May 2017

Dissertation

Automated Student Project Allocation System

CO3015 Computer Science Project
Department of Informatics, University of Leicester

Anita Lad

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Anita Lad

Signed:

Date: 04/May/2017

Table of Contents

1. Abstract

The informatics department at university carries out a yearly process of student project allocation for final year students. The process involves allocating projects to students by making sure that projects assigned to a proposer are in accordance with their maximum supervision capacity and a project, assigned to a student, is from their ranked project preferences.

The current web application used by the university only provides a Graphical User Interface (GUI) to carry out tasks. The interface is poorly designed and not offering better user experience. In order to carry out the allocation process, administrator has to run an external script and derive suggestions. Each allocation from the suggestion is then inserted into the system manually. The whole process involves a lot of time and effort and becomes a very tedious task when proposers' supervision capacities change frequently.

I have developed a web application that serves GUI to the users. The interface is designed with use of cutting edge technologies to provide a better user experience in terms of reliability, robustness and ease of use. The intuitive design provides validations, help tags for guidance and understandable explanations on invalid actions throughout the pages.

In addition, the application is equipped with an automation feature that can provide various suggestions with different trade-offs between the overall project priorities that are assigned to students and overall distribution of assigned projects between proposers. The provided suggestions serve a primary base for the allocation process so that administrators can make necessary manual amendments before finalising the allocations. The automation process is optimised with advanced and fast multi-objective evolutionary algorithm NSGA-II (Non-Dominated Sorting Genetic Algorithm – II). An administrator can also adjust the parameters and choose between different objective function approaches, to direct the algorithm's search to a desired direction.

This automation feature could save time and effort invested by administrators in comparison to the majority of the manual process involved within the existing system. It also offers flexibility, alterations and robustness throughout the entire process.

This dissertation explains the entire process of the project life-cycle from design to completion along with the reasons for the chosen technologies and approaches.

2. Introduction

University's informatics department is currently struggling with the undertaking of the project allocation process for final year students. The department is using a range of systems to carry out the entire process along with a lot of manual processing.

Projects are proposed by lectures (proposers) and are selected by students according to a rank of one to four. A project is supervised by its proposer. The allocation process involves allocating a project to a student. [1]

The **main aim** of this project is to develop an application that provides an interface for various users to submit project proposals/preferences.

The **secondary aim** is to combine the interface and the automation feature within one system. The automation feature suggests a reasonable allocation of students to projects, by considering the preferences made by a student and the overall distribution of project assignments of proposers whilst trying to not exceed their supervision capacity.

The project 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 as well as adding the automated allocation feature.

Objectives: In order to achieve the aim, I am required to –

1) Setup a server to host the system which will publish the web application.
2) Setup a database to store users' proposals, preferences and relevant data.
3) Build a web application using MVC framework to provide interface for all operations.
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. Evolutionary algorithms evolve a set of possible solutions until a "good enough" solution is found.
7) Learn about technologies involved within development-
   - 7.1.1. Using good programming practices such as Dependency Injections, Repository Pattern, Unit of Work Pattern, Message logging, etc.
   - 7.1.2. Privacy, Safety and Security such as Password Salting/Hashing, SSL certification, Individual users' authentication and authorization.
   - 7.1.3. Configurations such as Network settings, Database Configurations, RESTful API settings, etc.

Distinctive Features (Challenges):

I have managed to develop some enhanced features within the project.

1. **A Remote Server** is setup and configured from home which is hosting the application live.
2. The web application is secured by **SSL** (Secure Socket Layer) certificate therefore the data is encrypted within the transport along with providing a **secure DNS** for clients.
3. Use of **Mock Objects** allows forming the unit test cases without connecting to the database and each scenario has a different set of objects to test against.
4. Proposed a **formulation** and designed a **MOEA** for the problem including multiple project distribution fitness calculation approaches: **Entropy** and **Standard Deviation**
5. Implementation of an advanced algorithm **(NSGA-II)** to introduce **automation** of project allocations and facilitated a choice of solutions amongst different trade-offs for the administrators.
6. Passwords are stored with secure hashing algorithms **(SHA-256)** and random **Salting** technique with the facility of defining distinct random algorithm parameters for different users.
7. The system has been supported by customised **Message Logging** where errors and exception occurrences are automatically logged into files to review the causes.
8. Setup and configurations of SMTP server and client service to send **Automated Emails**.

Overview of the Dissertation:

Chapter-3 describes the research on the related work and evaluation of findings on the various algorithm approaches. Chapter-4 outlines the overall requirements of the system and Chapter-5 justifies the design choices made for developing the system. Chapter-6, 7 and 8 provide the details of the implementation process along with the reasons for the choices made for the system architecture, GUI and the algorithm approach respectively. Chapter-9 analysis the test results obtained and finally Chapters-10 and 11 summarises the overall achievements and challenges of the project.

3. Background and Related work

3.1. Related Work (GUI)

In order to gain detailed understanding, I have evaluated some systems offering similar functionalities.

University's existing system provides an interface to carry out the proposal and preference submissions; however, it is not equipped with automated allocation functionality. The allocation process is undertaken by running a script outside of the application and placing each allocation manually within the system.

The whole process requires a lot of manual processing thus become a very challenging task when proposers frequently change their supervision capacities.

Moreover, the interface of the existing system is not user friendly as it fails to provide sufficient help information on functionalities.

There is an online application available called, **ProjectList** [22] which provides similar functionalities. However, it only allocates the projects that are preferred by one student as a first choice and the rest of allocations are required to be carried out manually.

3.2. Related Work (Allocation Problem)

The well-known approach introduced by Abraham et al, (2003) [23]. This is based on "two-way-matching" algorithm and considers the *stable* matching of students to projects. The stable matching informally involves matching the student-project pairs without violating the constraints of the problem.

I have observed the following limitations which have been removed from the algorithm used within my approach -

3.2.1. It enforces non-zero project proposals by proposers.

3.2.2. Disallows the pre-allocations via self-proposals by students.

3.2.3. The approach is based on the student-optimal allocation that emphasis on student's priority regardless of proposers' project distribution.

3.2.4. Students at the beginning of the list are more likely to be assigned their higher priority projects.

3.2.5. The algorithm does not offer flexibility in terms of choosing solutions with different trade-offs.

3.3. Background

Due to weaknesses noted above, I decided to use an algorithm that can deal with multiple objectives and provide suggestions with different trade-offs between objectives.

The evolutionary algorithms are capable of dealing with multi-objectives and very efficient in terms of providing "good enough" solutions in accordance with defined objectives.

I decided to adopt the Multi-Objective Evolutionary Algorithm (MOEA) approach as it considers each objective separately and provide different trade-offs between defined objectives. This would suit this particular project's problem of dealing with two objectives: 1) Students' assigned priorities and 2) Assigned project distribution between proposers.

The different MOEA approaches I came across are explained below.

Strength-Pareto EA-(SPEA) is introduced by Zitzler and Thiele (1998, cited in Deb et al, 2002). The approach emphasises on preserving non-dominated solutions externally for each generation. The time complexity is measured as **$O(MN)^3$** where M is number of objectives and N is population size.

Pareto-archived evolution strategy-(PAES) is introduced by J. Knowles and D. Corne (1999, cited in Deb et al, 2002). This approach suggests single parent – single offspring method and based on maintaining archived solutions on each of the iterations. The complexity of the approach is **$O(MN^2)$** where N is the population size and the proportion of archive length and M is the number of objectives. PAES has no emphasises on diversity and this could result into loosing better solutions within the un-covered space.

3.4. Non-dominated Sorting Genetic Algorithm (NSGA)

The research performed by Deb et al, (2002) [17] shows that **NSGA-II** outperforms both the above mentioned SPEA and PAES in terms of convergence and diversity.

NSGA is a one of the most famous multi-objective evolutionary algorithms. It has been used in similar allocation problems such as software project scheduling.

How does it work?

NSGA first generates random N solutions (Population size). It then sorts solutions in various fronts according to their fitness of defined objectives. Each front (a group of solutions) contains solutions that dominate the solutions of the next front. If solution-A is equal or better than the solution-B in all objectives and solution-A is strictly better than solution-B by at least one objective, this means **solution-A dominates solution-B** [19]. The solutions that do not dominate each other are combined together in the same front.

It iteratively chooses the best solutions (children) from the available solutions (parents). The chosen solutions are then altered in pairs to improve their fitness (cross-over); each child solution is mutated (mutation) to introduce diversity. Changed children solutions are then combined with parent solutions and sorted in the different dominated front according to their fitness and crowding distance. The **crowding distance** is a degree to measure the distance from one solution to its neighbours. The resulted offspring solutions of size N are taken to the next iteration.

The final iteration returns a set of offspring solutions that provides suggestions with different trade-offs between defined objectives.

3.5. NSGA-II:

There are two approaches available in NSGA: naïve NSGA and NSGA-II. The NSGA-II uses the Fast Non-dominated Front Sorting Approach (FNFSA), which improves the sorting process to **O(MN^2)** (This is better than **O(MN^3)** in naïve NSGA). The NSGA-II also emphasises on diversity and elitism in the solution selection process as opposed to the naïve NSGA.

The benefit factors of the NSGA-II are mentioned below:

3.5.1. Fast Approach:

Calculate two entities per solutions:

1) Domination count `np` - The number of solutions which dominate the current solution
2) Dominated set `q` - A set of solutions that the solution dominates. This requires **O(MN^2)** comparisons.

3.5.2. Diversity Preservation:

NSGA-II ensures diversity by crowded-comparison approach. It emphasises on solutions that have highest crowding distance. This improves the chances of discovering better solutions from the un-covered region.

3.5.3. Elitist survivor selection:

This approach selects the best individuals according to their fitness and the crowding distance from the combination of parent and children solutions. Any solutions with lower fitness yet better crowding distance also have some probabilities to be selected.

