<u>COMP5047 Coursework-Member Report</u>

Student Number- 19333526
Group Number - 08
Subsystem - Student Union Management System (SUMS)

Quality Requirements Specification

Introduction

The Student Union Management System (SUMS) supports university level management by allowing Society Leaders to submit event proposals and Union officers to review, approve or reject proposal requests. For quality requirement specifications, the selected functional requirement is FR-SL-2 that is Propose Society Event, which requires the subsystem to allow society leaders to create and submit proposals, support validation and review workflows and manage approval, rejection and notification processes.
Because the FR-SL-2 involves multiple users, sensitive information and time critical actions, it's important to have reliability and security. This section defines the security, privacy, performance, reliability and scalability requirements which are necessary to FR-SL-2. All requirements are written as clear, measurable and testable statements.

Security and Privacy Protection Requirements

SPP 1: Authorised Access to Event Management

Only authenticated Society Leaders may submit proposals and only authenticated Union Officers may review, approve or reject proposals..
Testable Condition: 100% of unauthorised requests must be rejected in access control tests.

SPP 2: Secure handling of event data

All event proposal data like title, description, venue, date and time, endorsements and decisions must be encrypted during transmission and stored securely at the end.
Testable Condition: System checks must confirm encryption enabled for data in transit and at storage layers.

SPP 3: Privacy of Proposal Status and Decision History

A proposal's status, decision rationale and officer comments must be visible only to the submitting society leader and authorised union representative.
Testable Condition: Privacy tests must show 0% visibility of this data to unauthorised accounts.

SPP 4: Mandatory Logging of Proposal Lifecycle Actions

Every action related to proposal creation, validation, approval, rejection and notification must generate a timestamped audit entry containing actor ID, proposal ID and action type.
Testable Condition: Audit analysis must confirm a complete log entry for every lifecycle action performed during testing.

Performance Requirements

These requirements ensure the proposal workflow remains responsive for both students and officers.

PR 1: Responsiveness of Event Creation and Update

Submitting a proposal which includes data validation and storage must complete 3 seconds for 95% of request under normal load conditions that is less than or equal to 100 student users
Testable Condition: Performance testing must validate response time distribution

PR 2: Officer Review Response Time

When loading a proposal for review it must be completed within 2 seconds for 95% of requests during a typical load of 50 union officers.
Testable Condition: Review interface performance tests must meet this threshold.

PR 3: Notification Dispatch Latency

The system should send the approval and rejection notification to the society leader within 5 seconds of the officer submitting the decision.
Testable Condition: End to end timing tests must demonstrate notification latency within limits.

PR 4: System Throughput for Proposal Workflow

The system should handle at least 500 proposal related operations per minute, that is submission,validation, review request and notifications without degradation.
Testable Condition: Load tests must confirm throughput capacity.

Reliability Requirements

These requirements make sure that event management stays stable and consistent during heavy usage and unexpected failures.

RR 1: Availability of Proposal Services

Proposal submission, validation and review functions must maintain 99.5% availability during operational hours.
Testable Condition: Monitoring data must confirm availability over a standard measurement period.

RR 2: Preservation of Proposal Data During Failures

No proposal data that includes submitting information, officer notes, approval decisions or notifications may be lost during system crashes or unexpected failures.
Testable Condition: Crash recovery tests must confirm zero data loss.

RR 3: Automatic Recovery of Proposal Processing Services

If any microservice involved in FR-SL-2 fails, the system automatically recovers via restarts or failovers within 60 seconds of average recovery time.
Testable Condition: Crash recovery tests must demonstrate zero data loss in all validated scenarios.

RR 4: Daily Backup of Proposal and Decision Records

All proposal related data must be backed up every 24 hours. Restoration of the most recent backup must complete within 30 minutes.
Testable Condition: Backup and restoration tests must verify both frequency and recovery time.

Scalability Requirements

These requirements make sure that the subsystem can scale to handle the subtle increased volume of proposal, users and event related operations without compromising responsiveness or system stability.

SR 1: Horizontal Scaling of Proposal Processing Services

Core services like submission, validation, review and notification service must scale horizontally from 2 to 8 instances without code changes.

Testable Condition: Scaling tests must confirm correct behaviour under increased instance counts.

