CS7CS3 Advanced Software Engineering Group Project

VR Library - Functional & Technical Architectures:

Daoqi Xiong	Jiayun Zhang	Joshua O'Donnell	Keshav Tripathi	Priyansh Nayak
ID: 25335365	ID: 25334655	ID: 25353241	ID: 25347551	ID: 25350660
Rundong Chen	Sarvesh Jaiswal	Viraj Bhor	Xu Gong	Yokesh Kathiravan
ID: 25336554	ID: 24361554	ID: 25358463	ID: 25334562	ID: 25341409

Group 3

Submission Date: November 7th, 2025

1 Functional Architecture

1.1 Diagram

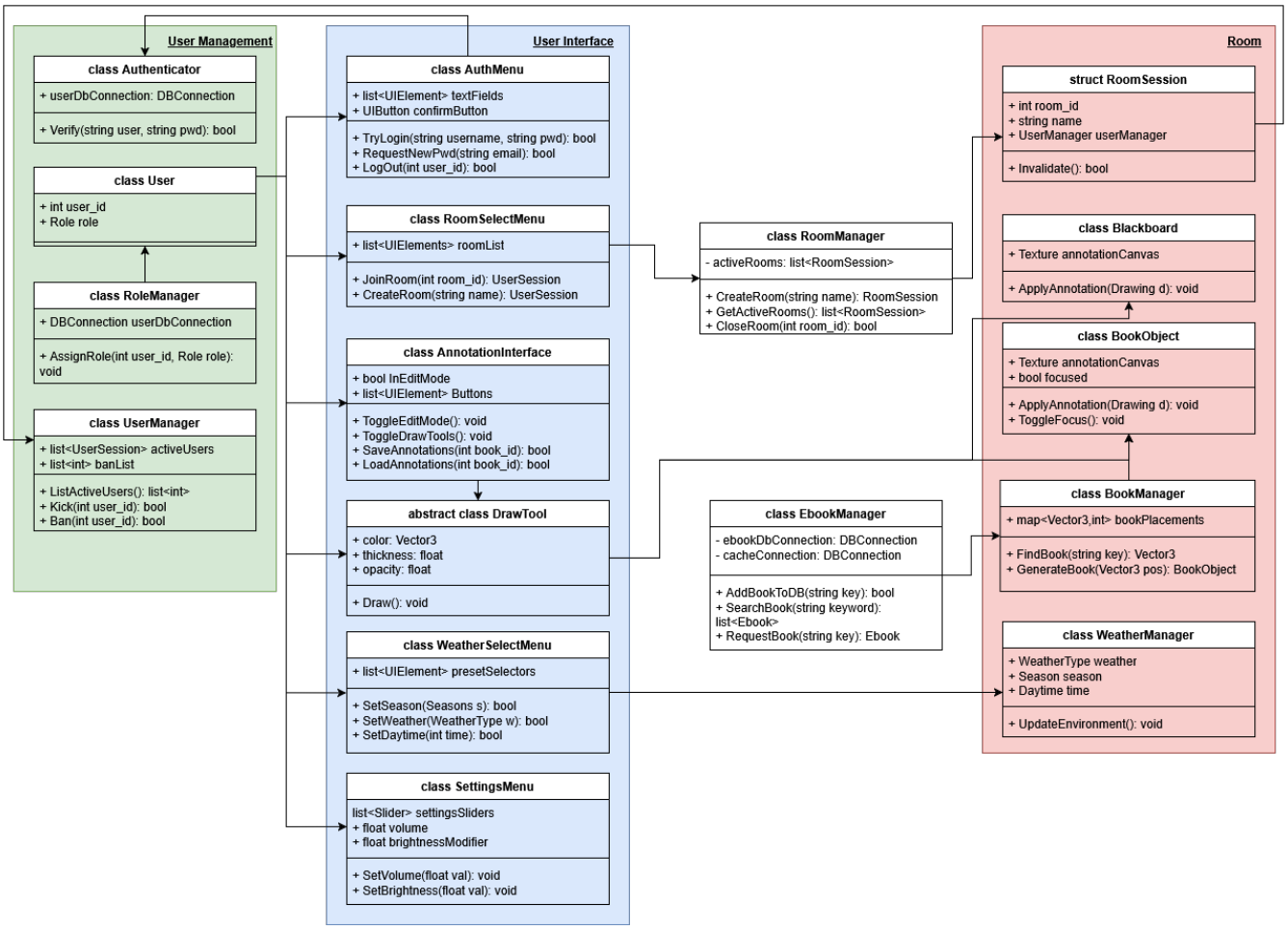

Figure 1: The functional architecture conceived for this project.

1.2 Component Descriptions

The system is composed of multiple interconnected components that together manage user access, room operations, environmental settings, library content, and auditing functionalities. Each major subsystem and its internal components are described below.

1.2.1 User Management

Authenticator

- Handles login and logout processes (Use Case: Sign In, Logout).
- Issues and validates access/refresh tokens (JWT-based).
- Connects with Role Manager and Session Manager for user authentication.

User

- Represents a regular user who joins rooms and interacts with books or other users.
- Can perform actions like ordering, grabbing, or annotating books.
- Limited to permissions granted by their assigned role.
- Has a role representing the user's access level with in the system.

Role Manager

- Defines, edits, and updates the global role catalog (Admin, Librarian, Member).
- Manages authorization scope and permissions system-wide.
- Maintains audit trail for role modifications.

User Manager

- Tracks active sessions, manages session creation and termination.
- Supports session listing and remote termination (Use Case: Session Listing & Remote Termination).
- Ensures single-device access rules and propagates session revocation.

1.2.2 User Interface

Authentication Menu

- User entry point for login and account management.
- Integrates with the Authenticator for sign-in/sign-out workflows.
- Displays error handling and feedback for invalid credentials.

Settings Menu

- Allows users to adjust environmental and system preferences.
- Connects to Weather Manager and Library environment modules.
- Hosts sub-menus like brightness, volume, and daytime settings.

Annotation Interface

- Provides tools for VR and desktop annotation (Use Cases: VR User Annotates Book, Desktop User Annotates Book).
- Allows highlighting, underlining, and note creation.
- Syncs with Book Manager for saving and reloading annotations.

Room Selection Menu

- Displays available rooms and manages join/create room workflows.
- Integrates with Room Manager and Session Manager for validation.
- Shows active participants and room details.

Weather Selection Menu

- Provides dropdowns and toggles for weather and ambiance customization.
- Communicates with Weather Manager for real-time scene updates.
- Supports both private and public synchronization.

Draw Tool

- Provides direct interaction capabilities for drawing or markup.
- Used for annotation and Blackboard editing functions.
- Integrates with Annotation Interface and Room Session components.

1.2.3 Room

Room Session

- Represents an individual's active session in a room (Use Case: Create Room Session).
- Handles user presence, join/leave events, and synchronization.
- Integrates with Session Manager and Room Manager.

Weather Manager

- Controls weather and ambient changes in real-time (Use Case: Weather Adjustment).
- Updates environment visuals and sound for all users in session.
- Synchronizes host-selected conditions in private sessions.

Book Manager

- Manages all book-related operations (ordering, grabbing, annotating).
- Interfaces with Book Object and Database for CRUD actions.
- Handles annotation saving/loading and synchronization.

Book Object

- Represents a specific book instance in the environment.
- Stores metadata, textures, annotations, and prefab references.
- Used for grabbing, reading, or entering focus mode.

Blackboard

- Shared interactive board within rooms for user collaboration.
- Allows editing, drawing, and presentation (Use Case: Interaction with Blackboard).
- Synchronizes changes between users in real-time.

1.2.4 Library Management

eBook Manager

- Handles digital book storage and retrieval.
- Communicates with Book Manager to provide eBook access.
- Supports searching, ordering, and metadata indexing.

1.2.5 Room Management

Room Manager

- Core controller for room lifecycle: creation, closure, and synchronization.
- Manages participants and permissions through scoped role bindings.
- Works closely with Session Manager, Weather Manager, and Audit Log.

1.3 Component Interactions

Each of the above components interacts through well-defined interfaces and data exchange patterns. For example:

- During user authentication, the **Authentication Menu** triggers the **Authenticator**, which validates credentials and communicates with the **Session Manager** to issue a session token.
- In room activities, the **Room Manager** coordinates with the **Weather Manager**, **Book Manager**, and **Blackboard** to synchronize updates across all participants.
- The **Audit Log** listens to every major event for traceability, ensuring compliance and accountability.

1.3.1 Adding a book to the library.

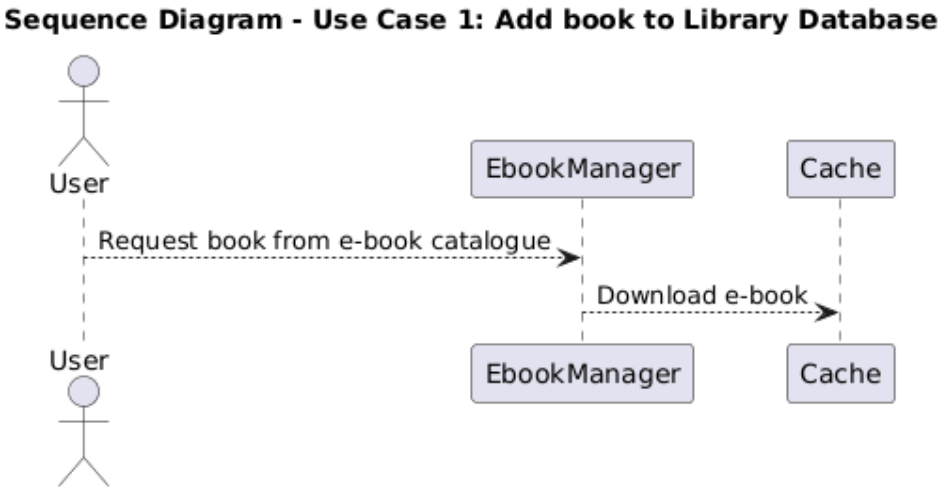

Figure 2: Sequence diagram describing component interactions for book additions.

