Group 13

Trainer Skills & Availability App

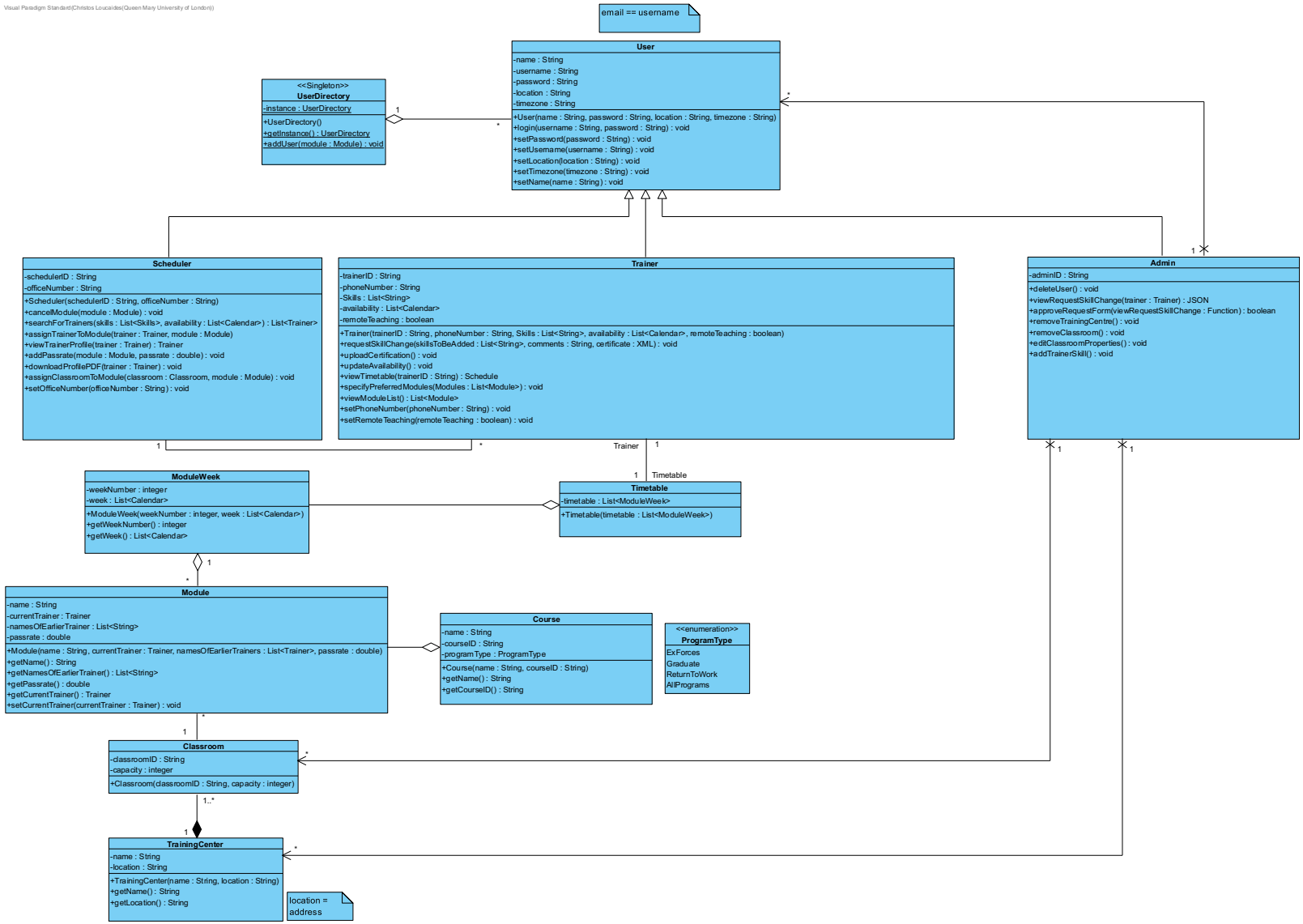
ECS506U Software Engineering Group Project

Design Report

Table of Contents

1. *Class Diagram* 2
2. *Traceability matrix* 3
3. *Design Discussion* 4
4. *Sequence Diagrams* 5

