CSCI 4210: Introduction to Software Engineering

University of New Orleans
Department of Computer Science

# Capstone Project: Touchless Kiosk System

---

## Cycle 0:
## Requirements Analysis and Roadmapping

---

# Cycle 0: Requirements Analysis and Roadmapping

In Cycle 0 of the Capstone project, the focus is on gathering insights from stakeholders to inform the development of the touchless kiosk system. This cycle is broken into several key tasks that span Weeks 1 through 4. It involves identifying key users, conducting interviews, translating findings into actionable user stories and requirements, prioritizing features, and developing a roadmap. The roadmap will guide the system's development through cycles and ensure alignment with stakeholder needs. By the end of Cycle 0, all pods will merge their work into a collaborative GitHub repository and create a roadmap to track progress toward the Minimum Viable Product (MVP).

## Key Objectives:

- **Part 1: Problem Definition and Stakeholder Analysis** Pods will define the problem that the proposed system aims to solve, ensuring it aligns with real-world needs. They will then identify relevant categories of stakeholders and identify their potential user stories.
- **Part 2: Stakeholder feedback and User Story Refinement** Interview stakeholders and based on their feedback, pods will refine the initial user stories to reflect how the system should meet stakeholder needs and begin documenting functional and non-functional requirements.
- **Part 3: Requirements Documentation and Feature Prioritization** Pods will compile a comprehensive requirements document, detailing both functional and non-functional requirements. Additionally, each pod will prioritize the identified features by criticality and necessity, marking those needed for the MVP.
- **Part 4: Developing the Roadmap Based on Features** Using the prioritized feature list, pods will organize features into development cycles and map out key milestones. Each cycle should have associated timeframes and clearly defined milestones (e.g., "MVP Complete").
- **Part 5: Collaborative GitHub Project Board** As the final step, all pods will merge their work into a shared GitHub repository, establishing a collaborative GitHub Project board. Tasks will be organized and tracked using columns like "Backlog," "In Progress," and "Completed," with tasks tagged by cycle (`cycle-1`, `cycle-2`, `cycle-3`) to allow filtering by development stage. This shared project board will manage task progress and ensure proper version control and documentation.

This multi-staged approach ensures that all development efforts are systematically aligned with real-world use cases and business goals, while also providing a clear roadmap for project progress.

# Part 1: Problem Definition and Stakeholder Analysis

**Objective:** Establish the motivation for the touchless kiosk system, identify relevant stakeholders, and draft initial user stories that capture possible use cases. This sets the groundwork for stakeholder interviews in Week 2.

## Tasks:

1. **Define the System's Motivation:** Each pod should collaboratively define the system's purpose and identify its target audience. Consider the following:
   - Why is this system needed?
   - What problems does it solve?
   - How will it benefit users and the organization?
2. **Identify Stakeholders:** Based on the system's motivation, pods will identify potential stakeholders who are likely to use or be impacted by the touchless kiosk system (e.g., faculty, students, tutors). Each pod should prepare a comprehensive list of relevant stakeholders.
3. **Develop Initial User Stories:** Using the identified stakeholders, pods should draft initial user stories that describe potential use cases and how the stakeholders might interact with the system. Each pod must submit **at least 10 user stories per pod member**, either exploring a single stakeholder deeply or covering multiple stakeholders. These stories should consider different user goals.

## User Story Template

When drafting user stories, use the following template format. Each pod should provide a minimum of 10 user stories per pod member, which can focus on a single stakeholder deeply or cover multiple stakeholders.

- "As a [stakeholder], I want [functionality] so that [desired outcome]."

**Explanation:**

- **Stakeholder:** Identify who the user is, such as a student, faculty, IT staff, etc.
- **Functionality:** Describe the feature the user wants to interact with.
- **Desired Outcome:** Explain the benefit or value this feature provides to the user.

## Example User Stories

Below are some example user stories related to the CSCI department's touchless kiosk system. Pods should draft similar stories specific to their stakeholders.

- "As a student, I want to check my professors' office hours and course schedules so that I can plan my visits efficiently."
- "As a student, I want to find available tutoring sessions for CSCI courses so that I can get extra help outside of class."
- "As a faculty member, I want to display announcements about upcoming CSCI events, like guest lectures or workshops, so that students stay informed."
- "As a faculty member, I want to update my course information on the kiosk so that students have access to the most current syllabus and assignment deadlines."
- "As a CSCI club member, I want to view a calendar of CSCI department events so that I can attend relevant technical talks and coding competitions."
- "As an IT staff member, I want to remotely update the kiosk content, like software engineering course schedules or department announcements, so that everything is accurate and up-to-date."
- "As an IT staff member, I want to monitor the kiosks to ensure they are functioning correctly and available for use by students and faculty."
- "As a prospective CSCI student, I want to explore CSCI course offerings and major requirements so that I can better understand the program."
- "As a tutor, I want to post my tutoring schedule for CSCI courses so that students know when and where they can get assistance."

**Outline Interview Topic Points:** Based on the user stories, define key questions and areas to explore in stakeholder interviews for Week 2. The interview questions should probe deeper into the stakeholders' needs, pain points, and desired features.

# Deliverable:

- **Week 1: Stakeholder Identification Report and Initial User Stories.**

  **Content:**

  1. **Stakeholder Identification:** Each pod must identify relevant stakeholders who will interact with the system (e.g., students, faculty, admins). This should include the role of each stakeholder and their potential impact on the system.
  2. **System Motivation:** Each pod must outline the motivation for developing the system. This should address why the system is needed, what problems it will solve, and how it will benefit the organization and users.
  3. **User Stories:** Each pod member must submit at least **10 user stories**, either focusing on one stakeholder in-depth or exploring multiple stakeholders. These stories must follow the format provided in the **Stakeholder Template Example**.

  **Submission Guidelines:**

  1. Each pod should create a markdown file for their stakeholder identification and user stories, named `stakeholder-[role].md`.
  2. All items must be submitted in a shared GitHub repository under the `requirement_analysis` folder.

  **Due:** End of Week 1.

## Stakeholder Template

```
# Stakeholder Analysis Template

## Stakeholder: [Stakeholder Role]
- **Role**: Describe the stakeholder's role in the organization and their interaction
    ↪ with the touchless kiosk system.
- **Needs**: List what the stakeholder requires from the system, focusing on
    ↪ functionalities or tasks.
- **Pain Points**: Identify current challenges or frustrations the stakeholder faces
    ↪ that the system could address.
- **Desired Outcomes**: Describe what success looks like for the stakeholder once the
    ↪ system is implemented.

### User Story:
"As a [stakeholder], I want [functionality] so that [desired outcome]."
```

## Example Output

Below is an example based on a single-user story for a faculty stakeholder:

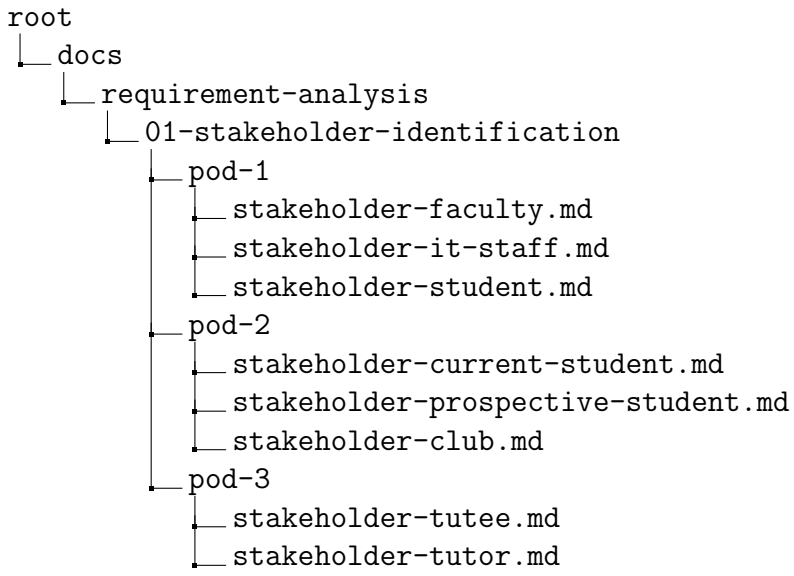
```
# Stakeholder Analysis Template

## Stakeholder: Faculty Member
- **Role**: Faculty members are responsible for teaching courses and providing
    ↪ students with the necessary information regarding their schedules, office hours
    ↪ , and course materials.
- **Needs**: Faculty need the ability to update their course information, event
    ↪ announcements, and office hours quickly on the kiosk.
- **Pain Points**: Faculty members currently rely on email and physical noticeboards,
    ↪ which may lead to delays in communication.
- **Desired Outcomes**: A system that allows them to remotely update all relevant
    ↪ information and ensures that it is displayed immediately for students.

### User Story:
"As a faculty member, I want to update kiosk content remotely so that the most current
    ↪  information is always available to students."
```

## Directory Structure (Example)

The following is an example of how the shared directory structure should look based on the submissions from the pods for **Part 1: Stakeholder Identification**. Actual stakeholder types and file names may vary depending on each pod's analysis and choices.




**File Naming Convention:**

- Each stakeholder's analysis and corresponding user stories should be saved as markdown files with the following naming convention: `stakeholder-[role].md`
- Examples:
  - `stakeholder-faculty.md`
  - `stakeholder-it-staff.md`
  - `stakeholder-student.md`

# Concluding Task: Empty GitHub Project Board

At the conclusion of Part 1, the entire class must collaborate to create a shared, empty GitHub Project board. This board will serve as the central platform for managing tasks and tracking progress throughout the project.

**The GitHub Project board must minimally include the following columns:**

- **To Do:** Tasks that are yet to be started.
- **In Progress:** Tasks currently being worked on.
- **Done:** Completed tasks.

**Purpose:** This shared board will be used by all pods to organize and track tasks throughout the duration of the capstone project, ensuring a collaborative and transparent workflow. Each pod will add their tasks to this board in subsequent cycles as they progress through their respective work.

## Supplemental Resource:

GitHub Projects is a project planning tool that allows you to:

- Create issues, break them into tasks, track relationships, and add custom fields.
- Visualize large projects using tables, boards, or roadmaps.
- Automate workflows with code.

Learn more at: `https://github.com/features/issues`

# Part 2: Refining User Stories with Stakeholder Input

**Objective:** Validate and refine the initial user stories by conducting interviews with selected stakeholders. This process will provide deeper insights into their needs, challenges, and expectations, allowing us to adjust the system's functionality based on real-world feedback.

## Tasks:

1. **Schedule Stakeholder Interviews:** After reviewing the work submitted in Part 1, each pod must schedule **2-3 interviews** with **unique** stakeholders. To do this:
    - Select stakeholders from the combined stakeholder list submitted in Part 1.
    - Add tasks for each interview to the GitHub Projects board, specifying the stakeholder role and pod member responsible for conducting the interview.
    - Ensure the interview schedule includes the proposed dates and times for the interviews.
    - These tasks should be tracked on the project board, with statuses moving from "To Do" to "In Progress" and then "Done" after the interviews are completed.
2. **Conduct Stakeholder Interviews:** Each pod must conduct the interviews they have scheduled. Focus on the following during the interviews:
    - What specific features or functionalities do the stakeholders consider essential?
    - What pain points do they currently face that the system could address?
    - Are there any surprising insights that were not initially considered during the ideation cycle?
    - How does the stakeholder envision integrating the system into their daily workflow?
3. **Summarize Interview Insights:** After conducting the interviews, summarize the key findings for each stakeholder. Focus on:
    - Essential features the stakeholders emphasized.
    - Any pain points or challenges stakeholders identified.
    - New insights that might influence the design or features of the kiosk system.
4. **Refine User Stories:** Using the feedback from stakeholders, refine the initial user stories to reflect their actual needs and priorities. Be sure to:
    - Remove any user stories that are no longer relevant based on the feedback.
    - Add new user stories to address unanticipated needs or challenges.
    - Make existing user stories more specific or adjust them based on stakeholder insights.

## Interview Summary Template

Each pod should summarize the findings from their stakeholder interviews using the following template. This will ensure consistency and completeness across submissions.

```
# Stakeholder Interview Summary

## Stakeholder: [Stakeholder Role]
- **Interview Date**: [Date]
- **Interviewee**: [Name or Identifier]

### Key Insights:
- **Essential Features**: Describe what the stakeholder considers must-have features.
- **Pain Points**: Highlight the challenges or pain points identified during the interview.
- **Insights**: List any unexpected insights or new use cases that emerged during the interview.
- **Workflow Integration**: How does the stakeholder envision using the system in their daily work?

### Refined User Stories:
1. "As a [stakeholder], I want [functionality] so that [desired outcome]."
2. "As a [stakeholder], I want [functionality] so that [desired outcome]."
```

## Example Interview Summary Output

```
# Stakeholder Interview Summary

## Stakeholder: Faculty Member
- **Interview Date**: September 28, 2024
- **Interviewee**: Professor Ted Holmberg

### Key Insights:
- **Essential Features**: The faculty member emphasized the need to quickly update event announcements and
    ↪  office hours remotely.
- **Pain Points**: Current systems rely on email or manual updates to physical noticeboards, which are
    ↪ slow and prone to miscommunication.
- **Surprising Insights**: The faculty member suggested integrating a notification icon to alert students
    ↪ when changes were recently made.
- **Workflow Integration**: The faculty member anticipates using the system weekly to update office hours
    ↪ and event information.

### Refined User Stories:
1. "As a faculty member, I want to update kiosk content remotely so that I can quickly share changes with
    ↪ students."
2. "As a faculty member, I want a notification system to alert students when updates are made to the kiosk
    ↪ ."
```

# Deliverable:

- **Week 2: Stakeholder Interview Summary Report and Scheduled Interviews.** *Content:* A report summarizing the interviews, including key insights and the refined user stories. Each pod should submit their findings along with the scheduled interview tasks created in GitHub Projects. All interviews should be documented in the GitHub Project board for tracking. *Due:* End of Week 2.

## Guidelines for Submissions

- The interview summaries and refined user stories must be saved as `.md` files.
- Ensure all key insights from the interviews are captured in detail, as these will influence the next stages of development.
- Submissions should be placed in the GitHub repository under the `requirement_analysis` directory.

## Directory Structure (Example)

Below is an example of how the directory structure should look for Part 2 submissions. All files should be placed in the shared GitHub repository under the `requirement_analysis` folder.

```
root
└── docs
    └── requirement-analysis
        └── 02-stakeholder-interviews
            ├── pod-1
            │   ├── interview-summary-faculty.md
            │   └── interview-summary-it-staff.md
            ├── pod-2
            │   ├── interview-summary-current-student.md
            │   └── interview-summary-club.md
            └── pod-3
                ├── interview-summary-tutor.md
                └── interview-summary-tutee.md
```

**File Naming Convention:**

- Each interview summary should be saved as a markdown file with the following naming convention: `interview-summary-[role].md`
- Example: `interview-summary-faculty.md`, `interview-summary-it-staff.md`

# Part 3: Requirements Document & Feature Ranking with Priority and Difficulty

**Objective:** Compile a comprehensive requirements document based on stakeholder feedback and refine user stories. Prioritize features, assess difficulty, and identify the core features that will form the MVP (Minimum Viable Product).

