
Ticket 5 STP

CS5213 Group B

Feb 24, 2024

CONTENTS

1	Introduction	1
1.1	Purpose	1
1.2	Scope	1
1.3	References	1
2	Software Quality	2
2.1	Coding Standards:	3
2.2	Test-Driven Development (TDD):	5
3	Requirements	8
3.1	Functional Requirements	8
3.2	R1:User Registration	8
3.3	Detail Data Required/User Inputs	10
3.4	R3: For Project Proposal and Approval	11
3.5	R4 For Project Repository	13
3.6	R4.11 Team Formation	15
3.7	R5 For Communication and Collaboration Platform	17
3.8	R6 For Milestone Tracking	17
3.9	R7. For Feedback and Evaluation	18
3.10	R9 For Calendar and Notifications	19
3.11	R10 For Reporting and Analytics	20
3.12	R11 External Interface Requirements	20
3.13	R12 Non Functional Requirements	22
4	Test Strategy	26
4.1	Deployment Environments	26
4.2	Critical Unit Tests	28
4.3	Integration Testing Strategy	30
4.4	Test Execution	30
4.5	Reporting	30
4.6	Regression Testing	31
4.7	Continuous Monitoring	31
4.8	Test Schedule	32
4.9	Exit Criteria	33
5	Test Case	35
5.1	Test cases for CMS	35
6	Verification and Validation	82
6.1	Verification	82
6.2	Validation	82

7	Report Templates	84
7.1	Test Report Template	84
7.2	GitHub Issue Template	85
8	Traceability Matrix	88
9	References	98

INTRODUCTION

This document constitutes the Software Test Plan for the Capstone Management System (CMS). The CMS is a tool designed to improve the capstone experience for all participants: students, faculty, and industry partners. The Capstone Management System will provide an interface for all stakeholders to engage in the process of generating and performing capstone projects.

A thorough description of the Capstone Management System software is available in the reference documents listed below. Specifically, the software development plan (SDP) will cover the overall system description.

1.1 Purpose

The purpose of this document is to provide a complete picture of the testing approach for the CMS. It will outline the necessary requirements, test plan and issue templates, testing strategies, test cases, verification and validation, and test traceability matrix.

1.2 Scope

This document's scope is limited to the CMS software's development lifecycle. It should be a living document, updated by the development teams as the software is developed and a more nuanced understanding of the system is achieved. The intended use of this document is for development and testing team members to reference while developing the application and for stakeholders to review to understand the testing process.

1.3 References

Documentation used for reference in this document. If SDP, SQAP, SRS, or Use-Cases are mentioned, they correspond to one of the following documents:

- Capstone Management System Software Development Plan (SDP, GroupB)
- Capstone Management System Software Quality Assurance Plan (SQAP, GroupB)
- Capstone Management System Software Requirements Specification (SRS, GroupB)
- Capstone Management System Software Use-Cases (GroupB)

SOFTWARE QUALITY

Quality in software development is fundamental for creating successful products. It encompasses various attributes such as functionality, reliability, usability, efficiency, maintainability, portability, and security. Beyond merely avoiding defects, software quality strives for excellence, ensuring that a product consistently and effectively fulfills its intended purpose.

The dimensions of software quality include ensuring that the software meets specified features and operations, providing consistent and dependable performance, creating an intuitive and user-friendly interface, optimizing resource utilization for efficient operation, facilitating ease of modification and expansion, adapting to diverse environments and platforms, and implementing robust security measures.

The significance of software quality is evident in various aspects. It enhances customer satisfaction by delivering reliable and user-friendly products. It contributes to cost efficiency by identifying and rectifying issues early in development. A focus on software quality helps build a positive reputation for development teams and organizations within the industry. It also aids in proactively addressing potential risks associated with software development and ensuring compliance with industry regulations and best practices.

Software quality is not a static goal but an ongoing, integrated process that permeates every phase of the software development lifecycle. Achieving quality involves meticulous methodologies, rigorous testing practices, and a commitment to continuous enhancement. Prioritizing software quality empowers organizations to deliver products that exceed functional expectations, providing users with a dependable and satisfying experience.



2.1 Coding Standards:

In the domain of software development, adherence to robust coding standards is pivotal for ensuring not just the functionality but the overall quality and maintainability of codebases. The guidelines presented encompass a spectrum of technologies, including Python frameworks (Django, Wagtail, Django CMS), databases (MySQL, PostgreSQL, SQLite), and frontend frameworks (React.js, Angular, Vue.js).

2.1.1 Python (Django, Wagtail, Django CMS):

Naming Conventions: Naming is a cornerstone in code readability. Following the PEP 8 standard ensures a unified and comprehensible codebase. For Django models and fields, clear and descriptive names contribute to the clarity of the underlying data structure.

Indentation and Formatting: Consistent indentation, as per PEP 8 guidelines, fosters an organized and readable code structure. Code formatters like black not only maintain consistency but also automate the often subjective aspect of code formatting.

Comments: Effective use of comments, particularly docstrings for documentation, is crucial for understanding complex modules, classes, and functions. Striking the right balance between comments and self-explanatory code is an art that enhances code maintainability.

Whitespace: Whitespace consistency, enforced by tools like flake8, goes beyond aesthetics. It plays a crucial role in code readability and contributes to the overall visual coherence of the codebase.

Code Organization: The recommended Django project structure is not just a convention but a practical approach to organizing code for scalability and maintainability. Grouping related functionality within modules, classes, and functions further aids in comprehensibility.

Error Handling: Comprehensive error handling is not just about catching exceptions but also about logging and documenting errors effectively. Django's logging framework is a powerful tool for maintaining visibility into runtime

issues.

Code Duplication: Identifying and mitigating code duplication is a proactive step toward maintainability. Leveraging Django's class-based views and model mixins is a strategic approach to promoting code reuse.

Consistent Code Style: Automation, through pre-commit hooks with tools like flake8 and black, minimizes the cognitive load on developers and ensures that code adheres to the specified standards consistently. Continuous integration tools act as a safety net, catching deviations from these standards in pull requests.

Testing and Documentation: While unit tests verify critical paths, comprehensive documentation, including API endpoints and data models, is indispensable for ensuring that the codebase remains comprehensible over time.

2.1.2 Databases (MySQL, PostgreSQL, SQLite):

Naming Conventions: Clear and consistent naming conventions for tables, columns, and constraints contribute to a comprehensible database schema. Following PostgreSQL's lowercase identifier convention ensures uniformity.

Schema Design: Normalized schemas reduce redundancy and maintain data integrity. Regularly reviewing and optimizing queries is essential for sustained performance.

Indexes: Strategic indexing based on query patterns is a performance optimization technique. Regular monitoring and optimization ensure that indexes enhance retrieval speed without unnecessary overhead.

Error Handling: Beyond transaction handling, logging database-related errors with specificity using database-specific error codes is crucial for debugging and maintaining data integrity.

Security Practices: Applying the principle of least privilege to database users and conducting regular audits of user roles and permissions are proactive measures in safeguarding the database.

2.1.3 Frontend (React.js, Angular, Vue.js):

Naming Conventions: Each frontend framework has its naming conventions for components, variables, and functions. Adhering to these conventions contributes to code consistency and aids in collaboration.

Indentation and Formatting: Consistent indentation is not just about aesthetics; it's a readability factor. Code formatters like Prettier automate formatting, allowing developers to focus on logic rather than style.

Comments: Striking a balance between comments and self-explanatory code is crucial. Excessive comments can clutter code, but well-placed comments explaining complex components or decisions are valuable.

Whitespace: Whitespace conventions, enforced by linting tools like ESLint or TSLint, contribute to code readability and maintainability. Consistency in applying whitespace conventions is vital.

Code Organization: Organizing components and services as per recommended project structures enhances code maintainability. Grouping related functionality within components and services ensures clarity.

Error Handling: Implementing error boundaries in React.js and using Angular's error handling mechanisms are preventive measures for graceful error handling, contributing to a better user experience.

Code Duplication: Strategically extracting common logic into reusable components, directives, or services contributes to a modular and maintainable codebase.

Consistent Code Style: Linting tools integrated into the development workflow, coupled with continuous integration tools, ensure real-time feedback on code adherence to standards, preventing the introduction of non-compliant code.

Testing and Documentation: Writing unit tests for components and services using framework-specific testing libraries is a foundational practice. Documenting component APIs, state management, and usage guidelines ensures that future developers can work effectively with the codebase.

2.1.4 Enforcing Coding Standards:

Linting Tools: Integration of linting tools into the development environment ensures real-time identification of coding standard violations, promoting adherence during development.

Code Formatters: Automating code formatting through tools like Prettier or Black ensures consistency, minimizing debates over formatting preferences and promoting a unified code style.

Version Control Hooks: Version control hooks serve as a gatekeeper, ensuring that only code meeting specified coding standards enters the version control system.

Continuous Integration (CI): CI pipelines, configured to run linting, formatting, and unit tests on each pull request, establish a systematic approach to enforcing coding standards and preventing the merging of non-compliant code.

Code Reviews: Code reviews, a fundamental part of the development process, offer a human-centric layer to the enforcement of coding standards. Code review tools that highlight style violations enhance the efficiency of this process.

In summary, the implementation of these coding standards, complemented by a robust enforcement mechanism, establishes a solid groundwork for the development of not only high-quality code but also a collaborative and sustainable development process. Ongoing updates to coding standards, aligned with project needs and continuous improvement initiatives, guarantee adaptability and resilience in response to evolving project requirements.

2.2 Test-Driven Development (TDD):

2.2.1 Automated Test-Driven Development (ATDD):

1. Define Behavior:

- *Objective:* Clearly articulate the expected behavior of a specific feature in the Capstone Management System.
- *How:* Collaborate with stakeholders to create user stories and acceptance criteria.

2. Write Automated Test:

- *Objective:* Create an automated test outlining criteria and conditions for expected behavior.
- *How:* Use testing frameworks compatible with the technology stack (e.g., PyTest for Python/Django).
- *Considerations:* Cover various scenarios, including edge cases and potential user inputs.

3. Implement Code:

- *Objective:* Develop the minimum code required to pass the automated test.
- *How:* Focus on functionality addressing the test criteria.
- *Considerations:* Keep the codebase simple and modular.

4. Run Automated Tests:

- *Objective:* Execute all automated tests, including the new one, to confirm implemented code meets specified criteria.
- *How:* Integrate automated tests into the development workflow.
- *Considerations:* Maintain a continuous feedback loop, addressing failures promptly.

5. Refactor Code (if needed):

- *Objective:* Improve code without changing behavior, focusing on readability, maintainability, and performance.
- *How:* Refactor based on automated tests and code reviews.

- *Considerations:* Ensure all automated tests, including the new one, pass after refactoring.

2.2.2 Manual Test-Driven Development:

1. Define Expected Behavior:

- *Objective:* Clearly define expected behavior for a feature in the Capstone Management System.
- *How:* Collaborate with stakeholders to create detailed user stories, scenarios, and conditions.

2. Write Manual Test Cases:

- *Objective:* Develop detailed manual test cases as a guide for implementing corresponding code.
- *How:* Outline step-by-step instructions, inputs, and expected outcomes.
- *Considerations:* Ensure test cases cover a range of scenarios, positive and negative.

3. Implement Code:

- *Objective:* Write necessary code to fulfill requirements outlined in manual test cases.
- *How:* Refer to manual test cases as a roadmap for development.
- *Considerations:* Align code implementation with steps and conditions in manual test cases.

4. Execute Manual Tests:

- *Objective:* Manually execute test cases, following specified steps and inputting defined data.
- *How:* Act as a tester, simulating user interactions to validate functionality.
- *Considerations:* Document any deviations between actual results and expected outcomes.

5. Refine Test Cases (if needed):

- *Objective:* Update and refine manual test cases based on insights gained during code implementation.
- *How:* Adjust test cases to account for nuances or unexpected scenarios discovered during testing.
- *Considerations:* Continuously improve and expand the manual test suite with application evolution.

2.2.3 Benefits:

Automated Test-Driven Development (ATDD):

- *Automation Efficiency:* Rapidly run automated tests for quick and frequent feedback on code changes.
- *Regression Testing:* Automated tests act as a safety net, ensuring existing functionalities are not broken.
- *Continuous Integration:* Seamlessly integrate automated tests into CI/CD pipelines for continuous delivery.

Manual Test-Driven Development:

- *User-Centric Validation:* Manual testing provides hands-on validation of user-centric features, considering the overall user experience.
- *Exploratory Testing:* Manual testing allows exploration, uncovering issues not captured by automated tests.
- *Flexibility and Creativity:* Manual testing allows flexibility in adapting test cases to evolving requirements and creatively exploring potential scenarios.

By combining both Automated Test-Driven Development and Manual Test-Driven Development, we establish a comprehensive testing strategy that blends the efficiency of automation with the human touch of manual validation, ensuring the robustness and user satisfaction of the Capstone Management System.

REQUIREMENTS

3.1 Functional Requirements

Functional requirements for the Capstone Management System define the specific behaviors and capabilities that the system must possess to support the needs of students, faculty, and company representatives engaged in capstone projects. These requirements encompass user registration and authentication, with secure role-based access to ensure that users can perform actions pertinent to their roles—such as submitting project proposals, collaborating on documents, tracking project milestones, and evaluating progress. We also include a centralized dashboard for an overview of project statuses, a project repository for document and code storage, team formation tools, integrated communication facilities, a robust calendar with notifications, and comprehensive reporting and analytics features to monitor and assess project outcomes and student performance. Collectively, these functional requirements facilitate a seamless, efficient, and secure management process for all participants in the capstone project lifecycle...

3.2 R1:User Registration

3.2.1 R1.1: Account Creation Interface

- **R1.1.1** Our system will feature a user-centric interface for account creation, prioritizing clarity and ease of use.
- **R1.1.2** A form with mandatory fields will be presented for data entry, ensuring all necessary information is captured.
- **R1.1.3** We plan to apply strict form validation to maintain the integrity of the data. This includes checking the format of emails and requiring strong passwords.
- **R1.1.4** For students and faculty, the active directory servers located on-campus may be used to simplify the login process (see the diagram below).

3.2.2 R1.2 Essential Information Collection

- **R1.2.1 Full Name:** The registration form will accommodate entries for first, middle, and last names.
- **R1.2.2 Contact Details:** We will collect email addresses and phone numbers, supporting the international format for the latter.
- **R1.2.3 Affiliation:** Users will identify their role (Student, Instructor, or Company Representative) via dropdown menus or radio buttons.
- **R1.2.4 Password Creation:** Password fields will incorporate real-time strength evaluation and display criteria to encourage secure account creation.

3.2.3 R1.3 Identity Verification

- **R1.3.1** Submission of the registration form will initiate a verification process through an emailed link or SMS code.
- **R1.3.2** Users must complete the verification within a set timeframe, adding a layer of security to the registration process.

3.2.4 R1.4 Role Selection and Assignment

- **R1.4.1** During registration, users will select their intended role within the system.
- **R1.4.2** The system will notify administrators and faculty, allowing them to approve the assignment of role-based permissions, dictating access levels post-login.
- **R1.4.3** Roles will be integrated into user profiles for subsequent Role-Based Access Control.

An example of sponsor and mentor registration workflow is given below:

3.2.5 R2: Role-Based Access Control (RBAC)

R2.1: Role Definitions

In the system, every user role will have specific permissions that clearly outline which features and actions they can access. We will store these permissions in the database to ensure they are consistently applied across the system.

R2.2: Permissions for Instructors

Instructor will have the ability to oversee project proposals, participate in giving feedback, and supervise assessments. The system will enable monitoring of student advancement and allow for direct interactions with both students and colleagues.

R2.3: Permissions for Company Representatives

Company representatives will have access to customized interfaces designed according to their interaction needs, emphasizing project management and feedback provision. Access for these representatives will be strictly regulated to protect confidential information.

R2.4 Permissions for Students

- **R2.4.1 Access to Projects and Management Capabilities:** Students are granted the ability to access, manage, and interact with their project proposals, documents, and relevant materials within the project repository. This includes submitting proposals, updating documentation, and reporting progress.
- **R2.4.2 Collaboration within Teams:** Students are allowed to form teams, extend invitations to peers for project participation, and work collaboratively on shared documents and resources. The platform supports communication among team members through integrated messaging systems or discussion forums.
- **R2.4.3 Milestone Submission and Progress Tracking:** Students are permitted to submit work pertaining to project milestones, monitor their advancement towards these milestones, and view upcoming deadlines to effectively organize their time.

- **R2.4.4 Receiving Feedback:** Students can receive constructive feedback from instructors and company representatives, with the facility to view evaluations and comments on their work, fostering continuous improvement.
- **R2.4.5 Management of Personal Profiles:** Students have the facility to oversee their personal profiles, which includes the ability to update contact details and modify passwords.
- **R2.4.6 Access to Calendar and Notifications:** Students can access a calendar to track project timelines, deadlines, and scheduled meetings. They will also receive notifications and reminders concerning significant deadlines and events related to their capstone projects.
- **R2.4.7 Access to Resources:** Students are provided access to educational resources, guidelines, and templates to support their capstone project development.
- **R2.4.8 Generation of Limited Reports:** Students can produce reports on their project status and their performance metrics, where applicable.
- **R2.4.9 Restricted Access:** Students do not have access to administrative functions such as the approval of project proposals, grade assignments, or the viewing of other groups' confidential project details. Access to sensitive information regarding faculty assessments and internal company notes is also restricted.

3.3 Detail Data Required/User Inputs

3.3.1 R2.5 For User Registration

- **R2.5.1 Full Name:** The registration process should prompt users to enter their full legal name, ensuring accurate identification in the system. This should include separate fields for the first name, middle name(s), and last name to accommodate various naming conventions.
- **R2.5.2 Email Address:** Users must provide a valid email address that will serve as a primary means of communication and as part of their login credentials. The system should verify that the email address is in a valid format and is not already registered.
- **R2.5.3 Affiliation:** Users are required to select their affiliation with the institution from predefined options such as 'Student', 'Instructor', or 'Company Representative'. This selection will dictate the level of access and permissions granted to the user within the system.
- **R2.5.4 Password:** A password must be created by the user, adhering to defined strength criteria, such as a minimum number of characters, and the inclusion of uppercase letters, lowercase letters, numbers, and symbols. The system should provide real-time feedback on the password's strength and compliance with the criteria.
- **R2.5.5 Contact Number:** As an optional field for additional verification, users may enter a contact number. This number should support international formats and may be used for account recovery or multi-factor authentication purposes.

3.3.2 R2.6 For Login

- **R2.6.1 Username/Email Address:** To access their account, users will be required to enter their unique username or the email address associated with their account.
- **R2.6.2 Password:** The corresponding password for the account must be entered. This password should be transmitted securely and validated against the stored credentials.
- **R2.6.3 Multi-factor Authentication Code:** If MFA is enabled, after entering the username and password, the user will be prompted to provide a code that has been sent to their pre-configured device or email as an additional layer of security.

Sponsor and mentor login will behave similar to 4x4 login, but instead of using LDAP, accounts shall be stored in the database and the authentication backend of the CMS will approve authentication requests. See the diagram below.

3.3.3 R2.7 For Role-Based Access Control (RBAC)

- **R2.7.1 User Role Data:** Upon registration or by administrative assignment, users will have their role within the system recorded—whether they are an Instructor or a Company Representative. This role data is crucial for determining the features and data the user can access.
- **R2.7.2 Permission Levels:** Each role will have associated permission levels which are sets of access rights within the system. These define what actions users can take and what data they can view or modify. Permissions should be granular to allow precise control over user actions and access within the system.

3.3.4 R2.8 Pathways from Specific Requirement

- **R2.8.1 From Registration to System Access:** When you sign up, we make sure your details are correct by sending you an email to check. You need to click on a link in this email to confirm your account. After you confirm through the email, you'll need to finish setting up your account by adding any more information we need. We'll give you a starting role in the system based on if you're a student, teacher, or company person. This can be changed later if needed. Then, we'll send you over to the login page. Here, you can get into the system with the username and password you just made.
- **R2.8.2 From Login to Role-Specific Dashboard:** When you log in, we check your username and password to make sure it's really you. If you have extra security set up (we call this Multi-factor Authentication or MFA), we'll ask for a special code that's sent to your phone or email. Once we know it's you, we'll take you to your main page in the system. This page will look different depending on if you're a student, teacher, or from a company. It'll have everything you need for your projects.
- **R2.8.3 Role-Based Access Implementation:** Right after you log in, the system looks at what role you have. This decides what you can see and do in the system. As you move around in the system, what you can access depends on your role. Some things are only for teachers, some only for students, and some only for company people.

