Ain Shams University
Faculty of Computer & Information Sciences
Software Engineering Program
2022-2023

Automation Of Business Analyst Tasks (Generation Of UML Diagrams)

By

Team Members:

Tony Shereen Shawky Riad

Mai Adel El Saied Abdelshafy

Radwa Ibrahem Abdelrahman Ali

Mariam Nader Neseem Salama

Kiroloss Magdy Yassa

Under Supervision of

Dr. Mohamed Mabrouk

Information System Department,
Faculty of Computer and Information Sciences,
Ain Shams University

July - 2023

Acknowledgement

It gives us a great desire to acknowledge all those who supported us during the development process of this project.

First of all, our heartfelt gratitude goes out to Dr. Mohamed Mabrouk, our project supervisor, and our project proposal approver for his support throughout the project.

To all the academic and non-academic staff at FCIS for their support, and for allowing to continuing studies without any issues during the pandemic situation.

Last not least, all our colleagues at FCIS ASU for their extended corporation and inspiration from the beginning.

Abstract

Unified Modeling Language (UML) diagrams are in particular helpful for displaying the business requirements of any proposed system and assist in system design from the perspective of the end user. UML diagrams are created independently after the gathering of requirements in the software analysis process.

Due to the complexity of the business environment or the UML diagrams' technical capabilities, it takes a lot of time to draw these diagrams using current drawing tools, thus automated UML Diagram generation tool is necessarily needed. Automated UML Diagram generation tools can be identified with the amount of time spent on requirement analysis and low-quality human analysis. This project's goal is to build a Software tool for producing UML diagrams(use case and class diagrams)from input user’s text files using natural language processing.

The final results obtained by the developed system is that the system is capable of generating use case and class diagrams from the input files of user stories and functional requirements by utilizing NLP and ML techniques and creating a usable trustworthy Web application by successfully generating services ,service details, and generated UML diagrams for each project that are all easily updated, validated, or deleted. Diagrams were drawn using the Plant UML tool along with simple, plain text language, making it possible for user interaction. The developed system can recognize important actors, use cases, and their relationships to various kinds of relationships for use case diagrams. As for Class diagrams, Important classes, their attributes ,and their relationships can also be recognized.

تعد الرسوم البيانية للغة النمذجة الموحدة (UML) مفيدة بشكل خاص لعرض متطلبات العمل لأي نظام مقترح وتساعد في تصميم النظام من منظور المستخدم النهائي. يتم إنشاء مخططات UML بشكل مستقل بعد تجميع المتطلبات في عملية تحليل البرامج.
نظرًا لتعقيد بيئة الأعمال أو القدرات الفنية لمخططات UML ، يستغرق الأمر وقتًا طويلاً لرسم هذه المخططات باستخدام أدوات الرسم الحالية ، وبالتالي فإن أداة إنشاء مخطط UML الآلي ضرورية بالضرورة. يمكن تحديد أدوات إنشاء مخطط UML الآلي مع مقدار الوقت المستغرق في تحليل المتطلبات والتحليل البشري منخفض الجودة. هدف هذا المشروع هو بناء أداة برمجية لإنتاج مخططات UML(استخدام الرسوم البيانية للحالة والفئة) من الملفات النصية لمستخدم الإدخال باستخدام معالجة اللغة الطبيعية.

النتائج النهائية التي حصل عليها النظام المطور هي أن النظام قادر على إنشاء مخططات حالة الاستخدام والفئة من ملفات الإدخال لقصص المستخدم والمتطلبات الوظيفية من خلال استخدام تقنيات NLPو ML وإنشاء تطبيق ويب جدير بالثقة وقابل للاستخدام من خلال إنشاء الخدمات والخدمات بنجاح التفاصيل ومخططات UML المُنشأة لكل مشروع والتي يمكن تحديثها أو التحقق من صحتها أو حذفها بسهولة. تم رسم المخططات باستخدام أداة Plant UML جنبًا إلى جنب مع لغة نص بسيطة وبسيطة ، مما يجعل من الممكن تفاعل المستخدم. يمكن للنظام المطور التعرف على الجهات الفاعلة المهمة ، وحالات الاستخدام ، وعلاقاتهم بأنواع مختلفة من العلاقات لاستخدام مخططات الحالة. بالنسبة لمخططات الفصل ، يمكن أيضًا التعرف على الفئات المهمة وخصائصها وعلاقاتها.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
▪ **API**	Application programming interface
▪ **BA**	Business Analyst
▪ **ML**	Machine Learning
▪ **NLP**	Natural Language Processing
▪ **NER**	Named Entity recognition
▪ **NLTK**	Natural Language ToolKit
▪ **SDLC**	Software development life cycle
▪ **UML**	Unified Modeling Language
▪ **POS**	Parts of Speech

Chapter 1
1. Introduction

1.1 Motivation

The software development life cycle (SDLC) is a crucial process in software engineering that involves several stages, including requirements gathering, design, implementation, testing, and maintenance. The design phase is particularly important, as it lays the foundation for the rest of the SDLC. In this phase, designers manually generate UML Diagrams, for instance Use Case and Class diagrams, from previously collected Requirements to model the system and its functionality.

UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques and shapes to create visual models of object-oriented software systems. It combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the SDLC especially design and analysis phase and across different implementation technologies.

Software systems and applications have become more complex in recent years, developers are finding it more challenging to build complex applications within short time periods. Even when they do, these software applications are often filled with bugs, it can take programmers weeks to find and fix them and nowadays companies are looking for satisfying their customers by delivering fast, high-quality software.

Thus, this project is motivated by the need to improve the efficiency and accuracy of the design phase of the SDLC. Automating the process of creating UML Diagrams, Use Case and Class diagrams, which consequently speeds up the requirements specification, thus speeding up the software design, so that the implementation process begins. We hope to assist a variety of users in

generating comprehensive diagrams in a fraction of the time it would take to create them manually

1.2 Problem Definition

The specific problem that our project addresses is the manual process of creating UML Diagrams from Requirements Specifications, which can be a time-consuming and error-prone process, especially for complex systems. Our project aims to solve this problem by automating the process of creating UML Diagrams, especially Use Case and Class diagrams. This will help users to speed up the SDLC process, reduce errors, and improve the accuracy of their diagrams.

1.3 Objective

The objective of our project is implement an automated tool for UML modeling(class diagram and use case diagram)that are created during design phase. This thesis project aims to develop an open-source Software Tool that can process user stories and functional Requirements and convert them into comprehensive UML diagrams by Implementing natural language processing (NLP) techniques to extract relevant information from user's input files. In Addition, auto-generating Use Case and Class diagrams that are consistent with the requirements of the system, instead of drawing them manually, thus improving the efficiency and productivity of the design phase of the SDLC. Our project is developed to provide a flexible, user-friendly, and open-source tool for user's profile and project edits.

This thesis project's main purpose is to develop a usable software tool to speed up the Design phase in Software Development process ,which in turn speeds up the whole Software Development process while providing services open-source, unlike other paid Software tools, for all users in order to save cost and effort.

Finally, the "Automation of Business Analyst task "seeks to enforce consistency and standardization by automating the application of predefined modeling guidelines and notations, ensuring accurate and compliant UML models. Additionally, the tool targets rapid iteration and updates, automatically reflecting changes in the system within the diagrams, enabling faster adaptation and keeping the documentation up-to-date.

1.4 Document Organization

Chapter 2: Background

This chapter contains a detailed description of the field of the project and all the scientific background related to the project, highlighting its significance. The chapter proceeds with a detailed survey of the work previously conducted in the field.

Chapter 3: System Architecture and methods .

This chapter includes a detailed description of system architecture, showing main components of the system and relations between these components , description of used methods and procedures.

Chapter 4:Implementation

This Chapter includes a detailed description of all the functions in the system , a detailed description of all the techniques and algorithms implemented and description of any new technologies used in implementation.

Chapter 5:User Manual

This Chapter describes in details how to operate the project along with screenshots of the project representing all steps.

Chapter 6: Conclusions and Future work

In this chapter, the main findings and contributions of the project are highlighted. It reflects on the limitations and potential areas for improvement of the UML automation modeling tool. It discusses challenges or constraints encountered during the project

Chapter 2
2. Background

2.1 Scientific background and field of the project

2.1.1. Software Development Life Cycle

Software Development Life Cycle (SDLC) [2] is the application of standard business practices to building software applications. It's typically divided into five to eight steps: Planning, Requirements Definition, System Design, Software Implementation, Testing, Operations and Maintenance. Some project managers will combine, split, or omit steps, depending on the project's scope. These are the core components recommended for all software development projects.

SDLC is a way to measure and improve the development process. It enables an in-depth investigation of every stage of the procedure. As a result, businesses are able to maximize efficiency at every stage. As computing power increases, it places a higher demand on software and developers. Companies have to reduce costs, deliver software more quickly, and meet or exceed customer expectations. The SDLC helps companies accomplish these goals by identifying inefficiencies and higher costs and fixing them to run smoothly. It also simply outlines each task necessary to assemble a software application, reducing waste and boosting the development process efficiency. Monitoring also ensures the project stays on track and continues to be a feasible investment for the company.

The main goal of the SDLC is to enhance organizational systems. This procedure often involves developing application software, as well as training staff on how to use it. Application software, often known as a system, was designed to support a particular organizational process or function, such as market analysis, payroll, or inventory management. The goal of application software is to turn data into information and for that to be achieved the structured approach of the SDLC, shown in Fig 2.1, is a four-phased approach to identifying, analyzing, designing, and implementing an information system.

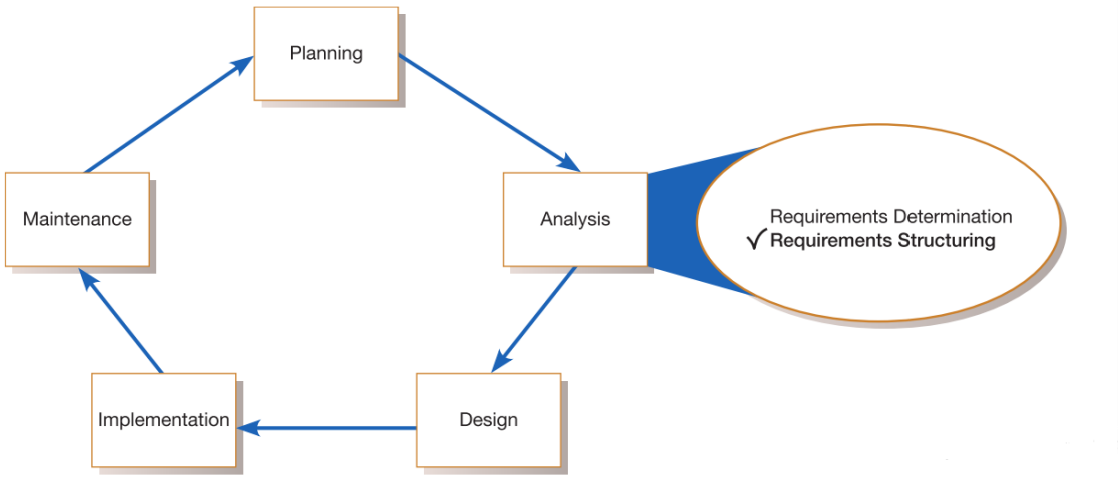

Figure 2.1 Software Development Life Cycle

1. Planning

In the Planning phase, project managers evaluate every aspect of the project. This involves estimating human and material costs, making a schedule with specific deadlines, and creating the project's teams and leadership structure. The systems analyst team then creates a detailed plan for the proposed project for the team to follow. In addition, stakeholders (everyone who stands to gain from the application) feedback is also set up, This baseline project plan describes the time and resources required for carrying out of the standard SDLC.

The scope and purpose of the application should be clearly defined as it maps the team's path and provides them with guidance for their second task in this phase. In addition, it sets boundaries to prevent the project from extending beyond its initial scope or changing its focus.

2. Requirement Definition

Defining requirements is considered part of planning to determine what the application is supposed to do and its requirements. For instance, a social media app would need to allow users to interact with friends. A search function may

necessary for an inventory program. Therefore, the business analysts or product owners collaborate with users and stakeholders to understand what they expect from the proposed system in order to begin a careful analysis of any existing systems, both manual and computerized, that may be replaced or improved as part of this project.

After studying the requirements, you organize them in a way that avoids duplication by considering how they relate to one another. You develop alternative initial designs as part of the structuring process that conforms to the specifications. The next step is to analyze these alternatives to see which one best satisfies the requirements while maintaining the organization's permitted cost, labor, and technical levels for the development process.

This phase's output is a description of the alternative solution the analysis team suggests. After the organization accepts the suggestion, you can make plans for purchasing any hardware and system software needed to construct or run the system as specified.

3. System Design

In the design phase, the system is described independently of any computer platform (logical design) and transformed into technology-specific details (physical design) for programming and system construction. Logical design is not tied to any specific hardware and systems software platform, and can be implemented on any hardware and systems software. It focuses on the business aspects of the system and its impact on functional units within the organization. During physical design, logical design is transformed into physical specifications, such as structured systems design, which can be broken into smaller components before being written in programming languages. The analyst team determines the programming languages, database systems, file structures, and hardware platform, operating system, and network environment for the system. This process ensures the system's functionality and functionality within the organization.

The final output of the design phase is the physical system specifications, presented in a form, such as a diagram or written report, willing to be shifted over to programmers and system builders for implementation. [1][2]

4. Software Implementation

In this step, the software code is actually written. A small project might be developed by just one developer, while a large project might be divided into multiple teams and worked by them all. Use a source code management or access control application during this phase. These tools aid programmers in keeping track of code modifications. They also support ensuring that various team initiatives are compatible with one another and ensuring that objectives are met.

Besides coding, there are numerous more tasks involved. Many developers need to work in teams in order to identify and fix errors and defects. Tasks often hold up the development process, such as waiting for test results or compiling code so an application can run. SDLC may anticipate these delays, allowing developers to be given other responsibilities.

Documentation may involve a formal process, such as wiring a user manual for the program. It may also be casual, such as comments in the source code that describe why a developer employed a specific technique. Even for companies who create simple software, documentation is important and helpful.
Documentation can be a quick guided tour of the application's basic features that display on the first launch. For difficult tasks, it may be helpful to take video lessons. Written documentation like user guides, troubleshooting guides, and FAQ's help users resolve problems or technical questions.[1]

5. Testing

Testing is crucial for an application's functionality and user satisfaction. It begins early in the SDLC, with a master test plan developed during analysis, followed by unit, integration, and system test plans during design phase. These plans are implemented during implementation phase, ensuring the system performs as expected. The Overall Plan and Testing Requirements outline a baseline project plan for testing, with a schedule of events, resource requirements, and standards of practice outlined . Procedure Control outlines the testing process, including documentation of errors and changes. Testing ensures that each function works correctly, and different parts of the application work seamlessly together, reducing

hangs and lags in processing. testing phase helps reduce bugs and glitches, leading to higher user satisfaction and better usage rates.[1]

6. Operations and Maintenance.

The maintenance phase of the SDLC is crucial step as it involves resolving bugs and resolving errors that may lead to new development cycles. The systems development group collects maintenance requests from users and interested parties, such as system auditors, data center and network management staff, and data analysts. Once collected, each request is analyzed to better understand how it will alter the system and what business benefits and necessities will result from such a change. If the change request is approved, the changes are designed and implemented, and formally reviewed and tested before installation into operational systems. The maintenance phase is a subset of the entire development process, resulting in the development of a new version of the software and updated design documents. This process represents the deliverables and outcomes of the entire development process, including the system itself.

2.1.2. User stories and Functional Requirement:

User stories [3] are a technique used in software development to capture and communicate the requirements of a system from the perspective of its users. A user story is a simple, one or two sentence statement that describes a specific feature or functionality that the user wants from the system. User stories are typically written in a format that follows the template:

- As a (**who** wants to accomplish something)
- I want to (**what** they want to accomplish)
- So that (**why** they want to accomplish that thing)

For example, a user story for a social media platform might be:
"As a user, I want to be able to comment on posts, so that I can express my opinions and engage with others."

User stories are typically used during the requirements gathering phase, where they can be used to capture and document the requirements of the system in a way that is focused on the needs of the user; user stories are often used in conjunction with other requirements gathering techniques, such as use cases or functional requirements documents.

User stories can be translated into functional requirements documents as part of the software development process. User stories are typically used to capture high-level requirements. Functional requirements documents, on the other hand, are typically more detailed documents that describe the specific features and functionality that the system must have in order to meet the needs of the user. The process of translating user stories into functional requirements documents typically involves breaking down the user stories into smaller, more detailed requirements, and documenting these requirements in a structured way. This might involve creating a use case diagram or flow chart that documents the specific steps and interactions involved in each user story.[3]

Functional requirements documents may include descriptions of individual features or functionality, use cases, or user stories, as well as additional details such as performance requirements, security requirements, and usability requirements. Functional requirements are important because they provide a clear and specific description of the software system's intended behavior, and help ensure that the system is designed and developed to meet the needs of its users and stakeholders. Functional requirements should be clear, concise, and testable, and should be written in a way that is understandable by all stakeholders, including developers, testers, and project managers.[4]

2.1.3. UML Diagrams

UML (Unified Modeling Language) diagram is a graphical representation of a software system or a process that uses standardized symbols and notations to depict the structure, behavior, and relationships of the system's components. The most important types of UML diagrams, include:[9]

1.Class Diagram:

Shows the classes and their relationships in a system, including attributes and methods. Main elements in Class Diagrams are :

- Classes: A list of the objects or entities that make up the system being modeled.
- Attributes: A list of the properties or characteristics of each class, such as name, type, and visibility.
- Methods: A list of the behaviors or actions that each class can perform, along with their parameters and return types.
- Relationships: A description of the relationships between the classes, such as "class A uses class B" or "class C extends class D".

