COMP5047

Applied Software Engineering: Coursework

Semester 1 2025-2026:1

Table of Contents

Software Development of a Modern Computer Application

Introduction
This document explores the design and development analysis of a Society Leader App subsystem within a modern computer application. The subsystem is designed to support the following requirement domains: FR-UO-3: Management of Societies, 4.2. Requirements of Society Leaders, and FR-SL-2: Management of events.

Together, these requirements form the foundation for the subsystem's architectural design, functional modeling, and detailed implementation strategies.

Task 2 - Analysis and Specify Software Quality Requirements

This task identifies and defines the quality requirements essential to the Society Leader App subsystem. It covers security and privacy protection (both functional and non-functional), performance, reliability, and scalability.

Security and Privacy Protection

Determines the extent to which a software system prevents access to data and information by individuals or other systems that have not the proper level of access based on their types and authorization levels.

Functional Requirements	Description
Mobile Application Architecture	1. A standard architecture shall be developed based on prescribed sets of security principles, rules, techniques, processes, and patterns to develop a secure mobile application. 2. The entire development of the mobile application must center around architecture principles, which can be revised according to insights gained throughout the development of application layers, as well as operational usage and user feedback.
User Authentication and Authorization	1. App owners must guarantee that explicit consent is obtained in a convenient manner prior to permitting the registration of a mobile application. 2. A login authentication and a risk-based financial-value-based transaction authentication must be established. 3. App owners are required to ensure that the initiation of mobile payments, along with access to sensitive payment and personal, is safeguarded by robust authentication mechanisms, which includes: ○ Implementation of Multi-Factor Authentication (MFA) for registration of a mobile app user account. ○ Strong and configurable password or pin or a biometric credential such as face recognition or fingerprint recognition. ○ Time-based one-time password (TOTP) for authentication. ○ OTP-fetching functionality. ○ Configure maximum number of failed authentication attempts which access to the mobile application's services will be denied.
Protection on Sensitive Payment Data	1. App owners are required to ensure that sensitive information is not stored in a shared store segment alongside other applications, but rather utilize the device internal storage, which is effectively sandboxed, or preferably a container app

and User Information	that does not interfere with other applications or security configurations of the devices.
App Vulnerability Assessment, Patching, and Updating	1. App owners must guarantee that the application has undergone thorough and iterative vulnerability assessment, scan and intrusion tests to identify weaknesses within the application, conducted by internal and external evaluators.

Performance

Illustrates how well the application operates in relation to the quantity of resources it uses when operating under specific conditions.

Functional Requirements	Description
Membership & User Management	1. **Onboarding & Verification:** Identity verification and approval processes for society members. ***Example****: Verifying identities of society leaders before granting access to core functionalities and sensitive information in the mobile application.* 2. **Member Directory:** Accessible directory with easy access to contact information, student profiles, and membership status. ***Example****: Reaching out to inquire additional information from a student seeking to establish a new society via contact information (student email address, mobile phone number).*
Communication & Notification	1. **Real-time Messaging:** Instant chat supporting private, group, and committee channels. ***Example****: A leader can privately message a society member or discuss within a committee group.* 2. **Push Notification:** Push alerts and emails for events, internal societal affairs, or maintenance scheduling for registered application users. ***Example****: Notify members about a new event or emergency alert via push notifications and emails.* 3. **Announcements & News Feeds:** Society leaders can send announcements and new feeds, optionally attach documents, links, or multimedia. ***Example****: Post a reminder for an upcoming society meeting with attached agenda documents.*
Event & Facility Management	1. **Event Creation & Scheduling:** Leaders can detail events with information including name, theme, organizing team, date/time, venue, scope (public/private), and description ***Example****: Schedule an annual society seminar*. 2. **Event Promotion & Reminders:** Leaders can promote upcoming events by sharing reminders or countdowns within the application. Automated reminders can be sent to registered attendees before an event.

	__Example:__ Send a reminder notification 24 hours before an annual gala event to RSVP-confirmed members, including time and venue. 3. **RSVP & Attendance Tracking:** Members can confirm their attendance and leaders can monitor responses. On-site or QR code can be used for quick check-in, and attendance data can be exported for records. ***Example**: Track attendance at a workshop and generate attendance reports.*
Non-functional Requirements	**Description**
Responsive Times	1. Loading the main dashboard, responding to user inputs, and data retrieval for lists should be accomplished within 2 seconds on standard devices under normal network conditions (Muthineni, 2025).
Cross-platform Compatibility	1. Ensure the app functions smoothly across multiple operating systems and device types to maximize accessibility.
Network Efficiency	1. The app should use data compression and caching strategies to reduce data transfer size, and implement offline capabilities for viewing cached data, syncing updates once reconnected.

Reliability

The extent to which a software system performs a specific function under predetermined conditions for a predetermined amount of time.

