Software Requirements Specification

Version 1.0
NOV 7, 2025

SyncBridge

Group 6
CHEN Tianyu 1155210962
GUO Mingkang 1155211019
HUA Zifan1155211089
Li Molin 1155191354
Qi Zihan 1155191644

Submitted in partial fulfillment
Of the requirements of
CSCI 3100 Software Engineering

Table of Contents

1.0 Introduction

1.1 Purpose

The purpose of this document is to present a detailed Software **Requirements** Specification (SRS) for the SyncBridge system. This document describes the system's intended functionality, performance, and constraints, as well as its interaction with users and external systems.
SyncBridge is designed to provide an efficient and structured platform for communication and requirement clarification between **clients** and **developers** during software development projects.

This SRS serves as a reference for all stakeholders, including project sponsors, clients, and developers. It ensures that all participants share a consistent understanding of the system objectives, functionalities, and design limitations. Developers will use this document as a basis for implementation, testing, and future maintenance, while stakeholders can use it to validate that the system aligns with their expectations.

1.2 Scope of Project

SyncBridge is a web-based requirement management and communication platform designed to streamline collaboration between clients and developers. The system facilitates the entire lifecycle of requirement clarification—from initial submission to discussion, approval, and completion—within a single, integrated environment.

Key features of SyncBridge include:

Requirement Submission: Clients can submit new requirements via standardized **forms** with structured fields such as title, description, tags, estimated delivery time, and budget.

Status Management: Developers can update the progress of each requirement (e.g., Pending, In Progress, Completed), with all changes automatically logged and visible to both parties.

Communication System: Clients and developers can exchange messages in threaded discussion areas tied to each requirement.

Email Notification: The system automatically sends email updates for key events such as new submissions, status updates, and urgent communication.

Audit and Traceability: Every modification, comment, and status change is recorded with timestamps and user identifiers to ensure transparency and accountability.

By providing these functions, SyncBridge improves communication efficiency, reduces misunderstandings, and enhances the traceability of software requirements throughout the development process.

1.3 Document Conventions

This document follows the conventions specified below to ensure clarity, consistency, and traceability throughout the SyncBridge Software Requirements Specification (SRS).

- Font and Formatting:
  All body text is written in Times New Roman, 12-point font size. Section titles are bolded, with hierarchical numbering following the IEEE SRS standard (e.g., 1.1, 1.2, 1.3).

- Terminology:
  The first occurrence of each technical term is written in bold. Formal definitions of all such terms

are provided in Appendix A: Glossary.

- Priority Levels:
  Each requirement is assigned a priority to indicate its implementation urgency:
    - High — Essential for core system functionality or compliance
    - Medium — Important but not critical to initial deployment
    - Low — Optional or enhancement feature for future releases

1.4 References

IEEE Computer Society. IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications.

University Course Document: CSCI 3100 Software Engineering Requirements Specification Template, Fall 2025.

SyncBridge Project Team. *Internal Oral Discussions and Design Meetings (Unrecorded)*, 2025.

1.5 Overview of Document

The remainder of this document is organized as follows:

Section 2 – Overall Description:
Provides a high-level overview of SyncBridge's environment, major components, user characteristics, and non-functional requirements.
Section 3 – Requirements Specification:
Details the external interfaces, functional requirements (e.g., login, submit form, status management, communication), and non-functional requirements such as performance, security, and scalability.
Appendices and Figures:
Include diagrams, logical data structures, and supplementary reference materials to support system understanding.

This structure ensures that both technical and non-technical readers can navigate the document efficiently and understand how the system fulfills its intended purpose.

2.0. Overall Description

2.1 System Environment

The Requirement Clarification System has two active actors and one cooperating system.
The client submits requirements to the Form System . The client sends **messages** to the Communication System. The Email System sends emails to the client. The Developer accesses the communication system directly to review, clarify, and manage requirements with clients. All the workflow will be submitted by the developer to update the status system.The email system sends the email to both client and developer to confirm the update.

