COM2008-001
170164425
170163705
170164377
170164883

The University Of Sheffield.

Group Assessed Work Coversheet

Assessment Code: COM2008

Description: Group Systems Design Project

Staff Member Responsible: Simons, Dr Tony

Due Date: 30-11-2018 15:00:00

☑ **I certify that the attached is all my own work, except where specifically stated and confirm that I have read and understood the University's rules relating to plagiarism.**

I understand that the Department reserves the right to run spot checks on all coursework using plagiarism software.

Student Registration Numbers:

170164883

170164377

170163705

170164425

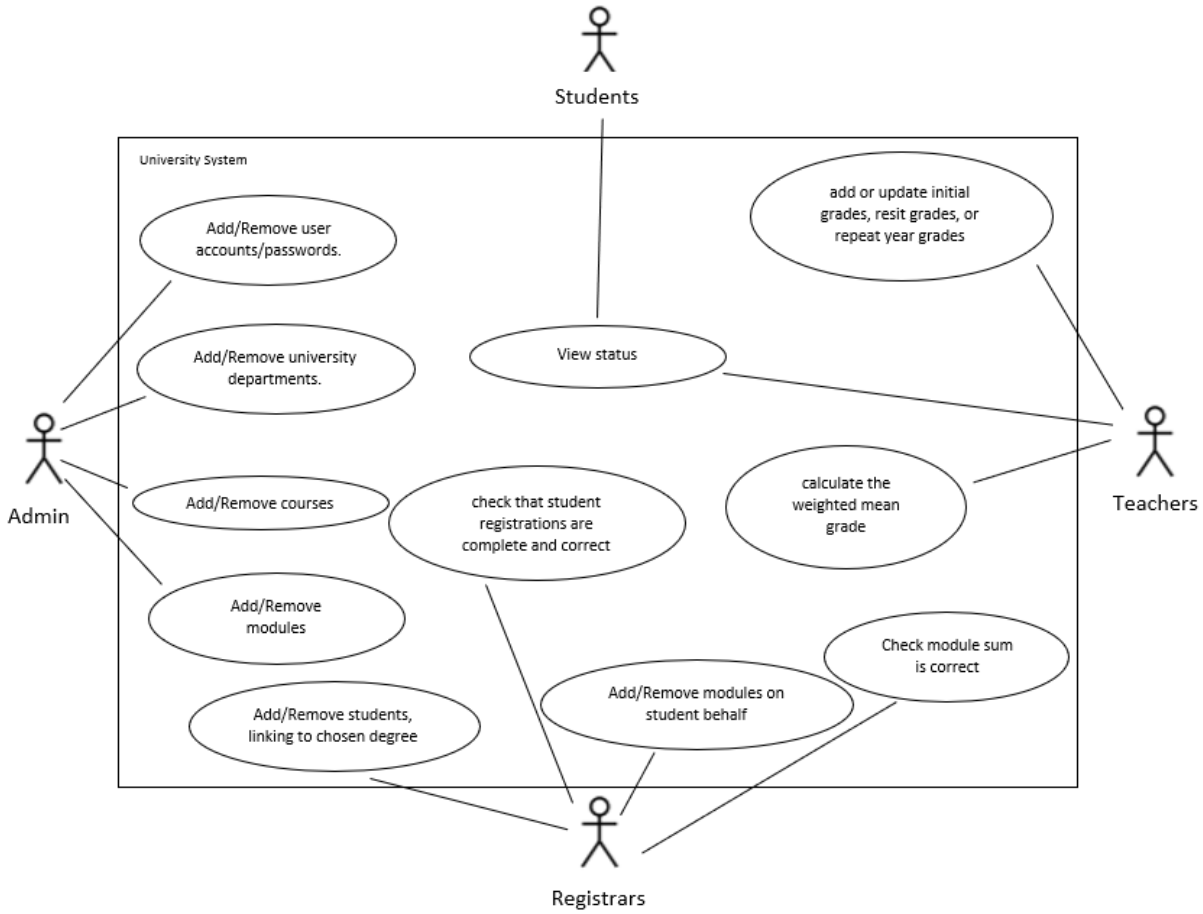
Assessment Code:

