A SEMANTIC BASED SOFTWARE REDOCUMENTATION USING ONTOLOGY WITH DISTRIBUTED PROCESSING TECHNIQUES

HIEW KHAI HANG

A project report submitted in partial fulfillment of the requirements for the award of Bachelor of Science (Honours) Software Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

October 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : HiewKhaiHang

Name : Hiew Khai Hang

ID No. : 1903181

Date : 6/10/2023

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"A SEMANTIC BASED SOFTWARE REDOCUMENTATION USING ONTOLOGY WITH DISTRIBUTED PROCESSING TECHNIQUES"** was prepared by **HIEW KHAI HANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Dr.Sugumaran a/l Nallusamy

Date : 6/10/2023

Signature :

Co-Supervisor :

Date :

ACKNOWLEDGEMENTS

I am deeply grateful for the collaborative effort and support of everyone who contributed to the successful completion of this project. Your dedication and contributions have been invaluable, and I want to express my heartfelt thanks.

First and foremost, I would like to extend my sincere appreciation to my supervisor, Ts Dr Sugumaran a/l Nallusamy. His guidance, expertise, and unwavering commitment to my project have been instrumental in its development. Ts Dr Sugumaran a/l Nallusamy provided invaluable insights and encouraged me to explore new horizons, shaping this project into what it is today.

Additionally, I want to express my gratitude to the Department of Internet Engineering and Computer Science (DIECS) faculty and staff for continued support and resources that facilitated the research and development process.

To my friends and family, who provided encouragement, understanding, and a listening ear throughout this journey, I offer my heartfelt thanks. Their unwavering belief in me sustained my motivation and determination.

Lastly, I want to thank all the participants and individuals who contributed their time, expertise, and insights to this project. Their contribution was indispensable in achieving the research goals.

This project would not have been possible without the collective efforts and encouragement of each and everyone of them. Together, we have achieved a significant milestone, and I am proud to have had the opportunity to work with such a dedicated and supportive group of people.

Thank you all for the contribution and support.

ABSTRACT

This project attempts to address the maintenance issues that industries experience as a result of the inadequate or non-existent documentation provided for the system, which drives up the cost to identify and fix system flaws. The time required to provide such documentation and the developer's belief that it is not important for the development process are the real causes of insufficient or non-existent source code documentation. Hence, the development of a web-based analysis system that manages user source code uploads and communicates with the Databricks cloud platform, which uses distributed processing techniques to quicken the redocumentation process, is the suggested solution for this project to address the root cause of the issue. The web-based analysis system is then able to retrieve and show the analysis data using the analysis result return. In addition, the web-based analysis system offers the creation of an ontology graph of the source code components, which illustrates the connections between each component. Three people were chosen to test the web-based analytic system as part of the evaluation process. The system usability score, which was determined by analysing the participants' responses, was 80.83%. This excellent result implies that the web-based analysis method is very user-friendly and usable. However, the participants' subsequent suggestions for improvement are also gathered in order to improve the operation of the web-based analytic system and meet the needs of the responders. With this semantically based redocumentation technique using distributed processing technology to produce documentation in order to enhance the efficiency of the development and debugging phases within a project team has been accomplished.

TABLE OF CONTENTS

DECLARATION i
APPROVAL FOR SUBMISSION ii
ACKNOWLEDGEMENTS iv
ABSTRACT v
TABLE OF CONTENTS vi
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF SYMBOLS / ABBREVIATIONS xix
LIST OF APPENDICES xx

CHAPTER

1 INTRODUCTION 1
1.1 Background of the Problem 1
1.2 Problem Statement 3
1.2.1 Time and budget consuming increased in maintaining the source code with inappropriate software documentation 3
1.2.2 Performance of current redocumentation tools reduce due to large source code 3
1.2.3 Inefficient in finding relevant information in different file in the source code with current redocumentation tools 4
1.3 Aim and Objectives 4
1.4 Proposed Solution 4
1.5 Proposed Approach 5
1.6 Project Scope 6
1.6.1 Extraction Module 6
1.6.2 Transformation Module 6
1.6.3 Store Data Module 7
1.6.4 Ontology Transformation Module 7
2 LITERATURE REVIEW 8
2.1 Introduction 8

	2.2	Reverse Engineering	8
	2.3	Software Redocumentation Process	9
	2.4	Redocumentation Tools	11
		2.4.1 Doxygen	11
		2.4.2 Javadoc	13
		2.4.3 PHPDocumentor	14
		2.4.4 Natural Docs	16
	2.5	Comparison and Analysis of Existing Redocumentation Tools	17
		2.5.1 Javadoc	18
		2.5.2 PHPDocumentor	18
		2.5.3 Doxygen	19
		2.5.4 Natural Docs	19
	2.6	Ontology	19
		2.6.1 Transformation from Data Repositories to Ontology Repositories	20
		2.6.2 Protégé – Ontology Editor	21
		2.6.3 HermiT Reasoner	23
		2.6.4 Querying Ontologies Using SPARQL	24
	2.7	Apache Spark	26
		2.7.1 Apache Spark Architecture	28
		2.7.2 Hadoop Distributed File System	28
3	**METHODOLOGY AND WORK PLAN**		**30**
	3.1	Introduction	30
	3.2	Software Development Methodology	30
	3.3	Project Plan	32
		3.3.1 Work Breakdown Structure (WBS)	32
		3.3.2 Gantt Chart	35
	3.4	Development Tools	38
		3.4.1 Databricks	38
		3.4.2 Protégé	38
		3.4.3 HermiT Reasoner	38
		3.4.4 RDFLib	38
		3.4.5 Amazon Simple Storage Service	39

		3.4.6 Amazon Lambda	39
		3.4.7 Flask API	39
		3.4.8 Laravel Framework	40
4	**PROJECT SPECIFICATION**		**41**
	4.1	Introduction	41
	4.2	Requirement Specification	41
		4.2.1 Functional Requirements	41
		4.2.2 Non-Functional Requirements	42
		4.2.3 Use Case Diagram	43
		4.2.4 Use Case Description	44
	4.3	Prototype Design	55
5	**SYSTEM DESIGN**		**59**
	5.1	Introduction	59
	5.2	System Architecture Design	59
		5.2.1 Laravel Framework Architecture	60
		5.2.2 Ontology Design	61
	5.3	Conclusion	65
6	**SYSTEM IMPLEMENTATION**		**66**
	6.1	Introduction	66
	6.2	Project Setup	66
		6.2.1 AWS S3 Setup	67
		6.2.2 Azure Databricks Workspace & Workflows Setup	69
		6.2.3 AWS Lambda Function	75
	6.3	System Modules	78
		6.3.1 Extraction Module	79
		6.3.2 Transformation Module	82
		6.3.3 Store Data Module	93
		6.3.4 Ontology Transformation Module	96
	6.4	System Deployment	132
	6.5	Conclusion	133
7	**SYSTEM TESTING**		**134**
	7.1	Introduction	134
	7.2	Unit Testing	134

7.2.1 Unit Testing for Extraction Module 135
7.2.2 Unit Testing for Transformation Module 137
7.2.3 Unit Testing for Store Data Module 145
7.2.4 Unit Testing for Ontology Transformation Module 148
7.3 Integration Testing 155
7.4 Performance Testing 171
7.5 System Usability Test 175
7.5.1 Test Scenario of Usability Testing 175
7.5.2 Result of Usability Testing 178
7.6 Manual Evaluate the Proposed OBSR with distributed processing techniques 180
8 CONCLUSION & RECOMMENDATION 187
8.1 Conclusion 187
8.2 Limitation and Recommendation for future work 188
9 REFERENCES 190
APPENDICES 195

LIST OF TABLES

Table 2.1: Functionalities Provided in the Redocumentation Tools 17

Table 4.1: Use Case Description for generate documentation 44

Table 4.2: Use Case Description for view documenetation 45

Table 4.3: Use Case Description for Search by keyword in generated documentation 47

Table 4.4: Use Case Description for generate & view graph 49

Table 4.5: Use Case Description for search by keyword in graph generated 51

Table 7.1: Unit Testing for Extraction Module 135

Table 7.2: Unit Testing for Transformation Module 137

Table 7.3: Unit Testing for Store Data Module 145

Table 7.4: Unit Testing for Ontology Transformation Module 148

Table 7.5: Integration Test Cases 156

Table 7.6: Execution Databricks Notebook Under Normal Load Test Case 172

Table 7.7: Ontology Construction Under Normal Load Test Case 173

Table 7.8: Used Time for Sending Different Flask API Request 174

Table 7.9: Sending Request to Flask API Under Normal Load Test Case 174

Table 7.10: Summary of Participants' Top Liked Features of the System 179

Table 7.11: Summary of suggestions for improving the system by participants 180

Table 7.12: Test Cases for evaluate the propose OBSR method 181

LIST OF FIGURES

Figure 1.1: Proposed OBSR Solution with Distributed Processing Techniques 4

Figure 2.1: Data Flow Overview (Doxygen, n.d.) 12

Figure 2.2: Javadoc Object Tree (Leslie, 2002) 13

Figure 2.3: XML Javadoc Tree (Leslie, 2002) 14

Figure 2.4: The Flow of the documentation process (phpDocumentor, n.d.) 15

Figure 2.5: Protégé Class Browser (Sivakumar & Arivoli, 2011) 22

Figure 2.6: Owlviz Graphical Representation (Sivakumar & Arivoli, 2011) 22

Figure 2.7: Results of the Performance Evaluation (Shearer, Motik & Horrocks, 2008) 24

Figure 2.8: Complete RDF graph (Castillo, Rothe & Leser, 2010) 25

Figure 2.9: SPARQL query patterns (Castillo, Rothe & Leser, 2010) 25

Figure 2.10: Query Result (Castillo, Rothe & Leser, 2010) 26

Figure 2.11: Apache Spark Architecture (Anurag Garg, 2023) 28

Figure 3.1: Evolutionary Prototype Model 30

Figure 3.2: Gantt Chart 1 35

Figure 3.3: Gantt Chart 2 36

Figure 3.4: Gantt Chart 3 36

Figure 3.5: Gantt Chart 4 37

Figure 3.6: Gantt Chart 5 37

Figure 4.1: Use Case Diagram 43

Figure 4.2: Home Page of the Redocumentation tools 55

Figure 4.3: Generate Documentation Page 55

Figure 4.4: Navigation Side Bar 56

Figure 4.5: Code Metrics of the whole source code folder 57

Figure 4.6: Class Page 57

Figure 4.7: User's Class Details Page 58

Figure 4.8: Search Class Page 58

Figure 5.1: Overview of the System Architecture Design 59

Figure 5.2: Source Code Components Ontology 63

Figure 5.3: Error prompt when the inconsistent of the ontology appear 65

Figure 6.1: Database Configuration In .env File 67

Figure 6.2: Amazon S3 Console 68

Figure 6.3: Create User Console 68

Figure 6.4: Set Permission Pages 68

Figure 6.5: Security Credentials Tab 69

Figure 6.6: Create Access Key Button 69

Figure 6.7: Retrieving Access Key 69

Figure 6.8: Fill in the information from S3 Bucket and Access Key Credential 69

Figure 6.9: Search Result in Azure Portal 70

Figure 6.10: Create A New Azure Databricks Workspace 70

Figure 6.11: Launching the new created workspace 71

Figure 6.12： Creating a new notebook 71

Figure 6.13: Creating New Compute Cluster 72

Figure 6.14: Attach Cluster to a notebook 72

Figure 6.15: Workflow Console 73

Figure 6.16: Create Automation Jobs with Databricks Workflows 73

Figure 6.17: Job Details 73

Figure 6.18: User Settings Button 74

Figure 6.19: Access Token Category in Developers Tab 74

Figure 6.20: Generating a new token 74

Figure 6.21: Token generate successfully 75

Figure 6.22: Create Function Button 75

Figure 6.23: Creating Lambda Function and Configuration 76

Figure 6.24: The Code to be execute when an event triggers this lambda function 77

Figure 6.25: Trigger Event Configuration 78

Figure 6.26: File Upload Page 79

Figure 6.27: Warning Message Display when validation failed 79

Figure 6.28: The Message Display After File has successfully uploaded 79

Figure 6.29: Code Segment of Upload File Controller 80

Figure 6.30: S3 Bucket Folder 80

Figure 6.31: Amazon Lambda Trigger when S3 Bucket has a folder name files/ 81

Figure 6.32: The script to be run when lambda function is trigger 81

Figure 6.33: The workflow to be triggered when the Databricks API is called 82

Figure 6.34: Mounting a S3 bucket into Databricks DBFS 82

Figure 6.35: Pre-processing of the data 83

Figure 6.36: Code Segment for declaring the class for each line of the source code 84

Figure 6.37: Code Segment to filtering null class and visualize the dataframe 84

Figure 6.38: Part of the visualization of the result 84

Figure 6.39: Code Segments of Finding Variable 85

Figure 6.40: Code Segments of Exception Handler and Result Visualization 85

Figure 6.41: Part of the Finding Variable Result 86

Figure 6.42: Code Segments of Finding Methods 87

Figure 6.43: Code Segments Result Visualization, Distinct and Count of Dataframe 87

Figure 6.44: Part of the Finding Method Result 88

Figure 6.45: Construct a new multidimensional collection and perform filter action 89

Figure 6.46: First Iteration of retrieve dependency function 90

Figure 6.47:Second Iteration of the retrieve dependency function 90

Figure 6.48: Third Iteration of the retrieve dependency function 91

Figure 6.49: Fourth Iteration of the retrieve dependency function 91

Figure 6.50: Merging and performing transformations action of dataframe 92

Figure 6.51: Part of the Dependencies Result 92

Figure 6.52: Function to Retrieve Code Metrics 93

Figure 6.53: Result of the Code Metrics Dataframe 93

Figure 6.54: Writing dataframe into CSV 94

Figure 6.55: Variable CSV Stored in S3 Bucket 94

Figure 6.56: Method CSV Stored in S3 Bucket 95

Figure 6.57: Dependencies CSV Stored in S3 Bucket 95

Figure 6.58: Metrics CSV Stored in S3 Bucket 95

Figure 6.59: Default Region Stored in config file 96

Figure 6.60: Access Key and Secret Access Key Stored in credentials file 97

Figure 6.62: Each of object retrieve from the response variable 98

Figure 6.64: Result of Dependencies Dataframe 99

Figure 6.65: Result of Methods Dataframe 99

Figure 6.66: Result of Variable Dataframe 99

Figure 6.67: Result of Metrics Dataframe 100

Figure 6.68: Import necessary classes and modules from RDFLib 100

Figure 6.69: Base Class Reference Defined 100

Figure 6.71: Explain example 101

Figure 6.72: Namespace define earlier 102

Figure 6.73: Example Ontology Output in TTL format 103

Figure 6.74: Import Flask Library in Jupyter Notebook 103

Figure 6.75: Creating Instance of Flask 104

Figure 6.76: GetAllMethod Route Defined 104

Figure 6.77: Running of the Flask application 105

Figure 6.78: Part of the result retrieve with the query in Ontology Graph 105

Figure 6.79: Transformation of the Retrieve Data into Dataframe 106

Figure 6.80: Filtering the data with “Has Method” Relationship 106

Figure 6.81: Dropping the Unnecessary Column 107

Figure 6.82: Output of the Jsonify Data 107

Figure 6.83: Better Visualization of Jsonify Data in Postman 107

Figure 6.88: Ontology Data Retrieved 109

Figure 6.89: Dataframe Transformed from the Ontology Data 109

Figure 6.90: Jsonify Data Retrieved in Postman 109

Figure 6.91: Retrieve Method Data by Class User Input 110

Figure 6.92: Retrieve Variable Data by Class User Input 110

Figure 6.93: Route to Download Complete Ontology File 110

Figure 6.94: File Downloaded After Sending Request 111

Figure 6.95: Splitting the Ontology with Only Dependencies Data 111

Figure 6.96: Splitting the Ontology with Only Variable Data 112

Figure 6.97: Splitting the Ontology with Only Method Data 113

Figure 6.98: Route declares to handle different URL request 114

Figure 6.99: Different Controller Has been Created to handle different request 114

Figure 6.100: Blade View Components 115

Figure 6.102: multipleUpload Method in FilesController 116

Figure 6.103: Index Method in FilesController 117

Figure 6.104: File Model 117

Figure 6.105: Output of the file-upload blade template with retrieved data 118

Figure 6.106: Error Prompt of the file-upload blade template with invalid file extension 118

Figure 6.107: Index Function of VariablesController 119

Figure 6.108: GuzzleHTTP Client used 119

Figure 6.110: Variable Data Display 120

Figure 6.111: Search Function in Variable Controller 121

Figure 6.112: Result Display by Searching “user” 121

Figure 6.113: Empty Result when the class name does not exist 122

Figure 6.114: Different Data Display 122

Figure 6.115: Index Method in MethodsController 123

Figure 6.116: Search Method in MethodsController 123

Figure 6.118: Method Data Display 124

Figure 6.119: Method Data Display by searching “user” 124

Figure 6.120: Method Data Display by searching non-existing class name 124

Figure 6.121: Index Method in DependenciesController 125

Figure 6.122: Search Method in DependenciesController 125

Figure 6.123: display-dependencies blade view component 126

Figure 6.124: Dependencies Data Display 126

Figure 6.125: Dependencies Data Display by searching "openstockcontroller" 127

Figure 6.126: Dependencies Data Display by searching non-existing class name 127

Figure 6.127: Index Method in MetricsController 128

Figure 6.128: display-metric blade view component 128

Figure 6.129: Metrics Data Display 129

Figure 6.130: Dropdown Navigation Bar 129

Figure 6.131: Step 1&2 To Generate Ontology Graph 130

Figure 6.132: Step 3 to generate Ontology Graph 130

Figure 6.133: Step 4 to search in the ontology graph 131

Figure 6.134: Example Output generated for method ontology graph 131

Figure 6.135: Some of the Git Action 132

Figure 6.136: GitHub Repositories 133

Figure 7.8: The Test Result 164

Figure 7.9: Test Cases for Method API 165

Figure 7.10: Test Cases for Variable API 166

Figure 7.11: Test Cases for Dependencies API 167

Figure 7.12: Test Cases for Metrics API 168

Figure 7.13: Test Cases for Different Ontology File Download 168

Figure 7.14: Integration Test in Retrieving Method Data 169

Figure 7.15: Integration Test in Retrieving Variable Data 170

Figure 7.16: Integration Test in Retrieving Dependency Data 170

Figure 7.17: Integration Test in Retrieving Ontology File 171

Figure 7.18: Total Used Time for Databricks Analysis & Produce Output 171

Figure 7.19: Total Used Time for Construct Ontology Graph 172

Figure 7.20: Variable in Stakeholder Class 180

Figure 7.21: Search Result with stakeholder's class variable 181

Figure 7.22: part of Openstockcontroller source code 183

Figure 7.23: Part of the dependencies output of openstockcontroller 184

Figure 7.24: Source code of stakeholder class 185

Figure 7.25: Result of the stakeholder class's method 186

LIST OF SYMBOLS / ABBREVIATIONS

API	-	Application Programming Interfaces
AWS	-	Amazon Web Service
CLI	-	Command Line Interface
CSS	-	Cascading Style Sheet
CSV	-	Comma-separated Values
DBFS	-	Databricks File System
HTML	-	Hypertext Markup Language
HTTP	-	Hypertext Transfer Protocol
JSON	-	JavaScript Object Notation
JSON-LD	-	JavaScript Object Notation for Linked Data
OBSR	-	Ontology Based Software Redocumentation Approach
OWL	-	Web Ontology Language
RDF	-	Resource Description Framework
S3	-	Simple Storage Service
SCSS	-	Sassy Cascading Style Sheet
SPARQL	-	Simple Protocol and RDF Query Language
SUS	-	System Usability Scale
TTL	-	Terse RDF Triple Language
URI	-	Uniform Resource Identifier
URL	-	Uniform Resource Locator
WAMP	-	Windows, Apache, MySQL, and PHP server
WBS	-	Work Breakdown Structure
WebVOWL	-	Web-based Visualization of Ontologies
XML	-	Extended Markup Language

LIST OF APPENDICES

Appendix A: Template of User Satisfaction Survey 195

Appendix B: Usability Test Responses 196

Appendix C: Keys Retrieving Process in S3 Bucket by Boto3 200

Appendix D: Information Extraction and Concatenate Dataframe 201

Appendix E: Iteration of different dataframe and transform into ontology 202

Appendix F: Different Routes to retrieve different Data 203

Appendix G: File-upload Blade view component 205

Appendix H: display-method blade view component 205

Appendix I: display-variable blade view component 206

Appendix J: API Testing Result 207

CHAPTER 1

INTRODUCTION

1.1 Background of the Problem

Software maintenance is a very broad term that refers to all changes made to a software system after it is put into use. Fixing problems, improving, eliminating, and adding capabilities, adjusting to changing operational settings and data needs, and improving performance, usability, or any other quality aspects are all examples of this (CANFORA & CIMITILE, 2001).

According to Kaur and Singh (2015), one of the maintenance challenges is due to the poor documentation quality which will lead to rise in the cost to detect any defects that are included in the system. This further suggests that maintenance effort is influenced by documentation quality. One of the main issues in comprehending a system is inadequate or inaccurate documentation(Freeman & Munro, 1992). Taking over development or maintenance tasks when system documentation was outdated, nonexistent, or inadequate and the source code is the only source for comprehending the written codes was a big difficulty for software engineers. Instead of developing a new module or maintaining the current software, the software engineer spent more time understanding the current code. So, to reduce maintenance work for the software developers, redocumentation is utilized to update the program documentation with the same abstraction which was in line with the most current code developments.

Our code should be documented for a variety of reasons. Future developers will be able to maintain and update the code more easily because of the documentation since they will be able to better comprehend our work and know where to make changes when modifications are required (Ryan, 2022). According to the study by Kaur and Singh (2015), The developer is able to save 12% of the cost of maintenance with up-to-date system documentation support compared to those developers without up-to-date system documentation support. The length of time required for the existing redocumentation strategy has gotten longer and longer as the system has increased in size as a result of the quick development of software systems with ever-increasing features and functionalities. When the

redocumentation tools are not able to handle the large size of source code, mostly the organization would just ignore the redocumentation part as long as the system is well functioning, the system documentation is not important from the organization's view. But once the maintenance and upgrading system process is being implemented, the developers and maintainers would take more time to understand how the system operates before they start implementing the changes. Moreover, those new developers who just joined the team would also face the same problem mentioned above. If the organization treats system documentation as an important item for the whole system developing process and the redocumentation tools are not able to handle the redocumentation process, it will cost the organization to have a staff who took responsibility for it and with low efficiency and time-consuming.

According to Nallusamy et al. (2021), the study of the class of redocumentation approaches—which includes XML-based method, incremental redocumentation, model-oriented redocumentation, etc.—was of poor quality in terms of granularity and efficiency. A trustworthy tool to extract software components, handled by the parser in the software redocumentation process, was necessary when a new software developer assumes development or maintenance responsibilities. The present parsers of the redocumentation tools might not be able to handle this massive quantity of data. Using the technologies now available on the market to redocument was time-consuming and unproductive. Hence, distributed processing approach was used to implement this challenge and resolve it.

Other than that, the current software redocumentation approach often comes out with HTMLs which represent different files in the system. Software developers or maintainers needed to navigate through different hyperlinks which represent different files in the source code to find relevant information on a specific component. This lacks the transitive closures that enable the construction of all pertinent information with a minimal amount of content navigation in the documentation. In order to facilitate the redocumentation process and represent information in the source code, ontology is therefore particularly employed. To aid program comprehension during maintenance activities, browsing, and semantic searching functionality is included in the produced HTML documentation. (Nallusamy,2015)

1.2 Problem Statement

To offer consistent and up-to-date software documentation that enables the software developers to understand the written codes and system documentation in an efficient way which allows they able to handle updating and maintenance of the source code with better code structure understanding, the software documentation is intended to develop and update automatically and regularly. There are now three key areas where redocumentation were having issues.

1.2.1 Time and budget consuming increased in maintaining the source code with inappropriate software documentation

The documentation is frequently manually produced in conventional software development, which can result in contradictions or obsolete information and cost developers important time and resources from the organization. The quality of software documentation is an important factor that affects the ability of developers to understand the system quickly. If the documentation was inappropriate and poor in quality, it will cost the developer time when dealing with the problem in the system (Kaur & Singh, 2015). When new software developers took longer time in understanding the code with inappropriate software documentation rather than developing or maintaining the code, which will cause a decrease in their performance and leads to the developing schedule or maintaining schedule becoming longer which cost additional time and money from the client.

1.2.2 Performance of current redocumentation tools reduce due to large source code

According to Nallusamy, Hao, and Zulkifle (2021), as technology advances and software gets complex over time, source code sizes increase as more files are added to handle more functions. As the complexity of the source code increases, the system's limited processing capacity will make redocumenting big sources of code challenging and time-consuming. Nevertheless, the issue may be resolved by dividing the source code and assigning a cluster of computers to handle each of their pieces of source code in the system constructed using distributed processing techniques.

1.2.3 Inefficient in finding relevant information in different file in the source code with current redocumentation tools

According to Nallusamy (2015), Software maintainers must visit several links from diverse pieces of data in the source code to get a single solution, and the provided link is unable to continuously traverse the content to create a new idea from the knowledge that was already there. The effectiveness and efficiency of the program documentation's exploration tools, such as browsing, searching, and visualizing the source code, were impacted by this.

1.3 Aim and Objectives

This project aims to study the existing redocumentation approaches and tools to understand their advantages and drawback in order to reduce the time consumed in the redocumentation task and deal with the uncertainty occurs when current redocumentation tools dealing with large source code using OBSR with distributed processing technique approach.

Objectives:

1. To develop a web application to handle the redocumentation process by the source code uploaded by the user and generate documentation and dependency diagram of the source code.
2. To create a data transformation method in the cloud platform which uses distributed processing technique and generate output return to the web application.
3. To evaluate the proposed OBSR with distributed processing technique approach using validating the correctness of the information and diagram generated.

1.4 Proposed Solution

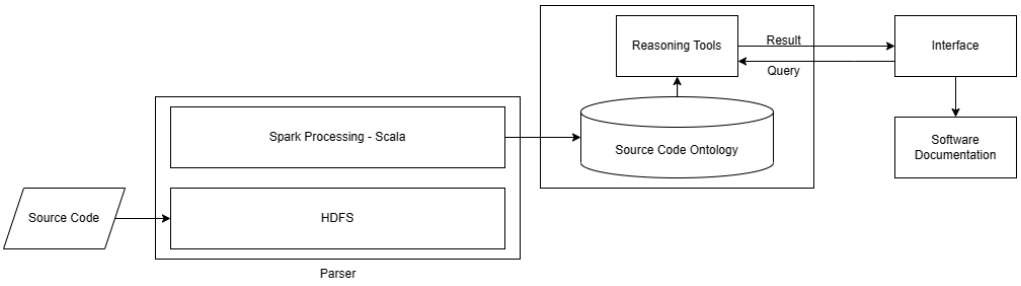


Figure 1.1: Proposed OBSR Solution with Distributed Processing Techniques

To resolve the problems outlined in the given problem statement, a web-based application for the deployment of semantically based software redocumentation employing ontology and distributed processing techniques was required. The web application first will receive the source code uploaded by the user and load it into the parser. The parser was used to extract crucial information from the source code and save it in the repository. The suggested method made use of HDFS to store, process, and analyze the source code across several commodity hardware nodes. A master node and a slave node will both exist. Blocks from the same file were distributed across multiple workstations by a master node, which also distributes the tasks to slave nodes at load time (Nallusamy, Hao & Zulkifle, 2021).

Other than that, the implementation of the OBSR method involves creating software redocumentation from the source code ontology and enabling software maintainers to browse and search for transitive relations and concept hierarchies using semantics. Source code ontology provides a knowledge repository, whereas the reasoning tool provides a query technique and verification method to improve the browsing and searching functionalities in the software documentation (Nallusamy, 2015).

1.5 Proposed Approach

The evolutionary prototyping technique was used in this project. It allows us to continuously roll out prototypes and improve them to make sure the client is happy with the product before installing the application, making it far more flexible than a waterfall approach. This approach combines incremental and extreme modelling. This technique reduces the likelihood of failure by enabling us to describe the development's scope and make new changes to meet client expectations since it identifies risk early on. When there are no defined requirements papers, using the prototype approach is a helpful way to gather and assess demands (Martinez, 2021).

The first phase in the evolutionary prototyping strategy was to plan, collect initial needs from the research, and assess the benefits and shortcomings of the current redocumentation approaches and tools. Through the analysis, we were able to identify the issue with the present redocumentation and design a solution to address the issue with a specific goal in mind. The next step was to create a prototype of different independents module which separated in the redocumentation tools based on the initial requirements acquired and design that helps the user understand the features and layout that were used in

the system. The client or user next assesses the prototype and provides suggestions and comments to help the redocumentation tools prototype develop. The user or client's feedback and comments were analyzed and used to improve the prototype through multiple iterations until the prototype was refined and improved and all requirements are met. Then, we move into the development phase to create an actual system using the prototype as a guide and additional testing to ensure that the redocumentation tools meet the client's expectations and requirements, and finally, the system was prepared for deployment.

1.6 Project Scope

By utilizing an ontology-based approach to software redocumentation using distributed processing technique, the approach should come out with the idea of integration of ontology-based repository and distributed processing techniques to produce a software documentation for maintaining the source code. The project scope involves several modules to meet the objective of this project which include Extraction Module, Transformation Module, Store Data Module, and Ontology Transformation module.

1.6.1 Extraction Module

The source code itself was the sole thing utilized in this project to extract the information. The following components, which may have originated from different programming languages, were taken out of the source code: forms, modules, functions, processes, event processes, tables, data reports, and variables, among others. The source code has been imported into the HDFS environment. To extract data for analysis, the next step was to load the source code into RDD, the main Spark storage structure.

1.6.2 Transformation Module

After the extraction module had finished loading the source code into RDD, transformations were used to tidy up, organize, and prepare the data for analysis. Filtering out extraneous data, aggregating data, joining data from many sources, and applying complicated calculations were examples of transformation-related jobs. The data can be saved in a dataframe and stored in as many output types, including XML, CSV, and others, after being filtered out using the built-in RDD function.

1.6.3 Store Data Module

After being transformed, the data was imported into the destination system, which was often a data lake, a data warehouse, or another type of storage solution. The data from the dataframe created in the transformation module was loaded into this module and saved in several output formats before being stored in a particular storage place, such as Amazon S3, Azure Data Lake Storage, and others.

1.6.4 Ontology Transformation Module

In this module, the source code ontology was created from the information in the CSV file. Data in the SCO are the source code components taken from the CSV file, which consists of class, method, variable, and dependency information. The individual was mapped to the relevant idea and role using the object property and data property. Finally, the SCO was used to display the information in the technical HTML software documentation.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter begins with a review of the existing literature on software maintenance, software documentation process, software documentation tools, reverse engineering and the concept and platform used during the redocumentation process.

2.2 Reverse Engineering

The use of program understanding technology is among the most promising solutions to the issue of software evolution. According to estimates, between fifty and ninety percent of evolution effort is focused on understanding or comprehending programs. While attempting to comprehend a program, programmers employ comprehension techniques, domain knowledge, and programming expertise. For instance, one may take the source code's syntactic information and use programming skills to create semantic abstractions (Müller, Wong & Tilley, 1993)

Reverse engineering is one technique to improve the programme comprehension process for large software systems as manual matching of such plans is challenging. Although while reverse engineering takes many different forms, the main objective is always to get data from already-in-use software systems. Using this information, future development can be enhanced, maintenance and re-engineering can be made simpler, and project management can be made easier.

Information extraction and abstraction are generally thought of as the two steps in the reverse engineering process. While abstraction develops documents and views that are user-oriented, information extraction analyses the relevant system artefacts to acquire raw data. (Nallusamy, 2015).

Redocumentation and design recovery are two major methodologies that can be used in reverse engineering approaches. The goal of redocumentation is to create or update different perspectives of a given artefact at the same level of abstraction, such as beautiful printing source code or visualising CFGs. Besides that, according to Gannod & Cheng (1999),

there are two types of reverse engineering: informal and formal. Informal methods essentially use pattern matching or synthetic structure analysis to retrieve artefacts from the source code. Formal approaches, in contrast, rely on mathematical logic, which necessitates running the specification and evaluating the program's characteristics. The benefit of formal approaches is that they give well-defined syntax and semantics for formal specification. Also, the syntax and semantics of the formal approach are well defined. Moreover, the formal approach offers inference rules that may be used to confirm the accuracy of each stage in the reverse engineering process.

In this project, the suggested solution makes use of formal approaches to redocument the source code in order to offer semantic knowledge representation. The inference rules, which can extract the semantic relationship from the repository, may be used by the formal procedures to convey the findings.

2.3 Software Redocumentation Process

In order to establish a procedure to use reverse engineering architecture to save the information in software, redocumentation mainly consists of four primary components. This procedure would result in documentation. The Software Work Product (SWP), the parser, the repository, and the software documentation make up the four primary parts.

- Software Work Product

  Source code, configuration files, generated scripts, and auxiliary artifacts make up SWP. A data collection tool, a handbook, a job control system, or a visual user interface that aids in source code comprehension can all be considered auxiliary artifacts. (Nallusamy, 2015)

- Parser

  The required data is extracted from SM using a parser, then it is stored in a repository and a system knowledge base. The parser's role is crucial in returning pertinent data with a certain approach. There are parsers that are exclusively interested in a certain language and those that are interested in different kinds of computer languages, like Universal Report. As specified by Marlow (2002), based on

a freely accessible generic Haskell parser supplied with GHC, haddock implements it. However, they were unable to utilize the parser because they wanted to modify the abstract syntax to include documentation annotations and supplement the grammar with additional products to handle documentation. As a result, the Haddock implementation includes a modified version of the original generic parser that was changed in the sections for lexical, abstract, and grammatical information. (Nallusamy, 2015)

- Repository

A repository stored information about source code metrics, and code-level relationship and dependencies. Usually, the data gathered in the repository is required to process and create knowledge, or to find hidden information which can show some interrelation within the source code. This information is published as a document by extracting it using a query language such as SQL or graph-based queries (Nallusamy, 2015). According to Kienle and Muller (2010), they use RSF text file as a repository and represent this as directed graph. The directed graph is presented as structural documentation in the Rigi Tools by extracting the data using the Rigi Command Language (RCL) query. (Nallusamy, 2015)

Generally, redocumentation uses the repository to analyse and present a hidden relationship in a source code. The query techniques used in the repository play an important role in browsing and searching the content in a documentation to present knowledge in different abstraction levels. (Nallusamy, 2015).

- Documentation

A number of documentation forms, including directed graphs, annotation, visualisation, metrics, or documentation, are then used to provide the processed data to the end user which include software maintainers. Modules, procedures, classes, subclasses, interfaces, control flows, composition, and enslavement are among the software components that are deleted. There are two types of written and graphical documentation for software. There are many different writing styles that may be used to create text documentation, from informal inline language to custom views that are dynamically created from a document database. Since they offer automated indexing and the creation of links between document portions or divisions, HTML and XML

are seen as more flexible forms of textual documentation. Using the advantages of HTML documentation, with all the tasks listed, is able to increase programme comprehension compared to a plain-text document. Static Images, which may employ a non-standard depiction of software artefacts and relationships, are the least well-established sort of graphical documentation. The most complex graphical documents may be modified by the user, giving them a greater opportunity to create original representations of the topic system. A software visualisation technique is used to give graphic documentation so that the maintainer may more easily understand (Nallusamy, 2015).

2.4 Redocumentation Tools

The most current state-of-the-art tools that have created solutions for the redocumentation process are presented in this area. The majority of the tools have been created for reverse engineering, which is sometimes equated to the process of redocumentation. However, several important tools have been found that can help with the continuing generation of high-quality documentation resulting from the redocumentation process.

2.4.1 Doxygen

Developed in 1997, Doxygen is a program that creates technical software documentation from source code. A stable version (1.8.3) of this specific utility was released on December 26, 2012, and it has since become the de facto industry standard. C, C++, C#, Java, and Python are just a few of the numerous programming languages that are supported by the tool. The tool employs improvised reverse engineering methods based on artificial source code analysis. Accepting the source code and processing it to an XML repository using lexical and synthetic analysis starts the redocumentation process. (Heesch, 2004)

Doxygen's major benefit is its ability to provide documentation in a number of forms, including HTM, RTF, LaTeX, and MAN. Doxygen has added indexing and searching features that let users look for data using the index that is supplied. The database is queried through the CGI protocol, which then returns the answers.

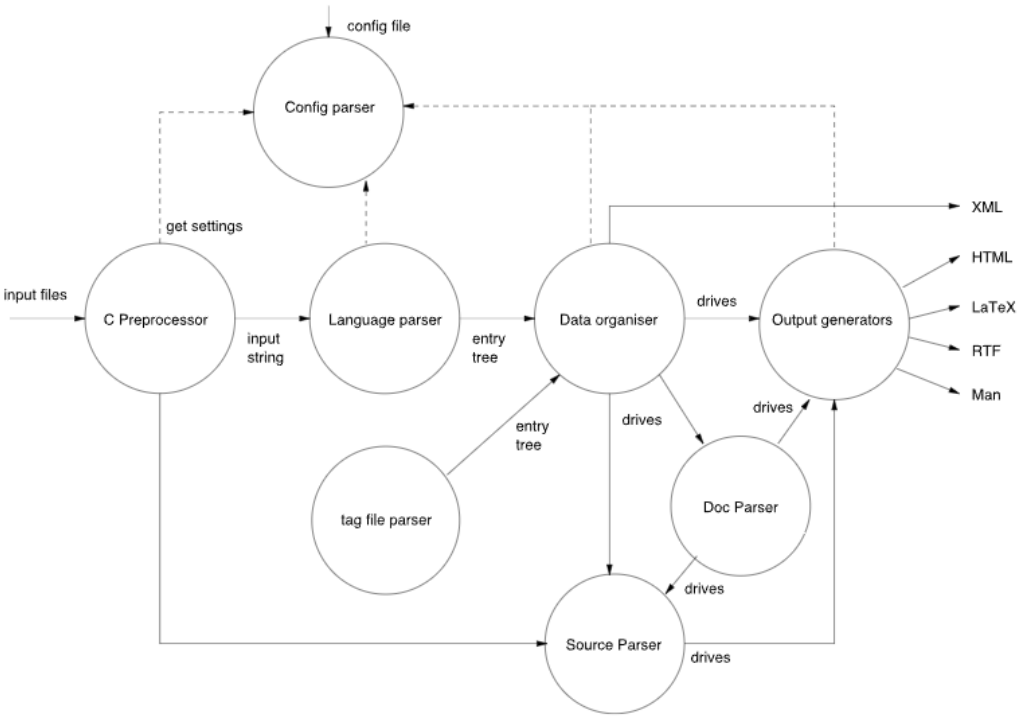


Figure 2.1: Data Flow Overview (Doxygen, n.d.)

According to the Data Flow Diagram above, each parser and processor deals with various duties independently before passing them on to the following parser to further extract the source code. A singleton class file named Config is utilized by the config parser to manage and save the settings of a project that is processed. The language parser, which is built as a large state machine, receives the input buffer that has been through the C Preprocessor's handling and loading of the input file. There is one parser for all the supported languages such as C/C++/Java. The parser's job is to turn the input buffer into an entry tree. An entry is described as a collection of unorganized data. It creates dictionaries of the extracted classes, files, namespaces, variables, functions, etc. throughout the data organizer process. The relationship between the extracted entities is computed too. If tag files are supplied in the configuration file, an XML parser based on SAX will read them and put Entry objects in the entry tree, marking the entry as external and containing the tag file's metadata. Special remark blocks that offer a brief or extensive description are saved as strings in the entities that they document by the documentation parser. Source parsers attempt to cross-reference the source code they parse with the entities that are described. The sources' syntax is highlighted as well. The output generators receive the output directly. Following the collection and cross-referencing of data, doxygen produces output in a number of formats. Directly from the collected data structures, XML is produced.

The XQuery-based XML query used by the Doxygen tool is seen as being preferable to SQL-based queries. It is possible to do searches for different data structures, including tree- and graph-based structures, using XML because of the way it is displayed in the recursive entity. For example, a database model and code structure may be created using XML. As a result, the same structure may be used by other sources, including existing documentation, to deliver the same XML information. The XML query language-created documentation can more precisely study important information than the relational approach can.

2.4.2 Javadoc

Javadoc scans a series of java source codes, which are frequently the source code for several Java packages, using the Java compiler. With each class definition and class-member declaration, the compiler produces information. This data is combined with the comments by the Javadoc tool to produce a class object tree structure that exactly matches the XML tree structure. The underlying Javadoc tool completes the parking process and gives the doclet access to a RootDoc object whether you are using a built-in HTML doclet or a custom doclet (Leslie, 2002).

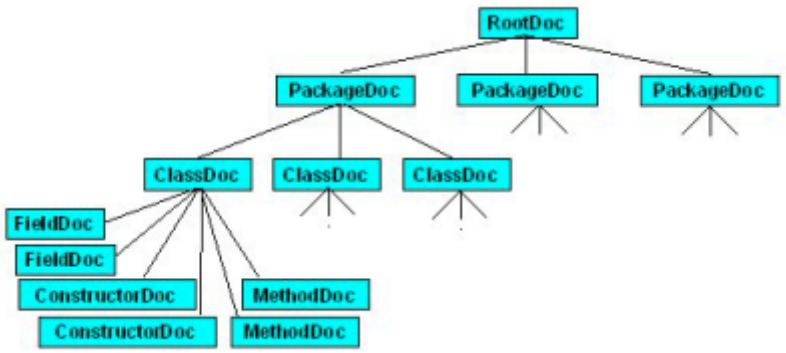


Figure 2.2: Javadoc Object Tree (Leslie, 2002)

The mapping of the Javadoc object hierarchy to XML comes next. To create XML that complies with our specific requirements, we may utilize element names and change the organization as necessary. We are able to generate a tree of XML documents along with the tree of HTML pages the conventional doclet produces, as opposed to producing a single XML document containing the full JavaDoc output.

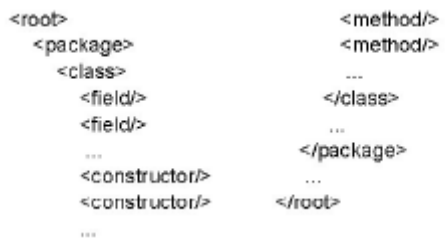


Figure 2.3: XML Javadoc Tree (Leslie, 2002)

2.4.3 PHPDocumentor

An open-source PHP documentation tool is called PHPDocumentor. PHPDocumentor automatically parses PHP source code and provides coherent API and source code documentation in a number of formats, depending on the structure of the source code itself and PHPDoc-formatted comments. You may produce documentation in HTML, PDF, CHM, or Docbook formats with PHPDocumentor..

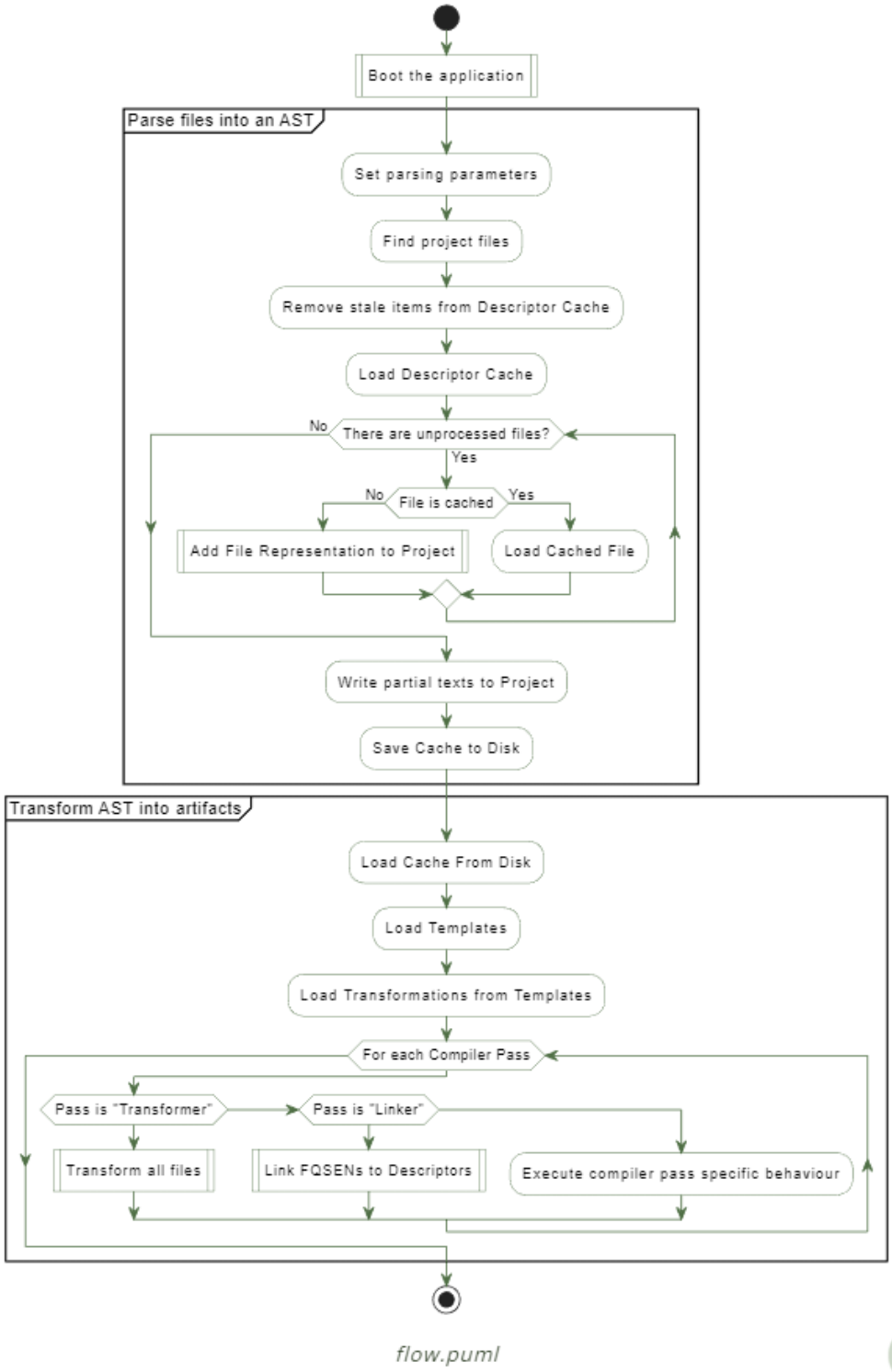


Figure 2.4: The Flow of the documentation process (phpDocumentor, n.d.)

The three-step procedure shown in the diagram above—Boot the application, Parse files into an AST, and Transform AST into artifacts—allows PHPDocumentor to deconstruct a project into its structural pieces and, depending on the template used, provide different sorts of output.

In order to correctly produce documentation, PHPDocumentor needs to locate all files in the project that we would like to document in the Parse files into an AST process. Many factors, such as directories and a listing indicating which files are disregarded, affect which files are eligible for our documentation based on the parameters and settings supplied.

If a cache of a prior PHPDocumentor run is present in the specified destination folder, it is loaded. In order to ensure that it doesn't include any items that aren't intended to be documented, PHPDocumentor will delete all files from that cache that aren't listed in the file listing that was discovered previously. After that, PHPDocumentor ought to have a description of our project, represented by an instance of the ProjectDescriptor class, which might be prepopulated with the Abstract Syntax Tree found from a prior run. PHPDocumentor will cycle over all files that were found before creating, or refreshing, the AST. Each file is hashed, and the cache is used to determine if the file is still recent. If the hash for a particular file does not exist in the cache or differs, PHPDocumentor will construct a new representation of that file and replace the old one (phpDocumentor, n.d.)

2.4.4 Natural Docs

A multi-language open-source documentation generator is called Natural Docs. We write documentation for our code using a natural syntax that is readable as plain English, and Natural Docs then scans our code and generates high-quality HTML documentation from it. (NaturalDocs, n.d.).

Natural docs use a range of parsing approaches, such as regular expressions and semantic analysis, to extract information from source code and output documentation only in HTML format. Natural Docs can produce documentation for non-code components like configuration files and database schemas as well as extract information about code elements like functions, variables, and classes.

Natural Docs' ability to handle code that is poorly commented or for languages without well-defined comment syntax is one of its advantages. Natural Docs uses its semantic analysis skills to deduce the structure and meaning of the code while working with poorly commented code. After classes, methods, functions, and other code components have been identified through code analysis, the tools employ a number of approaches to infer the documentation for those elements. For instance, it may look up the names of variables,

functions, and classes to figure out what they're for, or it might check the arguments and return types of functions to guess what they'll do.

Natural docs employ a combination of regular expressions and semantic analysis to extract information from the code itself for languages without well-defined comment syntax. The tools identify the components of the code through analysis, and then use regular expressions to retrieve details about those components. The name and type of a variable or function, or the argument list and return type of a function, for instance, may be determined using regular expressions.

2.5 Comparison and Analysis of Existing Redocumentation Tools

Table 2.1: Functionalities Provided in the Redocumentation Tools

	JavaDoc	PHPDocumentor	Doxygen	Natural Docs
Supported Language	Java	PHP	C/C++, C#, D, IDL, Fortran, Java, PHP, Python	Any Languages with comment
Generated Diagram		Class Inheritance Diagram	Caller & Callee Graphs, Inheritance Diagram, Dependency Graph, Collaboration Diagrams	Inheritance Diagram
Supported Format	HTML	HTML, CHM, PDF, XML	HTML, CHM, RTF, LaTeX, XML	HTML
Highlighting And Linking of Generated Doc		✓		
Searching	✓	✓	✓	✓

Function				
Browsing Option	Hypertext Link	Hypertext Link	Graphics	Hypertext Link

Based on the functionalities provided in Table 2.1, the limitations, and strengths of the existing redocumentation tools are identified as follows:

2.5.1 Javadoc

Without any created illustrations, Javadoc just facilitates the development of documentation for Java source code. Without the created diagram, the documentation is unable to describe how the parts of a Java class object relate to one another, and the maintainer must ascertain how each Java class is connected independently in order to completely understand the source code before doing their maintenance duties. The documentation output is produced using a very basic description search technique and only supports HTML. As a result, the selection and ranking of search results occasionally failed to live up to expectations. The software maintainer must manually navigate to the page with a hypertext link by using the navigation bar of the HTML documentation generated in order to explore specific classes in the document because the output generated does not highlight or link the class object in the HTML to the appropriate class. In addition, Javadoc might take a while to create documentation for big codebases, particularly if the source code is poorly organised. Moreover, it mainly relies on code comments to produce documentation. Incomplete or incorrect documentation may be created if the code is not well-documented using comments.

2.5.2 PHPDocumentor

As PHPDocumentor can only create class inheritance diagrams of the extracted source code and can only give documentation for PHP source code, it is rigid for a team that must work on other projects that utilise other development platforms. It creates documentation in more forms than Javadoc, such as HTML, CHM, PDF, and XML, making it simpler for programme maintainers. Using hypertext links, the generated documentation may also be accessed. The generated documentation allows the programme maintainer to look up the relevant class or function without using the navigation bar by providing matching highlights and links to the pages that are connected with each class name and function. PHPDocumentor includes a straightforward search option that periodically influences the ranking and selection of search results, much like Javadoc does. Moreover, the syntax-based parsing engine used by

PHPDocumentor might result in inaccurate documentation and sluggish performance if the massive source code base is poorly organised or contains mistakes.

2.5.3 Doxygen

Doxygen does not provide the ability to link generated documentation to each source code file, in contrast to the other two documentation tools. Although the query mechanism is simple, programme maintainers can browse the documentation's contents visually and textually. Software maintainers must therefore navigate through a lot of links from different pieces of information in the source code in order to discover a single knowledge element. Apart from that, configuring Doxygen for use with complicated codebases can be difficult and call for in-depth tool and codebase expertise. Dealing with large codebases takes time as well when the code is poorly organised.

2.5.4 Natural Docs

Natural Docs can only create inheritance diagrams and documentation output in HTML, which makes any customization beyond simple HTML formatting difficult. Other than that, it did not enable linking classes or modules together with corresponding headings and connections to related sites. This makes the system maintainers work harder as they examine the entire system to do a maintenance task.

