StudyBuddy Full-Stack System Overview and Architecture

CSL7090 Software and Data Engineering

| V Sujay (B22CS063) |Aiswarya K (B22CS028) |

26 SEPTEMBER 2025

Code Files

Contents

1 Executive Summary

StudyBuddy is a web-based platform designed to connect university students with compatible study partners. Its primary goal is to facilitate collaborative study by matching students based on shared courses and schedules. The system provides end-to-end account management (signup, login, profile editing), private and group chat functionality, and session scheduling, all within a secure, responsive single-page application.
The front-end is built as a React/TypeScript SPA served by an Express.js backend. The backend exposes RESTful APIs and real-time endpoints (using Socket.IO), with MongoDB for data persistence. Key implemented features include:

- **Account Management:** Users can create an account, authenticate (login/logout), edit their profile, and upload an avatar. Session-based authentication (using secure cookies) manages user sessions.
- **Matching & Courses:** Students are scoped to their university and can add courses to their profile. The system matches students by common courses and availability, showing potential study buddies and overlapping courses.
- **Real-Time Chat & Notifications:** Users can communicate in one-to-one or group chatrooms. Messages are saved to history, and real-time notifications alert users to new messages, users joining/leaving chats, and scheduling updates.
- **Scheduling & Availability:** Users set weekly availability and can propose or join study sessions. Session details (title, time, location, description) are coordinated through the app.
- **Responsive UI & UX:** The interface is built with Bootstrap and a custom theme to ensure usability across desktop, tablet, and mobile devices.
- **Security & Observability:** The application uses modern security practices (bcrypt password hashing, HTTP-only cookies, helmet headers, optional rate limiting) and provides monitoring endpoints (healthz, /readyz, /status) plus Prometheus metrics and structured logs.

This report gives an overview of StudyBuddy, with all the main features, requirements, and design details. It contains architecture diagrams, a project timeline, and notes on risks and mitigation. It also points out the design patterns and quality measures used while building the system.

2 Objectives

The StudyBuddy system is a web application that helps students connect for collaborative studying. Its frontend is built as a React/TypeScript single-page app, while the backend uses Node.js and Express. The frontend talks to the backend through REST APIs and real-time WebSocket events powered by Socket.IO. Data is stored in MongoDB, and user sessions are handled securely using cookies and an express-session store.

- **University Scoping:** Restrict connections so students only match with others at the same institution.
- **Account Management:** Allow users to sign up, log in, and manage their profiles and avatars
- **Course Enrollment:** Enable users to add and remove courses in their profile and see course overlaps with other students.
- **Matching:** Provide functionality to search for and list compatible study partners based on shared courses and availability.
- **Communication:** Support one-to-one and group chat with message history and real-time updates.
- **Scheduling:** Allow students to set weekly availability and propose/accept study session invitations (with time and location).
- **Usability:** Offer a responsive user interface that works on desktop, tablet, and mobile layouts.
- **Security & Observability:** Implement industry-standard security practices (password hashing, secure cookies, helmet headers) and provide monitoring (health checks, metrics, structured logging).

3 Functional Requirements

FR1
Title: Account Management
Priority: High
Description: Users can sign up, log in, and log out; manage profile details and avatar.
Acceptance Criteria: Signup creates a new user; login creates a session; logout ends the session; profile changes (including avatar upload) are persisted.
Implementing Files: backend/routes/users.js, backend/user.model.js, backend/multer-config.js; frontend/src/Components/SignupPage.tsx, LoginPage.tsx, EditProfile.tsx.

FR2
Title: University Scoping
Priority: High
Description: Enforce that users only match with others from the same university.
Acceptance Criteria: Matching and search results return only users who share the same university.
Implementing Files: backend/user.model.js (stores university); matching/search endpoints filter by university.