COM2008-001

Introduction

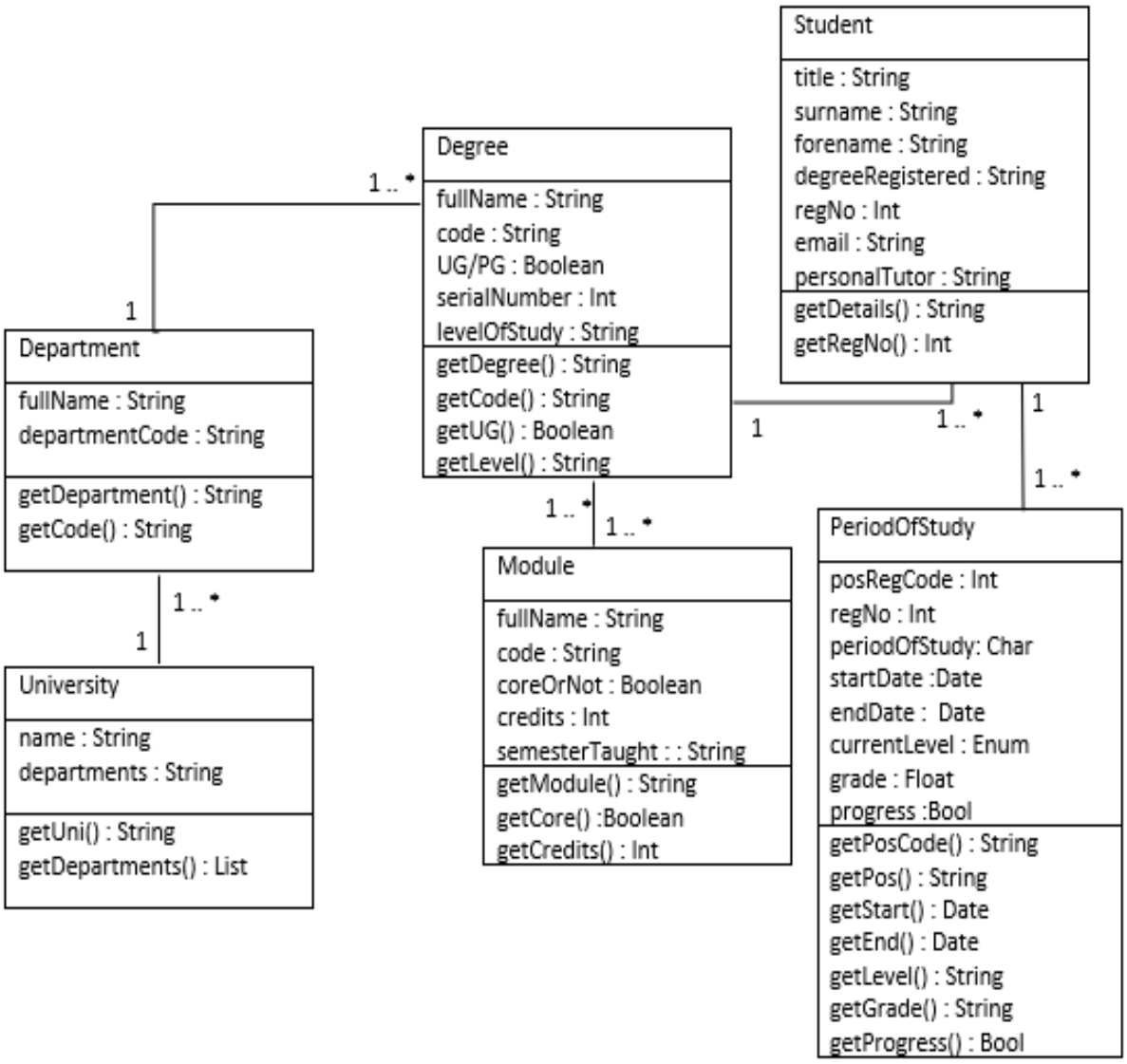
The task, as understood by the whole team, is to build a system to record the progress of students through a University. It should have the functionality to add and remove modules for multiple courses in multiple university departments with which multiple kinds of people can interact. Administrators manage the account creation and the assignment of roles to each account, as well as being able to add and remove modules and courses from the system. Registrars can link the courses and modules to the students and Teachers can give marks and enter the progress of students in each module and course. Students can view their own progress but nothing else. The objective is to build a single system which all four types of user can interact and which dynamically changes depending on the level of user using it. The system will have to be secure and resistant to a number of different methods of attack.

UML Use Case Diagram

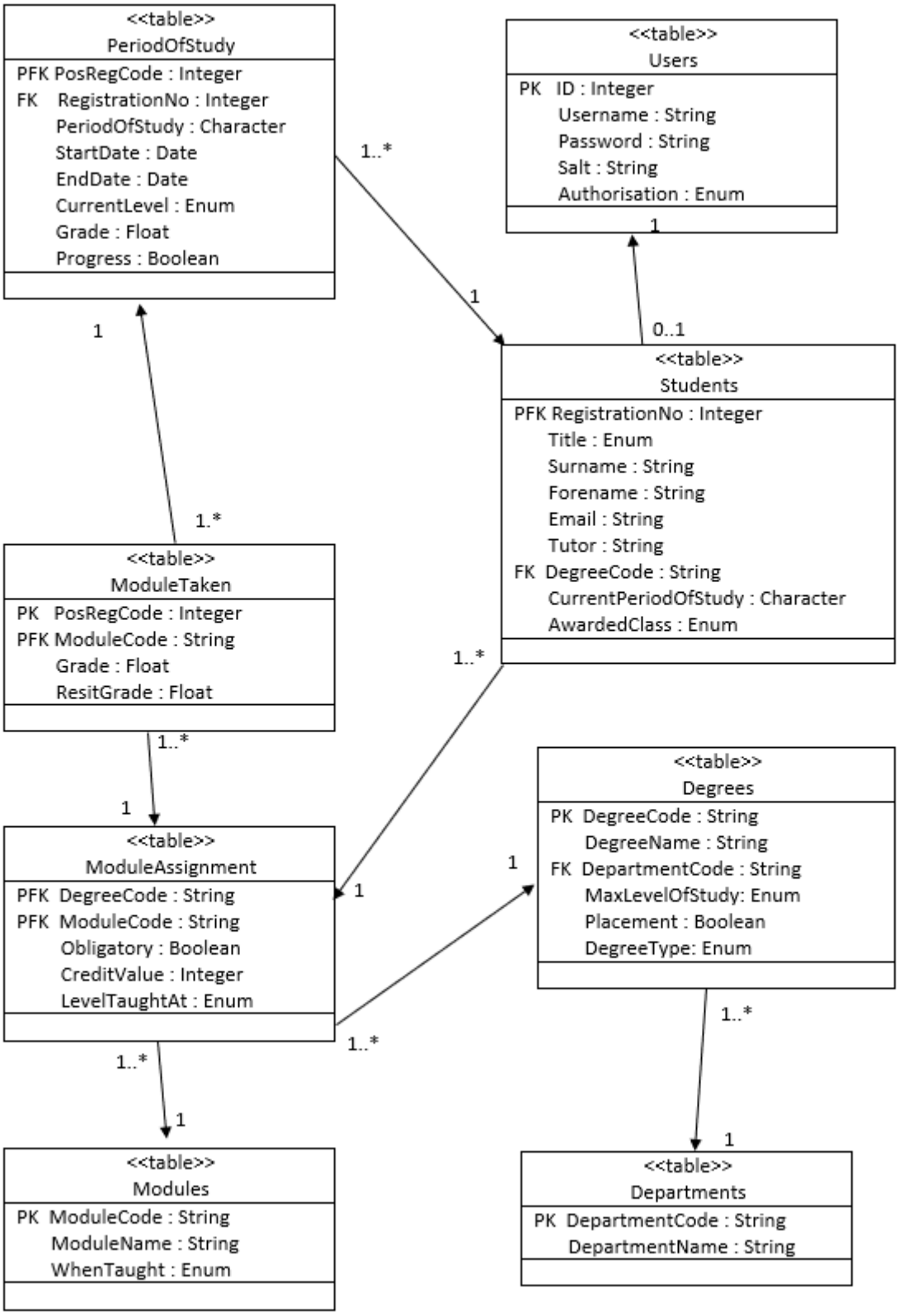


This diagram shows the main actors and use cases that need to be implemented in the system. It shows what actions will need to be performed by which users and which actions overlap between classes of user.

