DOI AND ISBN CITATION CONVERTER

YEN PEI XUAN

UNIVERSITI TUNKU ABDUL RAHMAN

DOI AND ISBN CITATION CONVERTER

YEN PEI XUAN

A project report submitted in partial fulfilment of the
requirements for the award of
Bachelor of Software Engineering with Honours

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

September 2025

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.

Name : Yen Pei Xuan

ID No. : 2105999

Date : 18 September 2025

COPYRIGHT STATEMENT

This final year project report is submitted in partial fulfilment of the requirements for the degree of software engineering at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

This project developed a web-based citation generator to automate the accurate formatting of academic references, addressing a significant need among students and researchers who struggle with manual citation processes. The system supports both Digital Object Identifiers (DOIs) and International Standard Book Numbers (ISBNs), generating citations in multiple styles including APA, Harvard, and IEEE. The methodology adopted is an Iterative approach. The application employs a three-tier architecture consisting of a lightweight HTML/CSS/JavaScript front end, a high-performance Python backend built on FastAPI, and DuckDB for persistent metadata storage. The system fetches metadata from external APIs asynchronously, using CrossRef for DOIs and Open Library for ISBNs, and using Pydantic models to provide strict input validation. A key innovation involves using DuckDB as a local cache to minimise redundant API calls and significantly improve response times. Testing of the system demonstrated its reliability in generating accurate citations, managing invalid inputs with ease, and efficiently processing multiple references through its advanced bulk upload functionality. The project provides a practical and user-friendly solution that saves time while maintaining academic integrity by reducing formatting errors. Future developments may include additional citation styles, export features, and cloud-based deployment to enhance scalability.

Keywords: citation, APA style, IEEE style, Harvard style, citation converter, web application

Subject Area: PN172 – Literary Composition Techniques

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

c_p	specific heat capacity, J/(kg·K)
h	height, m
K_d	discharge coefficient
M	mass flow rate, kg/s
P	pressure, kPa
P_b	back pressure, kPa
R	mass flow rate ratio
T	temperature, K
v	specific volume, m^3
α	homogeneous void fraction
η	pressure ratio
ρ	density, kg/m^3
ω	compressible flow parameter
ID	inner diameter, m
MAP	maximum allowable pressure, kPa
MAWP	maximum allowable working pressure, kPa
OD	outer diameter, m
RV	relief valve

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Citing sources is a fundamental component of academic and professional writing, forming the foundation of academic integrity and intellectual honesty. A citation is the acknowledgment of the source that the authors have used in their essay, assignment, or journal to support their arguments and provide evidence. The authors provide a reference to the source, which means they have acknowledged that they have read the work and recognize its ideas in their own writing. (Montana.edu, 2020) A responsible author gives credit to original authors, supports claims with evidence, and prevents plagiarism by quoting ideas used by other authors.

Citations have a number of practical applications, other than preventing plagiarism. They provide readers with a clear framework for tracing the evolution of ideas, enabling them to verify facts, investigate related studies, and gain a more comprehensive understanding of the subject. As besteditproof.com (2022) notes, the proper citations can help the reader find the sources from which the authors acquired the idea and check the facts. Good citation habits build trust and keep writing clear and honest, and scholars place a high value on and track the evolution of concepts. It shows the reader that the writer has done proper research by listing sources that were used to get information. The citation includes the author's name, journal title, date, DOI(Digital Object Identifier) or ISBN (International Standard Book Number), depending on the reference style used. Each citation style specifies the required information for a citation in a unique order and punctuation. There are many reference styles to use, for example, APA(American Psychological Association) style, Harvard style, and IEEE(Institute of Electrical and Electronics Engineers) style.

Why are there so many different reference styles? This is because the different reference styles are used for different fields and are suitable for them. For example, the APA style is favored in the fields of social science, education, and psychology, which is suitable for quantitative studies; the IEEE style is suitable for use in engineering, while the Chicago Manual of Style is suitable

for history. (Hunter, 2006) Another difference is that the source's information in reference will have a different order, although they have the same basic information, such as the authors' names, publication year, and the source's title. In APA style, the order of information starts with the author's name, followed by the publication year. This two information are the most important in the social science and education fields. It is important to use the right style because referencing is also a professional skill. (Which referencing style should you use?, 2019) It is important to note that understanding citation styles goes beyond simply knowing how to format references correctly. It also reflects a commitment to the standards set out in each discipline, adherence to established academic traditions, and a professional approach to all aspects of work. Correct citation methods demonstrate respect for others' intellectual contributions and contribute to the greater academic debate by clearly acknowledging the sources of ideas.

The Digital Object Identifier (DOI) System is used for recognising content objects in a digital environment. DOIs are assigned to any sources utilised on digital networks, such as journals and books. They can provide the newest information and are easy to find on the Internet because the DOI name will never change, although the content may change. The DOI System is a framework for easier management of content, metadata, and enables the automated management of media. Nowadays, we can get the information about sources by the DOI easily from the Internet, because each DOI is a unique alphanumeric string to identify the digital content, such as a journal and an article, and then provides the link on the Internet. (DOI Foundation, 2022) The DOI is commonly written on the first page of sources, so we can find it easily.

An International Standard Book Number (ISBN) is a product identifier used for publishers, libraries, and internet retailers to easily list the book’s sales records and stock control. The ISBN can identify the book's title, edition, format, and other information. The ISBNs are assigned to text-based monographic publications like academic books. Every book will have its unique ISBN that is built with 13 digits, and the beginning of the number should start with either 978 or 979. Each book edition is given a different ISBN, enabling easy identification of differences between versions by libraries, bookshops and readers alike.

This project aims to develop an automated citation converter that is convenient and highly accurate in reference style for users. This citation converter will support APA, Harvard, and IEEE style references by the DOI or ISBN provided by users. Furthermore, the citation converter will be able to provide a high-speed response for the users, and users will be able to copy or download the converted reference from the website.

1.2 Importance of the Study

The challenges of citation converters are that reference styles will have different editions, and as time goes by, the reference styles might have huge differences. Although numerous citation tools exist, many of them may use outdated editions of citation styles and generate incorrect formats, which can compromise the accuracy of academic work. In this case, users often need to manually format citations, which may lead to errors in referencing. Other than that, some of the citation converter tools require users to manually input the information of sources, as they fail to detect the necessary information from metadata. This will let users take more time to check back the sources and input by themselves, and also need to take time to verify citation details.

Secondly, the academic penalties for improper citation could be severe, including paper rejection, reduced grade, and accusations of academic misconduct. In such cases, plagiarism detection tools are employed to identify improper citations and plagiarism, thereby underscoring the significance of accurate referencing. The objective of this project is to reduce the risk of citation errors and enhance academic integrity by automating the citation process through the extraction of DOI and ISBN metadata.

1.3 Problem Statement

Manually formatting citations is highly prone to making mistakes, especially when formatting with multiple reference styles across different academic disciplines, and users might use the incorrect reference style format. The users may not understand the citation rules and lead to inconsistencies and incorrect reference formatting, for example, missing publication year, improper source titles, and missing commas. These errors can have serious consequences for academic integrity and professional reputation.

Although many online reference tools are available, some produce incorrect citation formats or incomplete metadata. Additionally, some citation converters operate with outdated versions of reference styles; many are still using APA 6th edition, even though the current version is 7th edition. This may result in inconsistencies when academic institutions require adherence to the most recent guidelines. Secondly, some of them used metadata retrieved from APIs or databases, which is not always complete or accurate. This will lead to incorrect or incomplete information, and users need to manually input the information. Furthermore, many of the reference converters online, few of the tools support DOI and ISBN metadata extraction while ensuring accurate citation formation across multiple reference styles, which creates challenges when citing books and journals.

These limitations indicate a market requirement for a robust, reliable and highly accurate citation converter. This converter should be able to automatically detect and extract metadata using DOI and ISBNs, while formatting the citations according to updated APA, Harvard and IEEE styles. Therefore, an accurate citation converter tool is needed that can effectively address this discrepancy by offering expeditious, precise, and adaptable citation services that are aligned with contemporary academic requirements. This project aims to address the problem and develop a tool that can convert DOI and ISBN references into multiple reference styles.

1.4 Aim and Objectives

This project's primary objective is to develop an automated citation converter that can efficiently obtain metadata from DOI and ISBN inputs and accurately format citations in accordance with APA, Harvard, and IEEE to reduce manual citation effort and prevent formatting errors. This project is developed to reduce the manual burden on academics, students, and researchers when citing their sources, increasing citation accuracy and consistency, while minimising citation errors and promoting academic accuracy. The objective of the project is not merely to transform input into output, but also to establish a system that provides metadata validation, robust error handling, and an enhanced user experience. It is anticipated that this tool will offer users rapid response times, enabling them to acquire citations in seconds.

Moreover, the tool will be designed to be scalable and stable, with the capability to manage multiple requests without compromising performance. This project is designed with modularity to enable easy future updates to accommodate changes in citation styles as the citation standards evolve over time. This project places a significant emphasis on accessibility and usability, ensuring that individuals with limited technical expertise are able to generate high-quality citations through a straightforward web interface. The integration of automation, validation, and usability is expected to yield substantial contributions to academic writing standards and promote optimal scholarly practices.

1.4.1 Project Objectives

The project will pursue the following objectives to attain the above-stated goal. Firstly, develop an automated system that retrieves metadata from the CrossRef for the DOI's information and the Open Library for the ISBN's information. The objective of this undertaking is to use the CrossRef and Open Library, with the intention of facilitating the automated retrieval of key citation metadata. The system has been designed to ensure the highest standards of data accuracy by sourcing information from well-known and reputable sources. The retrieval routines must be capable of gracefully managing problems, such as missing fields or API outages, and have fallback measures in place if data retrieval fails.

Secondly, create a user-friendly website to enable users can have a seamless experience and generate citations efficiently and correctly. The project will result in the creation of an accessible and adaptable web interface, in which users may enter their DOI or ISBN, select their needed citation style, and swiftly obtain a reference prepared to a standardised format. The principles of user experience (UX) that will be considered include clear feedback messages, quick input validation, and immediate visibility of system status. The website will also include certain guidelines, such as an example of a Digital Object Identifier (DOI) and International Standard Book Number (ISBN), as well as an explanation of the importance of referencing.

Thirdly, this project will implement error handling and validation to prevent incomplete or incorrect citations and verify the metadata consistency. The system is designed to provide rigorous validation tests on both the front end

and the back end. The system validates user input formats and ensures that API responses include all essential metadata, and issuing warnings to users when human input or edits are required are all features of the system. This objective is to ensure that, even in the event of incomplete data, the system will direct the user to complete an accurate citation.

Lastly, this project will ensure that the system is scalable, maintainable, and extensible in the future. The system's architecture will be characterised by modularity, with a clear delineation of components, such as API retrieval, citation preparation, and user interface. The modular design of the system will facilitate the future updating of citation formats, addition of new reference styles, and enhancement of metadata retrieval capabilities, without the need for substantial system changes.

1.5 Scope and Limitation of the Study

1.5.1 Target Users

The primary users of this citation converter are university students, academics, and researchers who frequently cite sources in APA style, Harvard style, and IEEE style. Undergraduate and postgraduate level students are often required to strictly adhere to the citation guidelines as a part of their academic thesis, dissertations, and assignments. Academics and researchers will need to ensure that their references meet rigorous formatting standards when preparing conference papers, research reports, and journal submissions. This citation converter aims to provide accurate citations efficiently and in compliance with the newest edition of the reference style. This project especially focused on APA, Harvard, and IEEE styles, which are among the most extensively utilised across educational institutions globally, as different academic disciplines have differing preferences for citation styles. Beyond the confines of academic settings, professional writers engaged in the creation of technical documentation, policy reports, or white papers may find value in a tool that ensures consistent and high-quality formatting of citations, thereby enhancing the professional presentation of their work.

1.5.2 Limitations of the Study

The project has several limitations of the study, as the citation converter focuses on DOI and ISBN-based references and is designed for a limited type of citation references. Firstly, the converter is only able to consider references that are specified by DOI or ISBN numbers because the functionality of the tool in question is contingent upon the presence of a Digital Object Identifier (DOI) or International Standard Book Number (ISBN). If the user inputs sources that do not conform to these criteria, such as informal blog postings, personal emails, or unpublished works, it will show an error message to users. Therefore, the users are required to manually create citations for these sources in accordance with standard procedures.

Secondly, the system will only support three citation styles: The reference styles employed in this text are APA, Harvard, and IEEE. It should be noted that alternative styles, including those designated as Chicago, Vancouver, and AMA, are not supported during this particular project phase. While these styles may be incorporated into subsequent iterations, the present project places emphasis on the three styles that are most pertinent to the specified target user base.

Thirdly, the system's reliance on external APIs, such as CrossRef and Open Library, creates a dependency risk. If these services modify their API formats, experience periods of downtime, or furnish insufficient metadata, the accuracy of citations may be adversely affected. While local caching or manual input alternatives can help mitigate this risk, reliance on external sources in real-time remains a limitation. Furthermore, if the metadata provides incomplete or incorrect information, the citation accuracy may be affected.

Lastly, the citation converter is web-based and reliant on external data retrieval; consistent internet access is a prerequisite for its functionality. In environments characterised by limited connectivity, users may encounter delays or interruptions when attempting to generate citations.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Citation is essential in academic writing, as it ensures that authors have conducted proper research and acknowledged their sources. The citation formatting was done manually before the citation tools were introduced, and this may lead to incorrect citation format and waste time. With the progress of science and technology, a lot of citation tools have been developed to help users quickly format citations correctly and reduce errors. A literature review is an integral component of any research endeavor, as it provides a comprehensive overview of preceding research, identifies lacunae in existing knowledge, and substantiates the necessity for the study in question. In the context of this project, it is imperative to examine existing citation and reference management technologies in order to understand their capabilities, strengths, and limitations. This chapter provides an overview of the existing citation tools with their strengths and limitations, mainly the DOI and ISBN citation, and the process of metadata retrieval. This chapter also writes the newest APA, Harvard, and IEEE style reference format, and the comparison between manual and automated citation formatting. Therefore, the subsequent sections will discuss and evaluate existing citation tools, metadata extraction methods, and reference formatting systems relevant to this study.

2.2 Literature Review

This section reviews existing citation tools, analyzing their advantages, disadvantages, and the gaps that motivate the development of this project.

2.2.1 Existing Converter Tools

In the literature review, three citation tools were compared about their strengths and limitations, as these tools are widely used and well-known on websites.

The Zotero reference tool is a tool that can automatically sense the content, like using a web browser, and is easy to use. It is a free and open-source desktop software that can store website content on local storage. It works well

with Chrome and Safari to detect and save links from web pages, and helps users organise their searches the way they want, such as sorting items in the collection and tagging them with keywords to make them easier to find. Furthermore, Zotero can capture metadata automatically from many sources, like videos and PDFS, which is a very unique function. Zotero also supports over 9,000 citation styles that users can use to format their references and allows sharing libraries for group collaboration, using cloud sync features so that users can synchronise the data across devices. (Zotero, 2025) However, Zotero also has its limitations; firstly, it provides 300 MB of free storage, and the additional storage needs to be paid for with a subscription. Compared to other existing citation converter tools, its interface is too simple and less modern. Although it has community-based support, it doesn't have a customer support team to solve issues.

