Functional Specification: Collaborative Code Editor

CSC1049
Hoang Xuan Hai Mai - 22115838
Bailey Anthony Scanlan - 23316363

Table of Contents

1. Introduction

1.1 Overview

This document defines the functional specifications of this project, which we are calling the “Collab code”. Hoang Xuan Hai Mai and Bailey Anthony Scanlan were supervised by Dr. Geoff Hamilton. Collab Code Editor is a Code Editor IDE that allows multiple users to edit the code in the same document or folder in real time, making it a “Google Doc for developers”. The system synchronises changes instantly across all connected clients, ensuring that participants can see the newest version of the code without conflicts.

1.2 Glosarry

Term	Definition
API	**Application Programming Interface**: A set of rules and protocols that allow different software applications

	to communicate with each other.
IDE	**Integrated Development Environment**: a software application that provides developers with a comprehensive set of tools in a single interface to write, compile, and debug code more efficiently.
HTTP/HTTPS	**Hypertext Transfer Protocol**: protocols for client-server communication, with GET, PUT, POST, and DELETE methods
Websocket	A communication protocol that provides full-duplex, bidirectional communication over a single, persistent TCP connection, allowing for real-time data transfer between a client and a server.

2. General description

2.1 System functions

This system will allow a user or users to be able to edit and work on a folder or file in a code IDE environment. The application will be a team-focused app with other team-focused aspects, such as a chat feature, as well as a call feature.

Users will be able to sign up to the collab code in order to be able to identify them, and then when they either are added to or create a multi-user document or folder, other users will be able to see what each individual user is typing, where each individual user's cursor is, as well as what document each individual user is currently editing. Remote users can open and edit the same document and see each other's changes instantly. Remote edits are merged so that all users see a consistent version of the document. Users will also be able to create, rename, and delete projects, and within a project, users can create, open, rename, and delete code files. The system persists documents so that they can be reopened and edited across sessions.

The editor provides syntax highlighting for one or more common programming languages (Python, JavaScript, etc). Login access control ensures that only authorised users can view or edit specific projects or documents. The system will periodically save the current state of the document for persistent storage, and optionally, the system may store a history of document revisions or major snapshots for recovery purposes.

For supported languages, users may execute code within a controlled environment, execution requests are sent to a backend component, and the output is returned to the users. Execution is subject to safety limitations

2.2 User characteristics and objectives

2.2.1 User community and Expertise

The primary user groups of the Collaborative code editor are: Students/Novice, these users are likely to have basic familiarity with coding at least one language and should also be comfortable using web-based tools and development environments, but may have limited experience with professional IDEs. But these users may also work individually or in small groups on assignments, labs, or projects. Teaching staff(lecturers, tutors, demonstrators) may also use this system, they are experienced with software development and programming tools and require the ability to monitor, demonstrate, and support students activities. These users will also need to be able to quickly set up example projects, share code with a class, or join student sessions. The final primary user group is general developers / collaborators, these users are familiar with version control systems and IDEs, and expect reasonably responsive editing and reliable synchronization between collaborators.

2.2.2 User objectives

From the user's perspectiv,e the main objectives of the system are collaborative learning and developmen,t meaning that users can work together on code in real time for lab exercises, assignments and

projects and also observe peers or instructors changes as they happen and learn from them
Clear visibility of shared work, see who is working in the same file and also understand what parts of the code are being edited, by whom, and when this will require reliable preservation of work meaning that users need to be able to trust their work is regularly and safely saved, and to also be able to reconnect and resume editing without loss of progress.

2.2.3 User “wish list” vs Feasibility scope

From the user's perspective, an ideal collaborative editor would include basic IDE features such as intelligent code completion, refactoring tools, and integrated debugging. As well as this full language support with automatic detection and language servers. Deep integration with external version control systems like Github is also needed. For better team management real time audio and video communication within the editor would also be wanted from the users perspective. And finally fine-grained access control and role based permissions (owner, editor, reviewer, etc).

2.3 operational scenarios

This section outlines representative scenarios that describe how users interact with the collaborative code editor in practice. These scenarios are written from the users perspective and illustrate the end-to-end system behaviour
Scenario 1: Pair Programming Session

1. Student A logs in to the system.
2. Student A creates a new project and adds a Python file for an assignment
3. Student A shares the project or document link with Student B
4. Student B logs in and opens the shared document.
5. Both students see the same code in the editor, along with indicators that the other user is present.
6. Student B edits comments and fixes a variable name at the same time; Student A sees these edits without conflict.

7. The system periodically saves the document to persistent storage while both students work.
8. When they finish, both students close the system. Later, either student can log in and reopen the document with the latest saved content.

Scenario 2: Instructor Demonstration

1. An instructor logs in and opens a prepared example project for a lecture.
2. The instructor shares the document link with all students in the class.
3. Students join the session and open the document.
4. As the instructor types or modifies code, all students see the changes in real time.
5. The instructor adds explanatory comments and may highlight or edit specific lines while talking.
6. A few students are given edit permission and can suggest changes live, which all participants see.
7. After the demonstration, the project remains available so students can revisit the example and explore the code independently

Scenario 3: Asynchronous Group Work

1. A three person group is assigned a project.
2. Student C creates a project and initial files in the editor.
3. Students D and E are added as collaborators.
4. The group members work at different times during the day.
   - When student C is online, they implement some functions.
   - Later, Student D logs in, opens the same files, and continues work from the last saved state.
   - Student E logs in at night, refactors part of the code, and saves.
5. At any time when two or more group members happen to be online simultaneously, real-time collaboration is possible; otherwise, they simply see the latest persisted state when they open the files.
6. The group can track their progress by reopening the project and reviewing the final integration code.