Functional Requirements	**Description**
Availability	1. Use dependable server and hosting solutions to guarantee a high uptime for the application and its backend services. 2. To prevent overload and guarantee availability during periods of high usage, use load balancing to distribute traffic evenly across servers (Deepak, 2024).
Robustness	1. Strong input validation is essential to avoiding crashes or unexpected behaviour brought on by erroneous data. 2. To manage error handling effectively and maintain the app's stability, implement thorough exception handling. 3. To avoid memory leaks and ensure seamless operation, take care of memory management, especially in mobile environments (Deepak, 2024). ***Example**: Post a reminder for an upcoming society meeting with attached agenda documents.*
Fault Tolerance	1. Describe errors to users in a non-technical, understandable, and friendly manner and offer guidance on what actions to take. 2. To facilitate analysis and debugging, implement thorough error and exceptions logging. Error logs can be captured and analysed using tools such as Sentry, Crashlytics, and Loggly.

	3. Design the application to handle errors without disruption to usability for users, even if some features are unavailable (NetGuru, n.d.).

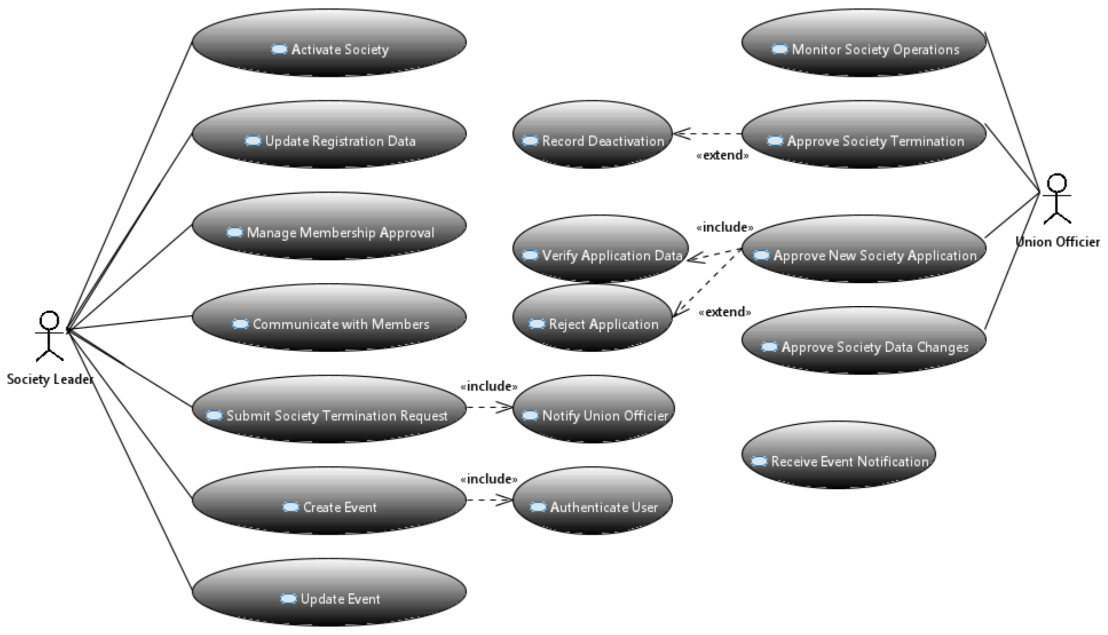
Scalability The capacity of the system to support growth or to handle an increasing volume of work without compromising performance.	
Functional Requirements	**Description**
Horizontal Scalability	1. Design the backend architecture to add additional servers or resources as per user increases for workload distribution. 2. Utilize cloud services such as AWS, Azure, or Google Clouds to dynamically allocate resources during peak usage and prevent the application from
Modular Design	1. Flexible architecture that allows for implementation of additional features without affecting performance or user experience through modular component design. 2. A microservices architecture enables independent scaling of different application sections, providing greater flexibility and easier management (NetGuru, n.d.).
Performance Maintenance	1. Monitor system performance continuously using analytical tools such as New Relic, DataDog, or Prometheus (Miguel, 2025). 2. To provide a satisfactory user experience is for the application to remain responsive, with fast load times and smooth operations.

Task 3 - Specification and Modelling Software Functional Requirements

This task models the functional requirements of the subsystem using a use case diagram and activity diagram. It highlights workflows between Union Officers and Society Leaders, focusing on three requirement domains:

- FR-UO-3: Management of Societies.
- 4.2 Requirements of Society Leaders.
- FR-SL-2: Management of events.