## Tasks:

1. **Complete the Requirements Document:** Each pod will finalize functional and non-functional requirements for the touchless kiosk system using the refined user stories from Part 2. Ensure the requirements document captures:

    - **Functional requirements:** These define what the system must do. They describe specific behaviors or functions the system needs to support, typically focusing on processes, data handling, and interactions with users or other systems. Examples include user authentication, report generation, and content management.

    - **Non-functional requirements:** These define how the system must perform its functions. They focus on quality attributes such as performance, security, usability, reliability, and scalability. Examples include the system's speed, security measures like encryption, and its ability to support a large number of users.

2. **Compile, Prioritize, and Assess Feature Difficulty:** Using the refined user stories, create a comprehensive feature list. This process involves translating user stories into tangible software components or abstractions. For each feature, assign:

    - **Priority:** Categorize each feature as Critical (must-have), High (should-have), Medium (could-have), or Low (nice-to-have/bonus).

    - **Difficulty:** Assess the complexity and effort required for each feature, labeling them as Easy, Medium, or Hard.

3. **Identify MVP Features:** Select the core features that are essential for delivering a functional product. This set of features will define the MVP, focusing on Critical priority and Medium or Easy difficulty where possible.

## Gauging Task Complexity

When prioritizing features by difficulty, it's essential to gauge the complexity of each task accurately. Assigning a difficulty rating helps set realistic expectations for how long tasks will take and how to distribute work across the team. Below are insights and examples to help assess the complexity of tasks:

### 1. Understanding Task Complexity

Task complexity is determined by several factors, including:

- **Size and Scope:** How many subtasks or steps are involved in completing the task?
- **Technical Requirements:** How technically challenging is the task? Does it require specialized knowledge, tools, or libraries?
- **Dependencies:** Does the task rely on the completion of other tasks? How many external systems or components need to be integrated?
- **Uncertainty:** How familiar are you with the problem space? Is there uncertainty about the task or unexpected challenges that might arise?

### 2. Difficulty Ratings

The most common way to categorize task difficulty is through three levels:

- **Easy:** These tasks are typically straightforward, require little research or planning, and have no significant dependencies.
- **Medium:** These tasks may require more steps or involve some technical challenges. They often include minor dependencies or require coordination with other teams.
- **Hard:** These tasks involve complex technical challenges, significant dependencies, or are critical to the system's architecture. They may also require extensive research, prototyping, or coordination across multiple teams.

### 3. Time Estimates

One way to help gauge difficulty is by associating the complexity with a rough time estimate:

- **Easy:** Tasks that can typically be completed in a few hours or less.
- **Medium:** Tasks that require about a day or two of work.
- **Hard:** Tasks that will take several days to a week to complete, or that need to be broken down into multiple subtasks to manage.

*Note: Time estimates are relative and will vary based on the skill level of your team and familiarity with the task.*

### 4. Examples of Task Complexity

Here are examples of different tasks and how they might be classified:

**Easy:**

- Create a basic UI mockup for a dashboard using wireframing tools (time: 2 hours).
- Write documentation for the user authentication module (time: 3 hours).
- Set up a new project in GitHub and initialize a repository (time: 1 hour).

**Medium:**

- Implement basic API endpoints for user authentication (time: 1-2 days).
- Develop a dashboard UI component for tutor scheduling (time: 1-2 days).
- Integrate a third-party library for gesture recognition in Leap Motion (time: 1-2 days).

**Hard:**

- Design and implement the backend logic for user role management with multiple permission levels (time: 3-5 days).
- Build an entire system feature like Leap Motion gesture-based control for the kiosk, with user testing and integration (time: 5+ days).
- Develop a data synchronization system across multiple kiosks, ensuring real-time updates and conflict resolution (time: 1 week+).

### 5. Factors to Consider When Gauging Task Difficulty

To more accurately assign difficulty levels, consider the following factors:

- **Skill Level of Team:** Is the team familiar with the technology or system? A task may be easy for experienced team members but medium or hard for newer developers.
- **Level of Detail:** A task requiring high levels of detail or precision (e.g., integrating security measures) may be more difficult even if it seems simple conceptually.
- **External Dependencies:** Tasks relying on third-party APIs, external services, or other teams' work might be harder due to dependency risks.
- **Potential Roadblocks:** Identify areas where bugs, performance issues, or design flaws might emerge. If a task is prone to unforeseen issues, it may warrant a higher difficulty rating.

**6. How to Use Difficulty Ratings**

- **Task Allocation:** Assign easier tasks to junior team members or as warm-up tasks for everyone to get familiar with the project. Harder tasks should be taken on by more experienced developers or distributed across the team to prevent bottlenecks.
- **Project Timeline:** Consider task difficulty when planning sprints or project cycles. Ensure that high-complexity tasks are allocated more time and are started early to allow for adjustments or fixes.
- **Iterative Approach:** For complex tasks, break them down into smaller, manageable subtasks. This not only reduces difficulty but also helps track progress and prevent delays.

By considering these factors and using the examples provided, students can better estimate the complexity of tasks, leading to more accurate roadmaps and smoother project execution.

## Grouping Features by Systems

Each pod will group the features based on systems that implement them. This approach mirrors real-world software development by modularizing the functionality into subsystems. Grouping features by systems ensures that technical dependencies and shared components (such as user authentication or Leap Motion Integration) are prioritized and completed before system-specific features are built.

## Example of System-Based Features

### Tutor System

- **Tutor Dashboard:** Interface for tutors to manage schedules and resources. (Priority: Critical, Difficulty: Medium)
- **Schedule Management:** Back-end system to manage tutor schedules. (Priority: High, Difficulty: Medium)

### Student System

- **Course Material Access:** Allows students to access course-related materials. (Priority: Critical, Difficulty: Hard)
- **Office Hours Search:** Enables students to find professor office hours. (Priority: Medium, Difficulty: Easy)

### Public System

- **Event Display:** Displays public events on the kiosk. (Priority: Low, Difficulty: Medium)
- **News Updates:** Shows relevant news to public users. (Priority: Low, Difficulty: Easy)

### Shared System

- **Authentication System:** Handles user authentication and role management. (Priority: Critical, Difficulty: Hard)
- **Leap Motion Integration:** Gesture-based control system for all users. (Priority: High, Difficulty: Hard)

## Rationale:

In industry, software systems are typically developed in modular components or subsystems, which allows for better encapsulation, scalability, and maintainability. By grouping features into systems, we ensure that the development process aligns with real-world practices. This approach allows for clearer task assignments, easier management of technical dependencies (e.g., between authentication and user management systems), and more realistic roadmapping where shared systems are completed before building user-specific features.

Grouping by systems also promotes independent evolution of each system, meaning changes to one system (such as the authentication process) can be made without impacting unrelated systems. This modularity supports concurrent development by different teams, making it easier to map user stories to technical components and ensure well-defined API interactions.

