CS.025 Requirements Document

OPEN RESPONSE - OSU CS SENIOR CAPSTONE PROJECT

AUTHORS

Evan Baumann - 934-076-910
Sean Gibson - 934-356-114
Sage Morgillo - 934-070-648
Tran Cong Son Nguyen - 934-529-215
Nathan Rumsey - 934-129-273
Nathaniel Wood - 934-143-984

2025-02-16

Change Log

Table 1: Winter, 2025 Change Log

Date	Version	Change Description	Reason for Change	Author/Initiator
11/10/2024	1.0	Initial version of the Requirements Document.	Enumerate project functional and non-functional requirements, scope, and timeline.	Team
02/16/2025	1.1	Converted document from Word to LaTeX.	Standardized formatting for professionalism and easier maintenance.	Nathan Rumsey
02/16/2025	1.2	Renamed "Verification and Validation/Documentation and Reporting" to "Verification and Validation/-Validation Reporting."	Improved clarity and alignment with standard terminology.	Nathan Rumsey
02/16/2025	1.3	Minor formatting adjustments to enhance LaTeX presentation.	Improved readability and consistency with the document style.	Nathan Rumsey
02/16/2025	1.4	Adjusted language in documentation discussions to be less imprecise and reduce word count.	Balanced the wording to maintain clarity while outlining expectations and reducing word count.	Nathan Rumsey
02/16/2025	1.5	Removed "Feature Satisfaction" as a criterion for user testing in Section 3.2 Verification and Validation.	Avoid redundancy, as it was already covered earlier.	Nathan Rumsey
02/16/2025	1.6	Moved "user testing timeline" from Verification and Validation to Timeline/Milestones.	Section reorganization for better alignment with content structure.	Nathan Rumsey
02/16/2025	1.7	Consolidated reporting descriptions within Verification and Validation.	Reduced unnecessary word count while maintaining essential details.	Nathan Rumsey
02/16/2025	2.0	Removed Canvas API from requirements.	No longer relevant to project scope.	Team

Contents

1 Project Overview

1.1 Purpose and Objectives

The purpose of this project is to develop an **open-source classroom polling software** that provides a free, accessible alternative to commercial solutions like Top Hat and Poll Everywhere, which often come with high costs (up to $60 per term) that can restrict equitable access for students. By eliminating these costs, this software aims to enhance student participation and engagement in real-time, interactive classroom polling.

The primary objective is to create a user-friendly and responsive platform that allows **instructors to conduct live polling** during lectures. This enables educators to gauge students' understanding and adjust their teaching dynamically, fostering a more interactive and adaptable learning environment.

Specifically, the project will focus on developing essential user-interface features to facilitate live polling, support multiple question formats, provide automatic grading, and offer real-time feedback for instructors. The platform will also incorporate key features such as account creation for various roles (administrators, instructors, and students), course and lecture management, and live in-classroom polling.

This project builds on two years of prior work and will leverage web development frameworks and real-time data processing to support these functionalities. Additionally, it will be developed with scalability, data security, and accessibility in mind, adhering to standards such as FERPA (Family Educational Rights and Privacy Act) to ensure it securely manages student information.

By providing a free, flexible, and open-source solution, this software not only reduces the financial burden on students but also offers greater flexibility in polling formats compared to available commercial solutions.

1.2 Target Audience and Stakeholders

1.2.1 Primary Users

- **Students**: Students require the software to be easy to understand, set up, and able to record responses in a way that feels accurate to their needs. The software enables students to participate in live classroom polls, providing them with an interactive and engaging learning experience. By responding to polls in real time, students can check their understanding of the lecture content and actively engage with course materials.
- **Instructors**: Instructors need the software to be able to create polls, gather real-time feedback on students' understanding, and use the software to adjust their teaching strategies accordingly.

They need to be able to access live responses and view aggregated data, allowing for quick assessment and insights into class performance. Instructors also need quality-of-life features such as automatic grading of responses and an interface that integrates with existing Learning Management Systems (LMS), such as Canvas, for seamless tracking of grades and student engagement.

- **Educational Institutions**: Educational institutions need polling software that supports student engagement and improved learning outcomes while remaining affordable. They prioritize tools that integrate with existing LMS systems and provide interactive features without increasing financial burdens on students. Cost-effectiveness is key, as institutions aim to offer valuable, engaging learning resources that align with their budget constraints and ensure accessibility for all students.

1.2.2 Stakeholders

- **Project Partner**: Dr. Evan Thatcher, a physics professor at Oregon State University is the primary stakeholder and project sponsor. He is responsible for overseeing the project's direction and progress. Dr.Thatcher will be updated regularly on the status of the project and can influence key design and feature prioritization decisions.
- **Capstone Instructors**: Kirsten Winters and Alex Ulbrich are indirectly involved as they oversee the project from an educational standpoint, providing guidance to the development team but not making design or architectural decisions.
- **Project TA**: Ethan Vrhel is the project TA and is closely involved in tracking the team's progress and requires weekly updates to ensure that the project remains on schedule and aligned with objectives. Ethan does not make design or architectural decisions but plays a role in coordinating project activities and providing feedback.

1.3 Success Criteria

Success for this project will be measured by evaluating the software against defined criteria that ensure it meets user needs, performs reliably, and aligns with project goals. These criteria will provide a clear standard for determining the project's effectiveness and readiness for deployment in educational environments.

Key Success Metrics

- **Core Functionality and Feature Completeness**: All essential features, such as user authentication, poll creation, real-time response collection, and result viewing, must be fully implemented, functional, and user-tested. Feedback from user testing should indicate high levels of satisfaction with the functionality and ease of use of each feature, ensuring the software meets the needs of both students and instructors.

- **User Feedback and Usability**: Success will be assessed through surveys and structured usability testing, evaluating dimensions such as ease of use, feature satisfaction, interface clarity, navigation, and overall design. Positive user feedback across these dimensions will confirm that the software is intuitive and accessible. Validation will confirm that the UI is user-friendly, with intuitive workflows, a responsive design, and clear visual hierarchies that help guide users through core tasks.
- **Performance and Stability Benchmarks**:
  - **Code Coverage**: High code coverage (80% or above) across unit and integration tests will be targeted to ensure comprehensive testing, validating the software's stability and robustness.
  - **Response Time**: The software should achieve response times of less than 500 milliseconds per action, meeting performance standards for real-time classroom interaction.
  - **Uptime**: The platform must meet an uptime target of 99.9%, demonstrating that it is reliable enough for uninterrupted classroom use.
- **WCAG 2.1 Level AA Compliance**: Automated and manual accessibility testing will validate that the platform meets these accessibility standards, ensuring an inclusive experience for all users, including those with disabilities.
- **Cost-Effectiveness**: Given that the software is free and open-source, cost-effectiveness will be evaluated by comparing it to existing paid alternatives. Success in this metric will be measured by the financial savings it offers to students and institutions.

