**Coursework 1: Football Club - Design**

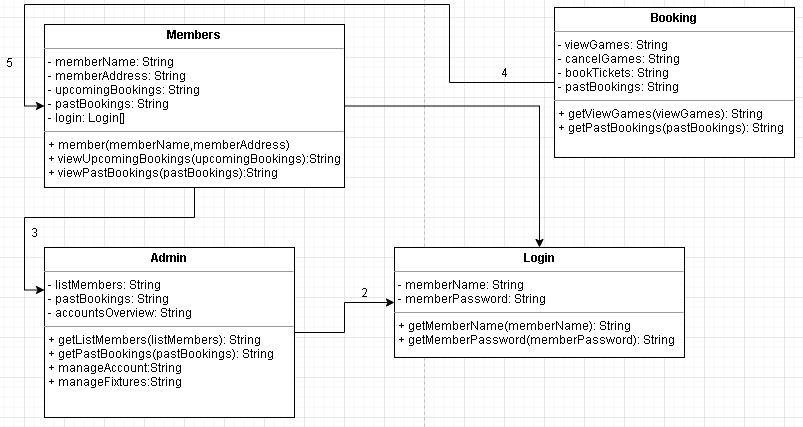
**Analysis**

After reading the brief I am required to design a new local football member record system which will include specific functional requirements such as the personal details of the members of the club like name and address, past and upcoming bookings that have been made and the ability to add or remove bookings from the system. The system should also ensure that the members keep all their data secure by ensuring private data such as passwords and other credentials are kept safe. In order to keep the system as secure as possible, the privacy and security will be the main factors of the design.

In the first stage of the system, the user is required to input some personal data about them in order to create their account. The data included is their full name, address and the system will include the upcoming games they have booked and a list of their past bookings. The log in system will require the user to store their name and a password that they will have to create in order to gain access to the system, and the password will be encrypted as a hash function as this is more secure than a Caesar cipher. Encrypting the password instead of storing it in plain text is crucial as the user needs to feel full protection from hackers so the password needs to be stored in a manner which no one can view it but the user. Although the password is stored as a hash on the system, the user is the only one with the knowledge of the password as they input it in plaintext using their private key, which is given back to them from their public key. As soon as the user has gained access, they are able to view their upcoming games and past bookings. They also can book new tickets for games and change their personal details, so it is vital that drastic security measures are indispensable. If the user decides to change their password, they will enter a new password which will also be encrypted using a hash function and all other personal data will be stored as a hash compared to plaintext which is exposed to security threats.

As the administrators of the system are required to manage all accounts on the system, they also have priority to list the members and their bookings, an overview of the member accounts in the system, add or remove accounts, add or modify upcoming fixtures and list attendance of both past and present games. The user’s details should be encrypted for security but there is no need to encrypt data such as the specific games users have booked or their past bookings. The admins should have access to the members list in case they need to update their fixtures. For example, if a game is cancelled the admin can send a message or post a public message on the system explaining that a game has been modified so the user can have full awareness of this in case, they need to cancel the game or move it. As the admin has permission to add and remove accounts, they only need the users name on the account to be able to move the data, anything else belongs to the user such as their password. The admin should be able to see everyone’s name in the system but that is all as it would contradict their privacy policy. The attendance should also be present to the admin as they may need to make amendments such as if the capacity is too full for a future game, if this happens then they may have to add more space or put priority on who can view the games first. It is also important for the admin to see the capacity of the previous games as they will need to gain statistics in case they are required to do so, and it gives the football system a sense of achievement if they have maximum capacity games all the time.