## Requirements Template

The requirements document for each system should be structured using the following template:

```
# [System Name] Requirements Document

## 1. Functional Requirements
- [Requirement 1]: Description of functionality the system must have.
- [Requirement 2]: Description of another functionality.

## 2. Non-Functional Requirements
- [Requirement 1]: E.g., system must respond in under 2 seconds.
- [Requirement 2]: E.g., system must ensure data security.

## 3. Feature List, Prioritization, and Difficulty
### Critical Features (MVP)
- [Feature 1]: Must-have functionality. (Priority: Critical, Difficulty: Medium)
- [Feature 2]: Another critical feature. (Priority: Critical, Difficulty: Hard)

### High Priority Features
- [Feature 3]: Important for enhanced usability. (Priority: High, Difficulty: Easy)
- [Feature 4]: Another high-priority feature. (Priority: High, Difficulty: Medium)

### Medium Priority Features
- [Feature 5]: Nice-to-have functionality. (Priority: Medium, Difficulty: Medium)
- [Feature 6]: Additional improvements. (Priority: Medium, Difficulty: Hard)

### Low Priority Features
- [Feature 7]: Bonus feature. (Priority: Low, Difficulty: Easy)
```

## Example: Tutor System Requirements

```
# Tutor System Requirements Document

## 1. Functional Requirements
- **Tutor Dashboard:** The system must allow tutors to view and manage their schedules
    ↪ .
- **Schedule Management:** Tutors should be able to update their availability in real-
    ↪ time.

## 2. Non-Functional Requirements
- **Performance:** System updates should be reflected in the dashboard within 2
    ↪ seconds of a change.
- **Availability:** The system should be accessible 99.9% of the time during the
    ↪ school year.

## 3. Feature List, Prioritization, and Difficulty
### Critical Features (MVP)
- **Tutor Dashboard:** Must-have functionality for managing tutor schedules. (Priority
    ↪ : Critical, Difficulty: Medium)

### High Priority Features
- **Schedule Management:** Allows tutors to modify their schedules. (Priority: High,
    ↪ Difficulty: Medium)

### Medium Priority Features
- **Notification System:** Alerts tutors when their schedules are updated. (Priority:
    ↪ Medium, Difficulty: Easy)

### Low Priority Features
- **Export Schedule:** Tutors can export their schedules to a calendar app. (Priority:
    ↪  Low, Difficulty: Easy)
```

# Deliverable:

- **Week 3: Requirements Document, Feature Prioritization, and Difficulty.** *Content:* A finalized requirements document that includes functional and non-functional requirements, a prioritized feature list with difficulty ratings, and a clear identification of MVP features. *Due:* End of Week 3.

## Directory Structure (Example)

All documents should be organized in the shared GitHub repository under the `requirement_analysis` folder. Below is an example of how the directory should be structured for Part 3 submissions:

```
root
└── docs
    └── requirements-analysis
        └── 03-requirements
            ├── Tutor-System.md
            ├── Student-System.md
            ├── Public-System.md
            └── Shared-System.md
```

**File Naming Convention:**

- Each pod's requirements document and feature list should be saved as markdown files with the following naming convention: `[system-name].md`
- Examples:
  - `Tutor-System.md`,
  - `Student-System.md`

# Part 4: Develop the Roadmap Based on Requirements

**Objective:** Each pod will independently analyze their user stories and feature lists to identify both user-specific systems and shared architecture systems. Based on this analysis, pods will break down the identified systems into incremental cycles (priority-based), define milestones for each cycle, and outline the sprints and tasks required to complete each milestone. Each pod's deliverable is a roadmap document capturing the systems, cycles, milestones, sprints, and tasks, clearly indicating which milestones are part of the MVP.

These deliverables will serve as the foundation for further collaboration, where all pods will later merge their roadmaps into a unified project in future cycles.

## Cycle Development and Goals:

The development process will be divided into incremental cycles, where each cycle focuses on delivering a set of related features or improvements. Cycles are organized around priority levels, with each cycle having a specific goal that contributes incremental value to the overall project. This structure ensures that critical functionality is delivered first (MVP), followed by additional features in subsequent cycles.

Each cycle will be broadly defined by its purpose and broken down into the following components:

- **Milestones:** Major system achievements (e.g., "User System Complete"), representing critical deliverables within the cycle.
- **Sprints:** Smaller, time-boxed iterations that focus on specific tasks (e.g., "Develop login API," "Implement frontend integration") to achieve the milestones.
- **Tasks:** Specific actions completed during each sprint to ensure progress toward the milestone.

**Cycle-Specific Goals:**

- **Cycle 1: Minimum Viable Product (MVP)** focuses on delivering the core features that make the system usable and functional. This cycle prioritizes essential user interaction and shared system integration.
- **Cycle 2: High-Priority Features** adds features that enhance the system by improving user experience, robustness, and functionality, often based on feedback from the MVP cycle.
- **Cycle 3: Medium and Low Priority Features** introduces nice-to-have features that improve system efficiency, convenience, or extended capabilities, ensuring the system is refined and optimized.

## Milestones and Incremental Value:

Each cycle will be broken down into milestones that represent major system deliverables (e.g., **User System**, **Authentication System**, **Leap Motion Integration**). These milestones are tied to specific systems, and each milestone will be further broken down into sprints and tasks that ensure incremental progress toward the overall project goals.

## Shared Architecture as Milestones:

Systems such as the **User System**, **Authentication System**, and **Leap Motion Integration** are foundational shared components and must be treated as milestones. Depending on their priority and importance to the MVP, some or all of these components may be included in the MVP to ensure robust user-specific systems are built on top of them.

## User-Specific Systems:

User-specific systems represent features required by distinct user types (e.g., tutors, instructors, students). A complete system or a subset of its milestones might be part of the MVP depending on priority and use case. These systems will build on the shared architecture and reflect user stories specific to their interaction with the system.

**Example User-Specific Systems:**

- **Tutor System**: Manages tutor schedules and resources.
- **Student System**: Allows students to access course materials and track grades.
- **Instructor System**: Helps instructors manage assignments and review student progress.
- **Prospective Student System**: Provides a portal for prospective students to explore course offerings.

## Leap Motion as a Shared Component:

Leap Motion, as a core element of user interaction, must be treated as a shared milestone. Depending on its importance to the MVP, some Leap Motion functionality may be deferred to later milestones. It will be integrated early and used across user-specific systems for gesture-based control.

## Tasks for Each Pod:

Each pod will perform the following tasks:

1. **Identify Systems and Define Cycles, Milestones, Sprints, and Tasks:** Each pod will use their user stories and feature lists to identify the user-specific and shared systems they will work on.

    - Break down the systems into cycles, milestones, and sprints.
    - Define the milestones for each cycle, clearly indicating which milestones are part of the MVP.
    - Identify tasks required to achieve each sprint, including priority and complexity levels.

2. **Create Independent Pod Deliverables:** Each pod will independently document its roadmap, including the systems, cycles, milestones, sprints, and tasks. Pods will assign weights or complexity to each task (e.g., low, medium, high). Each roadmap will be submitted in markdown format and organized by system.

    **Example Systems:**

    - User System
    - Authentication System
    - Leap Motion Integration
    - Tutor System
    - Student System
    - Instructor System
    - Prospective Student System

3. **Submit Roadmap Analysis:** Each pod will submit their roadmap in markdown format, with clearly defined cycles, milestones, sprints, and tasks. Submissions should be organized into a shared directory with each pod responsible for its section.

