Software Requirements Specification

for

AssignMate

Version 1.0

Prepared by Maitree Borisagar

Roll No- IT014

Department of Information Technology

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document outlines the Software Requirements Specification (SRS) for an Assignment Submission & Review App. This application will facilitate the submission, review, and management of student assignments in a streamlined and efficient manner.

1.2 Document Conventions

Italics denote first use of a term.
Bold denotes emphasis.

1.3 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) is intended for a technical audience involved in the design and development of the IT Department Asset Management System. This document serves as a blueprint for the system's functionalities and features.

While this project is designed for academic purposes, adhering to professional SRS standards ensures clarity and lays a strong foundation for development. This document is primarily for your reference as the developer; however, it can also be used for the following purposes:

- **Project Guidance:** Throughout the development process, refer back to this document to ensure the implemented functionalities align with the intended system design.
- **Communication Tool:** Consider sharing this SRS with your instructor for feedback and to ensure project alignment with their expectations.

1.4 Project Scope

This project builds a web application, "Assignment Submission & Review App," for managing student code submissions.

Focus:

- Student Dashboard: Upcoming assignments, submission status (submitted, in review, etc.), optional past submission review.
- Submission: Public GitHub repo URL, branch selection (leveraging GitHub API - Bonus Feature).
- Review Process: Notifications for new submissions, claim system for reviewers, status updates.

- Bonus Feature: Unclaimed reviews revert to available after 24 hours.
- Feedback: Video review links, "complete" or "needs update" options.
- Rejected Assignments: Student notifications, re-submission with updated URL/branch.
- Review Completion**: Video review submission, student notification, status update to "completed."
- Admin API: User account creation for cohorts.
- Daily Batch Jobs: Reminders for stagnant reviews and revisions.

Exclusions:

- Advanced access controls
- External system integrations
- Plagiarism detection
- Downloadable materials, etc.

1.5 References

While this project aims to create a unique solution for assignment submission and review, it's valuable to consider existing platforms that address similar functionalities. Here are some references that may provide inspiration or insights during development:

- **Google Classroom:** A popular learning management system by Google that offers features for online assignment distribution, submission, and feedback [reference Google Classroom ON classroom.google.com]. While Google Classroom caters to a broader educational scope, its core functionalities for assignment management can be a valuable reference point.
- **GitHub Classroom:** An extension of GitHub that integrates with educational institutions to streamline code assignments and reviews. It leverages existing GitHub functionalities for version control and collaboration [reference GitHub Classroom ON classroom.github.com]. This can be particularly relevant considering your application's focus on code submissions via GitHub.
- **Turnitin:** A plagiarism detection tool often used in conjunction with assignment submission platforms. While not directly applicable to the core functionalities you're building, familiarizing yourself with plagiarism concerns in online learning can be beneficial for future iterations of the application [reference Turnitin website ON turnitin.com].

These references serve as starting points for exploration. Remember to prioritize your project's specific goals and functionalities while considering the strengths and limitations of existing solutions.

2. Overall Description

2.1 Product Perspective

The Assignment Submission & Review App is a web-based application designed specifically for educational institutions to manage student code submissions and instructor reviews. It is not intended as a commercial product but serves as a self-contained solution for this specific purpose. While it draws inspiration from existing platforms like Google Classroom and GitHub Classroom, it focuses on a more streamlined approach tailored for code submissions and reviews.

2.2 Product Features

- **Student Dashboard:** Provides an overview of upcoming assignments with due dates, optional access to past submissions, and the current status (pending submission, submitted, in review, needs update, completed) of ongoing submissions.
- **Submission Functionality:** Enables students to submit assignments by providing a public GitHub repository URL and selecting the appropriate branch containing their code.
- **Bonus Feature:** Leverages the GitHub API (optional) to verify repository validity and populate a dropdown menu with available branches from the provided repository.
- **Review Process:** Facilitates the review process by sending notification emails to reviewers upon new submissions and allows reviewers to claim assignments for review. The system becomes unavailable for others to claim once an assignment is claimed.
- **Bonus Feature:** Unclaimed reviews automatically revert back to available after 24 hours, allowing other reviewers to claim them.
- **Feedback Mechanism:** Enables reviewers to provide feedback through video recordings linked within the application and mark assignments as "complete" or "needs update" based on their evaluation.
- **Re-submission Process:** Provides students with the ability to re-submit assignments marked for updates. During re-submission, students can modify the provided GitHub URL and branch selection. Upon successful re-submission, the status changes back to "in review" and gets assigned back to the same reviewer (if applicable).

