Faculty of Technology, Design & Environment

Assessment cover

OXFORD BROOKES UNIVERSITY

Module No:	COMP5047	Module title:	Applied Software Eng

Assessment title:	Software Engineering of a Modern Computer Application

Due date and time:	23:00pm, 5th Dec. 2025

Estimated total time to be spent on assignment:	84 hours per student

LEARNING OUTCOMES

On successful completion of this assignment, students will be able to achieve the module's following learning outcomes (LOs):

1. Demonstrate an understanding of the role of requirements analysis and specification in software engineering and to be able to use this knowledge to create use case models and functional models of computer applications.
2. Demonstrate an understanding of the relationship between requirements and design and to be able to apply the knowledge to create structural and behavioural models of computer applications.
3. Critically evaluate and utilise design paradigms of object-oriented analysis and design, component-based design, and service-oriented design.
4. Use software modelling language such as UML and modelling tools in the context of model-driven software engineering.
5. Work in a group to apply the knowledge and skills developed in this module

Engineering Council AHEP4 LOs assessed

C3	Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed
C5	Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards
C6	Apply an integrated or systems approach to the solution of complex problems
C14	Discuss the role of quality management systems and continuous improvement in the context of complex

	problems
C16	Function effectively as an individual, and as a member or leader of a team

GROUP 10:	

STUDENT NAMES

	Student Id:	Student Name:	Subsystem:
1.	19310301	Malachai Broderick	USU Student App
2.	19320199	Ben Crosfield	Student Union Management System
3.	19324320	Tania Reyes	USU Operation System
4.	19267298	Lucas Oliveira	Society Leader App

Statement of Compliance *(please tick to sign)*

☐ We declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments *(www.brookes.ac.uk/uniregulations/current)*

RUBRIC OR EQUIVALENT:

School of Engineering, Computing & Mathematics

APPENDIX I.

Introduction

This coursework covers the practical application of some of the]software engineering principles as part of the course COMP5047 - Applied Software Engineering, the aim of this coursework is to document the project management structure and software quality requirements of the "Society Leader App", which is part of the larger United Student Unions system. The document in fact provides a detailed account of a collaborative structure of my team and the technical design of one functional requirement which are all in accord with the modules' lectures.

APPENDIX II.

Project Management

1. Team Structure

The project team has four members. Each member is responsible for one subsystem of the United Student Unions (USU) platform. The structure used is in direct accord with a distributed subsystem responsibility method of ensuring the modularity of development and the subsequent economy of integration processes, being as follows:

- Member A – USU Student App (mobile application for students).
- Member B – Student Union Management System (web interface for university-specific student unions).
- Member C – USU Operation System (federation management subsystem).
- Author – Society Leader App (mobile and web interface for society leaders).

My project team agreed to work independently but we agreed to carry out weekly meetings in person and to maintain all project documentation in overall common use for the project team via a shared GitHub Repository.

2. Group collaboration

The project team arrived at some clearly defined media for collaboration to maintain consistency and equitable clarity. This consisted of:

- A GitHub repository used as the central hub for documentation.
- Weekly in-person meetings to coordinate subsystem integration and progress.
- A WhatsApp group for real-time communication.

3. Meeting Schedule and Minutes

Week	Date	Type	Topics	Attendance	Time and Place
2	07/10	Kick-off meeting	Team structure, GitHub setup, subsystem allocation	All members	12:00–13:00, AB-115, Headington Campus
3	14/10	Progress meeting	Discuss due dates; start drafting tasks; plan deadlines for Tasks 1 & 2	Two members	12:00–13:00, AB-115, Headington Campus
4	21/10	Planning meeting	Complete Quality Requirements; finish Use Case Diagram; begin Activity Diagram	Two members	12:00–13:00, AB-115, Headington Campus
5	28/10	Online meeting	Work on individual tasks; update group; progress on Use Case and Activity Diagrams	Three members	Online
6	04/11	Progress meeting	Continue individual task work; complete Use Case; progress Activity Diagram; start Task 4	Two members	12:00–13:00, AB-115, Headington Campus
7	11/11	Progress meeting	Upload Tasks 2 & 3; continue Task 4(a); contact missing teammate	Two members	12:00–13:00, AB-115, Headington Campus
8	18/11	Progress meeting	Task updates; individual work; Tania finalises Task 4(a)	Two members	12:00–13:00, AB-115, Headington Campus
10	27/11	Online meeting	Progress update; subsystem integration work; component/interface renaming	Three members	Online
11	02/12	Online meeting	Finalise coursework; upload all files; check Task 5 completion	Three members	Online

