WEB-BASED SOURCE TO SOURCE CONVERTER

CHOOI KAR JIAN

A project report submitted in partial fulfilment 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

APRIL 2021

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	:	[illegible]
Name	:	Chooi Kar Jian
ID No.	:	1703498
Date	:	6/4/2021

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"WEB-BASED SOURCE TO SOURCE CONVERTER"** was prepared by **CHOOI KAR JIAN** 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	:	[illegible]
Supervisor	:	Chean Swee Ling
Date	:	3 May 2021

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Miss Chean Swee Ling for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement to complete this project.

Lastly, I would like to thank REXTESTER for allowing me to use their API for this project.

ABSTRACT

Software maintenance activity in the software development life cycle is becoming more difficult over time. Hence, many companies are interested in using automated code translation techniques to maintain their software. However, the existing automated code translators are still error prone and inefficient. Thus, this project is developed to improve accuracy of code conversion between high level languages, eliminate the need of manual conversion and promote universally compatible code conversion. The core functionality of the project will be developed based on a transpiler which convert codes into an abstract intermediate representation and to the desired target language. In this project, a code transpilation framework are developed with a frontend website. The code conversion model could achieve 90% accuracy. The result of the usability testing also showed that the system achieved a positive usability result. In conclusion, the project has been implemented successfully as it met the project's objectives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AST	Abstract Syntax Tree
API	Application Programming Interface
DOM	Document Object Model
SDLC	Software Development Life Cycle
SUS	System Usability Scale
UML	Unified Modelling Language
UI	User Interface
WBS	Work Breakdown Structure
REST	Representational State Transfer
ANTLR	Another Tool for Language Recognition
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Software needs to be maintained in order to keep up with growing requirements of the current world. Software maintenance are becoming more cumbersome as software complexity increases over time. While some company chooses to spend large expenditure on software maintenance each year, other companies opt to reimplement and migrate their software program into another platform for better performance and maintainability. Code migration into another programming language can be achieved through transpilation process. A transpiler has the same concept as the compilers, but instead of converting the codes into lower-level language, transpiler will convert the codes into same abstraction level of programming language.

Transpiler is certainly useful for automated source code conversion but it may still require manual intervention from the programmers because the technology is relatively new. Hence, this project is initiated to analyse the issues of the transpilation process and propose suitable solution to resolve the issues. This chapter shall discuss the background of the problem, problem statements, project objectives, proposed solution, proposed approach and the scope of the project.

1.2 Background of problem

Software maintenance is one of the most important activity in software development life cycle. In fact, 70% of the resources are allocated to maintain the software codes (Christa *et al.*, 2017). According to Hunt and Thomas (2002), programmers tend to fix software bugs using update patches without understanding the underlying problem that causes the failure to happen. These software patches will not only increase the complexity of the codes but also increase the difficulty of software maintenance process for the future programmers. In addition, the codes tend to be more complex especially in a software project development that involves a lot of developers (Midha, 2008).This is due to the code inconsistency and different style of programming introduced by different developers in the development team. As a consequence, large amount of time and effort will be wasted to understand the logic and the relationship of the source code rather than fixing it (Smith, Capiluppi and Fernández-Ramil, 2006). This ideology is supported by Subramanian, Pendharkar and Wallace (2006) who stated that the software maintenance cost is directly affected by the code complexity of the software.

Based on Lumb (2018), many companies are turning their attention towards automated code translation techniques to update and maintain their software. Despite convenience that the system provides, the code translation process still needs programmers to be involved because the system was not able to identify the dependencies between different modules. Hence the code translation process is done partially rather than fully automated. Furthermore, source code translation must be done properly because it comes with risks that could cause the software to fail (Kontogiannis *et al.*, 2010). Despite all the negative effect that might come with the code translation system, automated code conversion tends to have lower risk compared to the other approach of updating or maintaining the system (Dahaner *et al.*, 2018).

It is undoubtedly true that code maintenance is a time consuming and resource heavy process. Programming languages will receive updates periodically to ensure that it is good enough to cater for the growing software requirements. Hence, this paper shall look into the problems of source-to-source translation and shall propose a solution to resolve the stated problems.

1.3 Problem Statement

This section shall describe the problems in two approaches in code conversion process. The first statement will address the problems in the manual code conversion process and the second statement shall cover the problems in the currently available code conversion system. The following issues shall be resolved with the completion of the project.

1.3.1 Cost ineffectiveness of manual conversion

Source code conversion process is very tedious and time consuming especially without the usage of automation software. Although there are software tools that can aid the conversion process, some company still perform manual code conversion using man labour. Ultimately, this approach is not cost efficient and effective for the software company because of the reasons stated as below:

i. **Time consuming**

   Source code conversion requires deep understanding of the original code before the it can be carried out. Hence, programmers will spend most of the time understanding the codes rather than performing the code conversion (George *et al.*, 2010). Besides that, the time taken for the process is affected by the complexity of the codes which means that longer duration will be required to convert a complicated software than a simple and well-defined software.

ii. **Inconsistent conversion**

   Manual code conversion is prone to mistakes especially the software which is not well documented. The programmer who does not completely understand the workflow of the software might risks losing the of the business rules when performing manual rewrite of the program (Ilyushin and Namiot, 2016). Other than that, the translated code might be inconsistent due to the unique programming styles from different programmers who are involved in the project. As a result, future maintenance task on the software will become difficult.

iii. **Expensive**

Manual source code conversion is costly because most of the project expenditures are spent on human resource for the project. Besides that, the cost of the conversion project will increase significantly with the duration of the project. According to George *et al.* (2010), rewriting the program manually will take years and requires a lot of manpower. Worst of all, there are chances that manual rewriting of a program will results in broken functionality which will cause financial damage to the company.

1.3.2 Error prone code conversion system

One of the main concerns for a code conversion system is the accuracy of the translated source code. The main goal of the system is to translate the source code into another programming languages without changing the meaning to the original code. However, the accuracy of the translated code depends on the ability of the code conversion system to capture the code structure and translate it to the target language correctly. Incorrect translation will modify the definition of business rules and program flow. As a result, intervention from the programmers is required and the system are only able to perform partial translation rather than a full translation (George *et al.*, 2010).

1.3.3 Language specific architecture

Code conversion process involves a sequence of task to break down the codes so that it can be translated into another programming language. However, the internal components that are responsible to process the codes are highly dependent on a specific programming language. This reduces the flexibility to convert between languages as new intermediate representation of the codes are required to be generated for each conversion process (George *et al.*, 2010). As a result, the efficiency of the conversion process will be affected.

1.4 Project Objectives

This project aims to achieve the following objectives:

i. To identify the issues and practice of the current code conversion process.

ii. To develop a web-based transpiler that is universally compatible with mainstream programming languages.

iii. To design a code transpilation framework.

iv. To achieve code translation accuracy score of 90% for the proposed source to source converter.

1.5 Project Approach

To effectively solve the problems identified in the code conversion process, a web-based source to source converter has been developed. A web interface was prepared to allow user interaction with the system. The primary purpose of the solution is to provide automated source code conversion using transpiler technology.

1.5.1 Transpiler Architecture

A transpiler was used as the back-end processing of the source-to-source converter system. The architecture of the compiler is similar to the compiler which consists of front-end analysis and back-end synthesis phase (Aho *et al.*, 2007).

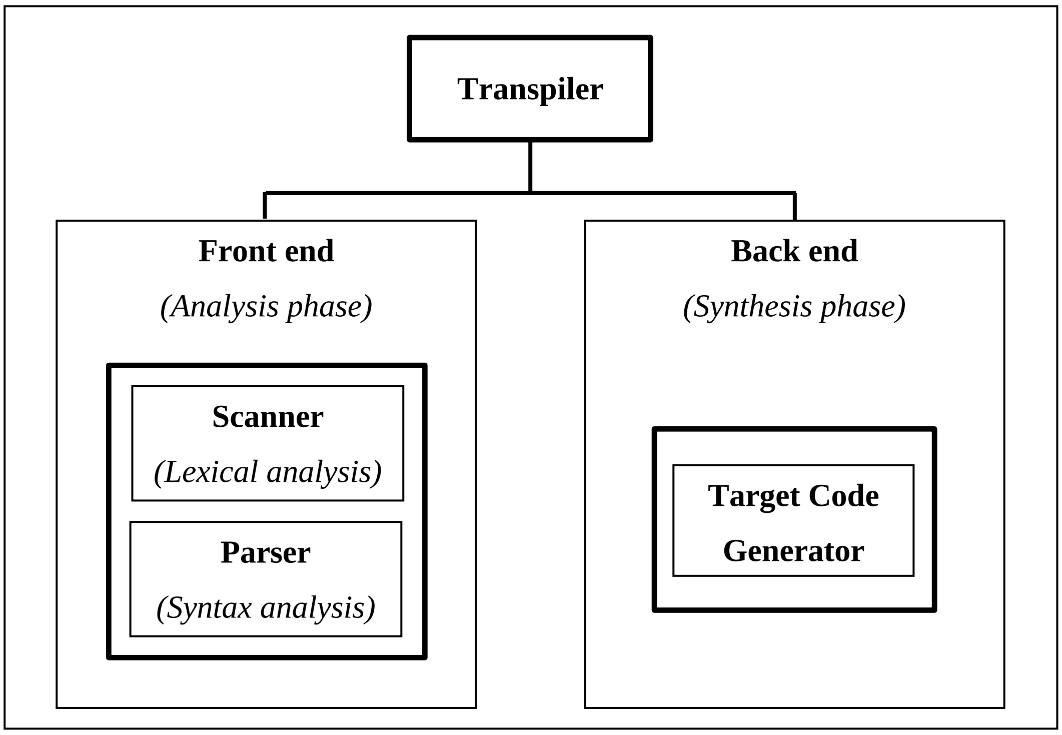


Figure 1-1 Architecture of the proposed transpiler

The front-end of the transpiler are responsible for tokenizing and parsing the source code into an AST meanwhile the back end of the transpiler will process the abstract syntax tree to the target code. The detailed implementation of the transpiler will be discussed in the later chapters.

1.5.2 General architecture of the system

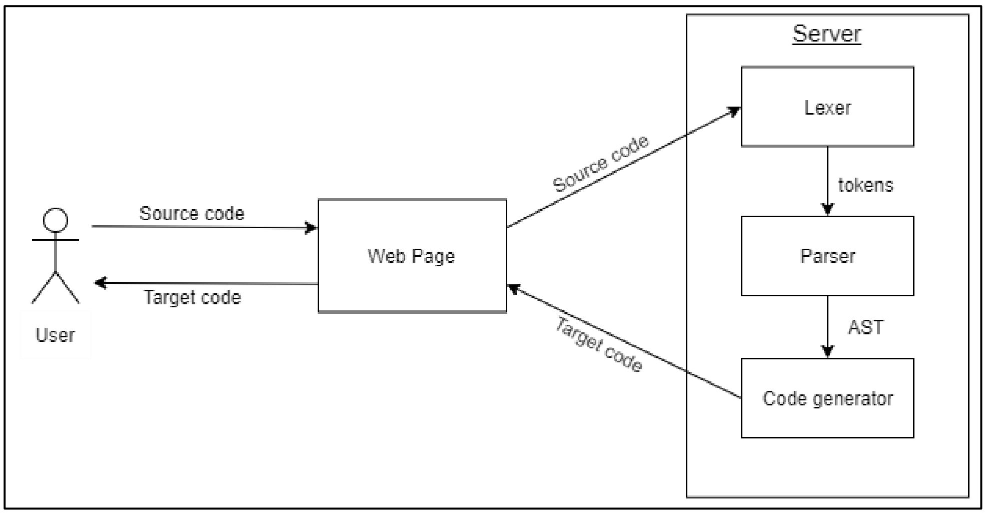


Figure 1-2 General architecture of the system

A web page was designed to allow user interaction with the proposed system. After user input the source code into the web page, then the server will process and translate the codes. Finally. The code in the targeted programming language will be returned to the user. The communication between the web page and the server is using RESTful API. The details of the communication between webpage and the server will be discussed in the later chapters.

1.6 Scope of the Project

This section includes the scope of the project which defines the backend transpiler module, supported conversion structure, the front-end features as well as the uncovered scope. Due to the time constraints and limited knowledge, the project scope has been narrowed down to focus on the conversion of the source code. Nevertheless, the code conversion system shall provide a web interface for the user to interact with the system.

1.6.1 Transpiler modules

A transpiler will be used as the backend processing of the system to translate the source codes. The transpiler will be written in JavaScript language since it provides a lot of flexibility. The backend processing system shall be divided into the following modules:

i. **Universal Lexer**

Lexical analysis will be carried out on the original source code by a lexer. The source code will be broken down into tokens where they are differentiated into literals, symbols and language specific keywords.

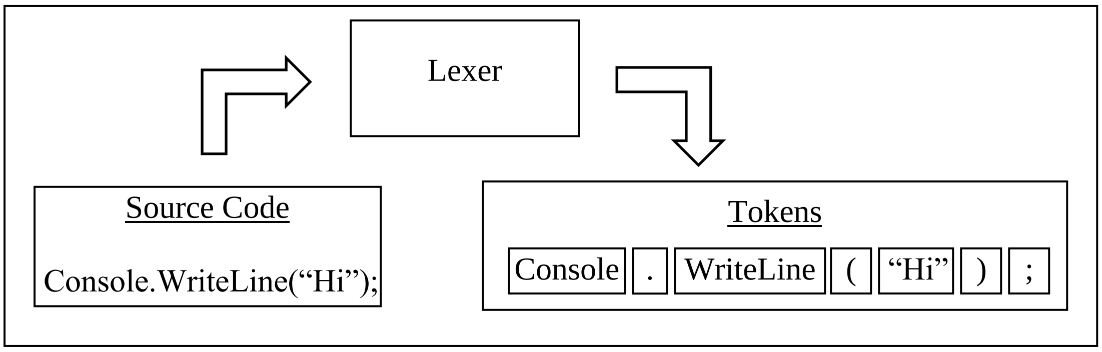


Figure 1-3 Lexical analysis process

ii. **Universal Parser**

A parser will take the sequence of tokens that are generated by the lexer to be parsed into an abstract syntax tree (AST). AST is an intermediary product that represents the abstract representation of the source code which are not dependent on any programming language.

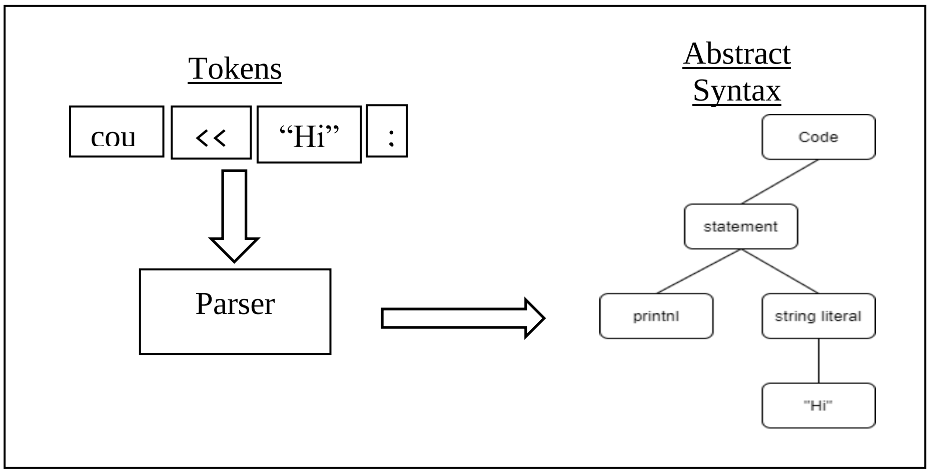


Figure 1-4 AST parsing process

iii. Code generator

The target code will be generated by the code generator using the AST created by the parser. The elements in the AST will be mapped onto the target language’s syntax and generate the code that have equivalent function from the source code.

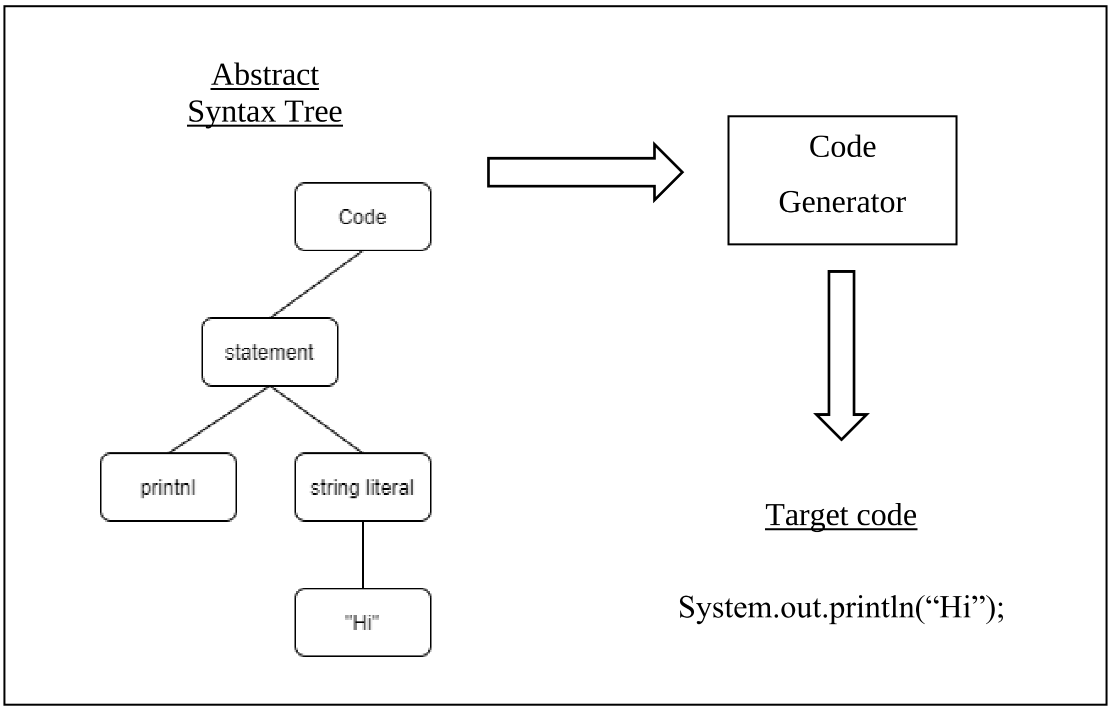


Figure 1-5 Code Generation Process

1.6.2 Supported conversion structure

Conversion of source code between two same level abstraction programming language is complicated because of their unique syntax and features. Hence, the proposed code conversion system is designed to convert the basic programming structure as the following:

A. Programming Fundamentals

i. Variables
ii. Mathematical operators (+, -, *, /)
iii. Logical operators (AND, OR, NOT)
iv. Selection operations (IF, IF...ELSE, SWITCH)
v. Looping operations (FOR, WHILE)

B. Object Oriented Programming

i. Class
ii. Object
iii. Inheritance
iv. Polymorphism

1.6.3 Web page features

As stated above, a web interface was provided to the user so they could interact with the system with minimal effort. The following features are included in the front-end website of the project:

A. Code conversion

The user shall be able to input the source code as plain text or as a programming language specific file (e.g., code.cpp). The webpage should communicate with the server for the translation of the source code. The output of the server will then be passed back to the user via the web page.

B. Code compilation

A code compiler will be integrated into the web interface using a third-party API. The user of the website can compile and execute the code. The rationality of this feature is to provide convenience to the user so that they could perform code debug at the website.

1.6.4 Uncovered scope

The project will not cover the following features:

i. API migration
ii. Database migration
iii. Code optimization
iv. Code semantic analysis

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Code conversion is not an easy process because it requires deep understanding of the programming languages and the conversion workflow. Therefore, literature review was conducted to gain understanding on areas related to the proposed idea of the project. Studies will be carried out to further improve the project. This literature review aims to:

1. Review similar system and past work
2. Understand the concept of a transpiler
3. Identify potential issue in code conversion process
4. Determine project methodologies to be used

2.2 Similar System

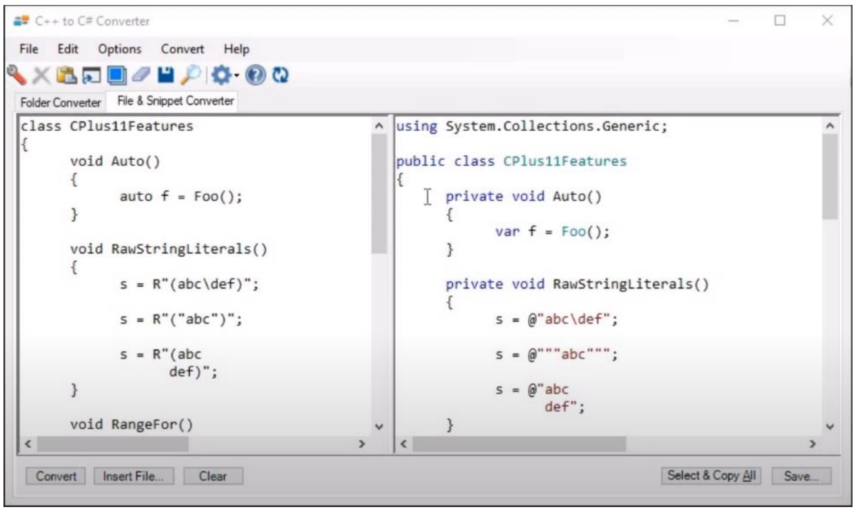
There are existing code conversion systems that can be accessed online whether it is published commercially or open sourced. Review on two popular code conversion system will be conducted to learn about the backend code conversion process and the additional functionalities that are provided to the users.

2.2.1 Java2Python

Java2Python is an open sourced code translation system that translates codes from Java language to Python language. Melhase {2012) explained that the system uses the concept of mapping where the identifiers and common operations were mapped from the source to target. However, problem arise when identifier name conflicts with keyword from another programming language. To solve the issue, explicit lexical transformation will be required to modify the identifier name so no error will occur in the translated program. The code conversion process is simple, the source codes are tokenized and sorted to build an abstract syntax tree using ANTLR. Then, tree traversal process will start extracting nodes from the tree and map them into target language.