UML Class Diagrams



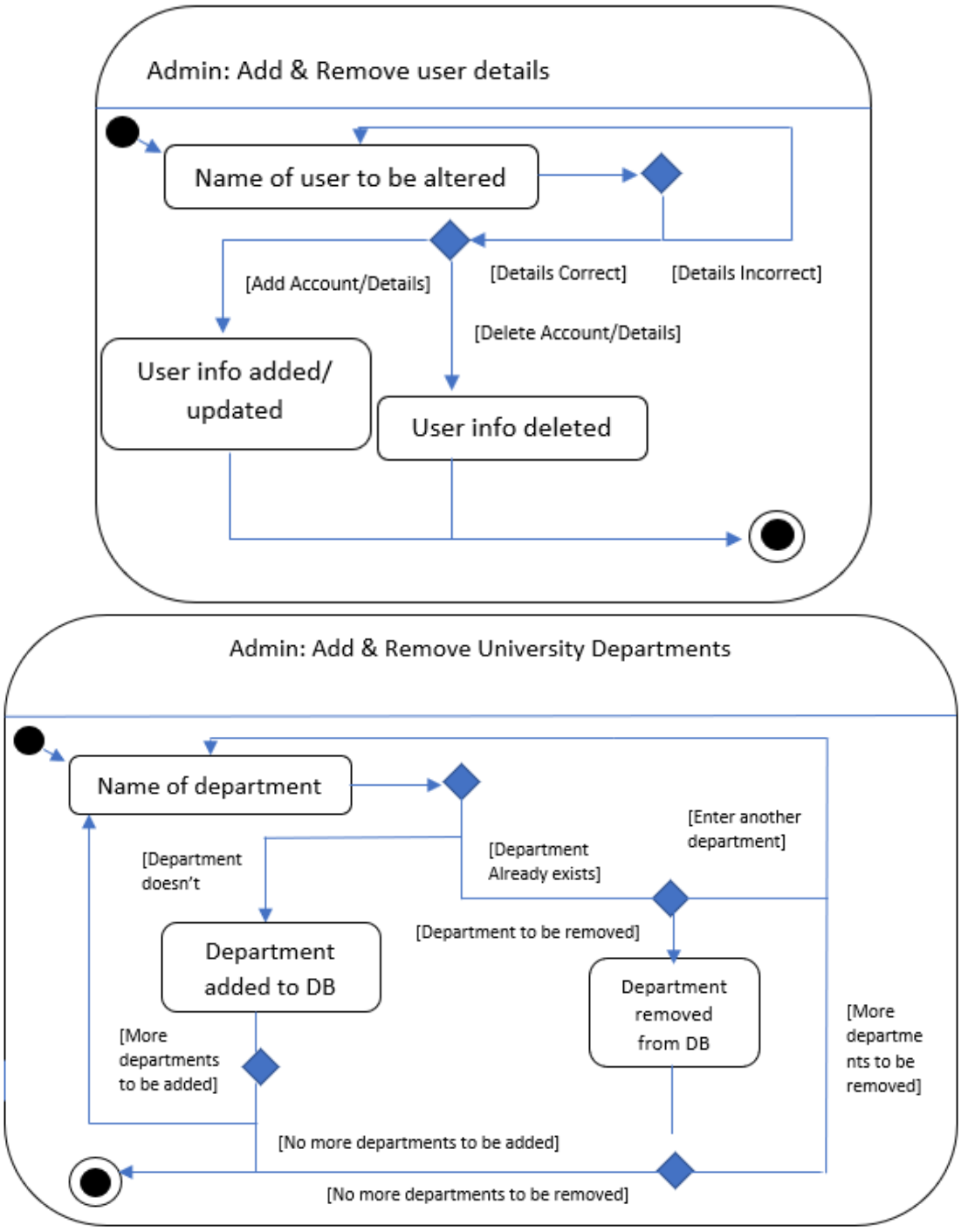
This is the initial information model that we devised and developed from looking at the contextual background information and determining what information would need to be stored and kept record of.

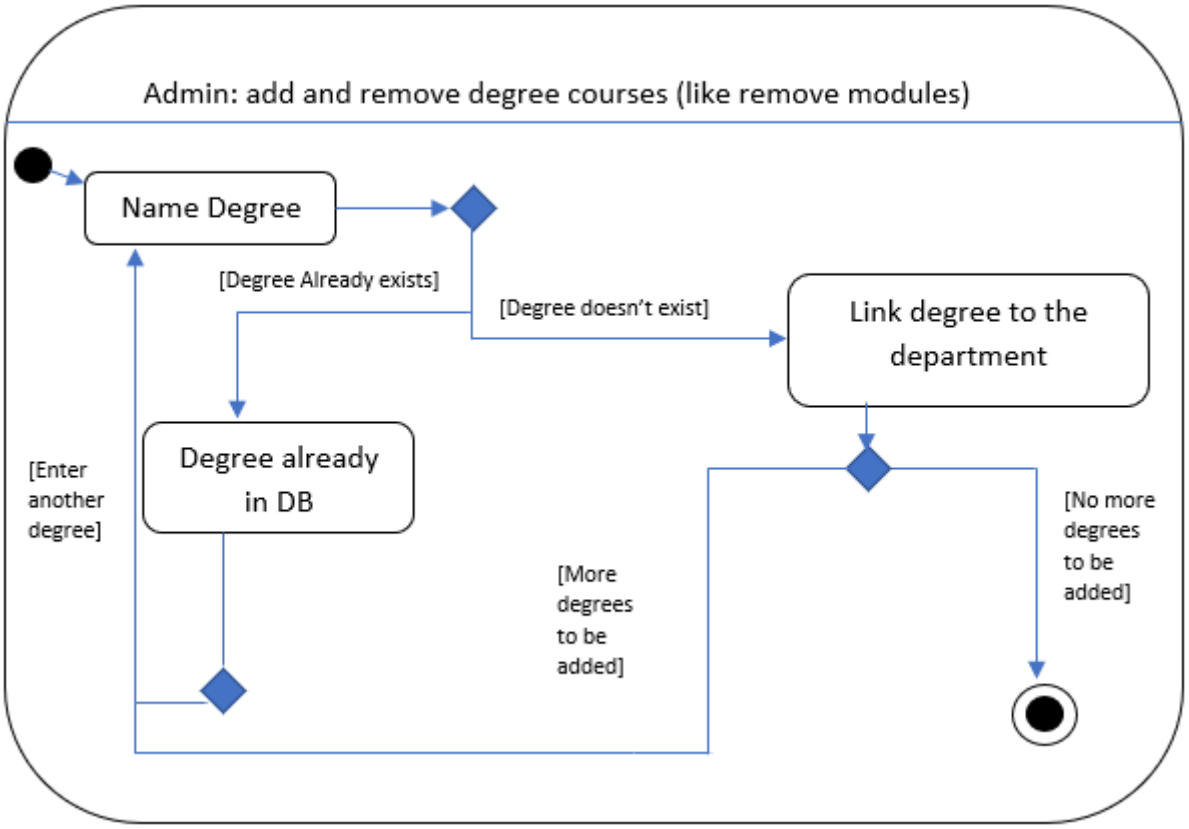


This is the UML class diagram of the normalised database model that we decided would be the most effective to implement. It breaks down all the necessary information so that there is no data redundancy and the highest possible data integrity.