3.4 R3: For Project Proposal and Approval

3.4.1 R3.1 Proposal Submission Interface

- **R3.1.1** A dedicated section where students can fill out and submit their project proposals.
- **R3.1.2** It should include form fields for project title, abstract, detailed description, objectives, methodology, expected outcomes, and any special requirements or resources needed.
- **R3.1.3** The system should allow for the attachment of relevant documents or images that support the proposal.

3.4.2 R3.2 Faculty Review Interface

- **R3.2.1** A portal for faculty members where they can view submitted proposals.
- **R3.2.2** Faculty should be able to filter proposals by various criteria such as submission date, student name, or project category.
- **R3.2.3** An area for faculty to write comments or request additional information from the student.

3.4.3 R3.3 Approval Workflow

- **R3.3.1** Once a proposal is submitted, an automated notification should be sent to the designated faculty for review.
- **R3.3.2** Faculty can approve, reject, or request revisions to the proposal.
- **R3.3.3** A tracking feature for students to follow the status of their proposal through the approval process.

R3.4 Feedback Loop

- **R3.4.1** A system for faculty to provide structured feedback, including comments, suggestions, and necessary improvements.
- **R3.4.2** Students should receive notifications when feedback is provided and be able to view and respond to the comments.

R3.5 Revision Submissions

- **R3.5.1** If revisions are requested, students should be able to submit an updated proposal through the system.
- **R3.5.2** The revised proposal should maintain a link to the original submission for reference.

R3.6 Final Approval and Project Initiation

- **R3.6.1** Once a proposal is approved, a project workspace should be created automatically in the system.
- **R3.6.2** Students and faculty should have access to this workspace based on their roles.

3.4.4 R3.7 Detailed Data Required / User Inputs

R3.7.1 For Proposal Submission

- **R3.7.1.1 Project Title:** A concise title for the project.
- **R3.7.1.2 Abstract:** A brief summary of the project's purpose and goals.
- **R3.7.1.3 Description:** A detailed explanation of the project including background information and the problem statement.
- **R3.7.1.4 Objectives:** Specific objectives the project aims to achieve.
- **R3.7.1.5 Methodology:** The approach or methods the students plan to use in their project.
- **R3.7.1.6 Expected Outcomes:** The anticipated results or products of the project.
- **R3.7.1.7 Resources:** Any special resources or support required for the project.

R3.7.2 Faculty Review

- **R3.7.2.1 Filters:** Options for sorting and searching through proposals.
- **R3.7.2.2 Comments Section:** A text area for faculty to write their observations and suggestions.

R3.7.3 Approval Workflow

- **R3.7.3.1 Notifications:** Automatic alerts for new submissions and status updates.

R3.7.4 Feedback Loop

- **R3.7.4.1 Feedback Form:** A structured format for providing feedback with specific sections for different types of comments.

R3.7.5 Revision Submissions

- **R3.7.5.1 Revision Tracking:** A system to track changes and link revisions to the original proposal.

R3.7.6 Pathway from Specific Requirement

- **R3.7.6.1** When a student submits a proposal, the system checks to make sure all required fields are filled and then sends it to the faculty for review.
- **R3.7.6.2** Faculty members receive a notification and review the proposal at their interface, where they can approve it, ask for changes, or reject it.
- **R3.7.6.3** The student gets a notification about the faculty decision. If changes are needed, they can submit a revised proposal.
- **R3.7.6.4** Once the faculty approves the revised proposal, the system will create a new project workspace and notify the student that their project can start.

3.5 R4 For Project Repository

3.5.1 Detailed Functionality

R4.1 Secure File Storage

- **R4.1.1** The system will provide a secure area online where all project-related files can be stored. This will include documents, images, spreadsheets, presentation files, and source code.
- **R4.1.2** It will use encryption to protect files from unauthorized access both during transfer (as they are uploaded or downloaded) and while at rest (stored on the server).

R4.2 File Management System

- **R4.2.1** Users will have the ability to create folders, rename them, and organize files as needed within the repository.
- **R4.2.2** The system should support version control for documents and code, allowing users to track changes over time and revert to previous versions if necessary.

R4.3 Upload and Download Capabilities

- **R4.3.1** Students and faculty will be able to upload files to the repository, with the system checking for any viruses or malware as part of the upload process.
- **R4.3.2** Users should be able to download files from the repository, with the system logging each download for security and auditing purposes.

R4.4 Access Control

- **R4.4.1** The repository will have a permissions system that controls who can view, upload, or download each file or folder based on their role in the project.
- **R4.4.2** Permissions can be set for individuals or groups, such as a project team or class section.

R4.5 Collaboration Tools

- **R4.5.1** The system will include features for commenting on documents, tagging other users, and suggesting edits.
- **R4.5.2** There should be a mechanism for managing and resolving conflicts when multiple users edit the same document simultaneously.

R4.6 Search Functionality

- Users will be able to search the repository for files based on name, content, date uploaded, and other metadata.

R4.7 Integration with Development Tools

- The repository will be able to integrate with software development environments (IDEs) and other tools commonly used in computer science projects.

R4.8 Backup and Recovery

- **R4.8.1** The system will perform regular backups of all files in the repository.
- **R4.8.2** There should be a process for recovering files in case of accidental deletion or loss.

3.5.2 R4.9 Detailed Data Required / User Inputs

R4.9.1 File Storage

- **R4.9.1.1 Files:** The actual files to be stored, which can be of various types and sizes.
- **R4.9.1.2 Metadata:** Information about the files, such as the author, date created, and a description of the contents.

R4.9.2 File Management

- **R4.9.2.1 Folder Names:** Custom names for folders created by users.
- **R4.9.2.2 File Versions:** Records of each file version when updates are made.

R4.9.3 For Upload and Download

- **R4.9.3.1 Upload Data:** Files selected by users for upload.
- **R4.9.3.2 Download Requests:** User requests for file downloads, specifying the file or files they wish to retrieve.

3.5.3 R4.10 Pathway from Specific Requirement

- **R4.10.1** When a user wants to store a file, they will log into the system, navigate to the project repository, and upload the file. The system will check the file for security and then store it, encrypted, in the correct location.
- **R4.10.2** To organize files, users can create new folders and move files into them. Each change will be recorded in the system's version history.
- **R4.10.3** If a user needs to find a file, they can use the search tool by entering keywords, file names, or other details that describe the file.
- **R4.10.4** When working on files, users can collaborate with teammates by leaving comments or making edits, with the system managing changes to prevent conflicts.
- **R4.10.5** Regular backups will be made of all files so that nothing gets lost. If a user deletes something by mistake, they can ask for it to be restored from the backup.

3.6 R4.11 Team Formation

3.6.1 Detailed Functionality

R4.11.1 Team Creation Interface

- **R4.11.1.1** A clear and simple-to-use interface where students can create a new team for their capstone project.
- **R4.11.1.2** The interface should allow the team creator to set a team name, description, and specify the number of members required.
- **R4.11.1.3** It should enable the team leader (creator) to define roles or skills needed for the project, such as 'programmer', 'designer', or 'researcher'.

R4.11.2 Teammate Search and Invitation

- **R4.11.2.1** A search functionality where students can look for potential teammates by various criteria such as skills, past project experience, academic focus, or availability.
- **R4.11.2.2** The ability to send invitations to join the team directly through the system.
- **R4.11.2.3** An option for students to post a “team member request” on a public board where other students can apply to join.

R4.11.3 Team Joining Mechanism

A feature for students to browse existing teams looking for members.

- **R4.11.3.1** A request to join option that allows students to apply to be part of a team and an approval process for the team leader to accept or reject applications.
- **R4.11.3.2** The system should manage membership requests, showing pending approvals and current team status to the team leader.

R4.11.4 Team Management Tools

- **R4.11.4.1** Tools for the team leader to manage the team such as adding or removing members, assigning roles within the team, and setting permissions for document access.
- **R4.11.4.2** A function to dissolve the team or leave the team for students who wish to exit their current group.

R4.11.5 Communication Tools

- **R4.11.5.1** Integrated messaging within the team formation interface to facilitate communication between team members and the team leader.
- **R4.11.5.2** Discussion boards or forums for team interaction, planning, and collaboration.

3.6.2 R4.12 Pathway from Specific Requirement

- **R4.12.1** A student starts by creating a team, entering all the required information about the project and what kind of team members they are looking for.
- **R4.12.2** Other students can search for teams that match their skills and interests and send a request to join, or respond to a public ‘team member request’.
- **R4.12.3** The team leader receives these requests and can review the profiles of the applicants before deciding to accept or reject them.
- **R4.12.4** Once the team is formed, the leader has tools to manage the team members and their roles, and all team members can communicate with each other through built-in communication tools.
- **R4.12.5** If needed, the team leader can make changes to the team, like adding more members or removing someone if they leave the project.

3.7 R5 For Communication and Collaboration Platform

The communication and collaboration platform component in our product includes integrated messaging or discussion forums for team communication and tools for collaboration on project documents and code. We have some functionality and data/user input involves detailing the specific capabilities and interactions users can expect from the system. Here's a more detailed breakdown:

3.7.1 R5.1 Functionality

- **R5.1.1 Messaging System:** Real-time text messaging allows for instant communication between team members and faculty, including group chats and private messages. Supports text formatting and embedding images or links.
- **R5.1.2 File Sharing:** Secure uploading and downloading of files with support for various file types (documents, images, presentations). Includes version tracking and permission settings to control access.
- **R5.1.3 Discussion Forums:** Spaces for broader discussions, allowing for threaded conversations organized by topic to facilitate class-wide or team-specific discussions.
- **R5.1.4 Task Management:** Enables teams to create, assign, and track tasks and deadlines within the project scope, enhancing collaboration and accountability.

3.7.2 R5.2 Data/User Input

- **R5.2.1 Text Messages:** Includes the actual text input by users, formatting choices (bold, italics, etc.), and any embedded media or links.
- **R5.2.2 Files for Sharing:** Information on the files being shared, such as file name, type, size, version number, and metadata (e.g., author, upload date).
- **R5.2.3 Discussion Posts:** Text content of posts, associated topics or tags, attachments, and user engagement metrics (likes, replies).
- **R5.2.4 Task Details:** Task descriptions, assigned members, due dates, status updates, and any associated files or links.
- **R5.2.5 Meeting Schedules:** Dates, times and participants, along with any necessary access links or codes.

This details provides a more comprehensive view of the communication and collaboration platform, specifying how users interact with the system and the types of data it needs to process. This detailed description will help ensure the development team fully understands the requirements and expectations for this component of the capstone management system.

3.8 R6 For Milestone Tracking

Here is a description of functionality and data/user input for the milestone tracking component in our product:

3.8.1 R6.1 Functionality

- **R6.1.1 Milestone Creation:** Allows users to define and set key project milestones with specific objectives, including start and end dates.
- **R6.1.2 Progress Monitoring:** Enables real-time tracking of milestone completion status, with visual indicators for on-track, at risk, and delayed milestones.
- **R6.1.3 Deadline Alerts:** Automated notifications and reminders for upcoming or missed deadlines to ensure timely completion of tasks.
- **R6.1.4 Update and Revision:** Facilitates the updating of milestone details and deadlines, allowing for project flexibility and adjustments.
- **R6.1.5 Integration with Calendar:** Milestones automatically populate in the system's calendar, providing a visual timeline of the project's key dates and events.

3.8.2 R6.2 Data/User Input

- **R6.2.1 Milestone Descriptions:** Textual information outlining the purpose and objectives of each milestone.
- **R6.2.2 Deadlines:** Specific dates for the completion of milestones, including any changes or extensions.
- **R6.2.3 Completion Status:** User-updated information on the progress toward milestone achievement, which can be marked as not started, in progress, completed, or delayed.

This detailed breakdown will guide the development of a robust milestone tracking system, ensuring users have the tools necessary to effectively plan, monitor, and adjust their project milestones.

3.9 R7. For Feedback and Evaluation

Here is a description of functionality and data/user input for the feedback and evaluation component in our product:

3.9.1 R7.1 Functionality

- **R7.1.1 Continuous Feedback:** A platform for faculty to provide ongoing, incremental feedback on project progress, with the ability to attach comments to specific project parts or deliverables.
- **R7.1.2.Final Evaluation:** Allows for comprehensive final assessments of projects, including qualitative feedback and quantitative scores based on predefined criteria.
- **R7.1.3. Peer Review:** Facilitates structured peer review processes among students, enabling them to provide and receive feedback on project work.
- **R7.1.4 Rubric-Based Evaluation:** Incorporates customizable evaluation rubrics, allowing faculty and peers to rate projects against specific criteria and standards.
- **R7.1.5 Feedback Compilation and Access:** Automatically compiles feedback for easy access by students, enabling them to review and respond to comments and evaluations.

3.9.2 R8 Data/User Input

- **R8.1 Feedback Comments:** Textual feedback provided by faculty and peers, including suggestions, commendations, and areas for improvement.
- **R8.2 Evaluation Criteria:** Predefined metrics or criteria used for assessing projects, which can include innovation, technical quality, adherence to objectives, presentation, and collaboration.
- **R8.3 Scores:** Numerical ratings assigned based on the evaluation criteria, which may be aggregated to form an overall project score.

This detailed approach ensures that the feedback and evaluation system supports a comprehensive and constructive assessment process, enhancing learning and project outcomes.

3.10 R9 For Calendar and Notifications

Here is a description of functionality and data/user input for the calendar and notifications component in our product:

3.10.1 R9.1 Functionality

- **R9.1.1 Event Scheduling:** Enable users to schedule project milestones, deadlines, meetings, and other key events within a personalized calendar.
- **R9.1.2 Automatic Reminders:** System-generated reminders for upcoming events, customizable to user preferences for lead time and notification method (e.g., email, SMS, in-app notifications).
- **R9.1.3 Calendar Integration:** Ability to integrate with external calendar services (such as Google Calendar or Microsoft Outlook) for seamless synchronization of events.
- **R9.1.4 Event Modification and Cancellation:** Facilities for users to modify or cancel scheduled events, with automatic notification to affected parties.
- **R9.1.5 Sharing and Permissions:** Options for users to share their calendar with team members or faculty, including configurable view/edit permissions.

3.10.2 R9.2 Data/User Input

- **R9.2.1 Event Details:** Information about each event, including title, description, date, time, and location (virtual or physical).
- **R9.2.2 Reminder Preferences:** User-selected preferences for how and when to receive notifications about upcoming events.
- **R9.2.3 Calendar Sharing Settings:** User choices regarding who can view or edit their calendar, including specific permissions for different users or groups.

This provides the development team with clear guidelines on the requirements for the calendar and notifications component, ensuring it supports effective time management and communication within the capstone management system.

3.11 R10 For Reporting and Analytics

Here is a description of functionality and data/user input for the reporting and analytics component in our product:

3.11.1 R10.1 Functionality

- **R10.1.1 Custom Report Generation:** Enables users to create customized reports on project progress, including specific data points such as milestone completion, team participation, and overall project status.
- **R10.1.2 Analytics Dashboard:** Provides an interactive dashboard that displays analytics on system usage, including user engagement, project submission rates, and feedback effectiveness.
- **R10.1.3 Performance Metrics:** Tracks and reports on key performance indicators (KPIs) for projects, such as adherence to deadlines, quality of submissions, and peer evaluation scores.
- **R10.1.4 Data Filtering and Segmentation:** Offers advanced filtering options to analyze data by time period, project type, department, or any other relevant criteria.
- **R10.1.5 Export and Sharing Options:** Allows users to export reports and analytics in various formats (e.g., PDF, Excel) and share them with stakeholders through email or integrated platforms.

3.11.2 R10.2 Data/User Input

- **R10.2.1 Report Parameters:** Specific criteria selected by users for customizing reports, such as date range, project category, or specific KPIs.
- **R10.2.2 Analytics Criteria:** User-defined metrics and dimensions for analyzing system performance and user engagement.
- **R10.2.3 Export Options:** Preferences for report format, resolution, and delivery method, ensuring reports are accessible and useful to all stakeholders.

This detailed description will guide the development of a comprehensive reporting and analytics system, ensuring stakeholders have access to meaningful insights into project and system performance.

3.12 R11 External Interface Requirements

This section aims to discuss the performance needs, applications utilized, and security protocols utilized to exchange data across the different endpoints throughout the application. By utilizing the proper programs and protocol, both high performance metrics and mitigate security risks. An important note is that these systems are custom to change as the tech stack, design features and requirements, and scale requirements are finalized by the customer and development team.

3.12.1 R11.1 User Interface

The User Interface (UI) is an essential component of the application, since this is where the users will directly access and interact with the application. The CMS that will be developed will be a GUI-driven interface, since the landing page once the user logs in will be the user-dashboard. While this application will be tailored for use on a computer, the frontend frameworks like React.js and Angular both have mobile app/web development features enables for use by the development team. The team may also decide to directly use HTML5 pages for the application, which are web pages that are specifically designed for use on both computer and mobile devices, which would allow for more flexibility for the users.

3.12.2 R11.2 External System Interfaces

A variety of system interfaces may be used due to the complexity and variety of the features that will be implemented for the application. For the general web pages, a RESTful API would be a strong choice for organization because the flow of the application through the web pages would be easy to follow for both the development team and the users. Additionally, this structure helps with the flow of data throughout the website, with the dashboard having much of the data displayed, and web pages for the smaller scale features only exhibiting only the data that is central to the functionality of the feature.

3.12.3 R11.3 Data IO Interfaces

The safe and secure exchange of data throughout the application's infrastructure is pivotal to the application's success. Data can be sent through the different channels of the program through a number of means, whether that be through reading CSVs, pulling JSON files, or exchanging XML data. While these tools are used in a variety of circumstances, Django and its derivatives have QuerySets, which are functions that query the data straight from the database, and Django plugs into React.js and Angular. Because of this, utilizing files to transfer data would be inefficient, require additional hardware, and expose the program data to additional vulnerabilities. When using these plug-ins from the application's endpoints to the backend, security protocols for the data transfers will be necessary, which will be discussed in the security interfaces section.

3.12.4 R11.4 Hardware and Software Interfaces

The development of this program will not require the team to manage hardware services, eliminating the need for maintainers to handle hardware malfunctions. Since this project will most likely be hosted on a server, the maintenance team will deal with connectivity and other software interface issues. This will deal with keeping the system up-to-date, clearing any potential dependencies within the different frameworks that will be used to develop and deploy the application, and maintaining the database. The frameworks that have been selected for the application at this time are compatible with each other and have a history of being so, however this is something that could change in the future.

3.12.5 R11.5 Communication and Security Interfaces

Maintaining security on this project is vital, and the communication the application performs between the various platforms that are utilized presents the highest risk associated with the project. The program will be password protected, which reduces the risk of non-users accessing data in the program. Additionally, being password protected will help make sure that users do not get access to information that their access level should not be able to see. In Django and its derivatives, Secure Sockets Layer (SSL) encryption and Hypertext Transfer Protocol Secure (HTTPS) communication protocol is available to be utilized. For authentication, Django also has a built-in username-password authentication system, which eliminates the need for the development team to develop such service. This mitigates the potential exposure of unauthorized users accessing information that otherwise they would not get access to.

3.12.6 R11.6 Performance Interfaces

Monitoring the application throughout its lifecycle is important for scalability, performance and maintenance adjustments, and implementing features in the future. While optimized coding practices and designing optimal processes from the onset, adapting to scaling requests, and performance metrics on the website post launch will help the maintenance team optimize the application to the user preferences throughout the service life of the application. There are a number of performance interfaces that allow for tracking the performance of the application that is compatible with Django. Here is a list and a description of what each program provides:

- 1. Google Analytics:** This tracks the interactions of the website from the frontend/UI. This is often free to access for your specified application. This tool may not help optimize the application, however it gives free insight on the traffic

experienced by the application **2. Solarwinds Appoptics:** This application provides sophisticated visualizations and tracks performance across the platform. **3. Atatus:** Provides real-time metrics on the application. Utilizing realtime data will allow the maintenance team to address concerns in a quicker and more targeted manner. This will improve the customer's experience with the application

There are many applications and performance tracking programs that can be used to track this application. The best program to use will depend on the customer needs, scale of the application, and risks identified throughout the development cycle of the application.