SR 2: Concurrent User Capacity

The subsystem must support at least 1000 concurrent society leaders viewing proposal status and at least 200 concurrent union officers performing review tasks without breaching the systems performance limits.

Testable Condition: Concurrent load tests must be performed to validate capacity.

SR 3: Growth in Proposal Volume

The subsystem must support at least 20000 stored proposals and 5000 active pending submissions while maintaining responsiveness thresholds.

Testable Condition: Testing with large datasets should confirm the system remains responsive.

SR 4: Scale Automatically During Heavy Load

If CPU usage for any proposal handling services stays above 70% for 5 minutes, a new instance should be added automatically.

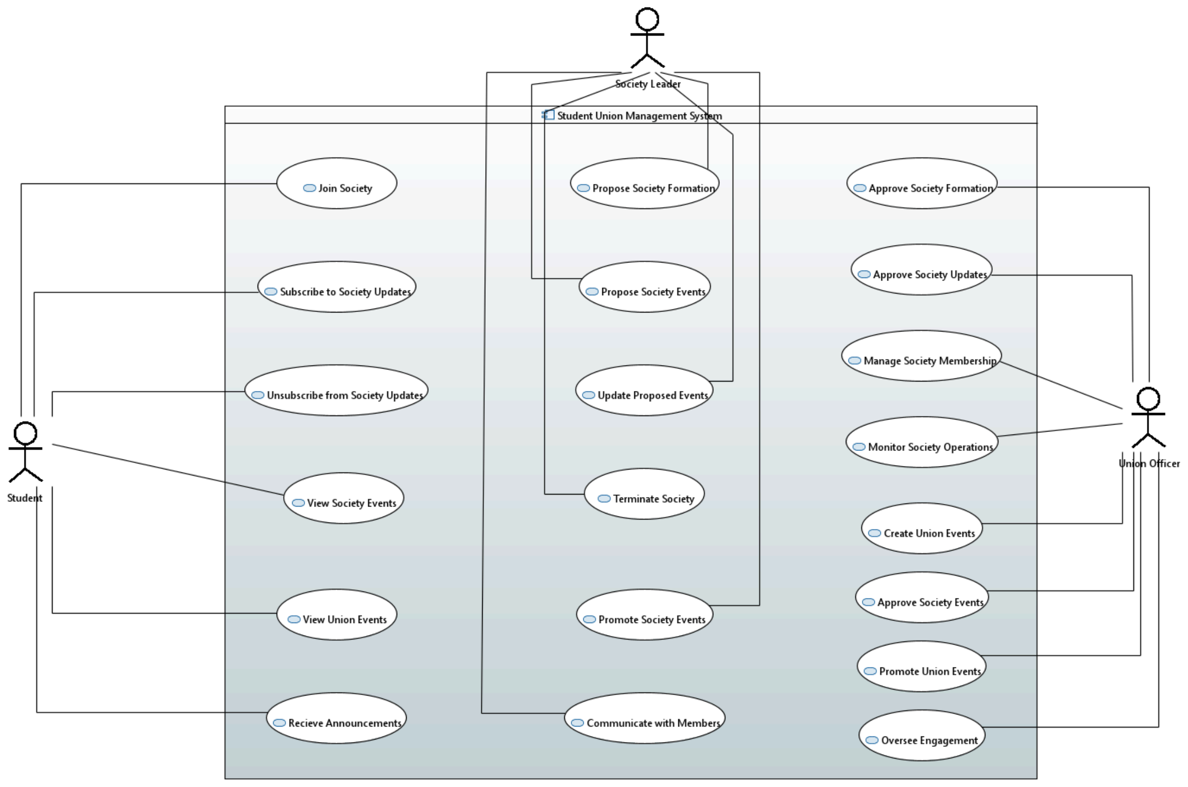
Testable Condition: Load testing should demonstrate the auto-scaling behaviour.

The quality standards outlined here make sure that the system is safe, effective and dependable operation of FR-SL-2 that are to propose society events in a multi user university setting. Sensitive proposal data and decision records are safeguarded by security and privacy regulations. Performance standards guarantee that the processes for creating, reviewing and notifying proposals continue to be quick and responsive. Requirements for reliability ensure that the subsystem remains stable and bounces back from errors quickly. SUMS is prepared to accommodate growing numbers of officers, students and event proposals over time because of its scalability requirements. When combined, these quality standards provide a solid start to deliver a dependable and effective event proposal workflow.