UML State Machine Diagrams

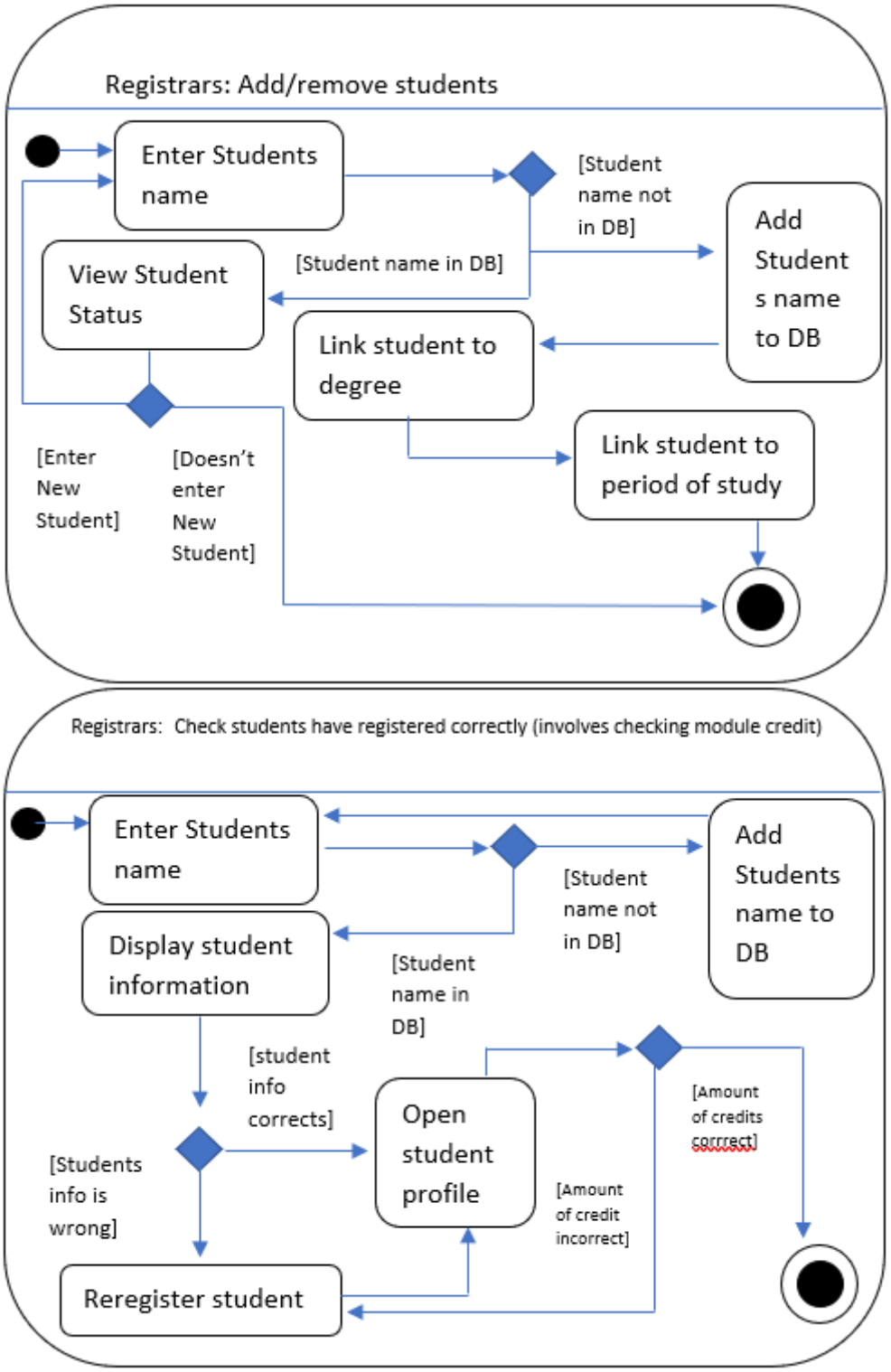
Administrators





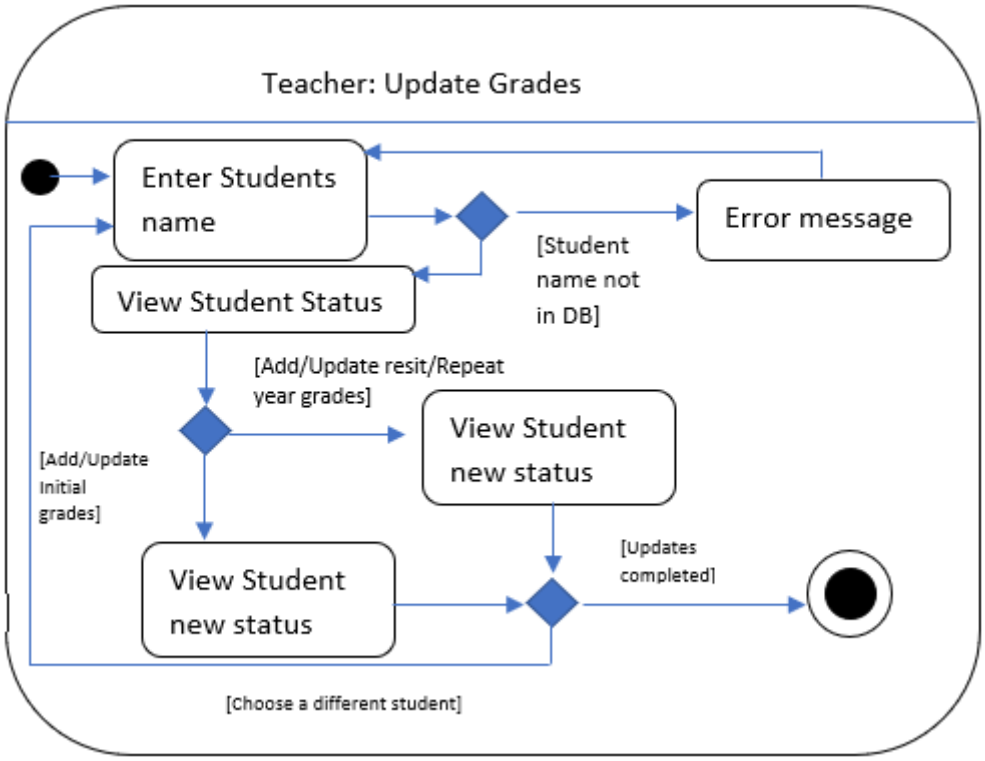
These diagrams show the different actions that the Admin class of users can perform. We started with these as they would have the highest amount of control over the system so it made sense to base the other groups around them. The first diagram shows the process that would have to be executed if the admin wanted to modify the records of existing users, the second shows the process required to add or remove departments and