In contrast to Zotero, Mendeley adopts a more research-oriented ecosystem with collaborative tools for academics. Mendeley is a desktop-based reference management software that requires users to download the application. Mendeley Cite is the new feature that adds to Microsoft Word and is compatible with Microsoft Office 365, so users can download it along with plug-ins for the Microsoft software to insert citations. It offers a range of features to help connect with discoverers and researchers, for example, a cross-device backup feature to sync the content automatically, intelligent filtering, and a PDF viewer. Mendeley aims to provide users with effective and fast reference management. Secondly, users can build their own Mendeley library to search, organize, and read their references. Additionally, it will also help users to collate all of the notes and highlights from a PDF file, so that users can easily find their highlights. (Mendeley, 2025) Mendeley is owned by Elsevier, which is a large publishing company, so users are concerned about issues of openness and privacy. In addition, the free version of Mendeley only allows users to create 5 groups and limits them to 25 members per group. Highlighting and comments are no longer shared between different group members and the user's original file. (Uw.edu, 2025)

On the other hand, EndNote is a more advanced commercial solution tailored for professional researchers and institutions. EndNote aims to simplify the complex task and organise complete references for users; it is widely used in research and academic institutions. EndNote also allows users to create a

smart group by customising the tag and field. It also supports cloud synchronisation and sharing, allowing users to access and collaborate via EndNote Web. It is available for Windows and Mac users, and has three versions: EndNote21, EndNote Web, and EndNote Basic. The EndNote Basic version is free for anyone, can integrate with MS Word, and does not require installation. The EndNote basic version has a file attachment storage limit of 2 GB only, and reference storage is just 50000. Different versions of EndNote have different maximum numbers of fields per reference and type. If users would like to use EndNote 21, they need to buy and install it on their devices, as it provides more services and efficiency. (EndNote, 2025)

Table 2.1: Tables of Tools Comparison

Tool	Strengths	Limitations
Zotero	Free, open-source, supports 9,000 styles, browser integration	Limited free storage, lacks customer support
Mendeley	MS Word integration, PDF annotation, and collaboration tools	Limited free groups, privacy concerns
EndNote	Advanced management, cloud sync, institutional support	Paid full version, limited free features

The strengths of these tools are that they all support multiple reference styles, such as APA style, Harvard style, Chicago style, and IEEE style. Some tools also offer a browser extension function that it will automatically captures the metadata. After users cite the resources, the completed references can be saved in a library for reuse purposes. The tools can easily format the imported citation for users and produce various types of style formats. Other than that, they also import citations from other databases to retrieve sources' information and cite them according to the reference style required by users.(Tutorspoint, 2024) They support many types of export citations, for example, BibTeX, copying to the clipboard, and downloading as a Word file.

Other than strengths, these citation generator tools also have their limitations. The application is built to earn profit; some features will be locked and require users to purchase the premium version to access all these features.

The accuracy of the input data is important because incorrect citation references may cause problems for users. The automatic data retrieval features will be convenient for users, but sometimes they bring some trouble. The tools can pull the incomplete details or incorrect information, and the users are required to manually edit the reference themselves. If users input the wrong information, the reference will generate an incorrect reference for users. Additionally, the citation generator may pull data from an untrustworthy source, which will affect the accuracy of citations. Thirdly, the citation generator might use an old version of the reference style or an incorrect format to cite the references, such as when to use a comma, which type of style uses the bracket, and different reference styles use different information. (Rephrasely, 2024)

2.2.2 DOI and ISBN Metadata Extraction

The metadata extraction function is a process that identifies and extracts the necessary information from an open source, for example, the authors' names, the title of the article, and the year of publication. This is also a core function to enable the automated citation generator. The DOI is a unique alphanumeric string to identify the digital content and provide a link on the Internet. To retrieve metadata using a DOI, the tools can access the DOI Foundation API and return a JSON-formatted data that contains the necessary information for the system.

A unique ISBN that is built with 13 digits and identifies text-based monographic publications like academic books. ISBN metadata can be retrieved through the public API, such as Open Library, and the necessary information will be returned to the system. To enhance the speed of the citation generator and the performance of the system, some tools will store the references that have been referenced by users in the databases. If other users cite the same DOI or ISBN, the tool will fetch the metadata from databases and return it to the users. This can also reduce reliance on external API calls and improve loading times.

2.2.3 Comparison between existing APIs Tools

This project requires the use of an API web application framework to assist in project development. This section will review the three frameworks: Django, Flask, and FastAPI, about their function, strengths, and limitations.

Django is a high-level, full-stack Python web framework that facilitates the fast construction of safe and maintainable websites with a clean, pragmatic design. The software contains integrated tools for managing administrative interfaces, database models, and more, allowing developers to focus on building the software rather than reinventing the wheel. (Django, 2024) Django provides a comprehensive solution that incorporates an ORM (Object-Relational Mapper), a templating engine, form handling, and security features as standard. The software in question has been exhaustively documented, is capable of extreme scalability, and boasts a substantial user base that provides substantial backing. The software adheres to the "batteries included" concept, thereby minimising the necessity for additional libraries. However, Django can present certain challenges in terms of its complexity, which may be more suited to larger projects and more substantial applications. Its monolithic nature has the potential to incur excessive overhead when only basic REST APIs are required. The customisation of such frameworks that exceeds the basic conventions established by Django has been shown to require a greater investment of time and effort.

Flask is a WSGI web application framework that is lightweight and adaptable, allowing users to initiate development processes expediently. Flask aims to facilitate the development of small to medium web applications and RESTful APIs. It encompasses the essential tools required for web development, while concurrently enabling developers to select their preferred components. (Flask, 2010) Flask is a straightforward and uncomplicated software, which renders it optimal for utilisation in diminutive applications for prototyping. It is a highly flexible system that facilitates rapid deployment with minimal coding. This paradigm shift empowers developers with unparalleled autonomy, granting them the freedom to select their authentication system, ancillary add-ons, and preferred database. Since Flask's simplicity has prompted the development of numerous third-party extensions to address its limitations, though this approach can lead to inconsistent architectures. In the case of Flask, additional configuration may be necessary for functionality included in frameworks such as Django for projects of a larger or more complicated nature.

FastAPI is a contemporary, high-performance web framework for the creation of APIs in Python, utilising conventional Python type hints. (FastAPI,

2023) The software has been developed on the basis of Starlette for web parts and Pydantic used for data validation, with asynchronous support and automatic API documentation generation. FastAPI is distinguished by its rapid execution, attributable to its asynchronous features and superior processing speed. These characteristics are comparable to those observed in Node.js and Go APIs. It generates OpenAPI (Swagger) documentation automatically, thus eliminating the need for any additional configuration. The microservice paradigm, modern web APIs, and asynchronous programming are well-suited to this approach. Additionally, FastAPI provides robust typing and validation using Pydantic models, which has been demonstrated to reduce problems and improve code clarity. The FastAPI ecosystem is in a state of development and expansion, given its relatively recent emergence as a software framework in comparison to more established options such as Django and Flask. It is evident that certain features, such as full-stack website support, like templating and ORM, are less comprehensive in their initial state than those offered by Django. Those developers unfamiliar with asynchronous programming may encounter an initial period of adjustment.

Table 2.2: Table of differences between frameworks comparison

Framework	Function	Strengths	Limitation
Django	Build complex websites	Comprehensive features, widely supported	Heavy for small projects, rigid structure,
Flask	lightweight micro-framework	Simple and flexible, easy to learn	Lacks built-in tools, Required manual setup extensions
FastAPI	Modern, high-performance	Asynchronous support, automatic validation with Pydantic	Newer ecosystem, learning curve for async handling

Based on the analysis above, FastAPI was chosen for this project due to its balance of speed, modern design, and validation capabilities. The primary reasons for this choice are as follows: the enhanced performance, the automated

validation of input, the asynchronous processing of requests, and the simplified development of RESTful APIs. The FastAPI software was found to offer an optimal combination of simplicity and functionality, aligning well with the project's requirements for a lightweight, efficient, and highly responsive citation converter with API-based metadata retrieval. The software's automated production of interactive API documentation (Swagger UI) has been demonstrated to streamline the development and testing processes. A comparative analysis of the structural intricacies of Django and the absence of integrated validation mechanisms in Flask reveals that FastAPI emerges as the most efficacious, contemporary, and extensible solution to meet the stipulated project requirements. Furthermore, the built-in data validation of FastAPI with Pydantic ensured the safe processing of DOI and ISBN queries, thus resulting in a more robust and error-free solution.

2.3 Citation Formatting Rules

The following section outlines the reference formatting rules for three major citation styles: APA, Harvard, and IEEE, along with examples for DOI-based and ISBN-based citations. The different style of reference requires different information; here is an example of each style of reference used by the DOI (Journal) and the ISBN (Books). The APA reference format was according to the Publication Manual of the American Psychological Association, 7th edition. The Harvard style was according to the Cite Them Right book. The IEEE reference format was according to the Universiti Tunku Abdul Rahman's IEEE Reference Guide.

2.3.1 APA style (7th Edition)

The APA reference format was according to the 7th edition of the Publication Manual of the American Psychological Association. (2020)

DOI Reference Format:

AuthorA, LastName. FirstName., & AuthorB, LastName. FirstName. (Year of Publication). Title of the article: Subtitle. *Title of the Journal,* Volume number(Issue number), page–page. https://doi.org/xxx

Example:

Smith, J., & Lee, A. (2020). Investigating citation tools in academia. *Journal of Information Systems*, *15*(2), 123–134. https://doi.org/10.1016/j.jis.2020.02.004

ISBN Reference format:

AuthorA, LastName. FirstName., & AuthorB, LastName. FirstName. (Year of Publication). *Title of book: Subtitle* (2nd ed.). Name of Publisher.

Example:

Clark, J. W. (2012). *AC Power Conditioners*. Academic Press.

2.3.2 Harvard style

The Harvard style was according to the Cite Them Right book. (2022)

DOI Reference Format:

AuthorA, LastName, FirstName., and AuthorB, LastName, FirstName., Year of Publication. Title of article. *Full Title of Journal*, [e-journal] Volume number(Issue number), pp.page-page. https://doi.org/xxx

Example:

QU, H.-B., CHEN, X., WANG, S.-T., & YU, M.,2015. Forward Affine Point Set Matching Under Variational Bayesian Framework. *In Acta Automatica Sinica,* [e-journal] 41(8), pp.1482–1494. https://doi.org/10.1016/s1874-1029(15)30001-x.

ISBN Reference Format:

AuthorA, LastName, FirstName., and AuthorB, LastName, FirstName., Year of Publication. *Title of book*. Edition. Place of publication (town or city): Publisher.

Example:

Clark, J.W. , 1990. *AC power conditioners: Design and applications*. San Diego: Academic Press.

2.3.3 IEEE styles

The IEEE reference format was according to the Universiti Tunku Abdul Rahman's IEEE Reference Guide. (2022)

DOI Reference Format:

[1] AuthorA, FirstInitial. SecondInitial. LastName, and AuthorB, FirstInitial. SecondInitial. LastName, "Title of the article," in *Abbreviated Name*. Year of Publication, pp.page–page. doi: (doi number)

Example:

[1] H.-B. QU, X. CHEN, S.-T. WANG, and M. YU, "Forward Affine Point Set Matching Under Variational Bayesian Framework," in *Acta Automatica Sinica*, 2015, pp.1482–1494, doi: 10.1016/s1874-1029(15)30001-x.

ISBN Reference Format:

[1] AuthorA, FirstInitial. SecondInitial. LastName, and AuthorB, FirstInitial. SecondInitial. LastName, "Title of chapter," in Title of Book, 3rd ed. City of Publisher (only U.S. State), Country: Abbreviated, Year of Publication, ch. 3, sec. 4, pp. page-page.

Example:

[1] R. Chellappa and S. Theodoridis, "Signal processing for massive MIMO communications," in *Academic Press Library in Signal Processing, Volume 7*, London, England: Academic Press, 2018, ch.8, pp.367- 401.

2.4 Summary

This chapter reviews and compares the existing citation generator tools, the strengths and limitations of the tools, the DOI and ISBN metadata retrieval, and the reference style format. A review of the tools under consideration reveals a number of factors that may be considered to be beneficial, alongside a number of factors that may be considered to be detrimental. The former include, but are not limited to, simplicity, cost barriers, the level of DOI/ISBN support, and the capacity for real-time updates. The latter comprises a number of disadvantages,

including, but not limited to, the complexity of the processes and the paucity of support for DOI and ISBN. Due to all the research, this project aims to provide an accurate automated citation generator to format APA, Harvard, and IEEE styles. Secondly, this project will also use trusted metadata sources, such as the DOI Foundation and Open Library, to retrieve the information. The citation generator will also support manual input in case the information is incomplete. Furthermore, this project will also create a high-speed DOI and ISBN citation converter with accuracy and reliability for users. The next chapter presents the system design, outlining how these findings inform the proposed architecture and implementation.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter discusses the methodology that outlines the approach used to develop the DOI and ISBN citation converter and ensure effective, accurate and usable. The methodology for this project follows a sequential workflow, beginning with planning and progressing through system analysis, implementation, testing and a final review phase. Each phase is important to ensure that the system is efficient, reliable, accurate, and user-friendly. This chapter will state the details about requirements and specifications for the project, the API integration used, and the Iterative methodology used to ensure continuous improvements and testing during development.

3.2 Requirement/ Specification/ Standards

The system requires a combination of front-end interfaces, backend processing services, and third-party APIs for metadata retrieval to develop a user-friendly website to convert the DOI and the ISBN into different reference styles.

The hardware requirements for this project are a minimum of 8 GB of RAM is recommended to support backend processing. The higher RAM capacity and speed lead to improved computer performance, and testing tools will work smoothly. Secondly, more RAM allows the system to handle more data and prevent slowdowns and lag while running multiple programs. The software requirements are important; the core programming language used for the project's backend is Python 3.13.

1. Pydantic and dataclasses are used for data validation, which enforces the type of data and ensures the input received via API requests is correctly structured to prevent missing data.
2. MyPy is a static type checker for Python; it can detect potential type-related errors before runtime and improve code reliability and maintainability.
3. DuckDB databases are used to store the citation history.

4. The third-party API calls the DOI Foundation to retrieve the journal information, for example, journal title, author, and publication data based on the DOI input. The Open Library for fetching book data, for example, book title, authors, and publisher, using the ISBN.
5. FastAPI was utilised in this project. It is a modern and high-performance web framework for constructing RESTful APIs using Python.

3.3 Software Development Methodology: Iterative Methodology

The iterative methodology is a software development methodology to develop a service in phases, and each phase builds on the previous phase. The phase involves planning and requirements, design, implementation, testing, and review, which is used to gain feedback and evaluation for refinement purposes. This methodology's advantage is that developers can test their ideas early and frequently to save time, reduce the risk of major flaws, and optimise the resources. (indeed, 2024)

The iterative process is where the developer starts to define a basic version of the service, defines the service's requirements, designs the service based on the requirements, develops some tests to test the service and then reviews the service's feedback. This process breaks a project into a few modules to let the developer develop the service step-by-step to improve the service. An iterative process begins with the first iteration, where the developer creates a simple planning task that fulfils the project requirements. Secondly, the developer should design the project interface or anything necessary and implement it. Then, conducting a test to ensure the project is well done and has no errors. After that, through the feedback, the developer can decide to refine the project or improve the project to meet the project goals.