FR3
Title: Course Management
Priority: High
Description: Users can add or remove courses from their profile and view overlaps.
Acceptance Criteria: Users can add courses via the UI; courses are stored in the profile; overlapping courses with other users are displayed.
Implementing Files: backend/routes/scheduling.js (courses endpoints), backend/scheduling.model.js; frontend/src/Components/EditProfile.tsx, ShowBuddies.tsx.

FR4
Title: Matching
Priority: High
Description: Find and list compatible study buddies based on shared courses and availability.
Acceptance Criteria: Search returns users enrolled in common courses and matching availability; results provided in JSON.
Implementing Files: backend/routes/scheduling.js (/find-partners endpoint), backend/routes/match.js; frontend/src/Components/Matching/MatchUsersList.tsx.

FR5
Title: Real-Time Chat
Priority: High
Description: One-to-one and group chat functionality with message history.
Acceptance Criteria: Users can create/join chatrooms and send/receive messages in real time; chat history is saved.
Implementing Files: backend/routes/chat.js, backend/sockets/chatSocket.js, backend/chatroom.model.js, backend/message.model.js; frontend/src/Components/ChatList/ChatList.tsx, Chatroom.tsx.

FR6
Title: Notifications
Priority: Medium
Description: Real-time notifications for new messages, user joins/leaves, and session updates.
Acceptance Criteria: In-chat notifications appear when others send messages or join; schedule updates trigger notifications.
Implementing Files: backend/sockets/chatSocket.js (emits notification events); frontend/src/Components/Notifications
Notifications.tsx.

FR7
Title: Scheduling
Priority: High
Description: Propose, accept, and manage study sessions; set weekly availability.
Acceptance Criteria: Users can set availability slots; create a session invitation; invited users see pending sessions.
Implementing Files: backend/routes/scheduling.js (/availability, /sessions endpoints), backend/scheduling.model.js; frontend/src/Components/Scheduling/Scheduling.tsx.

FR8
Title: Search/Filter
Priority: Medium
Description: Discover buddies by filtering on course and time availability.
Acceptance Criteria: The matching interface allows filtering by specific course and time; results update accordingly.
Implementing Files: backend/routes/scheduling.js (/find-partners with query parameters); frontend/src/Components/

FR9
Title: Responsive UI
Priority: Medium
Description: Application layout adapts to desktop, tablet, and mobile screen sizes.
Acceptance Criteria: The UI renders properly across different screen widths (as verified through responsive design testing).
Implementing Files: frontend/src/styles/theme.css; Bootstrap classes used in layout components.

The functional requirements listed above define the core capabilities and behaviors expected from the StudyBuddy system. Each requirement ensures that the application meets both user needs and technical standards, covering essential aspects such as account management, university-based scoping, course tracking, matching algorithms, real-time communication, scheduling, notifications, and responsive design. By specifying clear acceptance criteria and the implementing files for each functionality, this section provides traceability from requirements to code implementation. Together, these functional requirements serve as a foundation for designing, developing, and testing the system, ensuring a reliable and user-friendly experience for students collaborating in study sessions.

4 Non-Functional Requirements

ID / Title	Description / Implemented In (Key Modules)
NFR1: Security & Privacy	Use secure, tested libraries and best practices for authentication and data protection. Implemented in: utils/password.js (bcrypt); middlewares/validateAuth.js; backend/routes/users.js (signup/login); app.js (session config, helmet, CORS, rate limiting).
NFR2: Observability & Logging	Provide health checks, metrics, and structured logging for monitoring and debugging. Implemented in: backend/routes/health.js (/healthz, /readyz, /status endpoints); app.js (Prometheus prom-client for /metrics); logging/logger.js (Winston with request IDs).
NFR3: Maintainability (to some extent)	Organized code into layers and modules for ease of extension and debugging. Implemented in: Layered folder structure (routes/, models/, middlewares/); centralized configuration via .env and process.env; modular React components.

