System Design Document

Version 1.0
DEC 12, 2025

SyncBridge

Group 6
CHEN Tianyu 1155210962
GUO Mingkang 1155211019
HUA Zifan 1155211089
Li Molin 1155191354
Qi Zihan 1155191644

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

1. Introduction

1.1 Purpose

This Software Design Document (SDD) describes the complete technical design of the SyncBridge system, a web-based platform that supports structured requirement submission, negotiation, discussion, and status management between clients and developers.

The document defines:

- **System architecture (frontend + backend)**
- **Data models and database schema**
- **API specifications**
- **Messaging framework and WebSocket design**
- **Subform negotiation mechanism**
- **File storage logic**
- **Email notification logic (urgent/normal)**
- **Constraints, flows, and component responsibilities**

1.2 Scope

The system includes these primary features:

1. **Requirement submission (mainform)**
2. **Subform negotiation for requirement modification**
3. **Functional and non-functional requirement items**
4. **Structured message blocks: general / function / nonfunction**
5. **Real-time messaging and file preview**
6. **Automated email reminders based on block urgency**
7. **Role restriction via license (client / developer)**

The platform consists of:

- **Frontend: React + Vite, Shadcn UI, TailwindCSS, React Router, Zustand**
- **Backend: FastAPI, SQLAlchemy, MySQL, WebSocket, Poetry, Alembic**
- **File storage: local filesystem**
- **Email system: Resend API**
- **Transport: REST API + WebSocket**

1.3 Definitions and Terminology

Term	Description
Mainform	**The primary requirement form submitted by a client.**
Subform	**A modified version of the form used during negotiation; only one active subform is allowed at a time.**
Function	**A functional requirement item under a form.**
NonFunction (NFR)	**A non-functional requirement item under a form.**
Block	**A message thread area (general / function / nonfunction). Controls email notifications.**

Message	**Communication posted inside a block (text/file).**
Status	**The lifecycle stage of a form or subform.**
License	**Determines whether a user is client or developer.**
Urgent block	**Sends email if no reply in 5 minutes.**
Normal block	**Sends email if no reply in 48 hours.**
React Query	**Data fetching library used by frontend.**
Zustand	**Frontend state management library.**
Resend	**Email delivery provider.**

1.4 References

- **FastAPI official documentation**
- **React 18 + Vite documentation**
- **Shadcn UI documentation**
- **SQLAlchemy documentation**
- **IEEE SDD guidelines**

1.5 Document Overview

This SDD contains the following major sections:

1. **Introduction: System purpose, scope, and terminology.**
2. **Overall System Architecture: High-level architecture and interaction between frontend/backed/database.**
3. **Backend Detailed Design**
4. **Frontend Detailed Design**
5. **Database Design**
6. **API Specifications**
7. **Messaging & Email System Design**
8. **Security Design**
9. **Deployment Considerations**
10. **Appendix — UML Diagrams (UML Plot 1/2/3/4)**

2. Overall System Architecture

2.1 High-Level Architecture Overview

SyncBridge adopts a modular, service-oriented, three-tier architecture consisting of:

(1) Frontend Web Client（React + Vite）

(2) Backend Server（FastAPI）

(3) Database（MySQL）

(4) Local File Storage

(5) Email Notification Service（Resend API）

(6) WebSocket Real-Time Messaging

2.2 System Interaction Overview

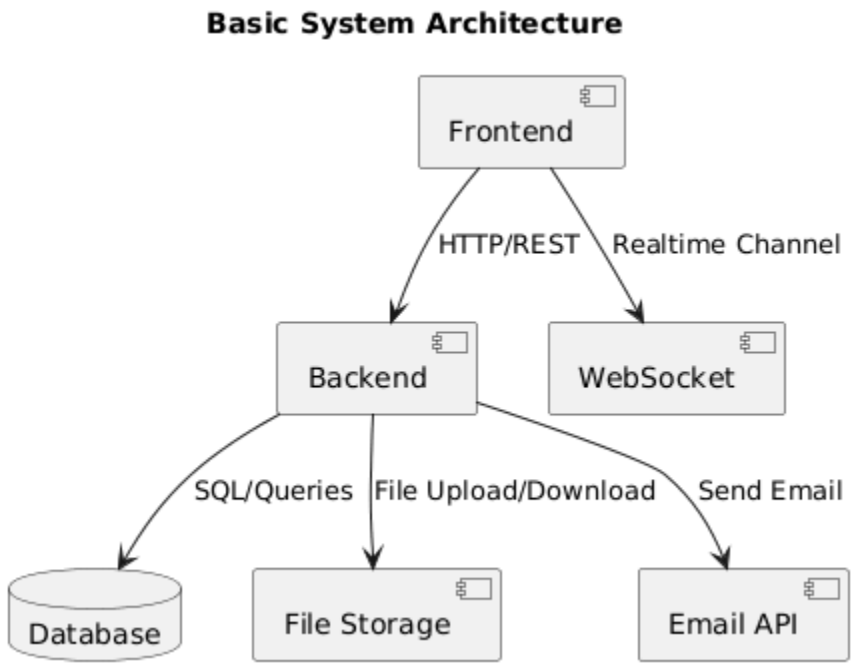
Sequence summary of a typical requirement lifecycle:

1. **Client submits a mainform (status = preview → available)**
2. **Developer sees available forms**

3. **Developer may**
    - **accept → status = processing**
    - **propose modification → create subform (rewrite stage)**
4. **Subform negotiation:**
    - **both comment in blocks**
    - **both must accept to merge**
    - **merged subform overwrites mainform**
5. **Messaging continues (WebSocket)**
6. **Email reminders sent based on block urgency**
7. **Form eventually ends (status = end)**

2.3 Component Diagram（UML Placeholder）

UML Plot 1 — System Component Diagram



2.4 Deployment Architecture

SyncBridge components can run in the following deployment structure:

- **Frontend: Static hosting (Vercel/Nginx/Netlify)**
- **Backend: FastAPI + Uvicorn + Gunicorn**
- **Database: MySQL 8.x**
- **Storage: Local file directory mounted to backend server**
- **Email service: Resend cloud API**
- **WebSocket: Backend integrated in FastAPI application**

Network flow:

Client Browser
→ REST API (HTTPS)
→ WebSocket (WSS)
→ Static Assets (CDN)
Backend Server
→ MySQL

→ Local File Storage
→ Resend API

2.5 Role-Based Data Flow Summary

Client

- Can view only his own forms
- Can create/modify/delete mainforms before processing
- Can reply in all blocks
- Can create or accept a subform only during rewrite
- Cannot see other clients' forms

Developer

- Can view:
  - available forms
  - processing forms assigned to self
  - end forms assigned to self
- Can accept available forms → processing
- Can create/edit/delete subforms
- Can modify functions / nonfunctions
- Can propose rewrite
- Messaging allowed in all blocks

2.6 Constraints

- Only one subform per mainform is allowed
- Form and subform have identical structure
- is_changed only meaningful on subform fields
- Block-level email triggers must be respected
- File size ≤ 10MB
- No multi-version history
- No admin role
- API surface must stay minimal

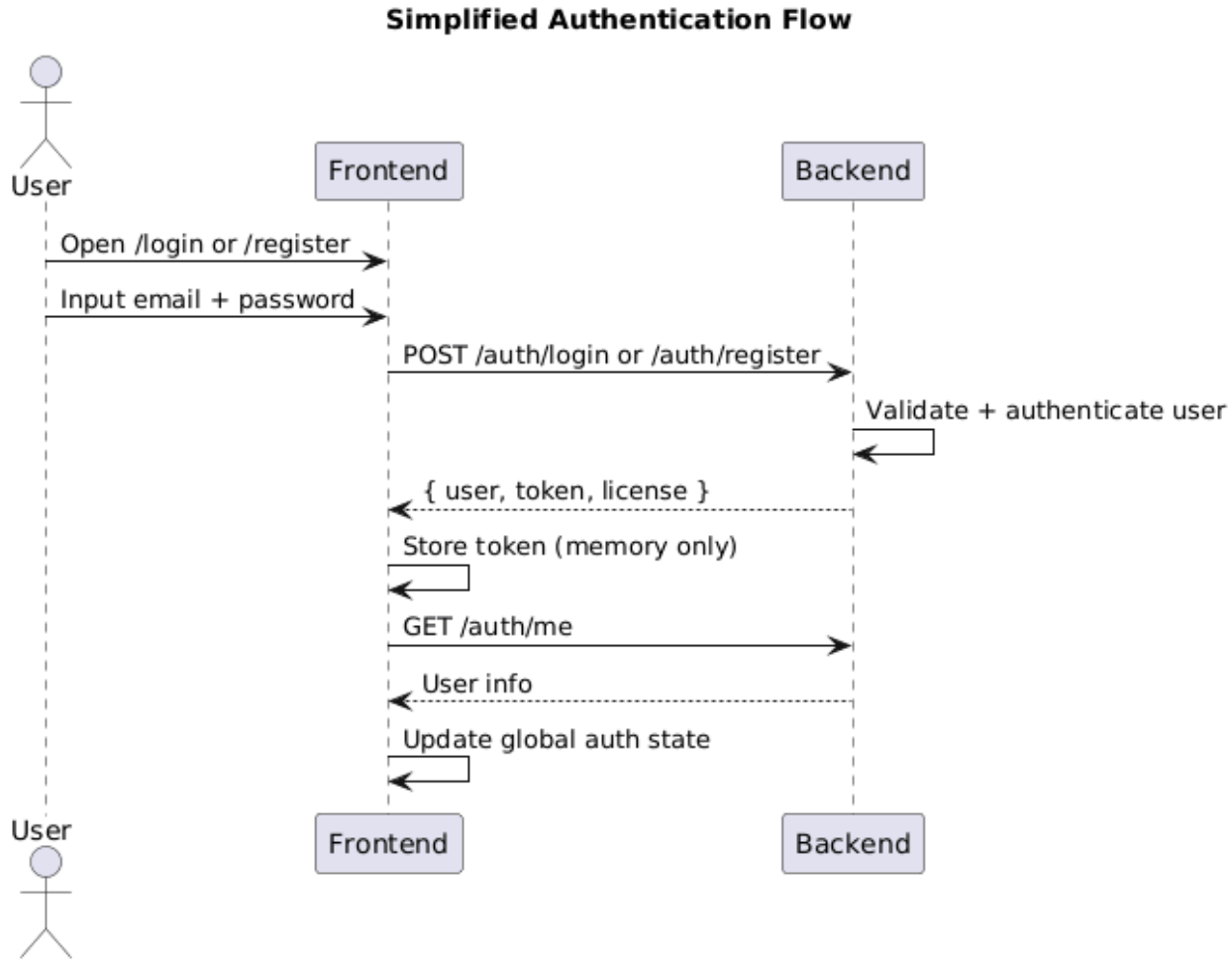
3. System-Wide Flow Diagrams

The diagrams cover:

1. User Authentication Flow
2. Form + Subform Lifecycle (Main Status Machine)
3. Function / NonFunction Interaction Flow
4. Message + WebSocket Flow
5. File Upload and Preview Flow
6. Email Notification Flow (Urgent / Normal Blocks)