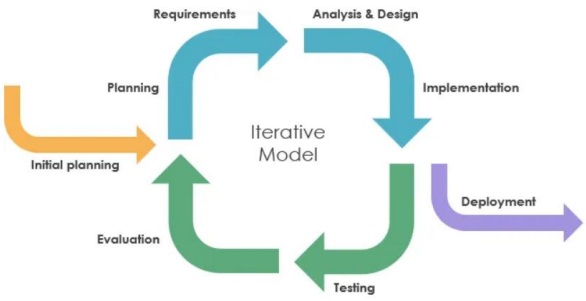


Figure 3.1: Iterative Methodology (Visual Paradigm, 2024)

3.3.1 Planning and Requirement Phase

The initial phase should start in the planning and requirements phase. The planning phase laid the foundation for the project's success. This phase should determine the project problem statement, project objectives, project scope, select suitable technologies, and set deliverables. The planning phase also identified the target users, requirements of the system, and key milestones based on the requirements. This phase will involve the project schedule to estimate the overall project duration and ensure the project is completed on time. The scope of the project was limited to creating a lightweight web-based citation converter that takes DOI or ISBN input, automatically retrieves metadata from reliable sources, and outputs citations in APA, Harvard, or IEEE format. Functionally, the system needed to allow users to input a DOI or ISBN, and then the system would automatically retrieve metadata and display formatted citations based on user-selected styles. Non-functional requirements included accuracy, fast response time, scalability, security, and maintainability. The flow of the application is for users to input the DOI and ISBN in the column and select the citation style needed. The system will validate the input and retrieve metadata from the CrossRef for DOI and the Open Library for ISBN, and then format the citation into the style that users choose. Other than that, users are allowed to manually input data if the metadata is not complete or incorrect.

The selection of a suitable technology stack constituted a pivotal element in the planning phase. Following a thorough evaluation of the available options, FastAPI was selected for the backend due to its speed, asynchronous capabilities, and automatic OpenAPI documentation compatibility. The httpx library was selected for use with HTTP requests due to its asynchronous request-handling capabilities, which facilitate expeditious metadata retrieval without compromising server performance. Pydantic was employed for the purposes of input validation and schema enforcement, thereby ensuring data consistency throughout the system. In order to enhance compatibility and reduce load times, the frontend was built with simple, lightweight technologies such as HTML5, CSS, and JavaScript.

3.3.2 Analysis and Design Phase

In this analysis and design phase, the system architecture was designed to represent the detailed information based on the project requirements. This system architecture provides a better understanding of the software and hardware requirements and their function. Firstly, the application block design is designed with a user-friendly interface and uses HTML language in VSCode. The comparison with existing citation converter tools analysis will be conducted to show the strengths and limitations of each website. After that, I can take the strengths and avoid the limitations for my project. The backend structure will use FastAPI to manage the user's request, the data classes, and Pydantic for validation metadata. Additionally, the reference format and sample are used to test the function and the output.

3.3.3 Implementation Phase

After the design phase, the implementation phase should be implemented using the backend services to develop a system using the Python language and FastAPI for this project. The CrossRef and ISBN will retrieve metadata using the DOI Foundation and Open Library because they are stable and widely used. The information that is retrieved will be converted to JSON format and will pick the necessary information to convert into different citation formats. Pydantic is implemented to validate the input, and MyPy uses static type checking to prevent runtime errors. Two Pydantic models were created: one to define the request structure (CitationRequest) and one to define the response structure (CitationResponse). The /cite endpoint is responsible for receiving user requests, assessing the necessity of retrieving metadata from DOI or ISBN sources, preparing the citation using the selected style, and returning the formatted citation. The httpx component facilitates asynchronous HTTP requests, thereby ensuring non-blocking performance and enhanced response times, even in scenarios where external APIs exhibit latency. With each iteration, the functionality is extended to avoid becoming too complex and confusing the system. For example, the second iteration adds new citation styles, and the third iteration implements a database.

The backend handles incorrect scenarios, such as an error DOI or ISBN, or a missing metadata field. Specific exceptions are reported using suitable

HTTP status codes and user-friendly error messages. The citation formatting logic adheres to APA 7th edition norms, Harvard reference standards, and IEEE regulations. It is important to note that particular care was taken to ensure that fields such as volume, issue, page numbers, and publication dates are properly handled, and that citations are generated in grammatically and stylistically accurate formats.

The frontend implementation placed significant emphasis on simplicity and practicality. The decision was taken to employ a rudimentary design to ensure that users would be able to input identifiers, select styles, and produce citations with minimal difficulty. The interface is composed of three distinct components: an entry field for a DOI or ISBN, a dropdown menu that facilitates the selection of a citation style, a button that initiates the process of generating a citation, and an example of a DOI and an ISBN. Upon the backend's return of the structured citation, it is presented within a read-only text field, thereby enabling users to seamlessly duplicate and paste the result. Frontend logic was handled using JavaScript, thereby circumventing the introduction of superfluous dependencies that could impede the processes of maintenance or deployment.

3.3.4 Testing Phase

The necessity for testing arises after the application implementation, in such a manner that it tests the coverage and functionality of the application. Testing can be a stabilising factor of the system, along with ensuring correctness in the citation format. In this project, unit testing would be done as a mode of testing the individual functions, like DOI/ISBN validation, citation formatting, and some functions of parsing the data from FastAPI. The test case creates a new test case to check for the precise response to a particular set of inputs. Aside from that, the test coverage for Python was performed to measure the amount of code exercised by tests. It would create a report on coverage showing which parts of the code base are covered by tests and which ones are not. (Keployio, 2024)

3.3.5 Review Phase

In the review phase, feedback from the test cases was collected, and code reviews were performed for continuous improvement. A meticulous review of

the citation formats was undertaken in order to ensure full compliance with the APA, Harvard, and IEEE standards, which were most recently updated. Following the testing phase, any bugs detected should be addressed, and each feature enhanced based on the results. By fixing the bugs, the system should run without error, or it may cause a domino effect, where the bugs are fixed, but an unexpected event occurs, and then causes huge issues for the system. The feedback will facilitate continuous improvement and ensure the application runs smoothly and fulfills the requirements. Each new iteration will enhance the application's system based on feedback to improve the user experience and optimise citation formatting logic.

3.4 Development Tools

This section will outline the tools used, the programming language used to write the program, the web framework, databases, and libraries used to retrieve metadata.

3.4.1 Visual Studio Code (VSCode)

Visual Studio Code (VS Code) is an open-source code editor used to write and debug Python code. It is lightweight and free to download from the website. It offers various features, including the AI copilot features for all users. It supports a wide range of extensions, the Pydantic, MyPy, and Git integration can be installed and used in VS Code. By using these tools in VS Code, the system can develop smoothly and efficiently.

3.4.2 FastAPI

FastAPI is a modern Python web framework used for building efficient and fast APIs. It can handle both synchronous and asynchronous operations, perform automatic documentation, and has built-in support for input data validation using Pydantic. It can help to create a readable, clean, and high-performance endpoint for this citation formatting.

3.4.3 Python

The main programming language used to develop the backend of the citation converter for this project is Python. Python is simple, readable, and can be used

for FastAPI. Its rich ecosystem of libraries can develop a system for rapid development and integration with APIs.

3.4.4 MyPy

MyPy is a static type checker that can check the input for Python. Implementing it can ensure the code follows the type annotations and help to catch type-related errors during development. the project will improve code reliability and maintainability by implementing MyPy.

3.4.5 Pydantic

Pydantic is a Python-based data validation and settings management package. It will validate the input data type based on type annotations. By implementing Pydantic, it can ensure the metadata retrieved from APIs fulfills expected formats and structures. It provides benefits to improve the accuracy and reliability of the citation data used in the converter.

3.4.6 DuckDB

DuckDB is an embedded analytical database system designed for online analytical processing (OLAP). DuckDB offers various features, including the rapid execution of SQL queries on large data sets, operating entirely within a process without requiring a separate server, and providing fast data transformation and aggregation. Additionally, DuckDB also helps manage and analyse data directly from local storage or memory, making it an ideal tool for data scientists and developers for exploratory data analysis.

3.4.7 HTML (HyperText Markup Language)

HTML (Hypertext Markup Language) is a standard markup language employed for the structuring of web content. The definition encompasses elements that serve as foundational components of web pages, including headings, paragraphs, forms, and buttons. In this project, HTML was employed to generate the input fields, dropdown menus, and display spaces for citation. The primary function of this process is to ensure that the user interface is well-organised, easily accessible, and semantically correct.

3.4.8 Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) are a language that is used to control the appearance and layout of HTML elements. In this project, it was employed to style the web interface, thereby enhancing both the visual appearance and the user experience. CSS is a set of style rules that allow for the flexible modification of visual elements on a webpage, including colors, fonts, margins, and padding. These rules can be applied to various screen sizes, ensuring that the webpage adapts accordingly, a process referred to as responsive design. The project is rendered with a clean, modern, and professional appearance that enhances usability by utilizing CSS.

3.4.9 JavaScript

JavaScript is a high-level programming language that facilitates the construction of dynamic and interactive web pages. In this project, the JavaScript programming language employs the Fetch API to facilitate asynchronous communication with the backend server. The system is designed to validate user input, initiate requests to the FastAPI backend, and subsequently update the page with the generated citations. The employment of JavaScript facilitates a seamless and responsive user experience, obviating the necessity for page reloads. This is of paramount importance to the real-time interaction and functionality of the application.

3.4.10 Git

Git is a version control system that can be used to manage source code changes. Git offers various features, such as tracking modifications, maintaining the history of the development process, and collaborating or sharing with others on GitHub. Additionally, Git helps to manage different versions of the project, and users can see the difference between versions using the rollback function.

3.4.11 Digital Ocean

Digital Ocean is a cloud infrastructure provider that provides scalable computing platforms and services. DigitalOcean provides a variety of features, including virtual private servers (Droplets), managed databases, Kubernetes clusters, and scalable storage. It also simplifies cloud computing for developers

by providing a user-friendly control panel, thorough documentation, and a predictable pricing scheme, allowing for quick web application development and management.

3.4.12 DuckDNS

DuckDNS is a free dynamic DNS service that allows you to assign a fixed domain name to a dynamic or changeable IP address. DuckDNS provides several features, such as a simple setup process, automatic IP address updates using an easy HTTP API, and support for multiple domain names. DuckDNS also allows users to operate personal servers, websites, or remote access tools from their home network without incurring the cost of a static IP address from their internet provider.

3.5 Project Plan

3.5.1 Work Breakdown Structure (WBS)

1.0 Initial Planning
- 1.1 Project Planning
  - 1.1.1 Define Project Problem Statement
  - 1.1.2 Define Project Aims and Objectives
  - 1.1.3 Define Project Scope and Limitations
- 1.2 Literature Review
  - 1.2.1 Comparison of Existing Converter Tools
  - 1.2.2 Define DOI and ISBN Metadata Extraction
  - 1.2.3 Define Citation Formatting Rules

2.0 Methodology and Work Plan
- 2.1 Define Requirements
  - 2.1.1 Define the hardware requirements of the system
  - 2.1.2 Define the software requirements of the system
- 2.2 Define the methodology for software development
  - 2.2.1 Develop Work Plan
  - 2.2.2 Define Development Tools

3.0 System Development
- 3.1 Develop Python on VS Code

3.1.1 Develop FastAPI

3.1.2 Develop Frontend using HTML, CSS, JavaScript

3.2 Adding features for the application

3.3 Improve the User-Interface (UI) and system

4.0 Testing Phase

4.1 Preparation for Test Case

4.1.1 Test Case for Valid Input

4.1.2 Test Case for Invalid Input

4.1.3 Test Case for Internet

4.1.4 Test Case for Frontend Function

4.2 Test with PyTest

4.3 Record and summarize results

5.0 Closure

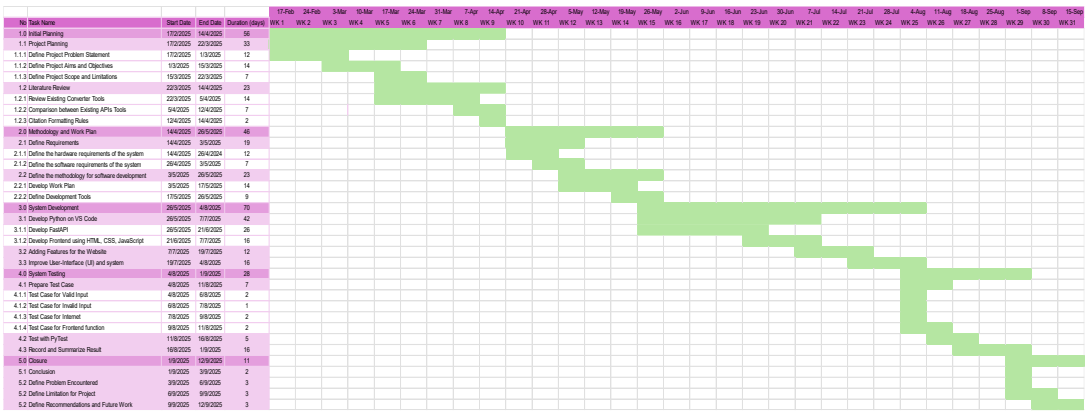
5.1 Conclusion

5.2 Define Problem Encountered

5.3 Define Limitations for the Project

5.4 Define Recommendations and Future Work

3.5.2 Gantt Chart



No	Task Name	Start Date	End Date	Duration (days)
1.0	Initial Planning	17/2/2025	14/4/2025	56
1.1	Project Planning	17/2/2025	22/3/2025	33
1.1.1	Define Project Problem Statement	17/2/2025	1/3/2025	12
1.1.2	Define Project Aims and Objectives	1/3/2025	15/3/2025	14
1.1.3	Define Project Scope and Limitations	15/3/2025	22/3/2025	7
1.2	Literature Review	22/3/2025	14/4/2025	23
1.2.1	Review Existing Converter Tools	22/3/2025	5/4/2025	14
1.2.2	Comparison between Existing APIs Tools	5/4/2025	12/4/2025	7
1.2.3	Citation Formatting Rules	12/4/2025	14/4/2025	2
2.0	Methodology and Work Plan	14/4/2025	29/5/2025	46
2.1	Define Requirements	14/4/2025	3/5/2025	19
2.1.1	Define the hardware requirements of the system	14/4/2025	26/4/2024	12
2.1.2	Define the software requirements of the system	26/4/2025	3/5/2025	7
2.2	Define the methodology for software development	3/5/2025	26/5/2025	23
2.2.1	Develop Work Plan	3/5/2025	17/5/2025	14
2.2.2	Define Development Tools	17/5/2025	26/5/2025	9
3.0	System Development	29/5/2025	4/8/2025	70
3.1	Develop Python on VS Code	29/5/2025	7/7/2025	42
3.1.1	Develop FastAPI	29/5/2025	21/6/2025	26
3.1.2	Develop Frontend using HTML, CSS, JavaScript	21/6/2025	7/7/2025	16
3.2	Adding Features for the Website	7/7/2025	19/7/2025	12
3.3	Improve User-Interface (UI) and system	19/7/2025	4/8/2025	16
4.0	System Testing	4/8/2025	1/9/2025	28
4.1	Prepare Test Case	4/8/2025	11/8/2025	7
4.1.1	Test Case for Valid Input	4/8/2025	6/8/2025	2
4.1.2	Test Case for Invalid Input	6/8/2025	7/8/2025	1
4.1.3	Test Case for Internet	7/8/2025	9/8/2025	2
4.1.4	Test Case for Frontend function	9/8/2025	11/8/2025	2
4.2	Test with PyTest	11/8/2025	16/8/2025	5
4.3	Record and Summarize Result	16/8/2025	1/9/2025	16
5.0	Closure	1/9/2025	12/9/2025	11
5.1	Conclusion	1/9/2025	3/9/2025	2
5.2	Define Problem Encountered	3/9/2025	6/9/2025	3
5.2	Define Limitation for Project	6/9/2025	9/9/2025	3
5.2	Define Recommendations and Future Work	9/9/2025	12/9/2025	3

Figure 3.2: Gantt Chart

3.6 Summary

This chapter determines the hardware and software requirements and the software development methodology used in the project. This project used a modern framework, FastAPI, to build RESTful APIs with Python, Pydantic and MyPy to validate the data. The system is being developed to be scalable, reliable,

and efficient. The iterative methodology was used to ensure the development process was step-by-step and remained adaptive throughout its lifecycle.

