VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Implementing a Test Generation Service For Flutter Framework

By
Dao Minh Huy

A thesis submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the degree of Bachelor of Computer Science

Ho Chi Minh City, Vietnam
June 2025

Implementing a Test Generation Service For Flutter Framework

APPROVED BY:

____

Committee name here

____

Committee name here

____

Committee name here

____

Committee name here

____

Committee name here

THESIS COMMITTEE

Acknowledgments

It is with profound gratitude and sincere appreciation that I extend my heartfelt thanks to Dr. Tran Thanh Tung for his unwavering support and exceptional professional guidance throughout the course of this thesis. His dedication, insightful feedback, and encouragement provided me with the optimal conditions to carry out and complete this research successfully. Dr. Tran Thanh Tung's invaluable knowledge and expertise have been a constant source of motivation and inspiration, significantly contributing to my learning process and academic growth.

I also want to express my thanks to all professors and lecturers who have followed and instructed me throughout my university journey. Their expertise and experience have enlightened and sharpened my skills to confidently enter the industry.

Lastly, I sincerely thank the thesis evaluation committee for their valuable time reviewing and assessing this thesis.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Listings

Abstract

Software testing is indispensable for ensuring the reliability and correctness of any software product before deployment. Despite its importance, developers often find writing unit tests and integration tests tedious and time-consuming. This is not due to the complexity of the process but to the cognitive effort required to work retrospectively, evaluating and validating code logic that has already been implemented without being bias from the logic of the source code.

This thesis introduces an innovative approach leveraging the capabilities of Artificial Intelligence (AI) called "Test Genie", which will alleviate developers' workloads by automating the generation of test cases. By offloading the task of test generation to an AI-driven system, developers can concentrate entirely on writing robust and functional source code. The proposed solution employs the Retrieval-Augmented Generation (RAG) technique to enhance the quality and relevance of the generated test cases, ensuring that the results align with the intended behavior of the code.

To further validate the practicality of the system, the service incorporates an embedded Software Development Kit (SDK) for the supported platform, with the initial implementation focused on the Flutter framework. This integration ensures that the AI-generated test files adhere to the platform's standards and are executable without manual intervention.

The results of this research aim to demonstrate how AI can transform the software testing process, reducing developer effort, improving testing efficiency, and fostering higher-quality code in modern software development.

Chapter 1
INTRODUCTION

1.1 Background

As software systems become increasingly complex, the demand for rigorous software testing has grown significantly. Modern applications often integrate multiple components, rely on distributed architectures, and interact with various external systems, making them more vulnerable to errors. According to a study by Capgemini (2021), the average cost of software failures has risen by 15% annually [1], underscoring the need for comprehensive testing to ensure reliability. Furthermore, the adoption of agile and DevOps methodologies has accelerated development cycles, necessitating continuous testing to maintain quality. The World Quality Report (2022) highlights that 78% of organizations have increased their investment in testing tools and resources over the past five years [1], reflecting the growing recognition of testing as a critical component of software development. Due to high demand in software testing, the market value of digital assurance also get higher. The average annual salary of Quality assurance tester have increased, from 60,000$ in 2015 to 82,000$ in 2024 [2].

1.2 Problem Statement

The rapid evolution of technology has led to the proliferation of programming languages and development frameworks, each with unique features and ecosystems. While this diversity offers developers powerful tools and improved syntax to enhance productivity, it also introduces significant challenges in the testing process. Developers must familiarize themselves with different testing languages, frameworks, and techniques for each platform, which can be both time-consuming and error-prone.

Although languages and frameworks are getting better in both syntax and community support, the testing process also getting trickier. Writing comprehensive unit and integration tests often requires developers to think "backtrackingly," reconstructing potential use cases and edge cases after implementing the functionality. A human can overlooking critical edge cases that might cause a costly consequence. According to CISQ, poor software cost the U.S. economy $2.08 trillion in 2020 alone [3].

To address these challenges, this thesis proposes the integration of an AI-driven Test Generation Service named Test Genie. By leveraging Large Language Models (LLMs), this service automates the creation of test cases, significantly reducing the burden on developers. Automating this process not only optimizes resource allocation but also minimizes the potential for human error, ensuring a more thorough and systematic approach to software testing.

1.3 Scope and Objectives

Initially, this thesis will only focus on one single framework: Flutter - a cross-platfrom framework that can build the product for many platform from one source code. Although Flutter is considered a new framework but the support community and the usage of this framework is increasing every year. This framework also support a testing module, enable users to develop different testing packages and techniques. The research will assess the feasibility of AI in test cases generation by using Langchain library to integrate API of LLM models. By using multiple LLM models, the thesis aim to present a suitable methodology that could provide support to reduce QA testers and developers's workload and effectively cover edge cases that human often miss.

To successfully implement this service, three primary objectives must be achieved. First, the AI must demonstrate the capability to analyze the business model and functional requirements directly from the project source code. This requires understanding the logical structure and intent of the application. Second, the AI must leverage an effective test generator model capable of producing test cases that align with the platform's standards while maintaining relevance to the identified business logic. Third, the generated code must be thoroughly validated to ensure its correctness and compatibility within the Flutter ecosystem. By meeting these requirements, the proposed service aims to establish a reliable and efficient solution for automating test case generation.

In this thesis, we will work on three components:

- Business Logic Analyzer module (BLA)
- AI-integrated test generation module
- AI test validation module

Each component will share the same tech stack:

- Python [6]: This is a popular high-level language that used widely by AI developers. Its simple syntax and wide range of supportive library help developers effectively implement complex system with minimal syntax.
- Python-Flask [4]: This is a micro web framework for Python. It is lightweight and easy to use, making it suitable for building small to medium-sized web applications.
- Python-Langchain [5]: Langchain is a framework for developing applications powered by Large Language Models (LLMs). This is an open-source framework and effectively utilize API provided by LLMs service provider as well as self-hosted LLMs.