2.Use Case Diagram:

Depicts the interactions between the system and its users or external systems. Main elements in Use Case Diagrams are :

- Actors: A list of the external entities (such as users or other systems) that interact with the system being modeled.
- Use cases: A list of the system's functions or services that the actors can perform.
- Relationships: A description of the relationships between the actors and use cases, such as "actor performs use case" or "actor initiates use case".

2.2 Scientific Background

Natural Language Processing :

Natural Language Processing is a theoretically motivated range computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications. The processing of natural languages is a difficult task, and it needs different techniques to be used than the

ones for processing artificial languages. Most requirements documents (e.g. User Stories) are written in NL rather than using modeling or structured techniques because it is much easier to be understood by all the stakeholders in the project. Besides that, NL allows the engineer to be as abstract or as detailed as required in a certain situation. On the other hand, NL can be ambiguous, leading to misunderstandings in the requirements specification.[7][8]

User requirement analysis can be an information extraction
application of Natural Language Processing. It will be identified specific semantic elements in the business requirements entered in textual form as mentioned. Therefore, this proposed system will come under information extraction over Information Retrieval. In natural language processing, several approaches were used to extract the information from the given text :

• Sentence splitting

With this approach all text will split into sentences.

• Lexical Analysis

This would get the split sentences and tokenize the sentences into words.

• Syntax analysis

This approach will receive the tokens generated by lexical analysis and applies a rule-based or machine-learning approach to generate and output parts of speech in a given text.

• Word chunking

By using this NLP approach, we can derive use cases from the input business requirement text. That will mostly be identify noun phrases, and verb phrases and also propositional phrases using tokenized text and POS tags.[7]

Basically, NLP can be viewed as a sequence of processing steps that starts from a raw text and proceeds through each higher level of representation. Under this approach the output of a lower level is the input for a higher level. Though there are some interdependencies, for simplicity each level is most often considered independently. This assumption greatly facilitates the identification of specific features at each level.[7]

2.3 Surveys

Based on surveys done on Sri Lankan, Singaporean and Australian IT Industries [5][6], two different surveys showed that some organizations use both UML as well as Non-UML Modelling methodologies (even if an Agile process is followed) to help design their software. Redundancy and unnecessary waste of time could be avoided, if other types of diagrams could be derived from one basic type. Some Important Survey results are given below:

According to the below figure, Figure 2.2, the majority (56%) seem to be using mainly UML Diagrams and other modeling techniques for visualizing their projects (Option b and c). Results also show that most of the system designers use User stories and entity Relationship Diagrams (ERD).[5]

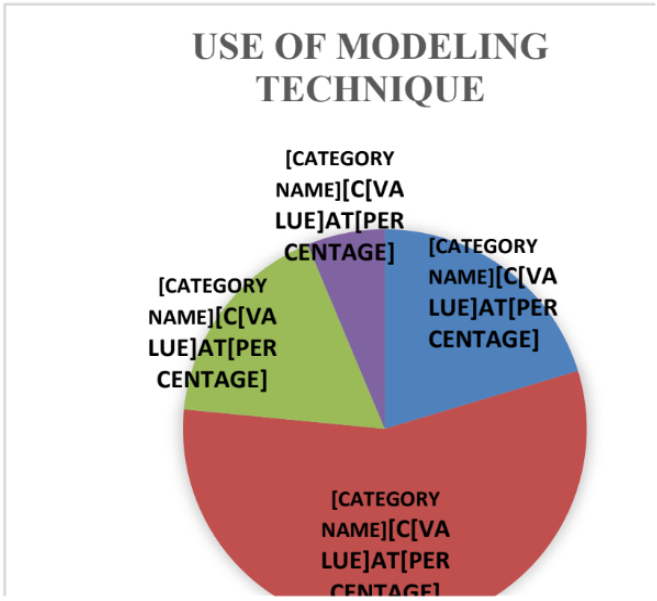

Figure 2.2 Modeling Techniques used for Software Engineering based on the Survey

Additionally in the Below table, Table 2.1 above shows the usage of UML diagrams based on a survey done exclusively on the Sri Lankan software development organizations. Out of the respondents who have answered Q1 by using UML Diagrams for Modeling, the majority seem to be using Class (86.5%) and Use Case Diagrams (86.5%) for their projects.[5]

UML Diagram	Usage	%
a. Use Case Diagrams	**32**	86.5%
b. Class Diagrams	**32**	86.5%
c. Sequence Diagrams	**21**	56.8%
d. State Diagrams	**13**	35.1%
e. Activity Diagrams	**20**	54.1%
f. Component Diagrams	**5**	13.5%
g.Composite Structure Diagrams	**1**	2.7%
h. Deployment Diagrams	**5**	13.5%
i. Object Diagrams	**3**	8.1%
j. Package Diagrams	**3**	8.1%
k. Profile Diagrams	**0**	0%
l.Communication Diagrams	**3**	8.1%
m.Interaction Overview Diagrams	**0**	0%
n. Timing Diagrams	**0**	0%

Table 2.1 Usage of UML Diagrams Based on the Survey

Thus, for our thesis project, we have given priority for the following popular diagrams and techniques for the initial stage to assist a large fraction of users.

1) Use Case Diagram –86.5% use Use Case diagrams

2) User Stories – From those who answered Q1 options b) and c), 55.1% use User Stories.

3) Class Diagram- of those who use UML diagrams 86.5% use Class diagrams.

2.4 Similar systems

There are many CASE tools for drawing the UML use case and class Diagrams, like Visual paradigm, StarUML and DrawIO are some of the famous software tools currently using in the industry, but all of these software tools are

generating UML Diagrams manually and using drag-drop approach. Other Software tools that automating the generation process like:

1. **SmartDraw:**

SmartDraw [10] is a diagramming tool that provides a feature for automatically generating UML diagrams, including use case and class diagrams, from text descriptions. The input format for auto-generating a use case and class diagram using SmartDraw may vary depending on the specific version and edition of SmartDraw you are using, but ,generally, the input consists of text descriptions of the various elements of the UML diagram. In addition, SmartDraw is a commercial, paid tool.

- For a use case diagram, the input might include descriptions of the actors, use cases, and the relationships between them in format:

Actor: [actor name]
Use Case: [use case name]
Description: [brief description of the use case]

- For a class diagram, the input might include descriptions of the classes, attributes, and methods in format:

Class: [class name]
Attributes: [list of attributes]
Methods: [list of methods]
Relationships: [list of relationships]

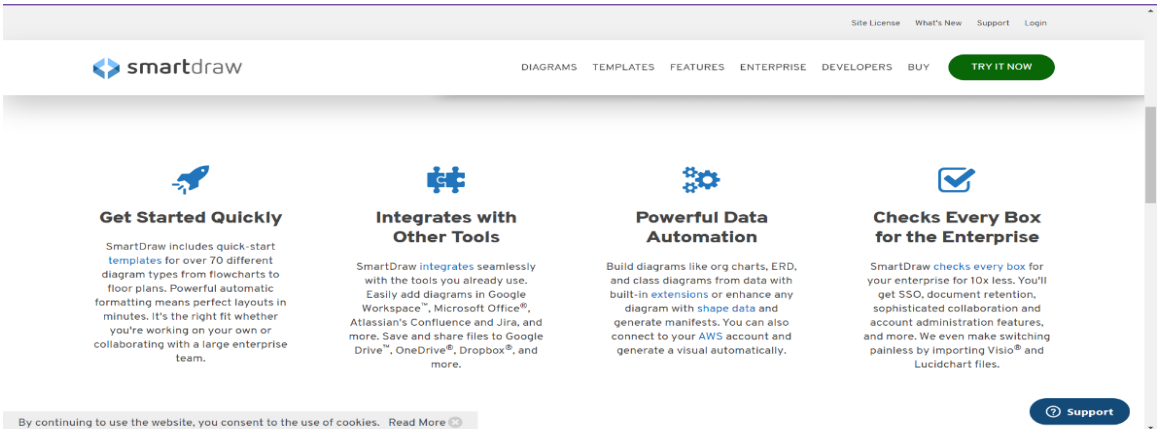

Figure 2.3 SmartDraw Software Tool

2. Rational Rose

Rational Rose [11] is a computer software tool that is widely used for object-oriented software development. Rational Rose was used primarily for modeling software applications using the Unified Modeling Language (UML), a standardized modeling language for software development. based on the information provided by the user or by reverse engineering existing code. Rational Rose is a paid, commercial tool and is not open-source. The cost of the tool may vary based on the licensing model, usage requirements, and other factors.

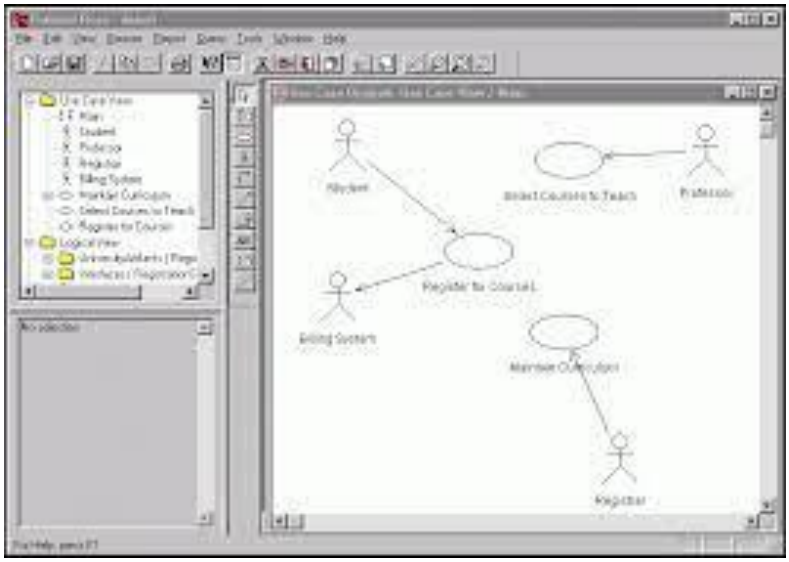

Figure 2.4 Rational Rose Software Tool

Chapter 3
3. System Architecture and methods

3.1 System Overview

3.1.1 System Architecture

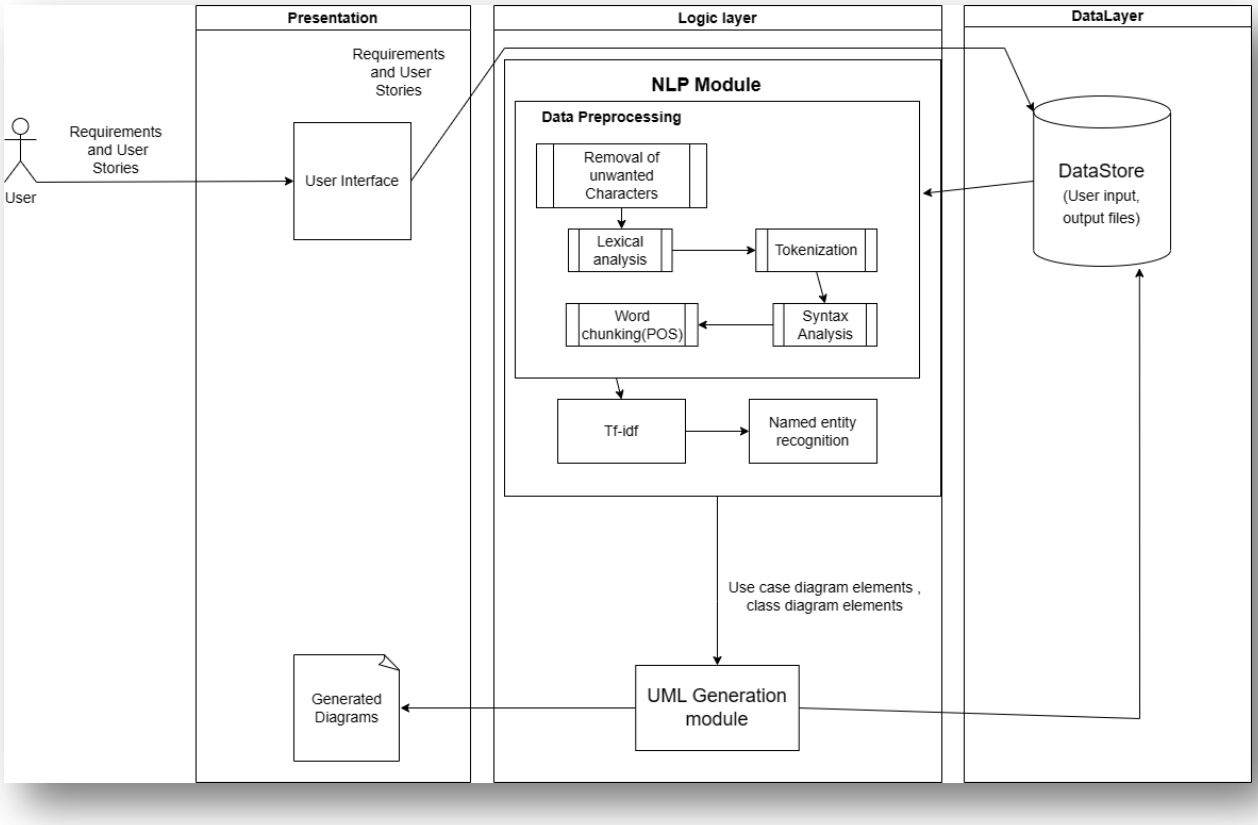

Figure 3.1 System Architectuure

Our project follows the Three-tier architecture pattern that is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the logic tier,

where data is processed; and the data tier, where the data associated with the application is stored and managed.

Presentation tier

The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect input and information from the user.

Application tier

The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed . In this layer all data processing and nlp techniques and algorithms are implemented.

Data tier

The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This is relational database management system NoSQL Database server.

In a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another but through the logic tier where some preprocessing and data validation is done.

3.1.2 Functional Requirements

1. User Sign-In:

Description: The user shall be able to log in by email and password to reach his Home Page.

Input: User data (Email, Password).

Processing: Authenticating the entered data using Firebase Authentication.

Pre-Condition: The user is already registered.

Post-Condition: The user will be signed in to his profile page.

Output: An error message appears in case of wrong email or password.

2. User Sign-Up:
Description: The user shall be able to register a new account in the system.
Input: User data (First Name, Last Name, Email, Password, and Role).
Processing: Check and authenticate if the entered data is valid using form validations and Firebase.
Post-Condition: Allow the user to log in into their account.
Output: An error message appears if the email already exists, if there is an empty field or if a field didn't satisfy its predefined constraint and a success message appears otherwise.

3. User Log out:
Description: The user shall be able to log out of the system easily.
Processing: End user's session in the system.
Pre-Condition: The user is already registered
Post-Condition: The user is no longer registered in the system and terminated his/her session.
Output: The user is redirected to the Home page.

4. Edit User Profile:
Description: The registered user shall be able to browse through their profile details, edit them, and save edited fields.
Input: User Data except Email (First Name, Last Name, Password, and Role).
Processing: Check if the entered data is valid, it will be valid if the data entered follows the predefined constraints.
Pre-Condition: The user is already registered in the system.
Post-Condition: Allow the user to update his/her profile with the edited fields.
Output: An error message appears if there is an empty field or if a field didn't satisfy its predefined constraint and a success message appears otherwise.

5. Add New Project:
Description: The registered user shall be able to add a new project to their profile.
Input: Project Data (Project Name, Company Name, Type, and Project Description).

Processing: Check if the entered data is valid and the project title doesn't already exist.
Pre-Condition: The user is already registered.
Post-Condition: Allow the user to save and successfully add the new project to their account.
Output: An error message appears if there is an empty field, otherwise a successful alert pop-up appears.

6. Edit Project Details:

Description: The registered user shall be able to browse through their project details, edit them, and save edited fields.
Input: Project Details (Project Name, Company Name, Type, and Project Description).
Processing: Check if the entered data is valid, then save those validated data into the database.
Pre-Condition: The user is already registered and there is a project in their account.
Post-Condition: Allow the user to update his/her project with the edited fields.
Output: An error message appears if there is an empty field and a successful alert pop-up appears otherwise.

7. Delete a Specific Project:

Description: The registered user shall be able to delete a project.
Processing: Delete the selected project from the database using Project identifier .
Pre-Condition: The user is already registered and the project already exists in the user's profile.
Post-Condition: The Project is Successfully Deleted and doesn't display in the user's projects pages.
Output: A validation pop-up appears and the user is redirected to his/her profile page.