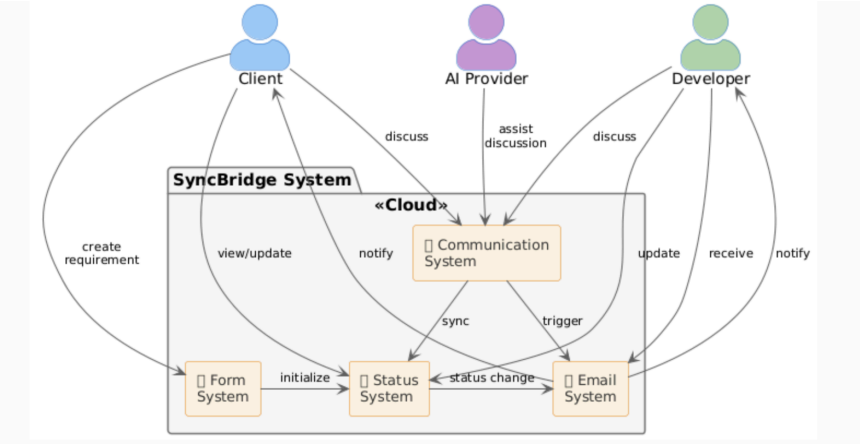


Figure 1 - System Environment

2.2 Functional Requirements Specification

This section outlines the use cases for each of the participant separately. The user and the developer have only one use case.

2.2.1 Client Use Case

**Use case: Submit Form
Diagram:

Brief Description

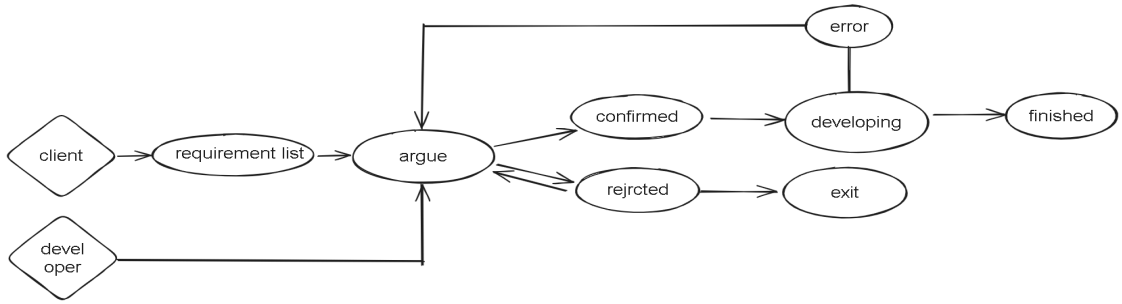
Users can submit forms including 3 parts: title,message and tag.

Initial Step-By-Step Description
Before this use case can be initiated, the Reader has already signed in an account.
1.The user logs into their account and navigates to the Submit page.
2.The user edits the following 3 parts:Title ,Message ,Functions ,Performance requirements ,Budget ,Expected time.
Users can customize the functions they need.
For performance requirements, they can be categorized into Lightweight, Commercial, and Enterprise levels based on the budget.
Each level has a different total percentage allocation, which users can distribute across various performance areas, such as security, high concurrency, and more.
3.The user reviews the filled form to ensure all required fields are completed and the information is accurate.
4.The user clicks the "Submit" button to submit.
5.The system saves the submitted data and displays a confirmation message.
6.The system assigns a unique ID to the submission for tracking purposes and revision.
Xref: Section 3.2.1, Search Article

2.2.2 Developer Use Case

In case of multiple authors, this term refers to the *principal author*, with whom all communication is made.

Use case: Update status
Diagram:



Use case: Update Status

Brief Description
The client and the developer collaborate to manage the full lifecycle status of requirements: The system supports both parties to update the requirement status on schedule (following the established process and recording changes). If any problems arise during development, the requirements can enter the "Error" state and then revert to the "Confirmed" state for further discussion. When either party is unable to meet the demand, they may also refuse it.To facilitate customers' clear understanding of the development progress, each function is equipped with a visual progress bar (developers can define the progress based on work such as code writing), and the overall work progress is visualized in percentage form. In addition, developers can set weights for each function to help clarify the project development structure plan and improve development efficiency.

