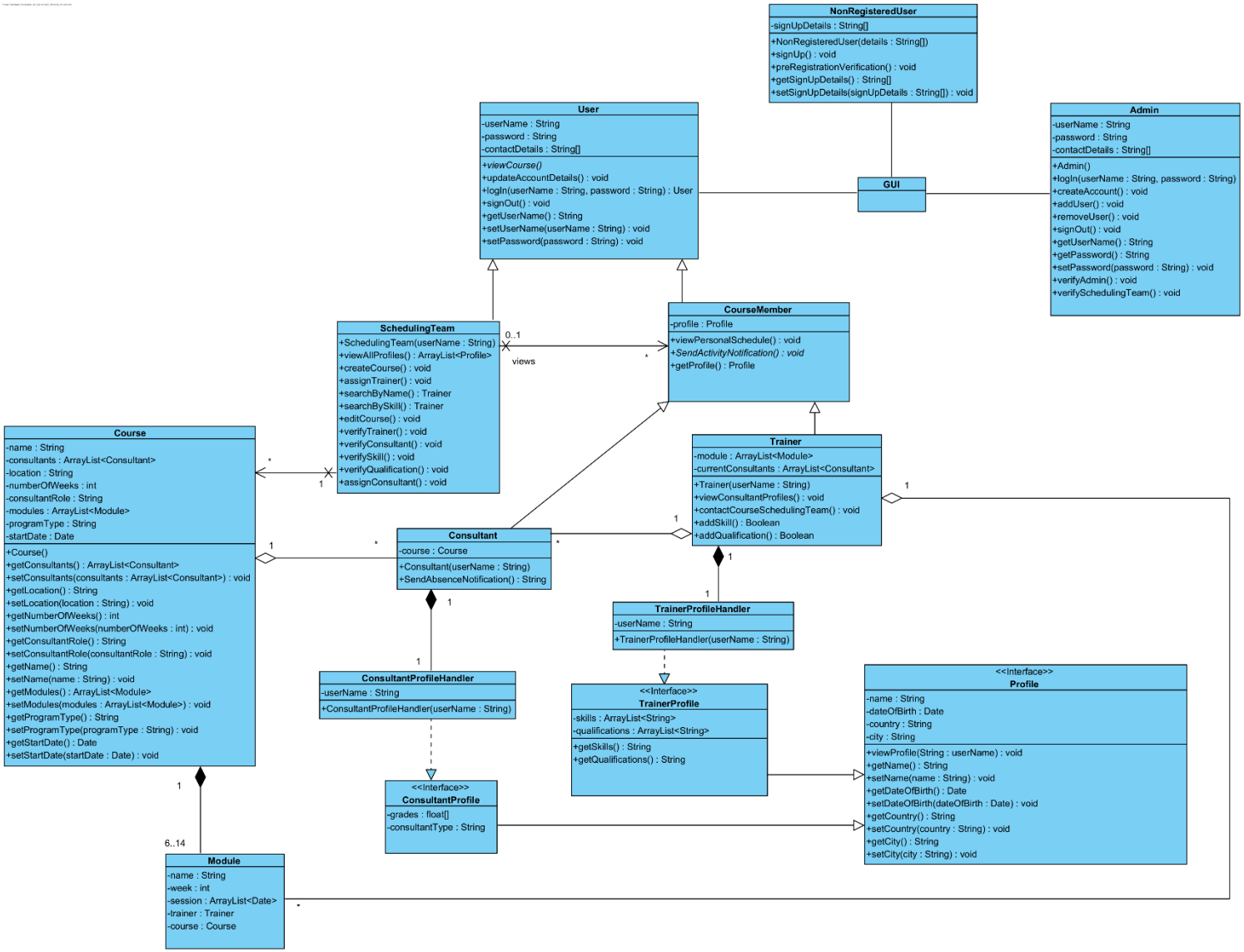
Group 18

FDM Trainer Skills and Availability

ECS506U Software Engineering Group Project

Design Report

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Explanation
User	RQ19	The class contains the variable username as String type.
Admin	RQ19	Like the User class this class contains the variable username as String type.
User	RQ20	The class contains the variable password as String type.
Admin	RQ20	Again, like the User class, this class also contains the variable password as String type.
User	RQ21	The class contains the variable contactDetails to store the user's contact details as a String type.
CourseMember	RQ26	Contains the method 'viewPersonalSchedule()' to allow the trainer to view their personal schedule.
CourseMember	RQ27	Contains the method 'viewPersonalSchedule()' to allow the consultant to view their personal schedule.
Module	RQ29	The training sessions are stored in an array list called Sessions.
Module	RQ30	The class contains a variable called Trainer which stores the allocated trainer for a session.
Course	RQ31	The class contains the array 'consultants' of type Consultants which stores a list of all consultants on the training session.
SchedullingTeam	RQ38	Contains the method 'assignTrainer()' to allow the scheduling team to assign trainers to courses.
SchedullingTeam	RQ39	Contains the method 'assignConsultants()' to allow the scheduling team to assign consultants to courses.
Course	RQ40	This class contains a variable called startDate which stores the starting date of a course. The variable is of type Date.
SchedulingTeam	RQ41	Contains the method 'createCourse()' to allow the scheduling team to create a course.
CourseMember	RQ51	The class contains the variable profile of type Profile.
Trainer	RQ55	Contains the method 'addSkill()' to allow the trainer to insert skills.
Trainer	RQ56	Contains the method 'addQualification()' to allow the trainer to insert qualifications.

3. Design Discussion

Differences between domain model/class diagram and why:

There were a few changes we made to our domain model that we implemented in our class diagram. First, we decided to remove the login and sign-up classes, and add the sign-up functionality to NonRegisteredUser and Login functionality to User instead as both classes were unnecessary and allowed us to assume that User was already authenticated and logged in.

We also removed the schedule class for viewing the course schedule. Instead, viewing course details will be implemented in the User class and viewing schedule will be implemented in the CourseMember class as each course member will have their own schedule and we believe it was more efficient to have each type of course member inherit it from the CourseMember class.