8. Upload Input Files:

Description: The registered user shall be able to upload user stories and functional requirements files to his/her project
Processing: Check that Uploaded files are valid and check files constraint , if fit constraints, add files to the user's project input files.
Pre-Condition: The user is already registered in the system, the project already exists in the user's profile, and there this still space for uploading files.
Post-Condition: User's database is updated with the uploaded files.
Output: The User's project is updated by displaying the user's files in a table.

9. Delete File:

Description: The user shall be able to delete any file from his/her uploaded files from Database.
Processing: Delete the selected file from the database using File identifier.
Pre-Condition: The user is already registered in the system and is on a Specific project page containing the file he/her wants to delete.
Post-Condition: The File is Successfully Deleted and doesn't display to the user.
Output: A validation pop-up appears.

10. Generate Use Case Diagram:

Description: The user shall be able to generate a Use case diagram from the selected uploaded user story file.
Processing: Process the selected user story file, generate a use case diagram, and save it in the user's Database.
Pre-Condition: The user is already registered in the system and there exists an input user story file already uploaded.
Post-Condition: The use case diagram is Successfully generated.
Output: An output progress bar showing feedback on the process.

11. Generate Class Diagram:

Description: The user shall be able to generate a Class diagram from the selected uploaded functional requirement file.

Processing: Process the selected functional requirement file, generate a class diagram, and save it in the user's Database.
Pre-Condition: The user is already registered in the system and there exists an input functional requirement file already uploaded.
Post-Condition: The Class diagram is Successfully generated.
Output: An output progress bar showing feedback on the process.

12. Export UML Diagrams:
Description: The registered user shall be able to export the generated use case or class Diagrams into PNG and Text Files.
Processing: Generate PNG and Text files (PlantUML file) .
Pre-Condition: The user is already registered and there is a validated service (generated diagrams exist) in his/her project.
Output: PNG and Text files for the user.

3.1.3 Non functional requirements

Capacity:

- The system shall have a capacity of 10 GB of data at least.
- The system user's connection pools shall handle 50 users at the same time.

Documentation:

- The system shall have a user manual that explains the functions of the System.

Modifiability:

- The system shall be able to accept the addition, and change of services and modules.

Fault Tolerance:

- The system shall have all functions implemented as services within a service-oriented architecture to allow the system to operate in the event of one or more services failing.
- The system must have Error handling features.

Privacy:

- The system shall protect the privacy of individuals identified in a collection.

Reliability:

- The Web application will be available almost all the time, and there are no page crashes and freezes.
- The system shall be available 99.9% of the time.

Access Control:

- The system shall maintain unique user identification for every person who will use the system.
- The system shall maintain a password for every unique user identification on the system.
- The system shall authenticate users while signing up .

Scalability:

- The application should be designed to deal with the increasing use and any size of data without errors, system failures, or degrading in its performance or reliability.

Usability:

- All users shall be able to use all system functionalities smoothly without the need of spending hours of training, and quickly reach their goals.
- The website shall be suitable and compatible with all Web screen device sizes.
- Feedback: The system shall inform the user what has been done.
- Simple: Attributes of an object that make its usage must be clear and understandable.
- Consistency: The system shall be externally and internally consistent.

3.1.4 System Users

A. Intended Users:

The intended user for this system, specifically, is the business analyst. Generally, the system can also be beneficial for anyone developing a Software system, for instance, Product Owner, Student, or Designers, with different job and role titles.

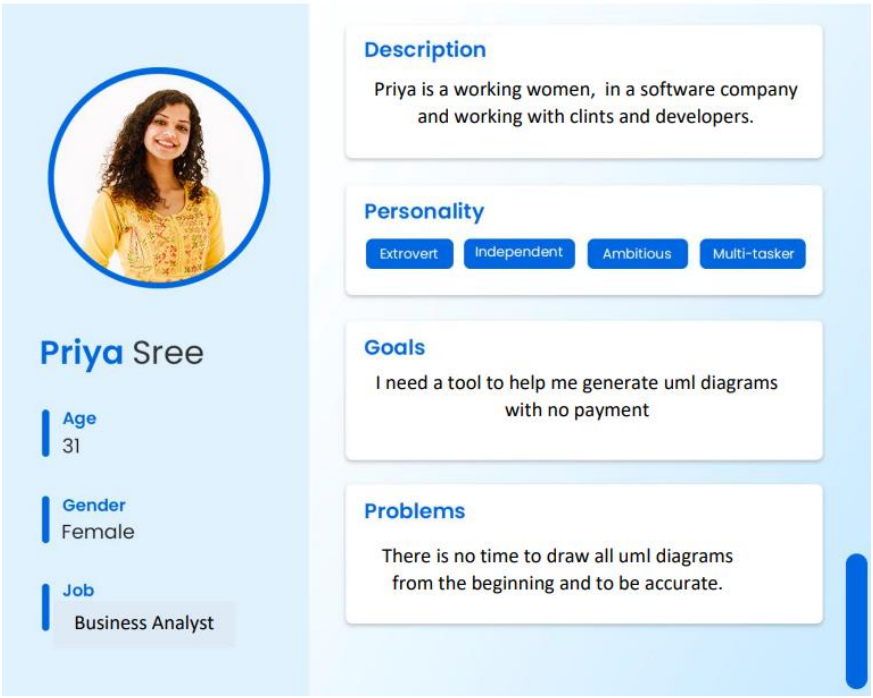

Figure 3.2 Intended User persona

B. User Characteristics

The user is expected to be qualified as a "business analyst" (beginner to expert level) who knows well how to gather stakeholders' requirements in user stories form, or anyone familiar with the SDLC phases, especially The Design phase, UML diagrams, and project documentation.

3.2 System Analysis & Design

3.2.1 Use Case Diagram

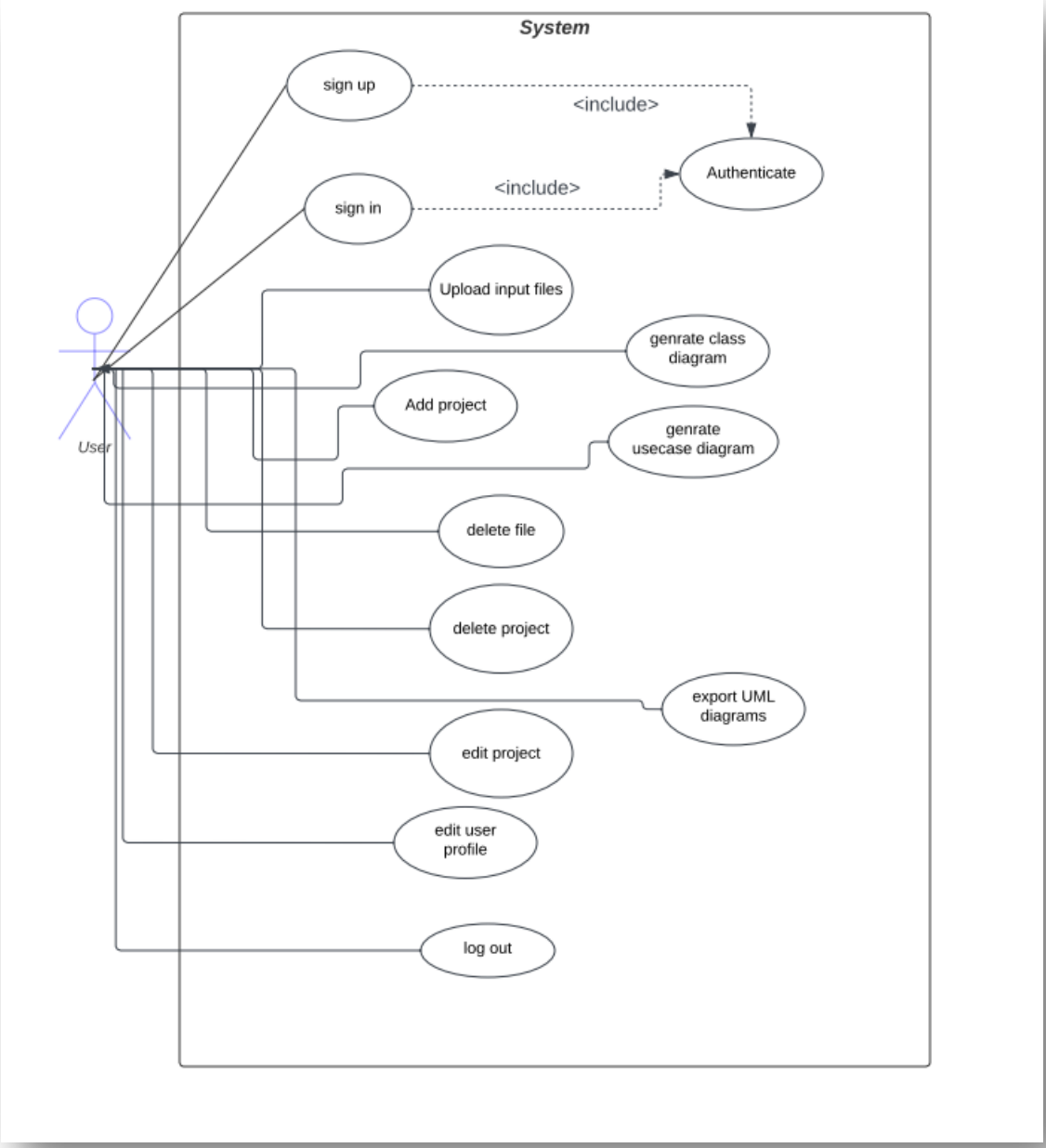

Figure 3.3 Use case Diagram

3.2.2 Class Diagram

Figure 3.4 Class diagram

Description of classes:

- Helper class: Responsible for the functions of getting the sentences from text files.
- UmlAutomationSystem class: Responsible for generating diagrams as use case and class diagram functions and also login and logout functions.
- AlgorithmsOfUseCaseDiagram class: A class of functions responsible for extracting actors, use cases and dependencies from for use case diagram.
- AlgorithmsOfClassDiagram class: A class of functions responsible for extracting attributes, methods, relations between classes as aggregation, inheritance and composition and also generalization.
- Actor class: Responsible for detecting the similarity between use cases to add use cases and dependencies to actors.
- PlantUML class: A super class that inherit from it UseCaseModel class and ClassModel class that responsible for draw the diagrams.
- UseCaseModel class: This class inherit from PlantUML class that responsible for adding use case model to the diagram.
- ClassModel class: This class also inherit from PlantUML class which is responsible for add classes, functions and dependencies to the class diagram.
- SharedAlgorithms class: The class prepare the input sentences and make preprocessing for words to remove duplicates and find synonyms.
- ClassEntity class: Adds the attributes and methods to classes.

