<u>CS673 Software Engineering</u>
<u>Team 1 - Eagles</u>
<u>Project Proposal and Planning</u>

Team Member	Role(s)	Signature	Date
Natasya Liew	Team Leader	*Natasya Liew*	September 7, 2024
Natthaphon Foithong	Design and Implementation Lead	*Natthaphon Foithong*	September 9, 2024
Ananya Singh	Security Lead	*Ananya Singh*	September 8, 2024
Battal Cevik	QA Lead	*Battal Cevik*	September 8, 2024
Poom Chantarapornrat	Requirement Lead	*Chan P.*	September 7, 2024
Yu Jun Liu	Configuration Lead	*Yujun Liu*	September 8, 2024

Revision history

Version	**Author**	**Date**	**Change**
1.0.0	**Yujun**	**09/10/2024**	**Sec 1,2,4,5,7**

Chatbot Web Application

Team: BU MET CS Course Building Chatbot (Group 1)

1. Overview

This project is a **web application** developed using **React, Java, and Python**, integrated with the **Llama 2** Chatbot to enhance the course selection process. The application provides students with personalized course recommendations based on data stored in a **MongoDB** database and a decision tree algorithm. It also includes features such as chat history sharing, emailing, caching, and user authentication to provide a secure and seamless user experience.
Since the BU Registration MS database is unavailable, the team has created its own database using MongoDB to store course and program data. The front end is built using React (JavaScript), while the backend leverages Java (**Spring Boot**) and Python for integrating **AI services**. The AI model and services are integrated with Llama 2 through the **Hugging Face API**. The application uses MongoDB to store course and program data, and **PostgreSQL** is used to store chat history. **Okta** is implemented for user authentication, ensuring a secure user login process.

2. Related Work

1. **Coursera:** Coursera provides course recommendations based on user history, using collaborative filtering. Our system uses Llama 2 for real-time, AI-driven conversations, offering personalized recommendations based on MongoDB-stored data and a decision tree algorithm.