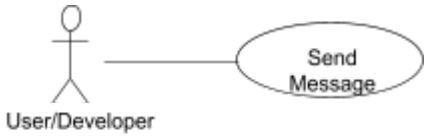
Initial Step-By-Step Description
Before this use case can be initiated, the client has created a requirement in the form of a online list, and both the client and the developer have access to the corresponding requirement record.

1. When the client submits a new requirement, the system automatically sets its status to Pending Confirmation.
2. The client and the developer discuss the requirement details through the system.
3. Based on the discussion, the requirement status is updated to:
    - Confirmed if both parties agree to proceed, or
    - Rejected if the requirement cannot be accepted.
4. If the requirement is rejected, the parties may either renegotiate (returning to Pending Confirmation or Confirmed) or end the process and handle any contractual obligations.
5. Once confirmed, the developer starts implementation and updates the status to In Progress, and every step finished by the developer will update the progress and make it visible to the user.
6. During development, if any issue is detected by the client or developer, the requirement enters an Error state and must be reverted to Confirmed for re-evaluation and further discussion.
7. After successful completion, the developer updates the status to Completed.
8. The system records all status updates with timestamps and displays the current status to both the client and the developer, shown in percentage form.

Xref: Section 3.2.4, Status Management

32.2.3 Common Use Case

Use case: Send Message
Diagram:



Brief Description

User or Developer sends a message to a comment area associated with a requirement. The comment can include text or files. The system supports previewing certain file types and handles various encoding formats.

Preconditions:

- The User/Developer has already accessed the main page of the Software.
- The selected requirement exists in the system.

Initial Step-By-Step Description

1. The user selects the requirement or the specific function sub-comment area they want to comment on.
2. The system displays the detailed view of the selected requirement, including all existing comments.
3. The user enters a message in the comment input area.
4. If the user wants to attach a file:
   - The system allows file uploads of **any type**, up to **1 GB**.
   - For larger files, the system prompts the user to provide a **GitHub link**.
   - Supported file types for preview: **images** and **standard encoded text files** (e.g., .txt, .c).
   - The system applies **plugins** as needed to correctly display files in different programming languages.
5. The user submits the message (and optionally attached files).
6. The system validates the message and attachments:
   - Ensures the content is not empty.
   - Confirms the file size and type constraints.
   - Verifies encoding for previewable files.
7. The system updates the comment section in **real-time**, displaying the new comment.
8. The system updates the comment section in **real-time**, displaying the new comment.
9. If the file is previewable, the system renders a **preview** (image or text/code with proper syntax highlighting).
10. The system confirms to the user that the comment has been successfully posted.

Xref: Section 3.2.6, Send Message

Use case: Login/Register

Diagram

Brief Description

The client and developer register their own account if they don't have one before or login their account.

Initial Step-By-Step Description

Before this use case can be initiated, the client and developer have already accessed the Online Website.# Todo

1.The client and developer sign up for a new account by email or phone number if they don't have one.
2.If client and developer have an account, they can log in with email verification or via mobile text message verification.

Xref: Section 3.2.1, Register and Login

Use case: Send email

Diagram

Brief Description

User/Developer sends an email notification related to form submission, status update, or message events.

Initial Step-By-Step Description

1.The user submits a form or triggers an event (e.g., update status, modify format, or send message).
2.The system identifies the event type and determines the recipient (e.g., developer or user).
3.The user can optionally mark the **email as urgent**.
4.The system generates the corresponding email content based on the event details.

- If Update Status, the system sends an email to notify the other party of the new status.
- If Format Modification, and the user has not replied within 5 minutes, the system automatically sends an email reminder.