Scenario 4: Handling Network Interruptions

1. A student is working in the editor with a collaborator.

2. The student’s network connection becomes unstable.
3. The editor detects loss of connection and indicates that collaboration is temporarily offline, while still allowing the student to edit locally.
4. When the connection is re-established the local changes reconcile with the remote changes and updates the document so that all users see a consistent final version.
5. The user receives a brief indication that synchronization has completed.

2.4 constraints

The design and implementation of the Code Editor are subject to the following constraints:

Platform and environment

- The system is intended to run in a standard environment.
- The backend will run on a typical server environment available for academic projects.
- The system should operate correctly on common student hardware and standard campus network conditions

Performance and responsiveness

- Real-time collaborative edits should appear to remote users with minimal perceived delay (aiming for sub-second latency under normal conditions)
- The system must handle a reasonable number of concurrent users and active sessions consistent with a small class or project group, rather than large-scale internet deployment.

Technology and protocols

- Communication between frontend and backend will use backend HTTP/HTTPS for standard API requests and a real-time protocol such as WebSockets for collaborative editing events.
- The backend will use a commonly available additional database system (SQKite, PostgreSQL, etc) supported by the chosen web framework.
- The project should rely on widely supported, open technologies suitable for an academic environment.

Security and privacy

- User authentication must prevent unauthorized access to private projects and documents.

- All communication may be required to use HTTPS when deployed in a real environment to protect credentials and content in transit.
- The system will not store highly sensitive personal data beyond what is necessary for user identification and access control.

Scope and development constraints

- The system is developed as a 3rd year project, with limited time, team size and infrastructure.
- Advanced IDE-like features are out of scope.
- Integration with external services is optional and not required for functionality.

3.Functional requirements

This section describes the functional requirements of the Collaborative Code Editor (CCE) in ranked order of criticality. Each requirement specifies what the system must do, independently of how it is implemented. Non-functional requirements are covered elsewhere.

For each requirement:

- Description - what the system must accomplish.
- Critical - importance to the system overall (high / medium / low)
- Technical issues - key design or implementation considerations.
- dependencies - other requirements that this requirement relies on

3.1 Core Functional Requirements (high criticality)

FR-1: User Authentication and Session Management

Description

- The system shall provide user registration, login, and logout functionality. Each user shall have a unique identity and an authenticated session to access personal and shared projects and documents

Criticality

- High - required to distinguish between users, enforce access control, and support collaboration

Technical issues

- Secure storage of user credentials (e.g password hashing)
- Session management and token handling for both HTTP and real-time connections (e.g websockets)
- Handling session expiry and re-authentication gracefully

Dependencies

- FR-2 (Project Management)
- FR-3 (Document management)
- FR-7 (Access Control and Sharing)
- Required by all collaborative and persistence features that are user-specific.

FR-2: Project Management

Description

- The system shall allow authenticated users to create, view, rename, and delete projects. A project represents a logical grouping of one or more code documents owned and shared by a user or a team.

Criticality

- **High** - central to organizing documents and collaboration sessions.

Technical issues

- Maintaining a clear project ownership and association with users.
- Ensuring that deletion of a project is handled safely, including deletion or archival of associated documents.
- Efficient retrieval of a user’s project list.

Dependencies

- FR-1 (User Authentication and Session Management)
- FR-3 (Document Management)
- FR-7 (Access Control and Sharing)

FR-3: Document Management

Description

- The system shall allow users to create, open, rename, and delete code documents within a project, The system shall persist the content and metadata of each document and make it available for later editing.

Criticality

- **High** - required for any editing or collaboration to occur.

Technical issues

- Representation of documents in persistent storage, including project association and language type.
- Ensuring consistent document metadata when documents are renamed or moved.
- Safe deletion, including preventing accidental data loss

Dependencies

  - FR-1 (User authentication and session management)
  - FR-1 (Project management)
  - FR-6 (Document Persistence and autosave)
  - FR4/FR-5 (Real-time collaboration features) when documents are opened collaboratively.

FR-4: Real-Time Collaboration Session Management

Description

- The system shall allow multiple authenticated users to join a shared editing session on the same document. The system shall manage active sessions, track connected participants, and handle user join/leave events.

Criticality

- **High** - Core requirement distinguishing this system as collaborative editor.

Technical issues

- Establishing and maintaining real-time connections per document
- Efficient management of concurrent sessions and connected users. Handling connection drops, reconnection, and consistent sessions state across clients.

Dependencies

- FR-1 (User Authentication and Session Management)
- FR-3 (Document Management)
- FR-5 (Real-Time Text Synchronization and Conflict Resolution)

- FR-8 (User Presence and Cursor Indication) for enhanced collaboration visibility

FR-5: Real-Time text synchronization and conflict resolution

Description

- The system shall transmit and apply text editing operations among all connected clients in real time, ensuring that all users eventually see a consistent document state even under concurrent edits.

Criticality

- **High** - Fundamental to collaborative editing behaviour.