1.4 Structure of thesis

This thesis consist of six chapters:

- **Chapter 1. Introduction:** Introduce the background story, how I identify the problem as well as the scope and objectives of this research. This chapter also lightly introduce the proposed solution of the stated problem.
- **Chapter 2. Liturature review/Related work:** This chapter focus on the related work that contributed to the thesis.
- **Chapter 3. Methodology:** Presenting the methodology behind the project, including the component of the system, method implemented for each module and the plan to validate the generated test from AI.

- **Chapter 4. Implement and results:** This chapter summarize the design and implementations of the system as well as the result of this research.
- **Chapter 5. Discussion and evaluation:** In this chapter, we will evaluate the result of this system.
- **Chapter 6. Conclusion and future work:** This chapter will conclude the research of this thesis, as well as the plan of development in the future.

Chapter 2
LITURATURE REVIEW/RELATED WORK

2.1 Unit test generator

LLMs approach compared to fomulated approach. To accurately give test case with correct syntax, I have researched some techniques that can handle different frameworks with just one centrialized system. There is a research that compares the performance of some common approaches including search-based, constraint-based and random-based. Tests generated by these methods frequently lack meaningful structure or descriptive naming conventions, making them difficult for developers to interpret and modify [7]. This limitation can hinder their practical usability, particularly in dynamic and iterative development environments.

In contrast, test case generation using Large Language Models (LLMs) offers a more intuitive and human-aligned approach [7]. LLMs, trained on vast amounts of programming-related data, possess the capability to generate test cases that not only adhere to syntactical correctness but also align closely with human developers' intentions and coding practices. This alignment results in unit tests that are more readable, contextually relevant, and easier to understand. Developers can quickly adjust and refine these tests as needed, enhancing their utility in real-world scenarios.

Moreover, the flexibility of LLMs enables them to adapt seamlessly to various programming languages and frameworks, providing a centralized solution for diverse development ecosystems. While traditional approaches may produce marginally higher percentages of technically correct test cases, they often lack the usability and adaptability that LLM-based methods provide. As a result, services leveraging LLMs for test generation consistently receive more favorable user feedback due to their focus on developer experience, ease of use, and alignment with real-world development workflows.

Disadvantages of LLMs. One of the most significant challenges is their propensity to generate hallucinations, where the model produces incorrect or fabricated outputs that lack grounding in factual data. This issue is particularly critical in tasks requiring precision, such as author attribution. For instance, research introducing the Simple Hallucination Index (SHI) revealed that even advanced LLMs like Mixtral 8x7B, LLaMA-2-13B, and Gemma-7B suffered from hallucinations, with Mixtral 8x7B achieving an SHI as high as 0.87 for certain datasets [8]. These hallucinations undermine the reliability and trustworthiness of LLMs, especially in contexts where factual accuracy is crucial.

Another drawback of LLMs is their lack of transparency in decision-making. These models function as black boxes, providing little insight into the reasoning behind their outputs [8]. This opacity complicates the debugging process and limits the ability to verify results, which is particularly problematic in applications requiring a high degree of explainability. Additionally, LLMs are highly dependent on the quality and diversity of their training data. Biases or inaccuracies present in the data can result in outputs that reinforce those biases or produce flawed results. Moreover, while these models excel at generating output based on their training corpus, they often struggle to generalize effectively when faced with novel or unseen cases.

2.2 Understanding Business Logic

The concept of Business Logic. An industry's business logic can be seen as a description of a number of basic conditions or circumstances that make up important starting points for understanding an established business and its conditions for change [9]. It encodes the real-world policies, procedures, and processes that govern how data is created, managed, and manipulated in a way that aligns with the objectives of the organization. Business logic acts as the foundation for decision-making and operational tasks, ensuring that the software performs actions that mirror the intended business behavior. This could involve calculating prices, validating transactions, or managing inventory, all based on predefined rules and conditions derived from the organization's requirements. Business logic serves as the intellectual layer of an application, translating business needs into functional processes that can be executed by the software. It defines the constraints, relationships, and actions that underpin the flow of data within the system, ensuring that each operation adheres to the intended policies and delivers accurate results. The clarity and accuracy of business logic are essential for maintaining the reliability of software systems, as it directly influences how well the software aligns with the real-world scenarios it is designed to address. By formalizing business rules into structured logic, it enables organizations to automate and scale their operations effectively while minimizing the risk of errors and inconsistencies.

Existing method. The extraction of business logic from source code has been a long-standing challenge, especially in the context of legacy systems. Traditionally, reverse engineering techniques have been employed to bridge the gap between low-level implementation details and high-level conceptual models of software systems. Tools such as SOFT-REDOC have been developed to support this process, particularly for legacy COBOL programs [9]. These tools rely on program stripping, wherein non-essential code is eliminated to focus on the logic that directly affects specific business outcomes. This involves identifying critical variables and their assignments, conditions, and dependencies to reconstruct the underlying business rules.

Challenges with Existing Approaches. The reliance on human analysts to interpret outputs and dependencies makes the process time-consuming and error-prone [9]. Furthermore, legacy programs often involve convoluted logic and scattered assignments, making it difficult to reconstruct business rules with precision. In cases where variable names and data structures lack descriptive clarity, analysts may struggle to comprehend the program's intent, leading to incomplete or inaccurate extraction of business logic. These limitations highlight the need for more automated and scalable approaches to understanding business logic in modern and legacy systems.

Chapter 3
METHODOLOGY

3.1 Overview

The methodology chapter provides a comprehensive overview of the approach taken in this research. It outlines the key components of the system, including the Business Logic Analyzer module (BLA), the AI-integrated test generation module, and the AI test validation module. Each component is designed to work seamlessly together, leveraging Python, Flask, and Langchain to create an efficient and effective solution for automating test case generation. The chapter also discusses the methods implemented for each module and the plan to validate the generated tests from AI, ensuring that the proposed solution meets its objectives and addresses the identified challenges in software testing.