4. Agenda Template

- Attendance check.
- Review of previous actions.
- Subsystem progress updates.
- Blockers and integration concerns.
- Action items and deadlines.

5. Project Management Approach

The agile methodology was adopted by the team, integrating short cycles of core team coordination with formal documentation practices. Responsibilities were decided and got defined as balanced by subsystem owners, provided so that they can focus on their own subsystem growth. The team meets weekly to enable fixing of engaging effort without overloading any team with workload.

APPENDIX III.

Software Quality Requirements

1. Functional Requirement Overview

FR-SL-2: The system will enable society leaders to create events by providing information on the events including the name of the event, theme of the event, details of the organising agent, time and date of the event, location of event, coverage of event promotion and promotion material. The successful creation of an event will tell us if the event will be up and promoted to the extent defined in the USU system. The suggested functionality relates to what is believed to be close to student engagement and total system calling into an event. This was analysed in the case study and the relative quality attributes referred to in lecture 2 (Zhu, 2025) sourced from ISO/IEC 25010 (2011).

2. Quality Requirements Specification

a) Security and Privacy Protection

Security and privacy are important aspects of user information gained from user effort and organisational databases. The requirements for this subsystem are considered from the principles outlined in lecture 2, and compatible with ISO/IEC 25010 (2011). The following events will take the following measures:

- Events creation activity action should only be performed by bona fide society leaders, and as such comprehensive action on the basis of access control to verified roles should be carried out.
- Data transmission will be harnessed using HTTPS protection on the basis of TLS 1.3.
- Event data will be stored on the basis of storage security assurances with encryption.
- Systemic action including events nor user information will go unlogged to the extent of total accountability. Auditability will be achieved.

b) Performance

Requirements for performance will lead to delivery of a fast experience from the Society Leader App for users, to the extent that the individual experience is a fast action, particularly meaningful during stressful peak usage periods. The Research guidelines of the lectures will be followed (Lecture 2):

- Request for event creation will have the confirmation returned to the user in the space of 2 seconds provided a normal load of request demands.
- Uploading for promotional material (≤ 10 MB) should occur in the space of 5 seconds.
- Asynchronous processing would be required for applications to enable user responsiveness.

- Naturally the fluidity of the UI should be available during data transmission.

c) Reliability

Reliability as such suggests that stability of platform and data systems are guaranteed during total event creation. The principles of lecture 2 should be followed:

- The availability of the system in question should be not less than 99.5% during academic periods.
- Total establishment of the event must be on the basis of atomic transaction type by event (the event creation).
- The results of failure will cause rollback of the result and clear notifications of specific fault necessitated.
- Event data will be stored with sources of information and data systems across Cloud Servers.

d) Scalability

Necessary are the methods to ensure growth in kind of utilisation to be gained at more and more universities. Considerations from lecture 1 and 2 are preferable:

- The system should cope accordingly with an abundance of (not more than) 100,000 concurrent society leaders.
- Versions of the systems infrastructure should expand on a horizontal basis to cover peak demand during event creations and tested estimated maximum periods.
- Processes for the calling of events should lead dynamic scalability in an audience of considerable range.
- The extent of cached data should be available from the system to ensure work performed will not lead to lapses in performance outcomes.

3. Summary Table of Quality Requirements

Attribute	Requirement Highlights	Reference
Security & Privacy	TLS 1.3, encryption, role-based access, logging	Lecture 2; ISO/IEC 25010 (2011)
Performance	≤ 2 s response, async processing, responsive UI	Lecture 2
Reliability	99.5 % availability, atomic transactions, error rollback, redundancy	Lecture 2.
Scalability	100 k concurrent users, horizontal scaling, distributed notifications, caching	Lecture 1 and 2.