5.The system sends the email to the recipient's registered email address.
6.The system confirms to the sender that the email has been successfully sent.

Xref: Section 3.2.4, Send email

2.3 User Characteristics

2.3.1 Client

Background: Typically non-technical individuals with limited understanding of development technology implementation details.They focus more on the business value and user experience of the requirements rather than implementation details.

Skills:Able to describe business needs and pain points, but may lack the ability to express them in structured manner.

2.3.2 Developer

Background:
Possess a technical background and expertise in software development, system design, and code implementation.Highly sensitive to the feasibility of requirements and technical limitations.

Skills:
Ability to quickly analyze requirements descriptions and assess their feasibility and implementation options.Adapt at asking questions to clarify ambiguities in requirements, thereby reducing misunderstandings.

2.4 Non-Functional Requirements

The system shall deliver high performance and reliability, ensuring that all submissions and queries are responded to within three seconds. It must support at least 100 concurrent users while maintaining an annual uptime of 99.5%.

The interface shall be intuitive and bilingual, supporting English and Chinese, and enabling non-technical users to complete key operations in no more than three steps.

Security shall be maintained using HTTPS encryption, username-password authentication, and role-based access control (RBAC) to ensure data protection and proper access permissions.

The system shall adopt a modular, maintainable, and scalable architecture, with at least 70% unit test coverage to ensure code quality. It must support deployment on Windows, macOS, and Linux, while remaining compatible with all major browsers.

All requirement changes, comments, and status updates shall be automatically logged with timestamps and user identifiers, ensuring full traceability. Additionally, the system shall provide exportable audit records for accountability and transparency.

2.5 Operating Environment

The system will support the latest two stable releases of the following desktop web browsers: Chrome, Firefox, Edge, and Safari (version 16 or higher).

For server deployments, the system will run on 64-bit Linux or Windows Server environments, with a minimum hardware configuration of 4 vCPUs and 8 GB of RAM. Persistent data storage will be handled by a relational SQL database.

All server-side timestamps will be stored in UTC, ensuring consistency across deployments. Client-side applications will display time in the user’s localized timezone, providing a seamless and region-specific experience.

2.6 User Documentation

The user documentation includes seven core modules: Product/Function Overview (core values, positioning, key functions), Quick Start Guide (installation/login steps, core operation shortcuts, interface button descriptions), Detailed Function description (operation paths, scenarios, parameters based on modules) Simple screenshots, FAQ (common issues such as login failure and actionable solutions), precautions/risk warnings (data backup, permission restrictions, prohibited operations, compatibility), support/contact information (customer service channels, response time).

2.7 Assumptions and Dependencies

The smooth implementation and operation of the system rely on external conditions (reliable databases, SMTP servers, file storage, and necessary third-party apis) and assumptions (stable infrastructure, sufficient operational support, and users' basic system literacy).

3.0. Requirements Specification

3.1 External Interface Requirements

3.1.1 User Interfaces

The system provides a unified web-based Graphical User Interface (GUI) optimized for clarity and responsiveness.
It includes a top navigation bar (Home, Forms, Messages, Settings) visible on all pages, with consistent colors and typography.
Core views: Dashboard for recent activities, Form View for creation and tracking, Message Panel for real-time discussion, and Settings for user preferences.
Error prompts appear as concise toast notifications; keyboard access is supported for main operations.
The design complies with WCAG 2.1 accessibility guidelines and supports both light and dark themes.

3.1.2 Hardware Interfaces

The software runs on standard consumer devices without proprietary hardware.
Supported platforms include desktop systems running Windows 10+, macOS 12+, and Linux 20.04+.
Interaction with devices uses OS-level APIs for storage, networking, and input.
Typical peripherals are keyboards, pointing devices, and optional external storage.
No specialized sensors or custom drivers are required.