2.6 Ontology

According to Ganapathy & Sagayaraj (2010), The collections of data known as ontologies are a crucial part of the semantic-based software redocumentation. The term "ontology," which is derived from philosophy, refers to the science that describes the many types of entities in the world and how they connect to one another. Ontology, according to Gruber, is the specification of conception. The axioms for limiting the relationship between words and the fundamental concepts and their relationships that make up the vocabulary of an application domain are defined by ontology. This definition describes the structure of an ontology. Taxonomy and a set of inference rules are features of the most common type of ontology used by software redocumentation tools. Taxonomy identifies groups of items and the connections between them. For usage on the Web, relations between entities, classes, and subclasses are extremely useful tools. By giving classes properties and enabling subclasses to inherit those characteristics, a huge number of relationships between entities may be described. Ontologies' inference rules add more power. A computer may be able to draw

inferences from an ontology's rules on the classes and relations. Although the computer does not actually "understand" any of this data, it is now far more capable of manipulating the words in ways that are helpful and clear to the user. Ontologies are used by more sophisticated applications to connect the data on a page to the underlying knowledge structures and inference procedures.

2.6.1 Transformation from Data Repositories to Ontology Repositories

An assortment of digital data may be found in a data repository, which one or more companies might employ to achieve a variety of goals. In literature, subject-specific datasets are sometimes referred to as data libraries. Also, a data library often maintains local data sets and provides access to them through a number of different channels. Although a data repository only offers basic operations like search, put, and get, a data library frequently provides access to the whole dataset (Hartmann, Palma & Gómez-Pérez, 2009).

Data warehouses, which analyze the stored data for management's decision-making, rose to prominence in the late 1980s and early 1990s. Periodically, data is appended to the repository, generally in this way. It may not, however, always have the analytical capabilities that a data warehouse offers.

A knowledge base is often a central archive for knowledge items. Ontologies are often used by knowledge bases to formally define its content and categorization system, but they can also contain unstructured or unformalized data that is expressed in procedural code or plain language. Furthermore, unlike a data repository, the goal of a knowledge base is often to enable automated deductive reasoning over the knowledge that has been recorded.

It is not unexpected that the ontology and semantic web communities started to show interest in using repositories to store semantic material a few years ago. Ontologies have had tremendous growth and application over the past few years, particularly in the semantic web's content. Ontologies are being created and used by academia and business to deliver new technologies and assist daily operations. As a result, there are presently many ontologies that have been created by several parties, making the ability to exchange and reuse them important.

Early attempts to compile a foundation of existing ontologies suggested developing a library system that provided a variety of tools for organizing, customizing, and standardizing collections of ontologies. With the help of this system, ontologies might be grouped and reorganized for later usage, integration, upkeep, mapping, and versioning (Hartmann, Palma & Gómez-Pérez, 2009).

2.6.2 Protégé – Ontology Editor

Ontology visualizations are used as information retrieval tools in applications that employ ontologies and have been integrated into ontology management systems like OntoUML and NavigOWL. Protégé's graphical user interface and Java API allow for interactive access to and modification of ontologies and knowledge bases. Pluggable components can be added to Protégé to bring additional features and services. A growing number of add-on plugins provide innumerable features, including additional ontology tools for management, multimedia support, querying and reasoning engines, methods for solving problems, and other features. A wide range of representation formats may be used to generate, view, and manipulate ontologies thanks to the extensive collection of knowledge-modelling structures and operations that Protégé supports. Protégé facilitates the construction of framework-based ontologies. An upgraded version of the frame-based system was produced in 2003 in order to support OWL with the benefit of the semantic web version. RDF, OWL, and XML schema are just a few of the forms in which Protégé ontology may be exported. (Sivakumar & Arivoli, 2011).

There are several ontology visualization methods in Protégé: Protégé Class Browser, Node Link and Tree, etc.

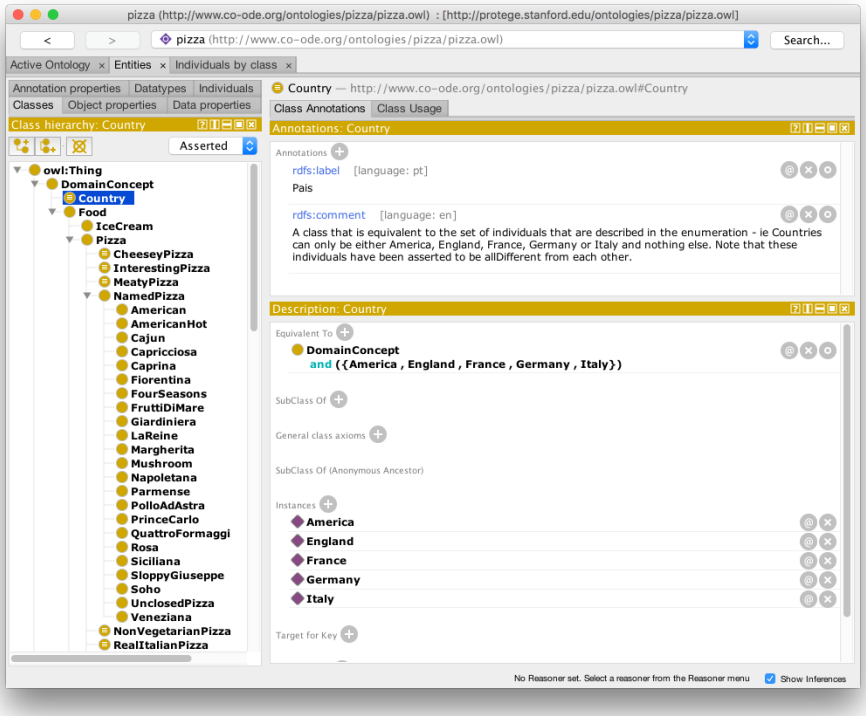


Figure 2.5: Protégé Class Browser (Sivakumar & Arivoli, 2011)

The taxonomy and axioms in the ontology are visualized using the Protégé plugin OWLviz as shown in Figure 2.6.

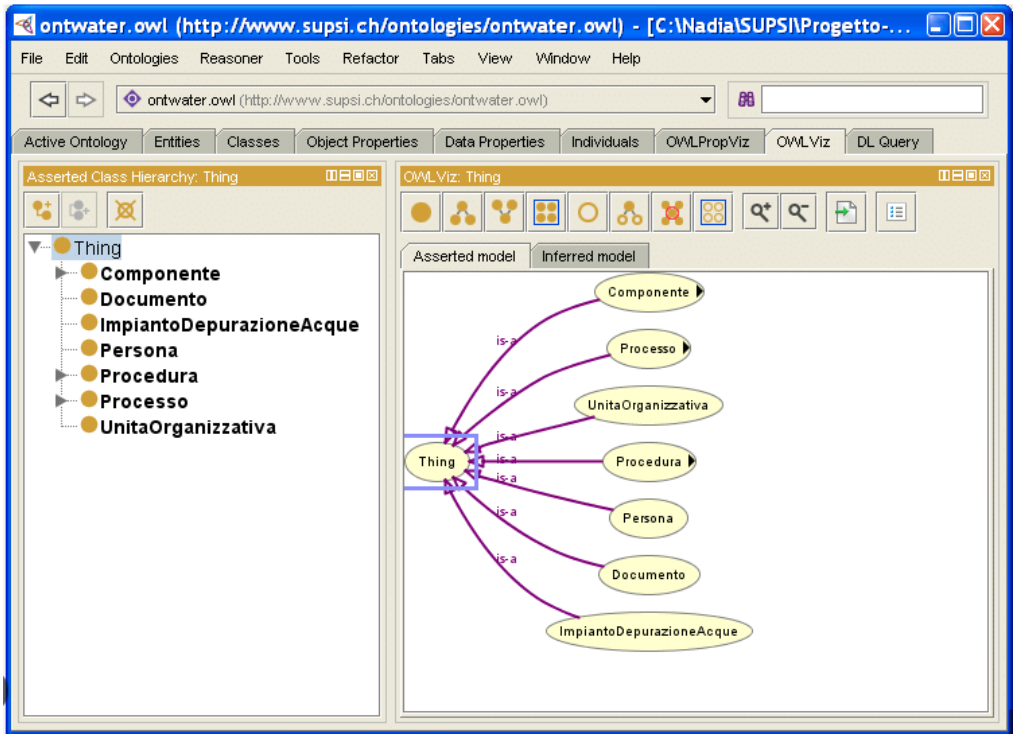


Figure 2.6: Owlviz Graphical Representation (Sivakumar & Arivoli, 2011)

2.6.3 HermiT Reasoner

Services for Description of Justification Subsumption testing and categorization for logic ontologies are often carried out by evaluating the consistency of many knowledge bases generated from the original ontology. For instance, analyzing the consistency of a knowledge base where at least one individual belongs to that class can quickly reveal whether a class is fulfilling. When they work to create a model of the knowledge base, Tableau reasoners do these consistency checks. Nevertheless, there are two challenges with building the models. First, there are frequently a large number of distinct structures that may be models; typically, a tableau algorithm must consider each of these possibilities before coming to the conclusion that no model is feasible. Second, even for ontologies that are somewhat small, the models created by tableau reasoners have the potential to be quite huge (Shearer, Motik & Horrocks 2008). These two kinds of complexity also commonly interact; for example, when building huge models, there are typically more potential models to consider, which makes practical reasoning hard.

Both of these causes of difficulty are addressed by the Descriptive Logic reasoning system HermiT Reasoner, which is built on a completely new architecture. The "hypertableau" calculus used by HermiT significantly decreases the number of potential models that must be taken into account (Shearer et al. 2008). The "anywhere blocking" method, which restricts the sizes of the models that are built, is another feature of HermiT. Lastly, HermiT employs an innovative and incredibly effective method for managing nominals when there are number constraints and inverse roles. A variety of other optimizations are also made possible by the combination of core algorithm advancements.

When an OWL file is fed into the HermiT reasoner, it determines if the ontology is coherent and establishes the connection of subsumption between the concepts. The test demonstrates that the HermiT reasoner is equally as quick as other reasoners and is even quicker when used to categorize complicated ontologies. The performance evaluation findings for several ontologies, which demonstrate how well they are categorized, are shown in Figure 2.2. The hypertableau rule application approach makes this feasible.

Ontology Name	Classification Times (seconds)			
	HermiT	HermiT-Anc	Pellet	FaCT++
Fly Taxonomy	1.1	1.2	1.2	5.3
GO Term DB	1.6	1.8	36.4	19.2
Biological Process	2.4	1.6	10.7	79.2
NCI	2.8	3.7	17.0	30.2
MGED	5.7	11.2	0.8	0.249
BP XP OBOL	8.7	8.5	505.1	1742.3
OWL Guide Food	19.3	29.6	14.2	1388.1
FMA Lite	43.8	error	error	error
DLP ExtDnS	95.8	error	7.1	0.1
FMA-constitutional part	error	error	error	error
GALEN-horrocks	1.5	1.5	13.5	156.9
Not-GALEN	1.6	1.8	54.1	200.4
GALEN-doctored	3.9	4.9	error	2836.1
GALEN-original	11.9	error	error	error
GALEN-module1	error	error	error	error
GALEN-full	error	error	error	error

Figure 2.7: Results of the Performance Evaluation (Shearer, Motik & Horrocks, 2008)

2.6.4 Querying Ontologies Using SPARQL

The management of the ontology by users through apps utilizing query-answering is crucial in ontology development. To access the data from the ontology, the query language for ontologies is required. The RDF-based query and the Logic/Ruled Based Query are two subcategories of the query language (Sirin & Parsia, 2007). SPARQL, which is essentially a data retriever based on the RDF triple format, supports RDF-based searches.

SPARQL is a query language designed for RDF, a data format represented as a directed labeled graph. SPARQL essentially function as a graph-matching query language, and it has three main components (Pérez, Arenas & Gutierrez 2009):

- Pattern Matching: This part of SPARQL allows to create queries that match pattern within RDF graphs. It has attributes like nesting, filtering, optional portions, union of patterns, and the option to indicate the data source to be matched.
- Solution Modifiers: After the pattern matching is performed, SPARQL offers solution modifiers to modify the results. These include classical operators like projection (selecting specific variables), distinct (removing duplicate result), order (sorting results) and limit (limiting the number of results returned).

- Query Output: SPARQL queries can produce various types of outputs. These include yes-or-no questions, choosing variable values that meet the patterns, creating new RDF data from these values, and resource descriptions (providing detailed information about specific resources in the RDF graph).

In essence, SPARQL is a versatile query language for working with RDF data, allowing users to efficiently retrieve, manipulate, and generate data in various ways based on graph patterns and criteria. Sample of the SPARQL queries and the result for the RDF graph are shown in the figures below:

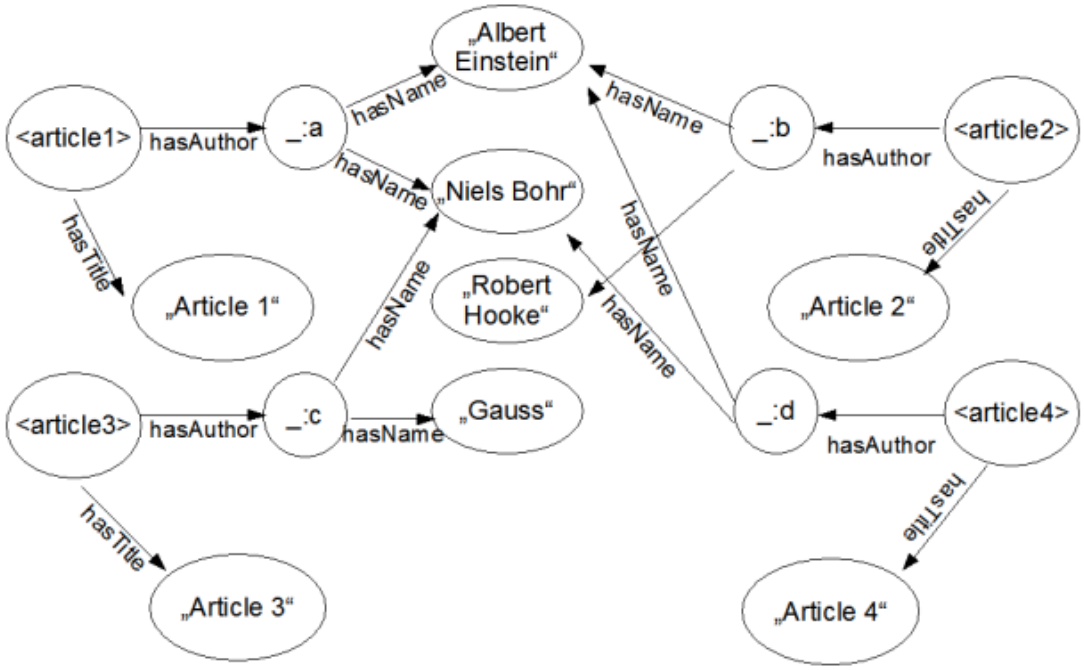


Figure 2.8: Complete RDF graph (Castillo, Rothe & Leser, 2010)

```
?article <hasTitle> ?title.
?article <hasAuthor> ?author.
?author <hasName> "Albert Einstein".
```

Figure 2.9: SPARQL query patterns (Castillo, Rothe & Leser, 2010)

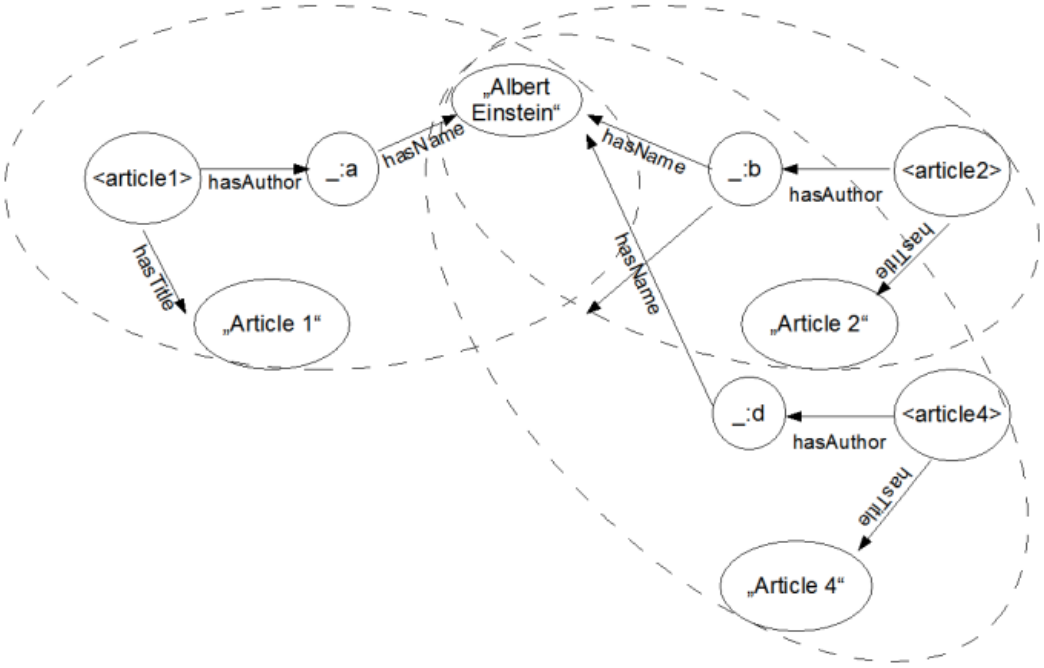


Figure 2.10: Query Result (Castillo, Rothe & Leser, 2010)

2.7 Apache Spark

According to (Pointer 2020), Apache Spark is a framework for data processing that can quickly perform operations on enormous amounts of data and divide processes over a number of servers, whether used alone or in cooperation with other distributed computing technologies. The fields of big data and machine learning, which demand the mobilization of huge computing capacity to manage enormous data warehouses, depend on these two features. Spark's simple API abstracts away the majority of the tedious work, relieving developers of some of the programming responsibilities related to distributed computing and large-scale data processing.

Apache Spark is one of the most well-known distributed big data analysis frameworks in the world. In addition to supporting the use of SQL, data streaming, machine learning, and graphical analysis, Spark provides native Java, Scala, Python, and R connectors. Spark can be used in a variety of contexts.

According to Mazzeschi (2021), Big data refers to a category of data whose analysis takes a significant amount of time and computational resources. Because of this, using huge data for analysis is never simple. First, we want a big data specialist, next we require a significant quantity of pricey computer capacity that may be rented from the cloud. Big data analysis is a talent that only a select few professionals have, as we can only learn it via practice and rigorous study and experimenting. This talent's worth is rising steadily.

Using our own equipment is the most typical method for carrying out any sort of data analysis. Due to the hardware's limited processing capability, it could take too long if the data is particularly large. Even with today's technology, processing a few Gigabytes of data while building a machine learning model will take a few hours. Scaling horizontally, which entails boosting GPU power to expedite the operation, is one potential approach. Thanks to cloud service providers, we can now rent GPU online. One of the greatest solutions for analyzing large amounts of data is still this technology, which is referred to as cloud computing.

Nevertheless, merely increasing computational power does not ensure that the analysis is carried out in the most effective manner. Thus, there's a potential that a few of the equipment we rented may use a lot of processing power inefficiently. Distributed computing can be helpful in this situation. Instead of using a single computer or a group of GPUs, distributed computing makes use of several distinct devices, each with its own GPU and CPU. In order to maximize process efficiency, the data is divided into several divisions before analysis in order to be used concurrently by all the machines in the cluster. As a result, the application will run as effectively as possible while using the least amount of computer resources. Cluster construction is a difficult process that calls very sophisticated software. Spark is only one of the alternatives.

A programming abstraction known as resilient distributed datasets (RDD), which may be scattered throughout a computer cluster, is the foundation upon which Apache Spark is based. Instead, operations on the RDDs might be spread out over the cluster and executed in parallel batches to provide rapid and scalable parallel processing.

Due to the combination of a driver core process, which divides a Spark application into tasks and distributes them to numerous executor processes, Spark operates in a distributed manner. These executors can be scaled up or down according to the application's

needs. In addition, Spark builds on Hadoop's MapReduce approach to accommodate various computation types, such stream processing and interactive searches, in a powerful way. In addition to that, Spark does not provide a framework for distributing file structure. Programmers install Spark on top of Hadoop to allow the advanced analytics applications of Spark to utilize the data stored using the Hadoop Distributed File System (HDFS)

2.7.1 Apache Spark Architecture

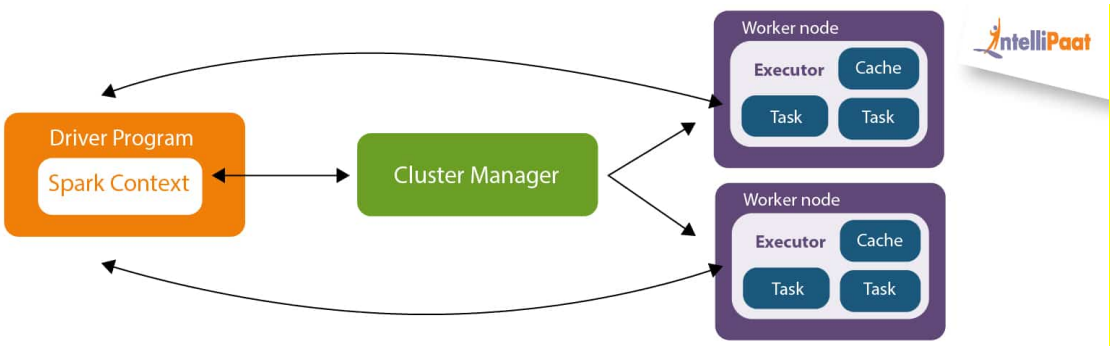


Figure 2.11: Apache Spark Architecture (Anurag Garg, 2023)

The driver program, which also constructs the Spark Context, invokes the application's main programme in the Apache Spark architecture. All of the necessary elements are present in a Spark Context. To keep track of how jobs are being completed in the cluster, Spark Driver and Spark Context work together. The responsibilities that Spark Driver is in charge of go much beyond those of the Cluster Manager. The Cluster Manager is in charge of assigning resources. After then, the job is divided up into several smaller jobs and dispatched to worker nodes. (Anurag Garg, 2023)

A number of worker nodes may be used to distribute and cache an RDD when it is formed in the Spark Context. Worker nodes carry out the tasks that the Cluster Manager gives them, finishing them, and sending the results back to the Spark Context. The executor is in responsible of fulfilling these obligations. Executor life expectancy is the same as that of the Spark Application. The system's performance may be improved by growing the worker node, which will allow the jobs to be split up into more logical chunks.

2.7.2 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is used to handle and store the enormous datasets typical of Big Data applications. Hadoop's primary file storage system is called HDFS. Because HDFS is fault resilient, it may be installed on low-cost, commodity hardware.

For applications requiring high-throughput data access, HDFS is available. Moreover, it enables Apache Spark and Hadoop to access file system data in streams. (Databricks, n.d.).

Hadoop is a system that utilizes distributed storage and parallel computing. Big data cannot be saved in a usual way, thus this may be used to sort and save it. An open-source Hadoop component project called HDFS offers several significant advantages when managing enormous amounts of data. Accepting mistakes will be the first step HDFS has been built to automatically identify issues and resolve them rapidly, ensuring dependability and stability. With the cluster design, 2 GB of data may be processed every second. Having access to new categories of data, particularly streaming data, will take third place. It is effective for managing streaming data because it was made to handle massive amounts of data for sequential processing and high information transfer rates. The fourth advantage is compatibility and mobility. Because HDFS is intended to be scalable to a variety of hardware configurations and works with several underlying operating systems, users are free to use it however they see suitable. Additional benefits include adaptability, affordability for large data quantities, and scalability (Databricks, n.d.).

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter discusses system development methods and project planning. The chapter provides a work breakdown structure (WBS) with a Gantt chart to see the planned timeline for each task and a description of the development tools used to construct this system.

3.2 Software Development Methodology

Prototyping is a software development method that involves creating, testing, and revising a prototype until it is functional. Also, it set the stage for the ultimate programme or system. It works effectively in situations when there are unclear project requirements. The process is one of repeated trial-and-error between the client and the developer.

In this project, our development method was evolutionary prototyping. The created prototype was progressively refined in response to supervisor feedback until it is eventually accepted. That allowed us to save time and effort. This was because it might occasionally be laborious to build a prototype from start. For initiatives utilising cutting-edge, poorly understood technology, this paradigm is helpful. Also, it was used in complex projects where each function only has to be checked once. When the demand was erratic or first poorly understood, it was helpful.

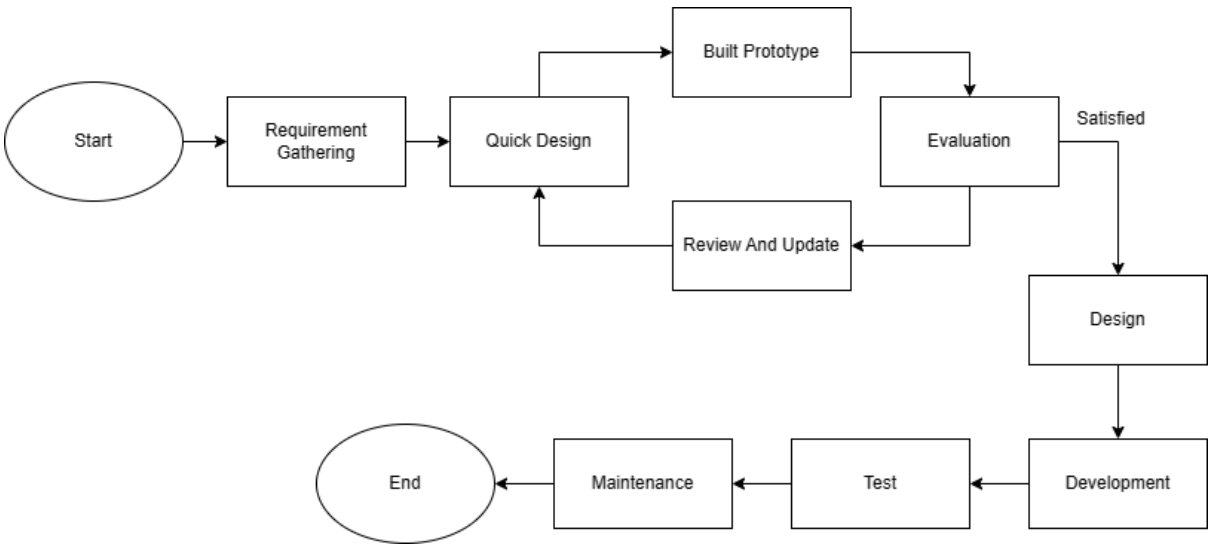


Figure 3.1: Evolutionary Prototype Model

1. Requirement Gathering

The goals, scopes, timetable, and restrictions of the project are all determined during this phase. The first requirement was gathered as part of the project planning process, which will dictate how the system was constructed. Initial needs were acquired by conducting research in the relevant subject, analyzing the present redocumentation system, and compiling information on the shortcomings and challenges of the current redocumentation tools. The need to address the shortcomings and challenges of the current redocumentation technologies was established using the information acquired. In addition, the new technology may make the same features of the old redocumentation tools, which were created in a new architecture, function better. The following action was to create a project plan that will list the actions required to complete the project. To help with project planning, the WBS and Gantt Chart were utilized to estimate the time needed to complete each subtask and meet the project's set milestones. WBS's main objective was to divide large, complicated jobs into a number of smaller subtasks. By presenting project activities that were declared in the WBS and their accompanying start and completion dates in a calendar format, the Gantt chart offers a standard structure for graphically illuminating information about project schedules.

2. Design

To give the project manager a visual representation of the proposed design solution, the Proposed OBSR Solution with Distributed Processing Techniques diagram was created in the quick design phase. It includes the design of the parser using the Spark platform and HDFS to implement distributed processing techniques in the project as well as the design of the knowledge repository to satisfy the goals and requirements in the first phase.

3. Built Prototype

After gathering requirements and doing some initial design work, the module prototype was made. This project make use of Azure Databrick, which provides an Apache Spark-optimized platform. Apache Spark tasks, such as the data extraction and data storage module, which shows some of the system's basic output, may be executed quickly and effectively thanks to this platform.

4. Evaluation

Users were shown the initial prototype produced in the previous step for assessment reasons. Following the assessment, the feedback and comments received from the user were recorded. This information was significant since it may help developers make the prototype better by revealing needs that aren't being satisfied. Any issues with the prototype also be uncovered throughout the examination.

5. Review & Refine

After several iterations, the prototype was refined and improved once the input has been examined. The prototype was iterated over and over again until the user is happy with it and all of the system's needs and goals were achieved. The prototype design was transformed into the finished actual system after the user approves it.

6. Development

The actual system was developed using the design from prototyping and new GUI which is not same as Databricks.

7. Testing

When the system's development is finished, testing will take place. The goal of software testing was to make sure that the system has been designed with the fewest possible flaws that could lead to system failure, that it complies with the technical specification established by its design and development, and that it effectively and efficiently satisfies the needs of the user, including handling all the exceptional and boundary cases.

8. Deployment & Maintenance

When a system successfully completes and passes each test that was run during testing, it is then ready for deployment. If a flaw was discovered while a user was using the system, further maintenance would permit the release of a new patch with bug remedies in the future.

3.3 Project Plan

3.3.1 Work Breakdown Structure (WBS)

0.0 A Semantic Based Software Redocumentation Using Ontology with Distributed Processing Techniques

1.0 Project Planning & Requirements Gathering

1.1. Preliminary Planning

1.1.1. Understanding Background of the Project

1.1.2. Identify The Problem of Current Existing Redocumentation Tools

1.1.3. Define Problem Statements

1.1.4. Determine Project Objectives

1.1.5. Define Project Proposed Solution

1.1.6. Select Project Proposed Approach

1.1.7. Define Project Scope

1.2. Literature Review

1.2.1. Review Software Redocumentation Process

1.2.2. Review Existing Software Redocumentation Tools

1.2.3. Review Ontology Concept in Constructing a Knowledge Repository

1.2.4. Review Spark Architecture and Distributed Processing Techniques

1.3. Methodology and Work Plan

1.3.1. Select Suitable Software Development Methodology

1.3.2. Develop Work Breakdown Structure

1.3.3. Develop Gantt Chart

1.3.4. Identify Software Development Tool

1.4. Requirement Identification

1.4.1. Requirement Specification

1.4.1.1. Gather Functional Requirement

1.4.1.2. Gather System Requirements

1.4.1.3. Gather Non-Functional Requirement

1.4.2. UML Modeling

1.4.2.1. Create Use Case Diagram

1.4.2.2. Create Use Case Description

2.0 System Development

2.1. First Iteration

2.1.1. Develop low-fidelity prototype with Databricks with built in GUI

2.1.2. Develop low-fidelity prototype for the web-based system which handles files upload and data visualization

2.1.3. Evaluation and gathering feedback

2.1.4. Refine prototype

2.2. Second Iteration

2.2.1. Design

2.2.1.1. Ontology Structure Design

2.2.1.2. System Architecture Design

2.2.2. Prototyping

2.2.2.1. Develop file-upload operation from web-application

2.2.2.2. Develop parser analysis algorithm

2.2.3. Evaluation and gathering feedback

2.2.4. Refine prototype

2.3. Third Iteration

2.3.1. Functionality Design

2.3.2. Web application prototyping

2.3.2.1. Develop Source Code Extraction Module

2.3.2.2. Develop Transformation Module in Databricks

2.3.2.3. Develop Store Data Module with CSV format into cloud storage

2.3.2.4. Develop Ontology Transformation Module with CSV Data

2.3.2.5. Develop Laravel Web Application to display Ontology Data

2.3.3. Evaluation and gathering feedback

2.3.4. Refine prototype

3.0 System Testing

3.1. Develop Test Plan & Test Cases

3.2. Unit Testing

3.3. Integration Testing

3.4. Performance Testing

3.5. System Usability Testing

4.0 Deployment

4.1. System Deployment

3.3.2 Gantt Chart

A SEMANTIC BASED SOFTWARE REDOCUMENTATION USING ONTOLOGY WITH DISTRIBUTED PROCESSING TECHNIQUES

Project Start: Mon, 30-Jan-2023
Display Week: 1

TASK	START	END
1.0 Project Planning & Requirements Gathering	30-Jan-23	17-Apr-23
1.1.Preliminary Planning	30-Jan-23	8-Mar-23
1.1.1.Understanding Background of the Project	30-Jan-23	20-Feb-23
1.1.2.Identify The Problem of Current Existing Redocumentation Tools	21-Feb-23	23-Feb-23
1.1.3.Define Problem Statements	24-Feb-23	25-Feb-23
1.1.4.Determine Project Objectives	26-Feb-23	28-Feb-23
1.1.5.Define Project Proposed Solution	1-Mar-23	2-Mar-23
1.1.6.Select Project Proposed Approach	3-Mar-23	4-Mar-23
1.1.7.Define Project Scope	5-Mar-23	8-Mar-23
1.2.Literature Review	20-Feb-23	17-Apr-23
1.2.1.Review Software Redocumentation Process	20-Feb-23	28-Feb-23
1.2.2.Review Existing Software Redocumentation Tools	1-Mar-23	15-Mar-23
1.2.3.Review Ontology Concept in Constructing a Knowledge Repository	16-Mar-23	1-Apr-23
1.2.4.Review Spark Architecture and Distributed Processing Techniques	2-Apr-23	17-Apr-23
1.3.Methodology and Work Plan	27-Mar-23	17-Apr-23
1.3.1.Select Suitable Software Development Methodology	27-Mar-23	30-Mar-23
1.3.2.Develop Work Breakdown Structure	31-Mar-23	5-Apr-23
1.3.3.Develop Gantt Chart	6-Apr-23	10-Apr-23
1.3.4.Identify Software Development Tool	11-Apr-23	17-Apr-23
1.4.Requirement Identification	6-Apr-23	17-Apr-23
1.4.1.Requirement Specification	6-Apr-23	12-Apr-23
1.4.1.1.Gather Functional Requirement	6-Apr-23	8-Apr-23
1.4.1.2.Gather System Requirements	9-Apr-23	10-Apr-23
1.4.1.3.Gather Non-Functional Requirement	11-Apr-23	12-Apr-23
1.4.2.UML Modeling	13-Apr-23	17-Apr-23
1.4.2.1.Create Use Case Diagram	13-Apr-23	14-Apr-23
1.4.2.2.Create Use Case Description	14-Apr-23	17-Apr-23

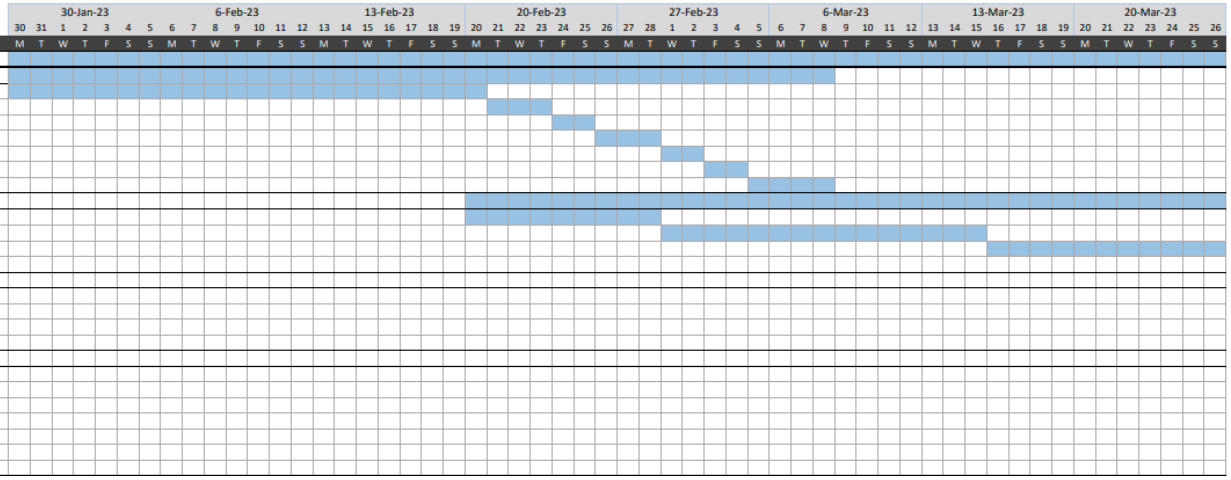


Figure 3.2: Gantt Chart 1

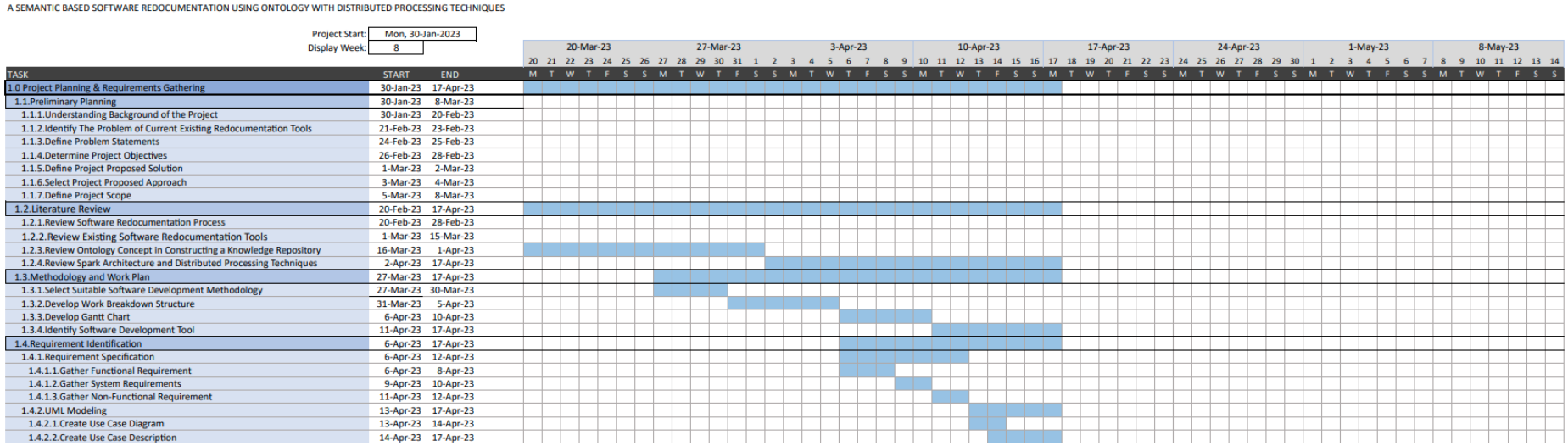


Figure 3.3: Gantt Chart 2

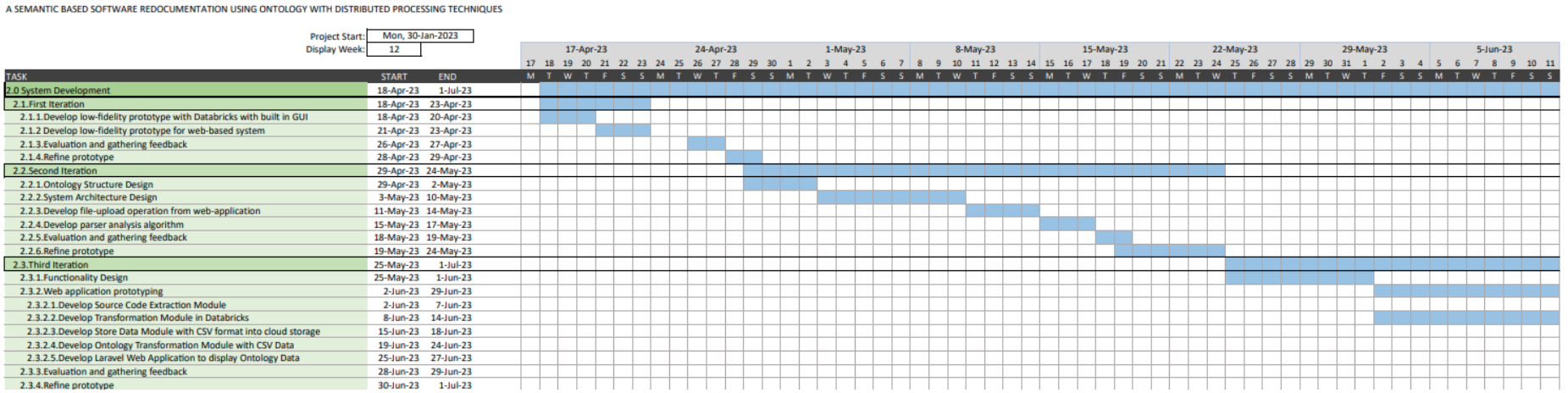


Figure 3.4: Gantt Chart 3

A SEMANTIC BASED SOFTWARE REDOCUMENTATION USING ONTOLOGY WITH DISTRIBUTED PROCESSING TECHNIQUES

Project Start: Mon, 30-Jan-2023
Display Week: 19

TASK	START	END
2.0 System Development	18-Apr-23	1-Jul-23
2.1.First Iteration	18-Apr-23	23-Apr-23
2.1.1.Develop low-fidelity prototype with Databricks with built in GUI	18-Apr-23	20-Apr-23
2.1.2 Develop low-fidelity prototype for web-based system	21-Apr-23	23-Apr-23
2.1.3.Evaluation and gathering feedback	26-Apr-23	27-Apr-23
2.1.4.Refine prototype	28-Apr-23	29-Apr-23
2.2.Second Iteration	29-Apr-23	24-May-23
2.2.1.Ontology Structure Design	29-Apr-23	2-May-23
2.2.2.System Architecture Design	3-May-23	10-May-23
2.2.3.Develop file-upload operation from web-application	11-May-23	14-May-23
2.2.4.Develop parser analysis algorithm	15-May-23	17-May-23
2.2.5.Evaluation and gathering feedback	18-May-23	19-May-23
2.2.6.Refine prototype	19-May-23	24-May-23
2.3.Third Iteration	25-May-23	1-Jul-23
2.3.1.Functionality Design	25-May-23	1-Jun-23
2.3.2.Web application prototyping	2-Jun-23	29-Jun-23
2.3.2.1.Develop Source Code Extraction Module	2-Jun-23	7-Jun-23
2.3.2.2.Develop Transformation Module in Databricks	8-Jun-23	14-Jun-23
2.3.2.3.Develop Store Data Module with CSV format into cloud storage	15-Jun-23	18-Jun-23
2.3.2.4.Develop Ontology Transformation Module with CSV Data	19-Jun-23	24-Jun-23
2.3.2.5.Develop Laravel Web Application to display Ontology Data	25-Jun-23	27-Jun-23
2.3.3.Evaluation and gathering feedback	28-Jun-23	29-Jun-23
2.3.4.Refine prototype	30-Jun-23	1-Jul-23

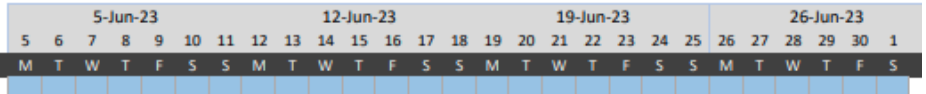


Figure 3.5: Gantt Chart 4

A SEMANTIC BASED SOFTWARE REDOCUMENTATION USING ONTOLOGY WITH DISTRIBUTED PROCESSING TECHNIQUES

Project Start: Mon, 30-Jan-2023
Display Week: 22

TASK	START	END
3.0 System Testing	1-Jul-23	1-Aug-23
3.1.Develop Test Plan & Test Cases	1-Jul-23	7-Jul-23
3.2.Unit Testing	8-Jul-23	14-Jul-23
3.3.Integration Testing	15-Jul-23	21-Jul-23
3.4.Performance Testing	22-Jul-23	23-Jul-23
3.5.System Usability Testing	24-Jul-23	1-Aug-23
4.0 Deployment	1-Aug-23	7-Aug-23
4.1.System Deployment	1-Aug-23	7-Aug-23

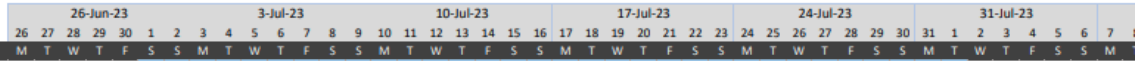


Figure 3.6: Gantt Chart 5

3.4 Development Tools

The proposed semantic based software redocumentation using ontology with distributed processing techniques tool has a web application for users to input their source code file in order to proceed with the redocumentation process of the source code. Therefore, web application development tools were used in this project to aid the development process. Other than that, Databricks was the platform to running the spark workloads.

3.4.1 Databricks

Databricks develops a web-based platform for employing Spark that includes Ipython programming notebooks and automated cluster administration. Using tools from BI to machine learning, it was used to process, store, clean, distribute, analyze, model, and monetize their datasets. The technology behind the Azure Databricks Lakehouse Platform, which powers the platforms' SQL warehouses and computer clusters, is called Apache Spark. Azure Databricks is a platform that is optimized for Apache Spark and provides a quick and simple way to run Apache Spark workloads.

3.4.2 Protégé

Protégé is a popular open-source ontology editor that allows us to create, edit and visualize ontologies. It provides a user-friendly interface for designing and managing ontologies and supports a wide range of ontology languages, including OWL, RDF and RDFS. This tool will be used to design the source code ontology with defining the logical class characteristics as OWL expressions.

3.4.3 HermiT Reasoner

The HermiT Reasoner is one of the new OWL reasoners that uses hyper tableau calculus. The designed source code ontology was required to be verified by classifying the ontology using this tool.

3.4.4 RDFLib

Working with RDF (Resource Description Framework) data is made easy with the help of the well-known Python package RDFLib. It is common practice to describe structured information and data in RDF, a standardized format for describing resources on the web in a way that is machine-readable. It's a crucial technology for the Semantic Web that makes it

possible to represent data in a form that computers can readily comprehend and interpret. With the use of SPARQL, which RDFLib provides, we can construct, manipulate, and query RDF graphs. Triples, or assertions with the pattern subject-predicate-object, make up RDF graphs. We can represent and interact in a systematic way with their triples thanks to the library. In addition, it supports a number of RDF serialization formats, including RDF/XML, Turtle, N-Triples, and JSON-LD. We can serialize RDF graphs into various file formats as a result.

3.4.5 Amazon Simple Storage Service

AWS provides the cloud-based storage of objects service known as Amazon S3. It is highly available and scalable. It enables us to easily and affordably store and retrieve data, including files, photographs, movies, backups, and more. A wide range of uses, including data storage, backup and archiving, data dissemination, content delivery, and data lakes, make extensive use of S3.

3.4.6 Amazon Lambda

AWS Lambda, or Amazon Lambda as it is more commonly known, is a serverless computing service offered by AWS. It enables us to execute code in response to different events without having to control infrastructure or servers. With Lambda, we can run our code in a highly scalable, on-demand way while only paying for the actual computing time we use. To automate the analysis process when S3 receives a file, we can trigger the Databricks workflow by using the Databricks API with the appropriate job ID using Lambda functions.

3.4.7 Flask API

Python has a simple and adaptable web framework called Flask. It is intended to make it simple to quickly and efficiently construct web apps. Flask gives developers the ability to select and include additional libraries as necessary while also providing the tools and components required to build online applications that can handle routing, templates, form handling, and more. In order to collect data from Amazon S3 storage and execute ontology altering with RDFLib function and return the result to our web application, we use Flask API with the RDFLib, a Python package.

3.4.8 Laravel Framework

In this project, the Python Flask Restful API's API endpoints were used in conjunction with the Laravel Framework's Blade Templates, Laravel Mix, to develop our web application. We may develop dynamic HTML views using Laravel's Blade templating engine to produce the application's front end. In order to include data from the backend in our HTML, we may utilize the Blade directive. In addition, Laravel Mix is a tool that makes it easier to compile front-end assets, such as SCSS into CSS and JavaScript. Popular JavaScript frameworks and libraries can be integrated with it as well.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

The project's preliminary specification, which involves defining the system's specification in terms of both functional and non-functional needs, is the chapter's main focus. It also contains explanations of each use case for tools for semantic software redocumentation that make use of ontologies and distributed processing methods, as well as a use case diagram. The high-level functionality and breadth of the system are graphically represented with the use case diagram, a specific type of diagram. The use case diagram also depicts the relationship between the actor and system. A documented record of the activities the actors take while utilizing the technology will be included in the use case description.

4.2 Requirement Specification

Reviewing current systems that are comparable to the proposed inventory management system outlined in Chapter 2 allows for the collection and identification of the requirements for the proposed system. Non-functional requirements and functional requirements were the two categories into which the requirement specification was divided. The system's services, as well as how it must react to different inputs and perform under diverse conditions, are listed in the functional requirements. Non-functional requirements were limitations on the system's ability to provide certain features or services, and they frequently apply to the system as a whole rather than to certain features or services individually.

4.2.1 Functional Requirements

Functional requirement for web-based application for semantic based software redocumentation tools are outlined in the list below:

- The system shall allow the users to input the source code file for analysis purposes.
- The system shall generate relevant documentation (Variable, Method, Dependency, Metrics) for each of the components in the source code.
- The system shall allow the users to search by the keywords to find the relevant components' documentation (Variable, Method, Dependency).

- The system shall generate a graph for different components and show the relationship between the components and classes.
- The system shall allow the users to search by the keyword to find the relevant dependencies in the dependency graph generated.

4.2.2 Non-Functional Requirements

- The interface of the web-based system should be easy to use, navigate, simple and consistent which allows the user to understand the workflow of the system easily.
- The time needed to generate documentation in this system should be shorter compared to other existing redocumentation tools.
- The system should ensure that the data inside the system will be protected.

4.2.3 Use Case Diagram

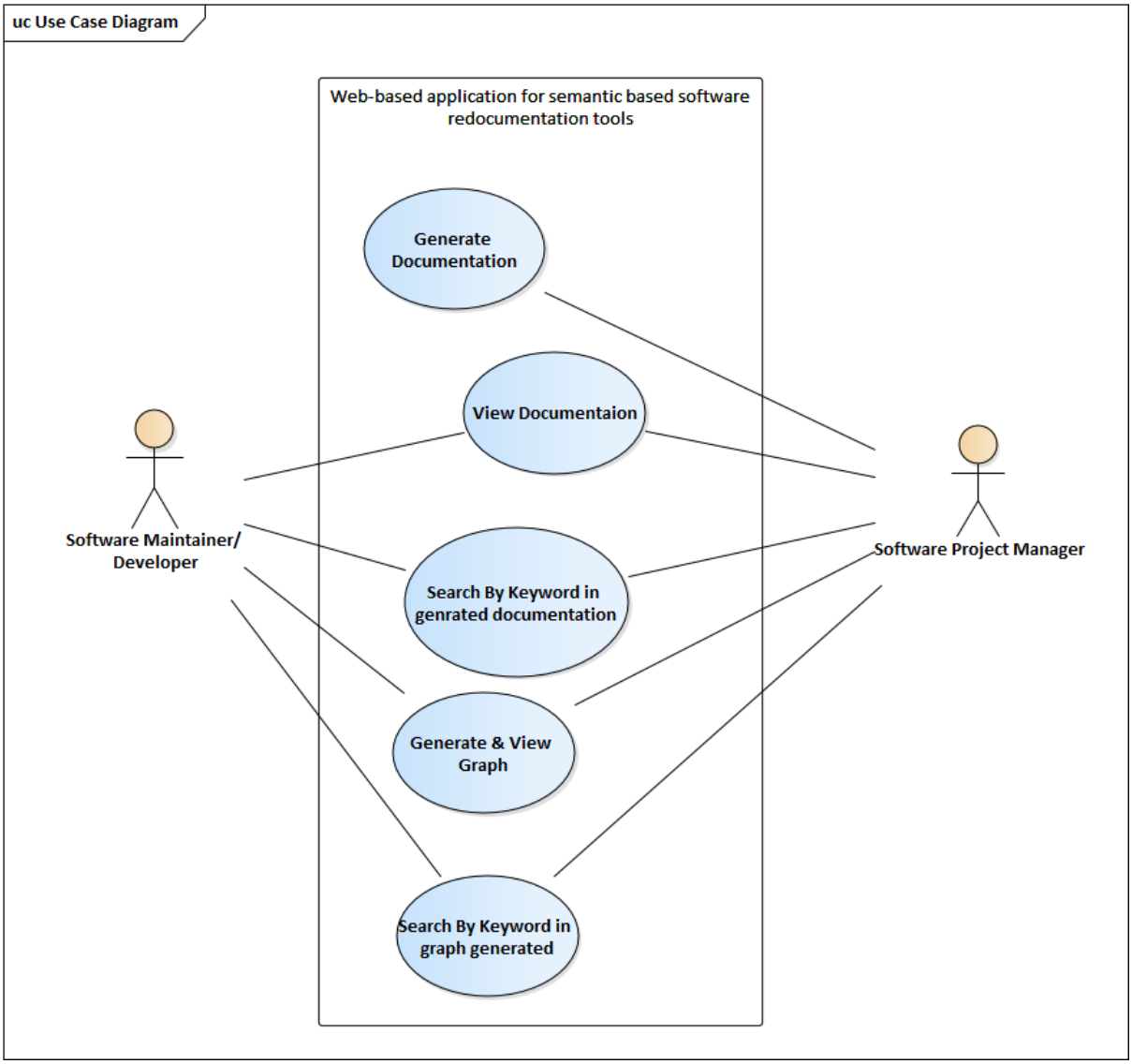


Figure 4.1: Use Case Diagram

4.2.4 Use Case Description

Table 4.1: Use Case Description for generate documentation

Use Case Name: Generate Documentation	ID: 1	Importance Level: *High*
Primary Actor: Software Project Manager	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Software Maintainer – The person who maintain the source code system and want to know the relationship and details of the source code without manually reviewing the source code.		
Brief Description: This use case describes how a software maintainer input the source code into the system and perform documentation process.		
Trigger: A software maintainer wants to know the relationship and details of the source code without manually reviewing the source code.		
Relationships: Association : software maintainer Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The software maintainers access the main page of the system which has the upload source code button. 2. The software maintainers click the 'upload' button. 3. The system will prompt an upload field to the software maintainers. 4. The software maintainers choose the folder which contain the source code components. 5. The software maintainers click the 'upload' button, and the system will store and start the documentation process.		
Sub-flows: 5.1 The system will validate the file extension of the file uploaded by the user		
Alternate/Exceptional Flows: 5.1 If the file upload by the user is not validate, an error message will be prompted to ask user to re-upload the file.		