2.2.2 Tangible Software solution

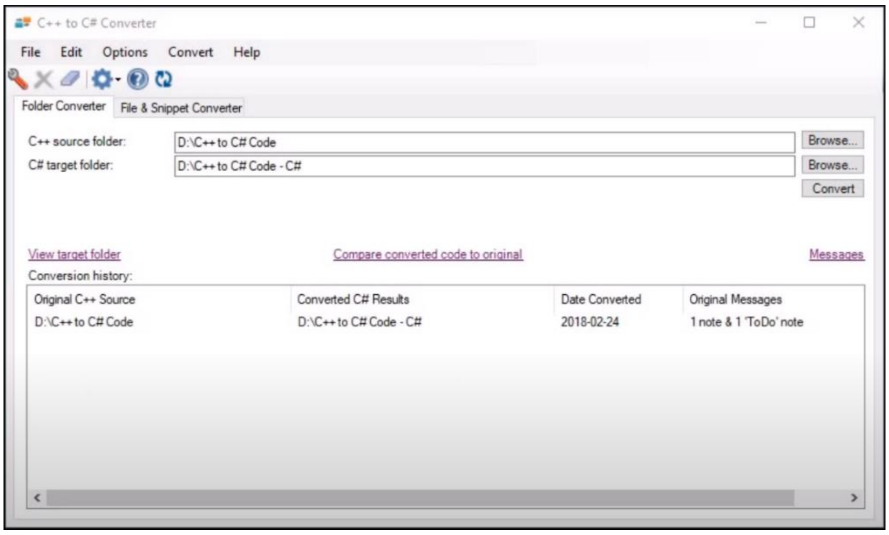
Tangible software solution is a company that specialize in creating code conversion software. Their software could translate between various programming languages which includes C++. C#, Java and VB.NET. The software was published commercially, and there is limitation on the conversion output for the free version. There were no implementation details for the code conversion process. However, analysis have been done on the software application to assess the features and user interaction design. The user interface is minimalistic and provides code conversion process through file upload or code snippet.



```
class CPlus11Features
{
        void Auto()
        {
                auto f = Foo();
        }

        void RawStringLiterals()
        {
                s = R"(abc\def)";

                s = R"("abc")";

                s = R"(abc
                      def)";
        }

        void RangeFor()
```

```
using System.Collections.Generic;

public class CPlus11Features
{
        private void Auto()
        {
                var f = Foo();
        }

        private void RawStringLiterals()
        {
                s = @"abc\def";

                s = @"""abc""";

                s = @"abc
                    def";
        }
```

Figure 2-1 Code conversion through code snippet



C++ source folder: D:\C++ to C# Code

C# target folder: D:\C++ to C# Code - C#

Conversion history:

Original C++ Source	Converted C# Results	Date Converted	Original Messages
D:\C++ to C# Code	D:\C++ to C# Code - C#	2018-02-24	1 note & 1 'ToDo' note

Figure 2-2 Code Conversion through file upload

Similar System	Strength	Weaknesses
Java2Python	• Perform conversion using existing technology (ANTLR). • Perform conversion by breaking down codes into tokens and forming a abstract syntax tree before mapping them into the target language.	• No user interface to enable user interaction. • Only converts between Java and Python.
Tangible Software solution	• Offers more programming language selection to the user to perform code conversion. • Have simple user interface for user to convert codes by importing the files.	• Limited conversion to free users.

Table 2-1 Comparison table on existing application

2.3 Past Work

The idea of translating programming languages has been discussed over the years because the code translation system has potential in various areas of software development lifecycle. Literature review will be conducted on two past research papers to discuss the common practise and future recommendation on the code translation process.

2.3.1 JPT: A Simple Java-Python Translator

In the research done by Coco, Osman and Osman (2018), they proposed that the code conversion shall analyse the similarities and differences between two different programming languages before performing code conversion. This is because different programming languages have different features that are unique to other languages, hence understanding of both programming languages are required to ensure that the code conversion process can be performed accurately and effectively. The paper proposed that the intermediate language that is created during the code conversion process can be written in XML format because XML are both human readable and machine readable. It will be easier for debugging effort. However, the process of parsing the source code to XML representation format is very time consuming and resource intensive. Hence, more effort will be needed to ensure that the intermediate language created will be efficient and effective.

2.3.2 Programming language Inter-conversion

George *et al.* (2010) had analysed many research papers that was relevant to the code conversion process and found out that the implementation of an intermediate language would benefit the code conversion process. The intermediate language should be abstract which means that it is not dependent on any programming language. Hence, it will be affective to store the logic of the program in an algorithmic format without disturbing the original structure of the program during the code conversion process. The converter can be designed in such a way that it could convert the common components of both programming languages and have the ability to map special functions between the programming languages. Lastly, George *et al.* (2010) suggested that predefined library can be prepared to convert algorithm between languages more efficiently.

2.4 Transpiler architecture

This project will involve transpilation process from a program called as a transpiler which is very similar to a compiler. Hence, an understanding of compiler technology is required before implementing the transpiler as the backend service of the proposed system.

In programming context, a compiler is a program that translate higher level abstraction source code into lower level target code that are semantically equivalent (Aho *et al.*, 2007). A typical compilation process will take high level language codes such as Java or C# and convert them into an intermediate representation of the source codes. Then, the intermediate representation of the source codes will be translated into the target language through mapping techniques. Unfortunately, most of the compilers are not universally adaptable to different programming languages as the internal modules of the compiler are highly specific to a programming language (Plaisted, 2013). In other words, many compilers are only able to recognise a specific syntax of a programming language.

A transpiler have similar components and workflow as the compiler which consists of a lexer, a parser and a code generator and the only difference between them is the abstraction level of the target language (Kulkarni, Chavan and Hardikar, 2015). Instead of conversion of source code to lower-level target codes, a transpiler would convert source code between programming languages that have the same level of abstraction. For example, a transpiler can convert Java code into C# code and vice versa. Other than that, the workflow of the transpiler and compiler are similar. The main tasks that need to be carried out by the program are:

a. Lexical analysis

According to Farhanaaz and Sanju (2016), lexical analysis is responsible for breaking down the source codes into lexemes using a language pre-processor. In other words, the lexical analysis will decompose lines of codes into tokens and remove any white spaces from the codes. Each lexeme contains a tag that describe the type of data they store. For example, “int” token will be tagged as a built-in system data type. Before breaking down the lines of codes, a symbol table will be needed to define the language specific keywords such as “goto” in C language. Then, the lexer will analyse the codes and tokenize the lines of codes according to the symbol table and place them into a queue to be passed to the parser to carry out syntactic analysis.

b. Syntactic analysis

The tokens that are generated from the lexer will be passed to the parser where syntactic analysis will take place. According to Kulkarni, Chavan and Hardikar (2015), syntactic analysis will parse the tokens to form a tree that is called as a syntax tree. The syntax tree can be considered as the intermediate representation of the source code because the syntax tree will only store all the details about the source code. There are two type of syntax tree with different abstraction level, a parse tree and an abstract syntax tree. A parse tree is highly specific to the source code. In other words, the tree is language dependent and less flexible. On another hand, the abstract syntax tree only preserve the structure and the process of the source code which means that it is not tied to any programming languages (Ilyushin and Namiot, 2016).

c. Code generation

After the intermediate representation of the code is generated, it will be passed to the code generator. The responsibility of a code generator is to generate the target code using the intermediate representation. If an abstract syntax tree was used, tree traversal will be performed on the tree to extract the nodes and map it to the corresponding target code programming language.

Based on the research done by Mu (2019), there are two architectures that define the workflow of a transpiler. The first architecture is called as Trans-To-IR (TTIR) which parses source codes to AST and then transformed the AST into language specific IR. The IR is then compiled and run by the interpreter. The advantage of this architecture is the converted code will be optimized and efficient and the disadvantage is the converted code are not human readable, hence it is impossible to perform debugging process on the converted code. The second architecture is called as Source-Lang-To-Target-Lang (SLTL). In this architecture, the source code will be parsed to AST and then translated to the target language. The advantage of this architecture is it promotes re-use of parser modules because the structure of intermediate representation is defined. Other than that, the target code generated is human readable. However, this architecture does not come with code optimization.

2.5 Concerns in code conversion process

A transpiler contains multiple components that work together to produce a specific functionality, that is, to translate a source code between different programming languages at the same level of abstraction without modifying the structure or business rules of the original source code. However, the process of building a transpiler system is not easy because it involves deep understanding of the system construction process. Moreover, testing the correctness of the transpiler will be a challenge because of the system complexity and the uncertainty to correctly evaluate the performance of a built transpiler. Hence, research is done on relevant articles and past research papers to find out the possible factors that will affect the decision making during the construction of the project and the evaluation method to test the correctness of the transpiler system.

According to Ilyushin and Namiot (2016), there are a few requirements that need to be achieved while building a transpiler. The first requirement to be achieved is to ensure that the transpiler could translate a source program to a different programming language program without modifying the original structure or semantic. This statement was supported by George *et al.* (2010) who commented that the aim of performing programming language conversion is to transform the codes into another language while ensuring the consistency of the program structure between the source code and the translated code.

Other than that, Ilyushin and Namiot (2016) also pointed out that both of the source program and translated program must be able to produce the same output. This is because the translation of codes should not affect or modify the functionality of the original program. It is important to ensure that the translated program could inherit the business rules defined from the original program so the translated program will not affect the business process. Lastly, the code conversion process should have minimal user interaction with the system. In other words, the system should be able to perform automated code conversion without interception from the user.

As mentioned above, the main concern of the transpiler is to ensure that the program structure and process can be translated to the target programming language. Hence, the accuracy of the transpiler can be measured according to the similarity of source program and translated program’s structure and output. An abstract intermediate representation of both source program and target program can be compared to measure the accuracy of the program translation process. This idea was motivated by Plaisted (2013) who suggests that two sets of codes which are syntactically equivalent should be able to produce a similar abstract representation. He also suggests that the implementation of an abstract intermediate representation during the code conversion process can effectively preserve the structural information of the source program.

After reviewing the relevant journal and past research papers, it is clear that a transpiler plays an important role in code conversion process because it can eliminate manual code conversion process which are error prone. However, intervention of programmers will still be needed for the process because different programming languages have their own specialized features. On the other hand, the idea of creating an abstract intermediate representation during code conversion process is adopted widely when constructing the transpiler. This is because it provides an abstraction level that could capture important component in the source program such as the working of an algorithm without dependency on any programming language syntax. On top of that, the abstract intermediate representation could be transformed into different programming languages because it is universal and contains only the details of implementation.

In short, a transpiler need to be able to convert a source program into another programming language without losing the structure and process of the original source program. Other than that, the entire code conversion process must be done with minimal user intervention to the process. Finally, the accuracy of the transpiler can be measured by comparing the abstract intermediate representation and output between the source program and the target program.

2.6 Intermediate representation

The transpilation process will involve a generation of an abstract intermediate representation which could represent both of the source code and target code. An understanding of intermediate representation is needed because it is crucial for the success of the code conversion process in the proposed system. This section shall summarize the observation and results regarding the performance of the different intermediate language that can be used to generate the intermediate representation.

Intermediate representation of the source code can be generated to help the code conversion process because it can effectively preserve the structure of the source code and translate the tree into the target code (George *et al.*, 2010). After conducting research on few relevant research papers, intermediate representation is most commonly written in three different languages which are XML, JSON and YAML. Based on the performance evaluation done by Eriksson and Hallberg (2011), YAML is better at storing deep hierarchical data or very complex data compared to XML and JSON. Other than that, JSON could provide better performance and parsing speed compared to YAML and XML. XML have the worst performance among the three because it uses tags to encapsulate the codes which uses a lot of resources. Hence, the performance of XML is poor. They also proposed a list of criteria for the selection of the intermediate representation.

The main selection criteria for the project depends on the functionality, readability and the performance of each intermediate language. JSON is the most suitable for the proposed project because it has the best performance in data retrieval process. Besides that, JSON is easy to be parsed and retrieved. The readability of the intermediate representation is given lower priority because it is not important for this project.

2.7 Development methodology

A software development methodology is a framework that can guide a developer to carry out software project more efficiently and more organised. It is important to choose a software development methodology based on the nature of the project to ensure the project can be carried out successfully. According to Kumar and Bhatia (2014), different methodology have different concept on the lifecycle. In this section, seven different models will be compared.

There are two main types of software development methodology that are predictive life cycle and adaptive software development life cycle (Schawalbe, 2020). Predictive life cycle will be suitable for project which the cost, time and requirements can be well defined at the early stage. One of an example of predictive life cycle is waterfall model. Waterfall model can be considered as the oldest methodology that still exists today. The methodology is not flexible because each phase needs to be signed off by the stakeholders before the next stage can begin. It also means that the requirements must be well defined at the early stage because the any changes from the previous phase would result in project schedule delay.

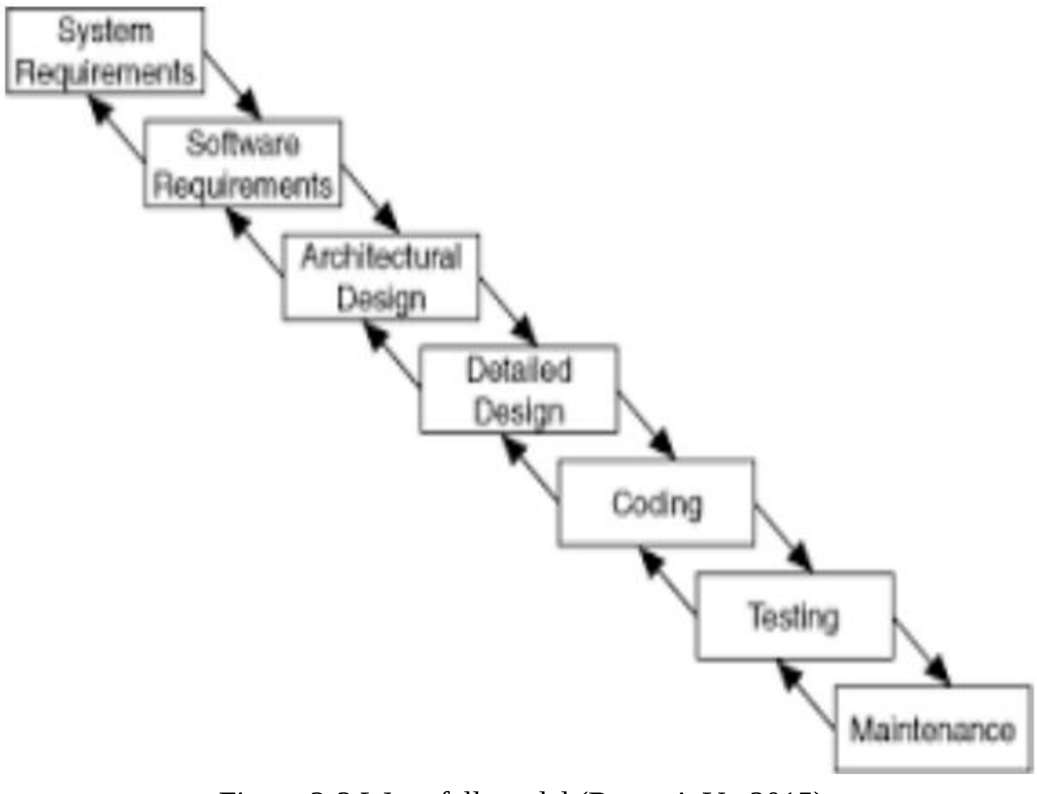


Figure 2-3 Waterfall model (Rastogi, V., 2015)

V-Shaped model is also one of the predictive life cycle models which means that the requirements of the project must be well defined at the early stage. This model is similar to the waterfall model, but it involves user in the early stages for software testing. It is certain that both waterfall model and V-Shaped model are inflexible in requirements change.

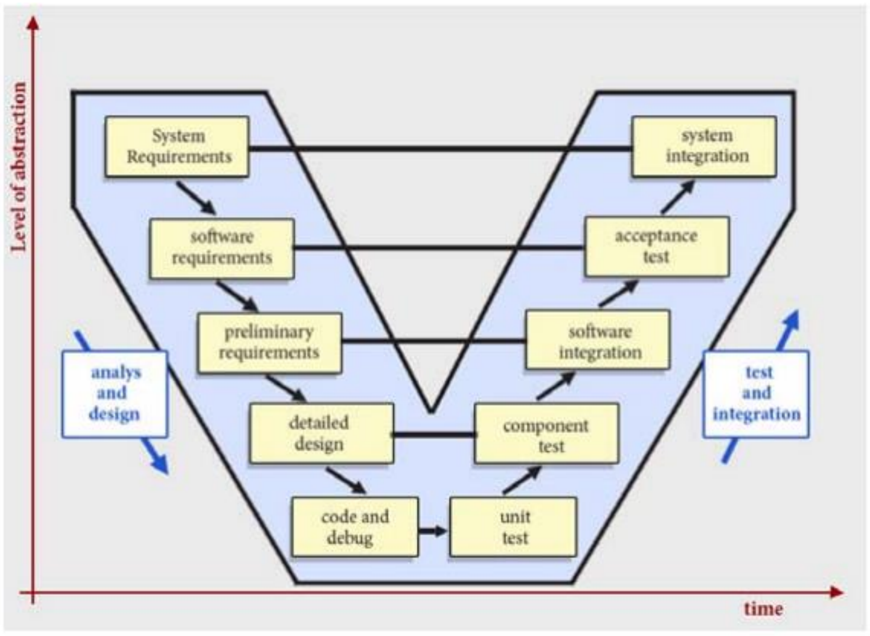


Figure 2-4 V-Shaped model (Kumar and Bhatia, 2014)

Moreover, predictive life cycle model also includes iterative model. Iterative model is different from the waterfall and V-Shaped model in the sense that it does not require all the requirements to be specified before the project started (Rastogi, 2015). The idea of this model is that the entire software development is divided into few iterations with waterfall model in each iteration. One of the benefits of this model is feedback can be gained from the previous iterations.

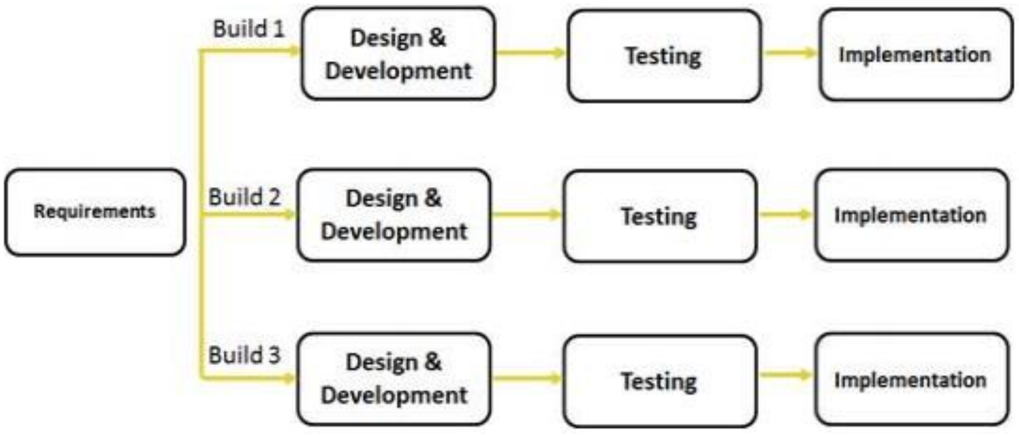


Figure 2-5 Iterative model (Rastogi, 2015)

On the other hand, adaptive life model consists of agile model which give emphasize on the customer satisfaction by providing continuous software delivery (Rastogi, 2015). In other words, the agile development could respond to the changing requirements rapidly in a quick succession. The main priority for this model is to achieve customer satisfaction.

Each methodology has their own unique workflow. The choice of software development methodology to be adopted depends on the nature of project. Iterative and iteration methodology is most suitable for the project because this project contains a lot of uncertainties from the technology that are not widely discussed. Hence, requirements might change from time to time so that the project can achieve the final goals of the project. Other than that, the proposed project contains multiple domain of knowledge such as back-end server and front-end webpage that could be messy in the later stage. Hence, iterative incremental model is adopted because it can effectively separate the development into few iterations to implement the most important feature at the beginning.

2.8 Summary

In brief, this literature review had covered 4 different areas that could benefit the development phase of the proposed system. First of all, the proposed system shall implement similar backend architecture and design that is similar to the existing system because they are proven to be beneficial for the system. Besides that, additional features will be added to the proposed system to provide convenience for the system user.

Secondly, the concept of a transpiler is similar to a compiler which contains internal components such as lexer, parser and code generator. The proposed system shall be able to perform lexical analysis, syntactic analysis and target code generation to perform code conversion process to another programming language.

Next, the proposed system shall include a backend transpiler that can perform code conversion between programming language without compromising the structure and process from the original source program. Other than that, the accuracy of the conversion can be measured by comparing the abstract intermediate representation and the output of the source program and translated problem.

Lastly, comparisons between different software development lifecycle models have led to a conclusion that iterative incremental model is the most suitable development methodology for the project.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter will cover the details of the phases in software development life cycle, work breakdown structure as well as the Gantt chart of the project development.

3.2 Iterative incremental model

Iterative incremental model will be used as the software development life cycle model of this project. The main concept of the model is to break down the entire software development process into few phases and implement each phase according to priority of the planned deliverables. In order words, higher priority deliverables will be implemented in the first iteration of the software project.

The software development will be divided into three main phases for this project. Each phase will contain requirement gathering, analysis and design, implementation and testing process as shown in the diagram below. The backend transpiler shall be implemented in the first iteration of the project as it is an important component in the project. Then, the frontend website will be delivered in the second iteration. Lastly, the connectivity of the frontend website and backend system will be established in the final iteration of the project.