APPENDIX IV.

Use Case Model

The use case model describes the functional scope of the Society Leader App Subsystem, based on the principles provided in the lecture 3 of the module, and its functional requirements FR-SL-1 and FR-SL-2 as described in the Coursework Case Study.

The primary actor is the Society Leader; he/she will interact with the Subsystem to administer both the society, and the events for that Society.

There are two major functional areas within that model:

1. **Society Administration (FR-SL-1)**

    - Activate a Society
    - Update a Society's Registration Data
    - Set approval rules
    - View members
    - Communicate with members
    - Request society termination

2. **Event Administration (FR-SL-2)**

    - Create an event
    - Update an event
    - Manage event promotions
    - View event status

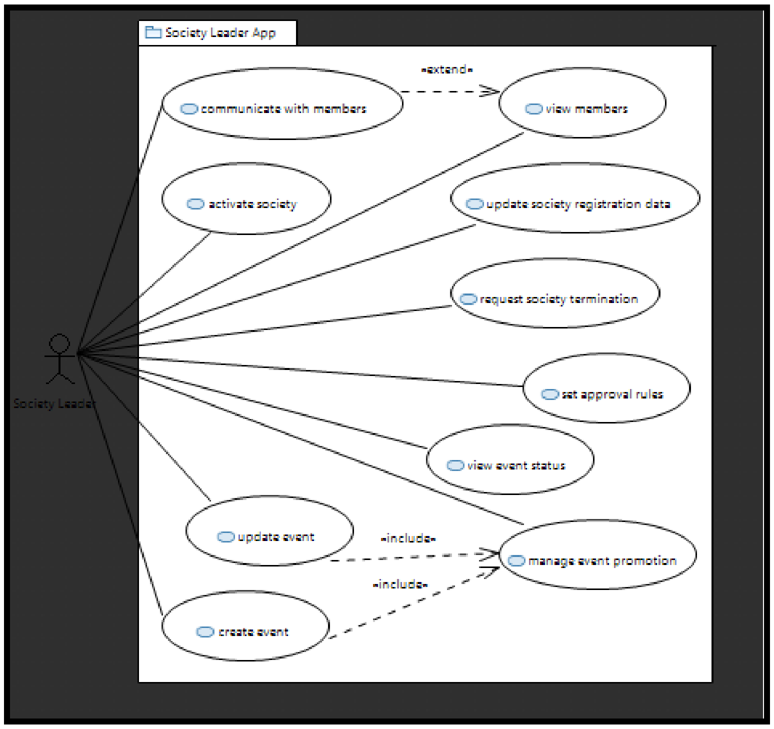
Two <<include>> relationships define mandatory shared behavior:

- Create an event and modify an event both include managing an event's promotions.

One <<extend>> relationship defines optional behavior:

- View and send messages to members

This model represents the core interaction between society leaders and the USU Platform, and provides the necessary functional foundation to perform the subsequent Architectural and Design Tasks.



Zoom in for better visualization or see the .png file uploaded in papyrus.

APPENDIX V.

Activity Diagram for "Create Event"

