**SP-100 — Campus AI Companion**

**Final Report**

**CS 4850 - Section 02 – Fall 2024**

**November 15, 2024**

*A group of people posing for a photo

Description automatically generatedwebsite link and github link…..*

**Team Members:**

|  |  |  |
| --- | --- | --- |
| Name | Role | Cell Phone / Alt Email |
| Dorian (Team Lead) | Developer | 470.439.9907  doriantaponzing@gmail.com |
| Fredy | Project Owner and Developer | 470.836.8552  atakoutene@gmail.com |
| Yann | Developer | 404.649.4887  thisisyanndev@gmail.com |
| Leopold | Developer | 470.838.2345  leopoldgatsing@gmail.com |
| Manuella | Documentation | 404.599.6647  manuellakoodjo@gamil.com |
| Sharon Perry | Project Advisor | 770.329.3895  [Sperry46@kennesaw.edu](mailto:Sperry46@kennesaw.edu) |

Table of Content

[**I.** **Introduction** 5](#_Toc183961769)

[1. Overview 5](#_Toc183961770)

[2. Project Goals 5](#_Toc183961771)

[3. Definitions and Acronyms 5](#_Toc183961772)

[4. Assumptions 6](#_Toc183961773)

[**II.** **Functional Requirement** 6](#_Toc183961774)

[1. Login and Account Management 6](#_Toc183961775)

[2. Authentication 6](#_Toc183961776)

[3. Recommendation/Suggestion Management 6](#_Toc183961777)

[4. Schedule Management: 7](#_Toc183961778)

[**III.** **Non-Functional Requirements** 7](#_Toc183961779)

[1. Usability: 7](#_Toc183961780)

[2. Scalability: 7](#_Toc183961781)

[3. Security: 7](#_Toc183961782)

[4. Maintainability: 8](#_Toc183961783)

[5. Portability: 8](#_Toc183961784)

[6. Compliance: 8](#_Toc183961785)

[**IV.** **External Interface Requirements** 8](#_Toc183961786)

[1. User Interface Requirements 8](#_Toc183961787)

[2. Hardware Interface Requirements 9](#_Toc183961788)

[3. Software Interface Requirements 9](#_Toc183961789)

[4. Communication Interface Requirements 9](#_Toc183961790)

[V. Design Considerations 9](#_Toc183961791)

[1. Assumptions and Dependencies 9](#_Toc183961792)

[Assumptions 9](#_Toc183961793)

[Dependencies 11](#_Toc183961794)

[1. General Constraints 13](#_Toc183961795)

[2. Development Methods 16](#_Toc183961796)

[Agile Development Methodology 16](#_Toc183961797)

[Test-Driven Development (TDD) 17](#_Toc183961798)

[Continuous Integration and Continuous Deployment (CI/CD) 17](#_Toc183961799)

[Code Review and Pair Programming 18](#_Toc183961800)

[Documentation and Knowledge Sharing 18](#_Toc183961801)

[Security and Compliance 19](#_Toc183961802)

[User Feedback and Usability Testing 19](#_Toc183961803)

[Deployment and Release Management 20](#_Toc183961804)

[Post-Release Monitoring and Maintenance 20](#_Toc183961805)

[VI. Architectural Strategies 21](#_Toc183961806)

[1. Modular Architecture 21](#_Toc183961807)

[2. Microservices Architecture (Backend) 21](#_Toc183961808)

[3. Scalable and Flexible Data Storage 21](#_Toc183961809)

[4. AI Integration Strategy 22](#_Toc183961810)

[5. Cross-Platform Frontend Development 22](#_Toc183961811)

[6. Security Strategy 22](#_Toc183961812)

[7. CI/CD Pipeline and Automated Testing 23](#_Toc183961813)

[8. Native and Serverless Considerations 23](#_Toc183961814)

[9. Performance Optimization 24](#_Toc183961815)

[10. Monitoring and Logging Strategy 24](#_Toc183961816)

[11. User Feedback and Analytics 24](#_Toc183961817)

[VII. System Architecture 24](#_Toc183961818)

[1. Overview 24](#_Toc183961819)

[2. Component Diagram 25](#_Toc183961820)

[3. Module Descriptions 25](#_Toc183961821)

[VIII. Detailed System Design 26](#_Toc183961822)

[Classification Section 26](#_Toc183961823)

[Presentation Layer (Frontend): 27](#_Toc183961824)

[Application Layer (Backend): 27](#_Toc183961825)

[Data Layer: 27](#_Toc183961826)

[Definition Section 28](#_Toc183961827)

[Frontend Subsystem: 28](#_Toc183961828)

[Backend Subsystem: 28](#_Toc183961829)

[Database Subsystem: 29](#_Toc183961830)

[External Services Subsystem: 29](#_Toc183961831)

[Constraints 29](#_Toc183961832)

[Performance Constraints: 30](#_Toc183961833)

[Security Constraints: 30](#_Toc183961834)

[Scalability Constraints: 30](#_Toc183961835)

[Compliance Constraints: 30](#_Toc183961836)

[Resources 30](#_Toc183961837)

[Hardware: 30](#_Toc183961838)

[Software: 30](#_Toc183961839)

[Third-Party Services: 31](#_Toc183961840)

[Interface/Exports 31](#_Toc183961841)

[Frontend Interfaces 31](#_Toc183961842)

[Backend Interfaces 32](#_Toc183961843)

[Data Exports 33](#_Toc183961844)

[IX. Architecture Drawings 33](#_Toc183961845)

[1. Scraped Data Source 35](#_Toc183961846)

[2. Pinecone (Vector Database) 35](#_Toc183961847)

[3. GPT-3.5 Model 35](#_Toc183961848)

[4. Firebase Services 36](#_Toc183961849)

[5. Mobile Client 36](#_Toc183961850)

[Data Flow and Interaction Summary 36](#_Toc183961851)

[X. Version Control Discussion 37](#_Toc183961852)

[XI. Challenges and Assumptions 38](#_Toc183961853)

[2. Limitations of the OpenAI API Free Tier: 38](#_Toc183961854)

[ Limited Access to the Most Updated Data: 38](#_Toc183961855)

[ Context Window Constraints: 38](#_Toc183961856)

[ Limited Prompt Comprehension: 38](#_Toc183961857)

[3. Matching Users’ Prompts to the Right Agent or Tool: 38](#_Toc183961858)