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Explanation
Module	RQ2	The module class has a string attribute to hold its name, another to hold its current trainer, a List<string> holding previous trainer's names and a double attribute representing the pass rate. The system will hold details on every different Module, identified by the name.
TrainingCenter, Classroom	RQ3	The training center class has two string attributes, one with the name of the center and another with the address of the center. Then, the Classroom class has an integer attribute to hold the capacity for each classroom. As there is a composite relationship between Classroom and TrainingCenter, classrooms can only appear as part of Training Centers. This means for each Classroom, the system is able to store its capacity and address (from the training center it belongs to). This is designed to reduce repeating unnecessary information, generally store information in a much more concise way, and create a much stronger, more obvious link between Classroom and TrainingCenter
User, Trainer, Calendar	RQ15	The trainer class is an implementation of the abstract class User. The user class holds a string attribute holding the location, and another holding the name. This is done as most if not all types of users will need to set their locations, and certainly their names, so this information is on the main User class rather than repeated across separate implementations. The trainer class then holds the phone number, as only trainers will need to be contacted by phone, and a list of Calendar dates representing their availability, as well as a list of strings that are the trainer's skills. Again, these pieces of information are only relevant to the trainer and so are on the trainer class only.
User, Scheduler	RQ23	Similarly to the trainer class, the email/username is stored in the abstract user class, as that will be used to login all types of users. Again, the location is on the user class too, as we can already see it is needed by different implementations of the user class, as well as the name. The scheduler will also hold the scheduler's office number, as a string attribute. This will most likely be unique to each scheduler and cannot be placed on the user class as not every user will have an office.

3. Design Discussion

There are a few differences between the entities in the domain model and the class diagram. The entities that were excluded from the diagram were ones related to the trainees, which are the consultant and graduate entities as it is not a requirement for the system. The general employee schedule was removed and a timetable entity for the trainers was put in place as it is also not a requirement for the system. This all comes as a conclusion from the analysis in the requirements elicitation stage.

A design pattern was implemented in the class diagram, namely the singleton pattern for the UserDirectory class. It ensures that only one instance of the UserDirectory is made, as it does not make sense to have several directories existing. It is also accessible to every class that needs it, which in this case are the users.

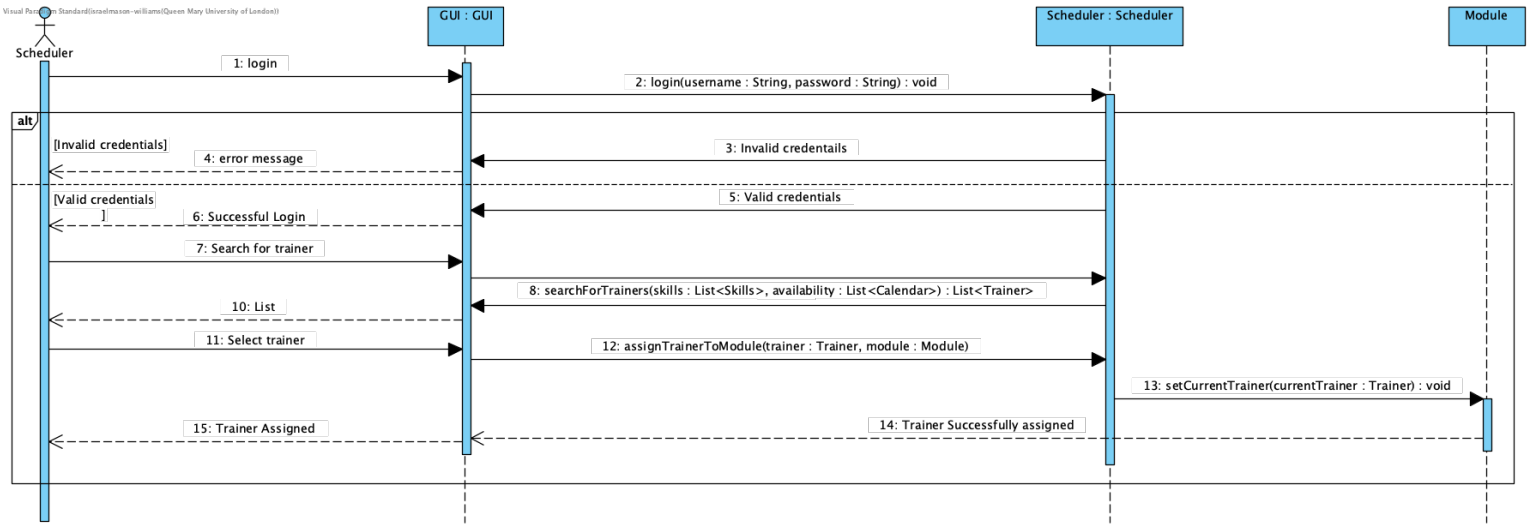
There are also design decisions that have been made that are justified. Some of these are in association/aggregation/composition and in multiplicities: for example, for UserDirectory – Trainer, there must be at least one user in the directory otherwise there is no point in having the directory, and users are not required to be in the directory and can exist independently from it. Classrooms must be part of a training centre as it is illogical to have them existing independently from a centre. Note that they also do not need to store a location value; each classroom is part of a training centre and the location data can be retrieved from there. Classrooms also can be part of as many modules as is required (perhaps one class takes part in multiple modules), and at least one classroom must exist. Some other considerations for association, aggregation, composition and multiplicities include each trainer having their own timetable, multiple modules making up a course and the specific week a module takes place being stored in a module week object.