This approach eliminates the limitations of fitness based delete-worst and age-based survival selection [19]. Delete-worst ends up in premature convergence state and age based approach would lose best solutions from the old generation.

4. Requirements

These are divided into two parts:

1) System Development
2) Implementation of an Algorithm

4.1. System Development Requirements:

I have implemented all the requirements that have been set at the start of the project, including essential, recommended and optional that are described below-(Objective 3).

Functional Requirements:

The system should allow three different users: Administrator, Proposer and Student

4.1.1. Administrator Functionalities:

4.1.1.1. Essential Requirements

1. Administrators are responsible for registering proposers and students. They can also amend details of registered users.
2. They can delete non-admin user accounts.
3. Are allowed to set/amend proposal and preference submission deadlines
4. Can request the system to provide suggestions on allocating projects to students.
5. Can save the final allocation to the system.
6. Constraints:
    1. Proposal deadline date has to be set before the project preference deadline date

4.1.1.2. Recommended Requirements:

7. Administrator can make changes to the suggested allocation provided by the system before finalising the allocations.

4.1.2. Proposer functionalities:

4.1.2.1. Essential Requirements

1. Proposers can sign up to the system to activate their account.
2. A new project can be added by a proposer.
3. Proposers can amend/delete projects proposed by themselves only
4. Can view projects not proposed by the proposer themselves.
5. Proposer can view their project allocations at any time.
6. Constraints:
    1. Proposer can delete their proposed project details before the proposal deadline date
    2. Any project can be deleted only if it has not been chosen by a student.

4.1.2.2. Recommended Requirements

7. Proposer can reject/accept self-proposal requests made by a student.

4.1.3. Student functionalities:

4.1.3.1. Essential Requirements

1. Students can sign up to the system to activate their account.
2. All projects proposed by lecturers can be viewed by the student and similarly can be selected as a preferred choice.
3. A maximum of four project preferences can be chosen.
4. Project preferences can be amended more than once
5. Once allocation process is successful, student can view their project allocations
6. Constraints:
    1. Four projects have to be chosen for a successful submission to avoid errors
    2. Preferences can only be amended before the preference end date
    3. Can only select one project/priority once in a submission
    4. Student cannot select more than two projects from the same proposer.

4.1.3.2. Recommended functionalities

7. Request a self-proposal to any available lecturers via system.
8. Constraints: Self-proposed project is automatically set as a first priority and cannot be overwritten unless the request has been rejected.

4.1.4. Common User functionalities:

4.1.4.1. Essential Requirements

1. Users can login/logout from the system.
2. On successful login, user can access the system related to their access level.
3. User can change their basic account details and password at any time.

4.1.4.2. Optional Requirements

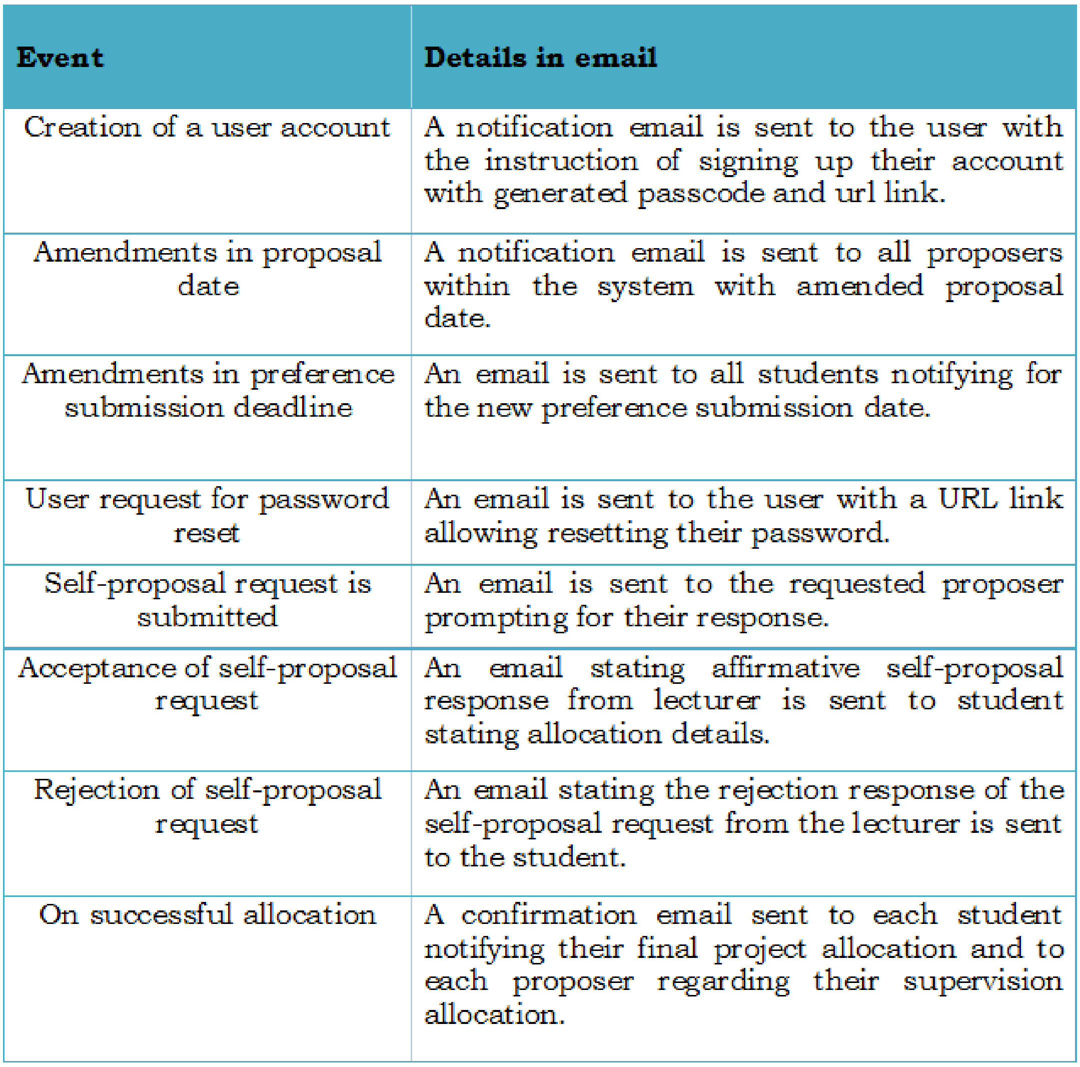
4. Users can requests for a link to reset forgotten password.

4.1.5. System functionalities:

4.1.5.1. Recommended Requirements

1. System provides client-side and server-side validations for data inputs. Users are notified of relevant messages on errors/exceptions.

2. Artificial Intelligence algorithms are used for student project allocations in order to produce efficient solutions.
3. Sends relevant notification emails to corresponding users on various occasions as described in figure-4.1-1.



Event	Details in email
Creation of a user account	A notification email is sent to the user with the instruction of signing up their account with generated passcode and url link.
Amendments in proposal date	A notification email is sent to all proposers within the system with amended proposal date.
Amendments in preference submission deadline	An email is sent to all students notifying for the new preference submission date.
User request for password reset	An email is sent to the user with a URL link allowing resetting their password.
Self-proposal request is submitted	An email is sent to the requested proposer prompting for their response.
Acceptance of self-proposal request	An email stating affirmative self-proposal response from lecturer is sent to student stating allocation details.
Rejection of self-proposal request	An email stating the rejection response of the self-proposal request from the lecturer is sent to the student.
On successful allocation	A confirmation email sent to each student notifying their final project allocation and to each proposer regarding their supervision allocation.

Figure_4.1-1: Email Notification list

Resources:

- **Internet**: Internet connectivity and a web browser are required to access the application.
- **Server**: A web application hosting server to host the application.
- **Database**: The application will be storing information on the database.
- **Third Party Libraries**: The whole system will make use of various third party libraries to provide various functionalities. Such as :
  Front-end support: HTML, CSS, Bootstrap, JavaScript, JQuery, JQuery-UI, AngularJS and JSON

.NET (backend) support: Entity Framework, OData/Web API services and LINQ

Quality Attributes:

- The system should provide better **user experience** than the current university system.
- The application is required to be **responsive** and **informative** to users' actions.
- Provide support for various **user roles** such as Administrator, Project Proposers and Students.
- The authentication and authorization checks are required to be implemented to **prevent unauthorised user** access.
- System will **safely** store users' passwords by using secure encryption methods such as SHA-256 hashing and use of random salting.
- Should ensure users' **data privacy and security** by implementing SSL certificate on the server to secure the data transfer between client and the server.

4.2. Algorithm Requirements:

A detailed description of algorithm is described in Ch:8.1.

A Multi-Objective Evolutionary Algorithm (MOEA) was required to be implemented. The greater detail on the implementation of the algorithm approach is described in Chapter 8.

5. System Specification and Design

This chapter describes the design phase of the project before the implementation.

Sub-section-5.1 illustrates the use case diagram to visualise the requirements. Sub-section-5.2 explains the database structure design. Finally, sub-section-5.3 draws attention to a navigation structure of the website design.

5.1. Requirement Structure

Following Use Case Diagram (Figure-5.1-1) will aid visualising the system's requirements, predecessor tasks and the overall requirement structure.