[ Keyword Matching: 39](#_Toc183961859)

[ Classifier-Based Approach: 39](#_Toc183961860)

[ Vector Similarity Matching: 39](#_Toc183961861)

[XII. Test Plans 39](#_Toc183961862)

[1. Test Items 39](#_Toc183961863)

[2. Testing Approach 40](#_Toc183961864)

[3. Entry and Exit Criteria 40](#_Toc183961865)

[4. Test Environment 40](#_Toc183961866)

[5. Test Cases 40](#_Toc183961867)

[1. Functional Requirements 41](#_Toc183961868)

[2. Non-Functional Requirements 41](#_Toc183961869)

[Software Test Report (STR) for Campus AI Companion 42](#_Toc183961870)

[Test Summary 42](#_Toc183961871)

[Detailed Test Results 42](#_Toc183961872)

[**Conclusion** 44](#_Toc183961873)

[**Appendix** 45](#_Toc183961874)

# **Introduction**

## Overview

The Campus AI Companion is an AI-powered mobile platform built on top of the OpenAI API to help CS students manage their academic and career paths. This app will offer them personalized guidance, connect students to resources, and help them achieve their goals. Initially focused on CS students, the platform will gradually expand to include other disciplines, providing tailored support based on each major’s requirements.

## Project Goals

* Provide personalized recommendations and suggestion based on user information, courses, and activities being involved in. These recommendations include: course suggestions, club and event recommendations, career path and Job recommendations.
* Enable students to manage their schedule.

## Definitions and Acronyms

1. **AI:** Artificial intelligence
2. **RAG:** Retrieval Augmented Generation
3. **API:** Application Programming Interface
4. **SRS:** Software Requirements Specification.
5. **SDD:** Software Design Document.
6. **LLM:** Large Language Model

## Assumptions

* Users will have access to mobile devices and the internet.
* Users will have basic technical knowledge to navigate mobile applications.

# **Functional Requirement**

## Login and Account Management

* **Create Account**: Users can register.
* **Login**: Users log in using their registered credentials.
* **Password Recovery**: Users can recover their passwords through email.

## Authentication

* **User Authentication**: Firebase Authentication to verify user identities.
* **Session Management**: Maintain user sessions with secure tokens.

## Recommendation/Suggestion Management

* Based on the user’s prompt and context provided to our LLM, the app will provide some answers, recommendation, and suggestions to guide the user.
* The app will also provide resources to assists students in their everyday activities and to grow into their academic life.

## Schedule Management:

* Based on the knowledge of users’ day to day activities, the app will generate a timetable for each user.
* The app will also provide reminders about the user’s schedule to make sure he misses no important tasks.

# **Non-Functional Requirements**

## Usability:

The platform should have an intuitive and user-friendly interface that allows students to easily navigate through features like personalized guidance and resource connections.

## Scalability:

The system should be designed to handle an increasing number of users as it expands from serving CS students to other disciplines.

It should support scaling both horizontally and vertically to accommodate growth in data, user interactions, and API calls.

## Security:

The platform must ensure secure authentication and authorization using Firebase Authentication, protecting user data with encryption both at rest and in transit.

## Maintainability:

The codebase should be modular, well-documented, and follow best practices in software development to facilitate easy updates, debugging, and feature enhancements.

The platform should use version control systems like Git, with clear branching strategies and code reviews to ensure code quality and manageability.

## Portability:

The application must be portable across different mobile operating systems (iOS and Android) with consistent performance and functionality.

It should also be easily deployable to different cloud environments, if needed, without significant reconfiguration.

## Compliance:

The platform must comply with relevant legal and regulatory requirements, including data protection laws like FERPA and COPPA ensuring that user data is handled responsibly.

It should also adhere to academic integrity policies and institutional guidelines where applicable.

# **External Interface Requirements**

## User Interface Requirements

* Consistent design using TailwindCSS.
* Mobile-responsive layout for both iOS and Android devices.
* Support for dark and light themes.

## Hardware Interface Requirements

Compatibility with standard smartphone features such as notifications, GPS (if needed for specific functions).

## Software Interface Requirements

* Integration with Firebase for user authentication and data storage.
* Use of OpenAI API for natural language processing and auto-completion.
* LangChain for external data integrations and context-aware responses.

## Communication Interface Requirements

* RESTful API communication between frontend and backend.
* Secure data transmission using HTTPS protocols.

# Design Considerations

## 

## Assumptions and Dependencies

By acknowledging the following assumptions and dependencies, the project team can better prepare for potential risks and challenges that may rise during the development and deployment of the Campus AI Companion:

### Assumptions

1. **User Base and Demographics**:

* The primary users of the app are computer science students from a university.
* Users have basic knowledge of mobile app usage and can navigate through standard UI components.
* Users have an active internet connection for the app to function optimally since real-time data retrieval and AI-powered recommendations require internet access.

1. **Platform Compatibility**:
   * The app will be compatible with both Android and iOS platforms, supporting at least Android 7.0 (Nougat) and iOS 12.0 and above.
   * Users have mobile devices capable of running React Native applications smoothly.
2. **Data Availability and Quality**:
   * The app assumes access to accurate and updated data from university system regarding course offerings, schedules, clubs, events, and job opportunities.
   * User-provided data, such as preferences and performance metrics, is accurate and up-to-date to ensure relevant recommendations.
3. **Performance and Load**:
   * The backend system (Node.js server and MongoDB database) is capable of handling the expected number of concurrent users without performance degradation.
   * The OpenAI API can handle multiple requests per minute without latency issues, given the rate limits and quotas defined by the OpenAI subscription plan.
4. **Security and Privacy**:
   * User data is stored securely in the backend database, and only authorized users can access their data.
   * The app assumes that users consent to share their academic performance and personal preferences for personalized recommendations.
5. **AI Model Accuracy**:
   * The recommendations provided by OpenAI are assumed to be relevant and accurate based on the training data and the input provided by users.
   * The app assumes OpenAI's LLM can understand natural language queries effectively to generate meaningful and actionable recommendations.
6. **Third-Party Service Stability**:
   * The OpenAI API and other third-party services (such as Firebase for notifications or AWS S3 for storage) are assumed to be reliable and available without significant downtime.
7. **Development and Testing Environment**:
   * The development team has access to the necessary tools, software licenses, and environments to develop, test, and deploy the application.
   * Adequate testing environments that mimic production environments are available for thorough testing of the app before release.