3.13 R12 Non Functional Requirements

Nonfunctional requirements (NFRs) are essential for guiding the design and development of a software system. They specify system qualities and attributes that are not directly related to functionality but significantly impact the system's success. For the Capstone Management System, the non-functional requirements (NFRs) play a crucial role in ensuring the system's usability, reliability, performance, and security. Here are the NFRs categorized by concern:

3.13.1 R12.1 Performance Requirements

- **R12.1.1 Response Time:** The system should respond to user interactions within 2 seconds under normal load conditions. Page load times for the dashboard and other critical features should be less than 2 seconds.
- **R.12.1.2 Concurrent Users:** The system must support up to 500 concurrent users during peak usage without significant degradation of performance.
- **R12.1.3 Data Processing:** Project proposal approvals, feedback submissions, and report generation should not exceed 5 seconds under normal conditions.
- **R12.1.4 Load Testing:** System should be tested under simulated peak loads to ensure performance meets requirements.
- **R12.1.5 Monitoring and Alerting:** System performance should be monitored continuously with alerts for potential issues. Performance metrics should be logged for analysis and optimization.
- **R.12.1.6 Geographic distribution of users:** If users are spread across continents, factor in latency and network performance variations.
- **R.12.1.7 Peak usage periods:** Identify predictable times of high activity (e.g., submission deadlines) and ensure optimal performance during those times.
- **R12.1.8 Performance benchmarks:** Research similar systems and compare their performance metrics to establish reasonable targets.

3.13.2 R13 Security Requirements

- **R13.1 Authentication:** The system must implement secure authentication mechanisms for all users, including two-factor authentication for administrators. Protection against unauthorized access and data breaches is critical.
- **R13.2 Authorization:** Role-based access control (RBAC) should be strictly enforced, ensuring users can only access information and functionalities pertinent to their roles.
- **R13.3 Data Encryption:** All sensitive data, including user information and project documents, must be encrypted in transit and at rest.
- **R13.4 Audit Trails:** The system should maintain detailed audit logs of all user actions, accessible only to authorized administrators.

- **R13.5 Vulnerability Management:** System should be regularly scanned for vulnerabilities and patched promptly. Secure coding practices should be followed to minimize vulnerabilities.
- **R13.6 Penetration Testing:** System should be subjected to regular penetration testing to identify and address security weaknesses.
- **R13.7 Data Backup and Recovery:** Regular backups of system data should be performed and stored securely offsite. Disaster recovery plan should be in place to ensure quick recovery in case of system failure.
- **R13.8 Access Logging and Auditing:** Logs should be retained for a defined period and be tamper-proof.
- **R13.9 Privacy Compliance:** System should comply with all relevant data privacy regulations (e.g., GDPR, CCPA). User data should be collected and used responsibly with clear consent mechanisms.

3.13.3 R14 Software Quality Attributes

R14.1 Availability & Reliability Requirements:

- **R14.1.1 System Availability:** The target availability should be at least 99.5% outside of scheduled maintenance windows.
- **R14.1.2 Data Backup:** Regular backups of all system data, with a recovery point objective (RPO) of 24 hours and a recovery time objective (RTO) of 4 hours.
- **R14.1.3 Fault Tolerance:** The system should be designed to gracefully handle failures, with mechanisms to automatically recover from common errors without user intervention.

R14.2. Scalability Requirements:

- **R14.2.1. Horizontal Scaling:** The system architecture must support horizontal scaling to accommodate growth in users and data volume.
- **R14.2.2 Load Balancing:** Implement load balancing to distribute traffic evenly across servers, ensuring no single point of failure.

R14.3. Usability Requirements:

- **R14.3.1. User Interface:** The interface should be intuitive and accessible, following best practices for UI/UX design. It should be accessible to users with disabilities, adhering to WCAG 2.1 guidelines. Consistent design elements and clear labeling are essential.
- **R14.3.2. Documentation:** Comprehensive user and technical documentation should be provided, easily accessible through the system.

R14.4. Compatibility Requirements:

- **R14.4.1 Browser Compatibility:** The web interface must be compatible with the latest versions of major browsers (Chrome, Firefox, Safari, and Edge).
- **R14.4.2 Mobile Responsiveness:** The system should offer a responsive design, ensuring usability on smartphones and tablets across different operating systems.

R14.5. Maintainability and Extensibility Requirements:

- **R14.5.1. Code Maintainability:** The system should be developed with clean, modular, and well-documented code to facilitate easy maintenance and future enhancements. Updates and modifications should not disrupt existing functionality.
- **R14.5.2 API Extensibility:** Where applicable, the system should provide well-documented APIs to allow for integration with external systems and future extensions.

R14.6. Legal and Regulatory Requirements:

- **R14.6.1. Data Privacy:** The system must comply with relevant data protection regulations (e.g., GDPR, CCPA) in handling personal information of users.
- **R14.6.2. Intellectual Property:** It should respect and enforce intellectual property rights, ensuring that users can only share and access content for which they have rights.

R14.7. Environmental Requirements:

- **R14.7.1 Energy Efficiency:** The system should be designed for energy efficiency, minimizing the carbon footprint of its operation.

3.13.4 Casptone Management System Requirements

1. User Authentication and Authorization:

- Login process should be secure and use strong password encryption.
- Role-based access control should be granular and ensure appropriate permissions for each user role.
- System should support two-factor authentication for additional security.

2. Dashboard:

- Dashboard should be customizable and allow users to personalize their view.
- Information displayed on the dashboard should be accurate and up-to-date.
- Users should be able to filter and search for specific information.

3. Project Proposal and Approval:

- Proposal submission process should be streamlined and efficient.
- Review process should be transparent and provide clear feedback to students.
- System should track the status of each proposal and notify relevant stakeholders.

4. Project Repository:

- Repository should support multiple file types and versions.
- Access to files should be controlled based on user permissions.
- Repository should allow for easy collaboration and file sharing within teams.

5. Team Formation:

- System should make it easy for students to find potential teammates based on skills and interests.
- Teams should be able to set their own rules and guidelines.
- System should facilitate communication and collaboration among team members.

6. Milestone Tracking:

- Milestones should be easily defined and tracked within the system.
- System should send automated notifications and reminders for upcoming deadlines.
- Progress towards milestones should be visible to all stakeholders.

7. Feedback and Evaluation:

- System should provide a structured way for faculty to provide feedback to students.
- System should facilitate peer evaluation and feedback sharing among team members.
- Mentorship relation should be traceable and transparent for both parties.

8. Notifications and Reminders:

- Automated notifications should remind users of upcoming deadlines, meetings, and milestones.
- Reminders can be sent via email, SMS, or in-app notifications.

9. Collaboration Tools:

- Integrated messaging or discussion forums should facilitate team communication.
- Version control for project documents and code is essential.

10. Reporting and Analytics:

- Generate reports on project outcomes, student performance, and faculty assessments.
- Analytics can provide insights into usage patterns and system effectiveness.

11. Calendar Integration:

- Integrate with existing calendar systems (e.g., Google Calendar, Outlook) for scheduling.
- Display project milestones, sprints, and deadlines on users' calendars.

By addressing these non-functional requirements, the Capstone Management System will be well-equipped to provide a secure, reliable, and user-friendly experience for its stakeholders, while also being prepared for future growth and changes in technology and regulations.

TEST STRATEGY

A test strategy is a high-level document that outlines the approach, objectives, and overall plan for the testing phase of a software development project. It provides a framework for the testing process, ensuring that it aligns with the project's goals and objectives. A comprehensive test strategy for the components of Capstone Management System should include both critical unit tests and integration testing via pull requests and CI/CD, So we should follow certain steps for each component as we will see in the following.

4.1 Deployment Environments

Deployment environments are a critical component of the software development ecosystem. When testing the system, it is paramount to understand the infrastructure on which the system is running. The organization of the deployed components of the operational system can inform how the system behaves under certain circumstances. Under development the software will be deployed to several environments. An environment is a set of hardware and software systems on which the software under development or under test is placed to run. These environments differ depending various factors, including but not limited to:

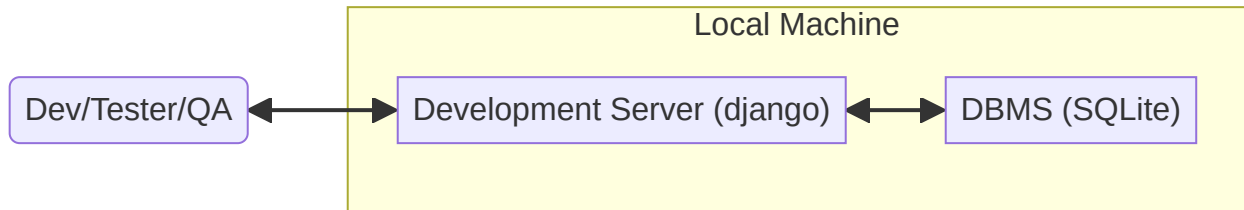
- Ease of code deployment.
- Tightness of the develop-build-test loop.
- Security of the deployed system.
- Accessibility of the deployed system.
- Comparability of the environment to production.
- Scalability of the system within the environment.

For an application with an extremely condensed development timeline, such as the CMS, the deployment environments should be kept simple and as similar as possible to avoid issues arising from differences in environments or complicated development setups. To this end, the tried and true approach of a web server and relational database management system (DBMS) makes sense for the CMS application. This architecture will easily deploy locally (on the developer's system), to a private server or VPS, or into the cloud. This leaves the deployment options for the CMS system open and make it easier to develop than more modern microservice oriented architectures.

4.1.1 Development Environment

The development environment of the CMS system is the environment developers will use to develop and test the software on their local machine. It consists of the developer's system (PC or laptop) capable of running a web server (django provide development server functionality) and a SQLite database (a local, file-base relational DBMS).

The flowchart below illustrates the structure of the development environment:

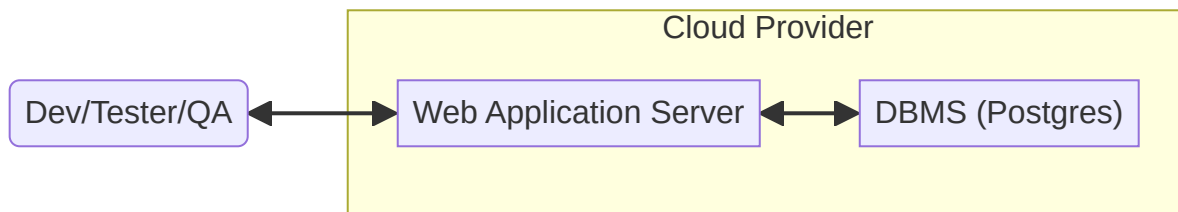


4.1.2 Testing/QA Environment

The testing/QA environment of the CMS system is the environment testers will use to perform integration, regression, and sell-off testing of the system. This system shall be set up in whichever cloud provider the production environment is set up in, but separated by some logical partition (e.g. in Azure a separate resource group or even subscription). This environment will be structured similarly to the production environment with a web application server (either VM or web app offering) and a fully-featured DBMS (likely PostgreSQL).

The construction and destruction of this environment can be automated to save on cloud service fees. The environment may be constructed when the testers need to perform tests, and removed when the tests are complete. For instance, at the end of a sprint, the test environment can be spun up (via CI/CD automation) and the latest development build of the CMS software could be deployed to it for the sprint demo.

The flowchart below illustrates the structure of the testing/QA environment:

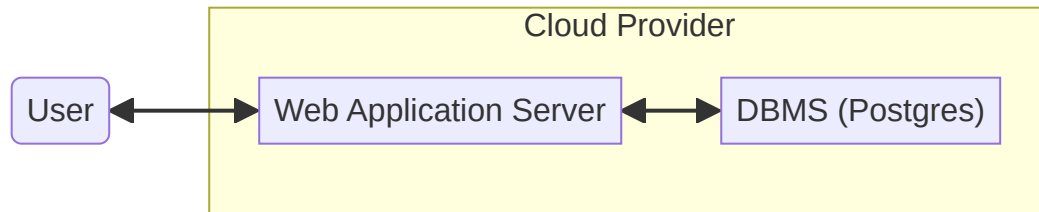


4.1.3 Production Environment

The production environment of the CMS system is the environment stakeholders will use to interact with the software and testers will use when necessary to recreate issues. This system shall be set up in a cloud provider (such as Azure). The environment will consist of a web application server (VM or web app offering) and a fully-featured DBMS. Other services, like caching or load balancing, can be brought online, as necessary (scaling).

In the extreme case of demand for scaling, future development efforts could focus on converting the monolithic structure of the CMS system into a microservices architecture. This would facilitate scaling to multiple institutions and thousands of users.

The flowchart below illustrates the structure of the production environment:



4.2 Critical Unit Tests

Critical unit tests are a subset of unit tests that focus on the most important and high-risk areas of the software. These tests target the key functionalities and components that are crucial for the system's operation and overall performance. The purpose of identifying and prioritizing critical unit tests is to ensure that the most essential parts of the software are thoroughly tested and validated early in the development process. In the following we will see the critical unit tests for our major components:

4.2.1 User Authentication

Test the collection of basic information with valid and invalid inputs. Unit test for email format validation and verification email or SMS sending process. Unit test for username and password validation (format, length, etc.). Unit test for role selection and assignment. Unit test for the role verification process by administrators. Unit test for the generation and validation of verification links or codes. Unit test for the activation of user accounts upon successful verification.

4.2.2 Dashboard

Unit test for the retrieval and display of project status, deadlines, and milestones. Unit test for the functionality of feedback and grade display. Unit test for the listing and status indication of supervised projects. Unit test for the tracking of student progress. Unit test for the generation and functionality of review and approval alerts. Unit test for the display and interaction with the review queue. Unit test for the reminders and functionality of feedback responsibilities.

4.2.3 Project Proposal and Approval

Unit test for the proposal submission form validation (required fields, file attachments, etc.). Unit test for the document attachment functionality (file size, format restrictions, etc.). Unit test for faculty access to review proposals (access control, proposal listing, etc.). Unit test for review tools functionality (adding comments, feedback submission, etc.). Unit test for the approval workflow logic (transition between stages, final approval, etc.).

4.2.4 Project Repository

Unit test for role-based access control functionality. Unit test for file upload functionality, including checks for file type and size restrictions. Unit test for file download functionality, ensuring that files are correctly retrieved and accessible to authorized users.

4.2.5 Team Formation

Unit test for keyword search functionality, verifying accurate search results. Unit test for the team creation process, ensuring all steps are functional and result in a complete team profile. Unit test for join request functionality, including sending, receiving, accepting, and declining requests. Unit test for the display of team profiles, verifying all necessary information is presented. Unit test for the listing view, ensuring all teams are listed with correct information.

4.2.6 Communication and Collaboration

Unit test for the real-time messaging functionality. Unit test for text messaging, including send, receive, and view operations. Unit test for media sharing capabilities, including integration with the repository component. Unit test for group chat creation and functionality. Unit test for contact list population and updating. Unit test for search and add contacts functionality.

4.2.7 Milestone Tracking

Unit test for milestone creation functionality, including input validation and data persistence. Unit test for the accuracy and updating of visual progress indicators. Unit test for deadline setting and modification, ensuring that changes are correctly saved and displayed.

4.2.8 Feedback

Unit test for peer feedback functionality, including the ability to submit, view, and manage feedback. Unit test for faculty evaluation functionality, ensuring that evaluations can be submitted and are accurately reflected in the system. Unit test for mentor feedback functionality, if applicable, to verify the submission and visibility of professional feedback.

4.2.9 Robust Calendar/Notifications and Reminders

Unit test for the automatic updating of the calendar with project milestones and deadlines. Unit test for scheduling and modifying team meetings. Unit test for sending deadline alerts and meeting reminders. Unit test for customization of notification types and frequencies. Unit test for the functionality of different calendar views. Unit test for syncing with external calendars and communication tools. Unit test for compatibility across platforms and devices. Unit test for rescheduling and modifying calendar events.

4.2.10 Reporting and Analytics

Unit test for the accuracy of project completion rate calculations. Unit test for the generation of quality metrics reports. Unit test for the accuracy of adherence to timeline reports. Unit test for the accuracy of grade reports. Unit test for the generation and accuracy of peer evaluations reports. Unit test for the accuracy of skill development reports.

4.3 Integration Testing Strategy

An integration testing strategy is a plan that outlines how individual software components will be combined and tested as a group to ensure that they work together correctly. The strategy defines the approach, methods, and tools that will be used to verify the interactions between different parts of the software system. We set up automated tests to run on every pull request that affects the all the 10 components of our product. Configure the CI/CD pipeline to automatically deploy changes to a staging environment for further testing. Use a combination of unit tests, integration tests, and end-to-end tests to cover the authentication workflow comprehensively.

4.4 Test Execution

Test execution is the phase in the software testing process where actual testing is carried out on the software application. During this phase, testers execute the test cases that have been prepared earlier in the testing lifecycle. The main activities involved in test execution include:

Setting up the Test Environment: Before executing the test cases, the testing environment needs to be set up. This involves configuring the hardware, software, network, and any other necessary components to create a setup that mimics the production environment.

Executing Test Cases: Testers run the test cases on the software application. This can be done manually, where testers follow the steps outlined in the test cases and compare the actual results with the expected results, or automatically, using test automation tools.

Logging Test Results: As test cases are executed, testers record the outcomes of each test case. This includes noting whether the test case passed or failed and documenting any discrepancies between the expected and actual results.

Reporting Defects: If any test case fails or reveals a defect in the software, testers log the issue in a defect tracking system. They provide detailed information about the defect, including steps to reproduce the issue, the expected and actual results, and any relevant screenshots or logs.

Retesting and Regression Testing: Once defects are fixed, testers retest the specific functionalities to ensure that the fixes are successful. Regression testing is also performed to check that the changes have not introduced new issues in other parts of the software.

Tracking Test Progress: Throughout the test execution phase, testers and test managers monitor and track the progress of testing. This includes keeping track of the number of test cases executed, passed, failed, and the status of defects.

Test execution is a critical phase in the testing process, as it directly involves validating the functionality and performance of the software application. The results of test execution provide valuable insights into the software's quality and readiness for release. We execute the test cases and unit tests as part of the pull request validation process and during the CI/CD pipeline runs.

4.5 Reporting

Reporting refers to the process of documenting and communicating the results, findings, and status of testing activities to stakeholders. Reporting is an essential part of the test strategy as it provides transparency, accountability, and insights into the quality of the software being tested. In the Reporting phase of our test strategy we prepare a test report summarizing the results, including any defects found and their severity. Report the results of the integration tests as part of the pull request review process and include test results in the CI/CD pipeline logs.

4.6 Regression Testing

Regression testing is a type of software testing that is performed to ensure that recent changes, such as code modifications, bug fixes, or enhancements, have not adversely affected the existing functionality of the software. The main goal of regression testing is to confirm that the software still works as expected after these changes and that no new defects have been introduced. In our testing strategy we conduct regression testing to ensure that changes to all the components do not negatively impact other parts of the system.

4.7 Continuous Monitoring

Continuous monitoring in a testing strategy refers to the ongoing process of observing, tracking, and analyzing the performance and stability of a software application throughout its development lifecycle and in production. The goal of continuous monitoring is to proactively identify and address potential issues before they impact the end-users or the system's functionality. Key aspects of continuous monitoring in a testing strategy include:

4.7.1 Performance Metrics

Monitoring key performance indicators (KPIs) such as response times, throughput, resource utilization, and error rates to ensure that the application meets performance standards.

4.7.2 System Health

Tracking the health and status of the application's infrastructure, including servers, databases, and network components, to detect any signs of degradation or failure.

4.7.3 Error and Exception Logging

Capturing and analyzing logs for errors, exceptions, and warnings to identify patterns or recurring issues that need to be addressed.

4.7.4 User Experience Monitoring

Observing user interactions and feedback to detect any usability issues or areas for improvement.

4.7.5 Security Monitoring

Continuously scanning for security vulnerabilities, breaches, or suspicious activities to maintain the application's security posture.

4.7.6 Automated Alerts

Setting up automated alerts to notify relevant stakeholders when predefined thresholds or conditions are met, enabling quick response to potential issues. Dashboards and Reporting: Utilizing dashboards and reports to provide real-time visibility into the application's performance and health, facilitating data-driven decision-making.

Continuous monitoring is an integral part of a modern testing strategy, particularly in agile and DevOps environments, where rapid development cycles and frequent deployments are common. It helps ensure the reliability, performance, and security of the application in an ever-changing landscape.

4.8 Test Schedule

To create a test schedule for all 10 components of the Capstone Management System, we should follow a phased approach that aligns with the software development lifecycle. Here's a suggested schedule:

4.8.1 Unit Testing

Timing: Start during the development phase, as soon as individual units or components are ready for testing.

Components: Apply unit testing to all components (User Authentication, Dashboard, Project Proposal and Approval, Project Repository, Team Formation, Communication and Collaboration, Milestone Tracking, Feedback, Robust Calendar/Notifications and Reminders, Reporting and Analytics) to validate their individual functionalities.