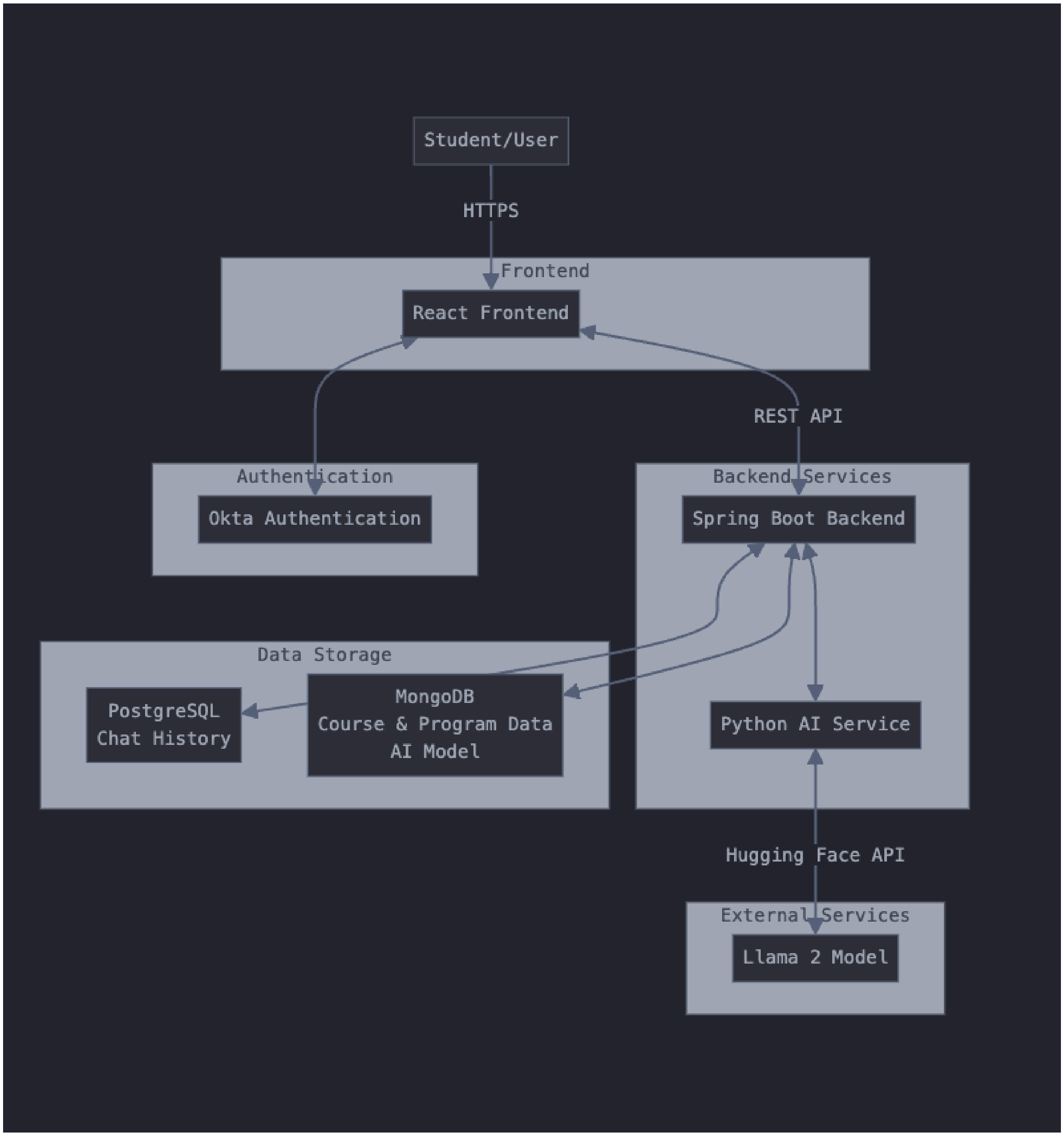
2. **EdX Chatbot**: EdX has a basic rule-based chatbot for general queries. Our chatbot, powered by Llama 2, provides more advanced, personalized course recommendations and interactive conversations.

3. **AI Chatbots for Higher Education (Ada, AdmitHub)**: These chatbots help with general student services, like admissions and support. Our system specifically focuses on personalized course selection, storing chat history in PostgreSQL, and offering chat history-sharing options.

4. **Degree Compass:** Uses predictive analytics for course recommendations. Our chatbot combines real-time AI responses with program-specific data and user inputs for more dynamic recommendations.

3. Proposed High-level Requirements

High-level design:



a. Functional Requirements
    i. Essential Features:
        1. **Provide Course Description**
            As a student, I want to read the course description before I enroll, so that I can decide whether the course is right for me.
            The agent must be able to provide the course description to the user as quickly as possible to save the time and effort of browsing through the syllabus from previous academic years.

        2. **Generate Class Schedule**
            As a student, I want to take fundamental classes before the advanced ones, so that I can understand the basics first and do well in every class.
            The AI must be able to create a recommended class schedule for the user without missing any prerequisites and comply with all program requirements to graduate.

    ii. Desirable Features:
        1. **Natural Language Interaction**
            As a user, I want to interact with the platform by using chat messages rather than clicking buttons, so that I can conveniently take a consult from the agent.
            The website must be able to receive and give information to the user in the form of natural language, such as the chat box interface.

        2. **User Authentication**
            As a student, I want to be able to see my previous activities like the chat history so that I can come back and review the class recommendation I used to ask for.
            The platform must be able to authenticate, save the chat history, and display it to the users whenever they want to.

    iii. Optional Features:
        1. **Data Caching**
            As a student, I want the website to load as fast and smoothly as possible so that I don't have to wait long.
            We may implement caching and web cookies so that the next time the user visits, the web page can be loaded quickly.

        2. **Smooth Text Transition**
            As a chat user, I want to see the text message smoothly appear letter by letter rather than a big chuck at once, so that I can slowly read along with the message.