### Dependencies

1. **React Native Framework**:
   * The project depends on the React Native framework for building the cross-platform mobile app. Any updates or changes to the framework could affect app functionality.
2. **OpenAI API**:
   * The app relies on the OpenAI API for generating recommendations and schedule management. Changes to the API, rate limits, pricing, or availability could impact the recommendation engine functionality.
3. **Node.js and Express.js**:
   * The backend of the app depends on Node.js and Express.js. Any major updates or security vulnerabilities in these technologies could affect the backend operations.
4. **MongoDB Database**:
   * The project depends on MongoDB for data storage. The availability, scalability, and performance of the MongoDB service could impact the app's ability to store and retrieve data efficiently.
5. **Third-Party Services**:
   * **Firebase**: For push notifications and real-time updates. The app's notification features depend on Firebase’s availability and performance.
   * **AWS S3**: For storing static content. Any changes in the AWS S3 service or costs could impact storage strategies.
6. **User Authentication and Authorization**:
   * The app relies on JWT (JSON Web Tokens) for user authentication. Any vulnerabilities in JWT or changes in how it is handled by browsers and mobile devices could affect app security.
7. **Network Connectivity**:
   * The app assumes that users have a stable and reliable internet connection. Any disruptions in network connectivity can impact the app’s performance, particularly for features requiring real-time data access or AI recommendations.
8. **Regulatory Compliance**:
   * The project must comply with data privacy laws such as GDPR or CCPA. Changes in these regulations could affect how user data is collected, stored, and processed.
9. **Team Expertise**:
   * The development and maintenance of the app depend on the availability of skilled developers with experience in React Native, JavaScript, Node.js, AI integration, and mobile development best practices.

## General Constraints

By recognizing and addressing the following constraints, the development team can better plan, prioritize, and execute the project to ensure the successful delivery of the successful delivery of the Campus AI Companion app:

**Platform Constraints:**

* The app must be compatible with both iOS and Android platforms.
* The minimum supported versions will be Android 7.0 (Nougat) and iOS 12.0, which may limit some advanced functionalities available in later OS versions.

**Performance Constraints:**

* The app must load within 3 seconds on an average device under normal network conditions.
* All API responses should return within 200 milliseconds to ensure smooth user experience.
* The app should handle up to 10,000 concurrent users without significant degradation in performance.

**Security Constraints:**

* All user data, including personal information and academic performance data, must be securely transmitted over HTTPS.
* Data at rest must be encrypted using industry-standard encryption techniques (e.g., AES-256).
* User authentication must be handled securely using JWT, and tokens should have a limited lifespan (e.g., 1 hour) with refresh capabilities.

**Data Privacy and Compliance Constraints:**

* The app must comply with data protection regulations, such as GDPR for users in the EU and CCPA for users in California, regarding data collection, storage, and sharing.
* Users must have the ability to view, edit, or delete their personal data within the app.
* Explicit consent must be obtained from users before collecting any personal or sensitive data.

**Third-Party Service Constraints**

* The app’s functionality depends on third-party services like OpenAI for AI recommendations, Firebase for notifications, and AWS S3 for storage. Any downtime or changes in these services can impact the app's features.
* The app must operate within the usage limits and quotas of the OpenAI API, Firebase, and AWS S3 services.

**Network Constraints:**

* The app requires an active internet connection for real-time data retrieval and AI-based recommendations. Offline mode support is limited to local data storage for schedule management.
* The app must gracefully handle slow or intermittent network connections, providing appropriate error messages or fallback mechanisms.

**Hardware Constraints:**

* The app should be optimized to run smoothly on devices with at least 2GB of RAM and dual-core processors to ensure broad device compatibility.
* High-resolution images or media content should be optimized to minimize storage and memory usage on user devices.

**User Interface Constraints:**

* The app must adhere to platform-specific UI/UX guidelines: Material Design for Android and Human Interface Guidelines for iOS.
* All user interfaces must be accessible, providing support for screen readers and other assistive technologies to ensure compliance with accessibility standards such as WCAG 2.1.

**Development and Deployment Constraints:**

* The app must be developed using React Native, Node.js, and JavaScript, which might limit the use of certain native functionalities or require bridging to native code.
* The deployment pipeline should support continuous integration and continuous deployment (CI/CD) to streamline updates and maintenance.

**Scalability Constraints:**

* The architecture must be scalable to accommodate future growth in the user base and additional features.
* The database (MongoDB) must be designed to handle increasing volumes of data without significant impact on performance.

**Maintenance Constraints:**

* The app must be maintainable, with a clean codebase, modular architecture, and comprehensive documentation for ease of future updates and feature enhancements.
* All dependencies and libraries must be kept up-to-date to avoid security vulnerabilities and compatibility issues.

**Time and Budget Constraints:**

* The project must be completed within the allocated time frame and budget, which might limit the scope of features and the use of certain premium third-party services or tools.
* The team should adhere to a fixed timeline for each phase of development (e.g., planning, development, testing, and deployment) to meet release deadlines.

## Development Methods

The following methods are designed to ensure efficient, high-quality development, testing, and deployment of the mobile application.

### Agile Development Methodology

The "Campus AI Companion" project will adopt the **Agile** development methodology. Agile emphasizes iterative development, where requirements and solutions evolve through collaboration between cross-functional teams. This approach ensures flexibility, quick adaptation to changes, and continuous improvement throughout the project lifecycle.

* **Iterations/Sprints**: The development process will be divided into **2-week sprints**. Each sprint will focus on specific features or components of the app, from design and development to testing and deployment.
* **Sprint Planning**: At the beginning of each sprint, a planning session will be held to define the sprint goals, prioritize tasks, and allocate resources. The Product Owner, Scrum Master, and development team will collaborate to ensure alignment with project objectives.
* **Daily Standups**: Daily standup meetings will be held to discuss progress, blockers, and plans for the day. These short meetings ensure team members are aligned, and any issues are promptly addressed.
* **Sprint Review and Retrospective**: At the end of each sprint, a review meeting will be conducted to showcase the completed work. Feedback from stakeholders will be collected and discussed during the retrospective to identify areas for improvement in the next sprint.

### Test-Driven Development (TDD)

**Test-Driven Development (TDD)** will be employed to ensure code quality and reliability. TDD is a software development process where test cases are developed to specify and validate what the code will do.