3.2.3 Sequence Diagrams

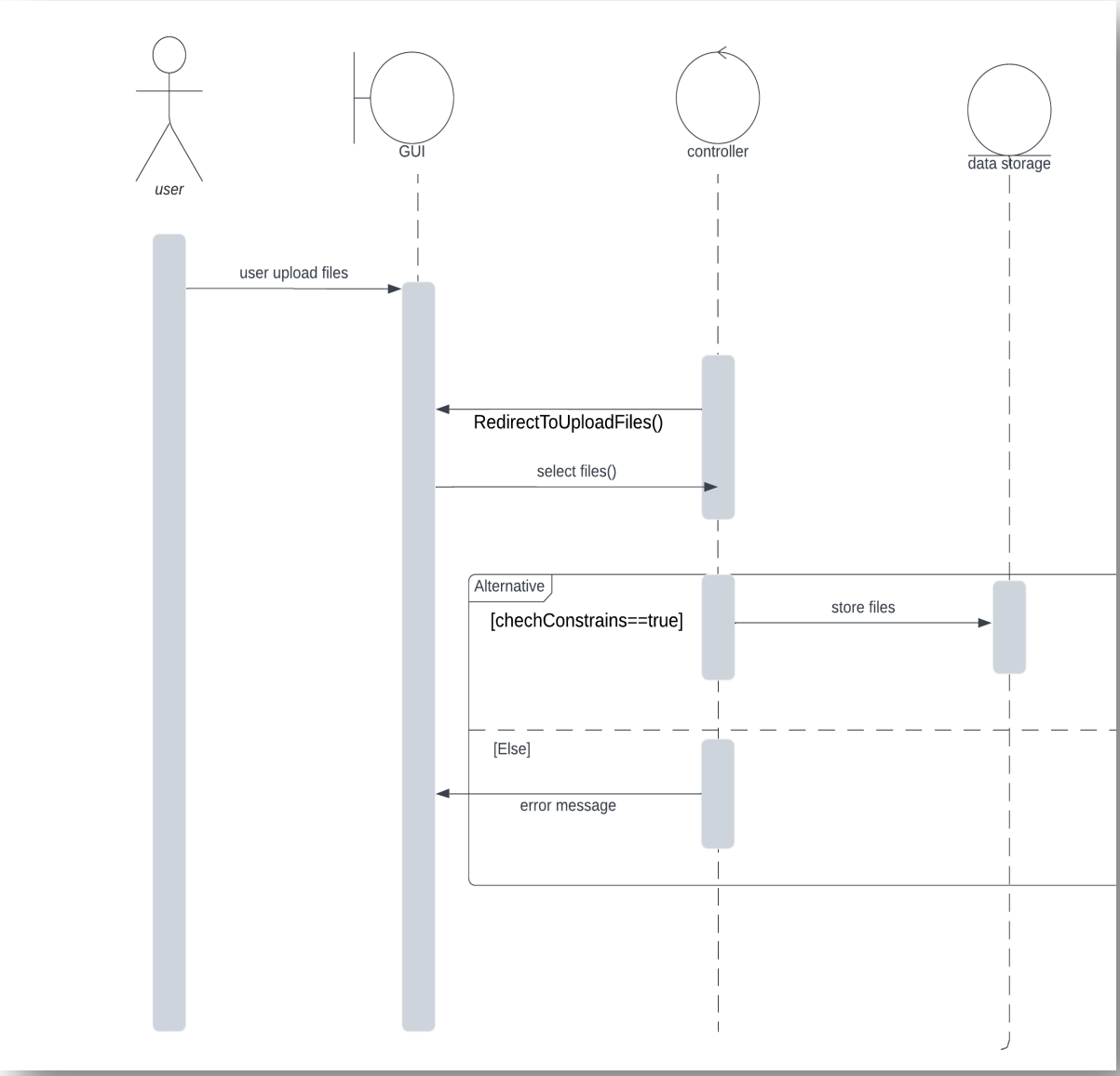

Figure 3.5 Sequence diagram for Upload Input Files

Upload Input File Sequence diagram

Generate class Diagram

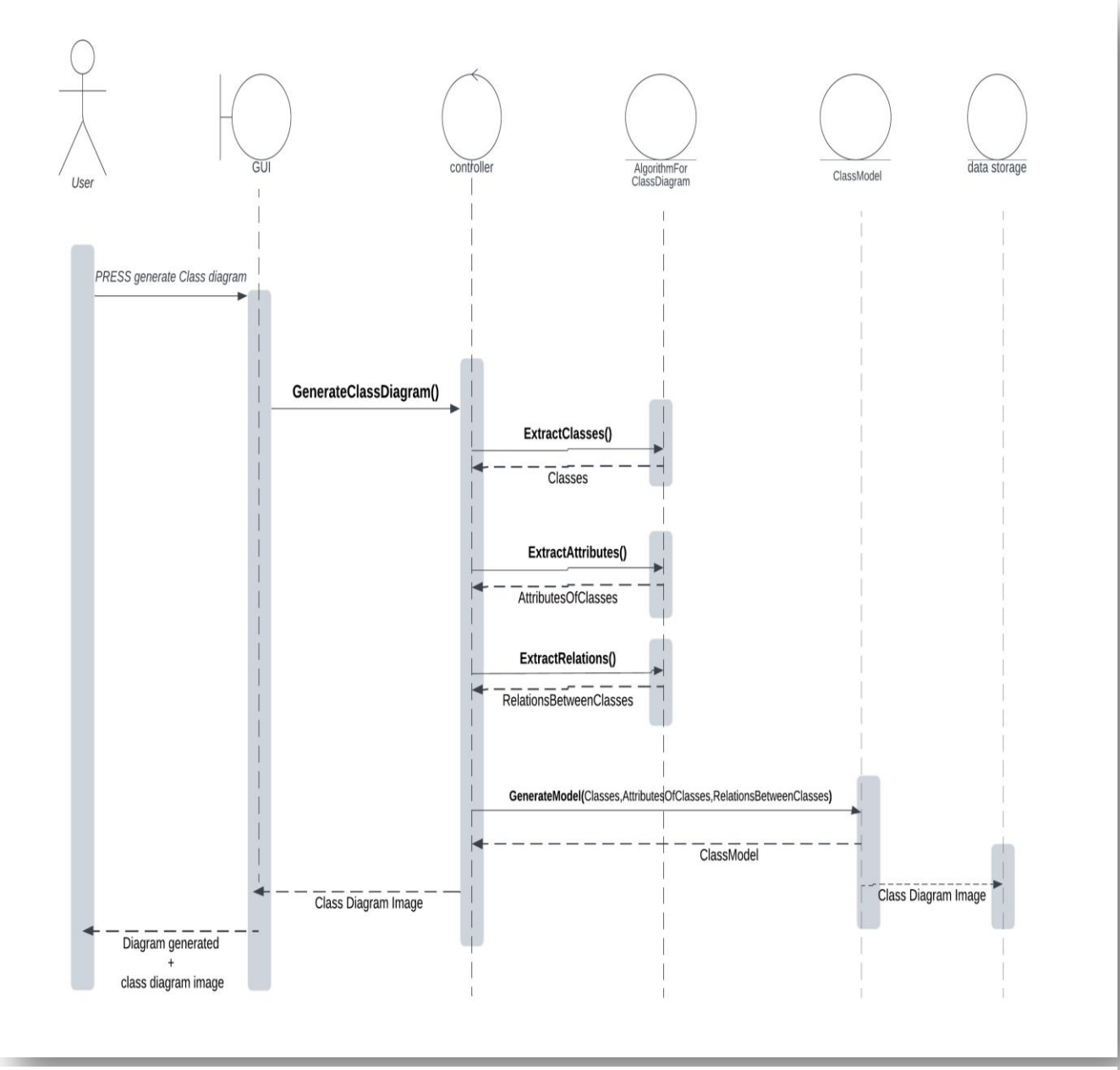

Figure 3.6 6 Sequence Diagram for Generate Class Diagram

3.2.4 Database Diagram

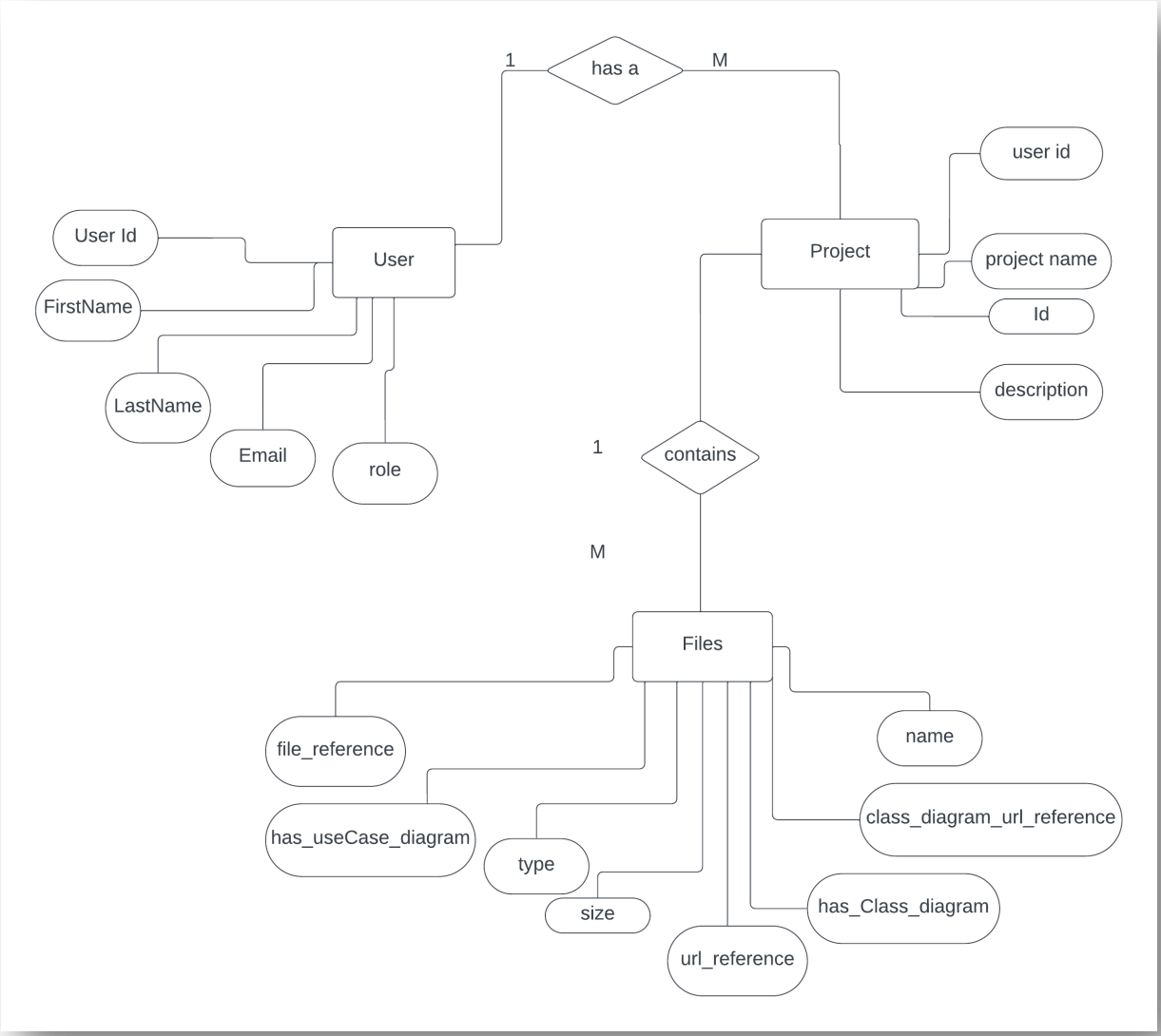

Figure 3.7 Database Diagram

Figure 3.7 represents the non-relational database diagram used in this project, it illustrates the documents structure of the project's database. The used database is Firestore which is NoSQL data model, used to store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that can be used to organize user's data and build queries.

User collection (table-like structure) has many documents. Each Document indicates a specific user and each one (users) has some fields as first name, last name, email, role all with a string data type.

In Addition ,there is another type of database has been used which is firebase storage, used to carry the documents(text files) entered to (User Input Files0 or generated from the system (UML Diagrams).

Chapter 4
4. Implementation

4.1. Detailed description of all the functions in the system

In order to use Plant-UML to render the text files into PNG, a class model or use case model should be written in certain syntax form in text file. Class model is implemented in terms of a class that has special functions. Each function types a text in a file in certain format in order to render this text file as an image by usage of Plant-UML.

4.1.1. User input

First step of implementing our project is to get input files with .txt extension from user, afterwards these files are saved to cloud firestore database.

4.1.2. File Preprocessing

Steps:

Step1:	User's input files (user story files or functional requirements) are cleaned which involves removing unwanted characters, punctuation, numbers, and special symbols that aren't relevant to the analysis. Regular expression ("re" module in python) are used for text pattern matching and removal.
Step2:	Stop words are removed to reduce noise and improve efficiency of downstream tasks. Spacy library provide predefined stop-word lists that can.

Step3:	Identifying sentence boundaries and splitting the whole text into list of sentences.
Step4:	Tokenization of each sentence is done; tokenization is the process of breaking the sentence into individual words or tokens. In Python, NLP library (SpaCy) is used in this step.

Table 4.1 Steps for File Processing

4.1.3. Generation of Use case diagram

4.1.3.1 Actors Extraction

Actors are extracted from sentences by means of noun chunks of SpaCy library. SpaCy's noun chunks extraction takes into account grammatical structures and dependencies in the text, allowing it to identify noun phrases accurately. User story is written in the following format : **Error! Reference source not found.**

As <role>, I want <feature> to <reason>

Since actors usually are the first noun found in any user story, Spacy noun chunks is used to extract actor where the first noun extracted is the actor.

4.1.3.2 Use Case Extraction

Use Case is extracted from user story as it's the feature mentioned in a user story .

Steps:

Step1:	**Tokens of each sentence(user story) .**
Step2:	Root verb is extracted by usage of POS tags . This root verb is compared to a list containing verbs that cant be a use case as “want, be, belong, able to”.
Step3:	Object of root verb is extracted by using spacy dependency utilities.
Step4:	By means of dependency parser , object dependents are extracted
Step5:	Use case is formed from the root verb and object and object dependents.

Table 4.2 Steps for Use Case Extraction

4.1.3.3 Use Case Extraction Relation Extraction

By the help of dependency subtree , relations are extracted between two use cases .Dependency subtree of tokens of root verb are extracted .These tokens of the subtree are compared with two words which are “after, depends on “ and what comes after these prepositions is extracted as another use case that is dependent on the root verb use case .

4.1.3.4 Generation of Use case diagram model as text file

After extraction process of use case diagram elements , a use case diagram model is written in certain syntax in a text file with the help of implemented use case model class that assists in typing the use case model in a text file.

4.1.3.5 Generation of Use case diagram image

After Use case model is written in text file in certain syntax. it passed to PlantUML tool to render it as image file that is displayed to the BA (user) at the end of processing.

4.1.4. Generation of class diagram

The primary task of this module is focusing on translating information extracted from text preprocessing module into useful conceptual modeling data. Semantic analysis is performed on preprocessed text to extract relevant information about UML class diagram candidate concepts. We have chosen rule-based NLP approach in order to extract class diagram elements (classes, attributes, relations). Because it deals wells with semantic analysis. The task of rule-based NLP approach is accomplished through the usage of pattern matching approach. Actually, pattern matching rules are regarded as one of the most successful approaches for extracting information from NL texts. The pattern-matching technique's essence is to generate a list of structural patterns based on human knowledge competence. Therefore, the approach rely on experts knowledge to construct a collection of patterns that will help us locate the candidate concepts in the UML class diagram. Patterns are considered as meta-model transformations. Basically, patterns describe a set of target sentences using regular expressions combined with lexical, syntactical, and semantic elements. Then, sentences of the processed text are matched against the defined patterns to make decisions and corresponding interpretations based on their similarity to these patterns. Correspondingly, in this study, a set of patterns is defined. Patterns are then matched with each sentence in the input requirements text in order to find out the candidate elements of the UML class diagram. [15]

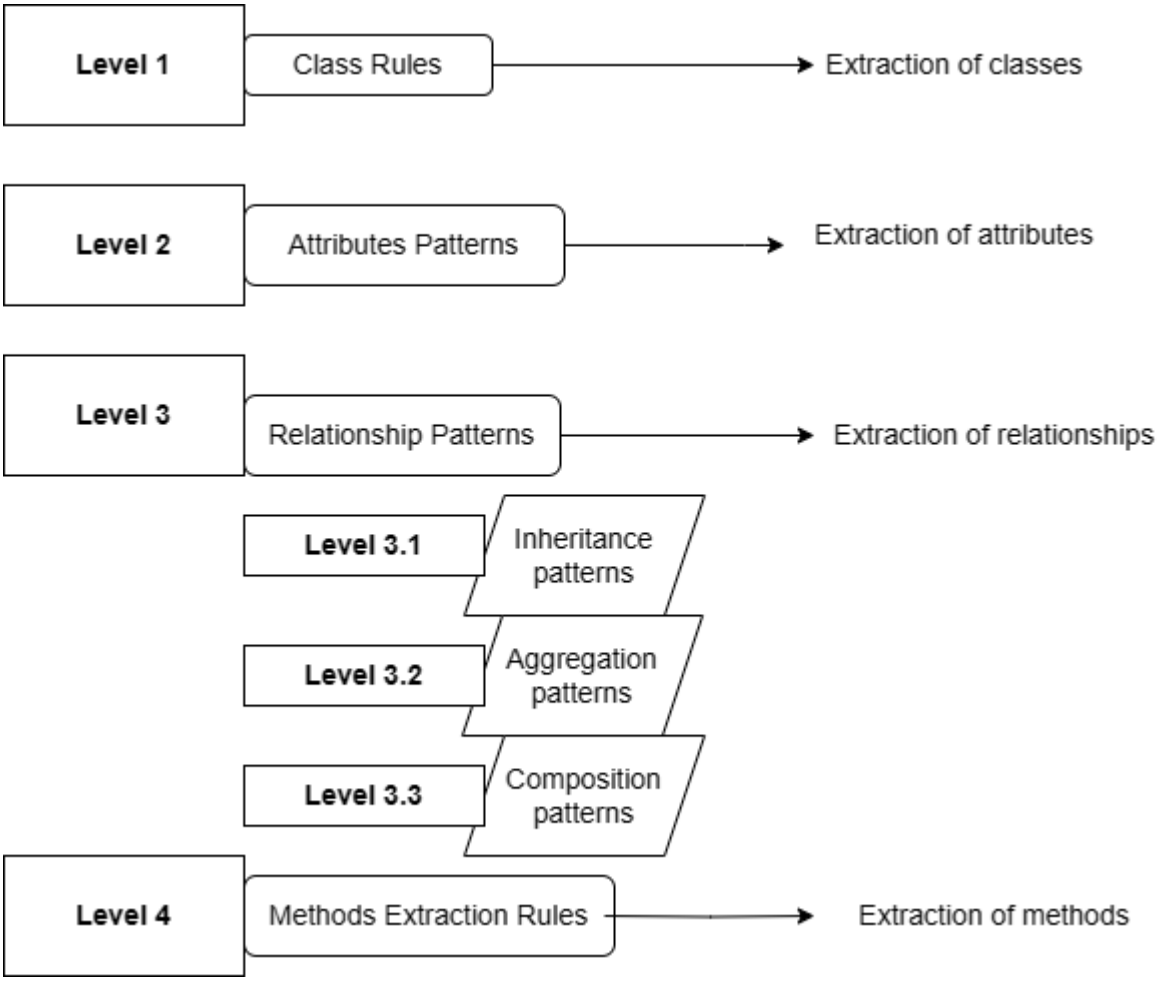

Figure 4.1 Priority Order Of Defined Patterns

4.1.4.1. Concepts Extraction

The aim of this module is to extract concepts according to the requirements document. This module uses SpaCy parser and stemming algorithm, and WordNet, to extract concepts related to the given requirements. We illustrate the algorithm of this module by the following step:

Step1:	**Use the requirements document as input.**
Step2:	Identify the stop words and save the result as {Stopwords_Found}
Step3:	Calculate number of words in the documents without the stop words. And Calculate the frequency of each word. Frequency (word) = number of occurrence (word) / Total number of words without stop words
Step4:	Use stemming algorithm module to find the stemming for each word and save the result in a list.
Step5:	Use SpaCy parser to parse the whole document (including the stop words)
Step6:	: Use the parser output to extract Proper Nouns (NN), Noun phases (NP), verbs (VB). And save it in {Concepts-list} list.
Step7:	Use Step2 and Step 6 to extract:{Noun phrases (NP)} - {Stopwords_Found}and save results to {Concepts-list}
	For each concept (CT) in {Concepts-list} if {synonyms_list} contains a concept (CT2) which have a synonym (SM) which lexically equal to CT, then CT and CT2 concepts are semantically related to each other.
Step8:	For each concept (CT) in {Concepts-list} if {synonyms_list} contains a concept (CT2) which have a synonym (SM) which lexically equal to CT, then CT and CT2 concepts are semantically related to each other.
Step9:	For each concept (CT) in {Concepts-list} if {hypernyms_list} contains a concept (CT2) which have a hypernyms (HM) which lexically equal to CT, then CT2 “is a kind of “ CT. Save result to {Generalization-list.

Table 4.3 Steps for Concept Extraction

4.1.4.2. Classes Extraction

This module uses different heuristic rules to extract the class diagram; However, We use domain ontology in this module to refine the extracted class diagram. We can summarize the heuristic rules used as the following:

Class Identification Rules [12]:

C-Rule 1 :	**If a concept is occurred only one time in the document and its frequency is less than 2 %, then ignore as class.**
C-Rule 2 :	If a concept is related to the design elements then ignore as class.
C-Rule 3 :	If a concept is related to Location name, People name, then ignore as a class.
C-Rule 4 :	If a concept is found in the high level of hyponyms tree, this indicates that the concept is general and can be replaced by a specific concept, then ignore as class.
C-Rule 5 :	If a concept is an attribute, then ignore as a class.
C-Rule 6 :	If a concept does not satisfy any of the previous rules, then it's most likely a class.
C-Rule 7 :	If a concept is noun phrase (Noun + Noun), if the second noun is an attribute then the first Noun is a class. The second noun is an attribute of that class.
C-Rule 8 :	If the ontology (if-used) contains information about the concept such as relationships, attributes, then that concept is a class.

Table 4.4 Class Identification Rules

We used Word Net feature to verify Generalization relationship where a noun phrase is supposed to be ‘a kind of’ another phrase. Word Net can used to find semantically similar terms, and for the acquisition of synonyms.

4.1.4.3.Attributes Extraction

In this task, extraction of attributes is done . A list of patterns are defined on some key words and expressions with the help of domain experts . Each Sentence in functional requirement file (user input) after preprocessing is compared to these patterns . The Phrase Matcher of SpaCy allows you to match lists of tokens on a text .
Table (4.5) illustrates pattern formats and their corresponding extraction of attribute.

[Note In the next table: X is used to indicate a class . while Y is used to indicate an attribute or more than one attribute speared with (,) or (and) or (or).]