We plan to have the generated text message appear letter by letter similar to OpenAI's ChatGPT so that the user does not have to wait for the whole paragraph to be completed to begin reading.

b. Nonfunctional Requirements
   i. Security requirements
      1. **Chat Privacy**
         As a student, I want to make sure no one can see my chat messages so that I can be more comfortable chatting with privacy.
         The website must be able to protect any unauthorized users from accessing the chat information without consent.

      2. **Protect Sensitive Information**
         As a user, I do not want to expose my password to other people so that I can be safe from malicious hackers.
         Our authentication system must encrypt and keep the login username and password secure to mitigate the risk of sensitive information leaks.

4. Management Plan

Objectives and Priorities

Objective 1: Front-end development (Front-End Engineer) - 1 member
- Priority 1: Web App UI Mock-up
- Priority 2: Implementation with React

Objective 2: Back-end development (Java Team) - 2 members
- Priority 1: High-level Architecture design
- Priority 2: Initiate server structure skeleton code/files with Spring Boot
- Priority 3: Implement MongoDB collaboratively with the AI Team

Objective 3: Integrated CI/CD and Testing (QA Lead) - 1 member
- Priority 1: Creating Github action YML file for CICD implementation
- Priority 2: Keep development integrity with each team
- Priority 3: Create test automation framework for functional, regression and integration testing

Objective 4: AI Service development (AI Team) - 2 members
- Priority 1: Prepare with AI model training
- Priority 2: Hugging Face API
- Priority 3: Collaborate with the Java team to handle responses and requests between the database and server APIs

Risk Management (need to be updated constantly)

The main risks identified by the group include team members

- **dropping out**
- **lack of motivation**
- **inconsistent work output**

which could impact the team's ability to meet deadlines.

Our team plans to frequently communicate and reassign tasks if necessary(**using Discord**), ensuring that work is equally distributed and all members are aligned with the project's goals.

Additionally, risks related to unclear requirements and scope creep were flagged, with mitigation strategies such as creating a clear requirements checklist and allowing changes only once per iteration.

The team is also aware of

- **potential integration challenges**
- **lack of testing**
- **technology learning curves**

which will be addressed by early integration efforts, setting aside extra time for testing, and providing resources for skill gaps.

Detailed Version: Risk Management Sheet

Link:https://docs.google.com/spreadsheets/d/1Niw1E7w-OHO8XMYnzISaqUA7xAgVSqSS/edit?gid=1743345481#gid=1743345481

a. Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

Iteration	Functional Requirements(Essential/ Disable/Option)	Tasks (Cross requirements tasks)	Estimated/real person-hours
1	Essential functions should achieved	Working Web App with UI, running server, responsive AI model.	20-30 hours for each team
2	Add on features	Better UI, service solutions, and well-train AI that able to achieve more functions	20-30 hours for each team
3	Fully tested and production-ready	Bug-free completed Web App production	10-20 hours for each team

5. Configuration Management Plan

a. **Tools***:*

Git & GitHub: Version control for source code, where GitHub will be the primary platform for repository hosting, code review, and collaboration.

IDEs: Team members are using different IDEs for development based on their preferences and skill sets (e.g., VSCode, IntelliJ, PyCharm, etc.).

CI/CD Tools: The CI/CD pipeline will be managed through GitHub Actions, allowing for automated testing and deployment.

GitHub Issues: GitHub issues help the team to create issues for each item/deliverable due for the agile development cycle, which clarifies the needs and current progress for the team to view.

SAST/DAST Tools: Not decided

Containerization: Docker(optional) will be used for containerizing the application to ensure consistency between development and production environments.

b. **Code Commit Guideline and Git Branching Strategy**

For our project, we will follow a Git Flow branching strategy to streamline the development process and encourage effective collaboration.

Main Branch: This branch will contain stable, production-ready code. Should be always runnable scripts and programs that achieve basic features(add on features should be released after each iteration)

Stage Branch: The stage branch acts as a pre-production environment where fully tested features are deployed before going into the main branch. It is used for final validation, such as staging deployments, where the application is tested in an environment similar to production. Once validated, the code is merged into the main branch.