CHAPTER 4

PROJECT SPECIFICATIONS

4.1 Introduction

The project specification provides a comprehensive set of instructions detailing the objectives of the DOI and ISBN citation converter system and the actions required to achieve its objectives. The purpose of specifications is to define a clear scope, set quantifiable expectations, and offer a solid foundation for the design, development, and testing phases. The specification guarantees that both functional and non-functional requirements are explicitly established, thus avoiding ambiguity and allowing for systematic evaluation of whether the system meets its objectives. By defining these needs, the project may be developed in an organised manner while remaining closely linked with the expectations of its end customers. This chapter also delineates the system's limitations, the prototype that will be produced, and the constraints of the system in its present implementation.

4.2 End Users

The end users of this project are primarily individuals engaged in academic and research activities. These include:

- University students who are regularly engaged in the preparation of assignments, projects, and theses should employ correctly formatted references to circumvent the occurrence of plagiarism and to ensure the maintenance of academic integrity.

- In academic writing, researchers are expected to acknowledge a range of sources when producing articles for journals or conference papers.

- Educators who guide students through the process of research writing require reliable citation tools to facilitate effective demonstration and academic support.

Consequently, the system has been developed with a focus on usability, minimising the complexity of citation formatting while still enabling the management of missing or inconsistent metadata.

4.3 Requirement Specification

The functional requirements define the system's needs, and the non-functional requirements define system qualities, performance, and constraints.

4.3.1 Functional Requirements

The functional requirements specify the features and behaviours that the system must provide. These include the ability to accept DOI or ISBN as input, retrieve metadata from external APIs, such as CrossRef and Open Library, and generate citations in APA, Harvard, and IEEE forms. The system allows users to download or copy citations and upload them in bulk via .txt files. Another important functional requirement is metadata caching in DuckDB, which ensures that repeated requests are delivered effectively and reduces API calls.

Table 4.1: Functional Requirements

ID	Functional Requirement
FR001	The system shall allow users to input a DOI and retrieve metadata from the CrossRef API.
FR002	The system shall allow users to input an ISBN and retrieve metadata from the Open Library API.
FR003	The system shall format metadata into APA style according to the latest APA referencing guidelines.
FR004	The system shall format metadata into Harvard style according to standard Harvard referencing rules.
FR005	The system shall format metadata into IEEE style according to IEEE referencing rules.
FR006	The system shall store generated citations in a DuckDB database for future retrieval and reuse.
FR007	The system shall validate user inputs and return error messages for invalid DOI/ISBN values.

FR008	The system shall provide a download or copy function, allowing users to incorporate the citation into their documents.
FR009	The system shall provide a basic user interface for inputting DOIs/ISBNS, viewing results, and downloading citations.
FR010	The system shall allow exporting citations in plain text format.
FR011	The system shall allow exporting citations to BibTeX format.
FR012	The system shall provide a clear form button to let users clear the input.
FR013	The system shall provide a simple FAQ section for users.

4.3.2 Non-Functional Requirements

Non-functional requirements describe system qualities such as performance, reliability, scalability, usability, and security.

Table 4.2: Non-Functional Requirements

ID	Non-Functional Requirement	Category	Priority
NFR001	The system shall return results within 3–5 seconds of DOI/ISBN input.	Performance	High
NFR002	The system shall have a clean and user-friendly interface that requires minimal training.	Usability	High
NFR003	The system shall generate citations with at least 95% accuracy according to citation guidelines.	Reliability	High
NFR004	The system shall be able to handle simultaneous requests without significant performance degradation.	Scalability	Medium
NFR005	The system shall support the addition of new citation styles with minimal changes in code.	Scalability	Medium

NFR006	The system shall ensure that data retrieved from APIs and stored in the database is secure and protected.	Security	High
NFR007	The system shall provide error messages and fallback options when API services are unavailable.	Reliability	High
NFR008	The system shall maintain compatibility with modern browsers (Chrome, Edge, Firefox).	Portability	High
NFR009	The system shall be documented and version-controlled with GitHub for future maintainability.	Maintainability	High
NFR0010	The system shall comply with academic citation guidelines and be easily extendable to support future updates in APA/Harvard/IEEE style formats.	Compliance	High

4.4 Prototype

The prototype developed for this project exemplifies the capacity for automating citation formatting with DOI and ISBN identifiers. The prototype includes the following key features: DOI/ISBN input, API metadata retrieval, citation formatting in APA/Harvard/IEEE style, and the FAQs section. The prototype is equipped with a rudimentary graphical user interface, facilitating interaction with the device. Despite its current functional limitations, the prototype serves as a robust foundation for future development, encompassing the capacity to export citations in various formats, enhanced search functionalities, and support for offline utilisation.

4.5 Summary

In summary, this chapter has outlined the functional and non-functional requirements that serve as the foundation for the DOI and ISBN citation converter system. The system's functionality is intentionally restricted to DOI

and ISBN inputs, the generation of citations in APA, Harvard, and IEEE styles, and the retrieval of metadata via APIs such as CrossRef and Open Library. The specification of these standards enables the structured development and measurement of the system, thereby ensuring that the end product satisfies user expectations while remaining within the project's schedule.

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

System design serves as the bridge between the abstract requirements of the system and its concrete implementation. It is imperative to establish a design that defines the architecture, data flow, storage models, and operational procedures before writing code. The design of this citation converter ensures that it can meet functional requirements, such as metadata retrieval and formatting, and non-functional requirements, including usability, reliability, and performance. An inadequately designed system is likely to exhibit inefficiencies, data discrepancies, and issues sustaining or extending features in the future. Conversely, a meticulously formulated design facilitates modular development, seamless integration of external APIs, and incremental enhancements through Agile methodology. Moreover, communication is another critical component of design. The provision of detailed descriptions and illustrations is instrumental in facilitating comprehension among stakeholders who do not possess a background in programming. These materials enable these individuals to grasp the conceptual underpinnings of the system. In this chapter, the design is described through system architecture to show the layered structure of the system, a data flow diagram that illustrates how information moves through the system, a database schema to capture the relationships between stored data, and system flow designs to highlight decision-making processes within the backend. These artifacts delineate the system's micro- and macro-level structures, and it is important to note that they also function as a guide for testing in subsequent phases. This is because test cases can be methodically developed because every flow and component is documented.

5.2 System Architecture Design

The design of the proposed citation converter system incorporates a three-tier architecture and integration with external metadata APIs. This architecture is a well-established design pattern that separates concerns into distinct layers,

making the system scalable, maintainable, and robust. Each layer facilitates the assurance that alterations made to a specific layer, such as database schema modifications, do not disrupt other layers, including the user interface. This architecture facilitates rapid reaction times and effective data retrieval by utilising contemporary frameworks and asynchronous technologies, two essential features for a citation creation system that communicates with external services. At a high level, the system is divided into three main layers: the presentation layer for the frontend, the application layer for the backend, and the data layer for the database, while also maintaining a critical dependency on external APIs, which are CrossRef and Open Library.

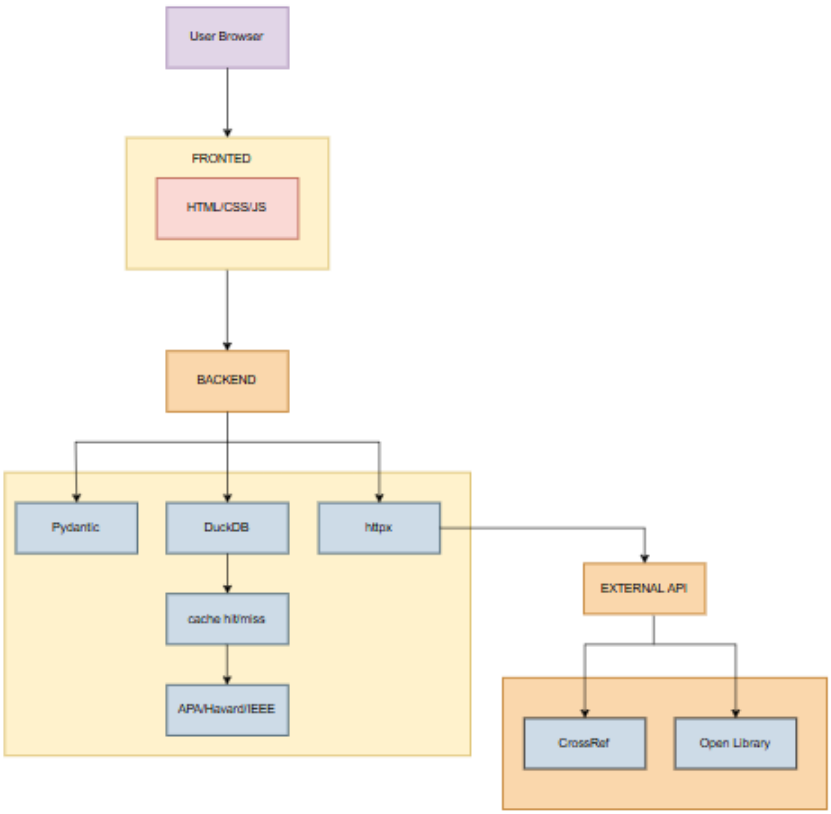


Figure 5.1: System Architecture Design Diagram

i. Presentation Layer

The presentation layer is the entry point of the system where users interact with the application. In this project, the frontend is developed using HTML, CSS, and JavaScript, providing a simple yet effective user interface. Firstly, it provides a single input that allows users to input the DOI or ISBN and also

an upload file system to upload the DOI file or ISBN file without inputting the reference number one by one. The users can choose the citation format they prefer from the style selector dropdown. After that, basic input checks are implemented in JavaScript to prevent the submission of invalid identifiers, and this can also help reduce unnecessary API calls to the backend. Once the user submits the form, the front-end sends a POST request to the backend's /cite endpoint. The request payload contains two fields: the identifier, which is a DOI or an ISBN, and the desired style. After processing by the backend, the frontend receives a JSON response containing the formatted citation string and supporting metadata. This is displayed neatly to the user, with options such as copy-to-clipboard and download as BibTeX. If an invalid DOI/ISBN is detected or the backend returns an error, for example, an ISBN is unavailable, the frontend displays a clear error message to guide the user. The design decision to make the frontend lightweight has been shown to result in faster load times, reduced complexity, and simpler integration with the backend REST API.

ii. Application Layer

The backend is the system's primary processing component, which is responsible for coordinating formatting, persistence, metadata retrieval, and validation. This system utilises the FastAPI framework, a modern Python framework known for its asynchronous capabilities, speed, and automatic API documentation production, in its construction. The backend is deployed using Uvicorn, an ASGI server designed to manage several concurrent requests, is used to deploy the backend. The incoming requests are validated against predefined Pydantic schemas to ensure that identifiers are properly structured and that the citation style field only accepts valid values. On the other side, the invalid requests are immediately rejected with a clear error response, reducing unnecessary backend processing. If the requested citation is not found in the cache, the backend makes an asynchronous API call using httpx. For DOIs, the request is directed to CrossRef; for ISBNs, the request goes to Open Library. This asynchronous function can handle multiple lookups to be processed simultaneously to improve the system's

responsiveness under load, so the system can retrieve metadata as soon as possible. Once metadata is retrieved, the backend applies the formatting rules based on the chosen style, such as APA, Harvard, and IEEE style. The engine will also ensure consistent and accurate formatting even when metadata fields are optional or missing. Before making external API calls, the backend first checks the local database for previously generated citations. If found, the cached result is returned instantly; if not, the metadata is fetched from the external API, formatted, and then stored in the cache for future requests and reducing calls to external APIs. By combining validation, caching, external integration, and formatting into a single backend service, the system achieves a clean separation of logic, enabling easier debugging and testing.

iii. Data Layer

DuckDB is used as the system's primary database in the data layer. DuckDB is an embedded analytical database designed to operate within the same process as the application. It is a simple database system, as no separate database server is needed; the database runs as part of the Python process, reducing setup complexity and deployment overhead. The data is stored in columnar format, which is efficient for analytical queries and supports fast lookups. After calling the external API, the metadata will be stored in the database to reduce call time and speed up the process. The system integrates with two external APIs to source bibliographic metadata: CrossRef, which provides metadata for DOIs, and Open Library, which provides metadata for ISBNs. When metadata cannot be located locally in DuckDB, these APIs are searched in real time. Even in the face of API volatility, the system manages timeouts, retries, and structured JSON parsing to guarantee seamless functioning. When metadata cannot be located locally in DuckDB, these APIs are searched in real time. Even in the face of API volatility, the system manages timeouts, retries, and structured JSON parsing to guarantee seamless functioning.

5.3 Database Design

Citation metadata obtained from Open Library for references based on ISBNs and Crossref for references based on DOIs are stored and managed by the database for this project. The system's storage engine, DuckDB, is characterised by its lightweight, integrated, and tailored architecture for analytical queries, ensuring effectiveness and adaptability. The database's primary function is to serve as a local metadata cache, thereby minimising the need for repetitive API calls and facilitating expeditious offline lookups. The metadata records for both Digital Object Identifier (DOI) and International Standard Book Number (ISBN) citations are stored in the database's central table named metadata_cache. Each record is associated with a JSON object, which contains the raw metadata response and is identified by a unique key (the DOI or ISBN value). This JSON format allows flexibility because citation metadata often varies in structure depending on the source. For example, Crossref metadata typically contains nested fields such as authors, publisher, references, and issue details, while Open Library metadata may emphasise ISBN, editions, and publication details. The database has the capacity to support both approaches by storing the response in JSON, thereby eliminating the necessity for frequent schema modifications. The following ERD (Entity Relationship Diagram) illustrates the conceptual design of the normalized schema.

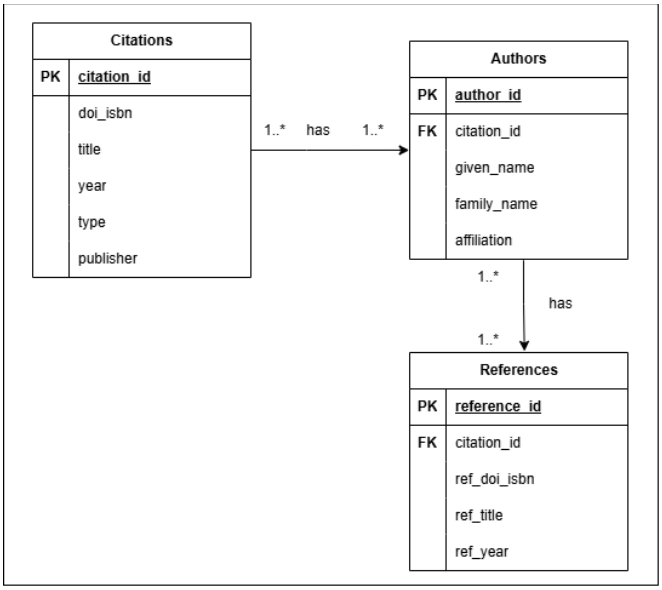


Figure 5.2: Entity Relationship Diagram for the databases

Table 5.1: Data dictionary

Column Name	Type	Key	Description
Id/key	String	Primary	DOI or ISBN. Example: 10.1016/S1874-1029(15)30001-X.
metadata	JSON	-	Full raw metadata response from Crossref/Open Library (authors, title, year, publisher, references, etc.).

