DEPARTEMENT OF COMPUTER SCIENCE

LANCER'S HUB: VIRTUAL OFFICE

A STUDY PRESENTED AS A PARTIAL FULFILMENT OF THE REQUIREMNT FOR BSC DEGREE IN COMPUTER SCIENCE

Done by:

GROUP MEMBERS

1. **Abenezer Sisay RCD/3064/2013**
2. **Betselot Desta RCD/O550/2013**
3. **Kidus Tesfu RCD/3068/2013**
4. **Getnet Tibebe RCD/0560/2013**

DATE : January-17-2025

SMU

ADDIS ABABA

Advisor: **Haileyesus Tilahun**

LANCER'S HUB: VIRTUAL OFFICE

Contents

Chapter 1. Introduction

1.1. Background of Organization

The rise of remote work and the increasing demand for flexible, cost-effective workspaces have driven organizations worldwide to adopt virtual office solutions. In Ethiopia, the need for such platforms is especially significant due to limited access to traditional office infrastructure and the growing entrepreneurial and freelance workforce.

Lancers Hub was developed as a virtual office solution tailored to meet the unique needs of Ethiopia's freelancers, startups, and remote workers. By leveraging cutting-edge technologies like **React**, **Node.js**, **Phaser3**, and **Colyseus**, the platform offers an immersive, interactive 2D environment designed to replicate the professional atmosphere of a traditional workspace. Originally inspired by platforms such as Gather.town, Lancers Hub evolves beyond the limitations of existing solutions by integrating advanced real-time communication, collaboration, and customizable workspaces.

The project initially considered Unity as the primary engine for the virtual environment. However, due to challenges like its steep learning curve, high licensing costs, and integration issues with other components such as the React-based landing page and office systems, the development team pivoted to Phaser3 and Colyseus for the game engine and multiplayer networking.

This change reflects the team's commitment to creating a seamless, affordable, and adaptable solution that empowers Ethiopia's burgeoning digital workforce to overcome geographical and infrastructural barriers while fostering a collaborative and innovative virtual ecosystem.

1.2. Statement of the problem

Problem 1: Lack of Dedicated Workspaces

Freelancers in Ethiopia often lack access to dedicated workspaces, leading to distractions, reduced productivity, and limited opportunities for networking and collaboration. Traditional solutions like coworking spaces are not always accessible or affordable. **Lancers Hub** addresses this gap by providing a distraction-free, virtual environment that enables professionals to focus and connect seamlessly.

Problem 2: High Costs for Startups

Startups face financial challenges when securing traditional office spaces, which often require long-term leases and incur significant overhead costs. The virtual office solution eliminates these financial burdens, offering cost-effective and scalable alternatives for collaborative work environments.

Problem 3: Isolation of Remote Workers

Remote workers often struggle with feelings of isolation, lack of social interaction, and difficulties maintaining work-life balance. By integrating features such as proximity-based video streaming and virtual coworking areas, **Lancers Hub** creates an engaging and supportive community for remote professionals.

Problem 4: Inflexible Traditional Platforms

Existing platforms for virtual collaboration are either too costly or limited in their customization options. **Lancers Hub** overcomes these limitations with an adaptable, modular architecture, enabling users to tailor their virtual office environments to their specific needs while remaining cost-effective.

1.3. Objectives of the Project

1.3.1. General Objective

To create a scalable, user-friendly, and immersive virtual office platform that fosters collaboration and innovation among Ethiopia’s freelancers, startups, and remote workers.

1.3.2. Specific objectives

- To identify and address the primary challenges faced by remote professionals in Ethiopia.
- To design and implement a modular, scalable virtual office platform.
- To integrate advanced real-time communication tools such as WebRTC and Colyseus.
- To develop a responsive and interactive 2D environment using Phaser3.
- To enable customizable office layouts and workspaces tailored to user needs.
- To support secure user authentication and access control mechanisms.
- To provide seamless integration with existing collaboration tools and APIs.
- To reduce the costs of maintaining traditional office spaces.
- To foster a supportive and engaging professional community.

1.4. Methodology and Approach

1.4.1. Data Collection

Data collection methods included:

- **Surveys and Questionnaires**: Distributed to freelancers, startups, and remote workers to gather insights into their challenges and requirements.
- **Competitor Analysis**: Evaluating existing platforms like Gather.town to identify gaps and opportunities.
- **Interviews and Focus Groups**: Conducted with industry professionals to validate the system design and features.

1.4.2. System Development and Process Model

The development followed an **Agile methodology**, allowing iterative improvements and adaptability to changing requirements. The team prioritized modularity through a microservices architecture, with independent components for real-time communication, workspace customization, and task management.

1.4.3. Design Pattern

The project employed the **Model-View-Controller (MVC)** pattern for separation of concerns, facilitating maintainability and scalability. The **Singleton pattern** was used for managing global states like session management and configuration.

1.4.4. Programming Language

- **Frontend**: React for dynamic and responsive user interfaces.
- **Backend**: Node.js for server-side operations and API integration.
- **Game Engine**: Phaser3 for interactive environments and Colyseus for multiplayer networking.

1.5. Tools

1.5.1 Hardware Tools

- Standard PCs and laptops with internet access.
- Devices for testing various screen resolutions and performance metrics.

1.5.2 Software Tools

- **Frameworks**: React, Node.js, Phaser3, Colyseus.
- **Database**: Firebase for real-time data management.
- **IDE**: Visual Studio Code for coding and debugging.
- **Collaboration Tools**: GitHub for version control and Jira for project management.

1.6. Scope and Limitation of the Project

1.6.1 Scope of the Project

- Platform: Web-based virtual office solution.
- Users: Freelancers, startups, remote workers.
- Features: Virtual workspaces, real-time communication, task management, and customizable layouts.
- Compatibility: Cross-platform compatibility for web and mobile devices.

1.6.2. Limitations of the Project

- Dependency on reliable internet access, which may limit usage in underserved areas.
- Lack of advanced sensory features (e.g., haptic feedback) present in physical office environments.
- Restricted to the features provided by the free tiers of tools like Firebase.

1.7 Significance of the Project

The Lancers Hub virtual office project has a wide range of benefits, including:

- **For Freelancers**:
    - Provides a distraction-free workspace, increasing productivity and professionalism.
    - Enables global networking opportunities through real-time communication and customizable portfolios.
    - Reduces overhead costs by eliminating the need for physical offices.
- **For Startups**:
    - Offers a scalable, cost-effective alternative to physical office spaces.
    - Facilitates team collaboration through features like task management and virtual meeting rooms.
    - Enhances resource allocation by reducing expenses associated with physical infrastructure.
- **For Remote Workers**:
    - Combats isolation by fostering a sense of community within virtual coworking spaces.
    - Supports work-life balance with customizable and adaptable environments.
    - Improves engagement and focus through immersive design elements.
- **For the Broader Community**:
    - Democratizes access to professional workspaces, fostering equity and inclusion.
    - Encourages innovation within Ethiopia’s entrepreneurial ecosystem.

- Contributes to the digital transformation of workplace solutions in the region.

1.8. Feasibility Study

1.8.1 Technical Feasibility

The project leverages mature and accessible technologies like React, Node.js, Phaser3, and Firebase, which are well-supported and suitable for a scalable virtual office platform. The team possesses the required skills to implement these technologies, ensuring the system's technical feasibility.

1.8.2. Economic Feasibility

The project minimizes costs by utilizing free or open-source technologies where possible. For example:

- Phaser3 and Colyseus eliminate licensing costs associated with Unity.
- Firebase’s free tier supports real-time communication and database management.

1.8.3. Operational Feasibility

The proposed system is operationally feasible, as it runs on standard hardware and software setups available to the target users. Post-deployment, the system is designed to be easy to maintain, with modular components ensuring low operational overhead.

1.8.4. Legal Feasibility

The project complies with data privacy regulations, includes secure user authentication and encryption to protect sensitive information.

1.9. Risk Assessment

1.9.1. Risks

Potential risks include:

- **Technical Risks**:
    - Integration challenges between Phaser3 and Colyseus.

- Dependency on third-party services like Firebase.

- **Operational Risks**:
  - Resistance to adoption by users unfamiliar with virtual platforms.

- **Infrastructure Risks**:
  - Unreliable internet access in some areas of Ethiopia.

1.9.2. Mitigation Strategies

- Regular testing and debugging to ensure seamless integration.
- Comprehensive user onboarding and training materials.
- Optimizing the platform for low-bandwidth environments to address connectivity issues.

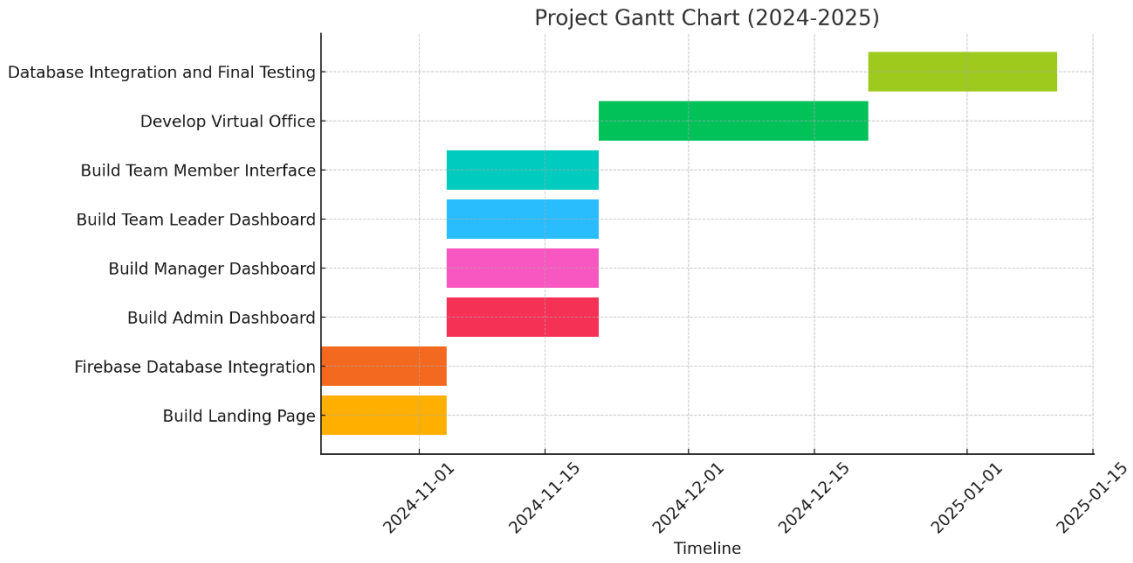
1.9.3. Assumptions

- Users will have basic technical literacy and access to internet-enabled devices.
- The selected tools and technologies will remain available and supported during development.

1.9.4. Constraints

- Limited budget and access to paid tools.
- Dependence on Ethiopia’s internet infrastructure, which may pose performance challenges in remote regions.

1.10. Work Break Down



Chapter 2: Business Area Analysis and Requirement Definition

2.1. Introduction

Lancers Hub is designed to address the unique needs of freelancers, startups, and remote workers in Ethiopia, combining administrative task management with an immersive virtual office environment. The system integrates a landing page for onboarding, a task management module for streamlined workflows, and an interactive 2D virtual office for enhanced collaboration.

This chapter delves into the business processes, current challenges, and proposed solutions that define Lancers Hub, alongside a detailed analysis of requirements.

2.2. Business Area Analysis

2.2.1. Detailed Analysis

Lancers Hub was developed to provide an efficient, collaborative, and scalable workspace solution. The platform's comprehensive functionality includes:

- **Landing Page**: Offers an overview of services, facilitating user onboarding through registration, login, or office purchase options.
- **Task Management System**: Allows managers and team leaders to oversee project progress, assign tasks, and manage user roles.
- **Virtual Office Environment**: A 2D interactive space for real-time collaboration with features like chat, screen sharing, and a whiteboard.

2.2.2. Current System

The existing solutions, such as traditional office spaces or basic task management software, fail to:

- Provide cost-effective, scalable options for remote teams.
- Enable immersive, interactive collaboration.
- Integrate role and task management with virtual coworking spaces.

2.2.3. Players of the existing system

- **Freelancers**: Require professional environments for focused work and networking.
- **Startups**: Face challenges with scalability and affordability in managing teams.
- **Remote Workers**: Seek engaging environments that reduce isolation and improve collaboration.

2.2.4. Proposed System

Lancers Hub addresses these limitations by:

- Enabling **managers** to purchase virtual office spaces and oversee task delegation and user roles.
- Allowing **team leaders and team members** to register with unique room IDs for task assignments and collaboration.
- Integrating an interactive office space to foster natural, productive communication and teamwork.

2.2.5. Forms and Reports Used

- **User Registration Form**: Collects room IDs, personal details, and CVs for role approvals.
- **Task Progress Reports**: Displays progress tracking on managers' and team leaders' dashboards.
- **CV Management**: Allows managers to view and download CVs for approval purposes.
- **Report management** : Allow members to download reports of their statstics

2.3. Requitement Gathering

2.3.1. Requirement Gathering Techniques

- **Surveys and Questionnaires**: Conducted among freelancers, startups, and remote workers to identify pain points and desired features.
- **Competitor Analysis**: Assessed platforms like Gather.town for benchmarking features and functionalities.

- **Interviews**: Held with industry professionals to validate system features and usability.

2.4. Method of communication

2.4.1. Communication Techniques

- Proximity-based Real-Time Communication: Integrated in the 2D office for natural interactions.
- In-System Messaging: Facilitates asynchronous communication within teams.

2.5. Requirement Definition

2.5.1. Functional Requirements

2.5.1.1. Essential Use case Modeling

The system revolves around key use cases like:

- **Office Purchase**: Managers purchase virtual office spaces, generating unique room IDs.
- **Registration and Role Assignment**: Users register using room IDs, while managers approve roles.
- **Task Management**: Managers and team leaders assign and track tasks.
- **Interactive Collaboration**: Users access the virtual office for real-time collaboration.

2.5.1.2. Actor Description

- **Manager:** Purchases offices, approves users, assigns roles, and manages tasks.
- **Team Leader:** Assigns tasks to team members and uploads CVs for approval.
- **Team Member:** Completes tasks assigned by managers or team leaders.
- **System Admin:** Approves manager registrations and monitors overall system functionality.

2.5.1.3. Essential Use Case Description

Use Case: Office Purchase

Actors: Manager

Description: The manager purchases an office and receives a unique room ID for distribution to team members.

Use Case: Task Assignment

Actors: Manager, Team Leader

Description: Managers assign tasks to team leaders, who further delegate tasks to team members.

2.5.1.4. Essential Use Interface Prototyping (Low Fidelity Prototype)

Screens include:

- Landing Page: Features login, registration, and office purchase buttons.
- Manager Dashboard: Displays user roles, task assignments, and progress tracking.
- Interactive Office Interface: Accesses real-time collaboration features.

2.5.2. Collaboration Modeling

The system ensures collaboration through:

- Real-time task updates.
- Interactive whiteboards and chat tools.

2.5.3. Nonfunctional Requirements

- **Scalability**: Handles increasing users and tasks efficiently.
- **Usability**: Intuitive interfaces for all actors.
- **Security**: Implements strong authentication and encrypted communication.

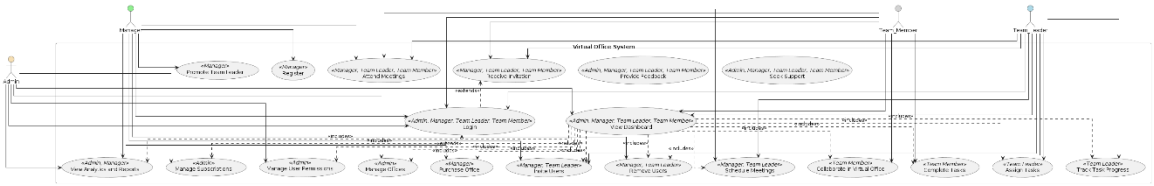
2.6. System Modeling

2.6.1. Introduction

The system design incorporates UML diagrams to visualize workflows and relationships.

2.6.2. System Use Case

Includes key functionalities like office purchase, role assignment, task management, and virtual collaboration.



