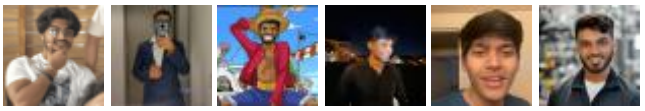
CAMPUS AGILE



- Dharmik Patel (Scrum Master & AI Engineer)
- Deep Patel (Front-end Lead)
- Dravya & Darshit Patel (Front-end Developers)
- Dev & Love Patel (Back-end Developers)

❖ Installation and Setup

The following instructions are inteded to be ran in a Linux/Unix Environment

▶ **Prerequisites**
▶ **Clone the Repository**

The next 2 instructions should be ran in 2 different termimal sessions. They need to be running concurrently.

▶ **Running the api**
▶ **Running the client**
▶ **Viewing the app**

❖ Information

CampusAgile is a revolutionary platform dedicated to equipping university students with invaluable real-world project management experience. Our mission is to empower the next generation of professionals by providing them with the tools and knowledge needed to excel in an ever-evolving job market. With CampusAgile, students gain access to a collaborative environment that seamlessly integrates agile methodologies, enabling them to break down complex projects, develop adaptability, master time management, and prioritize tasks effectively.

Key Features:

- **Agile Task Management:** CampusAgile introduces agile methodologies to students, teaching them how to dissect intricate projects into manageable components. This not only fosters project adaptability but also hones critical skills in time management and task prioritization.
- **Real-World Simulations:** Our platform offers a secure space where students can engage in real-life scenarios, allowing them to practice decision-making and tackle project challenges head-on. Through these simulations, students build confidence and acquire essential management skills that prepare them for the professional world.
- **Industry Integration:** CampusAgile collaborates with various organizations, bridging the gap between academic learning and industry requirements. By working on real projects sourced from industry partners, students not only enhance their networking opportunities but also increase their employability by gaining hands-on experience in their chosen field.

Major Features (MVP):

- **Task Assignment and Tracking:** Easily assign and monitor tasks, facilitating efficient project collaboration.
- **Interactive Agile Boards:** Visualize project progress through interactive agile boards, enhancing transparency.
- **Resource Management:** Optimize resource allocation and track utilization to ensure project efficiency.
- **Performance Analytics:** Access comprehensive analytics to evaluate individual and team performance, driving continuous improvement. **Stretch Goals:**
- **Mobile Application:** Develop a mobile app for on-the-go access, providing convenience and flexibility.
- **Integration with Popular Tools:** Integrate CampusAgile with widely-used productivity and collaboration tools to enhance workflow.

CampusAgile aims to be the premier platform for students seeking to bridge the gap between academia and industry, offering an immersive learning experience that prepares them for successful careers.

❖ Toolchain and Structure

▶ **Version Control**
▶ **Bug Tracking**
▶ **Build System**
▶ **Testing and CI**

❖ Team Policies

Communication and Tools:

- **Communication Platform:** We utilize Discord for real-time discussions, team meetings, and weekly updates. Discord enables us to maintain effective communication and quick responses among team members.
- **Version Control:** GitHub is our chosen platform for hosting the project's Git repository. This facilitates code versioning, collaborative development, and code reviews.
- **Issue and Task Tracking:** Jira Software is our primary tool for tracking project-related issues, tasks, and milestones. Jira helps us organize and manage our project workflow efficiently.

Policies:

- We expect all team members to actively participate in Discord discussions and weekly meetings to ensure effective communication.
- Git push requests to the repository are reviewed and approved by team leads (Deep Patel for front-end and Dharmik Patel for back-end) to maintain code quality and consistency.
- Task assignments and updates are tracked using Jira Software, ensuring transparency and accountability in project progress.

❖ Use Cases

Use Case 1: User Task Assignment

- **Actors:** User (Student or Faculty), System
- **Triggers:** User initiates the task assignment process.
- **Preconditions:** User must be logged in.

- **Postconditions (success scenario):**
  - User assigns tasks to team members successfully.
  - Task details and assignments are updated in the system.
- **List of Steps (success scenario):**
  - User logs in.
  - User navigates to the project.
  - User selects the "Assign Tasks" option.
  - User specifies task details (title, description, priority, due date, assignee).
  - User submits the task assignment.
- **Extensions/variations of the success scenario:**
  - If the task assignment fails, the user receives an error message and can retry.
- **Exceptions (failure conditions and scenarios):**
  - If the user is not logged in, they are prompted to log in before proceeding.