3.1.3 Software Interfaces

The system interacts with standard components such as a relational database, SMTP mail service, and optional SSO provider.
Data exchange uses JSON-based REST APIs over HTTPS; all calls are authenticated and logged.
Internal modules share session data, file metadata, and message objects through a common service layer.
All operating-system dependencies are abstracted through platform APIs, ensuring portability across major environments.

3.1.4 Communications Interfaces

All communications occur over secure protocols: HTTP/1.1 or HTTP/2 over TLS 1.2+ for general traffic and WSS for real-time messaging.
Email notifications use SMTP (STARTTLS); failed deliveries retry up to three times.
Messages are UTF-8 encoded JSON objects with standard fields for status and error codes.
End-to-end encryption protects data in transit, and authentication tokens refresh periodically to maintain session integrity.

3.2 Functional Requirements

The Logical Structure of the Data is contained in Section 3.3.1.

3.2.1 Register and login

Use Case Name	Register and Login
XRef	Section 2.2.3, Common Use Case
Trigger	The user accesses the software and selects “Register” or “Login.”
Precondition	The software is launched, and the login/registration interface is displayed.
Basic Path	1. The user chooses whether to register or log in.

	2. If the user selects Register, the system displays a registration form requesting username, password, and email. 3. The user fills in all required fields and submits the form. 4. The system verifies that the username and email are unique and that all fields meet validation criteria (e.g., password strength). 5. The system stores the new user information in the database and confirms successful registration. 6. The user is redirected to the login page. 7. The user enters the registered username and password. 8. Upon successful authentication, the user is directed to the home page or dashboard.
Alternative Paths	1. In step 4, if the username or email already exists, or if the password does not meet requirements, the system prompts the user to correct the information and resubmit. 2. In step 7, if the user clicks “Forgot Password,” the system displays a password recovery form. The user enters their registered email, the system sends a reset link, and the user resets the password. 3. In step 8, if the username or password is incorrect, the system displays an error message and allows the user to retry or reset the password.
Postcondition	The user is successfully authenticated and gains access to the main system functions.
Exception Paths	The user may close the application or abandon the registration/login process at any time.
Other	All passwords are securely stored in hashed form in the database. The system supports session timeout and logout functions for security.

3.2.2 Submit form

Use Case Name	Communicate
XRef	Section 2.2.1, Submit Form；Section 2.2.3,Send Message
Trigger	The user navigates to the Submit page or enters a message into the comment input area.
Precondition	The user has logged in. The user is on the Submit page.
Basic Path	1.The user logs into their account and navigates to the Submit page. 2.The user edits the following 3 parts:Title ,Message ,Functions ,Performance requirements ,Budget ,Expected time. Users can customize the functions they need. For performance requirements, they can be categorized into Lightweight, Commercial, and Enterprise levels based on the budget. Each level has a different total percentage allocation, which users can distribute across various performance areas, such as security, high concurrency, and more. 3.The user reviews the filled form to ensure all required fields are completed and the information is accurate. 4.The user submits a form. 5.The system saves the submitted data and displays a confirmation message. 6.The user selects the requirement for which they want to add a comment. 7.The system displays the detailed view of the selected requirement, including existing comments. 8.The user enters a message into the comment input area. 9.The user submits the comment. 10.The system validates the message, stores it in the database, and updates the comment section in real-time.

Alternative Paths	If in step 5,the data is saved.The system assigns a unique ID to the submission for tracking purposes. If in step 10,the data is saved.The system confirms to the user that the comment has been successfully posted.
Postcondition	The message is sent.
Exception Paths	The attempt may be abandoned at any time.
Other	Adding comments and submitting forms are in two different systems. Submitting forms needs to be done before commenting.