Technical issues

- Choosing and implementing a concurrency control mechanism.
- Maintaining document versions or operation sequences to transform or merge concurrent edits.
- Ensuring performance and responsiveness under rapid editing.
- Handling synchronization after temporary disconnection and reconnection

Dependencies

- FR-4 (Real-time collaboration session management)
- FR-3 (Document Management)
- FR-6 (Document Persistence and autosave) for storing consistent states.

FR-6: Document Persistence and Autosave

Description

- The system shall persist document content to durable storage at appropriate intervals and when users explicitly save. Upon reopening a document, the latest saved state shall be restored.

Criticality

- **High** - Necessary to prevent data loss and support asynchronous work.

Technical issues

- Deciding autosave frequency and triggers to balance safety and performance.
- Managing race conditions between autosave operations and incoming real-time edits.
- Handling partial or failed saves and providing fallback or recovery options.

Dependencies
- FR-3 (Document Management)
- FR-5 (Real-time text synchronization and conflict resolution)
- FR-4 (Real-time collaboration session management), since multiple users may be editing when autosave occurs.

FR-7: Access control and sharing
Description
- The system shall restrict access to projects and documents based on user privileges. Project owners shall be able to share access with other users, granting at least view or edit permissions.

Criticality
- **High** - Required to maintain privacy and control over collaborative workspaces

Technical issues
- Designing a permissions model that is simple yet sufficient.
- Enforcing permissions directly at the API level for all relevant operations.
- Handling changes in permissions during an active collaboration session.

Dependencies
- FR-1 (User authentication and session management)
- FR-2 (Project Management)
- FR-3 (Document management)
- FR-4 (Real-time collaboration session management for enforcing permissions in active sessions.

3.2 Important Supporting Functional Requirements (Medium Criticality)

FR-8: User Presence and cursor /selection indication
Description
- The system shall indicate which users are currently viewing or editing a document and, where feasible, show their cursor position and /or selected text regions in the editor.

Criticality

- **Medium** - Strongly improves user experience and collaboration clarity but not strictly required for basic functionality.

Technical issues

- Efficient propagation of cursor and selection updates over real-time channels.
- Avoiding excessive updates frequently that could affect performance.
- Distinguishing users visually while ensuring accessibility.

Dependencies

- FR-4 (Real-time collaboration session management)
- FR-5 (Real-time text synchronization and conflict resolution)
- FR-1 (User authentication and session management) for user identity

FR-9: Syntax Highlighting and Basic Editing Assistance

Description

- The system shall provide syntax highlighting for supported programming languages and basic editing features such as indentation support, bracket matching, and simple code formatting.

Criticality

- **Medium**- Important for usability and readability, but the system could technically function as a plain text editor.

Technical Issues

- Integrating or configuring an appropriate editor component with language modes.
- Ensuring that syntax highlighting remains correct as real time edits occur.
- Handling multiple languages and file types where applicable.

Dependencies

- FR-3 (Document management) - for language/file type metadata.
- FR-5 (Real-time text synchronization and conflict resolution) - for consistent text state.

FR-10: Error Handling and User Feedback

Description

- The system shall detect and report errors related to authentication, project and document operations, collaboration connectivity, and saving. Users receive clear messages indicating the nature of the problem and , where possible, recommended actions.

Criticality

- **Medium** - Necessary for robustness and user trust, though the editor could technically function with minimal feedback

Technical Issues

- Defining constant error codes and messages across the backend.
- Gracefully handling temporary network interruptions and reconnection.
- Differentiating between transient failures and permanent failures.

Dependencies

- FR-1 through FR-7 (all core operations may produce errors)
- FR-4 and FR-5 specially for real time connectivity issues.

3.3 Optional/Extended Functional Requirements (Low criticality)

FR-11: Basic Code Execution in a sandbox

Description

- The system may allow users to execute code from a document in a controlled, sandbox environment and display the resulting output within the interface.

Criticality

- **Low** - Valuable for learning and rapid feedback, but not essential for collaborative editing.

Technical issues

- Isolating execution from the main application to ensure security and stability.
- Setting and enforcing resource limits.
- Handling long-running or faulty programs gracefully.
- Transmitting code and input to the execution service and returning outputs.

Dependencies

- FR-3 (Document Management) to receive current document content.
- FR-6 (Document Persistence and Autosave) to ensure code being run is up to date (if required)

- FR-1 (User Authentication and Session Management) if execution is tied to user identity.

FR-12: In-Editor Text Chat or Comments

Description

- The system may provide a basic text based communication channel for collaborators working on the same project or document to discuss changes and coordinate tasks.

Criticality

- **Low** - Enhances collaborative experience but is not required for joint editing.

Technical issues

- Implementing a messaging channel and storing messages if persistence is required.
- Handling message ordering and delivery guarantees.
- Providing a lightweight, non-intrusive UI for communication.

Dependencies

- FR-1 (User Authentication and Session Management)
- FR-4 (Real-time collaboration session management) for live chat and active sessions.

FR-13: Document Version History and Restore

Description

- The system may maintain a history of document version or major snapshots, allowing users to inspect past states and optionally restore a previous version.

Criticality

- **Low**- Useful for recovery and auditing, but not essential for minimal operation.

Technical issues

- Choosing a versioning strategy.
- Managing storage overhead for multiple versions.
- Providing a clear UI to browse and restore previous versions while avoiding accidental data loss.

