Student Union Management System - Individual Report

COMP5047 – Software Engineering

1. Introduction

This document presents the individual analysis and design for the Student Union Management System. The analysis has been focused on the functional requirement FR-UO-3: Management of Societies.

2. Analysis and Specify Software Quality Requirements for FR-UO-3: Management of Societies.

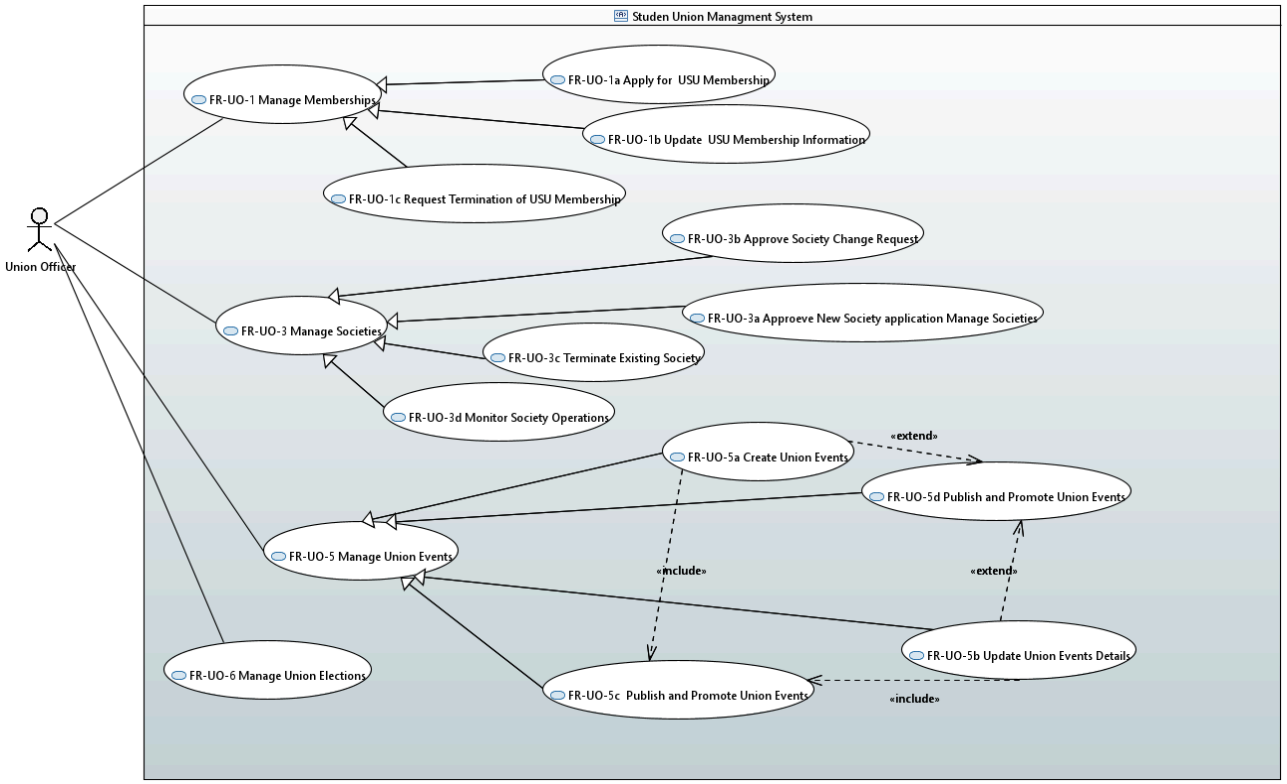
The quality requirements for FR-UO-3 address four critical attributes: security, performance, reliability and scalability.
Security is enforced through authenticated Union Officer access and logging for all society-management operations. Performance ensures that search and update operations execute within two seconds under normal load. Reliability requires 99.9% successful transactions and a maximum one-hour recovery time. Scalability guarantees the capacity to handle peak periods of 200 concurrent sessions and 500 society operations per minute without degradation of service.