1.3.2 Annotation Storage

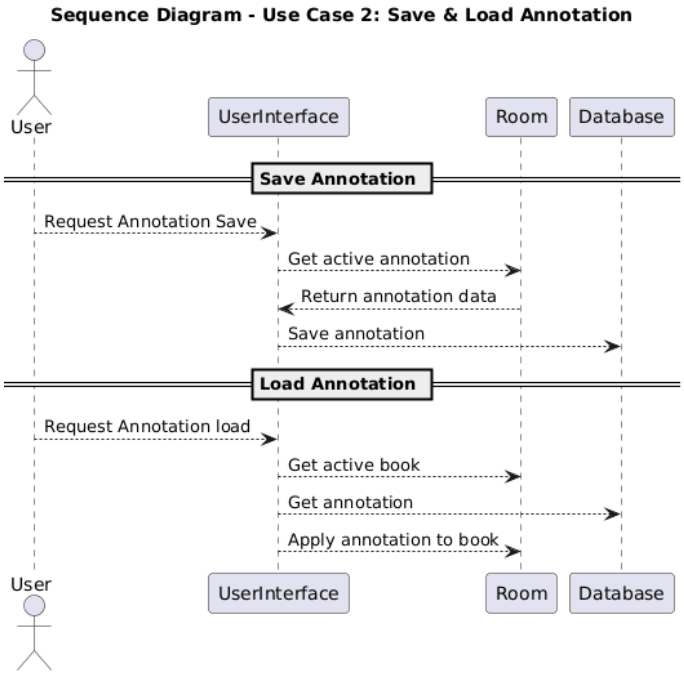

Figure 3: Sequence diagram describing component interactions for Annotation saving & loading.

1.3.3 Library Navigation

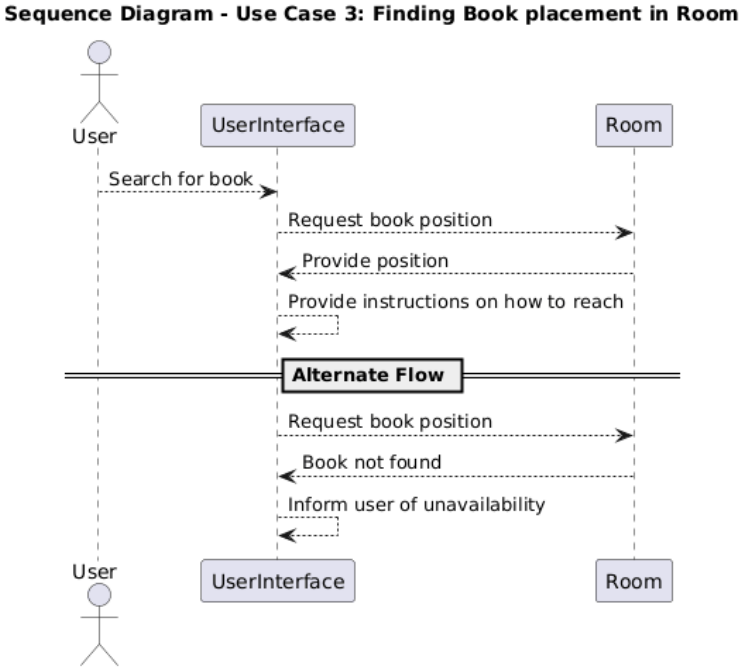

Figure 4: Sequence diagram describing component interactions for finding a book in the Room.

1.3.4 Ordering Book

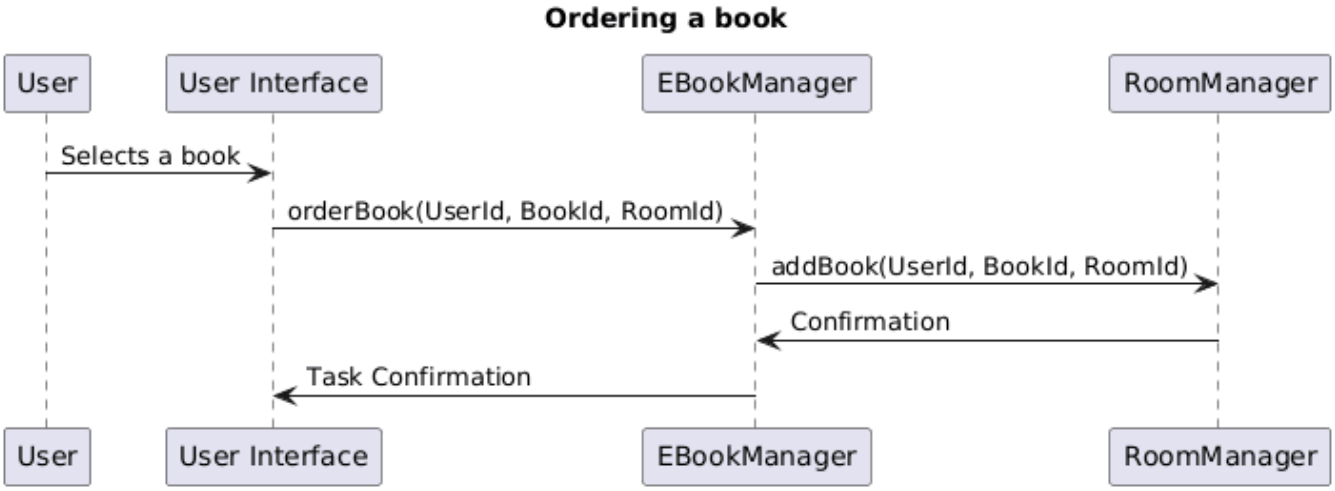

Figure 5: Sequence diagram describing component interactions for finding a book in the Room.

1.3.5 Book & Notice Board annotation

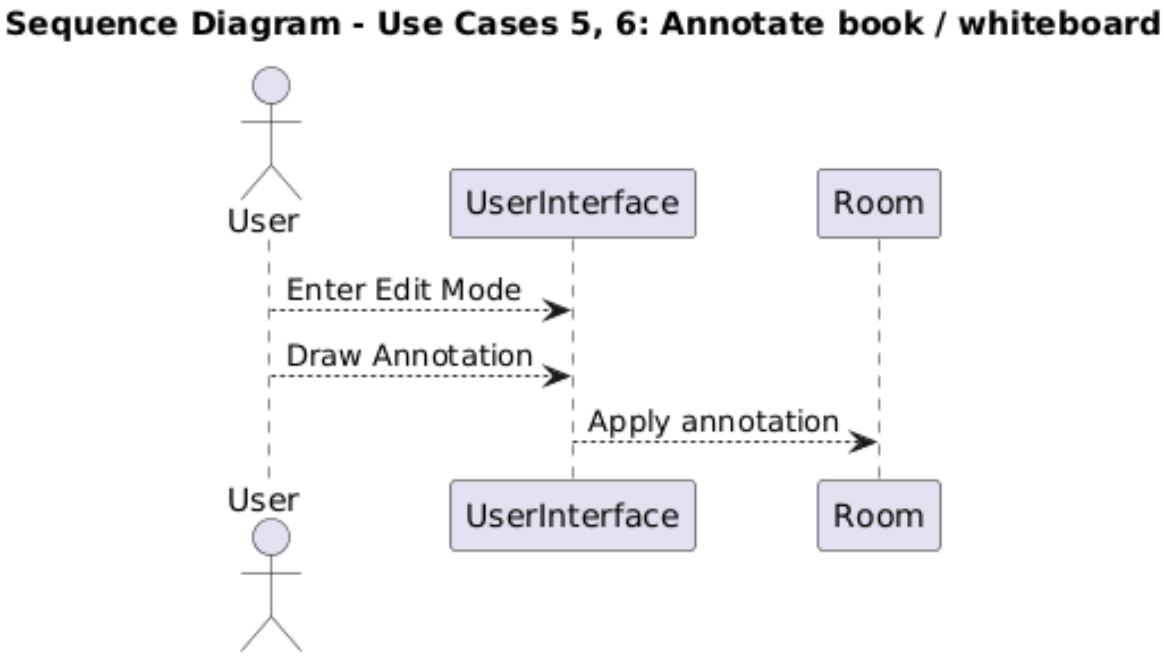

Figure 6: Sequence diagram describing component interactions for annotation of books, and the notice/whiteboard.

1.3.6 User Role Management

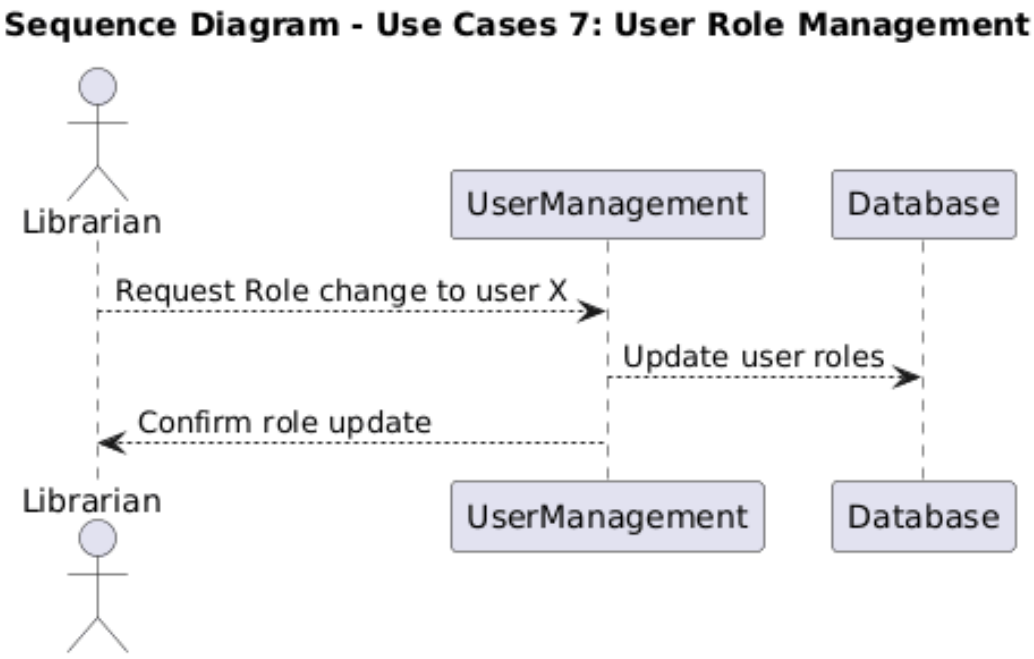

Figure 7: Sequence diagram describing component interactions for the assignment of global roles to users.