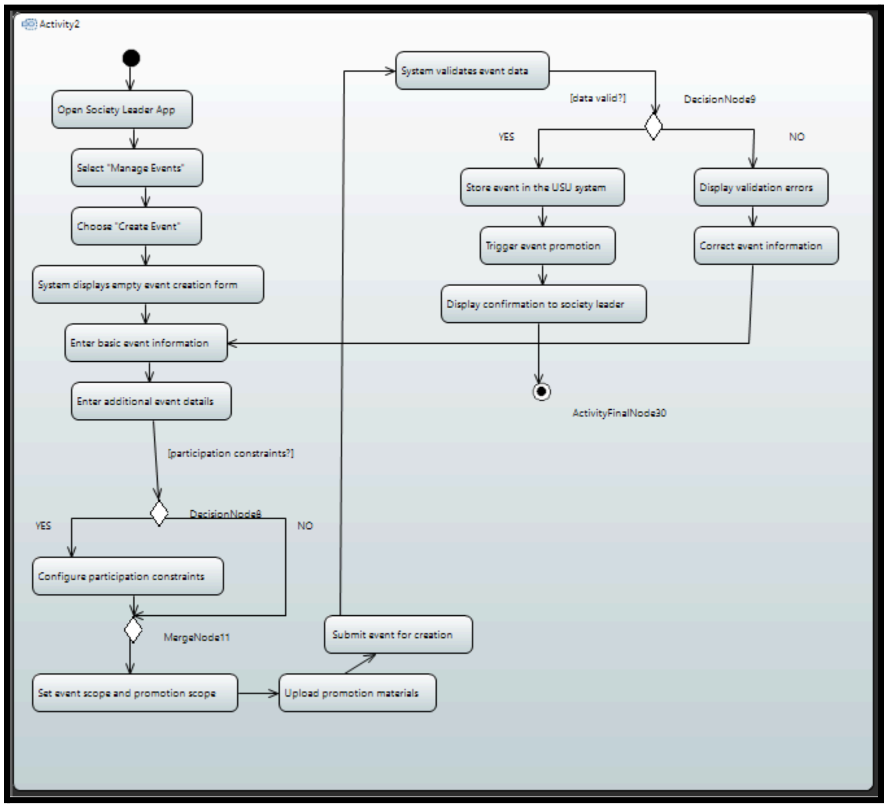
The Create Event activity diagram illustrates the internal workflow of the Create Event use case from the Society Leader App, as part of functional requirement FR-SL-2 of the case study.In terms of modelling conventions, it was completed using action steps, control flows, decision nodes and merge nodes structures for the handling of errors.

It works as the following: once the society leader has opened the app, navigated to the event management area and selected the "create new event" function, the system will display an empty event creation form where the leader can enter all the necessary basic and additional details about the event.

At this point the leader decides at a decision node, whether there are constraints for participation (for example registration or ticketing restrictions) - if there are then the leader configures them; otherwise the flow continues to the merge node. Once the merge node is reached, the leader can specify the event's scope and promotion scope, as well as upload promotional materials related to the event.

Once the event has been created, the system will perform a validation step, and once again, at another decision node the system checks whether the entered data is valid. If the data is invalid, the system will display the errors, and the society leader can go back to the form to make the necessary corrections to the entered data.

If the data is valid, the system will store the created event within the USU system, trigger the corresponding promotional measures, and show the society leader a confirmation message that the event has been successfully created, and therefore end the workflow.


Activity2
Open Society Leader App
Select "Manage Events"
Choose "Create Event"
System displays empty event creation form
Enter basic event information
Enter additional event details
[participation constraints?]
DecisionNode8
YES
NO
Configure participation constraints
MergeNode11
Set event scope and promotion scope
Upload promotion materials
Submit event for creation
System validates event data
[data valid?]
DecisionNode9
YES
NO
Store event in the USU system
Trigger event promotion
Display confirmation to society leader
Display validation errors
Correct event information
ActivityFinalNode30

Zoom in for better visualization or see the .png file uploaded in papyrus.

APPENDIX VI.

Architectural Design

As discussed in lectures 6 & 7 of this course, the Society Leader application uses an architectural model of microservices architecture, therefore I decided that a Service-Oriented Architecture approach would be used to decompose the system into four BackEnd Services. Each one exposes an interface and provides some functionality for society management:

- Society Management Service (ISocietyManagement): This service allows for the activation, update of registration information and configuration of approval rules for all societies and also initiates the termination of any society.

- Membership Service (IMembership): The Membership Service is the service that will provide the capabilities required for listing members and facilitating communications between members.

- Event Management Service (IEventManagement): The Event Management Service implements FR-SL-2 by providing the capability for creating, updating and retrieving events for a society.

