



HELWAN UNIVERSITY
Faculty of Computing and Artificial Intelligence
Computer Science Department

StudGo

A graduation project dissertation by:

Youssef Ahmed Abdulraouf Ahmed [20211057]

Yassin Mohamed Yassin Nasr [20211050]

Youssef Salah Youssef Mohamed [20211077]

Abdulrahman Amr Mohamed Mohamed [20210520]

Youssef Ahmed Mahmoud Ali [20211061]

Youssef AbdulMaqsood Mohamed [20211080]

Submitted in partial fulfilment of the requirements for the degree of Bachelor of
Science in Computing & Artificial Intelligence at the **Computer Science** Department,
the Faculty of Computing & Artificial Intelligence, Helwan University

Supervised by:

Dr. Salwa Osama

June 2025



جامعة حلوان
كلية الحاسبات والذكاء الاصطناعي
قسم علوم الحاسب

StudGo

رسالة مشروع تخرج مقدمة من:

يوسف احمد عبدالرؤف احمد [20211057]

ياسين محمد يس نصر [20211050]

يوسف صلاح يوسف محمد [20211077]

عبدالرحمن عمرو محمد محمد [20210520]

يوسف احمد محمود علي [20211061]

يوسف عبدالمقصود محمد الحسيني [20211080]

رسالة مقدمة ضمن متطلبات الحصول على درجة البكالوريوس في الحاسبات والذكاء الاصطناعي، بقسم
علوم الحاسب كلية الحاسبات والذكاء الاصطناعي، جامعة حلوان

تحت إشراف

د/ سلوي اسامة

يونيو/حزيران 2025



Table of Contents

Table of Contents

<i>Abstract</i>	8
<i>Keywords</i>	9
<i>Acknowledgement</i>	10
<i>Chapter 1: An Introduction</i>	11
<i>1.1 Overview</i>	12
<i>1.2 Problem Statement</i>	13
<i>1.3 Scope and Objectives</i>	14
<i>1.3.1 Scope</i>	14
<i>1.3.2 objectives</i>	14
<i>1.4 Document Architecture</i>	15
<i>1.5 Work Methodology</i>	18
<i>Chapter 2: Related Work (Literature Review)</i>	20
<i>2.1 Retrieval-Augmented Generation</i>	21
<i>2.2 Vector Databases and Semantic Search</i>	23
<i>2.3 LLMs in Educational Systems</i>	24
<i>2.4 Conflict Resolution Using Embedding Similarity</i>	24
<i>2.5 CV Analysis and Personalization</i>	25
<i>2.6 Comparative Analysis of Related Systems</i>	25



2.6.1 Feature Comparison	26
Chapter 3: The Proposed Solution	28
3.1 Solution Methodology	29
3.1.1 Architectural Overview	30
3.1.1 Core Innovations	31
3.2 Functional and Non-Functional Requirements	34
3.2.1 Functional Requirements	34
3.2.2 Non-functional Requirements	36
3.3 System Analysis & Design	37
3.3.1 Use Case Diagram	37
3.3.2 Sequence Diagram	47
3.3.3 Schema Diagram	53
3.3.4 Activity Diagram	54
Chapter 4: Experiment and Discussion	57
4.2 RAG-Based Intelligent Chatbot	59
4.3 Semantic Event Conflict Recommender	61
4.4 CV Analysis and Personalization	62
4.5 Discussion of Observations	64
Chapter 5: Methodology And Implementation Details	66
5.1 Tools	68



5.2 Phases	69
6.2.1 Frontend Development (React.js)	69
5.2.2 ASP.NET Core Backend (Main API)	70
5.2.3 FastAPI Chatbot (AI Assistant + RAG)	73
5.2.4 Internship Web Scraper (Python)	76
5.2.5 Integration Flow	76
5.3 Technology Stack Justification	77
5.4 UI Snapshots	80
5.4.1 Student Dashboard	80
5.4.2 Student Activity Dashboard	82
Chapter 6: Conclusions and Future Work	84
6.1 Conclusions	85
6.2 Future Work	86
References	88
Source Code	91



Table of Figures

Figure 1 RAG Comparison	21
Figure 2 Overview	29
Figure 3 Basic Overview	33
Figure 4 SA Usecase Diagram.....	38
Figure 5 Student Usecase Diagram.....	39
Figure 6 Activity Time Conflict Sequence Diagram	47
Figure 7 RAG System Sequence Diagram.....	47
Figure 8 SA Sequence Diagram.....	48
Figure 9 Auth Sequence Diagram.....	49
Figure 10 Student Sequence Diagram.....	50
Figure 11 Student Auth Sequence Diagram.....	51
Figure 12 Chatbot Sequence Diagram	51
Figure 13 Google Auth Sequence Diagram	52
Figure 14 Schema Diagram	53
Figure 15 SA Activity Diagram.....	54
Figure 16 Student Activity Diagram.....	55
Figure 17 Chatbot Activity Diagram	56
Figure 18 RAG Overview	59
Figure 19 RAG VS FINETUNING	64
Figure 20 Main Workflow	67
Figure 21 RAG ALG	74
Figure 22 RAG System Flow.....	74
Figure 23 Student Dashboard Home UI.....	80



Figure 24 Student Dashboard Jobs UI	80
Figure 25 Calender UI	81
Figure 26 SA Home UI.....	82
Figure 27 Manage Activity UI.....	82
Figure 28 SA Activity UI.....	83



Abstract

The idea for our Student Activity Companion App came from the need to support students in making the most of their university experience.

We noticed that many students struggle to find and engage with the right opportunities. Our app helps bridge that gap by allowing students to explore various student activity committees to join, stay updated on released internships, monitor their participation in different activities, and more. It also acts as a helpful guide for new students to discover different fields, build their network, and take their first steps toward career development.



Keywords

Student Activities, Career Development, Events, Workshops, Internships,
Student Organizations, Website, RAG System



Acknowledgement

We gratefully acknowledge the exceptional guidance and support of our supervisor, **Dr. Salwa Osama**, whose expertise and commitment were crucial throughout every stage of our project. Her thoughtful advice, patience, and encouragement greatly contributed to the development and success of our work.

Our deepest appreciation goes to our families and friends for their constant encouragement and belief in us, especially during the more difficult phases of this journey. Their unwavering support has been a vital source of motivation.

We would also like to thank all the participants who generously shared their time and perspectives, making this project possible through their valuable contributions.

Lastly, we extend our sincere thanks to the **Faculty of Computer and Artificial Intelligence at Helwan University** for providing the essential resources and a nurturing academic environment. Their dedication to fostering knowledge and innovation has played a significant role in the completion of our graduation project.

To everyone who supported us along the way, we express our heartfelt thanks.



Chapter 1: An Introduction



1.1 Overview

Student Activity Companion App is an all-in-one web platform designed specifically for university students and student-led organizations. It acts as a centralized hub for discovering, registering for, and managing a wide range of opportunities — such as events, workshops, internships, and more — that foster personal growth, professional development, and campus engagement.

Whether you're a student looking to enhance your skills and expand your career prospects, or a student organization aiming to promote your initiatives and reach a broader audience, the app streamlines the entire process. With features like personalized recommendations, event tracking, and conflict detection, the platform ensures students never miss out on valuable experiences that align with their goals.

By connecting opportunities with the students who need them most, the Student Activity Companion App empowers a more active, informed, and future-ready university community.



1.2 Problem Statement

University students—especially newcomers—often miss out on valuable extracurricular opportunities such as events, workshops, and internships. This is largely due to the scattered and uncoordinated nature of how these activities are promoted. Information is typically dispersed across various channels like social media, messaging apps, bulletin boards, and external websites, making it difficult for students to stay informed and organized.

This lack of a centralized, accessible platform leads to missed deadlines, low engagement, and underutilization of opportunities that are crucial for skill development and career readiness. Additionally, student organizations struggle to effectively reach their target audience and promote their events. A unified solution is needed to centralize access, streamline communication, and ensure students can easily discover and participate in activities that enrich their university experience.



1.3 Scope and Objectives

1.3.1 Scope

- Aggregating student activities (events, workshops, internships) in a single app.
- Supporting both student organizations (to post/manage activities) and students (to discover/register/track participation).

1.3.2 objectives

- Enable students to easily find and register for activities that match their interests and career goals.
- Provide student organizations with tools to promote and manage their activities.
- Integrate internship opportunities for career advancement.



1.4 Document Architecture

This document is systematically organized into six main chapters in addition to the preliminary pages and supplementary materials. Each chapter plays a crucial role in guiding the reader through the different phases of the project — from problem identification and related work to solution design, implementation, and evaluation.

Below is a detailed breakdown of the document structure:

Abstract, Keywords, and Acknowledgement

These preliminary sections provide a concise summary of the project, key terms used, and acknowledgements for contributors and supporters of the work.

Chapter 1: Introduction

Introduces the general context and motivation behind the project. It outlines the core problem the system addresses, defines the scope and objectives, and sets the stage for the rest of the report. This chapter also includes this overview of the document structure.



Chapter 2: Related Work (Literature Review)

Surveys relevant existing work and background research related to the system. This includes state-of-the-art techniques in Retrieval-Augmented Generation (RAG), vector databases, semantic search, LLM applications in education, embedding-based conflict resolution, and CV analysis. A comparative feature table highlights how our system differs from and improves upon existing solutions.

Chapter 3: The Proposed Solution

Describes the proposed approach in detail, starting with the overall methodology and architectural design of the system. It highlights key innovations and the rationale behind the chosen technologies. It also covers system requirements — both functional and non-functional — and presents detailed analysis and modeling using use case, sequence, activity, and schema diagrams.

Chapter 4: Experiment and Discussion

Demonstrates the system's core features through a set of focused experiments and components: the intelligent chatbot powered by RAG, the event conflict recommender using semantic similarity, and the CV analyzer for opportunity matching. The chapter also discusses observations, challenges, and insights gathered throughout testing and evaluation.



Chapter 5: Methodology and Implementation

This chapter describes both the development methodology adopted during the project and the detailed implementation of the system. The development process followed an iterative model, where each core feature — including the chatbot, recommender, and dashboards — was built and tested incrementally. The rationale behind choosing React.js, ASP.NET Core, and Python FastAPI is also discussed, followed by a breakdown of implementation phases and integration flow.

Chapter 6: Conclusions and Future Work

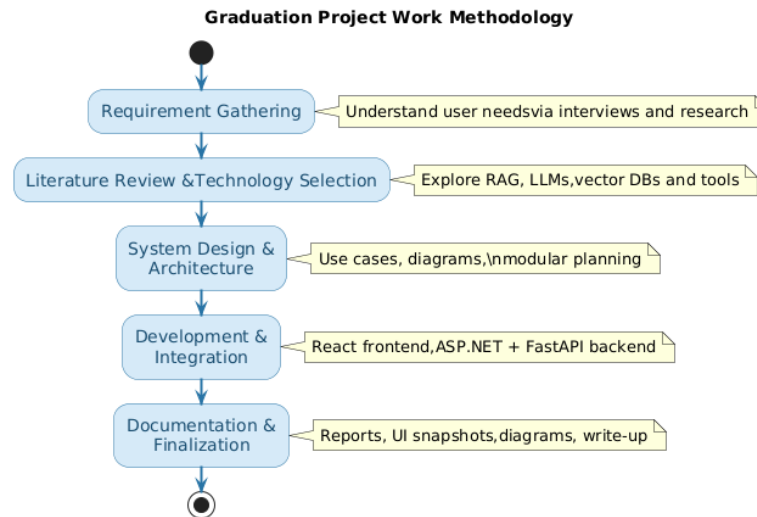
Summarizes the key achievements of the project, reflects on its impact, and discusses limitations. Suggestions for future work are provided, highlighting opportunities for system enhancement, scalability, and integration with wider educational platforms.

References and Source Code

A complete list of academic references, technical documentation, and online resources consulted during the project. This is followed by a link or appendix section containing the project's source code and deployment materials.



1.5 Work Methodology



The development of this graduation project followed a structured and iterative methodology that balanced both research and practical implementation. The work was divided into distinct phases, each building upon the previous to ensure logical and efficient progression. The methodology is summarized as follows:

1. Requirement Gathering and Problem Understanding

We began by identifying the main challenges faced by students and student activity organizations, using informal interviews, survey insights, and problem framing sessions.



2. Literature Review and Technology Selection

Extensive research was conducted to explore solutions involving Retrieval-Augmented Generation (RAG), semantic search, vector databases, and LLMs in education. This informed our choice of tools and techniques.

3. System Design and Architecture

The system was designed with modularity in mind, incorporating both traditional backend logic and AI components. Architecture diagrams, use cases, and schema models were developed to guide implementation.

4. Development and Integration

System components were developed in parallel: React.js for the frontend, ASP.NET Core for the main API, and Python (FastAPI) for the intelligent services. The modules were integrated using REST APIs to create a unified system.

5. Documentation and Finalization

The last phase involved writing technical documentation, preparing diagrams, generating UI snapshots, and finalizing the graduation report for submission.