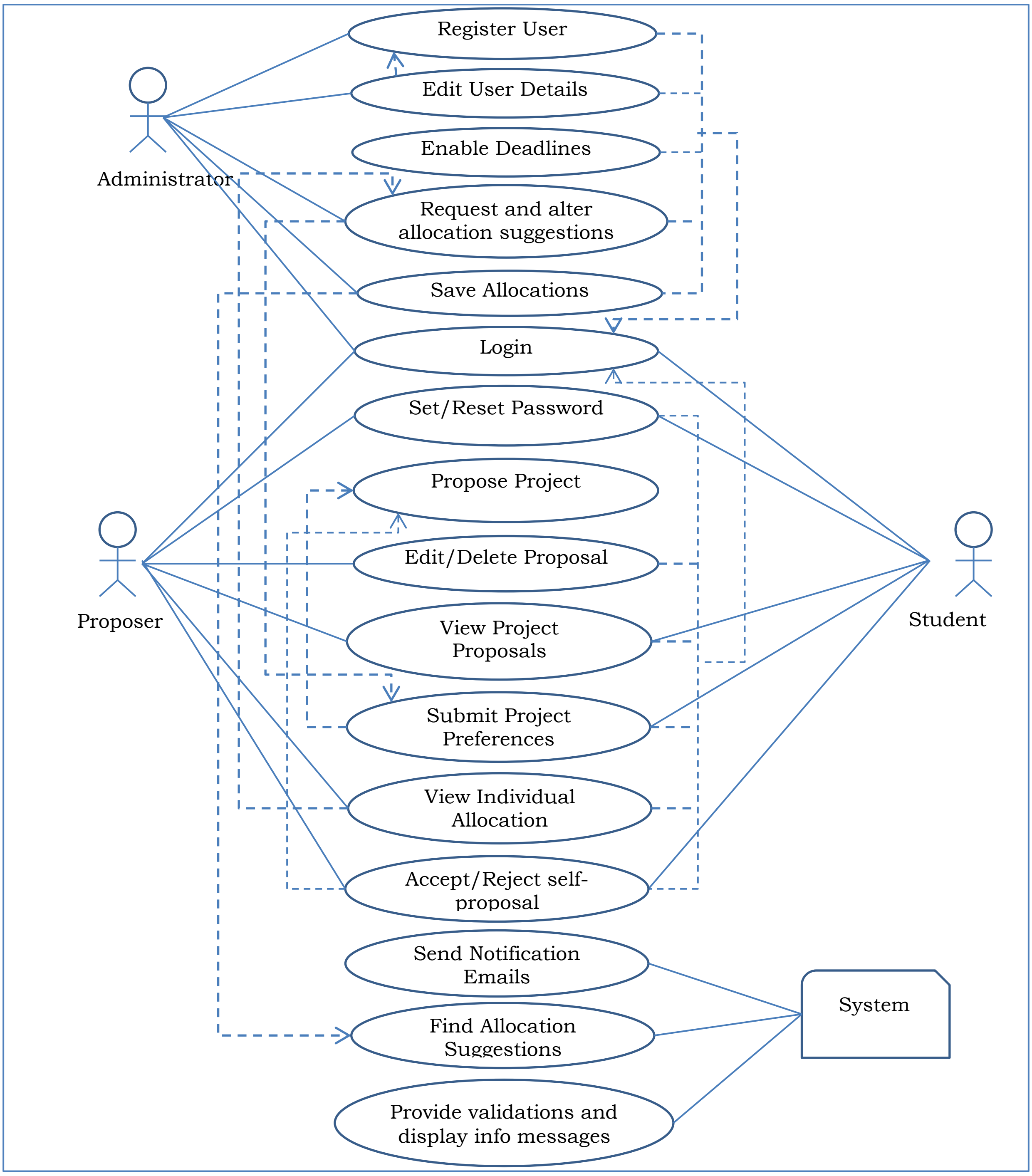


Figure 5.1-1:Use Case Diagram

5.2. Data Structures

There are eight data tables generated to store system data. I established various relationships to derive related data.

The `User` table stores information about users. `User` table is connected with `UserType` table with Many-to-one relationship which stores user roles. A user can have a password therefore `User` and `PasswordHash` table are connected with one-to-one relationship. The password related data is stored separately to the `User` table to keep vulnerable information separate from the general information. The `Token` table stores clients' token information for each logged in user. A user can be logged into the system from many devices at any time, thus can have many tokens so the one-to-many relationship has been setup between `User` and `Token` tables.

The `Project` table stores the data related to a project along with their proposers' information. A proposer can propose many projects thus `User` and `Project` tables are connected with one-to-many relationship.

Each student selects four projects by their priorities. The `StudentPreference` table stores student's preferred projects' information along with the priorities.

The `Deadline` table stores deadlines for submitting project proposals and preferences.

Finally, the `StudentProjectAllocation` table stores the student-project allocation. A student can be allocated at most one project thus one-to-one relationship is established between `StudentProjectAllocation` and `Project` tables. However, a project can be assigned to many students thus there is one-to-many relationship established between `StudentProjectAllocation` and `Project` tables.

Further information on relationships is available in ERD diagram figure-5.2-1.

Information on schema can be found in appendices_29.

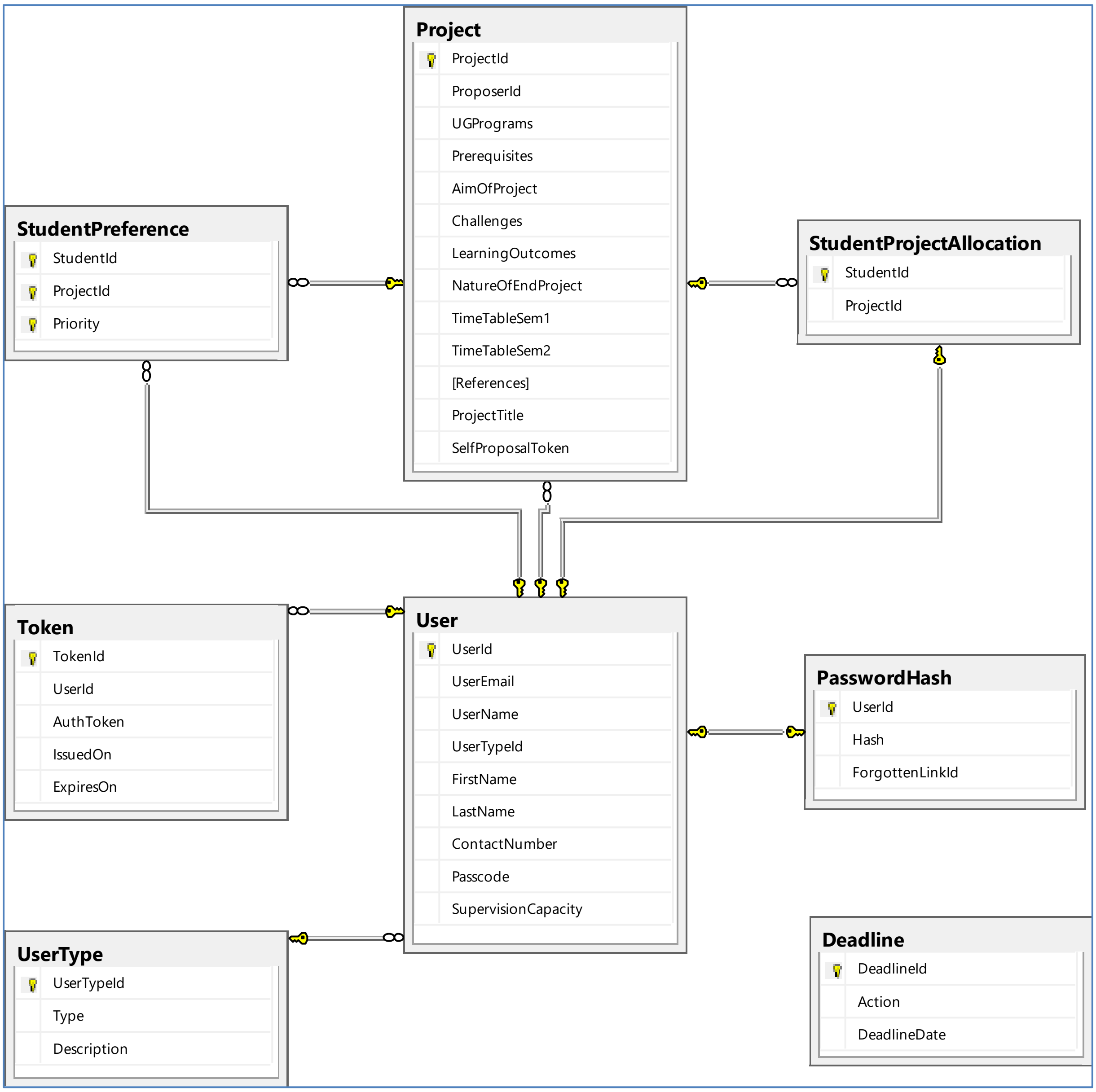


Figure 5.2-1:ERD_diagram (appendices_1)

5.3. Web Application Navigation Structure

The home page of the web application contains a login page. On successful login, the user is directed to an administrator, proposer or a student area depending on their access level.

The **help/info tags** are made available to provide support throughout the application.

Administrator and Student areas also provide facilities of **aside bar panels** to quickly access important functionalities without scrolling.

There are some blocks of the page which are hidden on the page load and are only displayed on invocation of the particular events. This is to hide unnecessary details to provide a **clear structure of the page**. Example explained below,

> *Project Allocation Area:* The “Suggested Allocation” section only becomes visible when user requests for the “Find Allocations”. The “Selected Suggestion” section only unwraps, when the user selects particular data points from the suggestion chart.

The navigation structure of GUI is described at figure-5.3-1.

