19324455 - USU Student App - Individual Section

1. ***USU Student App*: a mobile app to run on smart phones for students to participate in the student union's activities.**

Task 1

Teamwork Report

At the beginning of the project we randomly assigned roles to team members. Each member was then required to model their individual subsystem.

Github was used as a final repository to ensure we were able to submit our work together at the end of the project. Meeting notes for the first few meetings were stored in our group repository to show early understanding of responsibilities and progress.

As the project progressed in person meetings were heavily relied on especially nearing the deadline. This involved the discussion of challenges and reviewing and suggesting changes to others' work.

While documentation of meetings became less formal and documented over time the experience highlighted the importance of documentation of in person meetings and will be valuable experience to take into future projects.

Individual Report

My role individually

My individual role in the team was the USU student application. This involved using the case study to understand the requirements and producing the relevant UML diagrams. This included a use case diagram, activity diagram, class diagram, component diagram and sequence diagram. My responsibilities were to maintain consistency throughout the diagrams but also ensuring my component diagram was aligned with the whole system.

This role required me to make design decisions regarding the chosen classes, components and activities. These decisions were made based on following the case study and consideration of the system as a whole.

My role as a team

I took the responsibility in ensuring that we remained in touch. This included receiving progress updates and trying to do work reviews during our meetings and trying to ensure work aligned.

Subsystems were assigned randomly which required the team to own and understand their role.I aimed to understand my subsystem to the best of my ability whilst considering the interactions with other systems.

Technical contributions.

My technical contribution was designing and modelling the USU Student Application subsystem.

- Selecting relevant use cases for my subsystem
- Modelling user-system interactions
- Designing the architecture.
- Identifying provided and required interfaces in my case
- Developing class and sequence diagrams.

Collaboration

Collaboration presented challenges particularly around subsystem alignment and responsibilities. This made the full system diagram challenging to complete. As each team member had focused on their own subsystem a cohesive full system understanding was challenging.

I attempted to support my team by reviewing their component diagrams and providing suggestions to the best of my ability. I did not have full knowledge of their subsystem so I could only focus on inconsistencies with mine and other subsystems.

Project Management tools

GitHub was used as a shared repository for diagrams and project notes. Earlier in the project notes and project coordination was stored in GitHub, later on we relied on in person understanding so more questions could be asked and further clarification could be given.

Not all meeting notes were documented in GitHub but it was still the location to upload all finished diagrams and shared materials.

Challenges

One of the main challenges faced was ensuring interfaces were consistent across the subsystems to show the interactions. This was particularly evident and ongoing when trying to create the full system component diagram. Balancing the individual work and the group work also presented challenges.

I addressed these challenges to the best of my ability by suggesting mutual updates but found in areas it was not quite good enough to ensure a cohesive diagram matching the requirements in every area.

Reflection

This project highlighted the importance of communication throughout the project and it potentially if done better would have mitigated our challenges. I realised even if individual components are clearly defined, integration is still extremely difficult.

In future projects I would prioritise better communication and having more regular work reviews this would prevent mass inconsistencies addressing a few when they arise. Documentation of decisions and progress would also be a priority and conversations can easily be misinterpreted or forgotten.

Task 2

Functional requirement.

FR-ST-6: Participation in Events. The system should provide a facility to the student members to search for events, to view detailed information of events, and to subscribe to the updates of selected events, and to register to and/or buy tickets of the event of his/her choice.

Quality requirement

Security and privacy protection

The system as a whole has to deal with sensitive information, specifically the subsystem USU student application is dealing with sensitive student information. This means the data needs to be protected effectively, accounts need to be secure and roles need to be clearly defined.

Due to the nature of the data the application needs to be operating in accordance with GDPR regulations. It is important to also ensure that it aligns with other University related systems and it meets the requirements defined in the privacy policy to ensure the users are aware of how their data is going to be handled.

