CHAPTER THREE
SYSTEM ANALYSIS AND DESIGN

3.1. PREAMBLE

This section details the analysis and design of the proposed system. It presents the requirements for the system, the techniques taken to achieve the system as well as diagrams to model the proposed system.

3.2. THE PROPOSED SYSTEM

This proposed project is a web-based application designed to streamline code review. Users can connect their codebases through repositories like GitHub or import their code directly into the application. Once the code is imported, the application conducts an extensive review, highlighting areas for improvement and offering recommendations. Additionally, the system provides explanations and justifications for these suggestions, utilizing intelligent responses from a language model.

3.3. REQUIREMENTS ANALYSIS

In software development, requirement analysis includes the descriptions of the services that would be provided by the system, as well as its operational limits. It lays down the features that are required to suit the users' requirements, these requirements are then further separated into functional requirements, non-functional requirements and user experience requirements. The various requirements for the proposed system are detailed in this section.

3.3.1. FUNCTIONAL REQUIREMENTS

Functional requirements of the system are specific functionalities or services that the system is expected to perform. They describe what the application should do and include operations, activities, computational tasks, data manipulation, user interface behavior, and more. These requirements are as follows:

i. Users should be able to input their code or to link their respective codebases from code hosting sites.

ii. Users should be allowed to select the coding conventions they want to adhere to.
iii. The system should analyze the code and provide feedback on areas that need improvement.
iv. The system should generate appropriate and contextually relevant explanations and justifications based on the content of the code and the suggestions made.
v. The system should allow users to view the code and the suggestions side by side.
vi. The system should allow users to accept or reject the suggestions made.
vii. The system should provide a summary of the code review.
viii. The system should allow users to download the reviewed code.

3.3.2. NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are not directly related to the system functionality but rather are defined in terms of system performance. They explain the system behavior, features and overall attributes that can affect the user's experience. The non-functional requirements for the system in this study are as follows:

i. The system should be user-friendly and easy to navigate.
ii. The system should seamlessly engage with code hosting repositories like GitHub for easy retrieval and pushing of code.
iii. The system should provide feedback within a reasonable time frame.
iv. The system should be able to provide explanations and justifications that are contextually relevant to the submitted code.
v. The system should be able to provide a summary of the code review.

3.3.3. USER EXPERIENCE REQUIREMENTS

User experience requirements are the factors that specify the expected things the user should encounter while making use of the system and the outcome of their experience. In the case of this system, the user experience requirements include:

i. The system should be easy to use and navigate.
ii. The system should provide clear and concise feedback.
iii. The system should provide explanations and justifications that are easy to understand.
iv. The system should provide a summary of the code review.
v. The system should allow users to download the reviewed code.