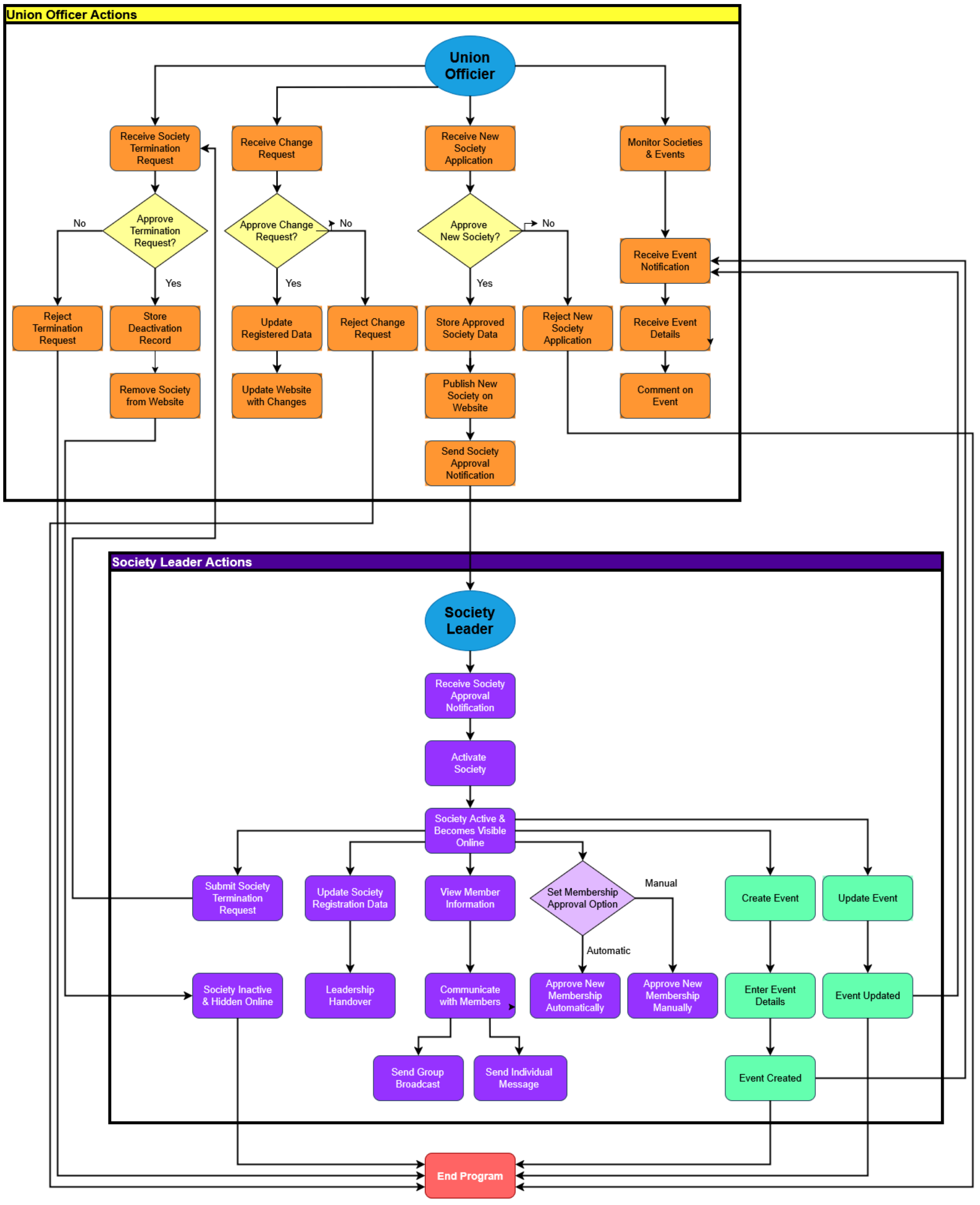
Use Case Model



The activity diagram below illustrates multiple selected use cases of a Society Leader Application. These use cases are colour-coded to facilitate identification of each use cases and highlight the relationships between two entities—the Union Officer and the Society Leader:

- Orange : FR-UO-3: Management of Societies.
- Purple : 4.2 Requirements of Society Leaders.
- Green-Cyan : FR-SL-2: Management of events.

Activity Diagram



Union Officer Actions
Union Officer
Receive Society Termination Request
Receive Change Request
Receive New Society Application
Monitor Societies & Events
Approve Termination Request?
No
Yes
Approve Change Request?
No
Yes
Approve New Society?
No
Yes
Receive Event Notification
Reject Termination Request
Store Deactivation Record
Update Registered Data
Reject Change Request
Store Approved Society Data
Reject New Society Application
Receive Event Details
Remove Society from Website
Update Website with Changes
Publish New Society on Website
Comment on Event
Send Society Approval Notification
Society Leader Actions
Society Leader
Receive Society Approval Notification
Activate Society
Society Active & Becomes Visible Online
Submit Society Termination Request
Update Society Registration Data
View Member Information
Set Membership Approval Option
Manual
Automatic
Create Event
Update Event
Society Inactive & Hidden Online
Leadership Handover
Communicate with Members
Approve New Membership Automatically
Approve New Membership Manually
Enter Event Details
Event Updated
Send Group Broadcast
Send Individual Message
Event Created
End Program

Task 4 - Software Architectural Design

This task defines the microservices architecture of the subsystem. The subsystem architecture and UML component diagram demonstrate how services interact with databases and various components.