## Roadmap Template:

The roadmap document should follow this structure, providing a clear breakdown of systems (both shared and user-specific), their milestones, sprints, and tasks, and mapping them into development cycles.

```
# Roadmap Document

## 1. Overview of Systems
### Shared Systems
- **System 1**: Description of a shared system (e.g., Authentication System).
- **System 2**: Another shared system (e.g., Leap Motion Integration).
- [Add more shared systems as needed]

### User-Specific Systems
- **System 1**: Description of a user-specific system (e.g., Tutor System).
- **System 2**: Another user-specific system (e.g., Student System).
- [Add more user-specific systems as needed]

## 2. Systems Breakdown: Milestones, Sprints & Tasks
### Shared Systems
#### System 1: [Shared System Name]
- **Milestone 1**: Description of the first milestone.
    - **Sprint 1**: Description of sprint 1.
        - **Task 1**: Description of task 1 (priority: high, complexity: medium).
        - **Task 2**: Description of task 2 (priority: high, complexity: low).
    - **Sprint 2**: Description of sprint 2.
        - **Task 1**: Description of task 1 (priority: high, complexity: high).

#### System 2: [Shared System Name]
- **Milestone 1**: Description of the milestone.
    - **Sprint 1**: Description of sprint 1.
        - **Task 1**: Description of task 1 (priority: medium, complexity: medium).

### User-Specific Systems
#### System 1: [User-Specific System Name]
- **Milestone 1**: Description of the first milestone.
    - **Sprint 1**: Description of sprint 1.
        - **Task 1**: Description of task 1 (priority: high, complexity: medium).
        - **Task 2**: Description of task 2 (priority: medium, complexity: high).

#### System 2: [User-Specific System Name]
- **Milestone 1**: Description of another milestone.
    - **Sprint 1**: Description of sprint 1.
        - **Task 1**: Description of task 1 (priority: low, complexity: low).

## 3. Incremental Cycles
### Cycle 1: Minimum Viable Product (MVP)
- **System 1 (Shared)**:
    - Milestone 1, Task 1 (priority: high).
    - Milestone 1, Task 2 (priority: high).
- **System 2 (User-Specific)**:
    - Milestone 1, Task 1 (priority: high).
    - Milestone 2, Task 1 (priority: medium).

### Cycle 2: High Priority Features
- **System 1 (Shared)**:
    - Milestone 2, Task 1 (priority: high).
- **System 2 (User-Specific)**:
    - Milestone 2, Task 1 (priority: medium).

### Cycle 3: Medium and Low Priority Features
- **System 1 (User-Specific)**:
    - Milestone 1, Task 1 (priority: low).
    - Milestone 2, Task 1 (priority: medium).
```

# Example Roadmap Submission: Pod 1

```
# Roadmap Document: Pod 1

## 1. Overview of Systems
### Shared Systems
- **Authentication System**: Enables secure login and session management.
- **Leap Motion Integration**: Provides gesture-based interaction for users.

### User-Specific Systems
- **Tutor System**: Allows tutors to manage schedules, access resources, and interact with students.

## 2. Systems Breakdown: Milestones & Tasks
### Shared Systems
#### Authentication System
- **Milestone 1**: Basic login functionality.
    - **Sprint 1**: Develop login API (priority: high, complexity: medium).
    - **Sprint 2**: Implement session management (priority: high, complexity: high).

#### Leap Motion Integration
- **Milestone 1**: Basic gesture recognition.
    - **Sprint 1**: Develop gesture recognition (priority: high, complexity: medium).

### User-Specific Systems
#### Tutor System
- **Milestone 1**: Design Tutor Dashboard.
    - **Sprint 1**: UI Mockups (priority: high, complexity: low).
    - **Sprint 2**: Backend Setup for Dashboard (priority: medium, complexity: medium).
- **Milestone 2**: Schedule management feature.
    - **Sprint 1**: API for schedule management (priority: medium, complexity: high).

## 3. Incremental Cycles
### Cycle 1: Minimum Viable Product (MVP)
- **Authentication System**:
    - Milestone 1, Sprint 1 (priority: high).
    - Milestone 1, Sprint 2 (priority: high).
- **Tutor System**:
    - Milestone 1, Sprint 1 (priority: high).
    - Milestone 1, Sprint 2 (priority: medium).

### Cycle 2: High Priority Features
- **Leap Motion Integration**:
    - Milestone 1, Sprint 1 (priority: high).
- **Tutor System**:
    - Milestone 2, Sprint 1 (priority: medium).

### Cycle 3: Medium and Low Priority Features
- **Tutor System**:
    - Milestone 2, Sprint 1 (priority: medium).
```

# Deliverables for Part 4:

- **Pod Deliverables:** Each pod will submit a markdown document detailing their roadmap analysis, including systems, cycles, milestones, sprints, and tasks.
  - User-specific systems
  - Shared architecture systems
  - Development cycles
  - Milestones for each cycle, indicating MVP components
  - Sprints and tasks required to achieve each milestone
- **Content for Submission:** The markdown file should break down each system by cycles, milestones, sprints, and tasks. Each task should have a weight or complexity level to assist in planning.

## Submission Directory Structure:

```
root
└── docs
    └── requirements-analysis
        └── 04-roadmap
            ├── roadmap-pod-1.md
            ├── roadmap-pod-2.md
            └── roadmap-pod-3.md
```

# Part 5: GitHub Project Setup & Collaboration

**Objective:** Now that each pod has developed a detailed roadmap in Part 4, the next step is to collaboratively track progress through a shared GitHub repository. Pods will merge their roadmaps into a unified GitHub Project board to ensure that all tasks, features, and milestones are properly tracked and organized.

Each pod's roadmap, including the incremental cycles, milestones, and tasks, will be converted into GitHub issues or project cards and managed using a GitHub Project board. This will enable efficient tracking of task completion, proper version control, and ongoing collaboration across pods.

## Tasks:

1. **Merge Roadmaps and Features into GitHub:** Each pod must consolidate the systems, milestones, and tasks defined in Part 4 into a unified GitHub Project board. All tasks should be organized by cycle (e.g., `cycle-1`, `cycle-2`, `cycle-3`) to allow filtering by development stage.

2. **Set Up Collaborative GitHub Project Board:** A shared GitHub repository will be created, and all roadmaps will be converted into a project board with the following columns:

   - **Backlog:** All tasks identified from the roadmap that are yet to be started.
   - **In Progress:** Tasks that are actively being worked on.
   - **Completed:** Tasks that have been fully completed and reviewed.

   Each task on the project board must be labeled with the appropriate cycle and should be assigned to pod members responsible for completing it.

3. **Version Control and Collaboration:** Pods will use GitHub branches to ensure that features or milestones can be developed concurrently. Use consistent naming conventions for branches (e.g., `feature-[name]`, `milestone-[name]`), ensuring that each branch represents a clear and isolated task from the roadmap.