1.3.7 Room Creation

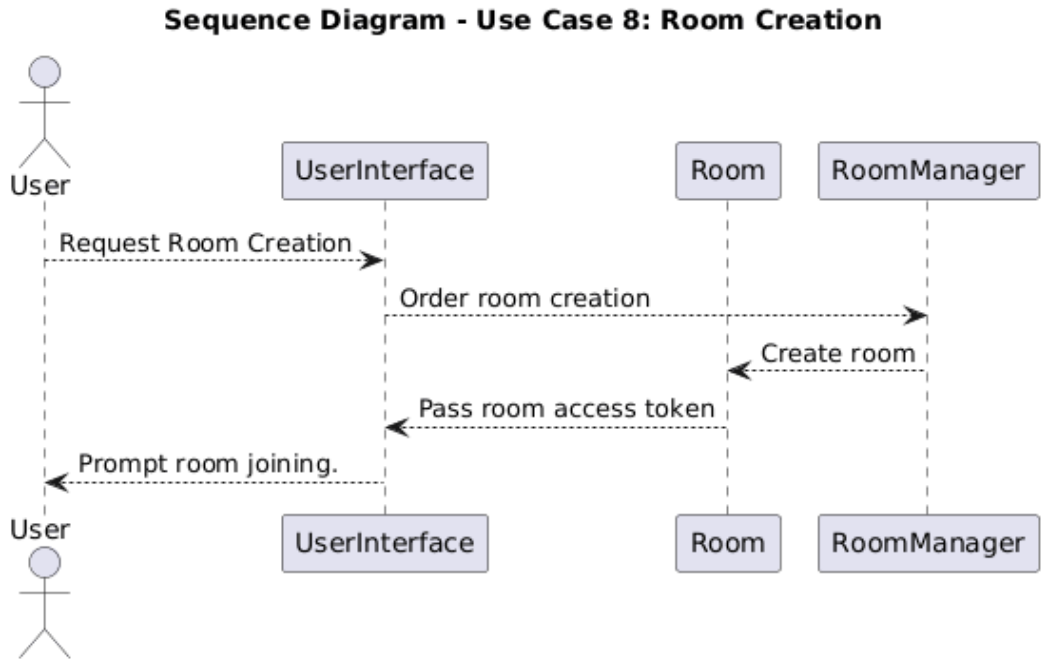

Figure 8: Sequence diagram describing component interactions for the creation of a new Room.

1.3.8 Join existing room

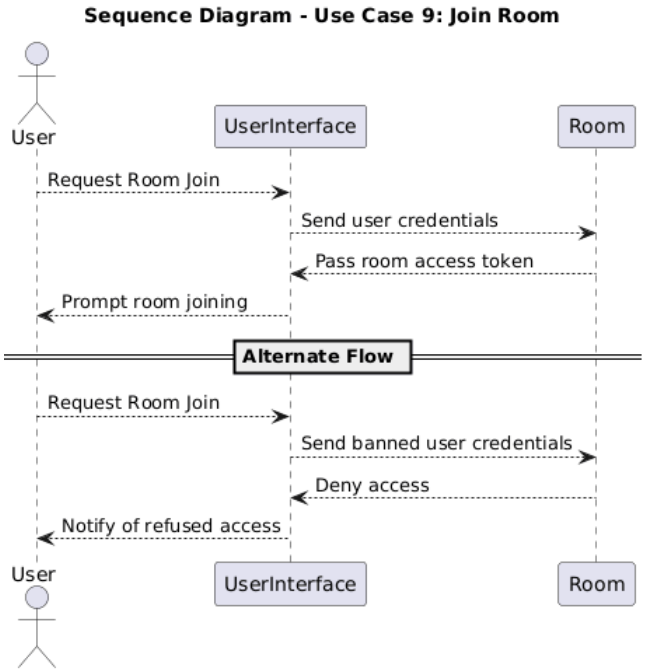

Figure 9: Sequence diagram describing component interactions for joining an existing Room.

1.3.9 Leave current room

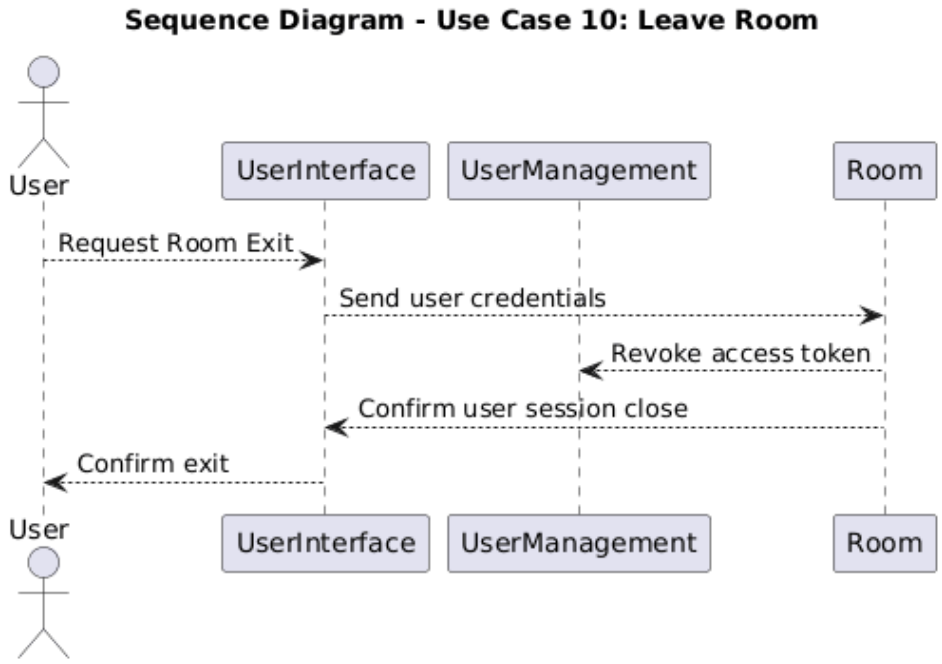

Figure 10: Sequence diagram describing component interactions for leaving the current room.

1.3.10 Kicking/banning users from rooms

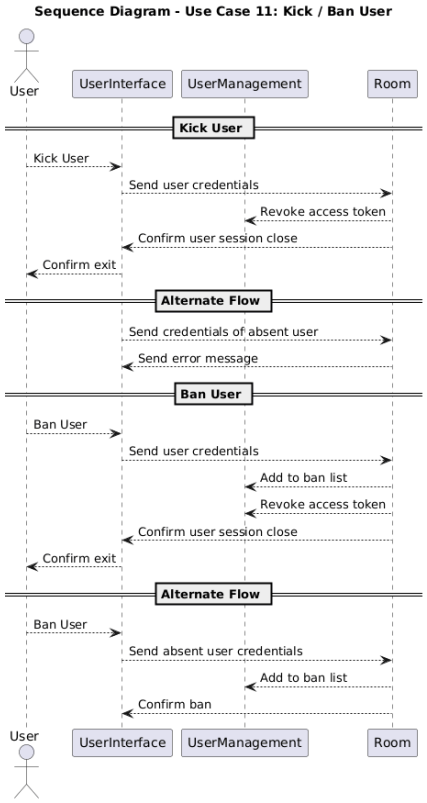

Figure 11: Sequence diagram describing component interactions for banning & kicking users from rooms.

