2016

Interim Report

Automated Student Project Allocation System

Anita Lad

Table of Contents

1. Aims and Objectives of the project:

Aim: The main aim of the project is to develop a web application that provides an interface for various users to submit project proposals and preferences and also suggests a good allocation of students to projects considering the available projects and the students' preferences [1].

The project will be an enhancement of the university's current student project allocation system. It will equip the system with cutting edge front-end technologies which will provide users a better look-n-feel and responsiveness than the existing system.

Objectives: In order to achieve the aim,–

1) Setup a server to host the system which will publish the web application.
2) Setup a database to store users' proposals, preferences and other data.
3) Build an application using MVC framework for building a web application.
4) Learn and Implement unit tests with mock objects and integration tests to improve its performance and encounter faults.
5) Implement an SMTP (Simple Mail Transfer Protocol) client service to systematically send notification emails to relevant users.
6) Develop an evolutionary algorithm to search for a good allocation automatically and efficiently. They evolve a set of possible solutions until a "good enough" solution is found.
7) Learn about technologies involved within development.
   For example –
   7.1) Good programming practices such as Dependency Injections, Repository Pattern, Unit of Work Pattern, Message logging, etc.
   7.2) Privacy, Safety and Security such as Password Salting/Hashing, SSL certification, Individual users' authentication and authorization.
   7.3) Configurations such as Network settings, Database Configurations, RESTful API settings, etc.

2. Planning and Timescales:

I have generated the Gantt chart to achieve the system's goals in a timely manner. Gantt chart file can be downloaded here.

In addition, I also will be using Scrum approach for system implementation.

Scrum consists of Backlog plan and Sprints. I have generated a Backlog plan which contains the lists of user stories/functionalities required to be implemented within the project along with their start date and end dates. Each week I setup weekly sprint by determining stories to be implemented for the week (weekly tasks) from the backlog plan.

Why Scrum? : Scum uses "divide and rule" approach which emphases on dividing the whole project in to small chunks (user stories/ functionalities)

and concentrates on completing a week worth of user stories at a time. This helps me easing my stress to not worry about the entire project and follow the plan in timely manner by focusing on sprints. If I can't achieve the dedicated targets for the week, I add incomplete stories to the next sprint, which indicates a transparent progress of the project and forces me to keep up with the plan before it becomes unmanageable.
The Scrum tool can be found here.

Milestones:

Completed:

1. **20-Oct- 16** - Finalise and Submit Project Plan
2. **02-Nov-16** – Project and Database Setup
4. **28-Nov-16** - Prototype Demonstration
5. **05-Dec-16** - Submit First Draft - Interim Report

Related objectives: **1)**, **2)**, **4)** (Unit Test with Mock objects), **7.1)**, **7.2)** and **7.3)** mentioned above within "Aims and Objectives"

Started:

3. 24-Nov-16 – Completion of Admin Functionalities.

Related objectives: 3) and **7.3)** (Network settings) mentioned above within "Aims and Objectives"

Not Started:

6. **08-Dec-16** – Final Submission of Interim Report
7. **16-Dec-16** – Completion of Proposer Functionalities
8. **07-Jan-16** – Completion of Student Functionalities
9. **23-Jan-17** – Setup Integration Test suits
10. **27-Feb-17** – Interview (Anticipated for earliest possible scenario)
11. **02-Mar-17** – Evolutionary Algorithm Literature Search
12. **12-Mar-17** - Evolutionary Algorithm Implementation
13. **23-Mar-17** - Complete Email Functionality
14. **16-Apr-17** - First Draft Dissertation Submission
15. **30-Apr-17** - Dissertation Submission
16. **08-May-17** - Presentation Delivery (Earliest possible date)

Related objectives: 4) (Integration Testing), **5)**, and **6)** mentioned above within "Aims and Objectives"

3. Description of Prototype

I have managed to setup a system with some functionality in place.

▪ Introduction:

The Student Project Allocation (SPA) System is implemented as a web application to allow users to interact via webpages. I have used Microsoft .NET framework to construct the application.

▪ Application Hosting (Server):

I have setup a server on my local machine (localhost) as well as on a virtual machine with the operating systems Windows 10 and Windows server 2012 r2 datacentre respectively. I am using Internet Information Service (IIS) web server to host the application which is provided by the Microsoft. Both local and virtual machines are already equipped with IIS.

The virtual machine is situated on my laptop and provides web hosting for any client machines sharing the same network. However, this doesn't work with university network due to security reasons; one cannot publish their applications to other clients by using the university network. In order to solve this problem, I will be working towards setting up a web server from home.

▪ Database Service:

I am using SQL Server Express as a database engine where I have generated a database called "SPA". The database will store information about various parts of the system, such as users' details, passwords, authentication tokens, proposed projects, project preferences, project allocations, deadlines for project proposals and preferences, etc.