Use Case Diagram

The diagram is enclosed within a system boundary and labelled as “Student Union Management System”, which indicates that all the use cases inside the boundary represent the functionality provided by this subsystem. The actors are positioned outside the boundary to emphasise that they are external users interacting with SUMS. Each use case inside the boundary directly corresponds to a functional requirement defined in the case study.
Students mainly use SUMS to take part in society activities. They can join or leave societies, manage their subscriptions, and keep up to date with society or union events. Society Leaders use the system in a more administrative way, they can propose new societies, submit or update event proposals, promote upcoming events, and communicate with their members. Union Officers have the supervisory role in the system. They review and approve society formations and event proposals, monitor how societies are operating, and manage overall student membership within the union structure.
The use case diagram therefore provides a high level of functional view of Student Union Management Service, clearly illustrating the system’s responsibilities, the actors who depend on the system, and the interactions necessary to support the workflows described in the

case study. It makes sure that every functional requirement is captured and linked to the appropriate user.



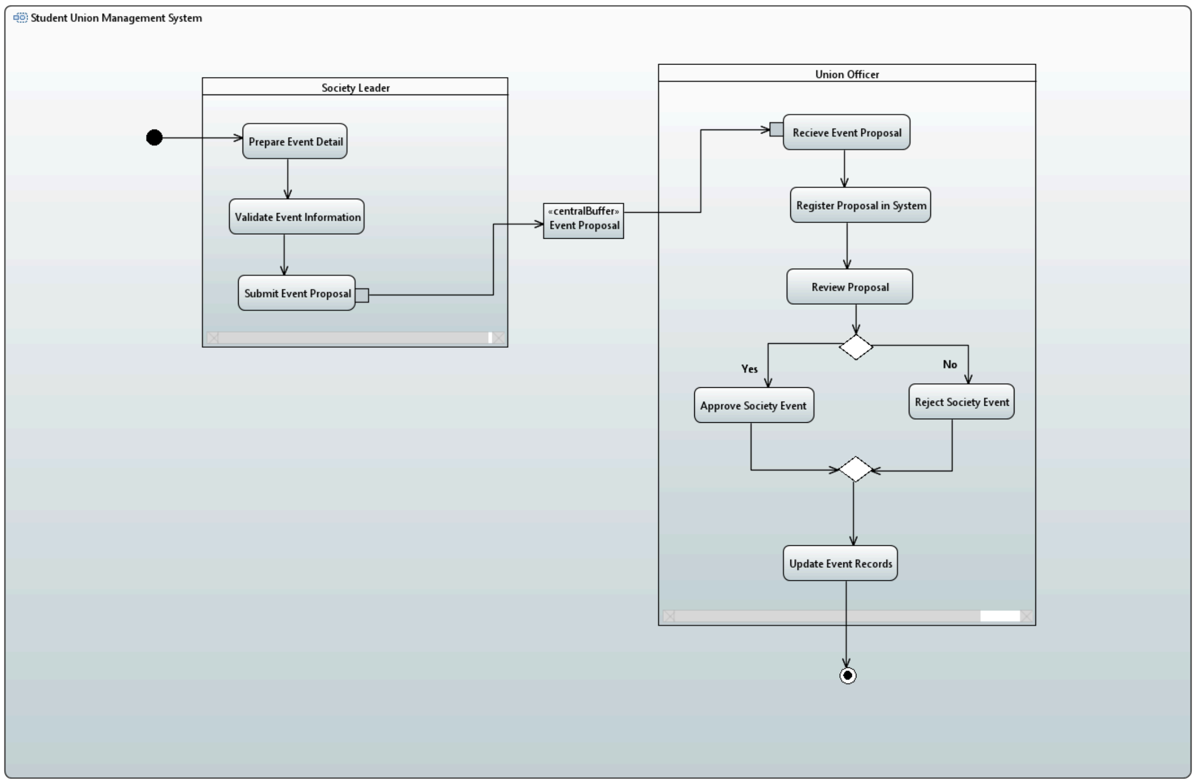
Activity Diagram

The activity diagram models the detailed behaviour of a single functional requirement from the case study. In this subsystem, the chosen requirement is FR-SL-2 that is Propose Society Event, which describes how a society leader submits a new event proposal and how a union officer reviews and approves or rejects it. The activity diagram provides a step by step representation of the workflow and shows how responsibilities are divided between two actors.

