5047 Personal Part Coursework

Subsystem 1:

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

Quality Requirements

Performance

Scalability

Reliability

Security

FR-C1. Registration: The system should enable a user to register with the following data: (*security will be a consideration because personal information is being shared)*

- username,
- password,:
  - *implemented to keep accounts secure*
  - *Used so that no one with unauthorized access can get in*
- contact email address,
  - *if the account is breached or password needs to be changed it is a point of contact*
  - *Can be used for added security (e.g. 2factor authentication)*
- contact telephone number,
  - Point of contact if account is breached or password change
  - Can be used for security (2 factor authentication)
- shipping address [optional],
  - Keeping these registered with the account will increase performance because the users won't have to enter it every time
  - However it is also a security consideration
- payment card details [optional].
  - *Increase performance because it isn't entered every time*
  - *Lots of money can be lost if payment details are stolen, so a very big security concern*

How do the following non-functional requirements relate to the functional requirements of Registration

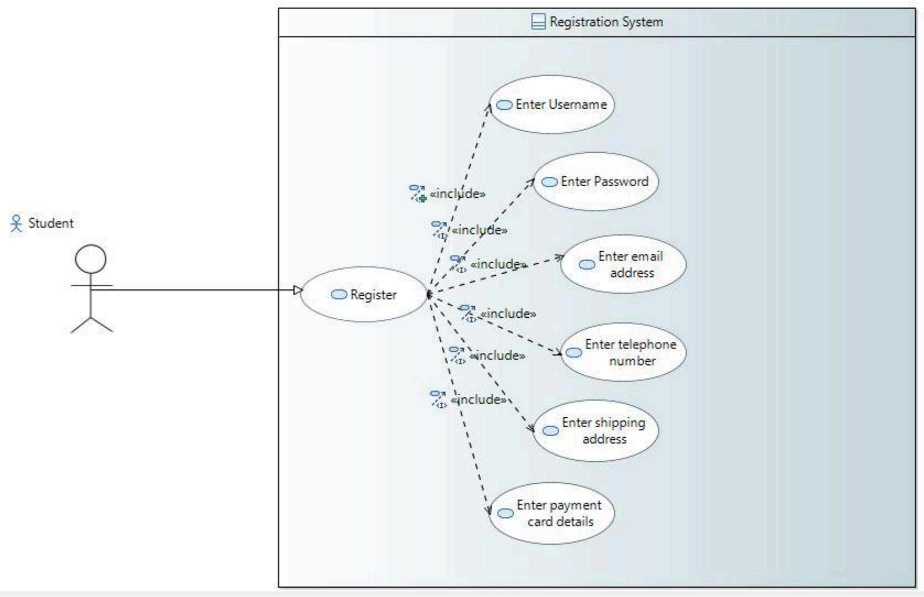
Performance: This is a quality requirement because while it's not a necessity, good performance and fast response time are important for users. For example, the system should be able to respond within two seconds when you have submitted your username and password.

Scalability: Scalability is the idea of whether the system will work if I had a much larger number of users, or did it on a much bigger scale. In this scenario, can the system handle receiving an increasingly growing number of registrations.