- **Review Completion:** Allows reviewers to submit their completed reviews by adding a URL to a video recording. A notification is then sent to the student informing them that their review is complete, and the assignment status is updated to "completed."
- **Admin API:** Provides an interface for administrators (e.g., instructors) to create user accounts for different student cohorts.
- **Daily Batch Jobs:** Automates sending reminder notifications for stagnant reviews (unclaimed for over 24 hours) and revisions (rejected assignments) requiring student attention.

2.3 User Classes and Characteristics

- **Students:** Enrolled in courses requiring code submissions. They will utilize the application to view assignments, submit their code via GitHub URLs, track submission status, and receive feedback from reviewers.
- **Code Reviewers:** Instructors or qualified personnel responsible for reviewing student code submissions. They will use the system to access new submissions, claim assignments for review, provide feedback through video recordings, and mark assignments as complete or needing revision.
- **Admins:** Personnel with access to manage user accounts through the Admin API. They will typically be instructors or course coordinators responsible for creating user accounts for their respective student cohorts.

2.4 Operating Environment

The application is designed as a web-based platform accessible through a standard web browser on various devices (desktops, laptops, tablets) with a stable internet connection. This ensures broad compatibility and eliminates the need for device-specific installations.

2.5 Design and Implementation Constraints

- Development tools and technologies chosen will prioritize scalability and maintainability to accommodate future growth and potential feature additions.
- Security measures will be a top priority to protect user data, code submissions, and reviewer feedback. Secure login protocols and data encryption will be implemented.

- The application design should be user-friendly and intuitive for all user classes (students, reviewers, admins), minimizing any complex learning curves.

2.6 User Documentation

The project will deliver comprehensive user documentation to guide all user classes through the application's functionalities. This documentation will likely include:

- **Student User Guide:** Explains how to navigate the student dashboard, submit assignments, track submission status, and receive feedback.
- **Code Reviewer Guide:** Provides instructions on claiming assignments for review, providing video feedback, marking submissions complete or needing revision, and managing their review workload.
- **Admin Guide:** Details the process for creating user accounts for different student cohorts using the Admin API.

The format of the user documentation will be determined during project development. Options include online help systems, downloadable PDFs, or a **1.7 Assumptions and Dependencies**

- The project assumes that users have basic computer literacy and internet access.
- The application's functionality relies on the availability and stability of the GitHub API for repository verification and branch retrieval (optional feature).
- The success of the project depends on instructor/administrator buy-in and their willingness to integrate the application into their existing workflows.

2.7 System Features

- The project assumes that users have basic computer literacy and internet access.
- The application's functionality relies on the availability and stability of the GitHub API for repository verification and branch retrieval (optional feature).
- The success of the project depends on instructor/administrator buy-in and their willingness to integrate the application into their existing workflows.

3. External Interface Requirements

3.1 User Interfaces

- **Web-Based Interface:** The application will be accessible through a standard web browser on various devices (desktops, laptops, tablets) with a stable internet connection.
- **User Interface Standards:** The UI will prioritize user-friendliness and consistency. A style guide will be developed to ensure a uniform look and feel across all screens.
- **Screen Layouts:** Layouts should be optimized for different screen sizes while maintaining clarity and ease of navigation.
- **Standard Elements:** All screens will include standard elements like a navigation bar, user profile information, and a clear indication of the current section/functionality.
- **Error Handling:** User-friendly error messages will be displayed in case of invalid inputs or unexpected system behavior. The messages will guide users towards resolving the issue.
- **Student Dashboard:**
    - Overview of upcoming assignments with due dates.
    - (Optional) Access to past assignment submissions.
    - Current status of ongoing submissions (pending submission, submitted, in review, needs update, completed).
- **Submission Functionality:**
    - Interface for entering a public GitHub repository URL.
    - Dropdown menu for selecting the relevant branch containing the code (optional, leverages GitHub API).
    - Submit button to initiate the submission process.
- **Review Process:**
    - Notification system to alert reviewers about new submissions.
    - Functionality for reviewers to claim assignments for review.
    - Indication of claimed/unclaimed status for each submission.
- **Feedback Mechanism:**
    - Option for reviewers to upload video recordings containing their feedback.

    - Buttons for marking assignments as "complete" or "needs update."
- **Handling Rejected Assignments:**
    - Notifications sent to students regarding rejected submissions.
    - Option for students to revisit the assignment for re-submission.
    - Interface for updating the provided GitHub URL and branch during re-submission.
- **Admin API:**
    - Web-based interface or API access point for creating user accounts for different student cohorts.