2.6.3. UI Identification

Key UI components:

- **Landing Page**: Provides service overview and onboarding.
- **Dashboards**: Role-specific dashboards for task and user management.
- **Interactive Office**: Accessible through the "Go to My Room" button.

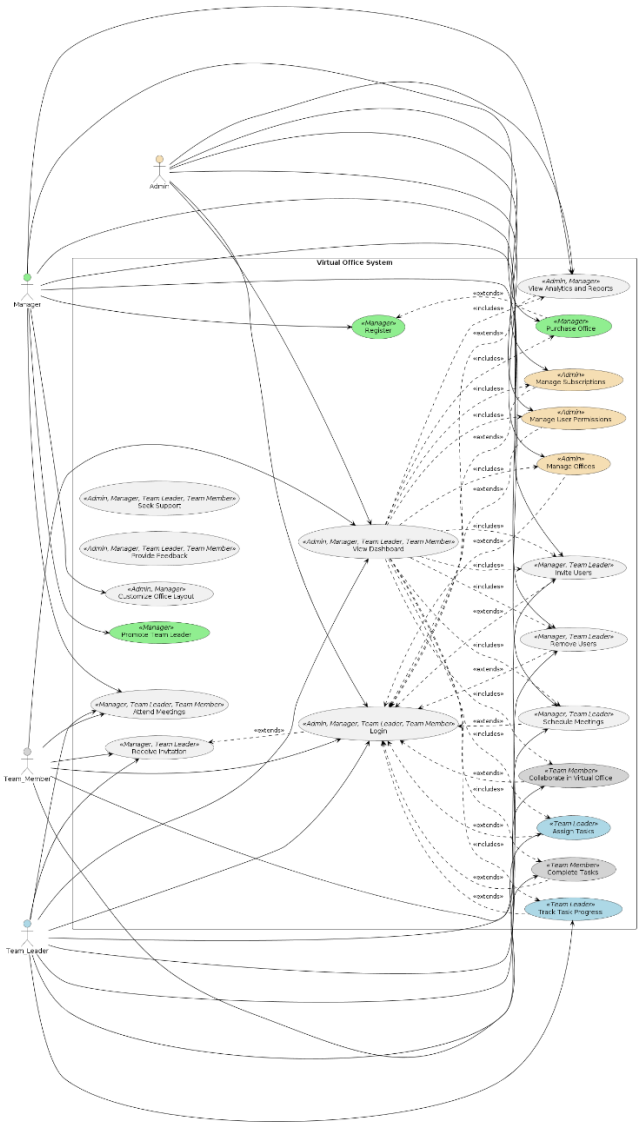
2.6.4. Business Rules Identification

- Only users with valid room IDs can register.
- Managers must approve all registrations before users can log in.
- Role assignments dictate task visibility and interaction privileges.

2.6.5. Actor Identification

Actors include Managers, Team Leaders, Team Members, and System Admins.

2.6.6. Use Case Diagram



2.6.7. Use Case Description

Scenario: Manager Approves User Registration

Actors: Manager, Team Leader, Team Member

Description: The manager validates user profiles and CVs before granting access.

2.6.8. Sequence Diagramming

- Registration: Users register with room IDs and await approval.
- Task Assignment: Managers assign tasks, and team leaders delegate them further.

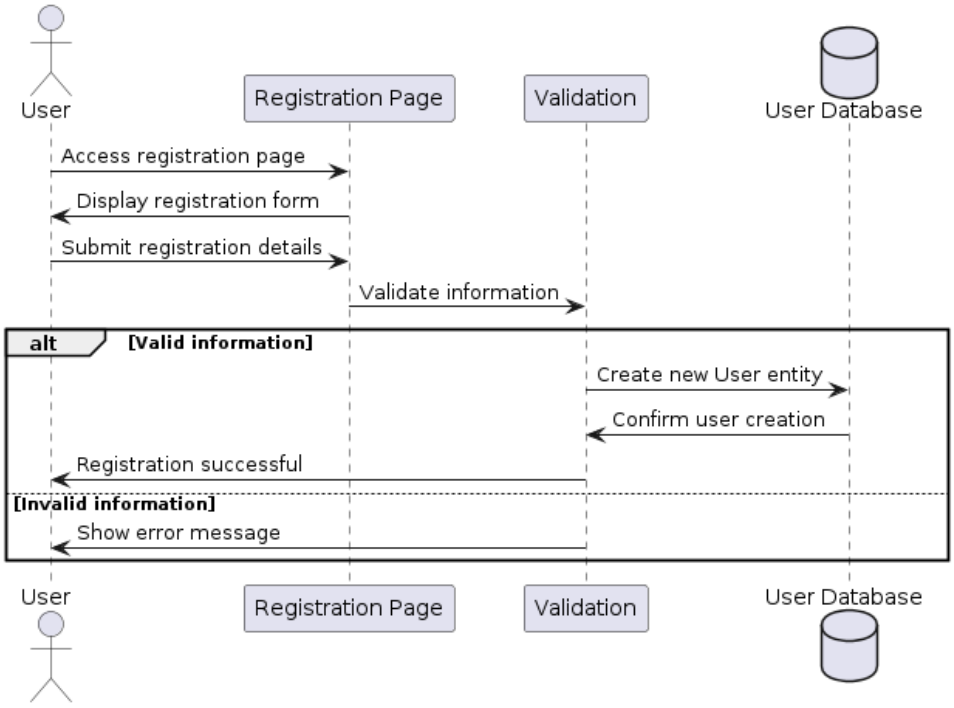


Fig 1.1 Sequence diagram for registration.

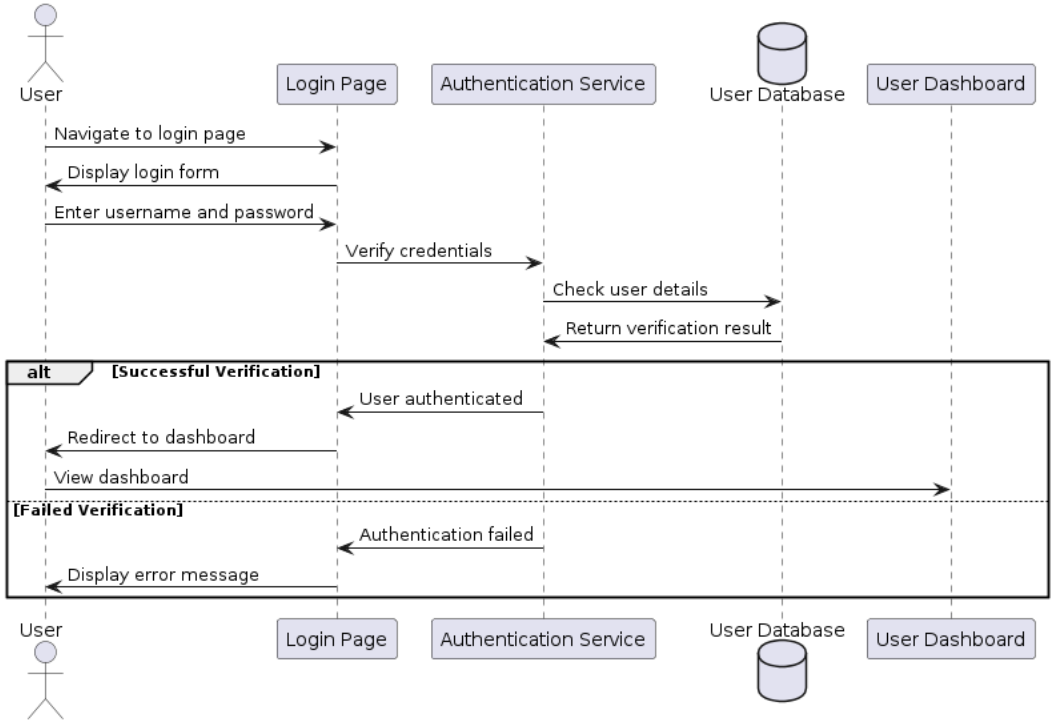


Fig 2.1 Sequence diagram for logging in

1. User Registration and Office Management:

- ✓ A Manager registers in the system.
- ✓ The Manager logs in and views the dashboard.
- ✓ The Manager purchases an office.
- ✓ The Admin manages offices and subscriptions.

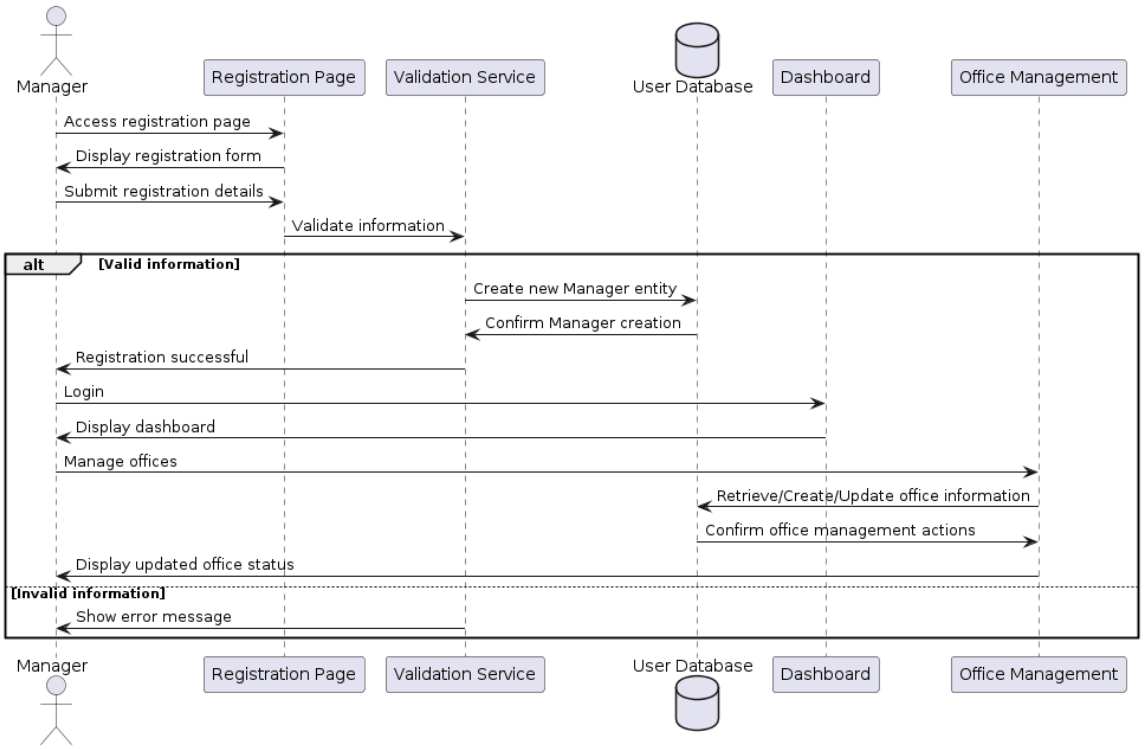


Fig 3.1 Sequence diagram for User Registration and Office Management

2. Task Assignment and Progress Tracking:

- ✓ A Team Leader receives an invitation and logs in.
- ✓ The Team Leader views the dashboard and assigns tasks to Team Members.
- ✓ The Team Leader tracks the progress of tasks.

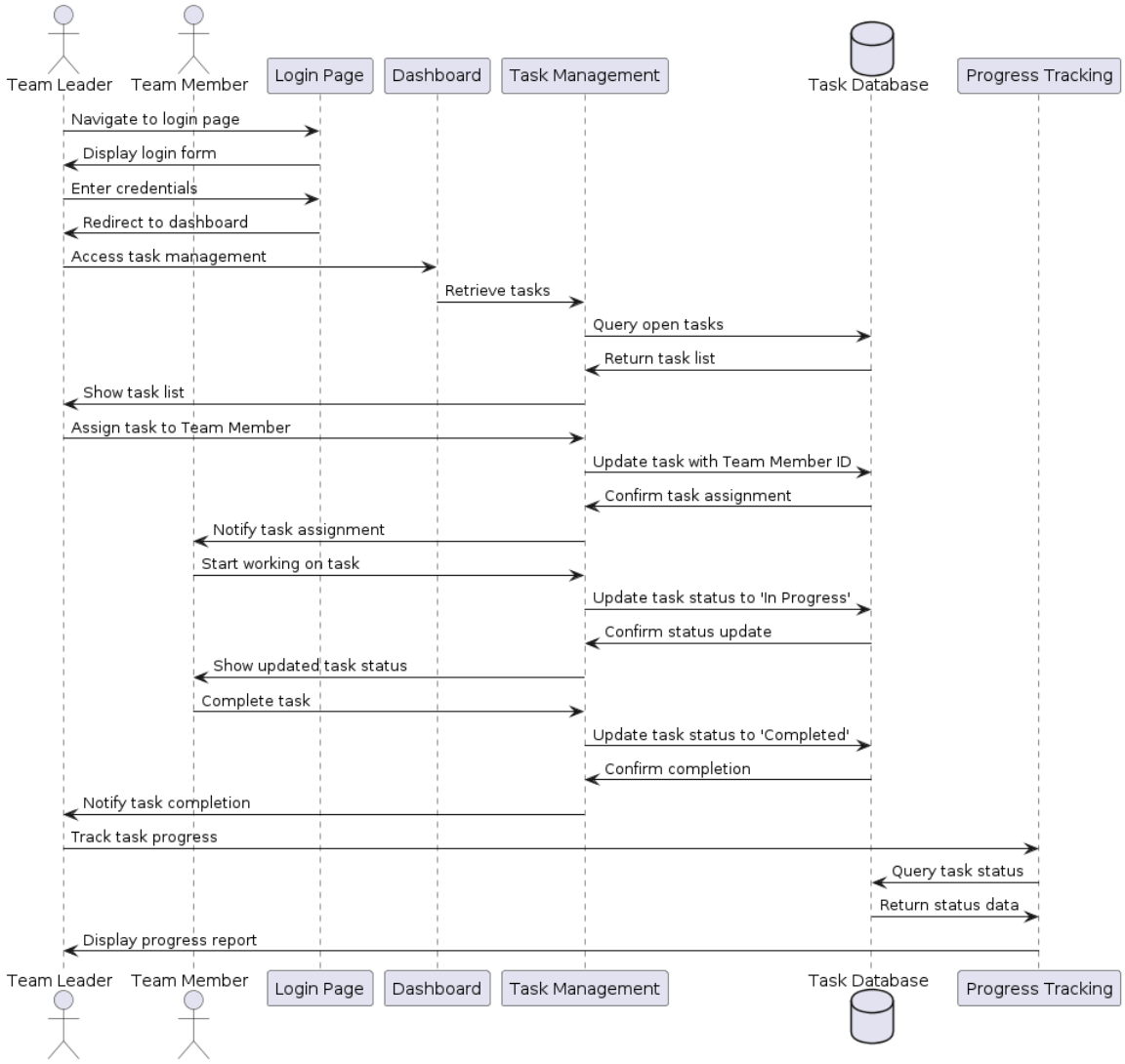


Fig 2.1 Sequence diagram of Task Assignment and Progress Tracking

3. User Invitation and Management:

- ✓ The Manager invites new users.
- ✓ The Manager or Team Leader removes users or promotes a Team Member to Team Leader.