Attp1	(X) (is |are) (identified| known|represented|denoted|described) (with|by) (Y)
Attp2	(details|data|information) about (X) are (Y)
Attp3	(X)(has|have|consistsof|consistof|show|shows|contain|contains|include|includes) (Y)
Attp4	(a|an|the) (Y) is unique within (a|an|the) (X)
Attp5	(Y) (is|are) required for (X)
Attp6	For (X) (need|like) to know (Y)

Table 4.5 Patterns for Attributes Extraction

Figure 4.5 illustrates an example.

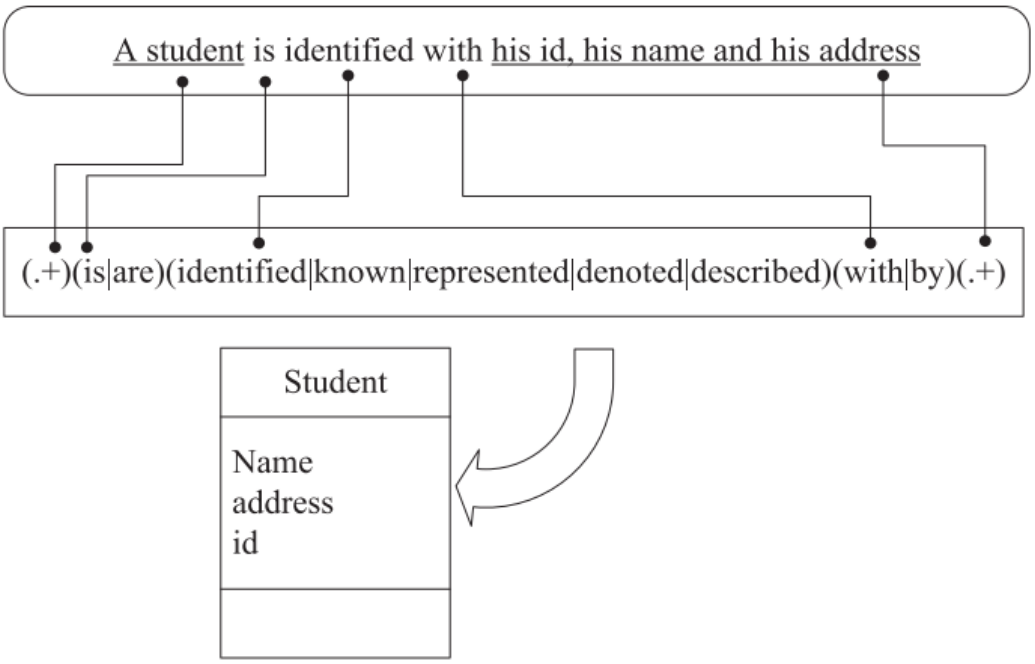

Figure 4.2 Illustration Of An Attribute Pattern Format.

4.1.4.4. Relationships extraction

A. Inheritance or generalization

[Note In next tables: X is used to indicate a parent class, while Y and Z are used to indicate subclasses.]

Inheritance pattern1:	(X) (is|are) either (Y) or (Z)
Inheritance pattern2:	Types of (X) are (Y) (and|or) (Z)
Inheritance pattern3:	(X) (may|can|could) (also)?be a (Y) (or) ? (Z)

Inheritance pattern4:	(Y) is a subclass of (X)
Inheritance pattern5:	(Y) (are|is) (kind|kinds) of? (X)

Table 4.6 Inheritance relationship patterns

B. Aggregation

Aggregation pattern1:	There are (Y) in (X)
Aggregation pattern2:	(X)(is|are) made up? (of|by) (Y)
Aggregation pattern3:	(X) (has|have) (Y)
Aggregation pattern4:	(X) (is|are) divided (into|to) (Y)
Aggregation pattern5:	X) (comprise|involve|carry|hold|embrace) (Y)

Table 4.7 Aggregation patterns

C. Composition

Composition pattern1:	(X) (is|are) composed (of|by) (Y)
Composition pattern2:	(Y) (is|are) part of (X)
Composition pattern3:	(Y) (belong to|belongs to) (X)
Composition pattern4:	(Y) (is|are) composite of (X)
Composition pattern5:	(X) (has|have) (.+) parts (Y)

Table 4.8 Composition patterns

4.3.4.3 Generation of class diagram model as text file

After extraction process of class diagram elements , a class diagram model is written in certain syntax in a text file with the help of implemented class model class that assists in typing the class model in a text file.

4.3.4.4 Rendering UML Diagrams from Plain Text Files

After Use case model is written in text file in certain syntax. it passed to PlantUML tool to render it as image file that is displayed to the BA (user) at the end of processing .Saving Render Resulted images to database .

4.2. Techniques

- **NLP Techniques :**

1. Tokenization:

Tokenization involves breaking a text into individual units, typically words or sentences. It is a fundamental step in NLP and serves as a basis for further analysis. Tokenization allows for the separation of text into meaningful components, enabling tasks as counting words frequency, analyzing grammar, or creating word embeddings or finding synonyms of specific word.
All functional requirements or user stories that are entered by user in form of text files is tokenized (splitted) into sentences then all sentences are tokenized into list of tokens where each sentence contains list of tokens not just words as strings .

2. Part-of-Speech Tagging:

POS tagging, also known as grammatical tagging , is the process of assigning grammatical tags or labels to words in a sentence, indicating their syntactic category or part of speech. POS tagging is a crucial step in natural language processing tasks and helps in understanding the structure and meaning of text.
POS tagging assigns tags to words based on their syntactic roles and categories. Some common POS tags include nouns (NN), verbs (VB), adjectives (JJ), adverbs (RB), pronouns (PRP), prepositions (IN), conjunctions (CC), and determiners (DT). These tags provide information about the word's function and its relationship with other words in a sentence.

POS is used through the project in extracting root verbs in extracting meaningful use case element. It is also used in pattern matching where

The Matcher allows you to define patterns based on linguistic annotations as part-of-speech tags, dependency labels, lemma, and

more. For example specifying a pattern that extract an aggregation relation is written as follow :

Pattern : xClass involve yClass .
[
{"POS": "NOUN", "DEP": "nsubj"}, {"POS": "VERB", "DEP": "ROOT", "LOWER": “verbsInvolve”},{"POS": "NOUN"}, {"OP": "*"}
]
[16]

4.3. Performance evaluation

This section presents the results of a practical performance evaluation of our system. In order to evaluate our system performance, we have compared the models produced by our system with the models produced by human analysts. We have used three metrics for it that is used in performance evaluation.
These are;

1. Recall:
It focuses Completeness parameter; how much completed model system has produced. It can be given as,

$$\text{Recall} = \text{Ncorrect} / \text{Nkey}$$

Where Ncorrect refers to number of correct responses made by the system. Nkey - number of information elements in answer key.

2. Precision:
It focuses on Accuracy parameter; how much of the information produced by the system is correct. It can be given as,

Precision= Ncorrect / (Ncorrect + Nincorrect)

Where Nincorrect refers to incorrect responses made by the system.

3. Overspecification:

It measures how much extra correct information in the system response is not found in the answer key.

Over specification= Nextra / Nkey

Where Nextra means number of responses judged correct but not found.

Generated Use case diagram Performance :

- Scenario 1: In case of small number of user stories less than 20, the Input User stories file (10 user stories) is shown in the following figure:

```
As a user,i can add my credit to pay so that order is completed.
As a user,i can view list of products for sale so that i can decide which products to buy.
As a user,i want to add items to cart so that i can check them out.
As a user,i can checkout products after paying by card.
As a user,i can delete products from cart so that i donot check them out.
As an admin ,i can add new products so that products are updated.
As an admin , i want to add new product to list of products .
As an admin , i can mark products as out of stock until they are refilled.
As an admin , i can change product details or price .
As an admin , i can delete product out of stock.
```

Figure 4.3 Small Scale of Input User Stories File

Generated Output Diagram:

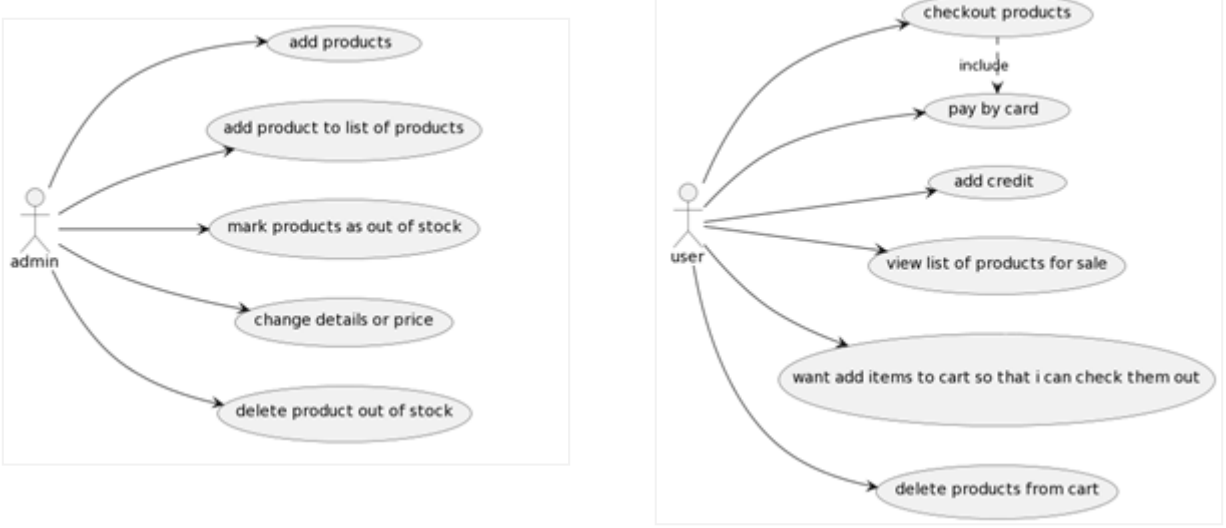

Figure 4.4Generated Use Case Diagram Result for small system

For admin Actor →

- Count of Use cases Found by actual BA, manually, is 5 .

Metric	Use Cases Found by our system
Recall	5/5 100%
Precision	5/5 100%
Overspecification	0

Table 4.9 Use Case Performance Measure for Admin Actor

For User Actor →

- Count of Use cases Found by actual BA, manually, is 6 .

Metric	Use cases Found by our system
Recall	6/6~100%
Precision	6/6 ~100%
Overspecification	0

Table 4.10 Use Case Performance Measure for User Actor

- Scenario 2: In case of larger input user stories file in size of 40 user stories. The Input User stories file is shown in the following figure :

```
As a customer, I want to see my borrows.
As a customer, I want to see my borrows.
As a customer, I want to see book categories.
As a customer, I want to see the availability of books.
As a customer, I want to reserve a book.
As a customer, I want to delete book recomendation.
As a customer, I want to create my borrows
As a librarian, I want to add new members.
As a librarian, I want to add new books so that members can see new books.
As a librarian, I want to add new books to book list.
As a librarian, I want to add books to the library.
As a librarian, I want to be able to add new members
As a librarian, I want to be able to calculate fees for delayed book returns
As a librarian, I want to be able to see the list of overdue books.
As a librarian, I want to be able to update members subscription plans.
As a librarian, I want to be able to update the status of a book as checked-out during check-out.
As a librarian, I want to block unwanted members.
As a librarian, I want to calculate the fee for delayed book returns.
As a librarian, I want to chat with users.
As a librarian, I want to check the maintenance dates of the books so that I can periodically.
As a librarian, I want to check the number of books available for borrowing.
As a librarian, I want to create book list.
As a librarian, I want to create information about borrowed books.
As a librarian, I want to create pages for book information.
As a librarian, I want to delete books.
As a librarian, I want to delete books from the library.
As a librarian, I want to delete information about borrowed books.
As a librarian, I want to delete members.
As a librarian, I want to detect unreturned books so that I can reject members.
As a librarian, I want to edit a member subscription.
As a librarian, I want to edit book list.
As a librarian, I want to edit current books shelf information.
As a librarian, I want to edit details about books and their locations.
As a librarian, I want to edit details about books.
As a librarian, I want to edit list of borrowed books.
As a librarian, I want to see late returns.
As a librarian, I want to update book list.
As a customer, I want to delete my borrows.
As a customer, I want to update my borrows.
As a customer, I want to make book recomendation.
```

Figure 4.5 Large Scale of Input User Stories File

Generated Output Diagram:

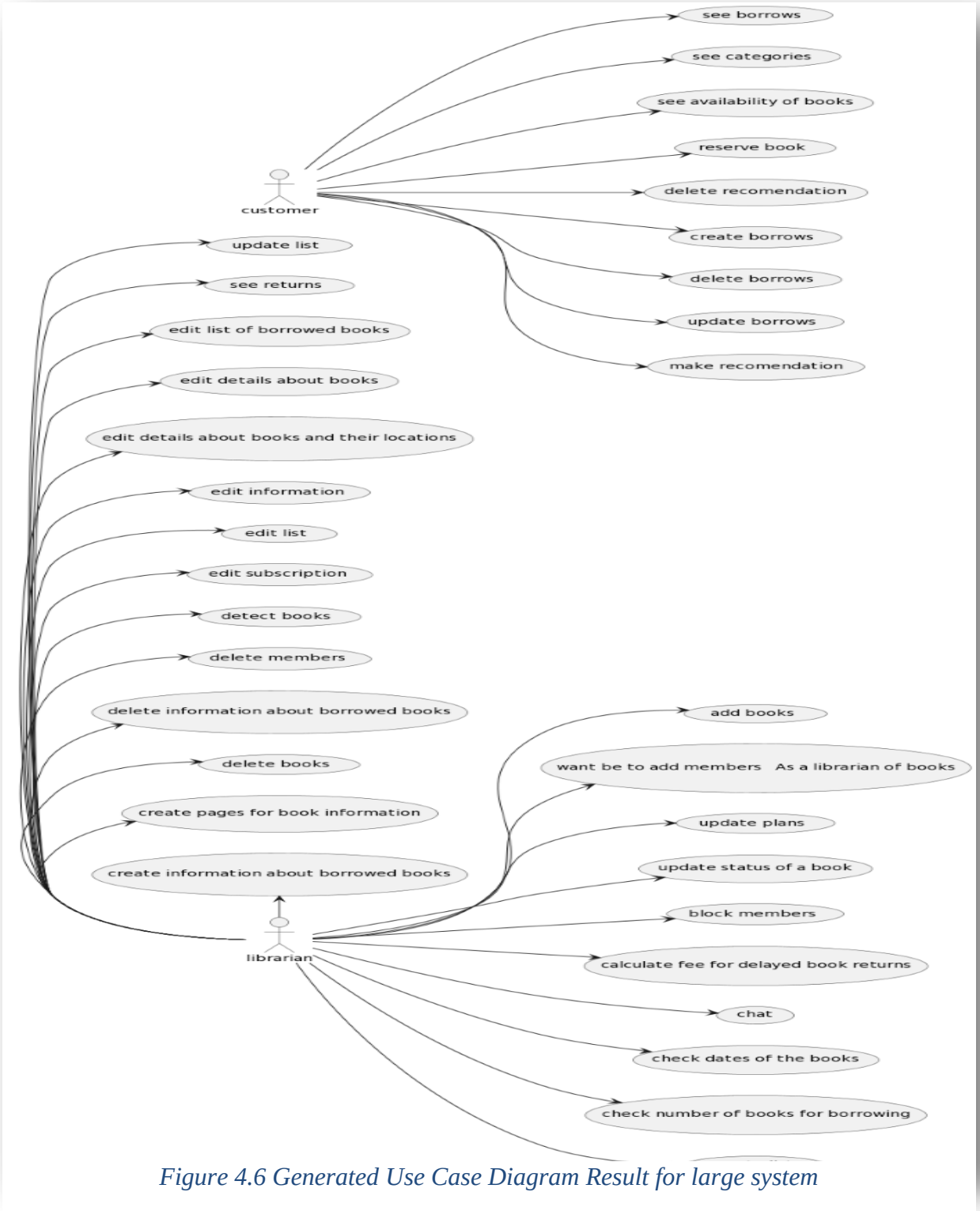

Figure 4.6 Generated Use Case Diagram Result for large system

For Customer Actor →

- Count of Use cases Found by actual BA, manually, is 9 .

Metric	Use cases Found by our system
Recall	9/10~100%
Precision	9/9~100%
Overspecification	0

Table 4.11 Use Case Performance Measure for Customer Actor

For Librarian Actor →

- Count of Use cases Found by actual BA, manually, is 27.

Metric	Use cases Found by our system
Recall	23/27~85%
Precision	23/25~92%
Overspecification	0

Table 4.12 Use Case Performance Measure for Librarian Actor

Class diagram :

This is a comparison between two class diagrams .