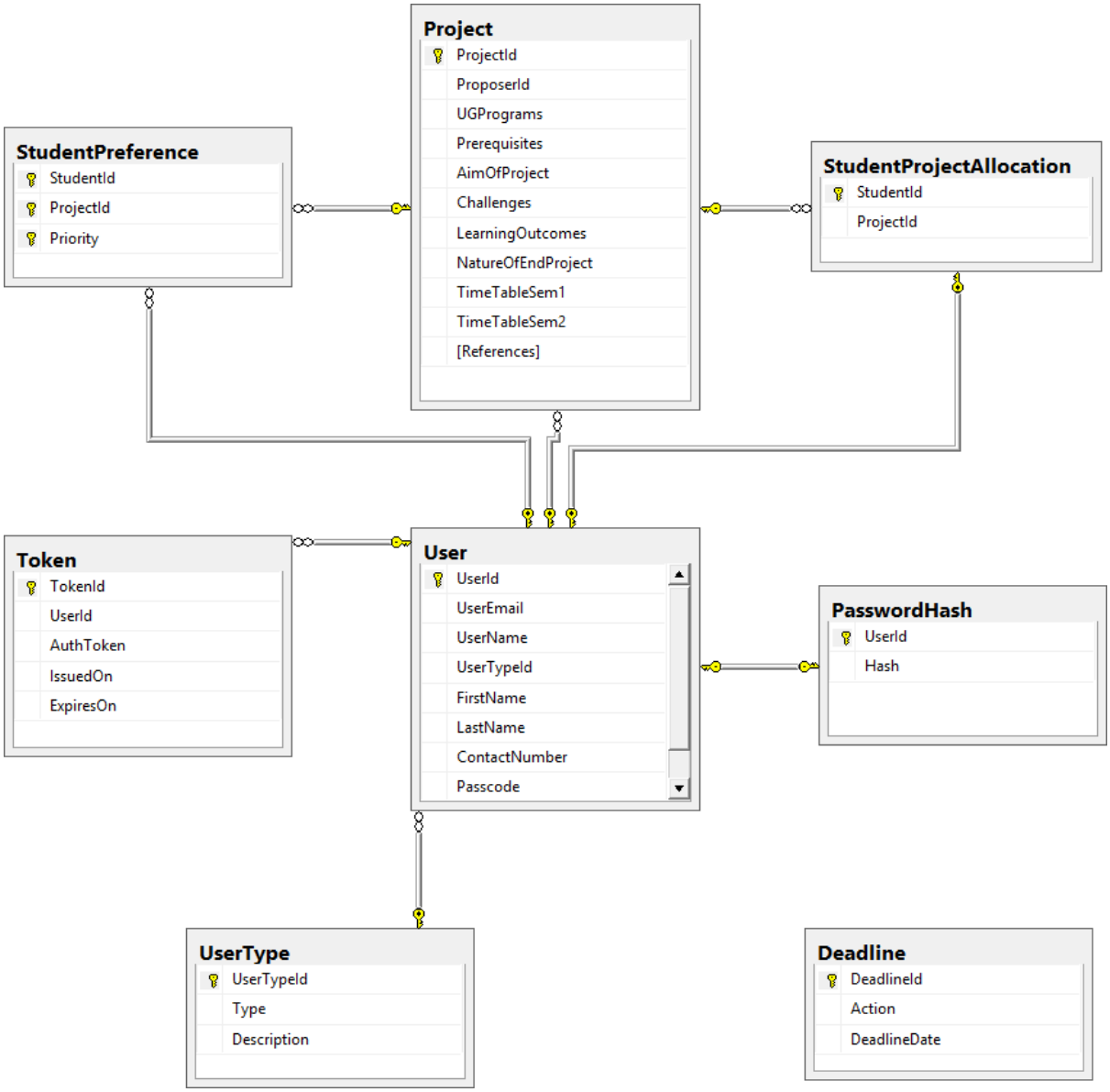

Figure 1 : ERD diagram (Download from here)

▪ Application Development:

The .NET framework (Back-end) uses C# programming language to write server side code. I have implemented AngularJS for Client side (Front-end) rendering and binding.

Implemented System Architecture:

The system is implemented using Visual Studio 2015 IDE and divided into following projects:

1. SPA.Batch
2. SPA.Entities
3. SPA.Data
4. SPA.Core
5. SPA.Web.Services
6. SPA.Web
7. SPA.Tests

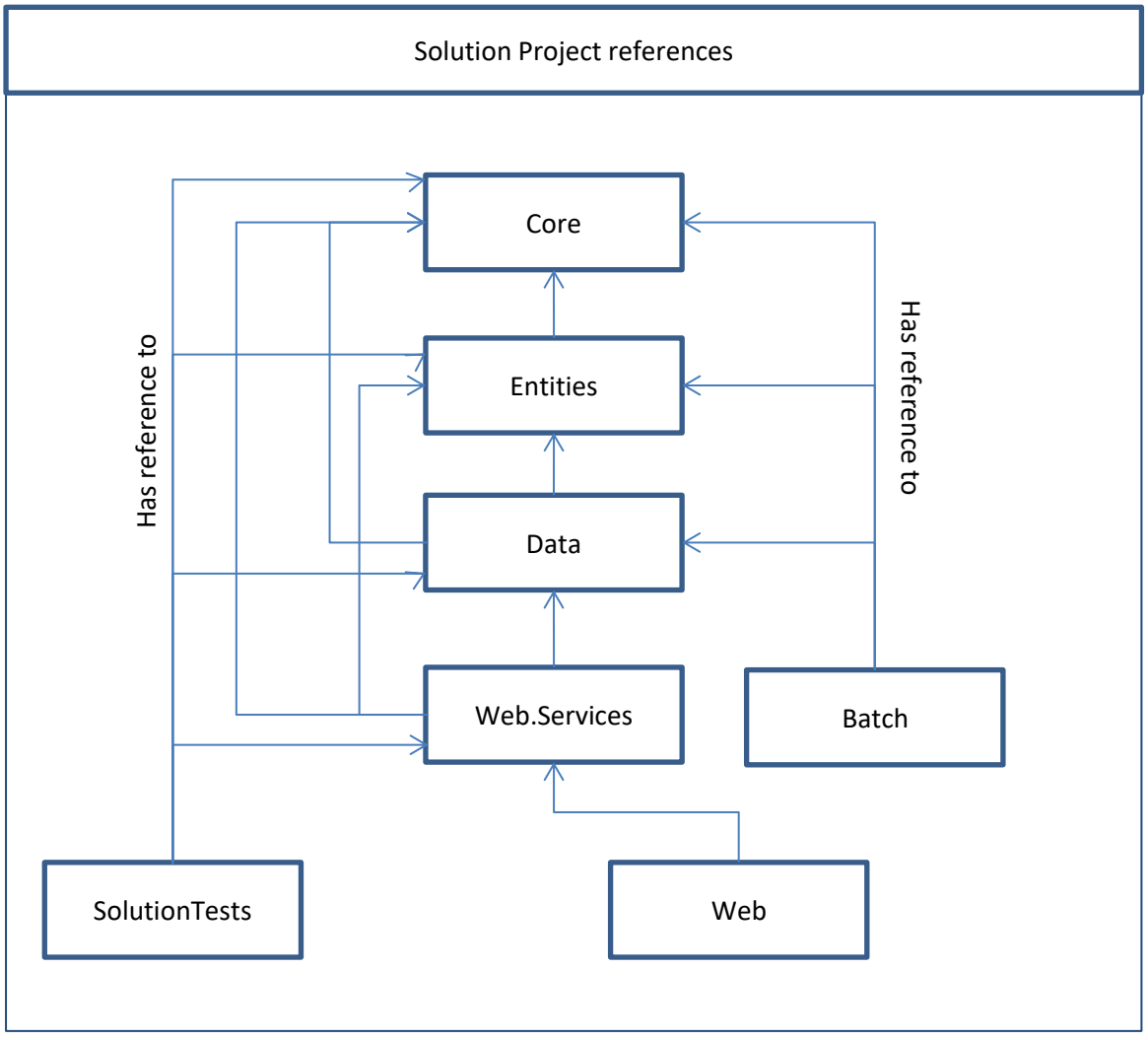

Figure 2: References of Solution's Projects (Download from here)

1. SPA.Batch:

The project contains the 'Main' method and has been used to create a database instance as well as to run quick console based tests.

2. SPA.Entities:

The project contains the entity/model classes which represents the tables within the database. These classes have been used by `SPA.Entities` project's `SPA.MasterDataContext` class to represent the database model; it also contains other helper classes.

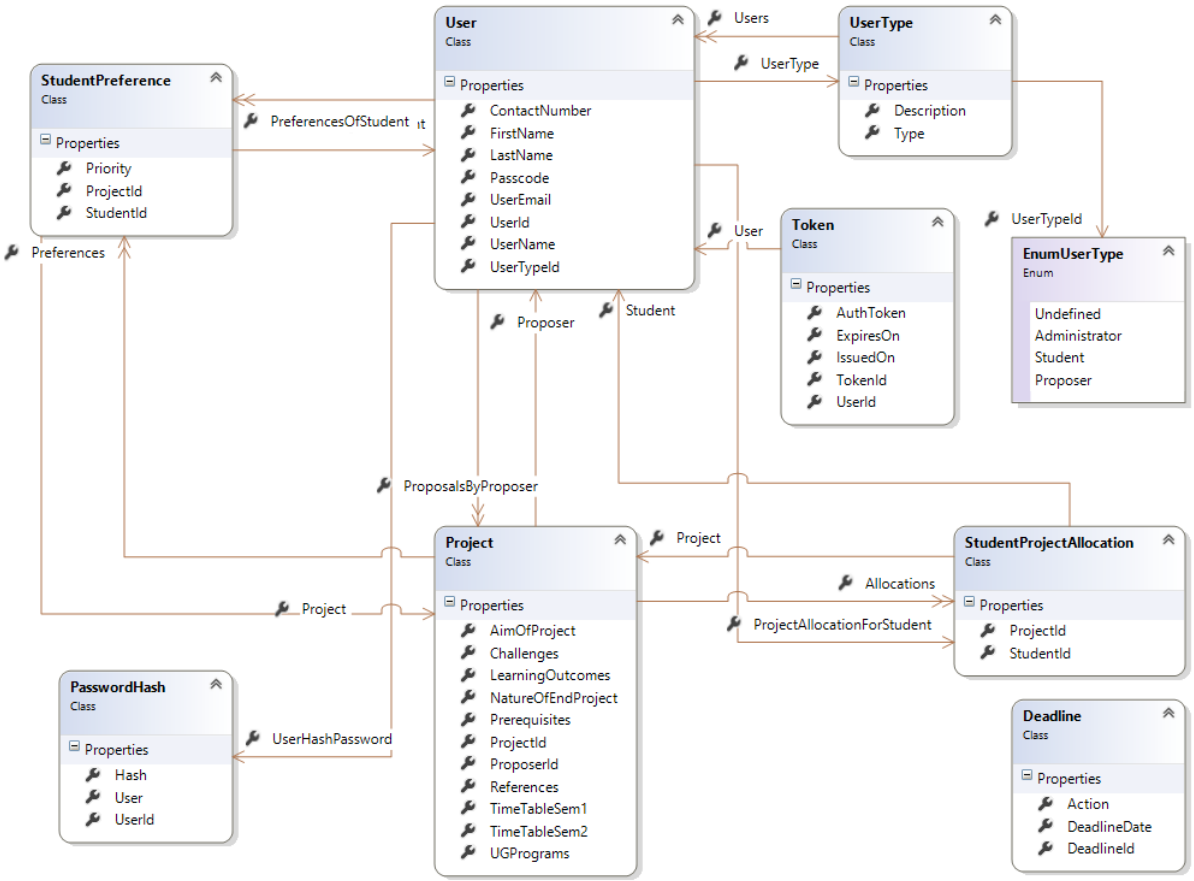

Figure 3: Entities Class Diagram (Download from here)

3. SPA.Data:

The project contains "`MasterDataContext.cs`" file which contains details of the database model and used by Entity Framework while interacting with the database entities. It also contains information about migration history which is used to keep track of model changes on server side and helps to create auto generated queries for updating database.