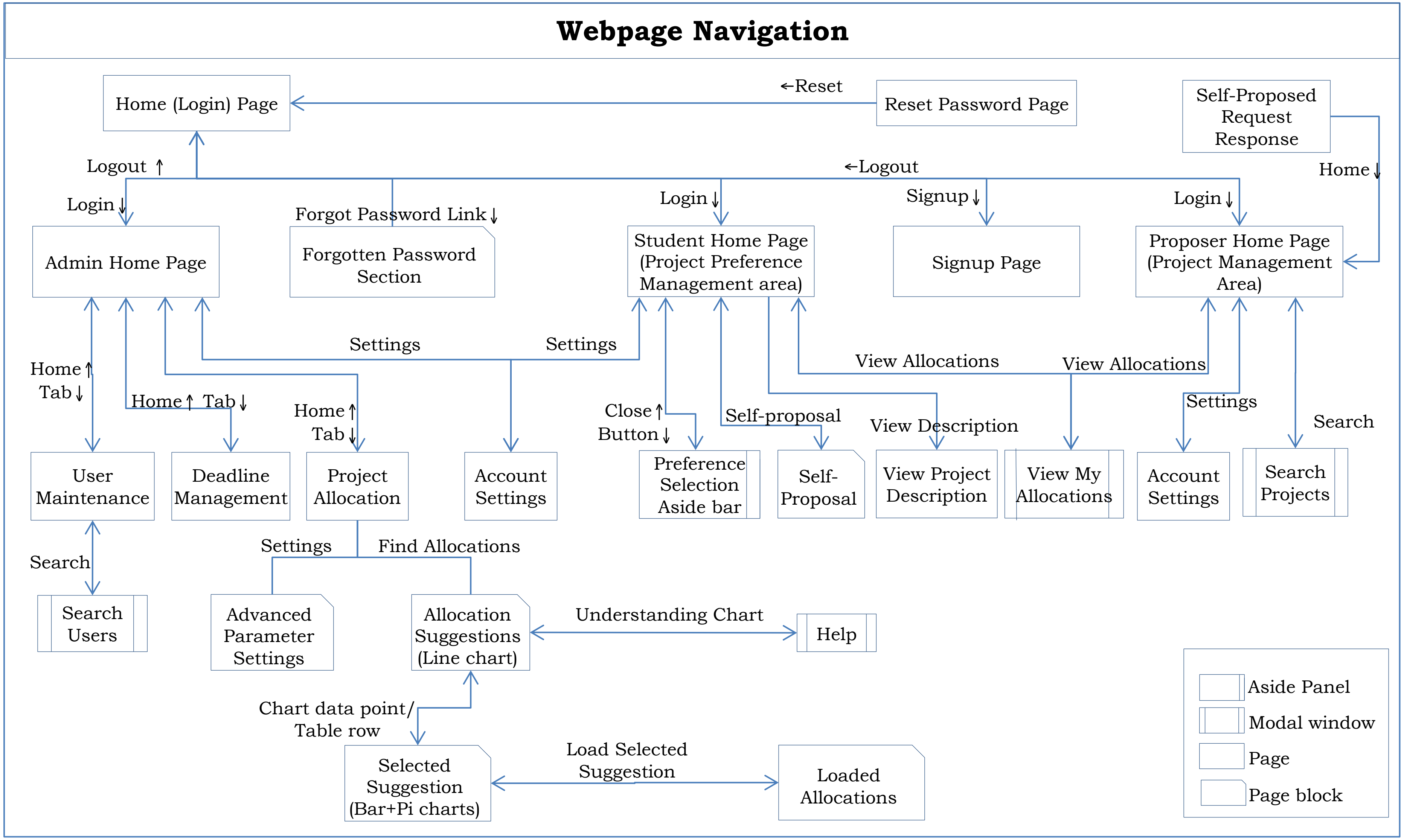


Figure_5.3-1:Webpage Navigation

6. System Structure Implementation (Solution)

This chapter explains the implementation process of the system.

Sub-section-6.1 clarifies the reasoning for the selection of the development environment. Sub-section-6.2 explains how I have setup a web server to run the application on localhost and remote server. The subsection-6.3 explains the DNS and networking configuration. Once, the web server, networking and database were setup, the subsection-6.4 describes the process of placing SSL certificate on the remote server. Sub-section-6.5 explains the installation and configuration of the database server.

Subsection-6.6 describes the application implementation structure along with each of its projects. Subsections 6.7, 6.8 and 6.9 describe the implementation of the design patterns to make the project maintainable and testable.

The important aspects of the application such as authentication/authorization and security of users' passwords are discussed within subsections 5.10 and 5.11.

Finally subsection-5.12 describes the implementation of the SMTP mail service within the system.

6.1. Nature of the System:

I decided upon developing the system as a web application due to future plans to move to the web development in .Net and C# area and also help me familiarize with the areas of improvement which I identified during my work placement.

6.2. Application Hosting (Server) (Objective-1)

I setup an IIS (Internet Information Service) web server available from Microsoft onto the localhost for day-to-day development and testing.

In addition to the localhost, I also decided to setup a system to host the web server which allows client machines to connect to the application via any network.

A Virtual Machine is an effective option for a web hosting platform as it doesn't require any additional hardware components to set up and run. I configured the virtual machine on my laptop (two systems on one machine) [1]. It was set to provide web hosting for any client machines sharing the same network. However, this is not supported by university network due to security reasons.

In order to solve the problem, I then decided to setup a server system on a separate physical machine and routed with a home router to allow client connections. I installed a Windows server 2012-r2 operating system on a machine and configured an IIS server to host the application. First, I completed the entire configuration and tested it through home network which was successfully allowing clients' requests. The next step was to make the server live and accept the requests from the clients that are connected to any network. I have discussed the networking configuration in the next section.

6.3. DNS and Network Configuration (Objective-7.3)

In order for server system to accept the client requests from the internet, I configured a port 8080 on a home router and assigned a DNS (obtained from No-IP) to the server machine to map clients to the server machine's IP address and a port number.

DNS host name: http://spasys.sytes.net/SpaSystem.Web/.

6.4. SSL (Secure Socket Layer) (Objective-7.3)

DNS routing successfully transferred data to and from the server. In order to make the transit secure, I was required to implement cryptographic approach on the data during transit.

SSL technology enables cryptography procedure on server and client via SSL certificates. I followed a tutorial by OrchardProjects,(no date)[16] and configured Self-SSL certificate by obtaining another DNS host name with https binding and configured port 443, on the home router, IIS server and on the No-IP service.

The secured DNS: https://anitalad.ddns.net/spaSystem.Web/

As the certificate is not certified by the trusted third party, the web browser would display a warning message which can be ignored to proceed to the webpage loading.

6.5. Database Service (Objective 2)

I have used Microsoft compatible SQL Server Express as a database engine and generated a database called "SPA" which stores information about various parts of the system. More information about database schema can be found at 5.3 Data Structures.

6.6. System Structure (Objective 3)

This section explains the implementation of the overall system structure and its components.

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

The figure-6.6-1 describes the dependency relations between projects.

The entire solution is setup as **MVC** tier approach. The Data project provides **Models** (represents database entities), Web project provides **Views** (GUI) and the Web.Services provides the **Controllers** (handles interaction from view to database via models).

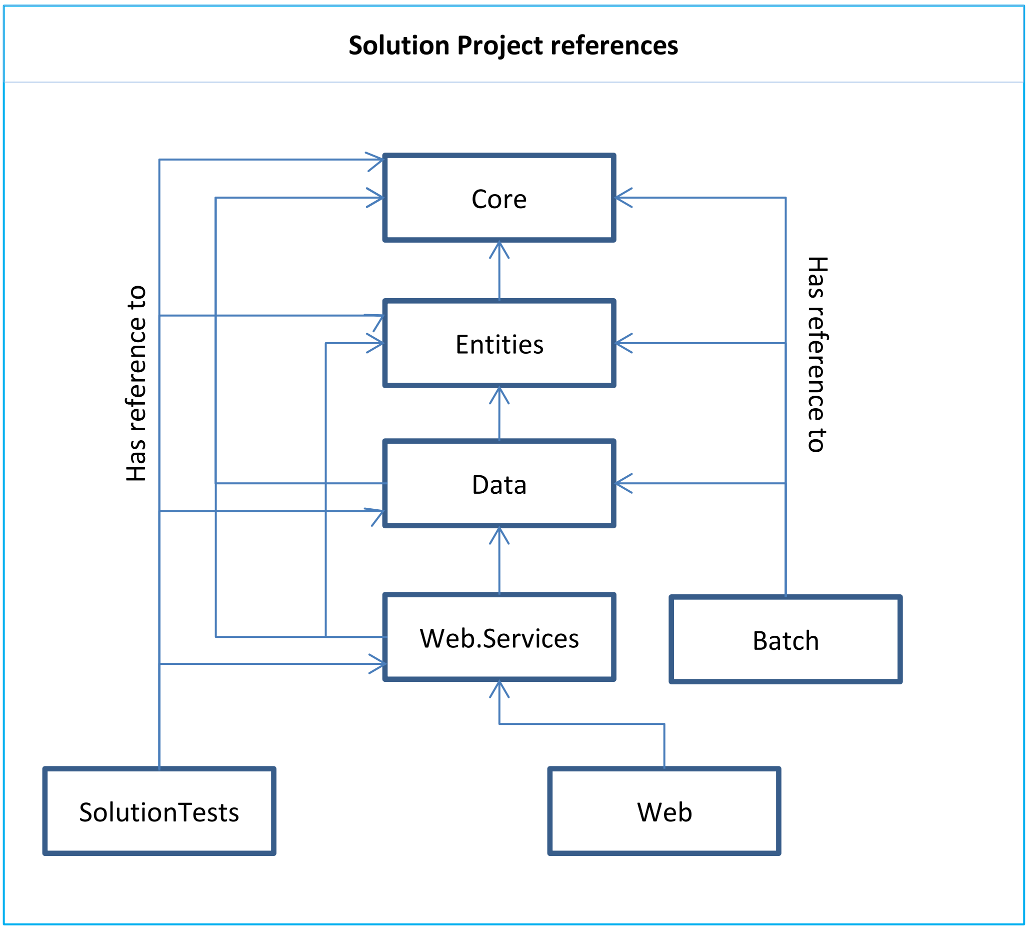


Figure 6.6-1: References of Solution's Projects

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. The Figure-6.6-2 displays the entity structure.