Quality attributes	Scenario	Requirement	Metric	Target Value
Security & Privacy	When a Union Officer approves, updates, terminates, or monitors a society record	Union Officers must be authenticated into their official account before performing any operation. All actions (approvals, modifications, terminations, and monitoring) must be logged and securely stored.	- Percentage of operations executed by authenticated users. - Percentage of operations successfully recorded in the log.	- 100% authorised access - ≥99.9% of actions logged and stored
Performance	Searching society by name or theme	The subsystem has to return the information within 2 seconds and show confirmation messages after each action	- Average response time	- ≤ 2 seconds per operation
Reliability	During the approval, update, or termination of societies	The subsystem must complete all society management transactions	- Percentage of success rate - Mean Time to Recovery after failure	- ≥ 99.9% successful operations - MTTR ≤ 1 hour

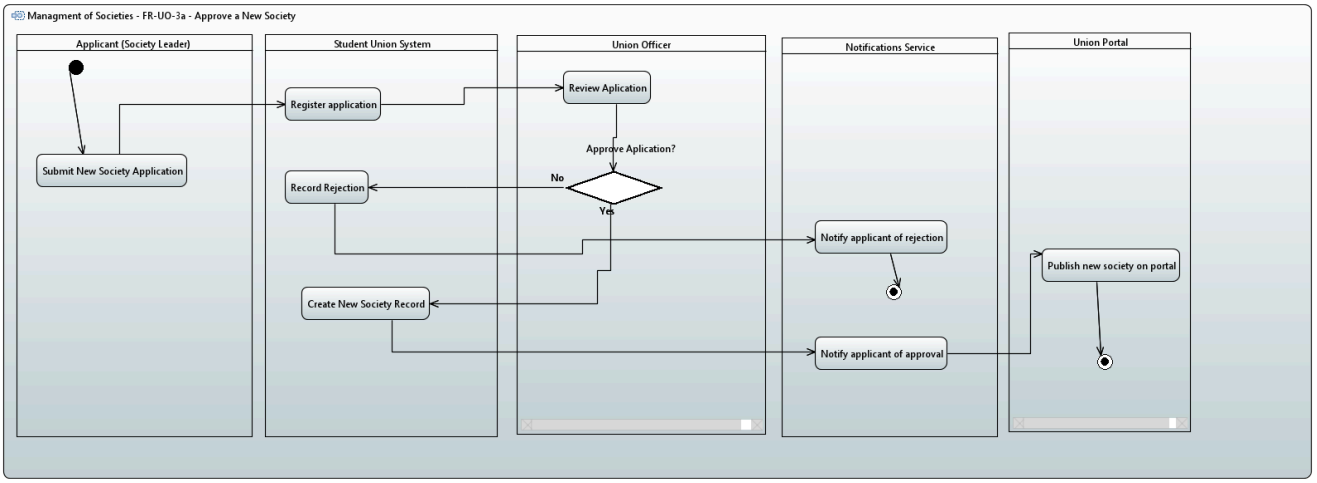
		without data loss or system failure in 99.9% of cases. In case of failure the system must recover within 1hour		
Scalability	During peak periods (e.g start of academic year)	The subsystem must support at least 200 user sessions simultaneously and 500 society operations per minute without reducing the response time or reliability	- Load testing results / concurrent session monitoring	- ≥ 200 simultaneous sessions; - ≥ 500 operations per minute

3. Modelling Software Functional Requirements

a. Use case Model



b. Activity Diagram



4. Software Architectural Design

The following tables describe the internal services of the Student Union Management System, together with the technical services it depends on, and the interfaces they provide and require.

Components:

Component	USU Membership Management
Description	Handles all union membership operations, including new membership applications, updates to membership details, termination requests, and membership status queries.
Stereo Type	Service
Provided Interface	iMembershipMgmt
Required Interface	iAuthentication, iDataStorage

Component	Union Admin
Description	Supports administrative functions such as scheduling union elections, consolidating election results, overseeing officer transitions, and managing system-level administrative tasks.
Stereo Type	Service
Provided Interface	iUnionAdmin
Required Interface	iDataStorage

Component	Society Management
Description	Manages societies: registers new society applications, approves or rejects requests, processes change requests, terminates societies, and monitors their operational status.
Stereo Type	Service
Provided Interface	iSocietyMgmt

Component	AuthenticationService
Description	Authenticates users, validates credentials, and issues session tokens for secure access across the system.
Stereo Type	Technical Service
Provided Interface	iAuthentication