3.2 Hardware Interfaces

The application is designed to be web-based and does not have any specific hardware interface requirements beyond a standard internet connection and a device with a web browser.

3.3 Software Interfaces

- **Optional: GitHub API:** The application may leverage the GitHub API to verify the validity of provided public GitHub repository URLs and retrieve a list of available branches for selection during the submission process.
- **Database:** A database will be used to store user information (student, reviewer, admin), assignment details, submission data, and review records. The specific database technology will be chosen based on project needs and scalability considerations.

3.4 Communications Interfaces

- **Email Notifications:** The system will utilize email notifications for various purposes:
    - Alerting reviewers about new submissions.
    - Informing students about the status of their submissions (e.g., review completion, rejection).
    - Daily batch job reminders for stagnant reviews and revisions requiring attention.
- **Web Communication Protocols:** Standard web communication protocols like HTTP/HTTPS will be used for all communication between the application and web browsers.
- **Security:** Secure communication protocols (HTTPS) will be implemented to ensure the confidentiality and integrity of data transmission, particularly for sensitive information like user credentials and code submissions.

4. Other Nonfunctional Requirements

4.1 Performance Requirements

- **Response Time:**
    - **Rationale:** Users expect a responsive and efficient system to avoid frustration and delays.
    - **Requirements:**
        - The application should load web pages within 3 seconds under normal conditions with a moderate user load.
        - Submission and review actions should provide feedback to users within 2 seconds of completion.
- **Scalability:**
    - **Rationale:** The application should accommodate a growing user base and potential increases in assignment submissions without significant performance degradation.
    - **Requirements:**
        - The system should be able to handle a minimum of X concurrent users (define a specific number based on expected usage) without exceeding a response time threshold of 5 seconds.
        - The database architecture should be designed to scale efficiently by adding hardware resources if necessary.
        -

4.2 Safety Requirements (Not applicable for this project)

There are no inherent safety concerns associated with this web application. No physical harm or damage can occur through its use.

4.3 Security Requirements

- **User Authentication:**

- **Rationale:** Protecting user credentials and sensitive data is crucial for maintaining user trust and preventing unauthorized access.
- **Requirements:**
  - Implement secure login mechanisms using strong password hashing (e.g., bcrypt) and consider two-factor authentication for added security.
  - Enforce password complexity requirements (minimum length, character types).
  - User sessions should automatically time out after a period of inactivity.

- **Data Encryption:**
  - **Rationale:** Sensitive data, such as user credentials and potentially student code submissions (depending on the privacy of the code), requires protection from unauthorized access.
  - **Requirements:**
    - Encrypt user credentials (passwords) at rest and in transit using industry-standard algorithms (e.g., AES-256).
    - Consider encrypting code submissions at rest, especially if they contain sensitive information.
- **Authorization:**
  - **Rationale:** Restricting access based on user roles ensures that users can only perform actions they are authorized for.
  - **Requirements:**
    - Implement user roles (student, reviewer, admin) with distinct access levels to functionalities and data.
    - Students should only be able to access their own assignments and review feedback.
    - Reviewers should only be able to access assignments assigned to them for review.
    - Admins should have access to manage user accounts and potentially view all assignments (without accessing code details unless necessary).

4.4 Software Quality Attributes

- **Usability:**

- **Rationale:** A user-friendly interface is essential for user adoption and efficient interaction with the application.
- **Requirements:**
  - The application interface should be intuitive and easy to navigate for all user roles (students, reviewers, admins).
  - Provide clear instructions and tooltips where necessary.
  - Maintain a consistent design style throughout the application.

- **Reliability:**
  - **Rationale:** Users rely on the application to be available and function correctly to manage their assignments and reviews.
  - **Requirements:**
    - The application should strive for an uptime of 99% during regular operating hours.
    - Implement error handling mechanisms to gracefully handle unexpected situations and provide informative messages to users.
    - Conduct regular testing to identify and fix bugs before they impact users.
- **Maintainability:**
  - **Rationale:** The application code should be easy to understand, modify, and extend for future updates and feature additions.
  - **Requirements:**
    - Document code clearly using comments and naming conventions.
    - Organize code into modular components with well-defined functions.
    - Implement unit and integration tests to ensure code quality and functionality.

5. Other Requirements

5.1 Internationalization (Optional)

- If the application is intended for a global audience, consider incorporating features to support different languages.
- This may involve designing the interface to accommodate different character sets and layouts, and potentially offering language selection options for users.