Dependencies

- FR-6 (Document Persistence and Autosave)
- FR-3 (Document Management)
- FR-5 (Real-Time Text Synchronization and Conflict Resolution) if versioning interacts with collaborative changes.

4. System Architecture

This section provides a high level overview of the anticipated system architecture for the Collaborative Code Editor. It describes how responsibilities are distributed across major system modules and identifies key third-party or reused components.

4.1 Architectural Overview

The Collaborative Code Editor is designed as a rich client application that runs on the user’s machine and connects to a remote collaboration backend over the network.
At a high level, the system consists of
Desktop Client Application

- Provides the graphical user interface (GUI).
- Hosts the code editor and local editing logic.
- Manages user sessions, local state and network communication.

Collaboration and data server

- Exposes network APIs for authentication, project and document management.
- Coordinates real-time collaboration sessions between multiple client instances.
- Persists users, projects, documents, and (optionally) version history and execution logs.

Data storage layer

- Provides durable storage for all application data managed by the server

Communication between client applications and the server uses standard network protocols.

- A request-response API for authentication, project and document operations.
- A real-time channel for collaborative editing events and presence updates.

4.2 Client-Side Architecture

The application is responsible for all user-facing interaction and local editing behaviour. It can be implemented using a cross-platform GUI framework.

Application Shell and Navigation Module

- Initializes the application and main window.
- Manages navigation between screens.
- Coordinates high-level application state.

Authentication and session modules

- Provides user logic and logout functionality.
- Stores session tokens or credentials securely on the client.
- Attaches authentication information to outgoing requests to the server.
- Handles session expiry and re-authentication prompts.

Project and document management UI

- Displays the list of projects and documents accessible to the user.
- Invokes server APIs to create, rename, open, and delete projects and documents.
- Manages local representations of project/document metadata to support responsive navigation.

Code editor and local editing Engine

- Embeds a third party code editor component to provide text editing, syntax highlighting, and basic code editing features.
- Maintains an in-memory representation of the currently open document's text.
- Captures local editing actions and forwards them to the collaboration module.
- Applies incoming remote operations so that the displayed text remains synchronized with other collaborators.

Real-time collaboration client module

- Establishes and maintains a bidirectional real-time connection to the collaboration server.
- Encodes local operations and presence updates into collaboration messages and sends them to the server.
- Receives transformed operations and presence information from the server and applies them to the local editor.
- Handles connection loss, reconnection attempts,and resynchronization after outages.

Feedback, error handling, and notification module

- Interprets error code and messages received from the server.
- Displays the user-friendly notifications.
- Provides visual indicators for synchronization status.

Optional Execution and output module

- Provides a panel or window within the application for running supported code and viewing its output.
- Sends “run code” requests to the backend execution service.
- Displays returned output and error messages to all collaborators as appropriate.

4.3 Server side Architecture

The server is responsible for enforcing business rules, coordinating collaboration, and managing persistence. It is logically divided into the following modules.

Network API and Routing Module

- Listens for incoming connections from desktop clients.
- Exposes structured APIs for authentication, project management, and document operations.
- Routes messages and requests to the appropriate internal services.
- Enforces authentication and basic input validation at the boundary.

Authentication and Authorization service

- Manages user accounts and credentials.
- Validates login attempts and issues session tokens or equivalent identifiers.
- Evaluates authorization rules to determine whether a user may perform a requested operation

Project and Document Management services

- Implements server-side logic to create, rename, delete, and list projects and documents.
- Validates operations according to ownership and shared access rights.
- Interacts with the data storage layer to read and persist metadata and document content.

Real-Time Collaboration Service

- Maintains live collaboration sessions for documents currently being edited.
- Tracks connected participants for each document and manages per-session state.
- Receives edit operations and presence updates from clients over real-time channels.
- Uses the collaboration engine to process and transform operations, then broadcast the resulting operations to all participants.

Collaboration Engine.

- Implements the algorithm used for conflict-free collaborative editing.
- Maintains versioning or operation logs per active document.
- Provides functions to:
    - Apply a new operation to the current document state.
    - Transform or reconcile concurrent operations from different clients.
    - Resynchronise a client that has reconnected after temporary disconnection.

Persistence and Autosave service

- Periodically writes the authoritative document state from active sessions to the database or storage system.
- Handles explicit save requests from clients.
- Manages optional versioning or snapshots to support later inspection and restore.
- Ensures that a consistent, recoverable state exists even when no session is active.

Monitoring, logging and Error handling

- Logs key events, errors, and even performance metrics.
- Supports debugging and operational monitoring.
- Provides structured error responses for both requests-responses APIs and real-time channels.

4.4 Data Storage layer

The data storage layer is dedicated to persistent application data. It is accessed only through the server side services.
Key stored entities include:

Users

- Username, hashed password, basic profile information.

Projects

- Project identifiers, names, owners, and lists of collaborators.
- time stamps for creation and modification.

Documents

- Document identifiers, associated project, filenames, language or type, and latest saved content.

Version History

- Historical snapshots or change records, each linked to a document, version number, timestamp and author.

Execution Logs

- Records of code execution requests and outputs for auditing or debugging.

A relational database is envisaged for structured entities such as users, projects, and documents. Document content may also be stored in text fields within the same database or in a specialized format, depending on size and performance needs.