1.4 Competitive Analysis

In the landscape of classroom polling tools, several platforms offer varying features, each with its own strengths and limitations. Our goal is to develop a free, open-source solution that not only matches but surpasses existing offerings, particularly in areas like Learning Management System (LMS) integration and advanced functionalities.

Competitive Classroom Polling Solutions

1. **Top Hat**: A comprehensive teaching platform that leverages student devices to increase in-class engagement and real-time feedback. It offers a wide range of interactive features, including polls, quizzes, discussions, and attendance tracking. Integrates with various LMS platforms and provides customizable content. However, it is a paid service, which may be a barrier for some institutions or students.
2. **Poll Everywhere**: A real-time polling tool facilitating immediate feedback and assessment in classrooms. It offers diverse question types, provides instant feedback, and integrates with

platforms like Canvas and PowerPoint. However, its free plan restricts the number of responses to polls, and the platform is not open-source.

3. **Canvas Polls**: A polling feature within the Canvas LMS that allows instructors to create and administer real-time polls. It seamlessly integrates with Canvas, is user-friendly, and open-source. However, it is exclusive to Canvas users and lacks the comprehensive features found in standalone polling tools.

4. **Google Forms**: A free, web-based tool from Google for creating surveys, quizzes, and polls. It integrates with Google Drive, is versatile, and easy to use. However, it lacks advanced features, does not integrate with LMS platforms, and managing responses can be challenging in large classes

5. **Mentimeter**: An interactive presentation tool enabling real-time audience engagement through various question formats. It delivers engaging visuals and a variety of question types. However, the free version limits the number of questions per poll, is not specifically designed for classroom use, and is not open-source.

6. **Kahoot!**: A game-based learning platform widely adopted in educational settings for creating interactive quizzes and polls. It offers a highly engaging interface, user-friendly design, and broad recognition among educators and students. However, it lacks LMS integration, limited collaborative features, and is not open-source.

The current market for classroom polling tools presents a spectrum of options, each with distinct advantages and drawbacks. While platforms like Kahoot! and Mentimeter excel in engagement, they often lack LMS integration and are not open-source. Poll Everywhere and Top Hat offer robust features and LMS compatibility but come with associated costs. Canvas Polls and Google Forms provide free solutions but are limited in scope and flexibility.

This analysis highlights an opportunity to develop a free, open-source classroom polling tool that combines the strengths of existing platforms, such as user engagement, diverse question types, and LMS integration, while addressing their limitations. By focusing on accessibility, comprehensive features, and seamless integration, our project aims to fill the gap in the current market, offering educators and students an effective and cost-effective solution for learning.

2 Scope and Requirements

2.1 Core Features

This project is designed to provide an accessible and efficient classroom polling tool. The functionalities focus on creating an intuitive interface that simplifies workflows for both instructors and students. By supporting real-time feedback, flexible question formats, and structured class and session management, this software will serve as a resource for improving classroom engagement and streamlining grading at a lower cost for both instructors and students.

1. **User Accounts and Roles**
    - **Account Creation and Management**: Allow students and instructors to create, manage, and delete their accounts securely.
    - **Roles-based Access Control**: Define specific permissions based on the user's roles. For example, only instructors should be able to create classes and post questions.
2. **Class Management**
    - **Create and Manage Classes**: Instructors should have the ability to create classes and manage details such as class names, descriptions, and settings.
    - **Class Enrollment**: Students should be able to join classes through a unique class code or invite link provided by the instructor.
    - **Student Roster**: Instructors should have access to a roster displaying all students enrolled in a class.
3. **Lecture and Session Management**
    - **Create and Organize Lectures**: Instructors should be able to create lectures within classes, each with a unique name, date, and associated questions.
    - **Session Scheduling and Management**: Lectures should support scheduling features for live sessions, as well as saving lecture sessions for asynchronous participation.
4. **Question Creation and Management**
    - **Question Types**: Instructors should be able to create multiple question types (e.g., multiple-choice, true/false, short answer, ranking, polling).
    - **Question Templates and Bank**: Provided templates for common question formats as well as the option to save questions to a reusable question bank for easy reuse in future lectures or classes.
5. **Student Interaction and Engagement**

- **Live Polling and Answer**: Students should be able to respond to questions in real-time and during live sessions.
- **Answer History**: Students should be able to view their answer history, both within and outside live sessions, for self-assessment.
- **Participation Tracking**: The system should track student participation for each lecture or class section.

6. **Real-Time Support**

   - **Live Mode with Real-Time Updates**: Use Socket.io for real-time updates to deliver instant feedback during polling sessions.

7. **Canvas Support**

   - **Grade Syncing**: Instructors should be able to sync grades or participation records from the polling software to Canvas using CSV import functionality.
   - **Student Data Import**: For ease of use, instructors should be able to import student data directly from Canvas, reducing manual data entry.

2.2 Use Cases and Workflows

The following use cases and workflows detail how instructors and students will interact with the software to achieve their objectives efficiently. Each scenario provides step-by-step guidance on key actions, from account creation and class management to live polling and data analysis. These workflows highlight the core functions of the platform, showcasing how it enhances both teaching and learning experiences through intuitive design and functionality.

1. **Account Creation and Onboarding**

   - **Use Case**: A new user, either a student or instructor, visits the platform to create an account. They select their role upon registration, which directs their access and functionalities within the software. Instructors can create classes, while students join classes using a unique code provided by their instructor.
   - **Workflow**: User accesses the registration page $\rightarrow$ Creates an account $\rightarrow$ Selects role (instructor or student) $\rightarrow$ Instructor creates a new class or student joins an existing class with a code $\rightarrow$ Confirmation email sent (if applicable).

2. **Creating and Managing Classes**

   - **Use Case**: An instructor logs in to the platform to set up a new class for an upcoming course, providing essential details and allowing students to enroll.

- **Workflow**: Instructor logs in → Navigates to "Create Class" → Enters course details (e.g., class name, description) → Saves class setup and shares the class code with students.

3. **Creating a Lecture and Adding Questions**
   - **Use Case**: Prior to a lecture, an instructor prepares polling questions to facilitate interactive classroom engagement.
   - **Workflow**: Instructor accesses class dashboard → Creates a new lecture within the class → Adds polling questions by selecting question templates (e.g., multiple-choice, short answer) → Saves lecture setup with questions for the session.
4. **Student Participation in Live Polling**
   - **Use Case**: During a live lecture, students join the session, view questions, and respond in real-time.
   - **Workflow**: Instructor starts a live session → Students join via session link → Students answer questions → Responses update in real-time on the instructor's dashboard.
5. **Data Analysis and Instructor Feedback**
   - **Use Case**: After the session, instructors review class participation and response data to identify knowledge gaps or challenging areas for students.
   - **Workflow**: Instructor logs in → Accesses class dashboard and session metrics → Reviews participation rates, question success rates, and identifies trends or students requiring further support.
6. **Canvas Support for Grade Uploads**
   - **Use Case**: Post-lecture, the instructor downloads CSV data to update grades and ensure accurate attendance records in Canvas.
   - **Workflow**: Instructor navigates to "Export Gradebook" → Selects section data for gradebook export → Initiates export → Confirmation of successful export displayed, CSV file will be downloaded.