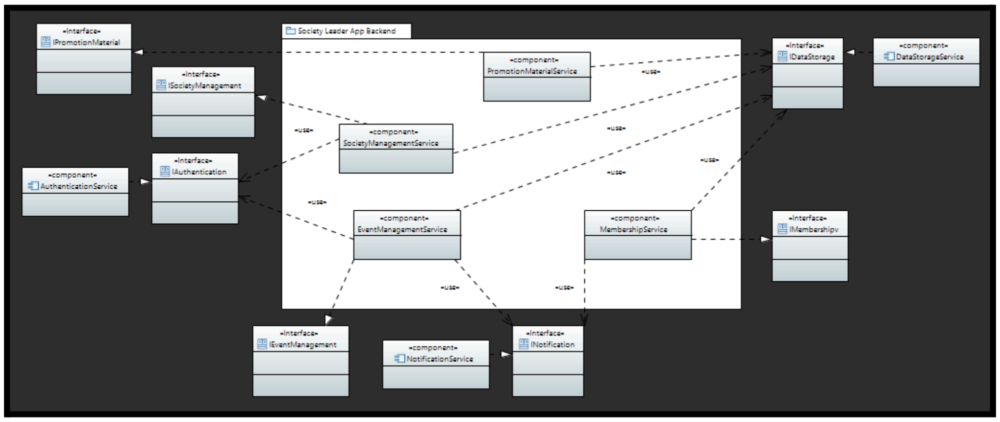
- Promotional Material Service (IPromotionMaterial): The Promotional Material Service is the service that will provide the capability to upload and manage promotional material for events.

Each of these services will be dependent upon three external services:

- Authentication Service (IAuthentication)
- Notification Service (INotification)
- Data Storage Service (IDataStorage).

Each backend service interacts only with the interfaces it requires. This is done so it gets in line with one of the key principles of micro-service architecture, namely low-coupling and high-cohesion among the system's parts.

So to be able to save the generated events for example, the Event Management Service will need to have access to the Data Storage Service (IDataStorage), additionally, to be able to include promotional material with the events and to inform event subscribers, the Event Management Service will only need to have access to the interfaces of the Promotion Material Service (IPromotionMaterial) and the Notification Service (INotification).



Zoom in for better visualization or see the .png file uploaded in papyrus.

For the 4b task, the architectures developed individually by each group member for the four USU sub-systems were then combined to create an overall architecture for the complete platform. As a result, the platform now has a set of shared services that are reused by all subsystems, including:

- AuthenticationService / IAuthentication
- NotificationService / INotification
- DataStorageService / IDataStorage

All of the duplicated components in the sub-systems (i.e. local versions of notification or storage managers) were eliminated and replaced with references to the shared services. The names of the interfaces were also made consistent across the diagrams (i.e. IDataStorage, INotification), so that all of the sub-systems rely on the same name of the interfaces.

The backend services of my sub-system (i.e. EventManagementService, SocietyManagementService, MembershipService, PromotionMaterialService) were incorporated into the other sub-systems by making them conform to the interfaces of the shared services and eliminating any duplication of services. For instance, EventManagementService now depends on the platform-wide IDataStorage and INotification services rather than creating its own versions of these services.

To make the role of my subsystem in the integrated architecture clearer, the following tables summarise the main components and interfaces that remain after the architectures were merged.

Components related to the Society Leader App in the integrated architecture

Component name	Provided interfaces	Required interfaces	Description in the integrated architecture
EventManagementService	-	IDataStorage, INotification, IAuthentication	Manages the generation, modification, and submitting of society events; however, after it has been integrated, it will utilize the interface to the common DataStorageService, NotificationService and AuthenticationService to execute these functions.
SocietyManagementService	-	IDataStorage, IAuthentication	Handles registration, activation and updates of societies. In the new architecture it will use the shared AuthenticationService for checking identities and the shared DataStorageService for all database persistence.
MembershipService	-	IDataStorage	Maintains lists of members for each society. Now all reading and writing functions are executed through the shared DataStorageService as opposed to being executed locally through a local storage component.
PromotionMaterialService	-	IDataStorage	Stores and retrieves promotional information

			regarding events. Although the PromotionalMaterialsComponent is unique to the Society Leader App, it utilizes the shared DataStorageService for all database persistence.
AuthenticationService (shared)	IAuthentication	-	The Platform-Wide Authentication Service (PWAS) is an authentication service that authenticates user's credentials and permissions for all components/subsystems within the platform, including the Society Leader App.
NotificationService (shared)	INotification	-	A central notification service utilized by the entire platform to deliver notifications/messages to students and society leaders. Local notification services have been removed from individual components/subsystems in favor of utilizing the PWAS.
DataStorageService (shared)	IDataStorage	-	Utilized by all components/subsystems as a shared persistence service. This service replaces local storage components and ensures that data is consistently stored and retrieved throughout the entire platform.