Download MasterDataContext file here.

4. SPA.Core:

The purpose of this project is to provide essential functionalities and details required by other projects.

The project contains "`AppConstants.cs`" file which holds the static values used within the solution.

The main functionality of this project is "message logging". I have implemented "Logger.cs" to write messages to the file and "LogHelper.cs" to allow other services to interact with the Logger class. Messages are divided into four stages - Error > Warning > Info > Verbose. A class calling methods within the "LogHelper.cs" provides the message stage and the message content. The message is then logged to the dedicated log file with its stage and current user's details.

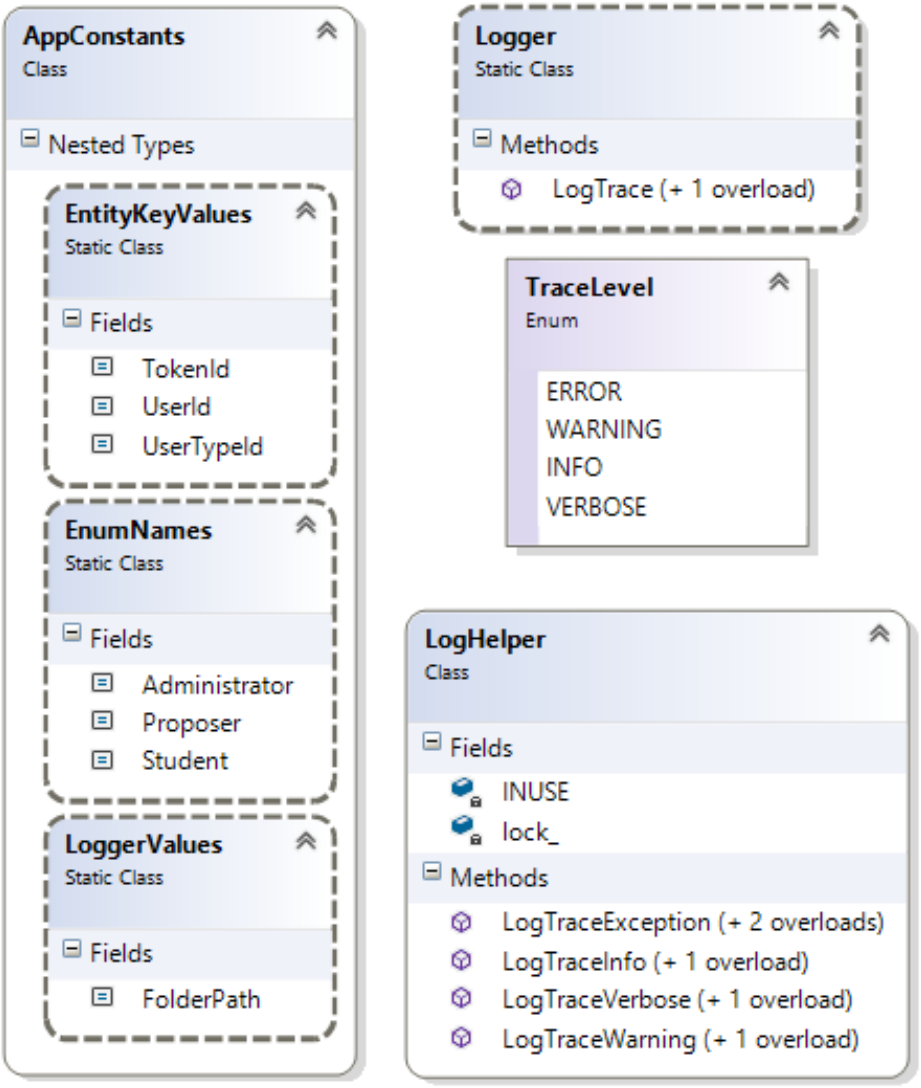

Figure 4: Core Class Diagram (Download from here)

5. SPA.Web.Services:

This is a stand-alone key component of the system providing RESTful (Representational State Transfer) API which can be wired up to any system to interact with the database.

App_Start:

This folder contains essential information required at the start of the service. This consists of two files: "WebApiConfig.cs" and "NinjectWebCommon.cs".

Please refer to the "WebApiConfig.cs explanation" and "NinjectWebCommon.cs explanation" for more details.

Controllers:

Controllers are the connection nodes of the OData service. Each action within the controller has its HTTP type such as GET for retrieving controller specific entities, POST for creating new entities, PUT for editing entity details, DELETE for deleting entity, etc.

A typical request (successful) process within a controller:
Controller receive the request > Accesses the request header > Identifies request type > Execute filters/attributes if applicable > Navigate to the specific Action > Passes details to corresponding repository > Send response that is returned from the repository.
So the controller acts as an intermediate between the client and the repository.