Chapter 2: Related Work (Literature Review)



The proliferation of digital platforms in education has created a data-rich environment, yet students often face information overload and fragmented user experiences. Addressing this challenge requires intelligent systems that can personalize, recommend, and communicate with contextual awareness. The integration of artificial intelligence (AI), particularly large language models (LLMs), has become a cornerstone for developing such sophisticated educational tools. This chapter provides a comprehensive review of the core research areas and technologies that form the foundation of the Studgo system. We examine the literature on Retrieval-Augmented Generation (RAG), the role of vector databases in enabling semantic search, the application of LLMs in educational contexts, novel techniques for conflict resolution using embedding similarity, and NLP-driven personalization through CV analysis. By synthesizing these domains, we position Studgo's unique contribution in creating a unified, context-aware platform for student development.

2.1 Retrieval-Augmented Generation

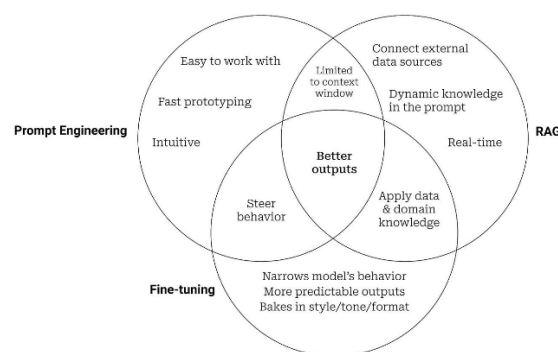


Figure 1 RAG Comparison

Retrieval-Augmented Generation (RAG) is a paradigm-shifting

architecture for language models, first proposed by Lewis et al. (2020). It



enhances purely generative models by dynamically incorporating information retrieved from an external knowledge base. The core process involves two stages: first, a retriever fetches relevant document chunks in response to a query; second, a generator synthesizes this retrieved information along with the original query to produce an answer.

The primary advantage of RAG is its ability to mitigate model "hallucination" and provide responses that are grounded in verifiable facts. This is particularly critical in educational applications where accuracy is paramount. While standard LLMs rely solely on their pre-trained parametric memory, RAG's non-parametric memory (the knowledge base) can be updated in real-time with new information, ensuring responses remain current and relevant. In the context of the Studgo system, RAG is implemented using the Mistral 7B model for its generative capabilities and ChromaDB as the retrieval backend. This architecture allows Studgo to access a dynamic knowledge base of



student activities, university events, and prior conversation history, ensuring all interactions are contextually rich and factually accurate.

2.2 Vector Databases and Semantic Search

The efficacy of a RAG system hinges on the quality of its retrieval mechanism, which is increasingly powered by vector databases and semantic search. Unlike traditional keyword-based search, which matches exact terms, semantic search operates on the conceptual meaning of a query. This is achieved by converting text into high-dimensional numerical vectors (embeddings), where semantically similar concepts are located close to each other in the vector space.

Vector databases, such as ChromaDB, are purpose-built to index and query these embeddings at massive scale. They employ similarity metrics like cosine similarity or Euclidean distance to find the vectors most relevant to a given query vector. A comprehensive survey by Han et al. (2023) highlights the superiority of vector databases for real-time,



large-scale semantic queries, which are essential for modern AI applications. For Studgo, this technology is crucial. It enables the system to understand nuanced user requests, matching a student's interest in "competitive programming" with an event described as a "hackathon," even when the keywords do not overlap directly.

2.3 LLMs in Educational Systems

LLMs, such as Mistral 7B, are widely adopted in educational NLP applications due to their language fluency and ability to process contextual inputs. Research by Liu et al. (2021) has shown that LLMs improve user satisfaction and engagement when used in educational chatbots. Studgo leverages this by using Mistral 7B to provide personalized and semantically rich responses.

2.4 Conflict Resolution Using Embedding Similarity

One of the innovative features in Studgo is the use of sentence embeddings to resolve conflicts between student events. By comparing the similarity between event descriptions and timings, the system can recommend non-conflicting alternatives. This approach is influenced by prior work on recommendation systems that use semantic similarity (Zhang & Chen, 2019).



2.5 CV Analysis and Personalization

Studgo uses spaCy to analyze student CVs and extract structured information such as skills, education, and work experience. This allows for improved recommendations and personalized assistance. Similar methodologies are used in Intelligent Tutoring Systems (ITS) and adaptive learning environments.

2.6 Comparative Analysis of Related Systems

Several platforms attempt to address similar user needs within different scopes:

Indeed and Wuzzuf are primarily job search engines offering access to job and internship listings. They focus on matching user-entered keywords with available positions but lack real-time personalization and context-aware conversation features. Luma Events and Meetup help users find and attend events but do not offer features like CV-based recommendations, inte, or semantic search for conflict resolution.

In contrast, Studgo offers a unified platform that combines AI-driven chatbot interaction, conflict-aware scheduling using semantic embeddings, and deep personalization through NLP-based CV analysis. It uniquely serves student activity discovery, opportunity alignment, and contextual career guidance.



2.6.1 Feature Comparison

While several platforms address components of a student's journey, none offer the integrated, AI-driven experience that Studgo provides. We analyze leading systems in adjacent domains to highlight Studgo's unique value proposition.

Job Portals (e.g., Indeed, Wuzzuf): These platforms are powerful search engines for job and internship listings. Their primary function is to match user-entered keywords and filters against a large database of opportunities. However, their architecture has significant limitations in the context of personalized student development. They typically lack a conversational interface, relying on static search queries. Furthermore, they do not perform deep CV analysis for proactive recommendations, nor do they integrate with a student's personal calendar or activity schedule to offer context-aware guidance.

Event Management Platforms (e.g., Luma Events, Meetup): These systems excel at helping users discover and manage attendance at events. They are generally organized around communities or interests. Their limitation, however, is a lack of focus on the specific academic and professional development context of a student. They do not offer features like CV-based opportunity matching, personalized career guidance, or the semantic conflict resolution crucial for a busy student's schedule. Their scope is social and logistical rather than developmental.



System	Domain	Key Features	Limitations
Indeed	Job Portal	Job listings, filtering	No chatbot, no CV parsing, no event recommendations
Wuzzuf	Job Portal	Internship/job search, user profiles	No smart recommendations, lacks personalized guidance
Luma Events	Event Management	Management	no educational context
Meetup	Community Events	Group-based events, interest-based search	No student focus, lacks academic relevance



Chapter 3: The Proposed Solution



3.1 Solution Methodology

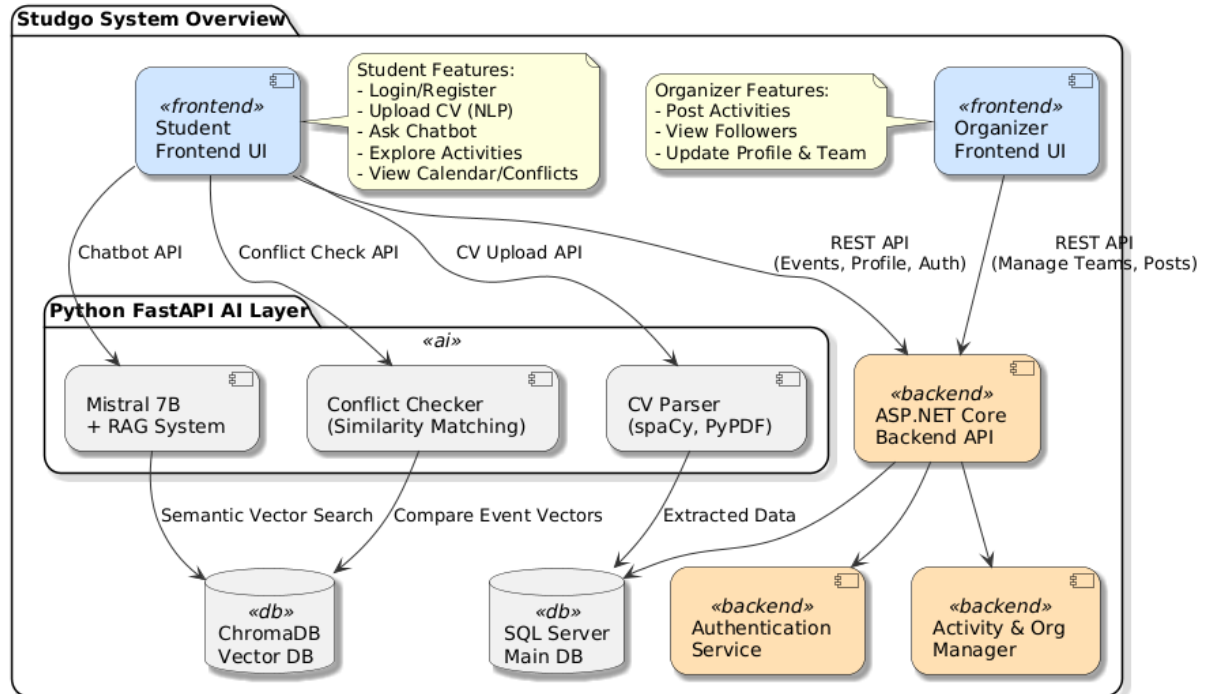


Figure 2 Overview

The proposed solution is designed to create a seamless and intelligent companion app tailored specifically for university students and student activity organizations. Recognizing the fragmented nature of current platforms where students struggle to find and manage opportunities like internships, workshops, and events, this system consolidates all essential functionalities into a single cross-platform app.



3.1.1 Architectural Overview

Modular & Scalable Architecture:

While the system is not fully microservice-based, it adopts a partially modular architecture by separating core application logic and AI functionality into distinct services.

The main backend is implemented using ASP.NET Core, handling user management, event handling, internship listings, and activity coordination.

The AI-powered chatbot, built with FastAPI and the Mistral LLM, operates as a separate lightweight service, allowing independent scaling and development.

Technology Stack:



- **Backend:** ASP.NET Core Web API — chosen for its performance, security features, and native support for RESTful APIs.
- **Frontend:** React.js SPA — provides a dynamic, responsive user experience that works smoothly on desktop, tablet, and mobile.
- **AI API:** Python FastAPI — ideal for AI workloads due to rich ML ecosystem, fast development, and asynchronous processing.
- **Database:** SQL Server — relational databases chosen for robustness, advanced querying capabilities, and data integrity.

3.1.1 Core Innovations

- **Intelligent Event Conflict Recommender:**

A service that automatically detects scheduling conflicts between activities a user has applied for. When a conflict is found, it analyzes the user's CV (skills, experience, and interests) to recommend the most relevant activity to attend. This ensures users make the most beneficial choice aligned with their career goals and background.



- **Chatbot Powered by Mistral LLM + RAG (Retrieval-Augmented Generation):**

A natural language interface where students can ask about opportunities, deadlines, eligibility, and get instant, personalized answers. The RAG system integrates external knowledge bases and real-time data for accurate, context-aware responses.

- **Web Scraping Engine for Internships:**

Continuously fetches and updates internships from popular portals Wuzzuf, aggregating opportunities without manual input.

- **Automated Agenda Generation:**

Generates a PDF agenda for a single activity, including all of its sessions, timings, descriptions, speakers, and other relevant details. This provides students with a clear, organized view of the full structure of the activity they are interested in.

- **Exploration Tools:**

Enables browsing through student organizations, their upcoming events, and workshops to encourage involvement and networking.



- **Dual Dashboards:**

- **Student Dashboard:** Focused on discovery, interaction, and personalized recommendations.
- **Organization Dashboard:** Enables event posting, tracking analytics, and communication tools.

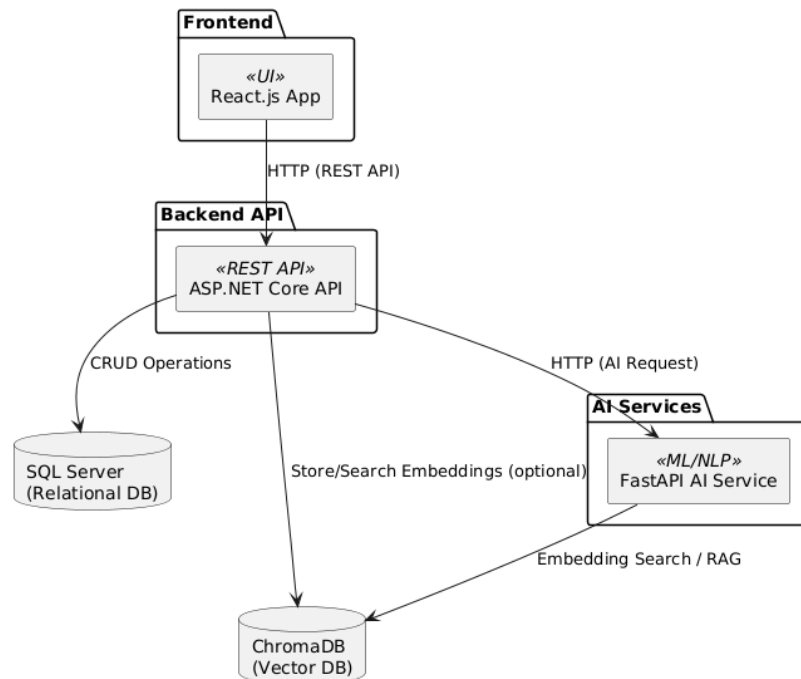


Figure 3 Basic Overview



3.2 Functional and Non-Functional Requirements

3.2.1 Functional Requirements

1. User Management:

- Registration and secure login for students and organization members.
- Role-based access controls differentiating student and organization functionalities.