2.3 Functional Requirements

The functional requirements define the essential inputs, outputs, and processes that enable the software to meet its core objectives. These requirements ensure that both students and instructors can seamlessly interact with the platform to create, participate in, and analyze classroom polling. Each functional requirement is given a ordered from top to bottom with decreasing priority within the scope of this year's development (for example, it is critical account management functionality is present, but not as critical that integration support is fully featured; basic functionality is still required).

1. **Inputs**
   - **User Data:** Collect and manage basic information, such as usernames, email addresses, and passwords, from students and instructors during account creation.
   - **Class and Lecture Details:** Instructors input class information, lecture schedules, and topics for creating and managing class sessions. Each class includes a unique code for student enrollment.
   - **Polling Questions and Responses:** Instructors add polling questions to lecture sessions, selecting from various question types (e.g., multiple choice, true/false). Students submit their responses in real-time during polling sessions.
   - **Integration Support:** For Canvas integration, a feature to download gradebook content from the polling software for easy importing into the Canvas LMS should be offered.
2. **Processes**
   - **Account Management:** Users register for an account, select their role (student or instructor), and confirm their email address. Instructors create classes that students join using a unique class code.
   - **Poll Creation and Management:** Instructors create and manage polls for each lecture, preparing questions and saving them for live sessions. Polls can be edited, archived, or deleted based on instructional needs.
   - **Live Polling and Real-Time Interaction:** During lectures, instructors initiate live polling sessions. Students access the session via a link, view questions, and submit responses, which update in real-time on the instructor's dashboard.
   - **Data Analysis and Feedback:** After the session, instructors can review participation metrics and individual question results. This allows them to gauge student comprehension, identify trends, and address learning gaps.
   - **Grade Export with Canvas Support:** Following each lecture, instructors should be able to export Gradebook and Attendance records, enabling them to update student grades and attendance records directly in their LMS solution of choice (Canvas).
3. **Outputs**
   - **Poll Results Dashboard:** Real-time display of student responses for each question, visible to instructors during the polling session.
   - **Post-Lecture Data Reports:** Analytical reports that include participation rates, average scores, and individual question success rates, enabling instructors to assess overall class performance.

2.4 Non-Functional Requirements

These non-functional requirements cover the platform's performance, scalability, reliability, security, usability, and accessibility, each essential to delivering an effective tool for classroom interaction.

1. **Performance:**
   - The platform must maintain fast response times, especially during live polling sessions. Actions such as creating a poll or submitting a response should have a response time of less than 500 milliseconds to support seamless real-time interaction.
   - Quick loading times and low latency are essential, particularly during high-traffic periods, to prevent interruptions and keep sessions engaging.
2. **Scalability:**
   - The software should be designed to handle high concurrency, supporting hundreds of users in a single class session and multiple active sessions across the platform without compromising performance.
   - Efficient data management is required to handle increased loads and data volumes, ensuring smooth operation as the user base grows and as simultaneous polling sessions occur.
3. **Reliability and Availability:**
   - To ensure consistent service, the platform should aim for 99.9% uptime, minimizing downtime during active classroom sessions. This includes implementing load balancing, failover mechanisms, and redundant server configurations.
   - The software should be resilient to server outages, ensuring that any disruptions are brief and do not interfere with classroom activities.
4. **Security:**
   - Sensitive data, such as student and instructor login information and class details, must be securely encrypted both in transit and at rest, using SSL/TLS for data transmission.
   - Role-based access control (RBAC) will limit data visibility and functionality based on user roles, ensuring only authorized users can access or modify information.
   - FERPA compliance is essential, particularly when handling student records and syncing data with the Canvas LMS. Security measures must align with educational privacy regulations to protect student data.
5. **Usability:**

- The user interface must be intuitive, minimizing the learning curve for both students and instructors. Navigation should be simple, with clearly labeled buttons, forms, and prompts that allow users to complete tasks without extensive guidance.
- Desktop optimization is required, as many users will access the platform through institutional computers. The platform should be user-friendly and responsive, providing an efficient experience across different devices.

6. **Accessibility:**

   - The platform must comply with WCAG 2.1 Level AA standards to ensure it is accessible to users with disabilities. This includes features such as high-contrast mode, screen reader compatibility, keyboard navigation, and customizable interface elements.
   - Accessibility compliance will make the platform inclusive, allowing individuals with various needs and abilities to use the software effectively.

2.5 Integrations and Dependencies

The project relies on key integrations and dependencies to support core functionality, real-time interaction, and data management. Each integration is selected to enhance the user experience and meet scalability, security, and usability requirements.

1. **Canvas API**: Integration with the Canvas CVS inport/export will enable seamless grade transfer, attendance tracking, and class updates directly within the Canvas LMS.

2. **Socket.io**: Socket.io will be used to facilitate real-time communication between the server and client applications. This enables instant polling updates, live feedback, and push notifications, ensuring a responsive, interactive experience during classroom sessions. Socket.io is essential for maintaining the real-time capabilities that are critical to the platform's functionality.

3. **Database System**: A robust and scalable database is required to manage user data, class information, polls, and responses. Currently, MySQL has been chosen for its ability to handle high read/write loads efficiently, making it suitable for concurrent users in a classroom environment. Integrated within OSU's infrastructure, MySQL is configured to support the target maximum of approximately 200 users per session, ensuring reliable performance and efficient data management.

4. **Legacy Constraints**: As a newly developed platform, there are no legacy system constraints affecting this project. However, the platform should be flexible enough to allow users to integrate with other grading systems in the future, should additional support be required. This flexibility will enhance the platform's adaptability for different institutional needs.

5. **Authentication**: A secure, role-based authentication system will be necessary to manage user access and permissions. This will ensure that sensitive data, such as student grades and responses, is protected, and that only authorized users (e.g., instructors, students) can access specific functionality based on their roles.

2.6 Assumptions and Constraints

The development and deployment of the classroom polling software are guided by key assumptions about technology, user behavior, and available resources, as well as certain constraints related to technology, time, and budget. Recognizing these factors provides a realistic framework within which the software will be developed, ensuring that it remains focused, feasible, and aligned with project goals.

Assumptions:

1. **Technology Assumptions**:
   - **API Access**: Consistent access to the required APIs, including Canvas and Socket.io, is assumed to enable real-time data updates and seamless LMS integration.
   - **Platform Compatibility**: The software will be optimized for modern web browsers and responsive across devices, prioritizing desktop use.
   - **User Proficiency**: Users (instructors and students) will have a basic level of technical proficiency and reliable internet access to support optimal functionality.
2. **User Behavior Assumptions**:
   - **Instructor and Student Roles**: Instructors are assumed to manage class setup, poll creation, and grading, while students will primarily interact through class enrollment, lecture participation, and answering polls.
   - **User Engagement**: Users are expected to understand and follow a simple login process and navigate the polling software effectively.
3. **Resource Availability**:
   - **Development Resources**: The project will have access to full-stack development expertise and OSU's LMS testing environment, enabling end-to-end development and testing.
   - **CI/CD Tools**: Continuous integration and deployment tools will be available for automated testing and deployment to streamline the development process.