Development Branch: This will serve as the integration branch for features developed by different members.

Feature Branches: Each team member will create their own feature branches off of the development branch. These branches will be used for developing new features or making improvements.

Naming Convention: feature/member-name-feature-description

Pull Requests (PRs):

- PRs must be submitted to the development branch for review.
- Code reviews are mandatory before merging. At least one other team member must approve the code.
- PRs should include tests and proper documentation, and all checks in the CI/CD pipeline must pass before merging.

c. **Deployment Plan if applicable**

The final plan should be designed after the meeting on 09/08 Sunday

Temperate plan:

The chatbot application will have a multi-phase deployment, involving API design, AI training, front-end, and back-end components.

Front-end: Design and Implement with React.js
Related tools: React.js, Axios or Fetch API, React Router, Redux or Context API, Material-UI or Bootstrap

Back-end (API): Design and Implement with Java(Spring Boot). Includes API, middleware, Auth, and servers accordingly(changes while agile dev cycle).
Related tools: Spring Boot, Spring Security, Hibernate, MySQL/PostgreSQL, Flyway or Liquibase, Swagger/OpenAPI, REST Assured, Docker

Back-end (AI): Python AI model that works alone with API that talks to back-end servers, and takes input requests rerouted from React app to server, outputs text generations to server as a response.
Related tools: TensorFlow or PyTorch, Flask or FastAPI, Gunicorn, Scikit-learn, NLTK or spaCy, Redis, Celery, Docker

6. Quality Assurance Plan

The Quality Assurance plan outlines the testing strategy, objectives, tools, and processes for ensuring the chatbot web application meets the desired quality standards. This includes both manual and automated testing, with a focus on functional, integration, and regression testing using Selenium, TestNG, REST Assured, and Java.

a. Metrics

Metric Name	Description
Number of Test Cases	The total number of test cases written for both manual and automated testing. Ensures proper test coverage.
Test Case Pass Rate	The percentage of test cases that pass successfully. Formula: (Passed Test Cases / Total Test Cases) * 100
Defect Density Rate	Measures the number of defects per KLOC (thousand lines of code). Formula: (Total Defects / KLOC).
Code Coverage	Percentage of code covered by unit and integration tests. Helps ensure that the most critical parts of the code are tested.
# of User Stories Completed	Tracks how many user stories are completed in each sprint/iteration. Helps measure progress and project velocity.

b. Coding Standard
The project will follow industry best practices and specific coding standards to maintain consistency, readability, and quality across the codebase.
- Naming Conventions: Variables, methods, and classes should use meaningful names that convey the purpose of the code.
- Classes: Use PascalCase (e.g., CreateBackendService).
- Methods and Variables: Use camelCase (e.g., getCourseDetails, studentName).
- Constants: Use ALL_CAPS for constants (e.g., MAX_RETRY_COUNT).
- Indentation: Use 4 spaces for each level of indentation (avoid tabs).
- Commenting: Use Javadoc-style comments for public methods and classes. Inline comments should only be used to explain complex logic.
- Code Structure: Keep methods short and focused on a single task (preferably less than 50 lines). Classes should follow the Single Responsibility Principle (SRP).
- Error Handling: Use proper exception handling (try-catch blocks) and avoid suppressing exceptions silently.
- Version Control: Each commit should address one logical change, with descriptive commit messages.
- Code Formatting Tools: Use Prettier for frontend (JavaScript) and Checkstyle for backend (Java).

c. Code Review Process

The code review process ensures high-quality code and consistency across the team, involving reviews for each pull request before merging into the main codebase.