3.1 User Authentication Flow

UML PLOT 1 — Authentication Flow Diagram



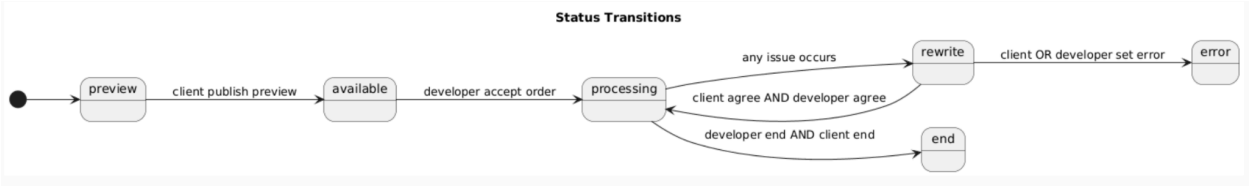
Flow Description

1. **User opens /login or /register.**
2. **User enters credentials:**
   - **email**
   - **password**
3. **Frontend sends request to:**
   - **POST /api/v1/auth/register**
   - **POST /api/v1/auth/login**
4. **Backend:**
   - **validates payload**
   - **hashes password (register)**
   - **checks email + password (login)**
   - **loads license.role (client or developer)**
   - **generates JWT token**
5. **Response:**

{status: "success", data: { user, token, license }}

6. **Frontend stores token in memory (not in localStorage).**
7. **Frontend requests user info via:**
   - **GET /api/v1/auth/me**
8. **React context updates global auth state.**

3.2 Form + Subform Lifecycle Flow
UML PLOT 2 — Mainform + Subform Status Machine
3.2.1 Mainform Status States



3.2.2 Detailed Lifecycle Explanation

1. Client Submits Form → preview
- Form created with status = **preview**
- After auto-validation, status becomes **available**

2. Developer Accepts Form
- status = **processing**

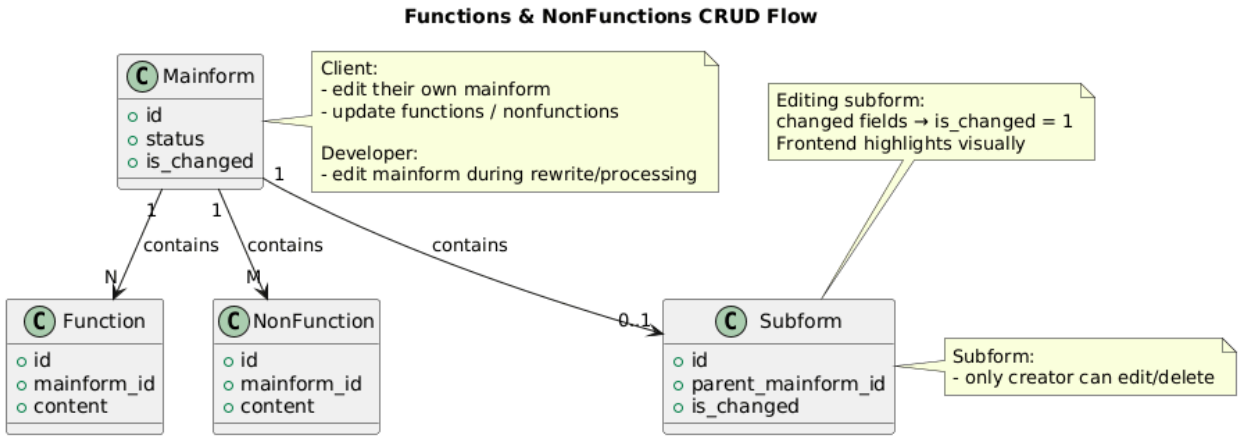
3. Developer Proposes Subform
- subform is created, linked via **mainform.subform_id**
- mainform.status = **rewrite**
- is_changed fields propagate

4. Discussion Phase
- both parties chat in block threads
- all fields displayed in "compare mode"

5. Merge
- subform overwrites mainform
- subform deleted
- status → processing

6. Reject Subform
- subform deleted
- mainform state resets to previous
- no version history saved (per your rules)

3.3 Function / NonFunction Flow
UML PLOT 3 — Functions & NonFunctions CRUD Flow



3.3.1 Relationship

Each form (or subform) contains:

1 mainform
0/1 subform
N functions
M nonfunctions

3.3.2 CRUD Rules

Actor	Allowed Actions
Client	edit their mainform, update functions/NFR
Developer	edit mainform during rewrite/processing
Subform	only creator may edit/delete