3.2.3 Status Management

Use Case Name	Status Management
XRef	Section 2.2.1 Client Use Case: Submit Form Section 2.2.2 Developer Use Case: Update Status Section 2.2.3 Common Use Cases: Send Message Section 3.2.4 Send Email
Trigger	Developer decides to update the status of a requirement, or an issue is detected during development.
Precondition	1. A requirement has been created through the Form System and is accessible to both the client and the developer. 2. The current status of the requirement is valid for transition (e.g., Pending Confirmation, Confirmed, etc.).
Basic Path	1. Developer selects a requirement from the system. 2. Developer updates the status of the requirement (e.g., Confirmed → In Progress). 3. System validates the status transition according to the defined workflow. 4. System records the status update with the following details: Timestamp User identifier Previous status New status Optional comment 5. System displays the updated status to both the client and the developer. 6. System sends an email notification to both parties about the status update.
Alternative Paths	A1: Status requires client confirmation Developer proposes a status change (e.g., Pending Confirmation → Confirmed). System sends a confirmation request to the client through the Communication System. Client reviews the proposed change and either confirms or rejects it. If confirmed, proceed to step 3 of the Basic Path. If rejected, the status remains unchanged, and the system notifies the developer. A2: Status change to "Error" Developer or client detects an issue during development. Requirement status changes to Error. System reverts the status to Confirmed for re-evaluation and further discussion through the Communication System.

Postcondition	The requirement status is successfully updated in the system. Both the client and the developer can view the updated status and its history. Email notifications are sent to both parties.
Exception Paths	E1: Invalid status transition Developer attempts an invalid status transition (e.g., Completed → In Progress). System displays an error message and prevents the update. E2: Client fails to confirm status change System sends a confirmation request to the client through the Communication System. Client does not respond within the defined timeframe. System cancels the proposed status change and notifies the developer.
Other	**Audit Logs**: The system shall maintain a complete audit log of all status updates, including timestamps, user identifiers, and comments. Notifications: Email notifications shall include the updated status, timestamp, and optional comments.

3.2.4 Send email

Use Case Name	Send email
XRef	Section 2.2.3, Common Use Case
Trigger	The system triggers an email notification event (e.g., user registration success, password reset, or system update).
Precondition	The user has a valid email address stored in the system database, and the mail server configuration is active.
Basic Path	1. A triggering event occurs (e.g., successful registration, password reset request, or admin announcement). 2. The system retrieves the user's email address from the database. 3. The system connects to the configured SMTP mail server. 4. The system sends the email to the user's email address. 5. The system logs the sending result (success or failure). 6. The user receives the email notification in their inbox.
Alternative Paths	1. In step 2, if the user's email address is invalid or missing, the system skips sending and logs an error message. 2. In step 3, if the SMTP server is unreachable, the system queues the email for retry and notifies the administrator.
Postcondition	The email notification is successfully sent or queued for retry.
Exception Paths	The system may abort the process if configuration or connection errors persist.
Other	All outgoing emails are sent via a secure SMTP connection (TLS/SSL). The system supports customizable templates for different notification types (e.g., registration, warning, updates).

3.2.5 Send message

Use Case Name	Send message
XRef	Section 2.2.3, Common Use Case

Trigger	The User/Developer chooses to add a comment to a requirement or a specific function sub-comment area.
Precondition	-The user/developer has logged in and accessed the system main page -A requirement item has been selected and its detail page is open
Basic Path	1. The user selects the requirement or the specific function sub-comment area they want to comment on. 2. The system displays the detailed view of the selected requirement, including all existing comments. 3. The user enters a message in the comment input area. 4. If the user wants to attach a file: ● The system allows file uploads of **any type**, up to **1 GB**. ● For larger files, the system prompts the user to provide a **GitHub link**. ● Supported file types for preview: **images** and **standard encoded text files** (e.g., .txt, .c). ● The system applies **plugins** as needed to correctly display files in different programming languages. 5. The user submits the message (and optionally attached files). 6. The system validates the message and attachments: ● Ensures the content is not empty. ● Confirms the file size and type constraints. ● Verifies encoding for previewable files. 7. The system updates the comment section in **real-time**, displaying the new comment. 8. The system updates the comment section in **real-time**, displaying the new comment. 9. If the file is previewable, the system renders a **preview** (image or text/code with proper syntax highlighting). 10. The system confirms to the user that the comment has been successfully posted.
Alternate Flows:	**- AF1:** File exceeds 1 GB ● The system prompts the user to upload via GitHub instead. **- AF2:** Unsupported file type for preview ● The system stores the file but does not attempt preview. **- AF3:** Encoding issues detected ● The system either applies a plugin to render correctly or prompts the user to re-encode.
Postcondition	● The comment and any attachments are saved in the database. ● The comment section displays the newly posted comment. ● Files that can be previewed are immediately visible in the interface.
Other	- All comment actions are logged with timestamp and user ID - The system supports real-time refresh (Ajax/WebSocket) for better user experience