Required Interface	iNotification, iDataStorage

Component	Union Event Management
Description	Creates and manages union events, updates event details, cancels events, and disseminates event information to students and society leaders.
Stereo Type	Service
Provided Interface	iUnionEventMgmt
Required Interface	iNotification, iDataStorage

Required Interface	iDataStorage

Component	NotificationService
Description	Sends email and in-app notifications to students, society leaders, and staff, for example when a society application is approved or a union event is published.
Stereo Type	Technical Service
Provided Interface	iNotification
Required Interface	-

Component	Data Storage Service
Description	Provides persistent storage for all subsystem entities (memberships, societies, events, elections). Handles CRUD operations and data retrieval.
Stereo Type	Technical Service
Provided Interface	iDataStorage
Required Interface	-

Interfaces:

Name	iMembershipMgmt
Description	Provides all operations related to union membership management, including creation, update, and termination of student memberships.
Provider	USU Membership Management
Operation	• applyForMembership(studentId:int, membershipType:String): Confirmation Submits a new membership application. • updateMembershipDetails(studentId:int, contactInfo:ContactDTO): boolean Updates the membership details of a student. • terminateMembership(studentId:int, reason:String): boolean Terminates a student's membership.

Name	iUnionAdmin
Description	Exposes administrative functions such as scheduling elections and closing or consolidating their results.
Provider	Union Admin
Operation	• scheduleElection(electionData:ElectionDTO): int Schedules a new union election. • closeElection(electionId:int): boolean Closes the election and computes final results.

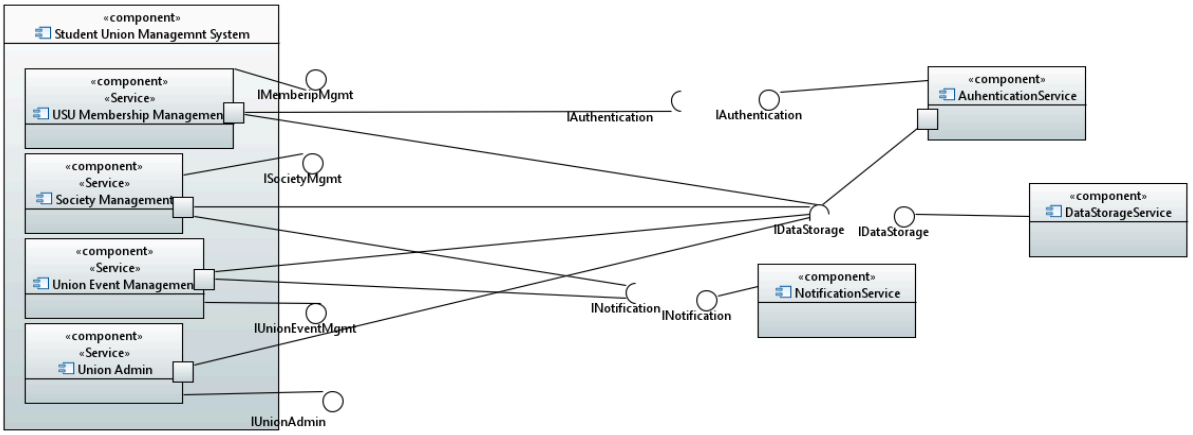
Name	iSocietyMgmt
Description	Offers API operations for society application handling, approval workflow, rejection workflow, and society monitoring.
Provider	Society Management
Operation	• submitNewSocietyApplication(applicationForm:SocietyApplicationInfo): int Registers a new society application. • approveSocietyApplication(applicationId:int, officerId:int): boolean Approves a society application. • rejectSocietyApplication(applicationId:int, officerId:int, reason:String): boolean Rejects a society application with a reason.

Name	iAuthentication
Description	Handles credential validation and secure authentication through tokens.
Provider	AuthenticationService
Operation	• authenticateUser(username:String, passwordHash:String): AuthToken Authenticates the user and generates an authentication token. • validateToken(token:AuthToken): boolean Validates the authenticity and expiry of a session token.

Name	iUnionEventMgmt
Description	Supports the creation, update, cancellation, and publication of union events.
Provider	Union Event Management
Operation	• createUnionEvent(eventData:UnionEventDTO): int Creates and publishes a new union event. • updateUnionEvent(eventId:int, eventData:UnionEventDTO): boolean Updates an existing union event. • cancelUnionEvent(eventId:int, reason:String): boolean Cancels the event and triggers notifications.