3.2 User requirement analysis

Understanding user requirements is a critical step in ensuring that the proposed system aligns with the needs and expectations of its target audience. This phase involves identifying and analyzing the specific functionalities, constraints, and preferences that users demand from the system. A thorough understanding of user requirements not only guides the development process but also ensures the system delivers value by addressing real-world challenges effectively. This section outlines the key user requirements identified for the proposed test generation service.

Req.ID	**Requirement Name**	**Detailed Description**	**Type**
001	Read project's source code	Users can send all project's source code at once via web-based Git repositories (e.g github, gitlab)	Functional requirement
002	Download/copy unit test/integration test	Users can download tests files or copy the file's content.	Functional requirement
003	Interactive business logic analyzation	Users can help AI correct the result of BLA process	Functional requirement
004	Performance	The system should generate test cases within a reasonable time frame, ideally under 5 minutes for a medium-sized project (e.g., 10,000 lines of code).	Non-functional requirement
005	Test file correctly reflect the given business model	The system should be able to generate test cases accurately reflect the business logic embedded in the source code.	Non-functional requirement
006	Validate generated test	A validation mechanism must be included to the system to ensure the syntax and logic is runnable	Non-functional requirement

Table 3.1: User requirements

3.2.1 Ability to send project's source code

The Test Genie system requires users to submit their project's source code via web-based Git repositories (e.g., GitHub, GitLab) rather than traditional methods like ZIP files. This design is intentional and aligns with modern development workflows since most modern projects have an online git repository. The biggest advantage is that this method will optimize unneeded directory that will be added to gitignore by users. Some modern framework use library that is sometimes heavy and not necessary during Business Logic Analyze process. Not adding these files will optimize the workloads of system much better.

User flow. Users will input the Git repository link via the User Interface (UI) and select the desired branch for analysis. If the system encounters access issues or cannot connect to the repository (e.g., internal Git systems), it will respond with an error message, prompting the user to resolve the issue.

System flow. After receiving the Git link and branch information, the system will clone the repository. Using predefined tokens or configuration files (e.g., pubspec.yaml for Flutter), the system will identify the framework and dependencies used in the project. Based on this information, the system will apply the most suitable strategy to analyze the source code and generate test cases.

3.2.2 Give user output

The output of the system is a full test file content that can be integrate into their existing workflows. The output is delivered through a live chat downloadable UI, ensuring a seamless and interactive experience for users.

Output format. Currently, this system only supports the Flutter framework, which has a built-in testing system. The system generates test files with the naming convention *"filename_test.dart"*, where the filename corresponds to the specific module or functionality being tested. This naming convention ensures that the test files are easily identifiable and organized within the project structure. The content of the test files is tailored to match the testing requirements requested by the user, including unit tests, integration tests, or widget tests, depending on the analysis of the source code. By adhering to Flutter's testing standards, the generated files are immediately compatible with the framework, allowing developers to run the tests without additional configuration. This approach ensures that the output is not only functional but also aligns with best practices for Flutter development.

Live chat interface. Users receive the generated test files through a live chat interface embedded in the system's UI. This interface provides a real-time, interactive experience, enabling users to communicate with the system as it generates and refines test cases. For example, if the user identifies an issue with the generated tests (e.g., incorrect logic, missing edge cases, or mismatched parameters), they can provide feedback directly through the chat. The system will then process this feedback and adjust the test cases accordingly. This two-way communication ensures that the final output meets the user's expectations and aligns with the project's requirements. Additionally, the live chat interface can provide explanations or suggestions for improving the tests, making it a valuable tool for both novice and experienced developers. This interactive approach enhances user satisfaction and ensures that the generated tests are accurate and relevant.

Downloadable Files. Instead of requiring users to manually create and organize test files, the system allows users to download the generated files directly and save them in the /tests/ folder of their Flutter project. This feature eliminates the need for manual file creation and ensures that the tests are placed in the correct directory, adhering to Flutter's project structure. The files are packaged in a format that is ready to be integrated into the user's project, requiring minimal manual intervention. This seamless integration reduces the risk of errors and saves developers' valuable time. Furthermore, the system ensures that the downloaded files are compatible with version control systems like Git, allowing users to immediately commit the tests to their repository. This feature is particularly useful for teams working in collaborative environments, as it streamlines the process of adding tests to the codebase.

Easy to adjust. Although the system is embedded with a validator to ensure that the generated tests are syntactically correct and runnable, it recognizes that real-world scenarios may require adjustments. For instance, the system might generate tests based on default parameters or assumptions that do not fully align with the user's specific use cases. In such situations, users can easily adjust the test parameters to better fit their requirements. The system provides clear and well-structured test files, making it straightforward for developers to modify variables, inputs, or assertions as needed. This flexibility ensures that the generated tests remain useful even in complex or unique scenarios. By combining automated test generation with the ability to manually refine the results, the system strikes a balance between efficiency and adaptability, catering to a wide range of development needs.

3.2.3 Interactive Business Logic Analyzating process

The Business Logic Analyzing (BLA) process plays a crucial role in ensuring that the system accurately interprets and applies business logic. If the output of this process is incorrect, it can lead to downstream malfunctions and errors, which can be costly and time-consuming to resolve. To address this, the system incorporates an interactive BLA process that allows users to collaborate with the AI to improve analysis results.

User interface. The interface for this process is designed to be intuitive and user-friendly, enabling users to interact with a visual representation of the project's modules, classes, and functions in the form of a graph. This graphical layout provides a clear overview of how different components of the application are interconnected and functioned. Users can inspect the analysis results by interacting with this graph, allowing them to identify potential issues or discrepancies in the current output.

One key feature of this interface is its ability to be manipulated by users. Through inspection, users can help guide the AI by highlighting specific areas of interest, providing context, or pointing out errors in the analysis. This interactive capability allows for a more precise and accurate understanding of how the business logic is being applied within the system.

Sytem flow. Once the project's source code has been submitted to the system, it undergoes an initial analysis phase that maps out the relationships between classes, modules, and functions. The system uses this information to generate a detailed breakdown of the project's structure and flow. After the analysis is complete, users receive access to a project insight webview that provides a comprehensive visual representation of how these components interact with each other.