Interfaces used by the Society Leader App after integration

Interface name	Provided by	Required by	Reference
IAuthentication	AuthenticationService	EventManagementService, SocietyManagementService	Performs authentication of users and authorization of user requests on behalf of backend services, enabling them to verify the identity of each requesting user and determine whether they have permission to execute their respective requests.
INotification	NotificationService	EventManagementServiceEventManagementService, SocietyManagementService	The EventManagementService uses this service to send notification requests (i.e., confirmations about new or updated events) to appropriate society leaders and/or members.
IDataStorage	DataStorageService	EventManagementService, SocietyManagementService, MembershipService, PromotionMaterialService	This service provides an abstraction layer for data storage/retrieval of events, societies, memberships and promotional materials across all backend services via a unified persistence API (save/update/query).

APPENDIX VII.

Detailed design

In this design of the OO structure, I focused on the event handling in the Society Leader App, specifically on the “create event” use case from Task 3 and the EventManagementService component from Task 4(a). I utilized the same Boundary-Control-Entity pattern that was presented in the OO Design lectures, where we divide responsibilities between UI, application logic and domain data.

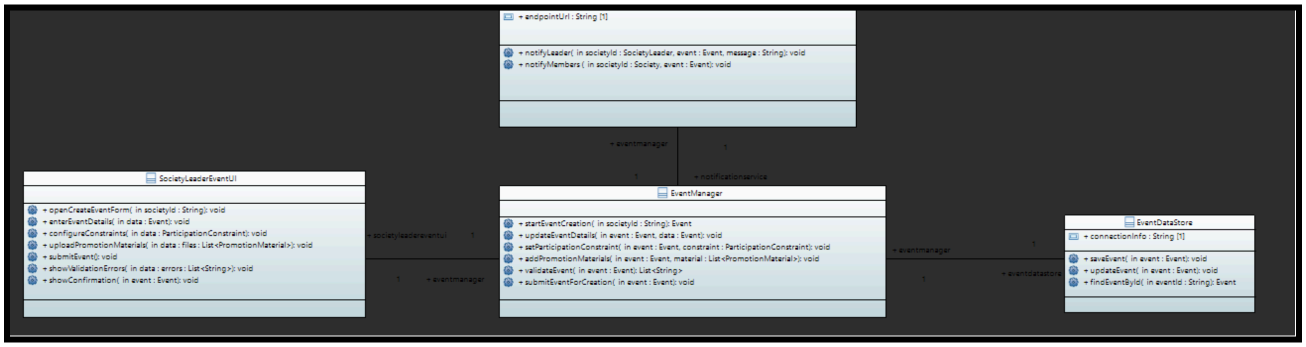
The Boundary Class (SocietyLeaderEventUI) includes all of the methods used to interact with the user; i.e. to get input from the user (event title, description, time and date, etc.) and submit a form to create or modify the event and to display possible either error messages or success messages after submitting a form.