4.5 Auxiliary and External Services

In addition to the main server, the architecture may incorporate

Code execution sandbox

- A separate process or service, possibly running on a different host.
- Executes user code under strict resource and security constraints.
- Communicates with the main server via an internal protocol or API.

Enterprise / institutional infrastructure

- Integration with institutional directories if deployed in an educational or organizational context.
- Use of existing logging monitoring, or backup system where available.

4.6 Distribution of Functions and Scalability

In the initial deployment, a typical configuration is:

- Multiple desktop client applications, each running on user machines.
- A single central collaboration and data server instance, hosting all server-side modules.
- A single database instance.

The architecture is intentionally modular so that if necessary, the following evolutions are possible.

- Separating the real-time collaboration service, REST API services, and code execution service into distinct server processes or containers.
- Scaling the server horizontally by adding additional instances behind a load balancer, while sharing a common database.
- Upgrading the database to a managed or clustered solution as the number of users or documents grow.

5. High Level Design

This section describes the high level design of the collaborative code editor as a networked desktop application. It introduces the main system components, how they interact with each other and with external entities.

5.1 Design Objectives and Approach

The system follows a layered, modular client-server design.

- A desktop client application provides the graphical user interface, code editing capabilities, and real-time collaboration client.
- A central collaboration server provides authentication, project and document management, real-time coordination, and persistence.
- A data storage layer provides durable storage for users, projects, documents, and
-
-

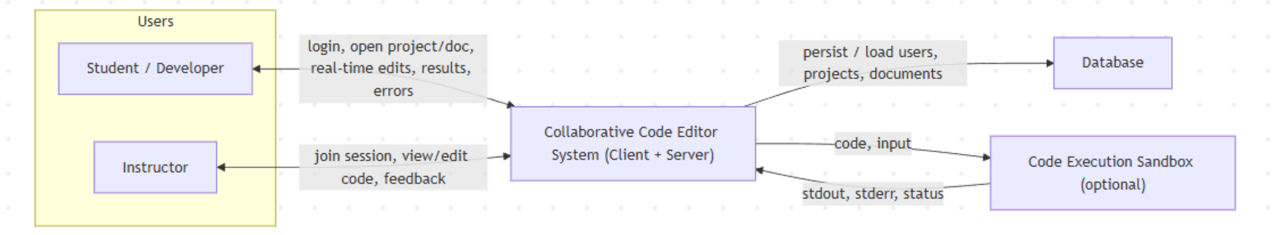
-
- optionally version history and execution logs.
- An optional code execution service runs user code in a sandboxed environment.

Design Goals.

- Clear separation of concerns.
- Direct traceability to functional requirements.
- Extensibility for future features.
- Robust handling of concurrent edits by multiple users.

5.2 System Context Model

The system context describes how the collaborative code editor interacts with external entities.


Users
Student / Developer
Instructor
login, open project/doc, real-time edits, results, errors
join session, view/edit code, feedback
Collaborative Code Editor System (Client + Server)
persist / load users, projects, documents
Database
code, input
stdout, stderr, status
Code Execution Sandbox (optional)

External entities:

- Student / Developer - runs the desktop client, creates projects, edits code, and participates in collaborative sessions
- Instructor - may join sessions to demonstrate or review code.
- Database - persistent storage for users, projects documents, and version history.
- Code Execution Sandbox - runs user code and returns output.

5.3 Major Subsystems and layers

5.3.1 Client-side subsystems

Authentication UI

- Login/Register forms.
- Store auth token.

Project and Document UI

- Project list with create/rename/delete.
- Document list in a project.
- Version history of a project.

Editor and collaboration UI

- Monaco editor.
- Connects WebSocket for the document
- Show cursor highlights.
- Participant name for users in the room.

Chat Panel

- For users to communicate with each other

5.3.3 Data storage subsystem

User Repository

- Stores user credentials, identities, and roles.

Project Repository

- Stores project metadata and links between projects and users.

Document Repository

- Stores document metadata and latest content.

Version/History Repository

- Stores snapshots or diffs of document versions,

Execution log Repository

- Stores records of code execution requests and outputs.

5.4 Data flow model

This model describes the main processes and how data flows between them, users, and data stores.

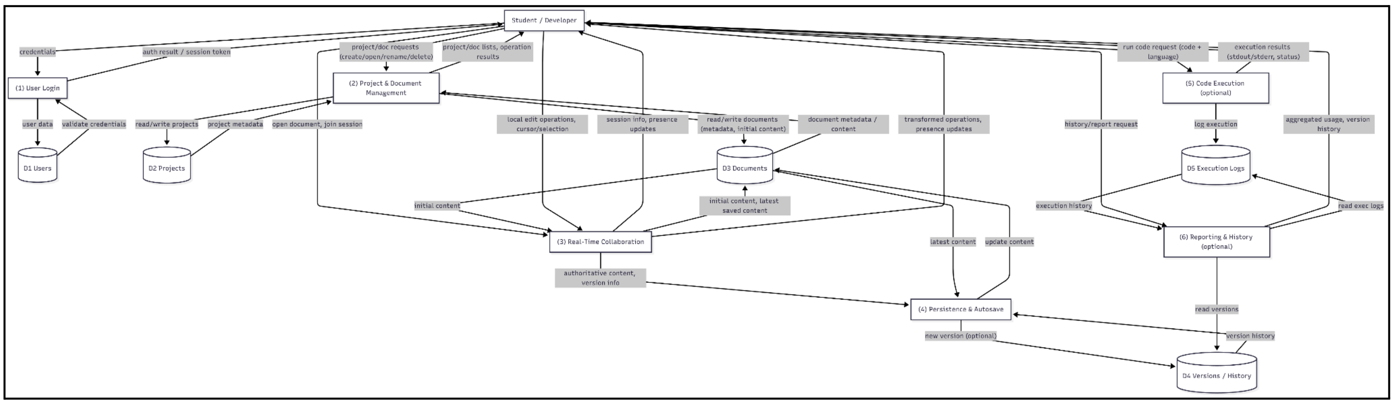
Main processes:

1. User login
2. Project and document management
3. Real-Time collaboration
4. Persistence and autosave
5. Code execution
6. Reporting and history

Data Stores

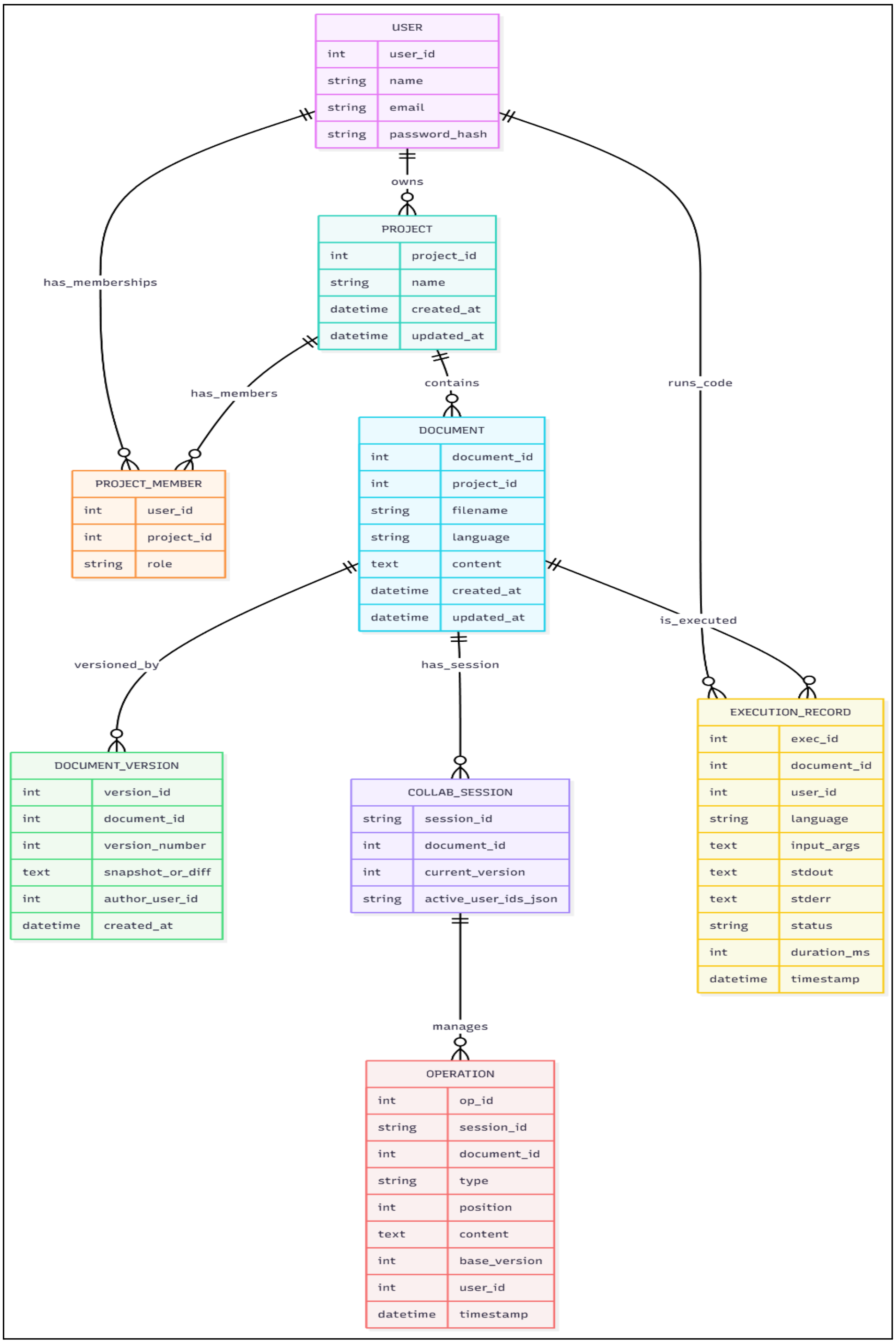
- D1 Users
- D2 Projects
- D3 Documents

- D4 Versions / History
- D5 Execution Logs


Student / Developer
credentials
auth result / session token
project/doc requests (create/open/rename/delete)
project/doc lists, operation results
run code request (code + language)
execution results (stdout/stderr, status)
(1) User Login
(2) Project & Document Management
(5) Code Execution (optional)
user data
validate credentials
read/write projects
project metadata
open document, join session
local edit operations, cursor/selection
session info, presence updates
read/write documents (metadata, initial content)
document metadata / content
transformed operations, presence updates
history/report request
log execution
aggregated usage, version history
D1 Users
D2 Projects
D3 Documents
D5 Execution Logs
initial content
initial content, latest saved content
execution history
read exec logs
(3) Real-Time Collaboration
latest content
update content
(6) Reporting & History (optional)
authoritative content, version info
(4) Persistence & Autosave
read versions
new version (optional)
version history
D4 Versions / History

5.5 Logical Data Structure (LDS)

This model captures the main data entities and their relationships.