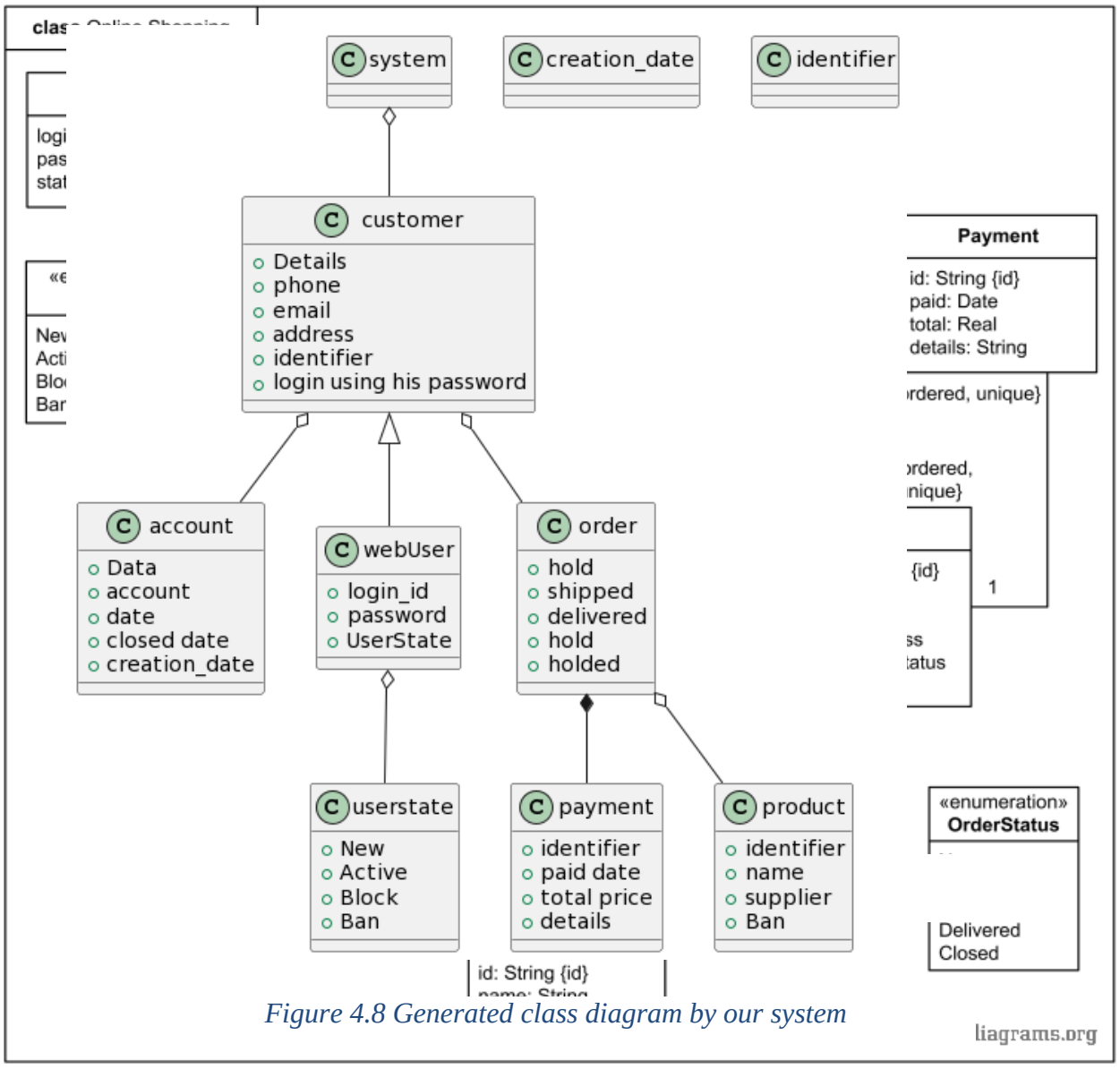

Figure 4.8 Generated class diagram by our system

Figure 4.7 Actual Ecommerce website class diagram

The actual class diagram contains 10 classes :

Metric	Use cases Found by our system
Recall	8/10~80%
Overspecification	3 out of 10 ~ 30%

4.4. Technologies used in implementation

Backend Technologies used :

- PlantUML :
  PlantUML is an open-source tool that allows you to create UML (Unified Modeling Language) diagrams using a simple and intuitive textual syntax. It enables expressing and visualizing software systems, designs, and processes using a human-readable format. Plant UML tool takes input in form of text in certain syntax form. The syntax of PlantUML is based on a set of keywords and symbols that represent UML elements. PlantUML then takes this textual representation and generates a visual diagram in various formats, such as PNG.[13]
- Python
- Fast API
- Firebase
- Cloud Firestore
- NLP libraries :
  - Spacy
  - Textacy
  - NLTK Word Corpus

Frontend Technology used :

- ReactJs
- HTML
- CSS
- JavaScript

Chapter 5

5. User Manual

5.1 Register User :

First step to benefit from System's functionalities is to register as a new user on the website.

Steps:

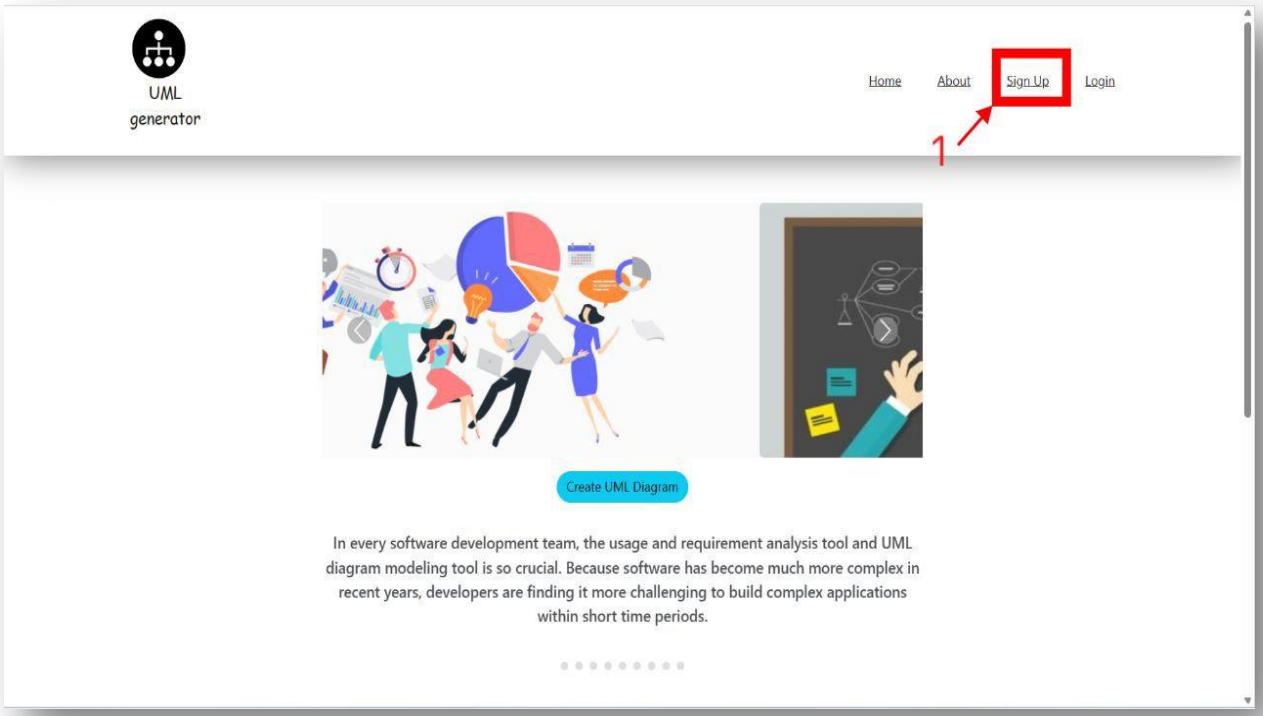

Figure 5.1 Register User

Step 1 : Click on Sign Up link.

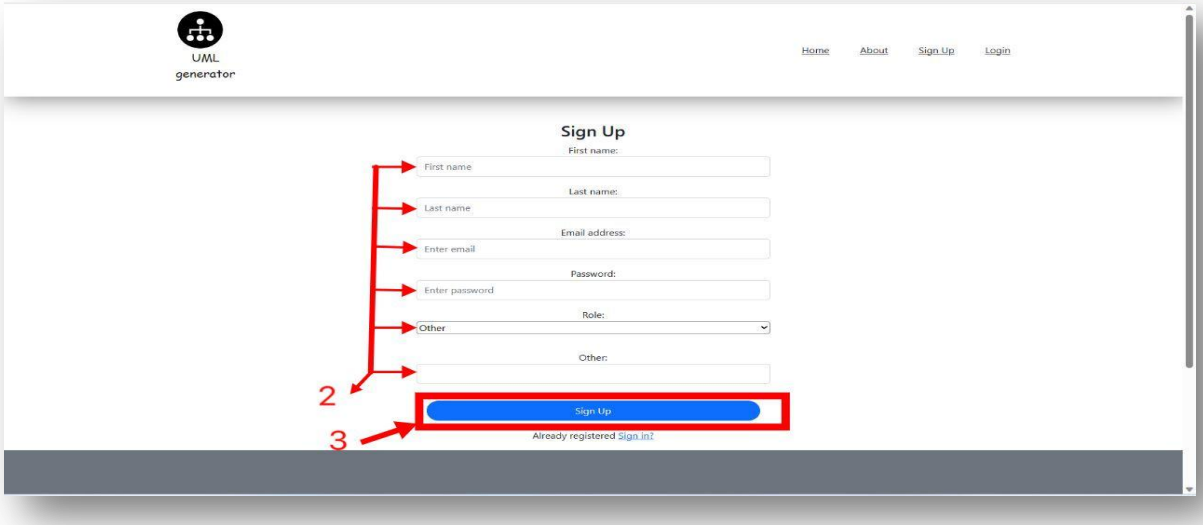

Figure 5.2 Sign Up Page

Step 2: Fill-out the form with required Data.
Step 3: Click on Sign up Button or Sign In link, if already registered.

5.2 Login:

Allows access to functionality of system which uploading files and generating diagrams.

Steps:

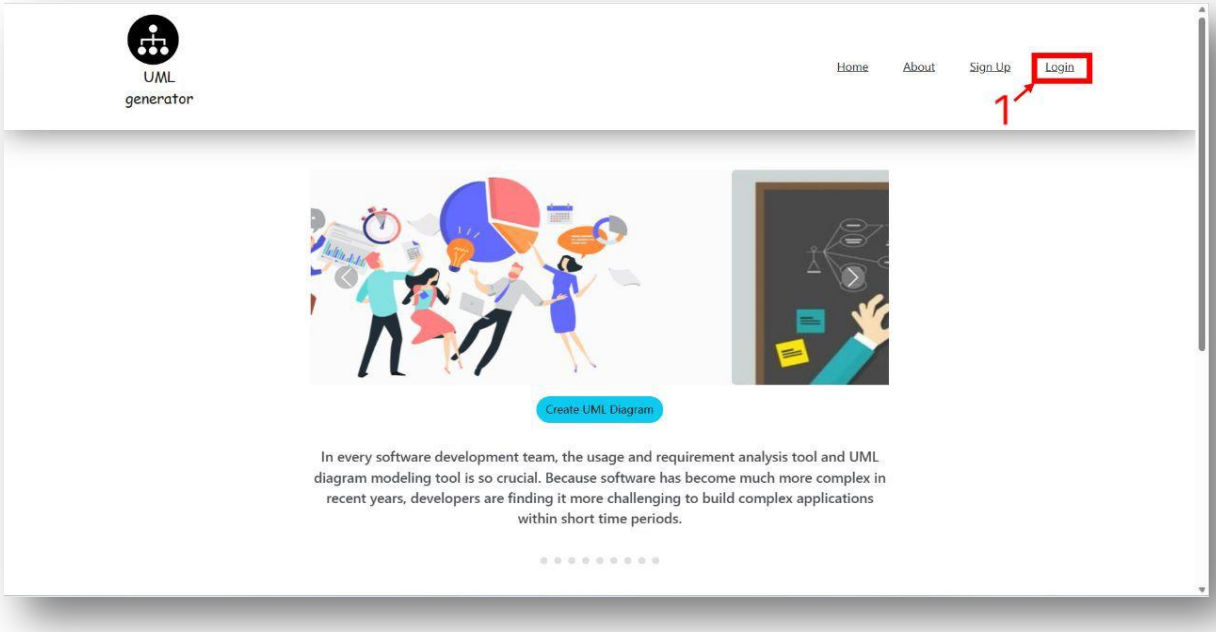

Figure 5.3 Login User.

Step 1 : Click on Login link.

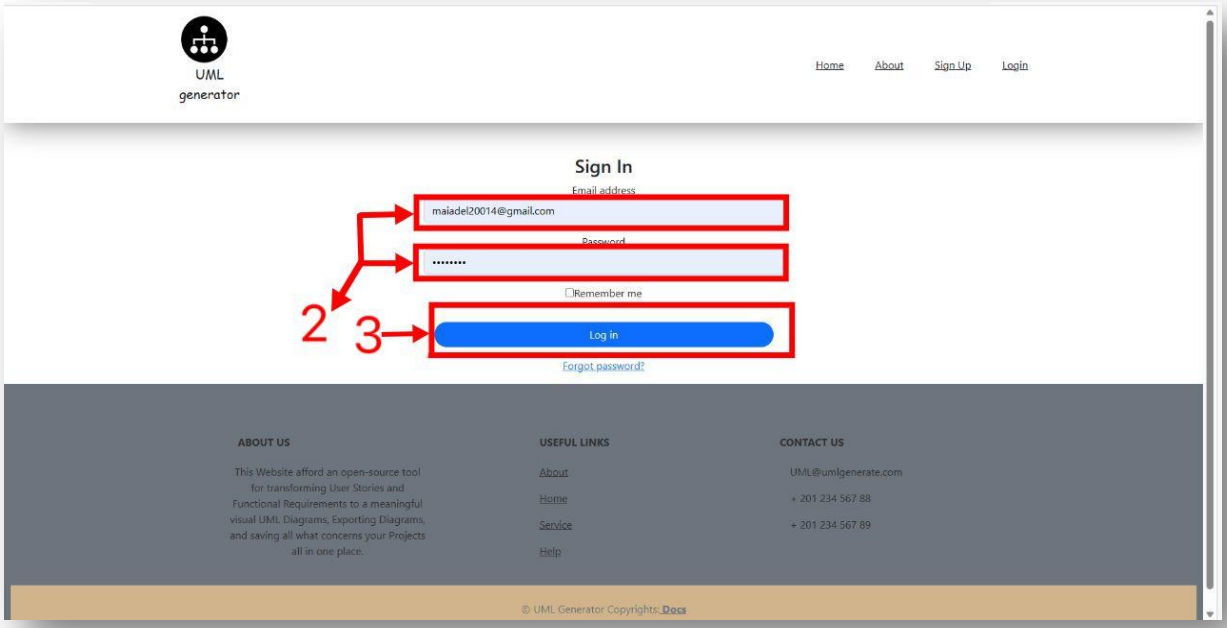

Figure 5.4 Login Page

Step 2: Fill-out the form with required Data.
Step 3: Click on Log in Button or Forgot password link to reset password.

5.2 : Add new project

Allows user to add projects to work on it. Projects separate files of different projects from each other. Each project contains files related to its own , separated from other user's projects.

Steps:

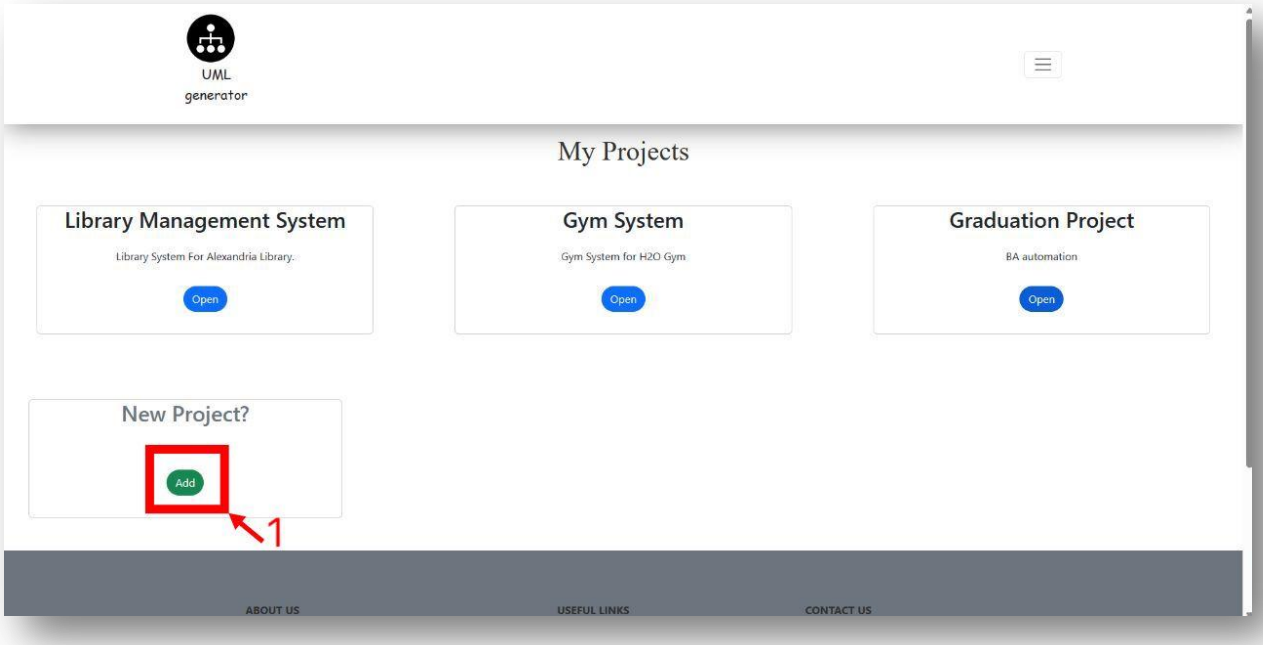

Figure 5.5 Add Project Button

Step 1: Click on Add Button.