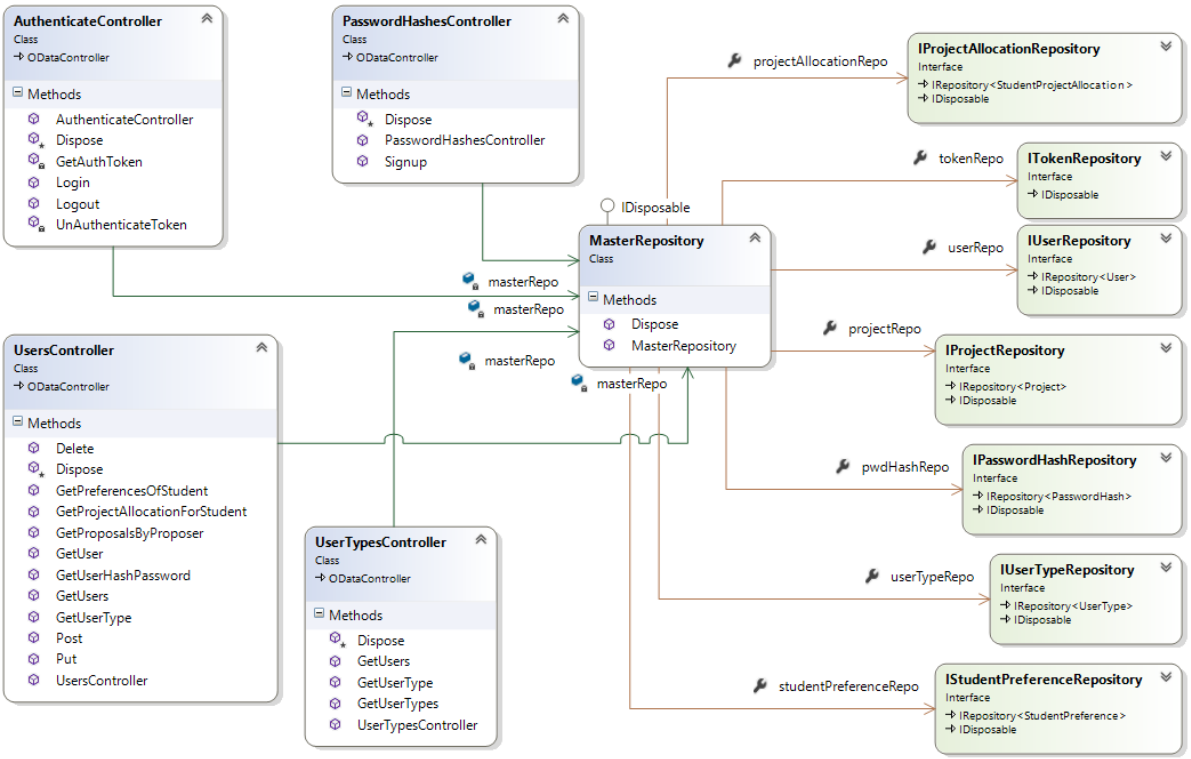

Figure 5: Controller Class Diagram (Download from here)

Repositories:

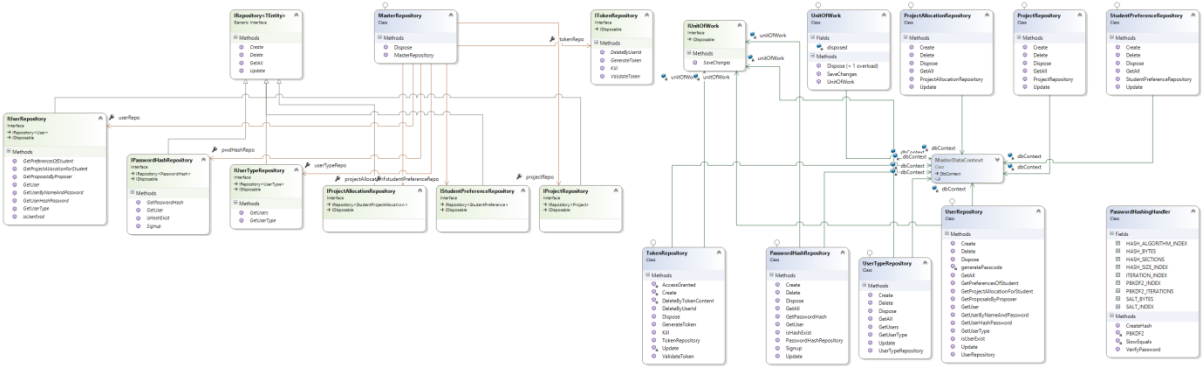

Figure 6: Repository Pattern Class Diagram (Download from here)

Repository Pattern:

Repositories are the active part of the system which use entity framework and interacts with the database.
Each entity is communicated via its controller and each controller derives information via its repository. The reason why these helpers are suffixed by "repositories", is because I have implemented repository pattern. Repository pattern allows all repositories to be accessed via a master repository. So controllers don't require creating instances of each repository, they are only required to access master repository to access all repositories. This makes the maintenance process quicker as any changes only need to be made in the master repository and its callers remain untouched.

Each repository implements its Interface to hide internal processes from the caller.
Example: UsersController access to MasterRepository and access IUsersRepository from available properties of the MasterRepository.
UsersController (caller) cannot access UsersRepository's `Dispose()` method even though it is a public method as it is not declared in the IUsersRepository.

Each interface of the repository implements the IRepository interface which enforces the CRUD (Create, Retrieve, Update and Delete) methods for each entity.

Unit of Work Pattern:

I have implemented Unit of Work pattern where I centralise the database interaction process.
Repositories make amendments in the objects and keep the instructions in the memory. If it is required to make those changes to the database, any repository calls SaveChanges() from the UnitOfWork class. When SaveChanges() method is called, entity

framework within the project gathers all the instructions, prepare SQL injections, connect to the database via dbContext and executes the queries.
The benefit of using the pattern is that I can observe the saveChanges() method and instruct it to behave in certain ways on specified circumstances. Such as catch exceptions, write messages to log file etc.
I can also combine many changes to various entities and save them all at once.