3.3 Detailed Non-Functional Requirements

3.3.1 Performance

The system shall be capable of responding to any client or developer query within three seconds under normal operating conditions. Each web page and function module (such as requirement submission and

discussion) shall load within two seconds on a stable network. The system is designed to handle at least 100 concurrent active users without noticeable degradation in response time or service quality.

3.3.2 Reliability & Availability

The system shall maintain an annual uptime of no less than 99.5%. In the event of unexpected server interruption, the system shall automatically restore data from the most recent backup within five minutes. Error events shall be logged automatically and notifications shall be sent to the system administrator for timely resolution.

3.3.3 Usability

The user interface shall be intuitive and consistent, allowing both technical and non-technical users to perform major operations such as submission, commenting, and status modification within three steps. The platform shall provide bilingual interfaces (English and Chinese) and maintain a clean, minimal design to enhance accessibility and reduce learning time.

3.3.4 Security

All users are authenticated through a secure username and password mechanism before accessing the system. All transmitted data—including requirement details, comments, and user credentials—shall be encrypted using HTTPS. Access control policies shall ensure that regular clients can view only their own requirements, developers can view assigned requirements, and administrators possess full system privileges.

3.3.5 Maintainability

The system adopts a modular, service-oriented architecture that enables independent maintenance and upgrading of the front-end and back-end components. All APIs shall be documented and automatically published through Swagger. Unit testing shall ensure at least 70% code coverage, supporting continuous integration and deployment.

3.3.6 Scalability

The system is designed to scale horizontally and vertically. It shall allow the easy addition of new user roles (e.g., Tester) and accommodate a tenfold increase in the number of requirements without major refactoring. The database schema and API structure will support future extensions with minimal redesign effort.

3.3.7 Portability

The platform shall be deployable across Windows, macOS, and Linux servers. The web client shall remain fully compatible with major browsers including Chrome, Edge, and Safari. The system’s architecture ensures minimal configuration differences across operating environments.

3.3.8 Auditability & Traceability

All requirement submissions, modifications, comments, and status updates shall be automatically recorded with timestamps and user identifiers. The system shall support exporting a complete change history in CSV or PDF formats for audit or review purposes. These records ensure accountability and transparency throughout the development cycle.

Appendix

Appendix A: Glossary

Term	Definition
Requirement	A user-submitted description of a needed feature, constraint, or objective, stored in the system for discussion, approval, and tracking.
Requirement Submission	The process through which a client fills out and submits a structured form including title, description, functions, performance requirements, budget, and expected time.
Requirement ID	A unique identifier automatically assigned by the system to each submitted requirement for tracking and traceability.
Status Management	The system-controlled lifecycle through which requirements progress (e.g., Pending Confirmation → Confirmed → In Progress → Completed).
Requirement Status	The current lifecycle state of a requirement. Possible values include: Pending Confirmation, Confirmed, Rejected, In Progress, Error, Completed.
Error State	A requirement status indicating that issues occurred during development and that the requirement must return to Confirmed for re-evaluation.
Progress Bar	A visual representation of percentage-based completion of a requirement's implementation, defined by the developer.
Function Weight	Developer-defined weight assigned to each function within a requirement to reflect its importance or workload contribution.
Client	A user responsible for submitting requirements and communicating business needs. Typically non-technical.
Developer	A user responsible for analyzing, clarifying, updating, and implementing client requirements.
User	A generic term referring to either a Client or a Developer interacting with the system.