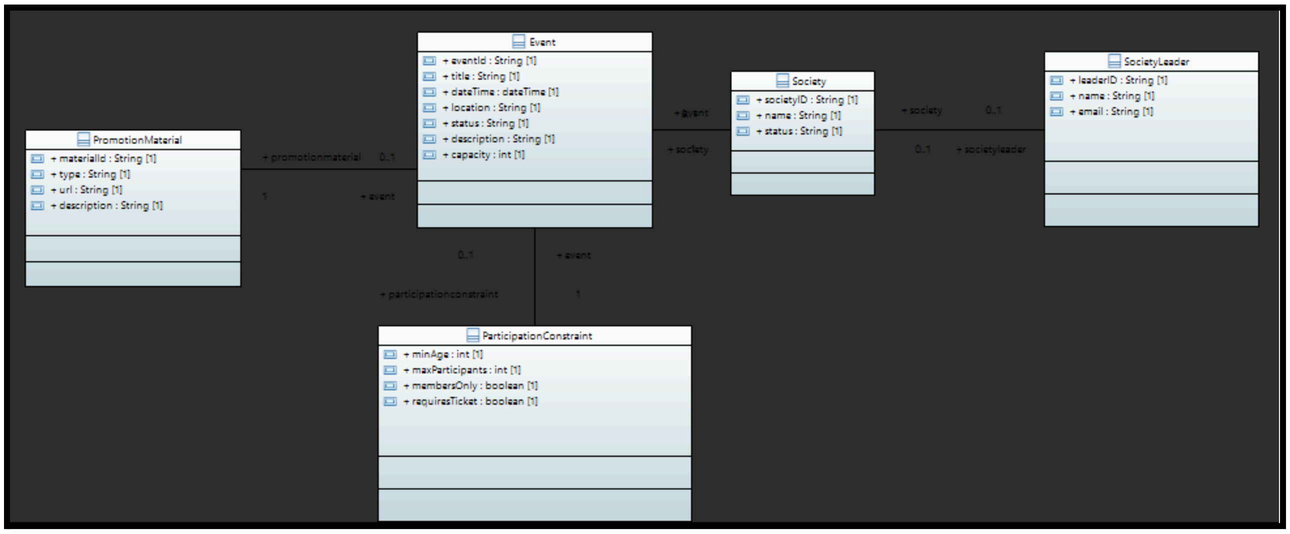
The main control class (EventManager), kind of manages the work flow for the creation/modification of an event. It receives requests from the SocietyLeaderEventUI, creates an event using the data that got submitted, enforces the participation constraints defined by the event, and finally validates the event before sending it to be stored in the database through an EventDataStore boundary class that serves as a facade for the Data Storage Microservice that I have previously defined in the Architectural Design.

The central domain entity (the Event class), represents a set of key properties (identifier, title, description, time and date, location, capacity, status and reference to the society that owns the event), and each event can have zero or more ParticipationConstraint objects associated with it. A ParticipationConstraint object defines constraints related to membership, the number of participants allowed, eligibility criteria, etc. As well as that, an event can have one or more PromotionMaterial objects associated with it too; each promotion material object defines a visual representation of the event (e.g. a poster or image).

The design also includes two simple entity classes, Society and SocietyLeader. The Society and SocietyLeader entities include attributes representing the society that owns the event and the current society leader that is creating/modifying the event.

Finally, the design also contains two service facade classes, NotificationService and EventDataStore, that represent the external notification and data storage services that were identified in the Component Architecture. When creating an event, the EventManager will invoke the NotificationService to send notifications to the leaders and members as well as use the EventDataStore to save/retrieve events.


+ endpointUrl : String [1]
+ notifyLeader(in societyId : SocietyLeader, event : Event, message : String): void
+ notifyMembers (in societyId : Society, event : Event): void
+ eventmanager
1
1
+ notificationservice
SocietyLeaderEventUI
+ openCreateEventForm(in societyId : String): void
+ enterEventDetails(in data : Event): void
+ configureConstraints(in data : ParticipationConstraint): void
+ uploadPromotionMaterials(in data : files : List<PromotionMaterial>): void
+ submitEvent(): void
+ showValidationErrors(in data : errors : List<String>): void
+ showConfirmation(in event : Event): void
+ societyleadereventui
1
1
+ eventmanager
EventManager
+ startEventCreation(in societyId : String): Event
+ updateEventDetails(in event : Event, data : Event): void
+ setParticipationConstraint(in event : Event, constraint : ParticipationConstraint): void
+ addPromotionMaterials(in event : Event, material : List<PromotionMaterial>): void
+ validateEvent(in event : Event): List<String>
+ submitEventForCreation(in event : Event): void
+ eventmanager
1
1
+ eventdatastore
EventDataStore
+ connectionInfo : String [1]
+ saveEvent(in event : Event): void
+ updateEvent(in event : Event): void
+ findEventById(in eventId : String): Event