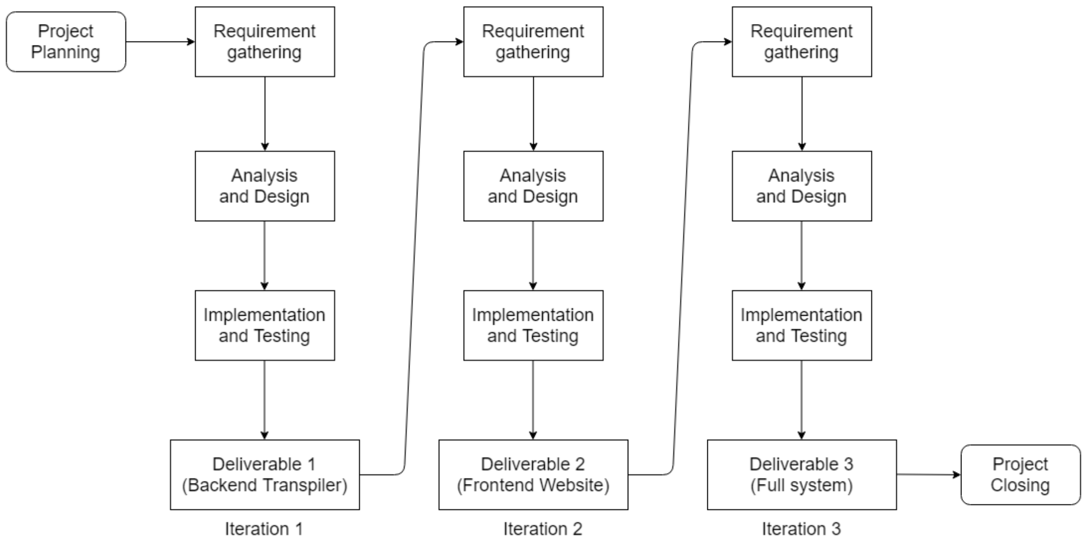


Figure 3-1 Proposed iterative and incremental model

3.2.1 Planning phase

3.2.1.1 Preliminary phase

Project planning is crucial for a project success because it sets expectations and understanding of the project that will be performed. In the planning phase, the first task to accomplish is to understand the background of the problem and identify the underlying issues of code conversion process. Few problems that are related to the code conversion process were found through relevant articles and journals. The first problem was the inefficiency and ineffectiveness of manual conversion. The second problem is the language specific architecture of most of the code conversion system and lastly, the error prone code conversion system.

After the problems are identified, few objectives were determined to provide a direction for the project so the main goal of the project could be achieved. The first objective of the proposed project is to identify the issues and practice of current code conversion process. The second objective is to develop a web-based transpiler that is universally compatible with mainstream programming languages. The third objective is to achieve code conversion accuracy score of 90% for the proposed source to source converter. These objectives were defined to achieve the main goal of the project, that is to provide a code conversion system that is universally compatible to improve code conversion process.

3.2.1.2 Requirements gathering and elicitation

Planning phase also includes information gathering process to define the requirements of the project. The purpose of the information gathering process is to investigate the approaches to carry out code conversion process, to analyze the user interface design and features provided by other relevant systems and to study about the intermediate language that is used to create an intermediate representation for the system. By gathering the information needed, requirements of the system can be outlined. Other than that, literature review will be conducted on past research papers to gather information regarding the best practices of code conversion process, and the issues and concerns that might affect the project.

3.2.1.3 Project scheduling

After the requirements of the system are gathered, the scope of the project can be defined to showcase all the necessary activities and tasks that need to be implemented. The project scope will describe all the work that needs to be done to achieve the project goal. A work breakdown structure is prepared to record all the project scope in an organized manner. The tasks can be broken down into work packages to distribute the tasks into different categories. Lastly, the work packages in the WBS will be scheduled using a Gantt chart so the project can be performed in a timely manner.

3.2.2 Analysis and Design phase

Analysis and design phase will provide the UML diagrams such as use case diagram, class diagram and system architecture to deliver visualization of the system design and workflow. The use case diagram was prepared to show the allowed user interaction with the system. The use cases will be explained in detail using use case description tables. Other than that, class diagram will also be prepared to showcase the relationship between different classes and the components of the back end transpiler system. Lastly, a prototype will be prepared to show the user interface of the webpage that work with the back-end server to provide a way for the user to interact with the system.

3.2.3 Implementation and Testing phase

The implementation and testing phase will be divided into three different iterations. The order of the phases will depend on the priority of the deliverable. Each iteration will consist of an implementation phase and a testing phase.

3.2.3.1 Iteration 1

The first iteration of the project implementation will be focusing on building the back-end service of the system. In other words, the deliverable of the first iteration would be the most crucial for the entire project. Since this project concern about the code conversion process the most, hence the transpiler need to be created before other modules. The transpiler is the main component of the proposed project. It will take the longest to finish because the system complexity is very high. Unit testing and integration testing will be performed on the modules to eliminate the bugs in the software codes as early as possible.

3.2.3.2 Iteration 2

The second iteration of the project will focus on the front-end webpage for the system. The user interface is the second most important for the project because it provides a platform where the user can interact with the system. After the completion of the webpage, user acceptance testing can be performed to analyze the user interaction with the system so changes can be made depending on the performance of the users.

3.2.3.3 Iteration 3

The final iteration of the project implementation process will focus on connecting the back-end service to the front-end website. Upon the completion of the integration of the back end transpiler and the front-end webpage, system testing can be performed to test the connectivity of the front-end and the back-end system.

3.2.4 Project Closing

After the implementation and testing of the system, documentation can be prepared for the project. The document shall include the lesson learnt throughout the project as well as the changes made during the implementation phase. The project would be considered as completed upon the achievement of project goals.

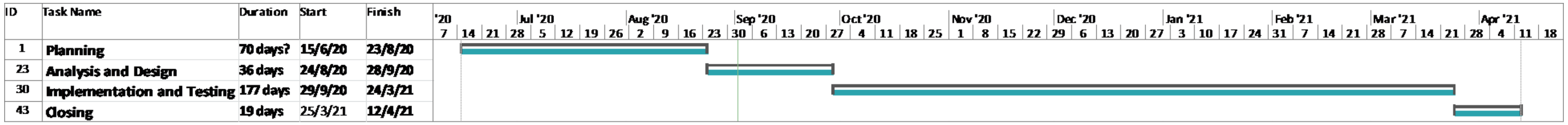
3.3 Work Breakdown Structure

1. Planning
    - 1.1. Study background of the problem
    - 1.2. Define problem statements
    - 1.3. Formulate project objectives
    - 1.4. Propose project solution
    - 1.5. Define project scope
        - 1.5.1. Identify transpiler modules
        - 1.5.2. Identify supported conversion Structure
        - 1.5.3. Identify web page features
        - 1.5.4. Identify uncovered scope
        - 1.5.5. Identify technologies and tools used
    - 1.6. Literature review
        - 1.6.1. Review similar system and past work
        - 1.6.2. Understand the concept of a transpiler
        - 1.6.3. Identify potential issue in code conversion process
        - 1.6.4. Determine project methodologies to be used
    - 1.7. Define system specification
        - 1.7.1. Define lexer dictionary
        - 1.7.2. Define AST structure
    - 1.8. Schedule project timeline
        - 1.8.1. Create WBS
        - 1.8.2. Create Gantt chart
2. Analysis and Design
    - 2.1. Create UML Diagrams
        - 2.1.1. Design Use Case Diagram
        - 2.1.2. Prepare Use Case Description
        - 2.1.3. Design Class Diagram
        - 2.1.4. Design level architecture diagram
    - 2.2. Develop Prototype
3. Implementation and Testing
    - 3.1. Phase 1 (Create back-end function)
        - 3.1.1. Create Lexer module

            - 3.1.2. Create Parser module
            - 3.1.3. Create Code generator module
            - 3.1.4. Carry out unit testing
            - 3.1.5. Carry out integration testing
        - 3.2. Phase 2 (Create front-end webpage)
            - 3.2.1. Design webpage
            - 3.2.2. Publish webpage
        - 3.3. Phase 3 (Implement entire software)
            - 3.3.1. Connect front-end and back-end
            - 3.3.2. Carry out system testing

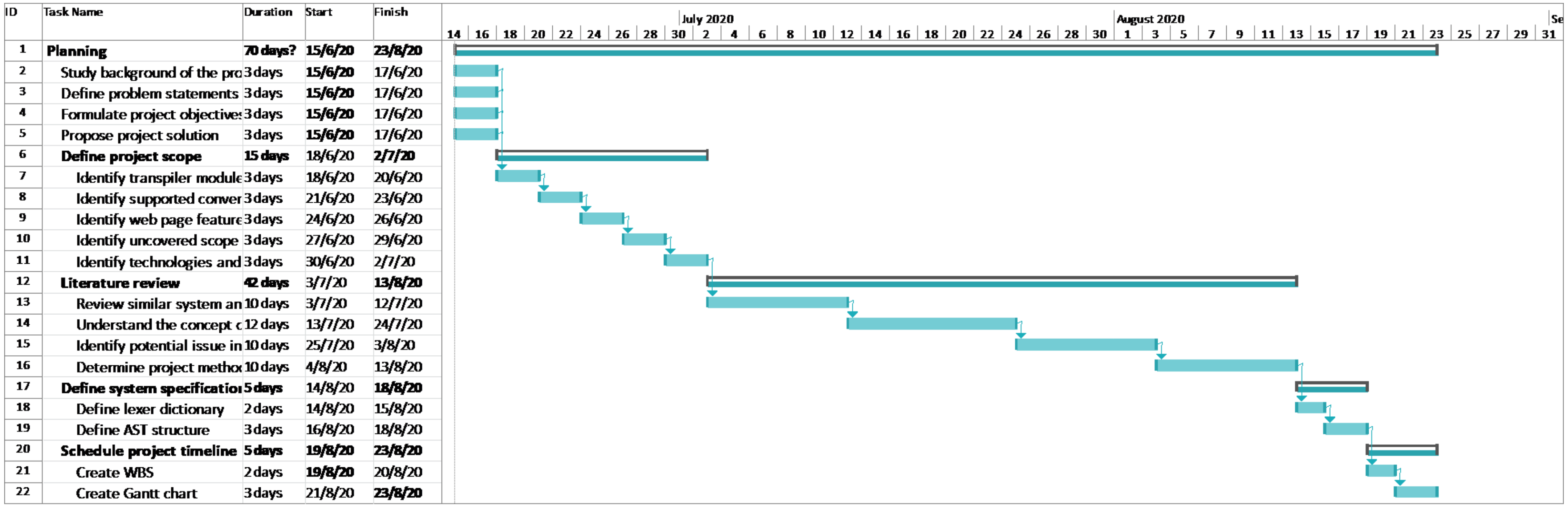
4. Closing
    - 4.1. Finalize the documentation of the system
    - 4.2. Prepare presentation slides

3.3.1 Gantt Chart



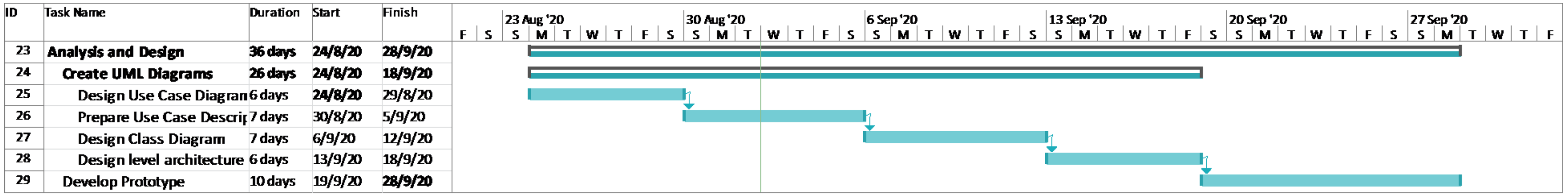
ID	Task Name	Duration	Start	Finish
1	**Planning**	**70 days?**	**15/6/20**	**23/8/20**
23	**Analysis and Design**	**36 days**	**24/8/20**	**28/9/20**
30	**Implementation and Testing**	**177 days**	**29/9/20**	**24/3/21**
43	**Closing**	**19 days**	25/3/21	**12/4/21**

Figure 3-2 Overview of the project schedule



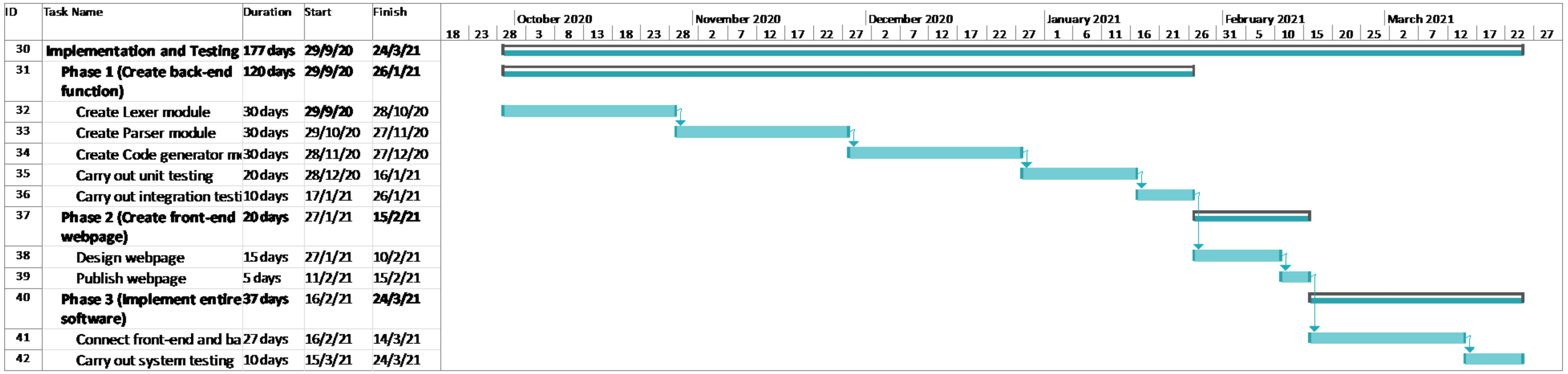
ID	Task Name	Duration	Start	Finish
1	**Planning**	**70 days?**	**15/6/20**	**23/8/20**
2	Study background of the pro	3 days	**15/6/20**	17/6/20
3	Define problem statements	3 days	**15/6/20**	17/6/20
4	Formulate project objectives	3 days	**15/6/20**	17/6/20
5	Propose project solution	3 days	**15/6/20**	17/6/20
6	**Define project scope**	**15 days**	18/6/20	**2/7/20**
7	Identify transpiler module	3 days	18/6/20	20/6/20
8	Identify supported conver	3 days	21/6/20	23/6/20
9	Identify web page feature	3 days	24/6/20	26/6/20
10	Identify uncovered scope	3 days	27/6/20	29/6/20
11	Identify technologies and	3 days	30/6/20	2/7/20
12	**Literature review**	**42 days**	3/7/20	**13/8/20**
13	Review similar system an	10 days	3/7/20	12/7/20
14	Understand the concept c	12 days	13/7/20	24/7/20
15	Identify potential issue in	10 days	25/7/20	3/8/20
16	Determine project metho	10 days	4/8/20	13/8/20
17	**Define system specificatio**	**5 days**	14/8/20	**18/8/20**
18	Define lexer dictionary	2 days	14/8/20	15/8/20
19	Define AST structure	3 days	16/8/20	18/8/20
20	**Schedule project timeline**	**5 days**	**19/8/20**	**23/8/20**
21	Create WBS	2 days	**19/8/20**	20/8/20
22	Create Gantt chart	3 days	21/8/20	**23/8/20**

Figure 3-3 Planning phase



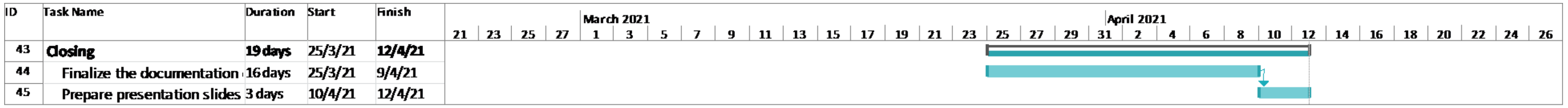
ID	Task Name	Duration	Start	Finish
23	**Analysis and Design**	**36 days**	**24/8/20**	**28/9/20**
24	**Create UML Diagrams**	**26 days**	**24/8/20**	**18/9/20**
25	Design Use Case Diagran	6 days	24/8/20	29/8/20
26	Prepare Use Case Descrip	7 days	30/8/20	5/9/20
27	Design Class Diagram	7 days	6/9/20	12/9/20
28	Design level architecture	6 days	13/9/20	18/9/20
29	Develop Prototype	10 days	19/9/20	28/9/20

Figure 3-4 Analysis and Design phase



ID	Task Name	Duration	Start	Finish
30	**Implementation and Testing**	**177 days**	**29/9/20**	**24/3/21**
31	**Phase 1 (Create back-end function)**	**120 days**	**29/9/20**	**26/1/21**
32	Create Lexer module	30 days	29/9/20	28/10/20
33	Create Parser module	30 days	29/10/20	27/11/20
34	Create Code generator m	30 days	28/11/20	27/12/20
35	Carry out unit testing	20 days	28/12/20	16/1/21
36	Carry out integration testi	10 days	17/1/21	26/1/21
37	**Phase 2 (Create front-end webpage)**	**20 days**	**27/1/21**	**15/2/21**
38	Design webpage	15 days	27/1/21	10/2/21
39	Publish webpage	5 days	11/2/21	15/2/21
40	**Phase 3 (Implement entire software)**	**37 days**	**16/2/21**	**24/3/21**
41	Connect front-end and ba	27 days	16/2/21	14/3/21
42	Carry out system testing	10 days	15/3/21	24/3/21

Figure 3-5 Implementation and Testing phase



ID	Task Name	Duration	Start	Finish
43	**Closing**	**19 days**	**25/3/21**	**12/4/21**
44	Finalize the documentation	16 days	25/3/21	9/4/21
45	Prepare presentation slides	3 days	10/4/21	12/4/21

Figure 3-6 Closing phase

3.4 Development tools and technologies

This section defines the development tools and technologies that will used in the project development.

3.4.1 Visual Studio Code

Visual Studio Code will be used as the main code editor for this project because it has a lot of features that could improve programming experience such as syntax highlighting and auto indentation. Since the proposed project involves intensive usage JavaScript and JSON files, this code editor is suitable for the project because it has support for hundreds of programming languages. Other than that, it also supports open-source plug-ins or extension to ease the coding process.

3.4.2 Git and GitHub

Git is a popular distributed version control system that can provide convenience to the developer in managing software folders meanwhile GitHub is a cloud-based repository to store project files. These tools are important for the project because it allows the developers to track the changes made to the program codes and revert the project back to previous version.

3.4.3 AxureRP 9

AxureRP is a prototyping tool to create high-fidelity prototype for software project. It is very convenient because the prototyping process does not involve any coding and uses drag-and-drop concept to design the prototype for the project. AxureRP 9 will be used to showcase the initial design of the front-end webpage.

3.4.4 React

React is a JavaScript oriented library that contains components for user interface development in the front-end system. React framework will be used in the project to implement the front-end webpage because it uses virtual DOM which promotes reuse of components in the project.

3.4.5 Node.js

Node.js is an open-sourced JavaScript-based server environment. It will be used as the backend server of the system which will communicate with the front-end webpage. The benefit of using node.js is that it allows third party packages or modules to be integrated to the project. Another reason to use Node.js is because time could be saved from learning other programming languages as the proposed project is mainly based on JavaScript programming language.

3.4.6 Jest

Jest is a testing framework maintained by Facebook that is specifically built for JavaScript. It is a popular testing framework for unit testing and integration testing for all types of project which includes React and Node.js. The testing framework will be implemented to carry out testing for the developed codes.

3.5 Summary

In short, this project will adopt iterative incremental development model which is divided into three phases. Other than that, the entire project duration will take up 302 days which includes public holidays and weekends.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

This chapter will describe the initial specification for this project which includes the requirements specification and the system design for both front-end development and back-end development. In addition, UML diagrams were modelled to allow visualization of the entire system process and workflow.

4.2 Requirements Specification

This section will list out all the functional requirements and non-functional requirements that would be implemented in the system. The user stated in the requirements is referring to the user who wants to use the system for code conversion or code compilation.

4.2.1 Functional Requirement

i. The system shall allow user to import source code file from local computer.
ii. The system shall allow user to convert Java code to C# code and vice versa.
iii. The system shall be able to compile Java code and C# code.
iv. The system shall allow user to modify the website’s theme.
v. The system shall allow user to export converted file with programming language specific file extension.
vi. The system shall prepare integrated code editor text area for user to enter programming codes.
vii. The system shall allow user to choose programming languages to be converted.
viii. The backend server must be able to handle multiple conversion process simultaneously.
ix. The system must display converted programming codes to the user upon completion of code conversion process.
x. The system must display the compilation result to the user.

4.2.2 Non-Functional Requirement

i. Usability
   a. The web application shall be designed to accommodate different screen sizes.
   b. The web application shall be easy to learn and intuitive.

ii. Performance
   a. The web page shall be able to be loaded within 3 seconds.
   b. The system shall handle multiple concurrent requests without causing a server crash.
   c. The system shall be able to perform operation asynchronously and output results within 10 seconds.

iii. Availability
   a. The web application shall be available to users at all time with the condition that they have access to Internet.

4.3 Use Case

This section will describe the set of actions that can be performed by the user. Since the project is mainly focusing on the code conversion process, hence there are only two simple use cases that can be performed by the user. The description of the use case will define the specific workflow for each use case.