The non-functional requirements focus on the quality attributes and operational aspects of the StudyBuddy system. Security and privacy are enforced through tested libraries, secure password handling, session hardening, and careful input validation. Observability and logging ensure that the system can be monitored, debugged, and maintained effectively through structured logs, health checks, and Prometheus metrics. Maintainability and modularity are addressed through a layered architecture, clear separation of concerns, and reusable components in both the frontend and backend. Together, these non-functional requirements complement the functional requirements by ensuring that the system is secure, reliable, scalable, and easy to operate and extend.

5 Architecture

The StudyBuddy system is built on a Client–Server Single-Page Application (SPA) model. The frontend is a React application that runs in the user's browser and communicates with the backend, which is implemented using Node.js and Express. The system supports both HTTP REST API requests and real-time communication through Socket.IO to enable interactive features. The backend follows a layered architecture: routers and controllers manage incoming requests, business logic is handled in services and models, and Mongoose models (following the Active Record pattern) handle database operations. Common middleware such as helmet, CORS, session management, rate limiting, logging, and metrics are applied across all requests.

User authentication is session-based. When a user logs in, an express-session cookie is set with httpOnly, sameSite=lax, and secure flags and is stored in MongoDB using connect-mongo. Passwords are stored securely with bcrypt, and legacy plaintext passwords are upgraded automatically upon login. Additional security measures include HTTP header protection through helmet and optional login rate limiting.

The system's observability features include /healthz, /readyz, and /status endpoints that provide service status, as well as a /metrics endpoint for Prometheus monitoring. Logging is handled with Winston in a structured format, with request IDs included for easier tracing. Real-time functionalities, such as chat and study session scheduling, use dedicated Socket.IO namespaces (/chat and /meet-up) to broadcast events and notifications.

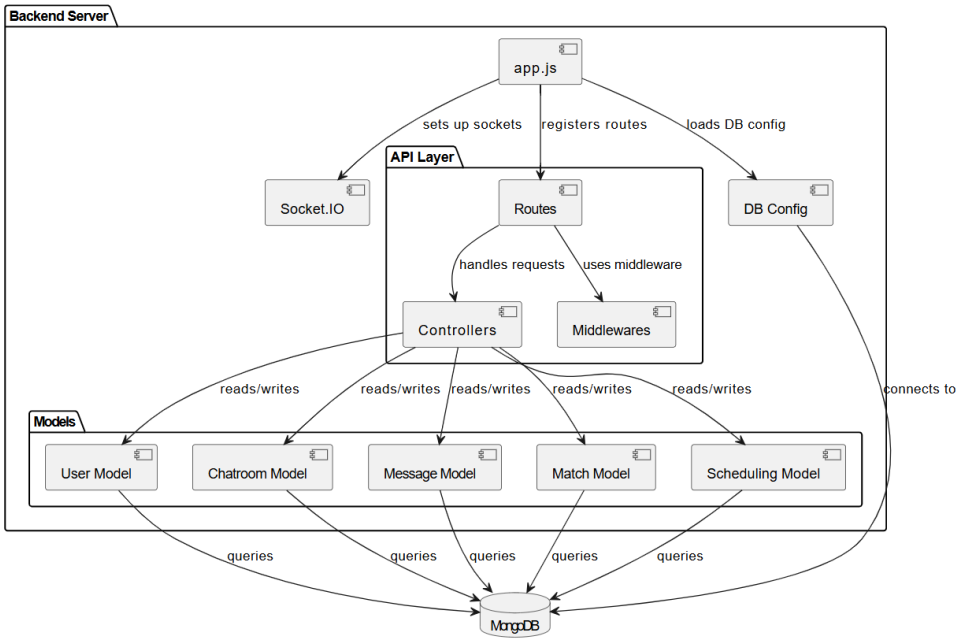

Figure 1: This diagram explains backend internals. app.js initializes the server, registers routes, configures middleware, and sets up Socket.IO for real-time messaging. Routes forward requests to controllers, which apply business logic and coordinate with models for users, chatrooms, messages, matches, and scheduling. Each model connects to MongoDB, ensuring consistent data management across the application.