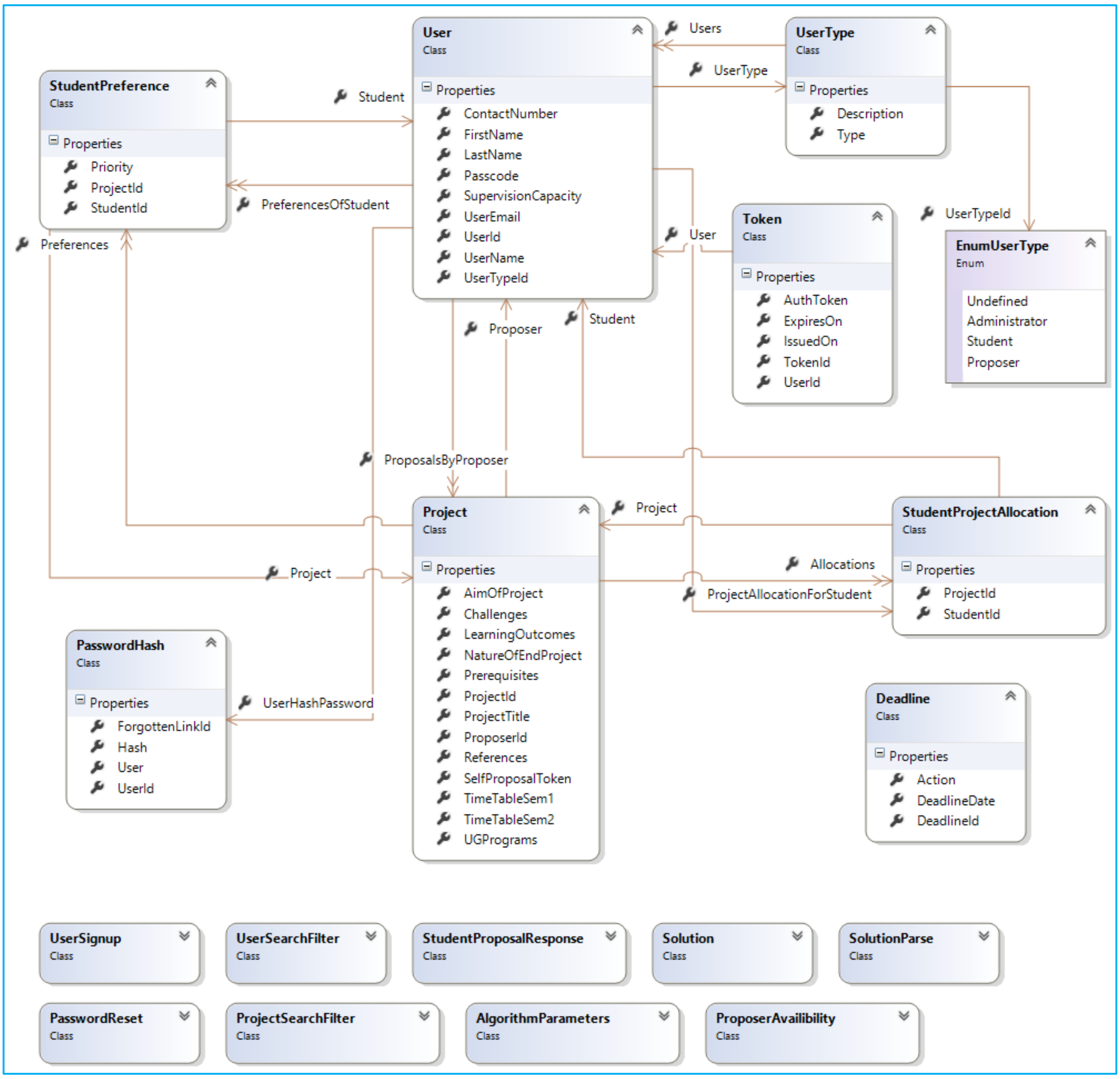


Figure 6.6-2: Entities Class Diagram (see appendices 3)

3. SPA.Data:

The project contains “MasterDataContext.cs” file which contains details of the database model bindings and used by Entity Framework while interacting with the database entities. The Entity Framework is

Microsoft provided ORM (Object Relation Mapping) from binding database tables to the memory objects. View MasterDataContext file for more details.

4. SPA.Core:

The main functionality of this project is a "message logging". I have implemented "`Logger.cs`" and "`LogHelper.cs`" classes to write messages to the file. Messages are divided into four stages - Error > Warning > Info > Verbose. The structure is displayed in figure- 6.6-3.

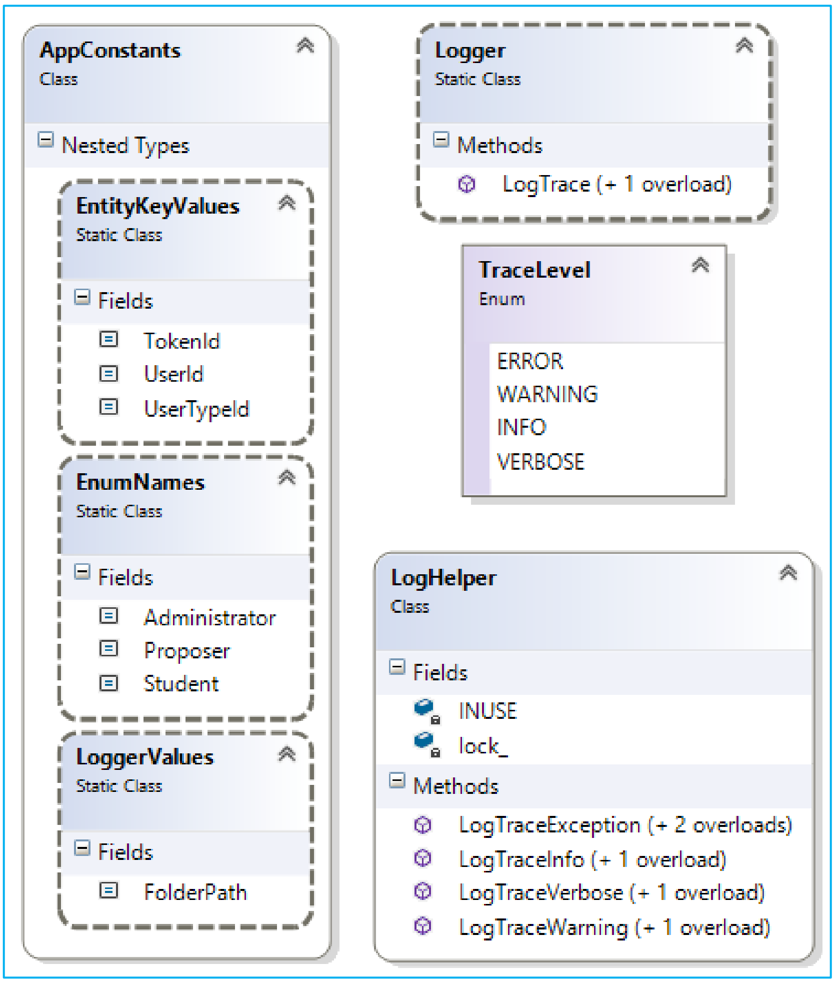


Figure 6.6-3: Core Class Diagram (see appendices 5)

5. SPA.Web.Services:

This is a stand-alone key component of the system providing RESTful (Representational State Transfer) API that 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`".

1. *“WebApiConfig.cs”*:

Any request will first trigger `Register()` method and then diverted to the specific action. Any features which are required to apply to all controllers such as authentication, enumeration rules, exception handling rules etc. are defined within this section.

The main operation of the method is to enable URL specifications of the entities and controllers’ connections.
For example:
(WebApiConfig.cs-Line-55) - builder.EntitySet<User>("Users");

This indicates that the entity set “User” table is connected with `UsersController` with the OData URL suffix “Users”.

I have used OData (Open Data Protocol) service to enable controllers’ communications. The OData service is a RESTful service which allows controllers’ actions to be triggered by using HTTP requests. In addition, it also allows constructing basic queries within URI string. Such as:

- `“http://localhost/SPA.web.Services/Users?$count”:` Count the numbers of users available

This practice reduces the numbers of actions implemented within the controller and provides robustness in queries.
See appendices_8 for the code sample.

Controllers:

Controllers are the connection nodes of the OData service. Each action within the controller can be communicated via certain HTTP calls such as GET for retrieving entities, POST for creating new entities, PUT for editing entity details, DELETE for deleting entity, etc. Each entity within the database has its own controller.

A typical successful request process within a controller:

```
> Controller receives the request
  > Accesses the request header
    > Identifies request type
      > Executes filters/attributes if applicable
        > Navigates to the specific Action
          > Passes details to corresponding repository
            >  Sends  response  that  is  returned  from  the
               repository
```

Figure 6.6-4: Controller Request Processing

The structure of the controller class is shown in figure-6.6-5.