4.3.1 Use Case Diagram

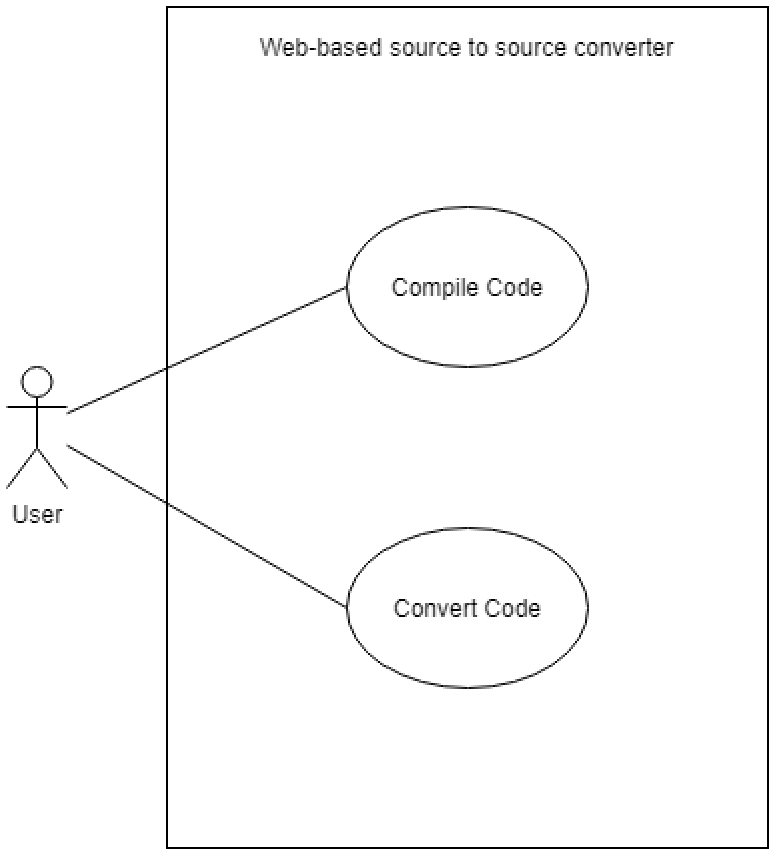


Figure 4-1 Use Case Diagram

4.3.2 Use Case Description

Use Case Name: Convert code		ID: UC01	Priority: High
Actor	User	Type: Detail, Essential	
Brief Description	This use case describes how the users use the source code converter to convert program codes.		
Trigger	The system user wants to convert program code into different programming language code.		
Relationships	-		
Flow of events	**<u>Normal Event Flow</u>** 1. User navigates to main page. 2. User select programming language to convert. 3. User input the program codes by pasting the program codes into the textbox or by uploading the program codes file. 4. User clicks on "Convert" button. 5. The system converts the program code and return the results to the user. If no codes are found, perform sub-flow 5.1. **<u>Alternative Event Flow</u>** 5.1 The system sends error message.		

Table 4-1 Use Case Description - Convert code

Use Case Name: Compile code		ID: UC02	Priority: Low
Actor	User	Type: Detail, Essential	
Brief Description	This use case describes how the users can compile program codes using the system.		
Trigger	The system user wants to compile the source program codes or the translated program codes.		
Relationships	-		
Flow of events	**Normal Event Flow** 1. User navigates to main page. 2. User select programming language to compile. 3. User input the program codes by pasting the program codes into the textbox or by uploading the program codes file. 4. User clicks on “Compile” button. 5. The system compiles the program code and return the results to the user. If no codes are found, perform sub-flow 5.1. **Alternative Event Flow** 5.1 The system sends error message.		

Table 4-2 Use Case Description - Compile code

4.4 Class Diagram

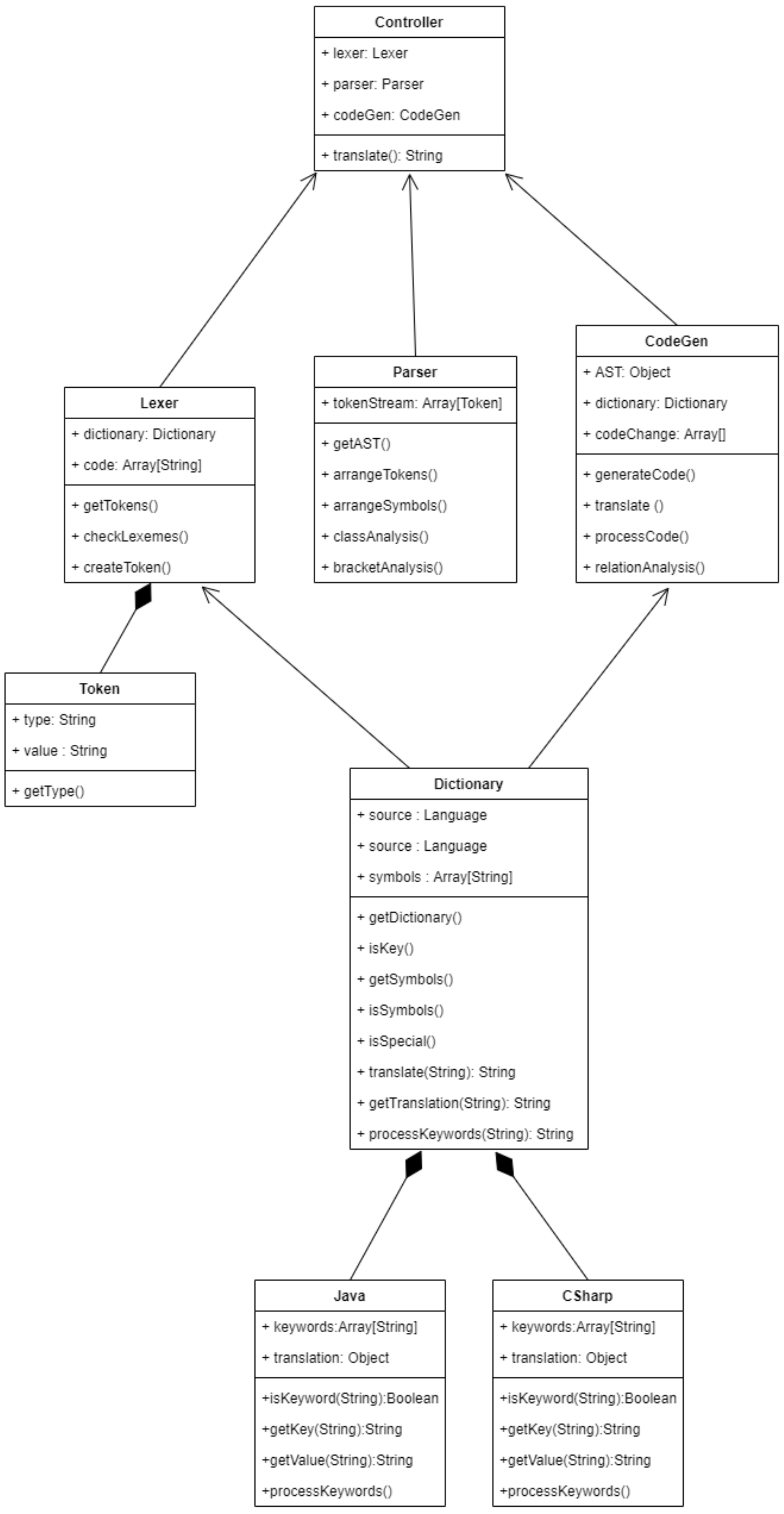


Figure 4-2 Class Diagram

4.5 High-level architecture

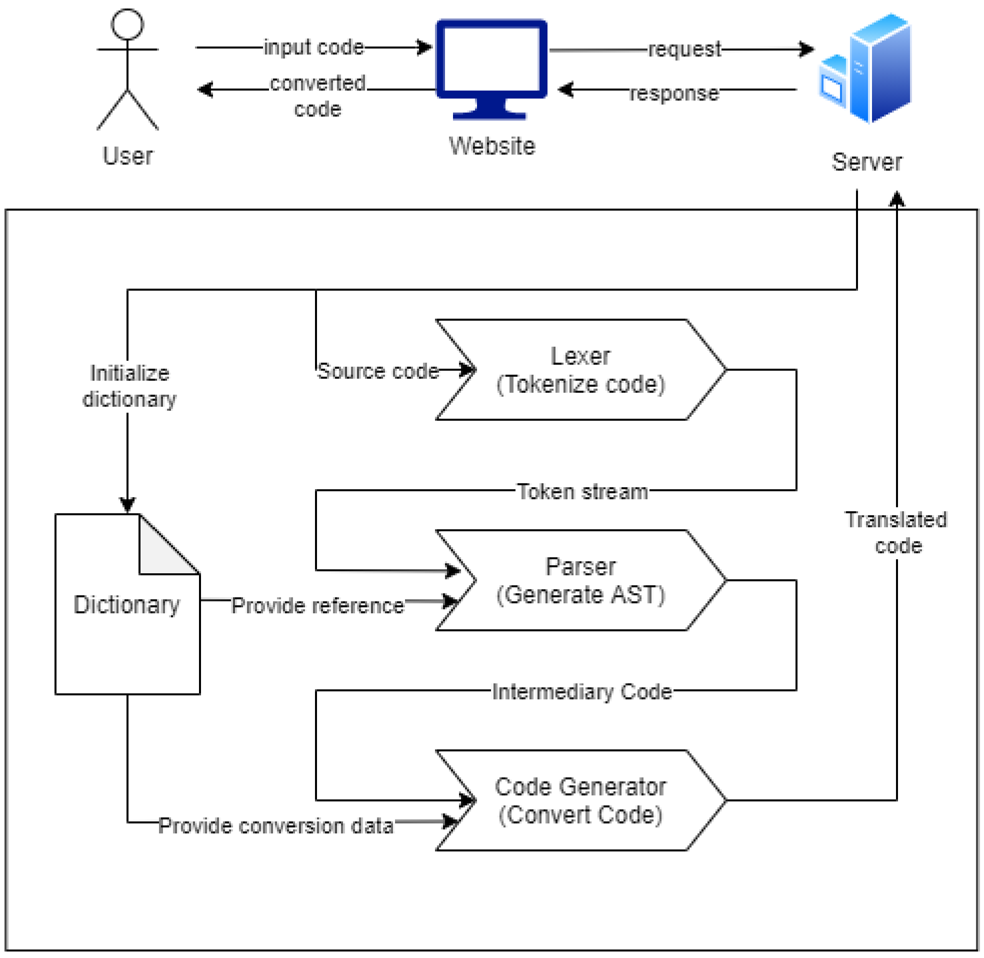


Figure 4-3 High level architecture

4.6 System specification

This section shall describe the back-end system specification. This section contains two sub-section which will describe the grammar dictionary for the lexer to conduct tokenization of the program codes and the structure of the abstract syntax representation after parsing process.

4.6.1 Lexer

Token type	**Regex rule**	**Example triggers**
Variable	[_a-z]([_a-zA-Z0-9])*	_variable, variable_1
Operator	[+\-*/%<>=!&|]	<=, &&, ||
Class	[A-Z][a-zA-Z]*	Integer, ArrayList
StringLiteral	["].*?["]	“Hello World!”
NumLiteral	\d.[0-9]*	12345

Table 4-3 Lexical grammar

Token type: ReservedKeyword			
Shared Keyword	• abstract • break • byte • case • catch • char • class • continue • default • double • else • enum	• false • finally • float • for • if • int • interface • long • new • null • override • private	• protected • public • return • short • static • switch • this • throw • true • try • void • while
Java specific	• boolean • extends • final • implements	• import • instanceof • package • super	
C# specific	• bool • const • foreach • in • is • namespace	• object • readonly • ref • sizeof • struct • typeof	

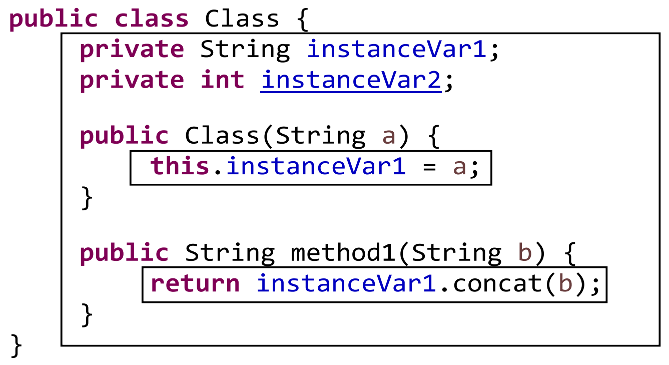
Table 4-4 System reserved keywords

4.6.2 Parser

Refer to appendix A for complete parsing example.

4.6.2.1 JSON structure description

Title	Description	Example
access	Describes the access modifier of the block or variables	public, private, protected
type	Describes the type of the block	class, method, function, variable, collection
kind	Describes the data type of class or variables, can be used for generic programming structure	string, int, <T>
name	Describes the identifier for the block	obj1, instanceVar1, employee_name

body	Describes the body of the block	 `public class Class {`
		`    private String instanceVar1;`
		`    private int instanceVar2;`
		
		`    public Class(String a) {`
		`        this.instanceVar1 = a;`
		`    }`
		
		`    public String method1(String b) {`
		`        return instanceVar1.concat(b);`
		`    }`
		`}`
		The box represents a block

content	Describes the value of the variable	`this.instanceVar1 = a;` Refer to appendix A(2), line 37-39
arguments	Describes the value passed to function calls	`return instanceVar1.concat(b);` Refer to appendix A(2), line 65-67
additional	Describes special operations from object or string	`return instanceVar1.concat(b);` Refer to appendix A(2), line 63-68

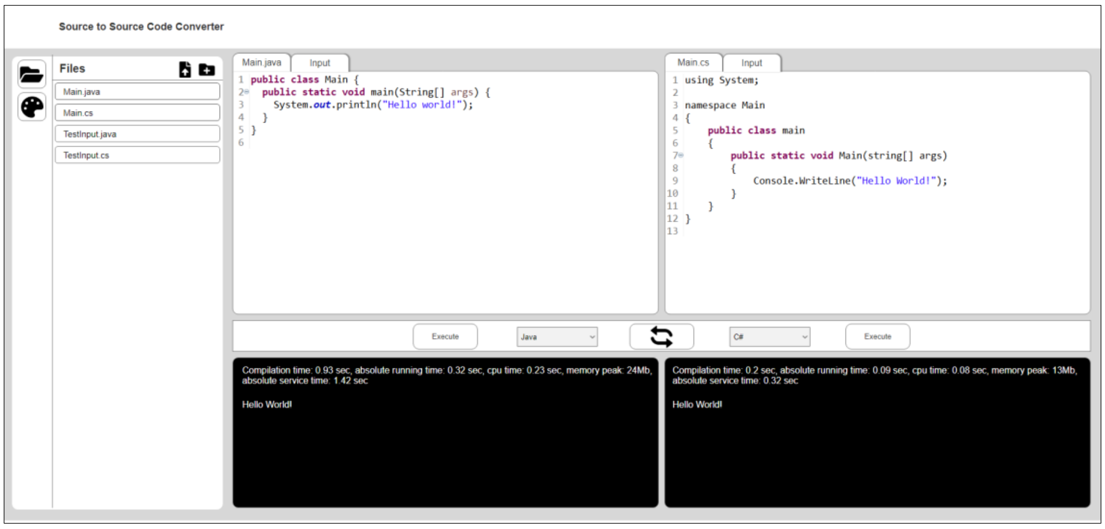
parameter	Describes the header parameter of a method	`public Class(String a) {`
		`    this.instanceVar1 = a;`
		`}`
		
		Refer to appendix A(2), line 24-30

return	Describes the return value of method	`public String method1(String b) {`
		`    return instanceVar1.concat(b);`
		`}`
		
		Refer to appendix A(2), line 71

Table 4-5 Components in abstract syntax representation

4.7 User Interface Design

The front-end system only contains one main page that allows user to interact with the system.



```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

```csharp
using System;

namespace Main
{
    public class main
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```
Compilation time: 0.93 sec, absolute running time: 0.32 sec, cpu time: 0.23 sec, memory peak: 24Mb,
absolute service time: 1.42 sec

Hello World!
```

```
Compilation time: 0.2 sec, absolute running time: 0.09 sec, cpu time: 0.08 sec, memory peak: 13Mb,
absolute service time: 0.32 sec

Hello World!
```

Figure 4-4 Main page

4.8 Summary

In short, this chapter describes the functional and non-functional requirements for the proposed system. Besides that, the specifications of the back-end system are also defined to standardize the structure of the abstract syntax representation and the tokenization process. Last but not least, UML diagrams are modelled to visualize the system structure and workflow.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Introduction

The entire development phase of the project was divided into three main stages as described in Chapter 3. However, there are changes in the ordering of implementation due to the complexity of the backend processing system.

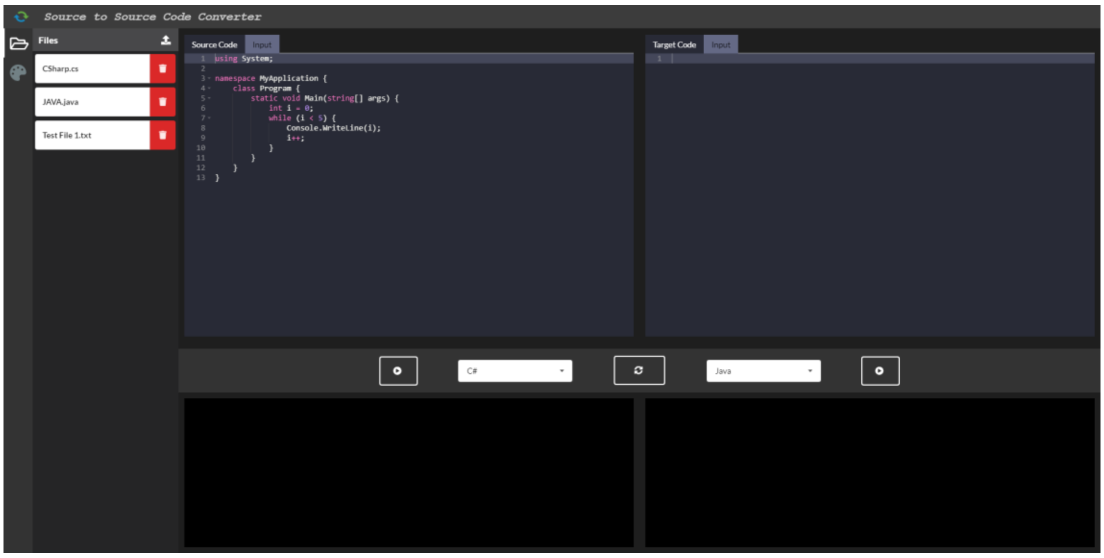
During the first stage in the development phase, a frontend website has been developed using React frontend library. The connectivity between the frontend website and basic structure of the backend system are configuring in the second stage. An automated testing framework was implemented to ensure that the project is tested after every code update. The core functionality of the system was implemented in the last stage of the implementation phase as it requires a lot of fine tuning and incremental updates.

5.2 First iteration phase

During the planning phase, the backend system was planned to be implemented first. However, implementation of the backend system will take up a lot of effort because it is the main focus of the entire project. Furthermore, a lot of incremental and iterative changes will need to be implemented for the improvement of the backend processing logic. Hence, the frontend website is developed before implementing the backend processing system.

5.2.1 Design of frontend website

The website was designed and developed based on the prototype defined in chapter 4. However, dark theme was used as the main colour palette for the website because the target audience of this website are the computer programmers who will look at computer screens for hours at a time. Kim et al. (2019) found out that dark mode will not only reduces visual fatigue for the users, but it will also improve usability of the website or content that they were browsing. React frontend library was used to create the website; it allows the website to update its appearance after state changes.

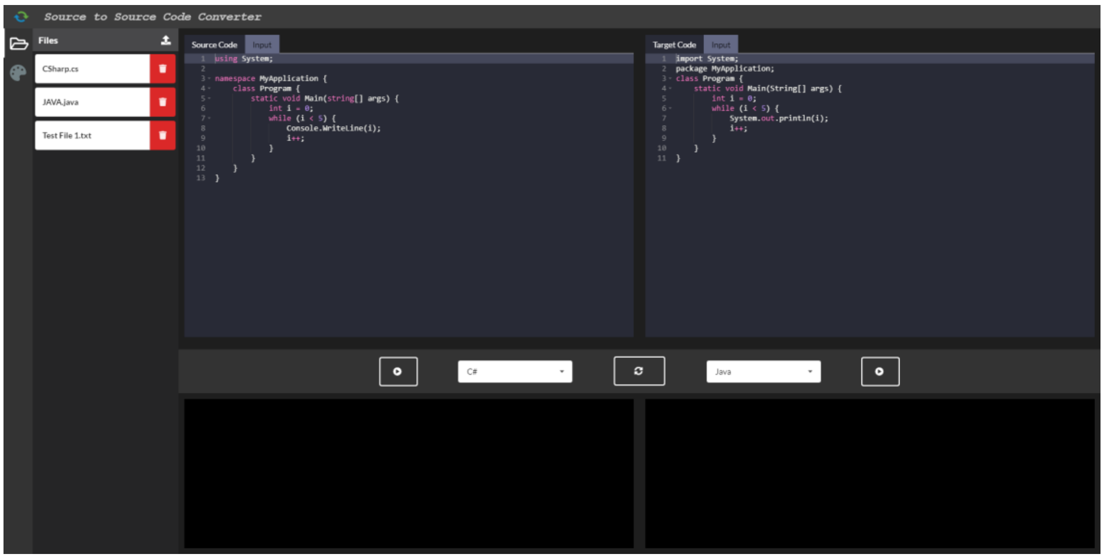


```
using System;

namespace MyApplication {
    class Program {
        static void Main(string[] args) {
            int i = 0;
            while (i < 5) {
                Console.WriteLine(i);
                i++;
            }
        }
    }
}
```

Figure 5-1 Source-to-source converter website

5.2.2 Features implemented for frontend website

There were few functionalities for the frontend website aforementioned in the earlier chapters. The users could use the system to perform code conversion, code compilation and configure the settings of the website. The user could enter the codes manually into the code editors of the website or select the input file from the left side file manager.

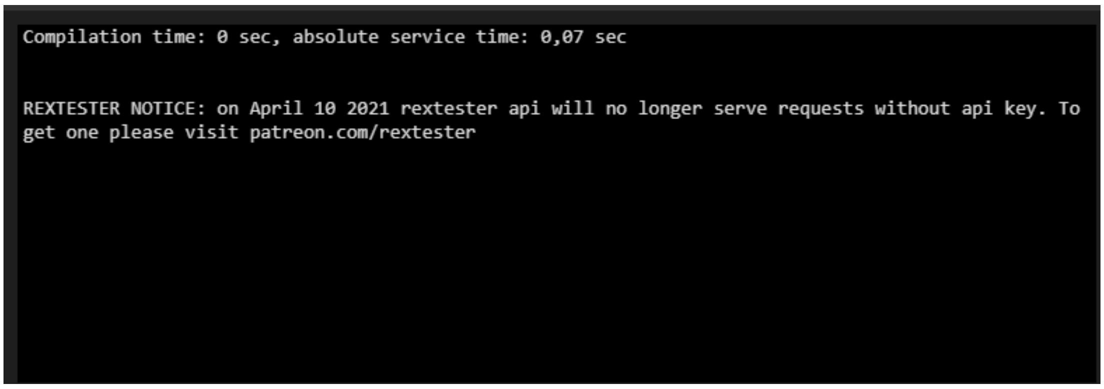