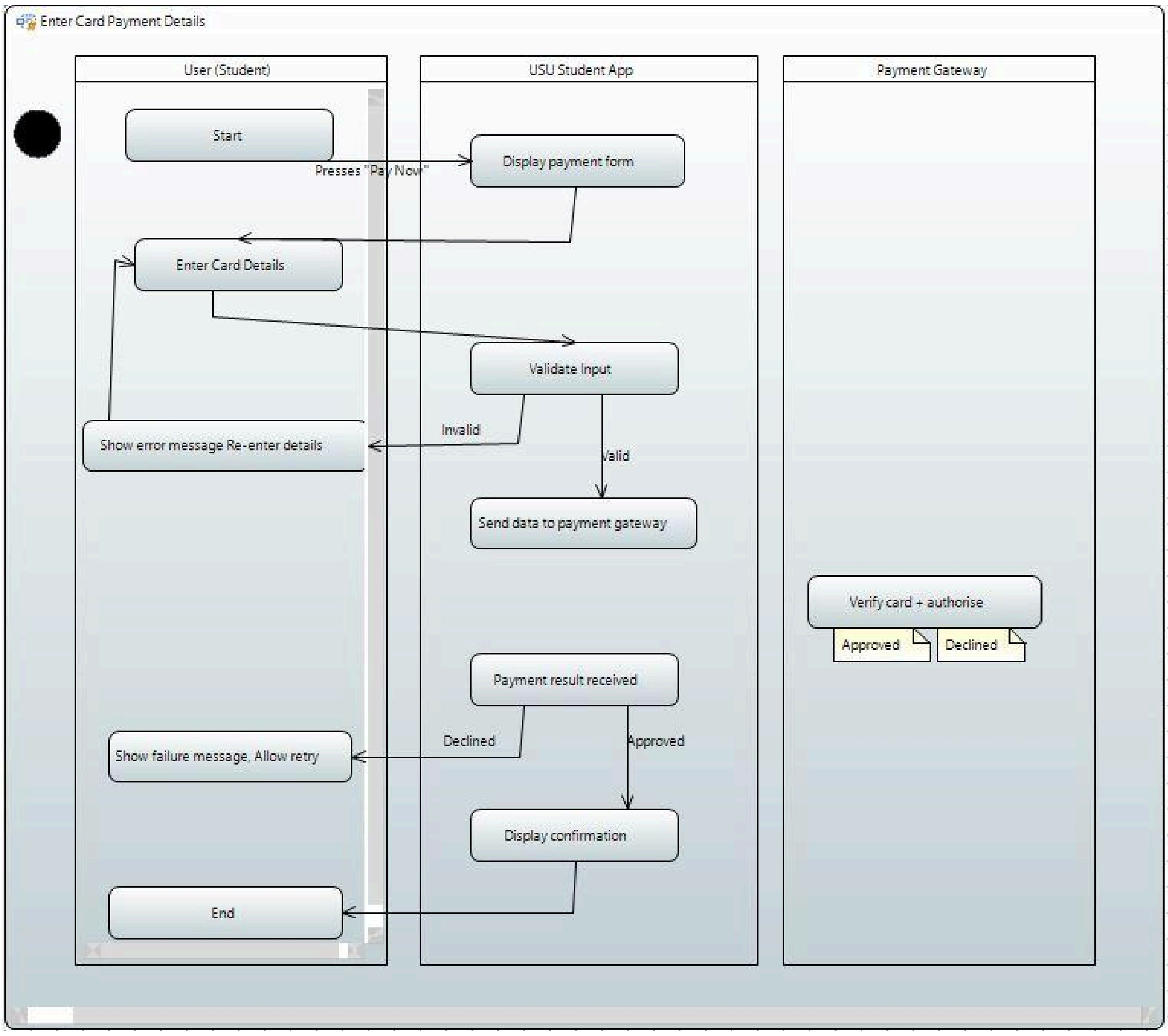
Reliability: This is whether the system performs consistently and as expected under different conditions. For example, the registration system should work with no crashes or errors, even with an increased number of users.

Security: In the registration section there is a lot of sensitive information collected from the users. Good security protects user data from unauthorized access.

Use Case Model



Activity Model



Software Architectural Design

First I'm going to bring together all the information needed for me to properly carry out this section from the case study, coursework specification, and other provided sources.

(a) *Architecture of the subsystem* (10 Marks, Individual effort): Each member of the team should produce an architectural design of your subsystem with focus on the microservices your subsystem provides and the microservices that your subsystem requests and other subsystems provides.

(b) *Students*. They are students at a university and the members of the university-specific student unions. They will use a mobile app to subscribe to societies, search for information about events, and register to events, receive information about university and national events, and participate in the events. A student can initiate a society and become the leader of a society subject to approval by the union.

FR-ST-1: Registration to University-Specific Student Union.

FR-ST-2: Joining and Subscribing to Society.

FR-ST-3: Quitting and Unsubscribing of a Society

FR-ST-4: Communication to Society

FR-ST-5: Initiation of New Society

FR-ST-6: Participation in Events

The main component is the USU Student App. The secondary component it interacts with is the Cloud in which the app will communicate with.

Within the student app we will have the components, StudentUI, EnrollSociety, StudentLogin, QuitSociety, InitiateSociety, EventParticipation, NotificationHandler.

Within the Cloud Backend System the components we will have are: AuthenticationManager, EnrollSociety, QuitSociety, SocietyManagement Service, UserProfile Service, Notification Service, EventParticipation Service.

The interfaces that are needed in this system are as follow: CheckPermission, Authenticate, EnrollForSociety, RequestInitiateSociety, LeaveSociety, ManageEventParticipation, SendUserNotification, RetrieveUserProfile, UpdateUserProfile.