Use Case 2: User Collaboration on a Task

- **Actors:** User (Student or Faculty), System
- **Triggers:** User selects a task to collaborate on.
- **Preconditions:** The User must be part of the project and the task must exist.
- **Postconditions (success scenario):**
  - User successfully collaborates on the task.
  - Task updates are reflected in real-time.
- **List of Steps (success scenario):**
  - User logs in.
  - User selects the project.
  - User navigates to the task.
  - User views task details and discussions.
  - Users add comments, attach files, or update task status.
- **Extensions/variations of the success scenario:**
- Users can mention other team members for quick collaboration.
- **Exceptions (failure conditions and scenarios):**
  - If the task does not exist or the user is not part of the project, they receive an error

message.

Use Case 3: Managing Agile Tasks

- **Actors:** User (Student or Faculty), System
- **Triggers:** Student or Faculty wants to create, assign, or update agile tasks.
- **Preconditions:** The User must be part of the project and the task must exist.
- **Postconditions (success scenario):**
  - Agile tasks are successfully created, assigned, or updated within a course.
- **List of Steps (success scenario):**
  - Users select the course in which they want to manage agile tasks.
  - User navigates to the "Agile Task Management" section.
  - Users create, assign, or update agile tasks as needed.
  - The system confirms the changes.
- **Extensions/variations of the success scenario:**
- If a task deadline is approaching, the system can send reminders to users.
- Instructors can review and approve tasks submitted by students.
- **Exceptions (failure conditions and scenarios):**
  - If the user is not logged in, they are prompted to log in before proceeding.

Use Case 4: Managing Project Progress

- **Actors:** User (Student or Faculty), System
- **Triggers:** Student or Faculty wants to monitor the progress of a project.
- **Preconditions:**
- The user must be part of the project and the task must exist.
- Users must have the role of faculty because only faculty(admins) can monitor the overall progress.
- **Postconditions (success scenario):**
  - Users can view project progress and make informed decisions.
- **List of Steps (success scenario):**
  - User selects the project they want to monitor.
  - User navigates to the "Project Progress" section.
  - Users review the project's agile board, resource allocation, and performance analytics.
  - Users gain insights into the project's status.
- **Extensions/variations of the success scenario:**
- If a project falls behind schedule, the system can highlight critical tasks.
- Faculty can provide feedback to students based on project progress.

- **Exceptions (failure conditions and scenarios):**
    - Users are not part of a project.
    - If the user is not logged in, they are prompted to log in before proceeding.

Use Case 5: Task Progress Tracking

- **Actors:** User (Student or Faculty), System
- **Triggers:** Need to monitor and assess task progress.
- **Preconditions:**
- Users (students and instructors) have registered accounts on CampusAgile.
- Users must be assigned tasks that they can work on.
- **Postconditions (success scenario):**
    - Users can monitor the progress of individual tasks within a project.
    - Users can add comments and such for their own benefit.
    - Instructors can assess student contributions and provide feedback.
- **List of Steps (success scenario):**
    - Instructor creates a project with defined tasks.
    - Students are assigned specific tasks.
    - Students update the status of tasks as "in progress", "awaiting approval", "completed."
    - Instructors review task progress and provide feedback.
- **Extensions/variations of the success scenario:**
- Task tracking will highlight task priority ranging from 1-4(critical to low). It can also tell if it is in progress, delayed, or finished.
- **Exceptions (failure conditions and scenarios):**
    - If the task does not exist or the user is not part of the project, they receive an error message.

Use Case 6: Task Dependencies

- **Actors:** User (Student or Faculty), System
- **Triggers:** There can be project tasks that depend on the completion of others.
- **Preconditions:**
- Users (students and instructors) have registered accounts on CampusAgile. Projects with task dependencies are available.
- Users must be part of projects and have tasks assigned to them.
- **Postconditions (success scenario):**

1. Users can define task dependencies to ensure proper project sequencing.
2. The platform helps manage task orders and notifies users of dependent tasks.