Name	iNotification
Description	Provides messaging and notification capabilities for workflows that require user or leader communication.
Provider	NotificationService
Operation	• sendEmail(recipientEmail:String, subject:String, body:String): boolean Sends an email notification. • sendInAppNotification(userId:int, message:String): boolean Sends an internal in-app notification.

Name	iDataStorage
Description	Supports persistent data access and retrieval for all business entities.
Provider	DataStorageService
Operation	• saveEntity(entity:Object): boolean Saves an entity to storage. • findById(entityType:String, id:int): Object Retrieves a specific entity by ID. • queryEntities(querySpec:QueryDTO): List<Object> Performs a complex query over the stored dataset.



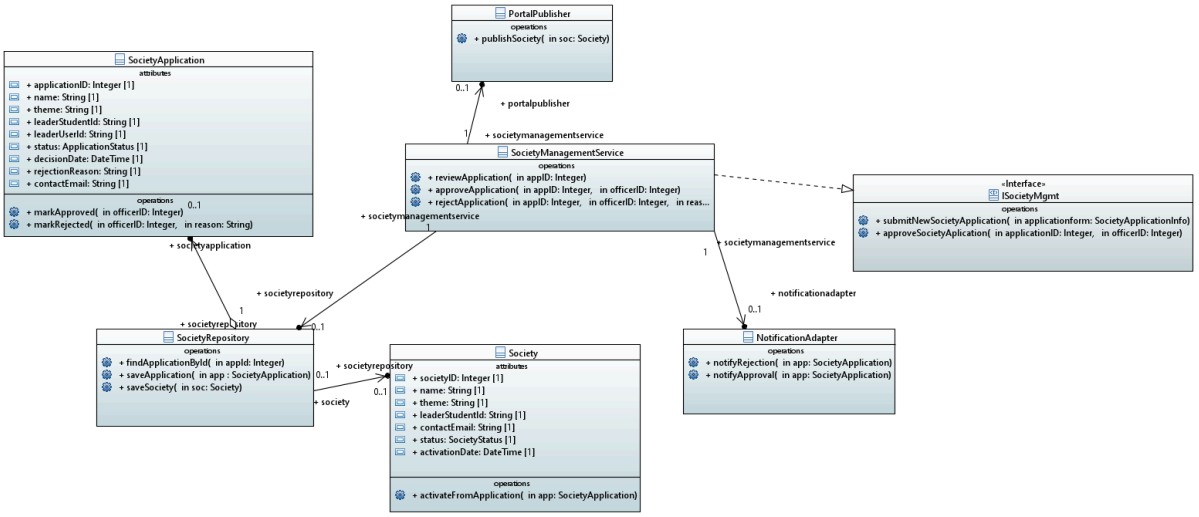
5. Software Detailed Design

a. Structural Model

The class diagram shows the internal structure of the microservice that manages societies.

- SocietyManagementService coordinates the whole process and uses SocietyRepository to store and retrieve data.
- SocietyApplication and Society represent the applications and the approved societies.

The service also communicates with other parts of the system through NotificationAdapter and PortalPublisher, which handle notifications and updates on the union portal.