Comment Area / Discussion Area	A message thread associated with a specific requirement or sub-function where users exchange messages and files.
Message	A text or file-based communication posted by a user in the comment area of a requirement.
Message Preview	The system's ability to visually display supported file types (images, text files, source code) inside the browser.
File Upload	The process of attaching files (≤1GB) to a message; larger files require GitHub links.
Encoding Format	The character encoding used in text/code files for preview (e.g., UTF-8).
Email Notification	A system-generated email triggered by actions such as new submissions, status updates, or messages.
Urgent Email	A user-marked email notification indicating a high-priority event.
Format Modification Event	A change to a requirement's structured fields that triggers an automated reminder email if unresponded within 5 minutes.
Audit Log	A chronological record of actions (status updates, comments, modifications) including timestamps and user identifiers.
Traceability	The ability to trace every requirement-related action through audit logs.
RBAC (Role-Based Access Control)	A security mechanism ensuring users can only access actions appropriate to their role (Client/Developer).
HTTPS	A secure communication protocol used for encrypted data exchange between browser and server.
SMTP	Protocol used by the system to send email notifications.
REST API	The application interface through which components exchange JSON-formatted data over HTTP/HTTPS.
WSS (WebSocket Secure)	Protocol used for real-time communication (e.g., live comment updates).
Dashboard	The main user interface showing recent activity, requirement summaries, and notifications.
Form System	The subsystem responsible for requirement submission and storing structured requirement data.

Communication System	The subsystem that manages messaging, comments, file uploads, and discussion threads.
Status System	The subsystem responsible for tracking and updating requirement lifecycle states.
Email System	The subsystem that generates and delivers event-triggered email notifications.
Session	A user's authenticated interaction period with the system, maintained through security tokens.
Session Timeout	Automatic logout after a period of inactivity to protect security.
Authentication	Verification of user identity through username/password or SMS/email verification.
Authorization	Determining whether a user has permission to perform a specific action.
Performance Requirement Level	One of three categories—Lightweight, Commercial, Enterprise—used to classify performance needs based on budget.
High Concurrency	A system capability allowing many users to interact simultaneously.
Non-Functional Requirement (NFR)	Requirements describing system qualities such as performance, security, availability, and scalability.
Availability	Percentage of time the system remains operational (target: 99.5%).
Scalability	The ability of the system to increase performance and capacity as user load grows.
Maintainability	Ease with which the system can be updated, fixed, or enhanced (e.g., modular design, 70% unit test coverage).
Portability	Ability of the system to run on different operating systems (Windows, macOS, Linux).
Accessibility (WCAG 2.1)	Standards that ensure the UI is usable by users with disabilities.
Timestamps (UTC)	Standardized times used in audit logs and server operations.
External Dependency	Any required external service such as a relational database, SMTP server, or file storage used by the system.

Assumption	A condition expected to be true for the system to function (e.g., stable internet connection, functioning email server).

Appendix B: TBD List

TBD Item	Description	Responsible Party	Deadline	Notes
Design Documentation Scope	**Confirm the level of detail required for the design document, including whether UML diagrams are mandatory or optional.**	**Development Team**	**TBD**	**While UML diagrams are not strictly required, their inclusion may enhance understanding.**
Key Components Definition	**Identify and finalize the key components of the system to be included in the design and implementation documentation.**	**Development Team**	**TBD**	**Components must align with system requirements and stakeholder goals.**
Testing Strategy	**Decide whether to adopt test-driven development (TDD) or a traditional code-test cycle for the project.**	**Development Team**	**TBD**	**The chosen approach will impact the testing process and documentation.**
Source Control Standards	**Define the standards for tracking released versions in the source control system (e.g., Git tags or branches).**	**DevOps Team**	**TBD**	**Ensure visibility into which versions of source files correspond to each release.**