Constraints:

1. **Technology Copnstraints**:

- **LMS Integration**: The software must integrate smoothly with OSU's existing Canvas infrastructure without requiring major changes or extensive customizations.
- **Technology Compatability**: All chosen technologies and frameworks should be compatible with OSU's current IT capabilities and avoid proprietary or costly tools to support open-source development and broad accessibility.

2. **Time and Budget Constraints**:
   - **Project Timeline**: Development is constrained by the academic calendar, necessitating incremental deliverables that align with key academic milestones.
   - **Budget Limitations**: The project has a limited budget, prioritizing open-source or institutionally licensed tools to minimize costs. The database is also budget-limited, so the system should initially support a modest user load.
3. **Scalability and Future Growth Constraints**:
   - **Design for Scalability**: The software should be built with scalability in mind to support potential expansion to more institutions or larger classes.
   - **Interoperability**: The design should allow flexibility for integration with other grading systems, assuming that individuals configuring external API connections have expertise in their respective systems.

2.7 Risks and Mitigation

Identifying and addressing risks early in the project is critical for ensuring a stable, functional, and secure software solution. The following section categorizes potential risks, mitigation strategies, and assesses risks using the Rumsfeld Matrix to ensure comprehensive coverage of both anticipated and unforeseen challenges.

1. **Integration Challenges with Canvas API**:
   - **Risk**: Issues with API compatibility, rate limits, or limited Canvas API access could hinder Learning Management System (LMS) integration, especially when attempting real-time grade updates and syncing student information.
   - **Mitigation**:
     - Conduct early testing in the Canvas API sandbox to validate workflows, identify compatibility issues, and ensure data synchronization works as intended.
     - Allocate buffer time in the project schedule specifically for handling API-related challenges, allowing for adjustments if compatibility issues arise.
     - Design integration with modularity in mind, enabling alternative approaches if certain API features are limited or unavailable.

- **Risk Type**: Known Unknown; potential issues are known, but the specific API limitations or bugs may only become clear during testing.

Real-Time Responsiveness and Scalability:

- **Risk**: Real-time polling using socket connections may face performance bottlenecks, particularly with large classes or multiple simultaneous sessions, potentially leading to lag, dropped connections, or slower response times.
- **Mitigation**:
  - Optimize socket connections for efficient data transfer by minimizing data payloads, using lightweight data formats, and fine-tuning connection settings.
  - Conduct extensive load testing to simulate large class sizes and multiple active sessions, identifying bottlenecks before deployment.
  - Prepare for horizontal scaling of the socket server, allowing for capacity expansion if high concurrent usage is anticipated.
- **Risk Type**: Known Known; the risk of scalability and performance issues is well understood, though the exact limitations may vary based on actual load.

Data Privacy and Security:

- **Risk**: Inadequate security measures could lead to unauthorized access or data breaches, exposing sensitive information such as student accounts, grades, and polling data.
- **Mitigation**:
  - Implement encryption for data in transit (using SSL/TLS) and encryption at rest for sensitive information.
  - Use role-based access control (RBAC) to restrict access based on user roles (e.g., instructors vs. students), ensuring only authorized users can view or modify sensitive data.
  - Schedule regular security audits and vulnerability assessments, allowing the team to identify and address potential weaknesses proactively.
  - Ensure compliance with FERPA guidelines to maintain data privacy, and incorporate best practices for managing and securing educational data.
- **Risk Type**: Known Known; the importance of data privacy and security is well understood; known techniques are available to mitigate the risk.

User Adoption and Usability:

- **Risk**: : If the platform is not intuitive or requires extensive onboarding, instructors and students may be reluctant to adopt it, reducing its effectiveness and limiting the project's impact.

- **Mitigation**:
  - Involve end-users early in the design process through interviews, surveys, and usability testing to gather feedback and tailor the interface to their needs.
  - Conduct usability testing throughout development to refine the user experience and minimize friction points for users.
  - Create clear documentation and tutorials for both instructors and students, ensuring they can quickly learn how to use the platform without extensive training.
- **Risk Type**: Unknown Known; the risk of user reluctance is anticipated, but specific user pain points may only be discovered after initial testing and feedback.

Dependency on External Libraries and Tools:

- **Risk**: Reliance on third-party libraries (e.g., socket.io for real-time connections, Courier for notifications) could lead to maintenance challenges if these libraries become deprecated, incompatible, or experience security vulnerabilities.
- **Mitigation**:
  - Choose well-maintained and widely used libraries that are actively supported by the community, minimizing the risk of sudden deprecation or lack of updates.
  - Maintain a minimal and modular dependency structure, allowing for the replacement of libraries if they become outdated or incompatible.
  - Regularly review and update dependencies as part of the CI/CD pipeline, ensuring the latest, most secure versions are used, and establish a contingency plan for replacing critical libraries if necessary.
- **Risk Type**: Known Unknown; while the potential for dependency issues is anticipated, specific compatibility or maintenance challenges may vary based on library updates and security vulnerabilities.

Resource Constraints (Time and Budget):

- **Risk**: Limited time and budget may restrict development, testing, and support for additional features, potentially impacting the quality or completeness of the final product.
- **Mitigation**:
  - Prioritize core features that meet immediate needs, deferring secondary features or "stretch goals" that may not be essential for initial functionality.
  - Set achievable milestones and conduct regular reviews of progress to identify areas where resources can be reallocated for maximum impact.
  - Adopt an incremental development approach, releasing early versions for feedback and iteration based on user input to maximize value within limited resources.

- **Risk Type**: Known Known; the constraints are well-defined, and mitigation relies on effective planning and prioritization.

Potential for System Downtime:

- **Risk**: Unplanned system outages could disrupt live polling sessions, impacting the user experience and potentially causing frustration among instructors and students.
- **Mitigation**:
  - Implement regular backups of data, enabling quick restoration in case of a server failure.
  - Develop failover strategies to ensure minimal downtime, such as load balancing or using a backup server during high-traffic periods.
  - Consider implementing a lightweight offline mode or cached fallback that allows students to submit answers locally if a connection is lost, which can then sync once the connection is restored.
- **Risk Type**: Unknown Known; the risk of downtime is anticipated, but specific causes may vary depending on traffic and infrastructure stability.

Unforseen Technical Challenges:

- **Risk**: Unexpected technical issues, such as compatibility problems with certain devices, hidden bugs, or unforeseen bottlenecks, could arise during development or deployment, affecting functionality or performance.
- **Mitigation**:
  - Schedule buffer time in the project timeline to allow for troubleshooting and addressing unexpected technical challenges.
  - Perform extensive testing across multiple environments and devices to catch potential compatibility issues early.
  - Maintain flexibility in the project design, allowing for adjustments to the technical stack or architecture if significant issues arise.
- **Risk Type**: Unknown Unknown; by definition, the specifics of these risks are unpredictable, but allocating time and flexibility helps mitigate potential impacts.