```
using System;

namespace MyApplication {
    class Program {
        static void Main(string[] args) {
            int i = 0;
            while (i < 5) {
                Console.WriteLine(i);
                i++;
            }
        }
    }
}
```

```
import System;
package MyApplication;
class Program {
    static void Main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println(i);
            i++;
        }
    }
}
```

Figure 5-2 Demonstration of code conversion

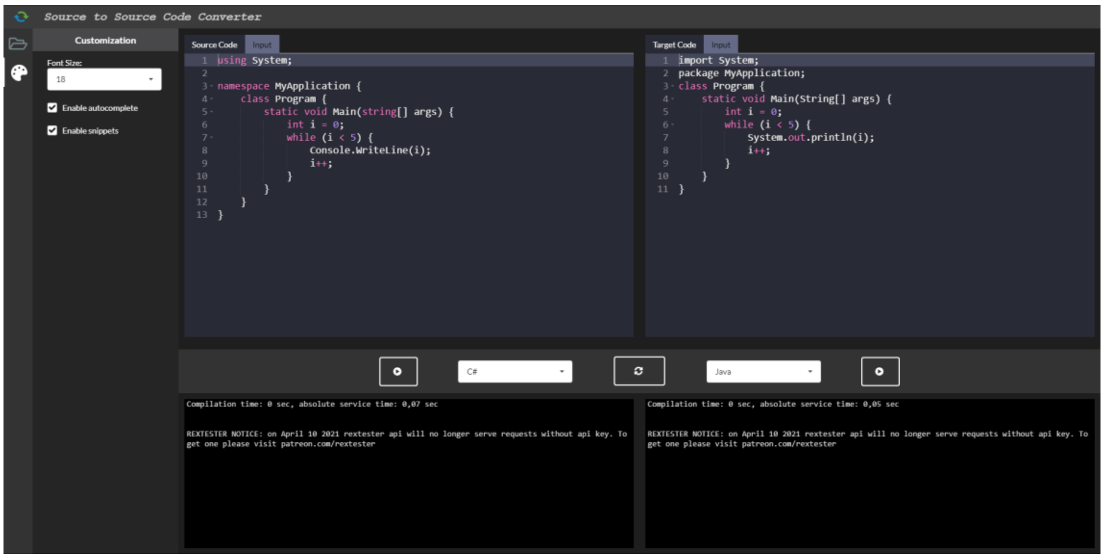
In the report, it was mentioned that the website could allow the users to compile the code based on their languages. However, the API supplier have stopped their free API from 10th April 2021 onwards. Due to time constraint to search for new code compiler API, the website will be unable to compile codes.



```
Compilation time: 0 sec, absolute service time: 0,07 sec


REXTESTER NOTICE: on April 10 2021 rextester api will no longer serve requests without api key. To
get one please visit patreon.com/rextester
```

Figure 5-3 Free code compiler API from REXTESTER

Other than that, the website also allows the user to customize the code editor settings using the options that were presented on the side bar. The user could customize the font size of the code editor, toggle autocomplete keyword setting, and toggle autocomplete snippet setting. React Ace is the code editor extension that were incorporated into the website to allow users enter codes. The extension is open-sourced and specifically made for React projects.



Font Size: 18

☑ Enable autocomplete

☑ Enable snippets

```
using System;

namespace MyApplication {
    class Program {
        static void Main(string[] args) {
            int i = 0;
            while (i < 5) {
                Console.WriteLine(i);
                i++;
            }
        }
    }
}
```

```
import System;
package MyApplication;
class Program {
    static void Main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println(i);
            i++;
        }
    }
}
```

C#

Java

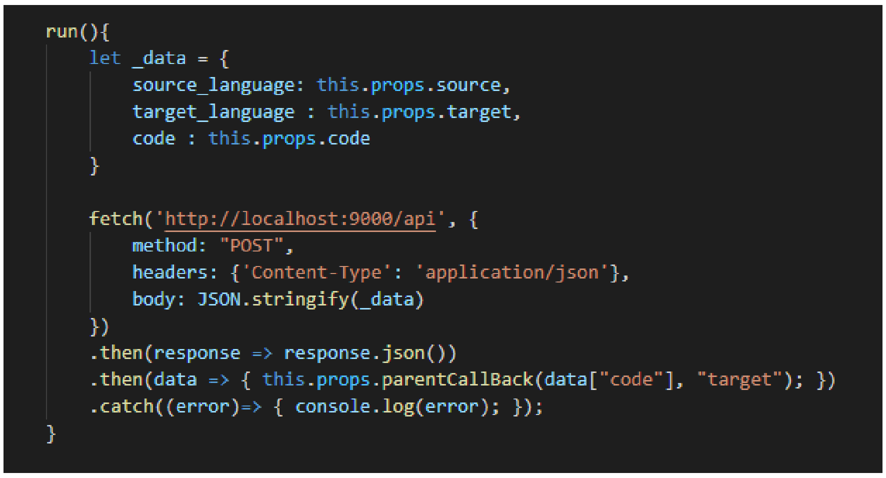
```
Compilation time: 0 sec, absolute service time: 0,07 sec


REXTESTER NOTICE: on April 10 2021 rextester api will no longer serve requests without api key. To
get one please visit patreon.com/rextester
```

```
Compilation time: 0 sec, absolute service time: 0,05 sec


REXTESTER NOTICE: on April 10 2021 rextester api will no longer serve requests without api key. To
get one please visit patreon.com/rextester
```

Figure 5-4 Code editor settings

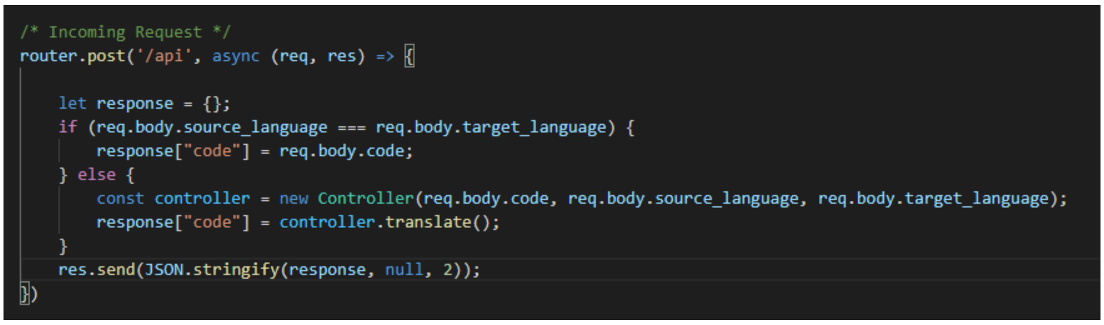
5.3 Second iteration phase

The second iteration phase in this project mainly focused on connecting the frontend and the backend of the system. The structure of the backend system was initiated, and a simple API were created using Axios at the backend JavaScript system. To connect the frontend website and the backend system, the frontend React code have consumed the backend API. The source language, target language and code were passed to the backend API in JSON format. The backend API will produce JSON replies.



```
run(){
    let _data = {
        source_language: this.props.source,
        target_language : this.props.target,
        code : this.props.code
    }

    fetch('http://localhost:9000/api', {
        method: "POST",
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(_data)
    })
    .then(response => response.json())
    .then(data => { this.props.parentCallBack(data["code"], "target"); })
    .catch((error)=> { console.log(error); });
}
```

Figure 5-5 Frontend API calls to backend



```
/* Incoming Request */
router.post('/api', async (req, res) => {

    let response = {};
    if (req.body.source_language === req.body.target_language) {
        response["code"] = req.body.code;
    } else {
        const controller = new Controller(req.body.code, req.body.source_language, req.body.target_language);
        response["code"] = controller.translate();
    }
    res.send(JSON.stringify(response, null, 2));
})
```

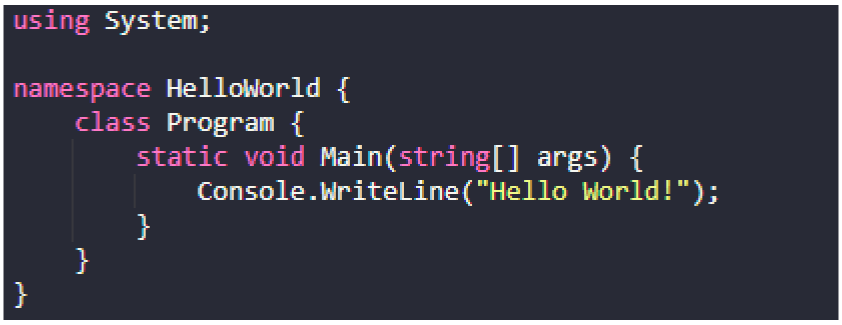
Figure 5-6 API set up at backend system

5.4 Third and subsequent iteration phase

The third and subsequent iteration phase will mainly focus on developing the backend processing system. The backend processing system was built based on the core concept, a transpiler. As mentioned in earlier chapters, a transpiler consists of two front end analysis modules that are the lexer and the parser. After performing analysis on the code, the intermediary product will be passed to a backend synthesis module that is the code generator. Each module has their purposes and output.

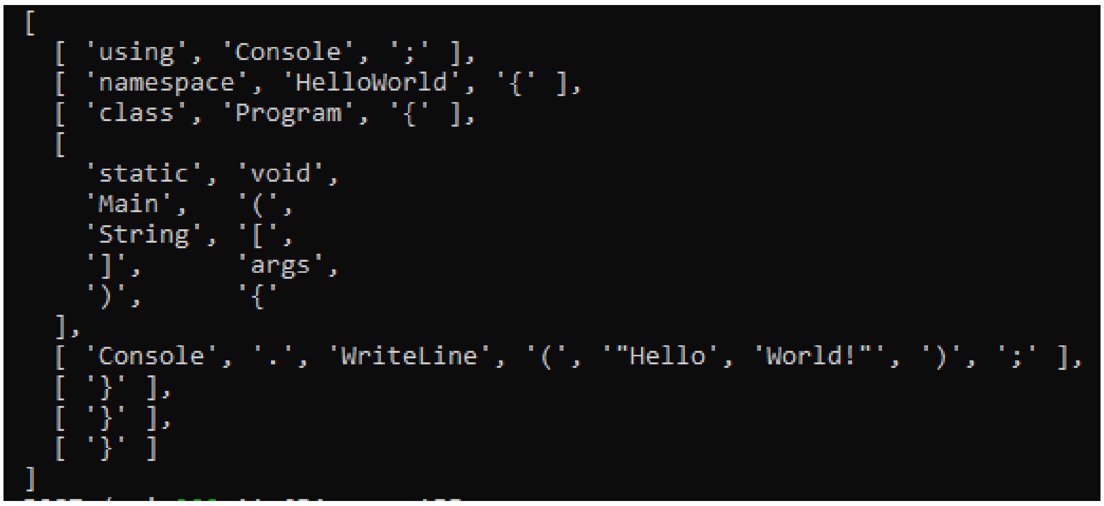
5.4.1 Lexer

The source code will first pass through a lexer. The lexer is responsible for splitting the source codes into lexemes, and then into tokens. The purpose of splitting the source code into literal chunks is to create convenience for the later stages of the code conversion process. The comments or string literal were undisturbed to maintain the formatting of the text as much as possible.



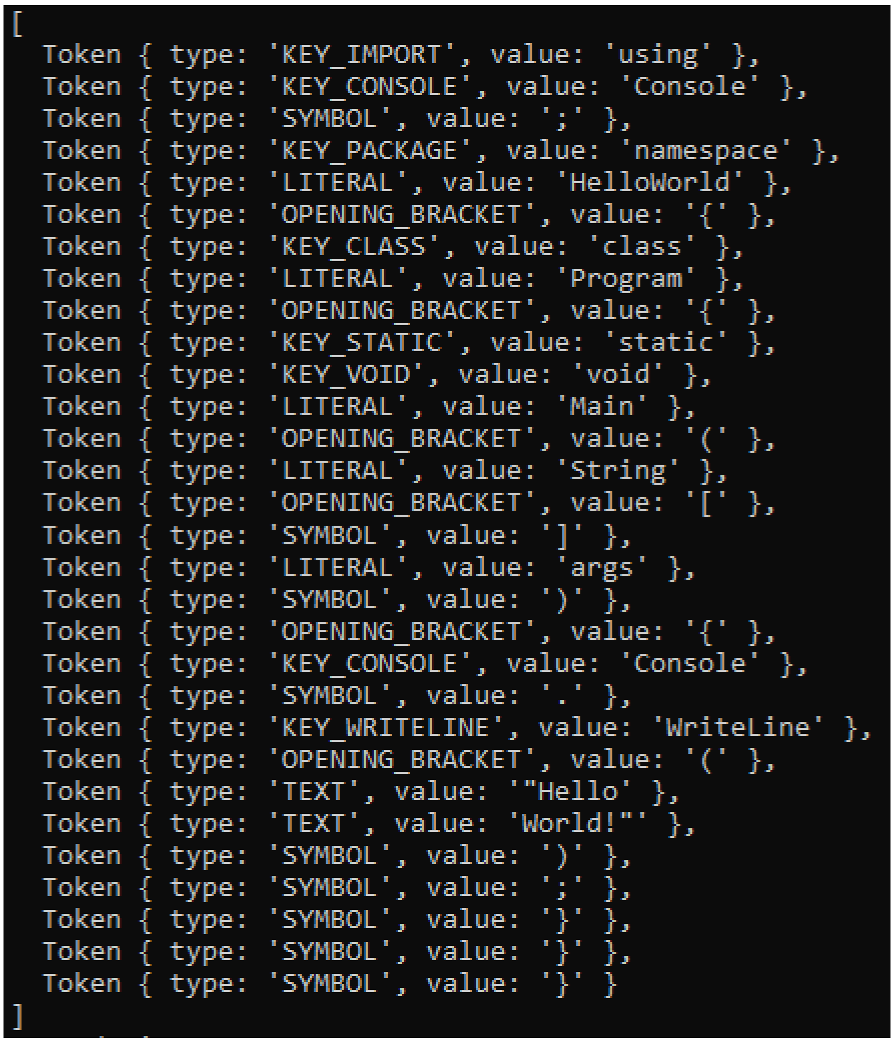
```
using System;

namespace HelloWorld {
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Figure 5-7 Example C# source code



```
[
  [ 'using', 'Console', ';' ],
  [ 'namespace', 'HelloWorld', '{' ],
  [ 'class', 'Program', '{' ],
  [
    'static', 'void',
    'Main',   '(',
    'String', '[',
    ']',      'args',
    ')',      '{'
  ],
  [ 'Console', '.', 'WriteLine', '(', '"Hello', 'World!"', ')', ';' ],
  [ '}' ],
  [ '}' ],
  [ '}' ]
]
```

Figure 5-8 Source code split into lexemes



```
[
  Token { type: 'KEY_IMPORT', value: 'using' },
  Token { type: 'KEY_CONSOLE', value: 'Console' },
  Token { type: 'SYMBOL', value: ';' },
  Token { type: 'KEY_PACKAGE', value: 'namespace' },
  Token { type: 'LITERAL', value: 'HelloWorld' },
  Token { type: 'OPENING_BRACKET', value: '{' },
  Token { type: 'KEY_CLASS', value: 'class' },
  Token { type: 'LITERAL', value: 'Program' },
  Token { type: 'OPENING_BRACKET', value: '{' },
  Token { type: 'KEY_STATIC', value: 'static' },
  Token { type: 'KEY_VOID', value: 'void' },
  Token { type: 'LITERAL', value: 'Main' },
  Token { type: 'OPENING_BRACKET', value: '(' },
  Token { type: 'LITERAL', value: 'String' },
  Token { type: 'OPENING_BRACKET', value: '[' },
  Token { type: 'SYMBOL', value: ']' },
  Token { type: 'LITERAL', value: 'args' },
  Token { type: 'SYMBOL', value: ')' },
  Token { type: 'OPENING_BRACKET', value: '{' },
  Token { type: 'KEY_CONSOLE', value: 'Console' },
  Token { type: 'SYMBOL', value: '.' },
  Token { type: 'KEY_WRITELINE', value: 'WriteLine' },
  Token { type: 'OPENING_BRACKET', value: '(' },
  Token { type: 'TEXT', value: '"Hello' },
  Token { type: 'TEXT', value: 'World!"' },
  Token { type: 'SYMBOL', value: ')' },
  Token { type: 'SYMBOL', value: ';' },
  Token { type: 'SYMBOL', value: '}' },
  Token { type: 'SYMBOL', value: '}' },
  Token { type: 'SYMBOL', value: '}' }
]
```

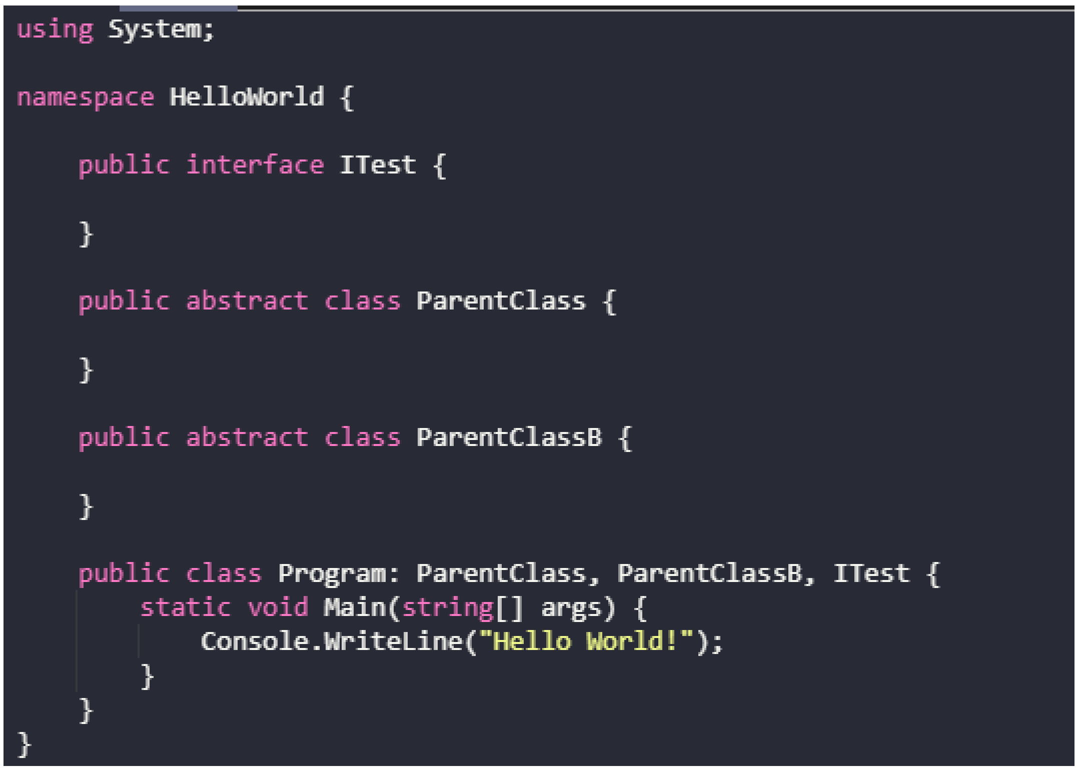
Figure 5-9 Lexemes were processed into tokens

Each lexeme was processed into a token. The token carries information such as the type of the token and the value of the token. If the system detects a keyword, an abstract keyword will be provided to the token. The abstract keyword is not specific to any language. In other words, it could be used universally with different languages. The vocabulary of the abstract language will increase as more programming language are added into the system.

After the source code is processed into a token stream as shown in Figure 5-9, it will be passed to the next analyser module, a Parser.

5.4.2 Parser

The parser will perform analysis on the token stream that were generated by the lexer. The two major analysis process that the parser will perform is structural analysis, object-oriented class analysis. The token stream will then be parsed into intermediary code which is not specific to any programming languages.