- **List of Steps (success scenario):**

1. Users create a project with interdependent tasks.
2. They specify task dependencies within the project.

3. The platform enforces task orders and provides notifications.

- **Extensions/variations of the success scenario:**

1. Allow users to connect other types of tasks to their projects such as Request Items, Ad hoc requests, Software requests, Hardware requests, etc.

- **Exceptions (failure conditions and scenarios):**

1. If the task does not exist or the user is not part of the project, they receive an error message.

❖ Other Requirements

Non-functional Requirements:

- **Usability:** The platform should have an intuitive user interface to ensure users can easily navigate, collaborate, and manage tasks without extensive training.
- **Security:** User data, project information, and communication should be encrypted to protect sensitive information. User authentication and authorization mechanisms must be robust.
- **Scalability:** The system should handle an increasing number of users, tasks, and projects without significant performance degradation.

External Requirements:

- **Robustness:** The platform should gracefully handle errors such as invalid user input, preventing system crashes or data corruption.
- **Installability:** CampusAgile should be accessible via a public URL for web-based access, and installation instructions should be provided for standalone applications.
- **Buildability:** All components, including clients and servers, should be buildable from source code. Comprehensive documentation should be available for developers to contribute and enhance the system.
- **Scope Alignment:** The scope of the project should align with the available team resources, ensuring that tasks and milestones are manageable within the team's capacity.

7 **Semester-Long Development Process:** Software Toolset: For our semester-long development process, we have carefully selected a set of software tools to facilitate efficient and effective development. These tools align with agile methodologies and provide the necessary support for our project.

1. Front-end Development:
    - React.js: We will use React.js for building our front-end user interface due to its component-based architecture and high reusability.
    - TypeScript: TypeScript will be employed to add type safety to our JavaScript code, reducing errors and enhancing maintainability.
    - HTML/CSS: Standard HTML and CSS will be used for structuring and styling our web pages.
    - GitHub: We will utilize GitHub for version control, enabling seamless collaboration and code management.
    - Jira: Jira will serve as our task-tracking tool to manage project tasks, issues, and sprints.
    - Discord: Discord will be our primary communication platform for team discussions and coordination.
2. Back-end Development:
    - Node.js: We have chosen Node.js for our back-end development, as it aligns well with JavaScript used in React, offering a unified development experience.

    - Database Management: Back-end developers will handle database management, selecting an appropriate database system based on project requirements.
    - Server Deployment: Deployment of the back-end server will be managed by the back-end lead, ensuring smooth operation.

Team Member Roles:

- Deep Patel (Front-end Lead): Deep will be responsible for leading the front-end development efforts. His role includes defining the front-end architecture, establishing a cohesive design and color palette, and assigning tasks to front-end developers. Deep's experience in front-end development makes him well-suited for this role.
- Dravya & Darshit Patel (Front-end Developers): Dravya and Darshit will work closely with the front-end lead to implement front-end components based on the provided design and requirements. Their responsibilities include coding, testing, and ensuring a seamless user experience.
- Dharmik Patel (Back-end Lead): Dharmik will take charge of the back-end development, overseeing the server-side architecture, database management, and server deployment. His experience in back-end development and NLP makes him the ideal candidate for this role.
- Dev & Love Patel (Back-end Developers): Dev and Love will collaborate with the back-end lead to develop server-side functionalities and integrate the database. They will work on implementing the logic necessary for the platform's core functionality.

Schedule and Milestones: Our development process is structured around key milestones to ensure progress throughout the semester:

1. Project Initiation (Week 1-2):
    - Define project scope, goals, and requirements.
    - Set up the development environment.
    - Establish communication channels.
    - Create a project plan and assign roles.
2. MVP Development (Week 3-6):
    - Develop a Minimum Viable Product (MVP) with core features.
    - Conduct initial testing and debugging.
    - Prepare for internal testing and review.
3. Feature Additions (Week 7-10):
    - Add additional features and functionalities.
    - Conduct thorough testing of new features.
    - Refine user interface based on feedback.
4. Testing and Refinement (Week 11-14):
    - Conduct comprehensive testing, including user acceptance testing.
    - Address any identified issues and bugs.
    - Optimize performance and security.
    - Finalize documentation and user guides.