Student data will require encryption securing the communication. This prevents the ability to eavesdrop on secure user information while it is being transferred. When being stored it is required to scramble sensitive data into a key which makes it almost impossible to break.

Roles need to be clearly defined to ensure that a logged in student is only able to access information specific to their account and when a sign in attempt is made there is a clear log displaying this information. Without this it could lead to other users gaining access to sensitive information and going undetected due to no logs..

Performance

The application should be responsive with a reasonable amount of users visiting, responsiveness should be maintained and operations should be able to be completed efficiently.

The application should allow users to browse the events smoothly and allow for easy registration events when there is traffic from a variety of Universities.

Notifications and updates should not show any delay as be as close to the real time as possible to ensure the user is able to action it in a timely manner.

Even during peak times such as freshers activities etc the application should remain consistently responsive.

Reliability

The application must always display accurate event availability information. The capacity should be updated along with the participants and cancellations. These updates should be visible a reasonable time after the action is completed in the backend. This will prevent students registering for events that are full or that have been cancelled.

If the user registration fails due to network or service issues the app must not submit incomplete participations or multiple as a result. The student should be notified there is an issue and they should be able to retry registration without corrupting the existing participation data.

The application should be constantly during peek times with consideration for potential issues on a one off basis. Making sure students can still register successfully near deadlines and when the application is experiencing high traffic.

Scalability

The USU student application should be able to scale to support an increase in users nationwide. The application should continue functioning features such as search and registration and should be able to handle many students from many universities using the app simultaneously.

The event should be able to handle spikes in traffic to the application for cases such as large and extremely popular events being posted and push notifications that will lead to many students clicking on it to visit the application.

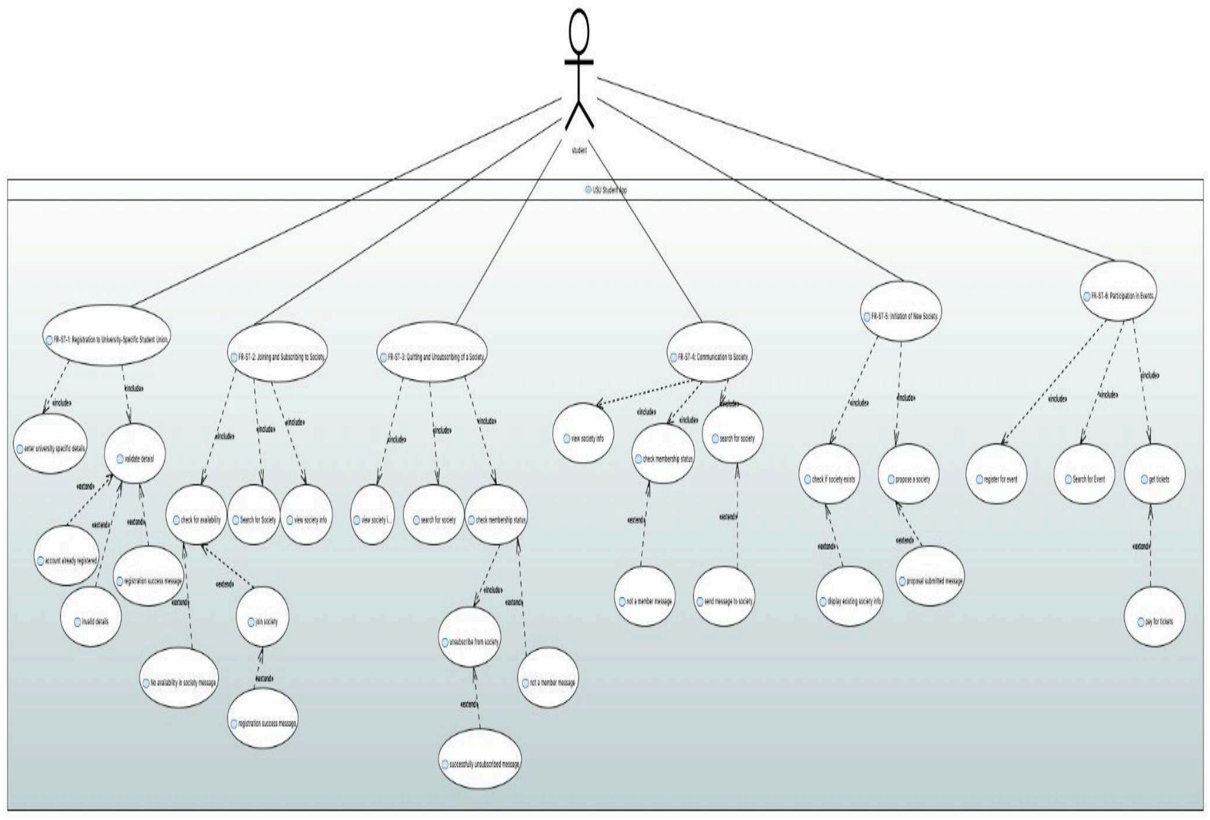
Scaling across all universities must be supported so registration can handle many different societies, many events and filtering via university specific events.