This webview not only displays the flow of the project but also highlights any potential issues or areas where the business logic may require adjustment. The system ensures that this visualization is clear and concise, making it easy for users to understand and address any discrepancies in the analysis.

3.2.4 Optimize performance

The input of this system is the user's source code of the project they needed to generate. A study show that the average number lines of code (LOC) of a project with 90 functions will have 90,000 lines of codes [10]. From AI perspective, that is an enormous amount of input tokens. To handle these input lighter, these inputs will be split into blocks of component to analyze.

Splitting strategy. In this system, relational database will be used to store project's source code. Each component will contain the input, output, related component information and the predicted business logic of that component. This structured approach allows for efficient handling and analysis of large inputs while maintaining clarity and organization.

Quering component. The graphical webview that was introduced above will be contruct by query the connection of these component.

Performance overall. By organizing the input into blocks of component and using efficient querying mechanisms, the system optimizes its ability to handle large-scale projects without compromising performance. The use of a relational database ensures that data retrieval is both organized and efficient, reducing the likelihood of bottlenecks during analysis.

This approach not only enhances the system's capacity to process extensive codebases but also improves overall efficiency by minimizing redundant data storage and retrieval processes.

3.2.5 Good test file generation - Quality control

To ensure high-quality test file generation while maintaining the abstraction of the LLM model, this thesis adopts the Retrieval-Augmented Generation (RAG) technique. This approach involves embedding relevant project framework documents (currently focused on Flutter) and providing them as input to the model through structured prompts. By augmenting the model with specific, context-rich information, the system can generate test cases that better align with the framework's requirements and coding standards.

Provided documents. The documents supplied to the LLM are carefully selected to include essential information related to testing syntax, techniques, and best practices for the Flutter framework. These resources guide the model in generating syntactically correct and framework-compliant test cases.

User-side documents. Users have the option to provide supplementary documents and sample test files from their projects. This customization allows the system to learn and adhere to the specific naming conventions, organizational structures, and testing styles already established within the project.

3.2.6 Test validation

In this thesis, the validation scope focuses on ensuring that the generated test files are runnable within the intended development environment. Rather than validating the correctness of test outcomes or the business logic they cover, the emphasis is placed on generating test files that can be successfully executed without syntax or framework-related errors. To achieve this, a Software Development Kit (SDK) is embedded for each supported framework, with the initial implementation targeting the Flutter framework.

This SDK integration ensures compatibility with the framework's testing infrastructure, allowing the generated tests to be seamlessly executed as part of the development workflow. By embedding the SDK, the system can identify and address potential issues during the test generation process, such as missing dependencies or incorrect file structures, thereby increasing the reliability of the output. While the current scope does not extend to evaluating the correctness of test assertions or coverage, this foundational validation approach ensures that developers receive test files that are syntactically correct, executable, and immediately ready for further refinement or deployment within their projects. Future enhancements may involve integrating more advanced validation techniques, such as logic verification

3.3 System Design

Overall, this system have two separate implementation: User Interface (Frontend) and Application service (Backend), connected through API Gateway. In this thesis, we will focus on how the service and each component inside is design.

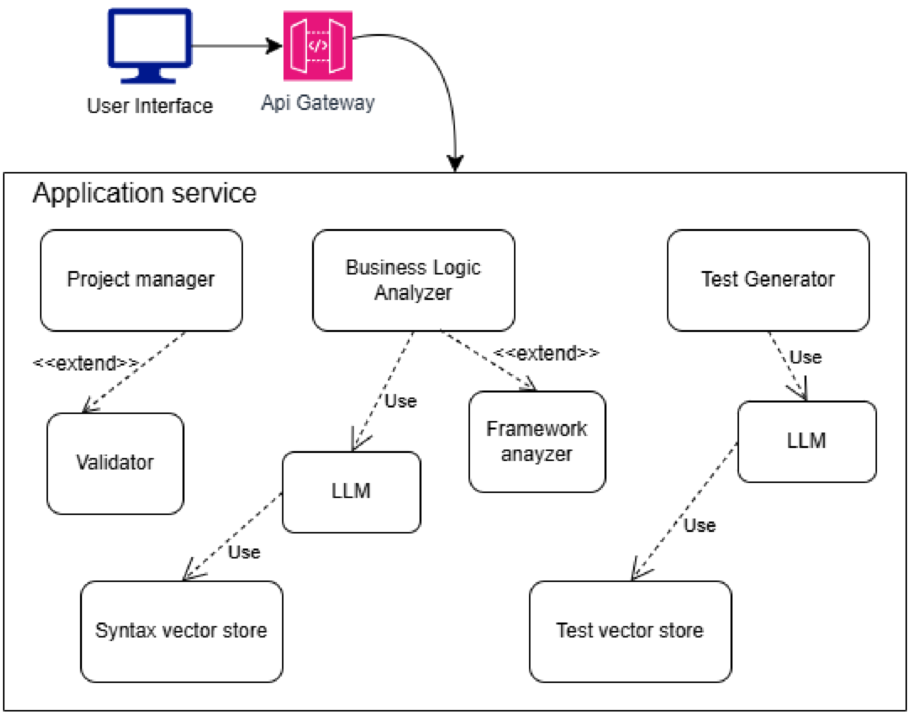

Figure 3.1: Test Genie's overall component design

Project manager. This component is abstracted by framework, handle anything related to the project's files. This component also interact with Git to clone the required project and responsible to use the SDK to validate existing tests.

BLA module. The BLA module plays a crucial role in analyzing the project's source code. It interacts with both project files and the database to break down the source code into components and identify relationships between classes and functions. These relationships are stored in a SQL database and visualized as component dependency diagrams to provide users with a clear representation of the project's structure. Analyzing strategy in this module will also be abstracted by framework.

Test Generator. This component communicates with Large Language Models (LLMs) to generate test files based on the analyzed business logic. It uses prompts and additional embedded documentation to produce runnable and framework-compliant test files that meet the project's requirements.