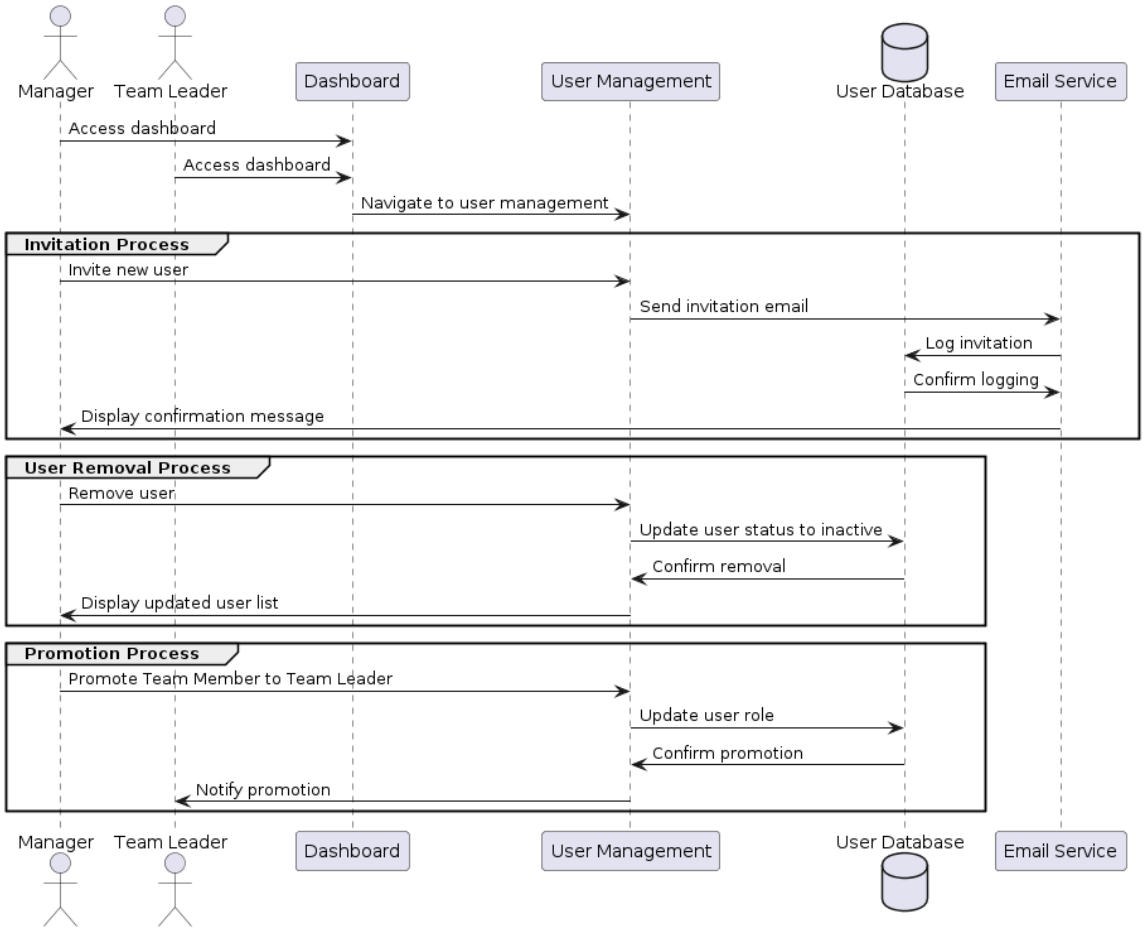
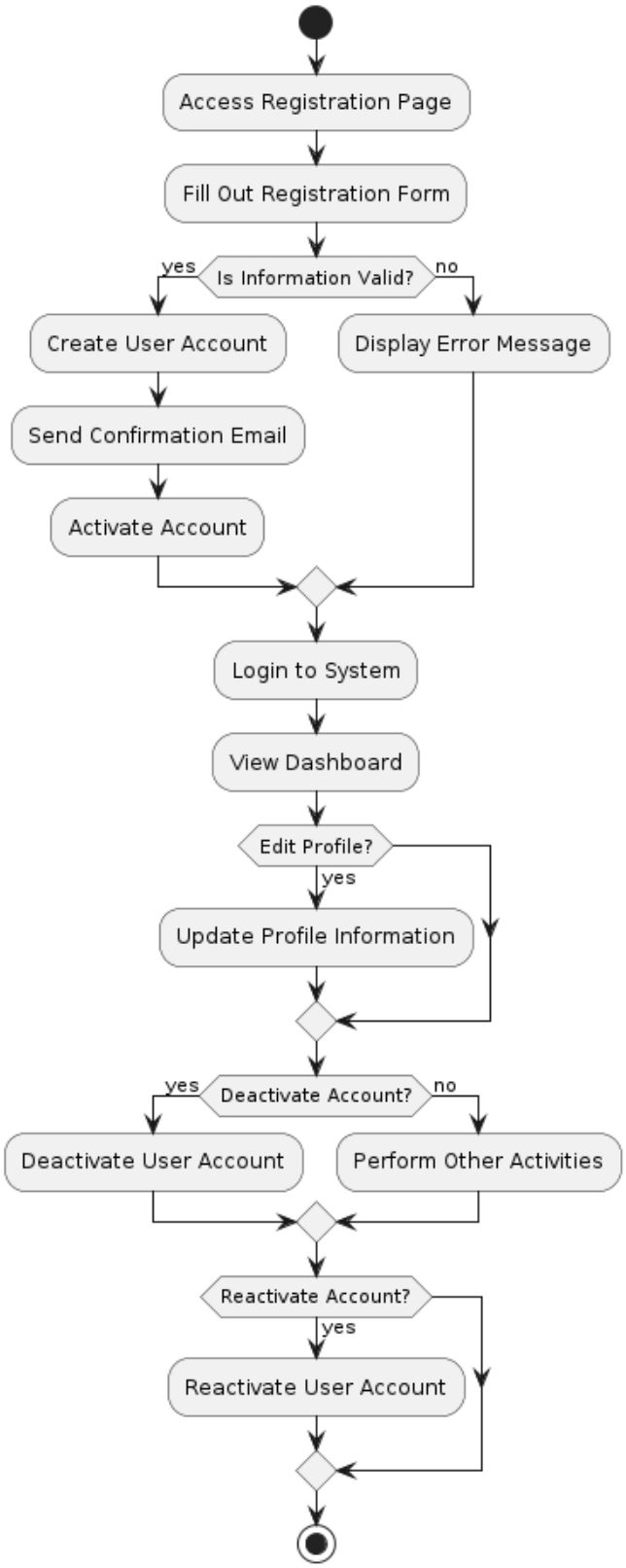


Fig 3.1 Sequence diagram for User Invitation and Management

2.6.9. Activity Diagram

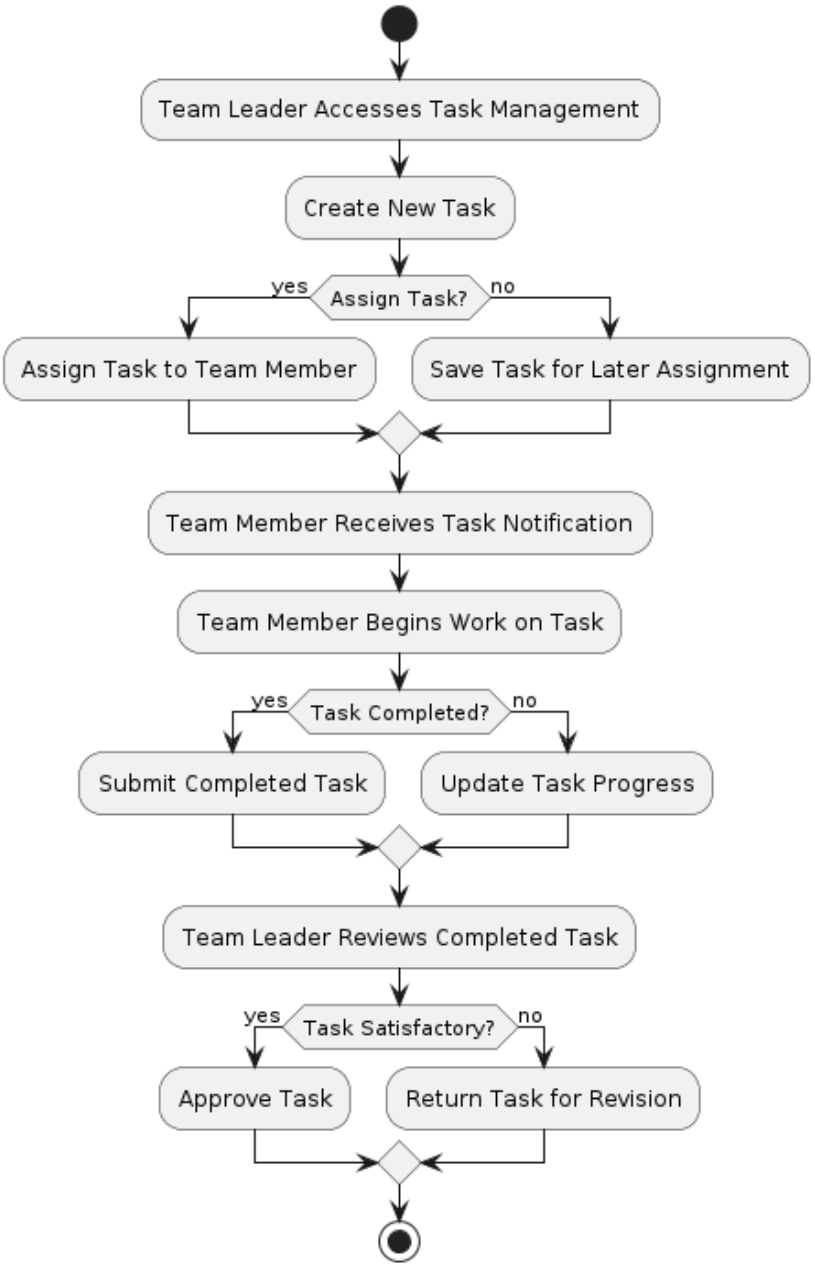
Here are some of the some of the key activities included in our Lancer's Hub system:

1. **User Registration and Account Management:**
   - ✓ Access registration page.
   - ✓ Fill out registration form.
   - ✓ Validate registration information.
   - ✓ Create user account.
   - ✓ Send confirmation email.
   - ✓ Log in to the system.
   - ✓ View and edit user profile.
   - ✓ Deactivate/reactivate account.


Access Registration Page
Fill Out Registration Form
Is Information Valid?
yes
no
Create User Account
Display Error Message
Send Confirmation Email
Activate Account
Login to System
View Dashboard
Edit Profile?
yes
Update Profile Information
Deactivate Account?
yes
no
Deactivate User Account
Perform Other Activities
Reactivate Account?
yes
Reactivate User Account

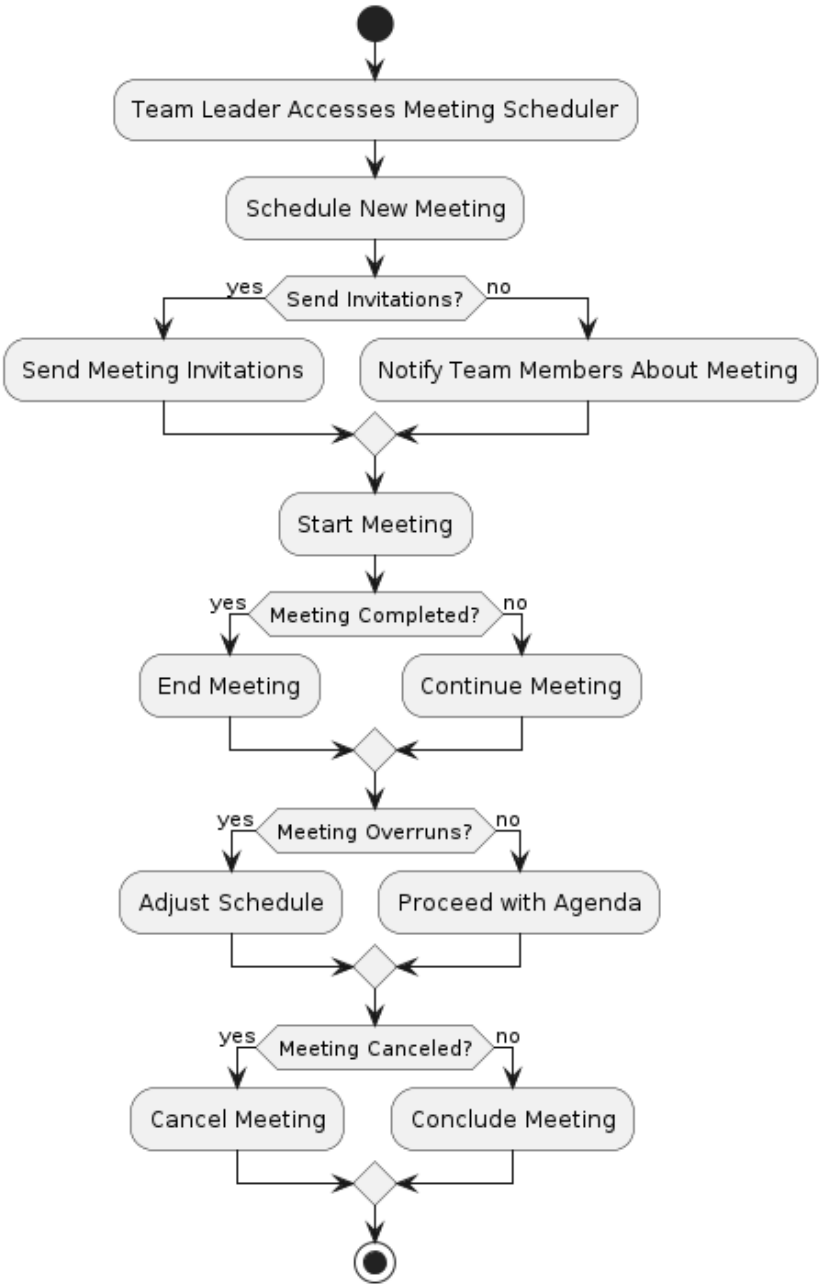
2. Task Management:

- ✓ Assign tasks to team members.
- ✓ Begin work on a task.
- ✓ Update task status.
- ✓ Complete and verify tasks.
- ✓ Handle overdue tasks.



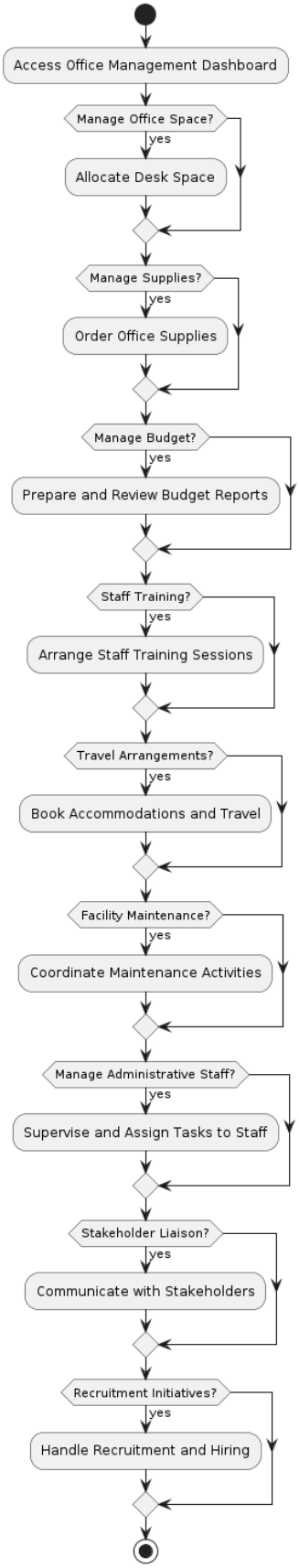
3. Meeting Scheduling and Management:

- ✓ Schedule a new meeting.
- ✓ Send meeting invitations.
- ✓ Start and end meetings.
- ✓ Attend meetings.
- ✓ Log meeting attendance.
- ✓ Cancel or reschedule meetings.



4. Office Management:

- ✓ List available offices.
- ✓ Purchase or rent an office.
- ✓ Manage office subscriptions.
- ✓ Perform office maintenance.
- ✓ Release or retire an office.