The application must balance processed information by balancing the load in the backend preventing bottlenecks that impact performance when the number of users increases.

Task 3a

The use case diagram for USU Student application shows the main operations the Student can perform on the application. The actor is the Student and the system interaction is shown in the diagram that matches the systems functional requirements.

The key use cases are to register, join a society, unsubscribe from a society, initiate a new society and participate in events. The diagram focuses specifically on what the student can do.


student
USU Student App
FR-ST-1: Registration to University-Specific Student Union.
enter university specific details
validate detaisl
account already registered
invalid details
registration success message
FR-ST-2: Joining and Subscribing to Society.
check for availability
Search for Society
view society info
No availability in society message
join society
registration success message
FR-ST-3: Quitting and Unsubscribing of a Society.
view society i...
search for society
check membership status
unsubscribe from society
not a member message
successfully unsubscribed message
FR-ST-4: Communication to Society.
view society info
check membership status
search for society
not a member message
send message to society
FR-ST-5: Initiation of New Society.
check if society exists
propose a society
display existing society info
proposal submitted message
FR-ST-6: Participation in Events.
register for event
Search for Event
get tickets
pay for tickets
«include»
«extend»

Task 3b

The activity model diagram shows the flow of Event Participation. The two swimlanes Student and application show how the responsibilities are divided.