Specification of Components

Student App Layer

Component	StudentUI
Description	A microservice running on the student app for the students to interact with the system.
Stereo Type	Component
Required Interfaces	UpdateUserProfile, RetrieveUserProfile
Provided Interfaces	

Component	EnrollSociety
Description	A microservice running on the student app to send a request for a user to join an society
Stereo Type	Component
Required Interfaces	Authenticate, EnrollForSociety
Provided Interfaces	

Component	StudentLogin

Description	Handles login workflow by collecting login input from the student and sending authentication requests to the backend service
Stereo Type	Component
Required Interfaces	Authenticate
Provided Interfaces	

Component	QuitSociety
Description	Handles the process of a student leaving/quitting a society, and sends a request to the backend service for the user to be removed from that society
Stereo Type	Component
Required Interfaces	LeaveSociety
Provided Interfaces	

Component	InitiateSociety
Description	A microservice running on the student app to request that a student be permitted to create a society through an application, and then to initiate it on the cloud system if it has been accepted
Stereo Type	Component
Required Interfaces	RequestInitiateSociety
Provided Interfaces	

Component	EventParticipation
Description	A microservice running on the student app to allow students to search for, view, and receive notifications for events
Stereo Type	Component
Required Interfaces	ManageEventParticipation
Provided Interfaces	

Component	NotificationHandler
Description	A microservice running on the student app to manage the delivery and display of notifications for each user based on the societies and events they're involved in.
Stereo Type	Component
Required Interfaces	SendUserNotification
Provided Interfaces	

<u>Cloud System - Services Layer</u>

Component	EnrollSociety
Description	Microservice on the backend that receives the request made by EnrollSociety on the app and starts the enrollment process, including validating the rules, records and checking authentication, before enrolling them onto the society.
Stereo Type	Service
Required Interfaces	CheckPermission, Authenticate
Provided Interfaces	EnrollForSociety

Component	QuitSociety
Description	Microservice on the backend that receives the request made by QuitSociety on the app and starts the leaving a society process
Stereo Type	Service
Required Interfaces	CheckPermission, Authenticate
Provided Interfaces	LeaveSociety

Component	AuthenticationManager
Description	This microservice handles the verification of student credentials, the token generated, and checking of the authentication policies.
Stereo Type	Service
Required Interfaces	

Provided Interfaces	CheckPermission, Authenticate

Component	SocietyManagement Service
Description	Manages the backend operations of all societies, creating, deleting, updating, managing members. Will interact with initiate society on the student app
Stereo Type	Service
Required Interfaces	CheckPermission
Provided Interfaces	RequestInitiateSociety

Component	UserProfile Service
Description	Manages backend operations for student profiles, creating, deleting, updating, event participation.
Stereo Type	Service
Required Interfaces	CheckPermission
Provided Interfaces	UpdateUserProfile, RetrieveUserProfile

Component	Notification Service
Description	Handles creation, management, and delivery of notifications to students, including event reminders, activity updates, system alerts.
Stereo Type	Service
Required Interfaces	
Provided Interfaces	SendUserNotification

Component	EventParticipation Service
Description	Handles requests from the app for users to search, view and receive notifications for events.
Stereo Type	Service
Required Interfaces	

Provided Interfaces	ManageEventParticipation

Specification of Interfaces

Name	CheckPermission	
Provider	Authentication Manager	
Operation	**Signature**	verifyAccess()
	Function	This verifies whether the user (student) has has the permissions necessary perform a specific action
Operation	**Signature**	decodeToken()
	Function	Its purpose is to get the information about the student from the token, for example Student ID, their societies, and permissions.

Name	Authenticate	
Provider	Authentication Manager	
Operation	**Signature**	studentLogin()
	Function	Logs the student into the USU system and return an access token
Operation	**Signature**	refreshToken()
	Function	Generates a new access token when needed

Name	EnrollForActivity	
Provider	EnrollActivity	
Operation	**Signature**	joinSociety()
	Function	Is to allow a student to enroll into a society, given they have the permissions of them
Operation	**Signature**	searchSociety()
	Function	Allows a student to search the database for societies that are offered at their university

Name	RequestInitiateSociety	
Provider	SocietyManagement Service	
Operation	**Signature**	submitApplication()
	Function	Lets the student submit an application to create a new society
Operation	**Signature**	trackApplication()
	Function	Student is able to check the approval status of their application by the union.