Repository Handlers:

Repository Handlers are help in hand to the repositories. Any processes, which are not directly related to the database, are implemented in handlers.

I have implemented Password hash process in handlers.

Please refer to the "Password Hashing and Salting" for more details.

Filters:

Filters are the properties which are executed prior to the action if specified. It is a gateway for actions. I have implemented class level and action level filters.

```
[ApiAuthenticationFilter(true)]
public HttpResponseMessage Login(){ … }
```

In the example above, the APIAuthenticationFilter will get executed before the login method.

There are two critical filter functionalities available within the project: Authentication ([`ApiAuthenticationFilter`]) and Authorization ([`AuthorizationRequired`]).

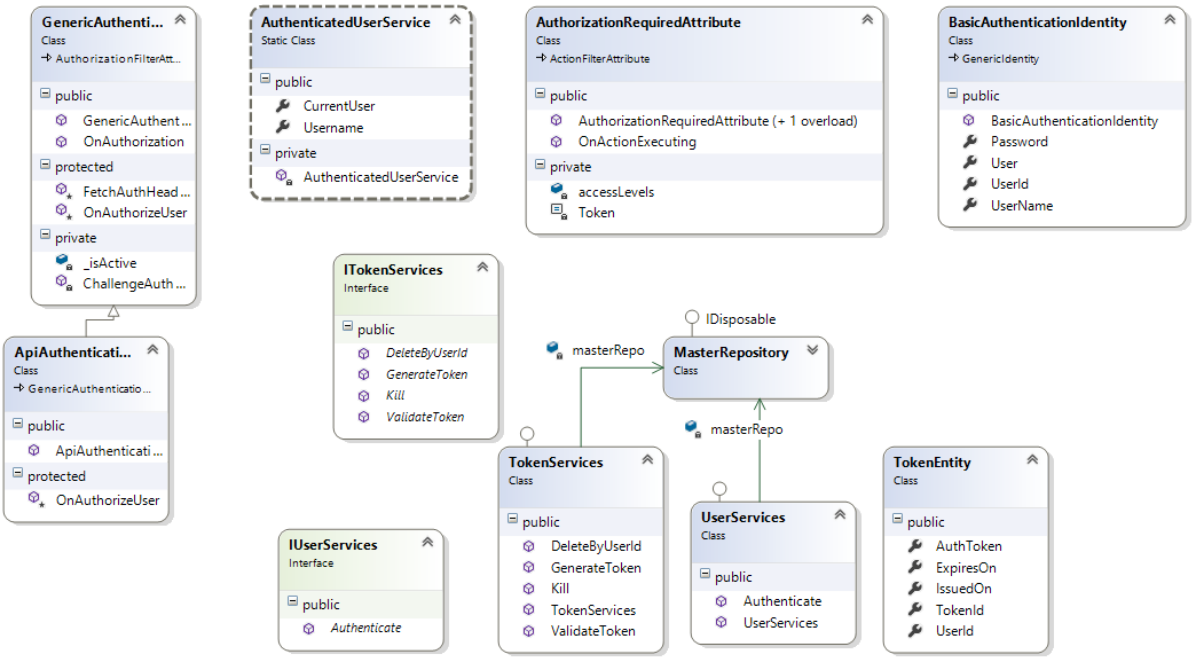

Figure 7: Filters Class Diagram (Download from here)

ApiAuthenticationFilter:

This filter has been used in two actions: Login() and Logout() within the AuthenticationController. The task of the filter is to extract the username and password from request header and authenticate user. If user is authenticated, then save the found user as an IPrincipal (current user request user) and respond the client via response header.

Please refer to the "ApiAuthenticationFilter explanation" for more details.

AuthorizationRequiredAttribute:

This filter attribute has been used in the actions where the user is required to be logged in to seek their identity and to confirm their authorization for the requested action.

For example: Within the UsersController class, the action Delete is decorated with

```
[AuthorizationRequired(AppConstants.EnumNames.Administrator)]
```

This means if the current user is an administrator, then they can delete the requested user. This filter assumes that the user is already logged in by `ApiAuthenticationFilter`. Therefore making sure that before a user uses any other services, they are logged in.

Let's say if the user has logged out from the system and then used the previous URL to delete a user, at this point the user is not logged in to the server, so they cannot execute the action and the server will return a 401- unauthorised response.

Please refer to the "AuthorizationRequired attribute explanation" for more details.

6. SPA.Web:

As we have seen the server how it behaves to various requests, let's see how the client is designed to send those requests.

I am using AngularJS 1.5.8 to render and process the client side. Front-end is using following technologies:

- **HTML** – to render pages
- **CSS** – to add custom themes
- **Javascript** and **Jquery** – are inherited by Angular for adding functionalities and animations
- **AngularJS** with **Typescript** – allows me to build components in typescript to make the client side coding more object-oriented. In other words, adds beauty to the code!

Please refer to the "AngularJS with Typescript" for more justifications.

App Folder Structure:

The AngularJS functionalities are designed within the `App` folder of the web project.

The App folder divided into following sub-folders.

1. **ClassLibrary :** Contains files defining classes such as Interfaces and their implementations, Enumerations etc. Generally reflects to the entities defined within the SPA.Entities project.
2. **Controllers:** Contains controllers of AngularJS.