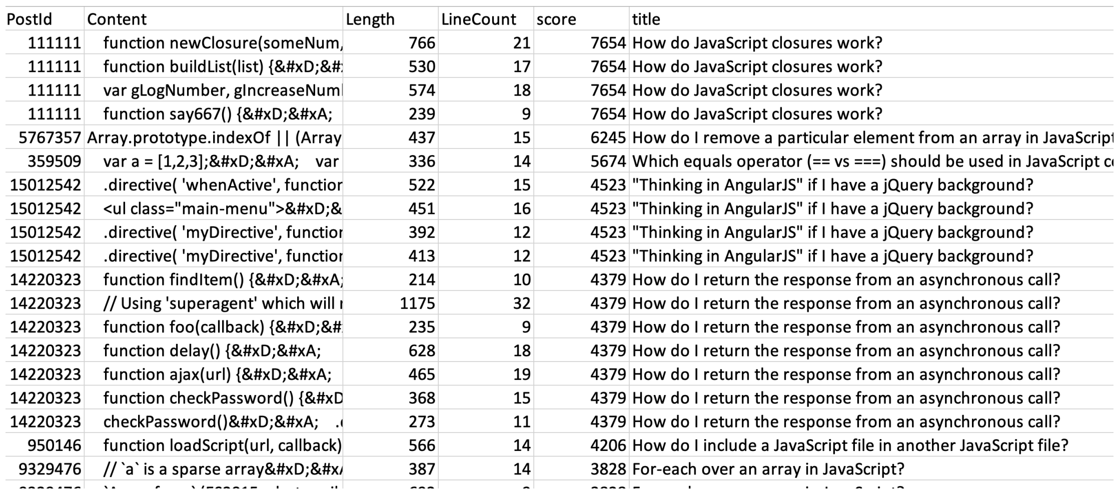
3.4. DATA COLLECTION

Two large datasets were used to re-train/fine-tune the Dolphin 2.5 Mixtral 8x7b model. The first dataset, MSR 2019, is a large dataset containing javascript code snippets extracted from Stack Overflow posts and the results of running ESLint to identify potential errors and stylistic issues (Ferreira Campos et al., 2019). The second dataset comprises parsed ASTs in JSON format used in the study by Raychev et al. (2016). These datasets are extensive, containing more than 480,000 JavaScript code files.

3.4.1. DESCRIPTION OF THE MSR 2019 DATASET

Ferreira Campos, Smethurst, Moraes, Bonifácio, and Pinto (2019) conducted a study to characterize the common coding violations in contemporary JavaScript code, resulting in the MSR 2019 dataset. It contains 336,000 JavaScript code snippets extracted from Stack Overflow posts. The code snippets were retrieved from SOTorrent (Baltes et al., 2019), a dataset that curates the extraction and evolution of Stack Overflow code snippets. The data was pre-processed and stored in a CSV file for further processing. The CSV file is in the following format:

i. ID of the question (PostId)
ii. Content (in this case the code block)
iii. Length of the code block
iv. Line count of the code block
v. Score of the post
vi. Title



PostId	Content	Length	LineCount	score	title
111111	function newClosure(someNum,	766	21	7654	How do JavaScript closures work?
111111	function buildList(list) {&#xD;&#	530	17	7654	How do JavaScript closures work?
111111	var gLogNumber, gIncreaseNum	574	18	7654	How do JavaScript closures work?
111111	function say667() {&#xD;&#xA;	239	9	7654	How do JavaScript closures work?
5767357	Array.prototype.indexOf || (Array	437	15	6245	How do I remove a particular element from an array in JavaScript
359509	var a = [1,2,3];&#xD;&#xA; var	336	14	5674	Which equals operator (== vs ===) should be used in JavaScript c
15012542	.directive('whenActive', functior	522	15	4523	"Thinking in AngularJS" if I have a jQuery background?
15012542	<ul class="main-menu">&#xD;&	451	16	4523	"Thinking in AngularJS" if I have a jQuery background?
15012542	.directive('myDirective', functior	392	12	4523	"Thinking in AngularJS" if I have a jQuery background?
15012542	.directive('myDirective', functior	413	12	4523	"Thinking in AngularJS" if I have a jQuery background?
14220323	function findItem() {&#xD;&#xA;	214	10	4379	How do I return the response from an asynchronous call?
14220323	// Using 'superagent' which will	1175	32	4379	How do I return the response from an asynchronous call?
14220323	function foo(callback) {&#xD;&#	235	9	4379	How do I return the response from an asynchronous call?
14220323	function delay() {&#xD;&#xA;	628	18	4379	How do I return the response from an asynchronous call?
14220323	function ajax(url) {&#xD;&#xA;	465	19	4379	How do I return the response from an asynchronous call?
14220323	function checkPassword() {&#xD	368	15	4379	How do I return the response from an asynchronous call?
14220323	checkPassword()&#xD;&#xA; .	273	11	4379	How do I return the response from an asynchronous call?
950146	function loadScript(url, callback)	566	14	4206	How do I include a JavaScript file in another JavaScript file?
9329476	// `a` is a sparse array&#xD;&#x	387	14	3828	For-each over an array in JavaScript?

Figure 3.9: A sample of the rows and columns of the pre-processed javascript code snippet dataset.

A simple Python script then processed the CSV file, extracting the code and merging all code blocks into their respective JavaScript files named after the PostID. This resulted in 336,000 Javascript files. Next, a script split the processed dataset into 20 evenly distributed parts and ran 20 instances of ESLint to generate reports, producing 20 JSON files.