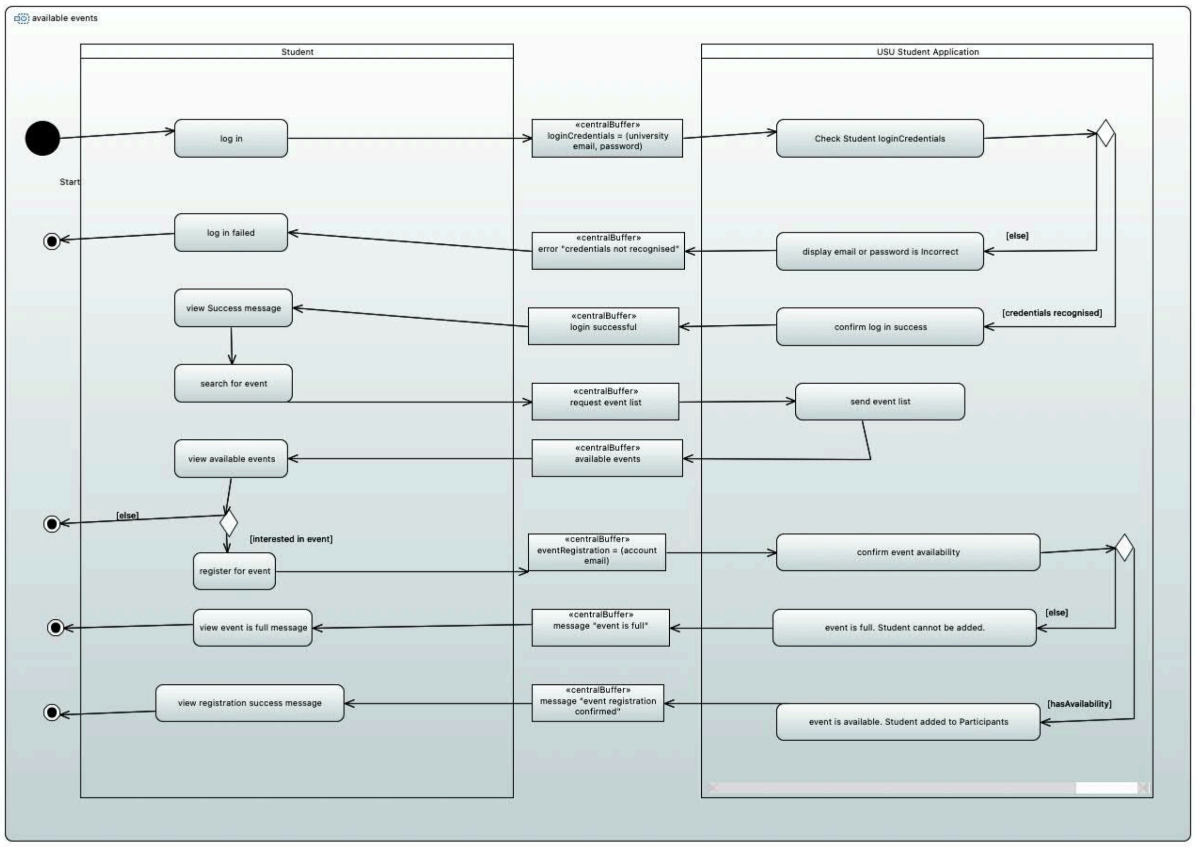
The student enters their login information which is validated by the system. The decision node checks the credentials and if they are not valid the event participation flow ends there.

If they are valid the user will see a login success message and then be able to search for events where they will be able to see all the current events that are available if none of them are of interest the flow ends there.

If the student sees an event they would like to register for their interest is sent over and availability is checked.

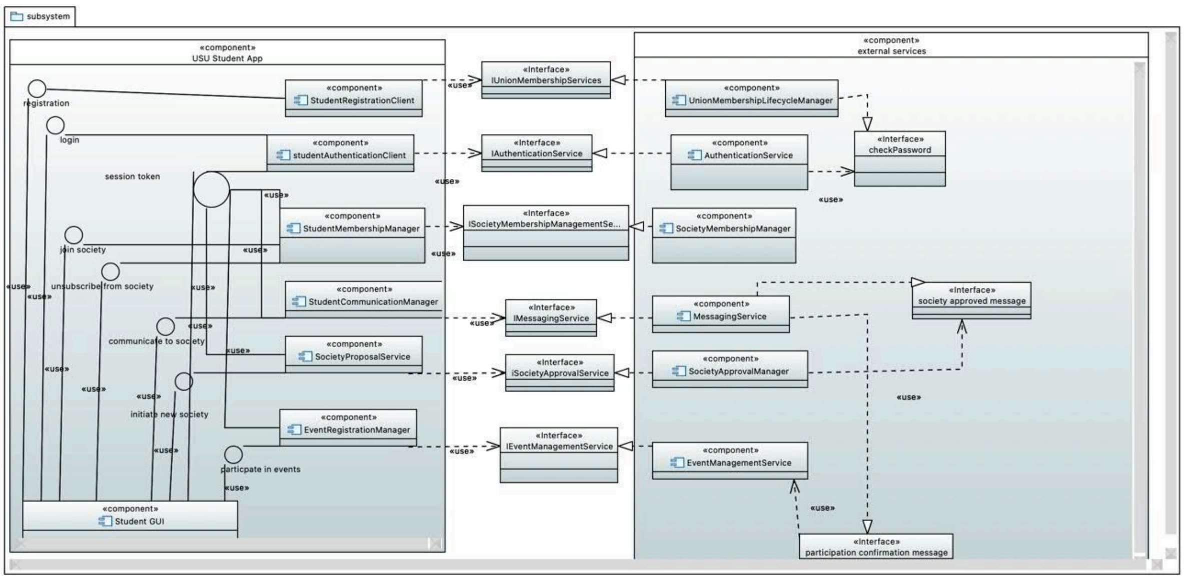
If the event is full they will receive an event with a full message and if there is space they will see a registration success message.