5.4 API Routes Design

FastAPI uses the Swagger API Documentation tool to create the design for every route. The route will contain the information about the parameters used to request and provide the response code and values.

5.4.1 DOI

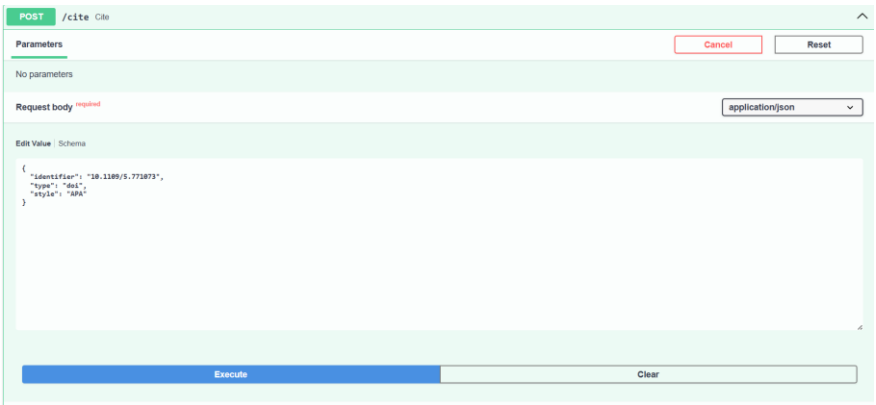


Figure 5.3: The design of the DOI route.

Figure 5.4: The response code and body.

5.4.2 ISBN

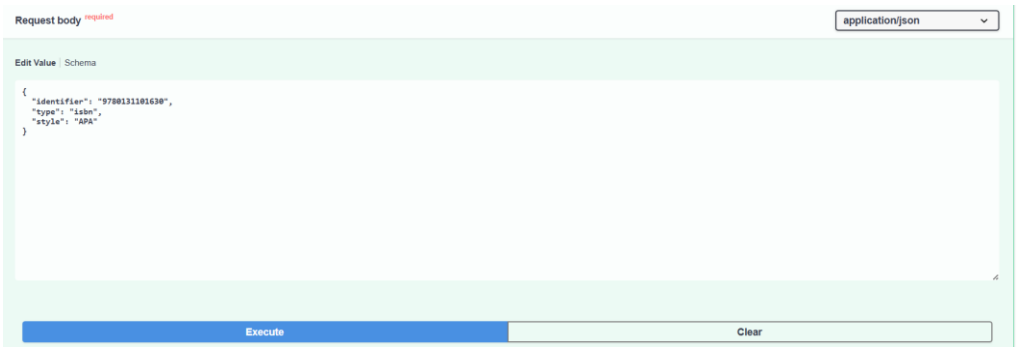


Figure 5.5: The design of the ISBN route.

Figure 5.6: The response code and body.

5.4.3 Error return

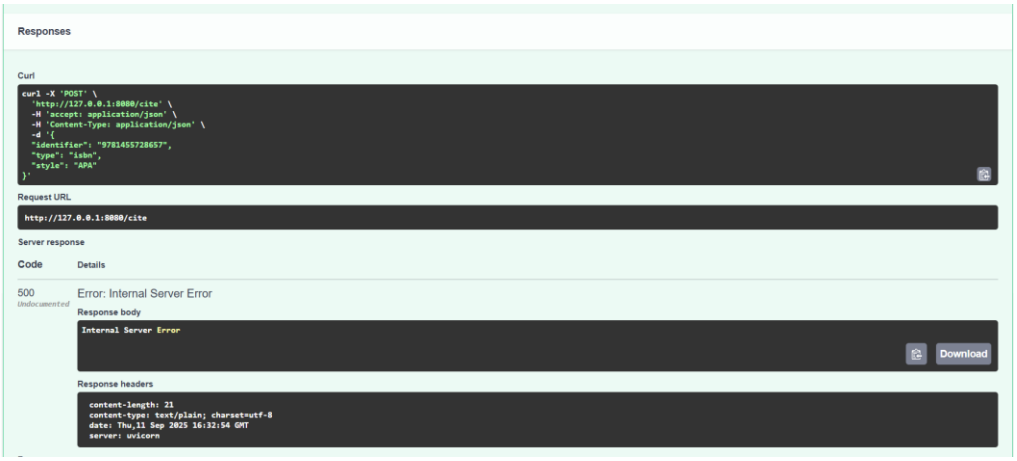


Figure 5.7: The response code and body when error.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

The implementation phase of this project involved converting the designs generated in Chapter 5 into a functional system. Several tools, programming languages, and frameworks were selected carefully to ensure that the system can satisfy both functional and non-functional requirements. The backend was built with Python 3.10 because Python has a wide ecosystem of libraries and is highly readable, which will make the system easier to maintain and extend in the future. FastAPI, a modern Python framework built for rapid and effective API development, was chosen as the web framework. FastAPI was selected due to its proven capabilities for asynchronous programming, enabling the system to process numerous requests concurrently. This feature was important because the system relies heavily on real-time calls to external APIs such as CrossRef and Open Library, which can occasionally result in latency. Asynchronous handling allows the backend to continue processing incoming requests while waiting for external API responses, resulting in greatly improved overall responsiveness.

The DuckDB database was used for the data layer because DuckDB is distinct from conventional databases such as MySQL and PostgreSQL in that it operates as an embedded, in-process database, whereas the latter run as separate services. This decision reduced system complexity by removing the need for an external database server. DuckDB is a single-file system, then optimised for analytical queries, making it ideal for caching bibliographic metadata and quickly retrieving previously retrieved results. This system uses DuckDB to store identifiers (DOIs or ISBNs), associated metadata in JSON, and request history. By caching this data, the number of repeated external API calls is reduced, enhancing efficiency while also protecting against rate limitations from the external metadata providers.

The front end of the website was built using HTML5, CSS3, and JavaScript to provide a lightweight and accessible user interface. All major browsers support these technologies, ensuring that users can access the system

without needing to install additional dependencies. The user interface features text input areas for entering DOIs or ISBNs, options for uploading .txt files or BibTeX files, and a dropdown menu for selecting citation styles. JavaScript communicates with the backend through asynchronous fetch calls, which allow the interface to remain responsive while waiting for results.

6.2 Backend Implementation

The backend implementation is in app.py, which defines the FastAPI application and implements all of its main capabilities. The backend is the system's central processing unit, which connects the user interface, database, and external metadata suppliers. The FastAPI framework provides a robust foundation for designing routes, managing requests, and returning structured results in JSON format. Each route corresponds to a distinct system function, ensuring the backend remains modular and extensible. The root endpoint (/) was the first to be implemented, which serves the index.html file as the homepage. This seamlessly integrates the frontend and backend, enabling users to access the interface directly without needing a separate static file server, and also provides the HTML page through FastAPI, which is an effective way for the backend to ensure that the entire system can be deployed in a single package.

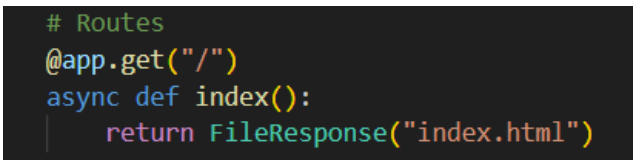


Figure 6.1: The route to call index.html

The most important endpoint is the /cite endpoint, which accepts a JSON payload as described by the CitationRequest Pydantic model. This payload comprises three essential fields: the identifier (either DOI or ISBN), the identification type, and the chosen citation style. The backend then validates this input through a series of steps. Firstly, it ensures that the citation style is one of the supported ones, such as APA, Harvard, and IEEE styles. If an unsupported style, such as "Chicago", is used, the system will instantly reject the request with a 400 Bad Request response. Next, the backend validates the

identifier's validity. The system returns a 404 Not Found error for known invalid identifiers such as "10.0000/invalid-doi" or "0000000000000." These validation processes help to prevent unnecessary calls to external APIs and provide users with instant feedback.

```python
@app.post("/cite", response_model=CitationResponse)
async def cite(request: CitationRequest):
    #Validate style
    if request.style not in VALID_STYLES:
        raise HTTPException(status_code=400, detail="Unsupported citation style")

    #Handle known invalid test identifiers
    if request.type == "isbn" and request.identifier == "0000000000000":
        raise HTTPException(status_code=404, detail="ISBN not found")
    if request.type == "doi" and request.identifier == "10.0000/invalid-doi":
        raise HTTPException(status_code=404, detail="DOI not found")

    if request.type == "doi":
        metadata = await fetch_doi_metadata(request.identifier)
        citation, in_text = format_doi_citation(metadata, request.style)
    elif request.type == "isbn":
        metadata = await fetch_isbn_metadata(request.identifier)
        citation, in_text = format_isbn_citation(metadata, request.style)
    else:
        raise HTTPException(status_code=400, detail="Invalid identifier type")

    return CitationResponse(citation=citation, in_text=in_text)
```

Figure 6.2: The cite endpoint for validation

If the input successfully passes validation, the backend determines the type of identification and retrieves the metadata. For DOIs, the system calls the CrossRef API, whereas for ISBNs, it calls the Open Library API. These calls are made asynchronously using httpx.AsyncClient, which ensures that the server remains responsive even during periods of high demand. When metadata is retrieved, the system sends it to the relevant formatting functions. The DOI will be sent to the format_doi_citation function, where the ISBN will be sent to the format_isbn_citation function. Then, the system produces the final formatted reference and in-text citation for the users. These are returned to the frontend as a CitationResponse, which is a structured JSON object that includes both outputs.

```python
def format_doi_citation(metadata, style, ref_index=1):
    authors = metadata.get("author", [])
    title = metadata.get("title", ["Unknown Title"])[0]
    journal = metadata.get("container-title", ["Unknown Journal"])[0]
    volume = metadata.get("volume", "")
    issue = metadata.get("issue", "")
    pages = metadata.get("page", "")
    year = metadata.get("issued", {}).get("date-parts", [[None]])[0][0]
    month = metadata.get("issued", {}).get("date-parts", [[None, None]])[0][1]
    doi = metadata.get("DOI", "")

    a = format_authors(authors, style, ieee=(style == "IEEE"))
    in_text = get_in_text_citation(authors, year)

    if style == "APA":
        citation = f"{a} ({year}). {title}. <em>{journal}</em>, {volume}({issue}), pp. {pages}. https://doi.org/{doi}"
    elif style == "Harvard":
        citation = f"{a}, {year}. {title}. <em>{journal}</em>, [e-journal] {volume}({issue}), pp. {pages}. https://doi.org/{doi}."
    elif style == "IEEE":
        citation = f"[{ref_index}] {a}, \u201c{title},\u201d {journal}, vol. {volume}, pp. {pages}, {month}. {year}. doi: {doi}."
    else:
        citation = "Error: Unsupported citation style"

    return citation, in_text
```

Figure 6.3: The format of a DOI citation.

```python
def format_isbn_citation(metadata, style, ref_index=1):
    authors = metadata.get("authors", [])
    title = metadata.get("title", "Unknown Title")
    edition = metadata.get("edition", "")
    publisher = metadata.get("publisher", "Unknown Publisher")
    year = metadata.get("publish_date", "Unknown Year")
    place = metadata.get("publish_places", [{}])[0].get("name", "Unknown Place") if metadata.get("publish_places") else "Unknown Place"

    a = format_authors(authors, style, ieee=(style == "IEEE"))
    in_text = get_in_text_citation(authors, year)

    if style == "APA":
        citation = f"{a} ({year}). <em>{title}</em>. {publisher}."
    elif style == "Harvard":
        edition_part = f" {edition}." if edition else ""
        citation = f"{a}, {year}. <em>{title}</em>.{edition_part} {place}: {publisher}."
    elif style == "IEEE":
        edition_part = f", {edition}" if edition else ""
        citation = f"[{ref_index}] {a}, \u201c{title}\u201d{edition_part}, {place}: {publisher}, {year}."
    else:
        citation = "Error: Unsupported citation style"

    return citation, in_text
```

Figure 6.4: The format of an ISBN citation.

Another feature is the /upload endpoint, which facilitates bulk citation generation. Users have the option to upload a .txt file containing multiple DOIs and ISBNs. The backend processes the file line by line, verifying the identifiers to determine if they are a DOI or ISBN, retrieving metadata from cache or other APIs, and generating citations. Each line of data is processed asynchronously, resulting in substantially faster bulk handling than sequential methods. The findings are returned as a collection of objects of the class "CitationResponse". This feature is useful for researchers who need to generate multiple citations at once.

```python
#Upload file
@app.post("/upload", response_model=List[CitationResponse])
async def upload_file(file: UploadFile = File(...), style: str = Form(...)):
    if not file.filename.endswith(".txt"):
        raise HTTPException(status_code=400, detail="Only .txt files are supported.")

    content = await file.read()
    lines = [line.strip() for line in content.decode().splitlines() if line.strip()]

    async def process_line(line, index):
        if line.startswith("10.") or "/" in line:
            metadata = await fetch_doi_metadata(line)
            citation, in_text = format_doi_citation(metadata, style, ref_index=index)
        elif line.isdigit():
            metadata = await fetch_isbn_metadata(line)
            citation, in_text = format_isbn_citation(metadata, style, ref_index=index)
        else:
            return CitationResponse(citation=f"Invalid identifier: {line}", in_text="")
        return CitationResponse(citation=citation, in_text=in_text)

    tasks = [process_line(line, i + 1) for i, line in enumerate(lines)]
    return await asyncio.gather(*tasks)
```

Figure 6.5: Upload file function

6.3 Data Validation

Data validation is a fundamental component of the system, ensuring that only properly structured requests are processed. In the app.py file, two Pydantic models are defined: CitationRequest and CitationResponse. The CitationRequest model specifies three fields: firstly, an identifier field containing the DOI or ISBN string; secondly, a type field indicating whether the identifier is a "doi" or "isbn" string; and thirdly, a style field which is a string representing the desired citation style. The FastAPI model automatically validates incoming JSON payloads before they reach the business logic. For instance, if a user submits a request that is missing the identifier field, FastAPI will immediately reject the request with a 422 Unprocessable Entity error, thereby specifying the absent field in the response. This eliminates the need for manual validation code and reduces the likelihood of runtime errors caused by malformed requests. The CitationResponse model defines the structure of responses returned through the /cite and /upload endpoints. Each response must comprise two fields, which include the citation with formatted reference and in-text citation, which is the recommended in-text citation. By utilizing this response model, the system ensures consistency in the output, irrespective of the input or style that is selected.