We also added a new module class to the course class which will be used to store weekly dates and other information about the module. This is because there will be multiple different modules for each course and each module's information will be different.

Multiple interfaces related to course member profiles were added, including a TrainerProfile interface for trainers and a ConsultantProfile interface for consultants which will store relevant information for each user. These were added due to feedback from FDM asking us to allow trainers to view consultant's profiles that they were teaching.

Design patterns used:

For design patterns, we applied abstraction-occurrence for the module/course classes. We decided to use this as for each course, we have multiple modules that represent what is being taught each week, and a course might even have multiple occurrences of the same module being taught but different trainers each time. Hence, we thought it was efficient to use this design pattern to avoid unnecessary duplication of data.

Justifications for design decisions (multiplicities, associations/aggregations/composition, generalizations):

We used inheritance and generalization for each type of user and course member as although they all have the same core functionalities such as viewing course details and profiles, each type of user has their own individual methods that other users do not have. For example, the scheduling team can create courses and assign trainers to them while trainers have an inbuilt way to contact the scheduling team. However, we separated User between SchedulingTeam and CourseMember, as both consultants and trainers have a lot of similar functionalities which differ from the scheduling team.

The reason we used generalization for distinct types of Profile interfaces (e.g., TrainerProfile and ConsultantProfile) instead of having one Profile interface is to make sure we make explicit use of the Interface Segregation Principle. This ensures that classes do not inherit methods that they do not need/use. For example, in the case of ConsultantProfile, we are storing the grades each consultant got for each module. However, since Trainers are only teaching the Module, they would have no reason to store grades on their profile, hence they inherit from a different interface.

As FDM asked us about verifying trainer skills and qualifications, we have added that functionality to the scheduling team, and since profiles are stored in the trainer or consultant objects, then anything regarding their profiles such as uploaded skills and qualifications will have to be handled through those. Which is why the scheduling team can view all course member profiles and trainers can view consultant profiles.

For the module class, we chose composition over aggregation as each course is composed of multiple modules. However, for the multiplicity we chose 6...14 as each course lasts 6 to 14 weeks therefore will have a lot more modules relative to the number of courses.