The tables below describe the internal services of the Society Leader App subsystem, showcasing their dependencies, methods, and parameters.

Name	**Communication Service**
Description	Provides messaging facilities between society leaders, members, and union officers.
Stereo Type	Microservice
Depenedency	Notification service, MongoDB Communication DB, Message Broker

Method	Parameters	Description
sendMessage	senderId: String, recipientId: String, content: String	Sends a direct message between two users.
broadcastMessage	societyId: UUID, message: String	Sends a group broadcast message to all members of a society.
getConversation	userId: int, recipientId: int,	Retrieves conversation history between two users.
getMessages	userId: int	Retrieves all messages for a user..
notifyMessageStatus	messageId: UUID, status: enum{Delivered, Failed, Sending}	Updates and notifies sender of message delivery or read status.

Name	**Membership Service**
Description	Manages membership registration and profile management.
Stereo Type	Microservice
Depenedency	MySQL Membership DB, Message Broker

Method	Parameters	Description
setJoinPolicy	societyID: UUID, policyType: enum{Manual, Auto}	Sets whether join requires manual approval or is automatic.
getMembers	societyId: UUID	Retrieves list of members in a

		society.
transferLeadership	societyID: UUID, newLeaderId: memberId	Transfer leadership to another member.
requestJoin	societyId: UUID, userId: int	Submits a request to join a society.
approveJoin	membershipRequestId: UUID	Approves a pending request to join a society.
removeMember	societyId: UUID, memberId: int	Removes a member from a society.

Name	**Society Management Service**
Description	Manages societies and union workflows.
Stereo Type	Microservice
Depenedency	MySQL Society DB, Union Website Publisher, Union Promotion Logic, Union Approval Service, Message Broker

Method	Parameters	Description
submitNewSociety	name: String, theme: String, orgTeam: OrgTeam, contactInfo: ContactInfo	Submits a new society application for union.
activateSociety	userId: String, messageHistory: String	Activates an approved society so it appears on the Union's website.
updateSocietyData	societyId: UUID	Requests changes to the society's registration data.
requestTermination	societyId: UUID	Submits a termination request for a society.
getSocietyData	societyId: UUID	Retrieves full registration data of a society.
listSocieties	filter: enum{Pending, Approved, Active, Terminated}	Lists societies by status.

Name	**Payment Handling Service**
Description	Processes payments, refunds, and transaction history for societies and events.
Stereo Type	Microservice
Depenedency	PostgreSQL Payment DB, Message Broker

Method	Parameters	Description
processPayment	userId: int, email: string, amount: double, paymentType: enum{Cash, Card}	Executes a payment transaction and returns a transaction ID.
refundTransaction	userId: int, transactionId: UUID, reason: string	Issues refund for a completed transaction.
getPaymentHistory	userId: int	Retrieves all past transactions for a user.
getTransaction	transactionId: UUID	Retrieves information of a specific transaction.
notifyPaymentStatus	transactionId: UUID, status: enum{Success, Failed, Refunded}	Sends a notification to a user of their payment outcome.