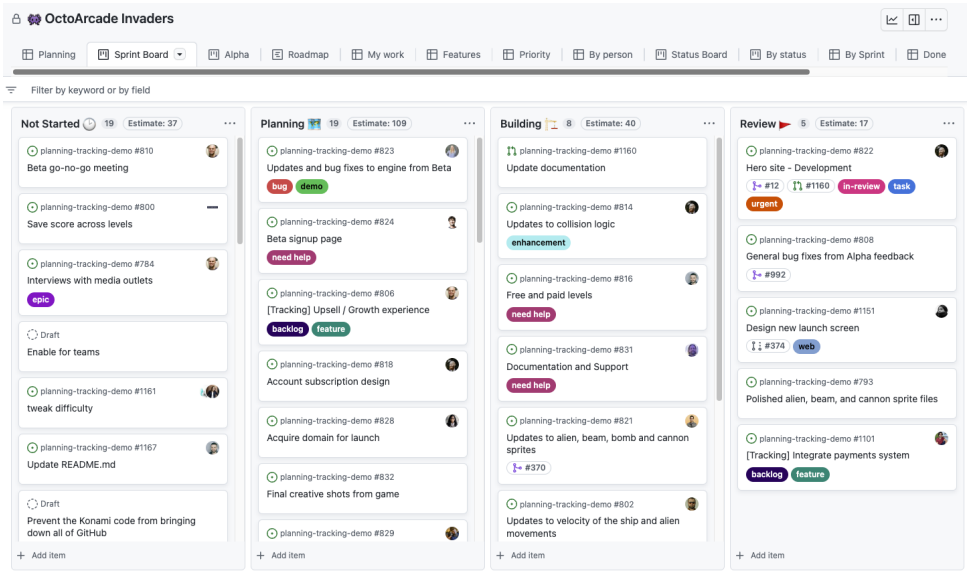
4. **Tagging and Filtering by Cycle:** All tasks on the GitHub Project board should be tagged with the relevant cycle label (`cycle-1`, `cycle-2`, `cycle-3`) to ensure that progress is tracked for each development stage. This allows pods to focus on the current cycle of development and plan for future cycles.

5. **Documentation in GitHub:** Each pod will centralize all relevant project documentation within the GitHub repository, using either the README file or GitHub Wiki. This should include:

   - The final requirements document and roadmap from Part 4.
   - A summary of current progress, upcoming tasks, and completed milestones.

## Example GitHub Project Board

Below is an example of how your GitHub Project board might be structured. Tasks should be added as issues or cards, and progress will be tracked using columns like "Backlog," "In Progress," and "Completed." Additionally, use the cycle labels to filter tasks as needed.




## Deliverables for Part 5:

- **GitHub Project Board Setup:** The unified GitHub Project board should be functional, with all tasks organized into cycles and properly tagged. Progress must be tracked through the **Backlog**, **In Progress**, and **Completed** columns.
- **Centralized Documentation:** All documentation related to the project, including the final roadmap, requirements document, and task progress, must be available in the GitHub repository, either in a README file or the Wiki.
- **Branching and Version Control:** Ensure that proper version control is established with branches for different features or milestones, using clear and consistent branch names.

**Due:** End of Week 4

# Grading Rubric:

**Week 1 Deliverable: Stakeholder Identification Report and GitHub Project Setup (20%)**

**Tasks for Week 1:**

| Task | Points |
|---|---|
| Collaborate to identify relevant stakeholders | 4 |
| Assign unique stakeholders to each pod | 6 |
| Draft initial user stories based on insights | 6 |
| Set up the shared GitHub Project Board | 4 |
| **Total** | **20** |

**Week 2 Deliverable: Stakeholder Interview Summary Report (20%)**

**Tasks for Week 2:**

| Task | Points |
|---|---|
| Conduct and document stakeholder interviews | 8 |
| Summarize stakeholder insights, pain points, and needs | 8 |
| Refine user stories based on interviews | 4 |
| **Total** | **20** |

**Week 3 Deliverable: Requirements Document & Feature Prioritization (30%)**

**Tasks for Week 3:**

| Task | Points |
|---|---|
| Develop a complete requirements document | 12 |
| Compile a feature list from user stories | 8 |
| Prioritize features by critical, high, medium, low | 6 |
| Identify MVP features | 4 |
| **Total** | **30** |

**Week 4 Deliverable: Project Roadmap (20%)**

**Tasks for Week 4:**

| Task | Points |
|---|---|
| Review and finalize the feature list | 5 |
| Map features to development cycles in the roadmap | 8 |
| Set milestones and timeframes for each cycle | 4 |
| Document the roadmap | 3 |
| **Total** | **20** |

**Week 4 Deliverable: GitHub Project Setup and Task Management (10%)**

**Tasks for Week 4:**

| **Task** | **Points** |
|---|---|
| Merge requirements and feature lists into GitHub project | 3 |
| Convert tasks into GitHub Issues or Projects with appropriate cycle tags | 3 |
| Set up the GitHub Project Board with columns (Backlog, In Progress, Completed) | 2 |
| Ensure proper version control with branches | 2 |
| **Total** | **10** |

# Appendix A: Student Submission Management

Since each pod is responsible for submitting markdown files and collaborating on shared tasks, it's important to have a clear process in place for managing contributions to the shared GitHub repository. This ensures that multiple students can work on different parts of the project simultaneously without overwriting each other's work or creating conflicts. Here are some best practices for managing student submissions and coordinating collaboration:

## 1. Using Branches for Each Pod

Each pod should work on their own branch in the GitHub repository to avoid conflicts when multiple groups are making changes. Here's the process:

- **Create a Branch:** Each pod creates a branch for their work using a naming convention like `pod-1-requirements`, `pod-2-roadmap`, etc.
  **Command:** `git checkout -b pod-1-requirements`
- **Commit to the Branch:** As the pod works on their task, they should commit changes to their own branch regularly to keep a history of progress.
  **Command:** `git add .` followed by `git commit -m "Initial draft of requirements"`

## 2. Pull Requests (PRs) for Submission

Once a pod is ready to submit their work, they should create a pull request (PR) to merge their branch into the main branch of the repository. This process allows for peer review and ensures that changes are coordinated without directly affecting the main codebase.

- **Create a Pull Request:** Once the pod is confident their markdown files are ready for submission, they should open a pull request on GitHub.
  **PR Title Example:** `"Pod 1: Requirements Document Submission"`
- **Assign Reviewers:** Assign at least one team member or another pod to review the PR before it is merged into the main branch.
- **Resolve Conflicts:** If there are any conflicts with other branches, GitHub will notify the pod during the PR. The pod should work with the other teams to resolve these conflicts by editing the markdown files or merging updates from the main branch into their working branch.

## 3. Coordinating Merges and Avoiding Conflicts

- **Pull Before Push:** Before pushing any changes, each pod should always pull the latest updates from the main branch to avoid conflicts.
  **Command:** `git pull origin main`
- **Rebasing if Necessary:** In the event of conflicts, pods can use rebasing to incorporate changes from the main branch into their branch without creating merge commits.
  **Command:** `git rebase main`

## 4. Best Practices for Markdown File Edits

- **Work on Separate Files:** To reduce the chances of conflicts, each pod should work on their designated files (e.g., `requirements-pod-1.md`, `roadmap-pod-2.md`). Avoid editing the same files as other pods unless necessary.
- **Clear Commit Messages:** Use descriptive commit messages that explain what changes were made, so team members and reviewers can easily track progress. **Example:** `git commit -m "Added user stories for the faculty role"`

## 5. Regular Updates and Communication

- **Frequent Merges:** Pods should submit and merge changes frequently, rather than waiting until all work is complete. This minimizes the risk of conflicts and ensures that the repository stays up-to-date.
- **Team Communication:** Use GitHub Issues or a team communication tool (like Discord) to coordinate submissions, discuss merge conflicts, and plan collaborative efforts across pods. Ensure that everyone knows when a major merge is happening and if any dependencies between pods exist.

By following these practices, you will avoid merge conflicts, maintain a clean commit history, and ensure a smooth collaborative workflow. Encouraging the use of pull requests, branching, and clear communication will help foster effective collaboration and improve project management skills.