Major Risks:

1. Technical Challenges: Complex technical issues or limitations in the chosen technologies could impede progress. We will address this by maintaining open communication within the team and seeking external expertise if needed.
2. Scope Creep: Expanding project scope beyond the defined requirements can lead to delays. We will closely monitor project scope and prioritize features based on their impact and feasibility.

3. Resource Limitations: Limited availability of team members or access to necessary resources may affect project timelines. We will mitigate this by allocating tasks effectively and having contingency plans in place.

External Feedback: External feedback will be most valuable during the MVP development and testing phases. We plan to seek feedback from professors, industry experts, and potential end-users to refine our platform. This feedback will help us identify usability issues, performance bottlenecks, and areas for improvement before the final release. We will gather feedback through surveys, user testing sessions, and expert reviews, incorporating valuable insights into our development process.

❖ Software Architecture

Part 1: Overview of CampusAgile Software Architecture:

CampusAgile is designed to provide students with a collaborative platform to gain real-world experience with agile methodologies, using Kanban boards and Scrum notes for organization. The software aims to facilitate teamwork, project management, and learning through interactive features. The major software components and their functionality are as follows:

- **Frontend**: Developed using HTML5, CSS, React, Typescript, and Chakra UI, the frontend is responsible for the user interface. It provides the user with the ability to access and interact with the various features of CampusAgile. Functionality includes user authentication, displaying Kanban boards, creating and managing Scrum notes, and interacting with the AI chatbot.
- **Backend**: Built using Nest.js, Express.js, and Typescript, the backend serves as the core of the application. It handles user authentication, manages project data, and communicates with the frontend. Key responsibilities include routing requests from the frontend, managing user data, storing and retrieving project-related information, and interacting with external services (if necessary).
- **Kanban Board Component**: This component allows students to create, manage, and collaborate on Kanban boards for their projects. It enables drag-and-drop functionality for cards, tracks the progress of tasks, and provides a visual representation of project status.
- **Scrum Notes Component**: Students can create, edit, and organize Scrum notes within this component. It provides a structured way to plan, execute, and review project tasks according to Scrum principles.
- **AI Chatbot (Optional)**: The AI chatbot, if implemented, would automate manual tasks and provide assistance to students. It could answer questions, generate reports, and help with project management. The AI component would use natural language processing (NLP) and machine learning algorithms to provide intelligent responses.

Interfaces Between Components:

- **Frontend to Backend**: The frontend communicates with the backend through API endpoints. This includes user authentication, project creation, updating Kanban boards, and managing Scrum notes.
- **Kanban Board Component to Backend**: The Kanban board component communicates with the backend to retrieve and update project data, such as task status and card placement.
- **Scrum Notes Component to Backend**: The Scrum notes component interacts with the backend to store and retrieve Scrum-related data, including sprint planning and task tracking.
- **AI Chatbot to Backend (Optional)**: If the AI chatbot is implemented, it communicates with the backend to access project data, answer user queries, and perform automated tasks.

Data Storage and Organization:

The data in CampusAgile is organized as follows:

- **User Data**: User profiles, authentication information, and roles are stored securely in a database.
- **Project Data**: Information about projects, including Kanban boards, Scrum notes, and associated files, is stored in a structured format in the database.
- **Kanban Cards and Task Data**: Individual task data within Kanban boards is stored in a way that allows tracking of task status and card placement.
- **Scrum Notes and Sprint Data**: Scrum notes and sprint-related data are organized to facilitate sprint planning, execution, and review.
- **Chatbot Interaction Data (Optional)**: If the AI chatbot is implemented, data related to user interactions and chatbot responses may be stored for analysis and improvement.

Assumptions:

- **Internet Connectivity**: It is assumed that users have consistent internet access to use CampusAgile as a web-based platform.
- **Security and Privacy**: Security measures are in place to protect user data and ensure privacy compliance with relevant regulations. **Alternative Software Architecture Decisions:**

Alternative 1 - Monolithic Architecture vs. Microservices:

- *Monolithic Architecture*: A single, integrated application with all features and components.
  - Pros: Simplicity in development and deployment.
  - Cons: Scalability and maintenance challenges as the system grows.
- *Microservices Architecture*: Breaking down the system into small, independent services.
  - Pros: Scalability, flexibility, and easier maintenance of individual components.
  - Cons: Increased complexity in managing multiple services.