1.3.11 User Logout

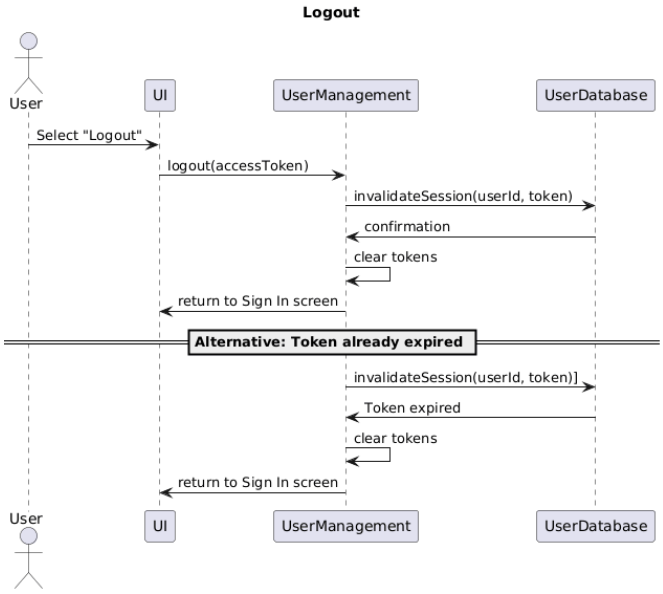

Figure 12: Sequence diagram describing component interactions for user Logout

1.3.12 User Sign In

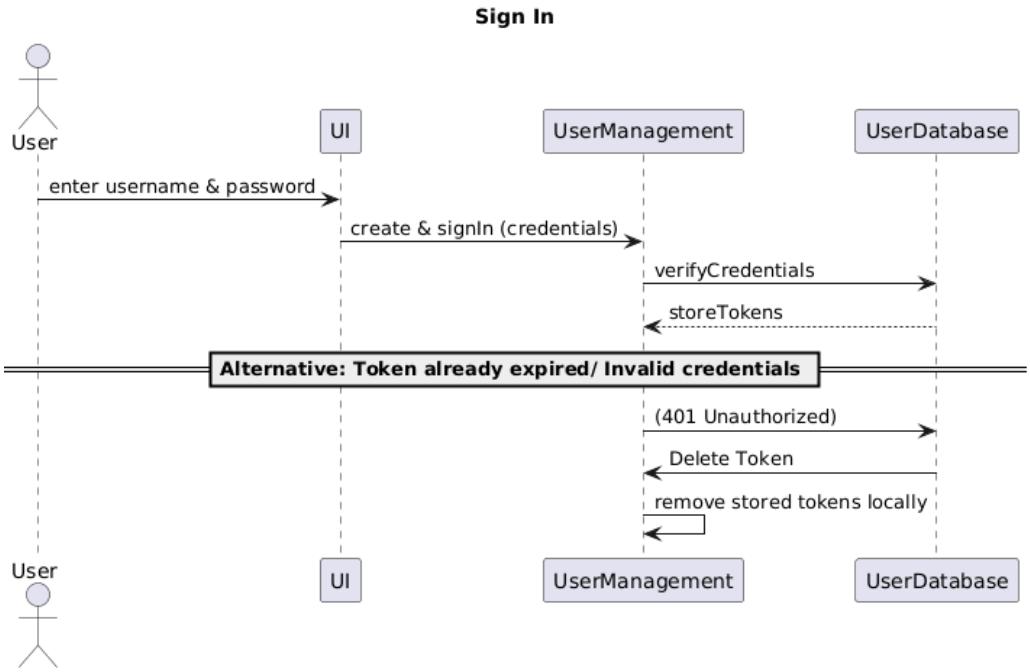

Figure 13: Sequence diagram describing component interactions for User Login

1.3.13 Closing Room

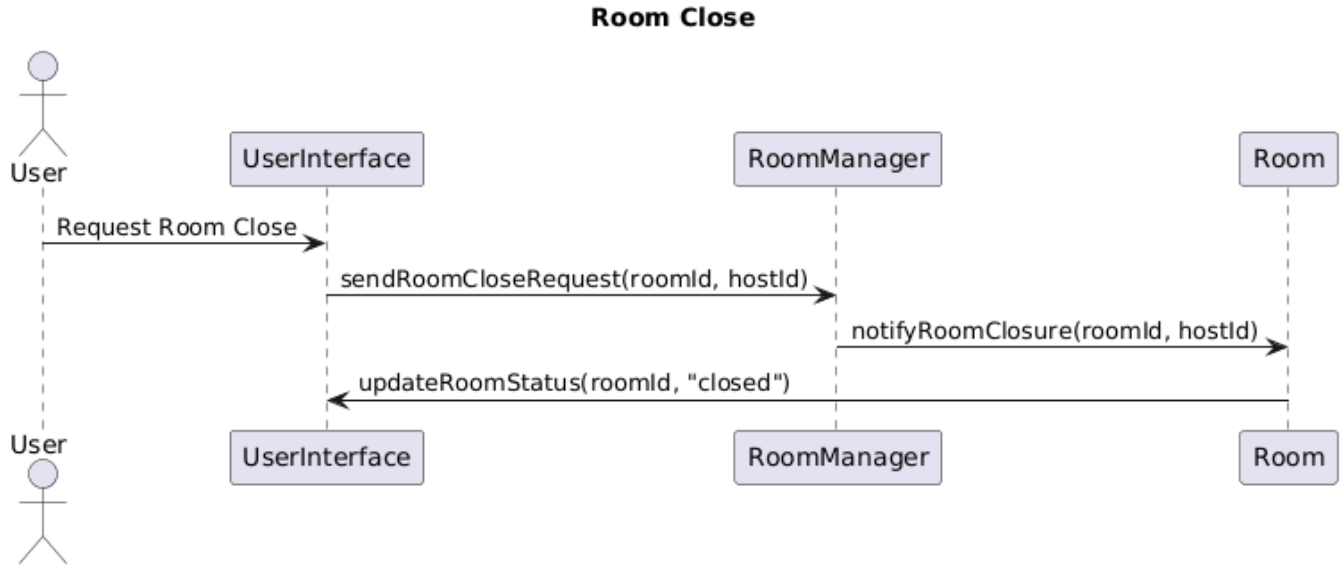

Figure 14: Sequence diagram describing component interactions for User to close the room session

1.3.14 Room session display and termination

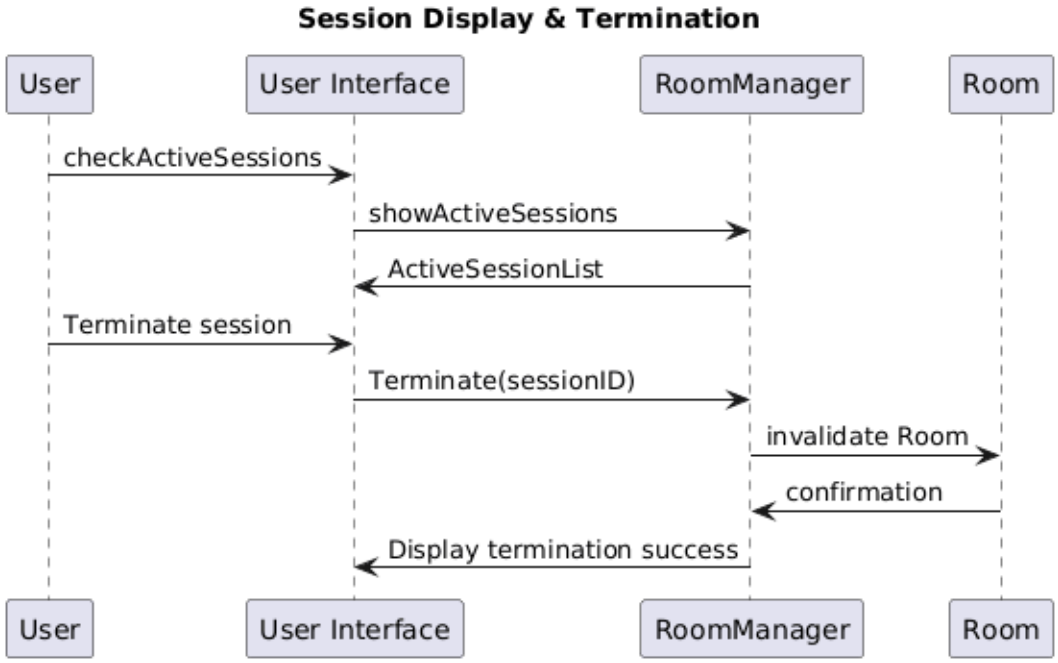

Figure 15: Sequence diagram describing component interactions for session display and termination

1.3.15 Change Weather, Season, Time

Figure 16: Sequence diagram describing component interactions for changing weather, season, and time.