In addition to Pydantic validation, type checking is supplemented by a static Python type checker, MyPy. Backend functions are marked with type hints, such as async def fetch_doi_metadata(doi: str) -> dict:, which makes the code more self-documenting and ensures correctness during development. MyPy aims to analyse the codebase to ensure that routines produce the expected values and that the arguments correspond to their declared types. This proactive method of type checking improves code reliability and also helps identify potential issues early in the development process to avoid huge bugs. This layered approach to validation, which combines Pydantic runtime enforcement and MyPy static analysis, ensures system robustness, error minimisation, and clear error signals for users.

```
async def fetch_doi_metadata(doi: str) -> dict:
    cached = await fetch_metadata_from_cache(doi, "doi")
```

Figure 6.6: MyPy check DOI type

```
async def fetch_isbn_metadata(isbn: str) -> dict:
    cached = await fetch_metadata_from_cache(isbn, "isbn")
```

Figure 6.7: MyPy check ISBN type

6.4 Database Implementation

The system uses DuckDB as its persistence layer, which offers a lightweight and embedded alternative to server-based databases. After initiation, the backend establishes a connection to a database file (db/citations.db) to verify the existence of the metadata_cache table. The table under consideration comprises three columns: firstly, an identifier, the DOI or ISBN string that serves as the primary key; secondly, a type of input, either DOI or ISBN; and thirdly, metadata, which is a JSON object containing bibliographic metadata.

```
# Connect to or create DuckDB database
con = duckdb.connect(database="db/citations.db")
con.execute("""
CREATE TABLE IF NOT EXISTS metadata_cache (
    identifier TEXT PRIMARY KEY,
    type TEXT,
    metadata JSON
)
""")
```

Figure 6.8: Connect the DuckDB database

The database functions as a cache for metadata obtained from CrossRef and Open Library. When a request is received, the backend first queries DuckDB to check whether metadata for the given identifier has already been saved, and this process is implemented by the function fetch_metadata_from_cache. If a record is found, the metadata is returned immediately without calling the external API. If no record is found, the backend performs an external API call to retrieve the metadata and stores it in the cache using the store_metadata_in_cache function. This caching approach enhances performance by reducing latency, particularly in cases of repetitive queries, and prevents unnecessary calls to external APIs that may have usage limits.

```python
# Metadata fetchers with DuckDB cache
async def fetch_metadata_from_cache(identifier: str, id_type: str):
    result = con.execute("SELECT metadata FROM metadata_cache WHERE identifier = ? AND type = ?", (identifier, id_type)).fetchone()
    return json.loads(result[0]) if result else None

def store_metadata_in_cache(identifier: str, id_type: str, metadata: dict):
    con.execute("INSERT OR REPLACE INTO metadata_cache VALUES (?, ?, ?)", (identifier, id_type, metadata))
```

Figure 6.9: Metadata fetchers with DuckDB cache

DuckDB's columnar storage format makes it efficient for analytical queries, though in this project, it primarily supports simple lookups and inserts. The primary advantage of this approach is its simplicity: the database is integrated into the application process, thereby eliminating the need for a separate server. This design helps to reduce deployment complexity, making the system portable and easy to run on any machine with Python installed.

6.5 Frontend Implementation

The frontend is implemented in index.html, serving as the user interface that links users to the backend API. The layout of the website is simple and effective, and is divided into two major sections. The left section is a help box that explains DOI and ISBN identifiers, and the right section is the main input form for generating citations. The initial form consists of three sections; first, there is an input field for users to enter a DOI or ISBN. Next, there is a file upload option for .txt files so users can process multiple references simultaneously. Thirdly, a dropdown menu for users to select a citation style, such as APA, Harvard, and IEEE styles. Finally, the form includes a button that, when selected, commences

the process of producing the citation. The results are presented in tabular form, with two columns: the full formatted citation and the proposed in-text citation. The functionality of the programme is further enhanced by the presence of additional buttons located beneath the table. These buttons let users interact with the system by performing some actions, such as copying citations to the clipboard, downloading citations in .txt or .bib format, and clearing the form. The bottom also features a Frequently Asked Questions section for users.

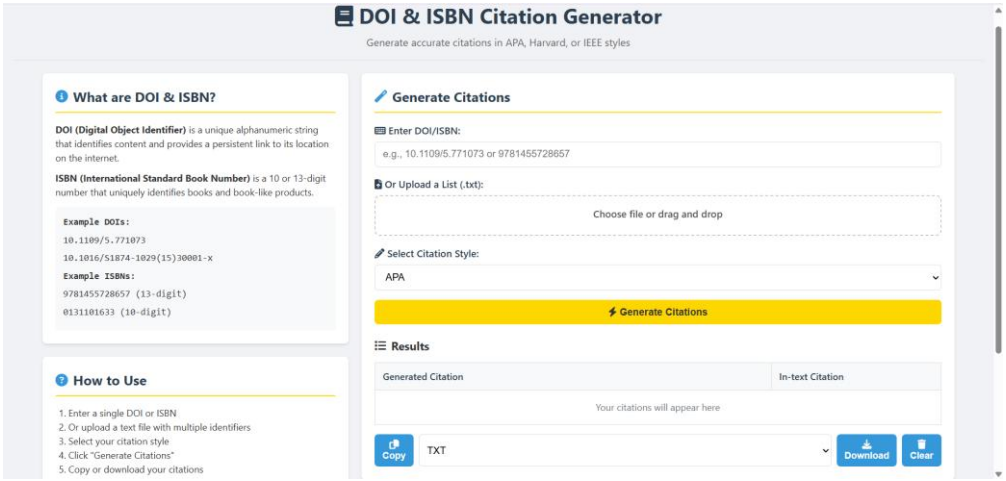


Figure 6.10: The Website Interface

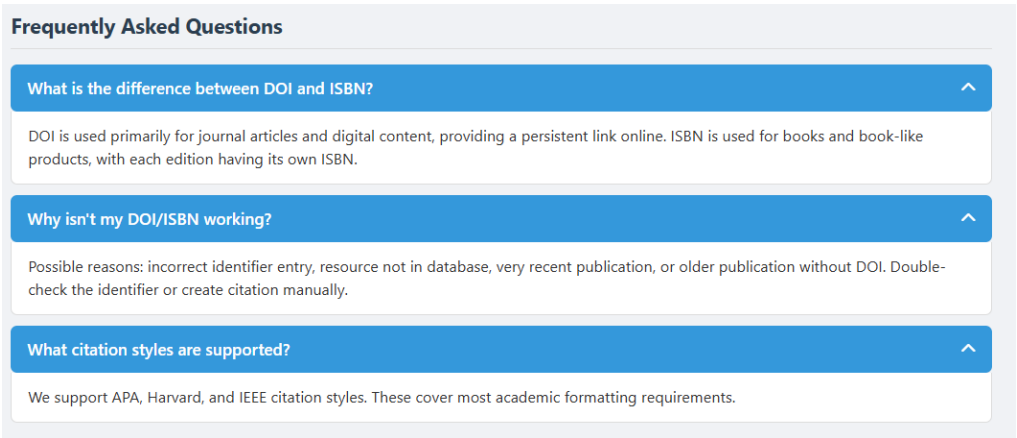


Figure 6.11: The Frequently Asked Questions section

JavaScript is used in the system to manage user interactions. For example, the handleGenerate() function is responsible for determining whether the user has supplied a single identifier or has uploaded a file. For single inputs, a POST request is sent to the /cite endpoint, while for file uploads, it sends a

multipart form to /upload. The results are dynamically inserted into the results table, allowing for instant feedback. Other than that, the error messages, such as "Invalid DOI or ISBN," are displayed in the table to ensure uniformity. Cascading Style Sheets (CSS) are utilised to style the interface, thereby endowing it with a contemporary design that incorporates clear typography, rounded boxes, and responsive layouts. Although the website is simple, the design ensures that the system is both user-friendly and accessible.

```
async function handleGenerate() {
  const citationBody = document.getElementById("citationBody");
  citationBody.innerHTML = "<tr><td colspan='2'>Generating...</td></tr>";

  const fileInput = document.getElementById("upload");
  const inputText = document.getElementById("input").value.trim();
  const style = document.getElementById("style").value;

  if (fileInput.files.length > 0) {
    const formData = new FormData();
    formData.append("file", fileInput.files[0]);
    formData.append("style", style);
    const response = await fetch("/upload", {
      method: "POST",
      body: formData,    });
    const data = await response.json();
    citationBody.innerHTML = Array.isArray(data) ? data.map(d =>
      `<tr><td>${formatCitationHTML(d.citation)}</td><td>${d.in_text}</td></tr>`
    ).join("") :
      `<tr><td>${formatCitationHTML(data.citation)}</td><td>${data.in_text}</td></tr>`;
  } else if (inputText) {
    let type = "";
    if (inputText.startsWith("10.") || inputText.includes("/")) type = "doi";
    else if (/^\d{9}[\dXx]$|^\d{13}$/.test(inputText)) type = "isbn";
    else {
      citationBody.innerHTML = "<tr><td colspan='2'>Invalid DOI or ISBN.</td></tr>";
      return;
    }
```

Figure 6.12: The handleGenerate() function for validation

6.6 Integration with External APIs

The system's functionality is significantly dependent on its integration with external metadata providers. For DOIs, the backend utilises the CrossRef API, which returns metadata in JSON format. The fetch_doi_metadata method is responsible for generating the API URL from the DOI, sending an asynchronous GET request, and parsing the message field of the JSON response. Metadata fields such as authors, title, journal, volume, issue, pages, and DOI are extracted and normalized before being formatted into citations. For ISBNs, the backend calls the Open Library API. The function fetch_isbn_metadata queries the

endpoint using the ISBN and parses the response, and because of the lack of standardisation inherent to Open Library responses, the function normalises the data by ensuring the inclusion of fields such as authors, publisher, publish date, and edition, even when replacement with placeholders such as "Unknown Author" is necessary.

The first check in both functions is checking the DuckDB cache to see whether metadata is already present to minimise external dependencies. If the API request fails, like the identifier does not exist, the functions will throw an HTTPException with a status code 404, to ensure the user receives a clear error message. The retrieved metadata is then passed to formatting procedures such as format_doi_citation and format_isbn_citation, which generate references in the chosen style. These functions are designed to handle differences between styles, including author name formatting, ordering, and punctuation, thereby ensuring that the output adheres to APA, Harvard, or IEEE rules.

```python
#Fetch metadata functions
async def fetch_doi_metadata(doi: str) -> dict:
    cached = await fetch_metadata_from_cache(doi, "doi")
    if cached:
        return cached
    url = f"https://api.crossref.org/works/{doi.strip()}"
    async with httpx.AsyncClient() as client:
        r = await client.get(url)
        if r.status_code != 200:
            raise HTTPException(status_code=404, detail="DOI not found")
        metadata = r.json()["message"]
        store_metadata_in_cache(doi, "doi", metadata)
        return metadata
```

Figure 6.13: Fetch DOI metadata function

```python
async def fetch_isbn_metadata(isbn: str) -> dict:
    cached = await fetch_metadata_from_cache(isbn, "isbn")
    if cached:
        return cached
    url = f"https://openlibrary.org/api/books?bibkeys=ISBN:{isbn}&format=json&jscmd=data"
    async with httpx.AsyncClient() as client:
        r = await client.get(url)
        if r.status_code != 200:
            raise HTTPException(status_code=404, detail="ISBN not found")
        d = r.json().get(f"ISBN:{isbn}", {})
        metadata = {
            "title": d.get("title", "Unknown Title"),
            "authors": [a["name"] for a in d.get("authors", [])] if "authors" in d else ["Unknown Author"],
            "publisher": d.get("publishers", [{}])[0].get("name", "Unknown Publisher"),
            "publish_date": d.get("publish_date", "Unknown Year"),
            "edition": d.get("edition_name", ""),
            "publish_places": d.get("publish_places", [])
        }
        store_metadata_in_cache(isbn, "isbn", metadata)
        return metadata
```

Figure 6.14: Fetch ISBN metadata function

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

Testing is one of the most critical phases in the software development life cycle, as it ensures that the system functions as intended, fulfills user requirements, and performs reliably under diverse conditions. A system that has not been thoroughly tested cannot be trusted to operate in a real-world environment, particularly when integrating with other services such as CrossRef and the Open Library. The objective of the project is to validate both functional needs, for example, the capacity to create citations in APA, Harvard, and IEEE formats, and non-functional criteria, such as response time, robustness to incorrect inputs, and error handling.

Testing also facilitates the identification of edge situations and inconsistencies that may have been overlooked during the development process. For example, metadata retrieved from other APIs might exhibit differences in structure, and the system must be capable of managing missing fields such as publisher names or page numbers. Moreover, invalid identifiers should not crash the system but instead return meaningful error messages. The structured testing facilitates a comprehensive evaluation of the system's resilience, correctness, and usability.

This system conducts the testing using Pytest, a widely utilised Python testing tool that functions seamlessly with FastAPI's TestClient. This facilitates the execution of automated testing procedures for API endpoints, getting rid of the necessity for requests to be routed through the front-end. In addition to utilising the automated backend testing, the user interface was manually tested to ensure usability and error handling. The combination of these tests provides confidence in the system's functionality, including its capacity to manage both expected and unexpected user behaviors.

7.2 Test Plan

The test plan was designed to cover the key functional and non-functional aspects of the system. Automated test cases were written in test_main.py, while manual UI tests were carried out using the browser interface. The main categories of tests are as follows:

i. Input Validation

- Test with valid DOIs and ISBNs to ensure that the system correctly retrieves metadata and creates citations.
- The system should be tested with invalid DOIs and ISBNs to ensure that clear error messages with code, such as 404 or 422, are returned.
- It is imperative to verify the absence of fields in the request payload to ensure that Pydantic validation produces structured error responses.

ii. Citation Output Correctness

- Verify that the citation output matches the rules of APA, Harvard, and IEEE formatting.
- Ensure that both the full reference citation and the in-text citation are included in responses.

iii. Database Storage (DuckDB Caching)

- When submitting the same DOI/ISBN several times, ensure that the metadata is retrieved from DuckDB rather than the external API following the first call.
- Ensure that cached info is stored correctly in JSON format and can be retrieved as needed.

iv. User Interface Usability

- Test the input form by inputting valid and incorrect identifiers.
- Upload a.txt file containing multiple identifiers, then validate that bulk citations are generated and displayed in the table.
- Test UI controls, such as copy citation, download citation, and clear form function, to ensure they perform as expected.

7.3 Test Case

Test cases provide a methodical method of determining if a system meets its requirements and expectations. The goal is to guarantee that each functional unit of the application performs as planned under both normal and extraordinary conditions. In the case of this citation converter system, test cases were critical not only for establishing that legitimate inputs produced correct citations, but also for ensuring that faulty inputs and deployment conditions were handled graciously. By developing structured test cases, the project was able to measure accuracy, robustness, and reliability in a controlled and repeatable manner.

7.3.1 Backend Test Case

For valid input scenarios, the test cases demonstrated that the system correctly retrieved metadata from CrossRef and Open Library and generated structured citations in APA, Harvard, and IEEE styles. These tests confirmed that the system's primary operation, converting identifiers into citations, worked properly and reliably. Test cases ensured that the system did not crash or give misleading results in circumstances of improper input, such as missing IDs, malformed ISBNs, or non-existent DOIs. Instead, it delivered structured error messages with corresponding HTTP codes. This demonstrated the system's reliability and capacity to guide users when mistakes occurred.