4.8.2 Integration Testing

Timing: Begin after unit testing, once multiple components are integrated.

Components: Focus on components with dependencies, such as Communication and Collaboration (integration with Project Repository), Team Formation (integration with User Authentication), and Dashboard (integration with all components).

4.8.3 System Testing

Timing: Conduct after integration testing, when the entire system is complete.

Components: Test the entire system as a whole, including all components, to ensure they work together seamlessly.

4.8.4 User Acceptance Testing (UAT)

Timing: Perform after system testing, before the final deployment.

Components: Involve end-users (students, faculty, mentors, administrators) to validate the system against their requirements, focusing on user-facing components like Dashboard, Team Formation, and Communication and Collaboration.

4.8.5 Regression Testing

Timing: Conduct throughout the development cycle, especially after new features are added or bugs are fixed.

Components: Apply to all components to ensure that new changes do not adversely affect existing functionalities.

4.8.6 Performance Testing:

Timing: Execute in the later stages of development, before UAT.

Components: Focus on components that are critical for system performance, such as Project Repository (for file management) and Communication and Collaboration (for real-time messaging).

4.8.7 Security Testing:

Timing: Perform throughout the development cycle, with a focus on the final stages.

Components: Prioritize components that handle sensitive data, such as User Authentication and Project Repository.

4.8.8 Continuous Testing (CI/CD)

Timing: Implement throughout the development cycle as part of the CI/CD pipeline.

Components: Integrate automated tests for all components into the pipeline to ensure continuous validation with each code commit. By following this test schedule, we can ensure that each component of the Capstone Management System is thoroughly tested at the appropriate stage of development, leading to a robust and reliable final product.

4.9 Exit Criteria

Exit criteria in a testing strategy are the predefined conditions that must be met before testing can be considered complete and the software can be deemed ready for release or the next phase of development. These criteria are typically agreed upon by stakeholders, including project managers, developers, testers, and clients, to ensure that the software meets the required quality standards. The exit criteria for the testing strategy of the Capstone Management System, particularly with a focus on stakeholder approval, should include the following:

4.9.1 Functional Completeness

All planned features and functionalities for each component have been implemented and verified through testing.

4.9.2 Test Coverage

Ensure that all critical test cases and scenarios outlined in the test strategy have been executed and passed.

4.9.3 Defect Resolution

All critical and high-priority defects identified during testing have been resolved and retested.

4.9.4 User Acceptance Testing (UAT) Approval

The system has been validated by end-users (students, faculty, mentors, administrators) during UAT, and their feedback has been addressed.

4.9.5 Performance and Security

The system meets the performance benchmarks and security standards set forth in the requirements.

4.9.6 Documentation

All necessary documentation, including user manuals, technical guides, and test reports, is complete and up to date.

4.9.7 Regulatory Compliance

The system complies with any relevant legal and regulatory requirements.

4.9.8 Stakeholder Sign-off

Key stakeholders, including project sponsors, faculty representatives, and IT administrators, have reviewed and approved the final test results and system readiness.

4.9.9 Post-Deployment Monitoring Plan

A plan is in place for monitoring the system's performance and user feedback post-deployment to ensure continued satisfaction and address any emerging issues. Once these criteria are met, the system can be considered ready for production deployment, with the approval of all relevant stakeholders.

TEST CASE

The scope of the test cases for the Capstone Management System encompasses a comprehensive evaluation of its functionalities to ensure a seamless and efficient experience for all users, including students, faculty, administrators, and sponsors. Each test case is designed to validate both the functionality and performance aspects of the system, ensuring that it meets all specified requirements and provides a user-friendly, reliable, and secure platform for managing capstone projects. The test cases are written considering the Use cases list and requirements mentioned in the references section, of the document

5.1 Test cases for CMS

5.1.1 User Authentication and Authorization

1. User Registration

- **Test Case ID:** TC_USER_REG_01
- **Description:** Tests the User Registration Process
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure the user registration process is intuitive, accessible from the login page, and successfully creates a new user account.
- **Pre Conditions:** The tester has access to the internet and a modern web browser. The Capstone Management System is up and running.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* N/A
 - * *Task:* Open the login screen/menu.
 - * *Expected Result:* The login screen is displayed correctly on all modern web browsers.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Click on the “Register” button.
 - * *Expected Result:* The user is redirected to the user registration form.
 - **Step 3**
 - * *Input data:* Valid user information
 - * *Task:* Fill in the registration form with valid user details (name, email, password, etc.).

- * *Expected Result:* All fields accept input appropriately, and the password is masked.

- **Step 4**

- * *Input data:* N/A

- * *Task:* Click the “Submit” button to create a new account.

- * *Expected Result:* A successful registration message is displayed, and the user is redirected to a confirmation page or login page.

- **Step 5**

- * *Input data:* User credentials

- * *Task:* Attempt to log in with the newly created user credentials.

- * *Expected Result:* The user can log in successfully with the new account.

- **Test Conclusion:**

- Ensure all mandatory fields are tested for validation checks (e.g., email format, required fields).
 - Test the behavior when entering invalid data or leaving mandatory fields empty.
 - Check if the system prevents the registration of a user with an already existing email address.
 - Review the privacy policy and terms of service links are accessible and correct.

2. User logs in with OU 4x4

- **Test Case ID:** TC_USER_OU4x4_LOGIN_01

- **Description:** Tests the OU 4x4 Login Process for Different User Roles

- **Applicable for:** All modern web browsers

- **Test Objective:** To verify that the OU 4x4 login process is secure, redirects unauthorized sessions to the login page, authenticates users against the OU LDAP servers, and assigns initial permissions based on the user role.

- **Pre Conditions:** The tester has access to the internet and a modern web browser. The Capstone Management System is up and running, and the OU LDAP server is accessible.

- **Test Steps:**

- **Step 1**

- * *Input data:* N/A

- * *Task:* Access the system without logging in.

- * *Expected Result:* The user is redirected to the login page due to an unauthorized session.

- **Step 2**

- * *Input data:* OU 4x4 credentials for a Student

- * *Task:* Log in using a student’s OU 4x4 credentials.

- * *Expected Result:* Authentication is successful, and the student is redirected to the dashboard with minimal permissions.

- **Step 3**

- * *Input data:* OU 4x4 credentials for a Teacher

- * *Task:* Log in using a teacher’s OU 4x4 credentials.

- * *Expected Result:* Authentication is successful, and the teacher is redirected to the dashboard with minimal permissions initially.

– Step 4

- * *Input data:* OU 4x4 credentials for an Admin
- * *Task:* Log in using an admin's OU 4x4 credentials.
- * *Expected Result:* Authentication is successful, and the admin is redirected to the dashboard with admin-level permissions.

– Step 5

- * *Input data:* Invalid OU 4x4 credentials
- * *Task:* Attempt to log in with invalid OU 4x4 credentials.
- * *Expected Result:* Authentication fails, and an error message is displayed without granting access to the system.

– Step 6

- * *Input data:* Valid OU 4x4 credentials + Incorrect Password
- * *Task:* Enter valid OU 4x4 username with an incorrect password.
- * *Expected Result:* Authentication fails, and an error message is displayed without granting access to the system.

– Step 7

- * *Input data:* N/A
- * *Task:* Verify if the password input is masked during entry.
- * *Expected Result:* The password is masked, ensuring privacy during the login process.

– Step 8

- * *Input data:* OU 4x4 credentials for a Student
- * *Task:* Log in with two-factor authentication enabled (if applicable).
- * *Expected Result:* The system prompts for a second factor, and upon successful verification, the student is logged in and redirected to the dashboard.

• Test Conclusion:

- Ensure to test the login process across different user levels (Student, Teacher, Admin) to verify role-based access control.
- If two-factor authentication is implemented, additional steps should be added to verify its functionality.
- The system should enforce strong password encryption during the authentication process; however, this may need to be verified through code review or security testing tools rather than functional testing.

3. User logs in with email

- **Test Case ID:** TC_USER_EMAIL_LOGIN_01
- **Description:** Tests the Email Login Process for Sponsors
- **Applicable for:** All modern web browsers
- **Test Objective:** To verify that the email login process is secure, redirects expired or new sessions to the login page, authenticates users against the corporate sponsor database, and correctly establishes user sessions with appropriate dashboard redirection and permissions.
- **Pre Conditions:** The tester has access to the internet and a modern web browser. The Capstone Management System is up and running, and the corporate sponsor database is accessible.

- **Test Steps:**

- **Step 1**

- * *Input data:* N/A
 - * *Task:* Access the system without logging in.
 - * *Expected Result:* The user is redirected to the login page due to an expired or new session.

- **Step 2**

- * *Input data:* Email and password for a Sponsor
 - * *Task:* Log in using a sponsor's email and password.
 - * *Expected Result:* Authentication is successful, and the sponsor is redirected to the dashboard with sponsor-level permissions.

- **Step 3**

- * *Input data:* Invalid email or password
 - * *Task:* Attempt to log in with an invalid email or incorrect password.
 - * *Expected Result:* Authentication fails, and an error message is displayed without granting access to the system.

- **Step 4**

- * *Input data:* Valid email + Incorrect Password
 - * *Task:* Enter a valid email with an incorrect password.
 - * *Expected Result:* Authentication fails, and an error message is displayed without granting access to the system.

- **Step 5**

- * *Input data:* N/A
 - * *Task:* Verify if the password input is masked during entry.
 - * *Expected Result:* The password is masked, ensuring privacy during the login process.

- **Step 6**

- * *Input data:* Email and password for a Sponsor
 - * *Task:* Log in with two-factor authentication enabled (if applicable).
 - * *Expected Result:* The system prompts for a second factor, and upon successful verification, the sponsor is logged in and redirected to the dashboard.

- **Test Conclusion:**

- This test case focuses on the sponsor user role, verifying that sponsors can log in securely using their email and password.
 - Ensure that the login process employs strong password encryption to protect user credentials.
 - If two-factor authentication is implemented, it's important to verify its functionality during the login process to ensure an additional layer of security.
 - Role-based access control should be tested to confirm that sponsors receive the correct level of permissions upon logging in.

4. Role-Based Access Control

- **Test Case ID:** TC_RBAC_01
- **Description:** Tests Role-Based Access Control for User Permissions
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that the system correctly implements Role-Based Access Control (RBAC) by allowing Administrators and Faculty to set roles and group memberships for users, which in turn accurately defines their access levels and permissions within the system.
- **Pre Conditions:** The tester is logged into the system with Administrator privileges. Test accounts for Student, Faculty, and Admin roles are available for testing, along with predefined roles and permissions.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Admin account credentials
 - * *Task:* Log in as an Administrator.
 - * *Expected Result:* Successful login with access to user role and permissions settings.
 - **Step 2**
 - * *Input data:* Faculty account to modify
 - * *Task:* Assign a Faculty role to a user account.
 - * *Expected Result:* The system updates the account with Faculty permissions, allowing access to Faculty-specific functionalities.
 - **Step 3**
 - * *Input data:* Student account to modify
 - * *Task:* Assign a Student role to a user account and group memberships.
 - * *Expected Result:* The system updates the account with Student permissions and any additional permissions or restrictions based on group memberships.
 - **Step 4**
 - * *Input data:* Sponsor account to modify
 - * *Task:* Attempt to assign a role not applicable to Sponsor users.
 - * *Expected Result:* The system prevents the assignment of inappropriate roles to Sponsor accounts, maintaining role integrity.
 - **Step 5**
 - * *Input data:* Admin account to modify
 - * *Task:* Assign another Admin role to a user account.
 - * *Expected Result:* The system updates the account with Admin permissions, including access to sensitive functionalities like user management.
 - **Step 6**
 - * *Input data:* Any user account
 - * *Task:* Verify two-factor authentication settings per role.
 - * *Expected Result:* Two-factor authentication settings are configurable and enforceable based on the role, enhancing security for sensitive roles.
 - **Step 7**

- * *Input data:* Any user account
 - * *Task:* Log out and log back in as the modified user account.
 - * *Expected Result:* The modified user account has access permissions consistent with its newly assigned role and group memberships.
- **Test Conclusion:**
 - This test case ensures the system’s RBAC mechanism functions as intended, allowing for granular control over user access based on roles and group memberships.
 - It’s crucial to test various combinations of roles and groups to ensure the system accurately enforces permissions without overlap or conflicts.
 - Two-factor authentication testing within this context ensures that enhanced security measures do not interfere with the correct application of roles and permissions.

5.1.2 Dashboard

1. Centralized Overview

- **Test Case ID:** TC_DASH_01
- **Description:** Tests Dashboard Functionality for Different User Levels
- **Applicable for:** All modern web browsers
- **Test Objective:** To verify that the dashboard provides a centralized overview of capstone project status, deadlines, and relevant information accurately and is customizable per the requirements of students, faculty, and administrators.
- **Pre Conditions:** Test accounts for Student, Faculty, and Administrator roles are available for login. Each account has pre-existing capstone project data to display.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Student account credentials
 - * *Task:* Log in as a Student.
 - * *Expected Result:* Successful login with the dashboard displayed as the home page, showing the student’s capstone project(s) and a list/calendar of upcoming deadlines related to these project(s).
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Verify the presence of capstone project status on the dashboard.
 - * *Expected Result:* The dashboard accurately displays the status, percentage of completion, current phase, and lists any outstanding tasks for the student’s project(s).
 - **Step 3**
 - * *Input data:* N/A
 - * *Task:* Verify the presence of capstone deadlines on the dashboard.
 - * *Expected Result:* The dashboard accurately displays all upcoming deadlines, ensuring the student is aware of key dates for their project(s).
 - **Step 4**

- * *Input data:* Customization options
- * *Task:* Customize the dashboard view, the project status view, view of deadlines on the dashboard.
- * *Expected Result:* The student can personalize the dashboard view, adding or removing widgets/information as allowed, personalize how project status is displayed, personalize how deadlines are displayed (list or calendar view) and prioritize information that is most important to them.
- **Step 5**
 - * *Input data:* Search and filter criteria
 - * *Task:* Use dashboard search and filter features.
 - * *Expected Result:* The student can successfully search for specific project information and filter results accurately.
- **Step 6**
 - * *Input data:* Faculty account credentials
 - * *Task:* Log in as Faculty.
 - * *Expected Result:* Successful login with the dashboard displayed, showing a broader range of information than the student view.
- **Step 7**
 - * *Input data:* N/A
 - * *Task:* Verify dashboard customization capabilities for Faculty.
 - * *Expected Result:* Faculty can customize their dashboard view, focusing on information relevant to their supervisory role.
- **Step 8**
 - * *Input data:* Administrator account credentials
 - * *Task:* Log in as an Administrator.
 - * *Expected Result:* Successful login with the dashboard displayed, showing comprehensive system-wide information.
- **Step 9**
 - * *Input data:* N/A
 - * *Task:* Verify accuracy of information for Administrator view.
 - * *Expected Result:* The dashboard for Administrators accurately reflects up-to-date, system-wide capstone project statuses and deadlines.
- **Step 10**
 - * *Input data:* N/A
 - * *Task:* Test the dashboard's responsiveness to real-time updates.
 - * *Expected Result:* After making changes to project information, the dashboard updates in real-time to reflect these changes across all user levels.
- **Test Conclusion:**
 - This test case ensures the dashboard functions as a centralized, customizable, and up-to-date information hub for all users involved in the capstone project.
 - Real-time update testing is crucial to verify the dashboard's effectiveness in providing timely information.

- Cross-role testing validates that the dashboard’s customization, filtering, and search functionalities meet the unique needs of students, faculty, and administrators.

2. Quick Links

- **Test Case ID:** TC_DASH_02
- **Description:** Tests Quick Links Feature on Dashboard for All Users
- **Applicable for:** All modern web browsers
- **Test Objective:** To verify that the dashboard provides direct access to frequently used tools such as project proposal submissions, document repositories, team communication channels, and milestone tracking systems, and that these links are customizable, accurate, and up-to-date for all users.
- **Pre Conditions:** User accounts for Student, Faculty, and Admin roles have been set up, and relevant tools and systems for capstone management are operational.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* User account credentials (Student, Faculty, Admin)
 - * *Task:* Log in as a user.
 - * *Expected Result:* Successful login with the dashboard displayed, showing quick links to frequently used tools.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Verify the presence and functionality of quick links for project proposal submissions.
 - * *Expected Result:* Clicking on the quick link navigates the user to the project proposal submission tool, which loads successfully and is up-to-date.
 - **Step 3**
 - * *Input data:* N/A
 - * *Task:* Verify the presence and functionality of quick links for document repositories.
 - * *Expected Result:* Clicking on the quick link navigates the user to the document repository, which loads successfully and displays the latest documents and resources.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Verify the presence and functionality of quick links for team communication channels.
 - * *Expected Result:* Clicking on the quick link navigates the user to the team communication channel, which loads successfully and allows for real-time communication with team members.
 - **Step 5**
 - * *Input data:* N/A
 - * *Task:* Verify the presence and functionality of quick links for milestone tracking systems.
 - * *Expected Result:* Clicking on the quick link navigates the user to the milestone tracking system, which loads successfully and displays up-to-date project milestones.
 - **Step 6**
 - * *Input data:* Customization options

- * *Task*: Customize the quick links on the dashboard.
- * *Expected Result*: Users can personalize which quick links are displayed on their dashboard, ensuring direct access to the tools they use most frequently.
- **Step 7**
 - * *Input data*: Search and filter criteria
 - * *Task*: Use dashboard search and filter features to locate specific tools or information.
 - * *Expected Result*: Users can successfully search and filter the dashboard, quickly finding specific tools or information among their quick links.
- **Step 8**
 - * *Input data*: Administrator account credentials
 - * *Task*: Log in as an Administrator.
 - * *Expected Result*: Successful login with the dashboard displayed, showing comprehensive system-wide information.
- **Test Conclusion**:
 - This test case ensures that the dashboard’s quick links feature is functioning properly for all users, providing them with efficient and customizable access to the tools they need most for capstone project management.
 - The ability to customize which quick links are displayed allows users to tailor their dashboard to fit their unique needs and preferences, enhancing the overall usability and efficiency of the system.
 - The test covers the functionality, accessibility, and up-to-dateness of the links, ensuring that users have a seamless experience when navigating to different tools and systems from their dashboard.

5.1.3 Project Proposal and Approval

1. Proposal submission interface

- **Test Case ID**: TC_PROJPROP_01
- **Description**: Tests Proposal Submission Interface for Students
- **Applicable for**: All modern web browsers
- **Test Objective**: To verify that the project proposal submission interface is streamlined, efficient, and allows students to submit their proposals with all required details and attachments, and that the system tracks the proposal status and notifies stakeholders appropriately.
- **Pre Conditions**: Student accounts are set up, and the proposal submission period is open.
- **Test Steps**:
 - **Step 1**
 - * *Input data*: Student account credentials
 - * *Task*: Log in as a Student.
 - * *Expected Result*: Successful login with access to the project proposal submission section.
 - **Step 2**
 - * *Input data*: N/A
 - * *Task*: Navigate to the proposal submission section.

- * *Expected Result:* The proposal submission interface is displayed, with fields for project title, abstract, detailed description, objectives, methodology, expected outcomes, and special requirements/resources.

– **Step 3**

- * *Input data:* Proposal details
- * *Task:* Fill in the form fields with a project title, abstract, detailed description, objectives, methodology, expected outcomes, and any special requirements or resources needed.
- * *Expected Result:* All form fields accept input as expected without errors.

– **Step 4**

- * *Input data:* Relevant documents or images
- * *Task:* Attach supporting documents or images.
- * *Expected Result:* The system allows for the attachment of documents or images, with a confirmation of successful attachment.

– **Step 5**

- * *Input data:* Completed proposal form
- * *Task:* Submit the proposal.
- * *Expected Result:* The proposal is submitted successfully, with a confirmation message displayed to the student.

– **Step 6**

- * *Input data:* N/A
- * *Task:* Check the status of the submitted proposal.
- * *Expected Result:* The system displays the current status of the proposal, allowing the student to track its progress.

– **Step 7**

- * *Input data:* N/A
- * *Task:* Receive notification of proposal review outcome.
- * *Expected Result:* The student receives a notification (e.g., email or system notification) regarding the outcome of the proposal review, including any feedback from reviewers.

• **Test Conclusion:**

- This test case ensures that the proposal submission process is user-friendly, enabling students to submit their project proposals with all necessary information and attachments efficiently.
- It also verifies that the system effectively tracks the status of each proposal and communicates the review outcomes and feedback to students, ensuring transparency in the review process.
- Ensuring the system supports attachments is crucial for students to provide all necessary documentation and evidence to support their proposals.
- The ability to track the proposal status and receive notifications keeps students informed and engaged in the process, contributing to a more transparent and user-friendly experience.