Alternative 2 - AI Chatbot Integration vs. No AI (Continued):

- *AI Chatbot Integration*: Implementing an AI chatbot to automate tasks.
  - Pros: Enhanced user experience, reduced manual work, and potential for intelligent assistance.
  - Cons: Development complexity, potential for errors in AI responses, and additional resource requirements.
- *No AI Chatbot*: Omitting the AI chatbot and relying solely on manual user interactions.
  - Pros: Simplified development, lower resource requirements.
  - Cons: Increased manual work for users, potentially reduced user engagement.

The choice between these alternatives depends on the team's capacity, user requirements, and the overall project scope. Implementing an AI chatbot can significantly enhance the user experience, but it requires additional development effort and maintenance. On the other hand, not implementing the AI chatbot may simplify development and reduce resource requirements but could result in a less automated and potentially less engaging user experience.

❖ Software Design

1. **Frontend Component**:
   - **Packages/Classes/Units of Abstraction**:
     - `AuthenticationService` : Responsible for user login and registration.
     - `KanbanBoardComponent` : Displays and manages Kanban boards.

        - `ScrumNotesComponent`: Manages Scrum notes and sprint planning.
        - `AIChatbotComponent` (if implemented): Handles AI chatbot interactions.
    - **Responsibilities**:
        - `AuthenticationService`: Validates user credentials, manages user sessions, and provides authentication-related functionality.
        - `KanbanBoardComponent`: Renders Kanban boards, allows card creation and movement, updates card statuses.
        - `ScrumNotesComponent`: Provides a user interface for creating, editing, and organizing Scrum notes, tracks sprint progress.
        - `AIChatbotComponent` (if implemented): Handles user queries, performs automated tasks, and interacts with external AI services.

2. **Backend Component**:
    - **Packages/Classes/Units of Abstraction**:
        - `UserAPI`: Handles user-related HTTP requests (e.g., registration, login).
        - `ProjectAPI`: Manages project data (e.g., creation, retrieval) and interfaces with the database.
        - `KanbanBoardService`: Contains business logic for Kanban board operations.
        - `ScrumNotesService`: Contains business logic for Scrum notes management.
        - `AIChatbotService` (if implemented): Orchestrates AI chatbot interactions and external service integration.
    - **Responsibilities**:
        - `UserAPI`: Validates and processes user registration and login requests, manages user data.
        - `ProjectAPI`: Receives and responds to requests related to project creation, retrieval, and updates.
        - `KanbanBoardService`: Contains logic for Kanban card movements, status updates, and data retrieval.
        - `ScrumNotesService`: Handles CRUD operations for Scrum notes, sprint planning, and task tracking.
        - `AIChatbotService` (if implemented): Orchestrates AI chatbot interactions, sends queries to AI services, and processes responses.

3. **Kanban Board Component**:
    - **Packages/Classes/Units of Abstraction**:
        - `KanbanBoard`: Represents a Kanban board, including cards and columns.
        - `KanbanCard`: Represents individual task cards on the board.
        - `KanbanColumn`: Represents columns on the board (e.g., "To Do," "In Progress," "Done").
    - **Responsibilities**:
        - `KanbanBoard`: Manages the overall structure of the board, including columns and cards.
        - `KanbanCard`: Contains data and metadata for individual tasks, including their status.
        - `KanbanColumn`: Represents and manages columns on the board, including adding/removing cards.

4. **Scrum Notes Component**:
    - **Packages/Classes/Units of Abstraction**:
        - `ScrumNote`: Represents an individual Scrum note.

        - `Sprint` : Represents a Scrum sprint, containing multiple Scrum notes.
        - `SprintPlanning` : Handles sprint planning and tracking.
    - **Responsibilities**:
        - `ScrumNote` : Stores data and metadata for individual Scrum notes, including their content and status.
        - `Sprint` : Organizes Scrum notes into sprints, tracks sprint progress.
        - `SprintPlanning` : Facilitates sprint planning meetings, assigns tasks, and monitors sprint progress.