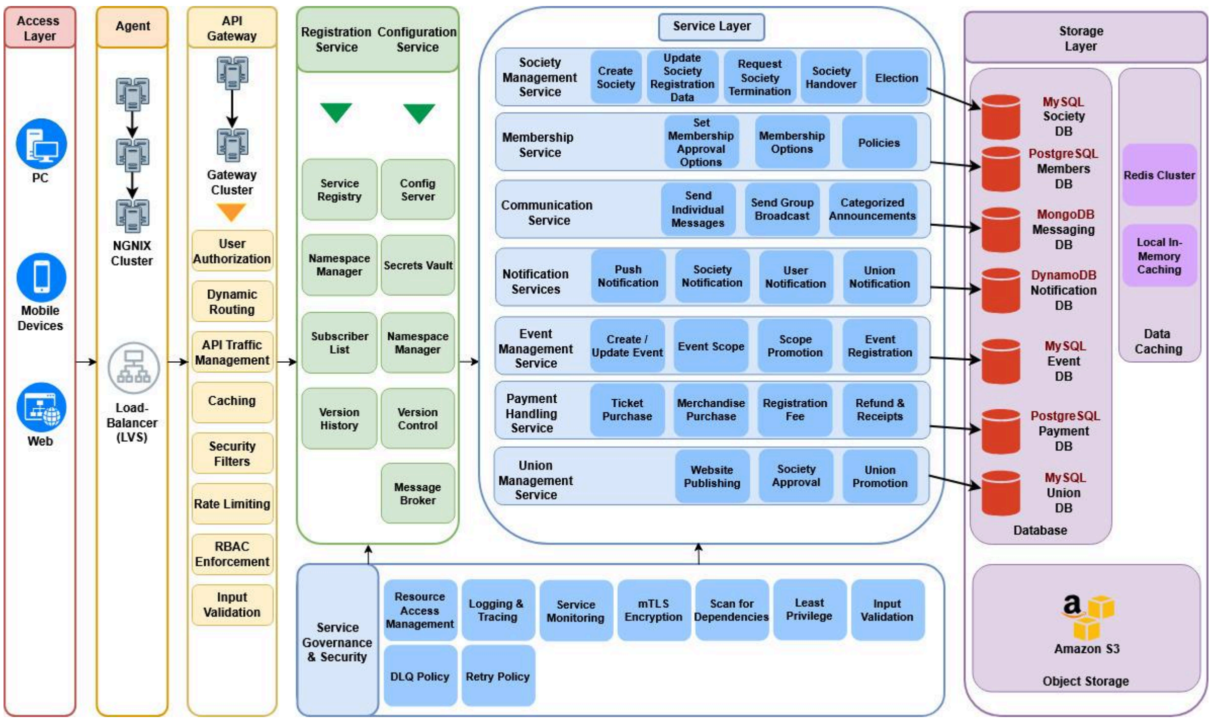
Name	**Event Service**
Description	Enables society leaders to organize and manage events..
Stereo Type	Microservice
Depenedency	MySQL Society DB, Payment Handling Service, Notification Service, Message Broker

Method	Parameters	Description
createEvent	eventData: EventData	Create a new event with details (name, theme, organizing team, connection, date & time, venue, constraints, promotion, scope).
updateEvent	eventId: UUID	Update event details or add progress information.
linkEvents	eventId: UUID	Links an event to related/previous events.
listEvents	societyId: UUID, scope: Enum	List events by societies or scope.
getEvent	societyId: UUID	Retrieves full details of an event.

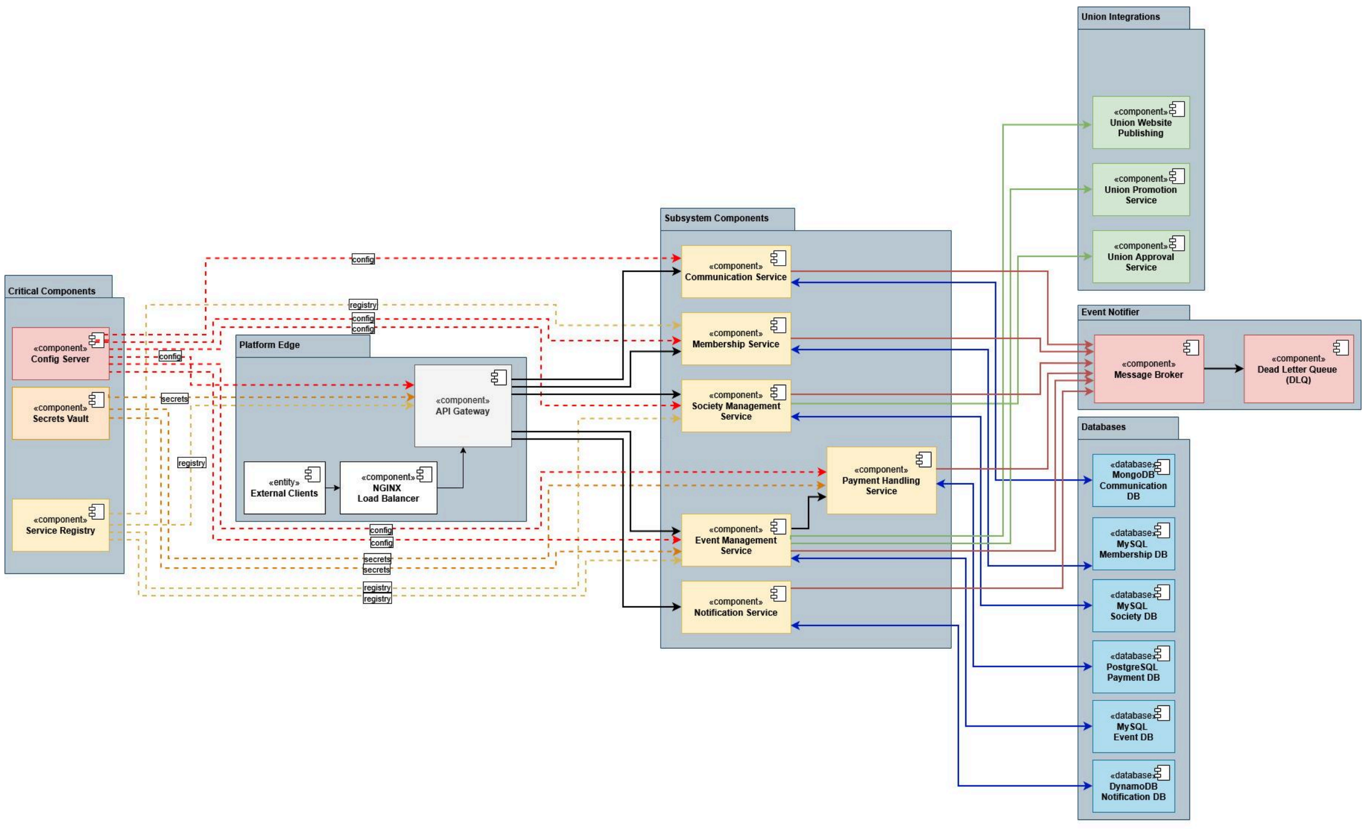
Name	**Notification Service**
Description	Sends notifications to users and union delegates
Stereo Type	Microservice
Depenedency	DynamoDB Notification DB, Message Broker