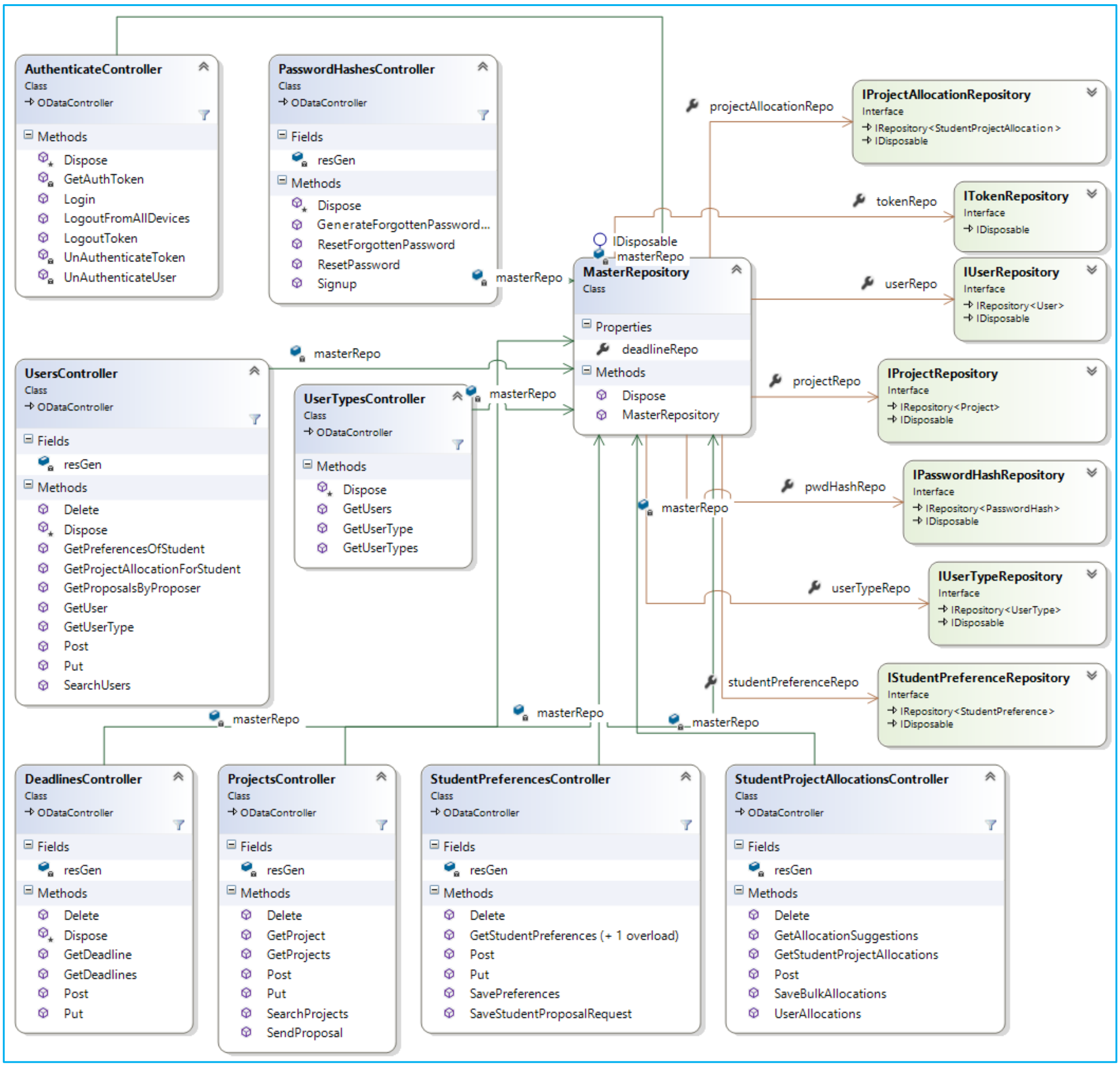


Figure 6.6-5:Controller Class Diagram (see Appendices 6)

Repositories:

Repositories are the active part of the system which interacts with the database. Please see Ch 5- Repository Pattern section for more details.
The structure/object diagram of the pattern is available in: appendies_7.

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 following processes within the handlers -

1. **Password hash** :
   Described at Section 6.11-Password Security
2. **SMTP emails** :
   Described at Section 6.12- SMTP Client Service
3. **Evolutionary Algorithms**:
   Described at Ch8 – Evolutionary Algorithm Implementation

Filters:

Filters are the properties which are executed prior to the action. They are described as a **gateway** for actions. Filters mainly contain the authentication and authorisation related features. Please read Ch 6- Authorisation section for further information.

6. SPA.Web:

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

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 building components in typescript to make the client side coding more object-oriented and less prone to the errors. In other words, adds beauty to the code!

Why AngularJS?

AngularJS is one of the frameworks in demand by most of employers within the web/mobile application development. It has advanced approaches to allow two-way DOM parsing. I am using AngularJS 1.5.8 to render and process the client side.

App Folder Structure:

The AngularJS functionalities are designed within the `App` folder of the web project. This is divided into following sub-folders. The structure diagram can be viewed at figure-6.6-5.

1. **ClassLibrary:** Generally clones the classes of SPA.Entities in a JavaScript format.
2. **Controllers:** Provides an intermediate between HTML pages and angular services, two-way DOM binding and handles invocation of events triggered by user via GUI elements.
3. **Services:** The folder contains services and resources that can provides interaction between controller and RESTful API by calling http $resource.
4. **Views:** Contains html pages for the front-end rendering
5. **App.js file**: Defines the angular module and inject basic references to run AngularJS. It also provides route configuration for easy page routing.

7. **SPA.SolutionTests: (Objective 4)**
   This project contains elements for testing. Please refer to the Ch 9-Testing for further details.

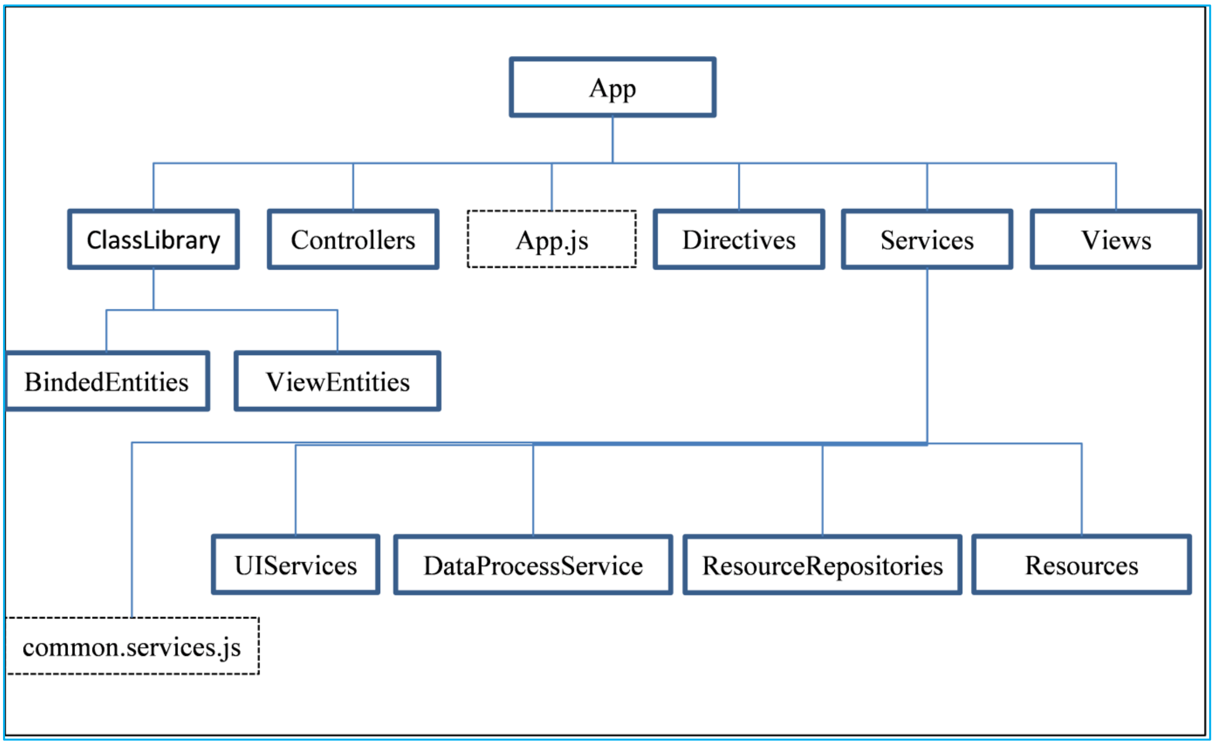


Figure 6.6-6: App folder structure

6.7. Repository Pattern (Objective 7.1)

Repository pattern and Unit of Work pattern provides an **abstraction layer** between data access layer and business logic layer and thus aids for **(TDD)** Test-Driven Development and allow application **to cope well with changes** [9].

Implementation of the Design Pattern:

Each entity is communicated via its controller and controller derives information via its repository. Repository pattern centralises the repository instances handling process. Controllers can access all repositories by only accessing `MasterRepository` class.

This makes the maintenance process **quicker** as any changes only need to be made in the `MasterRepository` and its callers remain untouched.

Each repository implements its interface to hide internal processes from the caller.

Example_(`SPA.Web.Services.Controllers`): `UsersController` accesses `IUsersRepository` from available properties of the `MasterRepository`. It (caller) cannot access `UsersRepository`'s `Dispose()` method, even though the method is public as it is not declared in the `IUsersRepository`.

6.8. Unit of Work Pattern (Objective 7.1)

I have implemented a 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, repositories call 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 this pattern is that I can control behaviour of the database executions from one method and place handlers such as catching exceptions, writing messages to log file etc. I can also combine various changes and save them all at once to limit numbers of query executions.

See appendices_9 for code sample.

6.9. Dependency Injections (Objective 7.1)

Dependency Injections is a technique to pass dependent objects of the class. I have used Ninject dependency resolver to ensure that the client object receives the dependent objects. The objects are kept as single copies per request thus eliminate the process of regenerating the copies of the same objects in the memory. This technique is called, singular object creation.

For example:-

UserController.cs

```
public UsersController(MasterRepository _masterRepo)
 {
     masterRepo = _masterRepo;
     resGen = new ResponseGenerator();

   }
```

NinjectWebCommon.cs (Definition of objects)

```
kernel.Bind<UsersController>().To<UsersController>().InRequestScope();
```

Figure 6.9-1: Code snippet: Dependency Injections

The Figure 6.9-1is an example of dependency injection sample.

The code within `NinjectWebCommon` class instructs the Ninject to reuse or instantiate any objects required within the constructor of `UsersController`. When Ninject tries to instantiate the `UsersController`, it finds that the controller requires other objects. It will try to resolve the entire dependency chain by making sure all required objects are in memory before returning the instance of the requested object.