To make the responsibilities clear, the diagram is split into two lanes from which one is for Society and one is for Union Officer. The diagram starts with the society leader preparing for the event and then submitting the proposal through the system. Once it is submitted, the proposal is held in a temporary buffer, which represents the system passing the event information from the leader of the officer.

In the Union Officer's lane, the workflow continues with the officer receiving the proposal, checking it, and making a decision. A decision node is used to show two possible outcomes, that is the officer can either approve or reject the proposal. If it is rejected, the system

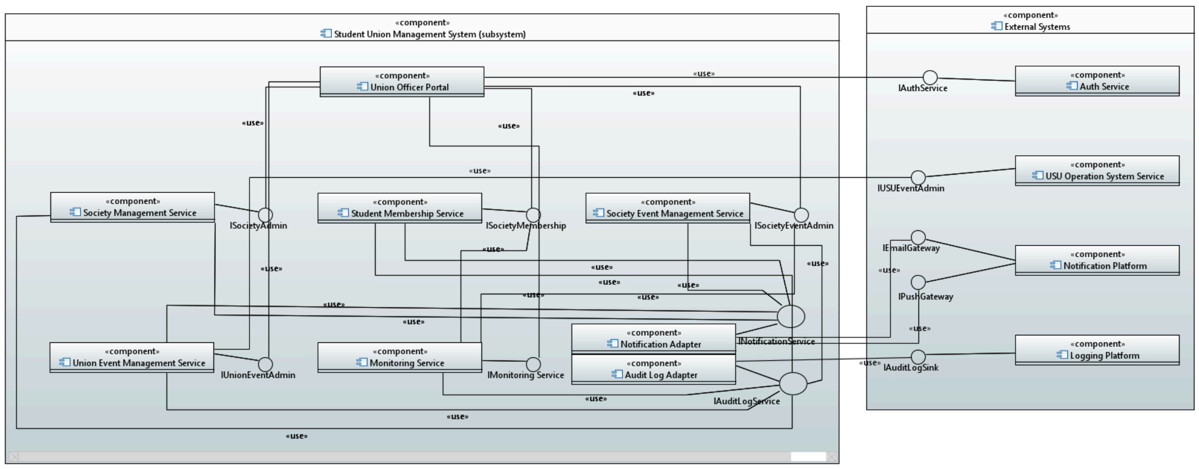
notifies the society leader. Both outcomes eventually join back together before reading the end point of the process.
Overall, the activity diagram gives a clear picture of how SUMS handles an event proposal from start to finish and shows the interaction between the society leader and the union officer in a structured way that follows the case study.



Subsystem Architectural Design (Component Diagram)

The Component diagram shows how the Student Union Management System is organised internally and how it connects to the rest of the whole system. The idea behind this diagram is to break the subsystem into smaller microservices, where each component has a specific responsibility and interacts with others only through clear, well defined interfaces. This helps the system stay modular, easier to maintain and easier to scale.
Inside the Student Union Management System, each major feature from the case study is represented as its own component. For example, the Social Management Service handles all the operations related to societies, while the Society Membership Service looks after student membership actions like joining or leaving societies. The Social Event Management and Union Event Management components deal with the creation and administration of society level and union level events. The Monitoring Service gives union officers the ability to track society behaviour and the Union Officer Portal is the interface officers use to interact with the system.