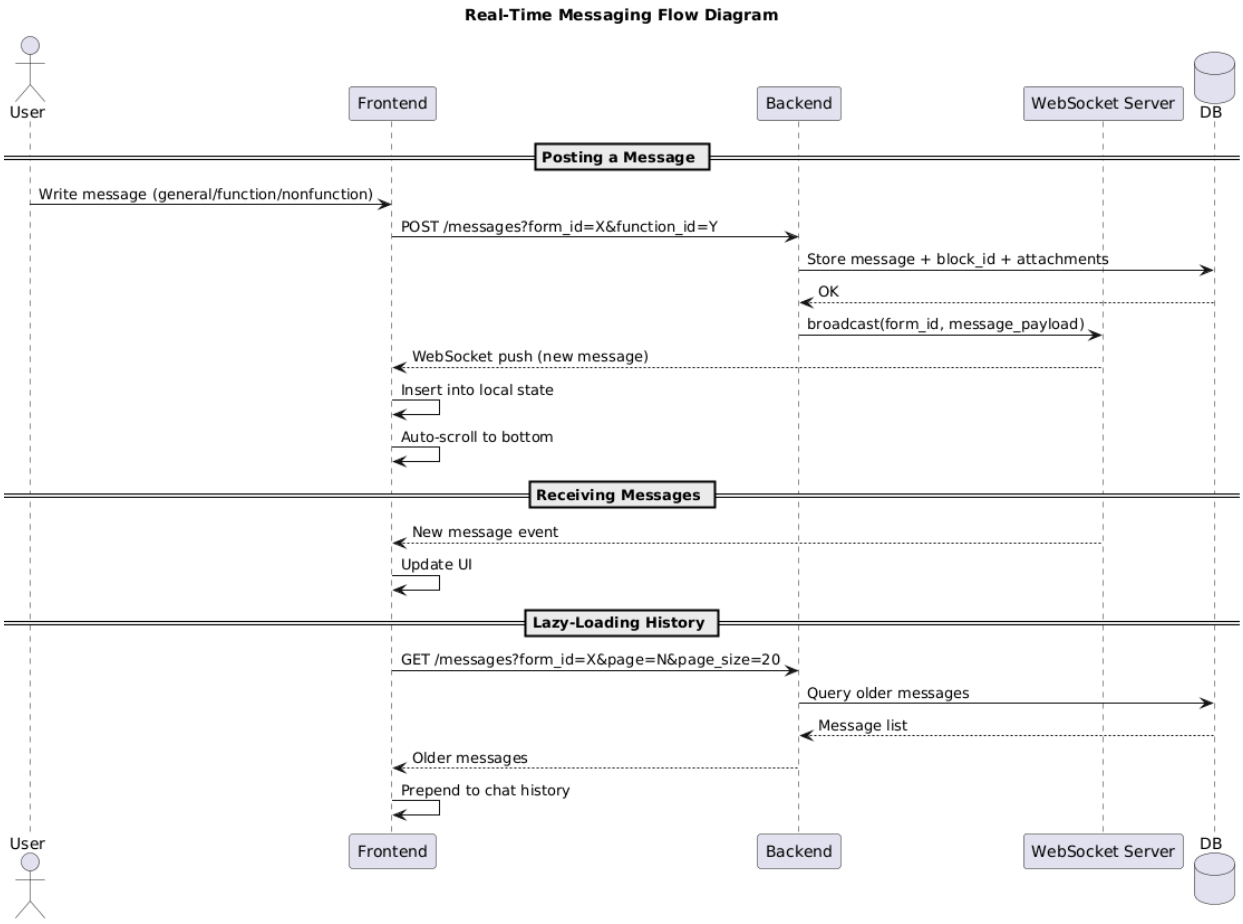
3.3.3 is_changed Propagation

When editing subform:

- **changed fields → set is_changed = 1**
- **frontend highlights changed fields visually**

3.4 Messaging + WebSocket Flow

UML PLOT 4 — Real-Time Messaging Flow Diagram



3.4.1 Posting a Message

1. **User writes message in a specific block:**
    - **general**
    - **function**
    - **nonfunction**
2. **Frontend sends:**

POST /api/v1/messages?form_id=X&function_id=Y

3. **Backend stores:**
    - **message**
    - **block_id**
    - **any attached files**
4. **Backend triggers WebSocket push:**

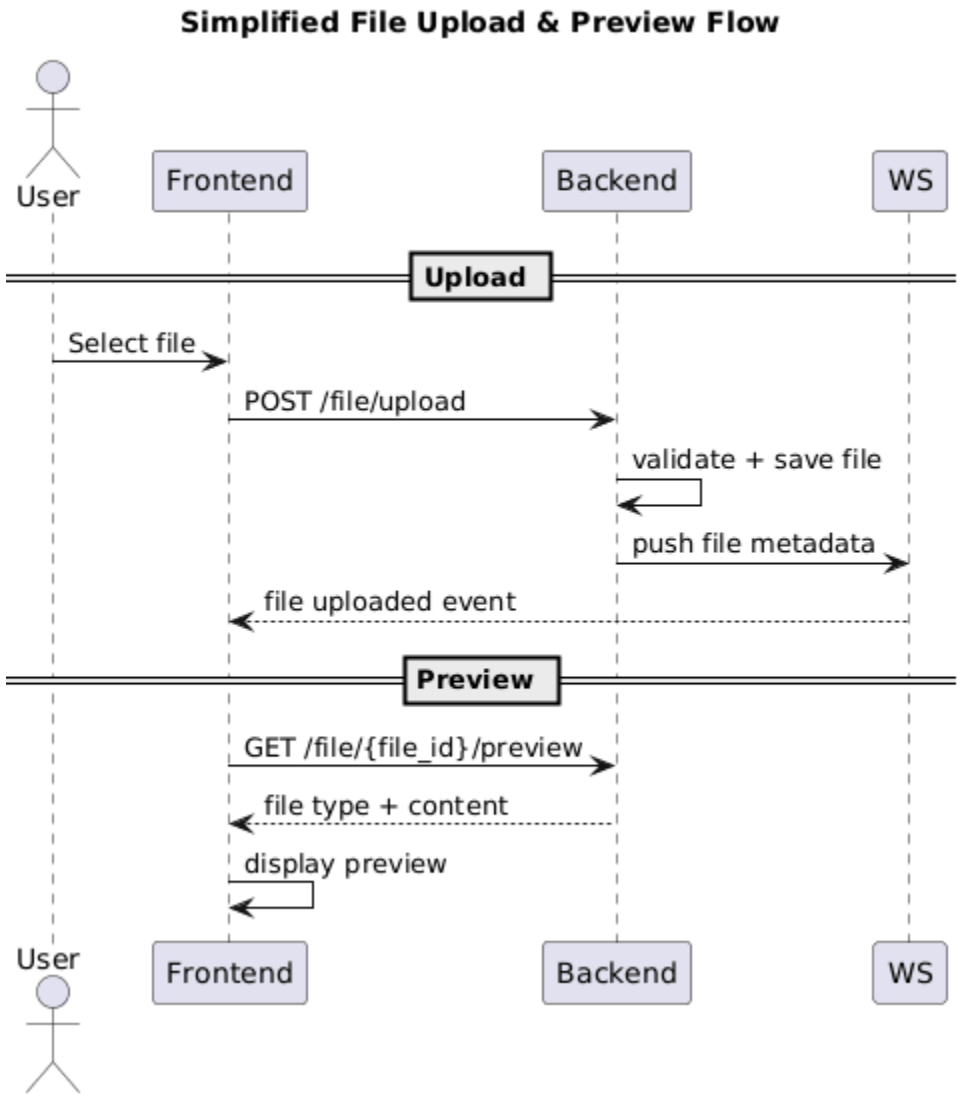
websocket_manager.broadcast(form_id, message_payload)

3.4.2 Receiving Messages
Frontend WebSocket client:

- **listens for new messages**
- **inserts into local state**
- **scrolls to bottom automatically**