2. Activity Management:

- Browse, search, and filter internships, events, and workshops.
- Apply for activities.
- Exploring Different Activity Contents And Generate Agenda

3. AI-powered Features:



- Chatbot for natural language Q&A regarding student activities and opportunities.
- Event Conflict Recommender to detect overlaps and suggest alternatives.

4. Dashboard Functionalities:

- Students: personalized feeds, agendas, conflicts, and chatbot access.
- Organizations: create and manage events, view participant data

5. Web Scraping:

- Automated background service to scrape internship postings regularly.
- Data cleansing and structuring to ensure consistent and useful internship listings.



3.2.2 Non-functional Requirements

- **Security:**
 - Use of OAuth 2.0 / JWT for authentication and authorization.
 - Input validation and secure coding practices to prevent common vulnerabilities.
- **Performance:**
 - Backend APIs optimized for low latency.
- **Maintainability:**
 - Clear separation of concerns and modular codebase.
- **Responsiveness and Accessibility:**
 - Responsive frontend design ensuring usability across devices.



3.3 System Analysis & Design

3.3.1 Use Case Diagram

Actors:

- **Student:** Primary end-user seeking activities and internships.
- **Student Activity Organization** Manages and posts activities/events.

Key Use Cases:

- Register/Login
- Browse and Filter Activities
- Apply for Activities
- Post Events (Organization)
- Conflict Detection & Recommendations



- Internship Data Fetching
- Chatbot Interaction
- Generate Activity Agenda

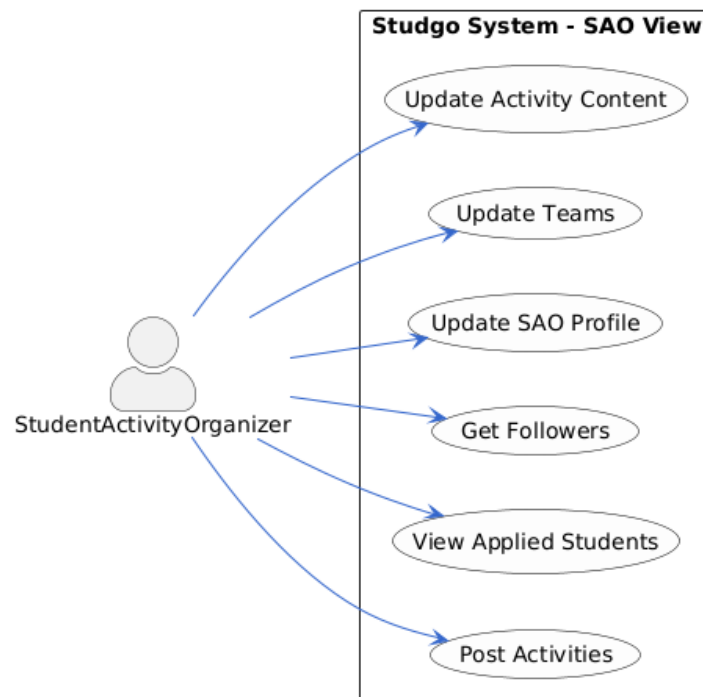


Figure 4 SA Usecase Diagram

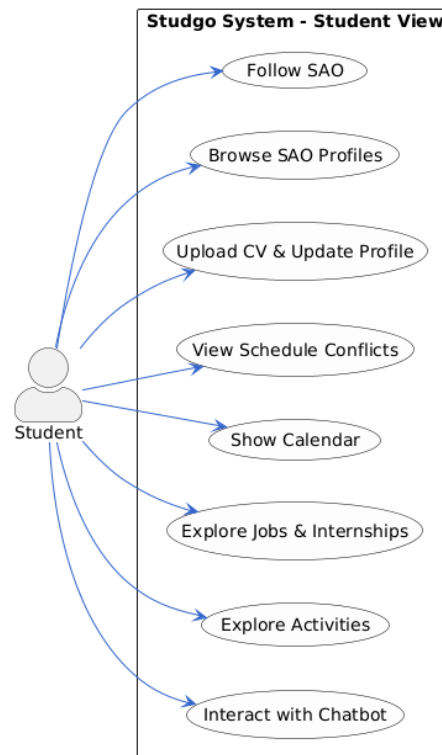


Figure 5 Student Usecase Diagram

Element	Description
Actor	Student
Use Case	Explore Activities
Description	Student browses activities and workshops posted by SAOs.
Preconditions	Student is logged in and has access to activity feed.
Postconditions	Student receives a list of available activities.



Element	Description
Actor	Student
Use Case	Explore Jobs & Internships
Description	Student browses job and internship opportunities.
Preconditions	Student is logged in
Postconditions	opportunities are displayed.

Element	Description
Actor	Student
Use Case	Show Calendar
Description	View a calendar with followed activities.
Preconditions	Student is logged in and has events to display.
Postconditions	Calendar is rendered with personalized schedule.



Element	Description
Actor	Student
Use Case	Show Calendar
Description	View a calendar with followed activities.
Preconditions	Student is logged in and has events to display.
Postconditions	Calendar is rendered with personalized schedule.

Element	Description
Actor	Student
Use Case	View Schedule Conflicts
Description	The system highlights conflicting activities.
Preconditions	Student has joined or is interested in overlapping events.
Postconditions	System notifies and suggests one to attend.



Element	Description
Actor	Student
Use Case	Upload CV & Update Profile
Description	Student uploads CV and updates personal details.
Preconditions	Student is authenticated.
Postconditions	System parses CV and used in RAG System.

Element	Description
Actor	Student
Use Case	Browse SAO Profiles
Description	Student views profiles of student activity organizers.
Preconditions	Student is logged in.
Postconditions	SAO profiles are rendered with activities and details.



Element	Description
Actor	Student
Use Case	Follow SAO
Description	Student follows an SAO to get updates.
Preconditions	Student is logged in and viewing an SAO profile.
Postconditions	Student receives future updates from the SAO.

Element	Description
Actor	Student Activity Organizer
Use Case	Post Activities
Description	Organizer posts a new activity to the system.
Preconditions	Organizer is authenticated and authorized.
Postconditions	Activity becomes visible to students.

Element	Description
Actor	Student Activity Organizer



Use Case	View Applied Students
Description	Organizer checks which students have applied to activities.
Preconditions	There are applications submitted by students.
Postconditions	Organizer sees a list of applicants and their basic info.

Element	Description
Actor	Student Activity Organizer
Use Case	Get Followers
Description	Organizer reviews the number of student followers.
Preconditions	Students have followed the organizer.
Postconditions	List or count of followers is shown.

Element	Description
Actor	Student Activity Organizer
Use Case	Update SAO Profile



Description	Organizer updates their organization's information.
Preconditions	Organizer is logged in.
Postconditions	Profile is saved and shown to students.

Element	Description
Actor	Student Activity Organizer
Use Case	Update Teams
Description	Organizer modifies team member roles or info.
Preconditions	Organizer has edit permissions.
Postconditions	Team changes are saved.

Element	Description
Actor	Student Activity Organizer
Use Case	Update Activity Content
Description	Organizer edits details of a posted activity.



Preconditions	Activity was previously posted.
Postconditions	Updated activity is reflected in listings.

Element	Description
Actor	Student
Use Case	Follow SAO
Description	Student follows an SAO to get updates.
Preconditions	Student is logged in and viewing an SAO profile.
Postconditions	Student receives future updates from the SAO.