We chose to make all course members and the course class unnavigable to the scheduling team class. This is because while the scheduling team is responsible for creating courses and assigning trainers/consultants to those courses, they do not need to be seen or accessed by either class.

4. Sequence Diagrams

Edit course

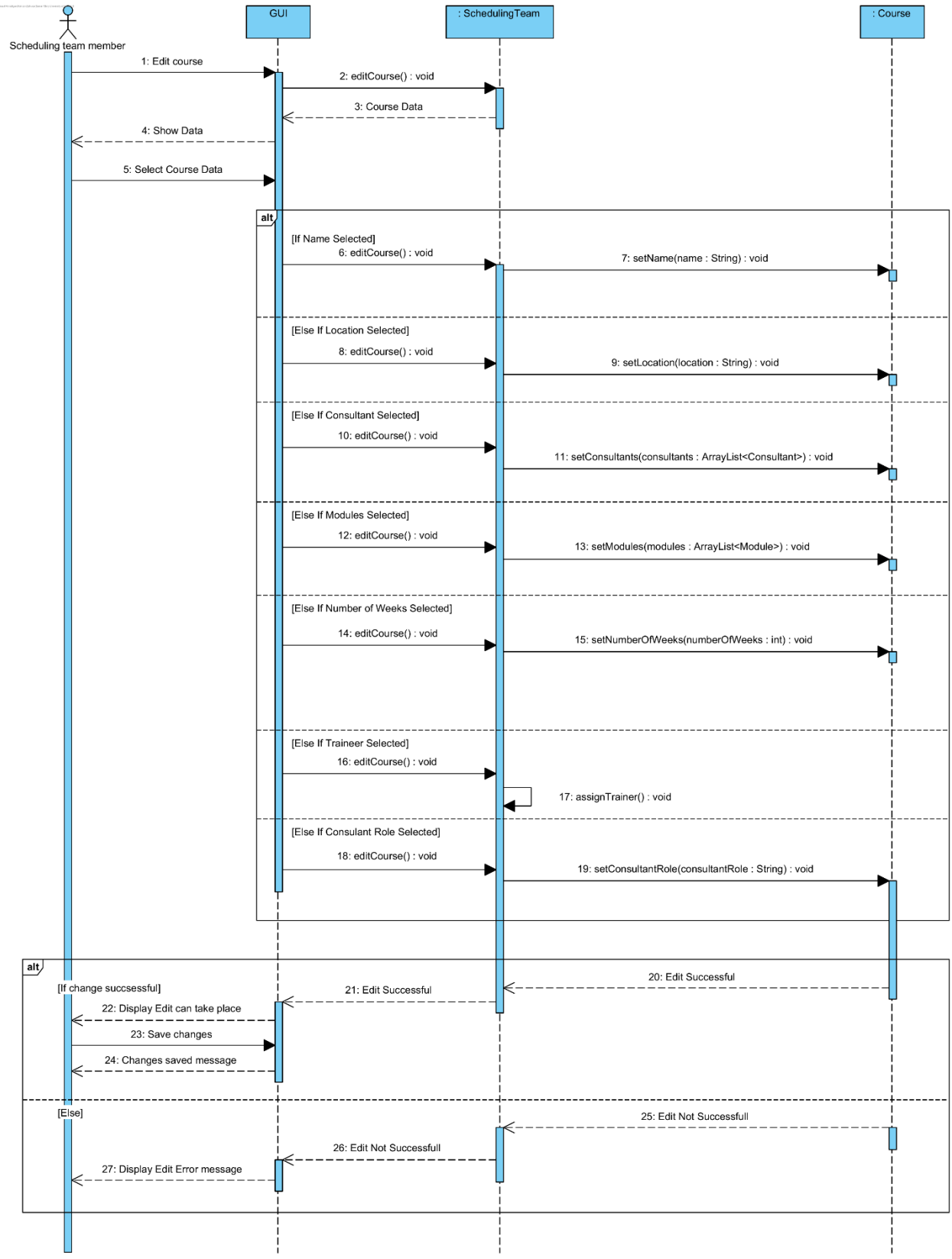
Edit course is one of the use cases that we describe in the Requirements Elicitation report.