Name	LeaveSociety	
Provider	QuitSociety	
Operation	**Signature**	quitSociety()
	Function	Removes the student from a given society
Operation	**Signature**	
	Function	

Name	UpdateUserProfile	
Provider	UserProfile Service	
Operation	**Signature**	updateUserProfile()
	Function	Update information about the student, for example emails, phone number, etc
Operation	**Signature**	changePreferences()
	Function	Adjusts the student preferences on their profile

Name	RetrieveUserProfile	
Provider	UserProfile Service	
Operation	**Signature**	getUserProfile()
	Function	Returns the stored data for the specified student

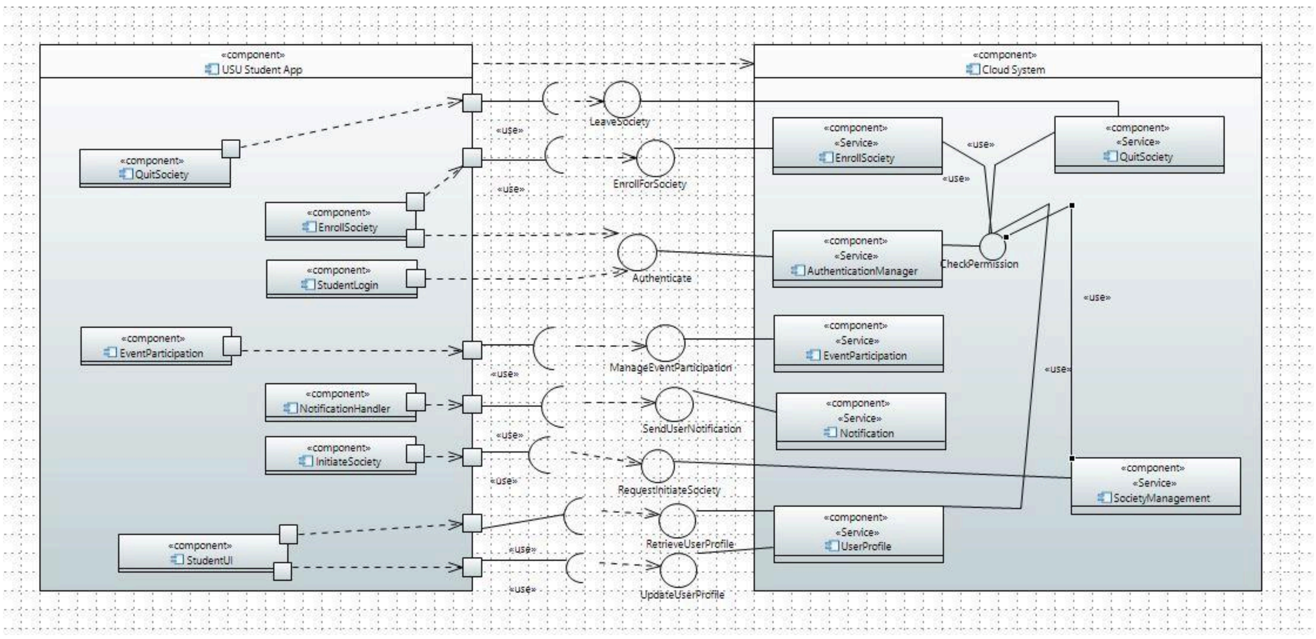
Operation	**Signature**	getStudentMemberships()
	Function	List the societies that the student is a part of

Name	SendUserNotification	
Provider	Notification Service	
Operation	**Signature**	sendNotification()
	Function	Sends a message to a specific society
Operation	**Signature**	sendEventUpdate()
	Function	Sends update to students who are subscribed to an event

Name	ManageEventParticipation	
Provider	EventParticipation	
Operation	**Signature**	searchEvents()
	Function	Allows student to browse or search for events
Operation	**Signature**	registerEvent()
	Function	Registers a student for an event the society is holding