3.3.2 Sequence Diagram

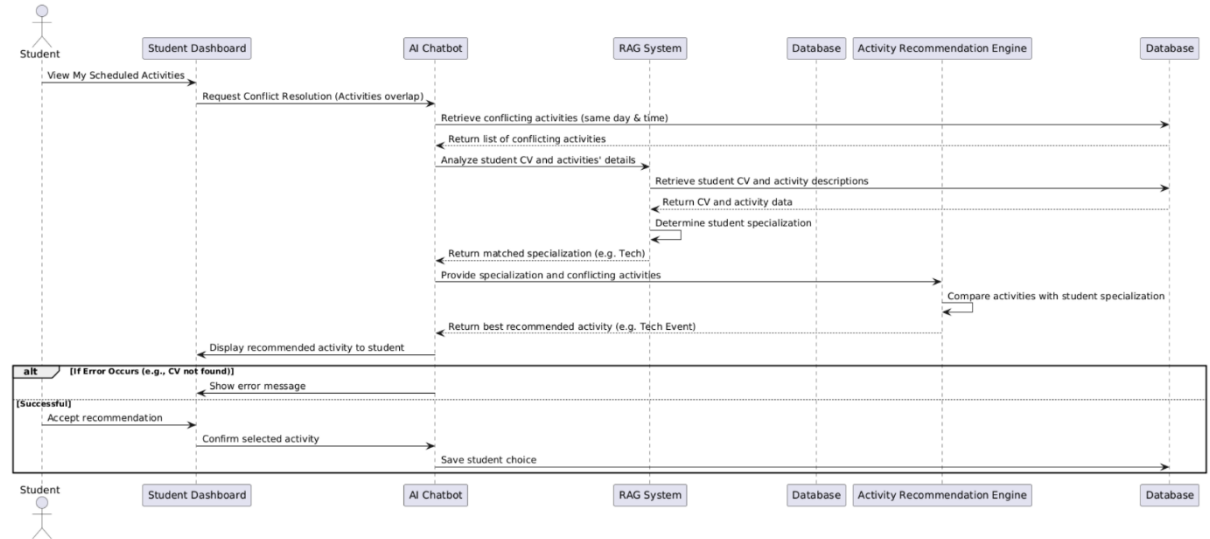


Figure 6 Activity Time Conflict Sequence Diagram

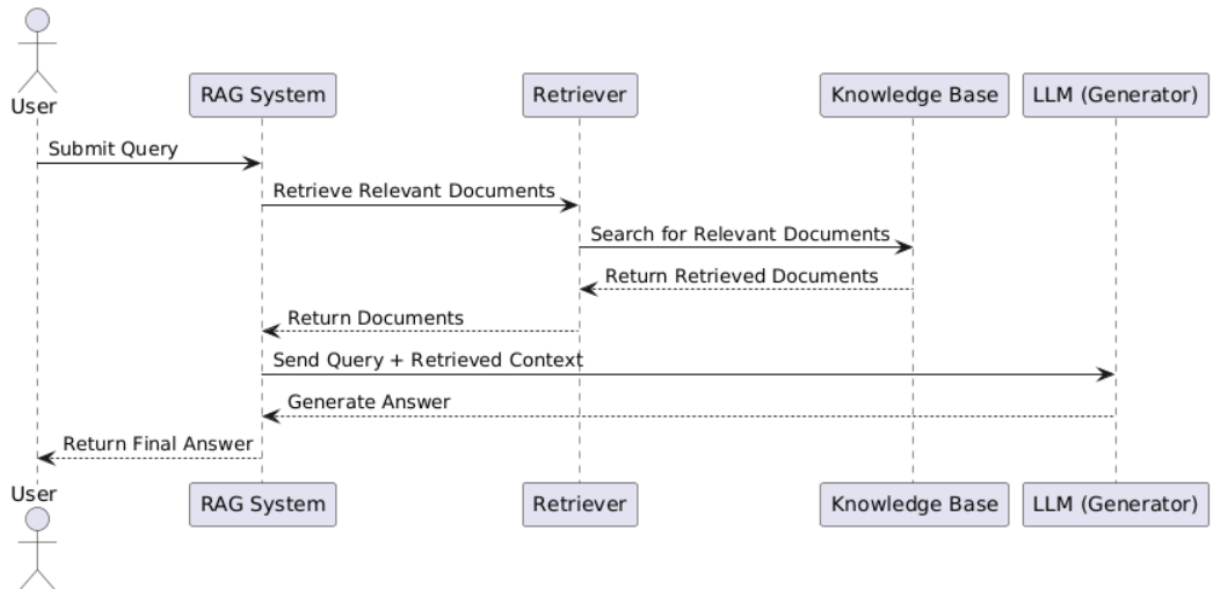


Figure 7 RAG System Sequence Diagram

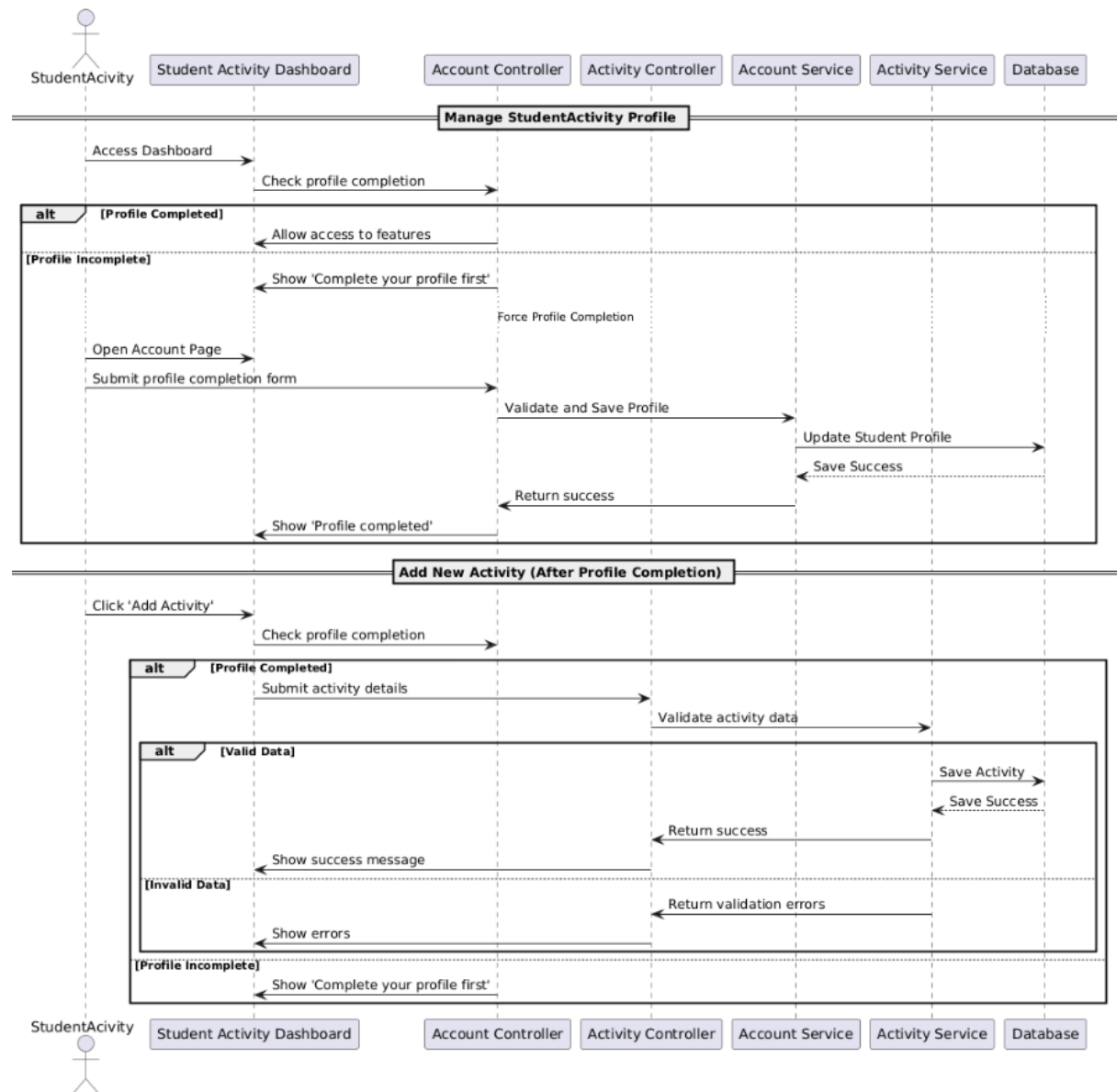


Figure 8 SA Sequence Diagram

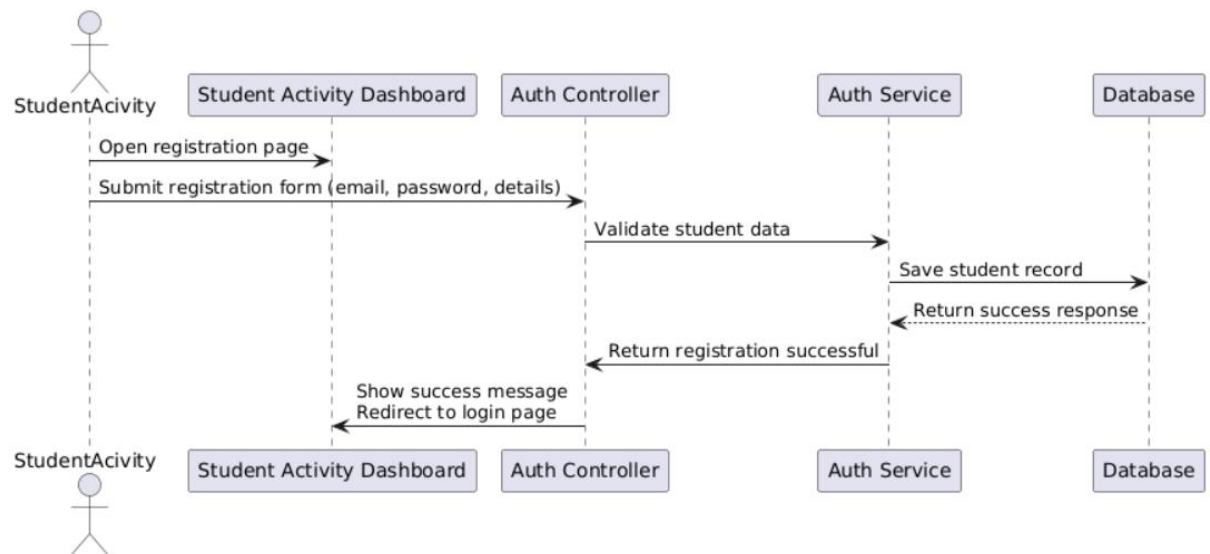


Figure 9 Auth Sequence Diagram

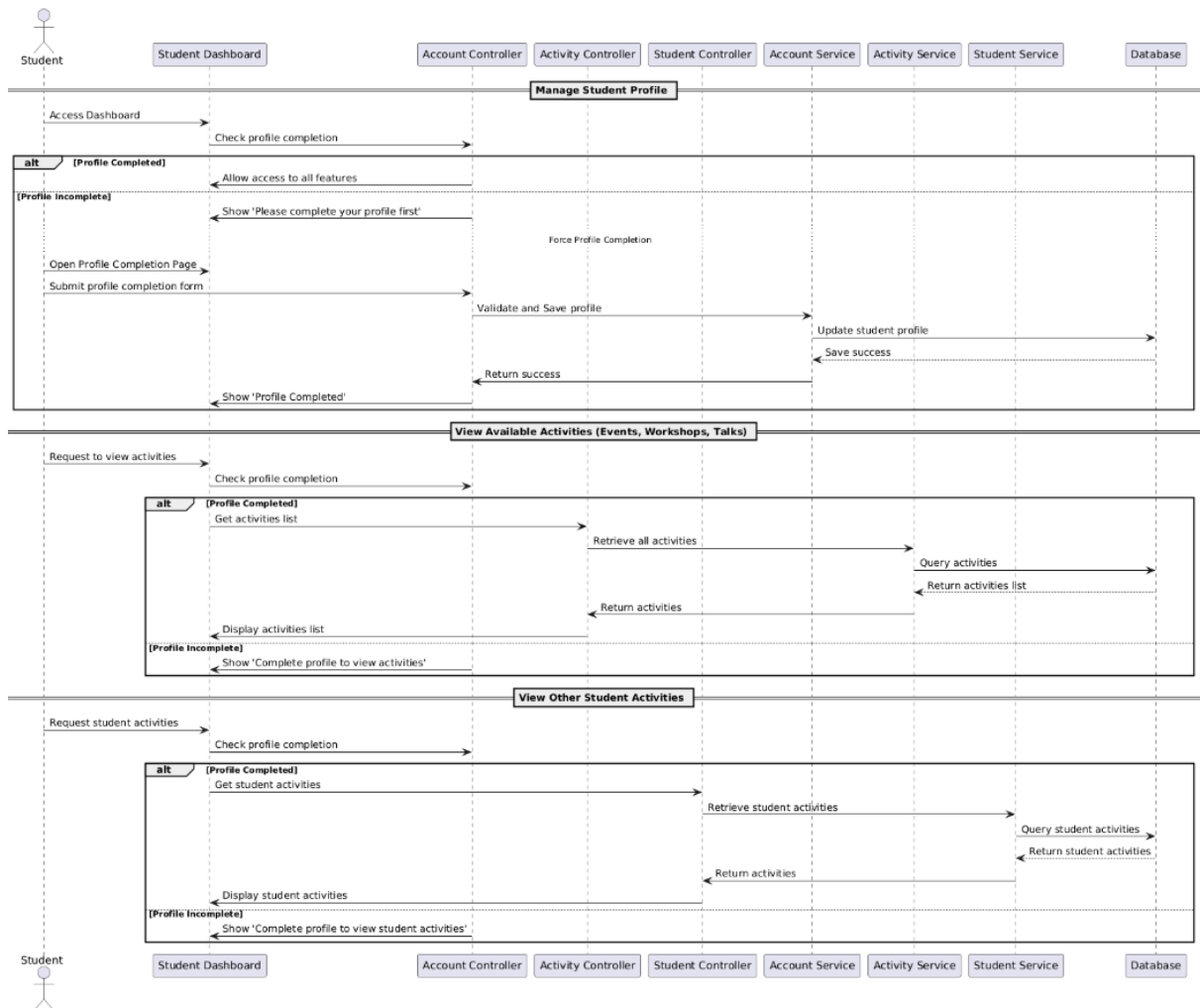


Figure 10 Student Sequence Diagram

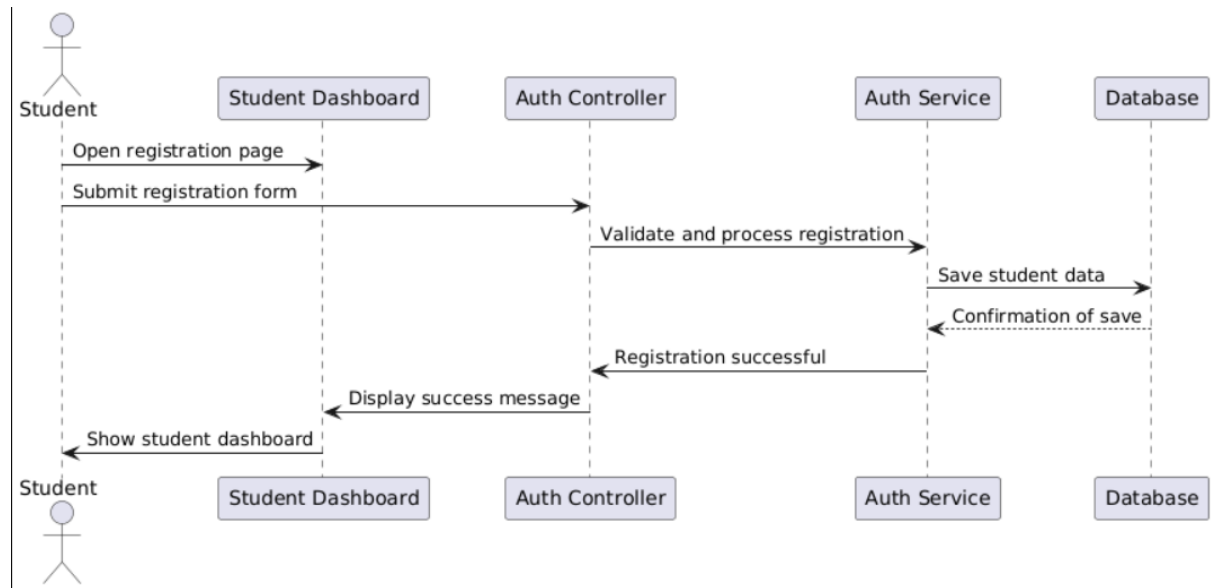


Figure 11 Student Auth Sequence Diagram

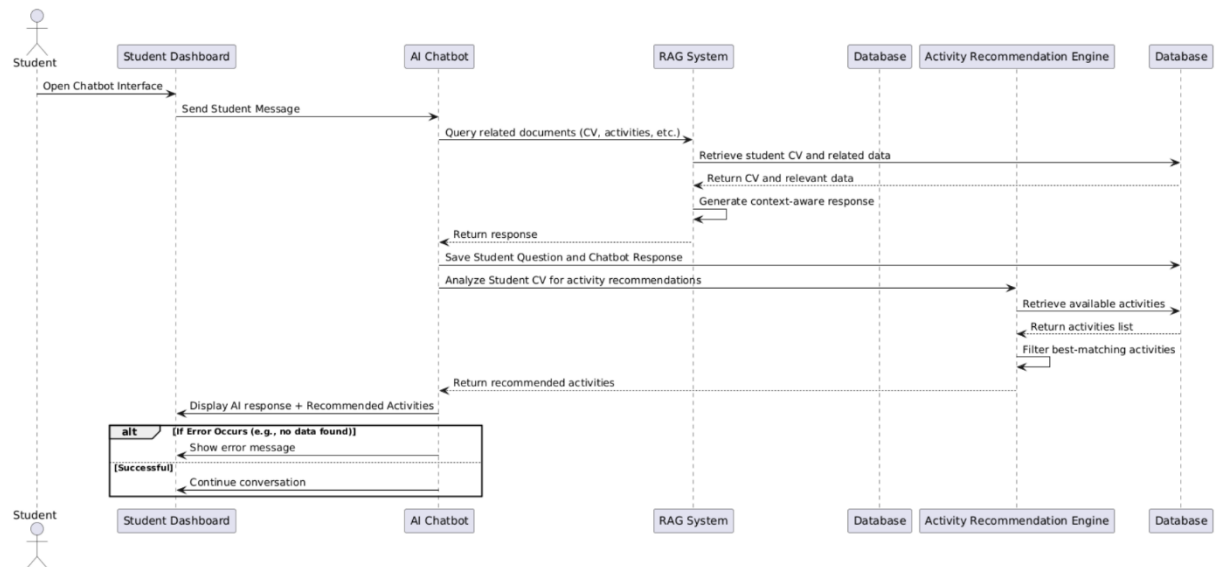


Figure 12 Chatbot Sequence Diagram

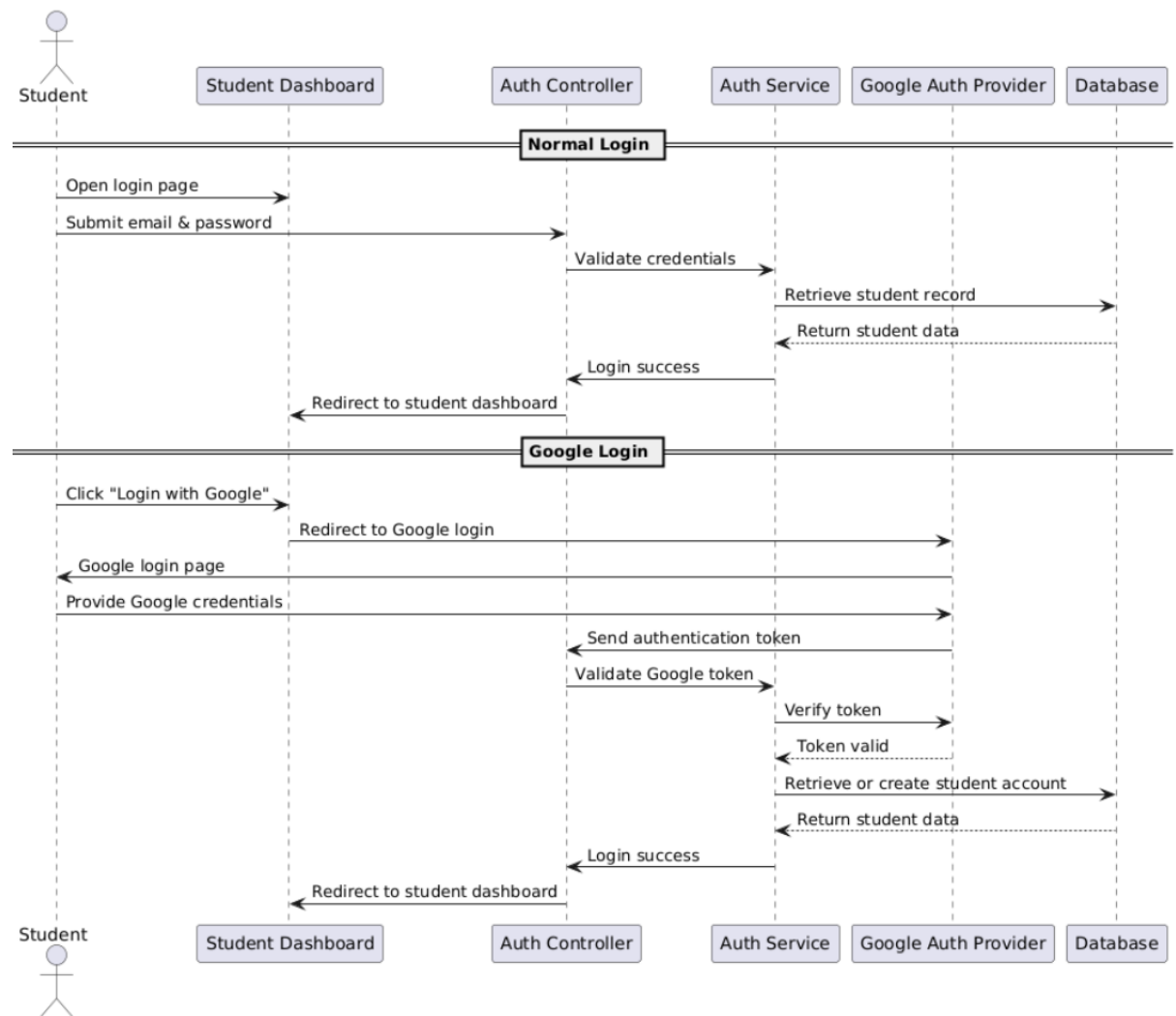


Figure 13 Google Auth Sequence Diagram





3.3.4 Activity Diagram

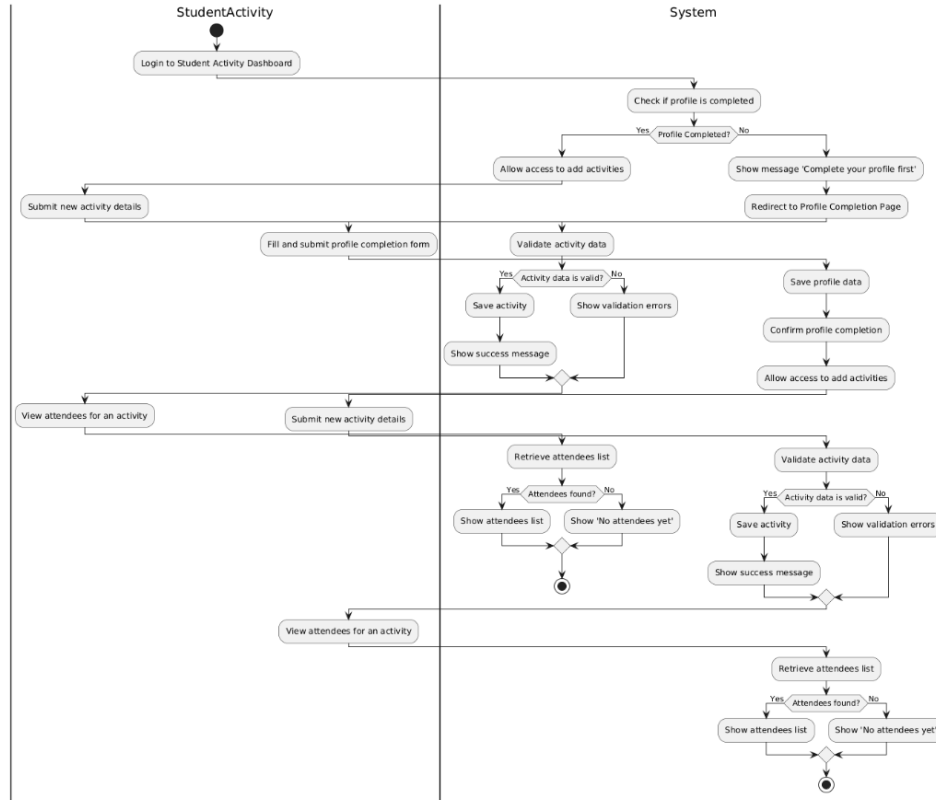


Figure 15 SA Activity Diagram

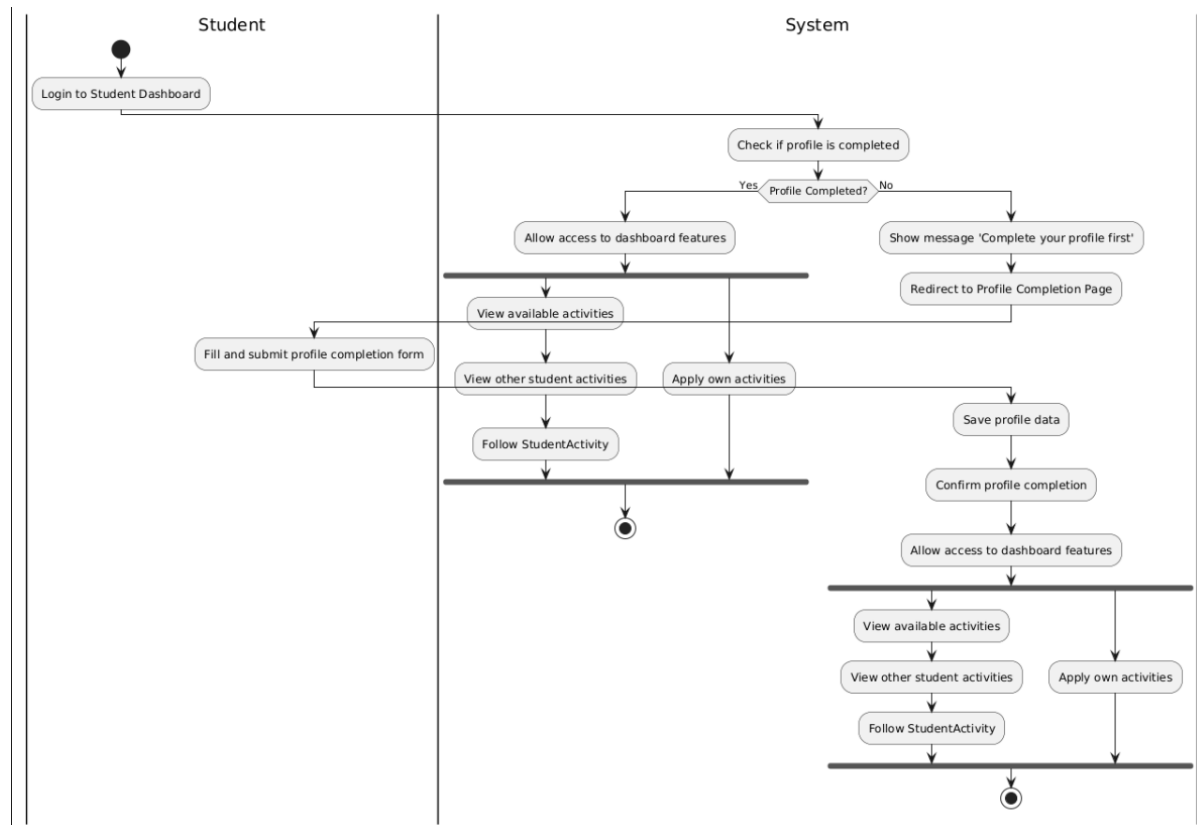


Figure 16 Student Activity Diagram