3.4.3 Lazy-loading History
GET /api/v1/messages?form_id=X&page=2&page_size=20
returns older messages as user scrolls up.

3.5 File Upload and Preview Flow
UML PLOT 5 — File Upload & Preview Flow



3.5.1 Upload

1. **User selects file (≤ 10MB).**
2. **Frontend sends:**

POST /api/v1/file/upload

3. **Backend:**
    - **validates size**
    - **generates file_id**
    - **saves file at /storage/files/{file_id}/{filename}**
    - **creates DB record**
4. **Backend pushes message via WebSocket with file metadata.**

3.5.2 Preview
Frontend requests:
GET /api/v1/file/{file_id}/preview

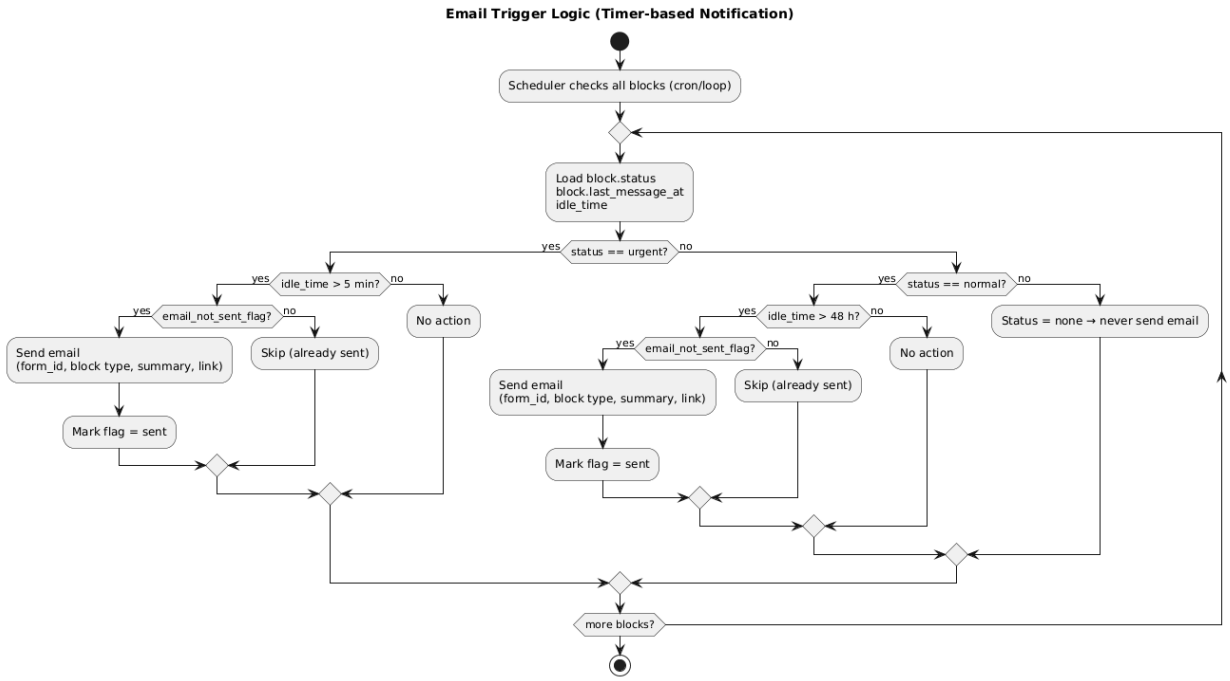
Backend returns:

```
{
  type: "image" | "text" | "code" | "binary",
  content: "..."   // base64 or text
}
```

Frontend displays:

- **image**
- **formatted text**
- **code editor**
- **“download only” when binary**

3.6 Email Notification Flow (Urgent / Normal Blocks)
UML PLOT 6 — Email Trigger Logic



3.6.1 Block Timer Rules

Block status	Email Rule
urgent	send email if no reply after 5 minutes
normal	send email if no reply after 48 hours
none	never send email

3.6.2 Backend Implementation

1. **Each block stores:**

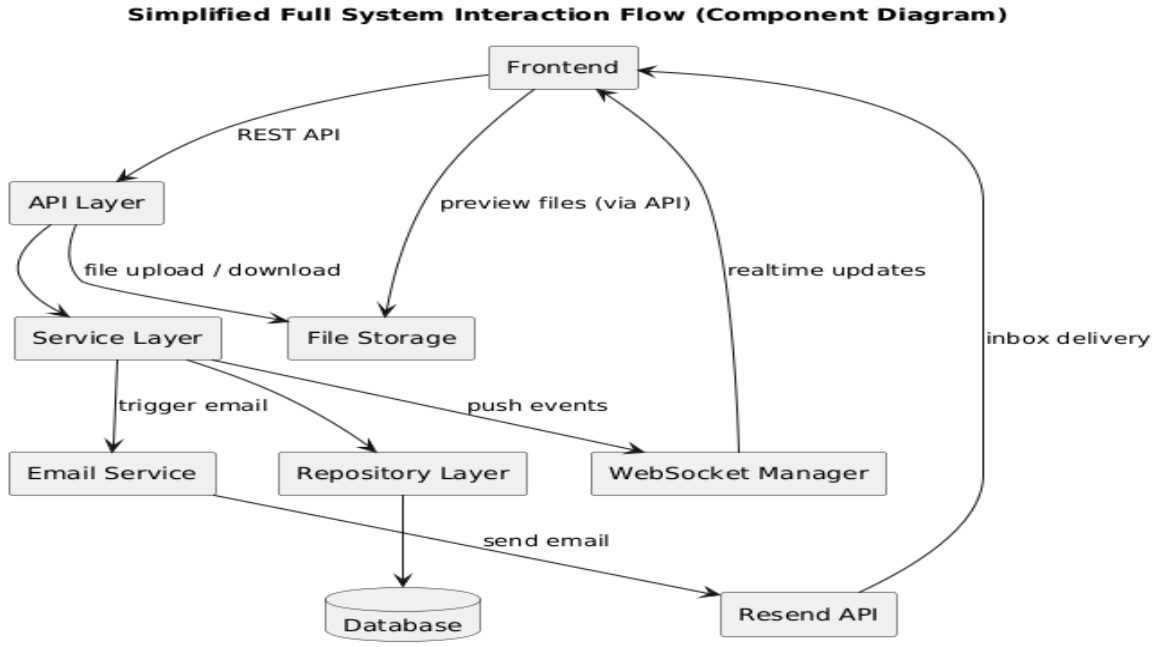
```
status
last_message_at
urgent_time = 5min
normal_time = 48h
```