5.2 If the file upload by the user is validate successfully, a file-upload successful message will be prompt and the file will be stored. 5.3 If the user leaves the file-upload field as empty, the system will prompt the user with a missing file input message.

Table 4.2: Use Case Description for view documenetation

Use Case Name: View Documentation	ID: 2	Importance Level: *High*
Primary Actor: Software Maintainer/Developer, Software Project Manager	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Software Maintainer – The person who maintain the source code system and want to know the relationship and details of the source code without manually reviewing the source code.		
Brief Description: This use case describes how a software maintainer access to the web-application to view the documentation generated.		
Trigger: A software maintainer wants to know the relationship and details of the source code without manually reviewing the source code.		
Relationships: Association : software maintainer Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: **View Metrics** 1. The software maintainers access the main page of the system. 2. The software maintainers select “DisplayMetric” button in the navigation bar. 3. The system will display the metric data of the whole source code component.		

View Variable 1. The software maintainers access the main page of the system. 2. The software maintainers select “DisplayVariable” button in the navigation bar. 3. The system will display the variable data with corresponding class component. **View Method** 1. The software maintainers access the main page of the system. 2. The software maintainers select “DisplayMethod” button in the navigation bar. 3. The system will display the method data with corresponding class component. **View Dependency** 1. The software maintainers access the main page of the system. 2. The software maintainers select “DisplayDependency” button in the navigation bar. 3. The system will display the dependency data with corresponding class component.
Sub-flows: **View Metric** 2.1 The system retrieves the metrics data from Flask Backend API. **View Variable** 2.1 The system retrieves the variable and class data from Flask Backend API. **View Method** 2.1 The system retrieves the method and class data from Flask Backend API. **View Dependency** 2.1 The system retrieves the dependency and class data from Flask Backend API.
Alternate/Exceptional Flows: **View Metric** 2.1 If the Flask Application is not up, a reject request error will be prompt to the user. **View Variable** 2.1 If the Flask Application is not up, a reject request error will be prompt to the user. **View Method** 2.1 If the Flask Application is not up, a reject request error will be prompt to the user. **View Dependency** 2.1 If the Flask Application is not up, a reject request error will be prompt to the user.

Table 4.3: Use Case Description for Search by keyword in generated documentation

<table>
<tr><td colspan="2">Use Case Name: Search by keyword in generated documentation</td><td>ID: 3</td><td>Importance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: Software Maintainer/Developer, Software Project Manager</td><td colspan="2">Use Case Type: Detailed, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
Software Maintainer – The person who maintain the source code system and want to know the details of a specific component in the source code.</td></tr>
<tr><td colspan="4">Brief Description: This use case describes how a software maintainer search and get the detail information of the component.</td></tr>
<tr><td colspan="4">Trigger: A software maintainer wants to know the details of a specific component in the source code</td></tr>
<tr><td colspan="4">Relationships:
Association : software maintainer
Include : N/A
Extend : N/A
Generalization : N/A</td></tr>
<tr><td colspan="4">Normal Flow of Events:
View Variable
1. The software maintainers access the main page of the system.
2. The software maintainers select “DisplayVariable” button in the navigation bar.
3. The system will then navigate to the variable data page will All variable data return.
4. The software maintainers key in the search key word on the top search input field and click on the “search” button.
5. The system will display the variable data search from the ontology by the user.
View Method
1. The software maintainers access the main page of the system.
2. The software maintainers select “DisplayMethod” button in the navigation bar.
3. The system will then navigate to the method data page will All method data return.
4. The software maintainers key in the search key word on the top search input field</td></tr>
</table>

and click on the “search” button.

5. The system will display the method data search from the ontology by the user.

View Dependency

1. The software maintainers access the main page of the system.
2. The software maintainers select “DisplayDependency” button in the navigation bar.
3. The system will then navigate to the dependency data page will All dependency data return.
4. The software maintainers key in the search key word on the top search input field and click on the “search” button.
5. The system will display the dependency data search from the ontology by the user.

Sub-flows:

View Variable

2.1 The system retrieves the variable and class data from Flask Backend API.

4.1 The system received the request with the user search key word and perform search query in the Flask Application.

4.2 The Flask Application then return the search variable data to the Web Application.

View Method

2.1 The system retrieves the method and class data from Flask Backend API.

4.1 The system received the request with the user search key word and perform search query in the Flask Application.

4.2 The Flask Application then return the search method data to the Web Application.

View Dependency

2.1 The system retrieves the dependency and class data from Flask Backend API.

4.1 The system received the request with the user search key word and perform search query in the Flask Application.

4.2 The Flask Application then return the search dependency data to the Web Application.

Alternate/Exceptional Flows:

View Variable

4.1 If the user search with empty keyword input, the Flask Application will return all variable data as default.

View Method

4.1 If the user search with empty keyword input, the Flask Application will return all

variable data as default. **View Dependency** 4.1 If the user search with empty keyword input, the Flask Application will return all variable data as default.

Table 4.4: Use Case Description for generate & view graph

Use Case Name: Generate and View Graph	ID: 4	Importance Level: *High*
Primary Actor: Software Maintainer/Developer, Software Project Manager	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Software Maintainer – The person who maintain the source code system and want to know the relationship of each component used in the source code.		
Brief Description: This use case describes how a software maintainer view the dependency graph between each of the used component.		
Trigger: A software maintainer wants to know the relationship of each component used in the source code in a graphical way which provide better understanding.		
Relationships: Association : software maintainer Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: **View & Generate Variable Graph** 1. The software maintainers access the main page of the system. 2. The software maintainers select “GenerateGraph” button in the navigation bar. 3. The system will then navigate to the generate graph page will procedure to generate the graph.		

4. The software maintainers click on “Variable Ontology” button to download the variable ontology file.
5. The software maintainers click on the link provide in the generate graph procedure and upload the variable ontology file.
6. The whole variable ontology graph is then generated with corresponding class relationship and display in the application.

View & Generate Method Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on “Method Ontology” button to download the method ontology file.
5. The software maintainers click on the link provide in the generate graph procedure and upload the method ontology file.
6. The whole method ontology graph is then generated with corresponding class relationship and display in the application.

View & Generate Dependency Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on “Dependency Ontology” button to download the dependency ontology file.
5. The software maintainers click on the link provide in the generate graph procedure and upload the dependency ontology file.
6. The whole dependency ontology graph is then generated with corresponding class relationship and display in the application.

View & Complete Ontology Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on “Complete Ontology” button to download the complete ontology file.

5. The software maintainers click on the link provide in the generate graph procedure and upload the complete ontology file. 6. The whole complete ontology graph is then generated with class, variable, method, metrics and depedencies relationship and display in the application.
Sub-flows:
Alternate/Exceptional Flows: **View & Generate Variable Graph** 5.1 If the file format is not valid, the WebVOWL will throw an invalid file error message. **View & Generate Method Graph** 5.1 If the file format is not valid, the WebVOWL will throw an invalid file error message. **View & Generate Dependency Graph** 5.1 If the file format is not valid, the WebVOWL will throw an invalid file error message. **View & Complete Ontology Graph** 5.1 If the file format is not valid, the WebVOWL will throw an invalid file error message.

Table 4.5: Use Case Description for search by keyword in graph generated

Use Case Name: Search By Keyword in graph generated	ID: 5	Importance Level: *High*
Primary Actor: Software Maintainer/Developer, Software Project Manager	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Software Maintainer – The person who maintain the source code system and want to locate the actual components with its corresponding relationship with the others.		
Brief Description: This use case describes how a software maintainer search through the		

generated graph to get the information of the relationship between each of the nodes.

Trigger: A software maintainer wants to know the relationship and details of the source code components which has linkage to each other.

Relationships:

Association : software maintainer

Include : N/A

Extend : N/A

Generalization : N/A

Normal Flow of Events:

Search Variable Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on the link provide in the generate graph procedure and upload the variable ontology file.
5. The whole variable ontology graph is then generated.
6. The software maintainer key in the search key word in the input field below.
7. The node with the search key will be highlighted with a red circle allow the software maintainer to locate the searched node.

Search Method Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on the link provide in the generate graph procedure and upload the method ontology file.
5. The whole method ontology graph is then generated.
6. The software maintainer key in the search key word in the input field below.
7. The node with the search key will be highlighted with a red circle allow the software

maintainer to locate the searched node.

Search Dependency Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on the link provide in the generate graph procedure and upload the dependency ontology file.
5. The whole dependency ontology graph is then generated.
6. The software maintainer key in the search key word in the input field below.
7. The node with the search key will be highlighted with a red circle allow the software maintainer to locate the searched node.

Search Ontology Graph

1. The software maintainers access the main page of the system.
2. The software maintainers select “GenerateGraph” button in the navigation bar.
3. The system will then navigate to the generate graph page will procedure to generate the graph.
4. The software maintainers click on the link provide in the generate graph procedure and upload the complete ontology file.
5. The whole complete ontology graph is then generated.
6. The software maintainer key in the search key word in the input field below.
7. The node with the search key will be highlighted with a red circle allow the software maintainer to locate the searched node.

Sub-flows:

Search Variable Graph

6.1 If the keyword input by the software maintainer exists, the search input will prompt a several option which contain the keyword input string by the software maintainer to choose and the chosen node will be highlighted.

Search Method Graph

6.1 If the keyword input by the software maintainer exists, the search input will prompt a several option which contain the keyword input string by the software maintainer to choose and the chosen node will be highlighted.

Search Dependency Graph

6.1 If the keyword input by the software maintainer exists, the search input will prompt a several option which contain the keyword input string by the software maintainer to choose and the chosen node will be highlighted. **Search Ontology Graph** 6.1 If the keyword input by the software maintainer exists, the search input will prompt a several option which contain the keyword input string by the software maintainer to choose and the chosen node will be highlighted.
Alternate/Exceptional Flows: **Search Variable Graph** 6.1 If the user search keyword is not found, there will be no option for the software maintainer to select, hence no highlighted node will be display. **Search Method Graph** 6.1 If the user search keyword is not found, there will be no option for the software maintainer to select, hence no highlighted node will be display. **Search Dependency Graph** 6.1 If the user search keyword is not found, there will be no option for the software maintainer to select, hence no highlighted node will be display. **Search Ontology Graph** 6.1 If the user search keyword is not found, there will be no option for the software maintainer to select, hence no highlighted node will be display.

4.3 Prototype Design

This section showing the prototype design which include the layout of each webpage and the design used in the actual development progress.

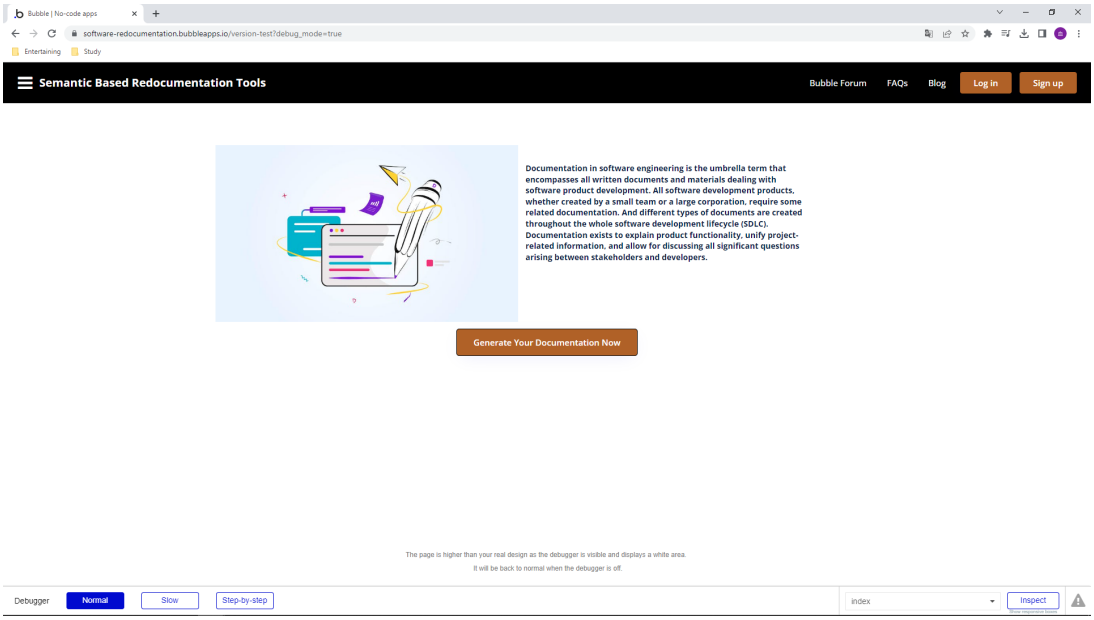


Figure 4.2: Home Page of the Redocumentation tools

The user was able to click on the generate documentation button to navigate to the page that allows the user to upload the source code and perform documentation process.

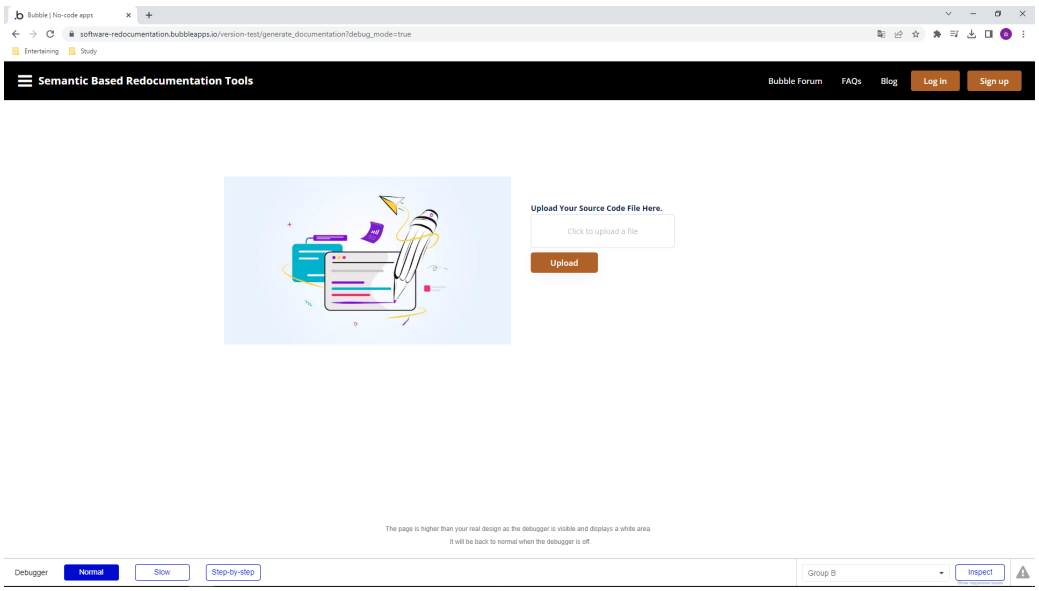


Figure 4.3: Generate Documentation Page

The user was allowed to upload the source code folder for analysis purposes, and the documentation will be saved to a web server which allow the user to be access by the navigation tab.

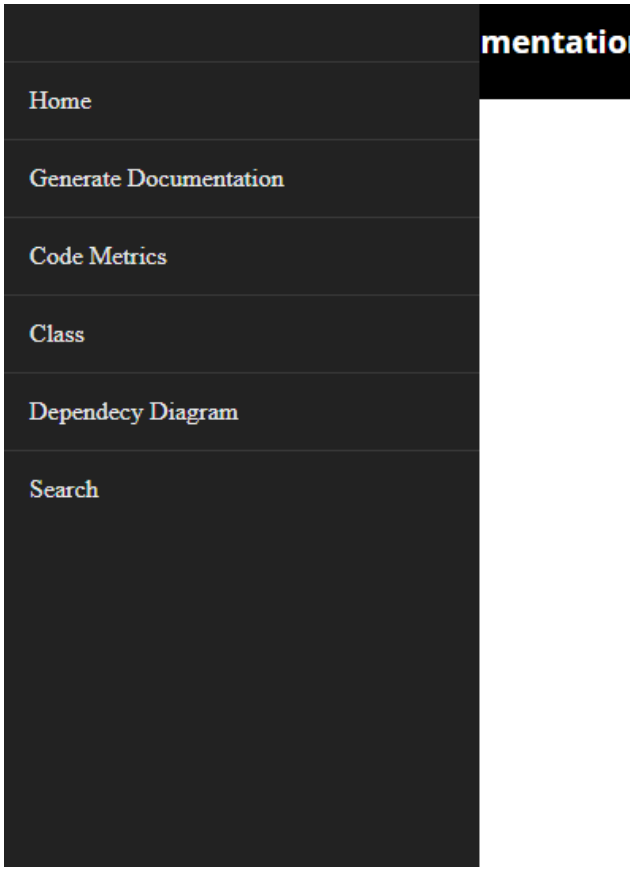


Figure 4.4: Navigation Side Bar

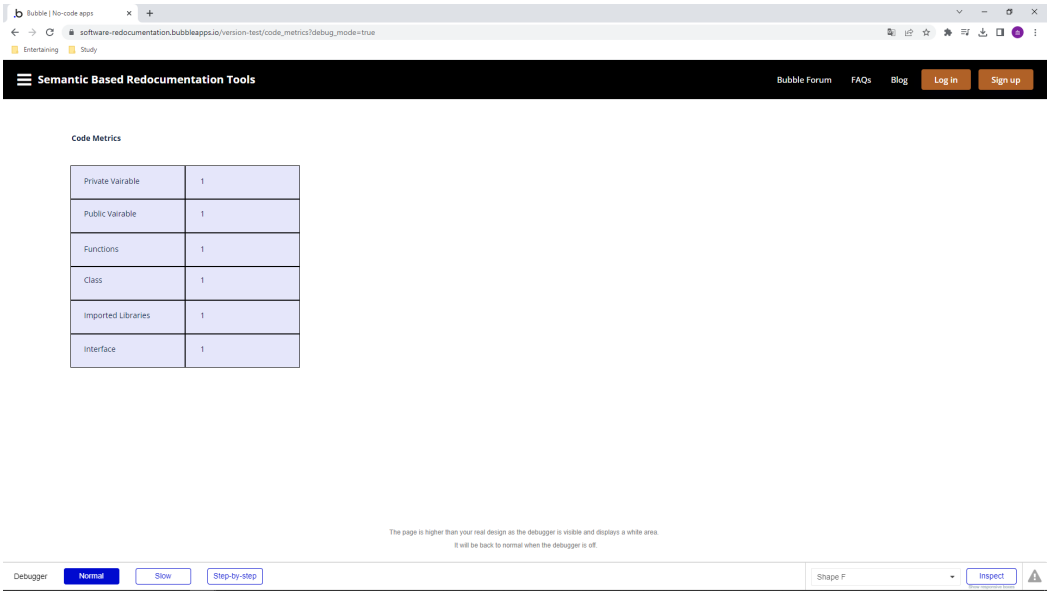


Figure 4.5: Code Metrics of the whole source code folder

Upon the end of the documentation process, information on other related components utilized in a specific source code as well as the software's measurement of the source code's complexity is presented in these pages.

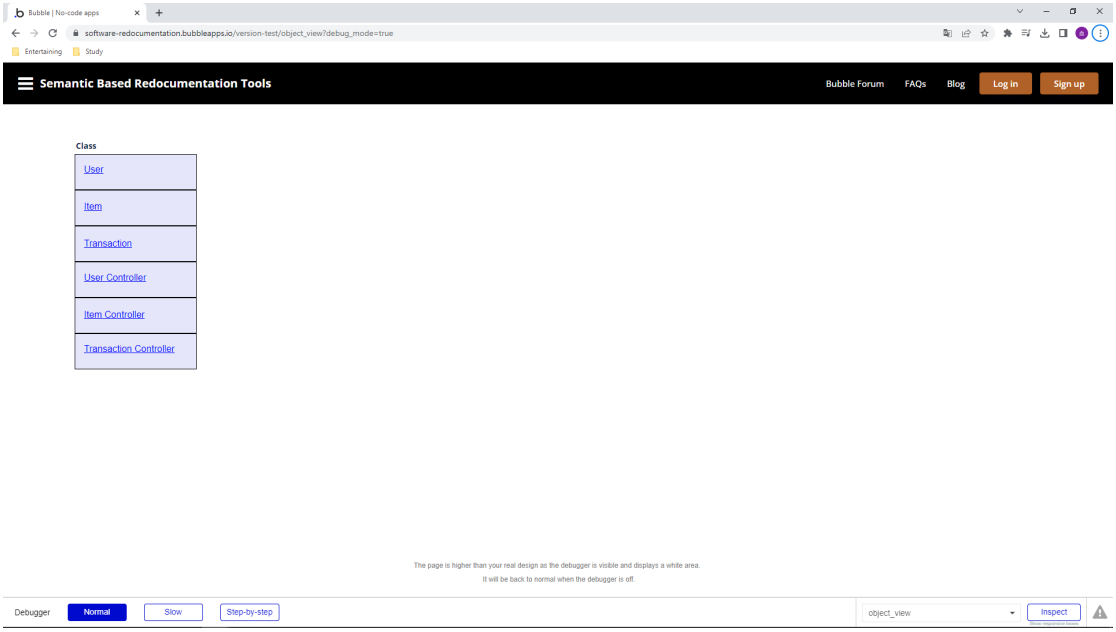


Figure 4.6: Class Page

This page provide the link to each of the class in the source code, in order to view the details of the selected class.

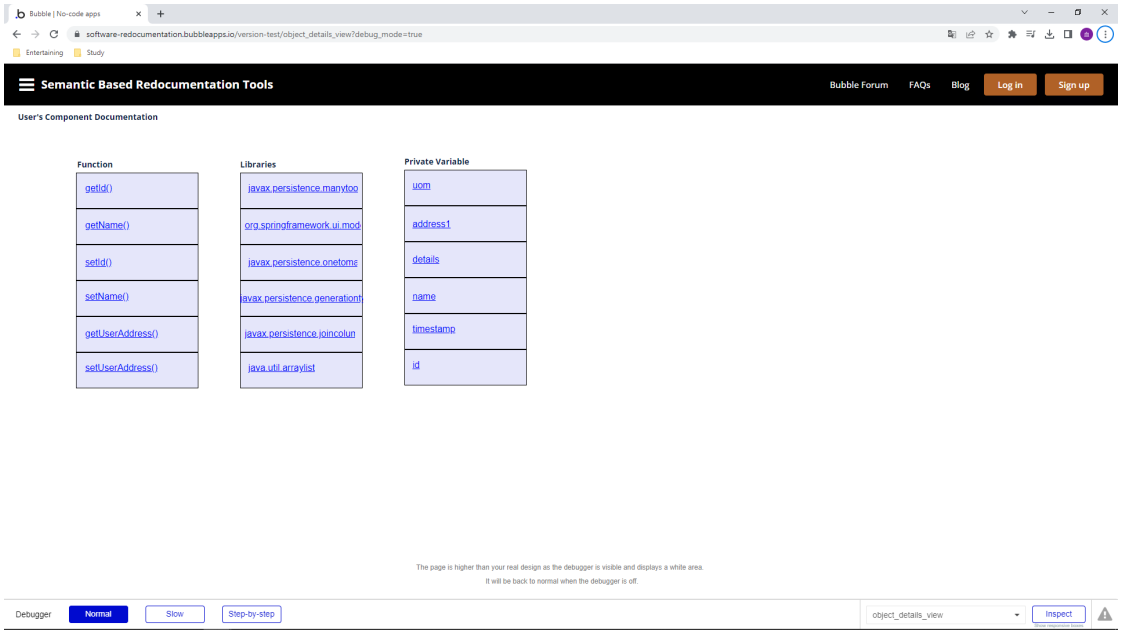


Figure 4.7: User's Class Details Page

This page show all the details such as function, libraries, private variables and etc information used in this class.

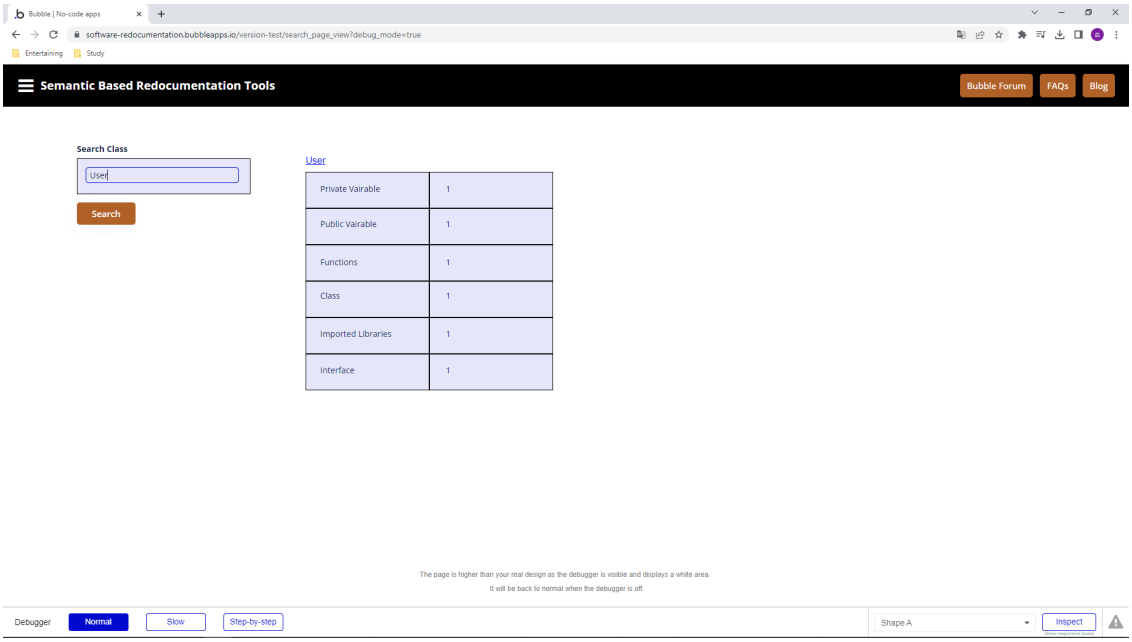


Figure 4.8: Search Class Page

This page allows the user to input the class name for search component purposes. If there was any result found, the class with link which enables the user to navigate to the corresponding class detail page will be provided. Other than that, the general information about the class will be shown too.

CHAPTER 5
SYSTEM DESIGN

5.1 Introduction

An overview of the system architecture and ontology design employed in this project is given in this chapter. The Laravel framework and the cloud platform used to build the system are both covered in-depth in the system's architectural design. However, using an external application called WebVOWL (Web-based representation of Ontologies), the developed ontology is used to describe the relationship between the classes and methods before being utilized to generate a graph representation of the dependencies of the entire source code.

5.2 System Architecture Design

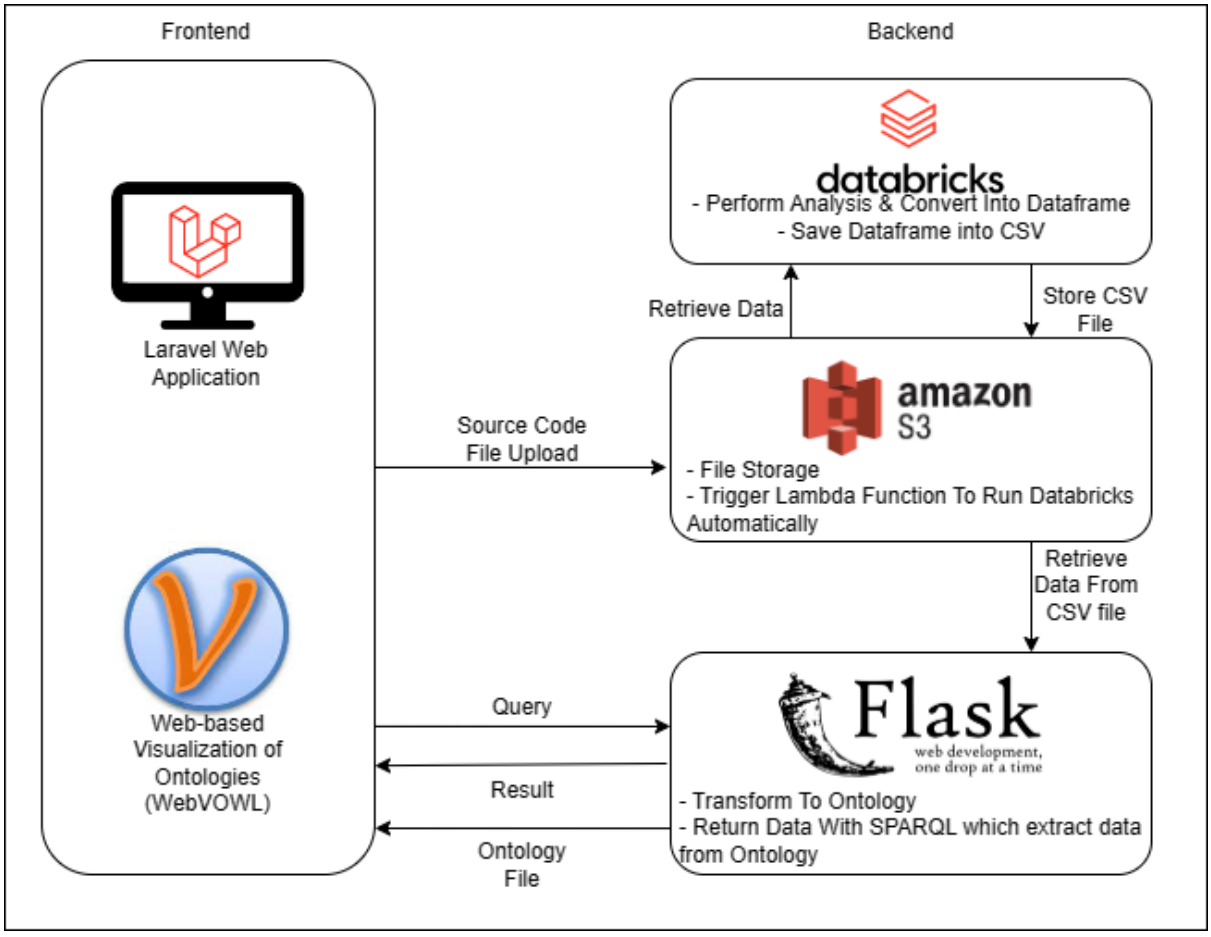


Figure 5.1: Overview of the System Architecture Design

Figure 5.1 shows an overview of the implemented system's architecture design. The system consists of one application which is a Laravel Web application. By sending a request to the Flask API in order to execute the query to retrieve the data, the Web application will be in

charge of the source code file upload and displaying the results. Once the files have been uploaded via the Laravel Web Application, the controller of Laravel will store them in Amazon S3 Storage and then launch an AWS Lambda Function to run Azure Databricks Notebook, which is used to perform the Extract, Transform, and Load process of the source code and produce dataframe output that will be written into a CSV file and stored in Amazon S3 Storage.

Following the upload of the CSV file containing the source code information to Amazon S3 Storage, while activating the Flask API, it will retrieve the information from the CSV file from Amazon S3 Storage using the built-in Python library and create an ontology using an external library called RDFLib that was imported. Following the creation of the ontology, the user can retrieve data by using a predefined SPARQL query in Flask or by searching through the class name, which the Flask API will accept as input and work in conjunction with to retrieve data about the class name input by the user. Last but not least, the user has the option to download various ontology file formats that will be uploaded to Web-based Visualization of Ontologies (WebVOWL) in order to produce an ontology visualization that will help the user comprehend the source code more clearly.

5.2.1 Laravel Framework Architecture

The Laravel Framework is a PHP web framework that is open-source and free. It is used to create sophisticated online applications. It is based on the architectural design pattern known as Model-View-Controller (MVC). There are many features in Laravel. These are listed below:

- Modularity: Laravel includes a large number of frameworks and modules that aid developers in creating PHP web applications that are responsive and modular. Additionally, this feature expedites development.
- Eloquent ORM: Object Relation Mapping is an acronym. Eloquent, a built-in ORM in Laravel, manages database-related tasks.
- Artisan: Laravel's command-line interface is called Artisan.
- Blade Templates: The Blade concept from Laravel generates a distinctive template to display data.
- Unit Testing: In Laravel, unit testing can be carried through using test cases.
- Email support: A built-in class in Laravel named Mail facilitates email sending.

- Authentication: The system's users are identified through authentication. It is typically accomplished by knowing the user's account and password.

Model-View-Controller (MVC) is an architectural design pattern used by Laravel that divides an application into three primary parts: Model, View, and Controller. The organization, maintainability, and scalability of the code are all improved by the separation of concerns.

- Model: The data and business logic of the application are represented by the model. The database is accessed, data is retrieved, stored, and operations are carried out on the data. Models in Laravel usually reside in the 'app' directory and extend the Eloquent ORM (Object-Relational Mapping) framework to streamline database interactions. Models offer techniques for data manipulation and querying while defining the structure of database tables. Eloquent's relationship methods, such as ‘hasOne’, ‘hasMant’, ‘belongsTo’, and others, are used to define relationships between models. CRUD (create, read, update, delete) activities on the database can be carried out using the model instance.
- View: Data presentation to the user is the responsibility of the View. It includes the UI elements that consumers view in their browsers, layout files, and HTML templates. Views in Laravel are normally kept in the directory ‘resource/views. The final HTML that is sent to the user's browser is produced using views. Laravel creates dynamic views using the Blade templating engine. Blade enables the usage of reusable components, template inheritance, and control structures like "@if" and "@foreach."
- Controller: Input from users is processed, models are interacted with, and data is prepared for the Views by the Controller. It functions as a go-between for the Model and the View, deciding how the program will behave based on user interactions. The directory 'app/Http/Controllers' has the controller files. A controller's methods each indicate a particular route or action that the application is capable of handling. Models are used by the controller to fetch or alter data as needed after receiving input from the request (such as form data). Following processing, controllers provide the user with responses in the form of views or JSON.

5.2.2 Ontology Design

Carefully envisioning, defining, and arranging the concepts, relationships, and attributes inside a given domain are necessary while designing an ontology. In order to improve

understanding, sharing, and reasoning about domain-specific information, ontologies are used to codify and express knowledge in a machine-readable fashion. There are some steps to guide through designing an ontology:

1. Clearly identify the ontology's scope and purposes: Outlining the domain, concepts, and knowledge we wish to represent.
2. Identify Important Concepts and Classes: List the important ideas, groups of ideas, or entities that are important to the domain. This could be things, intangible concepts, actions, or physical things.
3. Establish Relationships: Identify connections between the concepts. Relationships show the connections between concepts.
4. Define Properties: Specifying the properties of attributes that describe the concepts. Properties (values) can be data properties or object properties (link to other concepts).
5. Choose Ontology Language: Pick an ontology language based on the requirements. OWL and RDF are popular choices.
6. Create Class Hierarchy: Create a hierarchical structure (taxonomy) for concepts to reflect relationships between subsumption and specialization. The ontology is better organized and easier to understand because to this hierarchy.
7. Apply Domain Vocabulary and Standards: Use current standards, ontologies, and domain vocabularies as appropriate. This makes sure that the terminology is consistent and compatible.
8. Use Cases and Instances: Determine possible ontology applications and think about how specific instances (examples of a concept) might fit into the architecture.
9. Consider Restriction and Constraints: To encapsulate the logical principles and limitations that apply to the domain, define constraints and axioms.
10. Reuse and Extend Ontologies: In order to save time and improve interoperability, it may be appropriate to repurpose or augment existing ontologies.

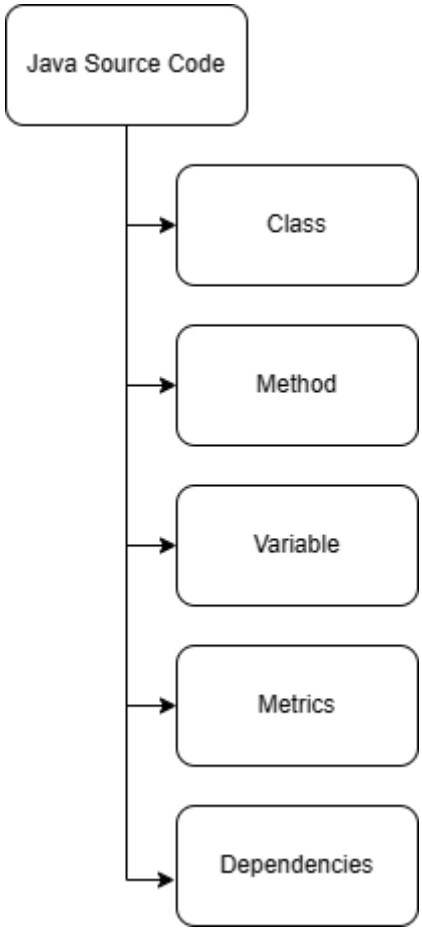


Figure 5.2: Source Code Components Ontology

Before any analysis of the source code analysis to be done, we need to design the source code ontology schema by using Protégé, which is an ontology editor, in order confirm the data structure and the analysis method to be done. As shown in Figure 5.2 above, the Java source code has 4 different concepts which are Class, Method, Variable and Metrics. For example, A Java source code has different classes which are User, UserController and etc. Inside the classes it contains different method such as getUserInfomation, getUserName and etc. Other than that, those classes also contain the variable such as username, user_age and etc. Moreover, some of the classes may have dependent on the other classes' method, for example, in the UserController class, it calls the getUserName function declare in User class, which represents a dependent relationship between these 2 classes. Last but not least, the whole Java Source Code has several metrics such as Total Line of Code (TLOC), Number of Class, Number of Method and etc.

With the concepts declare, we are now able to define the object properties which link the concepts together. An object property is a fundamental concept used to describe relationships between individual (instances) or classes (concepts) in a domain. Object property define how different classes are connected or related to each other. These relationships play a crucial role in modelling the structure and semantics of a domain's knowledge. For example, UserController **hasFunction** getUserName. The **hasFunction** is a property which shows the relationship between the UserController (domain) and getUserName (range).

To ensure that the general class and attributes across classes are defined consistently and categorised appropriately, the planned ontology has to be checked. An ontology is verified to make sure it is accurate, consistent, and in line with its intended uses. The goal of ontology verification is to find mistakes, contradictions, and problems within the ontology in order to verify its accuracy and dependability. Using the Protégé software's HermiT Reasoner, pick the HermiT Reasoner engine from the reasoner menu to carry out the verification procedure. The HermiT Reasoner computes the results and displays a hierarchy when the verification is required. The incorrect result is displayed to amend or change the class hierarchy or the axioms in the ontology, as shown in Figure 5.3, if the classification is inconsistent or incorrect. As a result, following each execution of the reasoning functions in Protégé, the ontology engineers must always examine the ontology and make any necessary modifications.

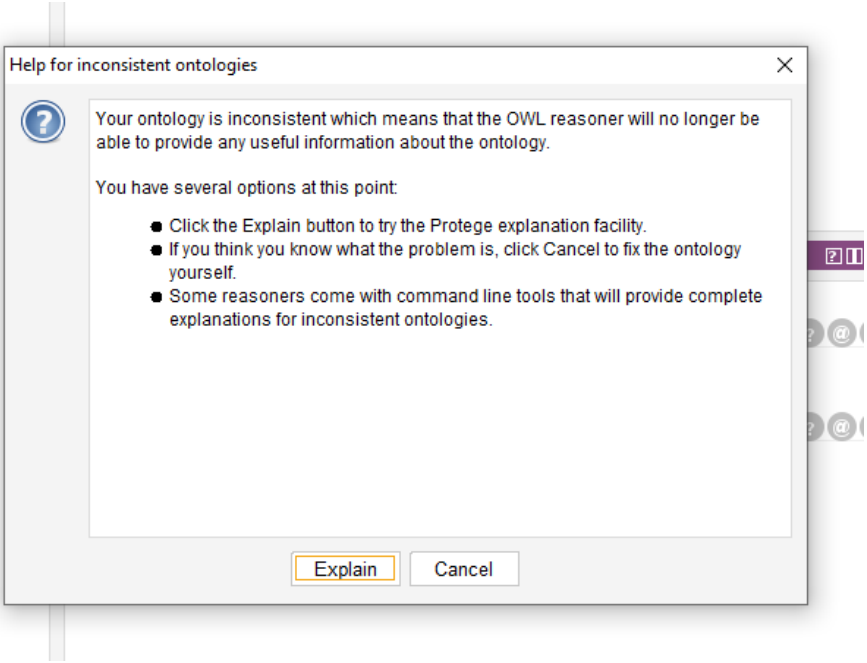


Figure 5.3: Error prompt when the inconsistent of the ontology appear

5.3 Conclusion

This chapter concluded by giving a general summary of the system architecture and ontology design that were employed in this project. The Laravel framework and the cloud services used by the system were both described in the system architectural design. Using the data output produced by the Databricks cloud platform, the ontology design will then be used to build the ontology idea. Overall, by outlining a clear and well-organized system architecture and ontology design, this chapter set the groundwork for the creation and application of the system.

CHAPTER 6
SYSTEM IMPLEMENTATION

6.1 Introduction

An overview of the project's setup and system implementation is given in this chapter. Each system component, such as a web application, a cloud-based analysis platform, and visualization tools, is described in detail. The use cases and requirements specifications addressed in Chapter 4 are the basis for the system modules' design. This chapter further details the characteristics and functionalities of each module to help the reader understand it better.

6.2 Project Setup

The first step in this project is to register for an Azure account at https://signup.live.com/signup We can subscribe to a service on Azure using the account. The service we'll be using for this project is Azure Databricks, which will take care of the extraction, transformation, and loading of the user-uploaded source code. Microsoft Azure offers Azure Databricks, an analytics platform built on Apache Spark, as a fully managed service. Microsoft and Databricks, the business formed by the Apache Spark developers, are working together on it. With the help of Azure Databricks, businesses will be able to handle and analyze enormous datasets in a more effective and scalable way. It offers a unified platform for data engineering, data science, and big data analytics.

To use the S3 and Lambda services, which enable us to store files on cloud storage and make them retrievable via Azure Databricks and mount as a DBFS under Azure Databricks, we must first register an Amazon account. Create an account at https://portal.aws.amazon.com/billing/signup#/start/email to get started. Popular cloud-based object storage is offered by AWS's Amazon S3. For many types of data, including files, photos, movies, backups, and more, it provides scalable storage. S3 is made to offer great performance, availability, and durability for online data storage and retrieval.

The next step is to decide on a framework for our web application. Laravel framework is what we use for our speedy and affordable development process. Installing a WAMP server and Composer are two of the technologies required to use the Laravel framework. On a Windows operating system, a WAMP server is a software stack used for web development. It offers a setting for locally running web apps on our machine for

development and testing needs. In addition, Composer, a tool for managing dependencies, is essential for managing the libraries, packages, and dependencies needed for our application. It makes it simpler to add, update, and manage external packages and libraries, which facilitates the development and maintenance of our Laravel projects. In addition, with Composer installed, we can use the command "composer global require laravel/installer" to download the Laravel installer to my machine worldwide. With this command, the laravel/installer package will be downloaded and installed in a PATH-compatible location on my system. In my command-line environment, I can now create new Laravel projects by using the 'Laravel' command from any location.

To create a new Laravel project, we can run “laravel new <project-name>” or “composer create-project laravel/laravel=8.* <project-name>” to configure which laravel version to use. Running these commands will create a new project with the specified name and all the necessary files and dependencies.