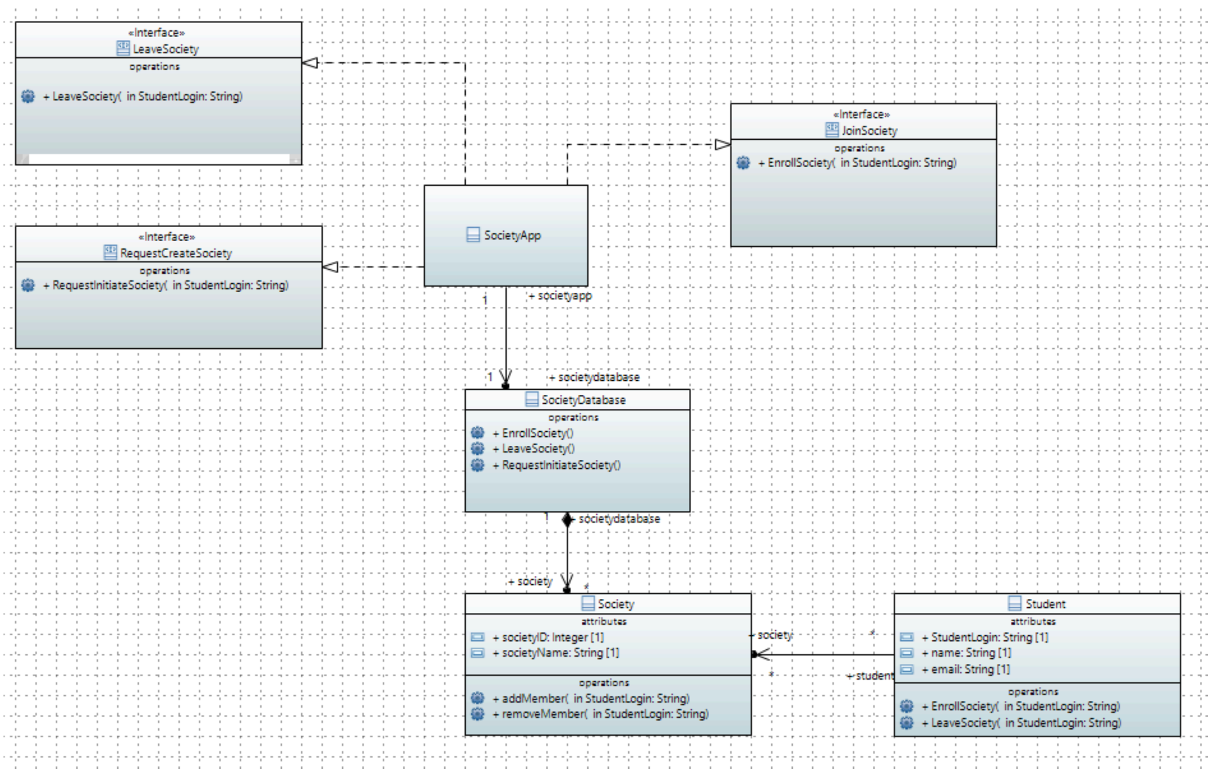
This is the process I followed to design the software architecture UML component diagram that I have completed.

1. Create the main two components, Student App and Cloud System
2. Add all the other microservice components in their corresponding system
3. Add required interface ports to app components
4. Add provided interface ports to app components
5. Connect each of the required interfaces to their matching provided ones using dependency lines
6. Make sure everything is represented clearly in the component diagram, and it is not too cluttered to read