1.3.16 Brightness Adjustment

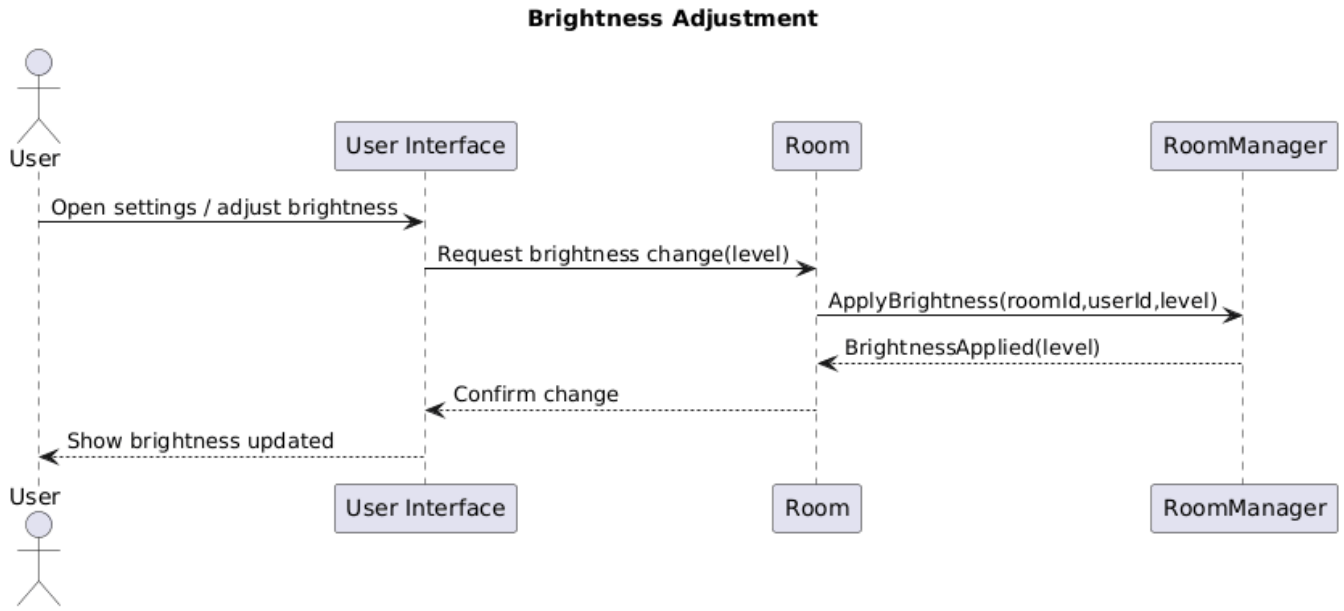

Figure 17: Sequence diagram describing component interactions for

1.3.17 Volume adjustment

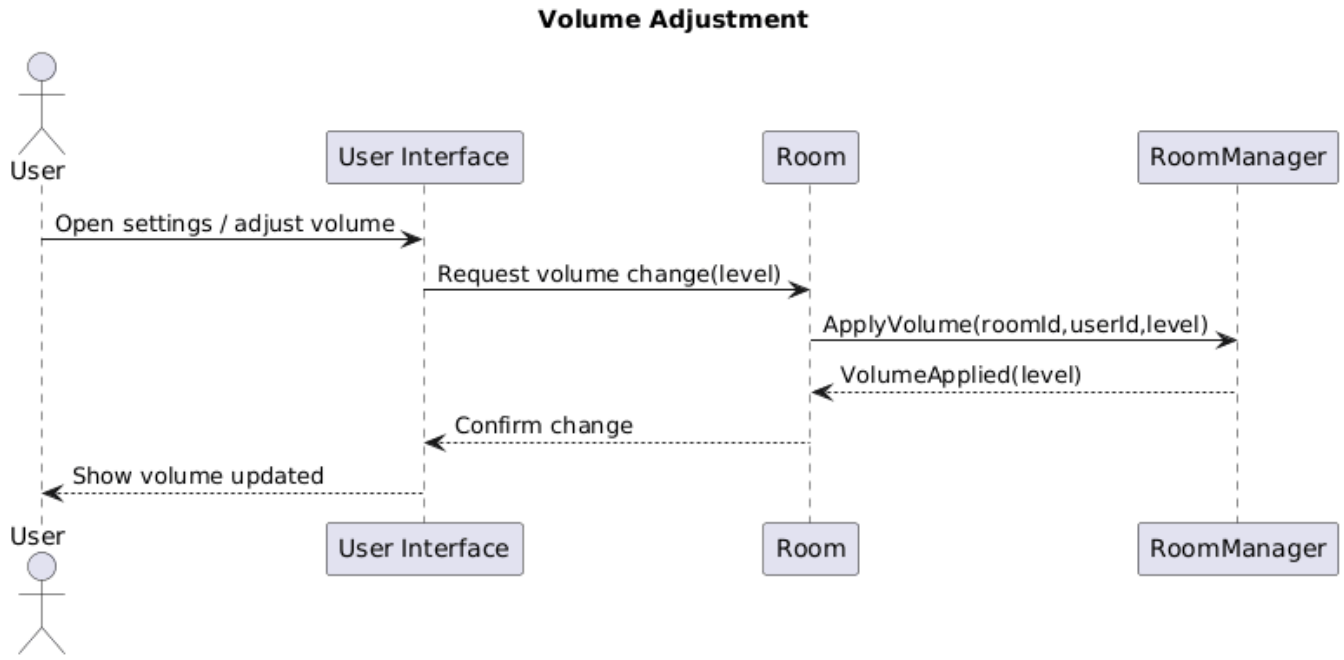

Figure 18: Sequence diagram describing component interactions for volume adjustment

1.3.18 Daytime adjustment

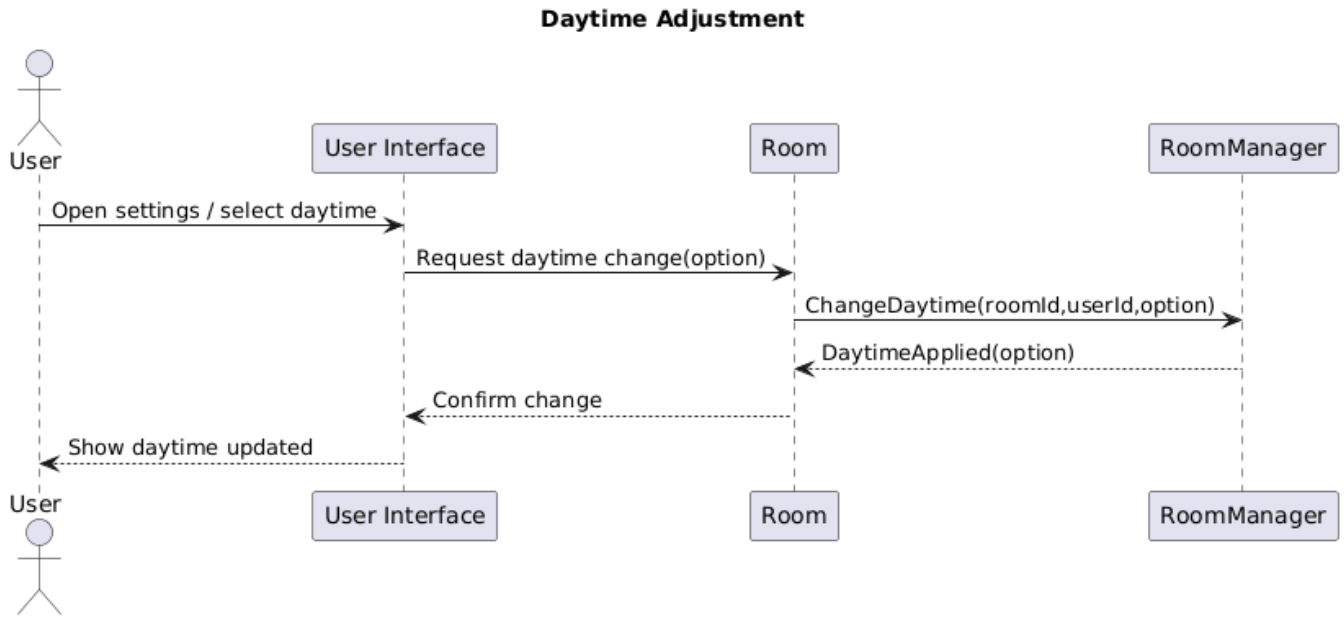

Figure 19: Sequence diagram describing component interactions for daytime adjustment

1.3.19 Entering Focus Mode

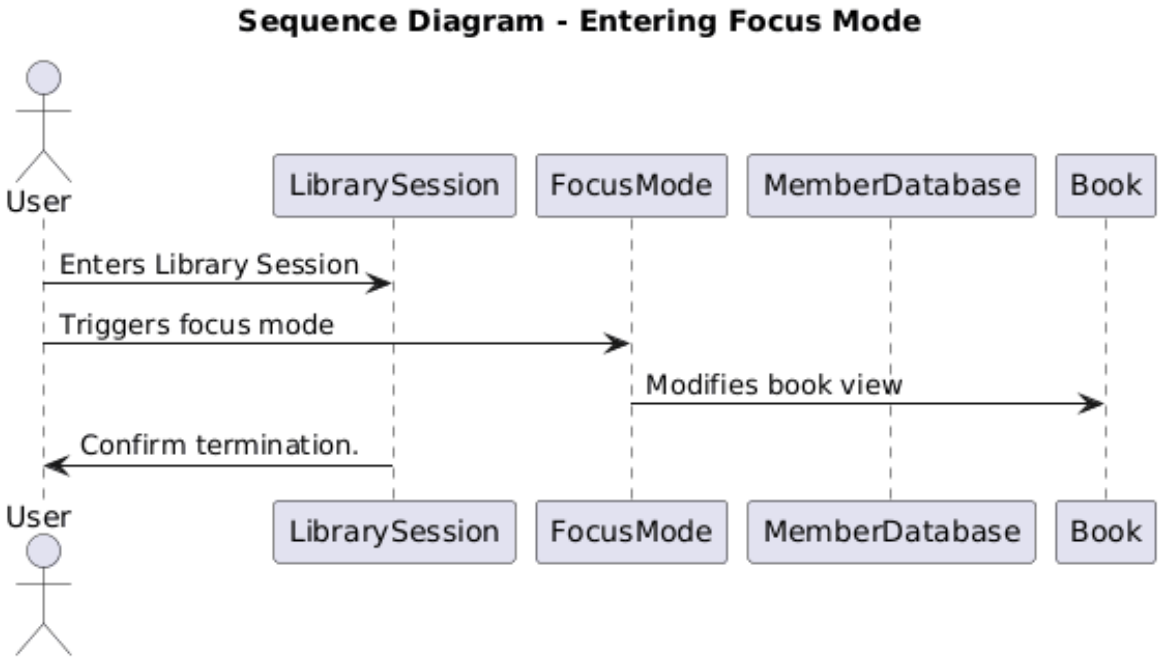

Figure 20: Sequence diagram describing component interactions for entering focus mode

1.3.20 Exiting Focus Mode

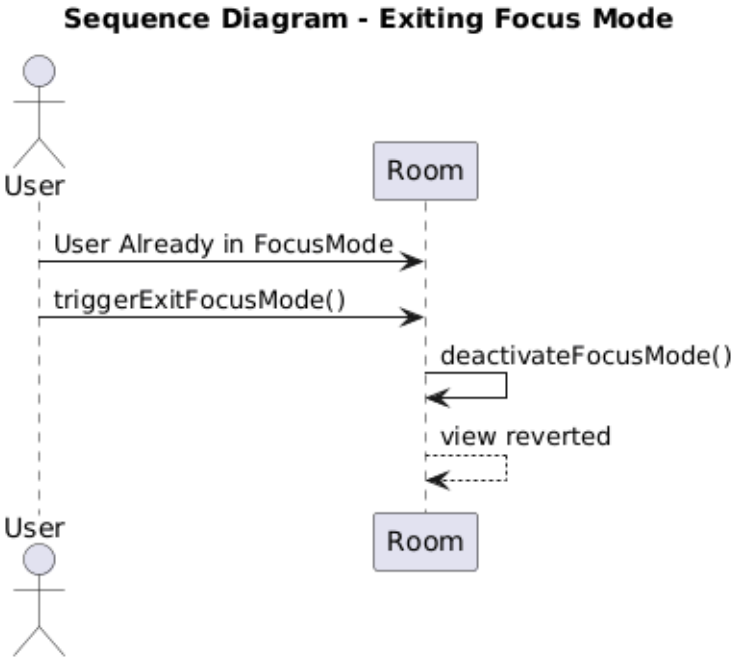

Figure 21: Sequence diagram describing component interactions for exiting focus mode