Method	Parameters	Description
sendToUser	userId: int, message, String, channel: enum{email, SMS, InApp}	Sends a notification to a user.
sendToUnion	unionId: int, eventId: UUID, eventUpdate: EventUpdate	Sends event update to a union delegate.
broadcastToMembers	societyId: UUID, message: String	Sends a broadcast message to all members or within a society.
directMessage	memberId: int, message: String	Sends a direct message to a specific member.
getStatus	notificationId: UUID, status: enum{Delivered, Failed, Sending}	Retrieves delivery status of a notification.

Subsystem Architectural Design



UML Component Diagram

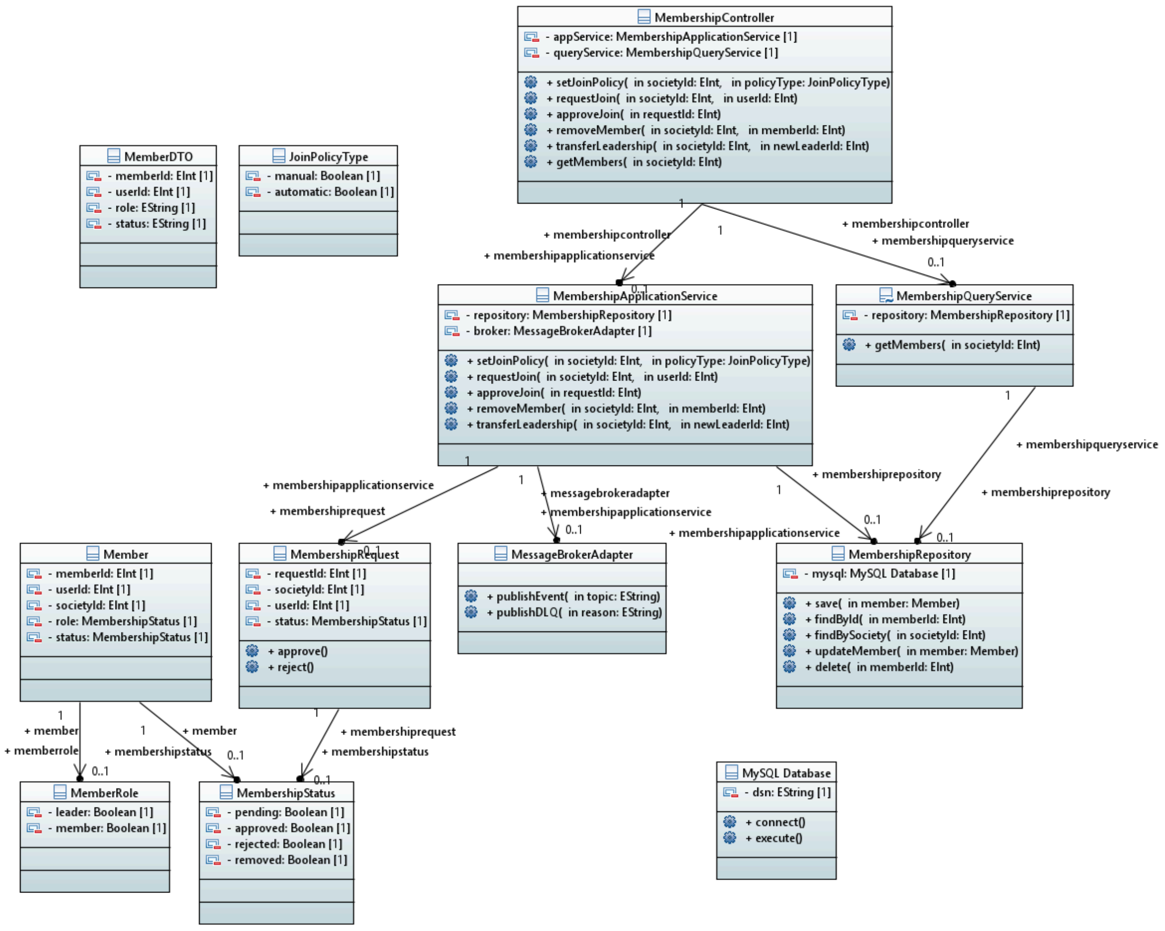

Critical Components
«component» Config Server
«component» Secrets Vault
«component» Service Registry
Platform Edge
«component» API Gateway
«entity» External Clients
«component» NGINX Load Balancer
config
registry
secrets
Subsystem Components
«component» Communication Service