2. **A scheduler (cron or async loop) checks all blocks:**

```
if block.status == urgent and idle_time > 5min:
   send email
if block.status == normal and idle_time > 48h:
   send email
```

3. **Email message includes:**
   - **form_id**
   - **block type**
   - **summary of last message**
   - **link to message room**
4. **Email is triggered only once per idle cycle (flag stored).**

3.7 System Integration Flow
UML PLOT 7 — Full System Interaction Flow



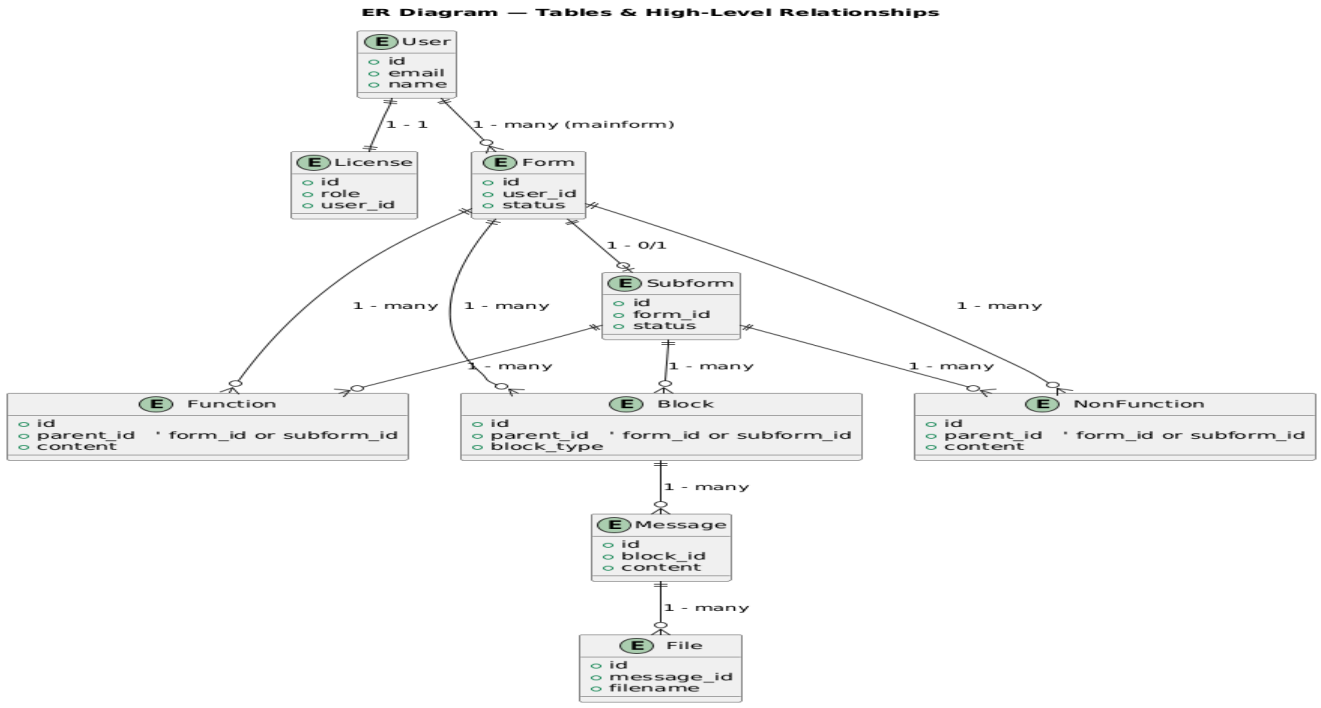
This plot shows how the three communication paths integrate:

- **REST API WebSocket Email**

4. Database Schema
This chapter defines the complete MySQL relational schema for SyncBridge.
The schema is designed to support:

- **mainform + subform (1-to-1 temporary version)**
- **functions & nonfunctions (1-to-many)**
- **blocks & messages (for general/function/nonfunction discussion)**
- **urgent/normal email triggering**
- **file attachments**
- **license-based roles (client/developer)**
- **controlled status transitions**
- **simple future extensibility**

4.1 Entity–Relationship Overview
UML PLOT 8 — ER Diagram (Tables & Relationships)



.2 Table Definitions

4.2.1 users

The users table stores basic user account information. Each user is identified by an auto-increment primary key id. The email field is unique and used for login, while password_hash stores the hashed password. The display_name field represents the user's nickname. The created_at and updated_at fields record the registration time and the last update time respectively.

4.2.2 licenses (each user has exactly one license)

The licenses table defines the system role and activation state for each user. Each license record has an auto-increment primary key id and a unique foreign key user_id referencing users.id, ensuring a strict one-to-one relationship between user and license. The role field is an enum limited to client and developer, defining the user's system capabilities. The activated_atfield records when the license becomes active. This design allows roles to be changed without modifying the user table, and no admin role is supported.

4.2.3 forms (mainform or subform)

The forms table represents both mainforms and subforms. Each record is identified by id and distinguished by the type field (main or sub). The owner_id references the user who created the form, typically a client. For subforms, parent_id references the associated mainform; this field is null for mainforms. Common form fields include title, message, budget, and expected_time. The status field applies only to mainforms and can be one of preview, available, processing, rewrite, end, or error. Timestamps are recorded in created_at and updated_at.