# Appendix B: Risk Management and Mitigation Plan

Once the roadmap has been developed, it's important to proactively address potential risks that could affect project success. This section outlines the major risks identified during planning and provides mitigation strategies for handling them.

## 1. Risk Identification

Each pod should identify risks that could impact their development based on the roadmap and assigned tasks. Common risks may include:

- **Technical Challenges**: Unfamiliarity with technologies or dependencies on external APIs or libraries.
- **Team Coordination Issues**: Lack of communication or difficulties coordinating task handovers between pods.
- **Scope Creep**: Uncontrolled expansion of features or changes in requirements that could delay delivery.
- **Timeline Delays**: Tasks taking longer than estimated due to unforeseen difficulties.
- **System Integration Risks**: Challenges arising from integrating subsystems or shared components, such as the Leap Motion controller or the authentication system.

## 2. Mitigation Strategies

For each identified risk, mitigation strategies should be outlined. This includes:

- **Technical Challenges**:
  - Conduct research and prototype key components early to identify possible technical issues.
  - Plan dedicated time for learning new tools or frameworks.
- **Team Coordination Issues**:
  - Implement clear communication channels (e.g., Slack, Discord).
  - Set regular stand-up meetings to align tasks and dependencies across pods.
- **Scope Creep**:
  - Define a clear scope during the roadmap development and avoid adding new features without approval from all stakeholders.
  - Prioritize features according to the MVP and high-priority list; additional features can be scheduled for later cycles.

- **Timeline Delays**:
  - Build in buffer time for critical tasks, and ensure high-priority features are started early.
  - Regularly review task progress on the GitHub project board to detect early signs of delays.
- **System Integration Risks**:
  - Ensure shared systems (e.g., authentication, Leap Motion) are stabilized early to avoid bottlenecks.
  - Pods working on user-specific systems should plan mock data integration to avoid being blocked by incomplete shared components.

## 3. Contingency Plans

For critical risks, contingency plans should be defined, which provide backup actions if the risk materializes:

- **Example**: If a technical challenge with Leap Motion arises and the required features cannot be implemented, the contingency plan may involve switching to an alternative input system (e.g., keyboard controls) while continuing to troubleshoot the Leap Motion integration in parallel.

By proactively addressing these risks, the project can proceed more smoothly, with predefined responses for potential issues. This planning also prepares the team for real-world challenges and ensures timely progress throughout the cycles.

# Appendix C: Testing Strategy

A well-defined testing strategy will help identify and fix bugs early in the development process. In addition, testing strategies help establish a contract between pods who may depend on data between each other, assigning accountability and standards for mocking data that does not yet exist. This ensures that each team can rely on predefined inputs and outputs, facilitating seamless integration. The following testing methods should be incorporated into the capstone project to ensure robustness and reliability.

## 1. Unit Testing

Unit tests focus on verifying the functionality of individual components or functions in isolation. These tests should be automated and executed frequently as part of the development workflow. **Important:** Each pod should create the unit tests for their assigned task before starting the implementation. Sharing these unit tests with other pods ensures that other teams can mock data and interactions appropriately, supporting seamless integration between different tasks.

- **Objective:** Validate that individual functions or modules perform as expected.
- **Tools:** Use Python's built-in `unittest` library, `pytest`, or equivalent testing frameworks in other languages.
- **Example:** For the `Authentication System`, a unit test should verify that the login function returns a valid user session upon providing correct credentials.

## 2. Integration Testing

Integration testing ensures that individual modules or subsystems work together correctly. These tests should cover the interactions between user-specific systems (e.g., Tutor System, Student System) and shared components (e.g., Authentication, Leap Motion).

- **Objective:** Test that combined components function properly when integrated.
- **Tools:** Use `pytest`, `unittest`, or integration testing libraries suitable for the project's framework.
- **Example:** Test that user authentication works correctly with the Tutor Dashboard, ensuring that only authenticated users can access tutor schedules.

## 3. User Acceptance Testing (UAT)

UAT ensures that the system meets stakeholder requirements and provides a satisfactory user experience. This type of testing should involve actual stakeholders or representatives simulating real-world use cases.

- **Objective:** Validate that the system meets business and user requirements.
- **Stakeholders Involvement:** Conduct tests with stakeholders like students, tutors, or faculty members to gather feedback on functionality.
- **Example:** Verify that students can search for professor office hours and access course materials using the kiosk system.

## 4. Continuous Integration/Continuous Deployment (CI/CD)

CI/CD automates the process of testing and deploying the system whenever new code is pushed to the repository. It ensures that changes are tested frequently and that deployments are automated.

- **Objective:** Automatically run tests and deploy code to production environments on each commit or pull request.
- **Tools:** Use GitHub Actions for automated testing and deployment. Set up a pipeline to run unit and integration tests on each pull request.
- **Example:** Set up a GitHub Actions workflow to run all tests when a new pull request is submitted, ensuring that no broken code is merged into the main branch.

## 5. Test Coverage and Regression Testing

High test coverage ensures that most of the codebase is tested and reduces the risk of undetected bugs. Regression tests help confirm that changes in the codebase do not break existing functionality.

- **Objective:** Ensure that at least 80% of the codebase is covered by tests, and run regression tests after significant updates.
- **Tools:** Use test coverage tools like `coverage.py` for Python or similar tools for other languages to measure test coverage.
- **Regression Testing:** Develop a suite of tests that run after every major feature or bug fix to ensure no new bugs are introduced.
- **Example:** After integrating the Leap Motion gesture control system, run regression tests to verify that previous functionalities, such as user login and dashboard navigation, still work as expected.

## 6. Testing Plan

Each pod should develop a detailed testing plan, outlining:

- Specific components or features to be tested.
- Tools and frameworks to be used.
- Expected test coverage percentage.
- How test results will be reported and reviewed.

A well-defined testing strategy will improve code quality, reduce bugs, and ensure that the final product meets user expectations.

# Appendix E: Deployment Plan

In the context of this project, the deployment will involve Raspberry Pis, both as client stations and as a central server. This section outlines the hosting environment, setup, release management, and automation processes for deployment.

## 1. Hosting Environment

The system will consist of two types of Raspberry Pis:

- **Client Stations:** Raspberry Pis connected to TV screens and equipped with Leap Motion sensors, serving as touchless kiosk clients. These clients will interact with users and run localized versions of the kiosk software.
- **Server:** A centralized Raspberry Pi that serves as the backend for the client stations, responsible for managing client updates and configuration. The server will host a web portal that administrators can use to update, monitor, and control client stations remotely.

## 2. Environment Setup

The deployment environments will be structured as follows:

- **Development Environment:** Individual pods will set up Raspberry Pi devices to test and develop features locally. This includes setting up the Leap Motion software and configuring the kiosk software on the client Raspberry Pis.
- **Staging Environment:** A staging environment will be established on a Raspberry Pi, simulating the final production setup. This will allow testing of the client-server interactions and ensure that the web portal can communicate with client stations effectively.
- **Production Environment:** Once the system is ready for deployment, the production environment will be set up with Raspberry Pi clients distributed across locations and the central server accessible via a secure web portal.

## 3. Release Management

The following practices will be adopted to ensure smooth deployment and maintenance.:

- **Versioning:** Use a versioning system (e.g., Semantic Versioning) to manage different releases of the software. This allows for tracking changes and rolling back if issues arise during deployment.
- **Release Pipeline:** Define a release pipeline that includes steps for pushing updates from development to staging and then to production. Each step should involve automated tests and manual verification to ensure stability before deployment.