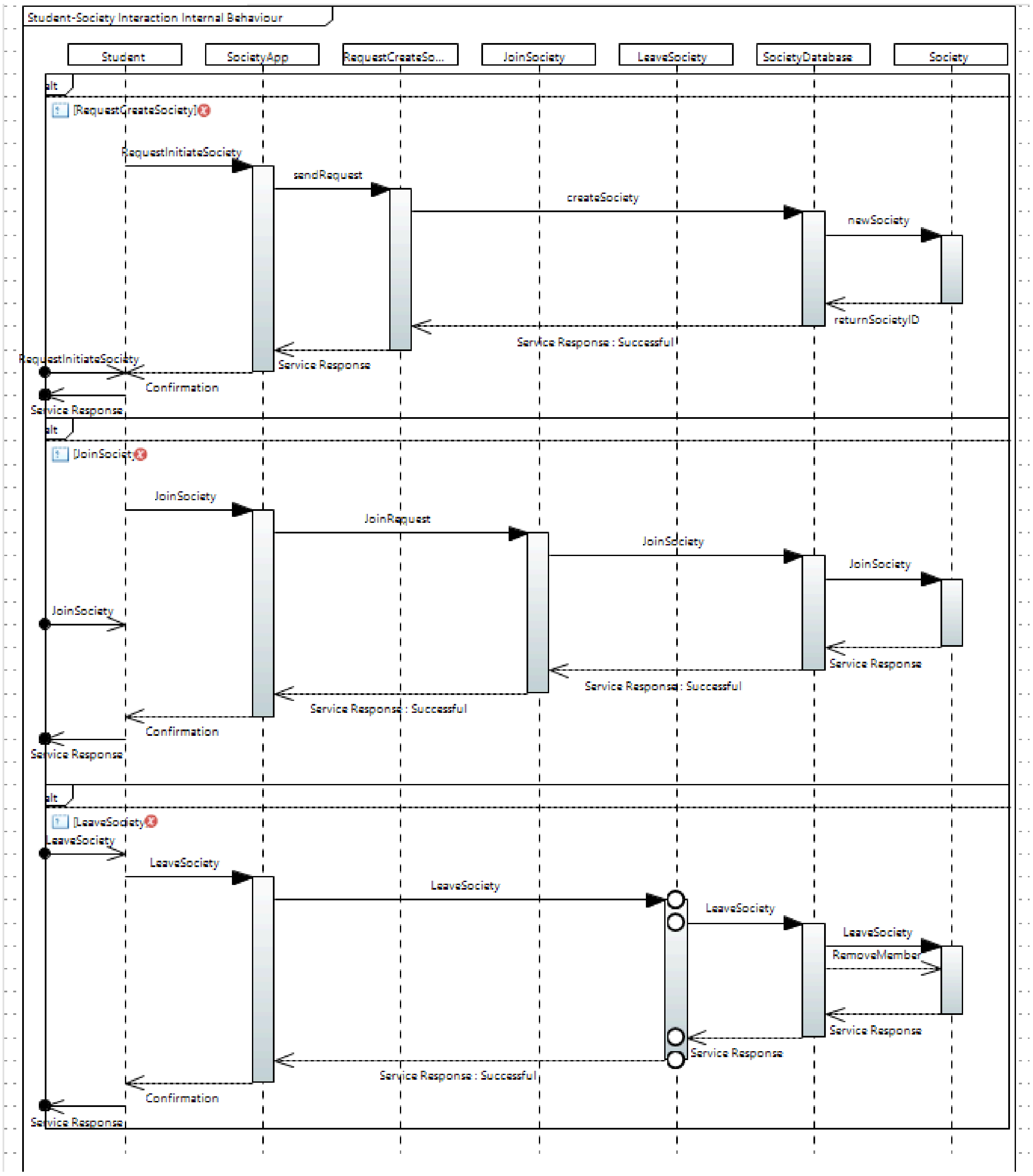

«component» USU Student App
«component» Cloud System
«component» QuitSociety
«component» EnrollSociety
«component» StudentLogin
«component» EventParticipation
«component» NotificationHandler
«component» InitiateSociety
«component» StudentUI
LeaveSociety
EnrollForSociety
Authenticate
ManageEventParticipation
SendUserNotification
RequestInitiateSociety
RetrieveUserProfile
UpdateUserProfile
CheckPermission
«use»
«component» «Service» EnrollSociety
«component» «Service» QuitSociety
«component» «Service» AuthenticationManager
«component» «Service» EventParticipation
«component» «Service» Notification
«component» «Service» SocietyManagement
«component» «Service» UserProfile

Software Detailed Design

Structural Model


«Interface» LeaveSociety
operations
+ LeaveSociety(in StudentLogin: String)
«Interface» JoinSociety
operations
+ EnrollSociety(in StudentLogin: String)
«Interface» RequestCreateSociety
operations
+ RequestInitiateSociety(in StudentLogin: String)
SocietyApp
1
+ societyapp
1
+ societydatabase
SocietyDatabase
operations
+ EnrollSociety()
+ LeaveSociety()
+ RequestInitiateSociety()
1
+ societydatabase
+ society
*
Society
attributes
+ societyID: Integer [1]
+ societyName: String [1]
operations
+ addMember(in StudentLogin: String)
+ removeMember(in StudentLogin: String)
+ society
*
*
+ student
Student
attributes
+ StudentLogin: String [1]
+ name: String [1]
+ email: String [1]
operations
+ EnrollSociety(in StudentLogin: String)
+ LeaveSociety(in StudentLogin: String)

Behavioural Model



Individual Report on Teamwork

In regards to our team project, we met up once a week, mostly on Tuesdays after the practical, to discuss our progress on the project. All of us took in turns each week to lead the meeting, which roles included taking notes, taking lead on the agenda from the previous meeting, and making sure we kept on track. When I was in charge of the meeting, I think I should have tried to get my work checked more week by week, although we did do this, I was slow to do it at the start. Overall, I think we worked well together, despite having one group member not present for a large amount of time.

I think we worked well together when it came to parts of the system such as the whole system software architecture, and we were able to fit everything we wanted together.
Overall, I’m happy with how our group worked, and I think we were able to make the best use of our time possible, and make the most out of the different skills each member of the group had.