The diagram shows where decisions are made and the user choices and how the student interacts with the system when participating in an event.


available events
Student
USU Student Application
Start
log in
«centralBuffer» loginCredentials = (university email, password)
Check Student loginCredentials
[else]
display email or password is Incorrect
«centralBuffer» error "credentials not recognised"
log in failed
[credentials recognised]
confirm log in success
«centralBuffer» login successful
view Success message
search for event
«centralBuffer» request event list
send event list
«centralBuffer» available events
view available events
[else]
[interested in event]
register for event
«centralBuffer» eventRegistration = (account email)
confirm event availability
[else]
event is full. Student cannot be added.
«centralBuffer» message "event is full"
view event is full message
[hasAvailability]
event is available. Student added to Participants
«centralBuffer» message "event registration confirmed"
view registration success message

<u>Task 4a</u>

The component diagram shows how the application is split into components each responsible for a specific application function. The student GUI interacts with the components by using their provided interface. The components rely on external services for validation, approval, registration and messaging. Overall the diagram shows the components communication and which component uses it and which one provides it. The external services act as a clear representation of where the USU Student application is using services from the whole system providing a foundation for completing the team diagram.


subsystem
«component» USU Student App
«component» external services
registration
login
session token
join society
unsubscribe from society
communicate to society
initiate new society
particpate in events
«component» Student GUI
«component» StudentRegistrationClient
«component» studentAuthenticationClient
«component» StudentMembershipManager
«component» StudentCommunicationManager
«component» SocietyProposalService
«component» EventRegistrationManager
«Interface» IUnionMembershipServices
«Interface» IAuthenticationService
«Interface» ISocietyMembershipManagementSe...
«Interface» IMessagingService
«Interface» iSocietyApprovalService
«Interface» IEventManagementService
«component» UnionMembershipLifecycleManager
«component» AuthenticationService
«component» SocietyMembershipManager
«component» MessagingService
«component» SocietyApprovalManager
«component» EventManagementService
«Interface» checkPassword
«Interface» society approved message
«Interface» participation confirmation message
«use»