«component» Membership Service
«component» Society Management Service
«component» Payment Handling Service
«component» Event Management Service
«component» Notification Service
Union Integrations
«component» Union Website Publishing
«component» Union Promotion Service
«component» Union Approval Service
Event Notifier
«component» Message Broker
«component» Dead Letter Queue (DLQ)
Databases
«database» MongoDB Communication DB
«database» MySQL Membership DB
«database» MySQL Society DB
«database» PostgreSQL Payment DB
«database» MySQL Event DB
«database» DynamoDB Notification DB

Task 5 - Software Detailed Design

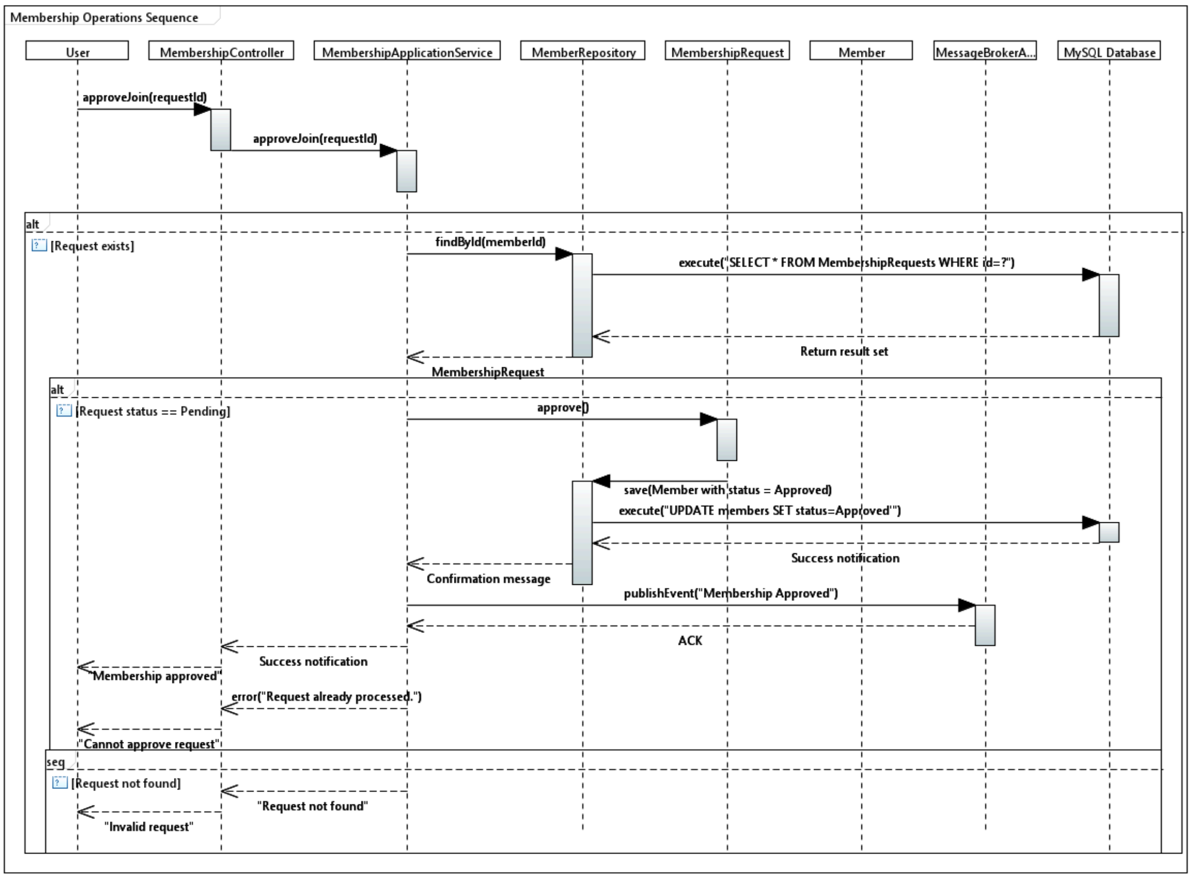
This task provides the structural design of the Membership Service, including controllers, services, repositories, adapters, database, and other essential components. It specifies class responsibilities, attributes, methods, and relationships, ensuring clear separation of concern.

