CSC3002 Car Sharing App

Sprint 1

11th November – 25th November

**WCF Services.**

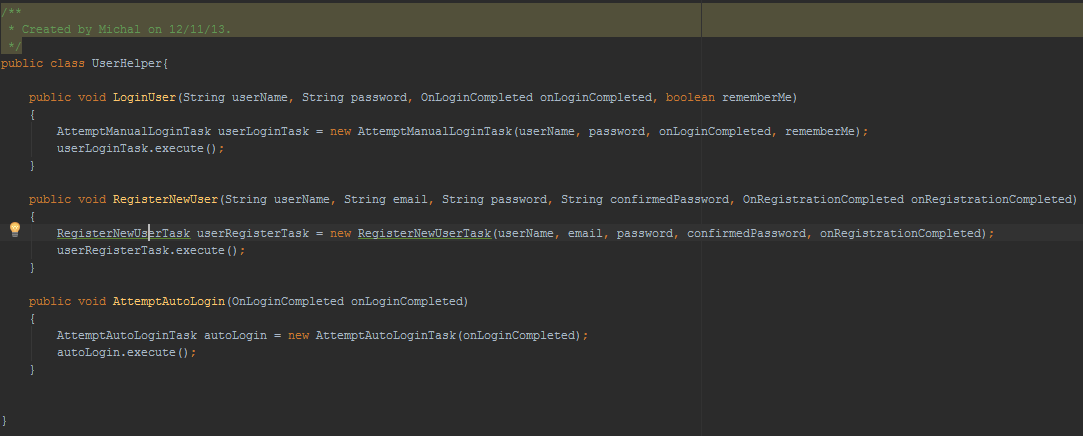
Developed first version of the WCF service. There are currently two service contracts defined for the Android app client to interact with. First is the user service that’s responsible for registering new users as well as logging in existing ones. There are currently two separate login methods, the manual login which is called by the client when no session cookie exists. The second method called automatic login uses client’s cookie file together with dynamically generated unique Android device ID to log the user in without using username and password. The system is described in more detail in the session management section. The WCF service uses both, WebGet and WebInvoke APIs.

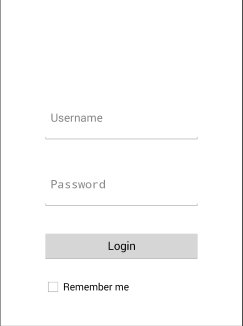
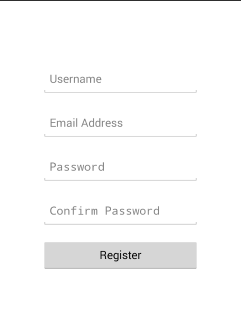


**What went well:** Services are up and running, hosted inside IIS and are accessible to the outside world. The user register and login methods are working successfully and communicating with the SQL database via Entity Framework.

**TODO:** Code is still to be properly refactored and commented. UserService and CarShare service methods to be partially redesigned and possibly relocated to another service to enable further separation of concerns.

**Android Client App.**The first version of the Android app supports registering and logging in of the user. The design is basic at this stage but my main focus was to get as much of the client’s architecture developed as possible. Since Android does not allow any networking activities to be performed on the main thread, every single interaction of the service had to be moved to a separate class which inherits from an async activity class which can be compared to a thread which performs its task in an async mode.



In Android, every screen that the user interacts with is known as an Activity. So far, I have developed a UserHelper class which acts as a glue between activities and the Networking tasks. This means that the UserHelper calls whatever networking task an activity wants to perform and invokes it. How does the async task inform an activity about a task which it just finished? Every time an async task is called, it gets an implementation of a callback interface passed in. This allows the async task to call a method on the activity class once the task is finished without having to pass an entire activity to the async task class.

**What went well:** The user interface although very basic, is intuitive and easy to use. Text boxes follow have labels inside them which saves a lot of space on the already limited screen area.

**TODO:** Revise the layout to make sure it scales properly on screens of various sizes and in both orientations.

**WCF Service Validation.**As well as validating data on the client side, I have implemented logic to perform additional validation on the WCF side as well. This is serves two purposes:

* Ensuring no incorrect data is saved into the database.
* If there is another client developed by someone else attempting to call the service, we do not want to rely on their validation logic.

Validation is performed on data annotation provided by entity framework. This provides a reliable quick and easy way of checking whether the data passed in from the client meets our criteria. As an example, consider the below screenshot:



**Entity Framework and the repository patterns.**

**HTTPS and encryption.**

**Session Management.**