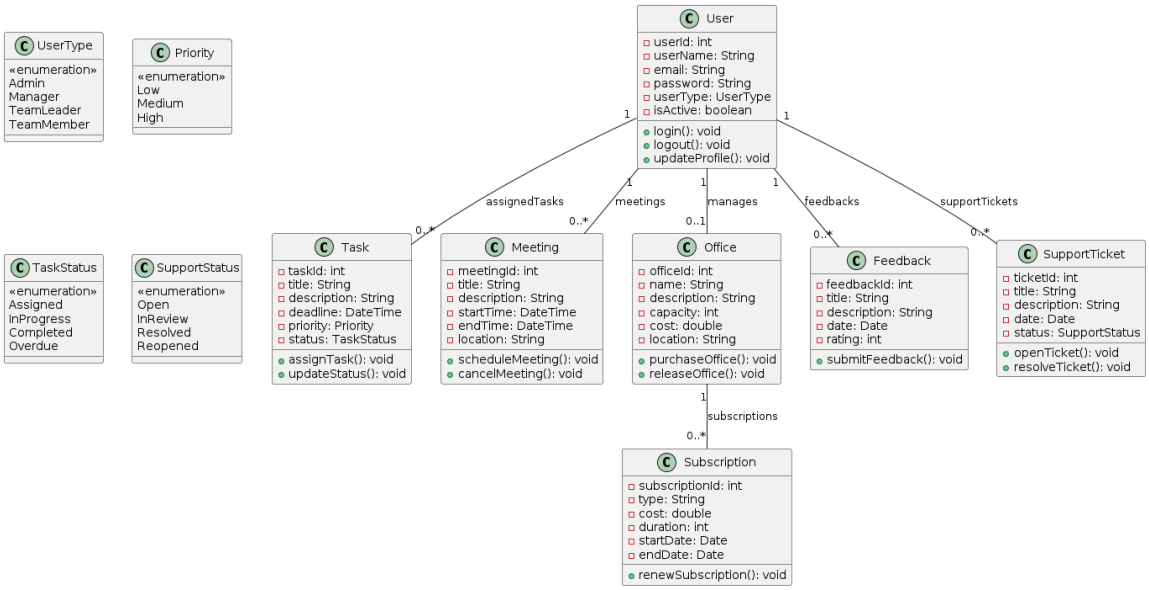


2.6.10. Class Diagram

1. **User**: Represents a system user with attributes like userId, userName, email, password, userType, and isActive. It has methods to login, logout, and updateProfile. The userType is an enumeration, indicating the role of the user within the system.
2. **UserType**: An enumeration that defines the different types of users in the system: Admin, Manager, TeamLeader, and TeamMember.
3. **Task**: Represents a task with attributes such as taskId, title, description, deadline, priority, and status. It includes methods to assignTask and updateStatus. The priority and status are enumerations, which define the importance and current state of the task.
4. **Priority**: An enumeration that categorizes tasks by their importance: Low, Medium, and High.
5. **TaskStatus**: An enumeration that indicates the current state of a task: Assigned, InProgress, Completed, and Overdue.
6. **Office**: Represents an office space with attributes such as officeId, name, description, capacity, cost, and location. It includes methods to purchaseOffice and releaseOffice.
7. **Subscription**: Represents a subscription with attributes like subscriptionId, type, cost, duration, startDate, and endDate. It has a method to renewSubscription.

Relationships:

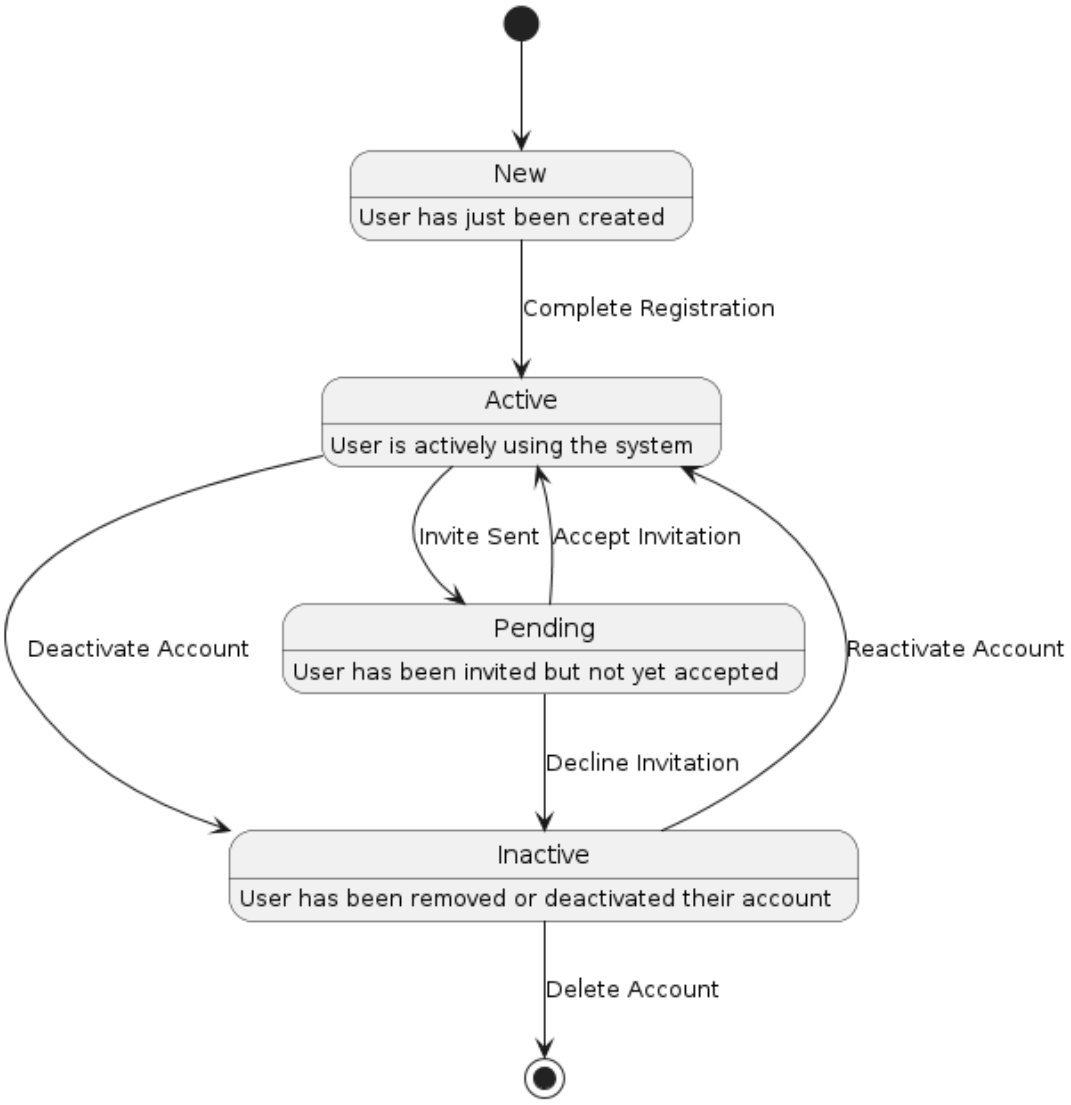
- ✓ A User can be assigned multiple Tasks and can have multiple Meetings.
- ✓ A User can manage zero or one Office and can submit multiple Feedbacks and SupportTickets.
- ✓ An Office can have multiple Subscriptions.



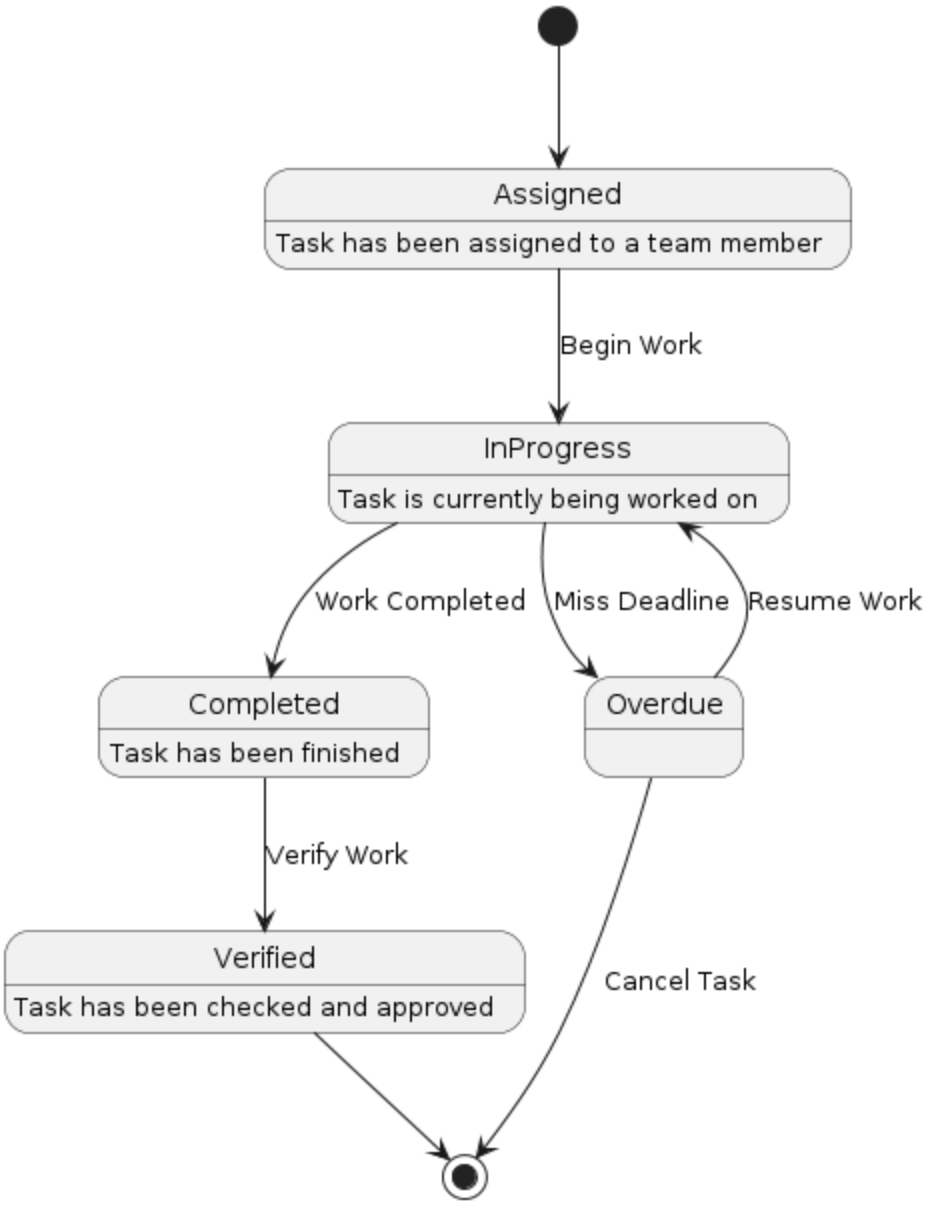
2.6.11. State chart diagram

Here are some of the key states included in our Lancer's Hub system:

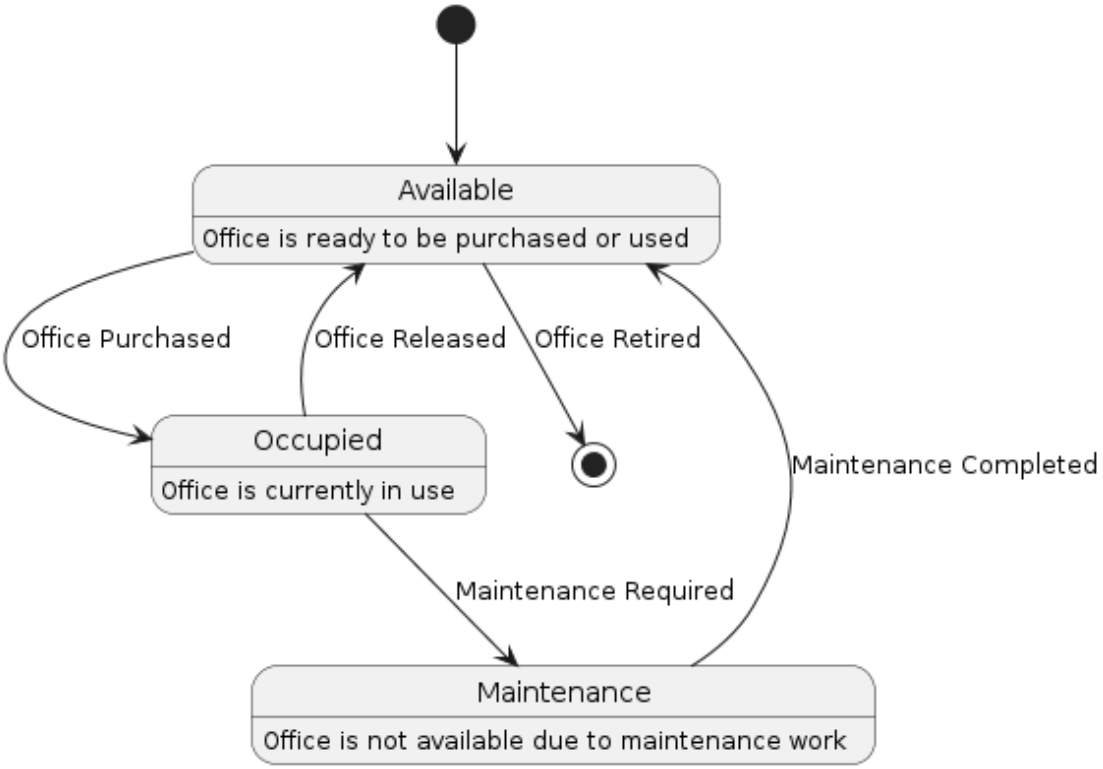
1. **User State:**
   - ✓ New: A user has just been created in the system.
   - ✓ Active: A user who has completed registration and is actively using the system.
   - ✓ Inactive: A user who has been removed or has deactivated their account.
   - ✓ Pending: A user who has been invited but has not yet accepted the invitation.



2. **Task State:**
   - ✓ Assigned: A task that has been assigned to a team member.
   - ✓ In Progress: A task that is currently being worked on.
   - ✓ Completed: A task that has been finished.
   - ✓ Overdue: A task that has not been completed by the deadline.

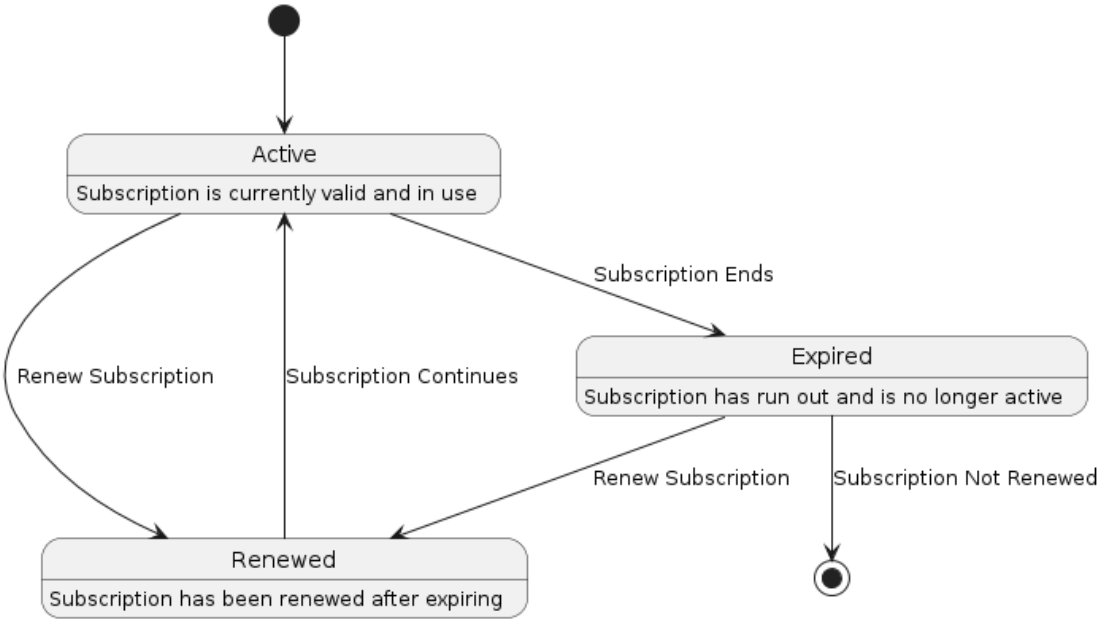


3. **Office State:**
   - ✓ Available: An office that is ready to be purchased or used.
   - ✓ Occupied: An office that is currently in use.
   - ✓ Maintenance: An office that is not available due to maintenance work.