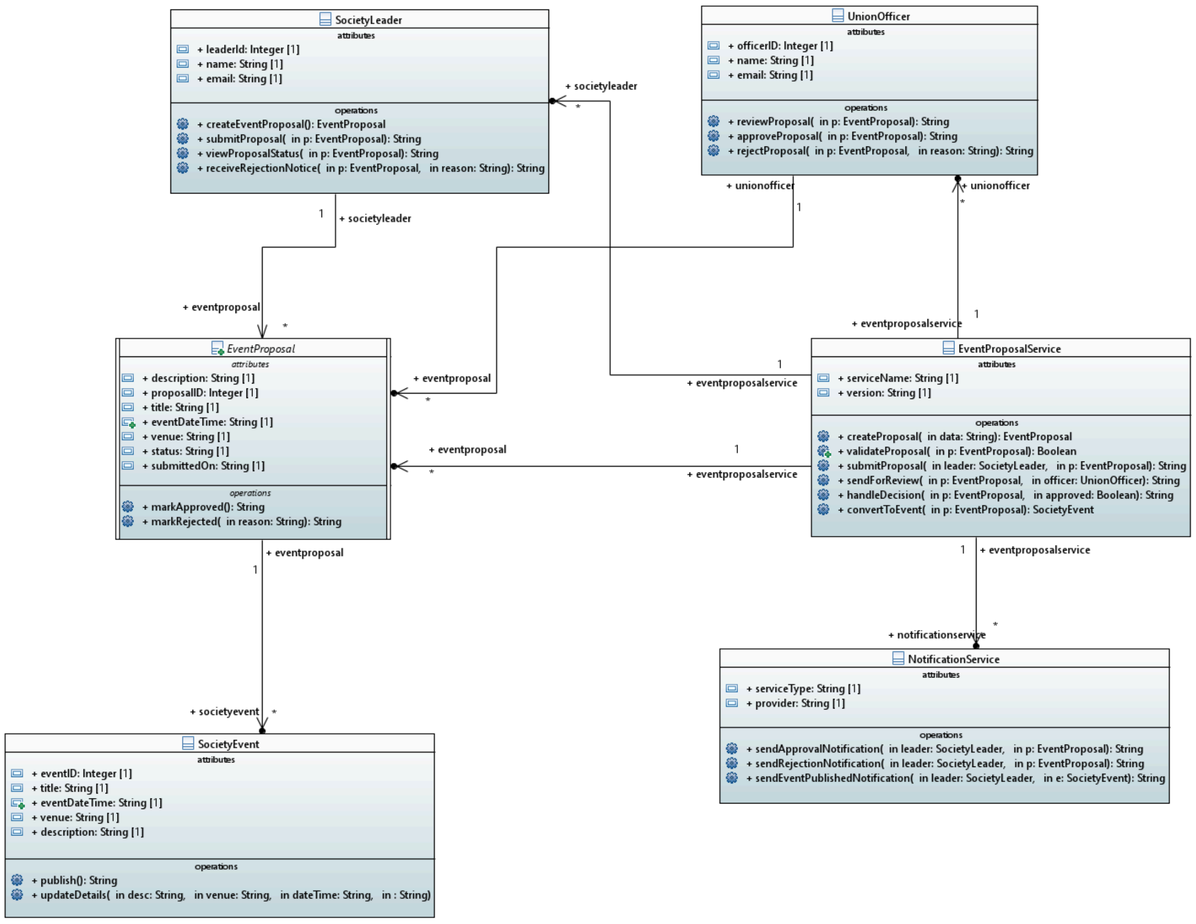
Each service provides its own provided interface which allows other subsystems to use Student Union Management System functions. SUMS also depends on several required interfaces, such as IAuthService for user authentication, notification gateways for sending messages and logging services to record system actions. These are shown using dashed usage arrows in the diagram.
Overall, the component diagram clearly illustrates how SUMS is structured as a set of independent microservices and how it relies on external services to complete its tasks. It gives a high level architectural view of the subsystem and shows how it fits into the larger system described in the case study.



«component» Student Union Management System (subsystem)
«component» Union Officer Portal
«component» Society Management Service
«component» Student Membership Service
«component» Society Event Management Service
«component» Notification Adapter
«component» Audit Log Adapter
«component» Union Event Management Service
«component» Monitoring Service
ISocietyAdmin
ISocietyMembership
ISocietyEventAdmin
INotificationService
IAuditLogService
IUnionEventAdmin
IMonitoring Service
«use»
«component» External Systems
«component» Auth Service
«component» USU Operation System Service
«component» Notification Platform
«component» Logging Platform
IAuthService
IUSUEventAdmin
IEmailGateway
IPushGateway
IAuditLogSink