Benefits: This practice makes unit testing easier as objects can be created once and passed within the various constructors allowing same objects to be used within various client classes and helps in the result prediction process. I have used mocking for unit testing where Ninject technique was a crucial part of the process. The more information on Unit test can be found at Ch 7- Testing: Unit Tests.

6.10. Authentication and Authorisation (Objective 7.2)

The system is required to support three different user types and thus have various access levels and rights. In order to restrict unauthorised access, I have implemented action and class level filters.

I was inspired from the article written by [10]- Mittal, A.(2015) regarding the implementation of the custom filters.

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

Authentication:

I have interpreted authentication as being an indication of whether the user is a valid user within the system or not.

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

In the snippet above, the APIAuthenticationFilter will be executed before the login method.

The ApiAuthenticationFilter has been used in two actions: Login() and Logout() within the AuthenticationController. The Authentication process flow is displayed in figure-6.10-2 and step-by-step explanation can be found in appendices_11.The filter structure is elaborated in figure-6.10-1.

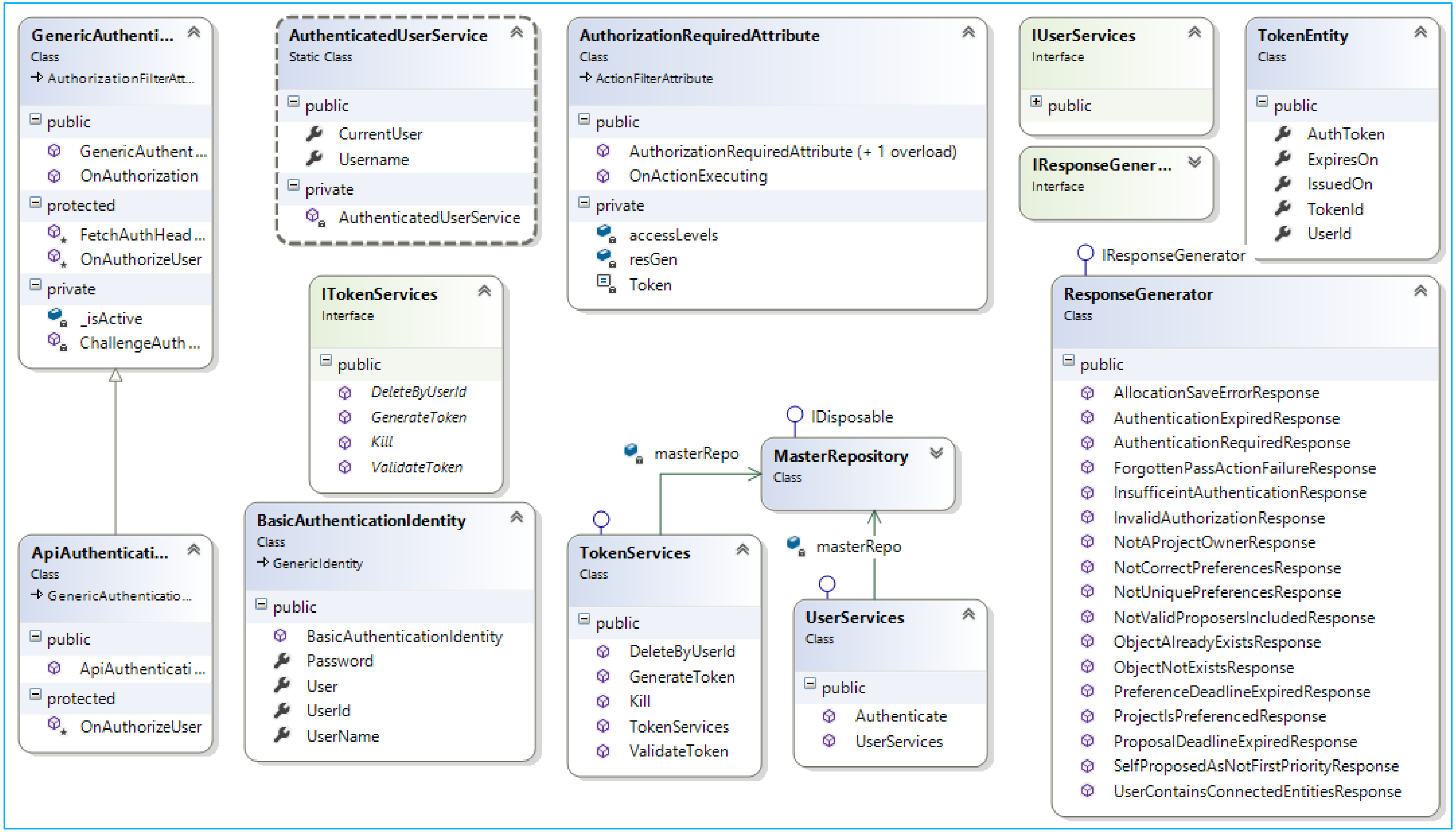
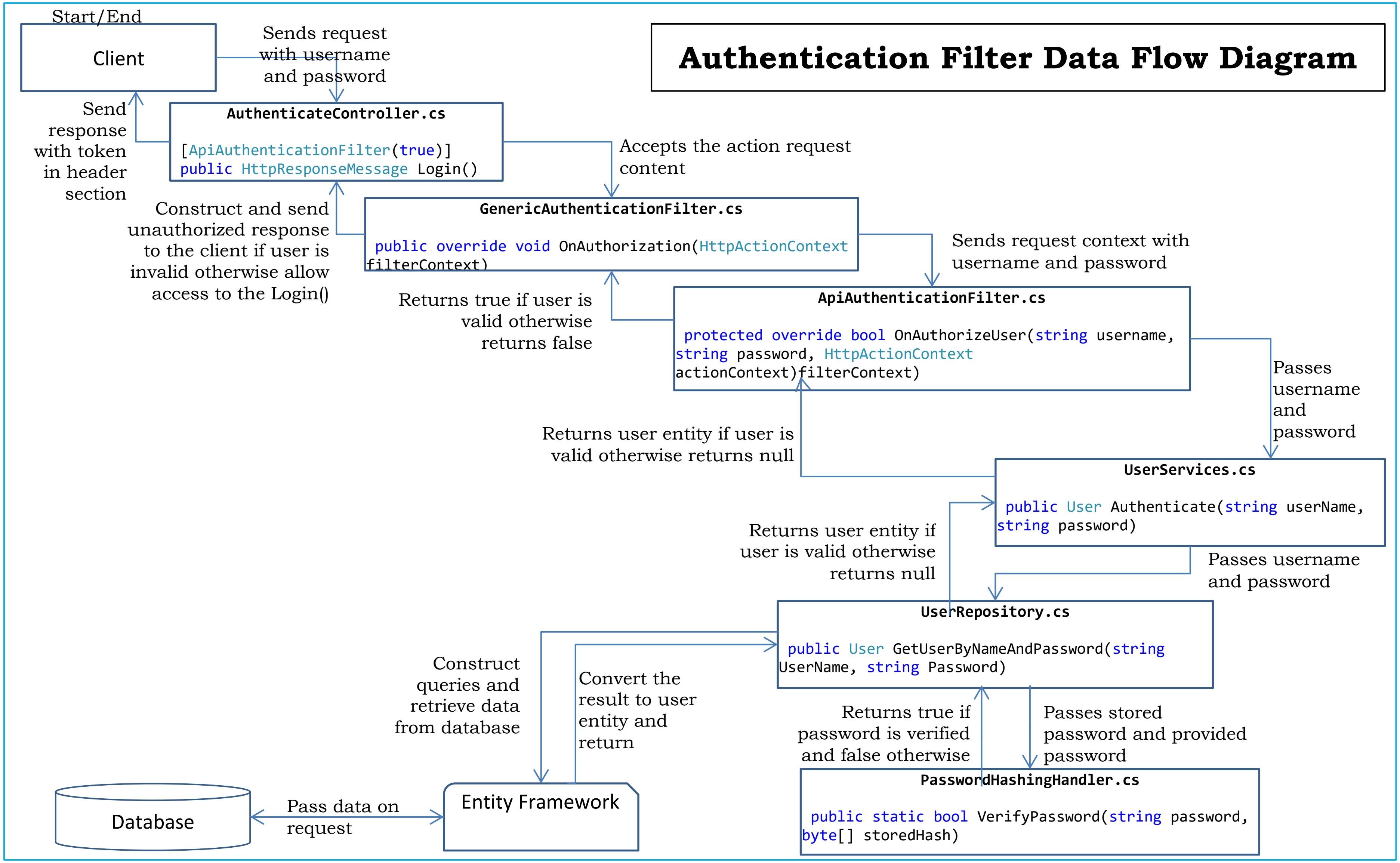


Figure 6.10-1: Filters Class Diagram (see appendices 12)



Authentication Filter Data Flow Diagram

Sends request with username and password

```
AuthenticateController.cs

[ApiAuthenticationFilter(true)]
public HttpResponseMessage Login()
```

Send response with token in header section

Accepts the action request content

```
GenericAuthenticationFilter.cs

public override void OnAuthorization(HttpActionContext
filterContext)
```

Construct and send unauthorized response to the client if user is invalid otherwise allow access to the Login()

Sends request context with username and password

```
ApiAuthenticationFilter.cs

protected override bool OnAuthorizeUser(string username,
string password, HttpActionContext
actionContext)filterContext)
```

Returns true if user is valid otherwise returns false

Passes username and password

```
UserServices.cs

public User Authenticate(string userName,
string password)
```

Returns user entity if user is valid otherwise returns null

Passes username and password

```
UserRepository.cs

public User GetUserByNameAndPassword(string
UserName, string Password)
```

Returns user entity if user is valid otherwise returns null

Construct queries and retrieve data from database

Convert the result to user entity and return

Returns true if password is verified and false otherwise

Passes stored password and provided password

```
PasswordHashingHandler.cs

public static bool VerifyPassword(string password,
byte[] storedHash)
```

Pass data on request

Figure 6.10-2: Authentication process flow

Authorization:

Authorization has been used as a level of access.