Next is to configure the database and environment credential. Using the WAMP server, it provides a MySQL database which hosting in 127.0.0.1 with a port 3306 in default. The only thing we need to do is to create a database and replace the name of the database in the project’s env file.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=fyp
DB_USERNAME=root
DB_PASSWORD=
```

Figure 6.1: Database Configuration In .env File

6.2.1 AWS S3 Setup

1) An AWS account is needed to set up a S3 Bucket.
2) Once a AWS account is prepared, go to https://s3.console.aws.amazon.com/s3/get-started?region=ap-southeast-1&region=ap-southeast-1 to create a new S3 bucket by clicking the “Create Bucket” button in the S3 console, as shown below.

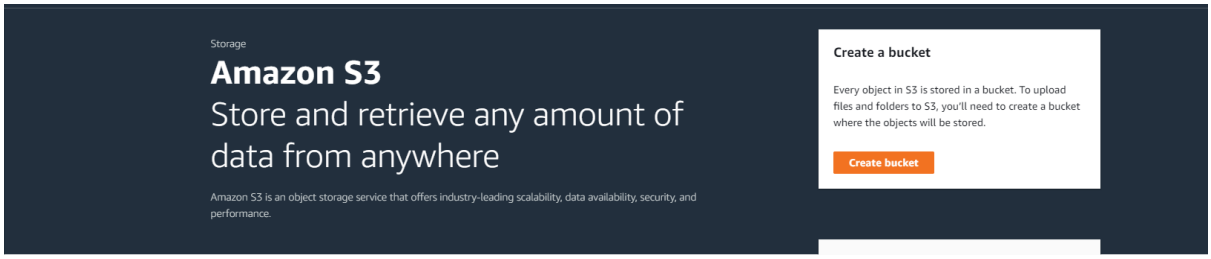


Figure 6.2: Amazon S3 Console

3) Follow the steps given by the S3 console to complete the bucket creation process.
4) After creating the S3 bucket, now we need to create a user in order to connect to the S3 bucket with the credentials and access key given.
5) Navigate to https://us-east-1.console.aws.amazon.com/iamv2/home?region=ap-southeast-1#/users to create a new user as shown below.

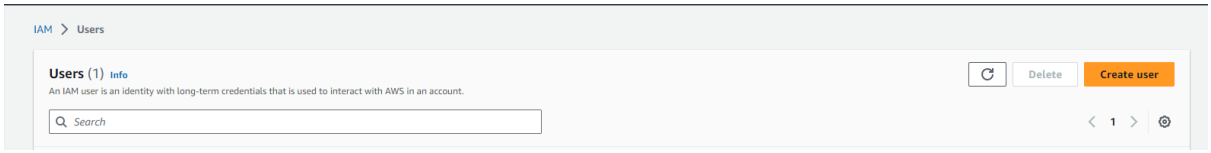


Figure 6.3: Create User Console

6) Follow the process of user creation, and in the "Set Permissions" process, select "Attach Policies Directly" and search "s3" in the search bar bellow, and select "AmazonS3FullAccess" and click next.

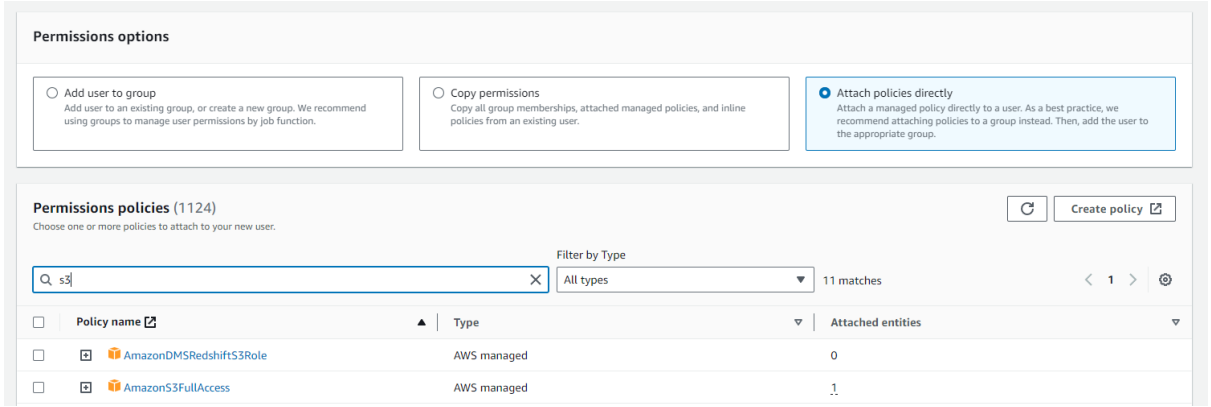


Figure 6.4: Set Permission Pages

7) Click on the "Create User Button" a new user will be created, next we will generate an access key based on the user created.

8) In the Users dashboard, select the user we have just created and navigate through the "Security Credentials" tab which shown in below.

Figure 6.5: Security Credentials Tab

9) Scroll down and find the access key category and click on the "Create Access Key" button.

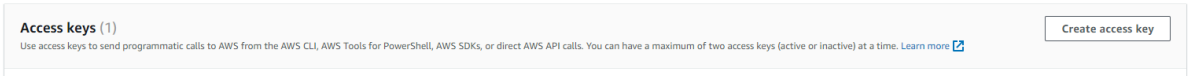


Figure 6.6: Create Access Key Button

10) Select the use case of the access key will be use and generate the access key.

11) Get the access key and secret access key, then change them in the Laravel project's env file.

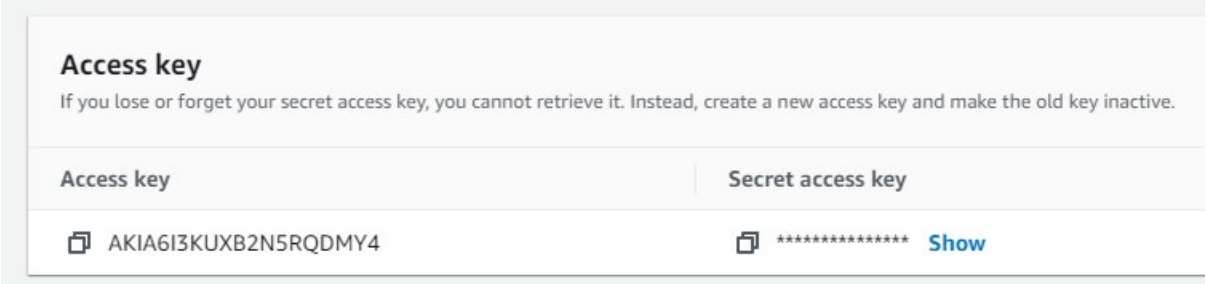


Figure 6.7: Retrieving Access Key

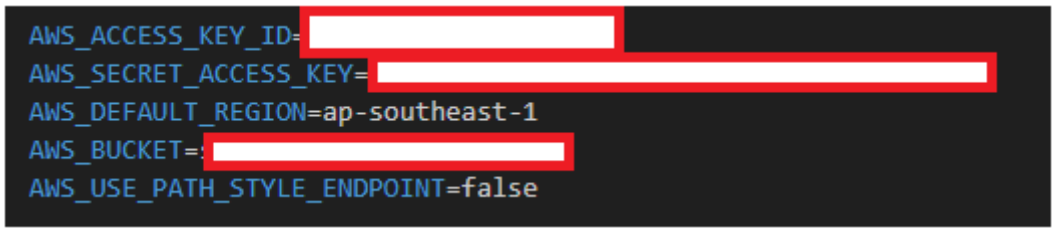


Figure 6.8: Fill in the information from S3 Bucket and Access Key Credential

6.2.2 Azure Databricks Workspace & Workflows Setup

1) An Azure account is needed to setup the Databricks Workspace & Workflows.

2) Once the Azure account is prepared, login into the account and search Azure Databricks, select and create a workspace in Azure Databricks console.

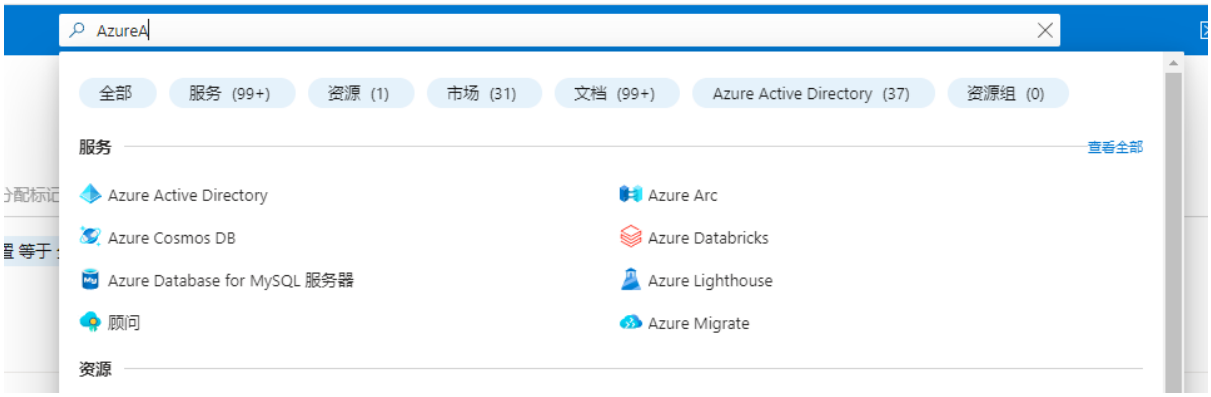


Figure 6.9: Search Result in Azure Portal

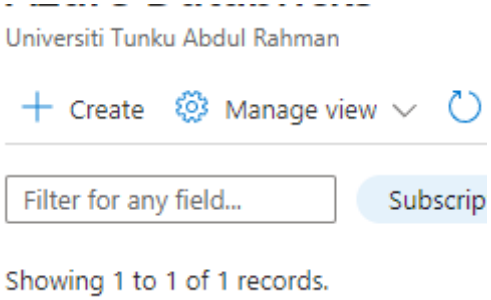


Figure 6.10: Create A New Azure Databricks Workspace

3) Follow the guidelines and procedure to create a new workspace with some configuration. Once the workspace is created successfully, launch the workspace.

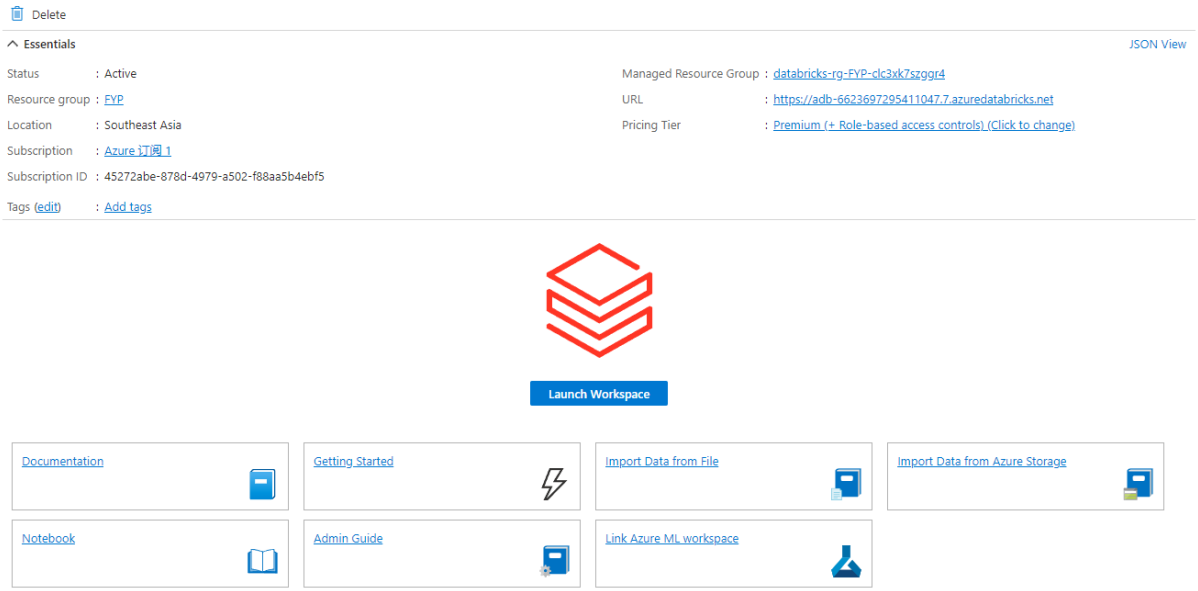


Figure 6.11: Launching the new created workspace

4) After launching the workspace, create a new notebook which used to perform analysis of the source code later.

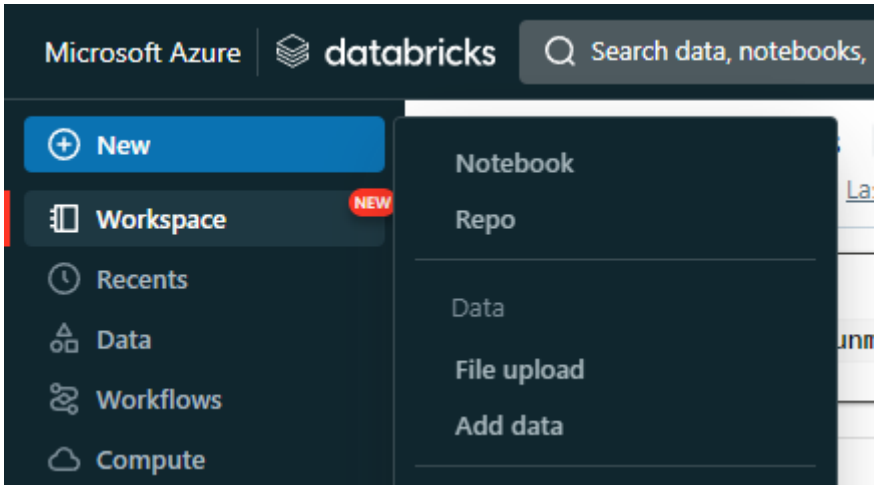


Figure 6.12：Creating a new notebook

5) Other than that, to perform the analysis action, a cluster will be needed to execute the notebook, hence we will need to create a compute cluster in order to execute the notebook by clicking on the compute button in the navigation bar.

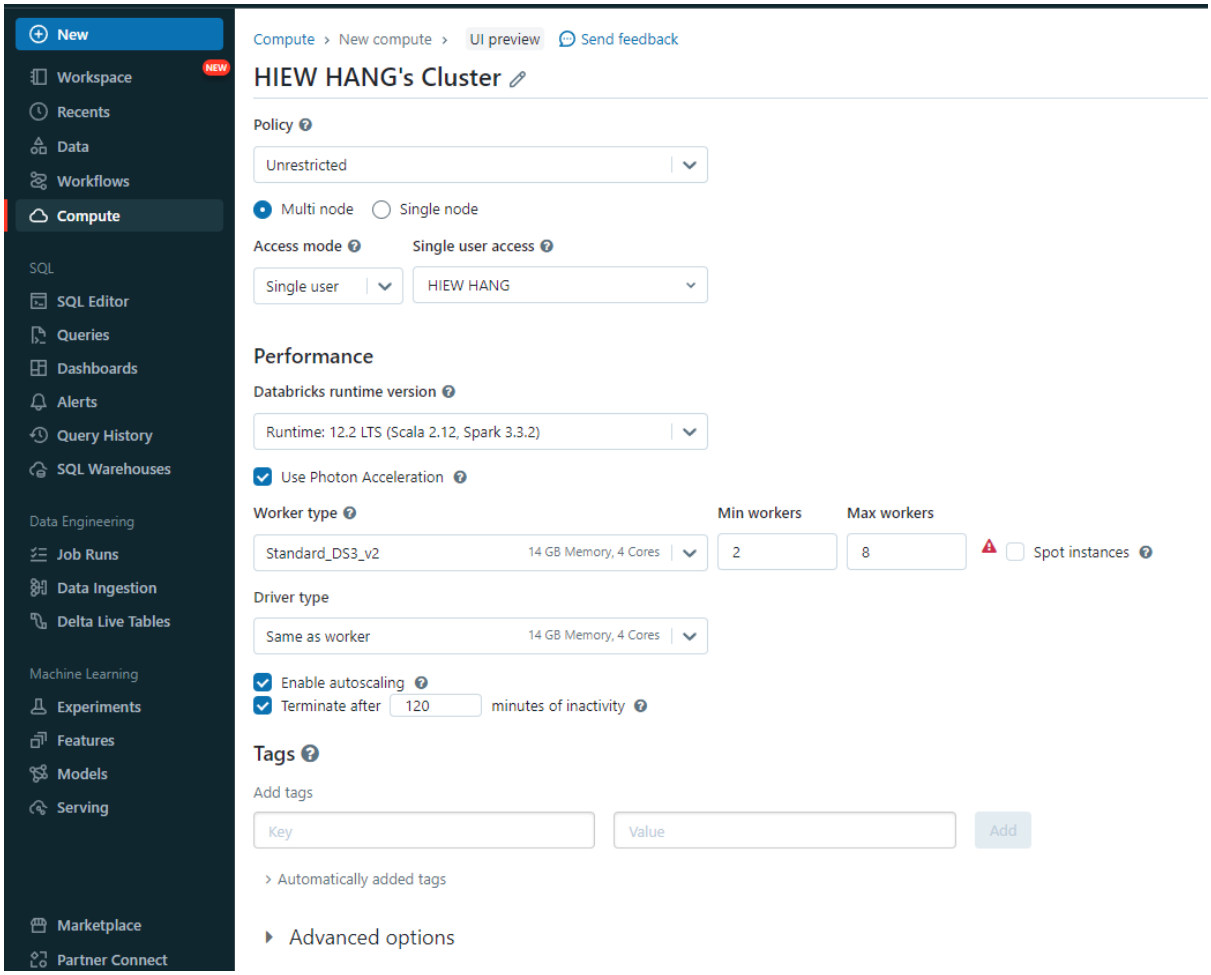


Figure 6.13: Creating New Compute Cluster

6) Next, attach the created cluster into the notebook by clicking the connect button locate on the top right of the notebook and start the cluster when executing the notebook.

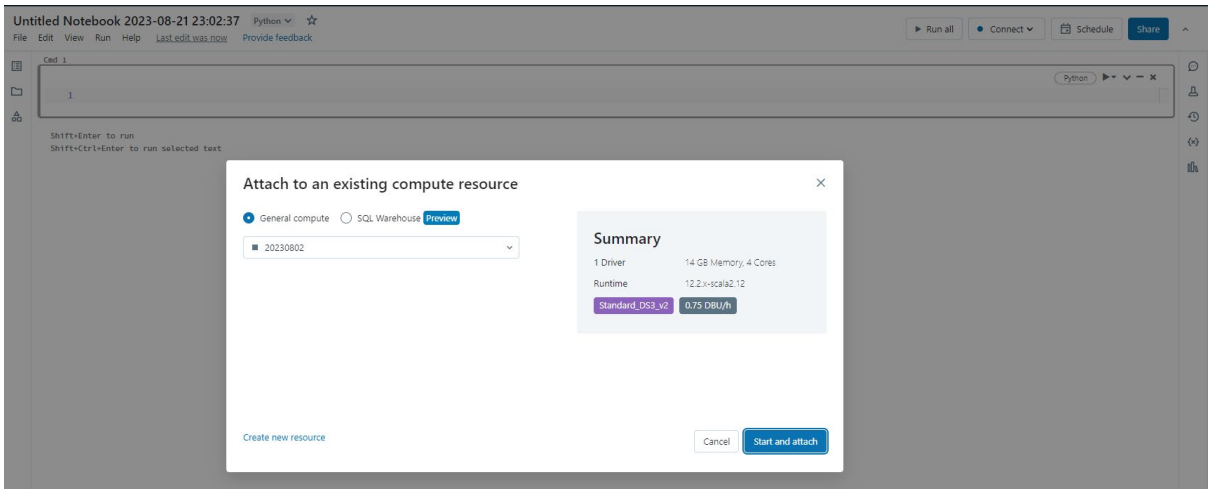


Figure 6.14: Attach Cluster to a notebook

7) To automate the executing of the notebook, we will be using the workflows' function which provide by the Azure Databricks, click on the Workflows button located in the navigation bar and select create job.

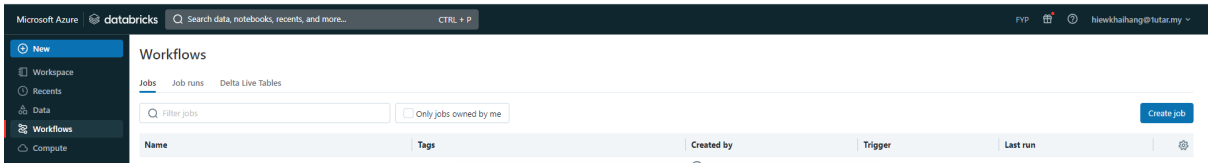


Figure 6.15: Workflow Console

8) Fill in the necessary details and configuration options and the most important is to select which path (notebook) to execute when a trigger event happens.

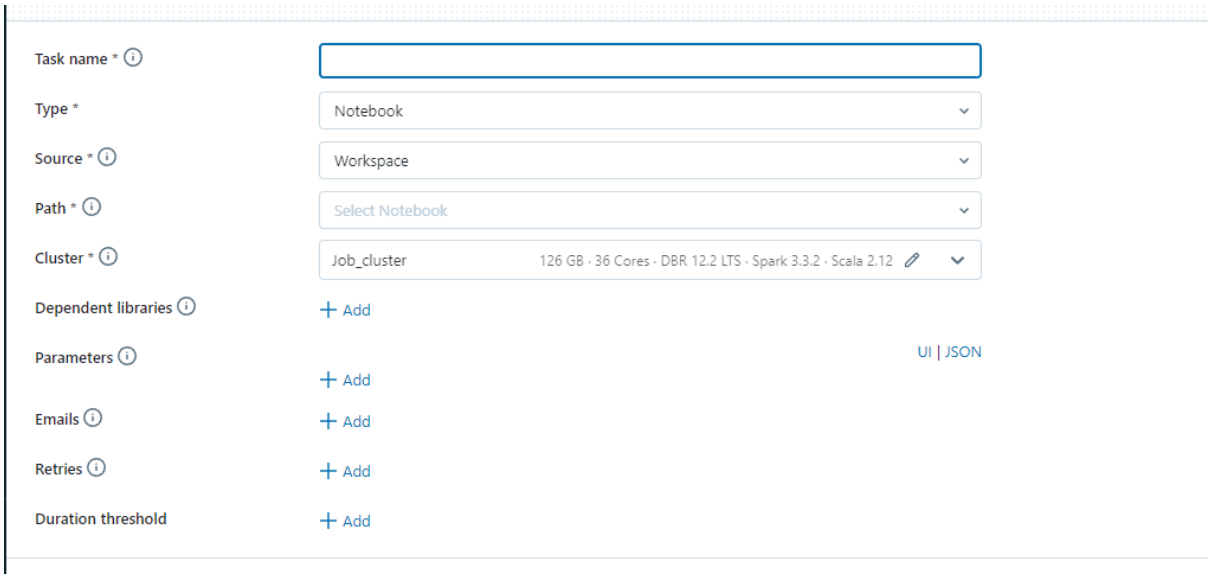


Figure 6.16: Create Automation Jobs with Databricks Workflows

9) Once the job created successfully, note down the Job ID declare in the job details category which will be used in the configuration of AWS Lambda Function.

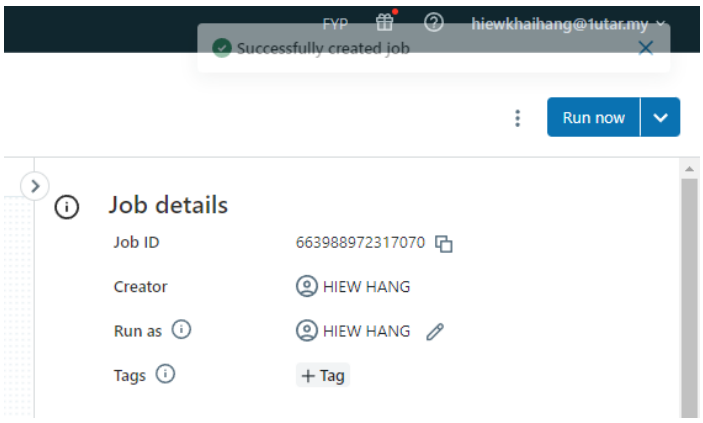


Figure 6.17: Job Details

10) Other things to mentioned to make the automation of the execution of notebook work successfully, is to generate a access token which used for authorize purpose when calling Databricks API, navigate through “User Settings” in the dropdown button which located in the most top right of the page.

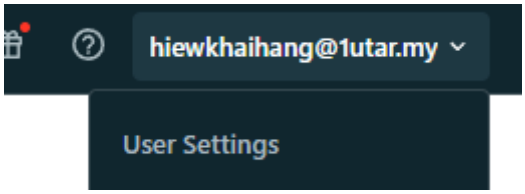


Figure 6.18: User Settings Button

11) Click on the “Developers” tab and find Access Token category and click “Manage”

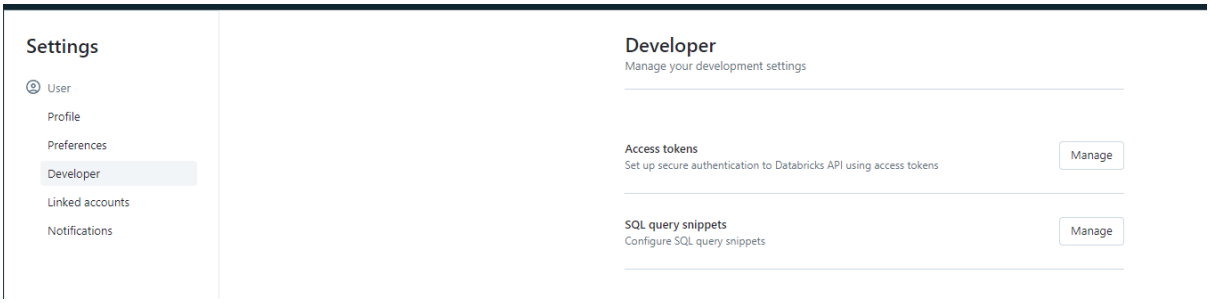


Figure 6.19: Access Token Category in Developers Tab

12) Click on the “Generate new token” button.

13) Enter the comment which used to represent the function of the token and tune the lifetime of the token.

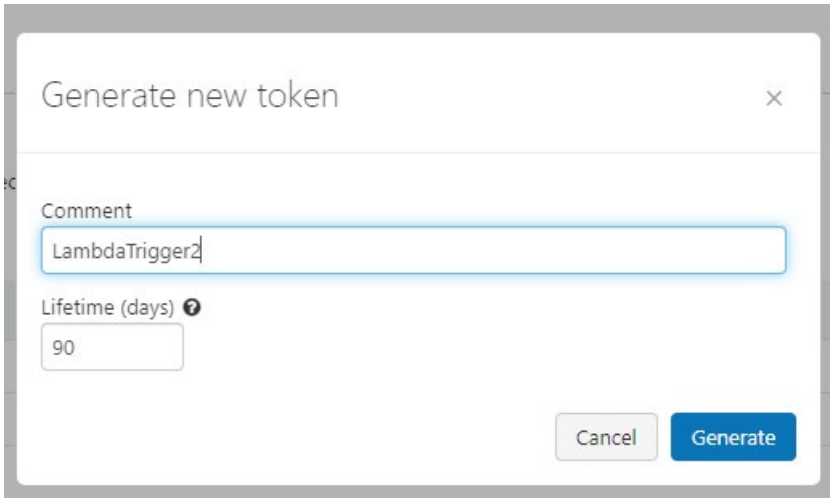


Figure 6.20: Generating a new token

14) Click on the Generate Button and take note on the token generated which will be used in the lambda trigger function.

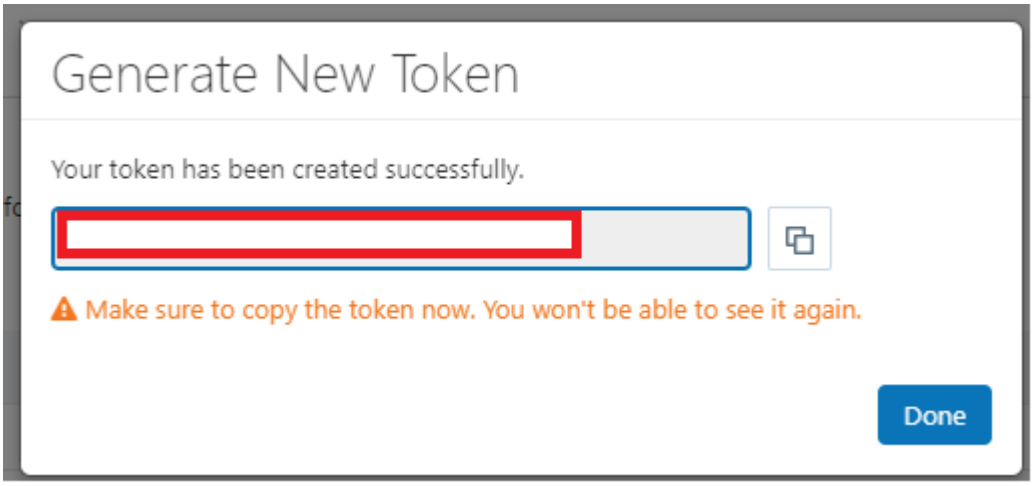


Figure 6.21: Token generate successfully

6.2.3 AWS Lambda Function

1) Using the AWS account created in section 6.2.1, login and navigate through the AWS Lambda dashboard with this URL: https://ap-southeast-1.console.aws.amazon.com/lambda/home?region=ap-southeast-1#/discover
2) Select the region which located beside the username drop-down button. Make sure the region is same with the region of the S3 bucket, if they are in different location, the lambda function could not function properly to get the trigger event from the S3 bucket.
3) Select the "Create Function" button located at the top right corner of the dashboard.

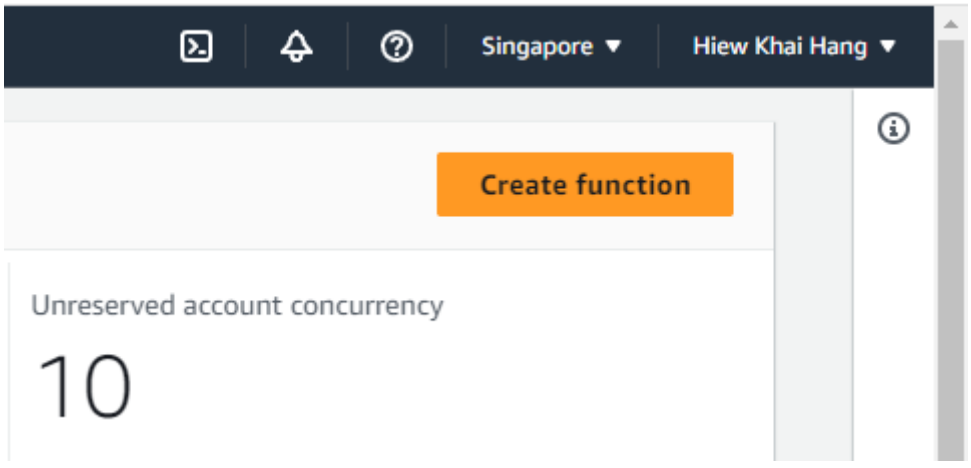


Figure 6.22: Create Function Button

4) Fill in the necessary information and select the runtime environment (Node.js 14.x) and using the x86_64 architecture as the configuration setting of the lambda function.

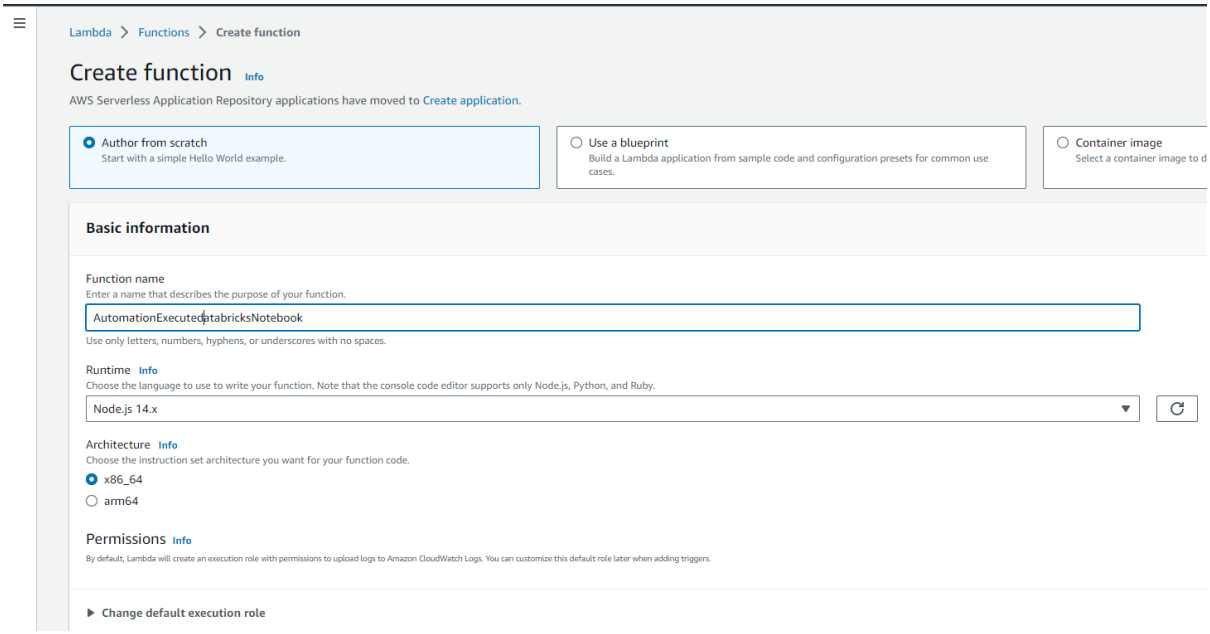


Figure 6.23: Creating Lambda Function and Configuration

5) After the Lambda Function created successfully, click on the function just created, scroll down and find the “Code” tab which will need to be execute when a trigger event happens.

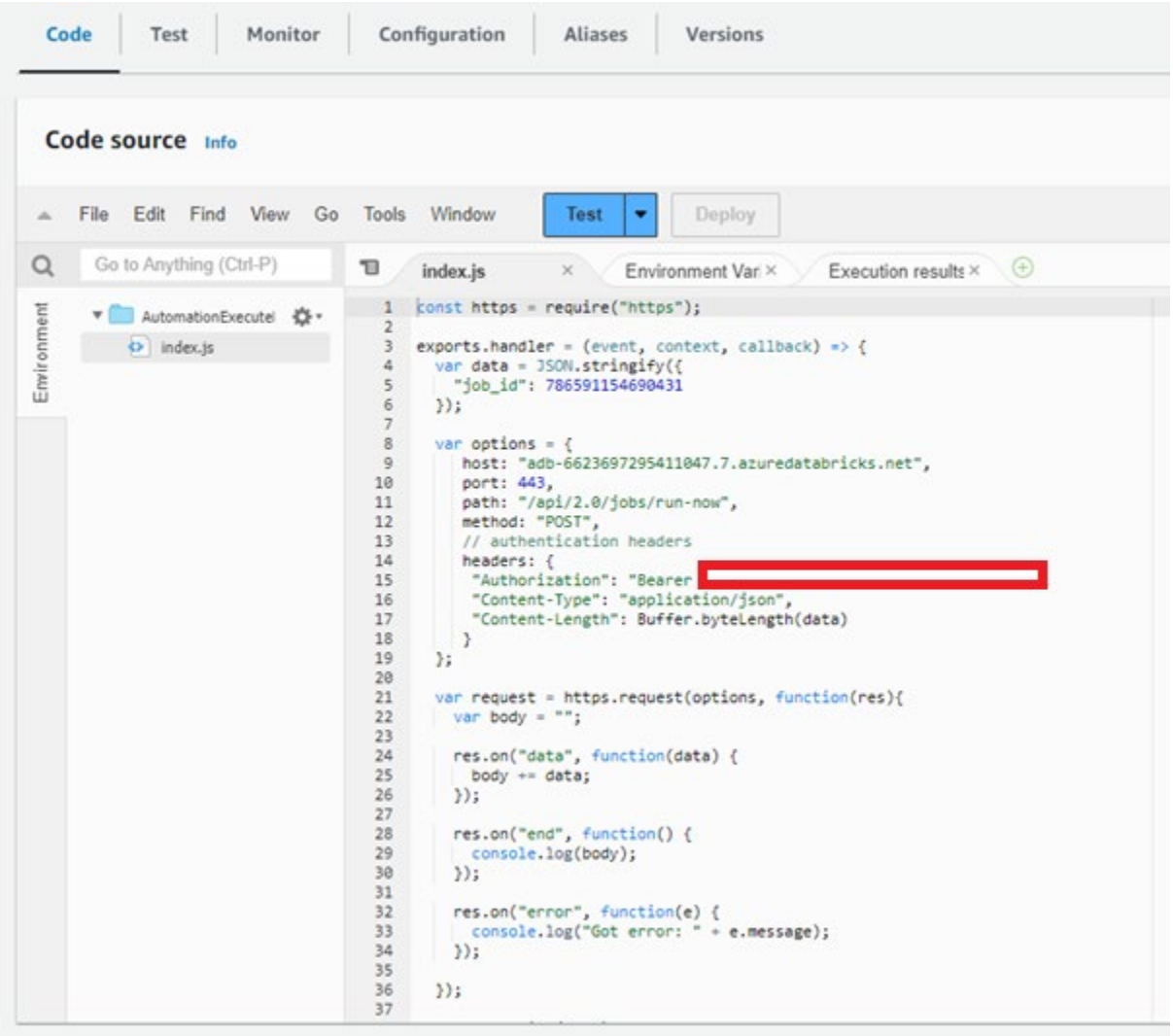


Figure 6.24: The Code to be execute when an event triggers this lambda function

6) Configure the code above, update the job_id with the corresponding job_id noted in the section 6.2.2 and change the Authorization header with your bearer token created in section 6.2.2 the created access token in databricks.
7) Next is to add a trigger event to call this lambda function, scroll up and find a “Add Trigger” button and select the source as S3 since we are using S3 bucket to trigger the function.

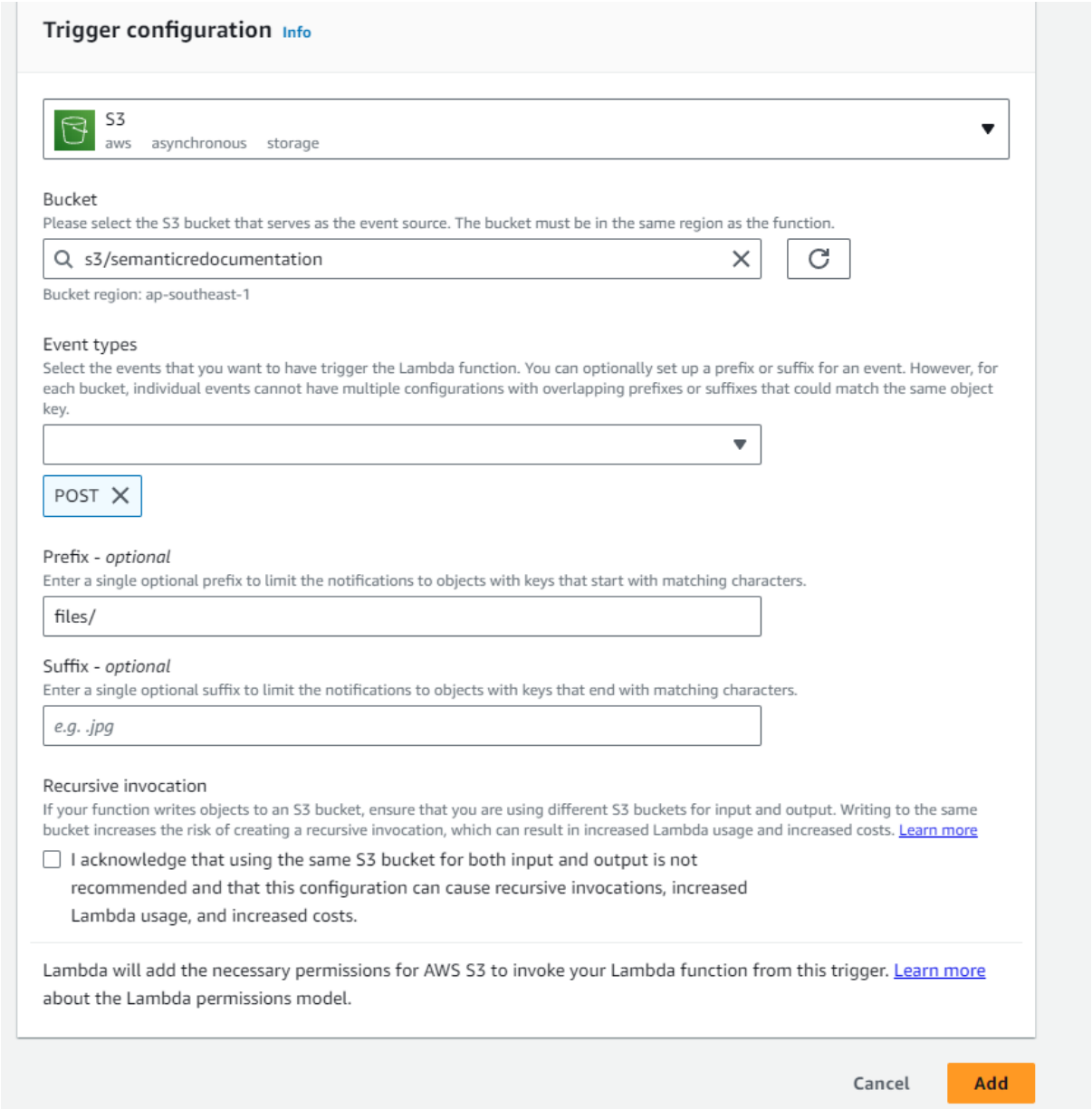


Figure 6.25: Trigger Event Configuration

8) Fill in the necessary information and condition to trigger this function. In my case, when a post action is occurred in files/ folder, this function will be triggered to call the Databricks API and execute the corresponding workflow job declare in the Databricks.

6.3 System Modules

This project only features one application, a web-based application which separate in front-end (client-side) using Laravel to render and sending request to the backend through RESTful API, and the backend (server-side) which we are using Python Flask to construct ontology

and retrieve data from it. Other than that, Laravel also took responsibility on upload the file by the user and storing it on the Amazon S3 bucket.

6.3.1 Extraction Module

First, the user uploads the source code through the web application and the controller handles the upload process.

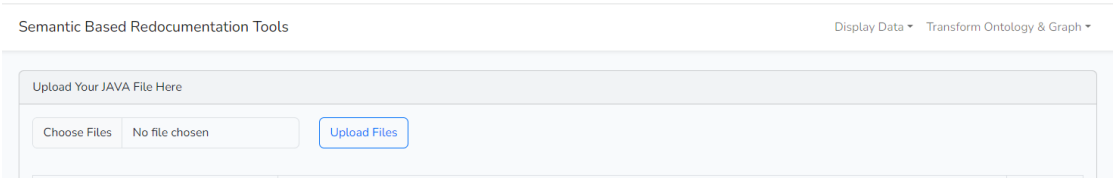


Figure 6.26: File Upload Page

Inside the controller, it will validate the file type of the user upload. Due to our analysis is suitable only for java source code, other type of source code will not be accepted when uploading the file.

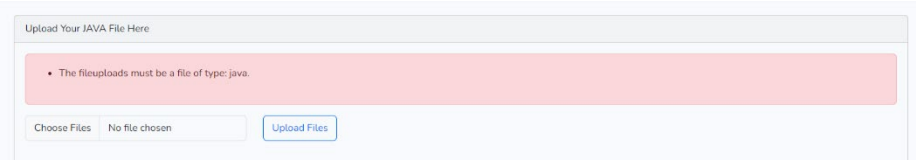


Figure 6.27: Warning Message Display when validation failed

The user can select a bunch of files and click on the “Upload Files” button, and the controller will take action saving it into the S3 storage.

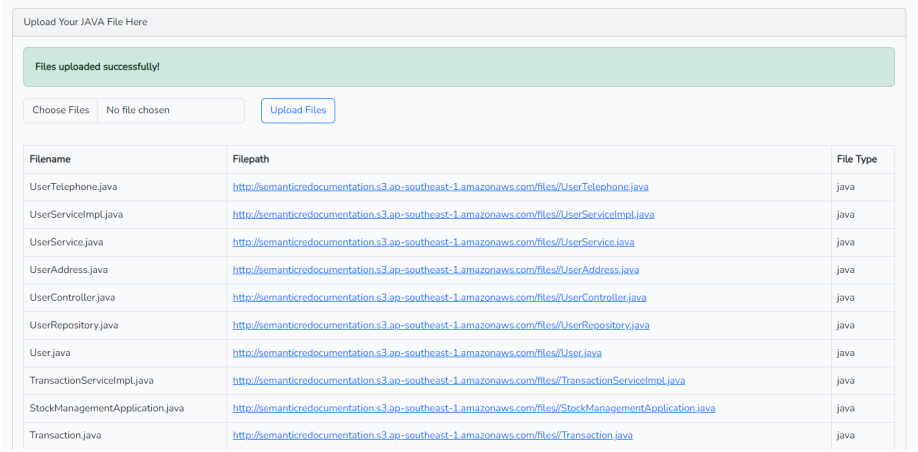


Figure 6.28: The Message Display After File has successfully uploaded

```php
public function multipleUpload(Request $request)
{
    $this->validate($request, [
        'fileuploads' => 'required'
    ]);
    $files = $request->file('fileuploads');
    foreach($files as $file){
        if($file->getClientOriginalExtension() != "java"){
            return redirect()->route('fileupload.index')->with('failed','Invalid File Extension');
        }
        $fileUpload = new File;
        $fileUpload->filename = $file->getClientOriginalName();
        $path = Storage::disk('s3')->putFileAs('files/', $file, $fileUpload->filename);
        // $path = $file->store('public/uploads');
        $fileUpload->filepath = Storage::disk('s3')->url($path);
        $fileUpload->type= $file->getClientOriginalExtension();
        $fileUpload->save();
    }

    return redirect()->route('fileupload.index')->with('success','Files uploaded successfully!');

}
```

Figure 6.29: Code Segment of Upload File Controller

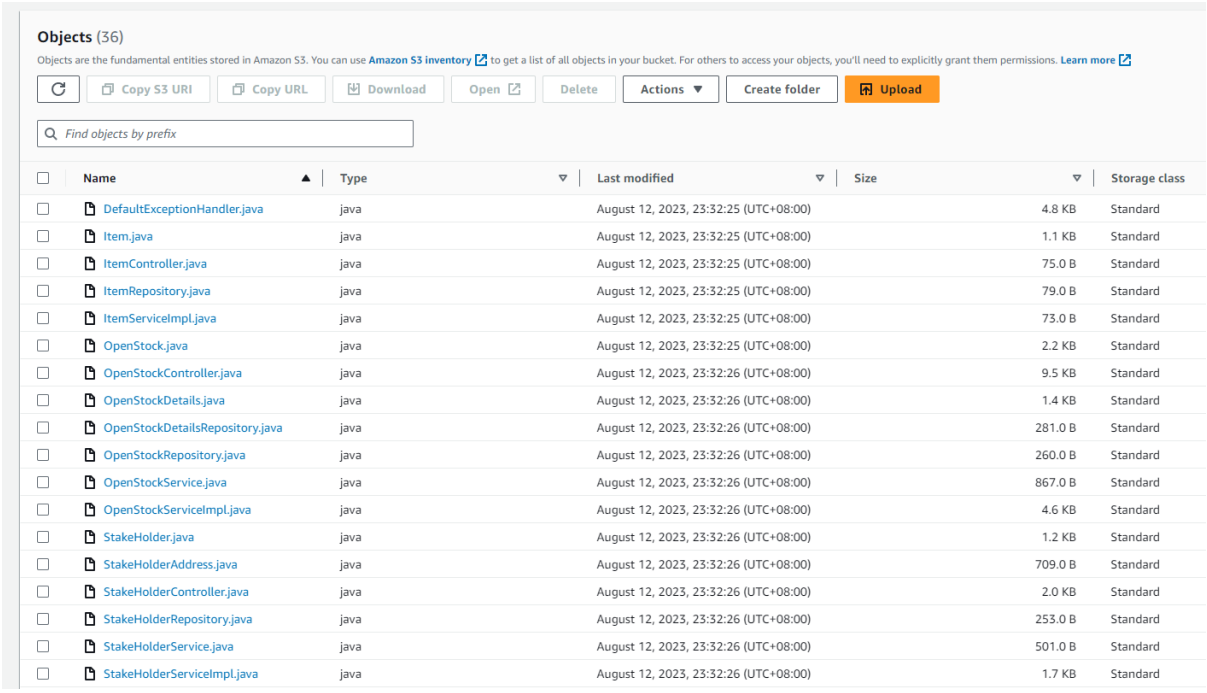


Figure 6.30: S3 Bucket Folder

After storing the files into Amazon S3 Bucket, a AWS Lambda Function will be triggered to execute a Databricks notebook which act as our cloud analysis platform to perform analysis and generate CSV file and store into Amazon S3 bucket again. This function will be trigger when a folder name files/ has store into the S3 bucket.

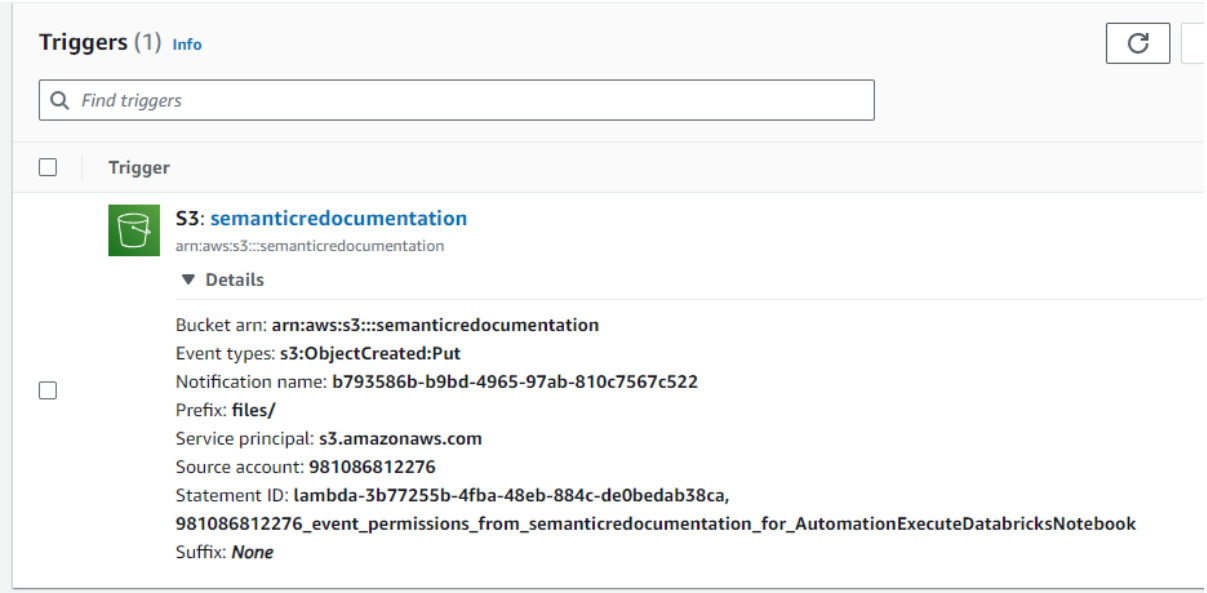


Figure 6.31: Amazon Lambda Trigger when S3 Bucket has a folder name files/

When this function is triggered, it will execute the scripts below which will call the Databricks API to execute the corresponding Databricks workflow with the bearer token as authorization and the job_id which created in Databricks.

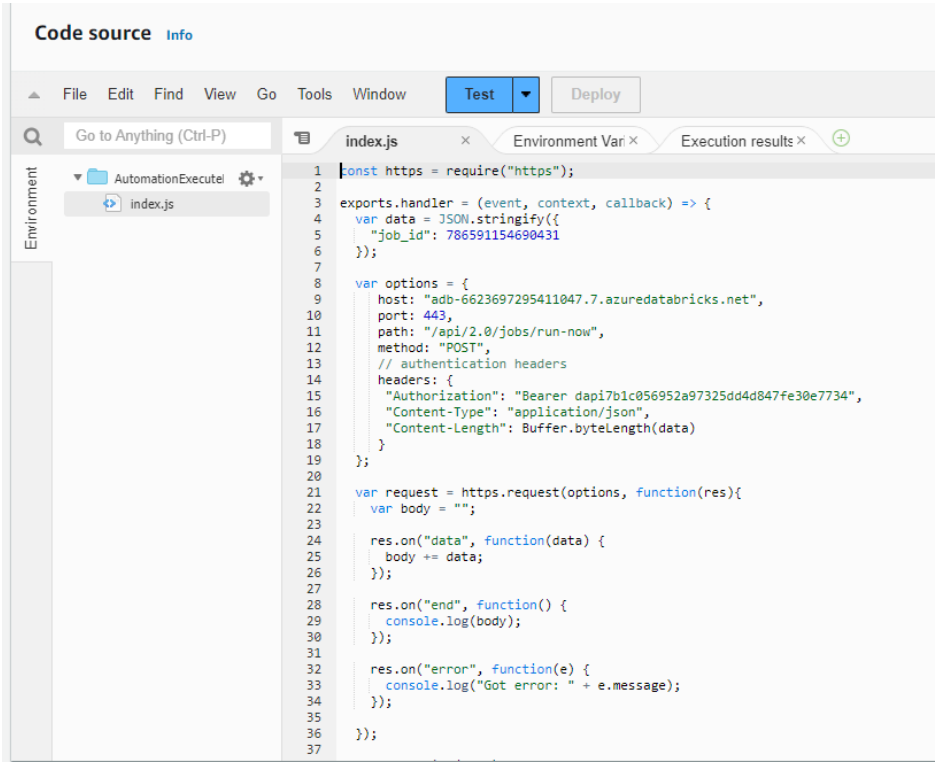


Figure 6.32: The script to be run when lambda function is trigger

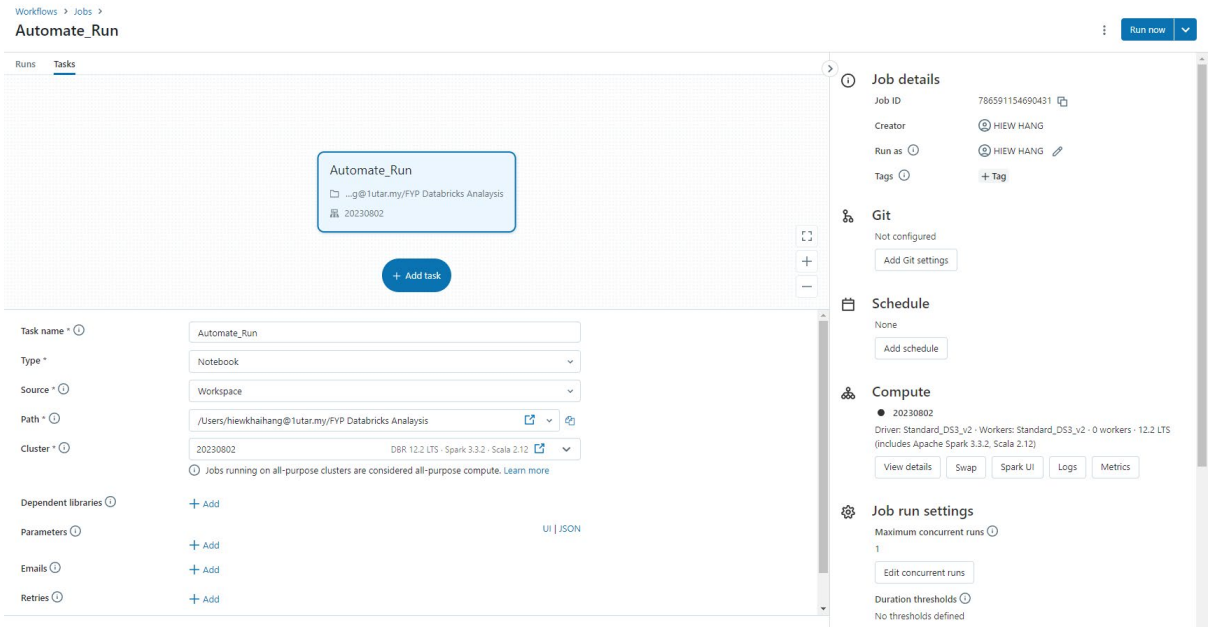


Figure 6.33: The workflow to be triggered when the Databricks API is called

After the Databricks notebook execute, the command inside the notebook will be execute follow by the sequence. The First command will be the mounting of the S3 storage to the Databricks DBFS, which allow us to retrieve the file and content after mounting the S3 into DBFS directly.

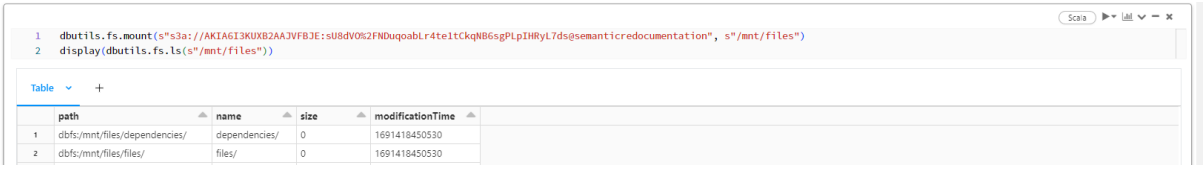


Figure 6.34: Mounting a S3 bucket into Databricks DBFS

6.3.2 Transformation Module

A unified data analytics platform called Databricks was created to assist enterprises in processing, analysing, and visualizing massive amounts of data. Big data processing, machine learning, and advanced analytics all frequently use it. A collaborative platform that interfaces with numerous technologies and data sources is offered by Databricks.

Scala analysis could be performed on the Databricks platform after the S3 storage was mounted into the DBFS. One of the programming languages that Databricks supports for creating code on the platform is Scala. The Java Virtual Machine (JVM) supports the flexible

programming language Sclaa, which is renowned for its object-oriented and functional characteristics. Given that it works especially well for distributed processing workloads, large data frameworks like Apache Spark frequently use it.

Extracting the source code from the files is the initial step in beginning the study of the source code. As seen in the figure below, you may get the entire content of the source code files saved in the DBFS as a Resilient Distributed Dataset (RDD) by utilizing the textFile functions offered by the SparkContext class. Every file in the directory will have its contents line by line read. After extracting the material is finished, the source code content is further transformed to aid in the analysis by pre-processing the data before the real transformation. The first step is to use Scala's map function to change every element in the RDD to lowercase, making analysis easier. A higher-order function called map is used frequently with collections in Scala to apply a given transformation function to each element of the collection, creating a new collection with the modified items. Next, leading and trailing whitespace, including spaces, tabs, and newline characters are removed from a string using the trim function supplied by Scala's String class. Pre-processing is completed in its final step, which involves filtering out extraneous lines of source code that contain comments and unclean lines that will impact the analysis process later.