<u>**Structural Model (Class Diagram)**</u>
The class diagram models structures of the Student Union Management System (SUMS) for the functional requirements <u>**FR-SL-2**</u>, which allows a society leader to create and submit an event proposal that will later be approved or rejected by a union officer. The diagram shows the main classes involved in this process, the data they store, the operations they offer and how they are connected.
At the centre of the model is the EventProposal class, which represents each proposal submitted by a society leader. It stores all the information described in the case study, such as the title, description, venue, date and time and current status. It includes operations like markApproved() and markRejected() that updates the proposal based on the officer's final decision.
The diagram also includes real users represented as classes like SocietyLeader and UnionOfficer. The SocietyLeader can create and submit proposals, view their status and

receive notifications while the UnionOfficer reviews proposals and either approves or rejects them. Their operations match the behaviour described in the functional requirement.
Two service classes, EventProposalService and NotificationService, handle the internal system logic. EventProposalService is responsible for creating proposals, validating them, sending them for review, processing the officer's decision and converting approved proposals into final SocietyEvent objects. NotificationService sends approval or rejection messages to the society leader once the decision has been made.
The associations between these classes model how they interact like a society leader can have many proposals and each proposal is linked to one officer during the review and an approved proposal leads to exactly one SocietyEvent. These relationships reflect the real sequence of actions in FR-SL-2 and provide the structural basis for the activity and sequence diagrams.
Overall, the class diagram gives a clear and organised picture of the subsystem and shows how data and operations are structured   to support the event proposal process.


SocietyLeader
attributes
+ leaderId: Integer [1]
+ name: String [1]
+ email: String [1]
operations
+ createEventProposal(): EventProposal
+ submitProposal(in p: EventProposal): String
+ viewProposalStatus(in p: EventProposal): String
+ receiveRejectionNotice(in p: EventProposal, in reason: String): String
UnionOfficer
attributes
+ officerID: Integer [1]
+ name: String [1]
+ email: String [1]
operations
+ reviewProposal(in p: EventProposal): String
+ approveProposal(in p: EventProposal): String
+ rejectProposal(in p: EventProposal, in reason: String): String
+ societyleader
*
1
+ societyleader
+ unionofficer
1
+ unionofficer
*
+ eventproposal
*
+ eventproposalservice
1
EventProposal
attributes
+ description: String [1]
+ proposalID: Integer [1]
+ title: String [1]
+ eventDateTime: String [1]
+ venue: String [1]
+ status: String [1]
+ submittedOn: String [1]
operations
+ markApproved(): String
+ markRejected(in reason: String): String
+ eventproposal
*
1
+ eventproposalservice
+ eventproposal
*
1
+ eventproposalservice
EventProposalService
attributes
+ serviceName: String [1]
+ version: String [1]
operations
+ createProposal(in data: String): EventProposal
+ validateProposal(in p: EventProposal): Boolean
+ submitProposal(in leader: SocietyLeader, in p: EventProposal): String
+ sendForReview(in p: EventProposal, in officer: UnionOfficer): String
+ handleDecision(in p: EventProposal, in approved: Boolean): String
+ convertToEvent(in p: EventProposal): SocietyEvent
+ eventproposal
1
+ societyevent
*
1
+ eventproposalservice
+ notificationservice
*
SocietyEvent
attributes
+ eventID: Integer [1]
+ title: String [1]
+ eventDateTime: String [1]
+ venue: String [1]
+ description: String [1]
operations
+ publish(): String
+ updateDetails(in desc: String, in venue: String, in dateTime: String, in : String)
NotificationService
attributes
+ serviceType: String [1]
+ provider: String [1]
operations
+ sendApprovalNotification(in leader: SocietyLeader, in p: EventProposal): String
+ sendRejectionNotification(in leader: SocietyLeader, in p: EventProposal): String
+ sendEventPublishedNotification(in leader: SocietyLeader, in e: SocietyEvent): String

Behaviour Model (Sequence Diagram)