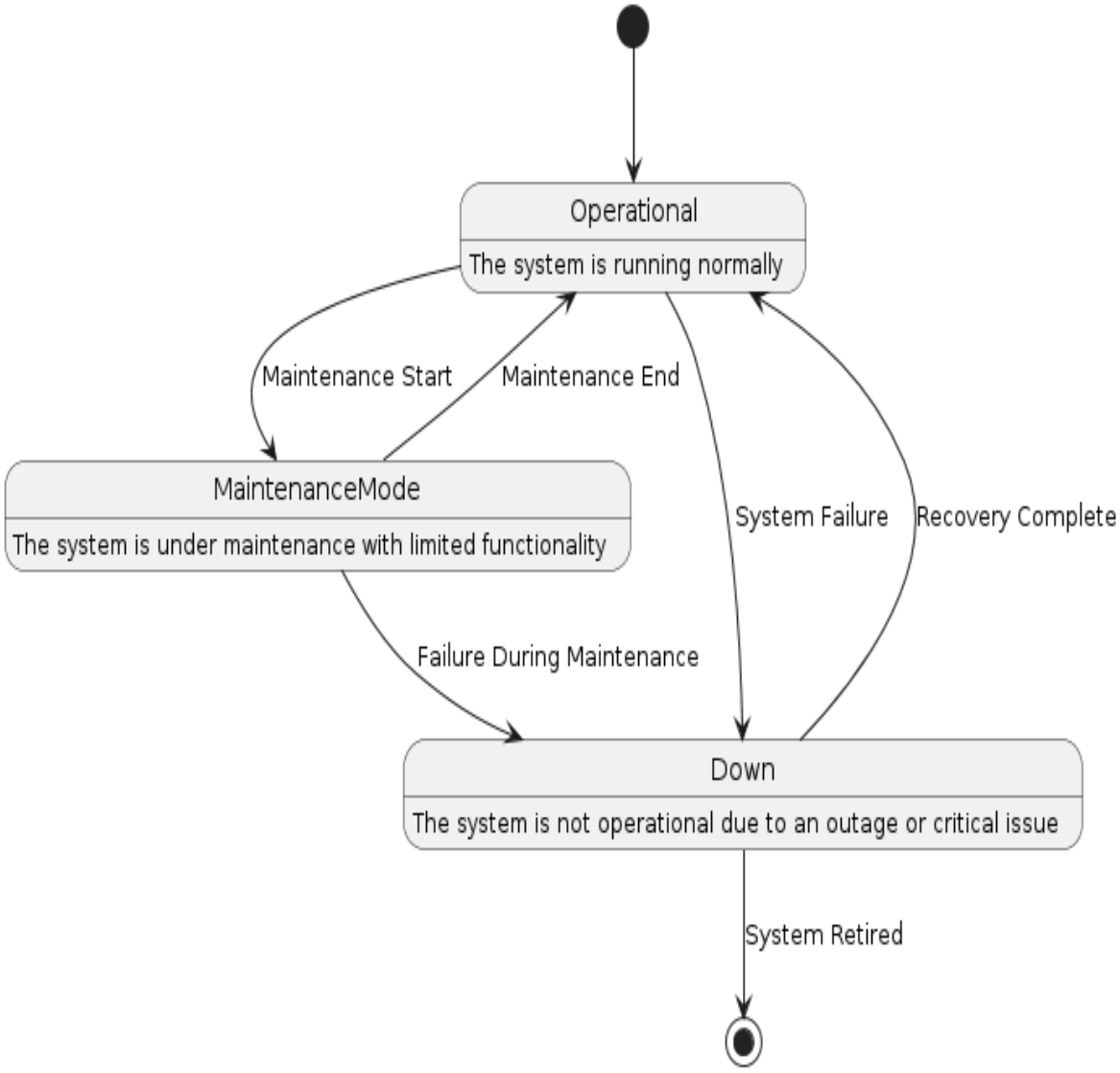
4. Subscription State:

- ✓ Active: A subscription that is currently valid and in use.
- ✓ Expired: A subscription that has run out and is no longer active.
- ✓ Renewed: A subscription that has been renewed after expiring.



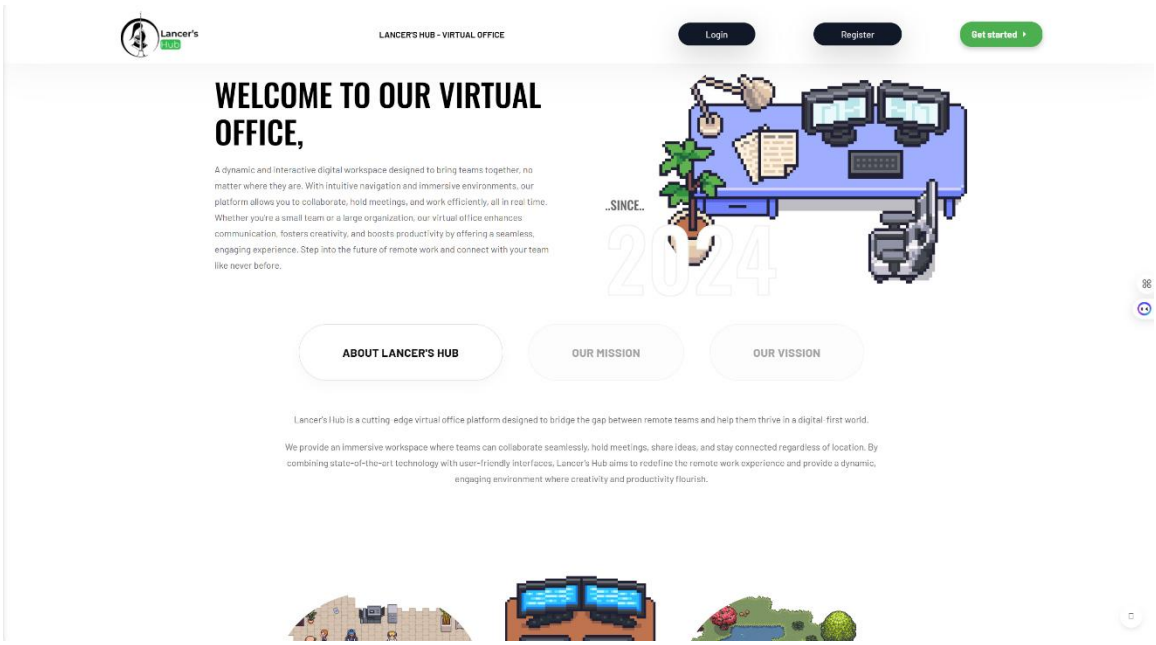
5. System State:

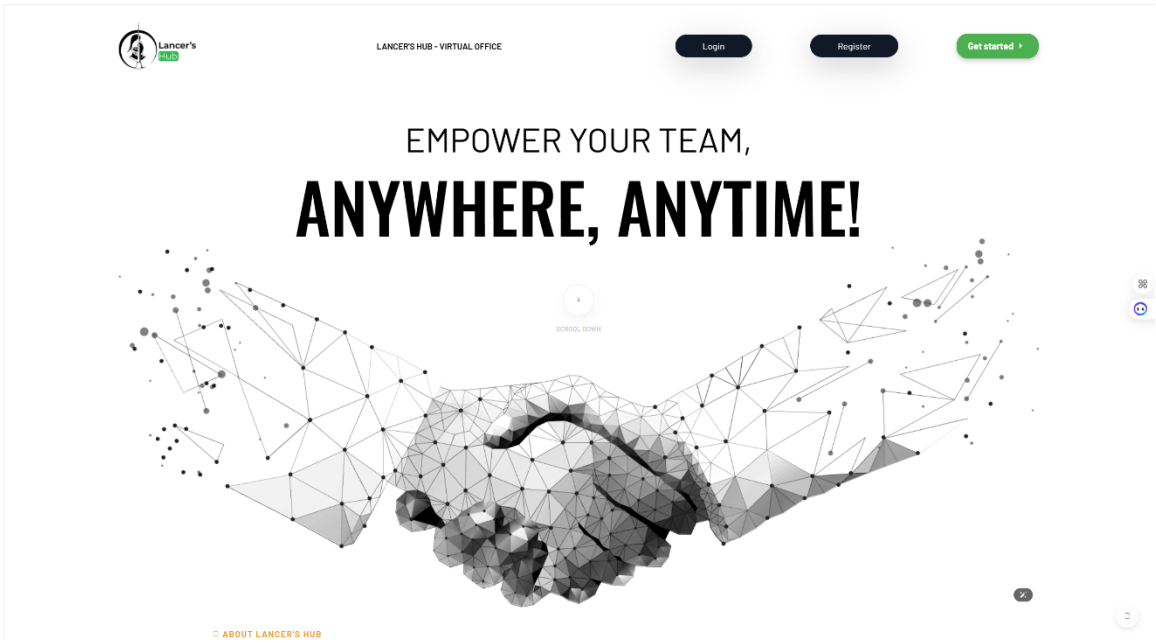
- ✓ Operational: The system is running normally.
- ✓ Maintenance Mode: The system is under maintenance and may have limited functionality.
- ✓ Down: The system is not operational due to an outage or critical issue.



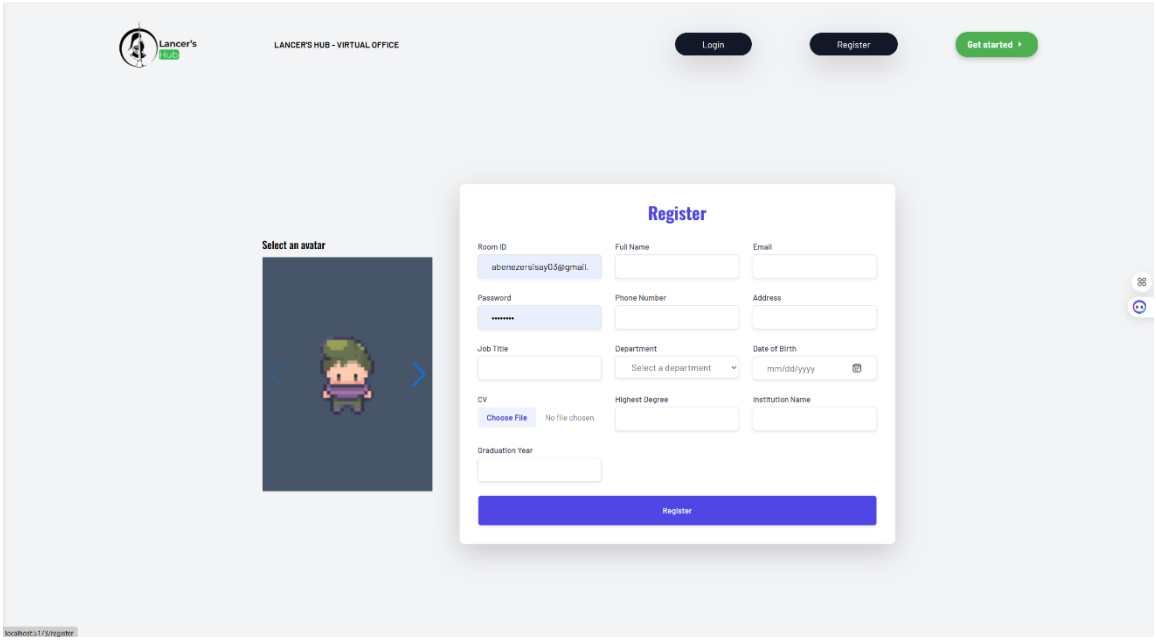
2.6.12. User Interface Prototyping

Landing Page:

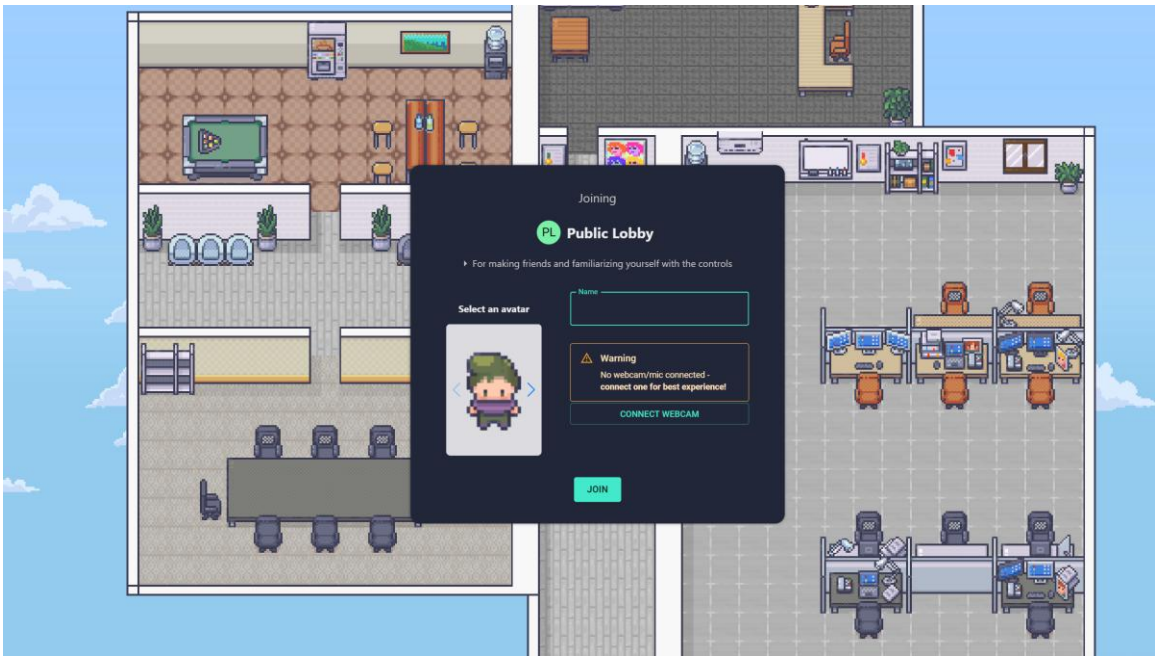




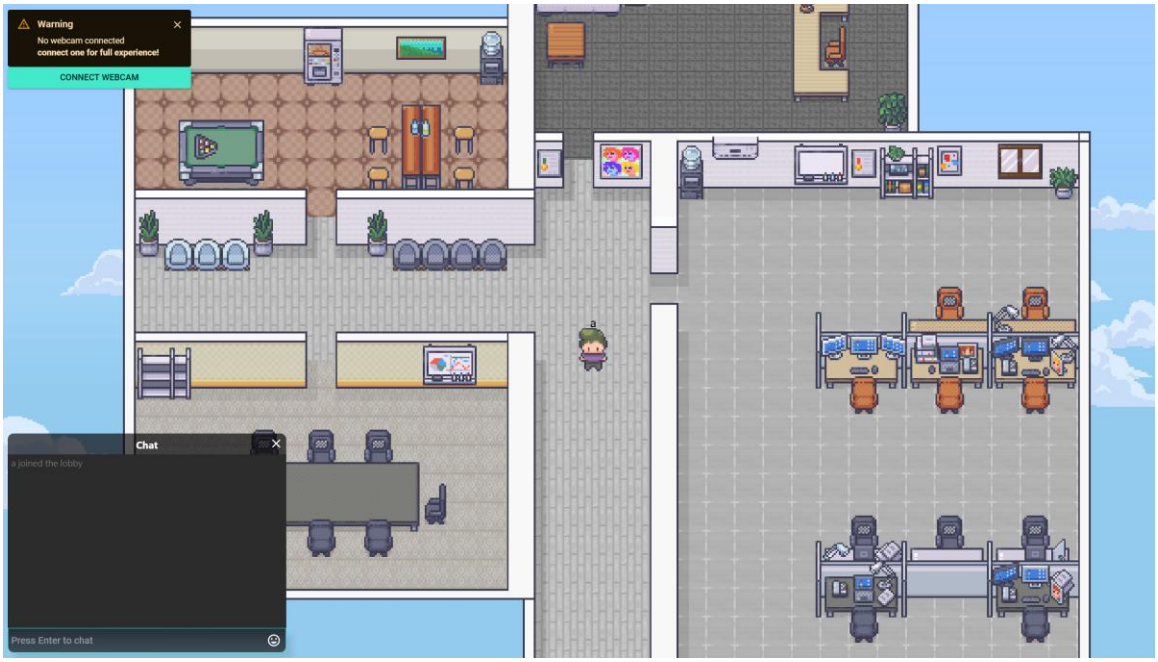
Office Registration Page:



Office Joining Page:



Office Layout:



Whiteboard layout:

Chapter 3 : System Design

3.1. Introduction

Lancers Hub relies on Firebase as its backend database for managing user data, office details, and task-related functionalities. This chapter explains the updated database schema and how it integrates with the system's components, ensuring scalability, security, and real-time synchronization.