* **Writing Tests First**: Before writing the actual code, developers will write automated test cases that define the desired behavior of the feature or function.
* **Code Development**: The code is written to pass the tests. Developers focus on writing just enough code to make the tests pass.
* **Refactoring**: After the code passes all tests, it is refactored to improve efficiency and readability without altering its behavior.
* **Tools Used**: Jest and Enzyme for unit testing in the frontend (React Native), Mocha, and Chai for backend testing (Node.js), and Cypress for end-to-end testing.

### Continuous Integration and Continuous Deployment (CI/CD)

To streamline the development and deployment process, the project will use **Continuous Integration and Continuous Deployment (CI/CD)** practices.

* **Continuous Integration (CI)**: Developers will frequently merge their code changes into the main branch. Automated builds and tests will run to ensure that the code is always in a deployable state.
* **Continuous Deployment (CD)**: Once the code passes all tests in the CI pipeline, it will be automatically deployed to the staging environment. After further manual or automated testing, the code will be deployed to production.
* **CI/CD Tools**: GitHub Actions or Jenkins will be used for automating the build, test, and deployment processes. These tools will ensure that the integration and deployment pipelines are efficient and reliable.

### Code Review and Pair Programming

To maintain high code quality and foster knowledge sharing among team members, **code reviews** and **pair programming** will be integral parts of the development process.

* **Code Reviews**: All code changes will go through a mandatory review process before being merged into the main branch. Peer reviews help catch potential issues early, enforce coding standards, and improve overall code quality.
* **Pair Programming**: For complex features or critical parts of the application, pair programming will be encouraged. This involves two developers working together at a single workstation, with one writing the code and the other reviewing it in real time.

### Documentation and Knowledge Sharing

Comprehensive documentation will be maintained throughout the development process to ensure clarity and facilitate knowledge sharing among team members and stakeholders.

* **Code Documentation**: Inline comments and external documentation (using tools like JSDoc for JavaScript) will be maintained to describe the purpose and functionality of the code.
* **Technical Documentation**: Detailed technical documentation, including system architecture, data models, and API specifications, will be maintained in a centralized repository like Confluence or a GitHub Wiki.
* **Knowledge Sharing Sessions**: Regular knowledge-sharing sessions or "lunch and learns" will be conducted to keep the team updated on new technologies, best practices, and learnings from the project.

### Security and Compliance

Security and compliance will be integrated into every phase of development to protect user data and ensure compliance with relevant regulations.

* **Security Reviews**: Regular security reviews and audits will be conducted to identify and mitigate potential vulnerabilities. Tools like OWASP ZAP or Snyk may be used for automated security scanning.
* **Data Protection**: All user data will be handled in compliance with GDPR, CCPA, and other relevant data protection regulations. Data encryption, secure communication protocols (HTTPS), and secure coding practices will be enforced.

### User Feedback and Usability Testing

User feedback will play a crucial role in shaping the app’s features and functionalities.

* **Usability Testing**: Periodic usability testing sessions will be conducted with a group of representative users to gather feedback on the app’s interface, features, and overall user experience.
* **User Feedback Loop**: Feedback collected from usability tests, user surveys, and in-app feedback tools will be regularly reviewed and used to inform feature development and improvements.

### Deployment and Release Management

A well-defined deployment and release management strategy will be followed to ensure smooth and controlled releases of the app.

* **Staging Environment**: All new features and updates will first be deployed to a staging environment for testing and validation.
* **Production Deployment**: After successful testing in the staging environment, changes will be deployed to the production environment.
* **Feature Toggles**: Feature toggles will be used to enable or disable specific features in production, allowing for controlled rollouts and quick rollbacks if necessary.

### Post-Release Monitoring and Maintenance

Post-release monitoring and maintenance are essential to ensure the app remains stable and functional after deployment.

* **Monitoring and Analytics**: Tools like Firebase Analytics and Sentry will be used to monitor app performance, user behavior, and error tracking. This data will help identify areas for improvement and detect issues early.
* **Bug Fixes and Updates**: A dedicated maintenance team will address any bugs or issues reported post-release, ensuring prompt resolutions and updates.
* **Continuous Improvement**: The team will continually iterate on the app, incorporating user feedback, performance data, and new technological advancements to enhance the app’s functionality and user experience.

# Architectural Strategies

## Modular Architecture

* **Component-Based Design**: The app will be designed using a modular approach where each feature (e.g., schedule management, course suggestions, event recommendations, job/career path guidance) is developed as a separate component. This will facilitate easier development, testing, and maintenance, allowing components to be independently updated or replaced without affecting other parts of the application.
* **Separation of Concerns**: By separating different concerns into distinct modules (UI, business logic, data access, etc.), the app will be more maintainable. This separation helps ensure that changes in one module do not impact others, reducing the risk of bugs and improving code readability.

## Microservices Architecture (Backend)

* **Microservices Pattern**: The backend will adopt a microservices architecture where each service (e.g., user management, course data service, recommendation engine, notification service) is developed, deployed, and scaled independently. This architecture promotes high scalability and allows different parts of the system to be developed in parallel by different teams.
* **RESTful API Design**: The microservices will communicate through well-defined RESTful APIs. This approach ensures loose coupling between services and allows the backend to be easily extendable and maintainable.

## Scalable and Flexible Data Storage

* **NoSQL Database (MongoDB)**: The use of MongoDB, a NoSQL database, allows for flexible schema design, which is ideal for evolving data models, such as user profiles and dynamic recommendations. MongoDB’s scalability features (sharding and replication) will help handle large volumes of data and high transaction loads.
* **Data Caching**: Implement caching strategies (using tools like Redis or in-memory caches) for frequently accessed data to reduce database load and improve response times. Caching will be particularly useful for static or semi-static data, such as course lists or event schedules.

## AI Integration Strategy

* **OpenAI API for AI-Driven Recommendations**: Leverage the OpenAI API to provide personalized recommendations. The integration will use RESTful API calls to send data and receive recommendations, ensuring that the AI component can be updated or replaced independently without impacting other parts of the system.
* **Edge AI Processing**: For privacy and performance reasons, consider local processing for sensitive or latency-critical AI tasks. This hybrid approach can balance the need for AI-driven functionality with concerns about privacy and data security.