Block of component ERD. To analyze user's project easier, BLA module will firstly split the source files into components. These components will be connected together through different type of connection.

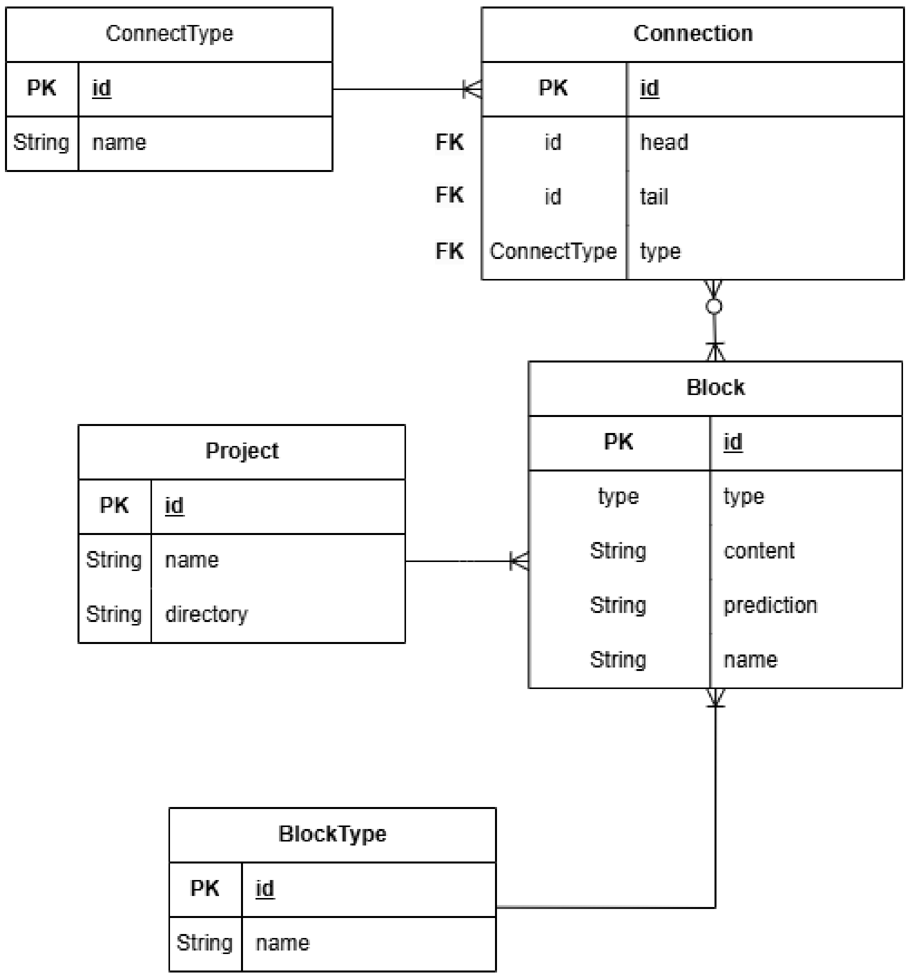

Figure 3.2: Block Relational Database Design

At the core of the diagram is the *Block entity*, which represents distinct units of the source code identified during the project analysis. Each components stores attributes such as its type, content, prediction, and a reference to the project it belongs to. The block entity will not need to store the id of its project because it is stored as system files in the server and Backend can access it directly. This kind of design will reduce the size of the database and optimize the performance of the system.

The *Connection entity* defines the relationships between blocks. It uses references to two distinct blocks: head and tail, forming a directional link between them. These connections are categorized by the ConnectType entity, which stores different types of connections that can exist between blocks, such as data flow, control flow, or dependency relationships. This architecture facilitates a comprehensive understanding of the project's code structure by mapping both the functional and logical connections between different blocks.

Furthermore, the *BlockType entity* is used to define the classification of blocks, storing various block types such as files, classes or functions. This separation of component types allows for better categorization and analysis. The Project entity ensures that each component is tied to a specific source code directory, while ConnectType maintains clarity by classifying relationships between them. This ERD structure enables the BLA module to effectively visualize and analyze complex relationships within user projects, making it easier to identify patterns and dependencies for test generation.

Just like *BlockType entity*, the *ConnectType entity* is used to define the classification of connections, storing various connection types such as Call, Contain, Use relationships, etc. This separation of connection types allows for better categorization and analysis. The Project entity ensures that each component is tied to a specific source code directory, while ConnectType maintains clarity by classifying relationships between them. This ERD structure enables the BLA module to effectively visualize and analyze complex relationships within user projects, making it easier to identify patterns and dependencies for test generation.

Chapter 4
IMPLEMENT AND RESULTS

This chapter delves into the implementation of each module inside Test Genie system. Overall, this system consist of three main modules:

- **Project Manager**: This module manages all the projects that are cloned to server. It mostly responsible for file-based activities and running CLI for each project.

- **Business Logic Analyzer**: This module will take various source file from Project Manager and break the source code into smaller pieces (blocks). Then, it will analyze each block and determine what each block does and how it should be tested if possible. A test plan will also be generated for each block and save it to the database.

- **Test Generator**: This module will take the test plan from Business Logic Analyzer and generate a test case for each block. The generated test cases will be saved as files directly in the project source code on server and can be used to run the tests later (validation).

Additionally, this system also have **DBMS** module to control the database but this module will not be explained thoroughly in this chapter.

4.1 Project Manager module

The **ProjectManager** module serves as the core backend functionality for handling projects within the Test Genie system. It provides a robust framework for managing software projects by integrating Git-based repositories, file management, and testing workflows. The module is built around the *Project* class, which encapsulates essential functionalities such as cloning repositories, recognizing project frameworks, and managing project files. Additionally, it features an abstract interface for test creation, validation, and execution, allowing for framework-specific extensions of functionality. For instance, the *Flutter* subclass extends the *Project* class to handle Flutter-specific tasks, including dependency management, ‘pubspec.yaml‘ parsing, and test execution. By modularizing these functionalities, the **ProjectManager** module streamlines project handling and enhances the system’s scalability for various software development frameworks.