3.2. Purpose of the System

The system aims to:

- Manage user authentication and authorization securely.
- Allow managers to purchase office rooms and distribute unique room IDs for user registration.
- Facilitate task assignment and progress tracking for seamless collaboration.
- Provide role-based dashboards with real-time updates

3.3. Design Goals

- **Data Integrity**: Ensure all user and office information is securely stored and consistently updated.
- **Real-Time Updates**: Use Firebase's real-time synchronization for immediate reflection of data changes.
- **Role-Based Access**: Implement strict role-based controls for system features and data visibility.
- **Scalability**: Accommodate growing user bases without affecting system performance.

3.4. Current Software Architecture

The architecture employs:

- **Frontend**: React for creating responsive and intuitive user interfaces.

- **Backend**: Node.js with Firebase for seamless integration and cloud-based operations.
- **Database**: Firebase Realtime Database structured for hierarchical data storage.
- **Game Engine**: Phaser3 and Colyseus for the interactive office environment.

3.5 Proposed Software Architecture

The proposed software architecture for Lancers Hub aims to create a robust, scalable, and maintainable platform that facilitates virtual office operations. This section outlines the architectural blueprint, detailing how the system's components interact to deliver a seamless user experience.

3.5.1 Subsystem Decomposition

To achieve a modular and maintainable design, the proposed architecture is decomposed into the following subsystems:

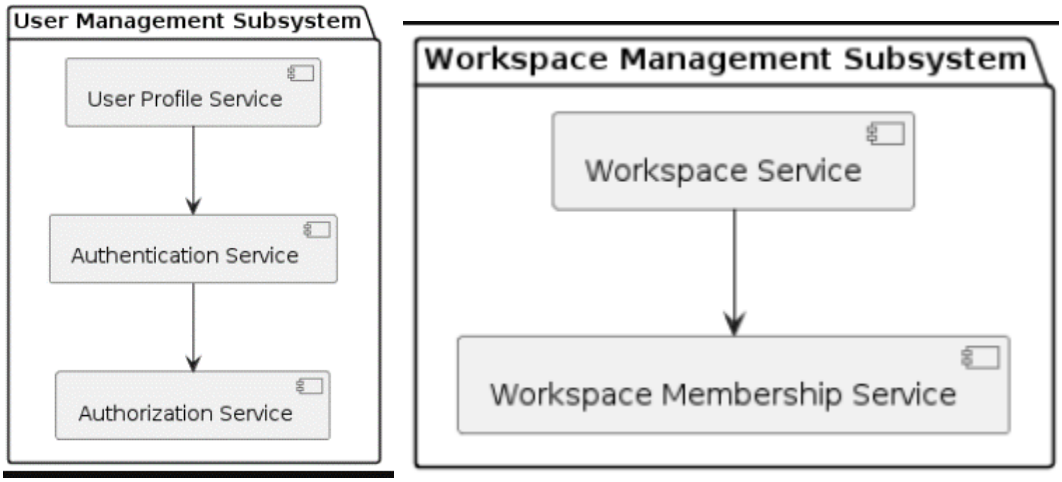
1. **User Management Subsystem**
    - **Purpose**: Handles user authentication, authorization, and profile management.
    - **Key Components**:
        - **Authentication Service**: Manages login, logout, and session handling.
        - **Authorization Service**: Enforces access control based on user roles.
        - **User Profile Service**: Manages user profile data and updates.
2. **Workspace Management Subsystem**
    - **Purpose**: Manages the creation and organization of virtual workspaces.
    - **Key Components**:
        - **Workspace Service**: Handles workspace creation, updates, and deletions.

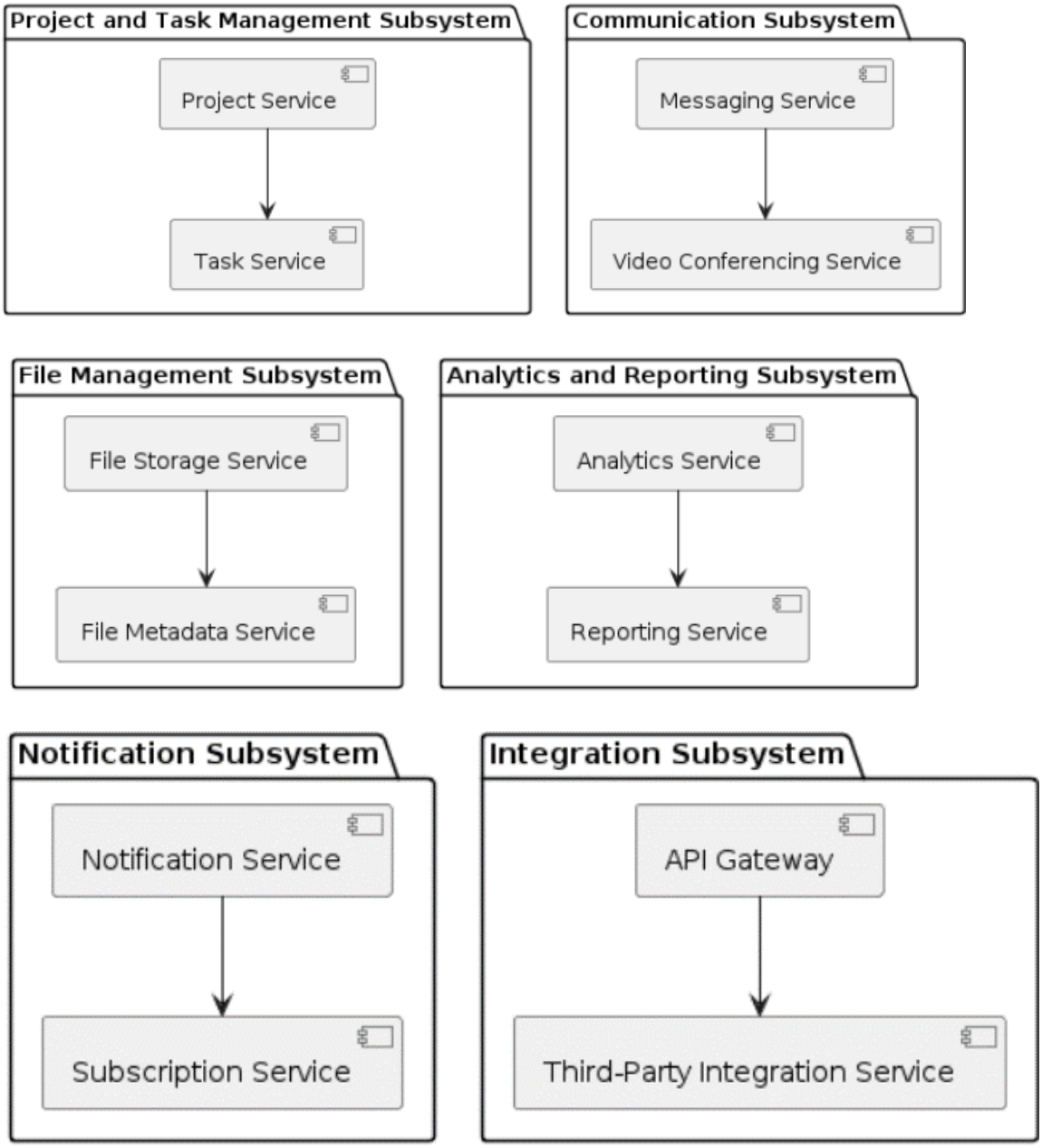
- **Workspace Membership Service**: Manages user membership and roles within workspaces.

3. **Project and Task Management Subsystem**
    - **Purpose**: Facilitates project creation, task assignment, and progress tracking.
    - **Key Components**:
        - **Project Service**: Manages project lifecycle including creation, updates, and deletions.
        - **Task Service**: Handles task assignment, status updates, and deadlines.
4. **Communication Subsystem**
    - **Purpose**: Enables real-time communication and collaboration among users.
    - **Key Components**:
        - **Messaging Service**: Manages sending, receiving, and storing messages.
        - **Video Conferencing Service**: Facilitates real-time audio and video communication using WebRTC.
5. **File Management Subsystem**
    - **Purpose**: Handles file storage, retrieval, and management.
    - **Key Components**:
        - **File Storage Service**: Manages file uploads, downloads, and storage.
        - **File Metadata Service**: Handles file metadata management and search functionality.

6. **Analytics and Reporting Subsystem**
    - **Purpose**: Provides insights into system usage and performance.
    - **Key Components**:
        - **Analytics Service**: Collects and processes usage data.
        - **Reporting Service**: Generates and delivers reports based on analytics data.
7. **Integration Subsystem**
    - **Purpose**: Facilitates integration with external systems and services.
    - **Key Components**:
        - **API Gateway**: Manages API requests and routing.
        - **Third-Party Integration Service**: Handles integration with external tools and platforms (e.g., calendar services, email).

Diagram for Subsystem Decomposition





3.5.2 Component Diagram

The component diagram for Lancers Hub illustrates the main components of the system and their interactions. This diagram provides a high-level view of the system's structure, showing how different components communicate and collaborate to deliver the overall functionality.

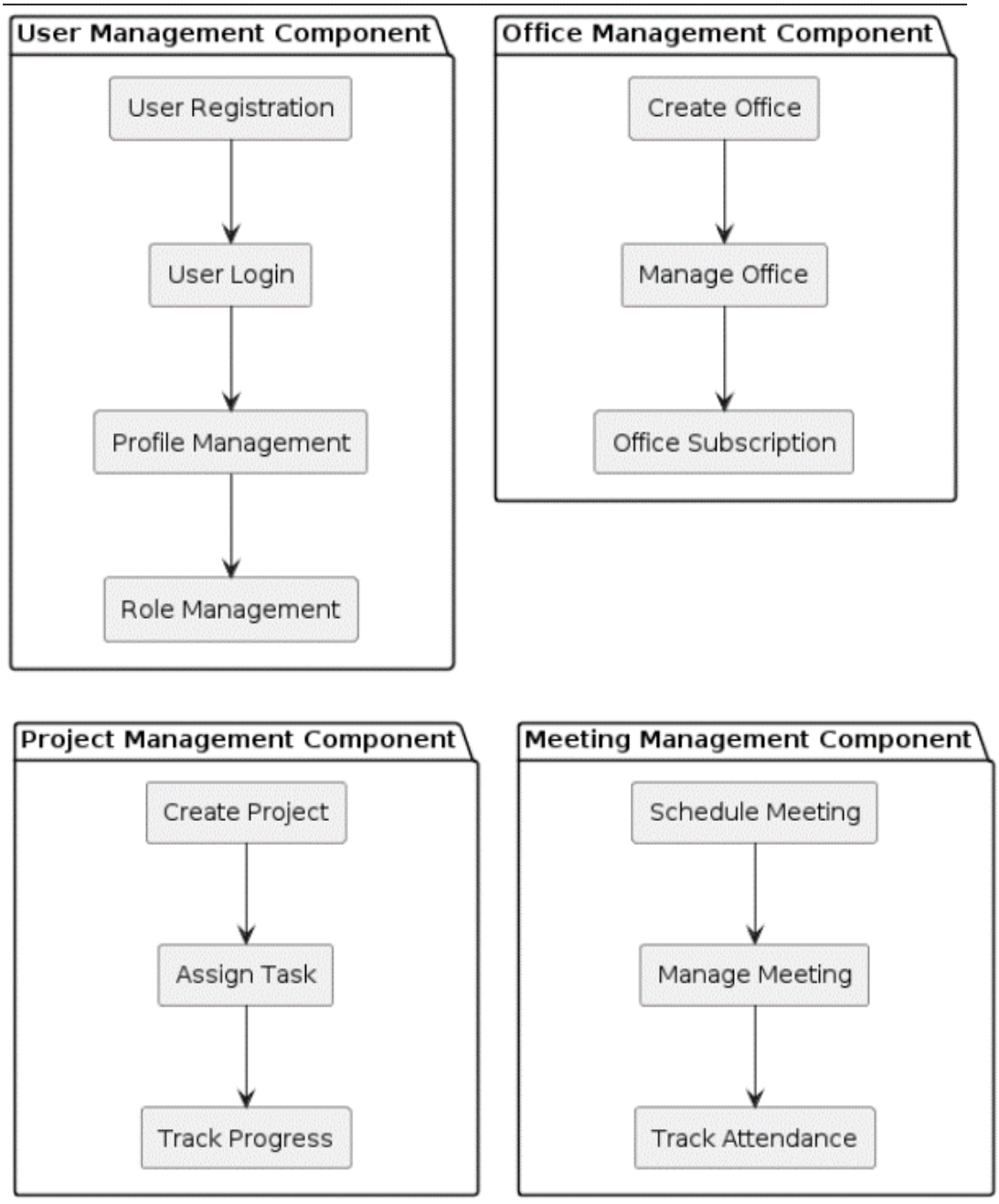
Key Components:

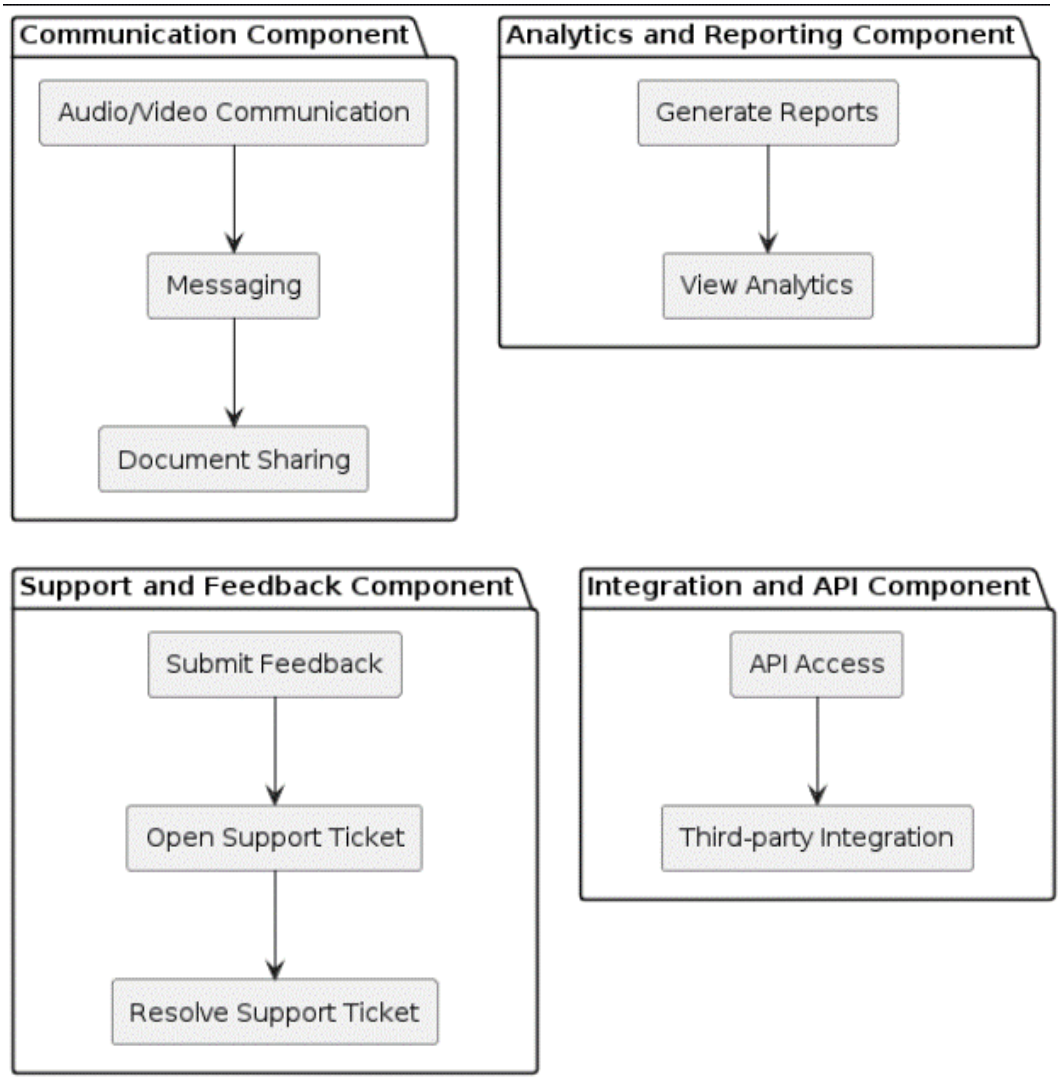
1. **User Management Component**
    - Manages user authentication, registration, profiles, and roles.
    - Interfaces:
        - User Registration
        - User Login
        - Profile Management
        - Role Management
2. **Office Management Component**
    - Manages virtual office creation, subscription, and maintenance.
    - Interfaces:
        - Create Office
        - Manage Office
        - Office Subscription
3. **Project Management Component**
    - Handles project creation, task assignment, and progress tracking.
    - Interfaces:
        - Create Project
        - Assign Task
        - Track Progress
4. **Meeting Management Component**
    - Schedules and manages virtual meetings and attendance.
    - Interfaces:
        - Schedule Meeting