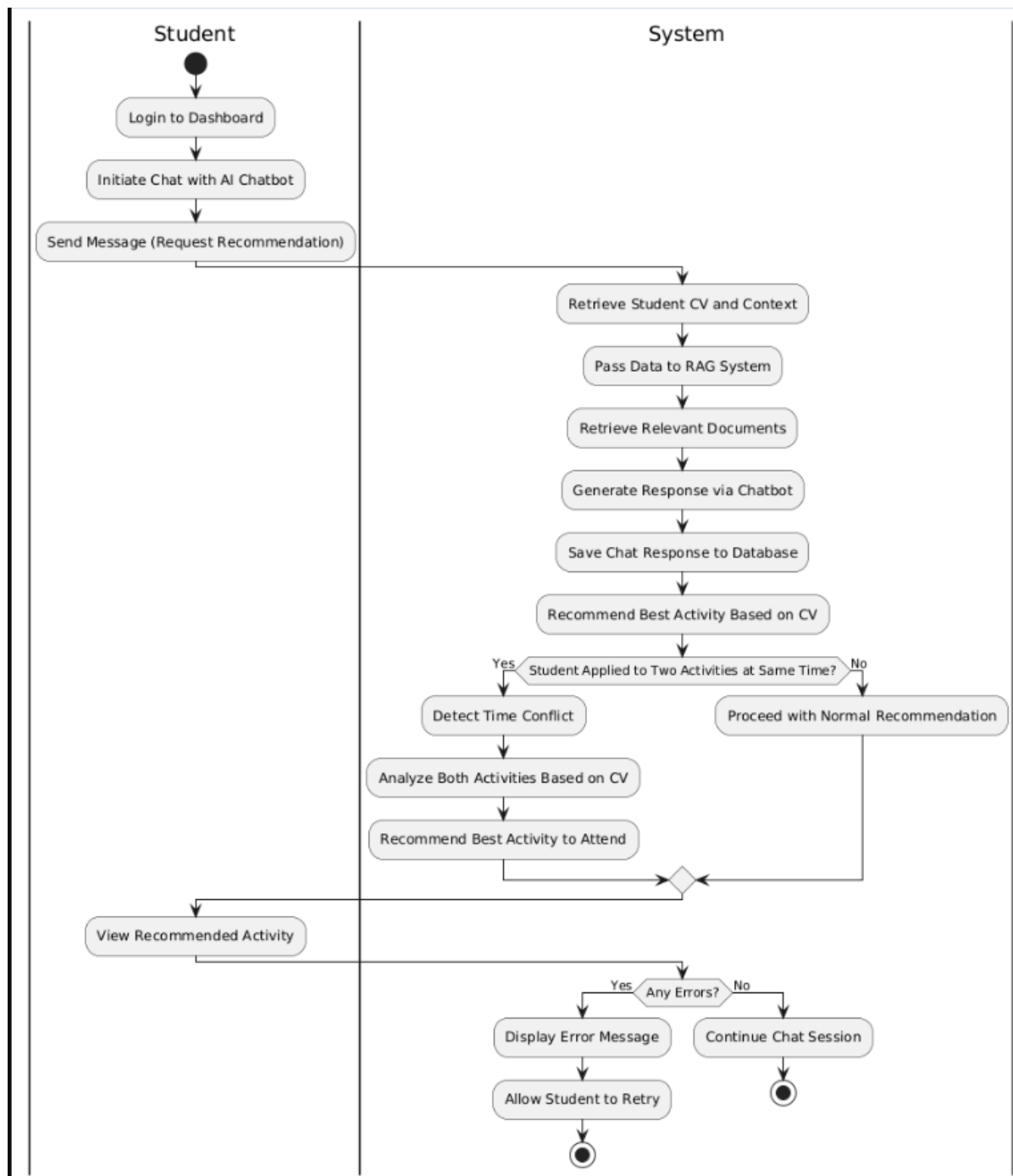


Figure 17 Chatbot Activity Diagram



Chapter 4: Experiment and Discussion



4.1 Introduction

This chapter presents the key AI-driven experiments conducted during the development of the Studgo system. These experiments demonstrate how the integration of natural language processing (NLP), vector search, and large language models (LLMs) contributed to creating an intelligent assistant and recommender engine for students. The goal was not to benchmark performance quantitatively but to validate the functional capabilities of the system components in real-world scenarios. The following sections describe the architecture, methodology, and system behavior observed during the design and deployment of two core features: the chatbot powered by a Retrieval-Augmented Generation (RAG) framework and the conflict-based event recommender.