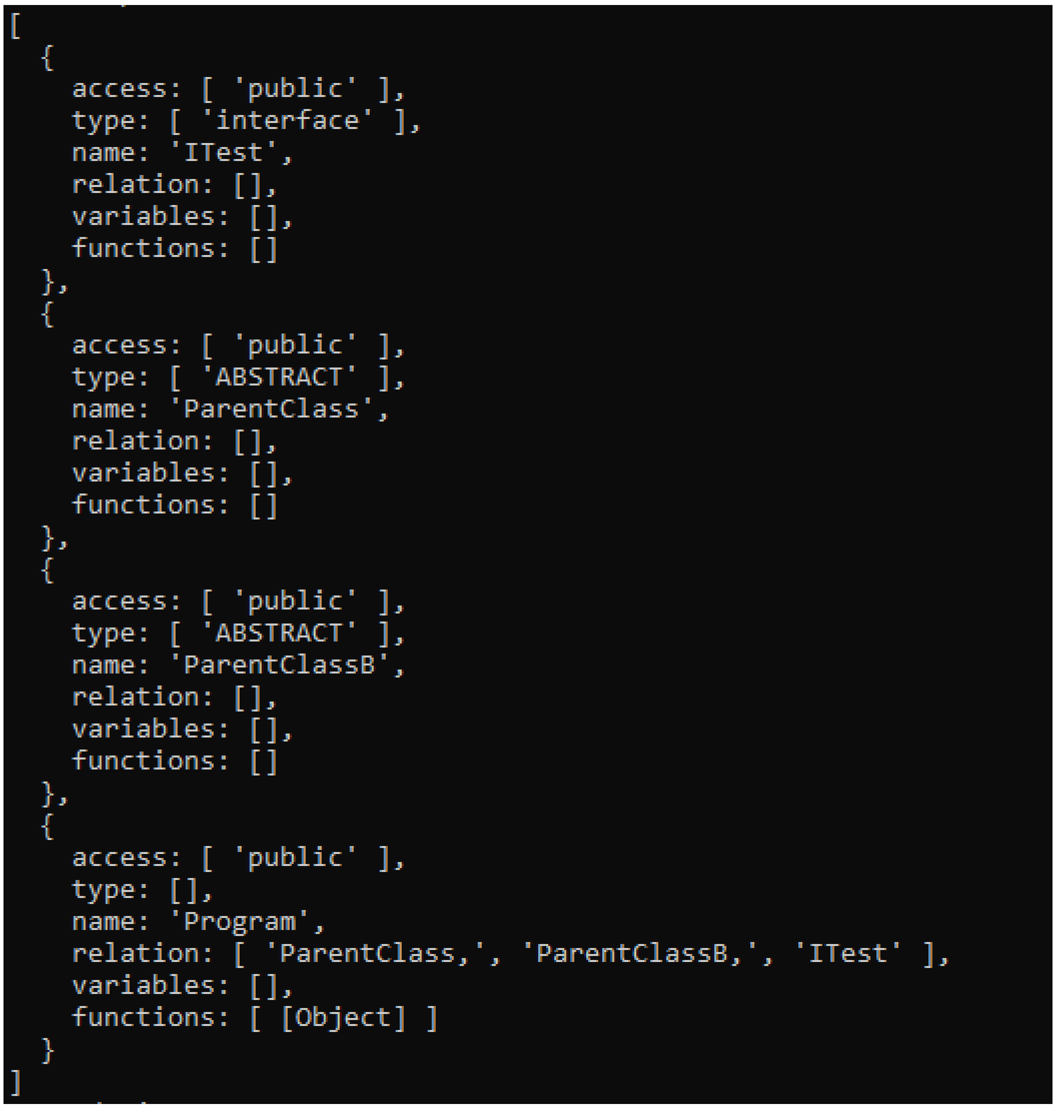


```
using System;

namespace HelloWorld {

    public interface ITest {

    }

    public abstract class ParentClass {

    }

    public abstract class ParentClassB {

    }

    public class Program: ParentClass, ParentClassB, ITest {
        static void Main(string[] args) {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Figure 5-10 Example C# code

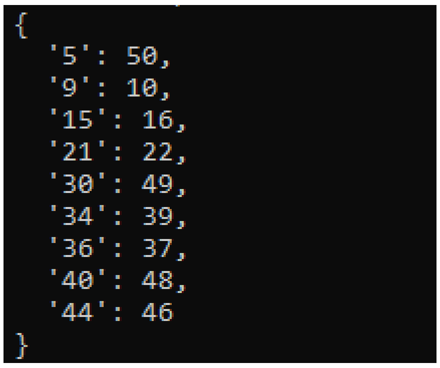
The codes that were shown in Figure 5-10 demonstrated object-oriented programming that were written in C# programming language. There four classes written in the codes includes an interface, two abstract classes and a normal class. The class analysis will find out all the classes in the code along with the details of the class which includes the access modifier, class type, class name, relationship between classes, instance variable declared in the class and the functions defined in the class.

On the other hand, a structural analysis has been carried out on the source code to find out the boundaries of the codes. The main search criteria for the structure are with brackets. This analysis technique is not suitable for programming languages that uses indentation and whitespace to identify code blocks such as Python. The structural analysis is useful for processing of complex codes in the later stage.



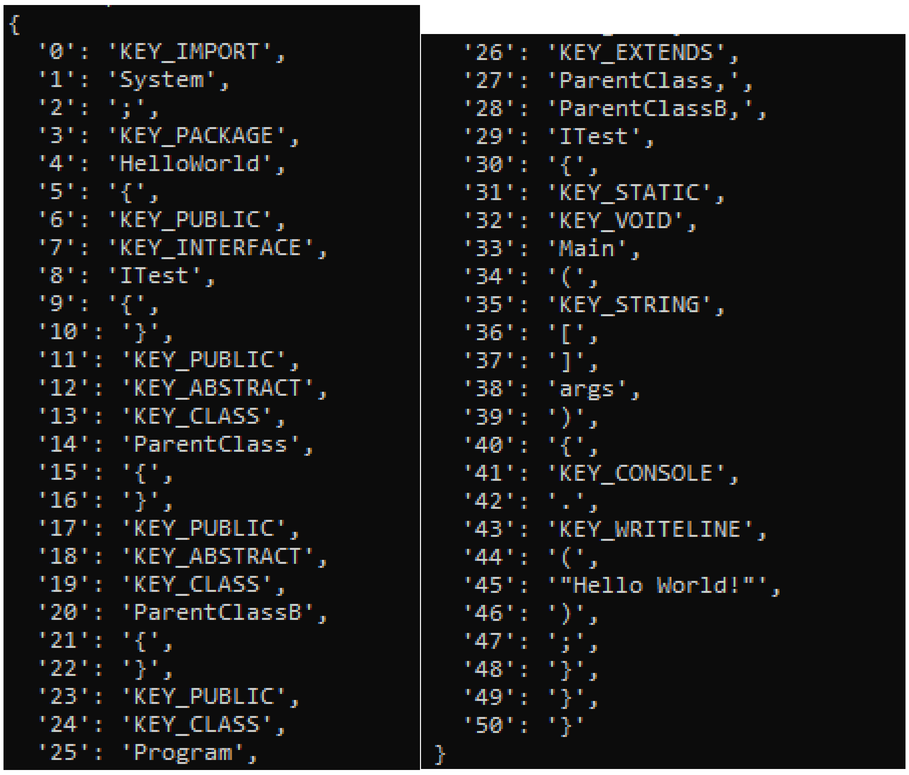
```
[
  {
    access: [ 'public' ],
    type: [ 'interface' ],
    name: 'ITest',
    relation: [],
    variables: [],
    functions: []
  },
  {
    access: [ 'public' ],
    type: [ 'ABSTRACT' ],
    name: 'ParentClass',
    relation: [],
    variables: [],
    functions: []
  },
  {
    access: [ 'public' ],
    type: [ 'ABSTRACT' ],
    name: 'ParentClassB',
    relation: [],
    variables: [],
    functions: []
  },
  {
    access: [ 'public' ],
    type: [],
    name: 'Program',
    relation: [ 'ParentClass,', 'ParentClassB,', 'ITest' ],
    variables: [],
    functions: [ [Object] ]
  }
]
```

Figure 5-11 Class analysis



```
{
  '5': 50,
  '9': 10,
  '15': 16,
  '21': 22,
  '30': 49,
  '34': 39,
  '36': 37,
  '40': 48,
  '44': 46
}
```

Figure 5-12 Structural analysis



```
{
  '0': 'KEY_IMPORT',
  '1': 'System',
  '2': ';',
  '3': 'KEY_PACKAGE',
  '4': 'HelloWorld',
  '5': '{',
  '6': 'KEY_PUBLIC',
  '7': 'KEY_INTERFACE',
  '8': 'ITest',
  '9': '{',
  '10': '}',
  '11': 'KEY_PUBLIC',
  '12': 'KEY_ABSTRACT',
  '13': 'KEY_CLASS',
  '14': 'ParentClass',
  '15': '{',
  '16': '}',
  '17': 'KEY_PUBLIC',
  '18': 'KEY_ABSTRACT',
  '19': 'KEY_CLASS',
  '20': 'ParentClassB',
  '21': '{',
  '22': '}',
  '23': 'KEY_PUBLIC',
  '24': 'KEY_CLASS',
  '25': 'Program',
  '26': 'KEY_EXTENDS',
  '27': 'ParentClass,',
  '28': 'ParentClassB,',
  '29': 'ITest',
  '30': '{',
  '31': 'KEY_STATIC',
  '32': 'KEY_VOID',
  '33': 'Main',
  '34': '(',
  '35': 'KEY_STRING',
  '36': '[',
  '37': ']',
  '38': 'args',
  '39': ')',
  '40': '{',
  '41': 'KEY_CONSOLE',
  '42': '.',
  '43': 'KEY_WRITELINE',
  '44': '(',
  '45': '"Hello World!"',
  '46': ')',
  '47': ';',
  '48': '}',
  '49': '}',
  '50': '}'
}
```

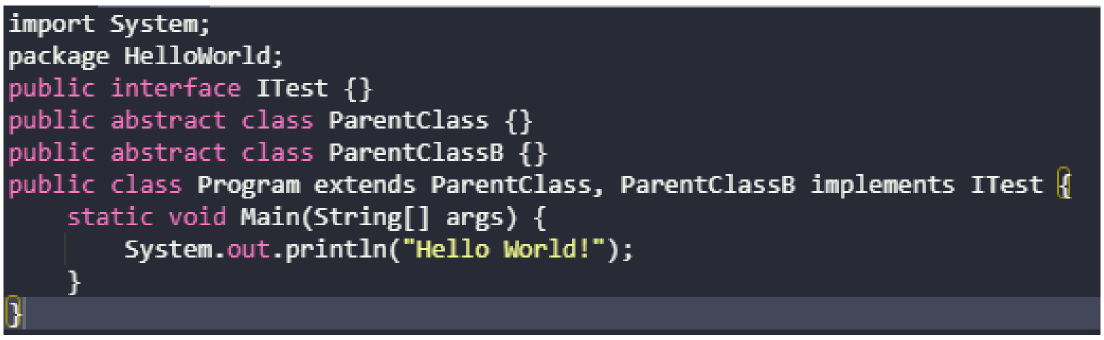
Figure 5-13 Intermediary code

The token stream is parsed into intermediary codes where abstract information of the codes are remained. The intermediary code does not store the "meaning" of each code, but only store the value of the codes. The text or string literal are pasted back together to maintain the original value of the source code. Finally, the class analysis and structural analysis result are bound together with the intermediary source code to form an Abstract Syntax Representation.

There were some discrepancies between the implementation phase and the planning phase. Originally, an Abstract Syntax Tree (AST) was supposed to be produced by the parser. However, the abstract syntax tree that were proposed in the earlier chapters are bias towards object-oriented programming languages as the structure of the AST revolves around classes and objects. After a few iterations of the project, a decision was made to change the structure of the parser so it could produce an intermediary code that were less bias towards any type of programming languages. A class analysis module was added to analyse the object-oriented languages meanwhile a structural analysis module was added to analyse the overall structure of the program to correctly identify the position of the block of codes.

5.4.3 Code Generator

After the front-end analysis phase by the lexer and parser, the abstract syntax representation that contains the intermediary codes, class analysis result and structural analysis result is passed to the backend synthesiser - code generator. The code generator is responsible to convert the intermediary codes into the target language chosen by the user with the help of the class analysis, structural analysis and dictionary. Some of the keywords might need special processing by the code generator.



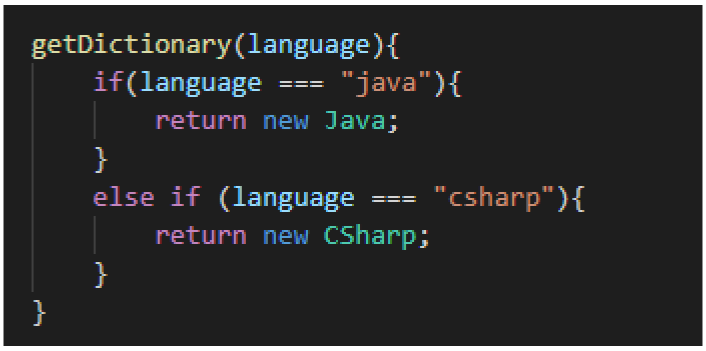
```
import System;
package HelloWorld;
public interface ITest {}
public abstract class ParentClass {}
public abstract class ParentClassB {}
public class Program extends ParentClass, ParentClassB implements ITest {
    static void Main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Figure 5-14 Result of conversion

The system successfully converts the C# code from figure 5-10 to Java code in figure 5-14. The system is able to detect the keywords that require special processing such as “namespace” keyword from C# programming language. Other than that, the system also successfully captures the relationship between different classes.

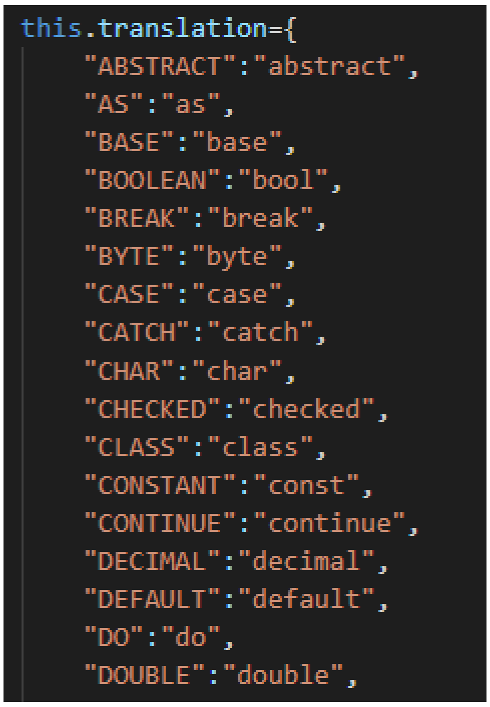
5.4.4 Dictionary & Language

The dictionary plays a crucial role in the system as it provides all the information about the conversion between the programming languages and the intermediary keyword. Moreover, the dictionary also contains the standard procedure when a special keyword that requires additional processing is found. The structure of the dictionary was repeatedly improved to accommodate more programming languages to be added in the future. Currently, the system only supports C# and Java programming languages.



```
getDictionary(language){
    if(language === "java"){
        return new Java;
    }
    else if (language === "csharp"){
        return new CSharp;
    }
}
```

Figure 5-15 Language packs are assigned dynamically



```
this.translation={
    "ABSTRACT":"abstract",
    "AS":"as",
    "BASE":"base",
    "BOOLEAN":"bool",
    "BREAK":"break",
    "BYTE":"byte",
    "CASE":"case",
    "CATCH":"catch",
    "CHAR":"char",
    "CHECKED":"checked",
    "CLASS":"class",
    "CONSTANT":"const",
    "CONTINUE":"continue",
    "DECIMAL":"decimal",
    "DEFAULT":"default",
    "DO":"do",
    "DOUBLE":"double",
```

Figure 5-16 Conversion rules

CHAPTER 6

SYSTEM TESTING

6.1 Introduction

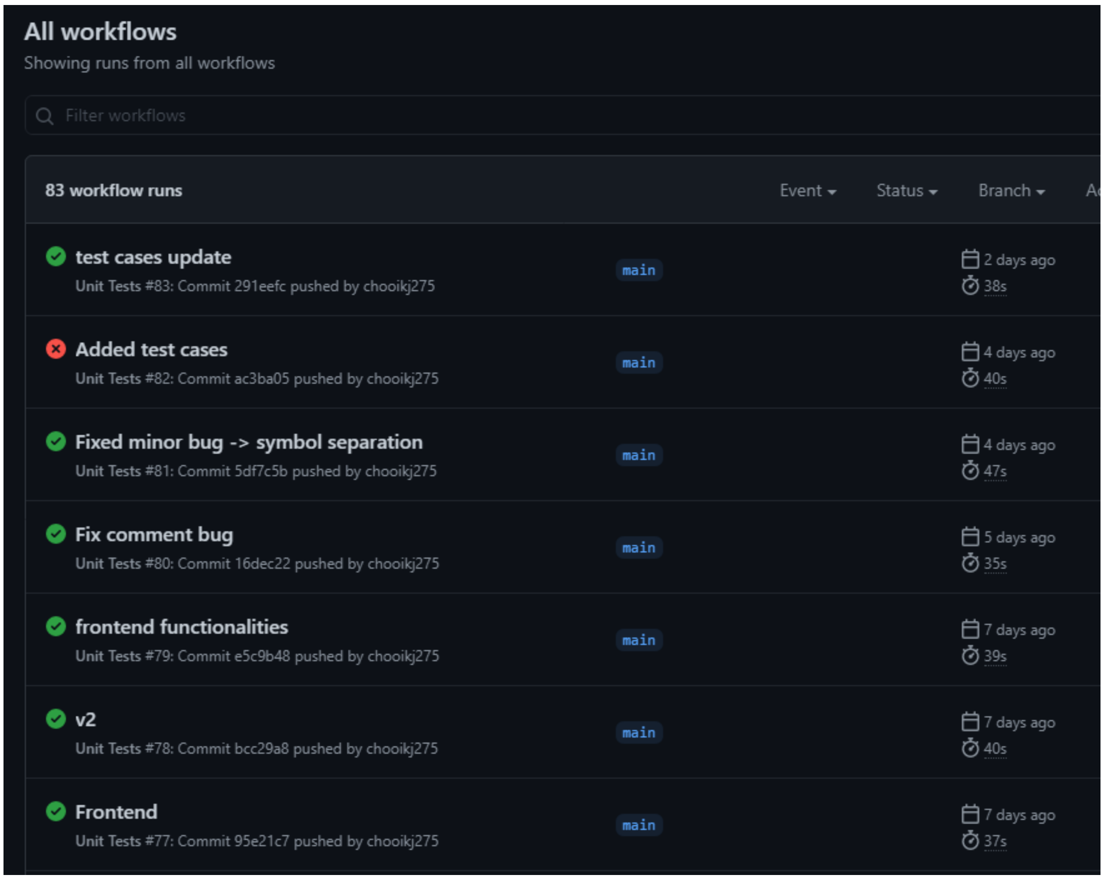
This chapter will describe the testing that have been executed after each implementation phase. A few types of testing method have been used in this project to identify bugs at the early phase to prevent software failure in the later stage of the development. The four testing methods that were carried out are unit testing, integration testing, UI test and user acceptance test.

Unit testing and integration testing were implemented using Jest which is a testing framework that were built specifically for JavaScript that works in different project such as Node.js and React that this project is using. The user interface of the website is tested manually to ensure that every components of the website are functioning as expected. Lastly, 30 volunteers are gathered to perform user acceptance test for the website.

6.2 Testing approach

Test-Driven Development (TDD) approach have been adopted in this project as the testing methodology. In this methodology, test code are written before the production code (Mäkinen and Münch, 2014). The three main steps of this approach are to start the development project with a failing test. After the tests are written for a module, the development of the module can begin. The development code is written repetitively until all test case passes. Finally, the new codes will be refactored and the whole process will start again.

An automated testing system have been implemented with the usage of GitHub workflow. Whenever codes are pushed to the GitHub repository, the preconfigured workflow will run the test against the new codes. This approach was taken to ensure that the new codes that are implemented in the system does not break other modules.


All workflows
Showing runs from all workflows

Filter workflows

83 workflow runs — Event ▾ Status ▾ Branch ▾ A

✓	test cases update Unit Tests #83: Commit 291eefc pushed by chooikj275	main	2 days ago	38s
✗	Added test cases Unit Tests #82: Commit ac3ba05 pushed by chooikj275	main	4 days ago	40s
✓	Fixed minor bug -> symbol separation Unit Tests #81: Commit 5df7c5b pushed by chooikj275	main	4 days ago	47s
✓	Fix comment bug Unit Tests #80: Commit 16dec22 pushed by chooikj275	main	5 days ago	35s
✓	frontend functionalities Unit Tests #79: Commit e5c9b48 pushed by chooikj275	main	7 days ago	39s
✓	v2 Unit Tests #78: Commit bcc29a8 pushed by chooikj275	main	7 days ago	40s
✓	Frontend Unit Tests #77: Commit 95e21c7 pushed by chooikj275	main	7 days ago	37s

Figure 6-1 Automated testing workflow

6.3 Unit Test

Unit testing have been implemented on few modules of the backend processing system which includes the lexer, parser, dictionary and code generator. Each module will be tested independently to verify its correctness.

Test case	Expected Result	Status
Split string of codes into lexemes	Every literal and symbol must be separated in an array except for comments and text to maintain original formatting	Pass
Parse lexemes into tokens	A token is created for every lexeme	Pass
Token type assignment (Symbols)	The token type for the symbols is assigned correctly.	Pass
Token type assignment (Keyword – Java)	The token type for Java keywords is assigned correctly.	Pass
Token type assignment (Keyword – C#)	The token type for C# keywords is assigned correctly	Pass
Token type assignment (Text)	The token type for comments or text is assigned correctly.	Pass
Token type assignment (Literal)	Token type for all non-keyword or symbol literal is assigned correctly.	Pass

Table 6-1 Unit Test Cases – Lexer

Test case	**Expected Result**	**Status**
Parse code into intermediary code format	Able to parse code into abstract syntax form by assigning abstract keywords and maintain value of the code.	Pass
Perform class analysis	Able to produce an array of classes that accurately describes the information of the class.	Pass
Perform structural analysis	Able to produce an array of objects containing the entire structural boundary of the code.	Pass
Parse code into AST	Able to parse stream of tokens into an AST that contains the intermediary codes, class analysis details and structural analysis details.	Pass

Table 6-2 Unit Test Cases – Parser

Test case	**Expected Result**	**Status**
Populate language file into dictionary	Language file is assigned to the dictionary accurately.	Pass
Find keyword	Able to translate language specific to intermediate keyword	Pass
Get translated keyword	Able to translate intermediate keyword to language specific keyword.	Pass

Table 6-3 Unit Test Cases - Dictionary

Test case	Expected Result	Status
Perform class relation analysis	Able to analyse relationship between classes.	Failed
Translate source keyword to target keyword	Able to convert keyword from source language to target language	Pass
Perform code tweaking for specialized keyword	Able to perform set procedure to modify the codes for specialized keyword	Pass
Perform code conversion from AST to target code	Able to produce an entire code in target language with information provided by AST.	Pass

Table 6-4 Unit Test Cases - Code Generator

The test case "perform class relation analysis" failed because the system is designed in a way that it will analyse all the classes during the parsing phase. All the classes in the source code will be identified, then the relationship between the class will be analysed depends on the type of classes. For instance, a relation to an interface class denotes that the base class is implementing the interface while a relation to an abstract class means that the base class is extending the abstract class. This test case will fail if the interface or abstract class are not present in the source code, hence the system cannot find the relation between the classes.