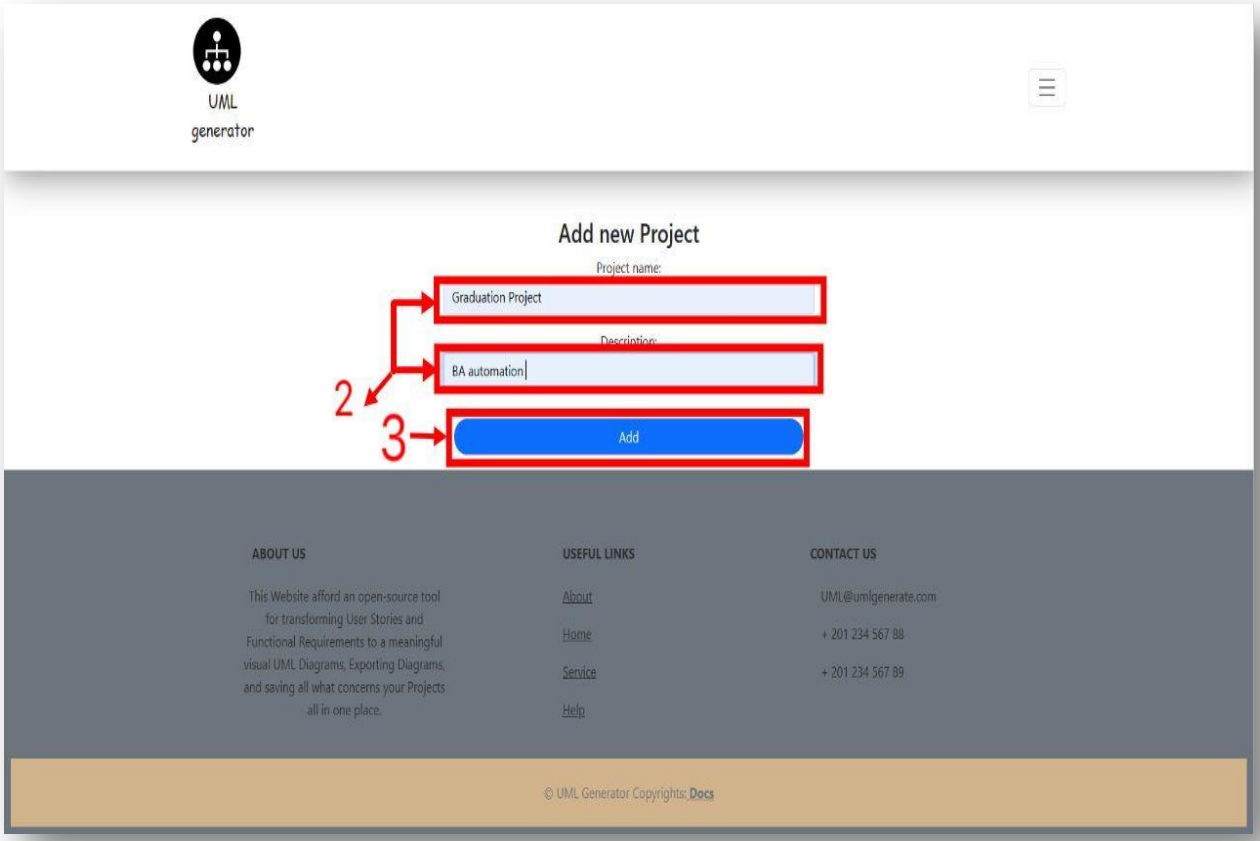

Figure 5.6 Add New Project Page

Step 2: Fill-out the form with required Data.
Step 3: Click on Add Button.

5.4 : Open Specific Project:

Open required project to upload related user stories and functional requirements files to this project, to generate use case and class diagrams, and to delete project.

Steps:

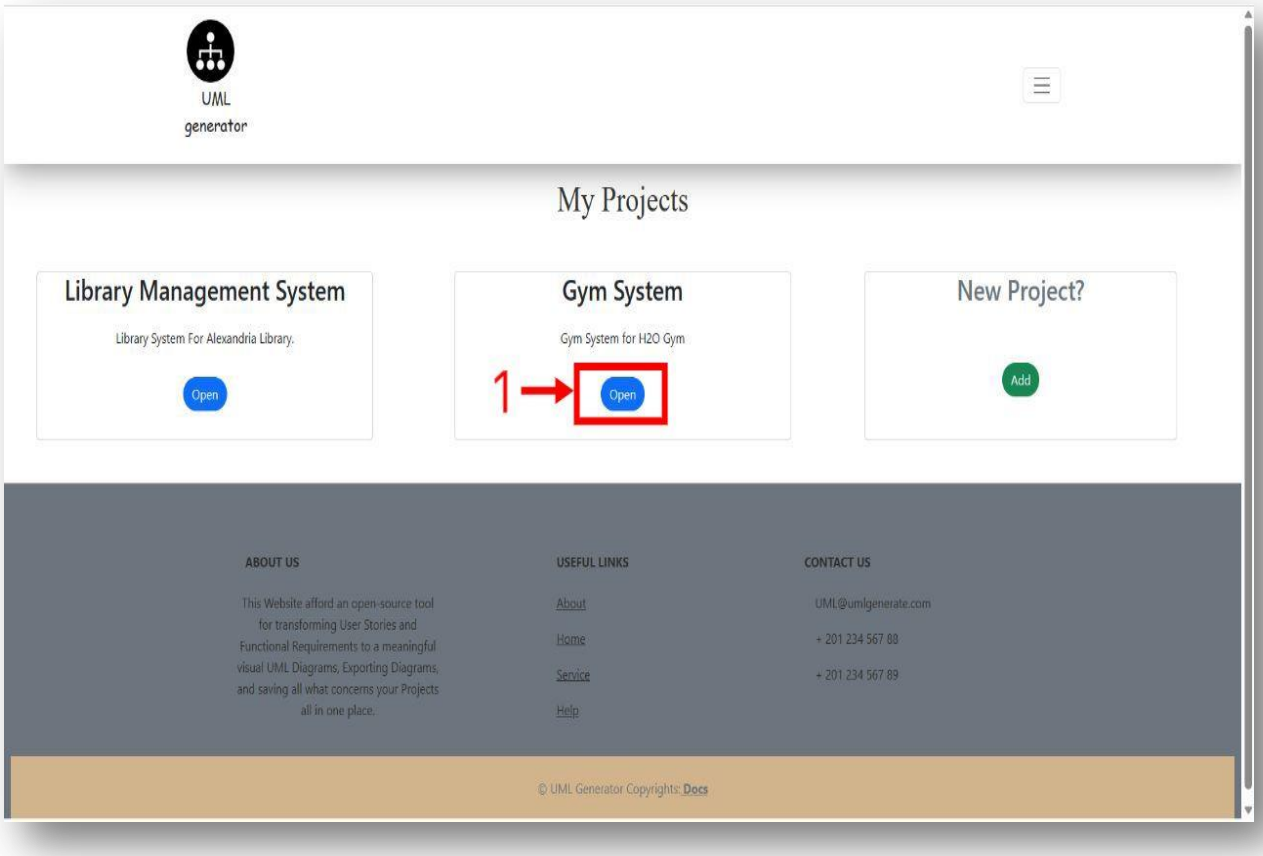

Figure 5.7 Open Project Button

Step 1: Click on Open Button.

5.4.1 Upload Files:

Steps:

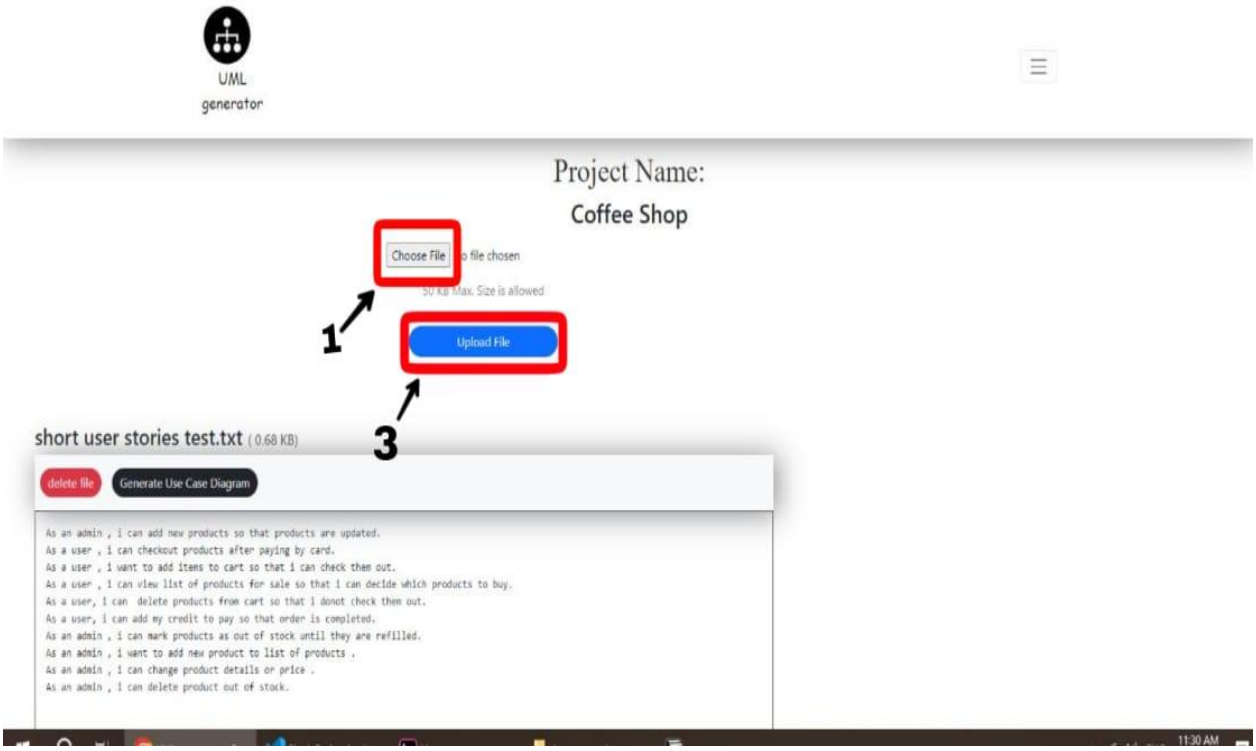

Figure 5.8 Choose and Upload File Button.

Step 1: Click on Choose Button.

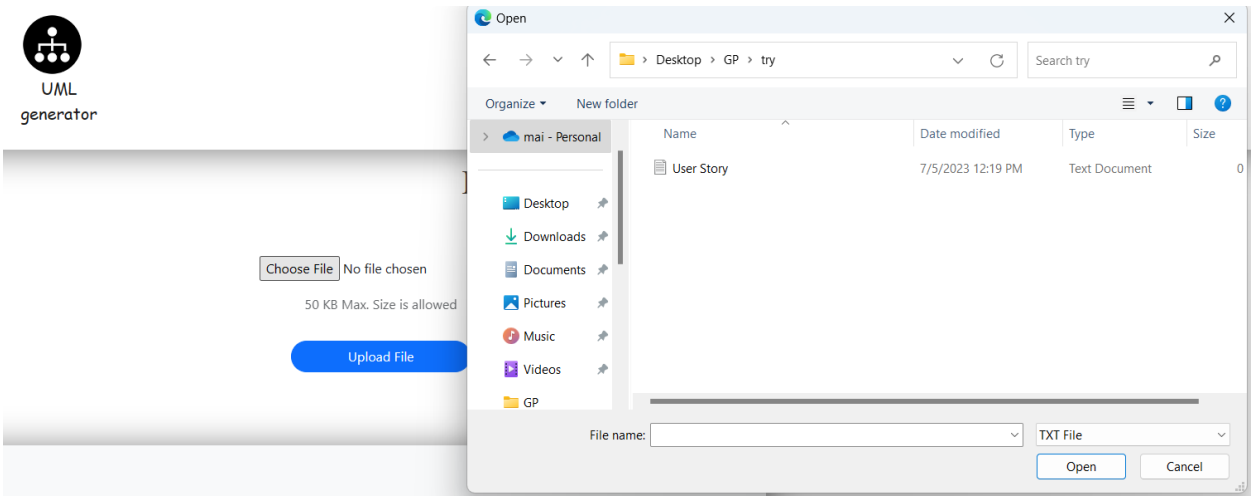

Figure 5.9 Figure 5.9 Choose File Window.

Step 2: Choose the file to be uploaded.
Step 3: Click on Upload Button.

5.4.2: Choose Diagram to generate :

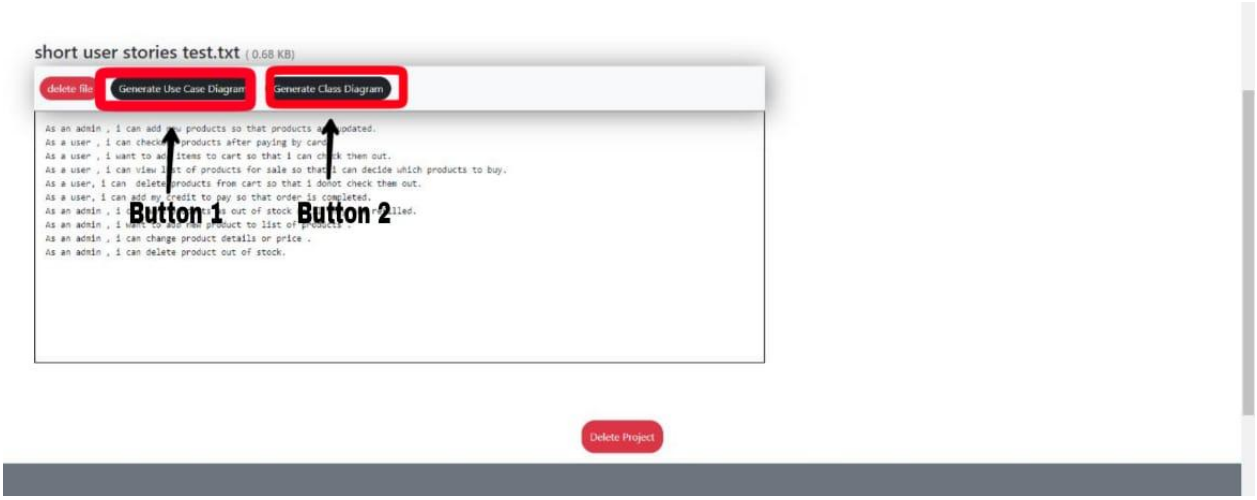

Figure 5.10 Figure 5.10 Generate Diagrams.

Step 1: Click on Generate Button 1 for generating Use Case Diagram and Generate Button 2 for generating Class Diagram.

5.4.3 Delete File:

Figure 5.11 Delete File

Step 1: Click on Delete File Button.

5.4.4: Delete Project :

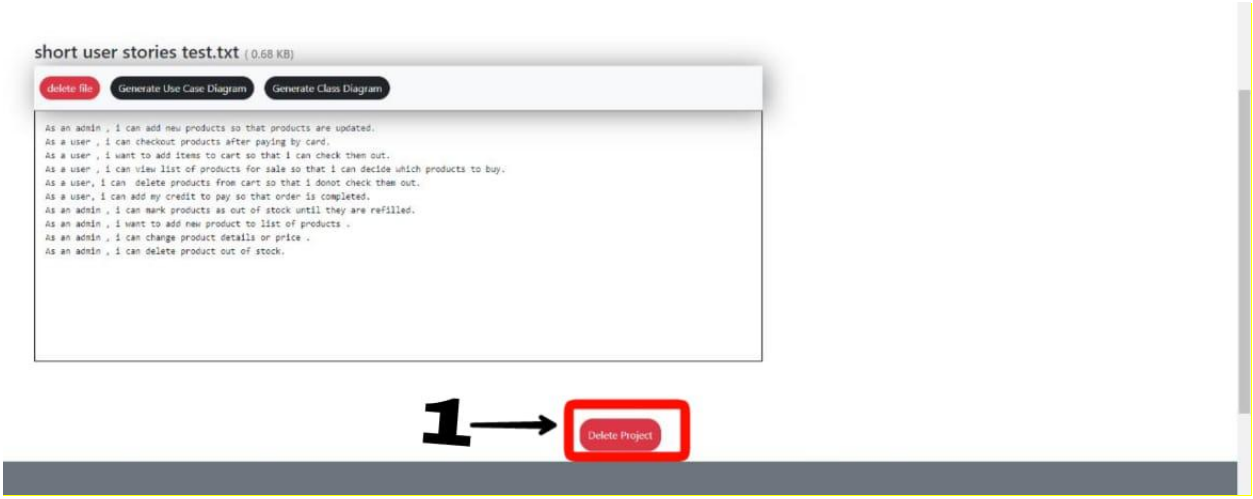

Figure 5.12 Delete Project Button.