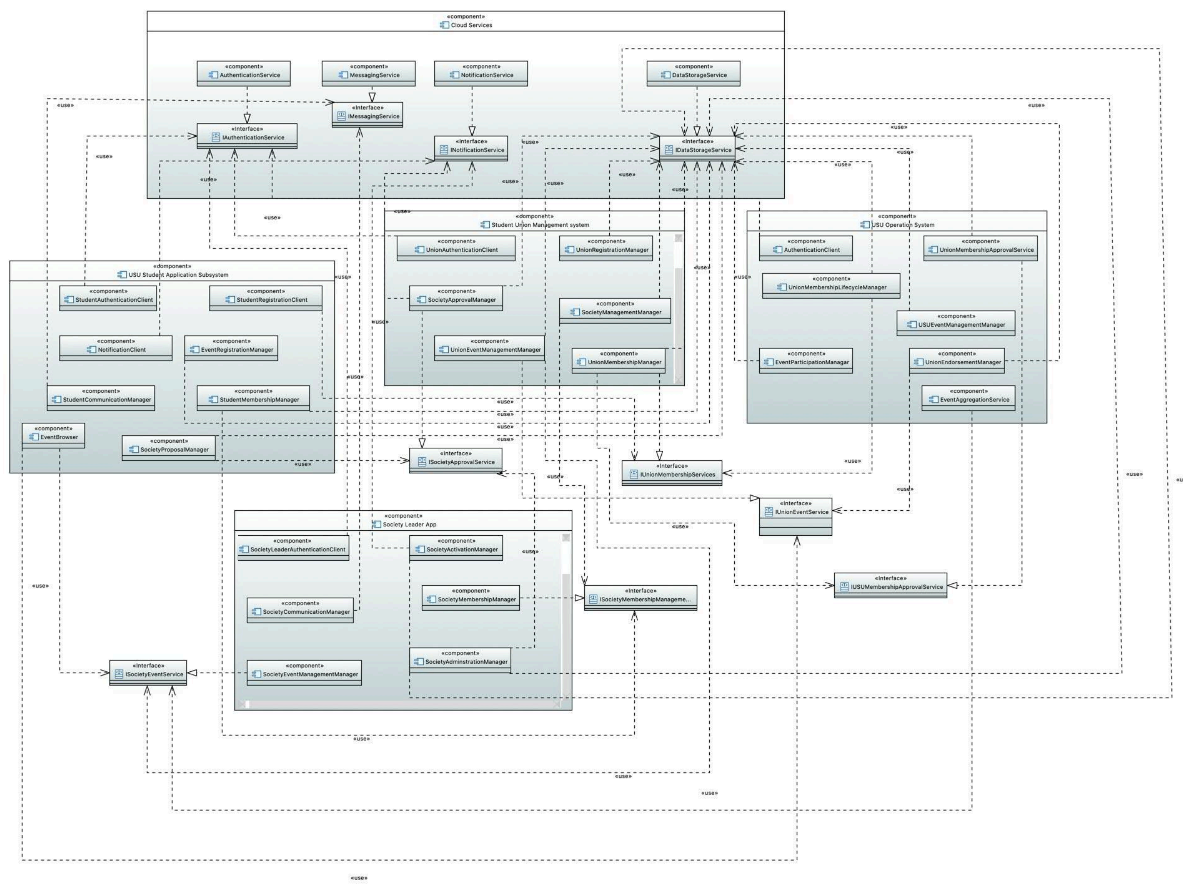
Note: some interfaces are shown as box and others lollipop as the format wouldn't work for some and it made the diagram look very unclear.

Task 4b

The whole system diagram shows all the team members' subsystems and how they are integrated through microservices. Each subsystem that provides or uses an interface is clearly defined. Shared cloud usage is also shown.

The microservice approach is shown by ISocietyMembershipManagement being provided by the Society Leader application and usage of this interface by the USU Student Application and the Student Union Management System. This separates the systems internal implementation while still providing the required functionality.

The interface communication ensures that there is not duplication of responsibilities across the system and still meets the requirements.


«component» Cloud Services
«component» AuthenticationService
«component» MessagingService
«component» NotificationService
«component» DataStorageService
«Interface» IMessagingService
«Interface» IAuthenticationService
«Interface» INotificationService
«Interface» IDataStorageService
«domponeht» Student Union Management system
«component» UnionAuthenticationClient
«component» UnionRegistrationManager
«component» SocietyApprovalManager
«component» SocietyManagementManager
«component» UnionEventManagementManager
«component» UnionMembershipManager
«component» USU Operation System
«component» AuthenticationClient
«component» UnionMembershipApprovalService
«component» UnionMembershipLifecycleManager
«component» USUEventManagementManager
«component» EventParticipationManagar
«component» UnionEndorsementManager
«component» EventAggregationService
«domponent» USU Student Application Subsystem
«component» StudentAuthenticationClient
«component» StudentRegistrationClient
«component» NotificationClient
«component» EventRegistrationManager
«component» StudentCommunicationManager
«component» StudentMembershipManager
«component» EventBrowser
«component» SocietyProposalManager
«Interface» ISocietyApprovalService
«Interface» IUnionMembershipServices
«Interface» IUnionEventService
«Interface» IUSUMembershipApprovalService
«component» Society Leader App
«component» SocietyLeaderAuthenticationClient
«component» SocietyActivationManager
«component» SocietyMembershipManager
«component» SocietyCommunicationManager
«component» SocietyAdminstrationManager
«component» SocietyEventManagementManager
«Interface» ISocietyMembershipManageme...
«Interface» ISocietyEventService
«use»

Task 5a

The class diagram shows participation in events in the USU Student app and the classes and interfaces involved..