## Cross-Platform Frontend Development

* **React Native Framework**: Use React Native to build a cross-platform mobile app, allowing a single codebase to be deployed on both iOS and Android. This strategy reduces development time and costs while ensuring a consistent user experience across platforms.
* **Responsive Design and Adaptive UI**: Implement a responsive design to ensure that the app looks good and functions properly on various device sizes and resolutions. Use platform-specific components when necessary to adhere to design guidelines (Material Design for Android, Human Interface Guidelines for iOS).

## Security Strategy

* **Secure Communication**: All data exchanged between the client and server will be encrypted using HTTPS to protect against man-in-the-middle attacks. JWT (JSON Web Tokens) will be used for secure user authentication and session management.
* **Data Encryption**: Sensitive user data stored in the database will be encrypted at rest using industry-standard encryption algorithms (e.g., AES-256).
* **Regular Security Audits and Penetration Testing**: Conduct regular security audits and penetration testing to identify and address potential vulnerabilities.

## CI/CD Pipeline and Automated Testing

* **Continuous Integration/Continuous Deployment (CI/CD)**: Implement a CI/CD pipeline using GitHub Actions or Jenkins to automate testing, building, and deployment processes. This will ensure that new code changes are quickly integrated and deployed with minimal manual intervention.
* **Automated Testing**: Employ automated testing strategies, including unit tests, integration tests, and end-to-end tests (using Jest, Cypress, or Detox), to ensure code quality and reduce the likelihood of introducing bugs during development.

## Native and Serverless Considerations

* **Cloud-Native Deployment**: Host the backend services on a cloud platform (e.g., AWS, Azure, Google Cloud) to take advantage of scalability, reliability, and managed services like databases and serverless computing.
* **Serverless Functions**: Consider using serverless functions (e.g., AWS Lambda, Google Cloud Functions) for certain backend operations that require scaling based on demand, such as processing recommendations or handling notifications. This reduces server management overhead and allows for better cost control.

## Performance Optimization

* **Lazy Loading and Code Splitting**: Implement lazy loading for non-critical components and code splitting techniques to reduce the initial load time of the app. This ensures a faster startup time and improves the overall user experience.
* **Asynchronous Data Loading**: Use asynchronous programming techniques (Promises, async/await) to handle data loading and avoid blocking the main thread. This ensures smooth and responsive UI interactions.

## Monitoring and Logging Strategy

* **Application Monitoring**: Use monitoring tools (e.g., New Relic, Datadog) to track application performance, identify bottlenecks, and monitor API response times. This allows for proactive maintenance and optimization of the app.
* **Centralized Logging**: Implement centralized logging (e.g., using ELK Stack - Elasticsearch, Logstash, Kibana) to collect and analyze logs from different components of the application. This facilitates easier debugging and error tracking.

## User Feedback and Analytics

* **Analytics Tools**: Integrate analytics tools (e.g., Google Analytics, Firebase Analytics) to track user behavior and app usage patterns. This data can be used to enhance user experience and prioritize new features.

# System Architecture

## Overview

The app follows a client-server architecture:

* Frontend: Developed using React Native to provide a cross-platform user interface.
* Backend: Developed using Node.js (JavaScript) to handle API request and data processing.
* Database: MongoDB for storing user data, schedules, course information, and recommendation data.
* AI Component: OpenAI API integration for generating personalized recommendations and schedules

## Component Diagram

OpenAI API

A black and grey circular object

Description automatically generated

Client

Android + iOS Mobile App

React Native

Firebase

Authentication and Authorization + File System

Backend Model using LangChain and Node.js

MongoDB Database

## Module Descriptions

* Frontend Module (React Native):
  + UI Components: Implements user interfaces for various features like schedule management, course suggestions, and recommendations.
  + Navigation: Handles app navigation using React Native.
  + State Management: Uses Redux or Context API for managing the app state.
* Backend Module (Node.js):
  + Restful API: Provides endpoints for the frontend to interact with.
  + Authentication: Manages user authentication and authorization using JWT (JSON Web Tokens).
  + Data Processing: Interacts with the database and processes user data for recommendations.
  + Integration: Communicates with the OpenAI API fetch recommendations.
* Database Module (MongoDB):
  + User Data: Stores user profiles, preferences, and performance data.
  + Course Data: Stores information about available courses.
  + Event Data: Stores information about clubs and events.
  + Recommendation Logs: Logs AI-generated recommendations for analysis and improvement.
* AI Module (Open AI):
  + Natural Language Processing: Uses OpenAI’s LLM to understand user queries and generate personalized responses.
  + Recommendation Engine: Provides recommendation and assistance based on user data and interaction history

# Detailed System Design

The following section provides a detailed discussion of the components described in the system architecture diagram for the Campus AI Companion system.

## Classification Section

The system is classified into three main layers:

### Presentation Layer (Frontend):

* + User Interface (UI) components built with React Native for cross-platform compatibility (iOS and Android).
  + Responsible for rendering the user interface, handling user interactions, and making API requests to the backend services.

### Application Layer (Backend):

* + Node.js with Express.js serves as the backend framework.
  + Handles business logic, processes API requests, manages sessions, and interacts with the database and third-party services like OpenAI.
  + LangChain which will manage the orchestration providing internal data and external data to LLM for contextual recommendations.
  + LangChain enables the integration and management of multiple AI models and tools, ensuring smooth and coherent responses. It is particularly suited for handling complex queries by enabling the backend to leverage OpenAI’s capabilities more effectively. For instance, LangChain can provide OpenAI with external resources such as PDF documents, allowing the model to deliver more informed and contextually relevant answers.

### Data Layer:

* + MongoDB is used as the primary database to store user data, schedules, course information, events, and job recommendations.
  + Includes data caching mechanisms using Redis to improve performance for frequently accessed data.