Project Name:	**DOI and ISBN Citation Converter**	**Test Designed by:**	**Yen Pei Xuan**
Module Name:	Local Test	**Test Designed date:**	8/8/2025
Release Version:	**1.0**	**Test Executed by:**	**Yen Pei Xuan**
		Test Execution date:	15/8/2025

Pre-condition	Input is valid
Dependencies:	
Test Priority	Medium

Test Case#	Test Title	Test Summary	Test Data	Expected Result	Post-condition	Actual Result	Status	Notes
TC1	Test Root Endpoint	Verify that the root '/' endpoint is accessible locally.	GET '/'	HTTP 200 OK	Page loads successfully	HTTP 200 OK	Pass	
TC2	Test Cite DOI	Verify citation generation for a valid DOI in APA style.	{"identifier": "10.1016/S1874-1029(13)60024-5", "type": "doi", "style": "APA"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	
TC3	Test Cite DOI	Verify citation generation for a valid DOI in Harvard style.	{"identifier": "10.1016/S1874-1029(13)60024-5", "type": "doi", "style": "Harvard"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	
TC4	Test Cite DOI	Verify citation generation for a valid DOI in IEEE style.	{"identifier": "10.1016/S1874-1029(13)60024-5", "type": "doi", "style": "IEEE"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	
TC5	Test Cite ISBN	Verify citation generation for a valid ISBN in APA style.	{"identifier": "9780131101630", "type": "isbn", "style": "APA"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	
TC6	Test Cite ISBN	Verify citation generation for a valid ISBN in Harvard style.	{"identifier": "9780131101630", "type": "isbn", "style": "Harvard"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	
TC7	Test Cite ISBN	Verify citation generation for a valid ISBN in IEEE style.	{"identifier": "9780131101630", "type": "isbn", "style": "IEEE"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	

Figure 7.1: Test case for valid input

Figure 7.1 demonstrates the result of the valid input test cases for DOI and ISBN. The figure shows that the system successfully retrieves metadata

from the CrossRef and Open Library APIs, formats the citations accurately in APA, Harvard, and IEEE styles, and returns both in-text and full citations. This confirms that the backend system handles valid inputs correctly and produces reliable citation outputs.

Project Name:	**DOI and ISBN Citation Converter**	**Test Designed by:**	**Yen Pei Xuan**
Module Name:	Local Test	**Test Designed date:**	8/8/2025
Release Version:	**1.0**	**Test Executed by:**	**Yen Pei Xuan**
		Test Execution date:	15/8/2025

Pre-condition	Input is Invalid
Dependencies:	
Test Priority	Medium

Test Case#	Test Title	Test Summary	Test Data	Expected Result	Post-condition	Actual Result	Status	Notes
TC1	Test Cite DOI	Verify system handles invalid DOI.	{"identifier": "10.0000/invalid-doi", "type": "doi", "style": "APA"}	HTTP 404 or 422 error.	Error response returned.	HTTP 404	Pass	Clear error message shown
TC2	Test Cite ISBN	Verify system handles invalid ISBN input.	{"identifier": "0000000000000", "type": "isbn", "style": "APA"}	HTTP 404 or 422 error.	Error response returned.	HTTP 404	Pass	Invalid ISBN handled properly
TC3	Test Cite Missing Identifier	Verify system rejects requests without identifier.	{"type": "doi", "style": "APA"}	HTTP 422 error (validation).	Error response returned.	HTTP 422	Pass	Validation error handled
TC4	Test Cite Unsupported Style	Verify system rejects unsupported citation style.	{"identifier": "10.1016/S1874-1029(13)60024-5", "type": "doi", "style": "Chicago"}	HTTP 400 or 422 error.	Error response returned.	HTTP 400	Pass	Unsupported style handled

Figure 7.2: Test case for invalid input

Figure 7.2 illustrates how the system responds to invalid DOI and ISBN identifiers. When an invalid or missing identifier is entered, the backend returns a proper error response, such as 404 or 422, without crashing or producing an incorrect citation. This verifies the robustness of the input validation and error-handling mechanisms in the backend API.

Finally, deployment-level test cases were run to ensure that the system functioned properly when hosted on DigitalOcean rather than only in a local development environment. These tests ensured that the deployed API endpoints were internet accessible, returned the same accurate results as local tests, and remained usable via the web interface.

Project Name:	**DOI and ISBN Citation Converter**	**Test Designed by:**	**Yen Pei Xuan**
Module Name:	Cloud Test	**Test Designed date:**	8/8/2025
Release Version:	**1.0**	**Test Executed by:**	**Yen Pei Xuan**
		Test Execution date:	15/8/2025

Pre-condition	Input from internet
Dependencies:	
Test Priority	Medium

Test Case#	Test Title	Test Summary	Test Data	Expected Result	Post-condition	Actual Result	Status	Notes
TC1	Test Root Endpoint	Verify deployed system's root endpoint is accessible.	GET https://citationconverter.duckdns.org/	HTTP 200 OK	Page loads successfully	HTTP 200 OK	Pass	Deployment success
TC2	Test Cite DOI (Valid)	Verify deployed system generates citation for valid DOI.	{"identifier": "10.1016/S1874-1029(13)60024-5", "type": "doi", "style": "APA"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	Matches local test
TC2	Test Cite ISBN (Valid)	Verify deployed system generates citation for valid ISBN.	{"identifier": "9780131101630", "type": "isbn", "style": "IEEE"}	JSON response with fields `citation` and `in_text`.	Valid formatted citation returned.	Correct citation + in-text shown	Pass	Matches local test

Figure 7.3: Test case for Internet

Figure 7.3 shows the results of testing the deployed application hosted on DigitalOcean. It demonstrates that both the root endpoint and the /cite endpoint work correctly in the live environment, retrieving metadata and

generating citations with the same accuracy as in local testing. This confirms that the deployment is stable and functions properly over the internet.

7.3.2 Frontend Test Case

In addition to backend test cases, additional user interface (UI) test cases were created to ensure that the citation generator's web-based front end functions properly and delivers a seamless user experience. These test cases focused on validating the interface's primary interactive components: the upload and download functions, the copy citation button, and the clear form button. The upload function was tested with .txt files containing both valid and invalid IDs. The system successfully parsed valid DOIs and ISBNs line by line, generating citations in the desired manner and displaying them in the output table. Invalid lines were also handled graciously, with the system displaying relevant error warnings rather than halting the entire operation.

The download function was tested by generating citations and exporting them in .txt and .bib formats. The .txt format created clear, plain-text citations suited for general use, but the .bib format generated BibTeX-compatible references that could be loaded straight into reference management software. Both file types were successfully downloaded, had valid citation entries, and were free of corruption. The copy citation button was checked to ensure that all generated citations could be copied to the clipboard with only one click. The test demonstrated that the citations were properly sent in structured form, allowing users to paste them immediately into their documents.

The clear form button was tested by first producing citations and then pressing the button to reset the input field and citation table. The test confirmed that all entries were cleared, and the system was restored to its original condition without refreshing the browser page, and rapidly initiated a fresh session. These UI test cases supplement the backend tests by ensuring that the entire system—from identifier input to citation production, export, and reset—functions consistently and quickly.

Project Name:	**DOI and ISBN Citation Converter**	**Test Designed by:**	**Yen Pei Xuan**
Module Name:	Frontend Test	**Test Designed date:**	8/8/2025
Release Version:	1.0	**Test Executed by:**	Yen Pei Xuan
		Test Execution date:	15/8/2025

Pre-condition	
Dependencies:	
Test Priority	Medium

Test Case#	Test Title	Test Summary	Test Data	Expected Result	Post-condition	Actual Result	Status	Notes
TC1	Test Upload Function	Verify that the system correctly processes a `.txt` file containing multiple DOIs/ISBNs.	Upload file: DOI.txt	System reads each line, generates citations for valid identifiers, flags invalid ones, and displays results in table.	Bulk citations displayed in output table.	Bulk upload successful, with errors flagged.	Pass	Confirms batch processing works.
TC2	Test Download Function	Verify that citations can be exported in `.txt` and `.bib` formats.	Generate citations → Click 'Download' (choose format).	File downloaded with all citations in correct format.	File saved on user's system.	Downloaded file opens correctly with citations intact.	Pass	Both TXT and BibTeX confirmed.
TC3	Test Copy Citation Button	Verify that all generated citations are copied to clipboard.	Generate citations → Click 'Copy Citation'.	All formatted citations copied to clipboard.	Citations available for paste into external document.	Copied text matches generated citations.	Pass	Clipboard content verified.
TC4	Test Clear Form Button	Verify that the input field and citation table are reset.	Generate citations → Click 'Clear Form'.	Input and table cleared, form reset to initial state.	System returns to ready state for new input.	Fields cleared and ready for new session.	Pass	Improves usability for repeated use.

Figure 7.4: Test Case for Frontend

Figure 7.4 displays the frontend testing of the citation converter interface. The figure shows that all user interactions, such as entering identifiers, uploading files, downloading results, and copying citations, operate smoothly without errors. The test verifies that the user interface components integrate effectively with the backend API to deliver a seamless experience.

7.4 Test Code

The testing phase showed that the system is functionally correct and stable in all scenarios. The pytest framework was used to create and run 14 automated test cases, all of which passed successfully in 3.43 seconds. The test suite consisted of three key sections: valid input tests, invalid input tests, and deployment-level tests. For valid inputs, the system accurately processed both DOI and ISBN IDs and generated citations in APA, Harvard, and IEEE formats. This showed that the citation formatting criteria were followed consistently and that the system supported a variety of academic standards.

Invalid input scenarios were handled as planned, with the system returning structured error messages and appropriate HTTP codes 404 or 422 to ensure robustness against wrong or missing user inputs. The deployment-level tests, which visited the system via its live DigitalOcean URL, confirmed that the application performed identically in the production environment, demonstrating the reliability of API integration and the stability of the deployed service. The fact that all 14 tests completed successfully in 3.43 seconds demonstrates the implementation's efficiency, with no notable performance bottlenecks identified. Overall, the findings show that the system meets its functional requirements, is resilient to edge cases, and can perform consistently in a real-world setting.

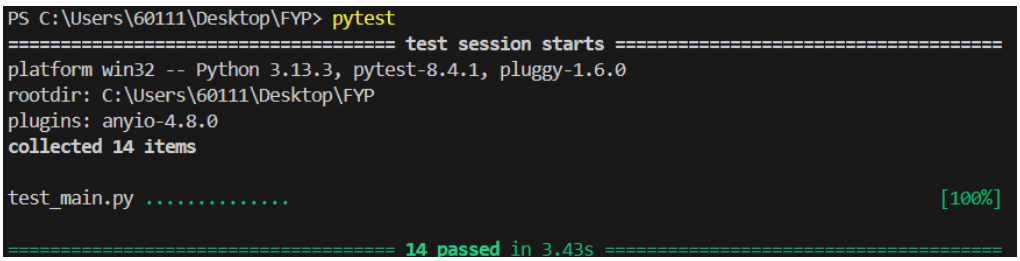


Figure 7.5: Test Result

Figure 7.5 presents the summary of all automated and manual test cases executed. It indicates that all 14 tests have been collected and passed successfully in 3.43 seconds, showing that the system performs efficiently and consistently. The overall result confirms that both the backend and frontend meet functional and performance expectations.

Table 7.1: Table of Summary of Test Cases

Test ID	Test Type	Description	Expected Result	Actual Result	Status
TC01	Backend	Test root endpoint	Homepage returns 200 OK	200 OK returned	Passed
TC02	Backend	Valid DOI in APA	Citation generated correctly	Citation accurate	Passed
TC03	Backend	Valid DOI in Harvard	Citation generated correctly	Citation accurate	Passed
TC04	Backend	Valid DOI in IEEE	Citation generated correctly	Citation accurate	Passed
TC05	Backend	Valid ISBN in APA	Citation generated correctly	Citation accurate	Passed
TC06	Backend	Valid ISBN in Harvard	Citation generated correctly	Citation accurate	Passed

TC07	Backend	Valid ISBN in IEEE	Citation generated correctly	Citation accurate	Passed
TC08	Backend	Invalid DOI identifier	Error handled gracefully	404/422 response	Passed
TC09	Backend	Invalid ISBN identifier	Error handled gracefully	404/422 response	Passed
TC10	Backend	Missing identifier	System returns 422	Validation triggered	Passed
TC11	Backend	Unsupported style	400/422 error	Error handled	Passed
TC12	Internet	Deployed root endpoint	200 OK response	200 OK returned	Passed
TC13	Internet	Deployed DOI test	Citation generated remotely	Citation accurate	Passed
TC14	Frontend	UI upload, download, copy, clear	All buttons work correctly	Functions validated	Passed

7.5 Discussion

The testing phase confirmed that the citation generation system functions correctly and reliably in a range of scenarios. Automated backend testing with Pytest demonstrated that the API endpoints respond appropriately to both valid and invalid queries. Valid DOI and ISBN inputs produced citations and in-text references in APA and Harvard styles that were consistently styled correctly. An additional IEEE formatting test confirmed that the criteria for formatting author names, titles, and volume/issue numbers were correctly applied. Invalid identifiers, such as erroneous DOIs or placeholder ISBNs, resulted in structured error responses with the relevant HTTP status codes (404 or 422).

Furthermore, missing fields in the request payload triggered Pydantic validation, demonstrating that the backend maintains strict input integrity. Database caching was also tested by repeatedly submitting the identical DOI/ISBN. The initial call initiated a fetch from the external API, while successive requests collected the stored metadata from DuckDB. This caching reduced the number of external API requests, increasing efficiency and demonstrating the system's capacity to optimise performance with repeated use. The DuckDB entries were validated to be saved in JSON format, allowing for simple retrieval for subsequent formatting processes.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

8.1 Conclusion

The project has successfully achieved its primary goal of developing a fully functional and reliable automatic citation converter system that is capable of processing both DOI and ISBN inputs. The system integrates seamlessly with the CrossRef API for journal articles and the Open Library API for books, to retrieve accurate metadata and convert it into citations that follow academic standards. The inclusion of three frequently used citation styles, APA, Harvard, and IEEE, guarantees that users are presented with the most relevant referencing forms for higher education and research. Also, DuckDB's effective implementation as a lightweight and embedded database enabled efficient metadata caching, resulting in reduced API calls and increased system responsiveness to repeated requests.

Aside from functionality, the system provided a clean and intuitive user interface that allows users to manually enter individual identifiers or upload bulk lists for batch processing. Furthermore, some additional features were implemented, such as the ability to copy citations, download them in various formats, and view clear findings, improving usability and efficiency. Furthermore, FastAPI was implemented because it supplied a modern, high-performance backend architecture, and Pydantic ensured stringent input validation and error handling. The system's reliability was further validated by automated testing with Pytest, which demonstrated its resilience and accuracy across many circumstances.

In conclusion, the project demonstrates the high degree of alignment between the design goals, implementation results, and testing outcomes. It helps solve a real-world academic problem by reducing the workload for students and researchers, improving citation accuracy, and streamlining the reference management process. Despite its limitations, the project provides a solid foundation that could eventually be expanded into a comprehensive academic citation platform.

8.2 Problems Encountered

Throughout the development and testing phases, a series of obstacles were encountered, posing significant challenges to the overall progress of the project. A significant challenge encountered pertained to the inconsistency of metadata from external APIs. While CrossRef and Open Library generally provided comprehensive information, there were many cases where essential details such as publisher names, author lists, or publication years were missing or formatted irregularly. This created challenges in generating properly formatted citations. To resolve this, placeholder defaults such as “Unknown Author” or “Unknown Year” were applied, ensuring that the citation output remained complete and usable.

Another concern was the dependency on external APIs, which periodically experienced outages or delayed response times. These outages disrupted the development and testing processes, highlighting the risk of relying on third-party services. The current solution includes storing metadata locally in DuckDB, so the repeated requests for the same identifier can be delivered quickly without re-querying the APIs, reducing delays and service interruptions.

Another problem encountered was the intricacy of citation formatting, particularly in the IEEE style. Unlike APA and Harvard styles, the IEEE has strict guidelines regarding author initials, punctuation, and order. To ensure compliance, these rules had to be implemented through careful coding and repeated testing. Even slight formatting errors could compromise academic legitimacy, so extra care was needed to ensure compliance with the standard.