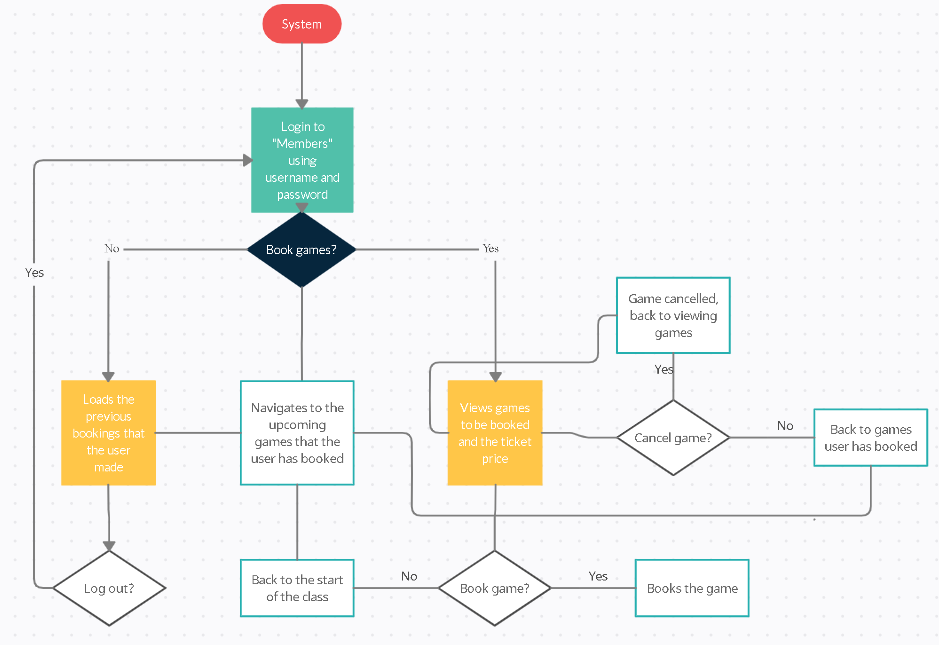
Overall, the credentials such as passwords and other private information should be stored as a hash function as the user should keep their private data safe and all other accounts should be encrypted in case of a threat or attack.

**Solution Design (UML Class Diagram)**

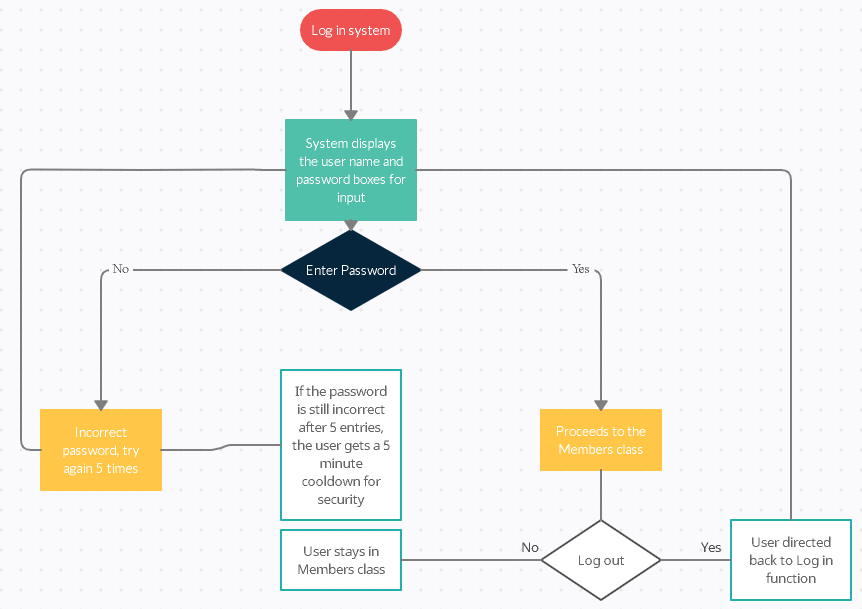
The UML (Unified Markup Language) diagram was designed in order to display how the different classes link together in the implementation and how the different methods in each of the classes can work together to run the program, making a standard way to visualise the design of the system. For example, the method “pastBooking” is present in the Members, Booking and Admin class and that is because it is needed to complete functions in different parts of the program. I decided to make all the methods strings as the system needs to be as accurate as possible in order to keep the overall functionality of the system consistent. In order to tackle the solution, I will need to implement various methods including classes. Classes allow the program to manage objects in the code and support inheritance which is useful as it allows for the reuse of code to be more consistent as writing new code out for a specific function may be a tedious and strenuous process. Classes also give a clear structure for the code as you can identify which functions are being used at a specific time and how they work together to bring great functionality to the system. Based off the requirements that I will use a few classes in order to keep my systems functionality smooth, including a member’s class, booking class, login and admin class. This will make implementing the code easier when I am required to do so. The members class will be used to add the user to the system which will store their personal details such as their name and address, which they can also edit or delete off the system. It will also display the upcoming games they have booked and their past bookings. The login class will be used to log in to the system to access the member’s class. The booking class will be used to view and cancel the upcoming games that they have booked and to book tickets for upcoming games. The admin class will be implemented in order to list all the members on the system to see who is on it, add/remove the members accounts in case they are required to do so, state the overall attendance of the past and upcoming games and to add/modify the upcoming fixtures in the system. Different methods will be inherited from other classes for the functionality of the system to remain consistent and to fetch any functions needed from different classes. In the first class “Members”, the user inputs their name, address after logging into the system. They are also able to view their upcoming games and past games in the class which is useful to them. In the “Booking” class, users can view upcoming games to see if they want to book them, cancel games in case they need to and they can also see their past bookings. This is the same as the “Members” class as they may need to see what games they have been to and if ticket prices were low or the venue was good, the user can then decide whether they are going to go to the same venue again.