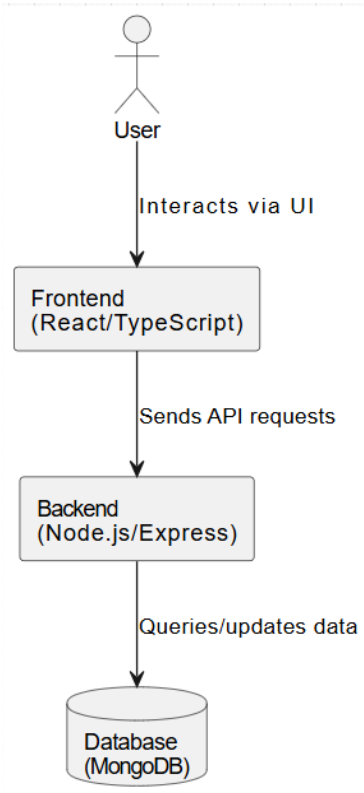

Figure 2: This diagram presents Studybuddy's system at a glance. The user interacts through the React/TypeScript frontend, which provides the interface and sends API calls or WebSocket messages to the Node.js/Express backend. The backend processes logic, manages sessions, and communicates with MongoDB for persistent storage of users, messages, matches, and schedules.

Figure 3: Explains request processing flow in Express. Incoming requests pass through middleware for authentication, validation, and logging before reaching controllers. Ensures security, modularity, and systematic handling of user requests.

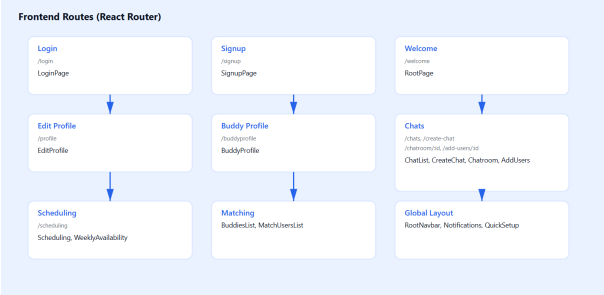

Figure 4: Shows React frontend routes mapping. Defines navigation paths for different features (dashboard, authentication, chat, scheduling, matching), providing a structured single-page application experience for Studybuddy users.

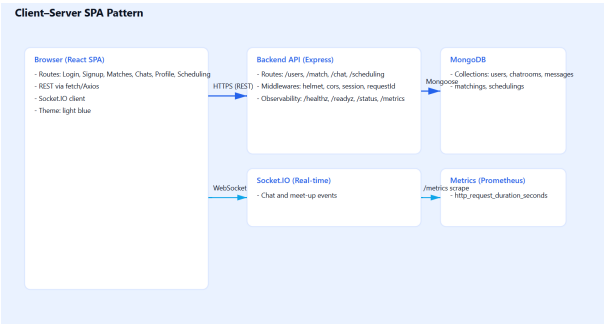

Figure 5: Demonstrates client-server interaction in a single-page app. React frontend requests data from Express backend, which queries MongoDB. Backend responds with JSON, enabling dynamic UI updates without page reloads.

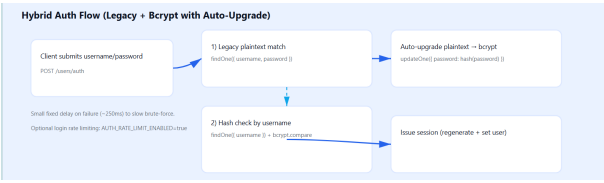

Figure 6: Outlines user authentication process: signup/login requests pass through middleware, controllers validate credentials, backend queries database, and successful responses return tokens, granting secure access to frontend features.