4.2 RAG-Based Intelligent Chatbot

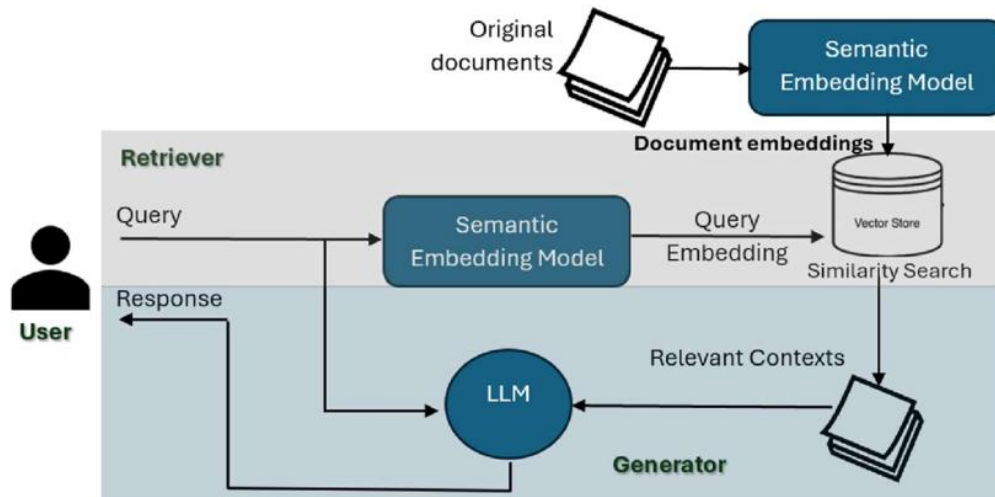


Figure 18 RAG Overview

The first experiment focused on designing a personalized assistant capable of responding to student queries with contextually accurate and semantically coherent answers. To achieve this, the system leverages a Retrieval-Augmented Generation (**RAG**) approach.

The chatbot's core functionality is built around the Mistral 7B large language model, accessed through an API backend. To provide the model with personalized and relevant information, we incorporated a vector database (**ChromaDB**) that stores semantically embedded knowledge chunks, such as previous conversations, workshop



descriptions, and extracted CV data. Upon receiving a user query, the assistant gathers relevant context by querying ChromaDB using cosine similarity over precomputed SentenceTransformer embeddings. The prompt is then dynamically constructed by combining the user query with the retrieved context and passed to the **LLM** for response generation.

This architecture enables the assistant to offer responses that are not only linguistically fluent but also tailored to the user's previous interactions and academic profile. For example, a student who uploaded a CV indicating experience in software development might receive internship recommendations aligned with programming skills.

Although no automated evaluation metrics were applied, the system consistently produced coherent and relevant answers during manual testing.



4.3 Semantic Event Conflict Recommender

A second key experiment involved designing a recommendation system that offers alternatives when a student attempts to register for overlapping events. The goal was to avoid hard rejections and instead promote student engagement by suggesting similar events.

When a conflict occurs, the system filters out overlapping events and uses a semantic similarity engine to rank remaining events. Each event description is converted into a high-dimensional embedding using **SentenceTransformers**. The similarity between the conflicted event and each alternative is calculated using cosine similarity. Top-ranked alternatives are then displayed to the user.

This approach allowed the system to recommend events that were not necessarily similar in title but shared similar themes or topics. For example, if a student tried to register for a workshop on “AI in Robotics” but it clashed with another commitment, the system might suggest



“Deep Learning in Healthcare” as a viable alternative, based on overlapping technical focus.

During development, the similarity model demonstrated robustness and efficiency. However, as with the chatbot, no formal validation was performed to measure recommendation relevance, and user feedback mechanisms were not yet in place.

4.4 CV Analysis and Personalization

To enhance personalization, a third experiment introduced the ability to process student CVs and integrate their content into the chatbot’s knowledge base. When a user uploads a CV, the system extracts textual content using PDF parsing tools and processes it through an NLP pipeline built with spaCy.

The extracted text is analyzed to identify named entities related to skills, education, work experience, and extracurricular activities. These entities are then structured and cached. When generating a response, the system retrieves relevant information from the CV cache to improve



personalization. For example, a student who mentions experience with machine learning may receive more specific opportunities or advice related to data science roles.

This feature proved particularly useful in crafting responses that felt tailored and contextually aware. Even though the NLP pipeline was not externally evaluated, informal observations during development indicated that the extracted CV information was correctly interpreted and integrated into the prompt.



4.5 Discussion of Observations

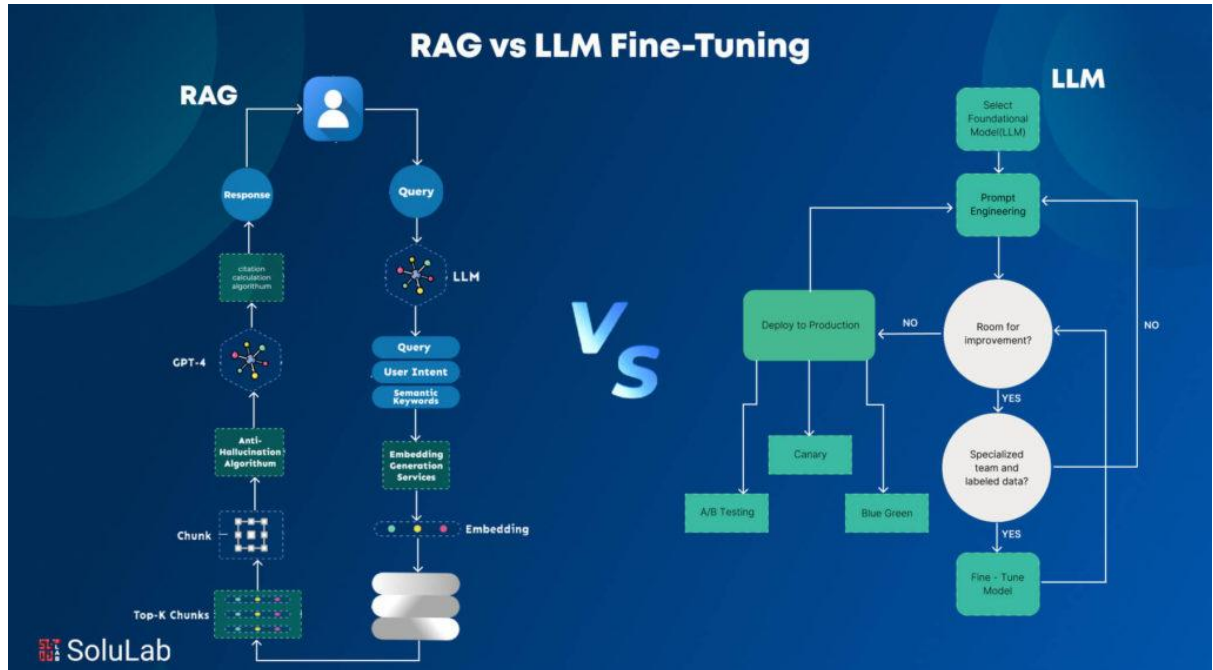


Figure 19 RAG VS FINETUNING

The experiments conducted in the development of Studgo demonstrate the successful integration of advanced AI techniques into a student-centered assistance platform. The Retrieval-Augmented Generation (RAG) framework significantly enhanced the chatbot's ability to provide intelligent, context-aware responses. By leveraging vector search via ChromaDB and embeddings from SentenceTransformers, the system effectively adapted to dynamic user input and maintained meaningful conversations. The use of the Mistral 7B model proved to be a powerful backbone for generating human-like, semantically accurate responses.



The semantic event recommender showcased another major achievement. Rather than simply notifying users of schedule conflicts, the system took a proactive approach by recommending alternative events based on content similarity. This not only improves the user experience but also encourages deeper engagement with student activities. Early observations indicated that the similarity-based filtering successfully surfaced relevant alternatives in most test cases.

Furthermore, the CV analysis component added a valuable dimension of personalization to the assistant. The ability to extract structured information from student resumes using NLP techniques allowed the system to contextualize advice and recommendations, aligning them with individual backgrounds. This significantly increased the perceived intelligence and usefulness of the assistant in informal trials.

Although formal testing and large-scale user feedback were outside the scope of the current phase, the engineering infrastructure is now in place to support such evaluations in future iterations. The modular and extensible design of the system ensures it is well-prepared for further development, refinement, and real-world deployment. Overall, the experiments successfully validated the core AI-driven features, demonstrating their feasibility, coherence, and strong potential for enhancing student engagement and support.



Chapter 5:Methodology And Implementation Details



This chapter presents a detailed explanation of the methodology adopted and the technical implementation decisions made throughout the development of the Student Activity Companion App. Designed to support student engagement in extracurricular activities, internships, and learning events, the application utilizes a modular architecture and a mix of modern technologies to ensure scalability, maintainability, and responsiveness. The methodology followed an agile-inspired approach, enabling iterative development, testing, and refinement.

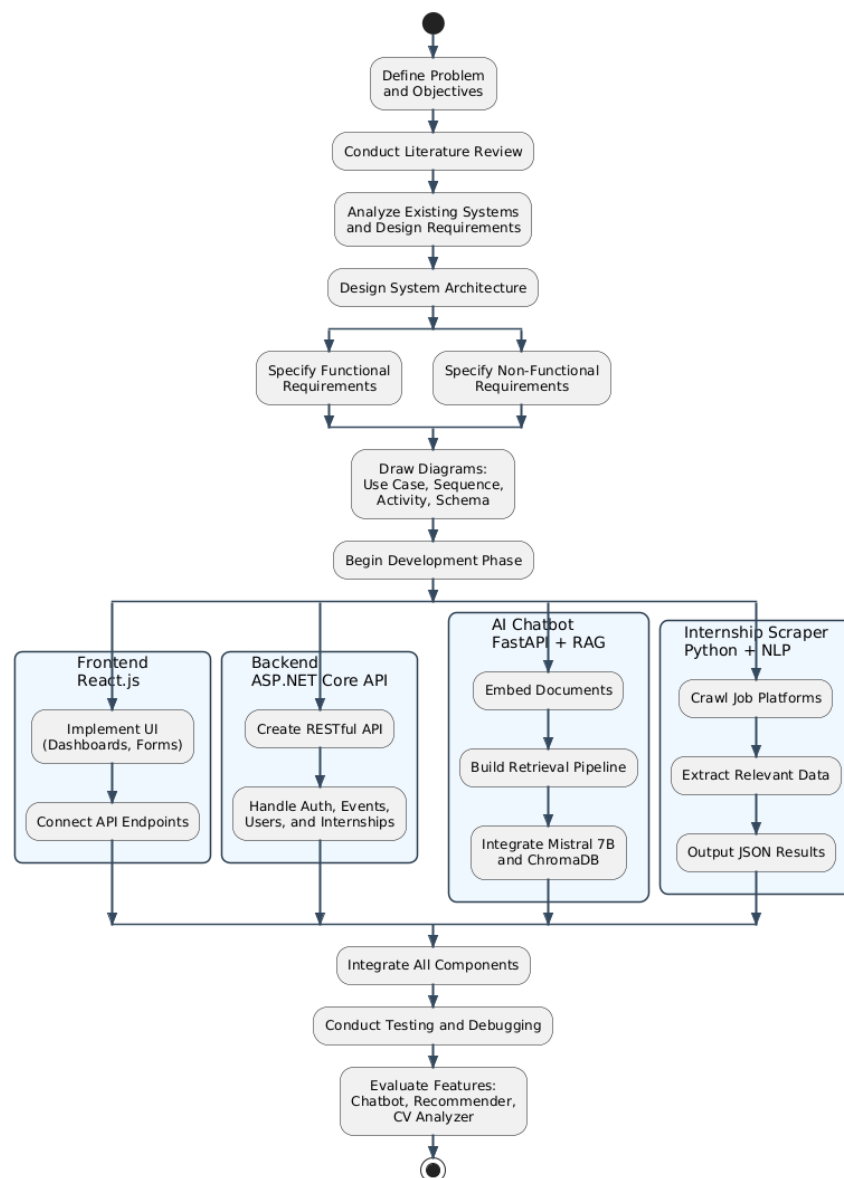
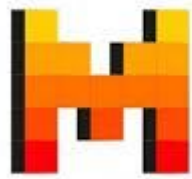


Figure 20 Main Workflow



5.1 Tools

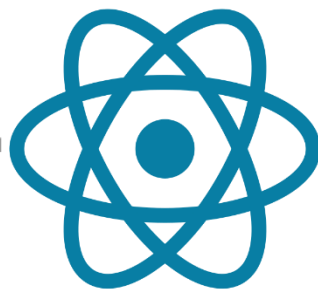


**MISTRAL
AI_**

ASP.NET



python™



Microsoft®
SQL Server®



Chroma



5.2 Phases

The development of the Student Activity Companion App was executed in multiple structured phases, ensuring efficient integration of features and technologies. The system centralizes student-related events, internships, and workshops within a smart, interactive platform. It uses **React.js** on the frontend, **ASP.NET Core** for the main backend API, and **FastAPI** for the AI chatbot. Data is stored using **SQL Server** and **ChromaDB** for vector-based document retrieval. Below is a comprehensive breakdown of the implementation process and system flow.