1.3.21 Physical Movement

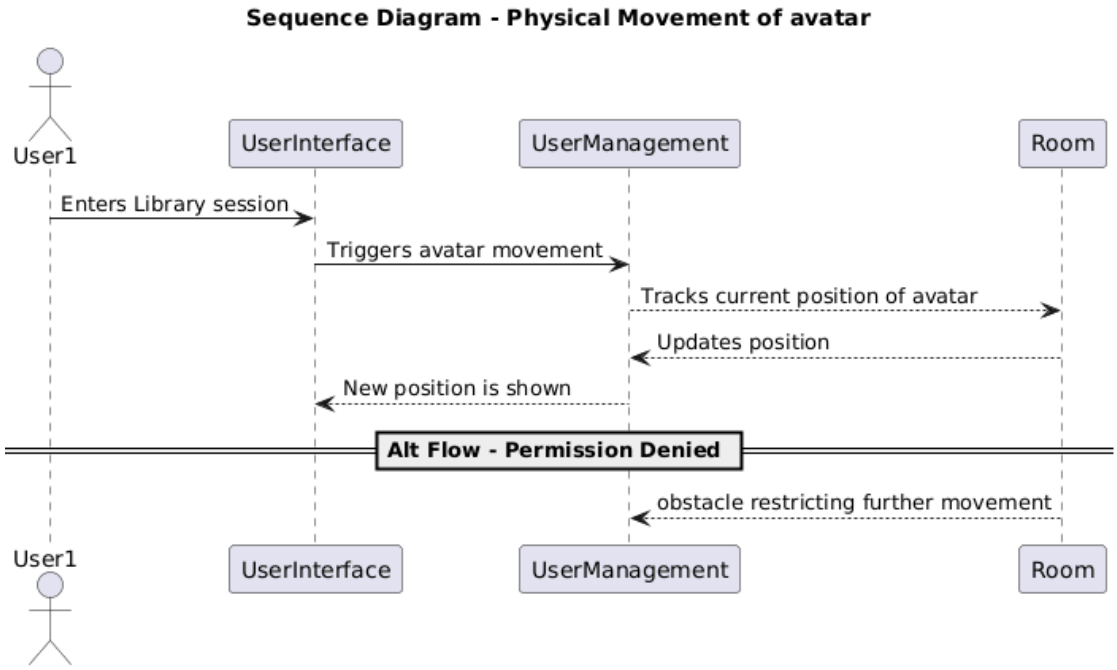

Figure 22: Sequence diagram describing component interactions for physical movement

1.3.22 Shared Book Session

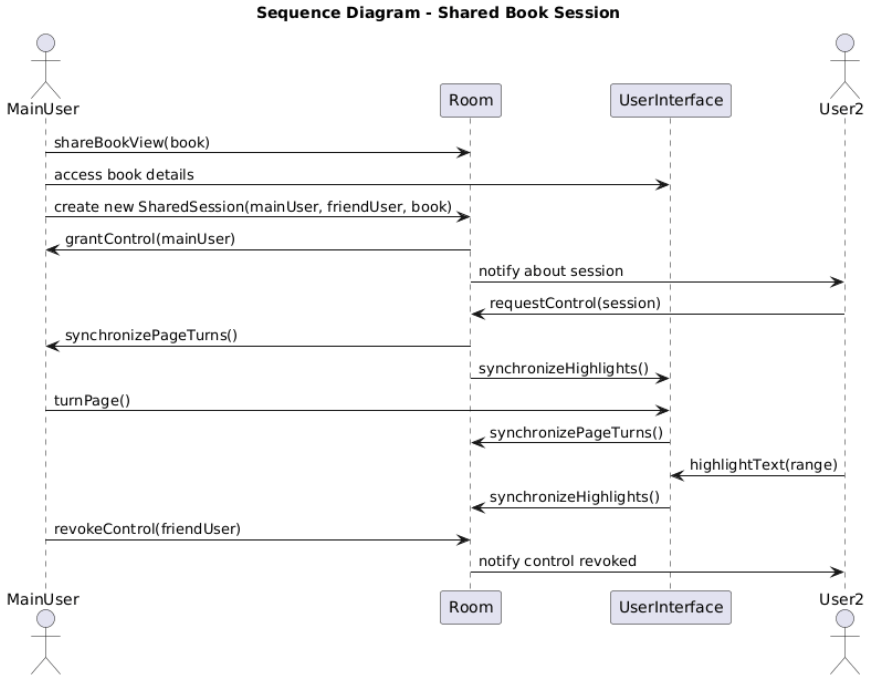

Figure 23: Sequence diagram describing component interactions for sharing a book

1.3.23 Notice Board Interaction

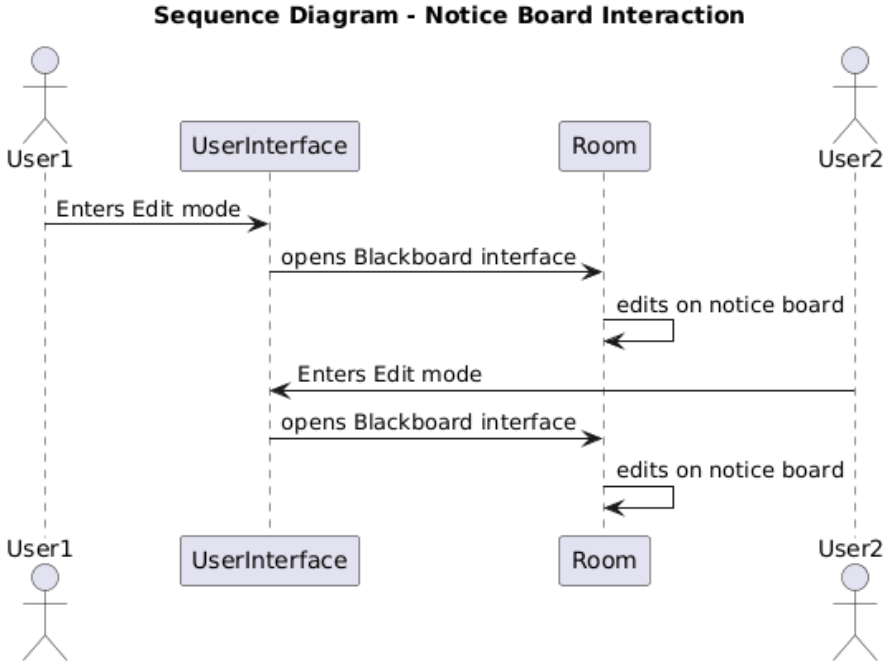

Figure 24: Sequence diagram describing component interactions for Blackboard

1.4 Component APIs

1.4.1 User Management

- *bool success* `Verify(string username, string pwd)`
  Compares user credentials with stored user information. If the credentials match, the verification is considered successful, and fails otherwise, with the status returned as a boolean value.

- *void* `AssignRole(string userID, string role)`
  Assigns a specified global role to the selected user, updating the user's status in the database in the process.

- *list<int>* `ListActiveUsers()`
  Returns a list of user ID's corresponding to each user active in the given room.

- *bool* `Kick(int userID)`
  Kicks an active user from the room, invalidating their session. The success of the action is returned as a boolean value.

- *bool* `Ban(int userID)`
  Bans a user from the room, invalidating any active session, and marking them for denial of access in the future. The success of the action is returned as a boolean value.

1.4.2 User Interface

- *bool* `TryLogin(string username, string pwd)`
  Requests user credential verification with the Authenticator. Returns authentication success status as a boolean value.

- *bool* `RequestNewPwd(string email)`
  Sends user an e-mail with a way to reset their password. Returns the action's success as a boolean value.

- *bool* `LogOut(int userID)`
  Requests termination of a user's authentication to the system. Returns log out success status as a boolean value.

- *UserSession* `JoinRoom(int roomID)`
  Requests to join an active room. Returns a user connection to the room (User Session) on success, or a null value otherwise.

- *UserSession* `CreateRoom(string name)`
  Creates a new Room, and connects the room's creator as a room admin. Returns a connection to the room (User Session) on success, or a null value otherwise.

- *void* `ToggleEditMode()`
  Flips the activation status of editing and annotation tools.

- *void* `ToggleDrawTools()`
  Shows or hides the drawing tool selection interface.

- *bool* `SaveAnnotations(int bookID)`
  Saves any annotations made to a book for later import. Returns the action's success as a boolean value.

- *bool* `LoadAnnotations(int bookID, int annotationID)`
  Loads a saved set of annotations to the corresponding book.

- *void* `Draw()`
  Follows the user's movement to create a drawing to apply to any editable object.

- *void* `SetSeason(Season season)`
  Changes the environment in the active room to fit the selected season.

- *void* `SetWeather(WeatherType weatherType)`
  Changes the environment in the active room to fit the selected weather preset.
- *void* `SetDaytime(Daytime time)`
  Changes the environment in the active room to fit the selected daytime
- *void* `SetVolume(float value)`
  Changes the volume output on the user's device to the new value.
- *void* `SetBrightness(float value)`
  Changes the brightness modifier on the user's device to the new value.

1.4.3 Room Manager

- *RoomSession* `CreateRoom(string name)`
  Creates a new Library Server (=Room) with the given name, and returns its head for reference & storage.
- *list<RoomSession>* `GetActiveRooms()`
  Returns a list of all active, public rooms.
- *bool* `CloseRoom(int roomID)`
  Orders a given room to terminate cleanly, returning the operation's success as a boolean value.

1.4.4 E-Book Manager

- *bool* `AddBookToDB(string key)`
  Requests a book from the external E-Book database, to add to the local cache. Returns the operation's success as a boolean value.
- *list<BookData>* `SearchBook(list<string> keywords)`
  Searches through the local cache for a book based on the provided keyword(s). Returns a list of hits.
- *BookData* `RequestBook(int bookID)`
  Gets the data from the book requested in the local cache, or a null value if there isn't a book with the given ID.

1.4.5 Room

- *bool* `Invalidate()`
  Instructs all systems present in the room to begin termination protocols. Returns a bool value once the termination is complete.
- *void* `ApplyAnnotation(Drawing d)`
  Applies a drawing to an editable object.
- *void* `ToggleFocus()`
  Focuses in or out of the selected book.
- *Vector3* `FindBook(int bookID)`
  Maps the exact position at which the selected book can be found in the current room, and returns it as a 3D vector.
- *BookObject* `GenerateBook(Vector3 position)`
  Deduces the selected book from its position, and generates a new instance of it in the user's hand.
- *void* `UpdateEnvironment()`
  Propagates changes to environment settings to any object in the room that is affected.