4.1.1 Module prequisites

This module require the SDK of supported frameworks to be installed standablone in folder *./SDKs* inside the module folder. This design not only allows the module to be easily extended and modifiled to support other frameworks, but also avoid more SDK installation on the server OS. Since the *Project* class (Listing A.1) just mainly control git management and file management, the subclass can freely control how the SDKs are used.

Subclass of Project are required to implement the following methods:

- **create_test**: This method will create the test file in the location that is required by the framework.

- **get_test_content**: This method will return the content of the test file that is created by the **create_test** method. The content of the test file is generated by the Business Logic Analyzer module.

- **run_test**: This method will run the test file that is created by the **create_test** method. The test result will be returned to the caller.

- **validate**: This method will run all the test files in the test directory and return the result. This method is used to validate the test files that are created by the **create_test** method.

- **getListSourceFiles**: This is an important method, which will partly decide how the source code is split into blocks. The starting point file (main file) should be placed on the first position of the list. The list will be used to split the source code into blocks. The list should contain all the source files in the project (relative to the project directory).

4.1.2 Flutter class

The **Flutter** class extends the **Project** class to provide framework-specific support for managing Flutter projects. This class is responsible for handling operations unique to Flutter, such as managing dependencies, running tests, and validating projects. It ensures that the Flutter SDK is installed and properly configured in the *./SDKs/flutter* directory before performing any operations.

Key methods of the **Flutter** class include:

- **_runFlutterCLI**: This method executes commands using the Flutter CLI within the context of the project directory. It supports arguments for various Flutter commands and handles errors if the command fails.

- **_checkSDK**: Ensures that the Flutter SDK is installed and operational by running the `flutter --version` command. If the SDK is not present or misconfigured, the method raises an exception.

- **_flutterPubGet**: Automatically installs dependencies listed in the *pubspec.yaml* file by running `flutter pub get`.

- **_addTestDependency**: Adds the Flutter *test* package as a dependency using `flutter pub add test`.

- **create_test**: Creates a test file in the designated *test* directory of the project. If the file already exists and overwriting is not allowed, an exception is raised.

- **get_test_content**: Retrieves the content of a test file from the *test* directory.

- **run_test**: Executes a specified Dart test file using the Flutter CLI and returns the results.

- **validate**: Iterates through all Dart test files in the *test* directory and validates them by running each test.

- **getListSourceFiles**: Collects and returns a list of all source files in the *lib* directory, ensuring that the *main.dart* file is prioritized as the entry point.

This design enables seamless integration of Flutter-specific features into the **Test Genie** system while adhering to the modular structure defined by the **Project** class. By implementing these methods, the **Flutter** class ensures compatibility with the broader system and provides developers with a streamlined process for managing and testing Flutter projects.

4.2 Business Logic Analyzer module

The **Business Logic Analyzer** module is designed to parse and analyze the source code of a project. It constructs a **Dependency Diagram** that represents the logical structure and relationships within the project. By leveraging framework-specific analysis strategies, such as the *FlutterAnalyzeStrategy*, this module identifies functional blocks and their interconnections. Each block is further enriched with predictions generated by the **AI Agent**, which analyzes the code to provide insights into its behavior and logic. This modular design allows the Business Logic Analyzer to be easily extended to support additional frameworks, making it versatile and scalable for various software projects. The output of this module serves as the foundation for the subsequent test generation process.

4.2.1 DependencyDiagram class

Diagram objects

FlutterAnalyzeStrategy Algorithm

4.2.2 AI_Agent class

4.3 Test Generator module

4.4 Other implementations

Chapter 5
DISCUSSION AND EVALUATION

5.1 Analysis

Fusce vitae porta quam. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Ut tempus semper metus at mollis. In hac habitasse platea dictumst. Ut at placerat mi, et tempor lorem. Suspendisse ut ultricies orci. Phasellus eu gravida orci. Quisque mollis a est et accumsan. Ut fringilla sem lobortis, fringilla magna non, feugiat quam. Ut aliquet neque sit amet dolor feugiat congue. Curabitur imperdiet, massa ut varius semper, turpis lacus luctus risus, in imperdiet ipsum metus vel ligula. Fusce a ultricies odio. Pellentesque sit amet mollis lectus. Integer auctor posuere venenatis.

5.2 Strengths

Nulla sollicitudin imperdiet mattis. Morbi dignissim non leo ut ultrices. Proin sit amet ex nec libero luctus accumsan. Interdum et malesuada fames ac ante ipsum primis in faucibus. Morbi id convallis dolor. Nullam commodo nec diam sit amet mattis. Mauris lacus magna, elementum non velit sit amet, vulputate commodo nunc. Nunc nec neque euismod, consectetur dui aliquet, placerat lacus.

5.3 Limitations

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Proin nibh tortor, blandit a congue eu, commodo id lacus. Nulla facilisi. Nulla id leo justo. Maecenas placerat elit quis nisi congue, et semper lorem dapibus. Nam a sodales nibh. Cras convallis, sem sed sodales venenatis, turpis tellus feugiat enim, lacinia hendrerit metus libero sed risus. Etiam quis lorem purus. Quisque sapien velit, consequat eu molestie ut, interdum non sem. Nunc sollicitudin mauris quis purus blandit convallis. Cras sem eros, porta eget malesuada sed, efficitur sit amet dolor. Etiam faucibus purus lectus, et elementum sem posuere sit amet. Sed non nisl quis felis semper finibus sit amet sit amet neque. Phasellus nec mi laoreet, fermentum lacus et, efficitur est. Nunc sit amet vulputate sem, at venenatis libero.