This risk assessment and mitigation plan provides an overview of both anticipated and unforeseen risks, with clear strategies to address each one. By proactively planning for these risks, the project aims to maintain functionality, security, scalability, and usability, ensuring a successful deployment of the classroom polling software.

2.8 Out of Scope Features

The following features and functionalities are considered out of scope for this project. These limitations are defined to maintain focus on core requirements and ensure successful delivery within the available time and resources.

1. **Non-Commercial Focus**: This project is developed as an open-source educational tool with no plans for monetization. There will be no commercialization features such as premium tiers, ads, or other profit-driven functionalities. Institutions interested in using the software may need to self-deploy and manage their own instances.
2. **Limited Targeted Audience**: The software is designed specifically for classroom environments within OSU, and it may not be adaptable for use by businesses or organizations with significantly different structures or requirements.
3. **SSO (Single Sign-On) Integration**: The software will rely on its built-in authentication service and will not include support for third-party Single Sign-On (SSO) systems. Users will create and manage accounts directly within the platform rather than using external authentication providers.
4. **Basic Analytics Only**: The software will provide essential feedback and polling metrics for instructors, but it will not include advanced analytics capabilities. There will be no predictive modeling, in-depth trend analysis, or complex data visualization options beyond basic reporting.
5. **Scalability Constraints**: The backend infrastructure is designed to support standard classroom sizes, with a maximum of a few hundred simultaneous users. The system is not intended for extremely high concurrency, such as thousands of users in a single instance, without significant changes to the underlying architecture.
6. **LMS Integration**: Integration with other external systems, including third-party grading tools or data repositories, is out of scope. This includes support for the Canvas API.
7. **Advanced Customization for Non-Educational Use Cases**: Custom features aimed at adapting the software for non-educational settings, such as corporate training or public events, are beyond the intended scope. Customization will be focused on meeting classroom-specific needs.

2.9 Technical Requirements

The infrastructure and deployment strategy for this project will be designed to ensure flexibility and adaptability, allowing it to be deployed across various environments and independent instances.

To accommodate potential future deployments by other institutions or developers, the codebase will be designed to be **deployment-agnostic**. This means the software will not be tied to a single deployment instance or CI/CD pipeline, making it easy for others to set up and maintain their own instances if they choose to fork the repository. Our focus will be on creating a flexible architecture that supports deployment on different cloud providers, such as **AWS**, **Google Cloud**, or Microsoft Azure, or **on-premises servers**.

Since the project partner has not finalized a specific deployment plan, the project will be set up to support **multiple deployment options**. This approach will ensure that the software can adapt to a variety of hosting scenarios without requiring significant changes to the code. To achieve this, deployment tools and processes will focus on:

- **Extensive Documentation**: Clear, step-by-step deployment instructions will be provided, covering common setup scenarios and any necessary configurations. This documentation will guide users through connecting the software to different environments, from setting up cloud-based infrastructure to deploying on a local server.
- **CI/CD Flexibility**: We will create a CI/CD setup that allows for easy adaptation by others, with scripts and processes that are not tied to a specific CI/CD tool or provider. This flexibility will enable future users or institutions to quickly establish their own CI/CD pipeline and maintain automated deployments.

By prioritizing flexibility, clear documentation, and deployment-agnostic practices, we aim to ensure that the project can be effectively deployed in any environment the project partner or future users choose.

2.10 Compliance and Legal Requirements

2.10.1 Data Protection and Privacy

The software must comply with the **Family Educational Rights and Privacy Act (FERPA)** since it will handle student information in an educational setting. In addition, we will ensure privacy-by-design and privacy-by-default principles are followed, minimizing data collection to only what is necessary and providing users with clear consent options and privacy settings.

FERPA mandates the protection of student education records, which includes data such as student names, grades, and polling responses, when these records are tied to identifying information. To comply with FERPA to the best of our ability, the software will incorporate the following data protection and privacy measures:

- **Data Encryption**: All sensitive data, particularly student records and response data, will be encrypted both at rest and in transit. SSL/TLS protocols will be used to secure data transfers between the client and server.

- **Access Control**: Role-based access control (RBAC) will be implemented to ensure that only authorized users (e.g., instructors, administrators) can access or modify sensitive student information. Student data will only be accessible to users who have the appropriate permissions.
- **Data Minimization**: The software will limit the collection and retention of personally identifiable information (PII) to what is strictly necessary for functionality. Data that is no longer needed for operational purposes will be anonymized or securely deleted in line with FERPA guidelines.
- **User Consent and Transparency**: Users, particularly students, will be informed of how their data will be used, stored, and protected. This transparency is critical for maintaining trust and adhering to FERPA's principles of privacy and informed consent. Please note that as this project is still in development, expanding user consent and data transparency is considered out of scope of our work this year.
- **Data Retention and Deletion**: State laws throughout the United States vary wildly in their requirements. Due to the complexity and inconsistency of these regulations, we consider data retention and deletion practices out of scope for our work on the project this year.
- **General Data Protectio Measures**: Although FERPA is the primary legal requirement for this project, the software's data handling practices will also follow general data protection best practices, such as secure storage and secure user authentication. These measures will help safeguard data against unauthorized access or breaches, ensuring the integrity and confidentiality of all stored information.

By implementing these data protection practices, the software aims to ensure compliance with FERPA and protect the privacy of all student data managed by the application. However, **FERPA compliance is not a requirement of this project**. We will instead be designing for the software to be FERPA compliant to the best of our abilities, understanding that compliance is a complex legal process that we are not committing to for the scope of this project.

2.10.2 Legal, Ethical, and Regulatory Considerations

In addition to FERPA, there are several other legal, ethical, and regulatory considerations to ensure the software remains compliant and accessible to all users. These include:

- **GPL v2.0 Licesnse**: This software is distributed under the **GNU General Public License v2.0 (GPL v2.0)**. As a free and open-source license, GPL v2.0 allows others to freely use, modify, and distribute the software, provided that any derivative works also remain open source under the same license. This license promotes transparency, encourages community contributions, and helps ensure the software remains freely accessible to educational institutions.

  - **Compliance with GPL v2.0**: Contributors to the project must follow the terms of GPL v2.0, meaning any code modifications or extensions to the software must also be licensed under GPL v2.0 if they are distributed. Additionally, when redistributing the software, contributors must include the original copyright notice and licensing terms, as well as any applicable source code.

- **Accessibility Compliance (WCAG Guidelines)**: The software aims to be accessible to all users, including those with disabilities, by adhering to the **Web Content Accessibility Guidelines (WCAG)**. WCAG outlines best practices for ensuring that digital content is accessible to individuals with visual, auditory, motor, and cognitive impairments. The project will strive to meet **WCAG 2.1 Level AA compliance**, which includes:
  - **Keyboard Accessibility**: Ensuring that all interactive elements (e.g., buttons, input fields) are navigable and operable via keyboard only, accommodating users who cannot use a mouse.
  - **Screen Reader Compatibility**: Ensuring that the software works seamlessly with screen readers by using appropriate HTML elements, ARIA (Accessible Rich Internet Applications) attributes, and descriptive alt text for images and icons.
  - **Color Contrast**: Adhering to color contrast guidelines to make text and interface elements readable for users with visual impairments, including color blindness.
  - **Scalable Text and Responsive Design**: Allowing users to adjust text size without loss of functionality and ensuring that the interface remains usable on various screen sizes and devices.