During the deployment to the DigitalOcean platform, a novel challenge manifested in the form of configuration issues pertaining to the domain and the SSL certificate. These issues engendered delays in the process of rendering the server accessible to the public. Furthermore, an attempt was made to arrange the deployment to operate with four worker processes for scalability. However, it was revealed that DuckDB does not support concurrent access across multiple workers. DuckDB has been developed as an embedded, single-process database; as a result, it is incapable of accommodating concurrent reads and writes from multiple Uvicorn workers. Consequently, the application was incapable of scaling to multiple workers according to standard methods and was constrained

to functioning with a single worker process. While this approach ensured reliability, it also imposed limitations on the system's throughput and concurrency, resulting in suboptimal scalability.

8.3 Limitations

Although the system offers a wide range of features, it has some limitations. One of the primary limitations of the system is the restricted number of supported citation styles. The system only supports APA, Harvard, and IEEE currently. While these are popular in academics, other styles such as Chicago, MLA, and Vancouver are also frequently required. The lack of these styles may limit the system's popularity among users who require more diverse formatting options.

Another limitation is the system's reliance on external APIs, such as CrossRef for DOIs and Open Library for ISBNs. Although these APIs provide huge and authoritative datasets, the system is vulnerable to downtime or data quality concerns caused by these services. If the API is unavailable, the citation generator will be unable to retrieve new metadata, reducing reliability for end users.

Thirdly, the system's reliance on an internet connection, because all information retrieval is based on online API calls, users cannot create new citations when offline. This reliance restricts usability in locations with weak connectivity or for researchers who require offline capabilities when traveling or conducting fieldwork.

Finally, the system's scalability is limited by the use of DuckDB. DuckDB is lightweight and efficient for caching, but its single-process nature prohibits it from growing horizontally to accommodate additional workers. This limitation limits the number of concurrent users that the system can support in its current configuration, making it unsuitable for large-scale institutional implementation.

8.4 Recommendations/Future Work

The system can be improved to become a better system and enrich its functions. Therefore, this chapter will provide some recommendations for the future to improve the system.

8.4.1 Expansion of Supported Citation Styles

One of the most important future enhancements for the system is the inclusion of additional citation styles beyond APA, Harvard, and IEEE. While these three formats cater to a wide academic readership, they do not meet the needs of specialties that rely significantly on alternative styles such as Chicago, Vancouver, and MLA. For example, the Chicago style is frequently used in the humanities, whereas Vancouver is the standard for medical and scientific articles. By extending the system to include additional formats, the tool will become more adaptable and appealing to a larger academic community.

8.4.2 Offline Caching and Local Metadata Storage

Another future development is the introduction of offline caching mechanisms. Currently, the system requires a constant internet connection to fetch metadata from CrossRef or Open Library, which causes limitations. Implementing offline caching would allow metadata to be saved locally after the initial retrieval, allowing users to generate citations even without internet connectivity. This might be accomplished by extending DuckDB to include an exportable and shareable cache file that can be moved between devices. Users could also select to pre-download metadata for a batch of identifiers, ensuring continuous citation production while offline. This offline functionality would further increase API robustness by allowing users to get previously cached results without relying on external services.

8.4.3 Integration with Browser Plugins and Word Processors

For optimal usage, the system can be expanded to include browser plugins and word-processor add-ins. Many students and researchers create their work directly in apps like Microsoft Word, Google Docs, and LaTeX editors. Frequently jumping between the citation generator and a writing platform might be distracting and inefficient. Users might generate citations in real time while

writing by embedding the system as a plugin in word processors, integrating correctly formatted references straight into their documents. Similarly, browser extensions could enable users to generate citations while perusing publications online, eliminating the need for manual copying of DOIs or ISBNs.

8.4.4 Migration from DuckDB to SQLite for Improved Concurrency

Finally, a recommendation for future work is to replace DuckDB with SQLite as the system's local database engine. Although DuckDB performed well during development for caching metadata, it does not enable real concurrent writes across several worker processes, as demonstrated during deployment attempts on DigitalOcean. The system could address this problem by migrating to SQLite, which supports multi-process concurrency more effectively. It supports safe concurrent reads and, with correct configuration, controlled write access, making it more suited to web applications deployed with numerous workers. SQLite is widely supported and has built-in transactional stability, so it ensures data consistency while allowing for horizontal growth among worker processes. This migration will preserve the simplicity of a file-based database while eliminating the bottleneck created by DuckDB, thereby increasing the system's scalability, dependability, and responsiveness in production settings.

REFERENCES

American Psychological Association (2020) *Publication Manual Of The American Psychological Association.* 7th ed. S.L.: American Psychological Association.

Best Edit & Proof (2021). *Importance of Citations in Academic Writing*. Available at: https://besteditproof.com/en/academy/importance-of-citations-in-academic-writing (Accessed: 10 April 2025).

Django (2025) *Meet Django.* Available at: https://www.djangoproject.com/ (Accessed: 10 April 2025).

EndNote (2025) *EndNote Desktop Comparison Chart.* Available at: https://endnote.com/product-details/compare-current-versions/ (Accessed: 10 April 2025).

FastAPI (no date) *FastAPI.* Available at: https://fastapi.tiangolo.com (Accessed: 10 April 2025).

Flask (2010) *Flask.* Available at: https://flask.palletsprojects.com/en/stable/ (Accessed: 10 April 2025).

Hunter, J. (2006) *The importance of citation*. Available at: https://web.grinnell.edu/Dean/Tutorial/EUS/IC.pdf (Accessed: 10 April 2025).

Indeed (2025) *What is a Software Process Model?* Available at: https://uk.indeed.com/career-advice/career-development/what-is-iterative-methodology (Accessed: 10 April 2025).

Keployio (2024) *Mastering Python Test Coverage: Tools, Tips, and Best Practices*. Available at: https://medium.com/@keployio/mastering-python-test-coverage-tools-tips-and-best-practices-11daf699d79b (Accessed: 10 April 2025).

Mendey (2025) *Mendeley Cite.* Available at: https://www.mendeley.com/reference-management/mendeley-cite (Accessed: 10 April 2025).

Montana.edu (2020) *Citation Help and Style Guide.* Available at: https://guides.lib.montana.edu/citation/overview (Accessed: 10 April 2025).

Rephrasely (2025) *Using Citation Generators: Benefits and Limitations - Rephrasely*. Available at: https://rephrasely.com/blog/using-citation-generators-benefits-limitations (Accessed: 10 April 2025).

Swagger (no date) *Crossref REST API.* Available at: https://api.crossref.org/swagger-ui/index.html (Accessed: 10 April 2025).

Tutorspoint (2024) *Revealing 5 Important Benefits of Using Citation Tools*. Available at: https://www.tutorspoint.com/blog/revealing-5-important-benefits-of-using-citation-tools (Accessed: 10 April 2025).

Universiti Tunku Abdul Rahman (2022) *IEEEReferenceGuide.* Available at: https://library.utar.edu.my/documents/Guides/IEEEReferenceGuide_12Aug2022.pdf (Accessed: 10 April 2025).

University of Reading Library (2019) *Which referencing style should you use?*. Available at: https://www.youtube.com/watch?v=vFSbpoWzA_4&t=2s (Accessed: 10 April 2025).

University of Washington (2025) *Mendeley: Working in Groups*. Available at: https://guides.lib.uw.edu/hsl/mendeley/organize/groups (Accessed: 10 April 2025).

Visual Paradigm (2024) Available at: https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model (Accessed: 10 April 2025).

Zotero (2025) *Your personal research assistant..* Available at: https://www.zotero.org/ (Accessed: 10 April 2025).

APPENDICES

Appendix A: Tables

Table 2.1: Tables of Tools Comparison

Tool	Strengths	Limitations
Zotero	Free, open-source, supports 9,000+ styles, browser integration	Limited free storage, lacks customer support
Mendeley	MS Word integration, PDF annotation, and collaboration tools	Limited free groups, privacy concerns
EndNote	Advanced management, cloud sync, institutional support	Paid full version, limited free features

Table 2.2: Table of differences between frameworks compared

Framework	Function	Strengths	Limitation
Django	Build complex website	Comprehensive features, widely supported	Heavy for small projects, rigid structure,
Flask	lightweight micro-framework	Simple and flexible, easy to learn	Lacks built-in tools, Required manual setup extensions
FastAPI	Modern, high-performance	Asynchronous support, automatic validation with Pydantic	Newer ecosystem, learning curve for async handling

Table 4.1: Functional Requirements

ID	Functional Requirement
FR001	The system shall allow users to input a DOI and retrieve metadata from the CrossRef API.
FR002	The system shall allow users to input an ISBN and retrieve metadata from the Open Library API.

FR003	The system shall format metadata into APA style according to the latest APA referencing guidelines.
FR004	The system shall format metadata into Harvard style according to standard Harvard referencing rules.
FR005	The system shall format metadata into IEEE style according to IEEE referencing rules.
FR006	The system shall store generated citations in a DuckDB database for future retrieval and reuse.
FR007	The system shall validate user inputs and return error messages for invalid DOI/ISBN values.
FR008	The system shall provide a download or copy function, allowing users to incorporate the citation into their documents.
FR009	The system shall provide a basic user interface for inputting DOIs/ISBNS, viewing results, and downloading citations.
FR010	The system shall allow exporting citations in plain text format.
FR011	The system shall allow exporting citations to BibTeX format.
FR012	The system shall provide a clear form button to let users clear the input.
FR013	The system shall provide a simple FAQ section for users.

Table 4.2: Non-Functional Requirements

ID	Non-Functional Requirement	Category	Priority
NFR001	The system shall return results within 3–5 seconds of DOI/ISBN input.	Performance	High
NFR002	The system shall have a clean and user-friendly interface that requires minimal training.	Usability	High
NFR003	The system shall generate citations with at least 95% accuracy according to citation guidelines.	Reliability	High
NFR004	The system shall be able to handle simultaneous requests without	Scalability	Medium

	significant performance degradation.		
NFR005	The system shall support the addition of new citation styles with minimal changes in code.	Scalability	Medium
NFR006	The system shall ensure that data retrieved from APIs and stored in the database is secure and protected.	Security	High
NFR007	The system shall provide error messages and fallback options when API services are unavailable.	Reliability	High
NFR008	The system shall maintain compatibility with modern browsers (Chrome, Edge, Firefox).	Portability	High
NFR009	The system shall be documented and version-controlled with GitHub for future maintainability.	Maintainability	High
NFR0010	The system shall comply with academic citation guidelines and be easily extendable to support future updates in APA/Harvard/IEEE style formats.	Compliance	High

Table 5.1: Data dictionary

Column Name	Type	Key	Description
Id/key	String	Primary	DOI or ISBN. Example: 10.1016/S1874-1029(15)30001-X.
metadata	JSON	-	Full raw metadata response from Crossref/Open Library (authors, title, year, publisher, references, etc.).

Table 7.1: Table of Summary of Test Cases

Test ID	Test Type	Description	Expected Result	Actual Result	Status
TC01	Backend	Test root endpoint	Homepage returns 200 OK	200 OK returned	Passed
TC02	Backend	Valid DOI in APA	Citation generated correctly	Citation accurate	Passed
TC03	Backend	Valid DOI in Harvard	Citation generated correctly	Citation accurate	Passed
TC04	Backend	Valid DOI in IEEE	Citation generated correctly	Citation accurate	Passed
TC05	Backend	Valid ISBN in APA	Citation generated correctly	Citation accurate	Passed
TC06	Backend	Valid ISBN in Harvard	Citation generated correctly	Citation accurate	Passed
TC07	Backend	Valid ISBN in IEEE	Citation generated correctly	Citation accurate	Passed
TC08	Backend	Invalid DOI identifier	Error handled gracefully	404/422 response	Passed
TC09	Backend	Invalid ISBN identifier	Error handled gracefully	404/422 response	Passed
TC10	Backend	Missing identifier	System returns 422	Validation triggered	Passed
TC11	Backend	Unsupported style	400/422 error	Error handled	Passed

TC12	Internet	Deployed root endpoint	200 OK response	200 OK returned	Passed
TC13	Internet	Deployed DOI test	Citation generated remotely	Citation accurate	Passed
TC14	Frontend	UI upload, download, copy, clear	All buttons work correctly	Functions validated	Passed

Appendix B: Open Access to Image Rights

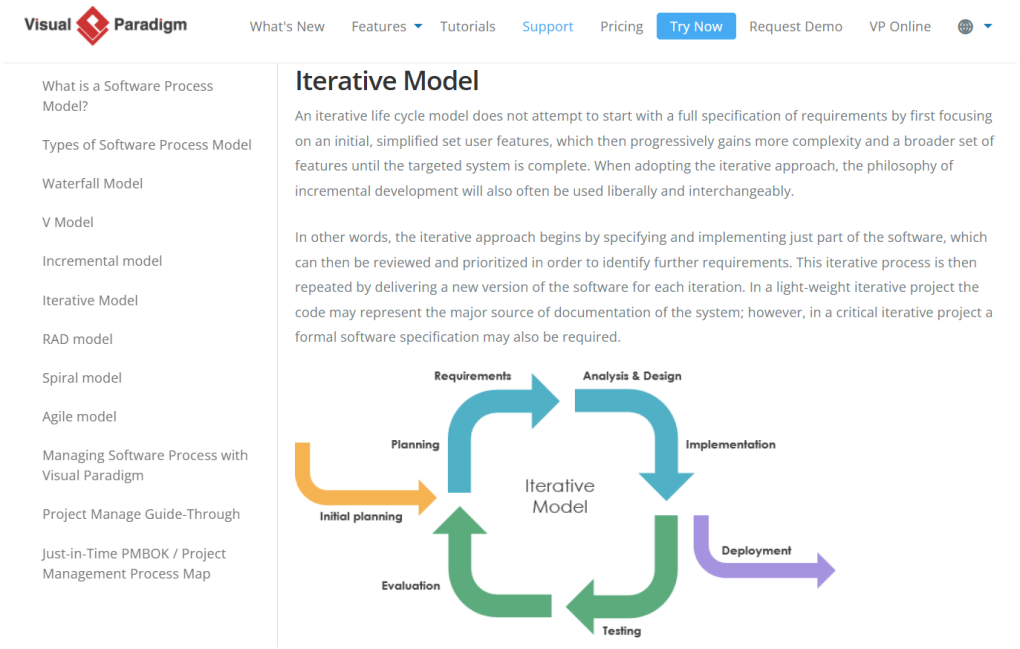


Figure C-1: Reprinted with Permission from Copyright 2024 Visual Paradigm to Reuse the What Is Iterative Methodology? (Visial Paradigm, 2024). Image in Figure 3.1.

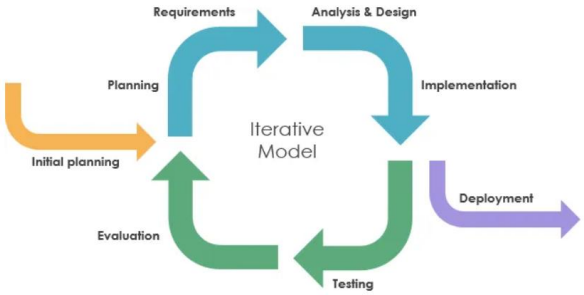


Figure 2.1: What Is Iterative Methodology? (Visual Paradigm, 2024). Reprinted with Permission from Copyright 2024 Visual Paradigm.