## Definition Section

The "Campus AI Companion" system is composed of multiple subsystems and components that interact to deliver a seamless user experience. Below is a detailed breakdown of the primary subsystems:

### Frontend Subsystem:

* **React Native Application:** A cross-platform mobile app that serves as the primary user interface, providing a responsive and intuitive user experience. The home screen is a chatbot interface that allows users to interact with the AI for queries and recommendations.
  + **Chatbot Component:** The central interaction point for users to communicate with the AI. It sends user queries to the backend and displays AI-generated responses. It provides functionalities such as voice commands and chat-based interactions
  + **Schedule Management Component:** Allows users to view, add, and manage their class schedules, assignments, and events.
  + **Course Recommendation Component:** Displays personalized course suggestions based on user preferences and academic history.
  + **Club and Event Recommendation Component: Provides** suggestions on clubs to join and events to attend based on user interests and schedule availability.
  + **Job and Career Path Recommendation Component**: Shows job opportunities and career advice tailored to the user's academic performance and interests.
  + **User Profile Component:** Manages user information, preferences, and settings

### Backend Subsystem:

* **Node.js Server:** Handles all server-side logic, including user authentication, data processing, and communication with external services.
  + **Authentication and Authorization Module:** Manages user login, registration, and access control using JWT for secure sessions.
  + **API Gateway:** Acts as the central point for all API requests, routing them to the appropriate services (e.g., user data, course recommendations, etc.).
  + **Recommendation Engine Module:** Integrates with OpenAI API to generate personalized course, club, event, and career recommendations.
  + **Data Processing Module:** Handles data manipulation and processing tasks, such as filtering courses or events based on user preferences.

### Database Subsystem:

* **MongoDB Database:** Stores all persistent data, including user profiles, schedules, preferences, AI-generated recommendations, and interaction logs.
  + **User Collection:** Stores user-specific data, such as personal information, preferences, and academic history.
  + **Schedule Collection:** Maintains user schedules, including classes, assignments, and events.
  + **Recommendation Collection:** Saves AI-generated recommendations for courses, clubs, events, and careers.
  + **Interaction Logs Collection:** Logs all user interactions with the chatbot for analysis and improvement of the AI model.

### External Services Subsystem:

* **OpenAI API**: Provides AI-driven recommendations based on user input and historical data.
* **Firebase**: Used for push notifications and real-time updates, as well as for analytics to track user behavior and app performance.
* **Admob**: Handles advertisements within the app.
* **Firestore** (optional): May be used for real-time data synchronization.

## Constraints

### Performance Constraints:

* + API response times should be under 200 milliseconds for optimal user experience.
  + The system should handle up to 10,000 concurrent users without significant performance degradation.

### Security Constraints:

* + All data must be transmitted over HTTPS to prevent data interception.
  + User data stored in MongoDB must be encrypted at rest using AES-256 encryption.
  + JWT tokens used for authentication should have a short lifespan and require periodic refreshes.

### Scalability Constraints:

* + The system must support scaling out horizontally to accommodate a growing number of users and increased data volumes.

### Compliance Constraints:

* + Must comply with data protection regulations such as GDPR and CCPA.
  + User consent is required for collecting and processing personal and sensitive data.

## 

## Resources

### Hardware:

* + Cloud servers (e.g., AWS EC2 instances) for hosting the backend services.
  + Managed MongoDB clusters (e.g., MongoDB Atlas) for database services.
  + Redis instances for caching services.

### Software:

* + **Frontend**: React Native, Redux/Context API, Axios (for HTTP requests), Firebase SDK.
  + **Backend**: Node.js, Express.js, Mongoose (MongoDB ODM), OpenAI SDK, Redis, JWT.
  + **Database**: MongoDB for primary data storage, Redis for caching.
  + **CI/CD Tools**: GitHub Actions or Jenkins for continuous integration and deployment.

### Third-Party Services:

* + **OpenAI API**: For AI-driven recommendations.
  + **Firebase Cloud Messaging**: For push notifications.
  + **AWS S3 or Google Cloud Storage**: For storing static content or user-uploaded data.

## Interface/Exports

### Frontend Interfaces

* **User Actions**:
  + **Login/Signup**: User authentication interface via a login/signup form, integrating JWT-based authentication.
  + **Chatbot**: Interface to communicate with the AI.
  + **Schedule Management**: Interface to add, edit, view, or delete schedule entries.
  + **Course/Event/Job Recommendations**: Interface to display personalized recommendations, allowing users to interact with suggested items.
* **API Endpoints** (Interacts with backend, \*\* might be updated\*\*):
  + **GET /api/user/profile**: Fetch user profile data.
  + **POST /api/user/update**: Update user profile settings.
  + **GET /api/schedule**: Retrieve user schedule data.
  + **POST /api/schedule/add**: Add a new schedule entry.
  + **GET /api/recommendations/courses**: Fetch personalized course recommendations.
  + **GET /api/recommendations/events**: Fetch personalized event recommendations.
  + **GET /api/recommendations/jobs**: Fetch personalized job recommendations.

### Backend Interfaces

* **Internal Services Communication**:
  + Microservices within the backend communicate using RESTful API calls and possibly asynchronous message queues (e.g., RabbitMQ or AWS SQS) for decoupled services.
* **Database Interaction**:
  + Uses Mongoose for MongoDB interactions, defining schemas and models for structured data access and manipulation.
* **External APIs**:
  + **OpenAI API**: To fetch recommendations based on user input.
  + **Firebase Cloud Messaging API**: For sending push notifications to users.
* **Exports**:
  + Provides RESTful API endpoints that the frontend can consume.
  + Sends data or notifications to the frontend or third-party services.

### Data Exports

* **To Frontend**:
  + JSON formatted responses for all RESTful API requests.
  + Real-time data updates via WebSockets or Firebase for certain features (e.g., notifications, real-time schedule changes).
* **To Third-Party Services**:
  + Sends data to OpenAI for processing and receives AI-generated recommendations.
  + Pushes notifications to Firebase for user alerts.

# Architecture Drawings

A diagram of a process

Description automatically generated

Fig: System Architecture of the Campus AI Companion

The following architecture outlines the components and data flow of the Campus AI Companion, a mobile application designed to assist students by providing them with relevant, contextual information powered by AI. This system integrates the OpenAI GPT-3.5 model with external databases and Firebase services to ensure a reliable, secure, and responsive user experience. Below is a detailed explanation of each component and how they interact within the system.

## 1. Scraped Data Source

To ensure that the Campus AI Companion has access to up-to-date and relevant information, data is regularly collected from various sources, such as official campus websites, learning resources, and event calendars. This scraped data forms the knowledge base, which the model can access and utilize when responding to user queries.

## 2. Pinecone (Vector Database)

The scraped data is transformed into vector embeddings—a mathematical representation that allows for efficient similarity searches—and stored in Pinecone, a high-performance vector database. Storing data as embeddings enables the system to retrieve information relevant to user queries quickly and accurately.