4.2.4 functions (functional requirements)

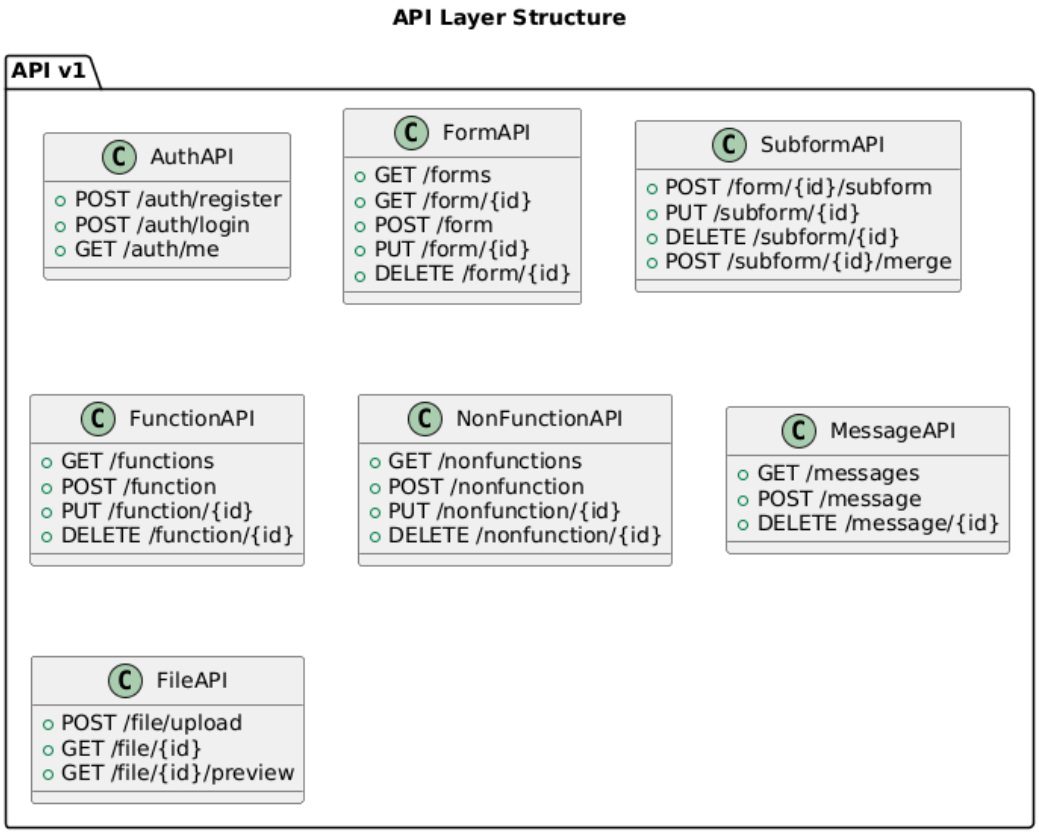
The functions table stores functional requirements associated with a form. Each function belongs to a mainform or subform via the form_id foreign key. The name and description fields describe the requirement, while choice indicates the service level (lightweight, commercial, or enterprise). The optional previous_function_id supports chained relationships between functions. The is_changed flag is set only when a subform's function differs from the mainform version. Creation and update times are recorded.

4.2.5 nonfunctions (non-functional requirements)
The nonfunctions table defines non-functional requirements linked to a form through form_id. Each record includes a name, a level enum (lightweight, commercial, enterprise), and a status enum (init, developing, finish, error). The function_weight field ranges from 1 to 10 to indicate relative importance. Similar to functions, is_changed marks differences in subforms, and timestamps track creation and updates.

4.2.6 blocks
The blocks table represents message groupings within a form or subform. Each block references a form via form_id and is categorized by type (general, function, or nonfunction). The optional target_id links the block to a specific function or nonfunction when applicable. The status field (urgent, normal, or none) controls email reminder behavior. Creation and update timestamps are maintained.

4.2.7 messages
The messages table stores individual messages within a block. Each message belongs to exactly one block via block_id and records the sender through user_id. The message content is stored in text_content, which may be null, and timestamps record when the message is created and updated.

4.2.8 files
The files table manages file attachments associated with messages. Each file record references a message through message_idand stores metadata including file_name, file_type, and file_size. The path field follows the format /storage/files/{file_id}/{filename}. The preview_generated flag indicates whether a preview has been created. File creation time is recorded in created_at.



4.3 Constraints and Integrity Rules

4.3.1 One mainform → zero or one subform

Each mainform may have zero or one subform only. This constraint is enforced at the service logic level by checking the number of forms whose parent_id equals the mainform ID, ensuring the count does not exceed one (e.g., SELECT COUNT(*) FROM forms WHERE parent_id = mainform.id <= 1).

4.3.2 Function & NonFunction belong to one form

Each Function and NonFunction entity must belong to exactly one form. This relationship is enforced through foreign key constraints, and deletions cascade automatically to maintain referential integrity.

4.3.3 Message flow

Each message block must belong to a valid mainform or subform. Messages are required to belong to exactly one block, ensuring that all message flows are correctly scoped and traceable.

4.3.4 File limitations

Uploaded files must be smaller than 10 MB, which is enforced at the service layer. Binary previews are not stored in the system.

4.3.5 License integrity

Each user is associated with exactly one license. User roles are not stored directly in the user table and are instead determined via the license mechanism.

---

5. API Specification

All API endpoints follow a unified design. The base URL is /api/v1. Responses use a standardized JSON format containing status, message, and optional data. Role-based access control is enforced on the backend using JWT authentication combined with the license table. All timestamps returned by the API are in UTC.

5.1 Authentication Endpoints
POST /auth/register
Registers a new user with email, password, and display name. On success, the API returns user information, a JWT token, and a default client license role. Errors include email conflicts (409) and invalid or missing input fields (400).
POST /auth/login
Authenticates a user using email and password. On success, the response includes user information, a JWT token, and the associated license role. Errors include incorrect credentials (401) and inactive licenses (403).
GET /auth/me
Returns information about the currently authenticated user, including basic user details and license role.