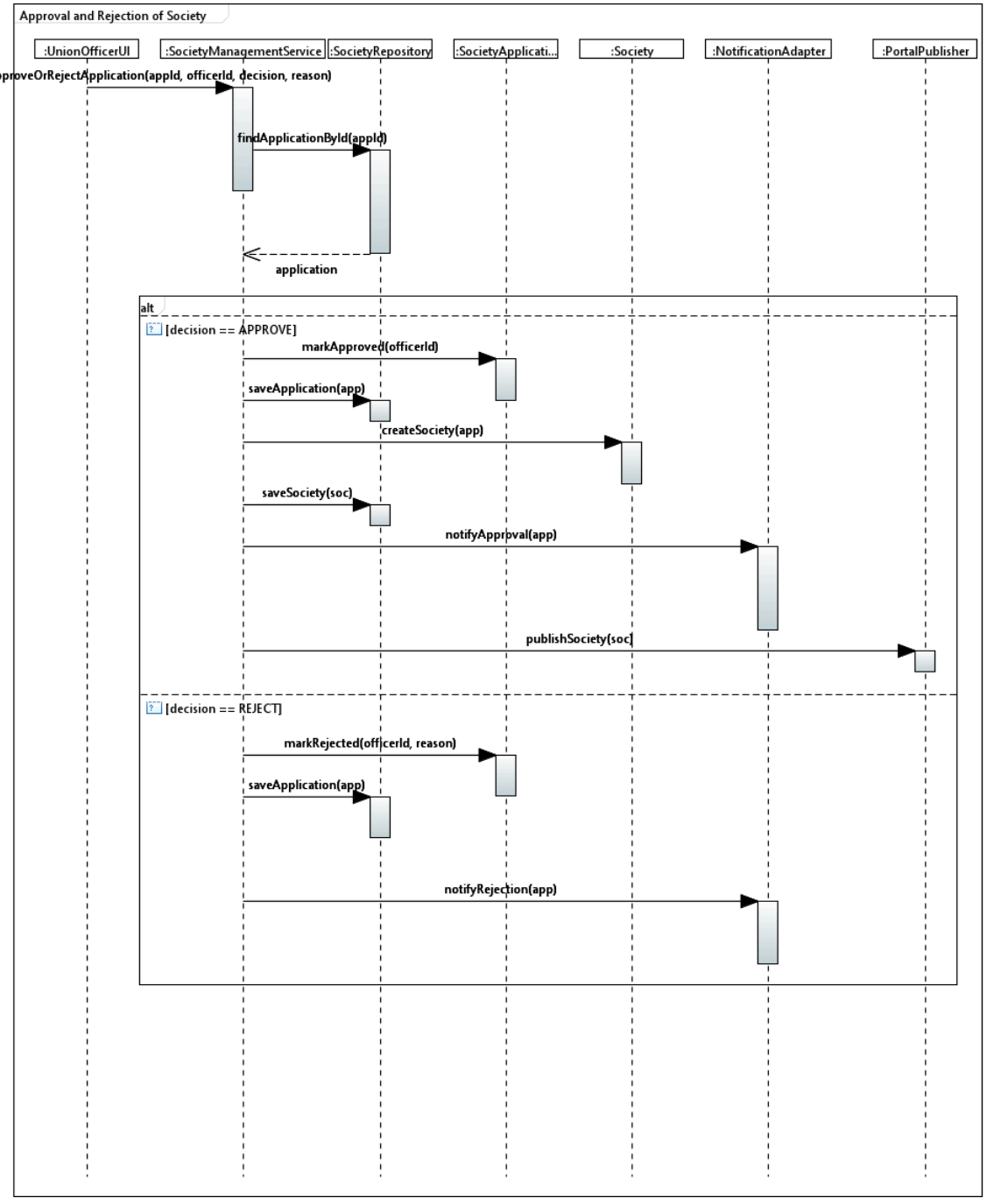


b. Behaviour Model

The sequence diagram shows how the approval process works inside the microservice. The service gets the application from the repository, checks the Union Officer's decision, and follows either the approval or rejection path.

- If it is approved, the application is updated, saved, turned into a new Society and published on the portal.
- If it is rejected, the application is saved with the rejection and a notification is sent.

Both paths make sure the information is stored correctly and the actions can be tracked.



6. Personal report on teamwork

a. Contributions to Team Organisation

Throughout the project, I contributed actively to the organisation of the team. I proposed several meetings during the semester and regularly updated my progress so that the rest of the group could follow how my work was advancing. I attended all meetings except one and wrote the meeting notes on several occasions.
I also uploaded my diagrams and Papyrus files to GitHub progressively, keeping my part of the repository organised and up to date. In addition, I suggested a common method for preparing the integrated component diagram by using shared tables to combine information from all subsystems. I communicated frequently with the group through our chat, helping to clarify questions and ensuring that everyone stayed aligned with deadlines and project expectations.

b. Technical Contributions

My technical contributions focused on the development of my assigned subsystem. I completed the use case model, activity diagram, architectural design, component and interface specifications, and the detailed design of the selected microservice. I kept my files updated in the repository and contributed ideas for the integrated component diagram of the whole system.
I also reviewed my teammates' work when needed, providing feedback to maintain consistency across the group's models. Throughout the semester, I made sure to complete my tasks on time and kept the quality of my diagrams and design work consistent with the project requirements