**Flow Diagrams**

Flow diagrams are diagrams which represent a flow or set of dynamic relationships in a system. These diagrams take on a step-by-step approach to solving a task and use specific functions along with it to link together.

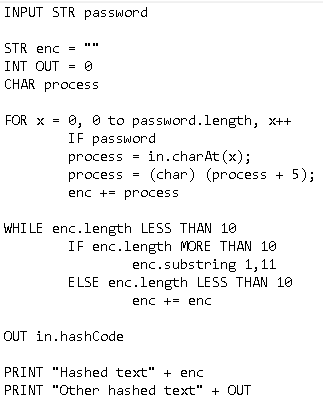


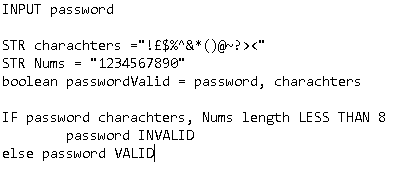
In this flow diagram, the process of logging in, booking, viewing and cancelling games is present. The user logs into the system with the username and password. They then have the choice of booking games and if they select yes then they are navigated to the function which displays all new games to be booked and if the user decides to book the game then they go to the book game function and do It there. If they decided not to book a game, then they are moved back to the start of the class and can also navigate previous games that they have booked in case they change their mind and decide to book again. If at the start of the class the user decides that they do not want to book a new game then they can load the previous games that they have made and can look back on them and decide whether they want to experience the game again, and if not, they have the option to log out which will take them back to the log in page. The user also has the option to cancel a game and if they decide to do so, they will have extra confirmation whether they want to cancel the game or not and if they do, they will be navigated back to the view games page where they can see if they want to choose another game to book and if they do not want to cancel then they will be navigated back to the games that they have booked.



In this diagram, the user is required to input their username and password to gain entry to the “Members” class. If the password if the password is incorrect, then the user will have to try again but if the password is wrong consistently five times in a row, then the user will have a five-minute cool down and can try again after that where they have another five attempts then another cool down if the password is still incorrect. Once access has been granted to the user, they can decide whether they want to log out as an option once they are in the class and if they say no, they will be redirected back to the start of the “Members” class but if they do decide to log out then they will be directed back to the log in screen where they can again input their username and password.