- Manage Meeting
- Track Attendance

5. **Communication Component**
    - Provides real-time audio and video communication, messaging, and document sharing.
    - Interfaces:
        - Audio/Video Communication
        - Messaging
        - Document Sharing
6. **Analytics and Reporting Component**
    - Generates and displays analytics and reports based on system usage.
    - Interfaces:
        - Generate Reports
        - View Analytics
7. **Integration and API Component**
    - Facilitates integration with third-party tools and services.
    - Interfaces:
        - API Access
        - Third-party Integration

Component Diagram:





3.5.3 Deployment Diagram

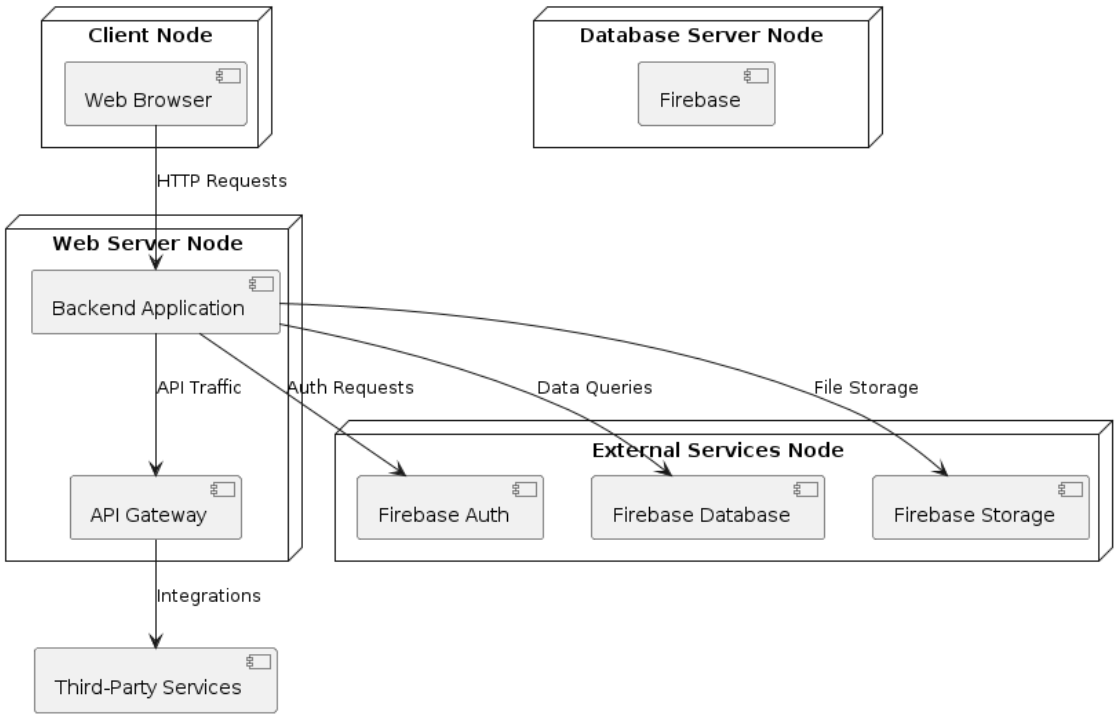
The deployment diagram for Lancers Hub illustrates the physical deployment of the software components across various nodes. It shows how different software components are distributed across hardware and network configurations. This helps in understanding the run-time processing of the application and its interactions with external services.

Key Elements of the Deployment Diagram:

1. **Client Node**
    - Represents the users' devices running the frontend application.
    - Software Components: Web Browser

2. **Web Server Node**
    - Hosts the backend services and API.
    - Software Components: Backend Application (Node.js), API Gateway
3. **Database Server Node**
    - Manages data storage.
    - Software Components: Firebase
4. **External Services Node**
    - Represents third-party services integrated with the system.
    - Services: Authentication (Firebase Auth), Real-time Database (Firebase), Cloud Storage (Firebase Storage)

Deployment Diagram:



Description:

- **Client Node:** Users interact with the system using a web browser
- **Web Server Node:** The backend application (e.g., developed in Node.js) handles business logic, and an API Gateway manages API traffic and integrations.
- **Database Server Node:** Firebase is used to manage data storage, including authentication, real-time database, and cloud storage.
- **External Services Node:** Represents external services like Firebase Auth, Firebase Database, and Firebase Storage

3.5.4 Persistent Data Management

For Lancers Hub, the persistent data management strategy involves using Firebase for database and storage solutions. This section outlines the approach to managing data persistently within the Lancers Hub system.

Key Elements of Persistent Data Management:

1. **Database Structure**
    - **Firebase Realtime Database:** Used for storing dynamic data that needs to be synchronized in real-time across multiple clients.
    - **Firebase Firestore:** A scalable and flexible database for more complex queries and hierarchical data structures.
    - **Firebase Storage:** For storing files such as documents, images, and other media.
2. **Data Models**
    - User Profiles
    - Projects
    - Tasks
    - Messages

- Meetings
- Files
- Analytics Reports

3. **Data Access Patterns**
    - **CRUD Operations:** Create, Read, Update, Delete operations on all data models.
    - **Real-time Synchronization:** Ensuring data consistency and synchronization in real-time across different clients.
    - **Security Rules:** Firebase security rules to ensure that only authorized users can access or modify data.
4. **Data Backup and Recovery**
    - Regular backups of the database and storage to prevent data loss.
    - Implementing recovery procedures in case of data corruption or loss.
5. **Data Integrity and Consistency**
    - Ensuring data integrity through validation rules and constraints.
    - Using transactions to maintain data consistency, especially for complex operations involving multiple data models.
6. **Data Privacy and Security**
    - Implementing robust security measures to protect sensitive user data.
    - Complying with data privacy regulations and ensuring user consent for data collection and processing.

Detailed Description:

1. Database Structure:

- **Firebase Realtime Database:** This is a NoSQL cloud database that stores data as JSON and synchronizes it in real-time with all connected clients. It is ideal for applications requiring real-time updates.
- **Firebase Firestore:** Firestore is a NoSQL document database that allows for more complex data structures and queries compared to the Realtime Database. It supports hierarchical data and offers powerful querying capabilities.
- **Firebase Storage:** Used for storing user-generated content such as images, documents, and other media files. It integrates seamlessly with Firebase Authentication to manage file access based on user permissions.

2. Data Models:

- **User Profiles:** Stores information about users, including their authentication details, profile information, and user settings.
- **Projects:** Contains details about the projects created within the system, including project metadata, status, and related tasks.
- **Tasks:** Stores information about tasks, including task details, assigned users, deadlines, and status.
- **Messages:** Manages communication between users, storing chat messages, timestamps, and read receipts.
- **Meetings:** Contains information about scheduled meetings, participants, agenda, and meeting notes.
- **Files:** Manages documents and media files uploaded by users, including metadata and access permissions.
- **Analytics Reports:** Stores generated reports and analytics data, used for monitoring and decision-making.

3. Data Access Patterns:

- **CRUD Operations:** Basic operations for creating, reading, updating, and deleting data are implemented for all data models.

- **Real-time Synchronization:** Leveraging Firebase's real-time capabilities to ensure data is synchronized across all clients instantaneously.
- **Security Rules:** Configuring Firebase security rules to enforce data access controls, ensuring only authorized users can access or modify data.

4. Data Backup and Recovery:

- **Regular Backups:** Implementing automated backup procedures to regularly back up the database and storage data to a secure location.
- **Recovery Procedures:** Establishing procedures to recover data in case of corruption or accidental deletion, including versioning and snapshots.

5. Data Integrity and Consistency:

- **Validation Rules:** Defining validation rules to ensure data integrity, such as required fields and data type constraints.
- **Transactions:** Using Firebase transactions to handle complex operations involving multiple data models, ensuring atomicity and consistency.

6. Data Privacy and Security:

- **Security Measures:** Implementing encryption, access controls, and secure communication channels to protect data.
- **Compliance:** Ensuring compliance with data privacy regulations like GDPR and CCPA, including obtaining user consent for data processing.

3.5.5 Detailed Database Design

3.5.5.1 Relational Tables

Although Firebase Realtime Database is NoSQL, its data structure can be represented relationally for clarity:

Table	Attribute	Data Type	Key
Users	userId	String	Primary Key
	fullName	String	
	email	String	Unique
	role	String	
	roomId	String	Foreign Key
	verified	Boolean	
Offices	roomId	String	Primary Key
	managerId	String	Foreign Key
	createdAt	Timestamp	
Tasks	taskId	String	Primary Key
	title	String	
	description	String	
	status	String	
	assignedTo	String	Foreign Key
	createdBy	String	Foreign Key

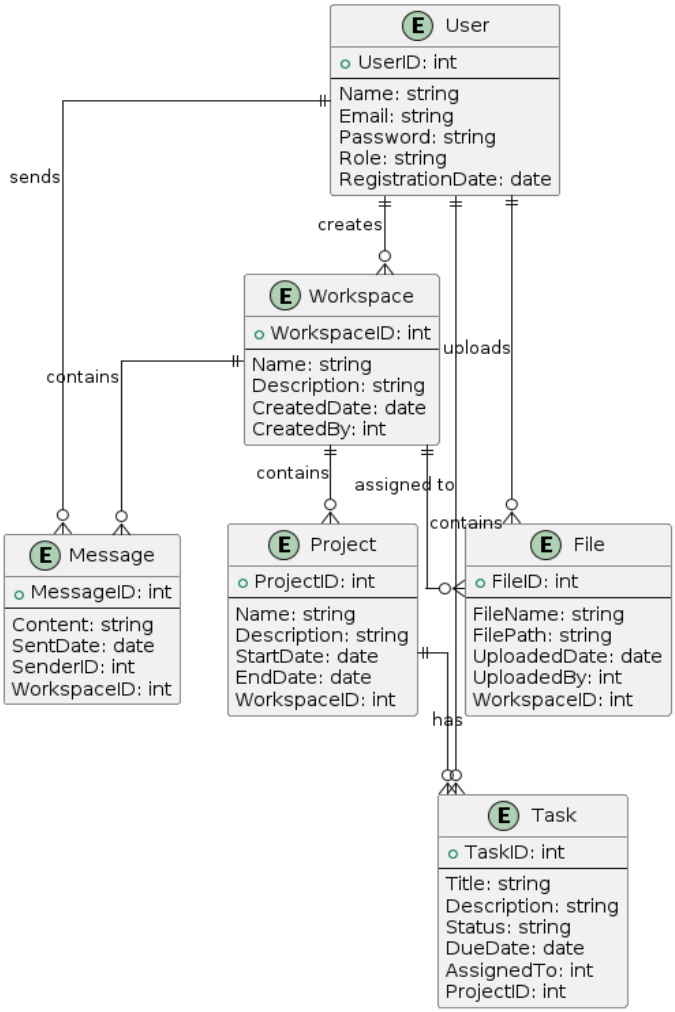
3.5.5.2 Normalization

1. **1st Normal Form (1NF)**:
    - Ensure all attributes hold atomic values.
    - Example: In the Users table, all user details are stored in individual columns.

2. **2nd Normal Form (2NF)**:
    - Eliminate partial dependencies.
    - Example: Separate user roles into a new table (Roles) and link via a foreign key.
3. **3rd Normal Form (3NF)**:
    - Remove transitive dependencies.
    - Example: Store office details (roomId, managerId, etc.) in the Offices table to avoid redundancy in the Users table.

3.5.5.3 Extended Entity-Relationship Diagram (EER)

- Relationships:



- Users → Offices (One-to-Many)
- Offices → Tasks (One-to-Many)
- Users → Tasks (One-to-Many as assignedTo)

3.5.5.4 Object-Oriented Relational Mapping (OO-Relational Mapping)

Mapping tables to objects (classes):

- **Users Table → User Class**:

```
class User {
  userId: string;
  fullName: string;
  email: string;
  role: string;
  roomId: string;
  verified: boolean;
}
```

- **Offices Table → Office Class**:

```
class Office {
  roomId: string;
  managerId: string;
  createdAt: Date;
}
```

- **Tasks Table → Task Class**:

```
class Task {
  taskId: string;
  title: string;
  description: string;
  status: string;
```

assignedTo: string;

createdBy: string;

}

3.5.6 Access Control and Security

Measures for Security:

1. **Authentication**:
    - Firebase Authentication is used to verify user identity.
    - JWT tokens secure API endpoints.
2. **Authorization**:
    - Role-based access ensures users can only perform actions assigned to their role.
    - Managers can approve users and assign tasks, while team members can only view their tasks.
3. **Data Privacy**:
    - Sensitive data like passwords are hashed using bcrypt.
    - Communication is encrypted using HTTPS.
4. **Security Practices**:
    - Input validation is performed on both frontend and backend.
    - CORS policies restrict API access to specific domains.

3.5.7 Global Software Control

Request Flow:

1. **Frontend**:
    - Users interact with React components that send requests via Axios.
2. **Backend**:

- Express.js routes handle incoming requests and validate JWT tokens.
- Firebase functions perform CRUD operations on the database.

Subsystem Synchronization:

- Firebase's real-time capabilities ensure updates propagate immediately across connected clients.

3.5.8 Boundary Conditions

Startup:

- Initialization involves loading default configurations like user roles and admin data.
- Any existing pending approvals are processed on server start.

Shutdown:

- Active sessions are logged out, and temporary data is cleared to maintain consistency.
- Tasks in progress are flagged for review on restart.

Error Conditions:

- **Database Disconnection**:
  - The system retries the connection automatically.
  - Users are notified of temporary unavailability.
- **High Traffic/Load**:
  - Firebase's scalability ensures performance remains consistent under increased user loads.

Extreme Scenarios:

- **Unusual User Behavior**:
  - Invalid inputs trigger validation errors to prevent malicious actions.

Chapter Four: Implementation

4.1 Development Tools

4.1.1 Server

4.1.1.1. Setup and Configuration:

- The server is running locally and is not deployed to any platform.
- The server uses **Node.js** with the following key dependencies:
    - express: For handling HTTP requests and middleware.
    - axios: For making RESTful API calls.
    - jsonwebtoken: For securing APIs using JWT.
    - cors: For enabling cross-origin requests.
    - dotenv: For managing environment variables.
    - bcrypt: For hashing sensitive data like passwords.
    - nodemailer: For handling email notifications.
- **Run Command**: The development server can be started using the script: npm run dev.
- **Integration with Firebase**:
- Firebase is used for authentication and as the Realtime Database for data persistence.
- JWT tokens are issued for user authentication after Firebase verification.