```
import spark.implicits._
import org.apache.spark.sql.functions._
import com.databricks.spark.xml._

//*************************************************************************************************************************************************************************
// Pre-processing

val firstRDD = sc.textFile("/mnt/files/files")
var className = ""
var interfaceName = ""

val lower=firstRDD.map(x=>x.toLowerCase)
val split=lower.map(x=>x.trim)
val TLOC = split.count
val preprocess=split.filter(!_.contains("//")).filter(!_.contains("/*")).filter(!_.contains("***")).filter(!_.contains("import")).filter(!_.contains("package")).filter(!_.contains
("this")).filter(!_.contains("system.out.println")).filter(!_.contains("50.0")).filter(x => x != "").filter(!_.contains("requestmapping")).filter(!_.contains("interface"))
```

Figure 6.35: Pre-processing of the data

The map method is then used once more to construct a multidimensional collection that contains showing the line of source code with the appropriate class name in order to declare the class of the line. The line will be further processed by using the indexOf, substring, and lastIndexOf functions to retrieve the class name in the line and save it inside a global variable defined before once a class name is found by the condition declared in the code, such as the line contains "public class" and "{" inside the string. The map function is

used to construct a multidimensional collection that displays the class name and the related line of code when the class name has been retrieved.

```
//*******************************************************
// Define Class For Each Line

val define_class_of_line = preprocess.map(x=>{
  if(x.contains("public class") && x.contains("{")){
    if(x.contains("extends")){
      val extend = x.indexOf("extends")
      val cut = x.substring(0,extend-2)
      val space_last_index = cut.lastIndexOf(" ")
      className = cut.substring(space_last_index+1)
      (className,x)
    }
    else{
      val bracket_index = x.indexOf("{")
      val cut = x.substring(0,bracket_index-1)
      val space_last_index = cut.lastIndexOf(" ")
      className = cut.substring(space_last_index+1)
      (className,x)
    }
  }
  else{
    (className,x)
  }
})
```

Figure 6.36: Code Segment for declaring the class for each line of the source code

Prior to receiving the results, we must use the filter function to remove any lines with null classes, which indicate that they are lines that fall within an interface. Next, use the toDF function to convert the RDD into a dataframe and display the resulting dataframe in the console for improved result visualization.

```
val remove_nullFunction = define_class_of_line.filter(x=> x._1 != "") // Remove those interface line which has null in class column
val df_import=remove_nullFunction.toDF("Line","Class")
df_import.show(1000,false)
```

Figure 6.37: Code Segment to filtering null class and visualize the dataframe

```
+-------------------------+----------------------------------------------------------------------------------------------------+
|Line                     |Class                                                                                               |
+-------------------------+----------------------------------------------------------------------------------------------------+
|defaultexceptionhandler  |public class defaultexceptionhandler  extends responseentityexceptionhandler{                      |
|defaultexceptionhandler  |public final string responsefailed = "failed";                                                      |
|defaultexceptionhandler  |@exceptionhandler(exception.class)                                                                  |
|defaultexceptionhandler  |public final responseentity<errormessage> someerror(methodargumentnotvalidexception ex ){           |
|defaultexceptionhandler  |string message = "";                                                                                |
|defaultexceptionhandler  |for (int i = 0; i < ex.getbindingresult().getallerrors().size(); i++) {                             |
|defaultexceptionhandler  |message = message + ex.getbindingresult().getallerrors().get(i).getdefaultmessage() +"\n";          |
|defaultexceptionhandler  |}                                                                                                   |
|defaultexceptionhandler  |errormessage errormessage = new errormessage(message, responsefailed);                              |
|defaultexceptionhandler  |return new responseentity<>(errormessage, httpstatus.bad_request);                                  |
|defaultexceptionhandler  |}                                                                                                   |
|defaultexceptionhandler  |@exceptionhandler(runtimeexception.class)                                                           |
|defaultexceptionhandler  |public final responseentity<errormessage> nullerror(runtimeexception ex ){                          |
|defaultexceptionhandler  |errormessage errormessage = new errormessage(ex.getmessage(), responsefailed);                      |
|defaultexceptionhandler  |return new responseentity<>(errormessage, httpstatus.bad_request);                                  |
|defaultexceptionhandler  |}                                                                                                   |
|defaultexceptionhandler  |@override                                                                                           |
|defaultexceptionhandler  |protected responseentity<object> handlemethodargumentnotvalid(methodargumentnotvalidexception ex,   |
```

Figure 6.38: Part of the visualization of the result

6.3.2.1 Finding Variable for Each Class

Up to this point, all the preliminary processes to convert the source code into a better data representation collection have been finished. We can then move on to the following phase, which finds the variable for each class, by using the collection we retrieved during the class-declaring phase. The procedure used in each phase is the same: data is filtered, mapped to a multidimensional collection, and then transformed into a dataframe. The procedure for locating the variable for each class is shown in the code segment below. First, filtering the line from the source code that contains the variable class and various types of variables from the collection that was retrieved in the previous phases during the class declaring phase. The next step is to change each line of the collection that is followed by a variable using the map function once more. Use the built-in functions lastIndexOf, indexOf, and substring function, which were used in the previous step, to extract the variable name inside the map function. A multidimensional collection that only contains the class name and the variable name is created at the end of each line of the collection that has been filtered to only contain the variable.

```
//*****************************************************************************************************************************************************************************
// Find Variable

var variable = ""
val find_variable = remove_nullFunction.filter(x=>{
  x._2.contains(";") && !x._2.contains("return") && !x._2.contains("set") && !x._2.contains(".") && !x._2.contains("(") && (x._2.contains("string") || x._2.contains("double") || x._2.
  contains("final") || x._2.contains("response") || x._2.contains("integer") || x._2.contains("openstock") || x._2.contains("openstockdetails") || x._2.contains("userservice") || x._2.
  contains("openstockrepository") || x._2.contains("openstockdetailsrepository") || x._2.contains("stakeholdertelephone") || x._2.contains("stakeholderservice") || x._2.contains
  ("stakeholderrepository") || x._2.contains("stakeholder") || x._2.contains("item") || x._2.contains("transaction") || x._2.contains("date") || x._2.contains("user") || x._2.contains
  ("stocklogservice") || x._2.contains("stocklogrepository") || x._2.contains("transactionserviceimpl") || x._2.contains("useraddress") || x._2.contains("usertelephone") || x._2.contains
  ("userservice") || x._2.contains("userrepository"))
})
val clean_variable = find_variable.map(x=>{
  if(x._2.contains("=")){
    val equal_index = x._2.indexOf("=")
    val cut = x._2.substring(0,equal_index-1)
    val space_last_index = cut.lastIndexOf(" ")
    val variable = cut.substring(space_last_index+1)
    (x._1,variable)
  }
  else{
    val space_last_index = x._2.lastIndexOf(" ")
    val comma_index = x._2.indexOf(";")
    val variable = x._2.substring(space_last_index+1,comma_index)
    (x._1,variable)
  }
})
```

Figure 6.39: Code Segments of Finding Variable

Similar to the earlier phase, various exception handlers are utilized to filter the empty variable caused by the unclean pre-processing result and produce a dataframe to improve the display of the findings.

```
val exceptionhandler = clean_variable.filter(x=> x._2 != "")
val find_variable_df=exceptionhandler.toDF("Class","HasVariable")
val distinct_find_variable_df = find_variable_df.distinct()
val find_variable_count = distinct_find_variable_df.count
distinct_find_variable_df.show(500,false)
```

Figure 6.40: Code Segments of Exception Handler and Result Visualization

Class	HasVariable
errordetails	details
errormessage	message
errormessage	response
errordetails	timestamp
errordetails	message
defaultexceptionhandler	message
defaultexceptionhandler	responsefailed
item	unitprice
item	salesprice
item	id
item	name
item	uom
openstock	id
openstock	percentage
openstock	stringdenote
openstock	user
openstock	reason
openstock	openstockdetails

Figure 6.41: Part of the Finding Variable Result

6.3.2.2 Finding Methods for Each Class

To find the methods from each line with their respective class in this section, we are using the collection from the class-defining phase. The method in the code segment below filters out the lines that contain the letters "{" and "public," but not the letters "class." The data must then be mapped into a multidimensional collection that includes the names of the class and function. We are using built-in functions like indexOf, lastIndexOf, and substring during the mapping process to extract the function name and save it in a global variable. The next step is to create a new multidimensional collection using the global variable.

```
//*****************************************************************************
// Find Methods

var function_name = ""
val find_method = remove_nullFunction.filter(x=>{
  x._2.contains("{") && x._2.contains("public") && !x._2.contains("class")
})
val clean_method = find_method.map(x=>{
  val bracket_index = x._2.indexOf("(")
  val cut = x._2.substring(0,bracket_index)
  val space_last_index = cut.lastIndexOf(" ")
  function_name = cut.substring(space_last_index+1)
  (x._1,function_name)
})
```

Figure 6.42: Code Segments of Finding Methods

The multidimensional collection is then converted, as per usual, into a dataframe for improved data display. After that, a unique filtering function is applied to the dataframe to exclude instances of classes with identical functions. This method is then used to determine how many methods will be included in the code metrics.

```
val find_method_df=clean_method.toDF("Class","HasMethod")
val distinct_find_method_df = find_method_df.distinct()
val find_method_count = distinct_find_method_df.count
distinct_find_method_df.show(500,false)
```

Figure 6.43: Code Segments Result Visualization, Distinct and Count of Dataframe

Class	HasMethod
errormessage	getresponse
errormessage	setmessage
errormessage	setresponse
defaultexceptionhandler	someerror
errordetails	gettimestamp
errordetails	getdetails
errordetails	getmessage
errordetails	errordetails
errormessage	errormessage
errormessage	getmessage
defaultexceptionhandler	nullerror
item	setunitprice
item	getsalesprice
item	getname

Figure 6.44: Part of the Finding Method Result

6.3.2.3 Finding Dependencies for Each Class

Databricks uses the indexOf, lastIndexOf, and substring built-in functions inside the map function of the multidimensional collection retrieved from the pre-processing phase to find each of the function's names with the following line inside a class in order to determine the dependencies of each class. The map function will create a multidimensional collection with the class name, function name, and the line of source code, which represents the line of code under which function and which class, after extracting the function name using condition apply. After creating the multidimensional collection of class names, function names, and line data, we filter out any lines that don't contain function names or lines that contain the character "." (which denotes a function that depends on other methods or functions). The code segment that performs the extract and transformation action is shown below.

```
//*******************************************************************************************
// Find Dependencies

var function_name2 = ""
val dependency = remove_nullFunction.map(x=>{
  if(x._2.contains("{") && x._2.contains("public") && !x._2.contains("class")){
    val bracket_index = x._2.indexOf("(")
    val cut = x._2.substring(0,bracket_index)
    val space_last_index = cut.lastIndexOf(" ")
    function_name2 = cut.substring(space_last_index+1)
    if(function_name == -1){
      (x._1,"",x._2)
    }
    else{
      (x._1,function_name2,x._2)
    }
  }
  else{
    (x._1,function_name2,x._2)
  }
})

var find_dependencies = dependency.filter(x=>{
  x._3.contains(".") && x._2 != ""
})
```

Figure 6.45: Construct a new multidimensional collection and perform filter action

The dependent function is then retrieved from the line retrieved from the earlier constructed collection. As previously established, each "." stands for a function that depends on other methods or functions; a line may have multiple "."s, which signify a class that depends on multiple functions that are declared on other classes or built-in functions. We can now obtain the dependent function that is stored in "cut2" variables in the code segments below by using the indexOf and substring functions with a condition that was used to determine the index of the function name.

```
val second_process = find_dependencies.map(x=>{
  if(x._3.contains(".")){
    val dotindex = x._3.indexOf(".")
    val cut = x._3.substring(dotindex+1)
    val frontColumnIndex = cut.indexOf("(")
    val backColumnIndex = cut.indexOf(")")
    if(backColumnIndex < frontColumnIndex || frontColumnIndex == -1){
      val cut2 = cut.substring(0,backColumnIndex)
      (x._1,cut,x._2,cut2)
    }
    else{
      val cut2 = cut.substring(0,frontColumnIndex)
      (x._1,cut,x._2,cut2)
    }
  }
  else{
    ("","","","")
  }
})

var handle_nullFunction = second_process.filter(x=> x._3 != "")
val find_dependencies_df1=handle_nullFunction.toDF("Class","Cut1","Function","HasDependencies")
```

Figure 6.46: First Iteration of retrieve dependency function

Iterating over the code above and creating a new collection with the same data structure but different data within is required to obtain all the dependencies of a line that contains multiple "." characters. We must filter out empty lines before the function iterates to prevent a break in the analysis. As a result, each time the function is used, a new dataframe with the same basic structure but different data representing the class dependencies will be created.

```
val third_process = handle_nullFunction.map(x=>{
  if(x._2.contains(".")){
    val dotindex = x._2.indexOf(".")
    val cut = x._2.substring(dotindex+1)
    val frontColumnIndex = cut.indexOf("(")
    val backColumnIndex = cut.indexOf(")")
    if(backColumnIndex < frontColumnIndex || frontColumnIndex == -1){
      val cut2 = cut.substring(0,backColumnIndex)
      (x._1,cut,x._3,cut2)
    }
    else{
      val cut2 = cut.substring(0,frontColumnIndex)
      (x._1,cut,x._3,cut2)
    }
  }
  else{
    ("","","","")
  }
})

var handle_nullFunction2 = third_process.filter(x=> x._3 != "")
val handle_exception = handle_nullFunction2.filter(x=>x._4 != "class, args")
val find_dependencies_df2=handle_exception.toDF("Class","Cut1","Function","HasDependencies")
```

Figure 6.47:Second Iteration of the retrieve dependency function

```
val fourth_process = handle_exception.map(x=>{
  if(x._2.contains(".")){
    val dotindex = x._2.indexOf(".")
    val cut = x._2.substring(dotindex+1)
    val frontColumnIndex = cut.indexOf("(")
    val backColumnIndex = cut.indexOf(")")
    if(backColumnIndex < frontColumnIndex || frontColumnIndex == -1){
      val cut2 = cut.substring(0,backColumnIndex)
      (x._1,cut,x._3,cut2)
    }
    else{
      val cut2 = cut.substring(0,frontColumnIndex)
      (x._1,cut,x._3,cut2)
    }
  }
  else{
    ("","","","")
  }
})

var handle_nullFunction3 = fourth_process.filter(x=> x._3 != "")
val find_dependencies_df3=handle_nullFunction3.toDF("Class","Cut1","Function","HasDependencies")
```

Figure 6.48: Third Iteration of the retrieve dependency function

```
val fifth_process = handle_nullFunction3.map(x=>{
  if(x._2.contains(".")){
    val dotindex = x._2.indexOf(".")
    val cut = x._2.substring(dotindex+1)
    val frontColumnIndex = cut.indexOf("(")
    val backColumnIndex = cut.indexOf(")")
    if(backColumnIndex < frontColumnIndex || frontColumnIndex == -1){
      val cut2 = cut.substring(0,backColumnIndex)
      (x._1,cut,x._3,cut2)
    }
    else{
      val cut2 = cut.substring(0,frontColumnIndex)
      (x._1,cut,x._3,cut2)
    }
  }
  else{
    ("","","","")
  }
})

var handle_nullFunction4 = fifth_process.filter(x=> x._3 != "")
val find_dependencies_df4=handle_exception.toDF("Class","Cut1","Function","HasDependencies")
```

Figure 6.49: Fourth Iteration of the retrieve dependency function

Due to the maximum number of times the "." character can appear in a line, which is 4, all dependencies have been removed after four iterations of the function. The next step is to combine all of the dataframes, delete any extraneous columns, and run a special filtering

algorithm to remove any objects that repeatedly appear in the dataframe. Last but not least, the function that uses Class and Function columns to sort the dataframe uses an ascending order. The code snippet that follows demonstrates how the function merges the dataframe and carries out the various modifications that were mentioned earlier.

```
val final_find_dependenvies_df = find_dependencies_df4.union(find_dependencies_df3).union(find_dependencies_df2).union(find_dependencies_df1)
val clean_final_find_dependenvies_df = final_find_dependenvies_df.drop("Cut1")
val distinct_clean_final_find_dependenvies_df = clean_final_find_dependenvies_df.distinct()
val sorted_distinct_clean_final_find_dependenvies_df = distinct_clean_final_find_dependenvies_df.sort(col("Class").asc,col("Function").asc)
sorted_distinct_clean_final_find_dependenvies_df.show(500,false)
```

Figure 6.50: Merging and performing transformations action of dataframe

```
+--------------------------+--------------------------+-------------------------+
|Class                     |Function                  |HasDependencies          |
+--------------------------+--------------------------+-------------------------+
|defaultexceptionhandler   |nullerror                 |getallerrors             |
|defaultexceptionhandler   |nullerror                 |size                     |
|defaultexceptionhandler   |nullerror                 |get                      |
|defaultexceptionhandler   |nullerror                 |getmessage               |
|defaultexceptionhandler   |nullerror                 |getbindingresult         |
|defaultexceptionhandler   |nullerror                 |bad_request              |
|defaultexceptionhandler   |someerror                 |getallerrors             |
|defaultexceptionhandler   |someerror                 |get                      |
|defaultexceptionhandler   |someerror                 |size                     |
|defaultexceptionhandler   |someerror                 |class                    |
|defaultexceptionhandler   |someerror                 |getbindingresult         |
|defaultexceptionhandler   |someerror                 |bad_request              |
|openstockcontroller       |createopenstock           |accepted                 |
|openstockcontroller       |createopenstock           |getuser                  |
|openstockcontroller       |createopenstock           |now                      |
|openstockcontroller       |createopenstock           |bad_request              |
|openstockcontroller       |createopenstock           |body                     |
|openstockcontroller       |createopenstock           |of                       |
```

Figure 6.51: Part of the Dependencies Result

6.3.2.4 Code Metrics for Whole Application

Using the filter method, Scala is able to extract every line that has the necessary information for us, such as the imported libraries, class, interface, and function, making it simple to obtain the Code Metrics of the entire program. Scala can filter the same element that appears in the collection by utilizing a separate function after getting the data by removing lines that contain the pre-defined keyword. Last but not least, use the count function on the collection to get the amount of information that has been saved; it will return that amount. The code snippets below demonstrate how to retrieve the application's overall Code metrics.

```
//*************************************************************************************************************************************************************
// Code Metrics
val find_import = split.filter(_.contains("import")).filter(!_.contains("//")).filter(!_.contains("@"))
val find_class = split.filter(_.contains("public class")).filter(!_.contains("//")).filter(!_.contains("@"))
val find_interface = split.filter(_.contains("interface")).filter(!_.contains("//")).filter(!_.contains("@"))
val find_function = split.filter(_.startsWith("public")).filter(_.contains("(")).filter(_.contains(")")).filter(!_.contains("//"))
val function_distinct=find_function.distinct
val library_distinct=find_import.distinct
val class_distinct=find_class.distinct
val interface_distinct=find_interface.distinct
val library_count = library_distinct.count
val interface_count = interface_distinct.count
val class_count = class_distinct.count
val count_method = function_distinct.count
val data = Seq(("TotalLOC", TLOC),("TotalClass", class_count),("TotalMethod", count_method),("TotalInterface",interface_count),("TotalImportedLibrary",library_count))
val metrics = spark.sparkContext.parallelize(data)
val metrics_df = metrics.toDF("Attribute","Quantity")
metrics_df.show()
```

Figure 6.52: Function to Retrieve Code Metrics

```
+--------------------+--------+
|           Attribute|Quantity|
+--------------------+--------+
|            TotalLOC|    1973|
|          TotalClass|      27|
|         TotalMethod|     126|
|      TotalInterface|      11|
|TotalImportedLibrary|      69|
+--------------------+--------+
```

Figure 6.53: Result of the Code Metrics Dataframe

6.3.3 Store Data Module

We can now write those data into a csv file and save it back in the S3 bucket declared in the previous module, which was used to extract the uploaded file from the user, after all the necessary data has been saved as a dataframe and stored in the variable declared in the Databricks notebook. We can change the options for the csv and the mode that is used to write the csv by utilizing a built-in function in Databricks. For instance, we may use the "option("header",true)" option to include the header of the dataframe in the csv file and the "overwrite mode" to allow the file writer to overwrite the file when the identical file already exists. The code snippets below demonstrate how the mounted S3 location in the DBFS writes the various dataframe into CSV and saves them to an S3 bucket.

```
//*********************************************************************
// Write Into Files and Save into S3

distinct_find_variable_df.write
  .option("header",true)
  .mode("overwrite")
  .csv("/mnt/files/variableCSV")

distinct_find_method_df.write
  .option("header",true)
  .mode("overwrite")
  .csv("/mnt/files/methodCSV")

sorted_distinct_clean_final_find_dependenvies_df.write
  .option("header",true)
  .mode("overwrite")
  .csv("/mnt/files/dependenciesCSV")

metrics_df.write
  .option("header",true)
  .mode("overwrite")
  .csv("/mnt/files/metricsCSV")
```

Figure 6.54: Writing dataframe into CSV

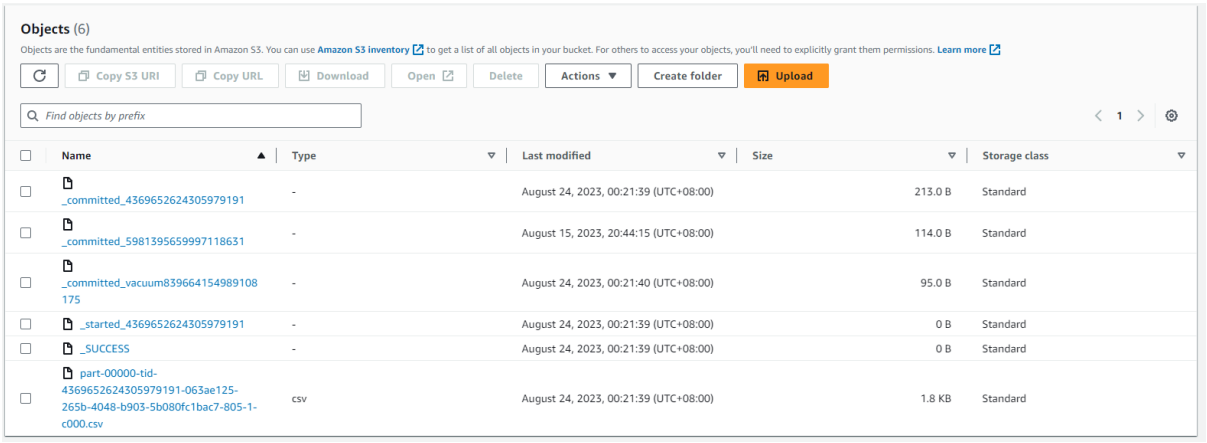
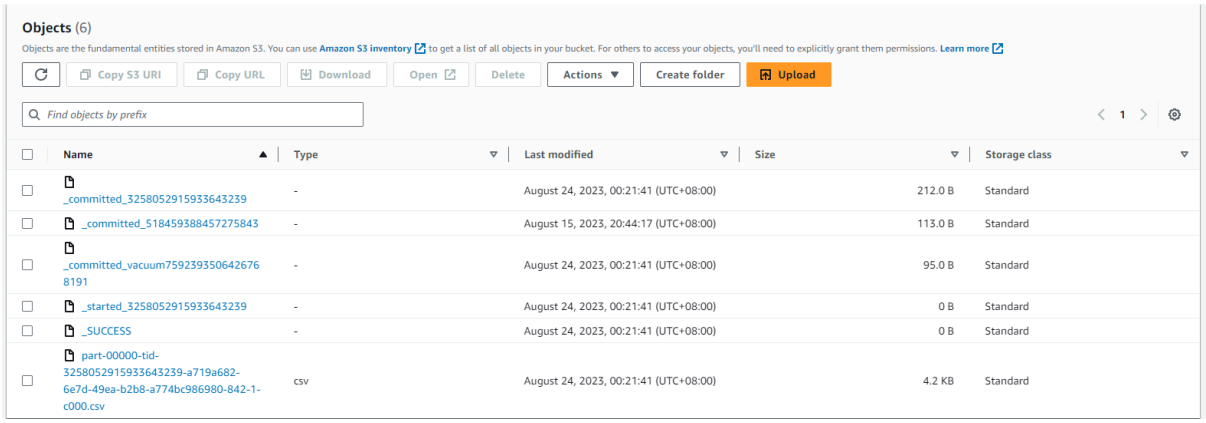


Figure 6.55: Variable CSV Stored in S3 Bucket



Objects (6)

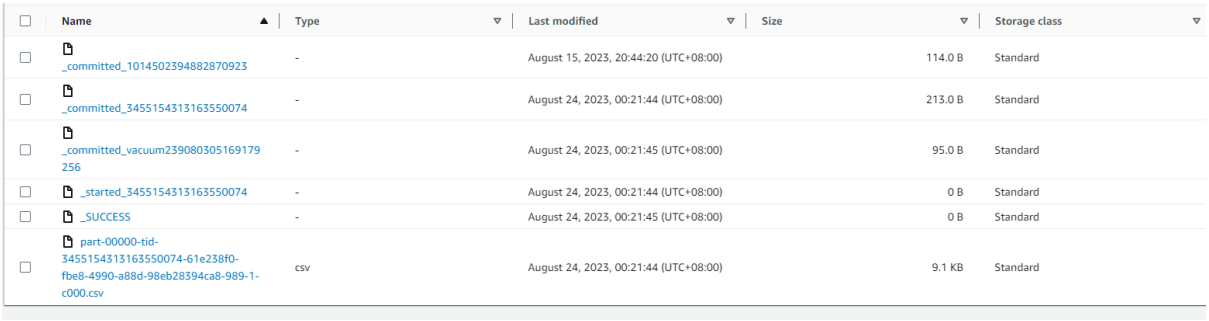
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

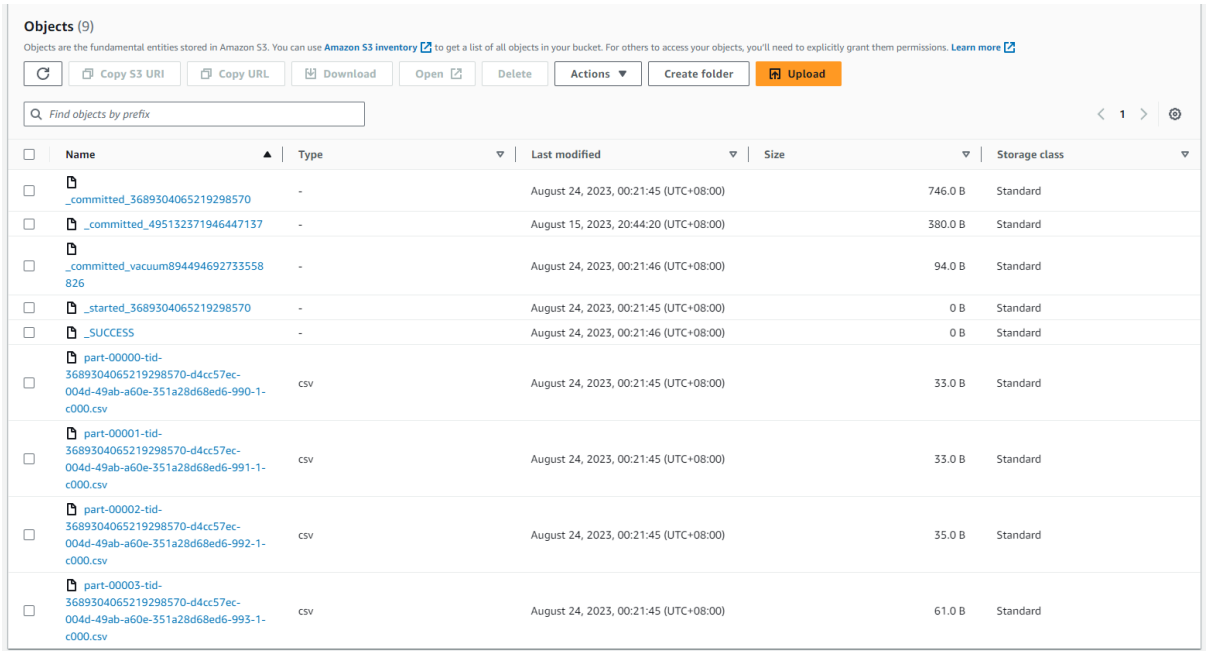
Name	Type	Last modified	Size	Storage class
_committed_3258052915933643239	-	August 24, 2023, 00:21:41 (UTC+08:00)	212.0 B	Standard
_committed_518459388457275843	-	August 15, 2023, 20:44:17 (UTC+08:00)	113.0 B	Standard
_committed_vacuum7592393506426768191	-	August 24, 2023, 00:21:41 (UTC+08:00)	95.0 B	Standard
_started_3258052915933643239	-	August 24, 2023, 00:21:41 (UTC+08:00)	0 B	Standard
_SUCCESS	-	August 24, 2023, 00:21:41 (UTC+08:00)	0 B	Standard
part-00000-tid-3258052915933643239-a719a682-6e7d-49ea-b2b8-a774bc986980-842-1-c000.csv	csv	August 24, 2023, 00:21:41 (UTC+08:00)	4.2 KB	Standard

Figure 6.56: Method CSV Stored in S3 Bucket



Name	Type	Last modified	Size	Storage class
_committed_1014502394882870923	-	August 15, 2023, 20:44:20 (UTC+08:00)	114.0 B	Standard
_committed_3455154313163550074	-	August 24, 2023, 00:21:44 (UTC+08:00)	213.0 B	Standard
_committed_vacuum239080305169179256	-	August 24, 2023, 00:21:45 (UTC+08:00)	95.0 B	Standard
_started_3455154313163550074	-	August 24, 2023, 00:21:44 (UTC+08:00)	0 B	Standard
_SUCCESS	-	August 24, 2023, 00:21:45 (UTC+08:00)	0 B	Standard
part-00000-tid-3455154313163550074-61e238f0-fbe8-4990-a88d-98eb28394ca8-989-1-c000.csv	csv	August 24, 2023, 00:21:44 (UTC+08:00)	9.1 KB	Standard

Figure 6.57: Dependencies CSV Stored in S3 Bucket



Objects (9)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
_committed_3689304065219298570	-	August 24, 2023, 00:21:45 (UTC+08:00)	746.0 B	Standard
_committed_495132371946447137	-	August 15, 2023, 20:44:20 (UTC+08:00)	380.0 B	Standard
_committed_vacuum894494692733558826	-	August 24, 2023, 00:21:46 (UTC+08:00)	94.0 B	Standard
_started_3689304065219298570	-	August 24, 2023, 00:21:45 (UTC+08:00)	0 B	Standard
_SUCCESS	-	August 24, 2023, 00:21:46 (UTC+08:00)	0 B	Standard
part-00000-tid-3689304065219298570-d4cc57ec-004d-49ab-a60e-351a28d68ed6-990-1-c000.csv	csv	August 24, 2023, 00:21:45 (UTC+08:00)	33.0 B	Standard
part-00001-tid-3689304065219298570-d4cc57ec-004d-49ab-a60e-351a28d68ed6-991-1-c000.csv	csv	August 24, 2023, 00:21:45 (UTC+08:00)	33.0 B	Standard
part-00002-tid-3689304065219298570-d4cc57ec-004d-49ab-a60e-351a28d68ed6-992-1-c000.csv	csv	August 24, 2023, 00:21:45 (UTC+08:00)	35.0 B	Standard
part-00003-tid-3689304065219298570-d4cc57ec-004d-49ab-a60e-351a28d68ed6-993-1-c000.csv	csv	August 24, 2023, 00:21:45 (UTC+08:00)	61.0 B	Standard

Figure 6.58: Metrics CSV Stored in S3 Bucket

In Addition, some of the dataframe is stored in several files due to the Apache Spark operates on data partitions, which are smaller chunks of the data that can be processed in

parallel. Spark distributes the data across multiple partitions to take advantage of parallel processing capabilities. Each partition becomes a separate file when the writer publishes the partitioned data to files.

6.3.4 Ontology Transformation Module

The next stage is to obtain the content of the CSV files and create the ontology using RDFLib, a Python library, which will be done once the source code has been converted into a structured dataframe and stored as CSV in an S3 bucket. Python will be used throughout the ontology creation procedure. Therefore, the first step in creating the ontology is to use Python to retrieve the data from the S3 Bucket. Using the "pip install boto3" command or the "conda install -c conda-forge boto3" command in the Anaconda environment to serve the Python language, the Boto3 library must be installed in the Python environment in order to link Python with S3 Bucket. Boto3 is the AWS Python SDK, which enables programmers to connect with a variety of AWS services. It offers a collection of APIs and libraries that simplify the process of creating software that uses AWS's cloud resources and services. Boto3 makes it easier to create, configure, manage, and interact programmatically with AWS services.

Before using Boto3, we must set up the login credentials for the AWS account using the IAM Console and AWS CLI. By repeating the steps in section 6.2.1 of the generating user process, we may create a new user for the S3 bucket using the IAM Console. That user can access the S3 bucket using the credentials and access key supplied. The S3 Bucket can be accessed using the same Access Key and Secret Access Key of the user created in chapter 6.2.1. The command "aws configure" sets up the credentials file and requests the Access Key, Secret Access Key, and the default region name. The AWS CLI must be downloaded in order to configure the authentication credentials using the "aws configure" command. The credentials and configuration file in the .aws folder, as shown in the image below, will save the Access Key and Secret Access Key with the default region once the configuration process is finished.

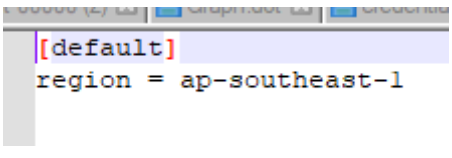


Figure 6.59: Default Region Stored in config file

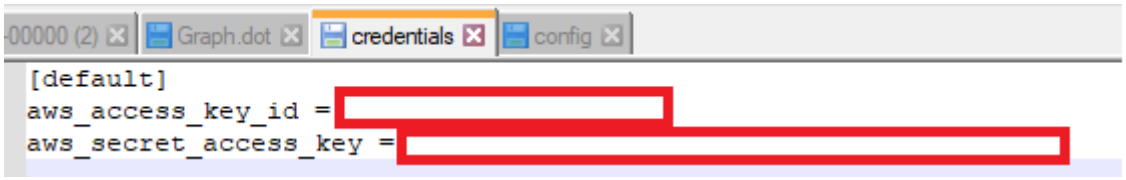


Figure 6.60: Access Key and Secret Access Key Stored in credentials file

Finally, after setting up the AWS credentials, we can now retrieve data from an S3 bucket. We first build a boto3 S3 client and specify the S3 bucket name in order to later obtain all the information about the S3 bucket. The keys to obtain the contents of the files stored in the dependenciesCSV folder in the S3 bucket will be saved in the dependencies_file array. Due to the folder may contain several files due to the partitioning of Databricks which store the information in several files. There are other arrays, such as the methods_file, metric_file, and variable_file arrays, that are used to hold different keys to access the information as well. Calling the list_object_v2 function after supplying the bucket name as the Bucket parameter will provide a list of all the objects in the bucket. Iterating the list objects allows us to collect the Keys required to later extract the files' contents. In addition, the earlier-created array is used to maintain the Keys by applying nested conditional expressions that are used by the startwith and endswith functions to choose which array to save the Keys in. The dependencies_file array will hold the key and the keys saved will be used to get the information inside the file, for instance, a key ended with ".csv" and start with "dependencies" showing the key is storing a part of the dependent information. Additionally, an exception handler is used to deal with any errors that may arise when trying to extract the Keys from the S3 bucket. The code snippets in Appendix C that follow show you how to get keys out of an S3 bucket.

Figure 6.61: Each of object retrieve from the response variable

By using the read and decode function of the Body attribute in the return object to convert the sequence of bytes encoded in UTF-8 characters into human-readable text, typically in the form of Unicode characters, the contents of the file can be retrieved by calling the get_object function of the S3 client. This will first obtain object information by iterating through the array that stores the keys. Since the read_csv method requires either a file-like object or a valid file path as input to read CSV data, we must use StringIO to convert the data from the string csv_content into a File-Like Object. StringIO provides a way to handle the CSV content as if it were a file when using the read_csv function. As a result, after the CSV data has been transformed into a File-Like Object, the read_csv function can read the data and transform it into a dataframe. The last step is to store the dataframe into an array.

Due to the partitioning of the Databricks Cluster, some of the data will be saved into different files in the S3 bucket, indicating that different keys will be used to access a subject's data, each of which will yield a single dataframe. To simplify the process later on, we must concatenate many dataframes in an array into a single dataframe. Using the concat function of the Pandas package, which also provides options like axis = 0 for concatenating the dataframe vertically, we can easily finish this work. The Code Segments in Appendix D illustrate the information extraction process and result presentation.

	Class	Function	HasDependencies
0	defaultexceptionhandler	nullerror	getallerrors
1	defaultexceptionhandler	nullerror	size
2	defaultexceptionhandler	nullerror	get
3	defaultexceptionhandler	nullerror	getmessage
4	defaultexceptionhandler	nullerror	getbindingresult
..	...	...	...
199	userservice	fetchallusers	findall
200	userservice	finduserbyid	findbyid
201	userservice	saveuserdetails	getusertelephones
202	userservice	saveuserdetails	setuser
203	userservice	saveuserdetails	save

[204 rows x 3 columns]

Figure 6.62: Result of Dependencies Dataframe

	Class	HasMethod
0	errormessage	getresponse
1	errormessage	setmessage
2	errormessage	setresponse
3	defaultexceptionhandler	someerror
4	errordetails	gettimestamp
..	...	...
154	usertelephone	setuser
155	usertelephone	getnumber
156	usertelephone	setnumber
157	usertelephone	getuser
158	usertelephone	setid

[159 rows x 2 columns]

Figure 6.63: Result of Methods Dataframe

	Class	HasVariable
0	errordetails	details
1	errormessage	message
2	errormessage	response
3	errordetails	timestamp
4	errordetails	message
..	...	...
70	usercontroller	userservice
71	userservice	userrepository
72	usertelephone	id
73	usertelephone	number
74	usertelephone	user

[75 rows x 2 columns]

Figure 6.64: Result of Variable Dataframe

	Attribute	Quantity
0	TotalLOC	1973
1	TotalClass	27
2	TotalMethod	126
3	TotalInterface	11
4	TotalImportedLibrary	69

Figure 6.65: Result of Metrics Dataframe

The concatenated dataframe can now be used to build an ontology in Python using RDFLib. Establishing classes, attributes, and relationships inside the ontology framework is one of the processes in this process. This focuses on the structure and schema of the ontology rather than specific instances or individuals. Python users can install RDFLib by issuing the pip command "pip install rdflib" or, in the Anaconda environment, "conda install -c conda-forge rdflib." Importing the necessary classes and modules from RDFLib is the next step, as seen in the accompanying figure:

```
from rdflib import Graph, URIRef, Literal, Namespace, RDF, BRICK, RDFS, OWL
```

Figure 6.66: Import necessary classes and modules from RDFLib

After that, create an RDF graph to hold the ontology. Custom namespaces may be specified for the ontology's elements, attributes, and literals. By adding triples to the graph, we can also define classes, subclasses, and properties. To further understand how the entities and attributes of the ontologies are defined, let's take a look at the Figure below.

```
g = Graph()
dependencies = Namespace("http://semanticBasedRedocumentation.org/class/")

mainClass_ref = URIRef(dependencies+'Class')
methodClass_ref = URIRef(dependencies+'Method')
variableClass_ref = URIRef(dependencies+'Variable')
dependenciesClass_ref = URIRef(dependencies+'Dependencies')
metricClass_ref = URIRef(dependencies+'Metrics')
quantityClass_ref = URIRef(dependencies+'Quantity')
g.add((mainClass_ref,RDF.type,OWL.Class))
g.add((methodClass_ref,RDF.type,OWL.Class))
g.add((methodClass_ref,RDF.type,OWL.Class))
g.add((variableClass_ref,RDF.type,OWL.Class))
g.add((dependenciesClass_ref,RDF.type,OWL.Class))
g.add((metricClass_ref,RDF.type,OWL.Class))
g.add((quantityClass_ref,RDF.type,OWL.Class))
```

Figure 6.67: Base Class Reference Defined

To begin, we create an RDF graph to hold the ontology in accordance with the illustration above. The following step is the development of a namespace, which acts as the foundation URI for the subsequent generation of further URIRef. A namespace is a method for supplying the prefix or abbreviation of a URI. URIs are used to uniquely identify resources in RDF data, which is crucial to the Semantic Web and linked data principles. RDF data may be handled and read by humans more easily by shortening long URIs using a namespace. All of the base class references declared in the above image are types of an ontology class, as was mentioned in chapter 5.2.2. The base class reference then corresponds to the design that was created and added to the ontology graph as a triple.
Accompanying the concatenation of dataframes in the previous phase, the ontology may be generated using the dataframe and RDFLib, as shown in the accompanying figure.

The figure in Appendix E shows the ontology's development process. We first iterate through the Dependency dataframe using the dataframe's iterrows method, which returns the data row by row with related columns. Using the row data received, the namespace specified previously, and the addition of the value by calling the row with the appropriate column name, we first create a URIRef (Uniform Resource Identifier Reference) for the class and dependent function. Then, using the URIRef of the defined resource, we add the triple to the graph to specify the classes. We also create a new URIRef with the class name and the string has_method appended at the end to represent the resources that a class has that rely on a function. Using the URIRef, we add the triple into the graph as a sub-property of topObjectProperty, which highlights the connection between the class and the dependent function. Once the ObjectProperty has been defined, the domain and range can now be added to the graph by including a triple with the appropriate class and dependant URIRef declared previously.

	Class	Function	HasDependencies
0	defaultexceptionhandler	nullerror	getallerrors
1	defaultexceptionhandler	nullerror	size
2	defaultexceptionhandler	nullerror	get

Figure 6.68: Explain example

```
cls = Namespace('http://semanticBasedRedocumentation.org/class/')
```

Figure 6.69: Namespace define earlier

Consider the dataframe in the figure above as an illustration. In the first iteration, the dataframe only retrieves the row of data with index 0. By using the namespace previously defined and adding the class name to the end of the namespace, we establish the class reference using the URIRef function. The dependent reference is the same. The generated reference is then combined with a triple representing the produced resource, which is a subclass of the primary class reference that was previously declared, to include both references that were previously declared as resources in the ontology graph. The dependant reference is equivalent, but it belongs to the dependent Class reference that was previously declared. The dependent function and class relationship are then represented by a new reference that is created next. We add a new ObjectProperty with multiple triples to the ontology network by repeating the previous procedure. We can now add the domain and range under this ObjectProperty using the URIRef of the ObjectProperty. The class to which an ObjectProperty can be applied as a subject is specified by its domain and the class to which it can be attached as an object is specified by its range. The class reference in this instance is the Object Property domain, and the dependent reference is the Object Property range, signifying that the class is dependent on the function. Referring back to figure 6.71, the defaultexceptionhandler class is dependent on the getallerrors function, which has a specific URI reference, in the row with index 0, following the building of the ontology.

URIRef list:

defaultexceptionhandler: http://semanticBasedRedocumentation.org/class/defaultexceptionhandler

getallerror: http://semanticBasedRedocumentation.org/class/getallerror

dependent (Object Property):

http://semanticBasedRedocumentation.org/class/defaultexceptionhandler_has_dependent

domain: http://semanticBasedRedocumentation.org/class/defaultexceptionhandler

range: http://semanticBasedRedocumentation.org/class/getallerror

Ontologies and RDF normally need URIs, which URIRef represents, to be globally unique. As a result, the RDFLib is able to recognize the repeated URI and put the information under it even after the dataframe has been iterated. For instance, the RDFLib will recognize the repeated class and dependent reference and add the new data under it while iterating over

the dataframe which has the same class reference as the previous row and dependent reference. In the figure below, the final result is depicted:

```
<http://semanticBasedRedocumentation.org/class/defaultexceptionhandler_has_dependent> a owl:ObjectProperty ;
    rdfs:domain <http://semanticBasedRedocumentation.org/class/defaultexceptionhandler> ;
    rdfs:range <http://semanticBasedRedocumentation.org/class/bad_request>,
        <http://semanticBasedRedocumentation.org/class/class>,
        <http://semanticBasedRedocumentation.org/class/get>,
        <http://semanticBasedRedocumentation.org/class/getallerrors>,
        <http://semanticBasedRedocumentation.org/class/getbindingresult>,
        <http://semanticBasedRedocumentation.org/class/getmessage>,
        <http://semanticBasedRedocumentation.org/class/size> ;
    rdfs:subPropertyOf owl:topObjectProperty .
```

Figure 6.70: Example Ontology Output in TTL format

As a result, now that we understand how an ontology is built, we can replicate the process with other dataframes, such as method, variable, and metrics dataframes, to build an entire ontology graph with the various classes defined and various types of object properties that represent various relationships, such as has method, has variable, and has attributes for different classes.

Now that the entire ontology graph has been created, we may extract the data from the ontology. Flask will be used to handle the request received and provide the data to the front-end application, which will streamline the retrieving process. Flask, a Python micro web framework for creating web apps and APIs, will be used to create the back-end services. Flask is able to immediately retrieve data from the ontology in the Python environment and deliver it to the front-end application for display purposes because we built the ontology using RDFLib in Python. The steps involved in utilizing Flask to provide a Python backend service include developing a Flask application, specifying routes, and managing requests and responses. Installing the flask package using the pip command "pip install Flask" or the conda command "conda install -c anaconda flask" in an Anaconda environment is required before using flask in a Python environment. Secondly, import the library into the Jupyter Notebook that was used to run the Python command indicated in the following figure:

```
from flask import Flask, render_template, request, send_file, jsonify
```

Figure 6.71: Import Flask Library in Jupyter Notebook

The following step is to create an instance of the Flask class and supply the unique Python variable "__name__" as an input, which aids Flask in determining the application's

root path. In relation to the application's location in the file system, it allows Flask to know where to seek for static files, templates, and other resources.

```
OntologyGenerator = Flask(__name__)
```

Figure 6.72: Creating Instance of Flask

Use the "@OntologyGenerator.route" decorator to define the routes next. The URL path is the parameter given to the decorator. Furthermore, by passing a list of methods as a parameter to the "@OntologyGenerator.route" decorator, we may specify which HTTP methods the route should support. For further information on how the Flask routes were created, let's take a closer look at the figure below.

```
@OntologyGenerator.route('/GetAllMethod',methods=['GET'])
def getAllMethod():
    data = []
    query = """
        SELECT ?domain ?property ?range
        WHERE {
            ?property rdfs:domain ?domain ;
                      rdfs:range  ?range .
        }
    """
    qres = g.query(query)
    for row in qres:
        if(row[1][-10::] == "_dependent"):
            data.append({"Class": row[0][46::], "Relationship": "Has Dependent" ,"Dependent": row[2][46::]})

        elif (row[1][-10::] == "has_method"):
            data.append({"Class": row[0][46::], "Relationship": "Has Method" ,"Method": row[2][46::]})

        elif (row[1][-10::] == "_attribute"):
            data.append({"Class": row[0][46::], "Relationship": "Has Attribute" ,"Attribute": row[2][46::]})

        else:
            data.append({"Class": row[0][46::], "Relationship": "Has Variable" ,"Variable": row[2][46::]})
    df = pd.DataFrame(data)
    df_filter = df[df['Relationship'] == "Has Method"]
    df_filter2 = df_filter.drop(columns=['Variable','Dependent','Attribute'])
    json_data = df_filter2.to_json(orient='records')

    return json_data
```

Figure 6.73: GetAllMethod Route Defined

The route defined as "/GetAllMethod" in the diagram above accepts client requests using the HTTP "GET" method. The "@OntologyGenerator.route" decorator's getAllMethod method will be run when someone has sent the request through the URL. After the route has been declare, we are now able to run the Flask application locally by the instance create before with a given port number as shown in the figure below:

```
if __name__ == "__main__":
    OntologyGenerator.run(port=5000)
```

Figure 6.74: Running of the Flask application

After the running of the Flask Application, we can now access the services by sending request to http://127.0.0.1:5000/{Route_Name}. For example, in the figure 6.76, we can access to the getAllMethod service by sending a "GET" request to http://127.0.0.1:5000/GetAllMethod and the service will return a Jsonify data which contains the methods data.

Returning to Figure 6.76, we first create an empty array inside the function to store the data that will be retrieved later. The "query" function of the RDFLib library is then used to create the SPARQL query that will be used to extract data from the ontology network. The prepared question is passed as an argument to the "query" function, which retrieves the data from the ontology network. The "g" stands for the ontology graph created in the previous stage. The query is used to get each object attribute along with its appropriate domain and range using a URI reference syntax. The figure below shows the piece of the ontology graph that the query returned:

```
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/getallerr
ors'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/size'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/get'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/getmessag
e'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/getbindin
gresult'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/defaultexceptionhandler'), rdflib.term.URIRef('http://semanticBasedRedo
cumentation.org/class/defaultexceptionhandler_has_dependent'), rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/bad_reque
st'))
```

Figure 6.75: Part of the result retrieve with the query in Ontology Graph

Following the data retrieval, we simply use the class name, relationship, and method to further process the data. We may handle the data using a conditional statement by examining the second column of the array, which indicates the object property because the data retrieval is in multidimensional array format. The data will be further processed and saved into the data array that was previously declared with several columns, including the Dependent, Method, Attribute, and Variable columns. The data array will then be

converted into a dataframe, and a filter procedure will be used to keep only the rows with the "Has Method" relationship and remove any unneeded columns from the dataframe, including variable, dependent, and attribute columns with null values. The dataframe is finally transformed into JSON format data and returned as jsonified data. The outcome is depicted in the following figure.

	Class	Relationship	Dependent	Method	Variable	\
0	defaultexceptionhandler	Has Dependent	getallerrors	NaN	NaN	
1	defaultexceptionhandler	Has Dependent	size	NaN	NaN	
2	defaultexceptionhandler	Has Dependent	get	NaN	NaN	
3	defaultexceptionhandler	Has Dependent	getmessage	NaN	NaN	
4	defaultexceptionhandler	Has Dependent	getbindingresult	NaN	NaN	
..	...	...	...	...	...	
358	TotalLOC	Has Attribute	NaN	NaN	NaN	
359	TotalClass	Has Attribute	NaN	NaN	NaN	
360	TotalMethod	Has Attribute	NaN	NaN	NaN	
361	TotalInterface	Has Attribute	NaN	NaN	NaN	
362	TotalImportedLibrary	Has Attribute	NaN	NaN	NaN	

Figure 6.76: Transformation of the Retrieve Data into Dataframe

	Class	Relationship	Dependent	Method	\
7	defaultexceptionhandler	Has Method	NaN	someerror	
8	defaultexceptionhandler	Has Method	NaN	nullerror	
38	openstockcontroller	Has Method	NaN	createopenstockdetails	
39	openstockcontroller	Has Method	NaN	getmessage	
40	openstockcontroller	Has Method	NaN	setmessage	
..	...	...	...	...	
350	usertelephone	Has Method	NaN	setuser	
351	usertelephone	Has Method	NaN	getnumber	
352	usertelephone	Has Method	NaN	setnumber	
353	usertelephone	Has Method	NaN	getuser	
354	usertelephone	Has Method	NaN	setid	

Figure 6.77: Filtering the data with “Has Method” Relationship

	Class	Relationship	Method
7	defaultexceptionhandler	Has Method	someerror
8	defaultexceptionhandler	Has Method	nullerror
38	openstockcontroller	Has Method	createopenstockdetails
39	openstockcontroller	Has Method	getmessage
40	openstockcontroller	Has Method	setmessage
..	...	...	...
350	usertelephone	Has Method	setuser
351	usertelephone	Has Method	getnumber
352	usertelephone	Has Method	setnumber
353	usertelephone	Has Method	getuser
354	usertelephone	Has Method	setid

[159 rows x 3 columns]

Figure 6.78: Dropping the Unnecessary Column

[{"Class":"defaultexceptionhandler","Relationship":"Has Method","Method":"someerror"},{"Class":"defaultexceptionhandler","Relationship":"Has Method","Method":"nullerror"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"createopenstockdetails"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"getmessage"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"setmessage"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"saveopenstock"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"createopenstock"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"deleteopenstockdetails"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"getresponse"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"fetchopenstockdetailsbyid"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"oncall"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"setresponse"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"updateopenstocklog"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"fetchallopenstock"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"fetchallopenstockdetailsbyid"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"updateopenstockdetails"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"deleteopenstocklog"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"fetchopenstockbyid"},{"Class":"openstockcontroller","Relationship":"Has Method","Method":"deleteallopenstockdetails"},{"Class":"openstockservic","Relationship":"Has Method","Method":"updateopenstockdetails"},{"Class":"openstockservic","Relationship":"Has Method","Method":"fetchallopenstockdetails"},{"Class":"openstockservic","Relationship":"Has Method","Method":

Figure 6.79: Output of the Jsonify Data

Figure 6.80: Better Visualization of Jsonify Data in Postman

By using the aforementioned procedure, we are now able to request the method data from a local host domain using the specified port. The same procedure is followed when retrieving variables, dependencies, and metrics data, with a few modifications like subject filtering to get the data we need for various purposes. The code snippets in Appendix F demonstrate the method and procedure used to retrieve the additional data.

All the variables, methods, dependencies, and metrics data are retrievable via the aforementioned routes. As a result, the next functionality to build is the ability to access data using a class provided by the user, which only returns the data the user specifically requested. Let's take a closer look at the route shown in the Appendix F for further information.

The route is declared as seen in Appendix F, and its goal is to retrieve the dependencies information provided by the user for the class. Pay attention to the string <class> that indicates the variable section in the route URL. Then, the variable will be passed as a keyword argument to the function below. The data retrieved and subsequently transformed from the data retrieved from the ontology are then stored in an array that is produced. Before utilizing the SPARQL search query, we next modify the Input variable by putting the class name that the user entered into a URL string that denotes the URLRef of the ontology resource we want to search for. The prepared SPARQL query is then combined with the input variable to search the object property with its domain and range using a URIRef that is identical to the input variable. The data that was retrieved is also further processed before being put into the data array that was earlier created and converted into a dataframe. The dataframe is finally transformed into JSON data and sent back to the front-end application, which then sends a request to the Flask application. By sending a request to http://127.0.0.1:5000/<class>, one can access the services; the class attribute in the URL denotes user input. The information obtained with the user input "usercontroller" is displayed below.