2. **Faculty Review Interface**

- **Test Case ID:** TC_PROPREV_01
- **Description:** Tests Faculty Review Interface for Project Proposals

- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that faculty members can efficiently review submitted proposals, filter them by various criteria, and provide comments or request additional information from students, all while maintaining a transparent review process with clear feedback.
- **Pre Conditions:** Faculty accounts are set up, and students have submitted project proposals.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Faculty account credentials
 - * *Task:* Log in as a faculty member.
 - * *Expected Result:* Successful login with access to the project proposal review portal.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Navigate to the review portal.
 - * *Expected Result:* The portal displays all submitted project proposals.
 - **Step 3**
 - * *Input data:* Submission date, student name, project category
 - * *Task:* Use filters to sort or search for proposals.
 - * *Expected Result:* The system effectively filters the proposals based on the selected criteria, making it easier for faculty to find specific proposals.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Select a proposal to review.
 - * *Expected Result:* The selected proposal's details are displayed, including all information submitted by the student and any attached documents or images.
 - **Step 5**
 - * *Input data:* Comments or requests for additional information
 - * *Task:* Enter feedback or request additional information in the provided section.
 - * *Expected Result:* The system allows for the input of comments or requests, which can be saved or submitted to be sent to the student.
 - **Step 6**
 - * *Input data:* N/A
 - * *Task:* Submit feedback/request.
 - * *Expected Result:* The feedback or request is successfully submitted, and a notification is sent to the student regarding the faculty's comments or additional information request.
 - **Step 7**
 - * *Input data:* N/A
 - * *Task:* Check the status of reviewed proposals.

- * *Expected Result:* The system tracks and displays the status of proposals that have been reviewed, including any pending actions or responses from students.

- **Test Conclusion:**

- This test case verifies that the faculty review interface is functional, user-friendly, and facilitates an efficient review process by allowing faculty to filter and select proposals for review.
- It ensures the review process is transparent by enabling faculty to provide clear and constructive feedback or request additional information, which is directly communicated to students.
- The ability for faculty to track the status of their reviews and any pending student responses is crucial for maintaining an organized and efficient review process.
- This test case contributes to verifying the overall requirement that the project proposal submission and review process is streamlined, transparent, and facilitates clear communication between students and faculty.

3. Feedback loop

- **Test Case ID:** TC_FEEDBACK_01

- **Description:** Tests the Feedback Loop for Project Proposals between Faculty and Students Applicable for: All modern web browsers

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure that faculty can provide structured feedback on project proposals, students receive notifications of such feedback, and can view and respond to comments in a transparent and efficient manner.

- **Pre Conditions:** Faculty and student accounts are set up, project proposals have been submitted by students, and are ready for review by faculty.

- **Test Steps:**

- **Step 1**

- * *Input data:* Faculty account credentials
- * *Task:* Log in as a faculty member.
- * *Expected Result:* Successful login with access to the project proposal review portal.

- **Step 2**

- * *Input data:* N/A
- * *Task:* Navigate to a specific proposal needing feedback.
- * *Expected Result:* The proposal is displayed with options for entering feedback.

- **Step 3**

- * *Input data:* Comments, suggestions, improvements
- * *Task:* Provide structured feedback on the proposal.
- * *Expected Result:* The feedback is successfully saved to the proposal, ready for student review.

- **Step 4**

- * *Input data:* N/A
- * *Task:* Submit the feedback.
- * *Expected Result:* A notification is automatically generated and sent to the student associated with the proposal.

- **Step 5**

- * *Input data:* Student account credentials
- * *Task:* Log in as the student who submitted the proposal.
- * *Expected Result:* Successful login with a notification about new feedback.

– **Step 6**

- * *Input data:* N/A
- * *Task:* Navigate to the proposal feedback section.
- * *Expected Result:* The feedback provided by the faculty is displayed clearly, including any comments, suggestions, and requests for improvements.

– **Step 7**

- * *Input data:* Response to feedback
- * *Task:* Enter a response to the feedback or make necessary revisions to the proposal based on the feedback.
- * *Expected Result:* The response or revised proposal is successfully submitted back to the faculty for further review.

– **Step 8**

- * *Input data:* N/A
- * *Task:* Faculty receives notification of student response or revised proposal.
- * *Expected Result:* Faculty is notified, ensuring continuous communication and allowing for further action if necessary.

• **Test Conclusion:**

- This test case ensures the functionality and efficiency of the feedback loop between faculty and students within the project proposal process. It confirms that faculty can provide meaningful feedback, students are promptly notified of this feedback, and can easily access and respond to it.
- The ability for students to respond to feedback directly or through revisions and for faculty to be notified of these responses supports a transparent and efficient review process.
- Verifying this feedback loop is crucial for maintaining high-quality project proposals and ensuring that students have clear guidance and opportunities for improvement based on faculty expertise.
- This test case contributes to verifying the overall requirement that the project proposal submission and review process is streamlined, transparent, and facilitates clear communication between students and faculty.

4. **Revised submissions**

- **Test Case ID:** TC_REVISION_01
- **Description:** Tests the Revision Submission Process for Project Proposals by Students
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that students can submit revised project proposals efficiently when revisions are requested, with the system maintaining a link to the original submission for historical reference.
- **Pre Conditions:** Student accounts are set up, and initial project proposals have been submitted and reviewed by faculty with feedback indicating the need for revisions.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Student account credentials

- * *Task*: Log in as a student who has received feedback requesting revisions.
- * *Expected Result*: Successful login with access to the project proposal submission interface.
- **Step 2**
 - * *Input data*: N/A
 - * *Task*: Navigate to the proposal needing revision.
 - * *Expected Result*: The original proposal and received feedback are displayed with an option to submit a revision.
- **Step 3**
 - * *Input data*: Revised project proposal details
 - * *Task*: Update the proposal based on the received feedback and prepare for resubmission.
 - * *Expected Result*: The revision interface allows for inputting updated project details, ensuring all required fields are completed based on feedback.
- **Step 4**
 - * *Input data*: Relevant documents or images
 - * *Task*: Attach any additional supporting documents or images if necessary.
 - * *Expected Result*: The system successfully attaches the new documents or images to the revised proposal.
- **Step 5**
 - * *Input data*: N/A
 - * *Task*: Submit the revised proposal.
 - * *Expected Result*: The revised proposal is successfully submitted, and a confirmation of submission is displayed. The system maintains a link to the original submission for reference.
- **Step 6**
 - * *Input data*: N/A
 - * *Task*: Check the status of the revised proposal.
 - * *Expected Result*: The system shows the revised proposal as “Under Review” and indicates a clear linkage to the original submission, preserving the history of changes.
- **Step 7**
 - * *Input data*: N/A
 - * *Task*: Faculty receives notification of the revised submission.
 - * *Expected Result*: Faculty associated with reviewing the proposal are notified of the new submission, enabling them to review the revisions.
- **Test Conclusion:**
 - This test case ensures that the process for students to submit revisions to their project proposals is both streamlined and effective, with clear mechanisms for updating based on faculty feedback.
 - It is crucial that the system maintains a link between the original and revised submissions to preserve the history of changes, providing transparency and context for reviewers.
 - The ability for faculty to be promptly notified of revised submissions ensures that the review process can proceed efficiently, maintaining momentum in the project proposal and approval process.

5. Approval Workflow

- **Test Case ID:** TC_APPROVAL_01
- **Description:** Tests the Approval Workflow for Project Proposals
- **Applicable for:** All modern web browsers
- **Test Objective:** To verify that the approval workflow for project proposals functions correctly, from submission through to notification of faculty, and the provision of clear, trackable feedback and status updates to students.
- **Pre Conditions:** Student and faculty accounts are set up. The student has prepared a project proposal ready for submission.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Student account credentials
 - * *Task:* Log in as a student and submit a project proposal.
 - * *Expected Result:* Successful submission of the project proposal is confirmed, and the system sends an automated notification to designated faculty for review.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* As a faculty member, receive the notification for review.
 - * *Expected Result:* Faculty member logs in and finds the new proposal notification in their dashboard or email.
 - **Step 3**
 - * *Input data:* N/A
 - * *Task:* Faculty reviews the proposal.
 - * *Expected Result:* Faculty has the option to approve, reject, or request revisions, with each option easily executable within the system.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Faculty decides to request revisions.
 - * *Expected Result:* Student receives notification of the request for revisions along with faculty comments. The proposal status is updated to “Revisions Requested.”
 - **Step 5**
 - * *Input data:* Revised proposal details
 - * *Task:* Student submits the revised proposal.
 - * *Expected Result:* The system updates the proposal status to “Under Review” and notifies the faculty member of the resubmission.
 - **Step 6**
 - * *Input data:* N/A
 - * *Task:* Faculty approves the proposal.
 - * *Expected Result:* The proposal status is updated to “Approved,” and the student receives a notification of approval.

- **Step 7**

- * *Input data:* N/A

- * *Task:* Student tracks the status of their proposal.

- * *Expected Result:* Throughout the process, the student can view the current status of their proposal, including when it is under review, if revisions are requested, and when it is approved.

- **Test Conclusion:**

- This test case ensures that the project proposal approval workflow is efficient, transparent, and provides clear feedback and status updates to students.
 - The ability to track the proposal status is crucial for maintaining transparency and ensuring students are informed of their proposal's progress through the approval process.
 - Automated notifications are essential for prompting timely reviews and keeping both students and faculty informed throughout the process.

5.1.4 Team Formation

1. Team creation interface

- **Test Case ID:** TC_TEAM_01

- **Description:** Tests the Team Creation Interface

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure that students and faculty can easily navigate to the team management section, view existing teams, create new teams through the team creation form, and set team names successfully.

- **Pre Conditions:** User accounts for students and faculty are set up and logged into the system.

- **Test Steps:**

- **Step 1**

- * *Input data:* Student/Faculty account credentials

- * *Task:* Log in and navigate to the teams link in the navbar.

- * *Expected Result:* User is redirected to the team management section, showing a list of teams they're part of and options to filter teams seeking members.

- **Step 2**

- * *Input data:* N/A

- * *Task:* Click on the button to create a new team.

- * *Expected Result:* User is redirected to a team creation form.

- **Step 3**

- * *Input data:* Team name: "Innovators"

- * *Task:* Fill out the team creation form with a name for the team and submit.

- * *Expected Result:* The form is submitted successfully, and the team "Innovators" is created and listed in the team management section.

- **Step 4**

- * *Input data:* N/A

- * *Task:* Verify the newly created team's presence.
- * *Expected Result:* "Innovators" team should be visible in the team management section to the user who created it and to others if the team is set to advertise for members.
- **Step 5**
 - * *Input data:* N/A
 - * *Task:* Check for communication tools.
 - * *Expected Result:* System provides mechanisms (e.g., message board, chat feature) to facilitate communication and collaboration among team members.
- **Step 6**
 - * *Input data:* N/A
 - * *Task:* Verify team rules and guidelines setting.
 - * *Expected Result:* Team creator can set and edit the team's rules and guidelines, visible to all team members.
- **Test Conclusion:**
 - This test case ensures that the process of team formation is user-friendly and supports essential features like creating a team, setting team names, and facilitating initial communication and guideline setup.
 - The ability to filter and view teams looking for members is crucial for students seeking to join teams that match their skills and interests.
 - Ensuring that teams can set their own rules and guidelines supports the requirement for teams to manage their internal processes and expectations effectively.

2. Teammate search and invitation

- **Test Case ID:** TC_TEAM_02
- **Description:** Tests Teammate Search and Invitation Functionality
- **Applicable for:** All modern web browsers
- **Test Objective:** To validate that students can search for potential teammates based on skills and interests, and send invitations to join their team.
- **Pre Conditions:** User (student) is logged in, and at least one team exists that the student is a part of. The system contains profiles of other students with visible skills and interests.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Student account credentials
 - * *Task:* Log in and navigate to the team management section.
 - * *Expected Result:* User is in the team management section, able to view teams they are part of.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Select the team they wish to add members to.
 - * *Expected Result:* User is presented with the team's details and an option to invite new members.
 - **Step 3**
 - * *Input data:* Keywords: "Software Development"

- * *Task*: Use the teammate search functionality, entering specific skills or interests.
 - * *Expected Result*: The system displays a list of students whose profiles match the entered skills or interests.
 - **Step 4**
 - * *Input data*: Select students from the search result
 - * *Task*: Send invitations to join the team.
 - * *Expected Result*: Selected students receive an invitation to join the team, and the inviter sees a confirmation of sent invitations.
 - **Step 5**
 - * *Input data*: N/A
 - * *Task*: Check invited students' response.
 - * *Expected Result*: Invited students have the option to accept or decline the invitation. Upon acceptance, they are added to the team.
 - **Step 6**
 - * *Input data*: N/A
 - * *Task*: Verify team communication tools.
 - * *Expected Result*: The system provides communication tools (e.g., message board, chat feature) that are accessible to new team members for collaboration.
 - **Test Conclusion:**
 - This test case checks the functionality and user experience of searching for potential teammates, ensuring that the search feature effectively filters students by skills and interests.
 - It also tests the invitation mechanism, ensuring that students can send and receive team invitations, and that these invitations function correctly, allowing for team membership changes.
 - The inclusion of communication tools in the test case addresses the requirement for facilitating collaboration among team members, verifying that newly added members have access to these tools.
- ### 3. Team joining mechanism
- **Test Case ID**: TC_TEAM_03
 - **Description**: Tests Team Joining Mechanism for Students
 - **Applicable for**: All modern web browsers
 - **Test Objective**: To ensure that students can effectively search for and send join requests to teams, and that team leaders can review and respond to these requests.
 - **Pre Conditions**: User (student) is logged in. Multiple teams exist within the system, each with distinct project topics, required skills, and preferred roles.
 - **Test Steps**:
 - **Step 1**
 - * *Input data*: Student account credentials
 - * *Task*: Log in and navigate to the team search section.
 - * *Expected Result*: User is in the team search section, able to search for teams.
 - **Step 2**

- * *Input data:* Keywords: “Web Development”, Skills: “JavaScript”, Role: “Developer”
- * *Task:* Use the search functionality with filters for project topic, required skills, and preferred roles.
- * *Expected Result:* The system displays a list of teams that match the search criteria.

– **Step 3**

- * *Input data:* Select a team from the search results
- * *Task:* View detailed information about the team and send a join request.
- * *Expected Result:* The join request is sent to the team leader, and the student sees a confirmation message.

– **Step 4**

- * *Input data:* N/A
- * *Task:* Log in as the team leader of the selected team to check for join requests.
- * *Expected Result:* The team leader sees the join request from the student, including the student’s profile and qualifications.

– **Step 5**

- * *Input data:* Review student profile
- * *Task:* Decide whether to accept or decline the join request based on the student’s qualifications and team needs.
- * *Expected Result:* If accepted, the student receives a notification of acceptance and is added to the team. If declined, the student receives a notification of the decision.

– **Step 6**

- * *Input data:* N/A
- * *Task:* Verify the updated team composition for both the team leader and the new member.
- * *Expected Result:* The team’s member list now includes the new student, and the student has access to the team’s communication and collaboration tools.

• **Test Conclusion:**

- This test case validates the functionality of the team joining mechanism from both the perspective of the student sending the join request and the team leader receiving and reviewing the request.
- It ensures that the search and filter functionalities work as expected, allowing students to find teams that match their interests and skills.
- The test also checks that the system supports the decision-making process for team leaders, enabling them to review requests and manage their team’s composition effectively.
- Finally, it verifies that once a student is added to a team, they have access to necessary tools for communication and collaboration, fulfilling the requirement to facilitate teamwork.

4. **Team management system**

- **Test Case ID:** TC_TEAM_04
- **Description:** Verify Integration with Development and Collaboration Tools for Team Management
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that students in teams can effectively use integrated development and collaboration tools (like GitHub, Microsoft Teams) for project collaboration and task distribution.

- **Pre Conditions:** User (student) is logged into the system and is a member of a team. The team has projects that require collaboration and task management.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Student account credentials
 - * *Task:* Log in and navigate to the team's project management section.
 - * *Expected Result:* User is on the project management page specific to their team, showing options for tool integration.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Select the option to integrate GitHub for version control and collaboration.
 - * *Expected Result:* System guides the user through the process of linking a GitHub repository to the team's project.
 - **Step 3**
 - * *Input data:* GitHub repository details
 - * *Task:* Complete the GitHub integration process.
 - * *Expected Result:* The team's project page now displays the linked GitHub repository, allowing for code sharing and version control.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Select the option to integrate Microsoft Teams for communication.
 - * *Expected Result:* System guides the user through the process of linking a Microsoft Teams channel to the team's project.
 - **Step 5**
 - * *Input data:* Microsoft Teams channel details
 - * *Task:* Complete the Microsoft Teams integration process.
 - * *Expected Result:* The team's project page now includes a direct link to the Microsoft Teams channel, facilitating communication among team members.
 - **Step 6**
 - * *Input data:* Task assignments
 - * *Task:* Use the integrated tools to distribute tasks among team members.
 - * *Expected Result:* Team members receive notifications of their assigned tasks through the integrated Microsoft Teams channel, and can access the GitHub repository to collaborate on code.
 - **Step 7**
 - * *Input data:* N/A
 - * *Task:* Verify the functionality of integrated tools by updating a task status on GitHub and discussing progress in Microsoft Teams.
 - * *Expected Result:* Changes made in GitHub are reflected in the project management section, and team members can successfully communicate through Microsoft Teams about project updates.

- **Test Conclusion:**
 - This test case ensures that the system supports integration with external development and collaboration tools, which is crucial for effective team management and project execution.
 - It verifies that students can link their team's projects with tools like GitHub and Microsoft Teams, streamlining the collaboration process.
 - The test also checks that task distribution and progress tracking are facilitated through these integrated tools, aligning with the requirement to make team formation and management efficient and effective.
 - Additionally, it confirms that the system's interface is user-friendly, guiding students through the integration process without issues.

5.1.5 Project Repository

1. Secure file storage

- **Test Case ID:** TC_REPO_01
- **Description:** Verify Secure File Storage in Project Repository
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that the project repository supports secure storage of multiple file types and versions, with access controlled based on user permissions, facilitating easy collaboration and file sharing within teams.
- **Pre Conditions:** Users (students, faculty) are logged into the system and have an existing project team with a designated project repository.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Various file types (e.g., .docx, .pdf, .jpg, .zip)
 - * *Task:* Upload these files to the project repository.
 - * *Expected Result:* The system successfully stores all uploaded file types in the project repository.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Check file versioning by updating an already uploaded file.
 - * *Expected Result:* The system retains the original file as version 1 and the updated file as version 2, indicating support for multiple versions.
 - **Step 3**
 - * *Input data:* User credentials with different roles (student, faculty)
 - * *Task:* Log in with different roles and attempt to access the project repository.
 - * *Expected Result:* Access is granted or denied based on the user's role and permissions set within the repository, demonstrating controlled access.
 - **Step 4**
 - * *Input data:* A new document for collaboration
 - * *Task:* Share the document with specific team members within the repository.
 - * *Expected Result:* Selected team members receive a notification and can access the shared document, indicating ease of collaboration and file sharing.

- **Step 5**

- * *Input data*: Comments or edits on a shared document
- * *Task*: Collaborate on the shared document by adding comments or edits.
- * *Expected Result*: All changes and comments are visible to those with access permissions, facilitating collaboration.

- **Test Conclusion:**

- This test case checks the project repository's capability to handle various file types and maintain version control, which is essential for managing project documentation and resources effectively.
- It validates that access control mechanisms are in place, ensuring that files in the repository are only accessible to users with appropriate permissions, enhancing security and privacy.
- The test also verifies the repository's collaboration features, such as file sharing and collaborative editing, which are crucial for teamwork and project development.
- The successful completion of this test case confirms that the project repository meets the specified requirements for secure file storage, version control, controlled access, and collaboration.

2. File management system

- **Test Case ID**: TC_REPO_02

- **Description**: Verify the File Management System in Project Repository

- **Applicable for**: All modern web browsers

- **Test Objective**: To ensure that the file management system within the project repository effectively supports secure storage, organized collaboration, and easy access to project materials, catering to the complex needs of capstone projects through the lifecycle of the project.

- **Pre Conditions**: Users (students, faculty) are logged into the system and have an active project with a designated project repository.