   Controllers' main task is to provide an intermediate between HTML pages and angular services. They provide two-way DOM binding and invocation of events triggered by user via GUI elements.
3. **Directives:** Contains custom defined directives (none at the moment)
4. **Services :** The folder contains subfolders:

   I. *DataProcessServices:* provides interaction between controller and RESTful API by calling resources.

   II. *ResourceRepositories:* Provides access to available resource classes.

III. *Resources:* Contains $resource objects for generating HTTP requests by setting up corresponding URI and header properties.

IV. *UIServices*: These services don't require interacting with API however provide common functionalities for the views.

5. **Views:** Contains html pages for the front-end rendering

6. **App.js file**: Defines the angular module and inject basic references to run AngularJS. It also provides route configuration for easy page routing.

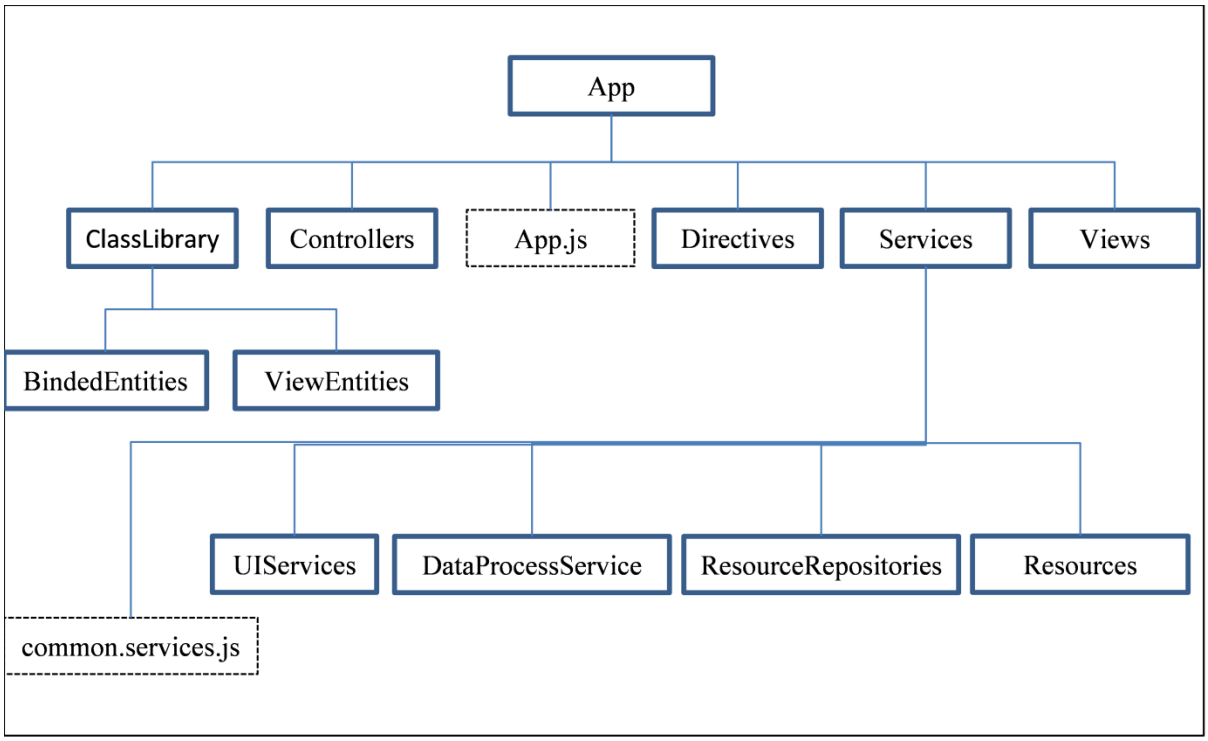

Figure 8: App folder structure (Download from here)

7. **SPA.SolutionTests:**

Unit Tests

I have implemented unit test project to test functionalities on the server. Some tests are in place such as testing custom logging and UserTypesController. More tests will be introduced as more functionalities put into place.

Unit Tests with Mock Objects

Mock objects allow defining database state and running controller actions with mock database context without executing queries to

the actual database. They generally used in conjunction with unit tests.
Please refer to the "Mocking functionality" for more details.

Implemented Features:

I have prepared GUI for user interactions and managed to cover following functionalities.

Login: Users can login to the system by providing their username and password. Appropriate messages are displayed on unsuccessful attempts. On login, user is directed to the relevant homepage according to their user types.

Signup: Homepage provides the signup functionality which allows users to create their accounts using a provided passcode generated on the user creation (for more information, refer to the User Registration section).

Logout: Users can logout from the system. This functionality will erase user's details from the browser's cookies, server session and delete user specific token from the database. User has to re-login to access the services again.

Administrator Functionalities:

Administrator's Homepage: When current user is recognised as an administrator, the system will redirect the user to the admin homepage. Administrators can access three different sections: Maintain Users, Setup Deadlines and Allocate Projects.

Maintain Users:

User Registration: Administrators can register new users by providing required information. When user is created, system generates a random 10 digit number and passes to the administrator which is then required to be passed to the created user.

Search Users: Administrators can search all users' details.

Edit User Details: Administrators can edit user details and save amendments on the database.

Delete User: Administrators can delete the loaded user.

Note: Application will display relevant error messages on unsuccessful attempts.

4. Software Architecture, Algorithms and Data Structures

1. Software Architecture:

The software architecture will remain the same as mentioned in the Prototype section.

2. Algorithms:

Research and implement appropriate algorithm to automate the student project allocation process.

3. Data Structures:

Data structure will remain the same as mentioned in the Database Service section.