5.2 Form Endpoints
Forms consist of mainforms and subforms, but only mainforms appear in form listings.
GET /forms
Returns a list of forms filtered by role. Developers can view all available forms and all mainforms assigned to them that are in processing or completed states.
GET /forms/{id}
Fetches full details of a mainform, including its optional subform, associated functions, nonfunctions, and all related message blocks. Both clients and developers can view mainform data, while subform editing privileges depend on role and allowed status transitions.
POST /form
Allows a client to create a new mainform with basic details and optional functional and nonfunctional requirements.
PUT /form/{id}
Allows editing of a mainform only when permitted by its current status. Clients may edit their own forms in preview or available states, while developers may edit forms only during rewrite.
DELETE /form/{id}
Allows a client to delete their own form before it enters the processing state.

5.3 Subform Endpoints
POST /form/{id}/subform
Allows a client or developer to propose a rewrite by creating a new subform.
PUT /subform/{id}
Updates fields of a subform and marks modified fields as changed.
DELETE /subform/{id}
Rejects a subform, deletes it, and reverts the mainform to its previous status.
POST /subform/{id}/merge
Merges a subform into the mainform once both parties agree.

5.4 Function Endpoints
Functions can be created, updated, and deleted via /function endpoints. All responses follow the unified response format, and role-based rules apply to modification and deletion.

5.5 Nonfunctional Endpoints
Nonfunctional requirement endpoints mirror the structure and behavior of function endpoints.

5.6 Messaging Endpoints
Messages are associated with specific blocks.

GET /messages
Returns a list of messages with sender information, text content, attached files, and timestamps.
POST /message
Sends a message to a specific block within a form or subform, optionally including file attachments.
DELETE /message/{id}
Allows only the original sender to delete a message.

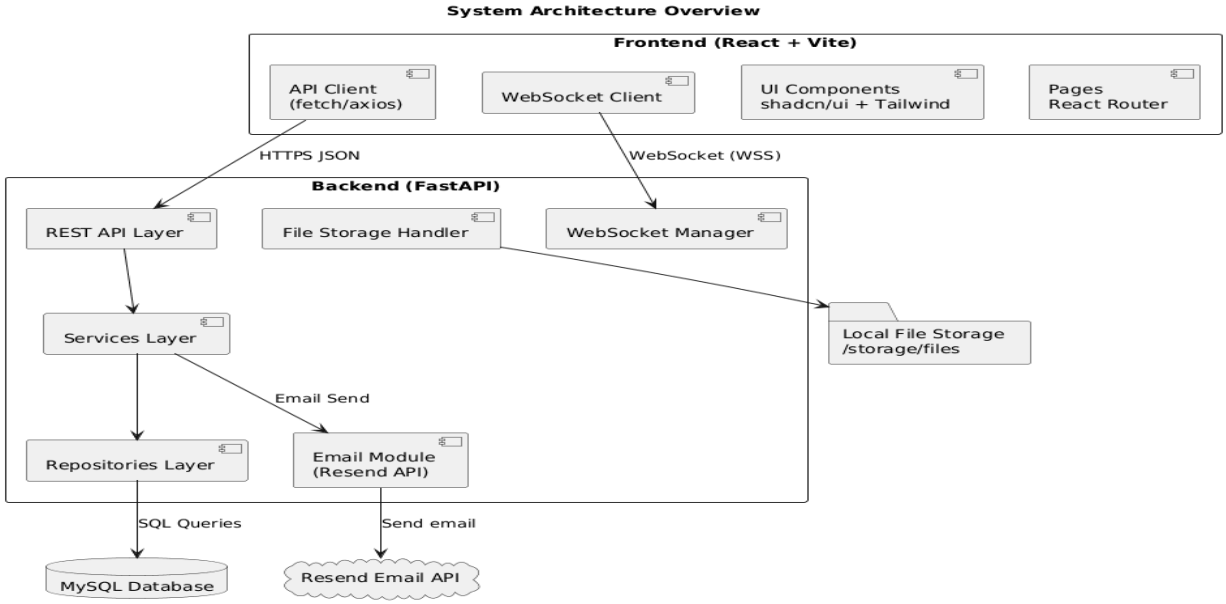
5.7 Files Endpoints
POST /file/upload
Uploads a file subject to the 10 MB size limit.
GET /file/{file_id}
Downloads a file.
GET /file/{file_id}/preview
Returns a preview of text, image, or code files in a base64-encoded format.

5.8 Error Codes Overview
400 invalid input; 401 authentication required; 403 not permitted by role; 404 resource not found; 409 status or rule conflict; 422 validation failed; 500 internal server error.

6. UML Diagrams
This chapter provides the core structural and behavioral diagrams for the SyncBridge system. They are intended for backend developers, frontend developers, and system architects to understand data relationships.
The diagrams included are:

1. **System Architecture Diagram**
2. **Use Case Diagram**
3. **Domain Model / ER Diagram**

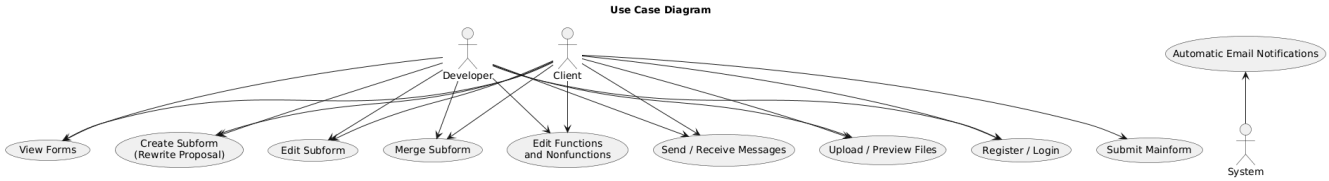
6.1 System Architecture Diagram



6.2 Use Case Diagram (Client + Developer)

- **Automatic email notifications**

UML Diagram 2 — Use Case Diagram



6.3 Domain Model (Entity–Relationship Diagram)