In addition to legal compliance, the project considers the ethical implications of data privacy, accessibility, and educational equity. As an open-source solution aimed at reducing financial barriers to classroom engagement, this software aligns with principles of equitable access and inclusivity in education. Furthermore, strict data privacy measures ensure that students' personal information is safeguarded, respecting their rights and maintaining trust within the educational community.

3 Verification and Validation

3.1 Objective of Verification and Validation

The objective of Verification and Validation for this project is to rigorously assess the software's effectiveness in meeting defined success criteria, ensuring it functions reliably, meets user needs, and aligns with project goals. Through a structured evaluation process, we aim to confirm that each feature performs as intended, that user interactions are intuitive and efficient, and that the software is stable, secure, and scalable.

This validation process will include comprehensive testing, user feedback, and performance assessments to verify that the software meets key benchmarks, such as real-time responsiveness, usability, accessibility, and data security. By following a structured approach, we aim to deliver a project that is not only functional but also adaptable and sustainable for long-term use across diverse educational environments.

3.2 Evaluation Metrics

The project will be evaluated through a combination of testing milestones, feature completeness, and user acceptance criteria. This means that all core functionalities required for classroom polling, such as user authentication, poll creation, real-time response collection, and result viewing, must be fully implemented, tested, and validated. Key metrics for evaluating success include:

- **Passing Test Suites**: All **unit**, **integration**, and **end-to-end** tests must pass with no critical issues. Additionally, **automated accessibility tests** must confirm WCAG Level AA compliance, and **performance** tests should demonstrate acceptable response times and load-handling capabilities.
- **User Testing and Feedback**: The project will also be evaluated by feedback from usability testing with instructors and students. Any significant usability issues identified during user testing must be addressed before release to ensure a smooth user experience. The user testing will ask users to evaluate the project upon the following design principles:
  - **User-centric Design**: The UI should be intuitive and easy to navigate for both students and instructors, with a focus on clear and accessible workflows for joining classes, launching polls, and viewing results. Minimal setup steps should ensure users can easily get started without difficulty.
  - **Responsive Design**: The UI should have key elements like buttons, input fields, and interactive components that should be usable across varying screen sizes.

  - **Clear Visual Hierarchy**: The design should use consistent icons, color schemes, and typography to guide users efficiently through each task. Clear distinctions between instructors and student views are essential to prevent confusion.

- **Metrics for Stability and Performance**
  - **Code Coverage**: Our project software must achieve a high level of code coverage at 80% or above in unit and integration tests to ensure thorough testing across the codebase. All tests must pass.
  - **Uptime Requirements**: Reliability and uptime metrics from testing should demonstrate that the platform can meet the 99.9% availability target, ensuring that it will perform reliably in real classroom scenarios.
  - **Performance Benchmarks**: The application should meet or exceed performance benchmarks for response time, typically under 500 milliseconds per action, to ensure real-time interaction in classroom settings.

By adhering to these rigorous testing and readiness standards, the team will ensure that the software is not only functional but also stable, reliable, and user-friendly.

3.3 Testing Plan

This testing plan is designed to ensure the software's functionality, performance, accessibility, and user satisfaction through a comprehensive approach that includes both **automated software testing** and **user testing**.

3.3.1 Automated Software Testing

Automated testing will cover unit, integration, performance, and accessibility aspects of the software, providing a consistent and repeatable process for validating the platform's functionality and stability.

1. **Unit Testing**:
   - **Frontend**: For the frontend, **Vitest** will be used to test helper functions and other traditional functions that do not rely on the DOM. This approach allows early detection of bugs in utility functions and data transformations.
   - **Backend**: Unit tests in the backend will be conducted using **Jest** to achieve complete code coverage. These tests will validate individual functions and methods, ensuring core logic, data processing, and utility functions perform as expected.
2. **Integration Testing**:

- **Frontend**: Integration testing on the frontend will leverage **jsdom** and **Vitest** to simulate the DOM environment, validating how different components interact within the application. Integration tests will focus on verifying workflows, such as button clicks, form submissions, and transitions between screens, to confirm a smooth user experience.
- **Backend**: Select backend integration tests will be conducted to validate interactions with external services, such as the MySQL database. This will ensure that API endpoints function correctly and maintain data integrity, particularly in complex scenarios. However, backend integration testing may be used sparingly based on the level of coverage achieved through other test types.

3. **End-to-End (E2E) Testing**:
   - E2E testing on the frontend will be performed using **Cypress**. These tests simulate real-world scenarios from a user's perspective, covering workflows such as creating and submitting polls, viewing aggregated results, and navigating core features. E2E tests validate that the entire system functions as expected when all components are integrated.
4. **Performance Testing**:
   - Performance testing will be conducted to confirm that the platform remains responsive, even under load. **k6** will be used to simulate concurrent user interactions, particularly in scenarios with multiple simultaneous polling sessions. This testing will focus on metrics like response times, data transfer rates, and resource usage to ensure the platform meets standards for real-time responsiveness.
5. **Uptime and Reliability Testing**:
   - Uptime testing will be essential to confirm that the application can meet the target of 99.9% availability during active classroom sessions. Monitoring tools such as **Pingdom** or **UptimeRobot** will be used to track uptime continuously and alert the team to any service interruptions, though we have yet to choose a tool at this time. Additionally, failover strategies will be tested to verify the resilience of the application under potential outages. This will only be able to be tested when we are able to deploy the project. If we are unable to deploy the project, we will informally track this through the number of failed requests over a period of time during live testing.
6. **Automated Accessibility Testing**:
   - Automated accessibility testing will be conducted using tools such as **axe** within the CI/CD pipeline. These tools will help identify common accessibility issues, such as color contrast, missing alt text, and keyboard navigation limitations, ensuring compliance with **WCAG 2.1 Level AA**. Automated accessibility testing will be supplemented with user testing to ensure a comprehensive approach.

3.3.2 User Testing

User testing will focus on evaluating the platform's usability, satisfaction, and practical effectiveness. This process will involve surveys, structured usability tests, and gathering feedback directly from the target audience to inform final adjustments to the software.