the third shows the process for adding and removing courses. The process for adding and removing modules would work in the same way as adding and removing degree courses as seen in the third diagram.

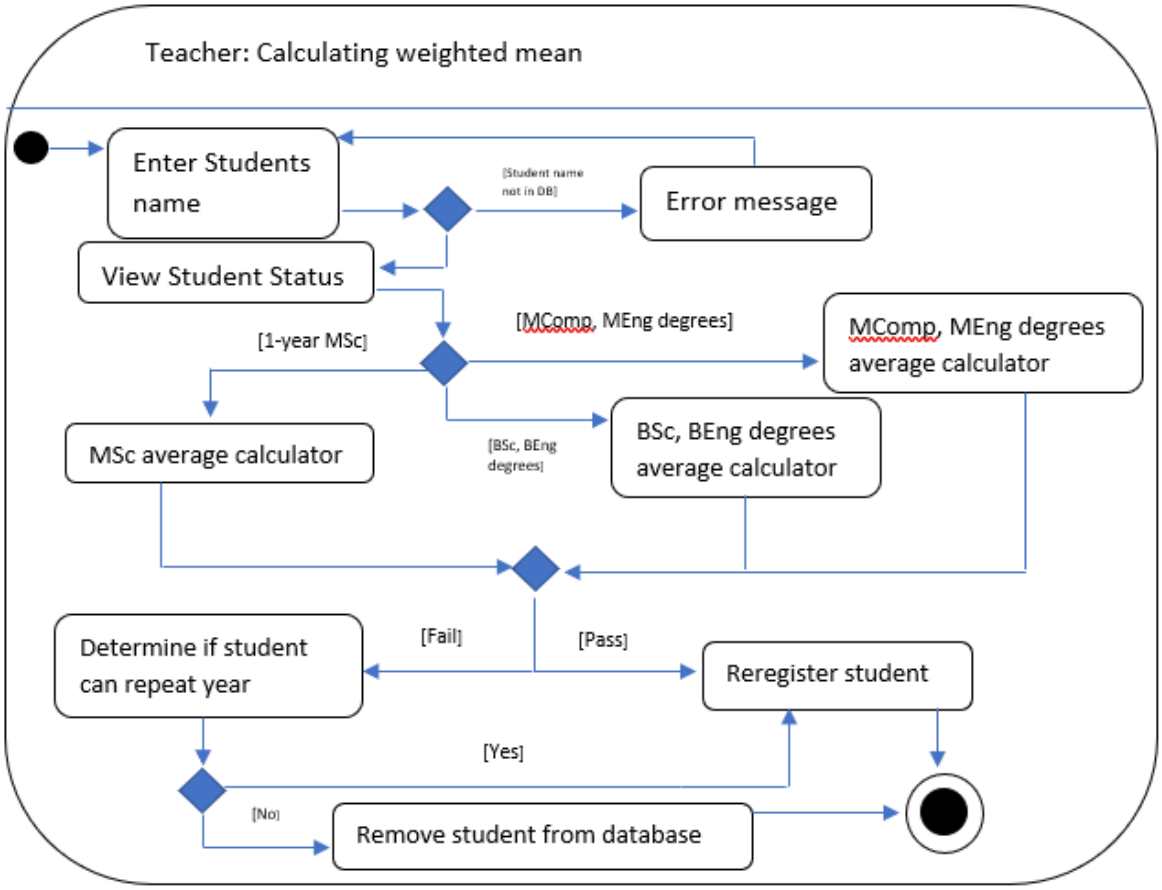
Registrars



The above diagrams show the actions that can be made by the group known as Registrars. They have the ability to add and remove students and assign them a degree as can be seen in the first diagram, as well as the ability to check that students are registered correctly as seen in the second diagram.