3.4.2. DESCRIPTION OF THE 150K JAVASCRIPT DATASET

The second dataset, which was used to train and evaluate the DeepSyn tool Raychev, Bielik, Vechev, & Krause (2016), comprises JavaScript programs collected from GitHub repositories. This dataset was meticulously preprocessed to remove duplicate files, project forks, and obfuscated files. The programs were then parsed using the error-tolerant Acorn parser and subsequently tokenized and converted into Abstract Syntax Trees (ASTs) in JSON format. For training and evaluation purposes, the dataset was divided into training, validation, and test sets. The resulting dataset is a comprehensive collection of parsed AST serialized in JSON format, ready for analysis and application.

As an example, given a simple program to print “Hello World!” in javascript:

```
console.log("Hello World!");
```

Figure 3.10: Example javascript code

The serialized AST in JSON format is as follows:

```
[
    {"id":0, "type": "Program", "children":[1]},
    {"id":1, "type": "ExpressionStatement", "children":[2]},
    {"id":2, "type": "CallExpression", "children":[3,6]},
    {"id":3, "type": "MemberExpression", "children":[4,5]},
    {"id":4, "type": "Identifier", "value": "console"},
    {"id":5, "type": "Property", "value": "log"},
    {"id":6, "type": "LiteralString", "value": "Hello World!"}
]
```

Figure 3.11: Example of a serialized AST in JSON format of the chosen dataset

To evaluate the performance after tuning the Dolphin Mixtral model, we will utilize 50,000 files from this dataset.

3.5. PHYSICAL DESIGN

A physical design represents the physical elements of a software system and relates to the actual input/output operations of the system. The focus is on how data is entered into the system, validated, processed, and output. It has two major categories, the input design and the output design. For this system, the physical design is concerned with how the code is inputted into the system, how the system processes the code, and how the system outputs the results of the code review. These design considerations are essential to generate a working system that specifies all the features of a candidate system.

3.5.1. THE PROPOSED SYSTEM ARCHITECTURE

The system architecture describes the structure of the system and the components that make up the system. It provides a high-level overview of how the system will be designed and how the components will interact with each other. The system architecture for the proposed system is detailed in this section.

The proposed intelligent code review application is composed of several components designed to streamline and enhance the code review process. These components include the Code Importer or Fetcher mechanism (Github integration or manually importing code), a simple static analysis engine, a user-friendly interface for users, a recommendations module or generator (Dolphin Mixtral model), and a comprehensive reporting system. The functions of the components of the architecture are as follows:

Code Importer/Fetcher:
This component allows users to import their code into the system either by linking their codebases from code hosting repositories like GitHub or by manually uploading their code. This component serves as the entry point for the code review process.

Static Analysis Architecture
The system's static analyzer plays a crucial role in examining user-submitted code. It begins by tokenizing the code, parsing it, and constructing an abstract syntax tree (AST) to comprehend the code's structure and semantics. Through this analysis, it identifies code quality concerns, adherence to coding conventions, and areas for enhancement. The static analyzer encompasses several key components:

i. Lexer: The lexer tokenizes raw source code into tokens, facilitating further analysis.

ii. Parser: The parser constructs an Abstract Syntax Tree (AST) using a recursive descent parsing method, capturing code hierarchy systematically.

iii. Semantic Analyzer: The semantic analyzer verifies the code's semantic correctness, ensuring adherence to programming language rules and identifying logical errors.

iv. Code Rule Enforcement: This component checks code against coding standards, detecting stylistic issues and ensuring consistency.

v. Report Generation: Detailed reports are created, highlighting the identified issues, the AST, and code snippets. This report serves as a query that would be fed to the LLM for contextual recommendations or suggestions based on the violations found in the analysis.

User Interface

The user interface component provides an intuitive and user-friendly platform for users to interact with the system. It offers functionalities for code submission, viewing analysis results, accepting or rejecting suggestions, and downloading the reviewed code.

Recommendations Module

The recommendations module, powered by the Dolphin Mixtral model, generates contextually relevant suggestions based on the code analysis results. Leveraging the model's deep learning capabilities, the module provides nuanced feedback, explanations, and justifications for the identified issues. It offers actionable insights to users, guiding them toward code quality improvement and best practices.

Reporting System

The reporting system compiles the analysis results, recommendations, and justifications into a comprehensive report. This report summarizes the code review process, highlighting the identified issues, suggested improvements, and contextual explanations. Users are allowed to accept the suggested changes from this report. It serves as a valuable resource for users to understand the code quality concerns and the rationale behind the recommendations.