- **Test Steps:**

- **Step 1**

- * *Input data*: Various file types (e.g., .docx, .pdf, .jpg, .zip)
- * *Task*: Organize these files into designated folders within the project repository based on project phases (e.g., Research, Development, Testing).
- * *Expected Result*: Files are successfully organized into folders, demonstrating the repository's support for complex project needs.

- **Step 2**

- * *Input data*: File with multiple versions
- * *Task*: Update a file and save as a new version within the same folder.
- * *Expected Result*: The system retains both the original and the updated file as separate versions within the folder, indicating support for version control.

- **Step 3**

- * *Input data*: User credentials with different roles (student, faculty, guest reviewer)
- * *Task*: Log in with different roles and access the project repository.
- * *Expected Result*: Access to files and folders is granted or denied based on user roles and permissions, ensuring controlled access.

- **Step 4**

- * *Input data*: New project materials for collaboration
- * *Task*: Upload new materials to the repository and share access with specific project team members.
- * *Expected Result*: Team members receive notifications and can access and collaborate on the shared materials, indicating effective collaboration support.

- **Step 5**

- * *Input data*: Search query for a specific document
- * *Task*: Use the repository's search feature to find a document based on name, type, or content.
- * *Expected Result*: The search returns accurate results, demonstrating the repository's support for easy access to project materials.

- **Test Conclusion:**

- This test case evaluates the file management system's ability to handle a variety of file types and organize them according to the project's structure, which is crucial for maintaining order and efficiency.
- It assesses the system's version control capability, ensuring that historical data is preserved and accessible for reference or audits.
- The controlled access test verifies that the system can adequately restrict or grant access based on predefined user permissions, which is critical for maintaining the integrity and confidentiality of project materials.
- Collaboration features are tested to confirm that team members can effectively share, access, and work on project materials together, which is essential for the dynamic nature of capstone projects.
- The search functionality check ensures that users can quickly find specific files or documents, enhancing the usability and efficiency of the project repository.
- Successful completion of this test case would indicate that the file management system within the project repository meets the complex needs of capstone projects, from secure storage and organized collaboration to easy access to project materials.

3. Upload and download

- **Test Case ID**: TC_REPO_03

- **Description**: Verify Upload and Download Capabilities in Project Repository with Git Integration

- **Applicable for**: All modern web browsers

- **Test Objective**: To ensure that the project repository supports upload and download capabilities for various file types, including documents and code, with Git integration for code management, facilitating a DevOps workflow for students within a team.

- **Pre Conditions**: Students are logged into the system, are part of an active project team, and have access to the project repository.

- **Test Steps**:

- **Step 1**

- * *Input data*: Document file (e.g., .docx, .pdf)
- * *Task*: Upload a document to the project repository.
- * *Expected Result*: The document is successfully uploaded, and team members are notified of the new addition.

- **Step 2**

- * *Input data:* Document file from repository
 - * *Task:* Download a previously uploaded document.
 - * *Expected Result:* The document is successfully downloaded, ensuring students can access and review project materials.
 - **Step 3**
 - * *Input data:* Source code file (.py, .js)
 - * *Task:* Upload source code to the repository with Git integration.
 - * *Expected Result:* The source code is successfully uploaded using Git, demonstrating the repository's support for DevOps workflows.
 - **Step 4**
 - * *Input data:* Updated source code file
 - * *Task:* Use Git commands to commit and push an updated version of the source code to the repository.
 - * *Expected Result:* The updated source code is successfully pushed to the repository, and version control is maintained, showcasing effective Git integration.
 - **Step 5**
 - * *Input data:* User credentials with different permissions (team member, non-team member)
 - * *Task:* Attempt to download a document or source code from the repository.
 - * *Expected Result:* Team members can successfully download files, while non-team members are denied access, verifying that file access is controlled based on user permissions.
 - **Step 6**
 - * *Input data:* New version of a document
 - * *Task:* Upload a new version of an existing document to the repository.
 - * *Expected Result:* The new version is successfully uploaded, and the repository maintains both the original and the new version, indicating support for multiple file versions.
 - **Test Conclusion:**
 - This test case evaluates the repository's capabilities to handle file uploads and downloads, which is crucial for sharing and accessing project materials among team members.
 - The integration with Git for source code management is specifically tested to ensure that students can utilize a DevOps workflow, enabling version control and efficient collaboration on code development.
 - The test also verifies that access to files is appropriately controlled based on user permissions, ensuring that only team members can access sensitive project materials.
 - The support for multiple file versions is assessed by uploading new versions of documents, an essential feature for tracking changes and maintaining the integrity of project materials over time.
 - Successful completion of this test case would indicate that the project repository effectively supports the collaborative needs of student teams, with robust upload, download, and version control capabilities, including seamless integration with Git for code management.
4. **Access control**
- **Test Case ID:** TC_REPO_04
 - **Description:** Verify Access Control in Project Repository
 - **Applicable for:** All modern web browsers

- **Test Objective:** To ensure that the project repository's access control mechanism effectively restricts or allows access to files based on user permissions, facilitating secure collaboration and file sharing within teams.
- **Pre Conditions:** Users (students, faculty, and administrators) are logged into the system and are part of or associated with active project teams.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* User credentials (student, team member)
 - * *Task:* Attempt to access files within their team's project repository.
 - * *Expected Result:* The student is able to access all files in their team's repository, demonstrating proper access control for team members.
 - **Step 2**
 - * *Input data:* User credentials (student, non-team member)
 - * *Task:* Attempt to access files in a project repository where they are not a team member.
 - * *Expected Result:* Access is denied, indicating that the repository correctly restricts access based on team membership.
 - **Step 3**
 - * *Input data:* User credentials (faculty, associated with the project)
 - * *Task:* Attempt to access files in a project repository where they are faculty advisors.
 - * *Expected Result:* The faculty member can access all files in the repository, showing that access control allows faculty oversight.
 - **Step 4**
 - * *Input data:* User credentials (faculty, not associated with the project)
 - * *Task:* Attempt to access files in a project repository where they have no association.
 - * *Expected Result:* Access is denied, confirming that faculty access is appropriately restricted to projects they are associated with.
 - **Step 5**
 - * *Input data:* User credentials (administrator)
 - * *Task:* Attempt to access files in any project repository.
 - * *Expected Result:* The administrator can access all files across repositories, validating that admin roles have overarching access for management and oversight purposes.
 - **Step 6**
 - * *Input data:* User credentials (team member)
 - * *Task:* Attempt to upload a new file type or version to their project repository.
 - * *Expected Result:* The upload is successful, and team members are notified, confirming that access control allows file contributions from team members.
 - **Step 7**
 - * *Input data:* Document sharing settings
 - * *Task:* Change access permissions for a specific document within a team's project repository.

* *Expected Result:* Access permissions are successfully updated, and the document's accessibility changes according to the new settings, demonstrating dynamic access control management.

- **Test Conclusion:**

- This test case focuses on validating the access control mechanisms of the project repository, ensuring that users can only access files for which they have the appropriate permissions.
- It tests various user roles (students, faculty, administrators) to verify that the repository enforces access control rules according to team membership, project association, and administrative oversight.
- By testing access denial for non-team members and non-associated faculty, it confirms the security and privacy of project materials.
- Successful completion of this test case would indicate that the project repository effectively secures project materials against unauthorized access, while facilitating collaboration among authorized users, by accurately implementing access control based on user permissions.

5.1.6 Collaboration Tools

1. Messaging system

- **Test Case ID:** TC_COMCOL_01
- **Description:** Verify Functionality of Integrated Messaging System
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that the integrated messaging system facilitates effective team communication among all user levels within the platform.
- **Pre Conditions:** Users are logged into the system and have established teams or groups within which they intend to communicate.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* Text message content
 - * *Task:* Send a message within a team group chat.
 - * *Expected Result:* The message is successfully posted, and all team members can view it, demonstrating the messaging system's basic functionality.
 - **Step 2**
 - * *Input data:* Text message with attachment
 - * *Task:* Post a message with an attached document in a team chat.
 - * *Expected Result:* The message and attached document are successfully posted, and all team members can download the attachment, showing the system supports sharing files through messages.
 - **Step 3**
 - * *Input data:* User credentials (student)
 - * *Task:* Attempt to access and send a message in a team chat where the student is a member.
 - * *Expected Result:* The student can access the chat and send messages, confirming student-level access to messaging functionality.
 - **Step 4**
 - * *Input data:* User credentials (faculty)

- * *Task*: Faculty sends an announcement message to all associated teams.
- * *Expected Result*: The announcement is successfully posted across all specified teams, indicating the messaging system supports faculty-to-student communication.
- **Step 5**
 - * *Input data*: User credentials (team member)
 - * *Task*: Attempt to create a private message thread with another team member.
 - * *Expected Result*: The private message thread is successfully created, and both participants can send and receive messages, validating the system supports private messaging.
- **Step 6**
 - * *Input data*: New discussion topic
 - * *Task*: Start a new discussion thread within a project's forum section.
 - * *Expected Result*: The new discussion thread is successfully created, and team members can post and reply to messages within it, demonstrating support for organized, topic-specific discussions.
- **Step 7**
 - * *Input data*: Search functionality
 - * *Task*: Use the messaging system's search function to find a specific message or discussion.
 - * *Expected Result*: The search returns accurate results, allowing users to quickly locate specific messages or discussions, confirming the effectiveness of the search functionality within the messaging system.
- **Test Conclusion:**
 - This test case assesses the integrated messaging system's ability to support various forms of communication, including group chats, file sharing, announcements, private messaging, and discussion forums.
 - It verifies that the system is accessible and functional across different user roles (students, faculty), facilitating appropriate communication channels for each.
 - By testing file attachment capabilities and the creation of new discussion threads, it ensures the platform supports comprehensive communication needs, including the sharing of resources and organized discussions on project-related topics.
 - Successful completion of this test case would indicate that the integrated messaging system effectively supports the collaboration and communication needs of teams within the platform, contributing to an efficient and cohesive project development environment.

2. File sharing

- **Test Case ID**: TC_COMCOL_02
- **Description**: Verify Functionality of File Sharing within Communication and Collaboration Tools
- **Applicable for**: All modern web browsers
- **Test Objective**: To ensure that the platform's collaboration tools support efficient file sharing with proper version control for all user levels.
- **Pre Conditions**: Users are logged into the system, are part of at least one team or group, and have files ready for sharing.
- **Test Steps**:
 - **Step 1**
 - * *Input data*: Document file (e.g., .docx)

- * *Task*: Share a document file in a team group chat or discussion forum.
- * *Expected Result*: The document is successfully shared, and all team members can access and download it, verifying basic file-sharing functionality.
- **Step 2**
 - * *Input data*: Source code file (e.g., .py)
 - * *Task*: Share a source code file within a project's discussion forum.
 - * *Expected Result*: The source code file is successfully shared, and team members can access and download it, demonstrating the system's ability to handle different file types.
- **Step 3**
 - * *Input data*: Updated document file
 - * *Task*: Upload a new version of a previously shared document file.
 - * *Expected Result*: The system recognizes the upload as a new version, maintains a version history, and allows team members to access both the new and previous versions, confirming version control capabilities.
- **Step 4**
 - * *Input data*: Large file (e.g., 100MB)
 - * *Task*: Attempt to share a large file with the team.
 - * *Expected Result*: The file is successfully shared, or an appropriate error message is displayed if the file exceeds the platform's size limits, testing the system's handling of file size limitations.
- **Step 5**
 - * *Input data*: User credentials (student)
 - * *Task*: A student shares a project-related file in their team's dedicated space.
 - * *Expected Result*: The file is successfully shared and accessible by all team members, verifying student-level access to file sharing functionalities.
- **Step 6**
 - * *Input data*: User credentials (faculty)
 - * *Task*: Faculty shares a file containing lecture notes with all associated project teams.
 - * *Expected Result*: The file is successfully shared across specified teams, indicating the system supports broad file sharing by faculty members.
- **Step 7**
 - * *Input data*: Confidential document with restricted access
 - * *Task*: Share a file with specified access permissions, restricting visibility to certain team members.
 - * *Expected Result*: Only the specified team members can access the file, testing the system's access control and permission settings for shared files.
- **Test Conclusion**:
 - This test case evaluates the platform's file-sharing capabilities within communication and collaboration tools, including handling different file types, version control, and access permissions.
 - By testing the sharing of various file types and sizes, it ensures the system's versatility and reliability in supporting project collaboration needs.

- Testing access control for shared files assesses the platform’s ability to handle sensitive or restricted documents, ensuring that file sharing is secure and respects privacy requirements.
- Successful execution of this test case would indicate that the platform’s collaboration tools effectively support file sharing, enhancing teamwork and project development processes.

3. Discussion Forums

- **Test Case ID:** TC_COMCOL_03
- **Description:** Verify Functionality of Discussion Forums within Communication and Collaboration Tools
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that the platform’s collaboration tools include fully functional discussion forums that support team communication and project collaboration for all user levels.
- **Pre Conditions:** Users are logged into the system and are part of at least one team or project group.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* N/A
 - * *Task:* Access the discussion forum feature from the dashboard.
 - * *Expected Result:* The user is able to navigate to and view the discussion forums associated with their teams or projects, verifying access to discussion forums.
 - **Step 2**
 - * *Input data:* Text message
 - * *Task:* Post a new message in a project discussion forum.
 - * *Expected Result:* The message is successfully posted, and visible to all team members, verifying basic posting functionality.
 - **Step 3**
 - * *Input data:* Question
 - * *Task:* Create a new discussion thread with a question regarding project methodology.
 - * *Expected Result:* The new thread is successfully created, and team members can reply, supporting threaded discussions for organized communication.
 - **Step 4**
 - * *Input data:* Reply to a post
 - * *Task:* Reply to an existing discussion post with additional information or clarification.
 - * *Expected Result:* The reply is successfully posted and correctly threaded under the original post, confirming the forum supports responses in threads.
 - **Step 5**
 - * *Input data:* Document file (e.g., .pdf)
 - * *Task:* Attach a file to a forum post for sharing with the team.
 - * *Expected Result:* The file is successfully attached and accessible for download by team members, verifying file attachment capabilities in forum posts.
 - **Step 6**
 - * *Input data:* User credentials (faculty)

- * *Task*: Faculty member initiates a discussion on upcoming project milestones.
- * *Expected Result*: The discussion is visible to all relevant team members, demonstrating that faculty can effectively communicate with students through forums.
- **Step 7**
 - * *Input data*: Search term
 - * *Task*: Use the search functionality within a forum to find discussions on a specific topic.
 - * *Expected Result*: Relevant discussion threads are successfully retrieved, verifying effective search functionality within forums.
- **Step 8**
 - * *Input data*: User credentials (student)
 - * *Task*: A student subscribes to notifications for a specific discussion thread.
 - * *Expected Result*: The student receives notifications for new posts in the subscribed thread, confirming subscription and notification functionalities.
- **Step 9**
 - * *Input data*: Sensitive topic discussion
 - * *Task*: Create a private discussion group for sensitive topics, inviting specific team members.
 - * *Expected Result*: The private group is successfully created, and only invited members can view and participate, testing privacy settings in forums.
- **Test Conclusion:**
 - This test case assesses the discussion forum feature of the platform's collaboration tools, ensuring it supports effective communication among team members.
 - By evaluating basic functionalities like posting, replying, and attaching files, it ensures that users can share information and collaborate on project discussions easily.
 - Successful execution of this test case would indicate that the platform's discussion forums are a robust tool for facilitating communication, collaboration, and information sharing among project teams.

5.1.7 Milestone Tracking

1. Project Milestones

- **Test Case ID**: TC_MILESTONE_01
- **Description**: Verify the ability to establish, track, and receive notifications for project milestones
- **Applicable for**: All modern web browsers
- **Test Objective**: To ensure that the system allows for the easy definition and tracking of project milestones, sends automated notifications for upcoming milestones, and makes progress visible to students, sponsors, and teachers.
- **Pre Conditions**: Users (students, sponsors, teachers) are logged into the system and have an active project created.
- **Test Steps**:
 - **Step 1**
 - * *Input data*: Milestone details (e.g., title, description, due date)
 - * *Task*: A student adds a new milestone to the project.

- * *Expected Result:* The milestone is successfully added to the project, visible in the project's milestone tracking section.
- **Step 2**
 - * *Input data:* N/A
 - * *Task:* A sponsor views the list of milestones for a project they are sponsoring.
 - * *Expected Result:* The sponsor can see all defined milestones, including details such as title, description, and due date, confirming visibility to sponsors.
- **Step 3**
 - * *Input data:* N/A
 - * *Task:* A teacher accesses the milestone tracking feature for a project they are overseeing.
 - * *Expected Result:* The teacher can view all milestones and track progress, confirming visibility to teachers.
- **Step 4**
 - * *Input data:* Due date approaching
 - * *Task:* System checks for milestones with approaching due dates.
 - * *Expected Result:* Automated notifications are sent to students, sponsors, and teachers about upcoming deadlines, verifying notification functionality.
- **Step 5**
 - * *Input data:* Milestone completion status
 - * *Task:* A student updates a milestone to “completed”.
 - * *Expected Result:* The milestone's status is updated successfully, and progress is reflected in the tracking system, visible to all stakeholders.
- **Step 6**
 - * *Input data:* N/A
 - * *Task:* A sponsor checks the progress towards milestones.
 - * *Expected Result:* The sponsor can see updated progress, including which milestones have been completed, verifying tracking functionality.
- **Step 7**
 - * *Input data:* Reminder settings
 - * *Task:* A teacher sets up custom reminder notifications for milestones.
 - * *Expected Result:* Custom reminder notifications are successfully set and will be sent according to the specified schedule, confirming customizable notification settings.
- **Step 8**
 - * *Input data:* Progress report
 - * *Task:* Generate a progress report for the project, including milestone status.
 - * *Expected Result:* A detailed progress report is generated, showcasing the status of each milestone, aiding in project overview and stakeholder updates.
- **Step 9**
 - * *Input data:* Milestone modification

- * *Task*: Modify the details of an existing milestone (e.g., extend the due date).
- * *Expected Result*: The milestone is successfully modified, and all stakeholders are notified of the change, ensuring all parties are up-to-date.

- **Test Conclusion:**

- This test case ensures that the system supports comprehensive milestone tracking, including the creation, modification, and completion of milestones.
- By testing the visibility of milestones to students, sponsors, and teachers, it verifies that all relevant stakeholders can monitor project progress.
- Customizable reminder settings and the ability to generate progress reports add flexibility and depth to the milestone tracking feature, enhancing project management capabilities.
- Successful execution of this test case would indicate that the system effectively supports the tracking of project milestones, facilitating better project management and stakeholder communication.

2. Task assignment

- **Test Case ID**: TC_MILESTONE_02
- **Description**: Verify the functionality for students to assign roles and tasks to team members in relation to project milestones
- **Applicable for**: All modern web browsers
- **Test Objective**: To ensure that students can effectively assign roles and tasks to each team member for specific milestones, with changes being visible and trackable within the system.
- **Pre Conditions**: Users (students) are logged into the system, have an active project created, and milestones are already defined.
- **Test Steps**:
 - **Step 1**
 - * *Input data*: Task details (e.g., task name, description, assigned team member, due date) related to a milestone
 - * *Task*: A student assigns a new task to a team member.
 - * *Expected Result*: The task is successfully added and linked to the specific milestone, visible in the project's task management section.
 - **Step 2**
 - * *Input data*: N/A
 - * *Task*: The assigned team member checks their task list.
 - * *Expected Result*: The new task appears in their task list, including all relevant details (task name, description, due date), confirming visibility to the assigned member.
 - **Step 3**
 - * *Input data*: Task completion status
 - * *Task*: The assigned team member updates the task to “completed”.
 - * *Expected Result*: The task's status is updated successfully, and this progress is reflected in the milestone's overall progress.
 - **Step 4**
 - * *Input data*: N/A

- * *Task*: A student views the milestone to check task completion status.
- * *Expected Result*: The student can see which tasks have been completed and which are still pending, providing a clear overview of progress towards the milestone.

– **Step 5**

- * *Input data*: Reminder settings
- * *Task*: A student sets up custom reminder notifications for tasks.
- * *Expected Result*: Custom reminder notifications are successfully set for tasks and will be sent according to the specified schedule to the responsible team members.

– **Step 6**

- * *Input data*: Task modification
- * *Task*: Modify the details of an existing task (e.g., change the due date or assigned member).
- * *Expected Result*: The task is successfully modified, and the assigned team member is notified of the change, ensuring all involved parties are up-to-date.

– **Step 7**

- * *Input data*: Role assignment
- * *Task*: Assign specific roles to team members for milestone completion.
- * *Expected Result*: Roles are successfully assigned, and each team member can view their roles and related tasks, ensuring clarity in responsibilities.