4. Additional features and functionalities:

Final system will have the same back-end and front-end structure as mentioned in prototype, however, there will be some additional features and functionalities as stated below:

a) SMTP Client service:

Open source SMTP client will be in place to send system generated emails to users on following occasions:

A. An email with passcode details to the (created) user when administrator creates a new user.
B. An email with new proposal submission date to proposers when a proposals deadline is amended.
C. An email with new preference submission date to students when preferences deadline is amended.
D. An email to users with a link to reset password when they request to reset forgotten password.
E. Automated emails to students and proposers with their final project allocation when administrator allocates projects to the students.

b) Implement remaining features of the application:

Implement remaining functionalities below for administrators, proposers and students.

All Users' Functionalities:

Change User's Details:

User can change their own account details limited to their first name, last name and contact details. User also can reset their password which will require current password from the user.

Administrator Functionalities:

Maintain Users:

Search Users: Administrator can search users' details by users' first name, last name, user id and username.

Constraints:

- Only allow administrator to search students and proposers. This is to ensure that administrator cannot change other administrators' details. Administrators are generated by the system thus not amendable by users.
- Administrator cannot search their own details in the "Maintain Users" section. This prevents them from changing their own user types or other unauthorised details.

Setup Deadlines:

Administrator can set/amend project preference and project proposal deadlines.

Allocate Projects:

Administrator can retrieve student project allocation suggestions from system's automated functionality. Once the suggestions have retrieved, administrator can finalise a suggestion and amend it before confirming the allocation assignment process.

Proposer Functionalities:

Propose Project: Proposer can propose projects by providing details. Once proposed, the project then can be selected by students as a preference.

Edit/Delete Project: Proposer can edit/delete project details proposed.

Constraints:

- Only the proposer of the project can edit/delete project details. Other proposers can only view the proposed project details.
- Proposer can only delete a project if it has not been selected by any student.
- Proposer can edit the projects before the proposal end date.
- Proposer can propose new projects until students' project preference date is expired.

View/Search All/Individual Project Proposals:

Proposer can view projects proposed by all proposers and view own project proposals.

They can search projects by project title, proposer's username and project id.

View individual allocation:

Proposer can view the project allocations of their proposed projects.

Student Functionalities:

Preference Submission/Alteration: A student can submit/Edit project preferences.

Constraints:

- Student has to choose four project preferences.
- Projects cannot be repeated within any one student's preferences.
- Student cannot assign the same priority for more than two different projects.
- More than two projects proposed by the same proposer cannot be selected by a student.
- Submission/Alteration of preferences is not allowed once preference deadline has expired.

View/Search Proposed Projects:

Student can view projects proposed by all proposers. Student can also search projects by project title, proposer's username and project id.

View individual allocation:

Student can view the project allocations of their chosen projects.

c) Place remaining unit tests

On implementation of each functionality, as mentioned above, place unit test of relevant controllers and methods.

d) Host the application through home network

Application is hosted on virtual machine; however it cannot be accessed while connecting to university Wi-Fi as mentioned in Application Hosting section. If the application is hosted on a home network and made available to the public, then anyone can access the application.

e) Place Integration testing

Integration testing will allow me to test the system's GUI based functionalities.

f) SSL certification

If I manage to run the application through the home server then I will try to place SSL certificate on IIS server. SSL (Secure Socket Layer) encrypts vulnerable data such as username and password before sending it over the internet. This provides security for data transfer between client and server.

g) Documentations:

I will place comments and method documentations for easy understanding of the code.

h) Evolutionary algorithms

I will research on **evolutionary algorithms** to solve the project allocation problem. I will use several different approaches such as various parent selections, cross-overs, mutations and survivor selection within the algorithm and observe their performance to decide the final approach.

The algorithm will try to search for a solution where it is desirable to allocate the projects to students' according to their highest priority project preferences. By doing so I will also keep an eye on the number of projects allocated to the proposers as project proposer will be assigned as a first marker of the proposed project. I will make use of penalties based on level of infeasibility to penalize the infeasible solutions which can then be used to compare for final solution selection. The system will prepare few suggestions by using the evolutionary algorithm and suggest to the administrator. The administrator can then observe the penalties suggested within the

summary and select a suggestion. Administrator can also change the suggestion manually before finalising the allocation.

5. Diaries

Link to the diaries

6. Career Plan

I acknowledge feedback received from my supervisor in the preparation of this plan.

7. Bibliography and Citations

[1] Dr. Minku L.L. (2016) "*Automated Student Project Allocation Tool*", Available at: https://campus.cs.le.ac.uk/teaching/proposals/v6819.automated_student_project_allocation_tool , [Accessed on: 14/10/2016]

[2] Microsoft Corporation (2016). "*UML Use Case Diagrams: Guidelines*". Available at: https://msdn.microsoft.com/en-us/library/dd409432.aspx. Accessed on: [03-Oct-2016].

[3] Oracle(2000). "*Developing Application - Drawing the Entity-Relationship Diagram*". Available at: https://docs.oracle.com/cd/A87860_01/doc/java.817/a81358/05_dev1.htm.
Accessed on: [04-Oct-2016].

[4] Brandenburg L.(2016) *"How to Create an Entity Relationship Diagram"*. Available at: http://www.bridging-the-gap.com/erd-entity-relationship-diagram/.
Accessed on: [04-Oct-2016].

[5] Ambler S. W. (2003). *"UML 2 Class Diagrams: An Agile Introduction"*. Available at: http://agilemodeling.com/artifacts/classDiagram.htm. Accessed on: [05-Oct-2016].

[6] Thomson P. (2014). *"aims and objectives – what's the difference?"*. Available at: https://patthomson.net/2014/06/09/aims-and-objectives-whats-the-difference/. Accessed on: [07-Oct-2016].