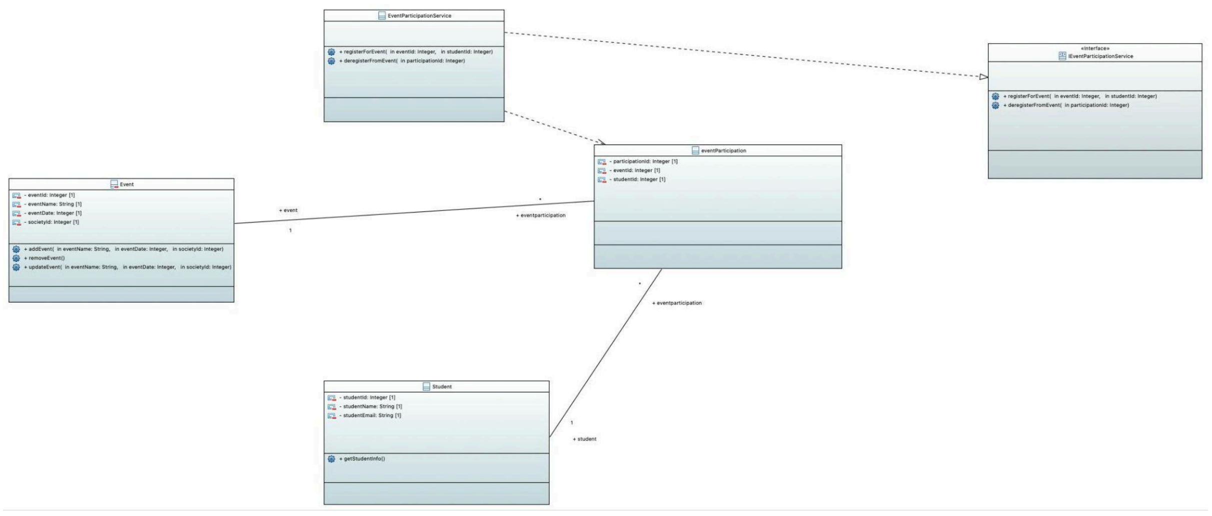
The three main classes are included in the diagram below.

Student - the application user that is able to participate in their chosen events.

EventParticipation - that links the student class with the event class holding the registration details which enables both registration and cancellation.

Event - where a posted event information is stored so it can be consistently added, updated and removed if required.