– **Step 8**

- * *Input data*: N/A
- * *Task*: The system checks for tasks with approaching due dates and sends automated notifications.
- * *Expected Result*: Automated notifications are sent to the responsible team members about upcoming task deadlines, verifying the notification functionality.

• **Test Conclusion:**

- This test case ensures that the system supports the assignment of roles and tasks to team members effectively, linked to specific milestones.
- By testing the visibility of assigned tasks and roles to each team member, it verifies that responsibilities are clearly communicated within the team.
- Successful execution of this test case would indicate that the system effectively supports task assignment and tracking in relation to milestones, facilitating better teamwork and project execution.

3. **Milestone progress tracking**

- **Test Case ID**: TC_MILESTONE_03
- **Description**: Verify the functionality for real-time tracking of milestone progress by students, sponsors, and teachers
- **Applicable for**: All modern web browsers
- **Test Objective**: To ensure that students, sponsors, and teachers can accurately track the progress made towards milestones in real-time, with appropriate details visible to each user type.
- **Pre Conditions**: Users are logged into the system, an active project with defined milestones is in place, and tasks are assigned to students.
- **Test Steps**:

– Step 1

- * *Input data:* N/A
- * *Task:* A student logs in to check the progress of assigned tasks within a milestone.
- * *Expected Result:* The student is able to see a list of tasks, including which ones are completed and which are pending, providing a detailed view of their contribution towards the milestone.

– Step 2

- * *Input data:* N/A
- * *Task:* A teacher logs in to view the progress of a group towards a milestone.
- * *Expected Result:* The teacher sees a progress percentage of the milestone based on the completion of student responsibilities or student-reported progress, giving a high-level overview of project advancement.

– Step 3

- * *Input data:* N/A
- * *Task:* A sponsor logs in to track project progress towards a milestone.
- * *Expected Result:* Similar to the teacher, the sponsor is presented with a progress percentage, allowing them to gauge the project's advancement towards completion.

– Step 4

- * *Input data:* Progress update
- * *Task:* A student updates the status of a task to “completed”.
- * *Expected Result:* The update is reflected in real-time across all user views, showing an updated progress percentage or task completion status.

– Step 5

- * *Input data:* Automated notification settings
- * *Task:* Check if the system sends an automated notification when a milestone is nearing its deadline.
- * *Expected Result:* Automated notifications are sent out to all stakeholders, reminding them of the upcoming deadline.

– Step 6

- * *Input data:* Milestone update
- * *Task:* A teacher or sponsor updates the milestone's expected completion percentage.
- * *Expected Result:* The update is reflected in real-time, and all stakeholders can see the adjusted expectations, ensuring everyone has the latest information.

– Step 7

- * *Input data:* Student report submission
- * *Task:* A student submits a progress report for their tasks.
- * *Expected Result:* The submitted report is accessible to teachers and sponsors, providing them with detailed insights into the students' progress.

– Step 8

- * *Input data:* Real-time dashboard
- * *Task:* Access the project's real-time progress dashboard.

- * *Expected Result:* The dashboard accurately reflects the current progress towards milestones, showcasing completed and pending tasks, as well as overall progress percentage.

- **Step 9**

- * *Input data:* Role-specific access
- * *Task:* Verify that each user type sees appropriate progress details relevant to their role.
- * *Expected Result:* Students see task-level detail, while teachers and sponsors see overall progress percentages, confirming role-based information visibility.

- **Test Conclusion:**

- This test case ensures that the system supports real-time progress tracking of milestones with details tailored to the user's role (student, teacher, sponsor), enhancing transparency and accountability.
- By testing the immediate reflection of progress updates across user types, it verifies the system's capability to provide real-time progress tracking.
- Successful execution of this test case would indicate that the system effectively supports comprehensive and real-time milestone progress tracking, facilitating improved communication and project management across all stakeholder groups.

4. Milestone analytics dashboard

- **Test Case ID:** TC_MILESTONE_04

- **Description:** Verify the functionality and effectiveness of the Milestone Analytics Dashboard for students, faculty, and administrators

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure the Milestone Analytics Dashboard provides comprehensive insights into project progress, deliverable completion rates, and team performance against predefined milestones.

- **Pre Conditions:** Users are logged into the system, multiple active projects with defined milestones and varying progress levels are present in the system.

- **Test Steps:**

- **Step 1**

- * *Input data:* N/A
- * *Task:* A student accesses the Milestone Analytics Dashboard.
- * *Expected Result:* The student views visual representations of their project's progress against milestones, including timelines and deliverable completion rates.

- **Step 2**

- * *Input data:* N/A
- * *Task:* A faculty member accesses the dashboard to view project progress across all their supervised projects.
- * *Expected Result:* The faculty member sees a comprehensive overview of each project's performance, with the ability to drill down into specific details for closer examination.

- **Step 3**

- * *Input data:* N/A
- * *Task:* An administrator uses the dashboard to monitor the overall progress and performance of capstone projects within the institution.

- * *Expected Result:* The administrator receives high-level analytics on project timelines, completion rates, and performance metrics, with the option to filter and compare different projects.

– **Step 4**

- * *Input data:* Filter settings
- * *Task:* Apply different filters (e.g., project topic, completion rate) to view specific data sets.
- * *Expected Result:* The dashboard updates in real-time to display data relevant to the selected filters, allowing users to tailor the information to their needs.

– **Step 5**

- * *Input data:* Progress update
- * *Task:* Check how the dashboard reflects real-time updates when a milestone is completed.
- * *Expected Result:* Once a project milestone is marked as completed, the dashboard immediately reflects this change, updating the visual data accordingly.

– **Step 6**

- * *Input data:* Data accuracy
- * *Task:* Compare dashboard data with actual project records for consistency.
- * *Expected Result:* The data presented on the dashboard matches the actual progress and performance records of projects, ensuring reliability and accuracy.

– **Step 7**

- * *Input data:* Usability
- * *Task:* Evaluate the ease of navigation and understanding of the dashboard features.
- * *Expected Result:* Users find the dashboard intuitive to use, with clear visual cues and easy navigation that allows them to quickly grasp project insights.

– **Step 8**

- * *Input data:* Customization
- * *Task:* Test the ability to customize dashboard views according to user preference.
- * *Expected Result:* Users are able to customize the dashboard layout and the type of data displayed, enhancing personal relevance and usability.

– **Step 9**

- * *Input data:* Export functionality
- * *Task:* Attempt to export data or reports from the dashboard for external use.
- * *Expected Result:* The dashboard provides an export function, allowing users to download data or reports for presentations, meetings, or further analysis.

• **Test Conclusion:**

- This test case ensures that the Milestone Analytics Dashboard serves as an effective tool for monitoring and analyzing the progress and performance of capstone projects against milestones.
- By verifying the dashboard's ability to provide real-time, accurate, and customizable insights, it confirms its utility in supporting informed decision-making and adjustments by students, faculty, and administrators.
- Successful execution of this test case would indicate that the Milestone Analytics Dashboard effectively meets the requirement for comprehensive milestone tracking and analysis within the Capstone Management System.

5.1.8 Feedback and Evaluation

1. Final Evaluation

- **Test Case ID:** TC_FEEDBACK_01
- **Description:** Verify the functionality and effectiveness of the Final Evaluation process for faculty providing feedback to students within the system
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure the system supports a structured way for faculty to provide comprehensive feedback and final evaluations to students, facilitating transparent and traceable mentorship and peer evaluation.
- **Pre Conditions:** Courses or projects with completed milestones are available. Faculty and students are registered in the system with appropriate permissions.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* N/A
 - * *Task:* Faculty member logs into the system and navigates to the Final Evaluation section.
 - * *Expected Result:* Faculty is presented with a list of students or teams eligible for final evaluation.
 - **Step 2**
 - * *Input data:* Evaluation criteria and scores
 - * *Task:* Faculty selects a student or team and completes the evaluation form, including feedback on performance, achievements, and areas for improvement.
 - * *Expected Result:* The system allows for detailed feedback and scoring based on predefined criteria.
 - **Step 3**
 - * *Input data:* N/A
 - * *Task:* Submit the final evaluation.
 - * *Expected Result:* The system successfully saves and submits the evaluation, sending an automated notification to the student or team.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Student logs in and navigates to the received evaluations section.
 - * *Expected Result:* The student can view the final evaluation feedback provided by the faculty.
 - **Step 5**
 - * *Input data:* N/A
 - * *Task:* Check the visibility of peer evaluations if applicable.
 - * *Expected Result:* Students can also see peer evaluations, fostering a transparent feedback environment.
 - **Step 6**
 - * *Input data:* Comments or response
 - * *Task:* Students respond to or acknowledge the feedback.
 - * *Expected Result:* The system records and notifies the faculty of the student's engagement with the feedback.

- **Step 7**

- * *Input data:* N/A
- * *Task:* Verify the traceability of the mentorship relation and feedback history.
- * *Expected Result:* Faculty and students can view the history of evaluations and feedback exchanges, ensuring transparency and traceability.

- **Step 8**

- * *Input data:* Accessibility
- * *Task:* Evaluate the ease of use and accessibility of the evaluation feature for both faculty and students.
- * *Expected Result:* Both user groups find the system intuitive, with straightforward navigation and clear instructions for completing and reviewing evaluations.

- **Step 9**

- * *Input data:* Confidentiality
- * *Task:* Confirm that evaluations are only accessible to the intended recipients and relevant faculty members.
- * *Expected Result:* The system ensures that evaluations are confidential, with access restricted to the student, their peers (if applicable), and their faculty mentor.

- **Step 10**

- * *Input data:* Report generation
- * *Task:* Faculty attempts to generate a summary report of evaluations for a course or project.
- * *Expected Result:* The system provides functionality to compile and export a summary report of evaluations, aiding in the overall assessment and feedback process.

- **Test Conclusion:**

- This test case ensures that the Final Evaluation process within the system is robust, user-friendly, and effective in providing meaningful feedback from faculty to students.
- By verifying the system's support for detailed evaluations, transparent peer feedback, and traceable mentorship relations, it confirms its utility in enhancing the learning and development experience.
- Successful execution of this test case would indicate that the system effectively meets the requirement for a structured feedback and evaluation mechanism, fostering a constructive and transparent educational environment.

2. Peer review

- **Test Case ID:** TC_FEEDBACK_02

- **Description:** Verify the functionality and effectiveness of the Peer Review process for facilitating evaluation and feedback sharing among team members within the system

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure the system supports a structured way for students to provide peer evaluations, enabling transparent feedback sharing and contributing to a comprehensive feedback loop among team members.

- **Pre Conditions:** Teams are formed, and projects or assignments are in progress or completed. Students are registered in the system with appropriate permissions to access team functions.

- **Test Steps:**

- **Step 1**

- * *Input data:* N/A
- * *Task:* Student logs into the system and navigates to the Peer Review section within their team or project workspace.
- * *Expected Result:* Student is presented with a list of team members eligible for peer review.
- **Step 2**
 - * *Input data:* Evaluation criteria and comments
 - * *Task:* Student selects a peer and completes the evaluation form, including feedback on contributions, collaboration, and areas for improvement.
 - * *Expected Result:* The system allows for detailed feedback based on predefined criteria and free-form comments.
- **Step 3**
 - * *Input data:* N/A
 - * *Task:* Submit the peer review.
 - * *Expected Result:* The system successfully saves and submits the review, sending an automated notification to the peer being reviewed.
- **Step 4**
 - * *Input data:* N/A
 - * *Task:* The reviewed peer logs in and navigates to the received reviews section.
 - * *Expected Result:* The peer can view the feedback provided by their teammate.
- **Step 5**
 - * *Input data:* Comments or response
 - * *Task:* The reviewed peer responds to or acknowledges the feedback.
 - * *Expected Result:* The system records and notifies the reviewer of the peer's engagement with the feedback.
- **Step 6**
 - * *Input data:* N/A
 - * *Task:* Faculty member logs in and checks the visibility of peer reviews for a given team or project.
 - * *Expected Result:* Faculty can view the peer reviews to monitor team dynamics and provide additional mentorship where needed.
- **Step 7**
 - * *Input data:* Confidentiality
 - * *Task:* Confirm that reviews are accessible only to the reviewer, the reviewed peer, and relevant faculty members.
 - * *Expected Result:* The system ensures that reviews are confidential, with access restricted appropriately.
- **Test Conclusion:**
 - This test case ensures that the Peer Review process is implemented effectively, promoting a culture of constructive feedback and continuous improvement among team members.

- By verifying the system's support for detailed peer evaluations, confidential feedback exchanges, and traceable mentorship relations, it confirms its utility in enhancing team collaboration and individual development.
- Successful execution of this test case would indicate that the system effectively meets the requirement for a structured peer evaluation mechanism, fostering a supportive and transparent educational environment.

3. Rubric based evaluation

- **Test Case ID:** TC_FEEDBACK_03
- **Description:** Verify the functionality and effectiveness of the Rubric-Based Evaluation process for standardizing the assessment of capstone projects by faculty and external evaluators.
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure the system supports a structured, fair, and transparent evaluation process through predefined criteria within rubrics, facilitating clear and objective grading.
- **Pre Conditions:** Capstone projects are completed or in a state ready for evaluation. Faculty and external evaluators have access to the system with permissions to evaluate projects.
- **Test Steps:**
 - **Step 1**
 - * *Input data:* N/A
 - * *Task:* Faculty/External evaluator logs into the system and navigates to the project evaluation section.
 - * *Expected Result:* Evaluator is presented with a list of projects ready for evaluation.
 - **Step 2**
 - * *Input data:* N/A
 - * *Task:* Select a project to evaluate.
 - * *Expected Result:* The system displays the project details along with the rubric-based evaluation form.
 - **Step 3**
 - * *Input data:* Scores and comments based on rubric criteria
 - * *Task:* Complete the rubric-based evaluation form by assigning scores and providing feedback for each criterion.
 - * *Expected Result:* The system allows for scores and comments to be inputted clearly and saves them as part of the evaluation.
 - **Step 4**
 - * *Input data:* N/A
 - * *Task:* Submit the completed rubric-based evaluation.
 - * *Expected Result:* The system successfully submits the evaluation and sends an automated notification to the student(s) associated with the project.
 - **Step 5**
 - * *Input data:* N/A
 - * *Task:* Student logs in and navigates to their project's feedback section.
 - * *Expected Result:* The student views the detailed feedback and scores based on the rubric from the evaluator.

- **Step 6**

- * *Input data*: Summary and action points
- * *Task*: Faculty/evaluator generates a summary report of the evaluation.
- * *Expected Result*: The system provides a comprehensive summary report highlighting strengths, areas for improvement, and overall scores.

- **Step 7**

- * *Input data*: Confidentiality
- * *Task*: Confirm that evaluations are accessible only to the evaluator, student(s), and relevant faculty members.
- * *Expected Result*: The system ensures that evaluations are confidential, with access restricted appropriately.

- **Step 8**

- * *Input data*: Consistency
- * *Task*: Evaluate multiple projects using the same rubric to ensure consistency.
- * *Expected Result*: The rubric-based evaluation process maintains consistency across different projects, ensuring fairness in assessment.

- **Test Conclusion:**

- This test case verifies that the Rubric-Based Evaluation feature meets the requirements for providing a structured, transparent, and consistent evaluation process.
- By ensuring that evaluations are based on predefined criteria, that feedback is detailed and accessible, and that the process supports mentorship and development, the system enhances the educational experience for both students and faculty.
- The successful execution of this test case would indicate that the system effectively facilitates fair and objective assessment, contributing to the overall quality and integrity of the capstone project experience.

5.1.9 Calendar and Notifications

1. Event scheduling

- **Test Case ID**: TC_CALENDAR_01
- **Description**: Validate event scheduling, sharing, permissions, and notification functionalities in the calendar feature for students and faculty.
- **Applicable for**: All modern web browsers
- **Test Objective**: To confirm that the calendar feature correctly schedules events, integrates with external calendars, allows for deadline adjustments, supports event completion marking, and sends timely notifications.
- **Pre Conditions**: Access to the system with appropriate permissions for students and faculty. Availability of external calendar systems for integration.
- **Test Steps**:
 - **Step 1**
 - * *Input data*: Course deadlines, Group milestones
 - * *Task*: Faculty sets course deadlines and group milestones in the calendar.

- * *Expected Result:* Deadlines and milestones are successfully added to the system's calendar and are visible to relevant students.
- **Step 2**
 - * *Input data:* N/A
 - * *Task:* Integrate with external calendar systems.
 - * *Expected Result:* The system offers options to integrate with Google Calendar, Outlook, etc., and syncs the added deadlines and milestones.
- **Step 3**
 - * *Input data:* Adjustment details
 - * *Task:* Faculty alters a course deadline.
 - * *Expected Result:* The altered deadline is updated in the system's calendar and synced with integrated external calendars. Notifications about the change are sent to affected students.
- **Step 4**
 - * *Input data:* N/A
 - * *Task:* Student views calendar events.
 - * *Expected Result:* Students can see all relevant deadlines and milestones on their calendar, including any changes.
- **Step 5**
 - * *Input data:* Event ID
 - * *Task:* Student marks a calendar event as completed.
 - * *Expected Result:* The system marks the event as completed and optionally sends a confirmation notification to the student and faculty.
- **Step 6**
 - * *Input data:* N/A
 - * *Task:* Automated notification settings for upcoming deadlines.
 - * *Expected Result:* The system sends automated notifications (e.g., email, SMS, in-app notifications) reminding users of upcoming deadlines and milestones.
- **Step 7**
 - * *Input data:* Permission settings
 - * *Task:* Verify that faculty can set and alter deadlines but students can only view and mark them as completed.
 - * *Expected Result:* Faculty have the ability to set, share, and alter deadlines while students can only view and mark events as completed, respecting permission settings.
- **Step 8**
 - * *Input data:* N/A
 - * *Task:* Check for reminders via chosen notification methods.
 - * *Expected Result:* Users receive reminders for upcoming events through their chosen method(s) (email, SMS, in-app notifications) at the set time before the event.
- **Step 9**

- * *Input data:* Shareability
- * *Task:* Faculty shares a milestone with a specific group or all students.
- * *Expected Result:* The shared milestone is visible only to the intended recipients, respecting shareability and privacy settings.

- **Test Conclusion:**

- This test case ensures that the calendar and notification system within the platform effectively facilitates the scheduling, sharing, alteration, and tracking of course-related deadlines and milestones.
- Successful execution of this test case would demonstrate the system's ability to integrate seamlessly with external calendars, respect user permissions, and provide timely and accurate notifications and reminders, enhancing the organizational aspect of course and project management for both students and faculty.

2. Calendar Integration

- **Test Case ID:** TC_CALENDAR_02

- **Description:** Validate the integration of the system's calendar with external calendar systems (Google Calendar, Outlook) and the effectiveness of notifications and reminders for students and faculty.

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure that the calendar feature successfully integrates with external calendars, accurately displays project milestones, sprints, and deadlines, and sends timely automated notifications.

- **Pre Conditions:** Users (students and faculty) must have accounts on external calendar systems (Google Calendar, Outlook) and access to email, SMS, or the application for notifications.

- **Test Steps:**

- **Step 1**

- * *Action:* User (student or faculty) connects their external calendar (Google Calendar, Outlook) to the system.
- * *Expected Result:* External calendar is successfully linked to the system.

- **Step 2**

- * *Action:* Add a project milestone, sprint, or deadline in the system.
- * *Expected Result:* Added items are visible in the system's calendar and synchronized with the external calendar.

- **Step 3**

- * *Action:* Modify a deadline in the system's calendar.
- * *Expected Result:* The change is reflected in both the system's and the external calendar.

- **Step 4**

- * *Action:* Check for automated notifications for an upcoming deadline.
- * *Expected Result:* Users receive notifications via their preferred method (email, SMS, in-app notifications) prior to the deadline.

- **Step 5**

- * *Action:* Disconnect the external calendar from the system.
- * *Expected Result:* The external calendar is successfully disconnected, and future changes in the system's calendar do not sync with the external calendar.

- **Test Conclusion:**

- This test case is designed to ensure seamless calendar integration between the system and external calendar services, providing users with a cohesive scheduling experience and reliable reminders for their academic commitments.