5. **AI Chatbot Component (Optional)**:
    - **Packages/Classes/Units of Abstraction**:
        - `Chatbot` : Represents the AI chatbot and its interactions.
        - `NLPProcessor` : Handles natural language processing tasks.
        - `ExternalServiceConnector` : Interfaces with external AI services (if used).
    - **Responsibilities**:
        - `Chatbot` : Manages user interactions, processes user queries, and coordinates AI responses.
        - `NLPProcessor` : Performs natural language understanding and generation tasks.
        - `ExternalServiceConnector` : Communicates with external AI services for specialized tasks, such as data analysis or report generation (if applicable).

❖ Coding Guidelines

1. **TypeScript**: **TypeScript Style Guide**
    - **Reason for Choice**: TypeScript is a statically typed superset of JavaScript, and following a well-established style guide helps ensure code consistency and maintainability.
    - **Enforcement**: The team will use linters and code analysis tools like TSLint or ESLint with TypeScript configurations to enforce the coding standards. Code reviews will also be conducted to catch and address violations.
2. **React (TypeScript)**: **Airbnb React/JSX Style Guide**
    - **Reason for Choice**: The Airbnb React style guide is widely adopted and provides best practices for writing clean and maintainable React components.
    - **Enforcement**: ESLint with the Airbnb React/JSX configuration will be used to enforce React-specific coding standards. Code reviews will also be conducted to ensure adherence to the guidelines.
3. **HTML**: **Google HTML/CSS Style Guide**
    - **Reason for Choice**: Google's HTML/CSS Style Guide is comprehensive and provides clear recommendations for writing consistent and accessible HTML.
    - **Enforcement**: HTML linters like HTMLHint or HTMLLint can be integrated into the development workflow to enforce HTML coding standards. Code reviews will also play a role in maintaining consistency.
4. **Chakra UI**: Chakra-ui
    - **Reason for Choice**: Chakra UI provides a set of accessible and customizable UI components that can help maintain consistency in the user interface.

- **Enforcement**: Since Chakra UI components are used within React and TypeScript, adherence to React and TypeScript style guides will naturally ensure consistency when working with Chakra UI.

5. **Nest.js**: **Nest.js Official Documentation**
   - **Reason for Choice**: The official Nest.js documentation provides comprehensive guidance and best practices for building and deploying applications using Nest.js.
   - **Enforcement**: The team will closely follow the Nest.js documentation and recommended practices, ensuring that the development process adheres to best practices. Code reviews will play a significant role in ensuring the adoption of these practices throughout the project.
6. **PostgreSQL**: **SQL Style Guide**
   - **Reason for Choice**: The SQL Style Guide provides guidelines for writing clear and maintainable SQL queries, which is essential for working with PostgreSQL.
   - **Enforcement**: SQL queries will be reviewed as part of the database schema design and data access layer. Code reviews will be conducted to ensure SQL queries adhere to the style guide and are optimized for performance.

❖ Process Description

Risk Assessment:

1. **Technical Complexity Risk**
   - Likelihood: Medium
   - Impact: High
   - Evidence: The project involves integrating multiple technologies (TypeScript, React, Next.js, Chakra UI, PostgreSQL, and potentially an AI chatbot), which can lead to technical challenges.
   - Steps to Reduce:
     - Conduct thorough technology research and prototyping.
     - Regularly assess the team's technical skills and provide training if necessary.
   - Detection Plan: Monitor development progress and identify technical challenges early.
   - Mitigation: Seek external expert consultation if significant technical challenges arise.
2. **Scope Creep Risk**
   - Likelihood: Medium
   - Impact: High
   - Evidence: Scope changes and feature additions are common in software projects, potentially affecting the project timeline.
   - Steps to Reduce:
     - Maintain clear project requirements and a change control process.
     - Educate stakeholders about the impact of scope changes.
   - Detection Plan: Regularly review project requirements and scope.
   - Mitigation: Analyze the impact of scope changes and negotiate priorities with stakeholders.
3. **Resource Availability Risk**
   - Likelihood: Medium
   - Impact: Medium
   - Evidence: Team member availability may fluctuate due to external commitments or unforeseen circumstances.
   - Steps to Reduce:
     - Maintain open communication about availability and commitments.