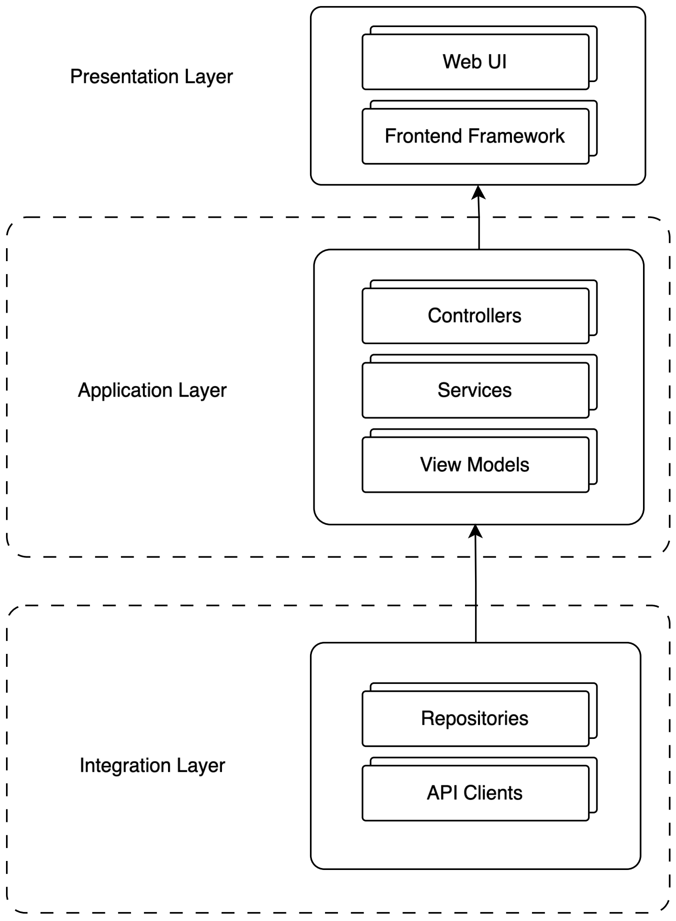


Figure 3.12: The Proposed System Architecture

The proposed system architecture consists of three layers. The Presentation Layer interfaces with the user, displays data, and handles inputs through a web-based front end built with the React Javascript framework. The Application Layer manages application logic, and workflows, featuring controllers for handling HTTP requests, and services for core functionalities like code importing from GitHub, static code analysis, generating recommendations with the Dolphin Mixtral model, and creating detailed analysis reports, alongside data transfer objects for formatting. The Integration Layer handles communication with external systems, including repositories for code interactions and API clients for integrating with services like GitHub.

3.6. LOGICAL DESIGN

The logical design of a system describes the functional requirements of the system and how these requirements are implemented. It refers to the conceptual and abstract representation of a system's architecture, functionality, and components. For the proposed system, the goal of logical design is to define the system's structure, data flow, interfaces, and interactions between various components in a way that is independent of any particular technology or platform. The logical design of the proposed system is detailed in this section.

3.6.1. USE CASE DIAGRAM

A use case diagram is a visual representation of the interactions between users and a system. It illustrates the various ways users can interact with the system and the system's responses to these interactions.

The use-case diagram for the proposed system shows that a user can interact with the system by importing their code, analyzing the code, viewing the analysis results, accepting or rejecting suggestions, and downloading the reviewed code. The system, in turn, processes the code, generates recommendations, provides explanations, and compiles a summary report for the user.