- Data Feed: Processed embeddings are continually fed into Pinecone, keeping the database updated with the latest available information.

## 3. GPT-3.5 Model

At the core of the Campus AI Companion is the GPT-3.5 model, which powers the AI’s natural language understanding and generation capabilities. The model is configured to use RAG (Retrieval-Augmented Generation), a technique that combines external data retrieval with response generation. Here’s how it works:

- Upon receiving a user query, the GPT-3.5 model first searches the Pinecone database for relevant information.

- This retrieved information is then used to enhance the response generation, ensuring answers are contextually accurate and specific to the query.

- Prompt Templates: To further refine its responses, the model uses predefined prompt templates. These templates guide the AI to produce responses that align with the tone and intent of the Campus AI Companion, ensuring consistency across various types of queries.

## 4. Firebase Services

The Firebase platform is integrated into the Campus AI Companion to handle user management and data storage, creating a secure and seamless experience for students. Two key Firebase services are utilized:

- Firebase Authentication: This component manages user authentication, ensuring secure login and access control within the app. Students must log in via Firebase Auth to interact with the AI, which allows for personalized responses and protects sensitive data.

- Firebase Storage: Firebase Storage is responsible for securely storing user-uploaded files and other relevant resources. The Campus AI Companion can access this data as needed, enhancing its ability to support students with personalized resources or documents.

## 5. Mobile Client

The Mobile Client is the front-end application interface that students interact with on their devices. It is designed to provide a user-friendly, intuitive interface for accessing the Campus AI Companion’s capabilities. The Mobile Client:

- Connects to the GPT-3.5 model to facilitate real-time question answering and information retrieval.

- Utilizes Firebase Authentication to manage user login and secure access.

- Communicates with Firebase Storage to retrieve or store files as necessary.

## Data Flow and Interaction Summary

1. Data Preparation: The scraped data is converted to embeddings and stored in the Pinecone vector database, ensuring the AI has access to relevant, searchable information.

2. User Query Processing: When a user submits a query via the Mobile Client, the GPT-3.5 model retrieves relevant data from Pinecone (using the RAG technique) and applies prompt templates for response customization.

3. Authentication and Storage: The Mobile Client leverages Firebase services for secure authentication and storage, ensuring a secure and personalized experience for each user.

4. Response Delivery: The processed response is then delivered back to the Mobile Client, providing the user with a contextually relevant answer or assistance.

# Version Control Discussion

Effective version control is essential for collaborative software development, and the team adopted GitHub to manage the "Campus AI Companion" project. GitHub facilitated seamless integration of contributions, efficient issue tracking, and comprehensive project documentation. The following practices were employed to maintain a robust version control process:

* Branch Management: Each team member worked on feature-specific branches, ensuring that the main branch remained stable. Feature branches were merged into the main branch only after thorough testing and approval.
* Commit Regularity: Regular commits were encouraged to ensure a detailed history of changes and facilitate easier debugging. Commit messages followed a structured format to describe changes clearly.
* Pull Requests and Code Reviews: Every pull request underwent peer review to identify potential bugs, ensure adherence to coding standards, and maintain code quality.
* Conflict Resolution: Merging branches was managed carefully to minimize conflicts. In cases where conflicts arose, the team utilized Git tools to resolve issues collaboratively.
* Automated CI/CD Integration: The repository was integrated with CI/CD pipelines to automatically test new code additions, ensuring that the application remained functional and stable after updates.

GitHub’s version control system played a vital role in fostering collaboration, improving code quality, and expediting the development process.

# Challenges and Assumptions

For the project, we are using the OpenAI API free tier. The free tier offers a limited yet functional version of the OpenAI API, which helps us leverage AI capabilities without significant costs. However, this comes with several constraints, which we outline below:

## Limitations of the OpenAI API Free Tier:

### Limited Access to the Most Updated Data:

Since the free tier often does not include real-time data, it restricts access to the latest models and information, resulting in a potential lag in the knowledge cutoff. This may affect the relevancy of the responses, especially if the query involves recent events or newly established facts.

### Context Window Constraints:

The free tier limits the size of the context window (the amount of text the model can process at once). When this window limit is reached, the model may omit parts of the input or the conversation’s prior context, which can impact the coherence and quality of responses for longer conversations or complex prompts.

### Limited Prompt Comprehension:

Certain complex or highly nuanced prompts may not be fully understood by the model due to these limitations, potentially impacting the depth and accuracy of responses. This is especially relevant for prompts that require fine-tuned comprehension or involve highly specialized knowledge.

## Matching Users’ Prompts to the Right Agent or Tool:

To optimize responses and improve user experience, we employ several methods to match prompts with the most suitable tools or agents. These methods include:

### Keyword Matching:

This approach identifies specific words or phrases in the prompt and routes the query based on predefined keywords. For instance, if a prompt includes terms like “summarize” or “translate,” it can be directed to agents specialized in these tasks. While simple, this approach is effective for clear-cut, task-specific queries.

### Classifier-Based Approach:

A classifier model categorizes prompts based on intent or subject matter, enabling more accurate matching. By using training data, the classifier can distinguish between types of requests (e.g., information retrieval vs. creative writing), ensuring each query is directed to the agent most likely to produce a relevant response.

### Vector Similarity Matching:

Using vector embeddings, this method measures the semantic similarity between a prompt and a set of predefined categories or sample prompts. This approach can capture nuanced meanings, matching prompts with similar underlying intents even if the specific words differ. Vector similarity matching is particularly effective for ambiguous or complex queries where simple keyword matching might fall short.

# Test Plans

# Test Items

The major components to be tested include:

1. Login and Account Management

2. Authentication

3. Recommendation/Suggestion Management

4. Schedule Management

5. Non-functional requirements like Usability, Scalability, Security, Maintainability, Portability, Compliance

6. External Interface Requirements (UI, hardware, software, and communication interfaces)

# Testing Approach

We conducted the following types of testing:

1. Functional Testing - Verify that each feature works according to the specified requirements.

2. Non-Functional Testing - Test usability, security, scalability, and compliance.

3. Interface Testing - Ensure all user interface components and API interactions function as intended.

Each feature was validated with a combination of white box, black box, acceptance, and performance testing to ensure comprehensive coverage:

* **White Box Testing**: Primarily focused on backend components for authentication and session management.
* **Black Box Testing**: Applied to user interface components, ensuring feature functionality from the end-user’s perspective.
* **Acceptance Testing**: Validated end-to-end workflows, ensuring the platform met all essential requirements.
* **Performance Testing**: Conducted load and stress tests to validate scalability and responsiveness under high demand.