The behavioural model complements the structural model by showcasing sequence flows for membership operations.

Structural Model


MembershipController
- appService: MembershipApplicationService [1]
- queryService: MembershipQueryService [1]
+ setJoinPolicy(in societyId: EInt, in policyType: JoinPolicyType)
+ requestJoin(in societyId: EInt, in userId: EInt)
+ approveJoin(in requestId: EInt)
+ removeMember(in societyId: EInt, in memberId: EInt)
+ transferLeadership(in societyId: EInt, in newLeaderId: EInt)
+ getMembers(in societyId: EInt)
MemberDTO
- memberId: EInt [1]
- userId: EInt [1]
- role: EString [1]
- status: EString [1]
JoinPolicyType
- manual: Boolean [1]
- automatic: Boolean [1]
+ membershipcontroller
+ membershipapplicationservice
+ membershipcontroller
+ membershipqueryservice
MembershipApplicationService
- repository: MembershipRepository [1]
- broker: MessageBrokerAdapter [1]
+ setJoinPolicy(in societyId: EInt, in policyType: JoinPolicyType)
+ requestJoin(in societyId: EInt, in userId: EInt)
+ approveJoin(in requestId: EInt)
+ removeMember(in societyId: EInt, in memberId: EInt)
+ transferLeadership(in societyId: EInt, in newLeaderId: EInt)
MembershipQueryService
- repository: MembershipRepository [1]
+ getMembers(in societyId: EInt)
+ membershipqueryservice
+ membershiprepository
+ membershipapplicationservice
+ membershiprequest
+ messagebrokeradapter
+ membershipapplicationservice
+ membershiprepository
+ membershipapplicationservice
Member
- memberId: EInt [1]
- userId: EInt [1]
- societyId: EInt [1]
- role: MembershipStatus [1]
- status: MembershipStatus [1]
MembershipRequest
- requestId: EInt [1]
- societyId: EInt [1]
- userId: EInt [1]
- status: MembershipStatus [1]
+ approve()
+ reject()
MessageBrokerAdapter
+ publishEvent(in topic: EString)
+ publishDLQ(in reason: EString)
MembershipRepository
- mysql: MySQL Database [1]
+ save(in member: Member)
+ findById(in memberId: EInt)
+ findBySociety(in societyId: EInt)
+ updateMember(in member: Member)
+ delete(in memberId: EInt)
+ member
+ memberrole
+ membershipstatus
+ member
+ membershiprequest
+ membershipstatus
MemberRole
- leader: Boolean [1]
- member: Boolean [1]
MembershipStatus
- pending: Boolean [1]
- approved: Boolean [1]
- rejected: Boolean [1]
- removed: Boolean [1]
MySQL Database
- dsn: EString [1]
+ connect()
+ execute()

Behavioural Model



Personal Report on Teamwork

Project Management Contribution

Although I was unable to attend some scheduled meetings due to other constraints, I contributed to project management by ensuring that each task was completed by our agreed scheduled deadlines. I progressively uploaded each diagram upon completion to our Github repository and recovered crucial files that were lost due to a faulty error in the repository. My role was more supportive than directive, but I ensured that my contributions were consistent.

Technical Contributions

My technical contributions focused on the design and development of my assigned subsystem—Society Leader App. Within the given timeframe, I was able to complete all tasks, including defining quality requirements, designing various models and diagrams (use case model, activity diagram, subsystem architectural design, UML component diagram, structural model, behavioural model), and detailed and thorough descriptive tables of different microservices while maintaining deliverable quality of work. I ensured my files were backed up, and stored and updated safely in the repository.

Collaborations with Team Members

Although I missed some meetings, I compensated by being responsive in our online group chat, documenting my contributions thoroughly to ensure continuity, even when real-time discussion was limited.

References

1. Muthineni, S. R. (2025). ‘Optimizing Mobile App Performance: A Comprehensive Guide’. *ResearchGate* [Online]. Available at: 8. 728-743. 10.34218/IJRCAIT_08_01_056.

2. Deepak, T. (2024). ‘Mastering Non-Functional Requirements for Mobile Application Development’. *Medium* [Online]. Available at: https://dtundwal.medium.com/mastering-non-functional-requirements-for-mobile-application-development-d77e3235fc0d.

3. NetGuru, (n.d.). ‘Mobile App Scalability: Mobile Development Explained’. *NetGuru* [Online]. Available at: https://www.netguru.com/glossary/mobile-app-scalability.

4. Miguel, P. G. (2025). ‘25 Best Application Monitoring Tools of 2025’. *TheCTOClub* [Online]. Available at: https://thectoclub.com/tools/best-application-monitoring-software/.