1. **Surveys**:
    - **Objective**: Collect quantitative and qualitative feedback on the platform's usability, satisfaction, and feature effectiveness.
    - **Survey Topics**:
        - **Ease of Use**: Assess users' perceptions of how intuitive and straightforward the platform is to use.
        - **Feature Satisfaction**: Gather feedback on whether the features meet user needs and expectations.
        - **Interface Clarity and Design**: Evaluate how clear and visually engaging the interface is.
        - **Navigation and Workflows**: Assess how easily users can navigate the platform and complete core tasks.
        - **Reliability and Performance**: Capture feedback on the software's reliability and speed.
        - **Suggestions for Improvement**: Allow users to provide open-ended feedback on how the platform could be enhanced.
2. **Structured Usability Tests**:
    - **Objective**: Observe real users as they complete key tasks on the platform to identify potential usability issues and areas for improvement.
    - **Process**
        - Participants will perform essential workflows, such as creating a poll, viewing real-time responses, and accessing results, with minimal guidance.
        - Observers will track task completion rates, note any difficulties encountered, and record time-on-task to identify areas where the interface could be more intuitive.
        - Each session will conclude with a debrief, allowing participants to share their impressions and suggestions for enhancing the user experience.

3.3.3 Additional Considerations

Cost-Effectiveness Validation: Given that this platform is offered free of charge, the cost-effectiveness analysis will primarily involve comparison with existing paid alternatives. This will demonstrate the financial advantage for users without charging fees. This has already been performed in **Section 1.4 Competitive Analysis**.

3.4 Validation Against Success Criteria

This validation will verify that the software addresses user needs, performs reliably, and complies with the defined success metrics. The validation process will integrate insights from both automated and user testing to ensure the platform is ready for deployment in educational environments.

3.4.1 Core Objectives for Validation

- **Meeting User Needs**:
  - **Usability**: The platform will undergo rigorous usability testing, with surveys and structured usability tests designed to evaluate **Ease of Use**, **Feature Satisfaction**, **Interface Clarity**, **Navigation and Workflows**, and **Suggestions for Improvement**. Success will be measured by user feedback indicating a high level of satisfaction with the interface, intuitiveness, and overall user experience.
  - **Feature Completeness**: All core functionalities, such as user authentication, poll creation, real-time response collection, and result viewing, must be implemented, thoroughly tested, and validated through user feedback. Each feature must align with the project's functional requirements, and users should express satisfaction with each feature's functionality and design.
  - **Accessibility**: Automated accessibility testing (using tools like axe or Lighthouse) and user testing with assistive technologies will validate that the platform meets **WCAG 2.1 Level AA** compliance. This criterion ensures that the software is accessible to all users, including those with disabilities, contributing to a fully inclusive classroom experience.
- **Achieving Performance and Stability Benchmarks**:
  - **Passing Automated Test Suites**: All unit, integration, and end-to-end tests must pass without critical issues. Unit and integration tests will cover at least **80% of the codebase**, ensuring comprehensive validation of both backend and frontend functionality.
  - **Performance Benchmarks**: The platform must achieve response times of under **500 milliseconds per action** to meet real-time interaction standards essential for classroom polling. Load testing through tools like **k6** will verify that the software performs efficiently under concurrent user loads typical of large classes or multiple simultaneous sessions.
  - **Uptime Requirements**: The platform must demonstrate a target uptime of **99.9%** in testing, indicating that it is reliable enough to support real-world classroom scenarios without disruption. Monitoring tools will track uptime and alert the team to any issues, with failover mechanisms in place to support continuous functionality.
- **Cost Effectiveness**: As a free and open-source platform, the software's cost-effectiveness will be validated through comparison with existing paid alternatives. This validation will confirm

the financial advantage the platform offers to educational institutions and students, as analyzed in **Section 1.4 Competitive Analysis**.

3.4.2 Validation Process

- **Automated Testing Results**: The results from unit, integration, performance, and automated accessibility tests will be reviewed to ensure that all components function as expected and meet the required standards for performance and accessibility. These tests provide consistent verification across the codebase, ensuring that individual components and workflows meet functionality, stability, and accessibility criteria.
- **User Testing Feedback**: Feedback from usability tests and surveys will be analyzed to ensure the platform meets user expectations and supports efficient, satisfying workflows. Any significant usability issues identified in testing will be addressed, with iterative improvements made based on user feedback. Success will be indicated by high user satisfaction scores across core usability dimensions (e.g., ease of use, feature satisfaction, reliability).
- **Final Validation Against Benchmarks**: After addressing user feedback and refining the software, a final round of testing will verify that all metrics and success benchmarks are met. This validation includes verifying feature completeness, passing all tests, and meeting or exceeding performance, accessibility, and uptime targets.

Through this structured validation process, the software's alignment with user needs, performance standards, and project goals will be confirmed, ensuring that the platform is ready for deployment in a real-world educational setting.

3.5 Long-Term Validation

As our team will conclude its involvement with this project at the end of the academic year, **long-term validation** will not be within the scope of our responsibilities. Future validation, improvements, and ongoing user support will be the responsibility of subsequent OSU Student Capstone development groups.

3.6 Validation Reporting

The documentation and reporting strategy for validation findings will focus on providing clear, actionable insights to guide improvements and inform stakeholders without excessive detail. The approach will be straightforward, capturing key results and essential feedback to ensure the software meets project goals and user needs.

1. **Data Collection and Summary**:

- **User Feedback Summaries**: Key findings from user surveys and usability tests will be documented in concise summaries. This will include overall satisfaction ratings, common usability issues, and prioritized improvement suggestions, helping us identify areas for enhancement.
- **Automated Testing Results**: Pass/fail results from unit, integration, end-to-end, accessibility, and performance tests will be recorded. Metrics such as code coverage percentage, accessibility compliance status, and load test performance will be noted to verify adherence to project benchmarks.

2. **Stakeholder Presentation**: For final project validation, a concise presentation will be prepared for stakeholders, focusing on:
   - Achievement of core objectives, highlighting successful test results and user satisfaction scores.
   - Key areas of improvement addressed, ensuring stakeholders see how feedback was incorporated.
   - Final validation metrics confirming that performance, accessibility, and functionality goals were met.

Validation Summary Report: A single summary document consolidating key findings from the final round of testing, presenting the status of all success criteria and any last recommendations for future teams.

4 Documentation and Training

4.1 User Manuals and technical Documentation

To ensure that the software is accessible, maintainable, and adaptable by both current users and future development teams, comprehensive documentation will be created. This documentation will serve a dual purpose: supporting developers and administrators in managing the software and providing end-users with clear, concise user guides. Some of this documentation has already been written by previous student development teams, but we will commit to updating and improving the following documentation.

Developer-Focused Documentation:

- **Setup and Installation Guide**: Detailed instructions on setting up the development environment, including dependency installation, configuration files, and initializing databases. This guide will cover both local development and a production-like setup to assist future developers in getting the project running smoothly.
- **Contribution Guidelines**: Clear guidelines for developers who want to contribute to the project. This will include instructions on issue creation, branching strategy, coding standards, testing requirements, and the pull request process, as well as expectations for documentation and code comments.
- **Deployment Instructions**: Comprehensive deployment documentation will enable other universities or institutions to host their own versions of the software. This guide will outline the necessary steps for deploying the software on cloud-based or on-premises infrastructure, including configuration, scaling considerations, environment variable management, and troubleshooting.
- **Administration and Maintenance Guide**: This documentation will cover the tasks required to maintain the software once deployed. It will include instructions on managing user roles, accessing logs, updating dependencies, performing backups, and other routine maintenance tasks essential for non-developer administrators.
- **Technical Documentation on Software Design**: Detailed documentation on the software's architecture, including the frontend and backend components, API structure, database schema, and integration points (i.e., external API integration).
- **Source Code Documentation**: The source code will include clear and informative comments to make the codebase more accessible to new developers. The comments will clarify the purpose, expected inputs/outputs, and key considerations for functions, allowing contributors to understand the code structure and logic without extensive background.