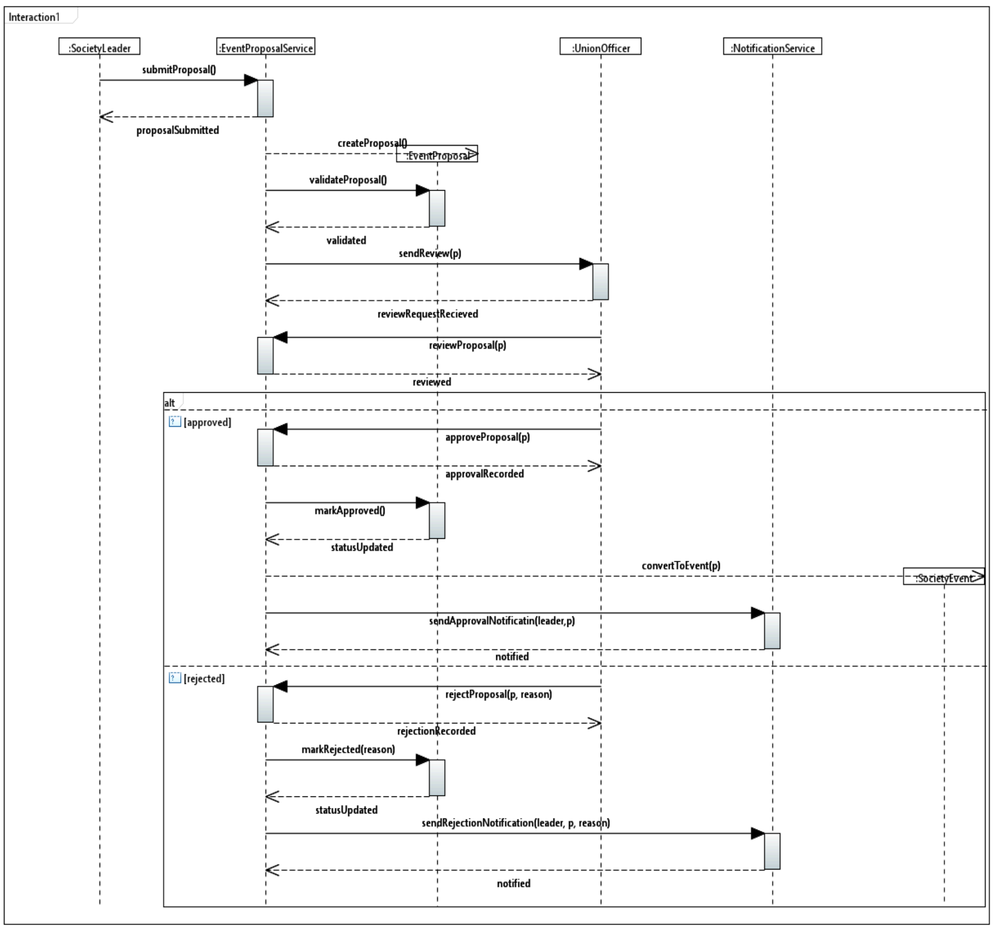
The sequence diagram shows how different parts of the SUMS subsystem interact over time to complete the event proposal process described in FR-SL-2. While the class diagram explains what the system is made of, the sequence diagram focuses on what the system actually does when a society leader submits an event proposal and a union officer reviews it.

The diagram starts with the society leader sending their proposal to the EventProposalService. The system then creates a new EventProposal object which checks that the information is valid and passes the proposal to the Union Officer for review. These steps always happen in the same order and represent the normal flow of the feature.

The most important part of the diagram is the alt fragment, which shows that there are two different outcomes depending on the union officer decision. If the officer approves the proposal, the system updates the proposal's status, creates a new SocietyEvent and sends a notification to the society leader. If the officer rejects it , the system marks the proposal as rejected and sends a rejection notification. Only one of these branches will run at a time and the alt fragment makes this very clear.

Return messages are shown as dashed arrows, which helps to make the communication between objects easier to understand. The diagram also reflects how responsibilities are shared as the union officer makes the decision, but the system handles the rest, which helps updating the proposal, creating events, sending notifications and keeping the process consistent.

Overall, the sequence diagram helps to visualise the full flow of the requirement and shows how the different classes in the subsystem work together to complete the event proposal process.



Design Discussion

The design for the Student Union Management System subsystem focuses on supporting FR-SL-2 that is Propose Society Event and the goal was to allow society leaders to submit event proposals and union officers to review, approve or reject them in a controlled and traceable manner. The subsystem is designed as a part of a larger microservice based architecture and therefore emphasises on clear boundaries, explicit interfaces and separation of capabilities.