USER	
int	user_id
string	name
string	email
string	password_hash

PROJECT	
int	project_id
string	name
datetime	created_at
datetime	updated_at

PROJECT_MEMBER	
int	user_id
int	project_id
string	role

DOCUMENT	
int	document_id
int	project_id
string	filename
string	language
text	content
datetime	created_at
datetime	updated_at

DOCUMENT_VERSION	
int	version_id
int	document_id
int	version_number
text	snapshot_or_diff
int	author_user_id
datetime	created_at

COLLAB_SESSION	
string	session_id
int	document_id
int	current_version
string	active_user_ids_json

EXECUTION_RECORD	
int	exec_id
int	document_id
int	user_id
string	language
text	input_args
text	stdout
text	stderr
string	status
int	duration_ms
datetime	timestamp

OPERATION	
int	op_id
string	session_id
int	document_id
string	type
int	position
text	content
int	base_version
int	user_id
datetime	timestamp

Key relationships (textual)

- A user owns zero or more projects and can collaborate on additional projects via “project_member”.
- A project contains one or more Documents.
- A document may have zero or more DocumentVersions.
- A CollaborationSession is associated with exactly one Document and coordinates Operations in memory.
- ExecutionRecord links a user and a document to a specific run of code and its outcome.

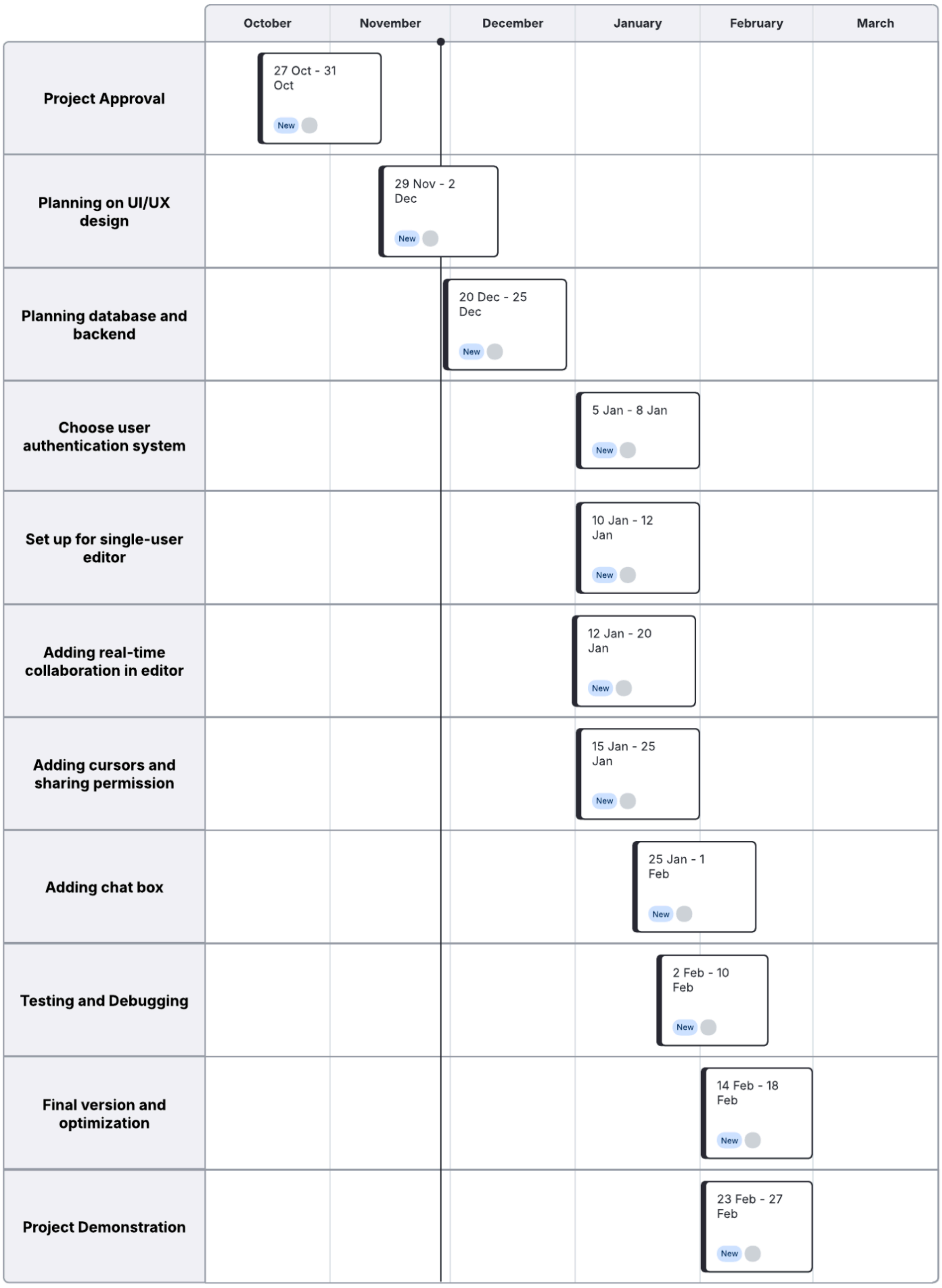
5.6 Summary

- The system context model shows how the collaborative code editor interacts with users, the database and the optional execution sandbox.]
- The data flow model outlines the main processes and how data moves between clients, server processes, and data stores
- The logical data structure defines the core entities and their relationships, guiding both database schema design and runtime data structures.