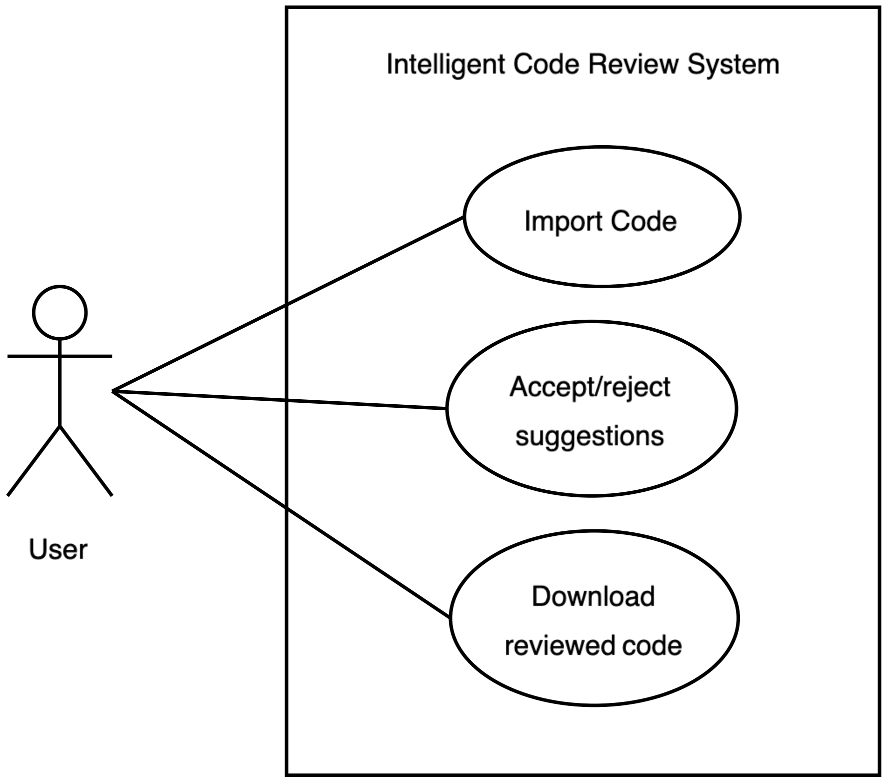
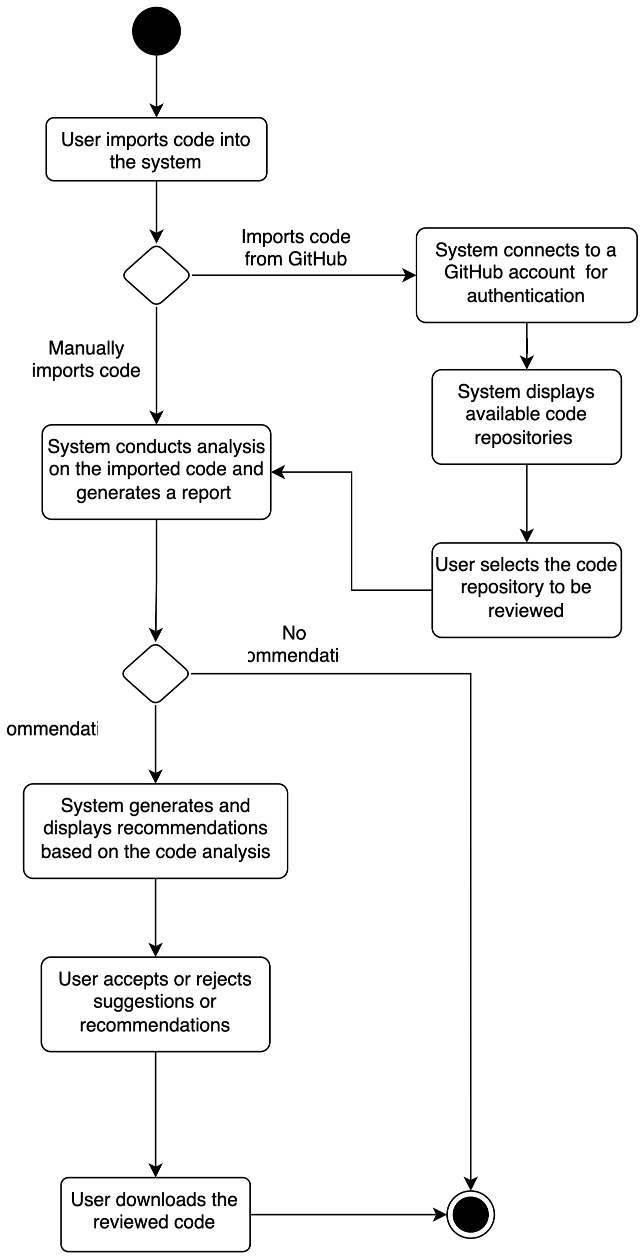


Figure 3.13: Use Case Diagram for the Proposed System

The use-case diagram provides a high-level overview of the system's functionality and user interactions, outlining the key features and capabilities of the proposed system.