```
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/deleteallusers'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/deleteuser'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/fetchallusers'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/get'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/build'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/ispresent'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/notfound'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/ok'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/finduserbyid'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/saveuserdetails'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/setid'))
(rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/usercontroller'), None, rdflib.term.URIRef('http://semanticBasedRedocumentation.org/class/nocontent'))
```

Figure 6.81: Ontology Data Retrieved

	Class	Relationship	Dependent
0	usercontroller	Has Dependent	deleteallusers
1	usercontroller	Has Dependent	deleteuser
2	usercontroller	Has Dependent	fetchallusers
3	usercontroller	Has Dependent	get
4	usercontroller	Has Dependent	build
5	usercontroller	Has Dependent	ispresent
6	usercontroller	Has Dependent	notfound
7	usercontroller	Has Dependent	ok
8	usercontroller	Has Dependent	finduserbyid
9	usercontroller	Has Dependent	saveuserdetails
10	usercontroller	Has Dependent	setid
11	usercontroller	Has Dependent	nocontent

Figure 6.82: Dataframe Transformed from the Ontology Data


```
[{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"deleteallusers"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"deleteuser"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"fetchallusers"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"get"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"build"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"ispresent"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"notfound"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"ok"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"finduserbyid"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"saveuserdetails"},{"Class":"usercontroller","Relationship":"Has
Dependent","Dependent":"setid"},{"Class":"usercontroller","Relationship":"Has Dependent",
    "Dependent":"nocontent"}]
```


Figure 6.83: Jsonify Data Retrieved in Postman

When finding variable and method data using user-inputted class information, the same procedure is done. Below is a diagram of the code segments used to search the variable and method.

```
@OntologyGenerator.route('/GetMethodByClass/<Class>',methods=['GET'])
def GetMethodByClass(Class):
    if Class != "":
        data = []
        Input = "<http://semanticBasedRedocumentation.org/class/" + Class +"_has_method>"
        query = "SELECT ?domain ?property ?range \n" + "WHERE { \n" + Input + " rdfs:domain ?domain ; \n" + "rdfs:range ?range . \n"  + "}"
        qres = g.query(query)
        for row in qres:
            data.append({"Class": row[0][46::], "Relationship": "Has Method" ,"Method": row[2][46::]})
        df = pd.DataFrame(data)
        json_data = df.to_json(orient='records')

        return json_data
```

Figure 6.84: Retrieve Method Data by Class User Input

```
@OntologyGenerator.route('/GetVariableByClass/<Class>',methods=['GET'])
def GetVariableByClass(Class):
    data = []
    Input = "<http://semanticBasedRedocumentation.org/class/" + Class +"_has_variable>"
    query = "SELECT ?domain ?property ?range \n" + "WHERE { \n" + Input + " rdfs:domain ?domain ; \n" + "rdfs:range ?range . \n"  + "}"
    qres = g.query(query)
    for row in qres:
        data.append({"Class": row[0][46::], "Relationship": "Has Variable" ,"Variable": row[2][46::]})
    df = pd.DataFrame(data)
    json_data = df.to_json(orient='records')

    return json_data
```

Figure 6.85: Retrieve Variable Data by Class User Input

Last but not least, in order for the user to download and upload the created ontology graph to the Web-based Visualization of Ontologies (WebVOWL) in order to examine the ontology graph, we would need to return it in file format. In order to handle the download request, a route is declared. The procedure of returning the Ontology File for the user to download is depicted in the image below.

```
@OntologyGenerator.route('/CompleteOntology',methods=['GET'])
def generateCompleteOntology():
    download_path="C:/Users/Zakaria/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication/storage/app/CompleteOntology.ttl"
    g.serialize(destination=download_path)
    return send_file(download_path, as_attachment=True)
```

Figure 6.86: Route to Download Complete Ontology File

The ontology is first serialized into a TTL file format using a machine path. The process of transforming RDF (Resource Description Framework) data from its internal representation into other formats that may be readily saved, communicated, or processed is known as serialization in RDFLib. Triples, which are assertions with subject-predicate-object relationships, make up RDF data. We can represent these triples using serialization in a number of different formats, including XML, Turtle (TTL), RDF/XML, JSON-LD, and more. The send_file function in Flask is then used to provide file download capability in the web

application in response to an HTTP request after the serialization process is complete. You can reach this service at http://127.0.0.1:5000/CompleteOntology.

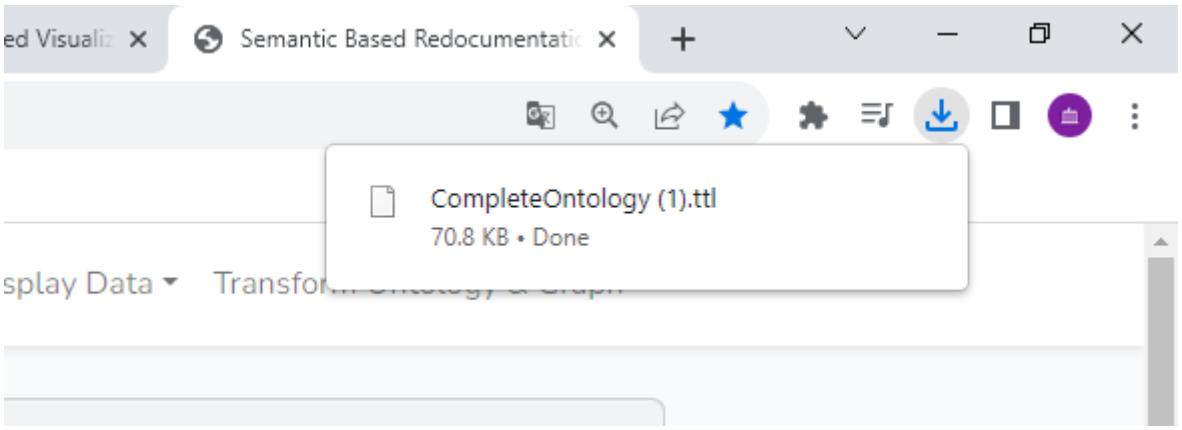


Figure 6.87: File Downloaded After Sending Request

Due to the Complete Ontology's high level of complexity, a large and complicated graph was formed, making it difficult to easily search for nodes inside it. As a result, we have chosen to divide the ontology into a number of smaller ontology graphs, which can produce a clearer and more accurate graph visualization. As indicated in the diagram below, a new route is established to handle the task of separating the ontology.

```
@OntologyGenerator.route('/DependenciesOntology',methods=['GET'])
def generateDependenciesOntology():
    ontology = []
    for Key in dependencies_file:
        csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
        csv_content = csv_object['Body'].read().decode('utf-8')
        df = pd.read_csv(StringIO(csv_content))
        ontology.append(df)

    ontology_frame = pd.concat(ontology, axis=0,ignore_index=True)
    Class = Namespace('http://semanticBasedRedocumentation.org/class/')
    Dependency_graph = Graph()

    main_ref = URIRef(Class+'Class')
    Dependency_graph.add((mainClass_ref,RDF.type,OWL.Class))

    for index,row in ontology_frame.iterrows():
        class_ref = URIRef(dependencies+row['Class'])
        dependent_ref = URIRef(dependencies+row['HasDependencies'])
        Dependency_graph.add((class_ref,RDFS.subClassOf ,mainClass_ref))
        dependent = URIRef(cls+row['Class']+'_has_dependent')
        Dependency_graph.add((dependent,RDF.type,OWL.ObjectProperty))
        Dependency_graph.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
        Dependency_graph.add((dependent,RDFS.domain, class_ref))
        Dependency_graph.add((dependent,RDFS.range, dependent_ref))

    download_path="C:/Users/Zakaria/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication/storage/app/Dependency.ttl"
    Dependency_graph.serialize(destination=download_path)
    return send_file(download_path, as_attachment=True)
```

Figure 6.88: Splitting the Ontology with Only Dependencies Data

This approach manages the ontology's division by creating a new Ontology Graph using solely Dependencies information. The procedure is the same as that used to build a full ontology graph: after retrieving the dependencies data from an S3 bucket with keys and converting it to a dataframe, predefining the namespace that will be used to create the URIRef later, creating a new instance of the RDFLib ontology graph, and beginning the process of iterating the dataframe and adding triples to the ontology graph. After the ontology graph has been built, it is serialized with our local machine's download route, allowing users to download the file by sending a request to http://127.0.0.1:5000/DependeciesOntology.

The user can download the Ontology containing the data they want with the route defined, and the same is true for the Variable and Method data. The routes that deal with the Variable and Method ontologies are displayed below.

```
@OntologyGenerator.route('/VariableOntology',methods=['GET'])
def generateVariablesOntology():
    ontology = []
    for Key in variable_file:
        csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
        csv_content = csv_object['Body'].read().decode('utf-8')
        df = pd.read_csv(StringIO(csv_content))
        ontology.append(df)

    ontology_frame = pd.concat(ontology, axis=0,ignore_index=True)

    Class = Namespace('http://semanticBasedRedocumentation.org/class/')
    Dependency_graph = Graph()

    main_ref = URIRef(Class+'Class')
    Dependency_graph.add((mainClass_ref,RDF.type,OWL.Class))

    for index,row in ontology_frame.iterrows():
        class_ref = URIRef(dependencies+row['Class'])
        variable_ref = URIRef(dependencies+row['HasVariable'])
        Dependency_graph.add((class_ref,RDFS.subClassOf ,mainClass_ref))
        dependent = URIRef(cls+row['Class']+'_has_variable')
        Dependency_graph.add((dependent,RDF.type,OWL.ObjectProperty))
        Dependency_graph.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
        Dependency_graph.add((dependent,RDFS.domain, class_ref))
        Dependency_graph.add((dependent,RDFS.range, variable_ref))

    download_path="C:/Users/Zakaria/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication/storage/app/Variable.ttl"
    Dependency_graph.serialize(destination=download_path)
    return send_file(download_path, as_attachment=True)
```

Figure 6.89: Splitting the Ontology with Only Variable Data

```
@OntologyGenerator.route('/MethodOntology',methods=['GET'])
def generateMethodsOntology():
    ontology = []
    for Key in methods_file:
        csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
        csv_content = csv_object['Body'].read().decode('utf-8')
        df = pd.read_csv(StringIO(csv_content))
        ontology.append(df)

    ontology_frame = pd.concat(ontology, axis=0,ignore_index=True)

    Class = Namespace('http://semanticBasedRedocumentation.org/class/')
    Dependency_graph = Graph()

    main_ref = URIRef(Class+'Class')
    Dependency_graph.add((mainClass_ref,RDF.type,OWL.Class))

    for index,row in ontology_frame.iterrows():
        class_ref = URIRef(dependencies+row['Class'])
        method_ref = URIRef(dependencies+row['HasMethod'])
        Dependency_graph.add((class_ref,RDFS.subClassOf ,mainClass_ref))
        dependent = URIRef(cls+row['Class']+'_has_method')
        Dependency_graph.add((dependent,RDF.type,OWL.ObjectProperty))
        Dependency_graph.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
        Dependency_graph.add((dependent,RDFS.domain, class_ref))
        Dependency_graph.add((dependent,RDFS.range, method_ref))

    download_path="C:/Users/Zakaria/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication/storage/app/Method.ttl"
    Dependency_graph.serialize(destination=download_path)
    return send_file(download_path, as_attachment=True)
```

Figure 6.90: Splitting the Ontology with Only Method Data

Finally, the front-end web application built on the Laravel Framework will receive the data return from the Flask services. In Chapter 5, the Laravel Framework's architecture is covered. Let's go over the Laravel Framework's controller and view components, which manage the data received and visualization.

Laravel first defines routes that relate URLs to particular controller functions, The 'routes/web.php' file in the Laravel project directory is normally where routes are defined. Each route includes an optional name, a controller method, and a URL. The map below depicts the declared path.

```php
Auth::routes();

Route::get('/home', [HomeController::class, 'index'])->name('home');

Route::get('/', [FilesController::class, 'index'])->name('fileupload.index');
Route::post('/multiple-file-upload', [FilesController::class, 'multipleUpload'])->name('multiple.fileupload');

Route::get('/Method', [MethodsController::class, 'index']);
Route::post('/searchMethodByClass', [MethodsController::class, 'search']);

Route::get('/Variable', [VariablesController::class, 'index']);
Route::post('/searchVariableByClass', [VariablesController::class, 'search']);

Route::get('/Dependency', [DependenciesController::class, 'index']);
Route::post('/searchDependencyByClass', [DependenciesController::class, 'search']);

Route::get('/Metric', [MetricsController::class, 'index']);

Route::view('/generateGraph', 'generate-graph');
```

Figure 6.91: Route declares to handle different URL request

According to the diagram above, every route manages various URL requests using various methods, such as the get and post methods that are bound to a method declared in the controllers. The application's logic is handled by the controllers. They take requests from routes, process the data, and then provide answers along with the appropriate view component. The 'app/Http/Controllers' folder in the Laravel project directory, which is depicted in the figure below, is where controllers are kept.

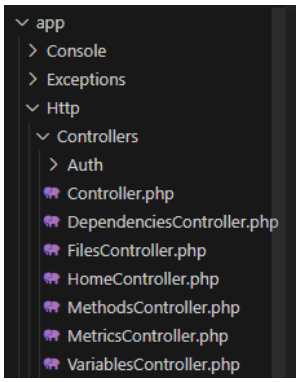


Figure 6.92: Different Controller Has been Created to handle different request

Other than that, users can access data thanks to view components. They are often created using Laravel's template engine, Blade. Views can use Blade syntax to display data that is sent by the controller. These Blade view components are kept in the Laravel project directory 'resources/views' folder, as seen in the figure below.

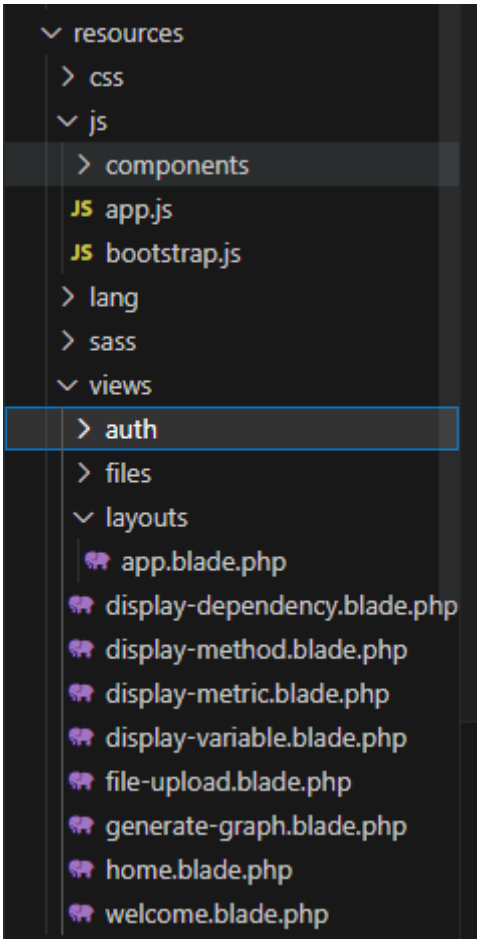


Figure 6.93: Blade View Components

Let's talk about the entire web application process, starting with the file-upload controller and the rendering of the file-upload page by the view components.

The Blade engine, which offers a practical and expressive approach to deal with views and produce dynamic HTML output, is used to write the view components. By separating the presentation layer from the application logic, it enables us to better organize and manage the code. The page-upload blade template is shown in the Appendix E. Which, if the problem was caught in the controller, will show the error and return to the view template using the blade engine's @if syntax. In addition, there is a form to manage file uploads using the "POST" method, which passes the uploaded files to the controller function with the URL declared in the "action" attribute and the previously described route file. The notification will appear on the website after the file is uploaded successfully; it will be in the same div as the error prompt but will have a different colour and message. Additionally, the details of the uploaded file will also appear on the page that will be displayed later.

```php
public function multipleUpload(Request $request)
{
    $this->validate($request, [
        'fileuploads' => 'required'
    ]);
    $files = $request->file('fileuploads');
    foreach($files as $file){
        if($file->getClientOriginalExtension() != "java"){
            return redirect()->route('fileupload.index')->with('failed','Invalid File Extension');
        }
    }

    foreach($files as $file){
        $fileUpload = new File;
        $fileUpload->filename = $file->getClientOriginalName();
        $path = Storage::disk('s3')->putFileAs('files/', $file, $fileUpload->filename);
        // $path = $file->store('public/uploads');
        $fileUpload->filepath = Storage::disk('s3')->url($path);
        $fileUpload->type= $file->getClientOriginalExtension();
        $fileUpload->save();
    }

    return redirect()->route('fileupload.index')->with('success','Files uploaded successfully!');

}
```

Figure 6.94: multipleUpload Method in FilesController

The request will be passed into this method to manage the file upload procedure when the submit event has been triggered in the blade view component. We check to see if the required attribute is present in the request first; if it is not, the validation will throw an error to the view component. The next step is to perform a second validation that checks the file extension because Databricks only handles the lexical analysis of Java Source Code; if the uploaded file is not Java source code, it will return an error message to the view component because we are handling multiple uploads. We obtain the information from each file and save it in the database using iteration. Other than that, we upload each file into the S3 into the files/folder within the S3 bucket and save their path into the database using the S3 disk that was explained and configured in Chapter 5. The controller will redirect the pages with a specified URL and send back the success message that displays in the view component after all the files have been successfully uploaded.

```php
public function index()
{
    //
    $fileUplaods = File::get();
    return view('file-upload', ['fileUploads' => $fileUplaods]);
}
```

Figure 6.95: Index Method in FilesController

The data for the index function will come from the File Model, which contains the application's data and business logic. They frequently use an Object-Relational Mapping (ORM) like Eloquent to connect with the database. The 'app' directory is normally where models can be found. as depicted in the following figure.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class File extends Model
{
    protected $table = 'files';

    protected $fillable = [
        'user_id',
        'name',
        'type',
        'size'
    ];
    use HasFactory;
}
```

Figure 6.96: File Model

With the help of the model, we are able to get the data out of the File database table and return it to the view. Because of this, the view is able to loop through the data return and present the information as a table.

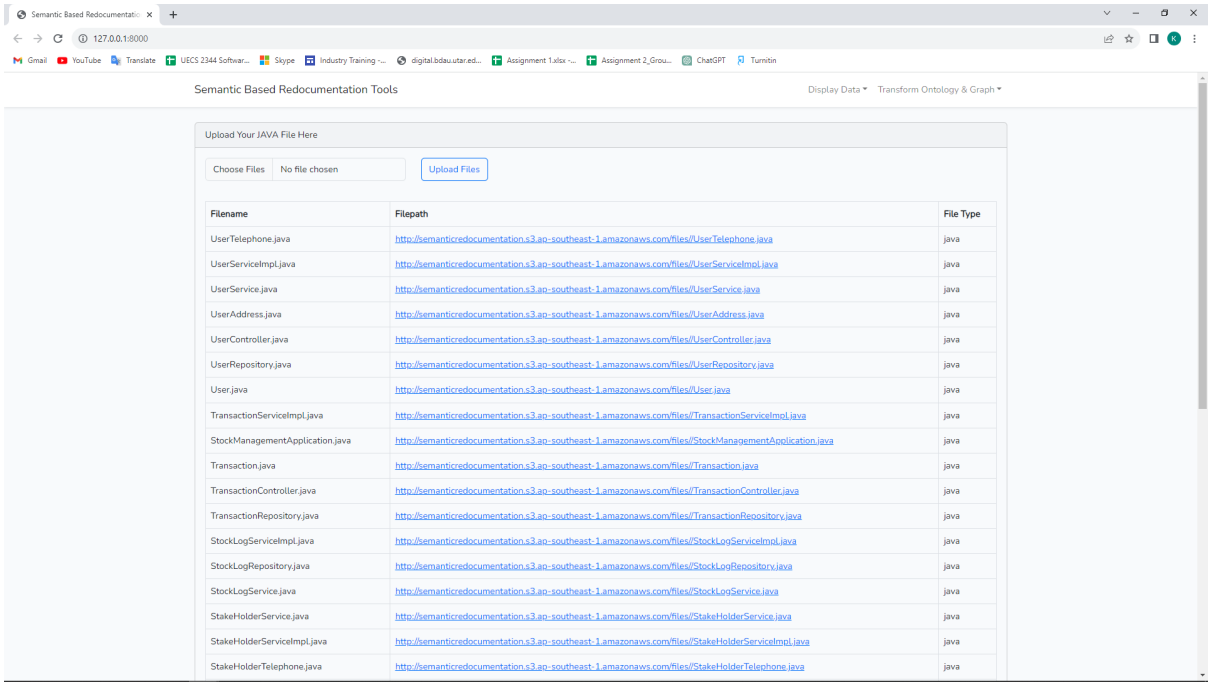


Figure 6.97: Output of the file-upload blade template with retrieved data

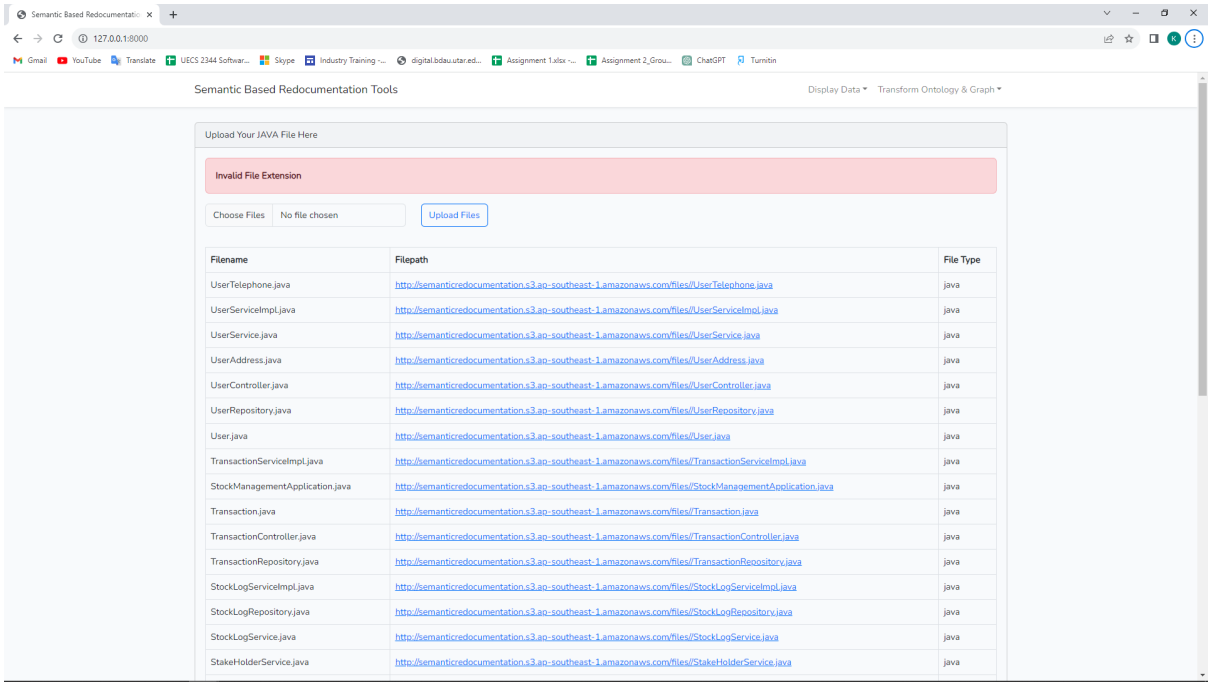


Figure 6.98: Error Prompt of the file-upload blade template with invalid file extension

The Lambda Function will start the Databricks workflow when the files have been uploaded to the S3 bucket, and the output of the analysis will then be saved back into the S3 bucket. The data will then be read from the S3 bucket and further transformed while the Flask API is operating, creating an ontology. The web application is then linked to the Flask

Service via the controller, which returns the data that has been transformed into an ontology and is then displayed.

```php
class VariablesController extends Controller
{
    //
    public function index()
    {
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllVariable';

        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-variable', ['Variables' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-variable', ['error' => $e->getMessage()]);
        }

    }
```

Figure 6.99: Index Function of VariablesController

The code snippets above demonstrate the VariablesController's index function, which returns the data returned by the Flask API. Guzzle is used in the Laravel Framework to send HTTP queries to external APIs. Guzzle HTTP client must first be installed using the composer command "composer require "guzzlehttp/guzzle". The Laravel Controllers, which must import the Guzzle namespace at the top of the project, can then use Guzzle:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Variable;
use GuzzleHttp\Client;
```

Figure 6.100: GuzzleHTTP Client used

The "Client" class instance is then created in the index method, where it will be used to send HTTP requests. The URL is then specified in the index function and saved into the $apiURL variable, which is used in the subsequent steps. The next step is to perform a GET

request to the URL defined previously using the get method of the "Client" instance, saving the response into the $response variable. We can get a variety of information from the response object after submitting the request. We retrieve the contents of the raw response body in the code segment above. In order to give the response data to the "display-variable" view components, we must first decode it using the json_decode method because the data is in JSON format. The function will throw an exception error code to the view component and display it on the web page if it encounters an exception.

After the data is passed to the view, the view will be in charge of rendering the data by iterating through the data array using the @foreach function from the blade engine and displaying the data in table format.

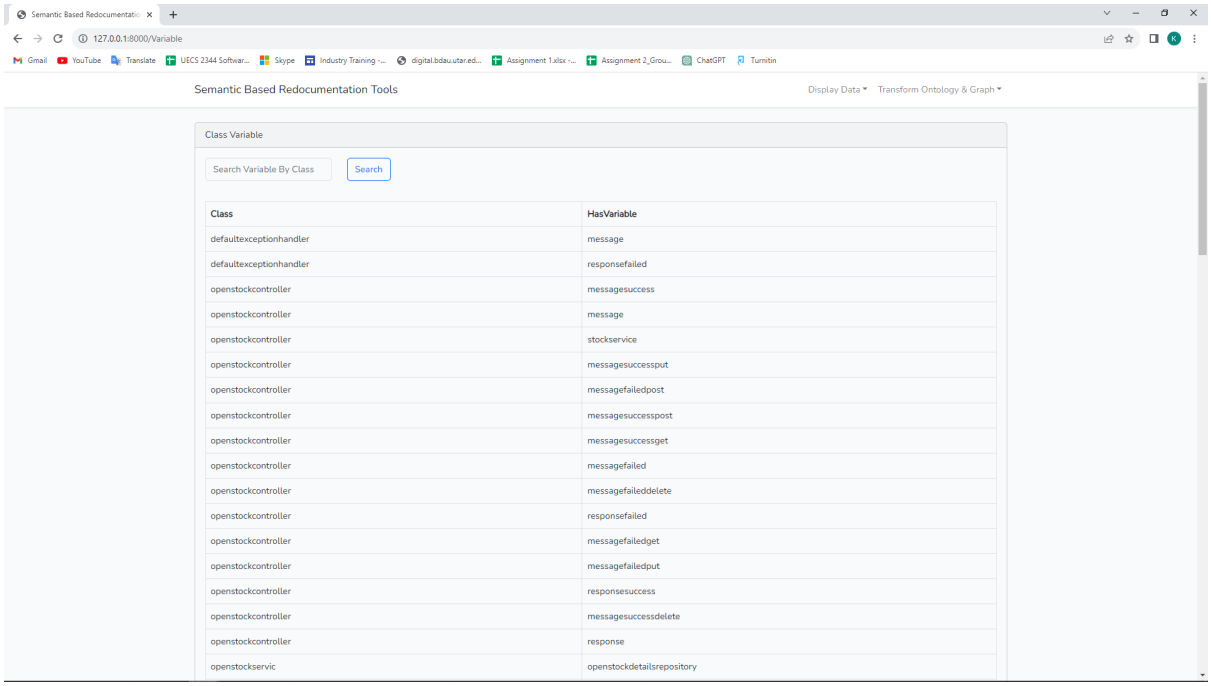


Figure 6.101: Variable Data Display

We can see every class and the accompanying variables in the diagram above. In addition, there is an input field where the user can type in a class name to search the ontology graph. Only the variable data with the user-inputted class name will be returned by the search procedure. In order to access only the essential data, the VariableController handles the searching function and connects to the Flask API in the code segments below.

```php
public function search(Request $req){
    if($req->className){
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetVariableByClass/'.''.$req->className;
        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-variable', ['Variables' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-variable', ['error' => $e->getMessage()]);
        }
    }
    else{
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllVariable';

        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-variable', ['Variables' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-variable', ['error' => $e->getMessage()]);
        }
    }

}
```

Figure 6.102: Search Function in Variable Controller

The code section above initially processes the input by using a conditional statement to see if the user entered any data into the field. If not, the entire variable data for all classes will be returned. After that, it proceeds in the same manner as the index function described earlier, establishing a client object and setting the apiURL in advance. The class name will be attached to the end of the apiURL, which serves as the argument to fetch in the Flask API, if the user entered the class name into the input field. The next step, which involves initiating a request, retrieving its contents from the body, and returning the data to the view component and display, is nearly identical. The results of the search are shown in the figure below.

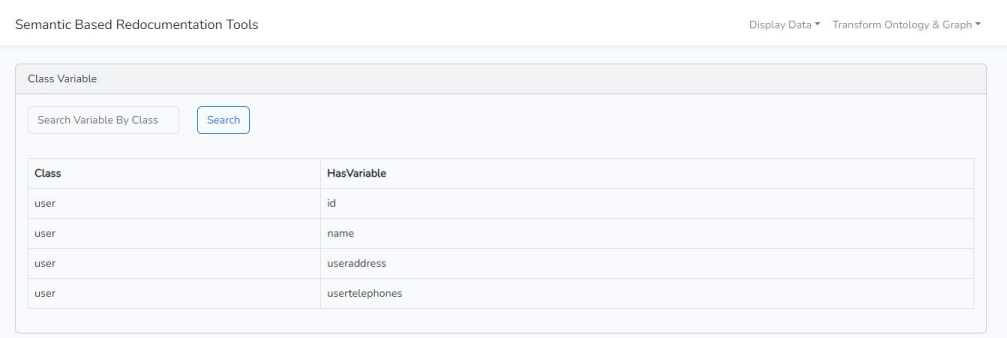


Figure 6.103: Result Display by Searching “user”

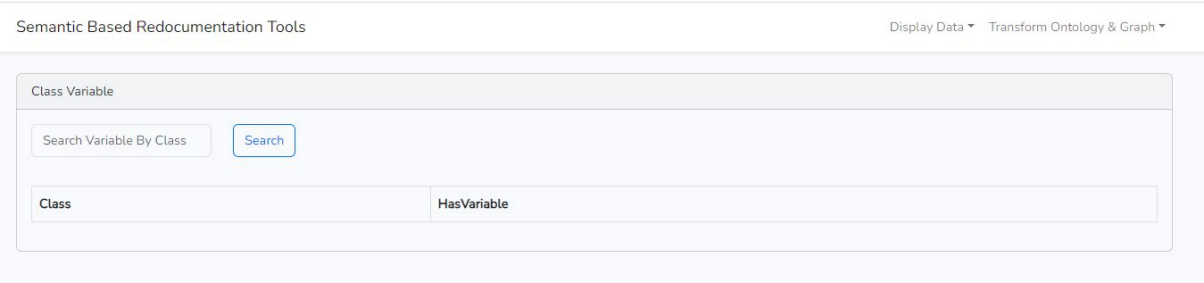


Figure 6.104: Empty Result when the class name does not exist

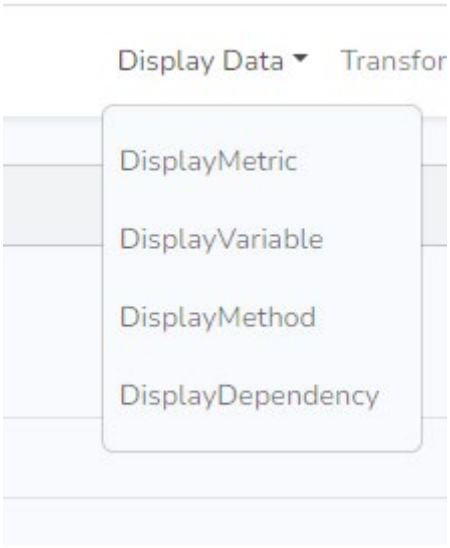


Figure 6.105: Different Data Display

Although there are many data displays, the controller's methods are essentially the same. The apiURL used to connect to the Flask API is the only variation. As a result, the code segments and display results for various purposes will be depicted in the picture below and share the same ideas as the variable retrieve process.

```php
class MethodsController extends Controller
{
    //
    public function index(Request $req){
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllMethod';

        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-method', ['Methods' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-method', ['error' => $e->getMessage()]);
        }
    }
```

Figure 6.106: Index Method in MethodsController

```php
public function search(Request $req){
    if($req->className){
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetMethodByClass/'.''.$req->className;
        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-method', ['Methods' => $dataArray]);
        }catch(\Excrption $e){
            return view('display-method', ['error' => $e->getMessage()]);
        }
    }
    else{
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllMethod';

        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-method', ['Methods' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-method', ['error' => $e->getMessage()]);
        }
    }
```

Figure 6.107: Search Method in MethodsController

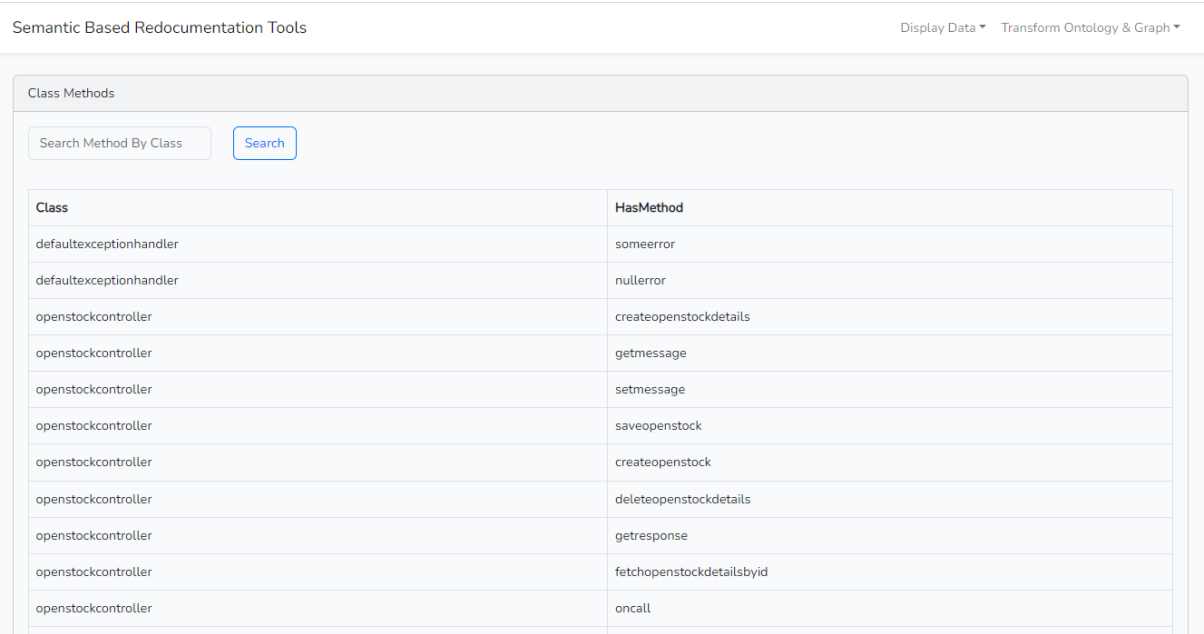


Figure 6.108: Method Data Display

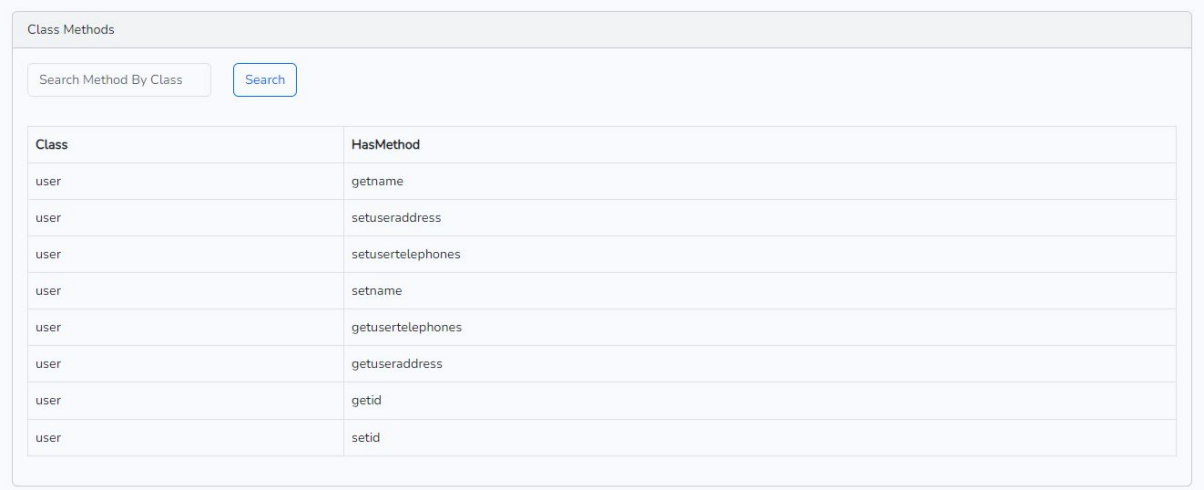


Figure 6.109: Method Data Display by searching “user”

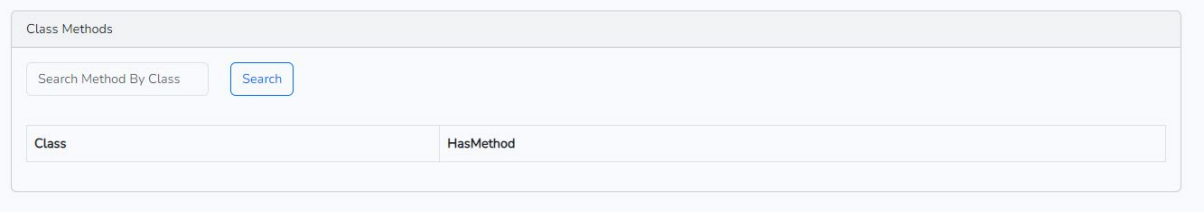


Figure 6.110: Method Data Display by searching non-existing class name

```php
class DependenciesController extends Controller
{
    //
    public function index()
    {
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllDependencies';
        try {
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-dependency', ['Dependencies' => $dataArray]);
        } catch(\Excrption $e){
            return view('display-dependency', ['error' => $e->getMessage()]);
        }

    }
```

Figure 6.111: Index Method in DependenciesController

```php
    public function search(Request $req){
        if($req->className){
            $client = new Client();
            $apiURL = 'http://127.0.0.1:5000/GetDependenciesByClass/''.$req->className;
            try {
                $response = $client->get($apiURL);
                $jsonResponse = $response->getBody()->getContents();
                $dataArray = json_decode($jsonResponse, true);
                return view('display-dependency', ['Dependencies' => $dataArray]);
            } catch(\Excrption $e){
                return view('display-dependency', ['error' => $e->getMessage()]);
            }
        }
        else{
            $client = new Client();
            $apiURL = 'http://127.0.0.1:5000/GetAllDependencies';
            try {
                $response = $client->get($apiURL);
                $jsonResponse = $response->getBody()->getContents();
                $dataArray = json_decode($jsonResponse, true);
                return view('display-dependency', ['Dependencies' => $dataArray]);
            } catch(\Excrption $e){
                return view('display-dependency', ['error' => $e->getMessage()]);
            }
```

Figure 6.112: Search Method in DependenciesController

```
@section('content')
<div class="container">
    <div class="card">
        <div class="card-header">
            Class Dependencies
        </div>
        <div class="card-body">
            <form class="row" method="post" action="{{url('searchDependencyByClass')}}">
                @csrf
                <div class="col-auto">
                    <input type="search" class=" form-control" placeholder="Search Dependency By Class" name="className">
                </div>
                <div class="col-auto">
                    <button type="submit" class="btn btn-outline-primary mb-3">Search</button>
                </div>
            </form>
            <table class="table table-bordered mt-3">
                <thead>
                    <tr>
                        <th>Class</th>
                        <th>HasDependencies</th>
                    </tr>
                </thead>
                <tbody>
                @foreach ($Dependencies as $dependency)
                    <tr>
                        <td>{{ $dependency['Class'] }}</td>
                        <td>{{ $dependency['Dependent'] }}</td>
                    </tr>
                @endforeach
                </tbody>

            </table>
        </div>
```

Figure 6.113: display-dependencies blade view component

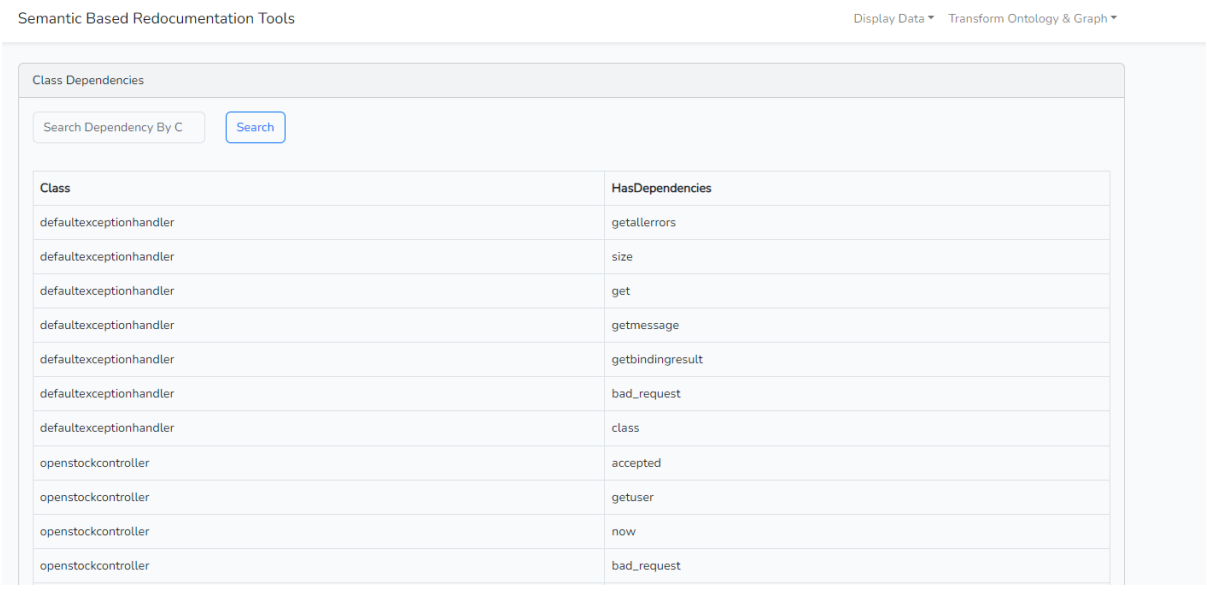


Figure 6.114: Dependencies Data Display

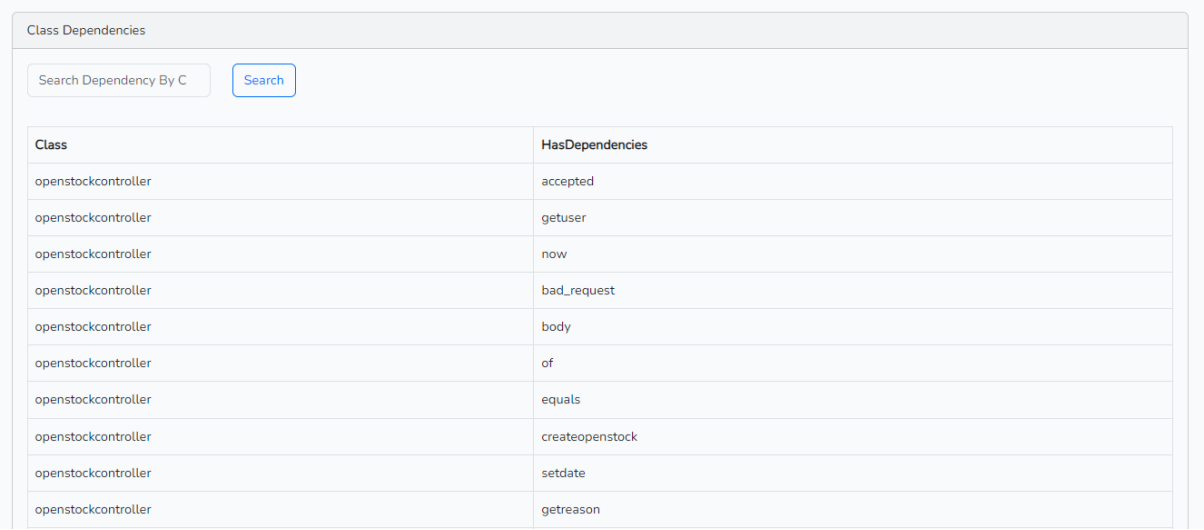


Figure 6.115: Dependencies Data Display by searching “openstockcontroller”

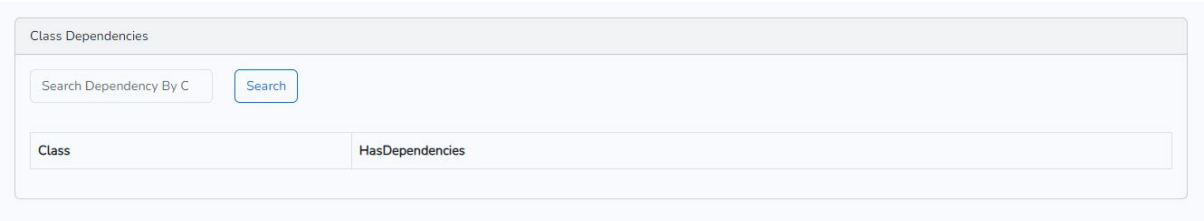


Figure 6.116: Dependencies Data Display by searching non-existing class name

```php
class MetricsController extends Controller
{
    //
    public function index()
    {
        $client = new Client();
        $apiURL = 'http://127.0.0.1:5000/GetAllMetric';

        try{
            $response = $client->get($apiURL);
            $jsonResponse = $response->getBody()->getContents();
            $dataArray = json_decode($jsonResponse, true);
            return view('display-metric', ['Metrics' => $dataArray]);
        }catch(\Excrption $e){
            return view('display-metric', ['error' => $e->getMessage()]);
        }

    }

}
```

Figure 6.117: Index Method in MetricsController

```blade
@section('content')
<div class="container">
    <div class="card">
    <div class="card-header">
            Metrics
        </div>
        <div class="card-body">
            <table class="table table-bordered mt-3">
                <thead>
                    <tr>
                        <th>Attribute</th>
                        <th>Quantity</th>
                    </tr>
                </thead>
                <tbody>
                @foreach ($Metrics as $metric)
                    <tr>
                        <td>{{ $metric['Class'] }}</td>
                        <td>{{ $metric['Attribute'] }}</td>
                    </tr>
                @endforeach
                </tbody>