- Who Reviews the Code: The design leader will review all architectural changes. The implementation lead will review all backend and API-related code.
- The security lead will review code related to user authentication and chat history security.
- Team members are encouraged to cross-review each other's code to foster collaboration and knowledge sharing.
- Review Checklist: Functionality: Does the code meet the requirements of the task? Are all use cases handled?
- Readability: Is the code easy to understand, and are the comments clear where necessary?
- Coding Standards: Does the code follow the established coding standards (naming, formatting, etc.)?
- Error Handling: Are exceptions properly handled? Is the code resilient to failure?
- Security: Does the code meet security best practices? Are sensitive data and authentication mechanisms secured?
- Testing: Are there sufficient unit tests, and do they cover edge cases? Are all tests passing?
- Performance: Is the code efficient and optimized for performance (especially for APIs and database calls)?
- Pull Requests: Every pull request (PR) must be reviewed by at least one other team member. PRs should include relevant unit tests and be merged only after all CI/CD tests pass. Use GitHub PRs to manage code reviews. Reviewers will leave comments and suggest improvements before approving the merge.
- Feedback: Reviewers should provide constructive feedback, highlighting areas of improvement as well as what was done well. If necessary, a follow-up review may be requested after changes are implemented.

d. Testing

1- Types of Testing:

Unit Testing: Developers will test individual modules (React components, Java services, Python AI model) using JUnit, PyTest, and Jest.

Integration Testing: Ensure that the backend (Spring Boot APIs, AI model) interacts seamlessly with the front end and the databases (MongoDB, PostgreSQL).

Regression Testing: Simulate the user's journey across the web app, ensuring the integration between the UI and backend services is smooth.

Performance Testing: Measure the response time of the chatbot, database queries, and API calls.

Security Testing: Ensure user authentication (Okta) and data protection (chat history) are robust.

2- Manual Test Plan:
The Manual Functional Test Plan focuses on verifying the core functionality of the web application by manually executing test cases. Testers will validate features like course recommendations, user authentication, and AI interactions to ensure they work as expected. Each test will be performed from the user's perspective, with any issues documented and reported for resolution. This plan ensures the application meets its functional requirements before moving to further testing phases.

3- Automation Test Plan and Framework:
A modular, scalable automation framework is essential for handling UI, API, and backend services.

Language: Java (for both Selenium & REST Assured). Framework: TestNG (for test case management), Selenium (UI), REST Assured (API), Maven (build tool). Automation will reduce manual efforts and increase the speed of regression testing.

- Frontend Automation: Use Selenium with TestNG for automating UI tests. Test cases include form submissions (login, course search), verifying course recommendations, and chat history functionality.
- Test Cases UI Testing (Selenium) Login and authentication using Okta. Course selection and recommendations. Chat history retrieval and display.
    - Tools: Selenium WebDriver, TestNG, Maven.

- Backend/API Automation: Use REST Assured for API testing (Java Spring Boot endpoints). Test cases include verifying API responses (course retrieval, chat history save, etc.), response times, and security headers.
- API Testing (REST Assured) Verify response of course list API (GET /courses). Test chat history save/retrieve (POST /chat/save, GET /chat/history). Check error responses for unauthorized access (403 Forbidden).

    - Tools: REST Assured, Postman for API testing, Spring Boot testing libraries.

- Database Automation: Verify MongoDB (course/program data) and PostgreSQL (chat history) through API queries. Ensure data consistency, retrieval accuracy, and security for user data.

- Performance Testing: Use JMeter for load testing API endpoints to simulate multiple users accessing the course recommendations at the same time.

Code Review Process for Automation Test:
PRs for all code changes, reviewed by at least one peer. All test cases must be validated, and automated tests should pass in CI/CD pipelines before merging.

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

e. Defect Management:
GitHub Issues for tracking bugs and issues.
Types of Defects: UI bugs (e.g., misaligned elements, broken navigation).
API issues (e.g., incorrect data returned, slow response).
Security vulnerabilities (e.g., unprotected data).
Defect Workflow: Issues will be logged in GitHub with details, assigned to relevant developers, and tracked until resolved. Bug fixes will be re-tested again.

7. References
UI - Natt-
https://www.figma.com/design/gjNG1bADwnFxgDqclMwQVQ/Chat-AI-Bot---673ONE?node-id=0-1&node-type=canvas

8. Glossary
AI = Artificial Intelligence