6.2.1 Frontend Development (*React.js*)

The frontend is built using **React.js** as a modern Single Page Application (SPA), featuring modular, reusable components and responsive design.

Key Features:

- **Role-based Interface:** Separate dashboards for students and activity organizations.
- **Student Dashboard:**
 - View events, register for activities, get conflict warnings, and access personalized recommendations.



- Embedded chatbot for support and exploration.
- **Organization Dashboard:**
 - Post and manage events and internships.
 - View student interactions and generate agendas.
- **Chat Interface:**
 - A real-time, floating chatbot interface for natural conversation.
 - Displays both student queries and assistant replies along with timestamps.
- **Routing & State Management:**
 - **React Router** and **Context API/Redux** used for navigation and state sharing across components.

5.2.2 ASP.NET Core Backend (Main API)

The main API is built using **ASP.NET Core**, structured around a layered architecture that separates controller, service, and data access responsibilities.

Key Modules:

- **Authentication & Authorization:**



- Role-based access using JWTs, with policies for students and activity organizations.
- **Event & Workshop Management:**
 - CRUD operations for events.
 - Schedule registration and validation based on availability and timing.
- **Internship Management:**
 - Fetch and present scraped internship listings.
 - Filter and categorize based on interests or domains.
- **Conflict Detection & Recommender:**
 - When a student attempts to register for an event, the system:
 - Retrieves all previously registered events.
 - Compares event start and end times.
 - Flags conflicts if there's overlap.
 - Recommends alternative sessions with similar content or different timings.



- Implemented using temporal comparison algorithms and topic similarity metrics.
- **Chat Memory & History:**
 - Every user interaction with the chatbot is saved to the **SQL Server** database.
 - Stored fields include: user ID, question, bot response, timestamp.
 - This enables:
 - Context retention across sessions
 - Smart follow-up queries
 - Auditing and personalization for future enhancements
- **Activity Agenda Generator:**
 - Generates weekly/monthly personalized event calendars for students.



5.2.3 *FastAPI Chatbot (AI Assistant + RAG)*

To deliver intelligent, context-aware assistance, a separate **FastAPI** service integrates a **Retrieval-Augmented Generation (RAG)** system powered by the **Mistral LLM** and **ChromaDB**.

Workflow:

1. User submits a message via the frontend chatbot.
2. FastAPI service:
 - Converts the query into an embedding.
 - Retrieves semantically relevant documents from **ChromaDB**.
 - Combines the retrieved context with the original question.
 - Sends the full prompt to the **Mistral LLM** for generation.
3. The reply is returned and saved in SQL Server for memory retention.



```
Algorithm RAG_Response_Generation(user_query)
  Input: user_query (string)
  Output: generated_response (string)

  Step 1: Embed the user_query
    query_embedding ← Embed(user_query)

  Step 2: Retrieve relevant documents
    retrieved_docs ← VectorStore.Search(query_embedding, top_k)

  Step 3: Concatenate retrieved_docs with user_query
    context_input ← Concatenate(retrieved_docs, user_query)

  Step 4: Generate response using LLM
    generated_response ← LLM.Generate(context_input)

  Step 5: Return generated_response
    Return generated_response
End Algorithm
```

Figure 21 RAG ALG

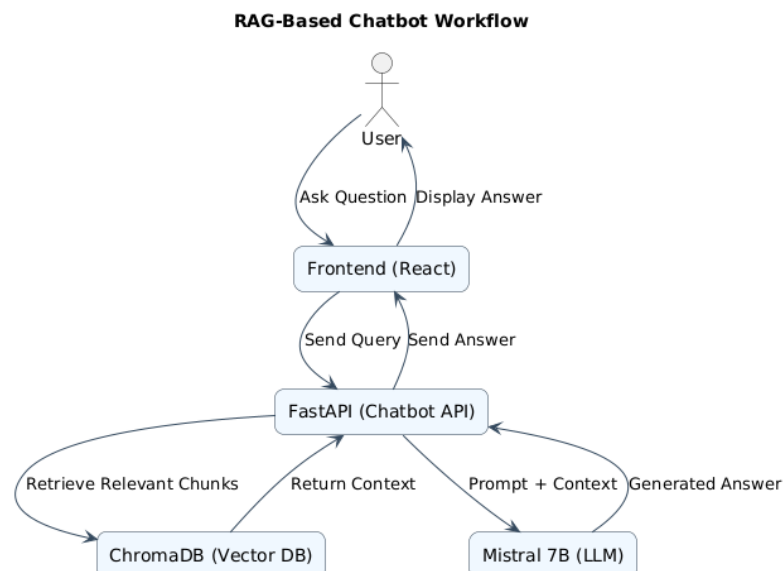


Figure 22 RAG System Flow



Why RAG Instead of Fine-Tuning:

- Fine-tuning LLMs requires:
 - Large annotated datasets
 - High training cost and compute
 - Limited flexibility for real-time content updates
- RAG allows:
 - Real-time document updates (new events, policies, FAQs)
 - No need to retrain models with every change
 - Faster deployment and better maintainability

Examples of Chatbot Capabilities:

- “What are today’s events?”
- “What is the deadline for internship registration?”
- “Tell me more about the Robotics Club.”



5.2.4 Internship Web Scraper (Python)

To provide up-to-date internships, a **Python-based scraper** was developed.

Implementation Details:

- Uses **BeautifulSoup**, **Selenium**, or **Requests** to extract listings from sites like **Wuzzuf**.
- Parsed fields:
 - Title, company, location, skills, description, duration, and deadline.
- Cleaned data is sent to be stored in the main SQL Server.
- Scheduled scraping runs periodically.

5.2.5 Integration Flow

The full system operates in a clean and maintainable pipeline.

Overall Flow:

1. **Frontend (React.js)** triggers requests based on user actions.
2. **ASP.NET Core API:**



- Handles routing, business logic, database interactions, conflict checking, and agenda generation.
- Saves and retrieves chat history.

3. FastAPI (Chatbot):

- Handles semantic understanding of student queries using **RAG + Mistral LLM**.
- Retrieves relevant documents from **ChromaDB**.

4. SQL Server:

- Stores structured data like users, events, chats, and roles.

5. ChromaDB:

- Acts as a vector database to store and retrieve semantically indexed information for the chatbot.

5.3 Technology Stack Justification

ASP.NET Core was chosen as the backend framework because it offers a robust, high-performance, and cross-platform environment capable of efficiently handling multiple concurrent users. The use of C# brings strong typing and comprehensive development tools that help improve code maintainability and reduce bugs. ASP.NET Core also provides essential built-in security features such as role-based access control and JWT authentication, ensuring student data remains protected.



Furthermore, its seamless integration with SQL Server and support for building RESTful APIs makes it an excellent choice for serving the React frontend and other clients.

React.js was selected for the frontend due to its component-based architecture, which facilitates the creation of reusable and modular user interface elements. This results in a dynamic, fast, and responsive user experience—critical for building interactive dashboards tailored for both students and student activity organizations. React's rich ecosystem of libraries and tools accelerates development while offering flexibility and modern UI/UX capabilities. Additionally, as a Single Page Application (SPA), React enables smooth navigation without full page reloads, contributing to a fluid user interaction.

For the AI chatbot functionality, **Python with FastAPI** was utilized. FastAPI is a modern, lightweight, and asynchronous framework that supports real-time interactions efficiently. Python's extensive AI and machine learning ecosystem makes it an ideal environment for integrating large language models like Mistral. Hosting the chatbot as a separate microservice allows it to be independently developed, maintained, and scaled, which improves overall system modularity and reliability.

The primary data storage relies on **SQL Server**, a mature and reliable relational database system. SQL Server's strengths lie in handling complex queries, ensuring data integrity, and supporting transactional operations. It provides a secure and structured environment for storing user profiles, event details, registrations, and chat history. Its tight integration with ASP.NET Core simplifies development workflows and database management, making it a natural choice for this application's backend.

Finally, **ChromaDB** was integrated as the vector database to support the Retrieval-Augmented Generation (RAG) system in the chatbot. ChromaDB specializes in semantic search through the storage and retrieval of vector embeddings, allowing the chatbot to return



contextually relevant responses efficiently. It supports real-time updates of the knowledge base, which means new information can be added or modified without retraining AI models—a critical feature for keeping data fresh. Moreover, ChromaDB is optimized for AI workloads and scalable, ensuring the system can maintain low latency even as the volume of data grows.

Together, these technologies provide a solid foundation for delivering a scalable, maintainable, and intelligent student activity companion app that effectively addresses the needs of its users.