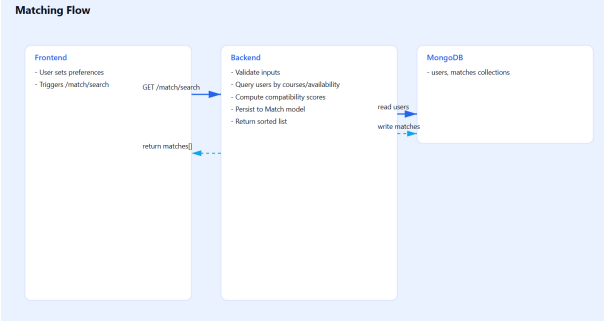

Figure 7: Shows matching logic. User preferences are compared, backend algorithms identify best matches, and results are returned to frontend. Facilitates intelligent peer connections within the Study-buddy platform.

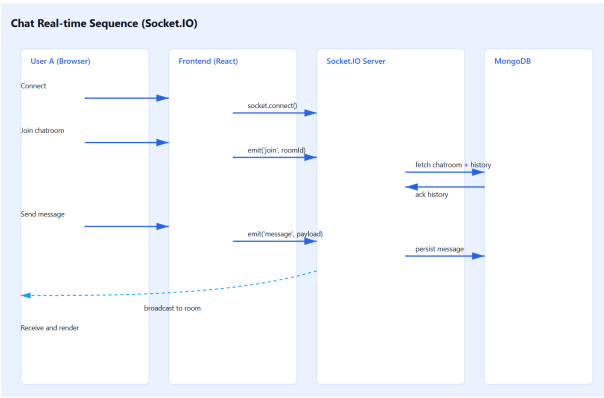

Figure 8: Depicts real-time chat sequence. Messages sent by a user travel through Socket.IO backend, are stored in MongoDB, and broadcast instantly to other connected clients.

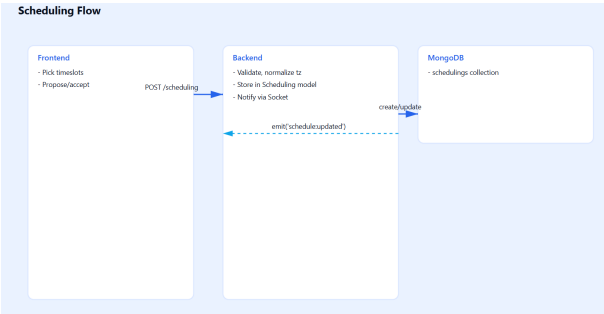

Figure 9: Illustrates scheduling process. User selects time, backend validates conflicts, stores event in database, and updates frontend. Ensures seamless booking and management of study sessions.

6 Design Patterns and Quality Attributes

- The backend is organized into layers—routes (presentation), controllers (business logic), and models (data access)—which separates concerns and simplifies debugging and maintenance.
- While the backend uses Express with layered logic, the overall system follows an MVC-like pattern where React (View) interacts with controllers (via APIs) and models (via Mongoose).
- Implemented using Socket.IO, this ensures clients are notified in real time when events occur (e.g., new chat messages, study session updates).
- MongoDB connections are created as singletons to avoid redundant connections, and middleware follows reusable factory-like design to enforce validation and security.
- The system emphasizes security (bcrypt password hashing, cookies with secure flags), maintainability (modular structure), usability (responsive UI), and observability (metrics and logs).

7 Conclusion

The development of StudyBuddy demonstrates how a practical web-based platform can support students in finding and collaborating with study partners. The project involved applying full-stack development concepts, combining a React frontend, an Express/Socket.IO backend, and MongoDB for persistent storage. Key features such as account management, matching, scheduling, and real-time chat were successfully implemented, supported by strong security and observability measures. While the current system addresses its core objectives, future enhancements such as AI-based recommendations, calendar integration, and mobile app support could significantly expand its capabilities. Overall, StudyBuddy reflects both the technical implementation and design considerations required to create a secure, scalable, and user-friendly system for academic collaboration.