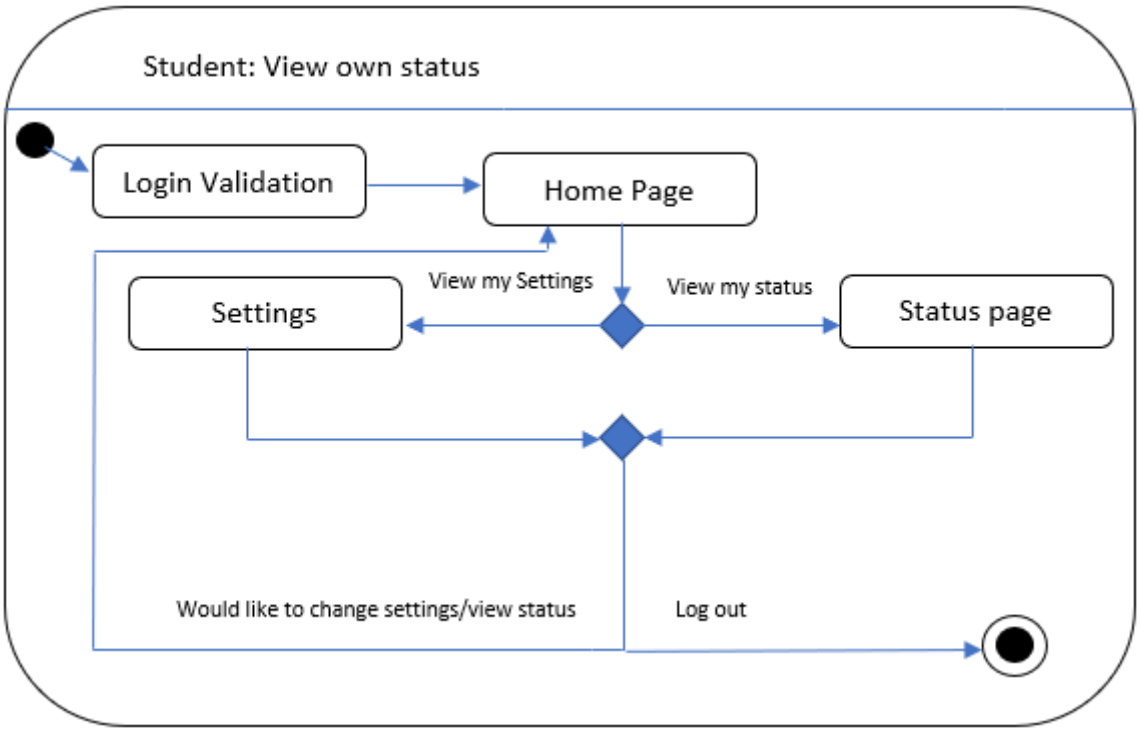
Teachers


Teacher: Update Grades
Enter Students name
Error message
[Student name not in DB]
View Student Status
[Add/Update resit/Repeat year grades]
View Student new status
[Add/Update Initial grades]
View Student new status
[Updates completed]
[Choose a different student]


Teacher: Calculating weighted mean
Enter Students name
[Student name not in DB]
Error message
View Student Status
[1-year MSc]
[MComp, MEng degrees]
MComp, MEng degrees average calculator
MSc average calculator
[BSc, BEng degrees]
BSc, BEng degrees average calculator
Determine if student can repeat year
[Fail]
[Pass]
Reregister student
[Yes]
[No]
Remove student from database

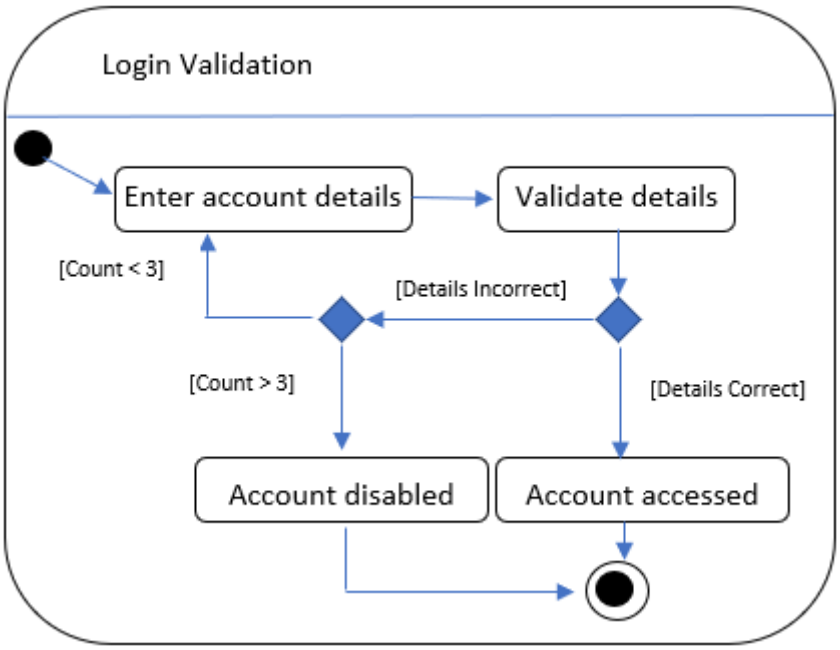
The diagrams above show the different actions that a user who has been designated a teacher can perform. They can update the grades of any student, as shown in the first diagram, as well as being able to calculate a weighted mean of a student's marks across a given course (as shown in the second diagram).

Students



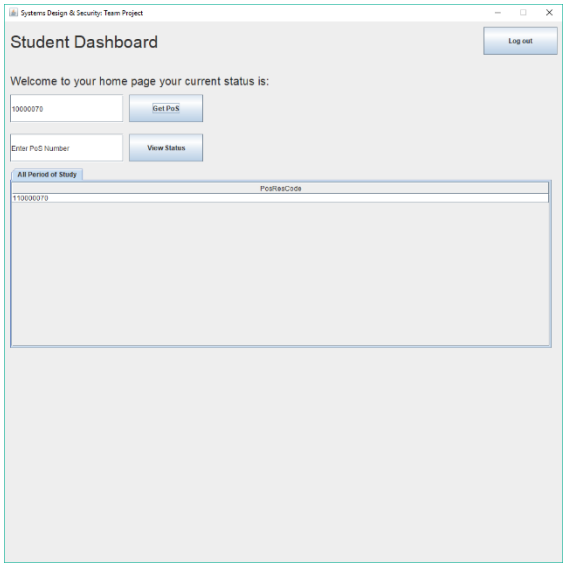
The final group of users, the students, only have the ability to view their own progress, the process of which is shown in the above diagram.