2 Technical Architecture

2.1 Diagram

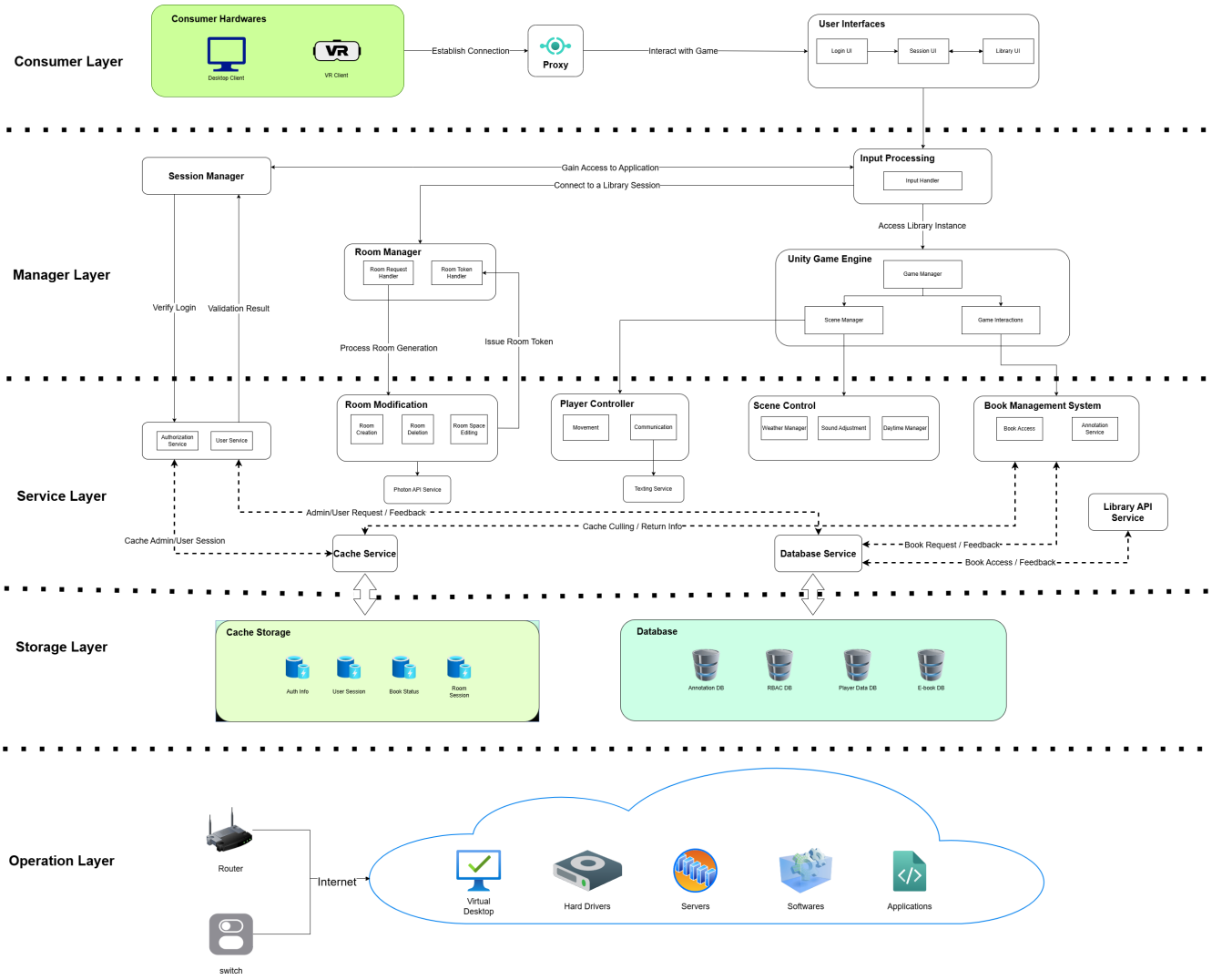

Figure 25: The technical architecture conceived for this project.

2.2 Quality of Service Technical Requirements

2.2.1 Security Requirements

Encryption

- Do transactions need to be encrypted?
- Level of encryption? (e.g., 40-bit encryption in US)

All data exchanges between client and server must be encrypted to protect user information and credentials. The system uses HTTPS with TLS 1.2 or higher (AES-256 typical) to ensure strong encryption.

User Identification

- uid/pw, cookies, certificates, application-level?
- Existing customer database to identify online visitors?

User authentication uses a username–password model with JWT tokens for session management. A dedicated user table handles account storage and identity management. Passwords and user roles are securely stored (hashed and salted) in the database. Authentication occurs at the application level.

Access to Data

- Do you need to restrict access to parts of the site?
- What privacy rules apply to user-provided information?

The system enforces role-based access control (RBAC). Different features are restricted by role, with administrative areas limited to authorized users only. User data follows strict privacy practices:

- Store only essential data (data minimization)
- Hash passwords with salt (bcrypt)
- Prevent sensitive data exposure through APIs
- Enforce database access control
- Apply GDPR-inspired principles: consent, right to delete, secure storage

Auditing and Legal Requirements

Question: What are the legal requirements and policies for auditing content, changes, and transactions? The system maintains audit trails for accountability and compliance with general data-protection and academic standards.

Auditing Policy Highlights:

- Structured and secure logging framework across all services (logins, data modifications, admin actions)
- No sensitive raw data (e.g., plaintext passwords)
- Tamper-resistant, timestamped entries for compliance-grade auditing
- Exportable timestamped logs for compliance and debugging
- Continuous improvement of auditing based on reviews

Demilitarised Zone (DMZ)

Question: Do you plan to use a secure demilitarised zone for server code? No dedicated DMZ is required due to the academic scope and resource limits of the project.

2.2.2 System Management

Infrastructure Access

Question: Do you have access to infrastructure for hosting and running your own server? Yes. The team can deploy and run its own server environment using Unity Cloud or equivalent services.

Response Time Targets

Goal: Maintain a smooth, responsive user experience.

- API responses: <300 ms
- UI actions: <1 s
- Database queries: <150 ms
- Heavy processing (AI/analysis): asynchronous, with immediate UI feedback

Availability

- Service hours: 24/7
- Scheduled maintenance: allowed during off-peak hours
- Target recovery time: 5–30 min depending on severity

The service is web-based, so high reliability is desirable but not mission-critical for academic use.

Failure Monitoring and Handling

Failures are monitored via Unity diagnostics and logs.

- Health checks with Unity's `Application.isPlaying`
- Logs via `Debug.Log`, persistent log files
- External monitoring: Unity Cloud Diagnostics or Sentry
- Alerting through the Unity console

Failure Recovery:

- Automatic reloads or subsystem restarts on crash
- Fallback UI ("Reconnecting..." overlay)
- Manual log inspection for debugging
- Data persistence via `PlayerPrefs` or local cache

Philosophy: Fail fast, debug fast, recover fast — prioritizing rapid continuity for academic prototyping.

Recovery Plan

Project data can be restored from version control and backups.

- Use Git or Unity Collaborate for versioning
- Store backups via Unity Cloud Build
- Follow a documented redeployment procedure

Tracking and Documentation

- Problem tracking through Unity logs and GitHub Issues/Jira
- Statistics: player sessions, crash reports, FPS, memory usage
- Performance instrumentation: scene load times, network latency
- Data stored locally or optionally uploaded to analytics service

For production scale, the architecture can integrate Prometheus, Grafana, or ELK.

2.2.3 Client-Side Management

Target Users

- Primary customers: Internet users
- Expected skill: basic computer literacy

The UI is intuitive and designed for general users; administrators have additional system-management capabilities.

Language Support English only.

Usage Patterns

Users will search and browse data.

- Keyword search for specific items
- Browsing categorized content
- Refining queries and revisiting past searches

State Management

Client-side:

- JWT in HTTP-only cookies
- UI state managed with browser storage

Server-side:

- Stateless backend using token-based identity
- Persistent state stored in database

Code Distribution

The client code (HTML, CSS, JS) is delivered directly via the web server. Users simply visit the URL — no installation needed. Version updates are deployed server-side for automatic propagation.

Client Architecture

Modern HTML, JavaScript, and Fetch/AJAX are used to maintain responsive communication. This avoids legacy dependencies (e.g., VBScript) and improves performance.

User Interfaces

General User Interface:

- Dashboard, search/browse view, detail pages, profile settings

Administrative Interface:

- Management dashboard, user control, content management, log view

Optional Views:

- Responsive layouts, error/status pages

2.2.4 Network Management

Network Scope

- Operates over the Internet via HTTPS + REST APIs

Application, Data, and Object Placement

Application: Backend runs on cloud VM; frontend delivered to browser.

Data: Stored on backend database; only transient state client-side.

Objects: Logic on backend, UI in browser.

Security Functions

HTTPS/TLS ensures encryption, integrity, and server authentication. JWTs and RBAC handle user-level authorization. Input validation and sanitization protect against injection and XSS.

Network Performance

Response time is affected by latency, bandwidth, and handshake overhead. DNS lookups, TLS negotiation, and routing contribute small delays, acceptable at this project's scale.

2.2.5 Server-Side Management

Server Architecture

Single-server setup for backend + database, suitable for project scope. Modular design allows future scaling into multiple servers if needed.

Security Functions

- JWT verification and role enforcement
- Input validation
- Secure password storage (hash+salt)
- HTTPS-only communication
- Database protection and event logging
- Rate limiting to deter brute-force attempts

Performance Considerations

Response-time factors:

- API and DB query latency
- Network delay
- Server load

Optimizations:

- Stateless backend design
- Indexed database queries
- Caching and minimal payloads
- Async tasks for heavy jobs

2.2.6 Application Logic

Client-Side Executables

No native executables required. The app runs in the browser with HTML, CSS, and JS, connecting via HTTPS APIs.

Logic Split
Client:

- UI rendering and validation
- Sending/receiving data

Server:

- Core logic, security, validation, authentication
- Database access and heavy processing

This split ensures performance and security balance.

2.2.7 Connectors

External Systems

- Online Library

Data Transfer
Data exchanged through REST APIs using JSON over HTTPS.

Data Freshness and Caching
Simple caching keeps data reasonably current for project needs.