The Society Event Management Service is a central component of the subsystem as it coordinates the complete proposal lifecycle which includes submission, validation, review and decision handling which helps to keep this in a single core service and makes sure to simplify the frontend interaction, as all proposal related actions are accessed through one interface.

Supporting components are separated to avoid overloading the core services, the notification adapter is responsible for triggering user notifications when proposals are approved or rejected while the audit log adapter ensures that all the important actions are recorded normally. This separation ensures that notification and logging behaviour does not interfere with the main proposal workflow and can resolve independently.

The student membership service and society management service support access control and validation as these components ensure that proposals are submitted only by valid society members and reviewed only by authorised union representatives. By isolating membership and society related checks into dedicated components, the design improves clarity and reduces duplication of logic across the subsystem and this approach directly supports the security and privacy requirements by enforcing role based access without embedding permission checks in every component.

All interactions with the external systems are handled through clearly defined interfaces as authentication is managed through the IAuthService, notifications are delivered via IEmailGateway and IPushGateway and logging is handled using IAuditLogSink. The subsystem does not depend on the internal implementation of these external services also, it relies only on the interfaces, which reduces coupling and makes the system easier to integrate and maintain. This interface based design is particularly suitable for frontend driven subsystems that primarily trigger backend processes.

Overall, the design prioritises clarity, modularity and scalability as each component has a well defined responsibility and interactions and explicit through interfaces and usage dependencies. This makes the subsystem easier to test, extend and integrate into the wider system and the design also supports future growth as new features or services can be added without reconstructing the existing components.

Conclusion

The work presented in this report focused on the design and quality analysis of the Student Union Management System subsystem, with specific emphasis on the functional requirement FR-SL-2 that is Propose Society Event as the subsystem was designed to support the complete lifecycle of the event proposal from submission by society leaders to review and decision making by union officers.
Quality requirements covering security and privacy, performance, reliability and scalability were defined to ensure that the subsystem operates safely, efficiently and consistently in a real university environment as each requirement was expressed in a measurable and testable manner which allows the system behaviour to be verified through testing and monitoring.
The architectural design was represented using component, class and sequence diagrams to illustrate both the structural and behavioural aspects of the subsystem because, the design separates responsibilities across clearly defined components and relies on interface based integration with external systems such as authentication, notification and logging service, this approach supports modularity, reduces coupling and makes the subsystem easier to maintain and extend.
Overall, the proposed design provides a clear and practical solution for managing society event proposals meanwhile it supports the future growth and integration within the wider USU architecture.

Groupwork Reflection

During this group project, my main responsibility was working on the Student Union Management System subsystem as my focus included defining the subsystem scope, contributing to the component diagram and supporting the design with the class and

sequence diagrams, as well as aligning the design with the selected functional requirement and quality requirements.
One of the main challenges I faced during the project was confusion around the subsystem allocation as initially, I was assigned to work on the USU Officer System, but the role was later swapped with a groupmate on his request but due to miscommunication and the complexity of the case study, I ended up spending additional time working on the USU Officer System again when i was actually expected to focus on the SUMS subsystem. Unfortunately, this misunderstanding was only fully realised at a later stage of the coursework which created time pressure and required me to restart the whole coursework which I was actually supposed to do for my work.
Once the issue became clear, I took the responsibility for the situation and decided to reorganise my work rather than rely on partially misaligned material after that i reviewed the case study again, clarified the SUMS requirements and rebuilt the whole subsystem design to ensure it correctly matched the functional requirements and the overall group architecture. This included reworking on every diagram, redefining component boundaries and updating the report sections to ensure consistency, although this was challenging but it helped me gain a deeper understanding of the system and improved the quality of my final contribution. Through this experience, I learned the importance of early clarification of responsibilities in group projects, especially when working on interconnected subsystems. I also developed a better understanding of how the individual subsystem designs must align with the overall system architecture, particularly when defining the interfaces and dependencies. My personal perspective on the situation helped me improve my time management and adaptability under pressure.
If I were to approach a similar project again, I would aim to confirm subsystem ownership and design boundaries earlier and request more frequent design checkins with the group as this would help to avoid duplicated work and reduce confusion during integration, despite the challenges, the project provided a valuable experience in collaborative software design and reinforced the importance of communication and flexibility in the group work.