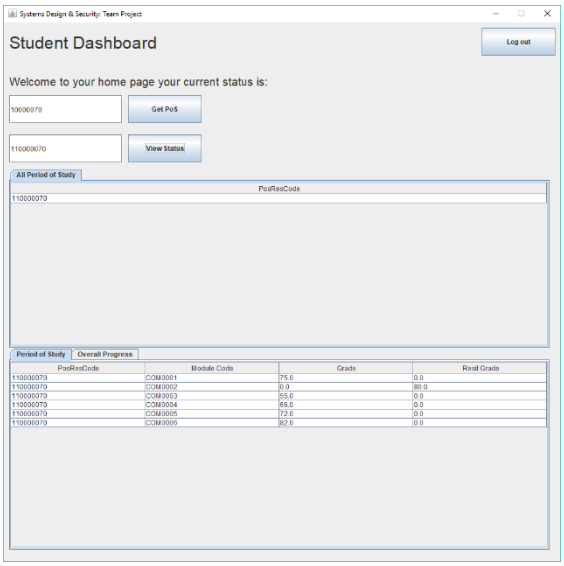
Other



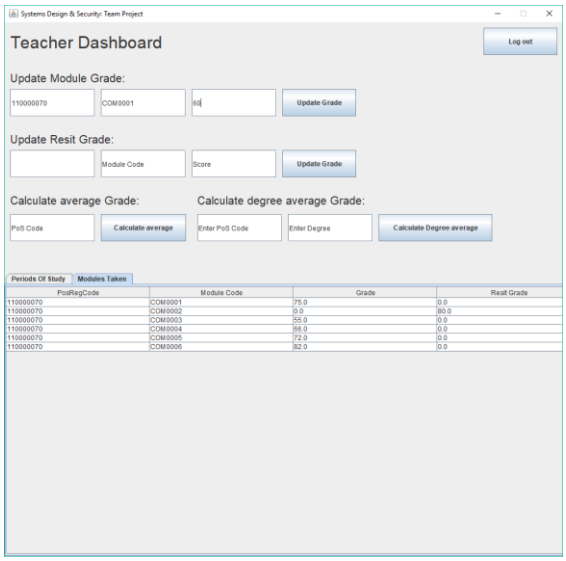
The login validation diagram applies to all groups of users and the process described in it would be applied before any level of user would be able to access the system, otherwise it would not be secure.

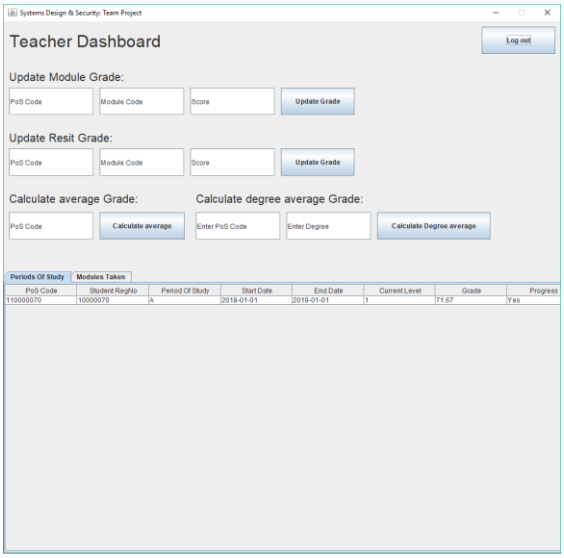
Screenshots



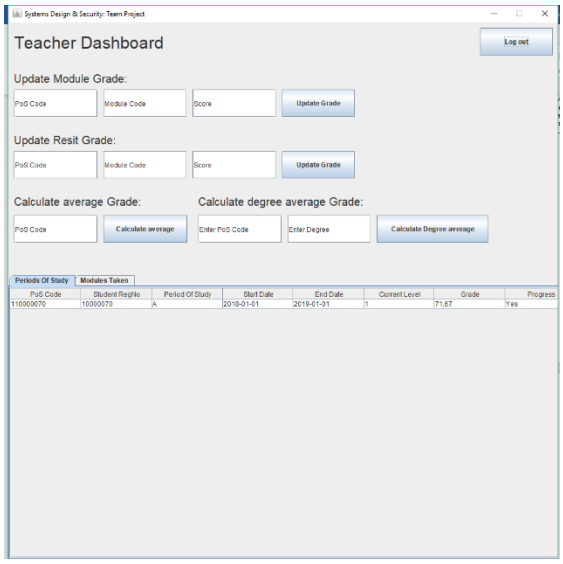


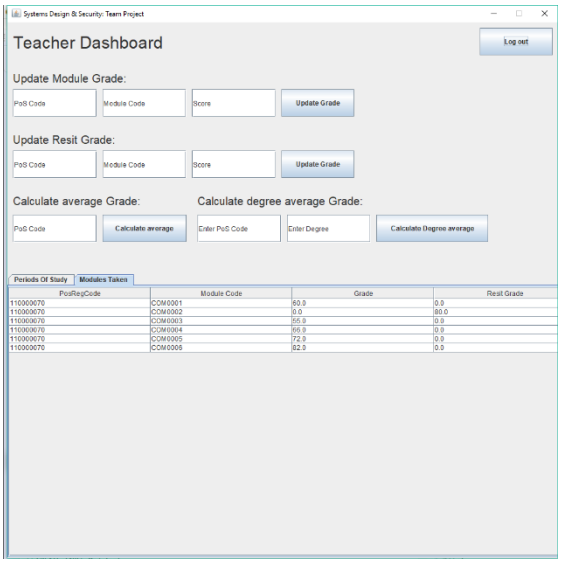
These screenshots show how a student would get their grades, for all modules they have sat at a whatever period of study they'd like to see, the overall progress tab shows the average of the student's grade and whether they will be progressing.