User-Focused Documentation:

- **User Manual**: A guide for instructors, students, and administrators on using the platform's core features, including account creation, poll setup, participation, and result analysis. This guide will include screenshots, step-by-step instructions, and explanations of features to ensure that users can navigate the software independently.
- **FAQ and Troubleshooting Guide**: A collection of frequently asked questions and common troubleshooting tips. This document will address typical issues users might encounter, such as login issues, connection problems during live polling, or difficulty accessing past poll results.

By providing extensive documentation for both developers and end-users, the project aims to ensure that the software remains accessible and manageable, enabling other institutions to adopt and benefit from it while ensuring a smooth transition to future development teams.

4.2 Training and Onboarding

While comprehensive training for future student development teams may not be feasible, we commit to creating robust documentation (outlined above) that will serve as a self-sufficient resource for onboarding new developers. This documentation will ensure that future teams can get up to speed quickly and understand the project's setup, features, and maintenance requirements.

However, we can provide some basic training for our project partner at the end of our development on the project to ensure they are equipped to maintain and administer the platform independently. This training will include:

- **Maintenance and Administration Training**: A session covering essential tasks for administrators, such as managing user roles, conducting backups, monitoring performance, and performing routine updates. This training will help our project partner manage the system from a non-developer perspective.
- **Handoff and Transition Guide**: Guidelines for how to transition the project to future student development teams, including a walk-through of key documentation, handover of administrator credentials, and recommendations for project continuity.

Through these efforts, we aim to support both immediate and long-term sustainability for the software, ensuring that it remains functional, well-maintained, and adaptable to future needs.

5 Maitenance and Support

5.1 Post-Deployment Support

We are unable to commit to long-term support or ongoing development after deployment. Therefore, we will focus on creating comprehensive documentation to enable others to manage the software independently. This includes a detailed troubleshooting guide and clear maintenance instructions to help future administrators and developers address common issues without needing direct support.

5.2 Ongoing Maintenance and Updates

The software will likely be maintained by future OSU Computer Science Capstone teams. To facilitate a smooth transition, we will ensure the software is as stable and reliable as possible through rigorous testing and validation. This approach aims to minimize the need for immediate intervention, allowing future developers to focus on new features or improvements rather than initial bug fixes or troubleshooting. Thorough documentation will be provided to guide them in ongoing maintenance and updates, as defined in the Documentation and Training section of this document.

6 Project Timeline and Milestones

As we are building on work initiated by a previous team, some features have already been partially implemented. However, issues within the existing codebase and a lack of documentation for setup and project management have created obstacles to accessing and fully understanding this prior work. Unresolved bugs and technical challenges ensure that this is an approximate, not an exact, timeline.

Each task is designed to function independently, allowing for flexibility and minimizing dependency impacts. Additionally, tasks with a higher likelihood of having been addressed by the previous team are scheduled earlier to expedite integration. The goal completion date for each task and milestone assumes a degree of concurrency for development, as we have a large team working on developing software, resulting in the estimated task duration not always adding up to the associated completion date.

Table 2: Project Timeline and Milestones

Task No.	**Task**	**Estimated Duration**	**Goal Completion Date**	**Priority**	**Dependencies**	**Details**
Milestone 1: Setup and User Account Management (Week 1 W25)						
1.1	Database Setup and Management	<1 week	Week 6 F24	High	None (initial setup task)	Set up a standardized development environment, including necessary tools, dependencies, and database connections.
1.2	User Registration and Account Creation	1 week	Week 8 F24	High	1.1	Complete user registration workflows, including personal details capture and role selection.
1.3	Role Selection and Access Control	1 week	Week 9 F24	High	1.2	Establish role-based access with dashboards tailored for instructors and students.
1.4	Account Management and Security	1 week	Week 1 W25	High	1.1, 1.2	Enable editing of personal details (name, email) and implement password reset functionality.

Continued on next page

Continued from previous page

Task No.	Task	Estimated Duration	Goal Completion Date	Priority	Dependencies	Details
1.5	Testing and Quality Assurance for User Management	1 week	Week 1 W25	Medium	1.2, 1.3, 1.4	Conduct usability and security testing for user account workflows.
Milestone 2: Classroom Creation and Management (Week 4 W25)						
2.1	Classroom Creation Functionality	1 week	Week 2 W25	High	1.3	Allow instructors to create new classes and access a classroom details page.
2.2	Student List Management	1 week	Week 2 W25	High	2.1	Enable instructors to upload a student list via CSV or retrieve student information from Canvas.
2.3	Student List Management	1 week	Week 3 W25	High	2.2	Implement process for students to select and link to their entry on the class roster.
2.4	Integrate Gradebook with Canvas API	1 week	Week 3 W25	Medium	2.3	Establish Canvas API connection, allowing instructors to export grades for students.
2.5	Testing and Quality Assurance for Classroom Management	1 week	Week 4 W25	Medium	2.1, 2.2, 2.3, 2.4	Perform functional and role-based testing on classroom and student management workflows.
Milestone 3: Lecture Management and Interactive Lecture Features (Week 10 W25)						
3.1	Lecture Creation and Management	1 week	Week 5 W25	High	2.1	Allow instructors to create and manage lecture sessions.
3.2	Live Lecture Control	1 week	Week 5 W25	High	3.1	Implement functionality for instructors to set lectures as "live," allowing students to respond in real time.

Continued on next page

Continued from previous page						
Task No.	**Task**	**Estimated Duration**	**Goal Completion Date**	**Priority**	**Dependencies**	**Details**
3.3	Real-Time Lecture Interactions	3 weeks	Week 8 W25	High	3.1, 3.2	Enable real-time updates for instructors as students respond to lecture questions.
3.4	Testing and Quality Assurance for Lecture Features	2 weeks	Week 10 W25	Medium	3.1, 3.2, 3.3	Conduct testing on lecture setup, live session handling, and real-time interactions.
Milestone 4: Test Deployment and Performance (Week 3 S25)						
4.1	Test Deployment and Load Simulation	1 week	Week 1 S25	High	1.5, 2.5, 3.4, Access to Deployment Server	Deploy the application in a test environment mimicking production.
4.2	Scalability and Performance Testing	1 week	Week 2 S25	High	4.1	Execute load tests to determine user capacity and measure response times.
4.3	Reliability and Stress Testing	1 week	Week 3 S25	High	4.2	Test application resilience by gradually increasing user load.

List of Tables