4.1.2 Database

- The Firebase Realtime Database was chosen for its real-time capabilities and hierarchical JSON structure.
- Data models include:
    - **Users**: Information about users, including their role, room ID, and status.

- **Offices**: Details about virtual office rooms, including manager IDs and descriptions.
- **Tasks**: Task management data, including assigned users and task progress.

4.1.2.1 Data Storage and Retrieval

- Firebase Realtime Database was used for storing hierarchical data in JSON format.
- Real-time data synchronization enabled dynamic updates to the user interface as data changed.

4.1.2.2 MVC or Other Design Pattern Followed

- The backend follows a **Model-View-Controller (MVC)** design pattern:
  - **Models**: Define the structure of data entities (e.g., users, tasks).
  - **Controllers**: Handle business logic for APIs (e.g., user registration, task management).
  - **Views**: In this case, API responses sent to the frontend.

4.1.2.3 Backend

- **Framework**: Express.js for routing and middleware management.
- **Middleware**: Body parsing (using body-parser), CORS handling (cors), and session management (express-session).

4.1.3 API Development

- RESTful APIs were developed for communication between the frontend and backend.
- Example endpoints include:
  - **POST /register**: Registers a new user using room ID validation.
  - **GET /tasks**: Fetches tasks assigned to a specific user.
  - **PATCH /task/:id**: Updates the status of a task.

4.1.4 Database Interactions

- CRUD operations were implemented as follows:
    - **Create**: Registering new users or creating tasks.
    - **Read**: Fetching user information, tasks, or office data.
    - **Update**: Updating task progress or user approval status.
    - **Delete**: Removing users or tasks (only accessible by managers).

4.1.5 ORM (If Used)

- No traditional ORM (Object-Relational Mapping) was used since Firebase operates as a NoSQL database.

4.1.6 Authentication and Authorization Mechanisms

- **Authentication**:
    - Managed through Firebase Authentication for initial login validation.
    - **JWT** (JSON Web Token) was used for session management and securing API endpoints.
- **Authorization**:
    - Role-based access control ensured that only authorized users (e.g., managers) could access certain features.

4.1.7 Session, Tokens, Cookies

- **Tokens**:
    - Access and refresh tokens were implemented using jsonwebtoken to ensure secure and scalable session handling.
- **Sessions**:
    - Short-lived sessions ensured users were logged out after inactivity.

4.1.8 Security

- Passwords were hashed using bcrypt for secure storage.

- API routes were protected with middleware that validates JWT tokens.
- Data validation was performed at both the frontend and backend to prevent malicious inputs.

4.1.9 Any Special Algorithm Used

- **Room ID Generation**:
    - Unique room IDs were generated for virtual offices using randomized alphanumeric strings.
- **Data Synchronization**:
    - Firebase's built-in real-time syncing ensured task and user updates propagated instantly across connected clients.

4.1.10 Third-Party API Integration

- **Nodemailer** was used for sending user verification emails and notifications.

4.1.11 Payment Service

- Payment services were integrated into the current system, Chapa test mode payment method was integrated.

4.1.12 Email

- User registration and confirmation is triggered using the **Nodemailer** library.

4.1.13 Frontend

- The frontend is built using **React** and styled with **Material-UI**.
- Key dependencies include:
    - react-beautiful-dnd: For drag-and-drop functionality in task management.
    - react-intl: For internationalization support.
    - react-toastify: For user notifications and alerts.

4.1.14 One Example for Few Components or One Page

4.1.14.1 Frontend Tasks

- **Filtering**: Tasks displayed in the dashboard are filterable by status (e.g., "In Progress," "Completed").
- **Front-End Security**:
    - All API requests include a JWT token in the authorization header.
- **Routing**: Implemented using react-router-dom for secure role-based navigation.
- **Responsiveness**: The layout adjusts seamlessly across devices, ensuring usability on both desktops and tablets.

4.1.14.2 API Integration

- The frontend communicates with the backend using axios.
- Example: The **Dashboard** component fetches tasks via the /tasks API and displays them in a paginated table.

4.1.15 Game Engine

- The 2D interactive office is built using **Phaser3** for rendering and **Colyseus** for managing multiplayer real-time interactions.
- Key features include:
    - Real-time chat.
    - Screen sharing and whiteboard collaboration.
    - Proximity-based video and audio interactions.

4.2 Prototype of the System

4.2.1 Actor 1: Manager

- The manager can:
    - Purchase a virtual office room and generate a unique room ID.
    - Approve or decline user registration requests.

- Assign roles (Team Leader or Team Member).
- Monitor task progress through a dashboard.

4.2.2 Actor 2: Team Leader

- The team leader can:
  - Register using a provided room ID.
  - Assign tasks to team members.
  - Track task progress and update statuses.

4.2.3 Actor 3: Team Member

- The team member can:
  - Register using a provided room ID.
  - View tasks assigned to them and update progress.
  - Collaborate in the virtual office.

4.3.Unit Testing

4.3.1. Tools Used:

- Manual testing was conducted during development for each feature.
- Debugging tools such as nodemon were utilized to monitor server-side changes.

4.3.2. Sample Unit Tests:

- Registration validation with and without a valid room ID.
- Task assignment and status updates.

4.4 Integration Testing

4.4.1. Integration Points:

- Firebase Authentication with the Node.js backend for user validation.

- Frontend integration with RESTful APIs for user and task management.

4.4.2 Tools Used

- Manual testing using Postman to test API endpoints and verify database updates.

4.5 System Testing

- Conducted by running the entire system locally, simulating manager, team leader, and team member workflows.
- Verified task management, role assignment, and the interactive office functionality.

4.6 Security Testing

- **JWT Tokens**: Used to secure all APIs, ensuring only authenticated users can access restricted resources.
- **Password Hashing**: User passwords are hashed using bcrypt to enhance security.

4.7 End-to-End Testing

- Testing the full workflow:
    1. Manager purchases a room and registers.
    2. Team members and leaders register using the room ID.
    3. Manager approves users and assigns roles.
    4. Tasks are created, assigned, and tracked.
    5. Users access the virtual office for collaboration.

4.8 Test Cases

Comprehensive Test Cases Table

Test Case ID	Scenario	Steps	Expected Outcome
TC001	User registration without a room ID	User leaves the "Room ID" field blank during registration.	An error message is displayed, and registration is not allowed.
TC002	User registration with an invalid room ID	User enters a non-existent or incorrect room ID during registration.	Error message "Invalid Room ID" is displayed.
TC003	Manager approval of a user	Manager approves a pending user.	The user’s status is updated to "Approved" and they gain access to their dashboard.
TC004	Team leader assigns a task to a team member	Team leader assigns a task and sets a deadline.	The task appears in the team member’s task list with the correct deadline.
TC005	Real-time chat in the virtual office	Two users exchange messages in the virtual office chat.	Messages are displayed instantly for both users.
TC006	Task progress update by team members	Team member updates a task's status from "In Progress" to "Completed."	The status change reflects instantly on the manager and team leader dashboards.

Test Case ID	Scenario	Steps	Expected Outcome
TC007	Database fetch error during user registration	Registration form attempts to retrieve room ID validation data but fails.	Error message is logged, and registration retries or prompts the user to try again.
TC008	Characters fail to load in the office part	Users join the virtual office but characters fail to display.	Placeholder graphics are shown, and the issue is logged for further debugging.
TC009	Task not saving properly	Team leader creates a task, but the task does not persist in the database.	Error logged; UI displays a retry option. Task is not added to the user dashboard.
TC010	Whiteboard connection fails	A user tries to connect to the whiteboard, but the action fails due to a server issue.	User receives an error, and an alert is logged.
TC011	Unauthorized API access	A user without a valid JWT token tries to access a protected API endpoint.	Request is denied with a 401 Unauthorized error response.
TC012	Real-time data synchronization	A task is created or updated by the manager.	All connected dashboards display the changes in real-time.
TC013	User logout scenario	A user logs out from the system.	Session tokens are invalidated, and the user is

Test Case ID	Scenario	Steps	Expected Outcome
			redirected to the login screen.
TC014	Screen sharing in the virtual office	A user starts screen sharing, and another user views it.	Screen sharing starts seamlessly without lag.
TC015	Database connection lost mid-session	The database disconnects while users are active.	Users are notified, and the system attempts to reconnect automatically.

4.9 Detailed Examples of Test Cases

Test Case: Database Fetch Error During User Registration

- **Scenario**: A user tries to register with a valid room ID, but the system fails to fetch room details due to a database connection error.
- **Steps**:
    1. Open the registration form.
    2. Enter valid user details and a valid room ID.
    3. Click the "Register" button.
- **Expected Outcome**:
    - Error message is logged: "Unable to fetch room details."
    - The registration form displays an error, and the user is prompted to retry.
    - Retry logic attempts to reconnect to the database.

Test Case: Characters Fail to Load in the Office Part

- **Scenario**: Users enter the virtual office, but their avatars fail to display due to asset-loading issues.
- **Steps**:
    1. User clicks "Go to My Room" from their dashboard.
    2. The virtual office environment loads.
    3. Avatars fail to appear on the screen.
- **Expected Outcome**:
    - Placeholder characters (e.g., generic gray icons) appear temporarily.
    - The system retries loading assets in the background.
    - A log entry is created for debugging: "Character asset failed to load for session ID [xxx]."

Test Case: Real-Time Data Synchronization

- **Scenario**: A manager assigns a task to a team leader, and all relevant dashboards should reflect the change instantly.
- **Steps**:
    1. Manager assigns a new task to a team leader.
    2. Team leader's dashboard is open in a different session.
    3. Observe the dashboard for real-time updates.
- **Expected Outcome**:
    - The task appears on the team leader’s dashboard instantly without requiring a refresh.
    - Firebase database reflects the change correctly.

Test Case: Unauthorized API Access

- **Scenario**: A user without a valid JWT token tries to access a protected endpoint for task data.
- **Steps**:
    1. Use an API testing tool (e.g., Postman) to call a protected API without including a JWT token in the request header.
    2. Observe the server's response.
- **Expected Outcome**:
    - API returns a 401 Unauthorized error response with a message: "Authentication required."
    - The request is not processed, and no data is exposed.

Test Case: Screen Sharing in the Virtual Office

- **Scenario**: A team member initiates screen sharing in the virtual office, and another team member joins the session.
- **Steps**:
    1. Team member clicks "Share Screen" in the virtual office.
    2. Another team member clicks to join the session.
    3. Observe the screen-sharing session for both users.
- **Expected Outcome**:
    - The shared screen displays without lag for the second user.
    - A notification is sent to all connected users: "User [name] started screen sharing."
    - Screen sharing stops gracefully when the session ends.

Test Case: Database Connection Lost Mid-Session

- **Scenario**: The database connection drops while users are active in the system.

- **Steps**:
    1. Simulate a database disconnection during active sessions.
    2. Observe how the system handles the event.
- **Expected Outcome**:
    - Users are notified: "Connection lost. Attempting to reconnect..."
    - The system retries connecting to the database automatically.
    - Data changes made during the disconnection are queued and synced once the connection is restored.

Chapter Five: Conclusion and Recommendation

5.1 Conclusion

The development of the Lancers Hub virtual office system has successfully addressed key challenges faced by freelancers, startups, and remote teams in Ethiopia. By combining a task management platform with an interactive 2D virtual office environment, the system delivers a comprehensive and user-friendly solution.

Key achievements of the system include:

- **Task Management**: Managers and team leaders can assign, track, and manage tasks effectively, ensuring clear accountability and streamlined workflows.
- **Role-Based Access Control**: The system ensures secure role-based access for managers, team leaders, and team members, maintaining data integrity and functional separation.
- **Interactive Collaboration**: The virtual office facilitates real-time communication, screen sharing, whiteboarding, and proximity-based interactions, enhancing team engagement.
- **Scalability and Security**: Using Firebase Realtime Database, the platform is designed for scalability, ensuring real-time synchronization of data while maintaining robust security protocols with JWT tokens and password hashing.

Despite being a local setup at this stage, the system has demonstrated its potential to improve productivity, collaboration, and resource management for remote and distributed teams. This project lays a strong foundation for further enhancements and scalability in the future.

5.2 Recommendations

For Immediate Enhancements

1. **Database Optimization**:
    - Resolve the occasional issues with data fetching and synchronization by refining database queries and implementing robust error handling.

2. **Deployment**:
    - Deploy the system to a cloud platform (e.g., Firebase Hosting or AWS) for broader accessibility and testing across multiple environments.
3. **User Feedback Integration**:
    - Conduct usability testing with actual freelancers, startups, and remote workers to gather insights and refine the platform's features.

For Future Developments

1. **Advanced Analytics**:
    - Integrate analytics to track user activity, task performance, and overall productivity. Insights can drive improvements and user engagement.
2. **Third-Party Integrations**:
    - Include integrations with tools like Google Calendar, Slack, or Trello to enhance task and time management.
3. **Cross-Platform Compatibility**:
    - Expand platform support to include mobile applications for iOS and Android, providing flexibility for users on the go.
4. **AI-Assisted Features**:
    - Introduce AI-powered task recommendations, virtual meeting assistants, and sentiment analysis for better collaboration insights.
5. **Improved Testing Framework**:
    - Implement automated testing pipelines (e.g., with Jest, Cypress) to ensure the system remains reliable and robust during further development.

For Business Scalability

1. **Monetization Options**:

- Introduce premium features, such as custom office designs or larger team support, for paid tiers.
- Offer subscription-based pricing for startups and enterprises.

2. **Localization and Accessibility**:
    - Provide localized language support to cater to diverse user groups.
    - Ensure compliance with accessibility standards (e.g., WCAG) to make the platform inclusive.
3. **Marketing and Outreach**:
    - Create a marketing strategy targeting freelancers, startups, and organizations transitioning to remote work to drive user adoption.

By implementing these recommendations, Lancers Hub can grow into a leading virtual office platform, not only in Ethiopia but also in global markets. Its ability to combine efficient task management with immersive collaboration makes it a unique solution tailored to the evolving needs of remote work.

Appendix

A.1 Data Collection Tools

The following tools were used during the requirement-gathering phase:

1. **Survey/Questionnaire**:
    - Distributed to freelancers, startups, and remote workers to understand their challenges.
    - Sample questions included:
        - "What are your main difficulties in managing tasks remotely?"
        - "What features would you expect in a virtual office platform?"
2. **Interview Questions**:
    - Conducted with potential users, asking:
        - "How do you currently manage team collaboration?"
        - "What specific tools or features do you find effective?"
3. **Observation**:
    - Monitored existing solutions like Gather.town to evaluate usability and gaps.

A.2 System Development Artifacts

1. **Prototype Screenshots**:
    - Login Page
    - Task Dashboard
    - Interactive Office
2. **Database Schema**:
    - Firebase Realtime Database structure as described in the design chapter.

3. **Use Case Models**:
    - Diagrammatic representations of key system functionalities.

A.3 Testing Tools

1. **Manual Testing**:
    - Scenarios for user registration, task assignment, and virtual office interactions.
2. **Postman**:
    - Used for API testing and validating database interactions.
3. **Browser Developer Tools**:
    - Debugging frontend issues and monitoring network requests.

A.4 User Guide

Steps to Use the System:

1. **For Managers**:
    - Purchase an office and share the room ID with team leaders and members.
    - Approve users and assign roles/tasks.
2. **For Team Leaders**:
    - Register using the room ID and start assigning tasks to team members.
3. **For Team Members**:
    - Register with the room ID and complete tasks assigned by team leaders.

References

1. **Frameworks and Tools**:
    - React: https://reactjs.org/
    - Node.js: https://nodejs.org/
    - Firebase: https://firebase.google.com/
    - Phaser3: https://phaser.io/phaser3
    - Colyseus: https://colyseus.io/
2. **Libraries Used**:
    - Axios: https://axios-http.com/
    - Express.js: https://expressjs.com/
    - Material-UI: https://mui.com/
3. **Documentation Sources**:
    - Agile Development Guide: https://www.agilealliance.org/
    - Gather.town (Competitor Analysis): https://www.gather.town/
4. **Code References**:
    - GitHub Repository (SkyOffice): https://github.com/kevinshen56714/SkyOffice