3. Cancellation

- **Test Case ID:** TC_CALENDAR_03
- **Description:** Validate the system's capability to modify and cancel events on the calendar and ensure the dissemination of appropriate notifications upon these changes to all relevant parties.
- **Applicable for:** All modern web browsers
- **Test Objective:** To verify that event modifications and cancellations are accurately reflected in the integrated calendar system and that automated notifications are sent to inform about the changes.
- **Pre Conditions:** Users have successfully integrated their accounts with external calendar systems (e.g., Google Calendar, Outlook) and have scheduled events visible on their calendars.
- **Test Steps:**
 - **Step 1**
 - * *Action:* User schedules a new event (e.g., project milestone meeting) and invites participants.
 - * *Expected Result:* The event is successfully added to the user's and participants' calendars.
 - **Step 2**
 - * *Action:* The event organizer modifies the event details (time, date, or location).
 - * *Expected Result:* The modifications are reflected accurately in both the system's calendar and the external calendars of all participants.
 - **Step 3**
 - * *Action:* Check for automated notifications regarding the event modification.
 - * *Expected Result:* All participants receive a notification (via email, SMS, or in-app notifications) detailing the modifications made to the event.
 - **Step 4**
 - * *Action:* The event organizer cancels the scheduled event.
 - * *Expected Result:* The event is removed from the system's calendar and the external calendars of all participants.
 - **Step 5**
 - * *Action:* Check for automated notifications regarding the event cancellation.
 - * *Expected Result:* All participants receive a notification (via email, SMS, or in-app notifications) informing them of the event cancellation.
- **Test Conclusion:**
 - This test case aims to ensure that the system effectively handles event modifications and cancellations, maintaining synchronization with external calendar systems and keeping all participants informed through timely and accurate notifications.

5.1.10 Reporting and Analytics

1. Report generation

- **Test Case ID:** TC_REPORT_01
- **Description:** Validate the system's ability to generate reports on project outcomes, student performance, and faculty assessments, and provide analytics insights into usage patterns and system effectiveness.
- **Applicable for:** All modern web browsers
- **Test Objective:** To ensure that users can generate and access comprehensive reports and analytics as required, with accurate data reflecting project outcomes, performance metrics, and system usage.
- **Pre Conditions:** Users must be logged into the system with sufficient permissions to access reporting and analytics features.
- **Test Steps:**
 - **Step 1**
 - * *Action:* Navigate to the “Reports and Analytics” section of the platform.
 - * *Expected Result:* User is presented with options for report generation and analytics insights.
 - **Step 2**
 - * *Action:* Select the option to generate a report on project outcomes.
 - * *Expected Result:* System prompts the user to specify parameters (e.g., time frame, project categories).
 - **Step 3**
 - * *Action:* Submit the request for a project outcome report with the specified parameters.
 - * *Expected Result:* System generates a report detailing project outcomes within the selected parameters and displays it to the user or offers a download link.
 - **Step 4**
 - * *Action:* Select the option to generate a report on student performance.
 - * *Expected Result:* System prompts the user to specify parameters (e.g., specific classes, time frame).
 - **Step 5**
 - * *Action:* Submit the request for a student performance report with the specified parameters.
 - * *Expected Result:* System generates a report detailing student performance metrics within the selected parameters and displays it to the user or offers a download link.
 - **Step 6**
 - * *Action:* Select the option to generate a report on faculty assessments.
 - * *Expected Result:* System prompts the user to specify parameters (e.g., faculty members, time frame).
 - **Step 7**
 - * *Action:* Submit the request for a faculty assessment report with the specified parameters.
 - * *Expected Result:* System generates a report detailing faculty assessments within the selected parameters and displays it to the user or offers a download link.
 - **Step 8**
 - * *Action:* Access the analytics dashboard to view insights into usage patterns and system effectiveness.

- * *Expected Result:* Analytics dashboard displays insights into usage patterns, engagement metrics, and system effectiveness, with visualizations and data summaries.

- **Test Conclusion:**

- This test case aims to confirm that the reporting and analytics functionality of the system works as intended, providing users with valuable insights into project outcomes, student performance, faculty assessments, and overall system usage. It is crucial that the data presented in reports and analytics are accurate and reflective of the actual usage and outcomes.

2. Performance analytics and metrics

- **Test Case ID:** TC_REPORT_02

- **Description:** Validate the functionality of performance metrics and analytics for faculty and administrators to monitor student contributions, group cohesion, and system utilization.

- **Applicable for:** All modern web browsers

- **Test Objective:** To ensure that the system can generate detailed reports and analytics on student performance, group dynamics, and feature utilization for faculty and administrator review.

- **Pre Conditions:** Projects, student groups, and users are properly set up in the system. Faculty and administrators have access rights to the reporting and analytics tools.

- **Test Steps:**

- **Step 1**

- * *Action:* Faculty logs into the system and navigates to the analytics dashboard.
- * *Expected Result:* The analytics dashboard loads successfully, displaying options for performance metrics and analytics.

- **Step 2**

- * *Action:* Faculty selects to view a report on group cohesion and student contributions for a specific project.
- * *Expected Result:* The system generates a detailed report showing each student's contributions, highlighting group dynamics and individual performances.

- **Step 3**

- * *Action:* Administrator logs into the system and navigates to the system utilization analytics section.
- * *Expected Result:* The system utilization analytics page loads successfully, displaying usage patterns and feature utilization statistics.

- **Step 4**

- * *Action:* Administrator selects to generate a report on feature utilization over the last quarter.
- * *Expected Result:* The system generates a report detailing the usage frequency of different features, identifying highly used features and those with minimal engagement.

- **Step 5**

- * *Action:* Administrator examines the report to identify any features that require maintenance or potential new features to implement based on app utilization.
- * *Expected Result:* The report provides actionable insights, identifying specific features that may require updates or suggesting areas for new feature development.

- **Test Conclusion:**

- This test case aims to verify that the reporting and analytics functionality meets the needs of faculty and administrators for monitoring student performance, assessing group dynamics, and evaluating system utilization. The ability to generate and review these reports is crucial for informed decision-making and the continuous improvement of the educational experience and platform effectiveness.

VERIFICATION AND VALIDATION

For the Capstone Management System (CMS), both verification and validation processes are critical to ensure the development and implementation of a system that meets the needs of its users—students, teachers, and administrators—while adhering to specified requirements. Below is an exploration of how these processes apply to the CMS.

6.1 Verification

Verification in the context of the CMS involves ensuring that the system is developed according to the specified requirements and design documents. This process includes a series of checks and tests to confirm that each feature of the CMS functions as intended, without any bugs or errors.

- **Test Case Execution:** This involves running predefined test cases that align with the system's requirements, as illustrated in the test cases written for various use cases like team formation, project repository, feedback and evaluation, calendar and notifications, and reporting and analytics. Each test case result is compared against expected outcomes to ensure the system's functionality aligns with the specifications.
- **Code Review and Analysis:** Code reviews by peers and continuous integration practices help verify that the software code meets quality standards and adheres to the project's coding guidelines. Static code analysis tools can also be employed to automatically detect potential issues.
- **System and Integration Testing:** Verifying that different modules of the CMS work together seamlessly. This includes testing integrations with third-party services like GitHub, Google Calendar, or email and SMS notification services.

6.2 Validation

Validation focuses on ensuring that the CMS fulfills the needs and expectations of its stakeholders, including students, faculty, and administrators. This process assesses the system's effectiveness in a real-world scenario and whether it achieves the intended outcomes.

- **User Acceptance Testing (UAT):** Stakeholders are involved in testing the system to ensure it meets their needs. This could include students forming teams and managing projects through the CMS, teachers tracking milestones and providing feedback, and administrators generating reports on student performance and system usage.
- **Feedback Loops:** Incorporating feedback from stakeholders during sprint reviews or at specific milestones. This feedback is critical for understanding whether the CMS aligns with user needs and expectations. Adjustments and enhancements can then be made accordingly.
- **Sprint Review and Sign-off:** During sprint reviews, stakeholders are presented with the functionalities developed during the sprint. This is a critical validation step where stakeholders confirm that the developed features meet their requirements and provide their approval or request changes.

Validation ensures that the CMS is not only built correctly according to specifications but is also the right product for its users, effectively supporting capstone project management, collaboration, and evaluation processes. By focusing on both verification and validation, the development team can deliver a CMS that is both technically sound and highly valuable to its intended users. A detailed description of verification and validation is given in the Group B's SQAP.

REPORT TEMPLATES

These are templates for test reports and GitHub issues.

7.1 Test Report Template

Tester Name: _____

Test ID:

Test Name:

Test Status:

Tester System Info:

SRS Requirements:

SRS Requirement	Requirement Status

Steps:

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.

10.

Results:

Conclusion

Actions Taken (if applicable)

7.2 GitHub Issue Template

Issues with software may be encountered at many times during the software development lifecycle and over the lifetime of the application. It is imperative to maintain a record of any bugs, defects, or regressions which have been discovered by developers, testers, and application users. It is also preferable that this collection be singular and searchable which reduces its management overhead.

GitHub issues offer a publicly accessible interface for stakeholders to search, create, and update issues. This feature is integrated with GitHub repositories and the GitHub pull request system which helps document the lifecycle of the issue; from reported, to backlog, to fixed, to merged, and finally closed.

When an issue is identified, whether it be during development, testing, or use of the software, the following template should be used on GitHub Issues. The “actions taken” section of the test report might include a link or reference to an opened GitHub issue for the issue(s) encountered during the test.

****Describe the bug****

A clear and concise description of what the bug is.

****Severity****

How much does this impact your ability to use the software:

- Totally Impaired (unable to use the feature as intended)

(continues on next page)

(continued from previous page)

- Slightly Impaired (feature does not perform as expected)
- Inconvenienced (feature performs as expected but is slow, difficult to use, or reports inaccurate information)
- Slight Problem (feature performs as expected, inconvenient, but there is a workaround)
- Cosmetic (visual glitch, misspelling, or other erroneous behavior that does not impact performance or useability of the feature)

****To Reproduce****
Steps to reproduce the behavior:

1. Go to '...'
2. Click on '....'
3. Scroll down to '....'
4. See error

****Expected behavior****
A clear and concise description of what you expected to happen.

****Screenshots****
If applicable, add screenshots to help explain your problem.

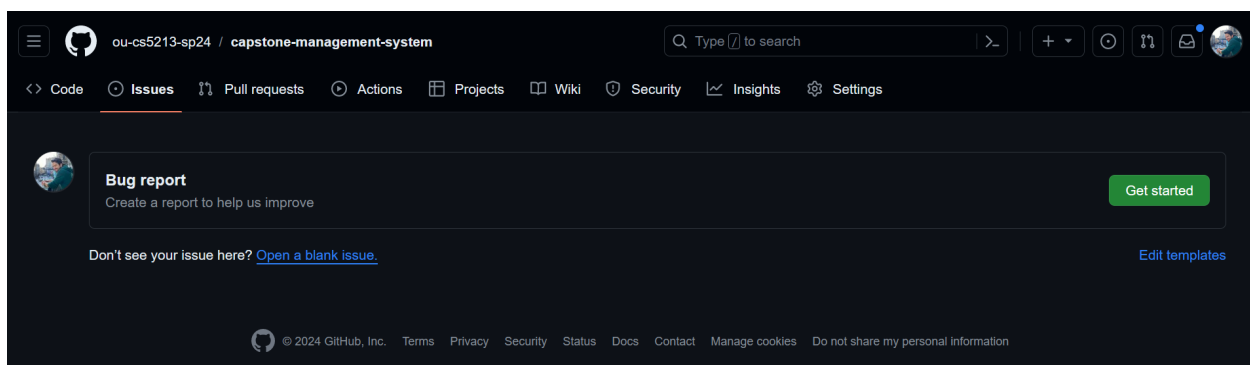
****System Info:****

- OS: [e.g. iOS]
- Browser [e.g. chrome, safari]
- Version [e.g. 22]

****Test ID:****
If applicable, the test id for the test being performed when the issue was encountered.

****Additional context****
Add any other context about the problem here.

When a GitHub user navigates to the issues page (link can be shared for bug reporting), they can open a new issue with the bug report template:



When the user clicks the “Get started” next to “Bug report”, they will be presented with the issue template to fill out a new issue:

ou-cs5213-sp24 / capstone-management-system

Type to search

<> Code

Issues

Pull requests

Actions

Projects

Wiki


Security

Insights

Settings

Issue: Bug report

Create a report to help us improve. If this doesn't look right, [choose a different type](#).



Add a title

Title

Add a description

Write

Preview

H

B

I

≡

<>

🔗

☰

☰

☰

📎

@

📧

↩

📎

****Describe the bug****

A clear and concise description of what the bug is.

****Severity****

How much does this impact your ability to use the software:

- Totally Impaired (unable to use the feature as intended)
- Slightly Impaired (feature does not perform as expected)
- Inconvenienced (feature performs as expected but is slow, difficult to use, or reports inaccurate information)
- Slight Problem (feature performs as expected, inconvenient, but there is a workaround)
- Cosmetic (visual glitch, misspelling, or other erroneous behavior that does not impact performance or useability of the feature)

****To Reproduce****

Steps to reproduce the behavior:

1. Go to '...'
2. Click on '....'
3. Scroll down to '....'
4. See error

****Expected behavior****

A clear and concise description of what you expected to happen.

****Screenshots****

If applicable, add screenshots to help explain your problem.

****System Info****

- OS: [e.g. iOS]
- Browser [e.g. chrome, safari]
- Version [e.g. 22]

****Test ID****

If applicable, the test id for the test being performed when the issue was encountered.

****Additional context****

Add any other context about the problem here.

Markdown is supported

Paste, drop, or click to add files

Submit new issue

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Assignees

No one—[assign yourself](#)

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Shows branches and pull requests linked to this issue.

Helpful resources

[GitHub Community Guidelines](#)

CHAPTER

EIGHT

TRACEABILITY MATRIX

Functional Requirement	Test Case						
	TC 3.1.1.1	TC 3.1.1.2	TC 3.1.1.3	TC 3.1.1.4	TC 3.1.2.1	TC 3.1.2.2	TC 3.1.3.1
<u>R1</u>							
R1.1.1	X						
R1.1.2	X						
R1.1.3							
R1.1.4							
R1.2.1	X						
R1.2.2	X						
R1.2.3	X						
R1.2.4	X						
R1.3.1	X						
R1.3.2	X						
R1.4.1	X						
R1.4.2	X						
R1.4.3	X						
<u>R2</u>							
<u>R2.1</u>		X	X				
<u>R2.2</u>		X	X				
<u>R2.3</u>		X	X				
<u>R2.4</u>		X	X				
R2.4.1				X		X	
R2.4.2				X		X	
R2.4.3				X		X	
R2.4.4				X		X	
R2.4.5				X		X	
R2.4.6				X		X	
R2.4.7				X		X	
R2.4.8				X		X	
R2.4.9				X		X	
R2.5.1	X						
R2.5.2	X						
R2.5.3	X						
R2.5.4	X						
R2.5.5	X						

R2.8.2				X	X		
R2.8.3				X	X		
R3							
R3.1.1						X	X
R3.1.2							X
R3.1.3							X
R3.2.1							
R3.2.2							
R3.2.3							
R3.3.1							
R3.3.2							
R3.3.3							X
R3.4.1							
R3.4.2							X
R3.5.1							X
R3.5.2							X
R3.6.1							X
R3.6.2							X
R3.7.1							X
R3.7.2							X
R3.7.3							X
R3.7.4							X
R3.7.5							X
R3.7.6.1							X
R3.7.6.2							X
R3.7.6.3							X
R3.7.6.4							X
R4							
R4.1.1						X	
R4.1.2							
R4.2.1							
R4.2.2							
R4.3.1							
R4.3.2							
R4.4.1							
R4.4.2							
R4.5.1							
R4.5.2							
R4.6							
R4.7							
R4.8							
R4.9.1.1							
R4.9.1.2							
R4.9.2.1							
R4.9.2.2							

R4.9.3.1							
R4.9.3.2							
R4.11.1.1							
R4.11.1.2							
R4.11.1.3							
R4.11.2.1							
R4.11.2.2							
R4.11.2.3							
R4.11.3.1							
R4.11.3.2							
R4.11.4.1							
R4.11.4.2							
R4.11.5.1							
R4.11.5.2							
<u>R5</u>						X	
R5.1.1							
R5.1.2							
R5.1.3							
R5.1.4							
R5.2.1							
R5.2.2							
R5.2.3							
R5.2.4							
R5.2.5							
<u>R6</u>							
R6.1.1						X	
R6.1.2							
R6.1.3							
R6.1.4							
R6.1.5							
R6.2.1							
R6.2.2							
R6.2.3							
<u>R7</u>							
R7.1.1							
R7.1.2							
R7.1.3							
R7.1.4							
R7.1.5							
<u>R8</u>							
R8.1							
R8.2							
R8.3							
<u>R9</u>						X	
R9.1.1							

Functional Requirement	Test Case					
FR's may be abbreviated from the previous list as to increase brevity across the tabled sections	TC 3.1.3.2	TC 3.1.3.3	TC 3.1.3.4	TC 3.1.3.5	TC 3.1.4.1	TC 3.1.4.2
R2.6.1	X	X	X	X	X	X
R2.6.2	X	X	X	X	X	X
R2.6.3	X	X	X	X	X	X
R3						
R3.1.1						
R3.1.2						
R3.1.3						
R3.2.1	X					
R3.2.2	X					
R3.2.3	X					
R3.3.1	X			X		
R3.3.2	X			X		
R3.3.3				X		
R3.4.1	X	X				
R3.4.2		X				
R3.5.1			X			
R3.5.2			X			
R3.6.1						
R3.6.2						
R3.7.1		X				
R3.7.2		X				
R3.7.3	X	X				
R3.7.4		X				
R3.7.5			X			
R3.7.6.1						
R3.7.6.2			X			
R3.7.6.3			X	X		
R3.7.6.4						
R4						
R4.1.1						
R4.1.2						
R4.2.1						
R4.2.2						
R4.3.1						
R4.3.2						
R4.4.1						
R4.4.2						
R4.5.1						
R4.5.2						

Functional Requirement	Test Case					
FR's may be abbreviated from the previous list as to increase brevity across the tabled sections						
	TC 3.1.4.3	TC 3.1.4.4	TC 3.1.5.1	TC 3.1.5.2	TC 3.1.5.3	TC 3.1.5.4
R2.6.1	X	X	X	X	X	X
R2.6.2	X	X	X	X	X	X
R2.6.3	X	X	X	X	X	X
R4						
R4.1.1			X	X		
R4.1.2			X	X		
R4.2.1			X	X	X	
R4.2.2				X	X	
R4.3.1			X		X	
R4.3.2					X	
R4.4.1			X	X	X	X
R4.4.2				X	X	X
R4.5.1		X				
R4.5.2		X				
R4.6				X		
R4.7		X	X	X	X	X
R4.8		X				
R4.9.1.1						
R4.9.1.2						
R4.9.2.1						
R4.9.2.2						
R4.9.3.1						
R4.9.3.2						
R4.11.1.1						
R4.11.1.2						
R4.11.1.3						
R4.11.2.1						
R4.11.2.2						
R4.11.2.3						
R4.11.3.1	X					
R4.11.3.2	X					
R4.11.4.1	X	X				
R4.11.4.2	X	X				
R4.11.5.1	X	X				
R4.11.5.2	X	X				

[illegible]

[illegible]

[illegible]

REFERENCES

1. <https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database>
2. <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>
3. <https://apps.dtic.mil/sti/tr/pdf/ADA301537.pdf>
4. [https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#:~:text=A software requirements specification \(SRS\) is a document that describes, stakeholders \(business%2C users\).](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#:~:text=A software requirements specification (SRS) is a document that describes, stakeholders (business%2C users).)
5. <https://www.geeksforgeeks.org/software-requirement-specification-srs-format/>
6. <https://www.javatpoint.com/software-requirement-specifications/>
7. <https://www.veraseti.com/insights/how-to-write-a-system-specification#:~:text=The SRS is a technical, its hardware and software requirements./>
8. [https://www.techtarget.com/searchsoftwarequality/definition/software-requirements-specification#:~:text=A software requirements specification \(SRS, will be expected to perform.](https://www.techtarget.com/searchsoftwarequality/definition/software-requirements-specification#:~:text=A software requirements specification (SRS, will be expected to perform.)
9. <https://forum.djangoproject.com/t/https-during-development/10509>
10. <https://forum.djangoproject.com/t/setting-up-ssl-in-settings-py/7147>
11. <https://docs.djangoproject.com/en/3.1/topics/security/#ssl-https>
12. <https://www.tek-tools.com/cloud/best-tools-to-monitor-django-performance>
13. Virtual Exam. 2023, December 4. CS619 Final Project | SRS | Software Requirement Specification | Functional Requirements |. Youtube. <https://www.youtube.com/watch?v=rAvlSHhodBw/>
14. GroupB_Ticket3_SRSCS5213spring2024.pdf