COMP5047: Applied Software Engineering

Coursework 1

Introduction

This project forms part of the COMP5047 module and the structured development of multi-subsystem computer applications within a team. The application in question was to be developed for the United Student Unions federation. The system is designed to be capable of supporting the procedures of several universities' student unions across the country. The following sections include an analysis of quality requirements for a specific functional requirement, a use case model outlining the functional scope of the Student Union Management System (SUMS), description of team structure and division of work, as well as explanation of both use case and activity diagrams produced during this coursework project.

1 Project Management

1.1 Team structure

For this project our team will consist of four members: myself (Malachai Broderick), Ben Crosfield, Lucas Oliveira, and Tania Reyes. In our initial meeting, I took the liberty of collecting the other members' contact details and making a WhatsApp group, in which we would be able to communicate seamlessly for the rest of the project. As for the leadership of our group, Tania happily volunteered for the role of group leader, meaning our team's structure was established swiftly. This meant being responsible for monitoring progress checks and representing the team in discussions with the module leader. All other team members contribute equally to both technical and administrative tasks to promote fairness and prevent burnout within the over the course of this project.

1.2 Division of work and Subsystem assignment

Each team member will be responsible for and focused on the development of one subsystem These were also delegated in our first meeting. Listed below are the specific subsystems we will each be responsible for:

- USU Student App – Tania Reyes
- Student Union Management System – Malachai Broderick
- USU Operation System – Ben Crosfield
- Society Leader – Lucas Oliveira

All components, including agendas, meeting minutes, design documents, and UML models are stored in a GitHub repository accessible to all members of our team. Version control allows us to manage the branches for subsystems, merge work effectively, and track all of our contributions to efficiently integrate our project.

The team's work process is Agile based, involving weekly reviews and incremental development. Each week we review our deliverable against the coursework specification to assure we are meeting its requirements and fulfilling learning objectives.

2 Analysis and Specify Software Quality Requirements

2.1 Overview of Functional Requirement

FR-UO-5: This functional requirement, as listed in the relevant case study for this coursework project, outlines a key functional requirement of this subsystem. It goes on to mention that the system should enable unions to organise events, including initial set-up and updates on event progress.

For the Student Union Management Subsystem (SUMS), I will be focusing on the functional requirement based on management of Union events. This function grants union officers the ability of a specific university's student union to create, update, and promote events through this subsystem's web interface.

Each event record must include data defined in the case study (name, theme, date and time, venue, plan, participation constraints, scope, promotion scope, and promotion materials).

After an event has been registered, it is stored in the USU cloud backend and shared with other subsystems.

2.2 Subsystem Functionality

The subsystem I am looking at is a web interface used directly by the user. As such, it must have the required functionality to allow for user interaction (e.g. registration/creation of events, event management), effective communication with the backend or other subsystems in general.

Listed below are its specific functions:

- Allows each University's Student Union committee members/officers to register their login information, storing it in the backend for re-login.
- Societies can be created, modified, and terminated with approval from their respective student unions.
- Societies and unions can create, promote, and manage event
- All updates (e.g. event modification) are shared across all subsystems and synchronised in real-time.
- The site should be capable of handling ticket sales, allowing SU members to release event tickets at the desired quantity and price.
- Union officers must be able to approve or reject new society applications.
- Approval from the Union should notify the applicant through communication with the Society Leader App.
- Rejections will record the reason and mark the application as closed.

Quality Requirements

Society Management

Security & Privacy Protection:

- Only authenticated officers or society leaders will access society management functions via a role-based access control system, ensuring that only users with sufficient privileges will be able to approve or modify that society's records.
- Modification logs shall be timestamped, including an IP address and Student ID, being kept for 12 months to support more extensive auditing.
- Approval decisions classed as 'sensitive' (e.g. society termination) shall trigger multi-factor confirmation steps, preventing unauthorised or accidental actions.
- All file upload that relates to new societies shall be validated and virus scanned to prevent injection of malicious content into the cloud.

Performance:

- Update transactions will complete within _ seconds. Page loads listing all union societies will occur within 4 seconds for 95% of requests (server-side processing time).
- Page listing societies should load within 2 seconds for 95% of requests under normal conditions.

- Frequently accessed data will be pre-cached by the system to improve response times and reduce database load.

Reliability:

- If the society creation process fails mid transaction, the system will automatically roll back to the previous consistent state.
- Society management data will be replicated across multiple cloud availability zones to increase fault tolerance.
- The system will queue pending society update requests and automatically retry them following service outages.

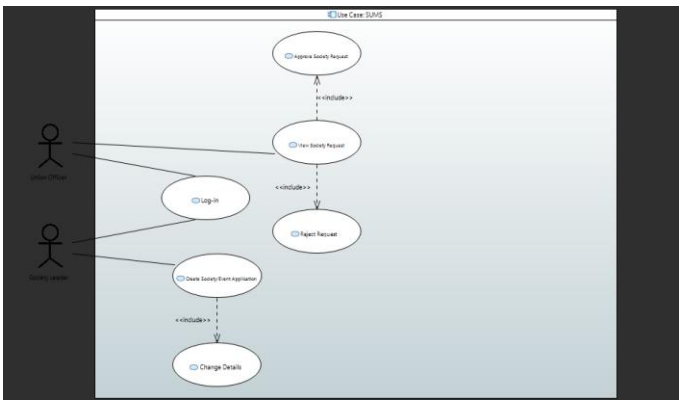
Scalability:

- The database shall support up to 5000 individual societies per university.
- The system will support concurrent operations for hundreds of society leaders and officers during peak usage periods.
- The system architecture will support the onboarding of new universities, without the need for downtime or restructuring.

Design Spec.

The design for this system was carried out in both collective and individual parts. We worked on both Use Case & Activity Diagrams separately in Papyrus, then produced diagrams representing the whole-system architecture.

The image here is a use case diagram for the SUMS I designed. It illustrates the core functionality of the SUMMS subsystem, identifying external actors and the services they require from the system in a specific use case. The two actors present in the diagram are a union officer and a society leader. The diagram includes numerous use cases in which the actors interact with the system. These include, logging in, viewing creation requests, and approval or denial of requests. As you can see, association links are used in the diagram. This makes it easy to recognise which actions are initiated by each user.



I also made an activity diagram to complement the use case diagram above. While the use case diagram outlines system functionality at a conceptual level, this diagram provides a bit more detail in showing the steps within a use case and the workflow involved in executing this functionality. It also uses swim lanes to separate responsibilities between the Society Leader, Union Officer, and SUMS subsystem. The diagram starts with a society application being submitted and progresses as this is then received and reviewed by the Union Officer.

A decision node is as a means of modelling the branching logic of the process, indicating the two scenarios where the society application is either approved or rejected. By including system-level activities such as notification, updates, and audit logging within the SUMS swim lane, the activity diagram shows clearly how the subsystem internally meets the criteria of the use case.