5.4 Comparison

Duis dictum, orci a pharetra semper, lacus lorem varius velit, et pharetra ex magna ut justo. Ut eu luctus lacus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus bibendum volutpat risus, vitae pellentesque nisl faucibus ac. Quisque et facilisis magna. Quisque sed egestas libero. Ut pretium, velit in porta rutrum, purus justo convallis tortor, id convallis lorem felis eget eros. Vivamus eleifend interdum risus et sollicitudin. Donec quis erat et purus tempus dignissim non eget mi. Nam ac dapibus urna. Donec molestie sagittis faucibus. Etiam turpis odio, consectetur non dictum non, pellentesque ut nulla. Nunc viverra massa nulla, vel scelerisque tellus varius id.

5.5 Future Work

Vestibulum pretium rhoncus eros ut finibus. Vivamus mi nisi, mattis nec ipsum vel, feugiat tincidunt libero. Etiam eu sagittis diam. Praesent gravida turpis a iaculis dictum. Aenean tristique libero laoreet, tempor magna dignissim, rhoncus dui. Nullam erat elit, dictum dictum odio at, pharetra ultrices dolor. Vestibulum vulputate blandit urna. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Morbi cursus vel ex sed consequat. Maecenas fringilla ante et mi imperdiet fringilla. Morbi consequat ligula et elit facilisis, vel condimentum sapien ullamcorper. Mauris nisi urna, pretium eu consectetur ut, molestie at nulla. Curabitur viverra sagittis urna.

Chapter 6
CONCLUSION AND FUTURE WORK

Pellentesque eleifend iaculis massa, ut blandit ipsum viverra nec. Pellentesque dictum ut magna sit amet aliquam. Aliquam erat volutpat. Nullam vel eleifend ex, hendrerit porttitor magna. Quisque fringilla neque tincidunt fermentum consequat. Maecenas tincidunt ex magna, ut hendrerit ipsum convallis nec. Mauris tellus arcu, semper ac massa ac, sagittis feugiat tellus. Nulla maximus urna in commodo vestibulum. Nam imperdiet ligula egestas, dapibus dolor eleifend, pellentesque nisl. Integer mattis rhoncus posuere.

References

[1] Capgemini - “World Quality Report 2021-22”, Thirdteenth edition.

[2] Glassdoor - “Qa Tester Salaries”, https://www.glassdoor.com/Salaries/qa-tester-salary-SRCH_KO0,9.htm

[3] Herb Krasner, Consortium for information & Software quality - “The Cost of Poor Software Quality in the US: A 2020 Report”.

[4] Migual Gringerg - “Flask Web Development”, 2014, Published by O’Reilly Media, Inc.

[5] “Langchain”, https://python.langchain.com/docs/introduction/

[6] Abhishek Singh - “Essential Python for Machine Learning”, new edition 2023.

[7] Shreya Bhatia, Tarushi Gandhi, Dhruv Kumar, Pankaj Jalote - “Unit Test Generation using Generative AI : A Comparative Performance Analysis of Autogeneration Tools”

[8] Tosin Adewumi, Nudrat Habib, Lama Alkhaled & Elisa Barney ML Group, EISLAB, Luleå University of Technology, Sweden - “On the Limitations of Large Language Models (LLMs): False Attribution”.

[9] Harry M. Sneed, Katalin Erdos - “Extracting Business Rules from Source Code”, April 1996

[10] Adrian S. Barb, Colin J. Neill, Raghvinder S. Sangwan, Michael J. Piovoso - “A statistical study of the relevance of lines of code measures in software projects”, May 7, 2014.

Appendix A
LISTINGS

```
import os
import subprocess

class Project:
    _framework = ''
    def __init__(self, git_url):
        self._git_url = git_url
        self._name = git_url.split('/')[-1]
        # print('Project name: ', self._name)
        if self._name.endswith('.git'):
            self._name = self._name[:-4]

        # check if project already cloned
        if os.path.exists(projectDir + '/' + self._name):
            return
        else:
            self._clone(git_url)

    def _clone(self, git_url):
        # clone the git repository to the project directory
        try:
            # if Project folder not exist, create it
            if not os.path.exists(projectDir):
                os.makedirs(projectDir)
            return subprocess.check_output(['git', 'clone', git_url, projectDir + '/' + self._name], universal_newlines=True)
        except subprocess.CalledProcessError as e:
            raise Exception(f'Error cloning project: {e}')

    def recognizeProjectFramework(self) -> str:
        # TODO: Implement project framework recognition
        return 'flutter'
        pass

    def _setFramework(self, framework) -> None:
        self._framework = framework

    def getFramework(self) -> str:
        return self._framework

    def getName(self) -> str:
        return self._name

    def getPath(self) -> str:
        return projectDir + '/' + self._name

    def getFileContent(self, fileDir: str) -> str:
        """_summary_

        Args:
            fileDir (str): file directory relative to project directory
```

```

        Returns:
            str: file content
        """
        with open(os.path.join(projectDir, self.getName(), fileDir), 'r') as f:
            return f.read()
```