        - Cross-train team members to mitigate skill-specific availability risks.
    - Detection Plan: Regularly update and review team availability.
    - Mitigation: Adjust work allocation and seek temporary resources if needed.
4. **Integration Risk**
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: Integrating various components and technologies can lead to compatibility and communication issues.
    - Steps to Reduce:
        - Plan for integration early and conduct regular testing.
        - Use standardized interfaces and communication protocols.
    - Detection Plan: Integrate components incrementally and perform thorough testing.
    - Mitigation: Develop contingency plans for integration issues.
5. **Data Security and Privacy Risk**
    - Likelihood: Low
    - Impact: High
    - Evidence: Handling user data and potential AI chatbot interactions requires a strong focus on security and privacy.
    - Steps to Reduce:
        - Implement robust security measures, including encryption and access controls.
        - Stay updated with data protection regulations (e.g., GDPR) and adhere to best practices.
    - Detection Plan: Regularly review and assess security measures.
    - Mitigation: Respond promptly to security incidents and comply with relevant regulations.

Changes Since Requirements Document Submission: The risk assessment section has been expanded to include a discussion of the top five risks, their likelihood, impact, evidence, steps to reduce the likelihood or impact, detection plans, and mitigation plans.

Project Schedule:

Milestones:

1. Project Planning & Architecture Design
2. Frontend Development (UI and User Authentication)
3. Backend Development (APIs, Database)
4. Kanban Board Component Development
5. Scrum Notes Component Development
6. AI Chatbot Integration (if applicable)
7. Integration and Testing
8. User Acceptance Testing
9. Documentation and Guides
10. Deployment and Release

Tasks and Dependencies: Tasks are organized within each milestone, and dependencies between tasks are indicated.

- *Project Planning & Architecture Design*
    - Define Project Requirements
    - Create Software Architecture

    - Choose Coding Style Guidelines
    - **Dependencies**: None
- *Frontend Development*
    - User Interface Design
    - User Authentication Implementation
    - Implement Kanban Board UI
    - Implement Scrum Notes UI
    - **Dependencies**: Architecture Design
- *Backend Development*
    - User Authentication API
    - Project Data API
    - Database Schema Implementation
    - **Dependencies**: Architecture Design
- *Kanban Board Component Development*
    - Card Management Logic
    - Drag-and-Drop Functionality
    - Card Status Updates
    - **Dependencies**: Frontend Development
- *Scrum Notes Component Development*
    - Scrum Note CRUD Operations
    - Sprint Management Logic
    - Sprint Planning UI
    - **Dependencies**: Frontend Development, Backend Development
- *AI Chatbot Integration (if applicable)*
    - Chatbot Interface Implementation
    - NLP Integration
    - External Service Integration
    - **Dependencies**: Frontend Development, Backend Development
- *Integration and Testing*
    - API Integration
    - Component Integration
    - Compatibility Testing
    - **Dependencies**: Frontend Development, Backend Development, Kanban Board Component Development, Scrum Notes Component Development, AI Chatbot Integration (if applicable)
- *User Acceptance Testing*
    - User Testing and Feedback
    - Bug Fixes and Improvements
    - **Dependencies**: Integration and Testing
- *Documentation and Guides*
    - User Guides
    - Admin Guides
    - Developer Guides
    - **Dependencies**: Integration and Testing
- *Deployment and Release*
    - Final Testing and Quality Assurance
    - Deployment to Production
    - Release to Users

    - **Dependencies**: User Acceptance Testing, Documentation and Guides

Team Structure: The team consists of six members with the following roles and responsibilities:

- **Project Manager**: Overall project planning, coordination, and risk management.
- **Frontend Developer**: UI design and development, user authentication.
- **Backend Developer**: API development, database management.
- **Kanban Board Component Developer**: Kanban board feature development.
- **Scrum Notes Component Developer**: Scrum notes feature development.
- **AI Chatbot Developer (if applicable)**: Chatbot integration and AI development.

Documentation Plan:

- User Guides: Detailed instructions for using the Campus Agile platform, including Kanban boards, Scrum notes, and AI chatbot (if applicable).
- Admin Guides: Guides for administrators to manage user accounts, project settings, and data.
- Developer Guides: Technical documentation for future maintenance and extensions of the system.
- Man Pages: Command-line utility documentation (if applicable).
- Help Menus: In-app help and tooltips for users.
- Wiki: Collaborative documentation for internal use and community support (if applicable).

Documentation will be developed incrementally alongside the software components. The team will allocate specific responsibilities for creating and maintaining documentation, ensuring that it remains up to date as the project progresses.