Some other subtle design decisions have been made with regards to generalisations. ProgramType is an enumeration, so each course has a ProgramType value of (e.g., ExForces, Graduate, etc.) Adding on to this, AllPrograms is an enum literal as some courses may be common across programs. Similarly, schedulers, trainers and admins all have common features private to an instance of each class (like username, password, location, etc) and is accordingly shown as a generalisation of the user class.

4. Sequence Diagrams

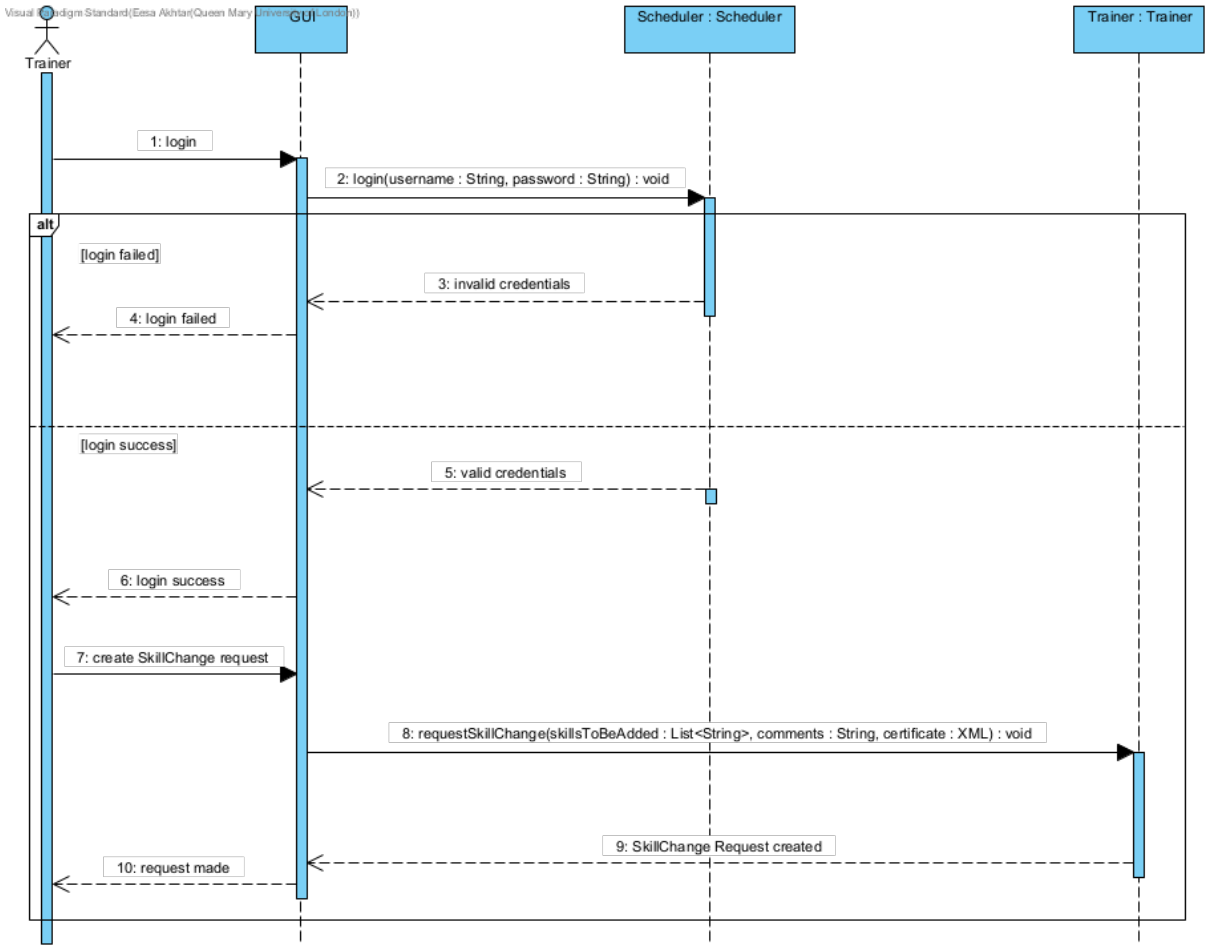
Assign Trainer to Module

This sequence diagram displays the order of events for a scheduler to assign a trainer to a specific module. This process utilises instances of scheduler and module respectively to perform this action and offers an alternative flow as outlined in the requirement elicitation document.



Request to Change Skills

This diagram shows the process of a Trainer submitting a skill-change request, this request gets logged onto our system which admins can then view and approve as needed.



View and Approve Skill Change Request

This diagram shows the process of an administrator approving a skill-change request. This involves them logging into the system, retrieving a skill-change request by viewing it, then subsequently approving it with a confirmation thrown back to the user.