Listing A.1: Project class.

```
from ProjectManager import Project, projectDir, os, subprocess, sdkDir

sdkDir = os.path.join(sdkDir, 'flutter')

class Flutter(Project): # Inherit from Project class

    def __init__(self, git_url):
        super().__init__(git_url)
        self._setFramework('Flutter')
        self._checkSDK()
        self._flutterPubGet()
        self._addTestDependency()
        self.yaml_name = self._getYamlName()
        # self._createSampleProject('sample')

    def _runFlutterCLI(self, args, isRaiseException=False) -> tuple:
        prjDir = os.path.join(projectDir, self.getName())
        flutterBatDir = os.path.join(sdkDir, 'bin', 'flutter')

        cmd = [flutterBatDir]
        # args handling
        # if args is a string that have space, convert it to list
        if isinstance(args, str) and ' ' in args:
            args = args.split()
        if isinstance(args, list):
            cmd.extend(args)

        # run cmd via subprocess
        try:
            process = subprocess.Popen(cmd, cwd=prjDir, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True, encoding='utf-8', shell=True)
            stdout, stderr = process.communicate()
            if process.returncode != 0 and isRaiseException:
                raise Exception(f'Error running flutter command: {stderr}')
            return stdout, stderr
        except subprocess.CalledProcessError as e:
            if isRaiseException:
                raise Exception(f'Error running flutter command: {e}')
            return e.__dict__, e.args

    def _checkSDK(self) -> None:
        # Check if flutter sdk is installed
        if not os.path.exists(sdkDir):
            print('Flutter SDK not found')
            return
        # run sdk from sdkDir
        try:
            self._runFlutterCLI('--version', isRaiseException=True)
```

```python
        except subprocess.CalledProcessError as e:
            raise Exception(f'Error checking flutter sdk: {e}')

        # print(result)

    def _getYamlName(self) -> str:

        yamlContent = self.getFileContent('pubspec.yaml')
        # print(yamlContent)
        # first line should define the name of the project: "name: ....."
        return yamlContent.split('\n')[0].split('name: ')[1].strip()


    # function for testing only. Do not use in production
    def _createSampleProject(self, prjName) -> str:
        try:
            # cannot use _runFlutterCLI because no project directory yet
            # result = self._runFlutterCLI(['create', prjName], isRaiseException=True)
            result = subprocess.check_output([os.path.join(sdkDir, 'bin', 'flutter'), 'create', prjName],cwd=projectDir, universal_newlines=True, encoding='utf-8',  shell=True)

        except subprocess.CalledProcessError as e:
            raise Exception(f'Error creating flutter project: {e}')
        return result

    def _flutterPubGet(self) -> None:
        # prjDir = os.path.join(projectDir, self.getName())
        # flutterBatDir = os.path.join(sdkDir, 'bin', 'flutter.bat')

        try:
            # result = subprocess.check_output([flutterBatDir, 'pub', 'get'], cwd=prjDir, universal_newlines=True)
            self._runFlutterCLI(['pub', 'get', '--no-example'], isRaiseException=True)
        except subprocess.CalledProcessError as e:
            raise Exception(f'Error running flutter pub get: {e}')

        # print(result)

    def _addTestDependency(self) -> None:
        # run pub add test
        try:
            self._runFlutterCLI(['pub', 'add', 'test'], isRaiseException=True)
        except subprocess.CalledProcessError as e:
            raise Exception(f'Error adding test dependency: {e}')
        # print(result)

    def create_test(self, filename, content, isOverWrite = False) -> None:
        # create test file in the test directory
        # check if test directory exists
        testDir = os.path.join(projectDir, self.getName(), 'test')
        if not os.path.exists(testDir):
            os.makedirs(testDir)
```

```
        # check if file exists
        fileDir = os.path.join(testDir, filename)
        if os.path.exists(fileDir) and not isOverWrite:
            raise Exception(f'File {fileDir} already exists')
        # create file
        with open(fileDir, 'w') as f:
            f.write(content)

    def get_test_content(self, filename) -> str:
        # use getFileContent to get the content of the test file
        testDir = os.path.join(projectDir, self.getName(), 'test')
        fileDir = os.path.join(testDir, filename)
        if not os.path.exists(fileDir):
            raise Exception(f'File {fileDir} does not exist')
        return self.getFileContent(fileDir)

    # return tuple (result, error)
    def run_test(self, filename) -> tuple:
        fileDir = os.path.join('test', filename)
        try:
            result = self._runFlutterCLI(['test', fileDir])
        except subprocess.CalledProcessError as e:
            raise Exception(f'Error running flutter test: {e}')
        return result
        pass

    def validate(self) -> str:
        # run all tests in the test directory
        testDir = os.path.join(projectDir, self.getName(), 'test')
        for file in os.listdir(testDir):
            if file.endswith('.dart'):
                result, err = self.run_test(file)
                if err:
                    return err

        return ''

    def getListSourceFiles(self) -> list[str]:
            """_summary_

            Returns:
                list[str]: list of source files in the project relative to project directory
            """
            prjDir = os.path.join(projectDir, self.getName())
            libDir = os.path.join(prjDir, 'lib')
            sourceFiles = []

            # find main.dart first
            if os.path.exists(os.path.join(libDir, 'main.dart')):
                sourceFiles.append(os.path.relpath(os.path.join(libDir, 'main.dart'), prjDir))

            for root, dirs, files in os.walk(libDir):
                for file in files:
                    if file.endswith('.dart') and os.path.relpath(os.path.join(root, file), prjDir) not in sourceFiles:
                        sourceFiles.append(os.path.relpath(os.path.join(root, file), prjDir))
```

```
            return sourceFiles

    def __str__(self) -> str:
        return f'Flutter project {self.getName()} created from {self._git_url}'

    pass

```

Listing A.2: `Flutter` class - subclass of Project.

```
from ProjectManager import Project
from .Flutter import FlutterAnalyzeStrategy
from .AI_Agent import AI_Agent

class DependencyDiagram:

    blocks = []
    connections = []

    def __init__(self, project: Project) -> None:
        self.project = project
        self._generateDiagram()
        self.ai_agent = AI_Agent()
        self._getPredictions()

    def _generateDiagram(self) -> None:
        # Analyze project abstractly to project's framework
        framework = self.project.getFramework()
        functionName = framework + 'AnalyzeStrategy'
        if functionName in globals():
            globals()[functionName](self)
        else:
            raise Exception('Framework not supported')

    def _getPredictions(self) -> None:
        for block in self.blocks:
            block.setPrediction(self.ai_agent.generate_BLA_prediction(source_code=block.getContentNoComment(), chat_history=[]))

    def __str__(self) -> str:
        """_summary_

        Returns:
            str: project name, list of blocks and connections in the diagram
        """
        res = f'Project: {self.project.getName()}\n'
        res += 'Blocks:\n'
        for block in self.blocks:
            res += f'{block.name} - {block.type}\n'
        res += 'Connections:\n'
        for connection in self.connections:
            res += f'{connection.head.name} -> {connection.tail.name} - {connection.type}\n'
        return res
```

Listing A.3: `DependencyDiagram` class.