This high level design forms the basis for more detailed class diagrams, interface specifications, and protocol definitions in subsequent design documentation.

6. Preliminary Schedule

Collaborative Code Editor



	October	November	December	January	February	March
Project Approval	27 Oct - 31 Oct New					
Planning on UI/UX design		29 Nov - 2 Dec New				
Planning database and backend			20 Dec - 25 Dec New			
Choose user authentication system				5 Jan - 8 Jan New		
Set up for single-user editor				10 Jan - 12 Jan New		
Adding real-time collaboration in editor				12 Jan - 20 Jan New		
Adding cursors and sharing permission				15 Jan - 25 Jan New		
Adding chat box				25 Jan - 1 Feb New		
Testing and Debugging				2 Feb - 10 Feb New		
Final version and optimization					14 Feb - 18 Feb New	
Project Demonstration					23 Feb - 27 Feb New	

7. Appendices

7.1 Appendix A

API - Application Programming Interface

CCE - Collaborative Code Editor

CRUD - Create, Read, Update, Delete

DB - Database

DFD - Data Flow Diagram

FR - Functional Requirement

GUI - Graphical User Interface

HTTP / HTTPS - HyperText Transfer Protocol / HyperText Transfer Protocol Secure

IDE - Integrated Development Environment

JSON - JavaScript Object Notation

LDS - Logical Data Structure

OTP / Token - One-time password or session token used for authentication

OT - Operational Transform (concurrency control technique)

CRDT - Conflict-free Replicated Data Type (alternative concurrency technique)

REST - Representational State Transfer (style of web API)

UI / UX - User Interface / User Experience

WS / WebSocket - Full-duplex real-time communication channel between client and server

7.2 Appendix B

This appendix provides a compact summary of all functional requirements defined in Section 3, including their identifier, title, and criticality.

ID	Title	Criticality
FR-1	User Authentication and Session Management	High
FR-2	Project Management	High
FR-3	Document Management	High
FR-4	Real-Time Collaboration Session Management	High
FR-5	Real-Time Text Synchronization and Conflict Resolution	High
FR-6	Document Persistence and Autosave	High
FR-7	Access Control and Sharing	High
FR-8	User Presence and Cursor/Selection Indication	Medium
FR-9	Syntax Highlighting and Basic Editing Assistance	Medium
FR-10	Error Handling and User Feedback	Medium
FR-11	Basic Code Execution in a Sandbox	Low
FR-12	In-Editor Text Chat or Comments	Low
FR-13	Document Version History and Restore	Low

7.3 Appendix C

Subsystems

DC - Desktop Client Application (UI, editor, collaboration client)

SV - Collaboration and Data Server (API, session management, collaboration engine)

DS - Data Storage Layer (database and repositories)

EX - Code Execution Sandbox (optional external service)

FR ID	Main Subsystems Involved
FR-1	DC, SV, DS
FR-2	DC, SV, DS
FR-3	DC, SV, DS
FR-4	DC, SV
FR-5	DC, SV
FR-6	SV, DS
FR-7	DC, SV, DS
FR-8	DC, SV
FR-9	DC
FR-10	DC, SV
FR-11	DC, SV, EX
FR-12	DC, SV
FR-13	DC, SV, DS

7.4 Appendix D

This appendix shows example (illustrative) data structures that may be exchanged between client and server. Final formats may change during implementation but should follow the same general shape.

7.4.1 Example Login Request

```json
{
  "email": "student@example.edu",
  "password": "SecretPassword123"
}
```

7.4.2 Example Login Response (Success)

```json
{
  "user_id": 42,
  "name": "Student A",
  "token": "eyJhbGciOi...",
  "expires_at": "2026-03-01T12:00:00Z"
}
```

7.4.3 Example Project Representation

```
{
  "project_id": 10,
  "name": "CA Assignment 1",
  "owner_id": 42,
  "collaborators": [
    { "user_id": 42, "role": "owner" },
    { "user_id": 51, "role": "editor" }
  ],
  "created_at": "2026-01-10T09:00:00Z",
  "updated_at": "2026-01-15T14:23:00Z"
}
```

7.4.4 Example Document Representation

```
{
  "document_id": 101,
  "project_id": 10,
  "filename": "main.py",
  "language": "python",
  "content": "print('Hello, world!')\n",
  "updated_at": "2026-01-15T14:23:00Z"
}
```

7.4.5 Example Collaborative Operation

```
{
  "session_id": "sess-abc123",
  "document_id": 101,
  "user_id": 42,
  "base_version": 7,
  "type": "insert",
  "position": 21,
  "content": "!",
  "timestamp": "2026-01-15T14:23:10Z"
}
```