            </table>
        </div>
    </div>

</div>
@endsection
```

Figure 6.118: display-metric blade view component

Figure 6.119: Metrics Data Display

Last but not least, the created ontology can be downloaded as a file. Using the WebVOWL tools, you can choose from a variety of ontology types to download and create a graph from.

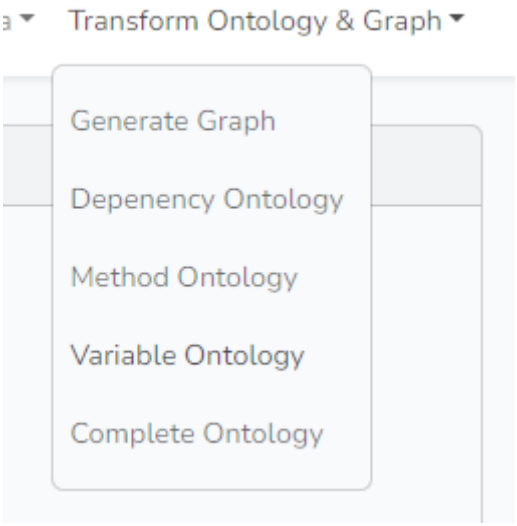


Figure 6.120: Dropdown Navigation Bar

There are numerous options to download the ontology files, including dependency, method, variable, and whole ontology, from the dropdown navigation bar above. When you click the generate graph button after downloading the file, step-by-step directions for creating the graph will then display in the page.

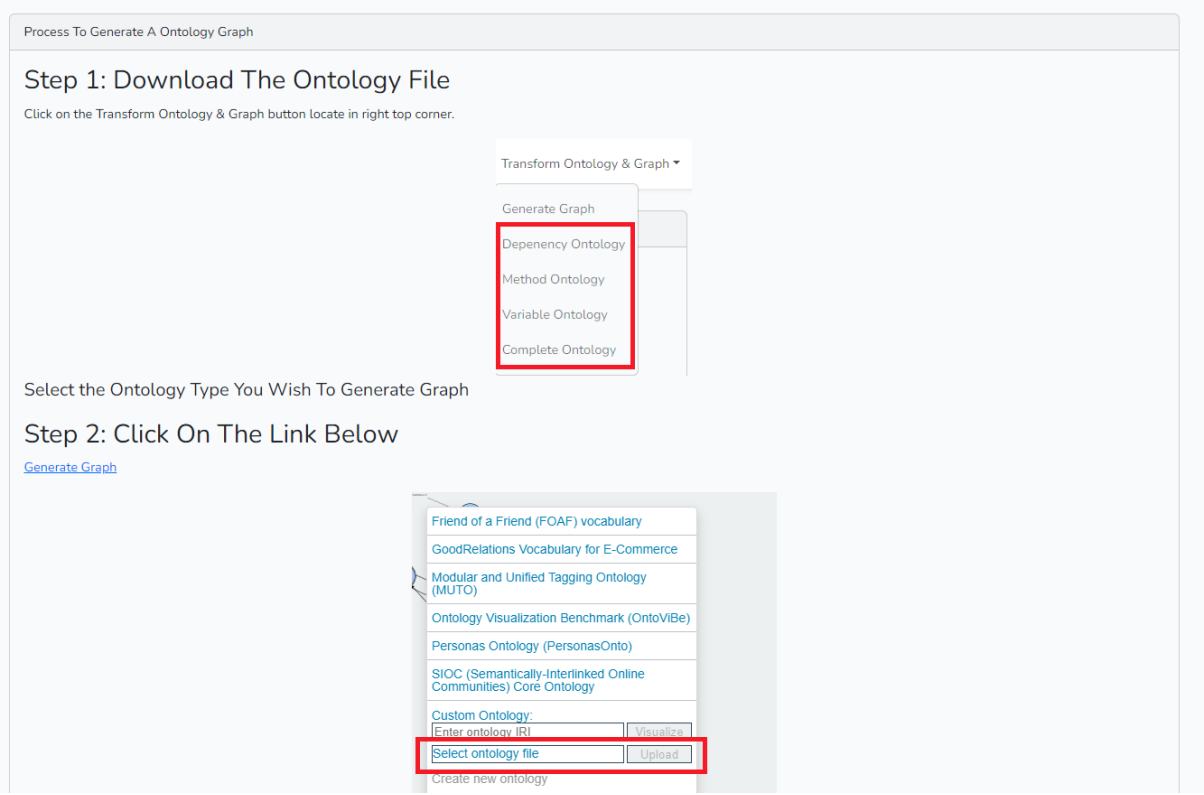


Figure 6.121: Step 1&2 To Generate Ontology Graph

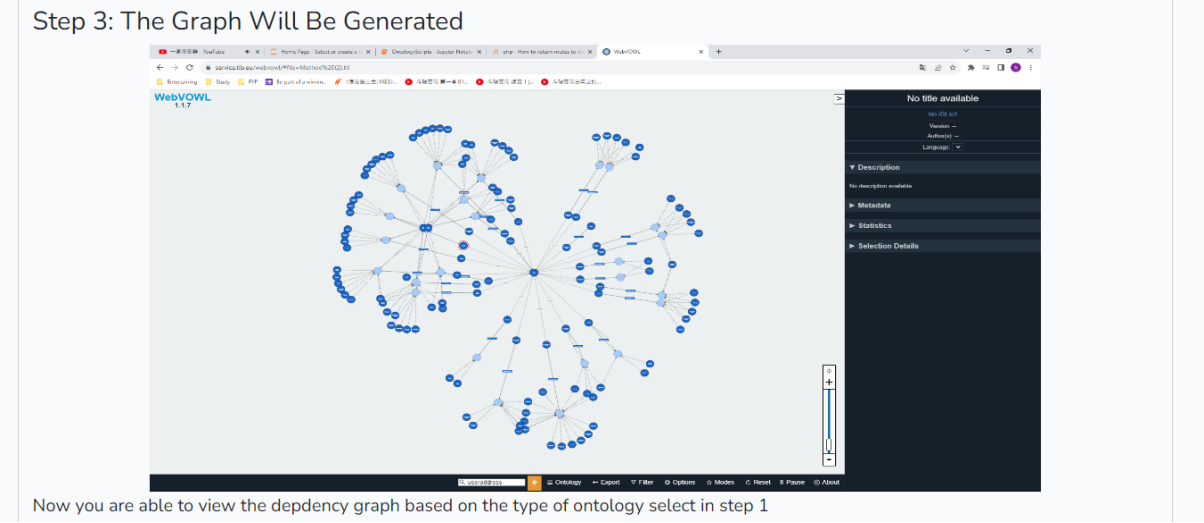


Figure 6.122: Step 3 to generate Ontology Graph

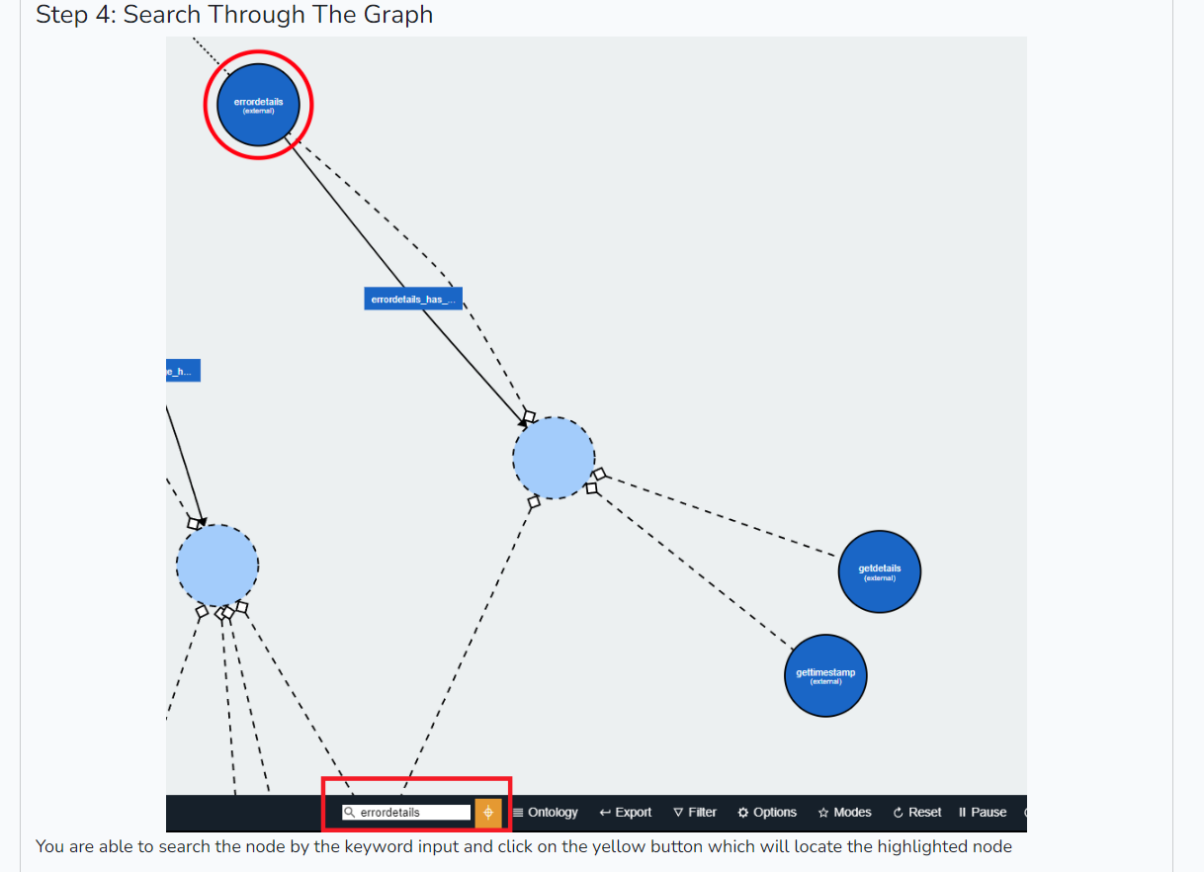


Figure 6.123: Step 4 to search in the ontology graph

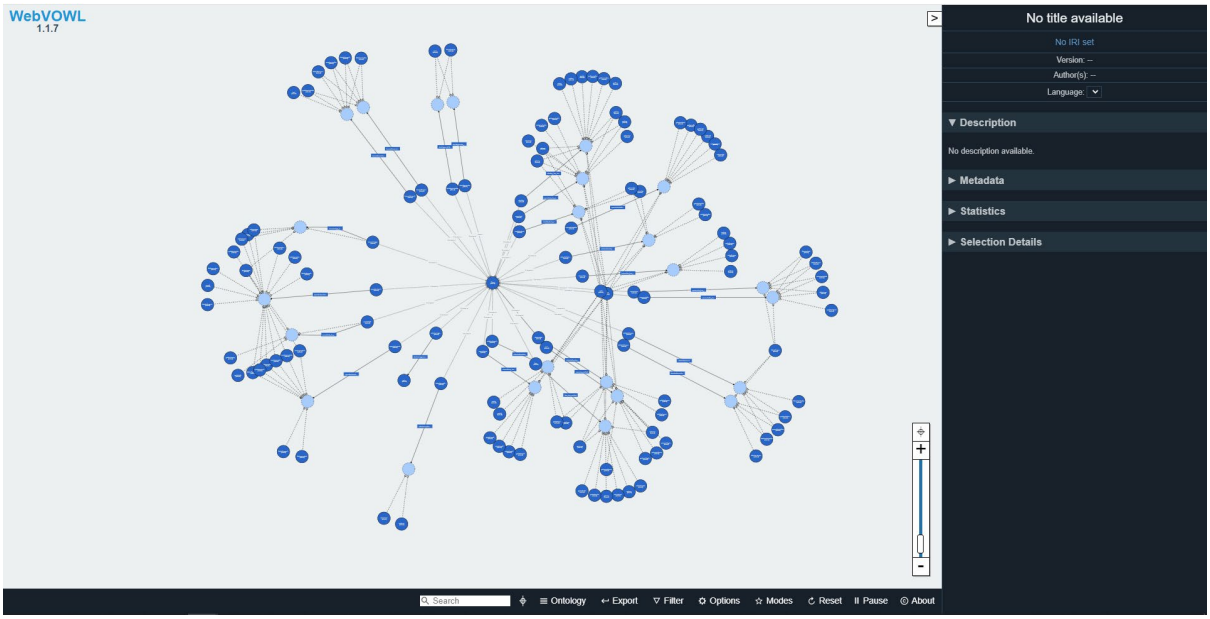


Figure 6.124: Example Output generated for method ontology graph

6.4 System Deployment

Git provides effective version control for this online application, which is hosted on GitHub Repositories. Git is a well-liked version control system that enables developers to effortlessly manage several project versions, track changes, and collaborate on code. Git makes it simple to integrate code updates and improvements while maintaining a well-organized development process.

The project repositories are housed on GitHub, a platform for collaboration and version control on the internet. A central location for developers to store, manage, and share their code repositories is provided by GitHub. With features like pull requests, branching, and issue tracking, it enables productive teamwork and makes it simpler to evaluate and integrate code changes.

```
Hiew Khai Hang@DESKTOP-L95UO39 MINGW64 ~/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication (master)
$ git add .

Hiew Khai Hang@DESKTOP-L95UO39 MINGW64 ~/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   app/Http/Controllers/FilesController.php


Hiew Khai Hang@DESKTOP-L95UO39 MINGW64 ~/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication (master)
$ git commit -a -m "remove comment of code segments"
[master 161f26e] remove comment of code segments
 1 file changed, 9 insertions(+), 9 deletions(-)

Hiew Khai Hang@DESKTOP-L95UO39 MINGW64 ~/Desktop/Study Materials/Y4S1/FYP/LaravelWebApplication (master)
$ git push origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 596 bytes | 596.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/KhaiHang1219/SemanticBasedRedocumentation.git
   7723c4e..161f26e  master -> master
```

Figure 6.125: Some of the Git Action

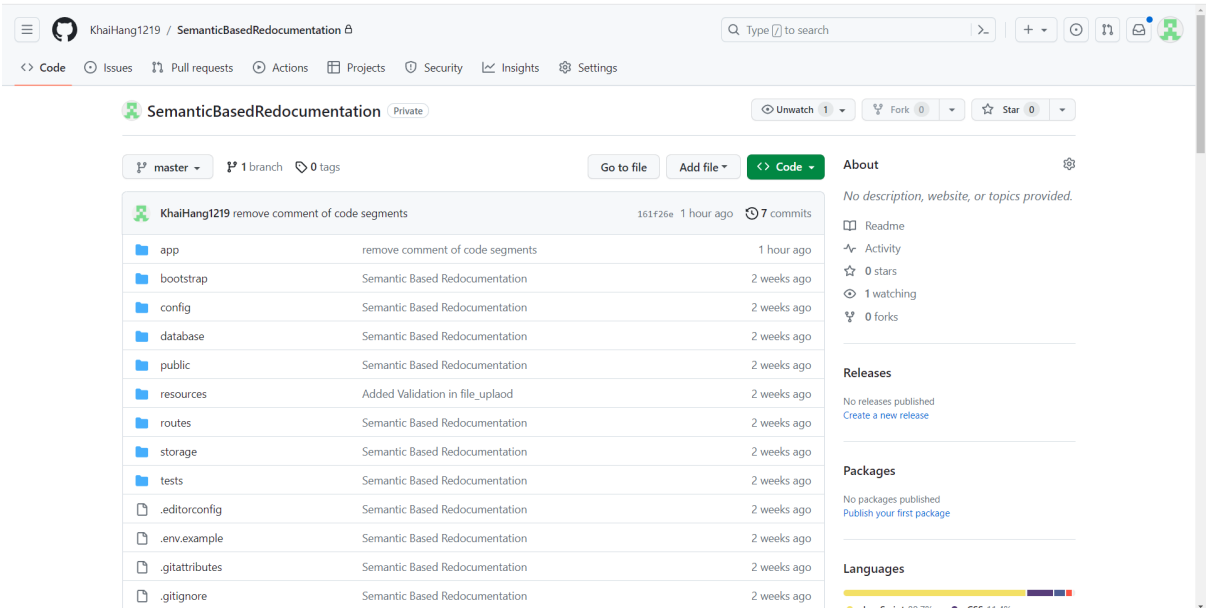


Figure 6.126: GitHub Repositories

6.5 Conclusion

This chapter concludes with an overview of the system implementation procedure. The project setup process starts with setting up the AWS and Azure accounts that will be used to leverage Amazon S3, Amazon Lambda Function, and Azure Databricks Platform for file storage and the use of Distributed Processing Techniques. This chapter also demonstrates how the Laravel project setup may be configured, as well as how a web application can interface with Amazon S3 services and immediately launch an AWS Lambda Function to run a Databricks workflow. The installation of necessary dependencies and tools is then covered in the chapter.

Extraction, Transformation, Store Data, and Ontology Transformation Module are the different system modules. Each module's implementation process is described in depth, including the use of various tools and proof-of-code segments. Each module result has been displayed since the module was implemented. Not to mention that the web application and Python Flask program are hosted on GitHub repositories and use Git for efficient version control, giving them a central location for development management and communication.

CHAPTER 7
SYSTEM TESTING

7.1 Introduction

Unit testing, integration testing, and usability testing were all conducted in this chapter to ensure that both the functional and the non-functional needs of the software were met in this project.

7.2 Unit Testing

In this project, unit testing is used to manually test each function to make sure the web application complies with the required definition. This method ensures that all functional and non-functional requirements are met while also assisting in the functionality verification of the applications. Other than that, this method will be conducted with manual testing and API testing with Postman. Postman will be act as a tool that helps in API Testing. Postman is a popular collaboration platform and toolset for testing, developing and documenting APIs. It provides a user-friendly interface that allows developers, testers, and API consumers to interact with APIs and perform various tasks related to API development and testing. Postman simplifies the process of making API requests, inspecting responses, and automating API workflows. Hence, the test cases and result of the Flask API testing and their accompanying findings are summarized below.

7.2.1 Unit Testing for Extraction Module

Table 7.1: Unit Testing for Extraction Module

Test Module	**Extraction Module**		**Test Title**	**File Upload from the Web Application**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-001	Upload Valid Source Code	1. Select the files the user wishes to upload for analyse purpose. 2. Click open on the file browsing console. 3. Click Upload Files button to proceed	I. Java Source Code Files	The source code files have been uploaded to S3 bucket and return the upload successfully message.	**Pass**
UNIT-002	Upload Invalid Source Code	1. Select files with invalid file extension the user wishes to upload for analyse	I. Source Code Files with other extension except from Java.	The source code files will not be uploaded to S3 bucket and return the Upload fail message with Invalid	**Pass**

		purpose. 2. Click open on the file browsing console. 3. Click Upload Files button to proceed.		File Extension.	
UNIT-003	Mounting S3 Bucket into Databricks DBFS	1. The files uploaded in the S3 Bucket are retrieve and mounted in Databricks DBFS	No Test Data	The S3 bucket which contains the source code files is mounted in Databricks DBFS and be access by Databricks Notebook	**Pass**

7.2.2 Unit Testing for Transformation Module

Table 7.2: Unit Testing for Transformation Module

Test Module	**Transformation Module**		**Test Title**	**Source Code Transformation Process**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-004	Extract the information from the Java Source Code File and perform pre-processing which filtering those unnecessary line, noisy data and comments in the source code.	1. Mapping the line retrieve from the source code into lower case 2. Trimming the line by removing the leading and trailing whitespace characters 3. Filter out the line with some specific character which represents the comment line and remove some line with special	No Test Data	A clean collection of the source code line has been retrieving and save into a variable for further process to retrieve Dependency, Method, Variable and Metrics data.	**Pass**

		word which indicates the unnecessary line and noisy data. 4. Retrieve the class name by using conditional statement and add a new column beside each of the line in the collection which representing the line is belong to which class. 5. Remove Those line without class name as noisy data.			

UNIT-005	Extract the information from the Collection retrieve in the pre-processing phase and retrieving **Dependency** data, transform into data frame.	1. Using Conditional Statement, insert a new column in the collection which represents the method name. 2. Using the collection, find the dependent method in the line with corresponding class and method using specific character and save into a new collection. 3. Remove those line with empty	No Test Data	The Dataframe result retrieved showing the relationship between the class, function and variable with 3 columns, Class, Function and HasDependencies.	**Pass**

		method name. 4. Transform into Dataframe. 5. Iterate the process until there is no specific characters occurs in the Cut column of the collections. During the iteration process, each iteration produces a Dataframe with same columns. 6. Merge the Dataframe and remove unnecessary			

		columns.			
UNIT-006	Extract the information from the Collection retrieve in the pre-processing phase and retrieving **Variable** data, transform into data frame.	1. Filtering out the line with specific string which represent the type of the variable. 2. Retrieve only the variable name with substring function. 3. Construct a new collection which storing the class name and corresponding variable. 4. Remove element with empty variable name. 5. Remove duplicate element occurs in	No Test Data	The Dataframe result retrieved showing the relationship between the class and variable with 2 columns, Class and HasVariable.	**Pass**

		the collection. 6. Transform Into DataFrame with the collection retrieved.			
UNIT-007	Extract the information from the Collection retrieve in the pre-processing phase and retrieving **Method** data, transform into data frame.	1. Filtering out the line with specific characters and string which represents the line contains a method. 2. Retrieve only method name with substring function. 3. Construct a new collection which stores the class and its corresponding	No Test Data	The Dataframe result retrieved showing the relationship between the class and method with 2 columns, Class and HasMethod.	**Pass**

		method name. 4. Transform into Dataframe			
UNIT-008	Extract the information from the Collection retrieve in the pre-processing phase and retrieving **Metric** data, transform into data frame.	1. Perform filtering process with different condition to retrieve the ImportClass line, Class line, Interface line, and Function line. 2. After the filtering process, using distinct to remove duplicate elements occurs in each of the collection. 3. Using Count function to get the	No Test Data	The Dataframe result retrieved showing the relationship between the Attribute and quanitty with 2 columns, Attribute and Quantity. The attributes contain Total Line of Code, Total Class, Total Method, Total Interface and Total Imported Library.	**Pass**

		quantity for each of the attribute. 4. Create a new collection to save the attributes and corresponding quantity. 5. Transform into Dataframe.			

7.2.3 Unit Testing for Store Data Module

Table 7.3: Unit Testing for Store Data Module

Test Module	**Store Data Module**		**Test Title**	**Writing the Data from the DataFrame into CSV File**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-009	Writing Variable Dataframe into CSV and save into S3 Bucket.	1. Using The Variable Dataframe Generated from the Transformation Module, write the data into CSV and save into mounted S3 bucket in DBFS.	No Test Data	The variableCSV file should save in the DBFS and sync to the S3 bucket in the AWS S3 console.	**Pass**
UNIT-010	Writing Method Dataframe into CSV and save into S3 Bucket.	1. Using The Method Dataframe Generated from the	No Test Data	The methodCSV file should save in the DBFS and sync to the S3 bucket in the AWS S3 console.	**Pass**

		Transformation Module, write the data into CSV and save into mounted S3 bucket in DBFS.			
UNIT-011	Writing Dependencies Dataframe into CSV and save into S3 Bucket.	1. Using The Dependencies Dataframe Generated from the Transformation Module, write the data into CSV and save into mounted S3 bucket in DBFS.	No Test Data	The dependenciesCSV file should save in the DBFS and sync to the S3 bucket in the AWS S3 console.	**Pass**
UNIT-012	Writing Metrics Dataframe into CSV and save into S3 Bucket.	1. Using The Metrics Dataframe Generated from	No Test Data	The metricsCSV file should save in the DBFS and sync to the S3 bucket in the	**Pass**

		the Transformation Module, write the data into CSV and save into mounted S3 bucket in DBFS.		AWS S3 console.	

7.2.4 Unit Testing for Ontology Transformation Module

Table 7.4: Unit Testing for Ontology Transformation Module

Test Module	**Ontology Transformation Module**		**Test Title**	**Retrieve CSV File Data and Transform into Ontology**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-013	Retrieve the CSV Files saved in S3 bucket and get the content of those CSV files.	1. Create a boto3 client and set up configuration of the boto3 to connect S3 bucket. 2. Retrieve all the keys with CSV file extension and defined prefix. 3. Add the key to corresponding array created. 4. Iterate through the arrays which contains the key to access the content of the files for different purposes and get the content of the file with the key. 5. Save the content into an array and concatenate the	No Test Data	The are 4 Dataframe generated which include Variable, Method, Dependency and Metric with the Key retrieve by the boto3 client.	**Pass**

		array become a complete Python Dataframe.			
UNIT-013	Using the Variable Dataframe, construct an ontology graph and add the data of the variable dataframe into ontology graph as triple.	1. Iterate Through the Variable Dataframe, creating unique URI reference with the value of the dataframe columns. 2. Creating triple with the unique URI reference and add into the ontology graph. 3. Creating unique URI reference for the relationship of the class and variable as object property 4. Creating triples with the relationship URI reference with corresponding domain (class URI ref) and range (variable URI ref)	No Test Data	The Ontology graph should have classes nodes and corresponding Variable with relationship (Object Property)	**Pass**
UNIT-014	Using the Method Dataframe, construct an	1. Iterate Through the Method Dataframe, creating unique	No Test Data	The Ontology graph should have classes nodes and	**Pass**

	ontology graph and add the data of the variable dataframe into ontology graph as triple.	URI reference with the value of the dataframe columns. 2. Creating triple with the unique URI reference and add into the ontology graph. 3. Creating unique URI reference for the relationship of the class and method as object property 4. Creating triples with the relationship URI reference with corresponding domain (class URI ref) and range (method URI ref)		corresponding Methods with relationship (Object Property)	
UNIT-015	Using the Dependencies Dataframe, construct an ontology graph and add the data of the variable dataframe into ontology graph as triple.	1. Iterate Through the Dependencies Dataframe, creating unique URI reference with the value of the dataframe columns. 2. Creating triple with the	No Test Data	The Ontology graph should have classes nodes and corresponding dependencies with relationship (Object Property)	**Pass**

		unique URI reference and add into the ontology graph. 3. Creating unique URI reference for the relationship of the class and dependency as object property 4. Creating triples with the relationship URI reference with corresponding domain (class URI ref) and range (dependency URI ref)			
UNIT-016	Using the Metrics Dataframe, construct an ontology graph and add the data of the variable dataframe into ontology graph as triple.	1. Iterate Through the Metrics Dataframe, creating unique URI reference with the value of the dataframe columns. 2. Creating triple with the unique URI reference and add into the ontology graph. 3. Creating unique URI	No Test Data	The Ontology graph should have attribute nodes and corresponding quantity with relationship (Object Property)	**Pass**

		reference for the relationship of the attribute and quantity as object property 4. Creating triples with the relationship URI reference with corresponding domain (attribute URI ref) and range (quantity URI ref)			
Test Module	**Ontology Transformation Module**		**Test Title**	**Retrieve all data from Ontology**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-017	Using the constructed Ontology graph, return all the Variable data by using SPARQL and Flask.	1. Sending Request to corresponding URL	No Test Data	The classes and corresponding variable data will be return as JSON format data.	**Pass**
UNIT-018	Using the constructed Ontology graph, return all the Method data by using SPARQL and Flask.	1. Sending Request to corresponding URL	No Test Data	The classes and corresponding method data will be return as JSON format data.	**Pass**
UNIT-019	Using the constructed Ontology graph, return all	1. Sending Request to corresponding URL	No Test Data	The classes and corresponding dependencies	**Pass**

	the Dependencies data by using SPARQL and Flask.			data will be return as JSON format data.	
UNIT-020	Using the constructed Ontology graph, return all the Metric data by using SPARQL and Flask.	1. Sending Request to corresponding URL	No Test Data	The attributes and corresponding quantity data will be return as JSON format data.	**Pass**
Test Module	**Ontology Transformation Module**		**Test Title**	**Retrieve data from Ontology by user input class name**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
UNIT-021	Using the constructed Ontology graph, return the Variable data by using SPARQL and Flask with the class name input by the user.	1. User Input the class name. 2. Sending Request to corresponding URL with the class name	Class name: usercontroller	The class with the user input class name and corresponding variable data will be return as JSON format data.	**Pass**
UNIT-022	Using the constructed Ontology graph, return the Method data by using SPARQL and Flask with the class name input by the user.	1. User Input the class name. 2. Sending Request to corresponding URL with the class name	Class name: usercontroller	The class with the user input class name and corresponding method data will be return as JSON format data.	**Pass**

UNIT-023	Using the constructed Ontology graph, return the Dependencies data by using SPARQL and Flask with the class name input by the user.	1. User Input the class name. 2. Sending Request to corresponding URL with the class name	Class name: usercontroller	The class with the user input class name and corresponding dependencies data will be return as JSON format data.	**Pass**

7.3 Integration Testing

Verifying how distinct software program modules or components interact with one another is the goal of software testing, sometimes known as "integration testing". Integrity testing is used to make sure that these components, which may have been developed and tested independently, work properly together. Using PHPUnit, this technique is used for both manual and automated testing. A well-liked unit testing framework for the PHP programming language is PHPUnit. It is made to make it easier to create and run unit tests, a sort of testing that focuses on ensuring the accuracy of distinct software units or components in isolation. In this integration test, the entire system flow is tested, including the file upload process, the Databricks notebook execution, the performance of the analysis process, and the return of the data to the S3 bucket with several CSV files. The data will then be fetched and extracted from the CSV files using the Flask application, which was used to build ontologies in a Python environment. The Laravel web application is able to retrieve the data from the ontology graph created by making requests to the Flask application using the routes provided in the Flask application and display it in the web application.

In the table below, the Integration Test Cases and Results are displayed next.

Table 7.5: Integration Test Cases

Test Module	**Extraction Module + Transformation Module**		**Test Title**	**Integration Testing of Laravel Web-Application, S3 Bucket, Lambda Function and Azure Databricks**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
INT-001	Uploading Source Code File from Laravel Web-Application into S3 Bucket, which trigger the execution of the Databricks Notebook and generate Dataframe output in the Transformation Module.	1. Upload the Java Source Code Files through Laravel Web-Application 2. Saving the source code files into S3 Bucket 3. Trigger AWS Lambda function to call Databricks API 4. Execution of Databricks Notebook 5. Display Dataframe Result	Java Source Code Files	The Dataframes should include the Variables, Methods, Dependencies and Metrics data.	**Pass**
INT-002	Uploading Invalid Source Code File from Laravel Web-Application into S3 Bucket, which trigger the execution of the Databricks Notebook and generate CSV outputs and saving	1. Upload the Java Source Code Files through Laravel Web-Application 2. Error Message Return	PHP Source Code Files	Error Prompt in the Web-Application Page with invalid file extension.	**Pass**

	into S3 Bucket.				
Test Module	**Store Data Module + Ontology Transformation Module**		**Test Title**	**Integration Testing of S3 Bucket with CSV Files Data, Flask Application, Laravel Web Application**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
INT-003	After the Transformation process, the data in the dataframes are save into S3 bucket as CSV Files and retrieve by the Flask Application and perform ontology transformation process.	1. Writing the Dataframe data as CSV files. 2. Saving the CSV files into S3 bucket. 3. Retrieve the key in the S3 Bucket. 4. Retrieve the content of the files with the key retrieved. 5. Perform Ontology Transformation process.	No Test Data	The data in the CSV files are retrieve completely and accurately. Moreover, the Ontology graph has been constructed successfully with the data retrieved.	**Pass**
INT-004	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve all Variable Data	1. Clicking the display data button and select display variable. 2. Sending Request to Flask Application	No Test Data	All the Class and corresponding variable are shown in the page.	**Pass**

		3. Received Data and Render the Class Variable Page.			
INT-005	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Variable Data with user input class name.	1. Clicking the display data button and select display variable. 2. Input the class name. 3. Sending Request to Flask Application 4. Received Data and Render the Class Variable Page.	Class name: usercontroller	The class with user input class name and corresponding variables is shown in the page	**Pass**
INT-006	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve all Method Data	1. Clicking the display data button and select display method. 2. Sending Request to Flask Application 3. Received Data and Render the Class Variable Page.	No Test Data	All the Class and corresponding methods are shown in the page.	**Pass**
INT-006	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Method Data with user input class name	1. Clicking the display data button and select display method. 2. Input the class name. 3. Sending Request to Flask Application 4. Received Data and Render the	Class name: usercontroller	The class with user input class name and corresponding methods is shown in the page	**Pass**

		Class Variable Page.			
INT-007	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve all Dependencies Data	1. Clicking the display data button and select display dependency. 2. Sending Request to Flask Application 3. Received Data and Render the Class Dependencies Page.	No Test Data	All the Class and corresponding dependencies are shown in the page.	**Pass**
INT-008	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Dependencies Data with user input class name	1. Clicking the display data button and select display dependency. 2. Input the class name. 3. Sending Request to Flask Application 4. Received Data and Render the Class Variable Page.	Class name: usercontroller	The class with user input class name and corresponding dependencies is shown in the page	**Pass**
INT-009	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve all Metrics Data	1. Clicking the display data button and select display metric. 2. Sending Request to Flask Application 3. Received Data and Render the Class Dependencies Page.	No Test Data	All the Attributes and corresponding quantity are shown in the page.	**Pass**

Test Module	**Store Data Module + Ontology Transformation Module**		**Test Title**	**Integration Testing of Flask Application, Laravel Web Application and WebVOWL**	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expected Result**	**Status**
INT-010	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Complete Ontology File and upload to WebVOWL to visualize the Ontology Graph.	1. Clicking the transformation ontology & graph button and select generate graph. 2. Follow the procedure by downloading the ontology file. 3. Clicking the transformation ontology & graph button and select complete ontology. 4. Sending Request to Flask Application and return the CompleteOntology.ttl file as a downloadable attachment. **5.** Clicking on the WebVOWL Link and upload the file follow the procedure.	No Test Data	The Complete Ontology graph is generated	**Pass**

		6. The ontology graph is generated.			
INT-011	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Variable Ontology File and upload to WebVOWL to visualize the Ontology Graph.	1. Clicking the transformation ontology & graph button and select generate graph. 2. Follow the procedure by downloading the ontology file. 3. Clicking the transformation ontology & graph button and select variable ontology. 4. Sending Request to Flask Application and return the Variable.ttl file as a downloadable attachment. **5.** Clicking on the WebVOWL Link and upload the file follow the procedure. 6. The ontology graph is generated.	No Test Data	The Variable Ontology Graph is generated	**Pass**
INT-012	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Method Ontology File and upload	1. Clicking the transformation ontology & graph button and select generate graph. 2. Follow the procedure by	No Test Data	The Method Ontology Graph is generated	**Pass**

	to WebVOWL to visualize the Ontology Graph.	downloading the ontology file. 3. Clicking the transformation ontology & graph button and select method ontology. 4. Sending Request to Flask Application and return the Method.ttl file as a downloadable attachment. **5.** Clicking on the WebVOWL Link and upload the file follow the procedure. 6. The ontology graph is generated.			
INT-013	Testing the integration between the Flask Application and Laravel Wen-Application to retrieve Dependencies Ontology File and upload to WebVOWL to visualize the Ontology Graph.	1. Clicking the transformation ontology & graph button and select generate graph. 2. Follow the procedure by downloading the ontology file. 3. Clicking the transformation ontology & graph button and select depepndecy ontology. 4. Sending Request to Flask	No Test Data	The Dependencies Ontology Graph is generated	**Pass**

		Application and return the Dependency.ttl file as a downloadable attachment. **5.** Clicking on the WebVOWL Link and upload the file follow the procedure. 6. The ontology graph is generated.			

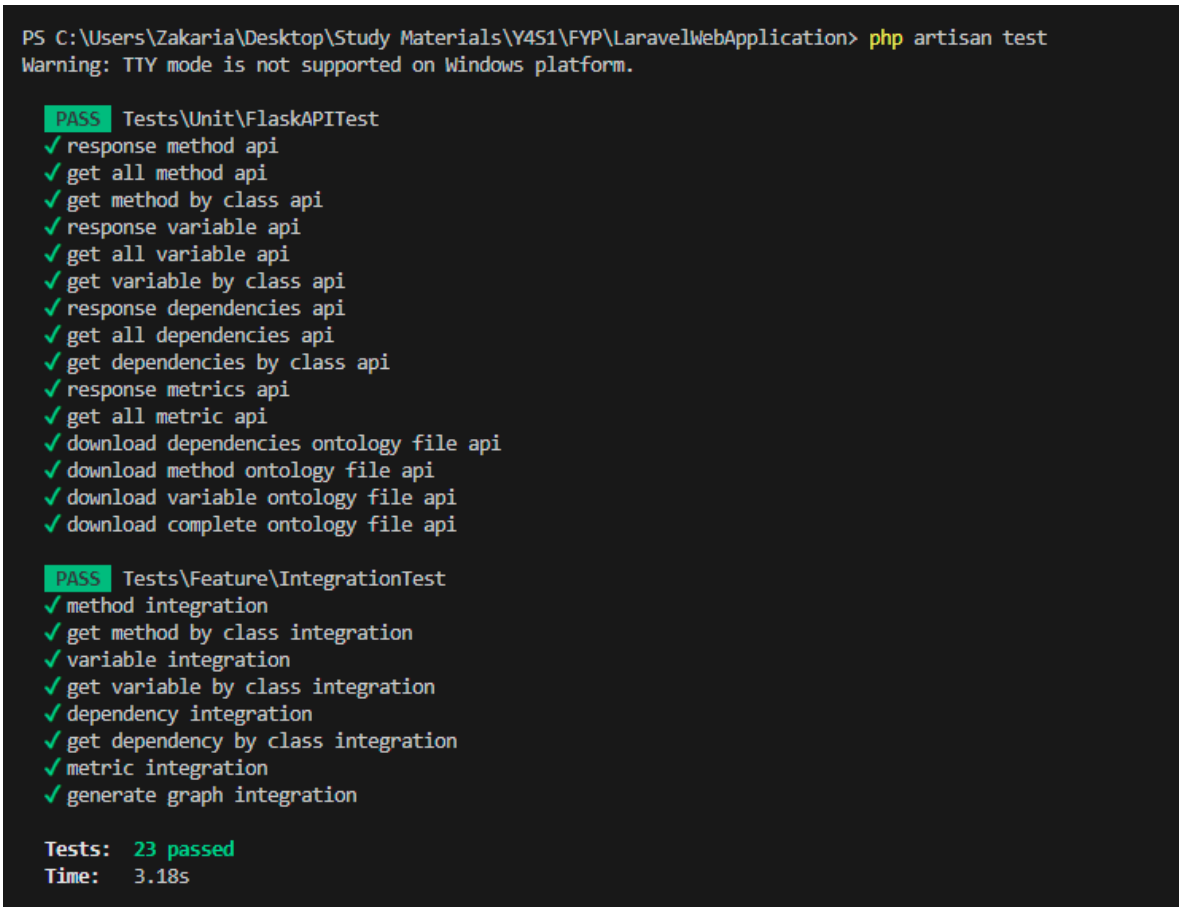


Figure 7.1: The Test Result

The results of testing the Flask API and the integration between the web application's route and the Flask API are shown in Figure 7.8, which uses the previously described Guzzle to handle the HTTP request between the Flask API and Laravel web application. With the aid of PHP Unit, some test cases are run, and as a result, all tests are passed. The Figures below demonstrate the construction of the test cases. Additionally, run the "php artisan test" command in the Laravel Web-Application directory and ensure that the Flask Application is running in order to execute the defined test cases.

```php
public function test_ResponseMethodApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllMethod';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
}

public function test_GetAllMethodApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllMethod';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("Has Method",$data['Relationship']);
    }
}

public function test_GetMethodByClassApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetMethodByClass/usercontroller';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("usercontroller",$data['Class']);
        $this->assertEquals("Has Method",$data['Relationship']);
    }
}
```

Figure 7.2: Test Cases for Method API

Guzzle allows us to send HTTP requests to Flask applications in order to retrieve JSON data. To verify that the URL is accurate, we first test the status of the response return after submitting the request to the Flask application. After that, retrieve data from the Flask application using the tested URL and compare each data value return with the anticipated outcome. The get data by class method is equivalent. The test cases that follow all relate to the same idea that is described and depicted in the drawings below.

```php
public function test_ResponseVariableApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllVariable';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
}

public function test_GetAllVariableApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllVariable';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("Has Variable",$data['Relationship']);
    }
}

public function test_GetVariableByClassApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetVariableByClass/usercontroller';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("usercontroller",$data['Class']);
        $this->assertEquals("Has Variable",$data['Relationship']);
    }
}
```

Figure 7.3: Test Cases for Variable API

```php
public function test_ResponseDependenciesApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllDependencies';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
}

public function test_GetAllDependenciesApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllDependencies';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("Has Dependent",$data['Relationship']);
    }
}

public function test_GetDependenciesByClassApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetDependenciesByClass/usercontroller';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("usercontroller",$data['Class']);
        $this->assertEquals("Has Dependent",$data['Relationship']);
    }
}
```

Figure 7.4: Test Cases for Dependencies API

```php
public function test_ResponseMetricsApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllMetric';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
}

public function test_GetAllMetricApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/GetAllMetric';

    $response =  $client->get($apiURL);
    $jsonResponse = $response->getBody()->getContents();
    $dataArray = json_decode($jsonResponse, true);
    foreach($dataArray as $data){
        $this->assertArrayHasKey('Relationship', $data);
        $this->assertEquals("Has Attribute",$data['Relationship']);
    }
}
```

Figure 7.5: Test Cases for Metrics API

```php
public function test_DownloadDependenciesOntologyFileApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/DependenciesOntology';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
    $this->assertEquals('attachment; filename=Dependency.ttl', $response->getHeaderLine('Content-Disposition'));
}

public function test_DownloadMethodOntologyFileApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/MethodOntology';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
    $this->assertEquals('attachment; filename=Method.ttl', $response->getHeaderLine('Content-Disposition'));
}

public function test_DownloadVariableOntologyFileApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/VariableOntology';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
    $this->assertEquals('attachment; filename=Variable.ttl', $response->getHeaderLine('Content-Disposition'));
}

public function test_DownloadCompleteOntologyFileApi(){
    $client = new Client();
    $apiURL = 'http://127.0.0.1:5000/CompleteOntology';

    $response =  $client->get($apiURL);
    $this->assertEquals(200, $response->getStatusCode());
    $this->assertEquals('attachment; filename=CompleteOntology.ttl', $response->getHeaderLine('Content-Disposition'));
}
```

Figure 7.6: Test Cases for Different Ontology File Download

Additionally, some integration testing scenarios are also carried out using PHP Unit by utilizing the Get method with a route that would call the controller's function and make a request to a Flask application. The web page will be rendered together with the data return. Applying the assertViewIs and assertSee functions to the content and the views file that were responsible for rendering the page will ensure that the rendered data is accurate. The code segments for the Integration Testing cases are shown in the images below. Figure 7.8 displays the outcomes of the integration testing instances declared below.

```php
public function test_MethodIntegration()
{
    $response = $this->get('/Method');
    $response->assertStatus(200);
    $response->assertViewIs('display-method');
    $response->assertSee('Class Methods');
}

public function test_GetMethodByClassIntegration()
{
    $data =[
        'className' => 'usercontroller'
    ];

    $response = $this->post('/searchMethodByClass',$data);
    $response->assertStatus(200);
    $response->assertViewIs('display-method');
    $response->assertSee('usercontroller');
}
```

Figure 7.7: Integration Test in Retrieving Method Data

```
public function test_VariableIntegration()
{
    $response = $this->get('/Variable');
    $response->assertStatus(200);
    $response->assertViewIs('display-variable');
    $response->assertSee('Class Variable');
}

public function test_GetVariableByClassIntegration()
{
    $data =[
        'className' => 'usercontroller'
    ];

    $response = $this->post('/searchVariableByClass',$data);
    $response->assertStatus(200);
    $response->assertViewIs('display-variable');
    $response->assertSee('usercontroller');
}
```

Figure 7.8: Integration Test in Retrieving Variable Data

```
public function test_DependencyIntegration()
{
    $response = $this->get('/Dependency');
    $response->assertStatus(200);
    $response->assertViewIs('display-dependency');
    $response->assertSee('Class Dependencies');
}

public function test_GetDependencyByClassIntegration()
{
    $data =[
        'className' => 'usercontroller'
    ];

    $response = $this->post('/searchDependencyByClass',$data);
    $response->assertStatus(200);
    $response->assertViewIs('display-dependency');
    $response->assertSee('usercontroller');
}
```

Figure 7.9: Integration Test in Retrieving Dependency Data

```php
public function test_GenerateGraphIntegration()
{
    $response = $this->get('/generateGraph');
    $response->assertStatus(200);
    $response->assertViewIs('generate-graph');
    $response->assertSee('Process To Generate A Ontology Graph');
}
```

Figure 7.10: Integration Test in Retrieving Ontology File

7.4 Performance Testing

Performance testing is a subset of software testing that focuses on assessing a software application's speed, responsiveness, scalability, stability, and overall performance under various circumstances. Performance testing's main objective is to confirm that the application meets performance standards and provides a positive user experience.

In our application, one of the main elements that affects how quickly an analysis is completed in Databricks is the responsiveness of the Web application. Performing performance testing for the total processing time of execution in Databricks notebook involves assessing how efficiently the notebook executes its code and generates results. The analytical process should be completed in less than three minutes. And the Databricks Notebook's execution result is displayed below. The analytic process and data storage process, which satisfy the requirements needed for the web application, take 1 minutes and 5 seconds to finish. The Flask Application is able to retrieve the data as quickly as possible and connect back to the web application when the analysis speed increases.

Figure 7.11: Total Used Time for Databricks Analysis & Produce Output

Table 7.6: Execution Databricks Notebook Under Normal Load Test Case

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-001	Measure the total execution time of a Databricks Notebook under normal load Condition	1. Upload File Through Web Application 2. Trigger Execution of the Workflow declare in Databricks. 3. Record the total execution time of the workflow which took responsibility for the execution of Databricks notebook.	The total execution time should be lower than 3 minutes to complete the analysis process and generate output	**Pass**

Additionally, the flask will be in charge of retrieving the data from the S3 Bucket and transforming it into an ontology after the analysis phase and data storage procedure are complete. The Flask application needs 0.5 seconds to process thousands of rows of data from CSV files calculated by the Time module imported. It could construct the ontology graph and provide the data by submitting a request to the Flask Application because it satisfied the non-functional requirements.

```
The ontology graph construct use time: 0.5100123882293701
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Figure 7.12: Total Used Time for Construct Ontology Graph

Table 7.7: Ontology Construction Under Normal Load Test Case

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-002	Measure the total construction time for the ontology graph construction which include data retrieving phase and generate ontology phase.	1. Retrieve Data from the S3 Bucket after the Databricks Analysis is done. 2. Extracting data from the files retrieved from S3 bucket. 3. Construct the ontology graph using the data extracted. 4. Record the total execution time of the function which took responsibility for the ontology construction.	The total execution time should be lower than 1 minutes to complete the data retrieving process and ontology generating phase.	**Pass**

Table 7.8: Used Time for Sending Different Flask API Request

API Request	Average Used Time (ms)
getAllMethod	45
getAllDependencies	45
getAllVariable	38
getVariableByClass	15
getMethodByClass	15
getDependentByClass	18
getAllMetric	34

The average used time for sending requests to the Flask application to retrieve data is displayed in the table above. All API queries are completed in less than 50 milliseconds, ensuring a smooth data return process and better user experiences.

Table 7.9: Sending Request to Flask API Under Normal Load Test Case

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-003	Measure the average response time to for the Flask Application to handle each of the request	1. Sending Request to Flask Application 2. Measure the response time for the corresponding request. 3. Iterate 5 time the step above and calculate the average response time. 4. Iterate the step above with all the requests.	Each of the request should return the result within 100 millisecond which provide near-instantaneous response.	**Pass**

7.5 System Usability Test

The System Usability Scale (SUS) is being utilized in this project to assess the application's usability. The "quick and dirty" methodology of the SUS was chosen because it produces accurate findings for usability testing. This method is especially helpful for this job because it only requires one person to finish the system in a short amount of time. The SUS questionnaire consists of 10 questions, each with five possible answers ranging strongly agree to neutral to strongly disagree. A score of 1 to 5, with 5 being a strong agreement and 1 a strong disagreement, is assigned to each response option. The System Usability Scale (SUS), which has been cited in more than 1,300 articles and publications, has developed into an industry standard. The SUS has the advantage of being a powerful testing instrument because it can generate trustworthy results with tiny sample quantities. Additionally, it is a reliable indicator of whether a system is useable or not (Klug, 2017).

The User Satisfaction Survey Template (Brooke, 1996), which consists of two sections, is shown in the table below and was used to conduct the usability testing. As was already indicated, Section A comprises of the ten rating questions, while Section B has three open-ended questions that let respondents offer brief observations on the current system The template and open-ended questions is shown in the Appendix A.

7.5.1 Test Scenario of Usability Testing

Test Scenario to act as a software developer / maintainer
Scenario 1 – Upload the Java Source code file to perform analysis purpose
Imagine you are a software developer / maintainer that are new to the team and would request to use this system to get basic understanding about the source code system by uploading them to a web application with a URL provided. What would you do to access this system and upload the source code?
Scenario 2 – Retrieve the basic variable data for each of the class
Imagine you are a software developer / maintainer. You need to get the basic understanding of the source code by knowing the variable for each of the classes. Your task is to navigate to specific pages that return the variable data for different classes. How would you access this information in the web application?
Scenario 3 – Retrieve the variable data for specific class by searching with class name
Imagine you are a software developer / maintainer. You need to get specific variable data for

specific class which helps you to solve the bug of the system. Your task is to navigate to specific pages, searching the data by input the class name and it return the variable data for different classes. How would you access this information in the web application?
Scenario 4 – Retrieve the basic method data for each of the class
Imagine you are a software developer / maintainer. You need to get the basic understanding of the source code by knowing the method for each of the classes. Your task is to navigate to specific pages that return the method data for different classes. How would you access this information in the web application?
Scenario 5 – Retrieve the method data for specific class by searching with class name
Imagine you are a software developer / maintainer. You need to get specific method data for specific class which helps you to solve the bug of the system. Your task is to navigate to specific pages, searching the data by input the class name and it return the method data for different classes. How would you access this information in the web application?
Scenario 6 – Retrieve the basic dependency data for each of the class
Imagine you are a software developer / maintainer. You need to get the basic understanding of the source code by knowing the dependency for each of the classes. Your task is to navigate to specific pages that return the dependency data for different classes. How would you access this information in the web application?
Scenario 7 – Retrieve the dependency data for specific class by searching with class name
Imagine you are a software developer / maintainer. You need to get specific dependency data for specific class which helps you to solve the bug of the system. Your task is to navigate to specific pages, searching the data by input the class name and it return the dependency data for different classes. How would you access this information in the web application?
Scenario 8 – Retrieve the metric data for whole source code system
Imagine you are a software developer / maintainer. You need to get the basic understanding of the source code by knowing the metrics of the whole source code system. Your task is to navigate to specific pages that return the metrics data. How would you access this information in the web application?
Scenario 9 – Download the complete ontology file to generate ontology graph which support ontology meaning
Imagine you are a software developer / maintainer. You need to get the understanding of different component include in the classes, obviously the web application only support showing one component of the class on a page. Hence the web application provides a

complete ontology file download options which support graph generation function. How would you access this download method and proceed to generate the graph?
Scenario 11 – Download the variable ontology file to generate ontology graph which support ontology meaning
Imagine you are a software developer / maintainer. You need to get the understanding of certain component include in the classes that sharing the same variable name, obviously the web application only support showing one component of the class on a page. Hence the web application provides a variable ontology file download options which support graph generation function to show the full visualization of the class and variable nodes and its relationship. How would you access this download method and proceed to generate the graph?
Scenario 12 – Download the method ontology file to generate ontology graph which support ontology meaning
Imagine you are a software developer / maintainer. You need to get the understanding of certain component include in the classes that sharing the same method name, obviously the web application only support showing one component of the class on a page. Hence the web application provides a method ontology file download options which support graph generation function to show the full visualization of the class and method nodes and its relationship. How would you access this download method and proceed to generate the graph?
Scenario 13 – Download the dependencies ontology file to generate ontology graph which support ontology meaning
Imagine you are a software developer / maintainer. You need to get the understanding of certain component include in the classes that dependent on the same dependencies function, obviously the web application only support showing one component of the class on a page. Hence the web application provides a dependencies ontology file download options which support graph generation function to show the full visualization of the class and dependency nodes and its relationship. How would you access this download method and proceed to generate the graph?

7.5.2 Result of Usability Testing

During the course of the usability testing procedure, as described in section 7.5.1, three respondents were chosen to offer input on 13 test scenarios. In Appendix B, you'll find a list of each tester's recorded responses.

By allocating a matching numerical score to each response, the respondent's responses are analysed to determine the SUS score. The following framework can then be used to tabulate the overall SUS score:

I. For every question with an odd number, one is deducted from the score to determine the final result.
II. For all questions with an even number, five is deducted from the score to determine the final result.
III. To get the percentage score, the total score from all the questions that a participant responded is summed up and multiplied by 2.5.
IV. Each participant's percentage scores are added and divided by the total number of participants. The entire percentage is divided by 3 in this instance.

The aforementioned procedure can be used to calculate each participant's SUS score. The SUS score is a total number out of 100, not a percentage, which is crucial to comprehend. A SUS score of 68 will only place you in the 50th percentile because the average SUS score for projects is 68. However, an SUS score above or below the average may provide a fast indicator of how usable the design solution is overall.

SUS Score	Grade	Adjective Rating
> 80.3	A	Excellent
68 – 80.3	B	Good
68	C	Okay
51 – 68	D	Poor
< 51	F	Awful

The system received an average usability score of 80.83% during testing, equivalent to a Grade A rating. The results are displayed in the table below. This demonstrates how extremely useful and user-friendly web applications are.

Participants Name	Usability Score for Each Question										Total	Percentage (%)
	1	2	3	4	5	6	7	8	9	10		
Chang Hao Jie	3	4	4	3	2	4	3	3	3	3	32	80
Ong Zhi Ying	4	3	3	3	2	4	4	4	3	2	32	80
Lim Jun How	2	4	4	4	2	4	3	4	3	3	33	82.5
Average SUS Score												**80.83**
Grade												**A**

A few open-ended questions were also produced in addition to the System Usability Scale (SUS) used in the System Usability Testing to allow respondents to offer succinct remarks on the present system. This strategy enhanced the quantitative data collected through the SUS by obtaining insightful feedback on how users felt and perceived the implemented system. Here is a list of the open-ended questions that were used:

1. What do you like best about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

Based on participant feedback, the following table lists the system's most popular features and functionalities. However, the investigation did not uncover any least-liked features or functionalities.

Table 7.10: Summary of Participants' Top Liked Features of the System

Summary of Participants' Top Liked Features of the System
The graph generate function using the Ontology File downloaded and WebVOWL is one of the system features that I most liked.
The time consumed to complete the analysis and return response is better than my expectation which comparing to the other documentation tools in used.
The splitting of the ontology graph provides better graph visualization of the source code. Since the Complete Ontology graph generated quite messy leads to difficulties in searching the nodes in the graph and corresponding relationship.

Participants offered suggestions for enhancing the current system, as indicated in the table below, despite the fact that no issues or least-liked features and capabilities were found during the testing. These suggestions are helpful for improving the system's overall efficacy and usefulness.

Table 7.11: Summary of suggestions for improving the system by participants

Summary of suggestions for improving the system by participants
The WebVOWL can be integrated directly in the Web Application since the WebVOWL is an open-source tool.
There is too limited source code analysis can be done, hope the further work of this project can handle more type of source code in this web application.
The lack of authentication and authorization which handle different purposes, for example the project manager is able to upload the source code file and the project member is able to view the documentation generate by creating some account and assign to different team member. Hope the further work can include the security features mentioned above.