6.4 Integration test

Integration test have been implemented on the entire backend processing system to test if the system is capable to convert codes in different scenarios with all the modules working together. The purpose of this testing is to find out defects in the interaction between the modules.

Test case	**Expected Result**	**Compilation Result**	**Status**
Perform code conversion with multiline comments	The position and the content of the comments should be same as the original code.	Pass	Failed
Perform code conversion with mathematical operators	The mathematical operators should maintain the same as the original source code.	Pass	Pass
Perform code conversion with selection structure	The selection structure should maintain the same as the original source code.	Pass	Pass
Perform code conversion with loop structure	The looping structure should maintain the same as the original source code.	Pass	Pass
Perform code conversion with class inheritance	The converted code should maintain the same class relationship as the original source code	Failed	Failed
Perform code conversion with access modifier	The context of the access modifier for both of the source code and target code should maintain the same.	Failed	Pass

Table 6-5 Integration test cases

The first integration test case failed because the system failed to capture the formatting of the original comments in the source code. The system does not emphasize on ensuring every whitespace is captured during the conversion process because it was converted into an intermediary code where the codes are abstract and meaningless. Hence the whitespaces that were included in the comments or string literals are not captured by the system. Despite the test case failed, the converted codes could produce the same output as the original source code.

The integration test case to test the code conversion with class inheritance failed in both expected compilation result and conversion result because the system was not able to detect the class relationship between the classes if the parent class is not included in the source code. The system will assume that all the classes will be extended and cause the final result to be inaccurate.

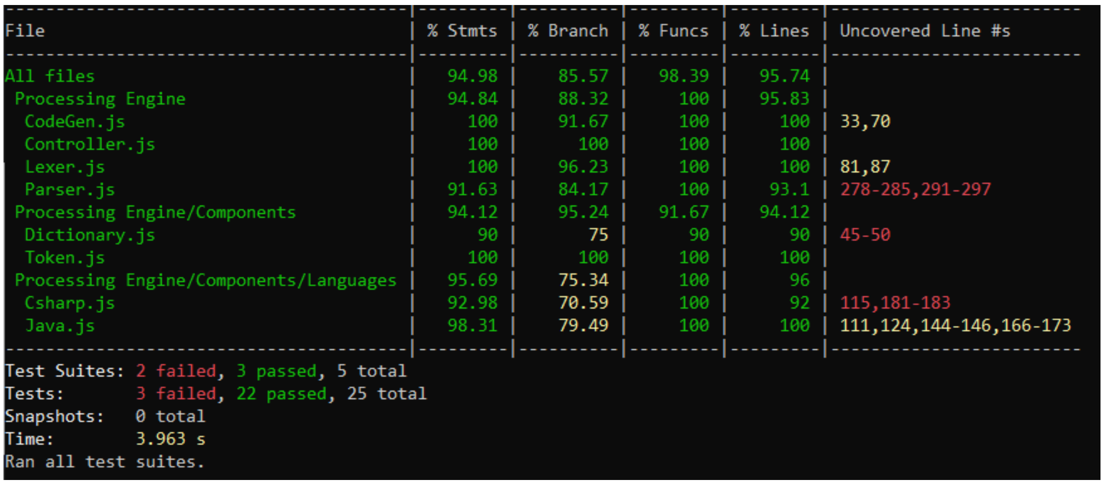
Lastly, the test case to perform code conversion with access modifier failed because compilation output of the converted code is different from the compilation result of the original source code. The output differs because the context of the access modifier of C# and Java programming language is different. For instance, “protected” keyword in C# programming language allows class to be accessible within the class and within the derived class. However, “protected” keyword in Java programming language allow the class to be accessible within derived class and anywhere in the same assembly or package. Hence, the converted code might convert the access modifiers successfully but the context of the accessibility of the code components still differs.

C#	**Java**	**Accessibility**
no modifier private	private	Accessible only within the class
protected	-	Accessible within the class and derived class
internal	no modifier	Accessible within same assembly or within same package
protected internal	protected	Can be accessed within derived class and anywhere in the assembly or package.
public	public	Can be accessed anywhere

Table 6-6 Different context of access modifier

6.5 Test coverage

The test coverage result below shows that more than 85% of the entire codebase is covered by the automated testing.



```
----------------------------------------|---------|----------|---------|---------|-------------------------
File                                    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------------------|---------|----------|---------|---------|-------------------------
All files                               |   94.98 |    85.57 |   98.39 |   95.74 |
 Processing Engine                      |   94.84 |    88.32 |     100 |   95.83 |
  CodeGen.js                            |     100 |    91.67 |     100 |     100 | 33,70
  Controller.js                         |     100 |      100 |     100 |     100 |
  Lexer.js                              |     100 |    96.23 |     100 |     100 | 81,87
  Parser.js                             |   91.63 |    84.17 |     100 |    93.1 | 278-285,291-297
 Processing Engine/Components           |   94.12 |    95.24 |   91.67 |   94.12 |
  Dictionary.js                         |      90 |       75 |      90 |      90 | 45-50
  Token.js                              |     100 |      100 |     100 |     100 |
 Processing Engine/Components/Languages |   95.69 |    75.34 |     100 |      96 |
  Csharp.js                             |   92.98 |    70.59 |     100 |      92 | 115,181-183
  Java.js                               |   98.31 |    79.49 |     100 |     100 | 111,124,144-146,166-173
----------------------------------------|---------|----------|---------|---------|-------------------------
Test Suites: 2 failed, 3 passed, 5 total
Tests:       3 failed, 22 passed, 25 total
Snapshots:   0 total
Time:        3.963 s
Ran all test suites.
```

Figure 6-2 Test coverage

6.5.1 UI test

User interface test is implemented for the frontend website to ensure that all the components on the website is working as expected.

Test case	Expected Result	Status
Upload file to website	The website could import files of predefined extension (txt, .cs or .java)	Pass
Populate file content to code editor	The content of the file at the sidebar will populate the code editor after clicking it.	Pass
Change website appearance	Any changes to the appearance control should change the appearance or settings of the website.	Pass
Convert code	The website should be able to send request to convert the code and display the code via the developed backend API.	Pass
Compile code	The website should be able to compile the code using an external API.	Pass

Table 6-7 Frontend UI test cases

6.6 Usability Testing

Usability testing for the frontend website is carried out using the remote computer software such as TeamViewer and AnyDesk. The purpose of this test is to get the user experience of real users when they are using the website. 30 volunteers are gathered to perform this testing. The demographic of the volunteers are programmers who are studying in university.

The volunteers are given a set of instructions that simulate a scenario (Appendix B). The volunteers will try to complete the set of actions on their own without any instructions to complete the scenario. The entire test process is observed. After all the scenarios have been executed, the volunteers are required to fill in a user satisfaction survey to measure the system usability scale (SUS). The results of the survey will be tabulated and analysed for any further improvement for the frontend website. The questions and responses that were used in this usability testing can be found in Appendix C.

6.6.1 System Usability Scale (SUS)

The SUS survey consists of 10 statements which the users will rate them from the lowest (1) to the highest (5) score. The calculation of the SUS score has few specific steps to follow:

i. The user score is subtracted by 1 for every odd numbered question
ii. The user score is subtracted from 5 for every even numbered question
iii. The total score is sum up and multiplied by 2.5

The evaluation of the website usability can be derived from the final SUS score that were calculated. The category of evaluation is as below:

i. 80.3 and above means that the website is well design and usable to the users.
ii. 68 is the average SUS score which means that the website is usable but need improvements.
iii. 51 or under means that the website needs a lot of improvement in terms of usability and design.

Question	Total Score for 30 participants	Average Score
Question 1	101	3.37
Question 2	105	3.5
Question 3	104	3.47
Question 4	101	3.37
Question 5	102	3.4
Question 6	108	3.6
Question 7	106	3.53
Question 8	108	3.6
Question 9	107	3.57
Question 10	108	3.6
Total Score		35.01
SUS score		87.53

Table 6-8 SUS score table

The results from the user satisfaction survey form are tabulated and calculated into SUS score. The results from the SUS score shows that the websites that were developed for this project is good and easy to be understandable by the users.

6.6.2 Descriptive feedback

Four descriptive questions were provided to the volunteers to get their feedback on the things that they liked the most about the website, the things that they least like about the website, a description about the website and additional comments.

From the collected survey, the users are happy with that the code conversion system works and the simplistic design of the website. However, some UI elements of the website needs improvement as it was less responsive and confusing to the users. The overall response from the users are positive but improvement are still needed to improve

6.7 Evaluation of accuracy

The accuracy of the transpilation framework can be calculated based on few products that were produced during the code conversion process. To measure the accuracy of the model, the converted code must have the same structure as the original source code, the compilation result that were produced by the converted codes must be the same as the original source code, and the intermediate code must be the same between the source code and the converted code. The following table shows the distribution of weights between the three criteria to measure the accuracy of the transpilation framework.

Criteria	Weight	Score	Weighted Score
Correctness of converted code structure	20%	22/25	17.6%
Correctness of the compilation result of the converted code	50%	23/25	46%
Similarity of the intermediate code	30%	22/25	26.4%
Accuracy of the model			90%

Table 6-9 Criteria to measure transpilation framework accuracy

Based on the tabulation of scores, each criterion was given a weight as a significance of the criteria to the model accuracy. The accuracy of the compilation result is the most important criteria because it will affect the behaviour of the entire system and its output. The similarity of the intermediate code and correctness of converted code structure were less important because some code might need to be changed during the code conversion process resulting alteration of the code structure.

The score of each criterion was taken from the test case that were created for this project. To further increase the reliability of the measurement of the accuracy, more real-world test cases were needed. The transpilation framework was able to convert most of the basic programming structure with an accuracy of 90%.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

7.1 Achievements

This project has successfully met all the objectives and requirements that are stated in the document, which includes the following:

i. To identify the issues and practice of the current code conversion process.
ii. To develop a web-based transpiler that is universally compatible with mainstream programming languages.
iii. To design a code transpilation framework.
iv. To achieve code translation accuracy score of 90% for the proposed source to source converter.

The web-based source to source compiler were able to conform to most of the initial project specification, to provide a web-based tool for high level programming language code conversion. The system was developed to enhance the current code conversion practices by creating an intermediate abstraction layer in the conversion process.

Despite many challenges faced during the development of the project, the resources and information gathered from the internet is sufficient to ensure that the development of the project is successfully executed. The system and framework created have provided contributions as follows:

i. Create an intermediate language that promotes universal code conversion between different programming languages.
ii. Replacing traditional XML data storage method with JSON format to enhance code retrieval and update process.
iii. Reduce inconsistencies for code conversion process
iv. Increase accuracy of code conversion between different programming languages
v. Reduce the need to perform manual intervention during code conversion process.

7.2 Limitations

There are few limitations of project that cannot be solved in time due to the project time constraint. Hence, the project is still considered as an early phase and proof of concept as few of the functionalities could not be fulfilled. The limitations of the system are:

i. Some keywords from the programming languages are incompatible or have different context from each other.
ii. Libraries from different programming languages are not compatible with each other as the methods they provide might differ from each other.
iii. Code conversion for object-oriented programming is still error-prone due to different context of the object-oriented keywords.
iv. Lack of test cases that demonstrate real world code which could aid in improving the design of the code structure.

7.3 Future Enhancement

At the moment, the design of the transpilation framework is still immature and require lots of research and improvements to ensure that the intermediary language produced during the conversion process could support all mainstream programming languages.

In the future, I hope few limitations of the system can be solved or improved by introducing new enhancement to the system as follows:

i. Develop libraries of algorithms and modules in such abstract languages or subsets of application languages
ii. Improve structural analysis to improve scalability
iii. Improve abstract syntax to accommodate different languages
iv. Introduce real world programming codes to test the system to improve the code structure design for better conversion accuracy.

REFERENCES

Aho, A. V. *et al.* (2007) *Compilers : principles, techniques, & tools*. second edi. Pearson Education.

Christa, S. *et al.* (2017) 'Software maintenance: From the perspective of effort and cost requirement', in *Advances in Intelligent Systems and Computing*. Springer Verlag, pp. 759–768. doi: 10.1007/978-981-10-1678-3_73.

Coco, E. J., Osman, H. A. and Osman, N. I. (2018) 'JPT : A Simple Java-Python Translator', *Computer Applications: An International Journal*, 5(2), pp. 01–18. doi: 10.5121/caij.2018.5201.

Dahaner, M. *et al.* (2018) *Empowering Graduates for Knowledge, Emerging Technologies for Developing Countries*. doi: 10.1007/978-3-319-67837-5.

Eriksson, M. and Hallberg, V. (2011) *Comparison between JSON and YAML for data serialization*.

George, D. *et al.* (2010) 'Programming Language Inter-conversion', *International Journal of Computer Applications*, 1(20), pp. 68–74. doi: 10.5120/419-619.

Ilyushin, E. and Namiot, D. (2016) 'On source-to-source compilers', *International Journal of Open Information Technologies*, 04(05). doi: 10.4236/jsea.2013.64a005.

Kontogiannis, K. *et al.* (2010) 'Code migration through transformations: An experience report', in *Proceedings of CASCON - 1st Decade High Impact Papers*. New York, New York, USA: ACM Press, pp. 201–213. doi: 10.1145/1925805.1925817.

Kulkarni, R., Chavan, A. and Hardikar, A. (2015) 'Transpiler and it's Advantages', *(IJCSIT) International Journal of Computer Science and Information Technologies*, 6(2), pp. 1629–1631.

Kumar, G. and Bhatia, P. K. (2014) 'Comparative analysis of software engineering models from traditional to modern methodologies', in *International Conference on Advanced Computing and Communication Technologies, ACCT*. Institute of Electrical and Electronics Engineers Inc., pp. 189–196. doi: 10.1109/ACCT.2014.73.

Mäkinen, S. and Münch, J. (2014) 'Effects of test-driven development: A comparative analysis of empirical studies', in *Lecture Notes in Business Information Processing*. Springer Verlag, pp. 155–169. doi: 10.1007/978-3-319-03602-1_10.

Melhase, T. (2012) *Java2Python*. Available at: https://github.com/natural/java2python/blob/master/doc/intro.md (Accessed: 2 September 2020).

Midha, V. (2008) *Does Complexity Matter? The Impact of Change in Structural Complexity on Software Maintenance and New Developers' Contributions in Open Source Software*. Available at: http://aisel.aisnet.org/icis2008/37 (Accessed: 13 July 2020).

Mu, L. (2019) 'gLua: A modern Lua transpiler in Scheme A technical report Guile Lua rebirth View project A flexible middle-ware for managing massive multi-robots View project gLua: A modern Lua transpiler in Scheme A technical report'. doi: 10.6084/m9.figshare.9210428.

Plaisted, D. A. (2013) 'Source-to-Source Translation and Software Engineering', *Journal of Software Engineering and Applications*, 06(04), pp. 30–40. doi: 10.4236/jsea.2013.64a005.

Rastogi, V. (2015) *Software Development Life Cycle Models-Comparison, Consequences, IJCSIT) International Journal of Computer Science and Information Technologies*. Available at: www.ijcsit.com168 (Accessed: 3 September 2020).

Schawalbe, K. (2020) *Information Technology Project Management*. 9th edn.

Smith, N., Capiluppi, A. and Fernández-Ramil, J. (2006) 'Agent-based simulation of open source evolution', in *Software Process Improvement and Practice*, pp. 423–434. doi: 10.1002/spip.280.

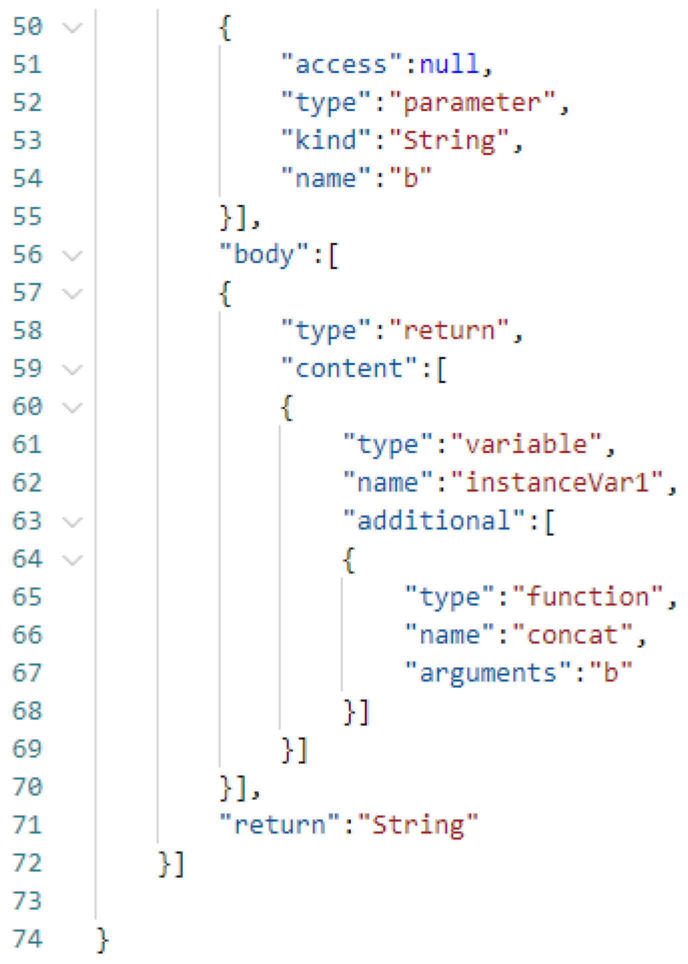
APPENDICES

APPENDIX A: JSON parsing to represent AST

```
public class Class {
     private String instanceVar1;
     private int instanceVar2;

     public Class(String a) {
          this.instanceVar1 = a;
     }

     public String method1(String b) {
          return instanceVar1.concat(b);
     }
}
```

Figure 7-1 Appendix A (1) source code to be parsed

```json
{
    "access":"public",
    "type":"class",
    "kind":null,
    "name":"Class",
    "block":[
    {
        "access":"private",
        "type":"variable",
        "kind":"String",
        "name":"instanceVar1"
    },
    {
        "access":"private",
        "type":"variable",
        "kind":"int",
        "name":"instanceVar2"
    },
    {
        "access":"public",
        "type":"method",
        "kind":null,
        "name":null,
        "parameter":[
        {
            "access":null,
            "type":"parameter",
            "kind":"String",
            "name":"a"
        }],
        "body":[
        {
            "type":"keyword",
            "name":"this",
            "additional":[
            {
                "type":"variable",
                "name":"instanceVar1"
                "content":"a"
            }]
        }],
        "return":"Class"
    },
    {
        "access":"public",
        "type":"method",
        "kind":null,
        "name":"method1",
        "parameter":[
```



```
        {
            "access":null,
            "type":"parameter",
            "kind":"String",
            "name":"b"
        }],
        "body":[
        {
            "type":"return",
            "content":[
            {
                "type":"variable",
                "name":"instanceVar1",
                "additional":[
                {
                    "type":"function",
                    "name":"concat",
                    "arguments":"b"
                }]
            }]
        }],
        "return":"String"
    }]

}
```

Figure 7-2 Appendix A (2) JSON representing AST

APPENDIX B: Test Scenario

<u>Test Scenario</u>

<u>Scenario 1: Perform code conversion from C# to Java</u>

Imagine that you are a user who wishes to use the source code converter website to perform code conversion from C# language to Java language.

How would you perform the action?

<u>Scenario 2: Perform code conversion from Java to C#</u>

Imagine that you are a user who wishes to use the source code converter website to perform code conversion from Java language to C# language.

How would you perform the action?

<u>Scenario 3: Input file(s) into the website for code conversion</u>

Imagine that you are a user who wishes to input some files into the website for code conversion.

How would you import the file?

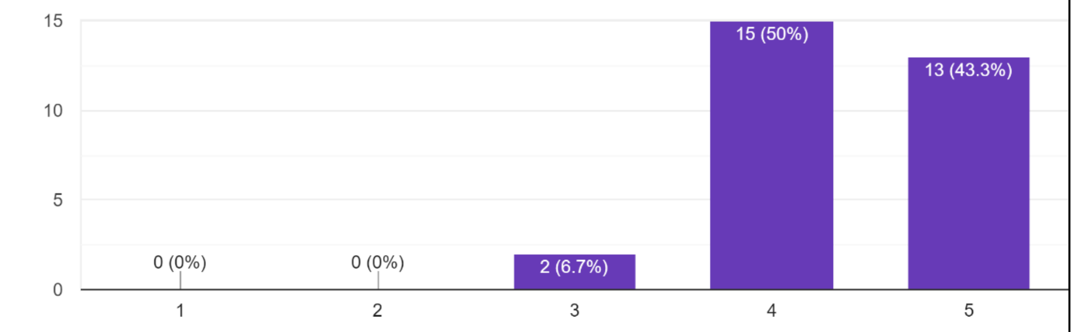
<u>Scenario 4: Change appearance of the website</u>

Imagine that you are a user who wishes to change the appearance and settings of the website.

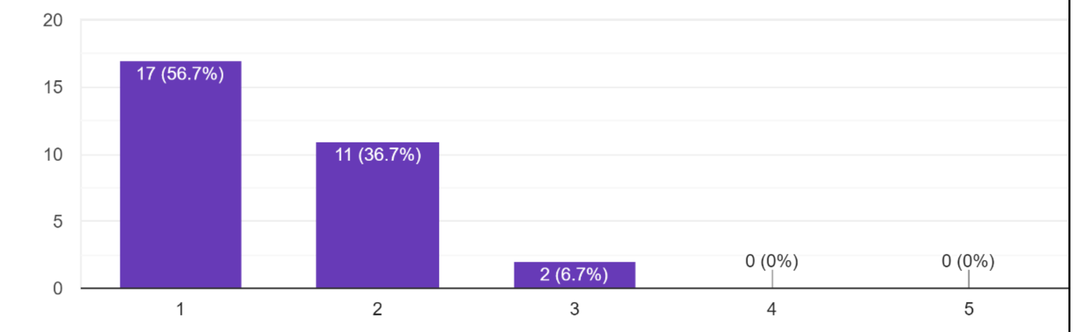
How would you change the settings?