5.4 UI Snapshots

5.4.1 Student Dashboard

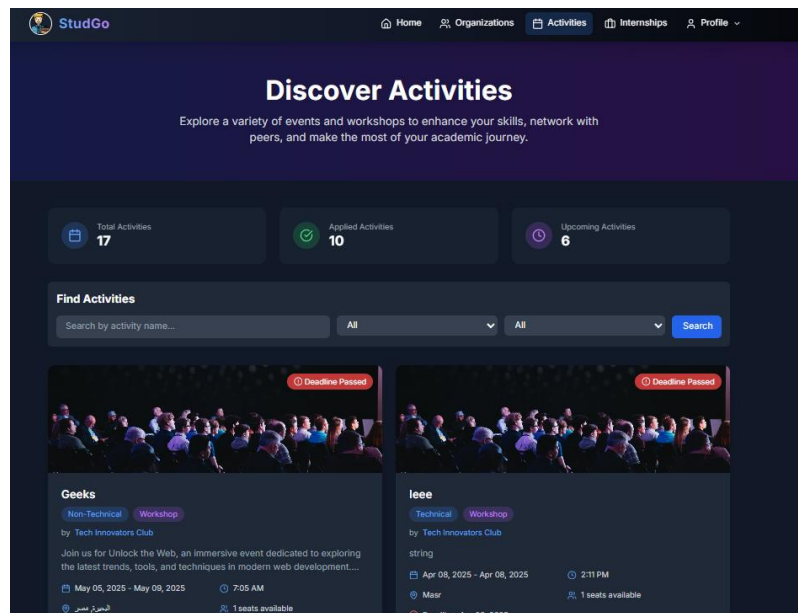


Figure 23 Student Dashboard Home UI

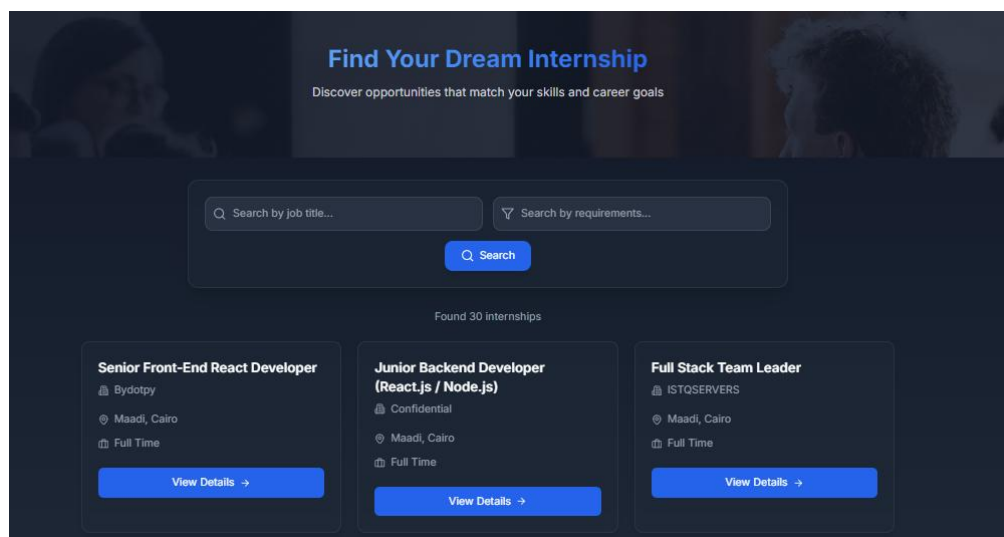


Figure 24 Student Dashboard Jobs UI

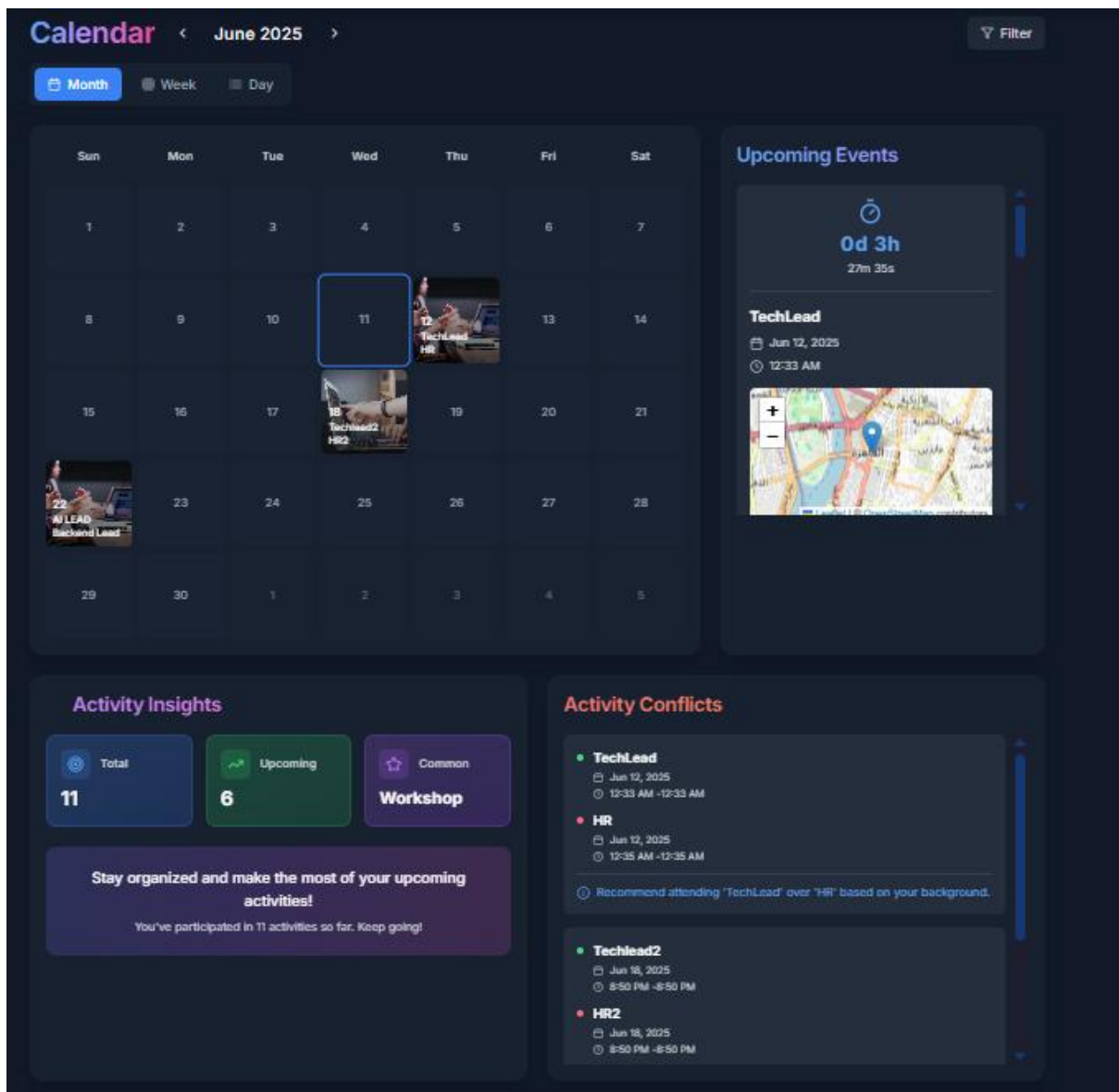


Figure 25 Calender UI



5.4.2 Student Activity Dashboard

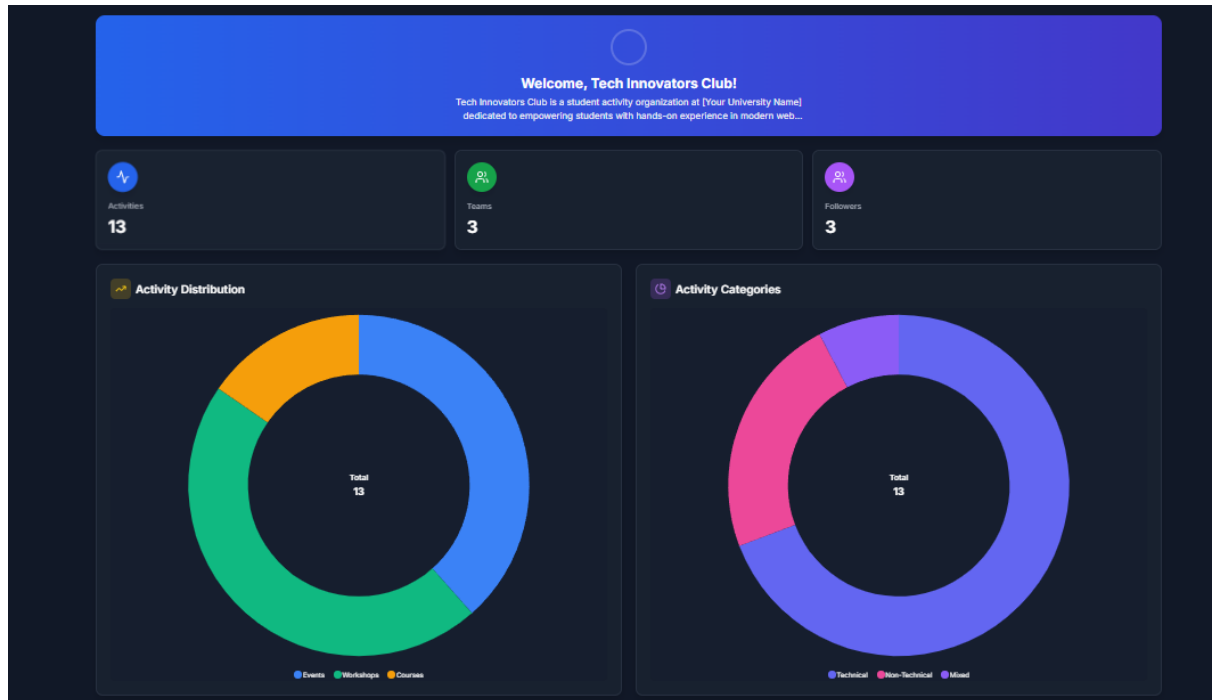


Figure 26 SA Home UI

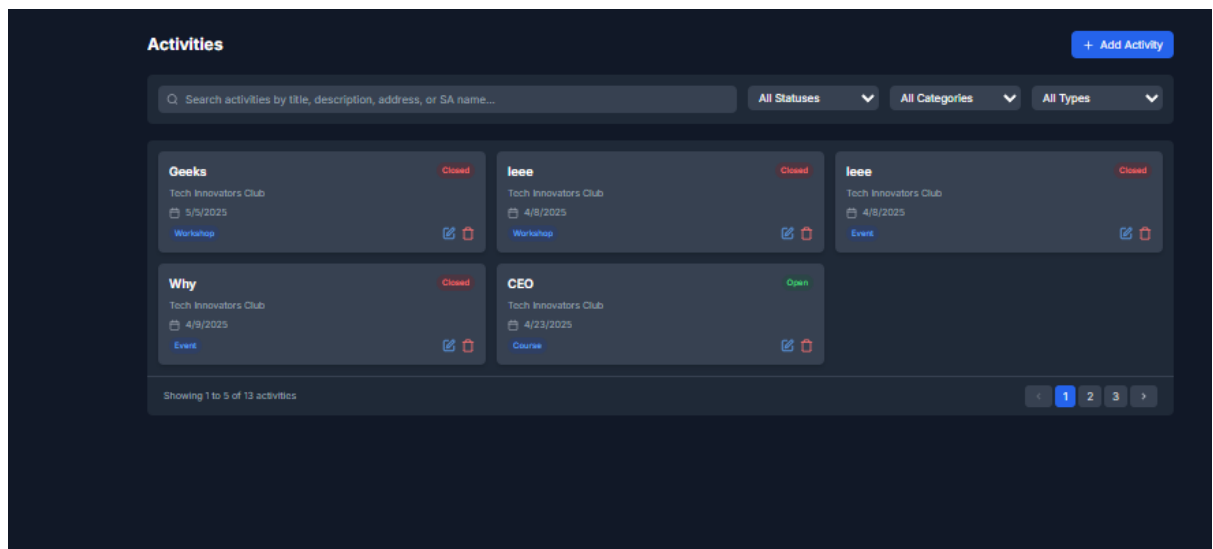


Figure 27 Manage Activity UI



[← Back to Activities](#)

Geeks

Tech Innovators Club

ClosedNonTechnicalWorkshop

[Details](#)[Contents](#)

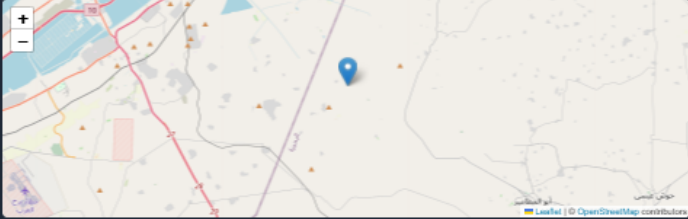
Description

Join us for Unlock the Web, an immersive event dedicated to exploring the latest trends, tools, and techniques in modern web development. Whether you're a curious beginner or a seasoned developer, this event is packed with engaging workshops, expert talks, live coding sessions, and networking opportunities.

Discover how to build responsive, accessible, and high-performance websites using technologies like HTML5, CSS3, JavaScript, React, and more. Learn best practices in frontend and backend development, deployment strategies, and how to create seamless user experiences.

Come ready to code, connect, and be inspired!

Location



الجيزة مصر

Resources

[View Agenda](#)

Date & Time

Starts
5/5/2025 7:05:00 AM

Ends
5/9/2025 7:05:00 AM

Registration Deadline
4/8/2025 8:05:00 AM

Capacity

1
Available Seats

Quick Info

- Organized by Tech Innovators Club
- Workshop Activity
- NonTechnical Category

Figure 28 SA Activity UI



Chapter 6: Conclusions and Future

Work



6.1 Conclusions

In summary, this project developed a feature-rich web application that brings together student activities, internships, and workshops into a single platform, making it easier for students to discover and participate in opportunities. The system includes a smart event conflict recommender that helps students avoid scheduling overlaps and suggests alternative options. It also features an AI-powered chatbot to assist students with inquiries, as well as an internship scraper that gathers opportunities from external websites. Users can explore different student activity organizations, view detailed agendas, and access dedicated dashboards tailored for both students and activity organizers. Overall, the platform aims to enhance student involvement, streamline event coordination, and support career development.



6.2 Future Work

Although the core objectives were achieved, several areas for future improvement and expansion were identified:

Mobile App Version: Develop a cross-platform mobile app using React Native or Flutter for more accessibility.

Real-Time Chatbot: Upgrade the chatbot to support voice input and live conversation threading.

Multi-language Support: Add Arabic and other language options to accommodate more students.

Gamification: Introduce badges or points to reward student participation in activities.

AI-Based Recommendation Engine: Expand recommendations to include personalized internship matches, event sequences, and career paths based on user profiles and past interactions.



Feedback & Review System: Allow students to rate and review activities or internships for future users.

Admin Analytics Dashboard: Provide advanced analytics and performance tracking for organizations and system admins.



References

- [1] P. Lewis et al., "Retrieval Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS 2020), arXiv preprint arXiv:2005.11401, 2020. [Online]. <https://arxiv.org/abs/2005.11401>
- [2] Y. Gao et al., "Retrieval Augmented Generation for Large Language Models: A Survey," arXiv preprint, arXiv:2312.10997, 2023. [Online]. <https://arxiv.org/abs/2312.10997>
- [3] S. Gupta, R. Ranjan, S. N. Singh, and M. Wang, "A Comprehensive Survey of Retrieval Augmented Generation (RAG): Evolution, Current Landscape and Future Directions," arXiv preprint, arXiv:2410.12837, 2024. [Online]. <https://arxiv.org/abs/2410.12837>
- [4] Y. Han, C. Liu, and P. Wang, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge," arXiv preprint, arXiv:2310.11703, 2023. [Online]. <https://arxiv.org/abs/2310.11703>
- [5] B. Han, S. L. Lehman, and D. J. Lisin, "Efficient similarity search with cosine and inner product," IEEE Transactions on Pattern Analysis and Machine Intelligence,



vol. 41, no. 5, pp. 1091–1103, May 2019. [Online].

<https://doi.org/10.1109/TPAMI.2018.2844170>

[6] Microsoft, “Introduction to ASP.NET Core,” Microsoft Learn, 2023. [Online].

Available: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core>

[7] ChromaDB Contributors, “Chroma: The AI-native open-source embedding database,” Chroma Documentation, 2024. [Online]. <https://docs.trychroma.com>

[8] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” arXiv preprint, arXiv:1908.10084, 2019. [Online].

<https://arxiv.org/abs/1908.10084>

[9] A. Beel, B. Gipp, S. Langer, and C. Breitinger, “Paper Recommender Systems: A Literature Survey,” International Journal on Digital Libraries, vol. 17, no. 4, pp. 305–338, Nov. 2016. [Online]. <https://doi.org/10.1007/s00799-015-0156-0>

[10] Microsoft, “SQL Server documentation,” Microsoft Learn, 2024. [Online].

<https://learn.microsoft.com/en-us/sql/sql-server/>



[11] Meta Platforms, Inc., “React – A JavaScript library for building user interfaces,” React Official Documentation, 2024. [Online]. <https://reactjs.org/>

[12] Microsoft, “Introduction to Razor Pages in ASP.NET Core,” Microsoft Learn, 2024. [Online]. <https://learn.microsoft.com/en-us/aspnet/core/razor-pages/>



Source Code

To review the Source code and all the data used in this project kindly

use this QR code.