## 4. Automated Deployment Tools

Automation will play a key role in managing the deployment process across Raspberry Pi devices. The following tools and processes will be used:

- **Ansible or similar tools:** Automate the deployment and configuration of software across multiple Raspberry Pis, allowing the central server to push updates to all client stations.
- **Docker (Optional):** Use Docker to containerize the kiosk software for consistent deployment across all Raspberry Pis. This can simplify the setup process by ensuring the same environment is deployed across all devices.

By defining a clear deployment plan, we can ensure the system is deployed efficiently and reliably across the Raspberry Pi client stations and the server.

# Appendix F: System Architecture and Design Documentation

In addition to the roadmap and feature breakdown, it is necessary to document the system architecture and design. This documentation will provide a clear understanding of how the system is structured, how components interact, and how data flows through the system. Proper design documentation ensures that future teams and stakeholders can easily understand, maintain, and extend the system. The following types of design documentation should be included in the final submission:

## 1. Architecture Diagrams

Architecture diagrams should provide an overview of the system's components and how they interact. These diagrams might include:

- **UML Diagrams:** Unified Modeling Language diagrams such as component diagrams, deployment diagrams, or system architecture overviews showing the interaction between Raspberry Pi clients and the central server.
- **Class Diagrams:** Illustrate the key classes, their attributes, and relationships, especially for modules like authentication or user management.
- **Sequence Diagrams:** Depict how different components or systems interact with each other during specific processes, such as user login or data retrieval between the kiosk and the server.

## 2. API Design

If your system exposes APIs (e.g., to manage data between the Raspberry Pi server and clients), it is crucial to document these APIs:

- **RESTful APIs or GraphQL:** Provide details on the available endpoints, the HTTP methods (GET, POST, PUT, DELETE), expected inputs (parameters, headers), and outputs (JSON responses).
- **Authentication and Security:** If authentication is required (e.g., token-based authentication), include details on how security is handled and how different roles (admin, user) are authorized.

## 3. Database Schema Design

If the system involves data storage, a clear schema design is required:

- **ER Diagrams:** Entity-Relationship diagrams showing the database structure, relationships between tables, and primary/foreign keys.
- **Table Structure:** Describe each table in the database, its fields, and data types, along with any constraints (e.g., unique fields, foreign key constraints).
- **Data Flow Diagrams:** Show how data moves through the system, from input (e.g., user interactions at the kiosks) to storage and processing (e.g., in the Raspberry Pi server).

## 4. Interaction Flow Diagrams

Document the interaction flows that represent how users will interact with the system. These could include:

- **User Journey Maps:** Illustrate typical use cases for the system, showing how users (students, faculty, admin) interact with different parts of the system (e.g., accessing information from the kiosk, using Leap Motion for gesture control).
- **State Diagrams:** Show the different states of the system or subsystems (e.g., logged-in vs. logged-out states, idle vs. active states in the kiosk).
- **UI Wireframes:** Include wireframes or mockups of the user interface for the kiosk, showing what users will see at various stages (e.g., login screen, content browsing, search functionality).

## 5. Design Rationale

Each design decision should be accompanied by a rationale that explains:

- Why a particular design pattern, architecture, or tool was chosen (e.g., using RESTful APIs for simplicity, opting for a relational database for structured data storage).
- The pros and cons of alternative approaches considered.
- How the design addresses scalability, maintainability, and security concerns.

Properly documenting the system architecture and design will not only help in the current project cycle but also provide future teams and external stakeholders with a comprehensive understanding of the system. This documentation should be maintained and updated as the system evolves.

# Appendix G: Team Roles and Responsibilities

To simulate a professional software development environment, each pod will rotate through key roles, both on a pod level and an individual team member level. Pods will take on project-wide responsibilities like Scrum Master and Project Manager, while individual team members will rotate between roles such as Tech Lead, Tester, and Documentation Specialist. This structure fosters collaboration, leadership, accountability, and a comprehensive understanding of various roles within a project team.

## 1. Pod-based Roles: Scrum Master and Project Manager (Rotating Weekly by Pod)

Each pod will rotate the responsibility of Scrum Master and Project Manager weekly. During their assigned week, the pod will manage sprint planning, conduct meetings, track progress, and ensure deadlines are met. These roles are distinct but complementary and help the pod gain experience in both team coordination and project management.

**Scrum Master:**

The Scrum Master focuses on facilitating the pod's work and ensuring that the sprint runs smoothly. This role is centered on enabling productivity by removing obstacles and fostering collaboration within the pod.

- **Responsibilities:**
  - Facilitate daily stand-ups and sprint planning meetings.
  - Manage the sprint backlog and ensure tasks are being actively worked on.
  - Resolve blockers preventing the pod from moving forward with their tasks.
  - Encourage communication within the pod to ensure clarity on tasks and goals.
  - Ensure that sprint goals are clear and achievable.

**Project Manager:**

The Project Manager focuses on the bigger picture, ensuring the pod adheres to deadlines, meets project milestones, and manages overall progress toward project completion. This role involves broader responsibility for the entire project timeline and coordination across pods.

- **Responsibilities:**
  - Oversee the project roadmap and ensure that key milestones are met.
  - Coordinate between pods to address dependencies and ensure smooth integration across teams.
  - Monitor the pod's overall progress and report to the instructor if the project is at risk of falling behind.

  - Track deliverables, ensuring all pull requests and tasks are submitted on time.
  - Ensure version control standards are followed when merging code to the main branch.

## 2. Member-based Roles: Tech Lead, Tester, and Documentation Specialist (Rotating Per Task)

Each task will have individual roles assigned to team members. These roles will rotate per task to ensure that every team member gains experience in leading technical development, ensuring quality through testing, and maintaining comprehensive documentation.

**Tech Lead:**

The Tech Lead is responsible for overseeing the technical direction of the task and ensuring that code development aligns with the project's goals. This role rotates for each task, allowing every team member to experience leading the development process.

- **Responsibilities:**
  - Lead the development of the task and make technical decisions.
  - Review code and ensure it meets the project's standards.
  - Provide technical guidance to other team members.
  - Conduct code reviews before pull requests are merged.

**Tester:**

The Tester ensures that each task is adequately tested, including unit, integration, and regression tests. This role rotates per task, ensuring all team members contribute to maintaining quality assurance across the project.

- **Responsibilities:**
  - Develop and run unit tests, integration tests, and regression tests for the task.
  - Collaborate with other pods by sharing mock data and ensuring that systems integrate properly.
  - Report any bugs or issues found during testing.

**Documentation Specialist:**

The Documentation Specialist ensures that all relevant project documentation is up to date and that the task is properly documented for future reference. This role rotates per task, giving each team member a chance to ensure that the project is well-documented and understandable for future teams or stakeholders.

- **Responsibilities:**

- Write clear and concise documentation for the task, including API documentation, system architecture, and any relevant meeting notes.
- Update the shared GitHub repository with relevant documentation.
- Ensure that all technical decisions are recorded and that documentation is easily accessible for future development.

## 3. Role Rotation Per Task and Week

The following rotations ensure that every team member and pod gains experience in various responsibilities:

- **Tech Lead, Tester, and Documentation Specialist:** Rotate per task, ensuring that every team member takes on all roles at different points.
- **Scrum Master and Project Manager:** Rotate weekly by pod. Each week, one pod will act as both Scrum Master and Project Manager, facilitating team-wide coordination and project governance.

This dynamic structure provides all members with valuable experience across multiple roles, preparing them for real-world software engineering teams where flexibility, collaboration, and role-based leadership are essential.