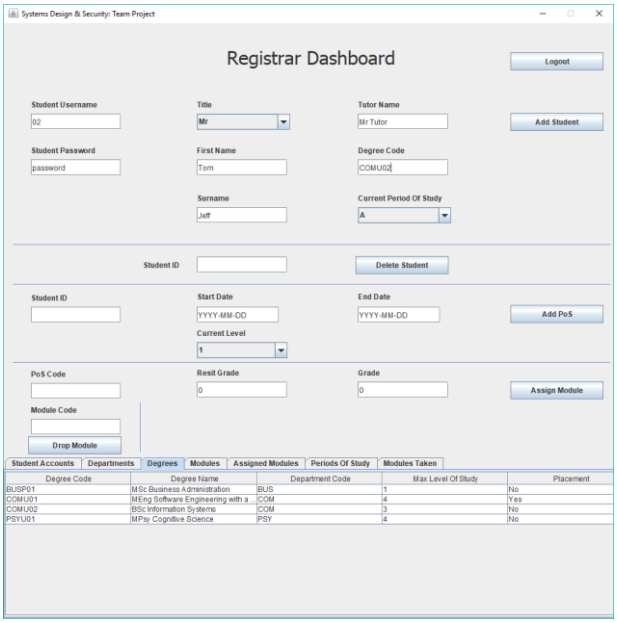


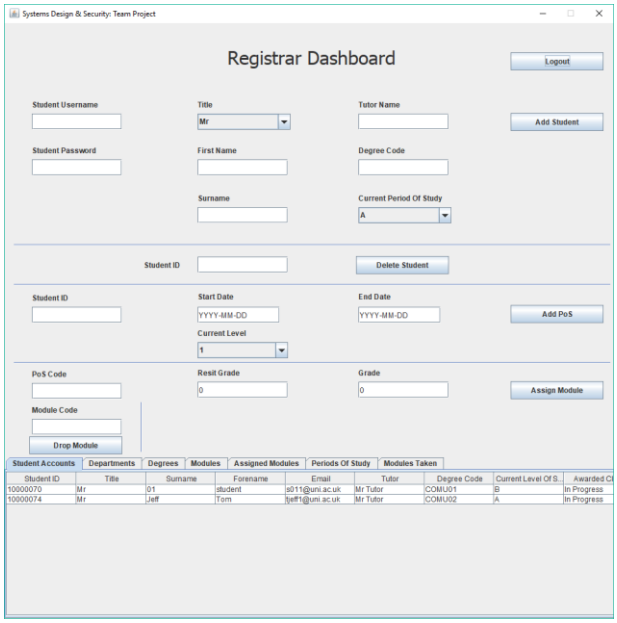


The screenshot to the left shows I am going to update the grade for a student with a period of study 110000070, which is specific to each student, for module COM001, a feature we have implement if that the overall grade is automatically updated once the grade is changed. Note the overall grade listed on the screenshot to right, the results are screenshotted below, so the module grade was correctly updated and overall degree score was automatically calculated and updated.

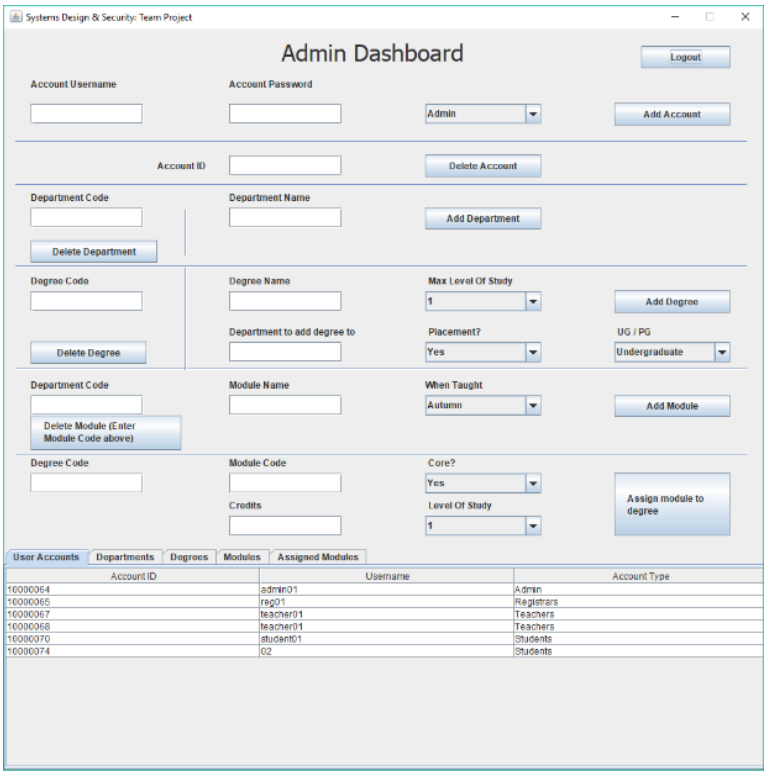








Above you can see the page for Registrars. As you can see you can add and delete students as shown in these screenshots. There is also the functionality to add Periods of Study and also to assign modules. Any information that can be accessed or changed by this user type can be viewed at the bottom of the screen.



Above is the Admin page, for administrators. This page has the most functionality, with users of this type being able to add and delete users of all types (as seen at the top of the screen), as well as being able to add and remove departments, degrees and modules respectively. At the bottom of the screen you can view all the relevant information for the processes above as with the registrars.

The Implementation

To speed up the implementation process and ensure that we worked at the highest possible efficiency we decided to divide the team up into two sub-teams, one of which would work on the database and providing functions to update, modify and retrieve information from it for the other sub-team, who's main focus would be on designing and implementing the GUI for the users to interact with.

Security considerations

Security was a high priority when designing the system, we knew that we had to make the passwords inaccessible to any outside force yet still be able to store and check if a correct password is entered. The solution that we devised to solve this problem was to design a function that encrypts a password before storing it using a salt and works as follows:

When a password is initially created the inputted string is put through a function which returns two strings, the encrypted password and the salt. The salt used to encrypt the passwords further before storing are randomly generated and stored as a string in the same table as the passwords and are are unique to each user. When you attempt to log in and enter a password, the password entered is put through the same function and if the resulting string matches the encrypted password then the user is allowed to log in and enter the system.

It was also important for us to consider the possibility of users being able to access functions that they should not be able to i.e. privilege escalation. For example, if a student would be able to somehow gain access to the functions that could update and modify grades and marks then the whole system would lose integrity and security, becoming useless. To counter the possibility of this happening we assigned different users different userTypes as well as ensuring that sensitive info that userType shouldn't be allowed to view such as passwords and info about about students (for Student types) is never passed

Another concern was preventing the use of SQL injection to trigger malicious updates to the database, which again would mean that the system would lose integrity and be open to manipulation from outside sources. To prevent this we implemented prepared functions instead of regular ones. This has a number of benefits that include making the whole system more efficient but the main one is that in prepared statements parameter values are passed separately which makes SQL injection impossible.

Team Member	**Contributions**	**Points Awarded**
Tom Eso	Team leader for the project, coordinated most meetings and delegation of the tasks. Also produced the UML case diagram and all the UML state machine diagrams. Alongside this coded the design and functions for the teacher and student pages for the final project.	27
Josh Barrows	Designed the Initial information model diagram, designed the GUIs for Login, registarpage and adminpage. Implemented all functions relating to logging in, registrar activities and admin jobs.	27
Adam Jones	Designed (UML for the normalised database), created and set up the database to the server. Coded most of the content of 'Sql.java' working on the security features such as password encryption as well as functions that allowed the GUI to insert, remove and alter records in the database.	27
Matthew Butt	Assisted with the development of the database code in 'Sql.java', editing UML diagrams as well as compiling and writing the team report.	19

Signatures of all team members:

Tom Eso: _____

Josh Barrows: _____

Adam Jones: _____

Matthew Butt: _____