# Entry and Exit Criteria

- Entry Criteria: Code is deployed in a staging environment and passes all unit tests.

- Exit Criteria: All critical test cases pass with no high-severity defects.

# Test Environment

- Mobile Platforms: iOS and Android

- Frameworks: React Native (frontend), Firebase, OpenAI API, LangChain (backend)

- Tools: Firebase Authentication, MongoDB, and TailwindCSS for UI design consistency.

# Test Cases

## 1. Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Test Case** | **Expected Result** |
| Login and Account Management |  |  |
| Create Account | Register with valid information | Account created successfully |
| Login | Login with registered credentials | User logged in and session started |
| Password Recovery | Request password recovery email | Password recovery email received |
| Authentication |  |  |
| User Authentication | Authenticate user with Firebase | Valid user session established |
| Session Management | Verify secure token handling for sessions | User remains logged in during session |
| Recommendation/Suggestion Management |  |  |
| Provide personalized guidance | Input prompt to LLM and receive recommendations | Relevant suggestions provided |
| Provide academic resources | Access recommended resources | Resources relevant to prompt displayed |
| Schedule Management |  |  |
| Generate timetable | Enter daily activities and receive schedule | Timetable generated as expected |
| Schedule reminders | Receive reminders for scheduled tasks | Notification sent at appropriate time |

## 2. Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Test Case** | **Expected Result** |
| Usability | Verify UI elements are intuitive and user-friendly | Users can navigate without confusion |
| Scalability | Test app response under high user load | App maintains performance without crashes |
| Security | Test secure login/logout and data handling | Data remains secure; only authorized access allowed |
| Maintainability | Code review and documentation inspection | Code is modular, documented, and follows best practices |
| Portability | Test app on both iOS and Android platforms | Consistent functionality across devices |
| Compliance | Verify data protection, FERPA, and COPPA compliance | All legal requirements are met |

# Software Test Report (STR) for Campus AI Companion

## Test Summary

All major functional and non-functional requirements were tested across iOS and Android platforms. Key areas such as login functionality, authentication, schedule management, and recommendations were validated.

## Detailed Test Results

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Pass** | **Fail** | **Severity** |
| Create Account | Yes | No | High |
| Login | Yes | No | High |
| Password Recovery | Yes | No | Medium |
| **Authentication** |  |  |  |
| User Authentication | Yes | No | High |
| Session Management | Yes | No | High |
| **Recommendation/Suggestion Management** |  |  |  |
| Provide personalized guidance | Yes | No | Medium |
| Provide academic resources | Yes | No | Medium |
| Schedule Management | Yes | No | Medium |
| Generate timetable | Yes | No | Medium |
| Schedule reminders | Yes | No | Medium |
| **Non-Functional Requirements** |  |  |  |
| Usability (UI/UX navigation) | Yes | No | Medium |
| Scalability under load | Yes | No |  |
| Security (secure login, data handling) | Yes | No | High |
| Maintainability (code quality, modularity) | Yes | No | Medium |
| Portability (cross-platform consistency) | Yes | No | Medium |
| Compliance with data protection laws | Yes | No | High |

# **Conclusion**

The "Campus AI Companion" project exemplifies the potential of AI-driven solutions in addressing the challenges faced by university students. By leveraging modern technologies such as the OpenAI API, Firebase, and React Native, the team developed a robust, scalable, and user-friendly platform tailored to enhance students' academic and professional journeys.

Throughout the project lifecycle, adherence to Agile methodologies ensured adaptability and responsiveness to user needs. Rigorous testing practices and continuous feedback loops further enhanced the application's reliability and usability. Effective version control and architectural strategies allowed the team to manage a complex system efficiently, setting a strong foundation for future iterations.

While the project encountered limitations, particularly with the OpenAI free tier, innovative approaches such as vector similarity matching and classifier-based routing mitigated these constraints. Moving forward, expanding the platform to include more disciplines and integrating advanced AI capabilities will unlock further opportunities for enhancing user experience.

The "Campus AI Companion" not only reflects the team's technical expertise but also their commitment to delivering a meaningful, impactful tool for student success.

# **Appendix**

Appendix A - Glossary

1. **API (Application Programming Interface)**  
   A set of rules and definitions that allow different software applications to communicate with each other. In this project, RESTful APIs are used for communication between the frontend and backend services.
2. **Authentication**  
   The process of verifying the identity of a user or system. In the app, authentication is managed using JWT (JSON Web Tokens) to ensure that only authorized users can access certain features.
3. **Authorization**  
   The process of granting or denying access to specific resources or actions within an application, based on the user's identity and permissions. It determines what a user can or cannot do after being authenticated.
4. **Backend**  
   The server-side component of the application that handles business logic, database interactions, and external API calls. It includes the server (Node.js with Express.js), the database (MongoDB), and various services (e.g., user management, recommendation engine).
5. **Frontend**  
   The client-side component of the application that users interact with. It is built using React Native, allowing it to run on both iOS and Android platforms. The frontend handles UI rendering, user input, and communicates with the backend via APIs.
6. **MongoDB**  
   A NoSQL database used for storing and managing data in a flexible, JSON-like format. It is the primary database for this project, chosen for its scalability and ease of handling dynamic data.
7. **OpenAI API**  
   An API provided by OpenAI that allows developers to integrate AI models into their applications. In this project, the OpenAI API is used for generating personalized recommendations based on user data.
8. **React Native**  
   A JavaScript framework for building mobile applications using React. It allows developers to build apps for both iOS and Android using a single codebase, ensuring consistency and reducing development time and cost.

Appendix B - References

1. **MongoDB Documentation**  
   MongoDB, Inc. "MongoDB Manual." [MongoDB Docs](https://www.mongodb.com/docs/)
2. **OpenAI API Documentation**  
   OpenAI. "OpenAI API." [OpenAI Docs](https://platform.openai.com/docs/)
3. **Node.js and Express.js Documentation**  
   OpenJS Foundation. "Node.js Documentation." Node.js Docs.
4. **React Native Documentation**  
   Facebook, Inc. "React Native – A framework for building native apps using React." React Native Docs.