5.2 Database Requirements

- The application will require a database to store user information (students, reviewers, admins), assignment details, submission data (including potentially file uploads containing code), and review records.
- The specific database technology (e.g., MySQL, PostgreSQL) will be chosen based on project needs, scalability considerations, and familiarity of the development team.
- Define any specific data integrity requirements or constraints to ensure data consistency and accuracy within the database.

5.3 Logging and Monitoring

- Implement a logging system to record user activity, system events (errors, warnings), and review actions.
- This information can be valuable for debugging issues, tracking user behavior, and auditing purposes.
- The application should also be monitored for performance metrics (response times, resource utilization) to identify potential bottlenecks and ensure optimal system health.
-

Appendix A: Glossary

- **API (Application Programming Interface):** A set of protocols and tools for building software applications that enables interaction with external systems.
- **Authentication:** The process of verifying a user's identity.

- **Authorization:** The process of granting access to specific resources or functionalities based on user roles or permissions.
- **Branch (Git):** A named pointer to a specific version of a codebase within a Git repository.
- **Encryption:** The process of transforming data into a scrambled format that can only be decrypted with a specific key.
- **Hashing:** A cryptographic function that transforms data into a fixed-size string (hash value) that can be used for data integrity verification.
- **Repository (Git):** A central location for storing and tracking changes to computer code and data.
- **Response Time:** The time it takes for a system to respond to a user action or request.
- **Scalability:** The ability of a system to handle increasing demands or workloads without significant performance degradation.
- **Secure Sockets Layer (SSL)/Transport Layer Security (TLS):** Security protocols that encrypt communication between a web browser and a server to protect data transmission.
- **Two-factor Authentication (2FA):** An additional layer of security that requires two verification methods (e.g., password and a code from a mobile device) to access an account.
- **User Interface (UI):** The graphical interface through which a user interacts with a software application.
-

Appendix B: Analysis Models

- Consider including Data Flow Diagrams (DFDs) to illustrate the flow of data within the application, including user interactions, system processes, and data storage.
- Entity-Relationship Diagrams (ERDs) can be used to depict the relationships between different data entities (e.g., users, assignments, submissions) and their attributes within the database.
-

Appendix C: Issues List

- Maintain a separate document or section within the SRS to track any identified issues, bugs, or areas requiring further clarification throughout the development process.
- This list can include TBDs (to be defined), pending decisions, missing information, or potential conflicts between requirements.

Other rough explanation

ASSIGNMENT SUBMISSION & REVIEW APP

Requirements

A student should be able to login to our application and see a dashboard that outlines the following info:
- What assignments are due, and when
- (perhaps) a page to review past assignment submissions
- Status of an assignment that's currently under review and if it needs tweaking
- A way to submit (or re-submit) an assignment for review

When submitting an assignment, the student will be asked to provide the following info:
- GitHub URL to public Repo
- Which branch is the correct one

Once submitted, the status updates from "pending submission" -> "submitted"

Bonus Feature: leverage GitHub API to verify if a URL points to a public repo, and get a list of branches to populate the "branch" dropdown

Once a student has submitted an assignment, an email notification should be sent to all active code reviewers letting them know that a new code review is ready. The first code reviewer to claim the review will be able to start working on it (and now it's unavailable to others for review).

Once a code reviewer has claimed a submitted assignment, the status changes from "submitted" -> "in review"

Bonus Feature: if a code reviewer has claimed a review and hasn't completed it within 24 hours it is sent back to the previous status, and anyone can claim it.

A code reviewer can reject a code review if it doesn't satisfy the assignment's criteria.
When a code reviewer rejects an assignment, the status will change from "submitted" -> "needs update"
When this happens, a notification will be sent to the student telling them they need to do more work on that assignment. Once they've done their work, they can revisit that assignment for re-submission (perhaps prompt them to make sure they've pushed their changes).

When a student re-submits an assignment, they will be given a chance to change their github url + branch, then it will move from "needs updating" -> "in review" and get assigned back to the same code reviewer. A notification will be sent to the code reviewer.

When a code reviewer has successfully completed their review, they will be able to submit their finished review (by adding a URL to a video recording) and then a notification will be sent back to the student. The status will change from "in review" -> "completed"

Also, for each cohort of students, we will need to have an API that allows for an Admin role to create user accounts

Daily batch jobs

Notifies code reviewers for reviews that are sitting stagnant and waiting for review
Notifies students if their rejected assignments are still needing tweaking

Appendix

Statuses: pending submission, submitted, in review, needs update, completed