APPENDIX C: User Satisfaction Survey

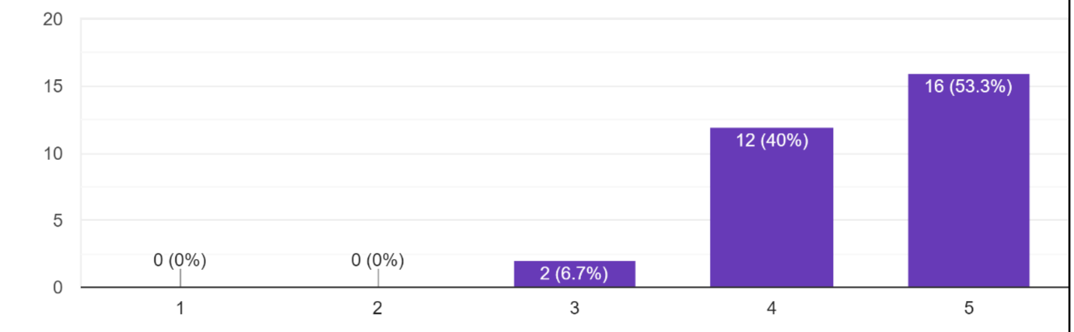
1. I think that I would like to use this system for code conversion related matter.
30 responses



2. I found the system unnecessarily complex.
30 responses

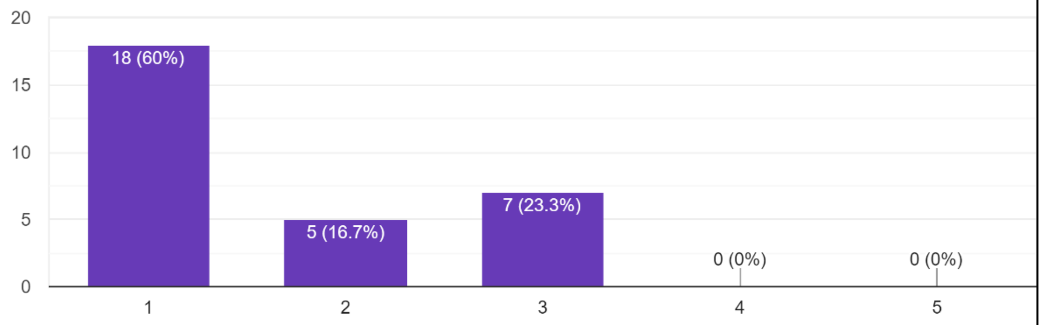


3. I thought the system was easy to use.
30 responses



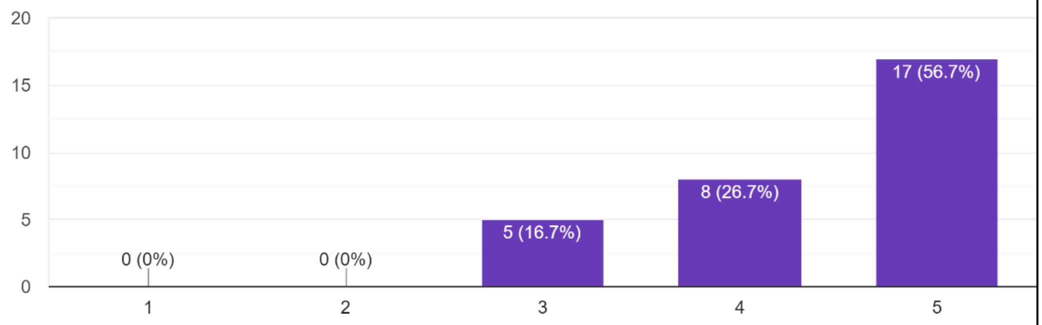
4. I think that I would need the support of a technical person to be able to use this system.

30 responses



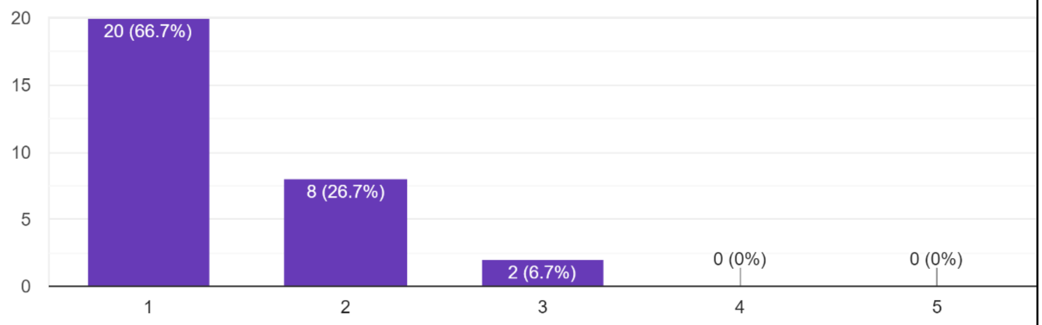
5. I found this system was easily moved thought without a lot of backtracking or data re-entry.

30 responses



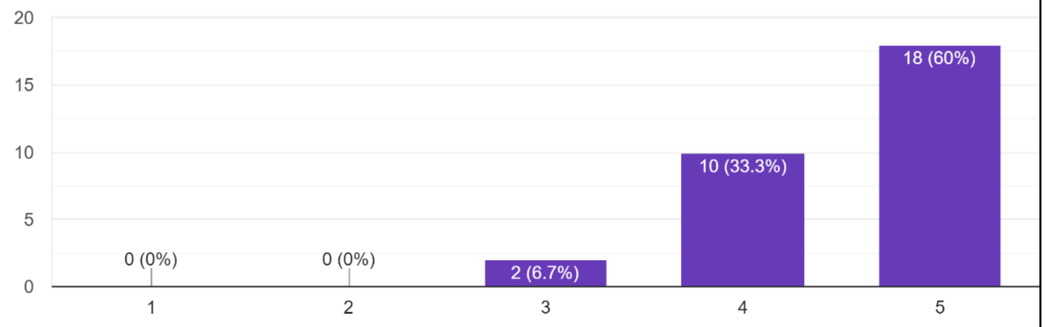
6. I thought there was too much inconsistency in this system.

30 responses



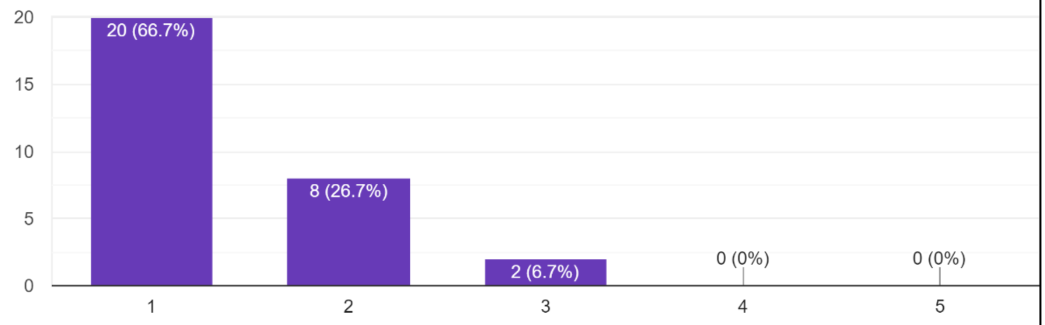
7. I would imagine that most people would learn to use this system very quickly.

30 responses



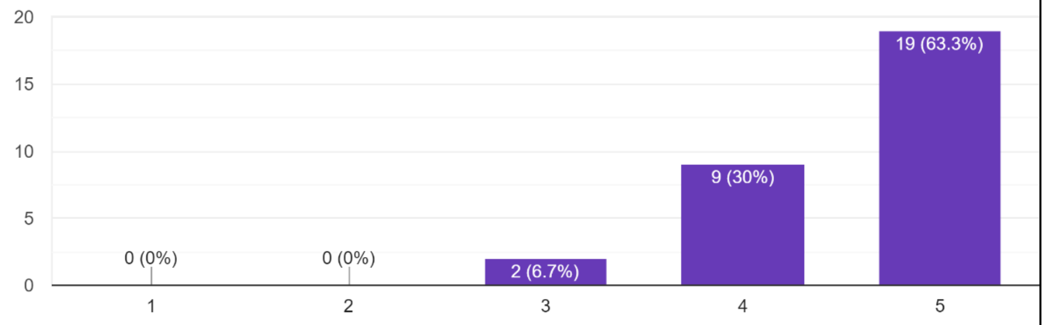
8. I found the system very awkward to use.

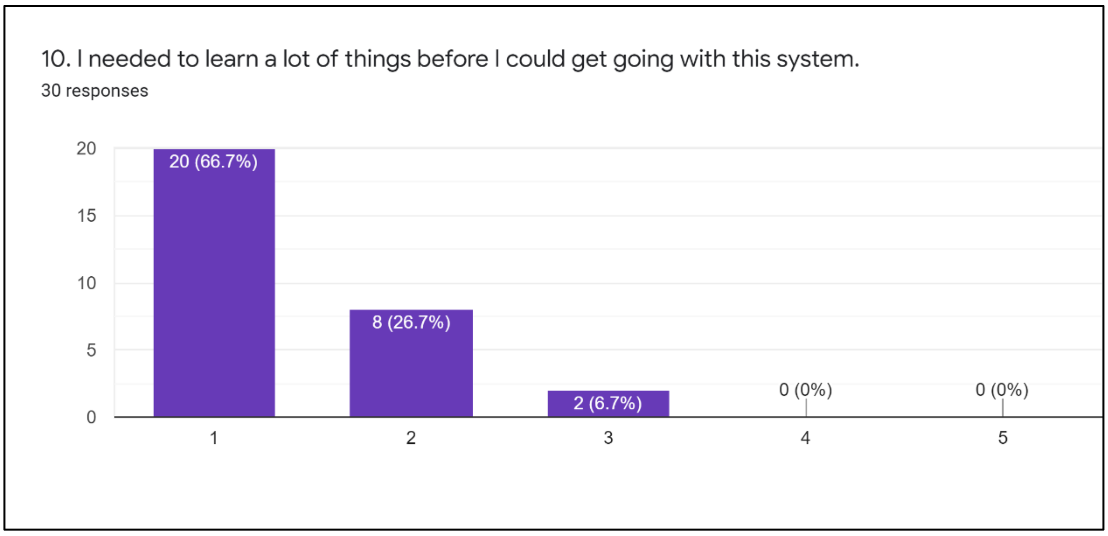
30 responses

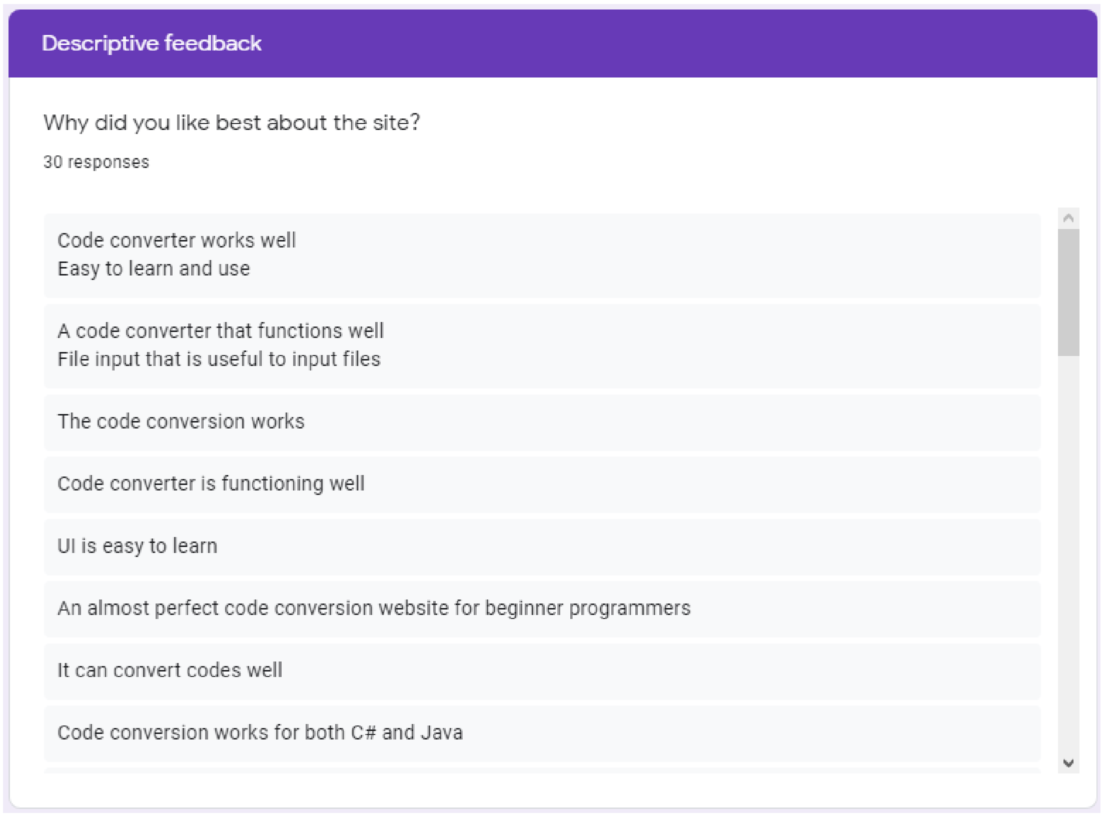


9. I felt very confident using the system.

30 responses




10. I needed to learn a lot of things before I could get going with this system.
30 responses
20
15
10
5
0
20 (66.7%)
8 (26.7%)
2 (6.7%)
0 (0%)
0 (0%)
1
2
3
4
5



Descriptive feedback

Why did you like best about the site?

30 responses

Code converter works well
Easy to learn and use

A code converter that functions well
File input that is useful to input files

The code conversion works

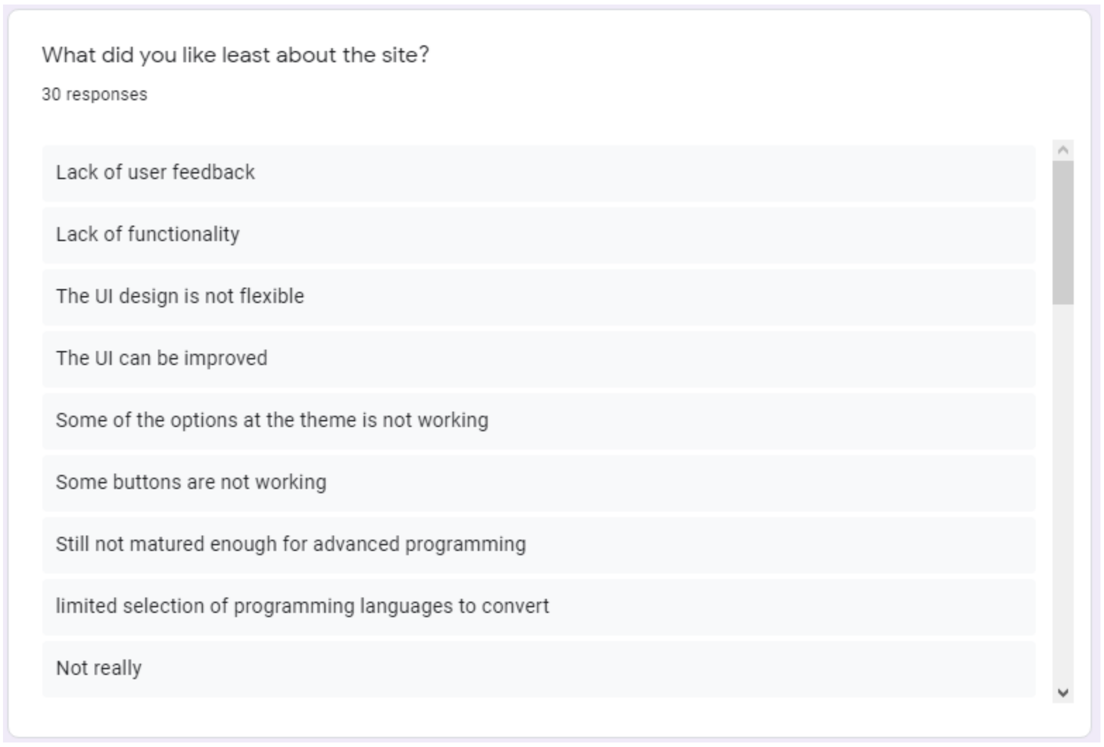
Code converter is functioning well

UI is easy to learn

An almost perfect code conversion website for beginner programmers

It can convert codes well

Code conversion works for both C# and Java



What did you like least about the site?

30 responses

Lack of user feedback

Lack of functionality

The UI design is not flexible

The UI can be improved

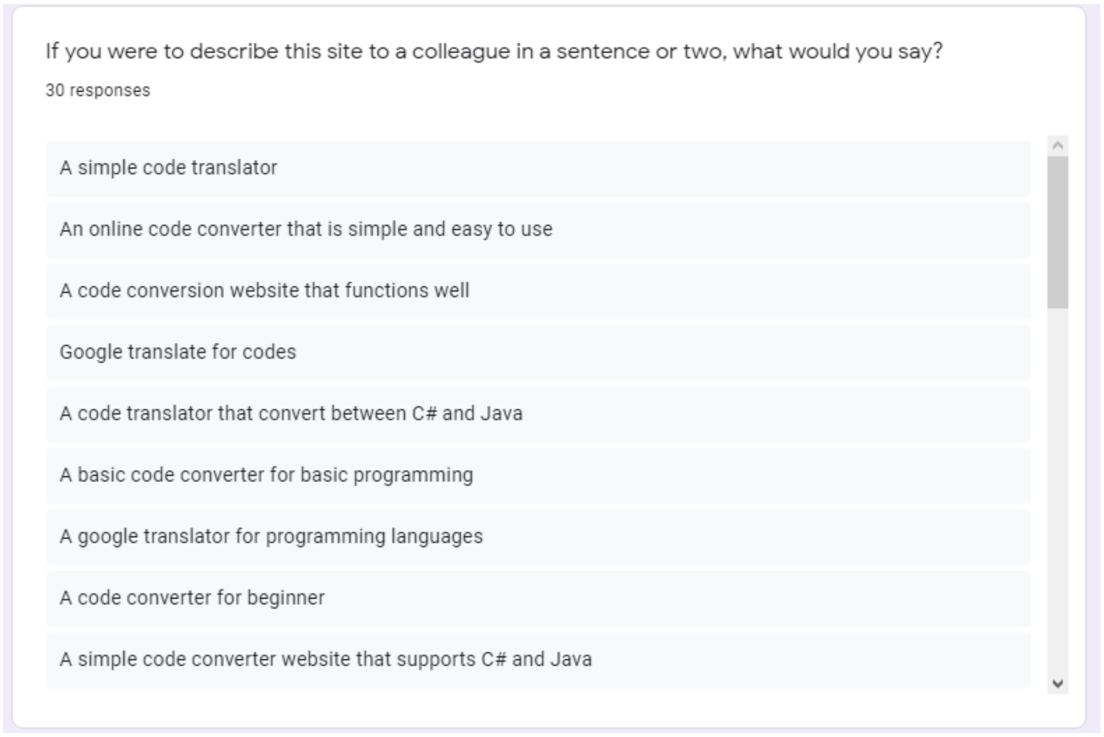
Some of the options at the theme is not working

Some buttons are not working

Still not matured enough for advanced programming

limited selection of programming languages to convert

Not really



If you were to describe this site to a colleague in a sentence or two, what would you say?

30 responses

A simple code translator

An online code converter that is simple and easy to use

A code conversion website that functions well

Google translate for codes

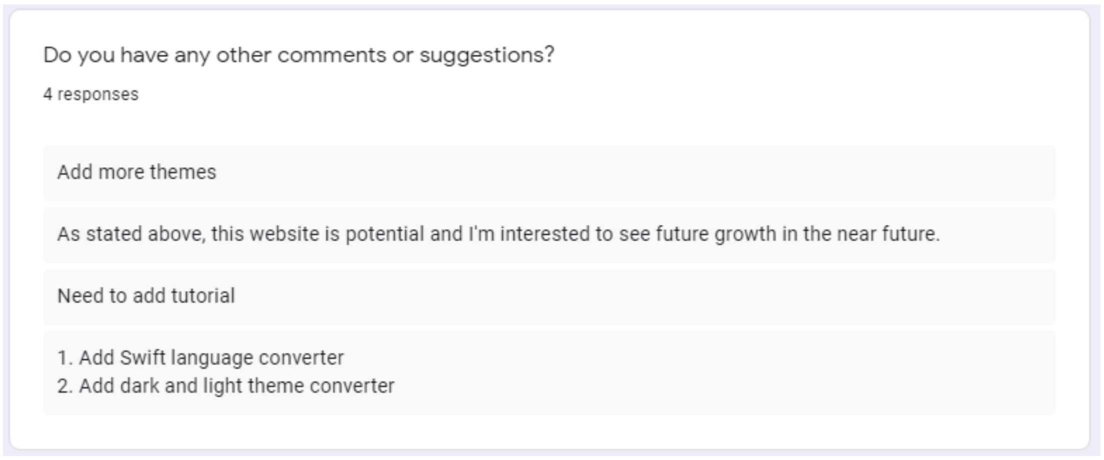
A code translator that convert between C# and Java

A basic code converter for basic programming

A google translator for programming languages

A code converter for beginner

A simple code converter website that supports C# and Java



Do you have any other comments or suggestions?

4 responses

Add more themes

As stated above, this website is potential and I'm interested to see future growth in the near future.

Need to add tutorial

1. Add Swift language converter
2. Add dark and light theme converter