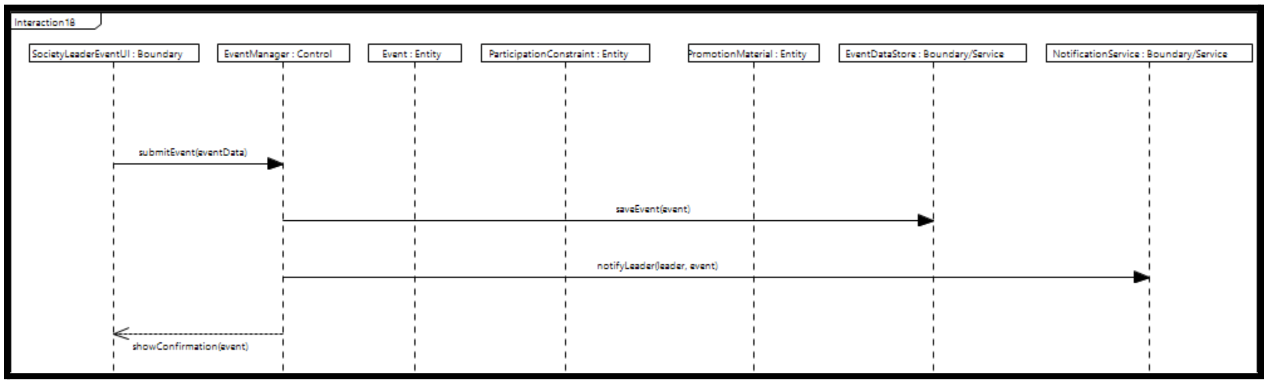
Zoom in for better visualization or see the .png file uploaded in papyrus.

In addition to the Structural Class Model, a Sequence Diagram was developed to illustrate the Dynamic Behavior of the "Create Event" Use Case of the Society Leader Application. This Sequence Diagram illustrates the interaction of the main Boundary, Control and Service Classes of the use case.

The Interaction of this diagram starts as the SocietyLeaderEventUI issues a submitEvent(eventData) request to the Event Manager (boundary). The Event Manager utilizes the data from the request to create or update an Event Entity Instance; and based upon the Participation Constraints previously defined for this application, the Event Manager then validates the submitted event data. Then once the submitted event data is validated, the Event Manager also invokes the saveEvent(event) method of the Event Data Store (service), which stores the Event within the underlying Data Storage Service of the application.

Following the successful persistence of the Event, the Event Manager notifies the External Notification Microservice via the invocation of the notifyLeader(leader, event) method of the NotificationService(boundary), which allows the appropriate society leaders and/or members to be notified about the creation of this new event.

Finally, the Event Manager returns control to the SocietyLeaderEventUI, where a showConfirmation(event) message is issued to allow the UI to display a confirmation dialog indicating that the event has been successfully created.


Interaction18
SocietyLeaderEventUI : Boundary
EventManager : Control
Event : Entity
ParticipationConstraint : Entity
PromotionMaterial : Entity
EventDataStore : Boundary/Service
NotificationService : Boundary/Service
submitEvent(eventData)
saveEvent(event)
notifyLeader(leader, event)
showConfirmation(event)

Zoom in for better visualization or see the .png file uploaded in papyrus.

APPENDIX VIII .

References

The references listed below reflect the core content of the COMP5047 lectures and coursework, with ISO/IEC 25010 (2011).

Zhu, H. (2025) COMP5047 Lecture 1: The Nature of Software. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 2: Software Quality. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 3: Use Case and Scenario Analysis. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 4: Activity Diagrams. Oxford Brookes University. [Powerpoint slides].

ISO/IEC (2011) Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models (ISO/IEC 25010:2011). Geneva: International Organization for Standardization.

Brooks, F. P. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, pp. 10–19.

Evans, E. (2003) Domain-Driven Design – Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

ISO/IEC 9126-1 (2001) Software Engineering — Product Quality — Part 1: Quality Model. Geneva: International Organization for Standardization.

Zhu, H. (2025) COMP5047 Lecture 8: Software Design – Structural Design. Oxford Brookes University. [Powerpoint slides].