7.6 Manual Evaluate the Proposed OBSR with distributed processing techniques

```
public class StakeHolder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id = 0;
    String name;

    @OneToOne(cascade = CascadeType.ALL)
    StakeHolderAddress address;

    @OneToMany(mappedBy = "stakeholder", cascade = CascadeType.ALL)
    List<StakeHolderTelephone> telephones;
```

Figure 7.13: Variable in Stakeholder Class

Class	HasVariable
stakeholder	id
stakeholder	address
stakeholder	telephones
stakeholder	name

Figure 7.14: Search Result with stakeholder's class variable

From the 2 figure above, the proposed OBSR with distributed processing techniques has successfully extract the variable result from the source code.

The test cases below showing the result and its test description:

Table 7.12: Test Cases for evaluate the propose OBSR method

Test Case ID	Test Description	Steps	Expected Result	Status
EVL-001	Testing the proposed OBSR with distributed technique approach using manual validating the variable of different classes in the source code.	1. Input the class name in the search text field and get the result. 2. Manual comparing the result with the actual source code for validating the correctness of the OBSR approach.	Each of the variable with corresponding class inside the source code are listed in the OBSR approach which show in the web page.	**Pass**
EVL-002	Testing the proposed OBSR with distributed technique approach using	1. Input the class name in the search text field and get the result. 2. Manual	Each of the method with corresponding class inside the source code are	**Pass**

	manual validating the method of different classes in the source code.	comparing the result with the actual source code for validating the correctness of the OBSR approach.	listed in the OBSR approach which show in the web page.	
EVL-003	Testing the proposed OBSR with distributed technique approach using manual validating the dependency of different classes in the source code.	1. Input the class name in the search text field and get the result. 2. Manual comparing the result with the actual source code for validating the correctness of the OBSR approach.	Each of the dependency with corresponding class inside the source code are listed in the OBSR approach which show in the web page.	**Pass**

The sample output of the method and dependencies are shown in the figures below:

```
    // create  stock log with its respective details
    @RequestMapping(value = "/openStockAll", method = RequestMethod.POST)
    public ResponseEntity<?> saveOpenStock(@Valid @RequestBody OpenStock openStock)  {

        if(openStock.equals(null) ) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(oncall(false,"POST"));
        }else {
            openStock.setDate(ZonedDateTime.now(ZoneId.of("UTC-4")));
            stockService.saveOpenStock(openStock);
            return ResponseEntity.status(HttpStatus.ACCEPTED).body(oncall(true,"POST"));
        }

    }
//  public OpenStock saveOpenStock(@RequestBody OpenStock openStock) {
//      openStock.setDate(ZonedDateTime.now(ZoneId.of("UTC-4")));
//      return stockService.saveOpenStock(openStock);
//    }

    // fetch all stock logs with its respective stock details
    @RequestMapping(value = "/openStockAll", method = RequestMethod.GET)
    public ResponseEntity<?> fetchAllOpenStock() {
        List<OpenStock> openStocks = stockService.fetchAllOpenStock();
        if(openStocks == null || openStocks.size() == 0) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(oncall(false,"GET"));
        }else {
            return ResponseEntity.ok(stockService.fetchAllOpenStock());
        }
    }

    // create a new stock log only
    @RequestMapping(value = "/openStockLog", method = RequestMethod.POST)
    public ResponseEntity<?> createOpenStock(@RequestBody OpenStock openStock) {
//      openStock.setDate(ZonedDateTime.now(ZoneId.of("UTC-4")));
//        return stockService.createOpenStock(openStock);

        if(openStock.equals(null) ) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(oncall(false,"POST"));
        }else {
            System.out.println("not null");
            if(openStock.getUser() == null || openStock.getUser()<1 ) {
                throw new RuntimeException("Please provide valid open stock log information");
            }else if (openStock.getReason() == null) {
                throw new RuntimeException("Please provide valid open stock log information");
            } else{
                openStock.setDate(ZonedDateTime.now(ZoneId.of("UTC-4")));
                stockService.createOpenStock(openStock);
                return ResponseEntity.status(HttpStatus.ACCEPTED).body(oncall(true,"POST"));
            }
```

Figure 7.15: part of Openstockcontroller source code

Class	HasDependencies
openstockcontroller	accepted
openstockcontroller	getuser
openstockcontroller	now
openstockcontroller	bad_request
openstockcontroller	body
openstockcontroller	of
openstockcontroller	equals
openstockcontroller	createopenstock
openstockcontroller	setdate
openstockcontroller	getreason
openstockcontroller	status
openstockcontroller	fetchallopenstock
openstockcontroller	deleteopenstocklog
openstockcontroller	size
openstockcontroller	deleteallopenstockdetails
openstockcontroller	deleteopenstockdetails
openstockcontroller	ok

Figure 7.16: Part of the dependencies output of openstockcontroller

```java
public class StakeHolder {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id = 0;
    String name;

    @OneToOne(cascade = CascadeType.ALL)
    StakeHolderAddress address;

    @OneToMany(mappedBy = "stakeholder", cascade = CascadeType.ALL)
    List<StakeHolderTelephone> telephones;

    public List<StakeHolderTelephone> getTelephones() {
        return telephones;
    }

    public void setTelephones(List<StakeHolderTelephone> telephones) {
        this.telephones = telephones;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public StakeHolderAddress getAddress() {
        return address;
    }

    public void setAddress(StakeHolderAddress address) {
        this.address = address;
    }

}
```

Figure 7.17: Source code of stakeholder class

Class	HasMethod
stakeholder	setaddress
stakeholder	setid
stakeholder	getname
stakeholder	getaddress
stakeholder	gettelephones
stakeholder	getid
stakeholder	setname
stakeholder	settelephones

Figure 7.18: Result of the stakeholder class's method

CHAPTER 8
CONCLUSION & RECOMMENDATION

8.1 Conclusion

This chapter's objective is to bring this work to a close which to include the achievement of the project and the limitation of current developed system and provide recommendation for further enhancement. All of the goals listed in Chapter 1 were accomplished, including:

1. To develop a web application to handle the redocumentation process by the source code uploaded by the user and generate documentation and dependency diagram of the source code.
2. To create a data transformation method in the cloud platform which uses distributed processing technique and generate output return to the web application.
3. To evaluate the proposed OBSR with distributed processing technique approach using validating the correctness of the information and diagram generated.

To accomplish the first objective by working together to build a web application, the Laravel framework and Flask Application were used. In order to create dependency diagrams that give software maintainers a greater understanding of the source code's structures and components, the built-in web application may fetch the source code's documentation and ontology file.

In addition, the second objectives is accomplish by using Azure Databricks cloud platform, which offers cloud analysis functionalities with distributed processing approaches to boost the analysis efficiency. The source code uploaded by the Software Project Manager is saved in the AWS S3 bucket and mounted in Azure Databricks HDFS which enable the analysis process to retrieve the data from the source code directly. The analysis's output will be saved in an AWS S3 bucket after it has been completed. The Flask Application can execute ontology transformation and extract the content of the result using the result. The ontology data can be extracted with the aid of SPARQL and returned to the web application by sending a request to the Flask Application.

Additionally, the third goal is accomplished through manual testing, which verifies the accuracy of the analysis by contrasting it with the real source code structure. To ensure consistency between the analytic result and the graph result, the created graph is then manually checked.

Not to mention, the project's goal of creating a semantically based redocumentation technique using distributed processing technology and an ontology to produce documentation for legacy systems in order to enhance the efficiency of the development and debugging phases within a project team has been accomplished.

8.2 Limitation and Recommendation for future work

Multiple constraints were found throughout the system's development and testing phases, both by me and the usability test participants. The following section will list these drawbacks in brief and offer suggestions for future research. This will make it possible to take care of any existing restrictions and guarantee that the system's performance may be improved in subsequent versions.

The first limitation of the system would be the low customization of the WebVOWL tool. It provides a complete ontology graph and does not handle too much of nodes on the ontology graph. If the ontology constructed is too complex with too many nodes and relationship, the graph will not be able perform smoothly and decrease the user experience. The WebVowl tool is not integrated in the Web Application due to its an open-source tool. If the customization on the WebVOWL can be done through, the visualization of the ontology dependency graph can be separate into several pages and provide navigation function to the element in the ontology graph with an anchor element.

The second limitation of the system would be the limited source code analysis can be done. The system only can handle Java source code analysis with corresponding ETL method in the cloud platform. Hence the future work of this project would focus on the enhancing of the web application which develop different ETL method in cloud platform to handle different source code uploaded by the Software Project Manager in order to provide more information and portability of the web application for different type of source code.

The third limitation of the system would be lack of authentication and authorization which handle different purposes. Hence, the future work can be done through adding a authorization module with authentication function. For example, adding a login, register account function and assign roles for the account created with the help of Laravel Framework, such as a manager has an account with project leader role which have the permission of uploading the source code to perform analysis, and the team member is only allowed to view the documentation generated.

The fourth limitation of the system it only handles one user, which representing if the other user uses this application, the previous data will be replace. To address this in the future work, with the authentication and authorization function, creating different folder with the account id for different user to store their data and retrieve the data from the corresponding file with their account id.

The fifth limitation of the system is the performance of the analysis will be affected by the developers coding style. Because the analysis is done through reading the source code line by line and extract the necessary information with some string handle methods. The possible solution is to encourage the developers to follow the coding guideline define in the team or using formatter before performing the analysis process.

With the limitations and suggestions gathered from the responses, we are confident that we can address each of the shortcomings with the suggestion and solution offered to improve the current approach and provide a better approach to documentation analysis that benefits the software developer or even the entire software industry.

REFERENCES

Brooke, J., 1996. Sus: a “quick and dirty’usability. *Usability evaluation in industry*, [e-journal] 189(3), 189–194. Available at: < http://www.tbistafftraining.info/smartphones/documents/b5_during_the_trial_usability_scale_v1_09aug11.pdf > [Accessed 30 August 2023].

CANFORA, G. & CIMITILE, A., 2001. SOFTWARE MAINTENANCE. *In Handbook of Software Engineering and Knowledge Engineering*, [e-journal] vol. 1, pp. 91–120. Available at: <https://www.worldscientific.com/doi/abs/10.1142/9789812389718_0005> [Accessed 1 March 2023].

Castillo, R., Rothe, C. & Leser, U., 2010, RDFMatView: Idexing RDF Data for SPARQL Queries. [e-journal] Available at: <https://www.informatik.hu-berlin.de/de/forschung/gebiete/ki/wbi/research/publications/2010/rdfmatview.pdf> [Accessed 7 April 2023].

Databricks. (n,.d.). *What is Hadoop Distributed File System(HDFS)?* [online] Available at: <https://www.databricks.com/glossary/hadoop-distributed-file-system-hdfs> [Accessed 9 April 2023].

Freeman, R.M. & Munro, M., 1992. Redocumentation for the maintenance of software. *Proceedings of the 30th annual Southeast regional conference on - ACM-SE 30*, 413, ACM Press, New York, New York, USA.

Ganapathy, G. & Sagayaraj, S., 2010. Automatic Ontology Creation by Extracting Metadata from the Source code. *Global Journal of Computer Science and Technology*, [e-journal] 10(14). Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a8a13eccb374c7a5fb361bcec97ced0d69b38ee2> [Accessed 8 April 2023].

Gannod, G.C. & Cheng, B.H.C., 1999. A Framework for Classifying and Comparing Software Reverse Engineering and Design Recovery Techniques. *Sixth Working Conference*, [e-journal] 77–88. Available at:

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=321c2f66ffefcfdbb2d0d59a631d26870652bfb4> [Accessed 2 March 2023].

Garg, A., 2023. *Apache Spark Architecture*. [online] Intellipaat Blog. Available at: <https://intellipaat.com/blog/tutorial/spark-tutorial/spark-architecture/> [Accessed 5 April 2023].

Hartmann, J., Palma, R. & Gómez-Pérez, A., 2009. Ontology Repositories. *Handbook on Ontologies*, [e-journal] pp. 551–571. Available at: <https://oa.upm.es/6430/2/OntologyRepositories.pdf> [Accessed 8 April 2023].

Heesch, D. van, 2004. Doxygen. [online] Available at: <http://www.doxygen.org/> [Accessed 5 April 2023].

Kaur, U. & Singh, G., 2015. A Review on Software Maintenance Issues and How to Reduce Maintenance Efforts. *International Journal of Computer Applications*, [e-journal] 118(1), 6–11. Available at: <https://d1wqtxts1xzle7.cloudfront.net/49247409/A_Review_on_Software_Maintenance_Issues_and_How_to_Reduce_Maintenance_Efforts-libre.pdf?1475263537=&response-content-disposition=inline%3B+filename%3DA_Review_on_Software_Maintenance_Issues.pdf&Expires=1693686419&Signature=gvG4gY2S9-v6Jm94JGm1Vep2zeV4MSEHjYbgGcOVn-Hd0-jg8mRXhy0JAfWxuGBPelxHpbXhDum0n-qbTCnJetkFo6U4Vr~r-nD0Y5dFKeJ-T9-E-hSkpje7mOWPw110ZGb8lslwF~c5MT7aIL40w4TOcwN2LwSgXpJPak6sOuInkgvdJ1VL0f-qbID6rwEkdc1gJ1pk6feoAv7fSTuyee51tW58VaLpK67CJJlJWCpEE5gO6eQ1raGY73CQ-y0KfREO2DRgjKFdYu5lmtoUn41~8kxQjevfEXuP3~oTA2VEGdZ3PH7eoFOu6YV-yllxYXPqUDYkpp9f2jD7iM07HYw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA> [Accessed 3 March 2023].

Klug, B., 2017. An Overview of the System Usability Scale in Library Website and System Usability Testing. *Weave: Journal of Library User Experience*, [e-journal] 1(6). Available at: <http://quod.lib.umich.edu/cgi/t/text/idx/w/weave/12535642.0001.602/--overview-of-the-system-usability-scale-in-library-website?rgn=main;view=fulltext> [Accessed 30 August 2023].

Leslie, D.M., 2002. Using Javadoc and XML to produce API reference documentation. *Proceedings of the 20th annual international conference on Computer documentation*, [e-journal] 104–109, ACM, New York, NY, USA. Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=164ad1f9b38f496e1773a5929e87c6782edde3c0> [Accessed 5 April 2023].

Martinez, P., 2021. *What is Evolutionary Prototype?*, [online] *Mockitt*. Available at: < https://mockitt.wondershare.com/prototyping/evolutionary-prototyping.html> [Accessed 3 April 2023]

Müller, H.A., Wong, K. & Tilley, S.R., 1993. Understanding Software Systems Using Reverse Engineering Technology Perspectives from the Rigi Project. *CASCON'93*, [e-journal] 217–226. Available at: <https://d1wqtxts1xzle7.cloudfront.net/49286960/Understanding_software_systems_using_rev20161002-6470-tlftpc-libre.pdf?1475404458=&response-content-disposition=inline%3B+filename%3DUnderstanding_Software_Systems_Using_Rev.pdf&Expires=1693686708&Signature=CZq6u3tdp3mt7DxAaJ2BP-opfY-pR69efM4qw-xwJWwAtv5~61sn3q3FjhOBKQYdl8LO7MQJ98iwld0X5HNIAjGOuw3mUuJbkvL~tcJcGY11oi~qX13ABEvlSzS8m9VXDLs7mofVm5laUlxMri20HAd-gesLoq-XZueMQMJZ5pbr9~RL5BRA3BZhEYb3PxKa6R35nQL0A4HP5D8~~nbm0zvTklyflmfbP5~2~izMFMjqSaMp1UaxX0-Fbk~Bz5qgYdc95kDOcZs7TL0uY1w7QbrU5HdGJR2d1Cx8ZVhonhIsUCJiAfD4NTACLWkSzNC-7ueZw~pFtM4gwnlgHAEnPw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA> [Accessed 5 April 2023].

Nallusamy, S., 2015. *An Ontology Based Software Reodumentation Approach TO Support Program Understanding For Event-Driven Programming* – PhD thesis, Universiti Teknologi Malaysia .

Nallusamy, S., Hao, H.M. & Zulkifle, F.A., 2021. Software Redocumentation Using Distributed Data Processing Technique to Support Program Understanding for Legacy System: A Proposed Approach. *In Advances in Visual Informatics: 7th International Visual Informatics Conference*, vol. 13051, pp. 239–252.

Pérez, J., Arenas, M. & Gutierrez, C., 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, [e-journal] 34(3), 1–45. Available at: <https://dl.acm.org/doi/abs/10.1145/1567274.1567278> [Accessed 31 August 2023].

Pointer, I., 2020. *What is Apache Spark? The big data platform that crushed Hadoop*. *InfoWorld*. [online] InforWorld. Available at: <https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html> [Accessed 5 April 2023].

Ryan, A., 2022. *Why documentation is important in software development*. [online] Available at: <https://www.linkedin.com/pulse/why-documentation-important-software-development-alexander-ryan> [Accessed 1 March 2023].

Shearer, R., Motik, B. & Horrocks, I., 2008. HermiT: A Highly-Efficient OWL Reasoner. [online] Available at: <http://www.cs.ox.ac.uk/boris.motik/pubs/smh08HermiT.pdf> [Accessed 7 April 2023].

Sirin, E. & Parsia, B., 2007. SPARQL-DL: SPARQL Query for OWL-DL. [e-journal] 258. Available at: <https://d1wqtxts1xzle7.cloudfront.net/6221666/10.1.1.142.9826-libre.pdf?1390845057=&response-content-disposition=inline%3B+filename%3DSparql_dl_Sparql_query_for_owl_dl.pdf&Expires=1693687149&Signature=gnwA~QWmmyuQ~z86SP79AFxgnFLBPAv7nPG1lRHYv15G5vRK-Y9QPSNySjYITyXs7Oq3Pxfhn7yY6bSr4-8W6UrHPiqhTo6-qDMcPquwfXgmaTflxMKscgeCAp0MkyercKTHOibdCoS9gJlikgGAiM93bpo8FPJHbypaBVB-NkXRY1gyHlNlh-~ZWbyNArvhrgQ9CEg-RIXIlcKYdH4P17ajrb7L-FMSWmS031CZQRNWaG4a4BwgmY5ldG6lGe4Hs8Q2Qy8pqFcDgQ1A3OxiKkRm5NcGUZcdhJEsVIpy8wKyz1oPDtDNPcCjNNx-7gmSUm8bXBzcKjLPpVhutzeD4w__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA> [Accessed 30 August 2023].

Sivakumar, R. & Arivoli, P. V, 2011. Ontology Visualization Protégé Tools – a Review. *International Journal of Advanced Information Technology (IJAIT)*, [e-journal] 1(4). Available at: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3429010> [Accessed 6 April 2023].

phpDocumentor, (n.d.). *Application Flow*. [online] Available at: <https://docs.phpdoc.org/guide/internals/flow.html> [Accessed 11 April 2023].

Natural Docs, (n.d.). *The Natural Docs Reference*. [online] Available at: <https://www.naturaldocs.org/reference/> [Accessed 12 April 2023].

APPENDICES

Appendix A: Template of User Satisfaction Survey

Participant No.: Name:					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.					
2. I found the system unnecessarily complex.					
3. I thought the system was easy to use.					
4. I think I would need the support of a technical person to be able to use this system.					
5. I found the various function in this system were well integrated.					
6. I thought there was too much inconsistency in this system.					
7. I would imagine that most people would learn to use this system very quickly					
8. I found the tool system cumbersome to use.					
9. I felt very confident using the system.					
10. I needed to learn a lot of things before I could get going with this system.					

1. What do you like best about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

Appendix B: Usability Test Responses

Participant No.: 1 Name: Chang Hao Jie					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.				✓	
2. I found the system unnecessarily complex.	✓				
3. I thought the system was easy to use.					✓
4. I think I would need the support of a technical person to be able to use this system.		✓			
5. I found the various function in this system were well integrated.			✓		
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly				✓	
8. I found the tool system cumbersome to use.		✓			

9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

   The graph generate function using the Ontology File downloaded and WebVOWL is one of the system features that I most liked.

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The WebVOWL can be integrated directly in the Web Application since the WebVOWL is an open-source tool.

Participant No.: 2 Name: Ong Zhi Ying					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.					✓
2. I found the system unnecessarily complex.		✓			
3. I thought the system was easy to use.				✓	
4. I think I would need the support of a technical person to be able to use this system.		✓			
5. I found the various function in this system were well integrated.			✓		

6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly					✓
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.			✓		

1. What do you like best about the system?

   The time consumed to complete the analysis and return response is better than my expectation which comparing to the other documentation tools in used.

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   There is too limited source code analysis can be done, hope the further work of this project can handle more type of source code in this web application.

Participant No.: 3 Name: Lim Jun How					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.			✓		
2. I found the system unnecessarily complex.	✓				

3. I thought the system was easy to use.					✓
4. I think I would need the support of a technical person to be able to use this system.	✓				
5. I found the various function in this system were well integrated.			✓		
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly				✓	
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.					✓
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

   The splitting of the ontology graph provides better graph visualization of the source code. Since the Complete Ontology graph generated quite messy leads to difficulties in searching the nodes in the graph and corresponding relationship.

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The lack of authentication and authorization which handle different purposes, for example the project manager is able to upload the source code file and the project member is able to view the documentation generate by creating some account and

assign to different team member. Hope the further work can include the security features mentioned above.

Appendix C: Keys Retrieving Process in S3 Bucket by Boto3

```
import pandas as pd
import boto3
import csv
from io import StringIO
from rdflib import Graph, URIRef, Literal, Namespace, RDF, BRICK, RDFS, OWL
from flask import Flask, render_template, request, send_file, jsonify

OntologyGenerator = Flask(__name__)

s3 = boto3.client('s3')

bucket_name = 'semanticredocumentation'
dependencies_file = []
methods_file = []
metric_file = []
variable_file = []
try:
    # List objects in the bucket
    response = s3.list_objects_v2(Bucket=bucket_name)
    # Filter for CSV files and retrieve the content of the first CSV file found
    for obj in response.get('Contents', []):
        if obj['Key'].lower().endswith('.csv'):
            if(obj['Key'].lower().startswith('dependencies')):
                dependencies_file.append(obj['Key'])
            elif(obj['Key'].lower().startswith('method')):
                methods_file.append(obj['Key'])
            elif(obj['Key'].lower().startswith('metrics')):
                metric_file.append(obj['Key'])
            elif(obj['Key'].lower().startswith('variable')):
                variable_file.append(obj['Key'])
except Exception as e:
    print(f"Error: {e}")
```

Appendix D: Information Extraction and Concatenate Dataframe

```
li = []
li2 = []
li3 = []
li4 = []

for Key in dependencies_file:
    csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
    csv_content = csv_object['Body'].read().decode('utf-8')
    df = pd.read_csv(StringIO(csv_content))
    li.append(df)

for Key in methods_file:
    csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
    csv_content = csv_object['Body'].read().decode('utf-8')
    df = pd.read_csv(StringIO(csv_content))
    li2.append(df)

for Key in variable_file:
    csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
    csv_content = csv_object['Body'].read().decode('utf-8')
    df = pd.read_csv(StringIO(csv_content))
    li3.append(df)

for Key in metric_file:
    csv_object = s3.get_object(Bucket=bucket_name, Key=Key)
    csv_content = csv_object['Body'].read().decode('utf-8')
    df = pd.read_csv(StringIO(csv_content))
    li4.append(df)

frame = pd.concat(li, axis=0,ignore_index=True)
frame2 = pd.concat(li2,axis=0,ignore_index=True)
frame3 = pd.concat(li3,axis=0,ignore_index=True)
frame4 = pd.concat(li4,axis=0,ignore_index=True)
```

Appendix E: Iteration of different dataframe and transform into ontology

```
for index, row in frame.iterrows():
    class_ref = URIRef(cls+row['Class'])
    dependent_ref = URIRef(cls+row['HasDependencies'])
    g.add((class_ref,RDFS.subClassOf ,mainClass_ref))
    g.add((dependent_ref, RDFS.subClassOf, dependenciesClass_ref))
    dependent = URIRef(cls+row['Class']+'_has_dependent')
    g.add((dependent,RDF.type,OWL.ObjectProperty))
    g.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
    g.add((dependent,RDFS.domain, class_ref))
    g.add((dependent,RDFS.range, dependent_ref))

for index, row in frame2.iterrows():
    class_ref = URIRef(cls+row['Class'])
    method_ref = URIRef(cls+row['HasMethod'])
    g.add((class_ref,RDFS.subClassOf ,mainClass_ref))
    g.add((method_ref, RDFS.subClassOf, methodClass_ref))
    dependent = URIRef(cls+row['Class']+'_has_method')
    g.add((dependent,RDF.type,OWL.ObjectProperty))
    g.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
    g.add((dependent,RDFS.domain, class_ref))
    g.add((dependent,RDFS.range, method_ref))

for index, row in frame3.iterrows():
    class_ref = URIRef(cls+row['Class'])
    variable_ref = URIRef(cls+row['HasVariable'])
    g.add((class_ref,RDFS.subClassOf ,mainClass_ref))
    g.add((variable_ref, RDFS.subClassOf, variableClass_ref))
    dependent = URIRef(cls+row['Class']+'_has_variable')
    g.add((dependent,RDF.type,OWL.ObjectProperty))
    g.add((dependent,RDFS.subPropertyOf, OWL.topObjectProperty))
    g.add((dependent,RDFS.domain, class_ref))
    g.add((dependent,RDFS.range, variable_ref))

for index, row in frame4.iterrows():
    Atribute_ref = URIRef(cls+row['Attribute'])
    Quantity_ref = URIRef(cls+str(row['Quantity']))
    g.add((Atribute_ref,RDFS.subClassOf ,metricClass_ref))
    g.add((Quantity_ref, RDFS.subClassOf, quantityClass_ref))
    relationship = URIRef(cls+row['Attribute']+'_has_attribute')
    g.add((relationship,RDF.type,OWL.ObjectProperty))
    g.add((relationship,RDFS.subPropertyOf, OWL.topObjectProperty))
    g.add((relationship,RDFS.domain, Atribute_ref))
    g.add((relationship,RDFS.range, Quantity_ref))
```

Appendix F: Different Routes to retrieve different Data

```
@OntologyGenerator.route('/GetAllDependencies',methods=['GET'])
def getAllDependencies():
    data = []
    query = """
        SELECT ?domain ?property ?range
        WHERE {
            ?property rdfs:domain ?domain ;
                      rdfs:range  ?range .
        }
    """
    qres = g.query(query)
    for row in qres:
        if(row[1][-10::] == "_dependent"):
            data.append({"Class": row[0][46::], "Relationship": "Has Dependent" ,"Dependent": row[2][46::]})

        elif (row[1][-10::] == "has_method"):
            data.append({"Class": row[0][46::], "Relationship": "Has Method" ,"Method": row[2][46::]})

        elif (row[1][-10::] == "_attribute"):
            data.append({"Class": row[0][46::], "Relationship": "Has Attribute" ,"Attribute": row[2][46::]})

        else:
            data.append({"Class": row[0][46::], "Relationship": "Has Variable" ,"Variable": row[2][46::]})
    df = pd.DataFrame(data)
    df_filter = df[df['Relationship'] == "Has Dependent"]
    df_filter2 = df_filter.drop(columns=['Variable','Method','Attribute'])
    json_data = df_filter2.to_json(orient='records')

    return json_data
```

```
@OntologyGenerator.route('/GetAllVariable',methods=['GET'])
def getAllVairable():
    data = []
    query = """
        SELECT ?domain ?property ?range
        WHERE {
            ?property rdfs:domain ?domain ;
                      rdfs:range  ?range .
        }
    """
    qres = g.query(query)
    for row in qres:
        if(row[1][-10::] == "_dependent"):
            data.append({"Class": row[0][46::], "Relationship": "Has Dependent" ,"Dependent": row[2][46::]})

        elif (row[1][-10::] == "has_method"):
            data.append({"Class": row[0][46::], "Relationship": "Has Method" ,"Method": row[2][46::]})

        elif (row[1][-10::] == "_attribute"):
            data.append({"Class": row[0][46::], "Relationship": "Has Attribute" ,"Attribute": row[2][46::]})

        else:
            data.append({"Class": row[0][46::], "Relationship": "Has Variable" ,"Variable": row[2][46::]})

    df = pd.DataFrame(data)
    df_filter = df[df['Relationship'] == "Has Variable"]
    df_filter2 = df_filter.drop(columns=['Dependent','Method','Attribute'])
    json_data = df_filter2.to_json(orient='records')

    return json_data
```

```python
@OntologyGenerator.route('/GetAllMetric',methods=['GET'])
def getAllMetric():
    data = []
    query = """
        SELECT ?domain ?property ?range
        WHERE {
            ?property rdfs:domain ?domain ;
                      rdfs:range  ?range .
        }
    """
    qres = g.query(query)
    for row in qres:
        if(row[1][-10::] == "_dependent"):
            data.append({"Class": row[0][46::], "Relationship": "Has Dependent" ,"Dependent": row[2][46::]})

        elif (row[1][-10::] == "has_method"):
            data.append({"Class": row[0][46::], "Relationship": "Has Method" ,"Method": row[2][46::]})

        elif (row[1][-10::] == "_attribute"):
            data.append({"Class": row[0][46::], "Relationship": "Has Attribute" ,"Attribute": row[2][46::]})

        else:
            data.append({"Class": row[0][46::], "Relationship": "Has Variable" ,"Variable": row[2][46::]})

    df = pd.DataFrame(data)
    df_filter = df[df['Relationship'] == "Has Attribute"]
    df_filter2 = df_filter.drop(columns=['Dependent','Method','Variable'])
    json_data = df_filter2.to_json(orient='records')

    return json_data
```

```python
@OntologyGenerator.route('/GetDependenciesByClass/<Class>',methods=['GET'])
def GetDependentByClass(Class):
    data = []
    Input = "<http://semanticBasedRedocumentation.org/class/" + Class +"_has_dependent>"
    query = "SELECT ?domain ?property ?range \n" + "WHERE { \n" + Input + " rdfs:domain ?domain ; \n" + "rdfs:range ?range . \n"  + "}"
    qres = g.query(query)
    for row in qres:
        data.append({"Class": row[0][46::], "Relationship": "Has Dependent" ,"Dependent": row[2][46::]})
    df = pd.DataFrame(data)
    json_data = df.to_json(orient='records')

    return json_data
```

Appendix G: File-upload Blade view component

```
@section('content')
<div class="container">
    <div class="card">
        <div class = "card-header">
            Upload Your JAVA File Here
        </div>
        <div class="card-body">
            @if ($message = Session::get('success'))
                <div class="alert alert-success">
                    <b>{{ $message }}</b>
                </div>
            @endif
            @if($message = Session::get('failed'))
                <div class="alert alert-danger">
                    <b>{{ $message }}</b>
                </div>
            @endif
            @if ($errors->any())
                <div class="alert alert-danger">
                    <ul>
                        @foreach ($errors->all() as $error)
                            <li>{{ $error }}</li>
                        @endforeach
                    </ul>
                </div>
            @endif

            <form class="row" method="post" action="{{url('multiple-file-upload')}}" enctype="multipart/form-data">
                {{ csrf_field() }}
                <div class="col-auto">
                    <input type="file" name="fileuploads[]" class=" form-control" multiple accept=".java">
                </div>
                <div class="col-auto">
                    <button type="submit" class="btn btn-outline-primary mb-3">Upload Files</button>
                </div>
            </form>
            <table class="table table-bordered mt-3">
                <thead>
                    <tr>
                        <th>Filename</th>
                        <th>Filepath</th>
                        <th>File Type</th>
                    </tr>
                </thead>
                <tbody>
                @foreach ($fileUploads as $fileUpload)
                    <tr>
                        <td>{{ $fileUpload->filename }}</td>
                        <td><a href="http://localhost:8000{{ $fileUpload->filepath }}" target="_blank" rel="noopener noreferrer">{{ $fileUpload->filepath }}</a></td>
                        <td>{{ $fileUpload->type }}</td>
                    </tr>
                @endforeach
                </tbody>
```

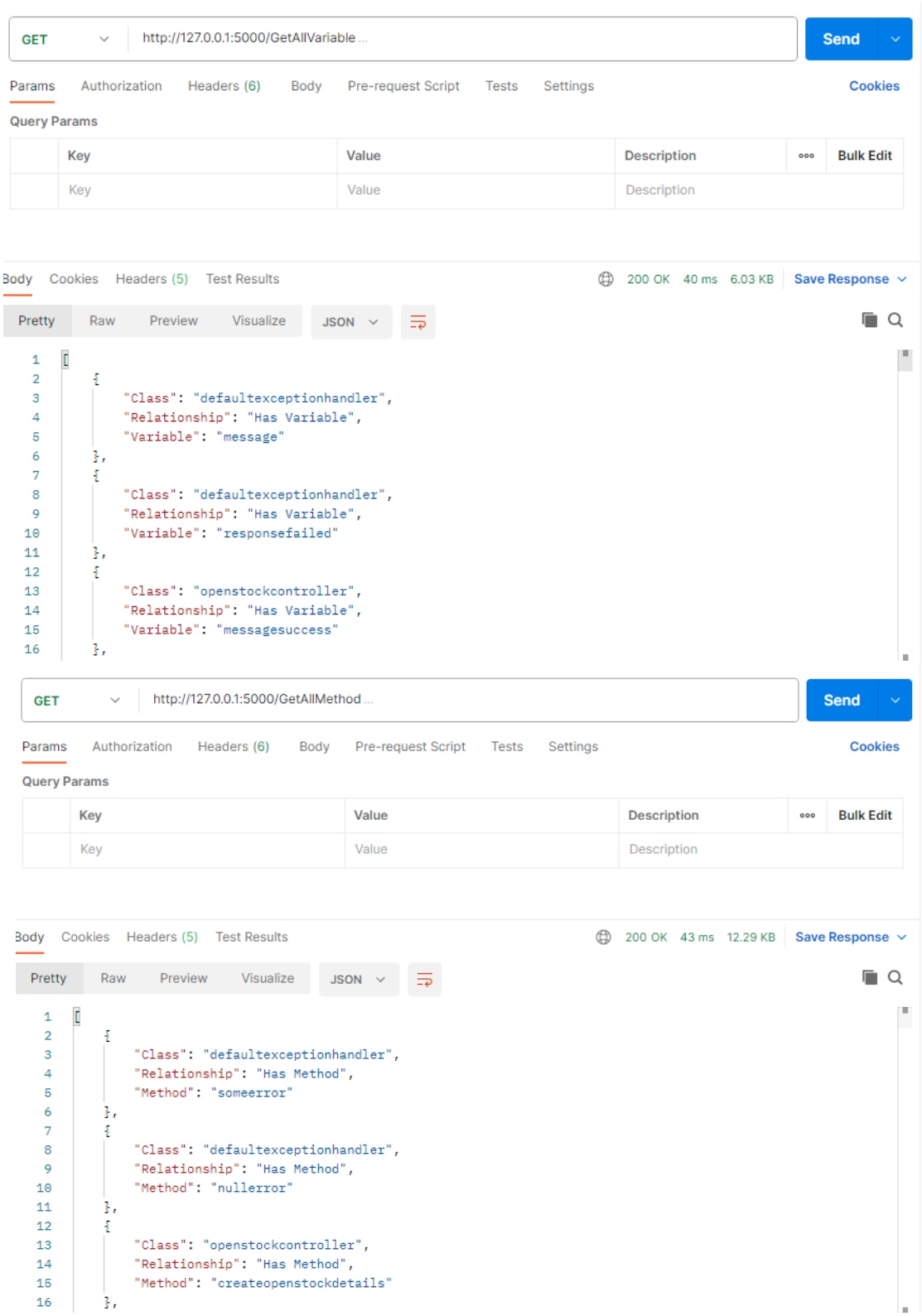
Appendix H: display-method blade view component

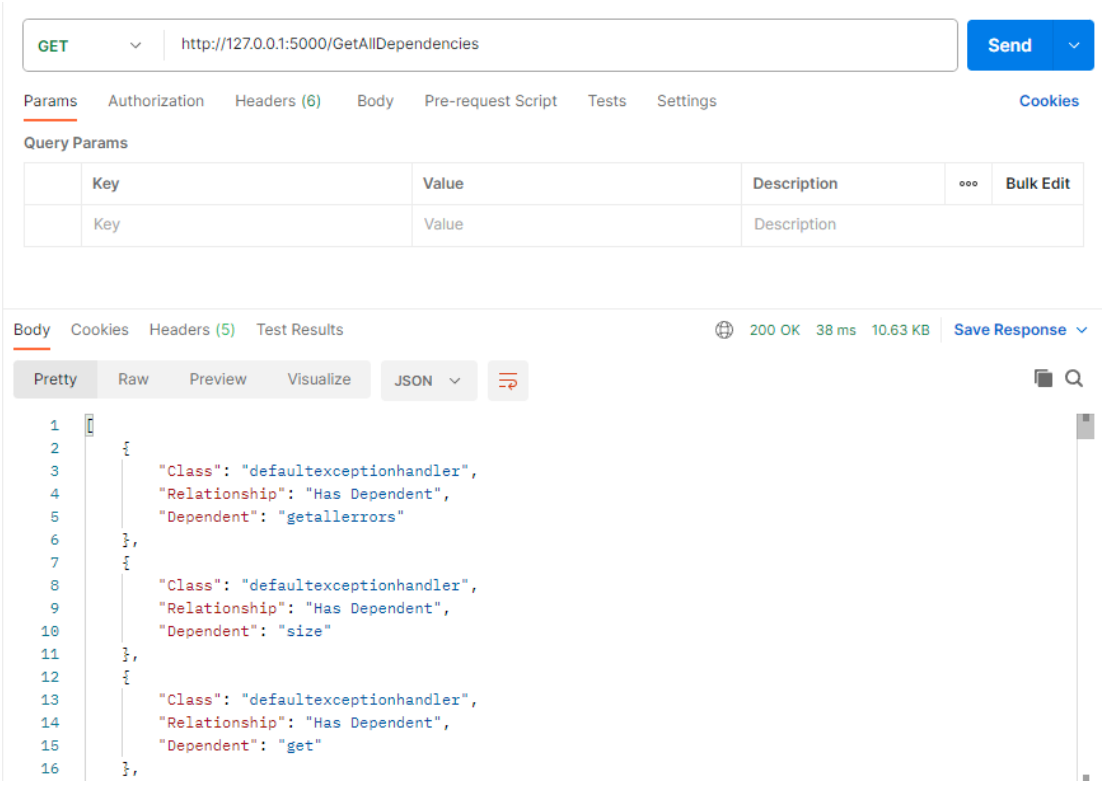
```
@section('content')
<div class="container">
    <div class="card">
    <div class="card-header">
            Class Methods
        </div>
        <div class="card-body">
            <form class="row" method="post" action="{{url('searchMethodByClass')}}">
                @csrf
                <div class="col-auto">
                    <input type="search" class=" form-control" placeholder="Search Method By Class" name="className">
                </div>
                <div class="col-auto">
                    <button type="submit" class="btn btn-outline-primary mb-3">Search</button>
                </div>
            </form>
            <table class="table table-bordered mt-3">
                <thead>
                    <tr>
                        <th>Class</th>
                        <th>HasMethod</th>
                    </tr>
                </thead>
                <tbody>
                @foreach ($Methods as $method)
                    <tr>
                        <td>{{ $method['Class'] }}</td>
                        <td>{{ $method['Method'] }}</td>
                    </tr>
                @endforeach
                </tbody>

            </table>
        </div>
    </div>
```

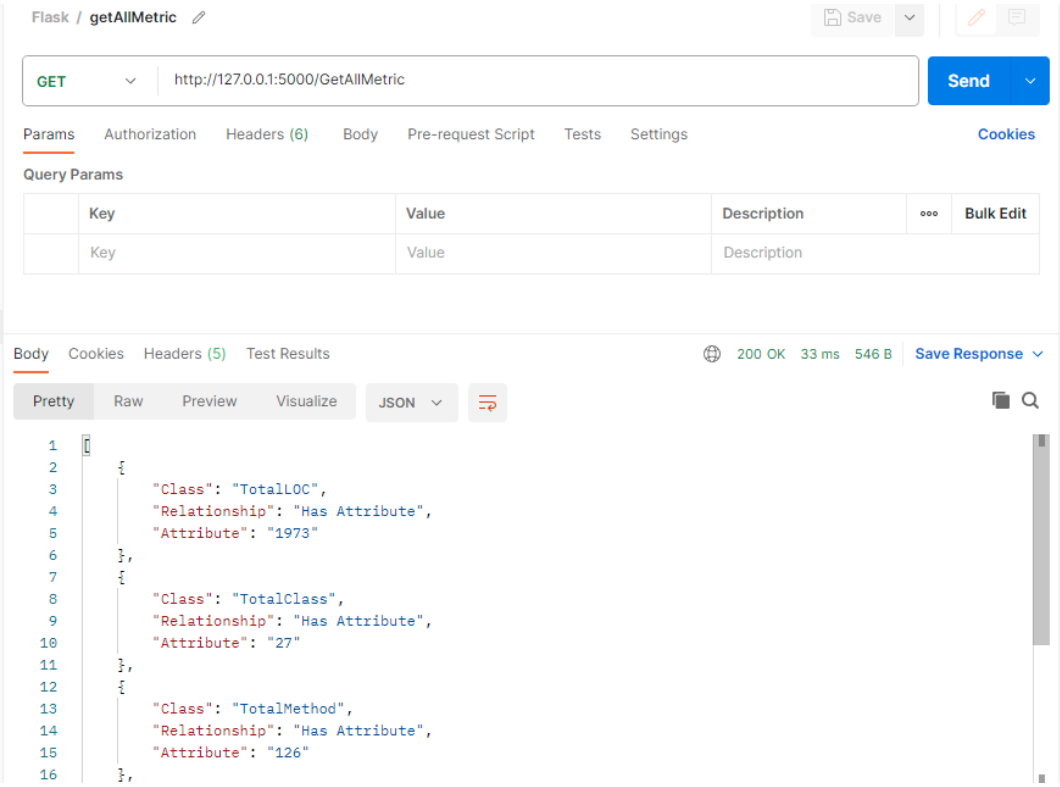
Appendix I: display-variable blade view component

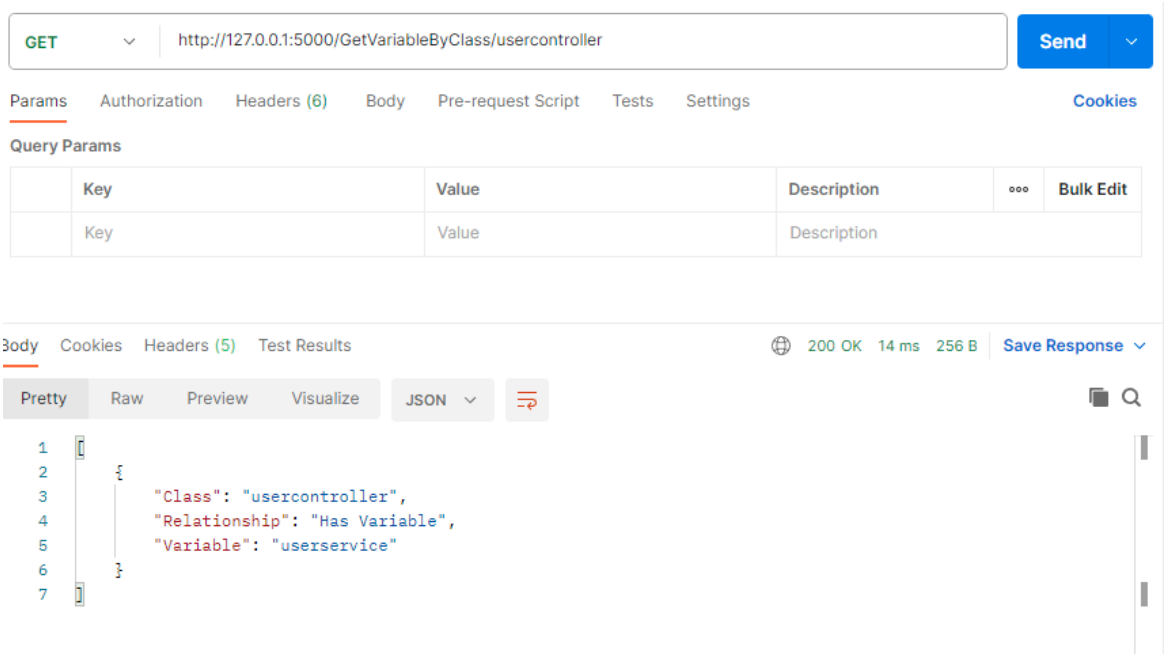
```
@section('content')
<div class="container">
    <div class="card">
    <div class="card-header">
            Class Variable
        </div>
        <div class="card-body">
            <form class="row" method="post" action="{{url('searchVariableByClass')}}">
                @csrf
                <div class="col-auto">
                    <input type="search" class=" form-control" placeholder="Search Variable By Class" name="className">
                </div>
                <div class="col-auto">
                    <button type="submit" class="btn btn-outline-primary mb-3">Search</button>
                </div>
            </form>
            <table class="table table-bordered mt-3">
                <thead>
                    <tr>
                        <th>Class</th>
                        <th>HasVariable</th>
                    </tr>
                </thead>
                <tbody>
                @foreach ($Variables as $variable)
                    <tr>
                        <td>{{ $variable['Class'] }}</td>
                        <td>{{ $variable['Variable'] }}</td>
                    </tr>
                @endforeach
                </tbody>

            </table>
        </div>
```

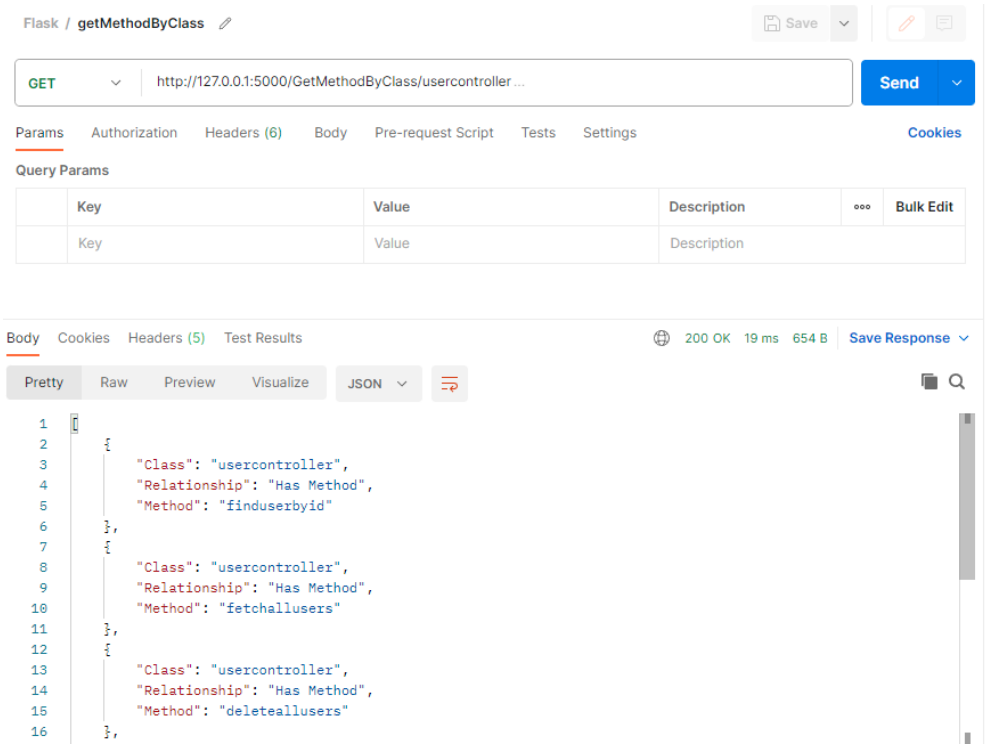
Appendix J: API Testing Result

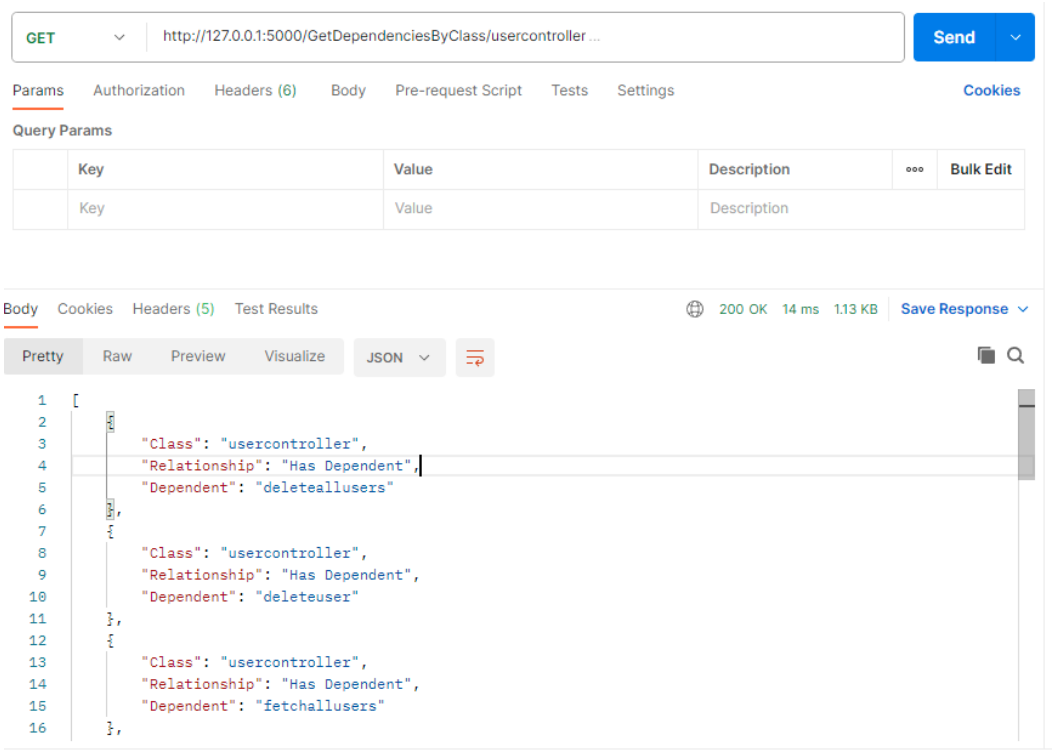



GET
http://127.0.0.1:5000/GetAllDependencies
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (5)
Test Results
200 OK
38 ms
10.63 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"Class": "defaultexceptionhandler",
"Relationship": "Has Dependent",
"Dependent": "getallerrors"
},
{
"Class": "defaultexceptionhandler",
"Relationship": "Has Dependent",
"Dependent": "size"
},
{
"Class": "defaultexceptionhandler",
"Relationship": "Has Dependent",
"Dependent": "get"
},


Flask / getAllMetric
Save
GET
http://127.0.0.1:5000/GetAllMetric
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (5)
Test Results
200 OK
33 ms
546 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"Class": "TotalLOC",
"Relationship": "Has Attribute",
"Attribute": "1973"
},
{
"Class": "TotalClass",
"Relationship": "Has Attribute",
"Attribute": "27"
},
{
"Class": "TotalMethod",
"Relationship": "Has Attribute",
"Attribute": "126"
},


GET
http://127.0.0.1:5000/GetVariableByClass/usercontroller
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
14 ms
256 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 [
2 {
3 "Class": "usercontroller",
4 "Relationship": "Has Variable",
5 "Variable": "userservice"
6 }
7]


Flask / getMethodByClass
Save
GET
http://127.0.0.1:5000/GetMethodByClass/usercontroller ...
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
19 ms
654 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 [
2 {
3 "Class": "usercontroller",
4 "Relationship": "Has Method",
5 "Method": "finduserbyid"
6 },
7 {
8 "Class": "usercontroller",
9 "Relationship": "Has Method",
10 "Method": "fetchallusers"
11 },
12 {
13 "Class": "usercontroller",
14 "Relationship": "Has Method",
15 "Method": "deleteallusers"
16 },


GET
http://127.0.0.1:5000/GetDependenciesByClass/usercontroller ...
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
14 ms
1.13 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"Class": "usercontroller",
"Relationship": "Has Dependent",
"Dependent": "deleteallusers"
},
{
"Class": "usercontroller",
"Relationship": "Has Dependent",
"Dependent": "deleteuser"
},
{
"Class": "usercontroller",
"Relationship": "Has Dependent",
"Dependent": "fetchallusers"
},