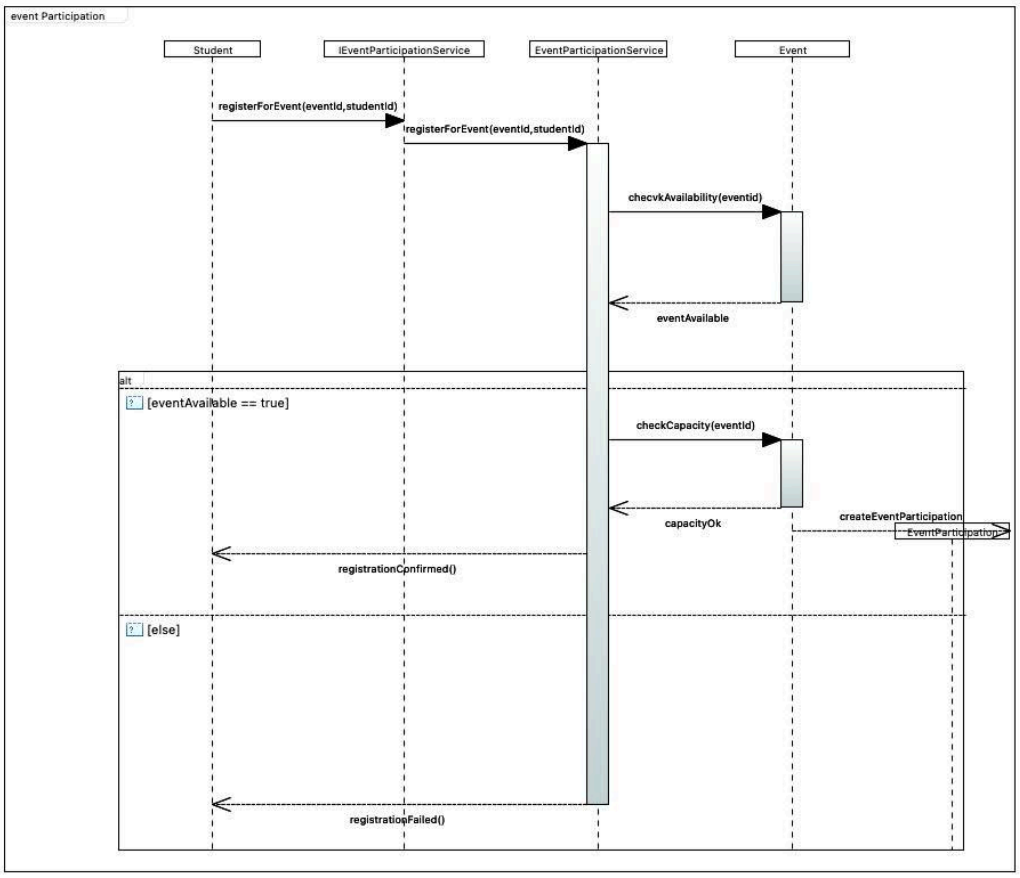
This structure shows the participation in events storing participation specific information in a consistent manner.


EventParticipationService
+ registerForEvent(in eventId: Integer, in studentId: Integer)
+ deregisterFromEvent(in participationId: Integer)
«Interface»
IEventParticipationService
+ registerForEvent(in eventId: Integer, in studentId: Integer)
+ deregisterFromEvent(in participationId: Integer)
Event
- eventId: Integer [1]
- eventName: String [1]
- eventDate: Integer [1]
- societyId: Integer [1]
+ addEvent(in eventName: String, in eventDate: Integer, in societyId: Integer)
+ removeEvent()
+ updateEvent(in eventName: String, in eventDate: Integer, in societyId: Integer)
+ event
1
*
+ eventparticipation
eventParticipation
- participationId: Integer [1]
- eventId: Integer [1]
- studentId: Integer [1]
*
+ eventparticipation
1
+ student
Student
- studentId: Integer [1]
- studentName: String [1]
- studentEmail: String [1]
+ getStudentInfo()

Task 5b

The sequence diagram shows the event participation component of the USU student application. It shows the interactions that happen when a student is attempting to register for an event.

The student attempts to register for the event through the event participation service interface which is sent to the event participation service. The service checks whether the selected event exists using the selected eventId. eventAvailable is returned if False the student receives a registration failed message. If eventAvailable,the capacity of the event is checked to ensure there is an available space for the student registering. If the capacity is ok a new participation is created and the student is alerted of registrationconfirmed.. If capacity is not ok the student is alerted of a registration fail.


event Participation
Student
IEventParticipationService
EventParticipationService
Event
registerForEvent(eventId,studentId)
registerForEvent(eventId,studentId)
checvkAvailability(eventid)
eventAvailable
alt
[eventAvailable == true]
checkCapacity(eventId)
capacityOk
createEventParticipation
EventParticipation
registrationConfirmed()
[else]
registrationFailed()