Step 1: Click on Delete Project Button.

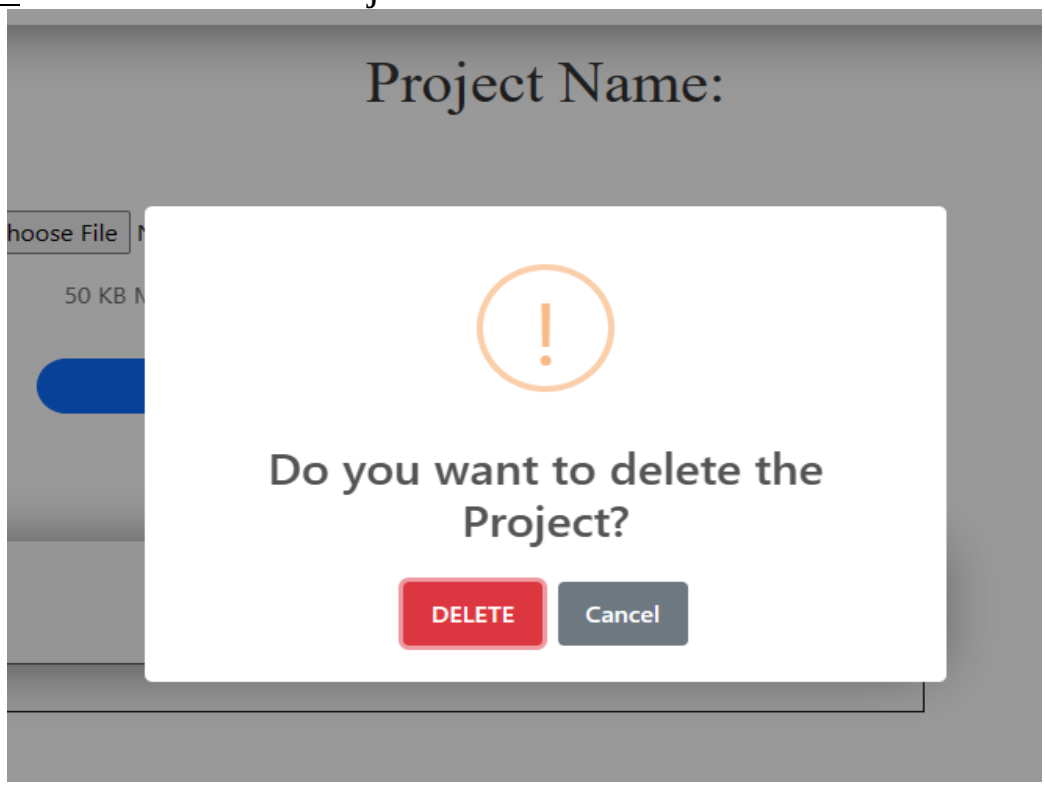

Figure 5.13 . Delete Project Validation Pop-up.

Step 2: Click on Delete Button or Cancel Button.

5 .5 *Logout :*

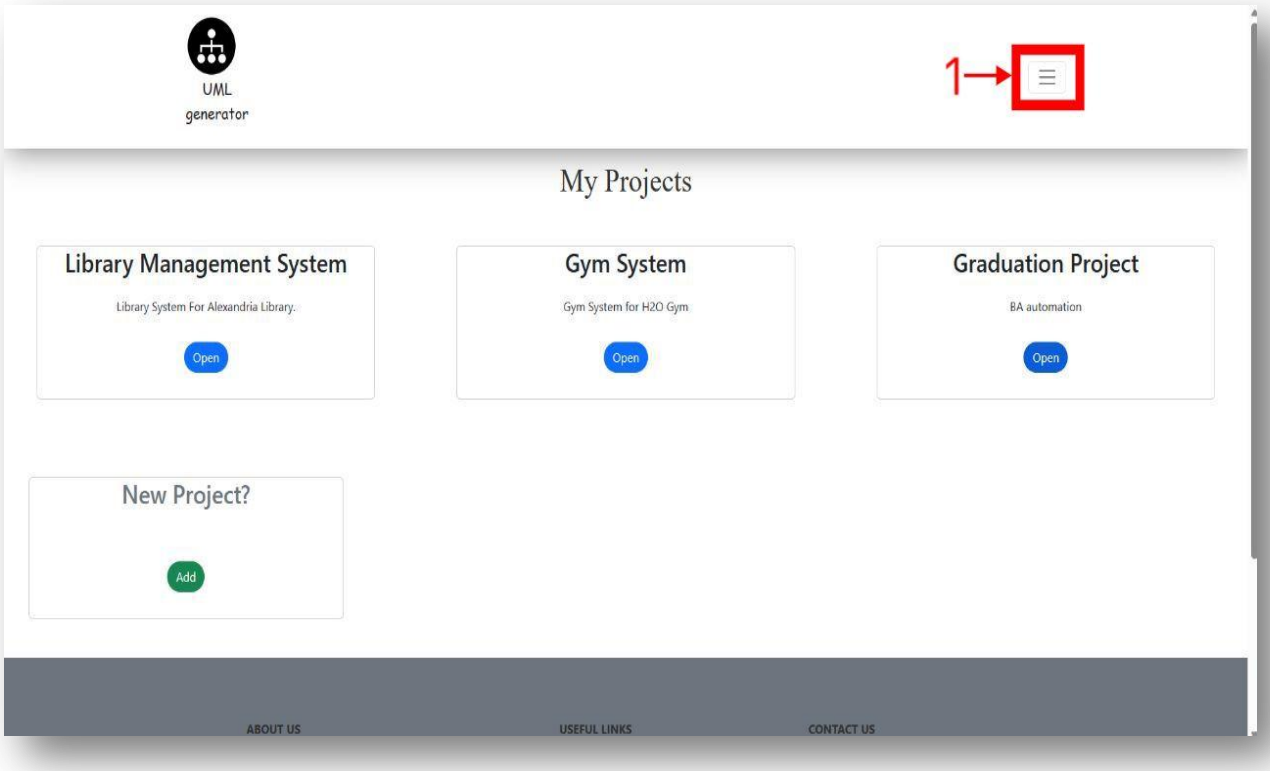

Figure 5.14 Menu icon Button

Step 1: Click on Menu icon Button.

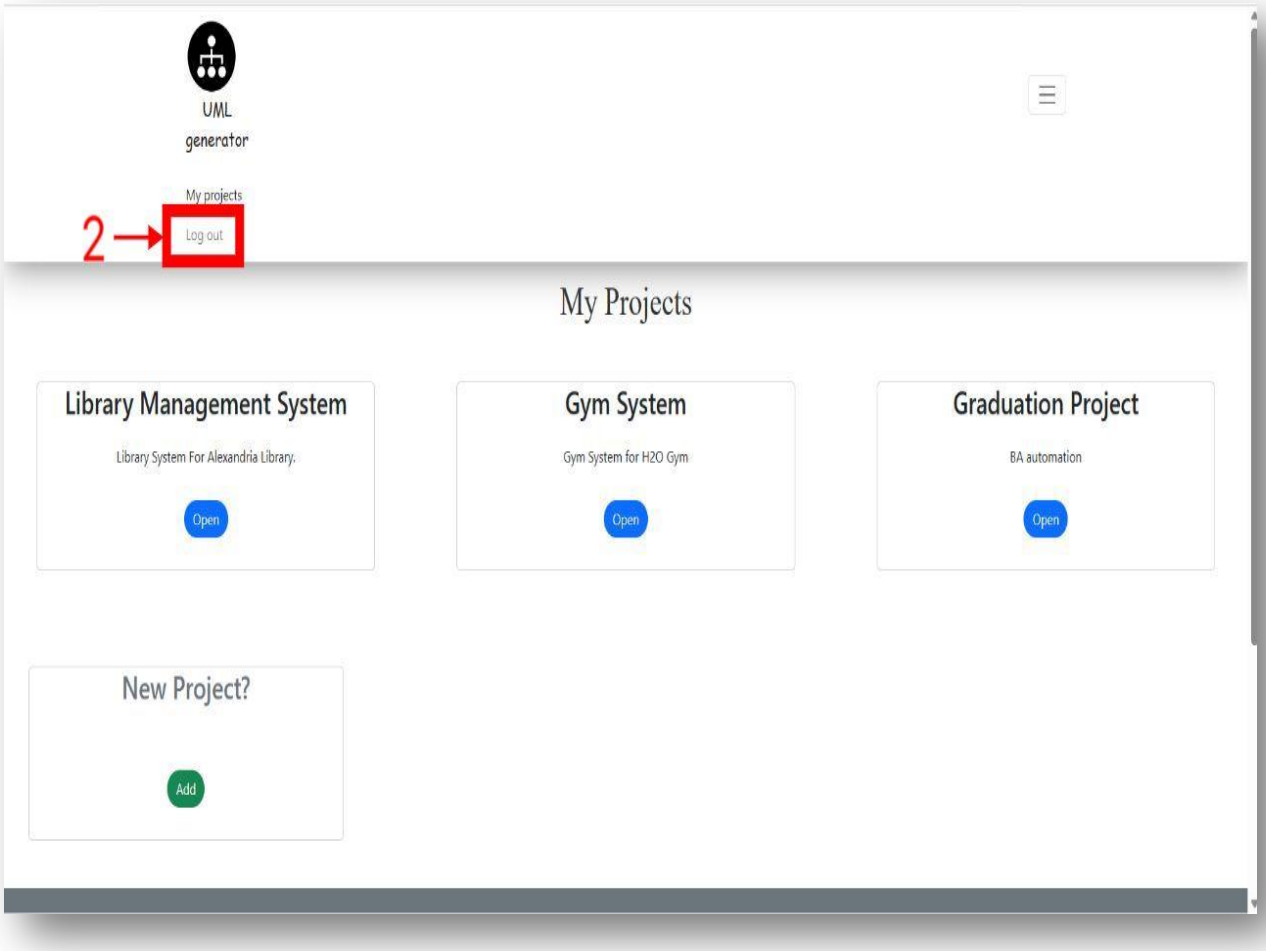

Figure 5.15Logout

Step 2: Click on Logout.

Chapter 6
6. Conclusions and Future Work

6.1 Conclusions

Design Phase consumes the most amount of time in SDLC, thus developing any assisting tool would highly be beneficial. Thus , this thesis Project develops a Software tool for automation UML Diagrams, use case and class Diagrams, for assisting BAs and designers in Design Phase. In Addition , "Automation of BA Tasks" provides a user-friendly , secure web application for BAs to save , manage , and upload files to the their projects easily. Also, unlike other commercial software tools , "Automation of BA tasks " is an open- source Website for all users.

Our Project makes use of NLP techniques and algorithms in the process of generating use case and class diagram by taking input files from users into the system , user stories for generating use case diagram and functional requirements for generating class diagram , processing them , and providing the output generated Diagrams as a PNG and Txt files for the user.

Throughout the Development of "Automation of BA tasks" , our team explored may research papers, meet experts , and reviewed carefully similar Software tools to ensure our project's value in field of study and to further enhance the consistency and quality of the tool. Our Algorithms and techniques choice to be implemented is merely a starting point to where this thesis could go. "Automation of BA Tasks " is a promising Research field that we believe could get even bigger resulting in better and more efficient results. The Design and User Interface of the tool were meant to be consistent with BAs to facilitate interaction and user's experience. The User Interface delivers a simple , easy to navigate and manipulate, and user-friendly Single-page web application.

6.2 Challenges

Serveal challenges were encountered during project design and implementation .

Firstly , extracting relevant information from natural language inputs posed a significant hurdle.NLP techniques had to be employed to understand the textual descriptions(user stories or functional requirements) provided by users and convert them into a structured representation models suitable for UML diagrams. Dealing with the nuances of language, variations in syntax, and the need to handle ambiguous or incomplete inputs required meticulous different preprocessing steps and advanced natural language understanding algorithms.

Secondly, ensuring accuracy and completeness in the automated UML diagrams proved to be a complex task. Translating natural language to precise UML elements and relationships requires robust algorithms capable of correctly interpreting the semantics of the textual inputs. Handling complex sentence structures, handling different levels of abstraction, and capturing the intended meaning behind user functional requirement were constant challenges in achieving accurate and

comprehensive UML diagrams. Patterns solved this problem .But it required huge amount of testing the results of these patterns.

Another challenge was handling the scalability and complexity of larger software systems. As the size and complexity of the system grew, the automated UML diagram generation system had to adapt and handle the increased volume of information. This included handling relationships between large number of different entities in sentences .

Lastly, ensuring user acceptance and usability of the automated UML diagrams was crucial. While the technical aspects were demanding, it was equally important to create a user-friendly interface that allowed users to interact easily and intuitively with the automated system. Balancing the need for automation with user control and customization options required careful design and user testing to accommodate a wide range of user preferences and scenarios. Ability to edit diagrams by editing diagram directly and get feedback in real time is included as future work.

Despite these formidable challenges, through continuous experimentation, research, rigorous testing, we were able to overcome these obstacles and develop a system capable of automating UML diagrams using NLP.

6.3 Future Work

The system can Successfully generate use case and class diagrams with identifying relationship and it's involved component. We hope to enhance our project and expand it by providing the following features and functionalities in Future works:

- Adding ML algorithms and models to extract more patterns.
- Adding multiplicity and cardinality ratio to relationship between classes in Class Diagram
- Easy and flexible User Edits to the Auto-generated Diagrams.
- Adding ML Unsupervised and semi-supervised models to Learn from manual User edits to the generated diagrams to enhance efficiency, accuracy, and performance.
- Enhancing class diagram elements extraction techniques , as patterns can be replaced with neural network and transformers .
- User can export diagrams in different formats and extensions.
- Auto-generating more UML diagrams , e.g. Sequence , state and ERD Diagrams.
- Adding Extensions with other Software tools and Plugins.

7. References

[1] HofferLecture, J. S. (2014). Modern System analysis and Design , Fourth Edition. Pearson.

F. GeorgeJeffrey A. HofferLecture.

[2] Jeffrey A. Hoffer, Joey F. George, and Joseph A. Valacich, 2000, "Essentials of System Analysis and Design", 6th edition.

[3] Gottesdiener, E., & Gorman, M. (2012). Discover to deliver: Agile product planning and analysis. EBG Consulting, Inc.

[4] Wiegers, K. E., & Beatty, J. (2013). Software requirements. Pearson Education.

[5] Transforming simplified requirement in to a UML use case diagram using .. (n.d.).https://www.researchgate.net/publication/315809182_Transforming_Simplified_Requirement_in_to_a_UML_Use_Case_Diagram_Using_an_Open_Source_Tool

[6] Tilakaratna, J.M.P.P., (2016) "Development and Evaluation of an Ontologically Complete and Clear Object-Oriented Conceptual Modelling Grammar", PhD Thesis, 251-256.

[7] Duc Janssens,2019, "Natural Language Processing in Requirements Elicitation and Analysis: A Systematic Literature Review "

[8] Prakash M Nadkarni, Lucila Ohno-Machado, Wendy W Chapman, 2011, "Natural Language Processing: an introduction".

[9] "UML Distilled: A Brief Guide to the Standard Object Modeling Language" by Martin Fowler .

[10] Flowchart Maker Org Chart Maker Drawing Program Floor Plan creator. SmartDraw.(n.d.). https://www.smartdraw.com/?id=104640&gclid=CjwKCAjwqZSlBhBwEiwAfoZ

UIHNGVAeVvRnZdGpxhr5l10SfalWTJ3YiLSWvHch8TFiDkrFL66vb9hoCVb0QAvD_BwE

[11] IBM rational rose enterprise 7.0.0.4 IFIX001. (2010, August 1). https://www.ibm.com/support/pages/ibm-rational-rose-enterprise-7004-ifix001

[12] Product Documentation | ServiceNow. (n.d.). https://docs.servicenow.com/en-US/bundle/utah-servicenow-platform/page/product/configuration-management/concept/c_IdentificationRules.html

[13] Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams. (n.d.). PlantUML.com. https://plantuml.com/

[14] [UML]Object Management Group. Unified modeling language specification (2.1.1). Tech-nical report, Frammingam, Mass, 2007. http://www.uml.org.

[15] Naik., A. a. ((2016).). *"Reflecting Natural Language Text in to UML Diagrams."*.

[16] Universal POS tags. (2014-2022). Retrieved from Universaldependencies: https://universaldependencies.org/u/pos/