Access Mode
Asynchronous communication keeps the UI responsive. Offline mode is not required.

Platform and Connector Requirements
No specialized enterprise connectors or OS-level dependencies required.

Security Policies
Standard web and user-data security practices apply.

Scalability and Performance
Predicted moderate traffic (VR library system).
Approach:

- Stateless backend for scalability
- REST API + async data for responsiveness
- Cached assets and database indexing
- Streaming of large VR assets

If future demand increases, scaling options include larger servers, CDN use, and separate asset hosting.

3 Detailed Project Development Plan

3.1 Methodology

3.1.1 What software are we going to use:

- **UI and 3D design:**

  Unity: We are going to use Unity as the primary game engine for constructing the entire virtual library environment, user interface, and interactive features. Utilizing Unity's robust VR support—including XR Interaction Toolkit and built-in VR templates we will try to create immersive spaces with real-time lighting, spatial audio, gesture-based interactions, physics, and intuitive controls. We might include features such as drag-and-drop UI elements and customizable menus that directly align with modules like SettingsMenu, AnnotationInterface, and navigation flows. Unity also supports rapid prototyping and iterative development, which is essential for testing and refining library layouts, UI feedback loops, and user-based scenarios.

  Blender: Blender is an open-source 3D modeling tool to generate detailed asset models—books, furniture, decor, and more. These 3D assets are designed in Blender, exported in compatible formats (FBX, OBJ), and then imported into Unity for placement and interaction in the virtual world. Blender supports advanced texturing, sculpting, and baking, ensuring visually compelling and optimized meshes for VR environments where performance and realism are critical.

- **Server/Networking:**

  Photon2: Photon is a networking solution integrated into Unity to enable real-time multiuser functionality, session management, and collaborative features. Through Photon, public sessions can synchronize the weather conditions outside the library, abient sound and lighting and private sessions can synchronize user activities, manage chat and collaborative annotations, and handle events like joining/leaving rooms or updating shared canvases. Photon provides reliable server infrastructure, and a matchmaking framework that supports persistent rooms and seamless communication between clients, essential for enabling immersive, shared VR experiences in the app.

- **Database Management System:**

  MySQL: MySQL functions as the backend for storing and retrieving persistent app data—including user authentication details (Authenticator), role assignments (RoleManager), ebook library content (EbookManager), and application state across sessions. With Unity, MySQL can be accessed via web APIs, cloud services, or direct database plugins. Its relational database structure efficiently organizes and queries information, supporting robust identity management, asset indexing, and library inventory needed for a scalable virtual library environment.

- **Version Control:**

  Git: Git is the version control backbone for all code, project scripts, and most asset types. Teams use platforms like GitHub or GitLab to collaborate, manage feature branches, and ensure integrity across distributed development environments. For larger files—such as 3D models or scene assets—Unity's Plastic SCM and Large File Storage (LFS) extensions may also be adopted for optimized asset tracking. Version control practices enable safe experimentation, rollback of changes, and continuous integration, supporting the rapid, iterative workflows required for VR projects.

3.1.2 Agile Methodologies:

Agile methodologies are widely used in software development because of their flexibility and iterative approach, making them ideal for VR projects where requirements and feedback evolve rapidly. Agile methods focus on building the application incrementally, with new functionalities added in short development cycles (iterations/sprints). Key practices include frequent meetings, real-time project tracking, and the ability for teams to rapidly incorporate changes or new client requests

- **Agile frameworks we plan to use:**

  Extreme Programming (XP): Highlights teamwork, client feedback, collective code ownership, pair programming, continuous integration, and coding standards to foster quality and adaptability.

These agile approaches promote collaboration, transparency, and fast response to change—essential for developing robust and user-centered VR applications.

3.1.3 Project Phases and Iterations:

The project is structured into weekly iterations, where each iteration serves as a focused sprint aiming to complete a meaningful set of features or improvements. During each iteration, team members are systematically rotated into different pair programming groups, as shown in the tables, ensuring that everyone collaborates with multiple partners and gains broad exposure to all aspects of the project. This approach facilitates knowledge sharing and collective code ownership while allowing iterative development of core components—such as navigation, browsing, interactive features, and collaborative tools—toward the final, fully functional virtual library application.

3.2 Coding Standards

General Principles:

- Respect of object-oriented programming principles,
- Regular code cleanup & refactoring,
- Test-driven, object-oriented development,
- modular, extendable code structure,
- comments to enhance code readability,
- consistent and explicit use of access modifiers,
- Aim for maximal algorithm optimization,
- Comprehensive and clean error handling

Project Structure & Naming:

- files and folders separated by feature/module,
- namespace separation by component,
- clear, but concise naming of files, folders, classes, variables and methods,

Naming Conventions:

- `PascalCase`: class names, public methods and fields,
- `camelCase`: local variables, properties and parameters,
- `foldername/subfoldername`: folder structures,
- `feature/branchname`: version control branches.

Version Control:

- Branching per feature to avoid conflicts,
- Regular merging to minimize size of merge conflicts.

3.3 Pair Programming Plan

3.3.1 Why to follow Pair Programming?

Pair Programming will be adopted throughout the coding phase to improve code quality, share knowledge among team members, maintain consistent coding practices, and reduce defects early.

3.3.2 Pairing Strategy to Be Followed

- Pairs will change every iteration.
- Members take turns as Driver and Navigator.
- Rotation ensures everyone works on all parts of the system.

3.3.3 Iterations

Person	Josh	Pri	Reid	Sarvesh	Daoqi	Gong Xu	Viraj	Vivian	Keshav	Yokesh
Josh		3	7	6	5	2	8	9	4	1
Pri	3		9	4	2	6	5	8	1	7
Reid	7	9		8	4	5	1	2	3	6
Sarvesh	6	4	8		7	1	3	5	9	2
Daoqi	5	2	4	7		3	6	1	8	9
Gong Xu	2	6	5	1	3		9	4	7	8
Viraj	8	5	1	3	6	9		7	2	4
Vivian	9	8	2	5	1	4	7		6	3
Keshav	4	1	3	9	8	7	2	6		5
Yokesh	1	7	6	2	9	8	4	3	5	

Figure 26: Team Division

Iteration	Group 1	Group 2	Group 3	Group 4	Group 5
Iteration 1	Viraj - Reid	Yokesh - Josh	Daoqi - Vivian	Pri - Keshav	Gong Xu - Sarvesh
Iteration 2	Reid - Vivian	Pri - Daoqi	Sarvesh - Yokesh	Josh - Gong Xu	Viraj - Keshav
Iteration 3	Josh - Pri	Reid - Keshav	Sarvesh - Viraj	Daoqi - Gong Xu	Vivian - Yokesh
Iteration 4	Gong Xu - Vivian	Josh - Keshav	Reid - Daoqi	Pri - Sarvesh	Viraj - Yokesh
Iteration 5	Sarvesh - Vivian	Keshav - Yokesh	Josh - Daoqi	Reid - Gong Xu	Pri - Viraj
Iteration 6	Josh - Sarvesh	Reid - Yokesh	Pri - Gong Xu	Daoqi - Viraj	Vivian - Keshav
Iteration 7	Pri - Yokesh	Sarvesh - Daoqi	Gong Xu - Keshav	Vivian - Viraj	Josh - Reid
Iteration 8	Reid - Sarvesh	Josh - Viraj	Pri - Vivian	Daoqi - Keshav	Gong Xu - Yokesh
Iteration 9	Daoqi - Yokesh	Pri - Reid	Josh - Vivian	Sarvesh - Keshav	Gong Xu - Viraj

Table 1: Pair Programming Rotations for 9 Iterations

3.3.4 Roles Within Pairs

Each pair will assume complementary roles in alignment with the system's technical domains. This division promotes balanced collaboration and ensures that all members gain exposure to both client- and server-side aspects of the project. Roles are fluid and may vary between milestones, but the following categories will be maintained to provide clarity and ownership:

- **VR/Unity Developer:** Responsible for immersive interaction design, object manipulation, lighting, environment effects, and VR hardware integration. Ensures that the virtual library offers realistic and intuitive interaction.
- **Shared Systems Developer:** Implements multi-user collaboration features, such as shared reading, annotations, and the notice board, maintaining synchronization between concurrent users.
- **Backend/API Developer:** Designs and, maintains the application server, handling authentication, roles, sessions, networking, and communication between the web and VR platforms.
- **Web Application Developer:** Develops the companion web dashboard for book search, purchase, and subscription management, linking seamlessly with backend services.
- **Infrastructure and Quality Engineer:** Oversees DevOps pipelines, continuous integration, automated testing, and performance monitoring to ensure system reliability.

Pairs will be composed so that each team includes complementary expertise, so that we encourage knowledge transfer and proper understanding of the system. Every weekly meeting, we will plan the reshuffle so that one member from each pair will rotate to a different while the other remains as an anchor point, ensuring that institutional knowledge is continuously shared without disrupting feature progress.

3.4 Milestones

Stage	**Milestone**	**Task**	**Group(s)**
Design Stage	Milestone 1	Project setup	Group 1
		Framework initialization	Group 2
		User story development	Group 3
		Server distribution	Group 4
		Book data preparation	Group 5
	Milestone 2	Data structure design	Groups 1, 2
		UI design	Groups 3, 4
		UX design	Group 5
Core Functions	Milestone 3	User login implementation	Group 1
		Room manager	Groups 2, 5
		Authorization service	Group 3
		User service	Group 4
	Milestone 4	Photon API integration	Groups 1, 4
		Text service	Group 2
		Library service	Groups 3, 5
	Milestone 5	Game engine programming	Groups 1, 2
		Room modification	Group 3
		Book management	Groups 4, 5
	Milestone 6	Game engine programming (continued)	Groups 1, 2, 5
		Player control system	Group 3
		Scene control	Group 4
Ops Stage	Milestone 7	Performance optimization	Group 1
		User testing	Groups 2, 3
		Bug fixing	Groups 4, 5
		Deployment preparation	All groups

Table 2: Project Stages, Milestones, Tasks, and Assigned Groups