**Pseudocode**

Pseudocode is needed to ensure that the key aspects of an algorithm are present before you write the proper code into a language. The code cannot be compiled but gives a good indication of what the real function will look like. This pseudocode was made in order to show how the hashing function will work in the system, as passwords needed to be encrypted or hashed for the user to feel full security. First, the user inputs their password into the function which is then compressed into a numerical value. A string is then made for the encapsulation of the hash. The keys are then stored, and the hash function is called. Keys are also stored in the hash codes index and the hashed password is then outputted along with another hash. This function also shows how the function will be stored as a hash and the password is validated.

This code shows how password validation will be implemented. The password is inputted by the user and checked to see if it meets the requirements. There are multiple characters and numbers that the code must follow and if it contains them then the password is valid, otherwise invalid.

**Security Analysis**

In order to keep the system as secure as possible there are multiple security features that can be implemented which make it almost impossible to breach. Using two step authentication such as two passwords or a code is very useful as it ensures that the user must break two security barriers before they can access the system. For example, if the system contained a regular password and a pin after that it is a lot more secure than just the pin or just the password alone. To keep the system as safe and secure as possible I will use two passwords with no hints so that the user must get through double the security. The passwords will also be hashed which is good as they cannot be reversed and will be almost impossible to guess or obtain. I decided to use a hash function over a Caesar cipher as I believe it is more secure and a more efficient way of implementing security when securing passwords. Whilst encryption is the process of plaintext being scrambled into text, which is unreadable, hashing is the process of converting an input into a fixed size of letters and numbers. I think that hashing is better as the hashed text is a different length compared to the original code. This means that it is almost impossible for someone to guess the hashed code as it is a variable length whilst encryption is the same length as the input. Each hash value must also be unique, meaning you would need to carry on guessing until you cracked the code. Hashing is also a one-way function, meaning that there is no way or reversing the hash when you can with encryption. For example, symmetric encryption involves only one key to both encrypt and decrypt the data which is a disadvantage as the key needs to be kept secure and this is a challenge where both encryption and decryption take place. Caesar ciphers are methods of encryption where there is a shift in letters, however this would not be useful in the system as the user needs to have full security. It is a simple structure used and only provides minimum security when the user needs full security and once you find the shift it is easier to break into the password. However, Caesar ciphers are good as they are easy to code and require only a few computing resources. I will also implement extra password validation so that the user must select multiple things for the password to be secure. For example, the user must ensure that the password is between 8 and 16 characters long, contains numbers and special character. This makes it much more difficult to break as there are many different types of characters to try and figure out apposed to just letters which is still hard to break but far easier. Password validation is also good as it is simple to implement into code, however storing it as a hash may be more difficult to recover if the password is forgotten by the user.

**Testing Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected outcome** | **Potential Issues** | **Issue Solutions** |
| Logging into the system | The user should be able to log into the system without any trouble | The password may not grant access to the Members class | Ensure that all functions link together if required |
| Running code in the program | The code should run without any errors | There may be an error in the code which forces the entire program to stop working, therefore ending the process | Ensure all code is grammatically correct and to comment out code that will be needed for later use. |
| Different import functions | All the functions imported are expected to work fully without and errors | A function may not have been implemented correctly therefore causing errors | Ensure that all Java files that contains library functions are inputted correctly before use. |
| Viewing past/present bookings | The system should display all the bookings that the user has made whether it is past or present | The user may experience difficulty listing all games as some may have been deleted | Include cancelled games in the list. |
| Viewing new games | The user should be able to view all games in the system | The admin may have accidently removed a game or forgotten to put on in | Ensure that the admin has full access to the games and accurately puts them all in the system |
| Cancelling games | The user should be able to cancel the games | The cancelled games may accidently be added to the games which happened in the past | Ensure all games are accurately placed in the correct format. |
| Booking tickets | The user should be able to book tickets for future games | The tickets may be from previous outdated games | Ensure that all past games are separated from the future games |