Scenario:

- Scheduling team wants to edit a course

Pre-requisites:

- User is part of the scheduling team
- User is signed in
- All data entered by user is Valid


Scheduling team member
GUI
: SchedulingTeam
: Course
1: Edit course
2: editCourse() : void
3: Course Data
4: Show Data
5: Select Course Data
alt
[If Name Selected]
6: editCourse() : void
7: setName(name : String) : void
[Else If Location Selected]
8: editCourse() : void
9: setLocation(location : String) : void
[Else If Consultant Selected]
10: editCourse() : void
11: setConsultants(consultants : ArrayList<Consultant>) : void
[Else If Modules Selected]
12: editCourse() : void
13: setModules(modules : ArrayList<Module>) : void
[Else If Number of Weeks Selected]
14: editCourse() : void
15: setNumberOfWeeks(numberOfWeeks : int) : void
[Else If Traineer Selected]
16: editCourse() : void
17: assignTrainer() : void
[Else If Consulant Role Selected]
18: editCourse() : void
19: setConsultantRole(consultantRole : String) : void
alt
[If change succsessful]
20: Edit Successful
21: Edit Successful
22: Display Edit can take place
23: Save changes
24: Changes saved message
[Else]
25: Edit Not Successfull
26: Edit Not Successfull
27: Display Edit Error message

Sign Up

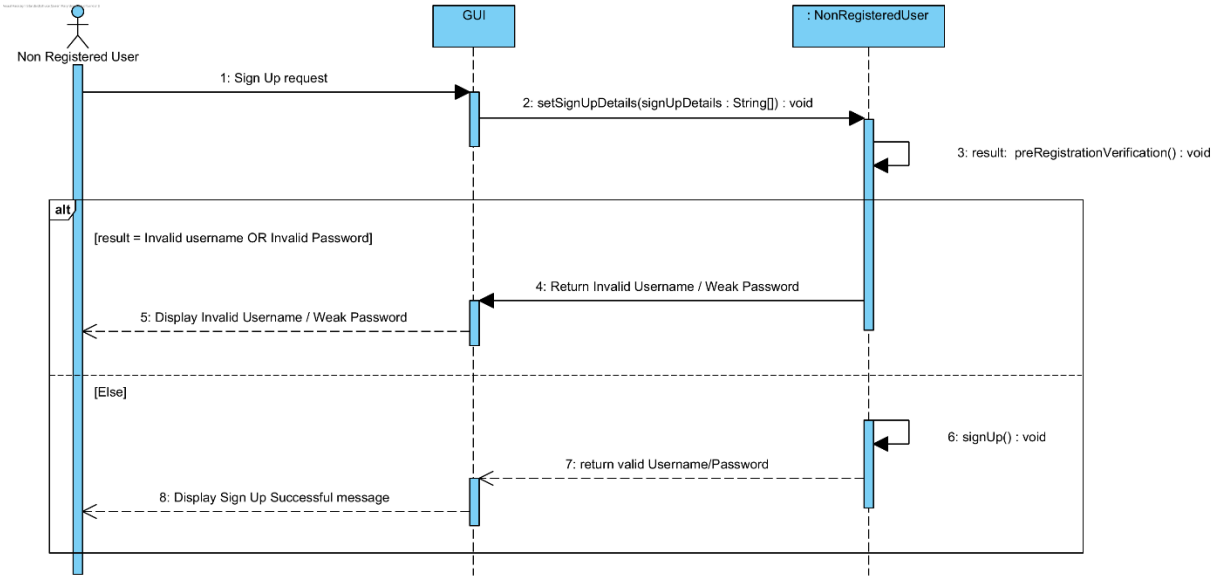
This use case is for non-registered users to sign-up to become a registered user.

Scenario:

- A non-registered user wants to sign up as either a consultant, trainer, system/data admin or scheduling team member

Pre-requisites:

- User has a valid email address
- User is signing up as either a consultant, trainer, system/data admin or scheduling team member only



Add Skills

Another key function is to be able to add skills, this to know what trainers can do what modules.

Scenario:

- Trainer wants to add more skills to their profile

Pre-requisites:

- Trainer is signed in