3.7. ACTIVITY DIAGRAM

An activity diagram illustrates the flow of activities within a system, showing the sequence of actions and decisions that occur during a process. It provides a detailed view of how the system processes information and performs tasks, describing the steps in a use case diagram.



User imports code into the system
Imports code from GitHub
System connects to a GitHub account for authentication
Manually imports code
System displays available code repositories
System conducts analysis on the imported code and generates a report
User selects the code repository to be reviewed
No
ommendati
ommendati
System generates and displays recommendations based on the code analysis
User accepts or rejects suggestions or recommendations
User downloads the reviewed code

Figure 3.14: Activity Diagram for the Proposed System

3.8. SEQUENCE DIAGRAM

A sequence diagram shows how objects interact in a particular scenario of a use case. It illustrates the sequence of messages exchanged between objects, highlighting the order of interactions and the flow of control in a system.

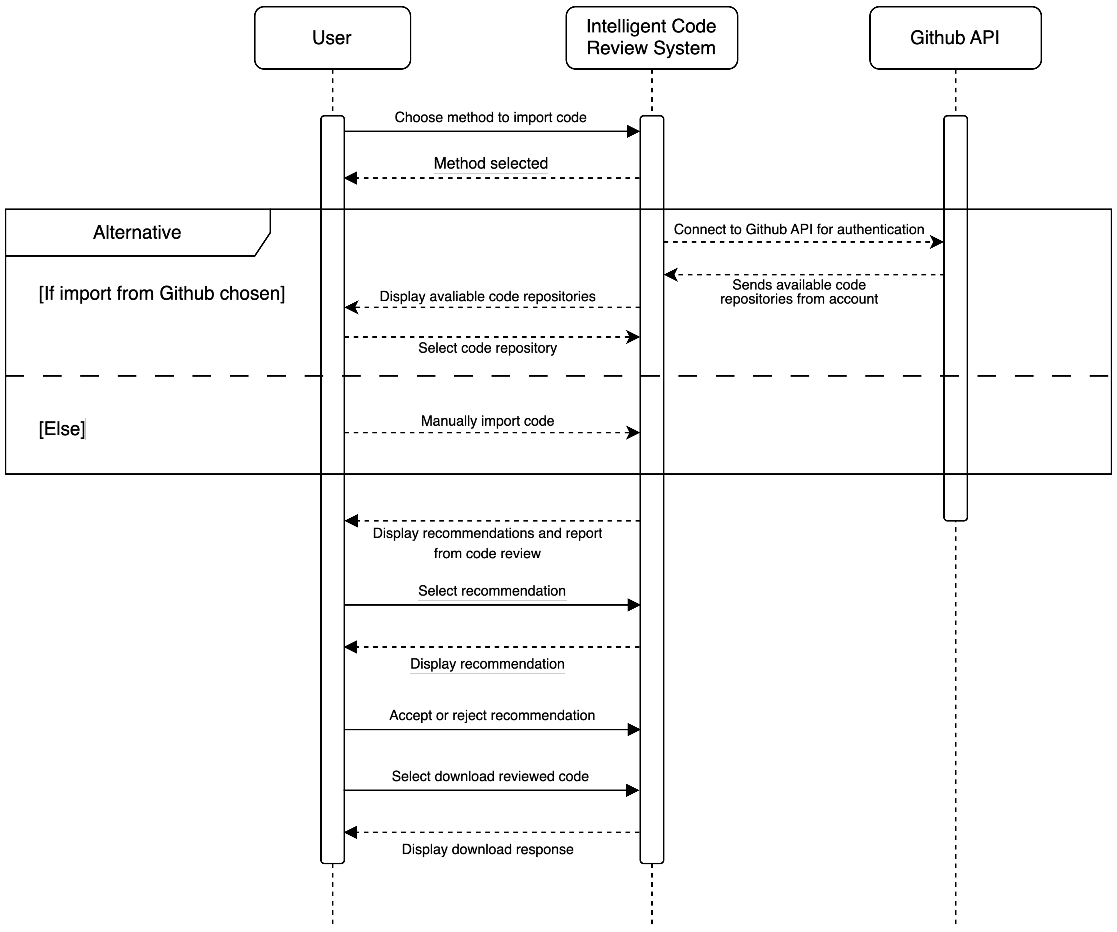

User
Intelligent Code Review System
Github API
Choose method to import code
Method selected
Alternative
[If import from Github chosen]
Connect to Github API for authentication
Sends available code repositories from account
Display avaliable code repositories
Select code repository
[Else]
Manually import code
Display recommendations and report from code review
Select recommendation
Display recommendation
Accept or reject recommendation
Select download reviewed code
Display download response

Figure 3.15: Sequence Diagram for the Proposed System

3.9. CONCEPTUAL DESIGN

The conceptual design of the system outlines the high-level structure and components of the system. This section shows the Entity Relationship Diagram (ERD) for the proposed system, detailing the entities and their relationships.

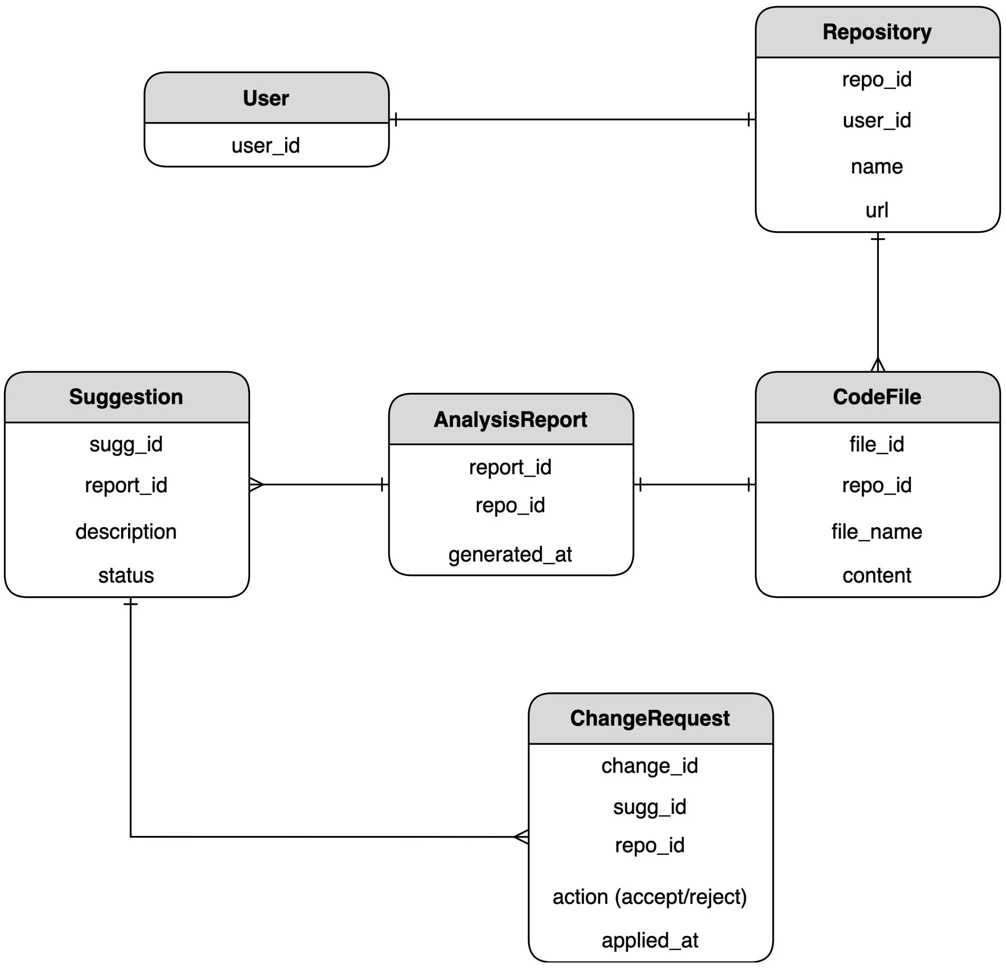


Figure 3.16: Entity Relationship Diagram for the Proposed System