### **Introduction**

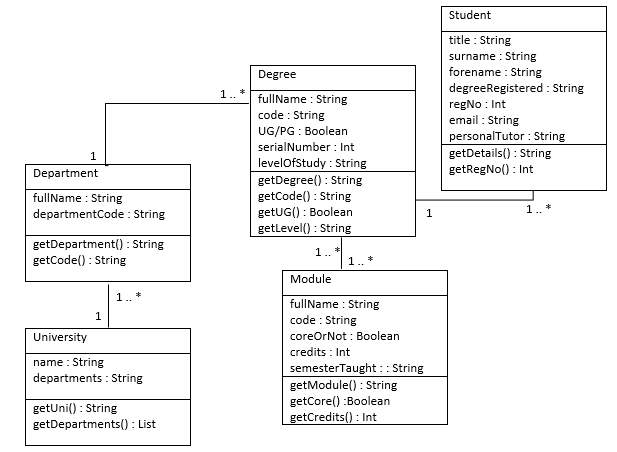
The task, as understood by the whole team, is to build a system to record the progress of students through a University. It should have the functionality to add and remove modules for multiple courses in multiple university departments with which multiple kinds of people can interact. Administrators 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



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.

#### 

### The Implementation Process

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.

### Screenshots

Put some screenshots & description here when it actually works.

|  |  |  |
| --- | --- | --- |
| **Team Member** | **Contributions** | **Points Awarded** |
| Tom Eso | Team leader |  |
| Josh Barrows |  |  |
| 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. |  |
| Matthew Butt | Assisted with the development of the database code, editing some UML diagrams as well as compiling and writing the team report. |  |