The `AuthorizationRequiredAttribute` 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.

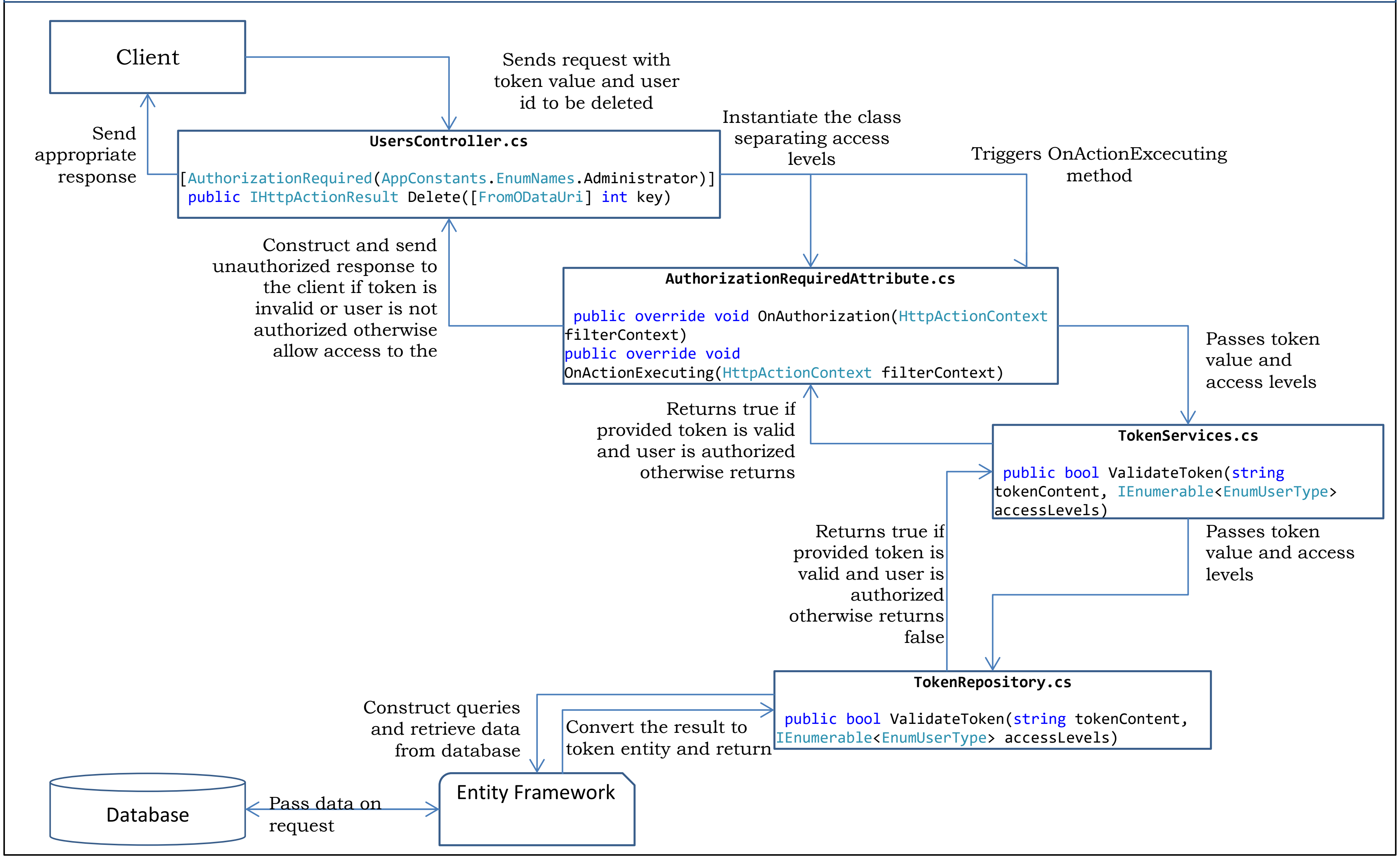
For example: Within the UsersController class, the action Delete is decorated with `[AuthorizationRequired(AppConstants.EnumNames.Administrator)]`

This means the current user has to be logged in as an administrator, to perform the action. This filter makes sure that the user is already logged in by `ApiAuthenticationFilter` because the authorization process only takes place if authentication has been successful and is an administrator.

Let's say if the user has logged out of the system and then used the previous URL to delete a user, then the server will return a 401-unauthorised response. This practice saves the use of resources for unauthorised request.

The work flow of the attribute is shown in the figure-6.10-3 and step-by-step information can be found at appendices_11.

Authorization Required Attribute Data Flow Diagram



```
[AuthorizationRequired(AppConstants.EnumNames.Administrator)]
 public IHttpActionResult Delete([FromODataUri] int key)
```

```
 public override void OnAuthorization(HttpActionContext
filterContext)
public override void
OnActionExecuting(HttpActionContext filterContext)
```

```
 public bool ValidateToken(string
tokenContent, IEnumerable<EnumUserType>
accessLevels)
```

```
 public bool ValidateToken(string tokenContent,
IEnumerable<EnumUserType> accessLevels)
```

Sends request with token value and user id to be deleted

Send appropriate response

Instantiate the class separating access levels

Triggers OnActionExcecuting method

Construct and send unauthorized response to the client if token is invalid or user is not authorized otherwise allow access to the

Passes token value and access levels

Returns true if provided token is valid and user is authorized otherwise returns

Returns true if provided token is valid and user is authorized otherwise returns false

Passes token value and access levels

Construct queries and retrieve data from database

Convert the result to token entity and return

Pass data on request

Figure 6.10-3: Authorization process flow

6.11. Password Security (Objective 7.3)

Security of users' vulnerable data is an important requirement of the system. In order to secure users' passwords, I have used password hashing/salting method. The main purpose of the practice is to convert users' original requested password to hash and salted binary data form and discard the original password. This technique decreases the chances of password leak. I have followed instructions and explanation provided at [12], [13], [14] and [15].

How does the Password Hashing Process work?

`PasswordHandler` has two main functionalities: `CreateHash` and `VerifyPassword`.
The **`createHash()`** function accepts the password string requested by the user and append a random salt with 24 bytes. The bytes are then encrypted using PBDF2 functionality that uses SHA "Secure Hashing Algorithm" process.

Salt is a randomly generated bytes, appended to the password for an added layer of security.

Why Salting?

- When random salt bytes are added to the password string, it increases password's length thus making it more difficult to decode.
- Hashing the password alone would be very insecure, if attackers have managed to reverse the algorithm, they can then easily retrieve other passwords. However, if it is salted, they still have to work out the real password.(contributing towards making the attack infeasible)
- If two users have the same passwords then hash string for both passwords would be identical, thus attacker can break one password to gain two accesses. If both passwords are salted with random salt, then hash strings will be different thus providing more security.

Once hashed and salted, the password is appended with algorithm properties such as algorithm name, iterations, salt and hashed bytes before converted into the ASCII bytes and stored away in the database.
The **benefit** of storing the password with algorithm properties is so that different properties can be assigned to different passwords making it even more secure.

`verifyPassword()`: This functionality matches the provided password with the stored password. In order to verify it, it separates the algorithm properties from the stored hash bytes and uses those properties to hash the provided password. Once hashed, both derived and stored passwords are compared byte-by-byte and returns true if hash bytes are completely identical.

The code snippet is available in appendices 10.

The inspiration and basic understanding are constructed from online articles and examples referenced in [12], [13], [14] and [15].

6.12. SMTP Client Service (Objective 5)

System is required to send automated emails on specific occasions to the relevant users. I have used available open source SMTP hosting service from google mail provider.

I setup and configured an account with google (spatooluser@gmail.com) such that the SMTP client from the home server can instruct google host (smtp.gmail.com) to send emails through the created account. I have implemented C# SMTP client service to construct mail objects and pass them to the SMTP host service.

The automated emails are implemented on occasions described in figure-4.1-1.

7. Website Implementation (GUI)

The application is running as a web application from home server with the URL: https://anitalad.ddns.net/spaSystem.Web/

7.1. Theme:

Colour combination:

Effect	colour	sample
Hover/Blur/ Static Background	#350404 rgb(53, 4, 4)	🟫
Hover/Blur/ Static Background	#690404 rgb(105, 4, 4)	🟥
Container background	#332c2c rgba(51,44,44,.95)	⬛

Font colour	#d8cece
	rgb(216, 206, 206)
		⬜

Figure 7.1-1: Application Theme

I have chosen the above colour theme for two reasons:

1. According to the OLED display screen energy consumption chart M. Linares-Vásquez, et al. (2015, cited in lecture notes at L.L. Minku. 2016) [20], the red colour components require less energy than green and blue. Moreover, the lecture notes [20] also describe the resulted GUI screen suggestions after optimisation. The suggestions clearly imply that dark colours are much more energy efficient than bright ones. This is particularly helpful when user decides to use the application on OLED screens.

2. Combined colours, blend in well with university's theme and logo.

7.2. Compatibility:

The application is accessible through any web enabled devices including but not limited to desktops, tablets and mobile devices. The application has also been regularly tested on leading web browsers such as Chrome, Firefox and IE.

The web application has been equipped with responsive feature such that it adjusts the layout of the application according to the size of the screen without having to compromise the usability.

7.3. Introduction to User Interface:

The homepage of the application contains login section which leads users to the corresponding authorised sections.

The website provides info tags on buttons to provide information on the functionalities. It also validates user inputs and provides suggestions in the case of errors/exceptions.

7.3.1. Home Page:

Home page contains a login section that allows users to request a password reset link. It also offers signup page for users to activate their account. The home page screenshot is available at figure-7.3-1.
(Requirements coverage: Ch-4.1.4-1,3,4)

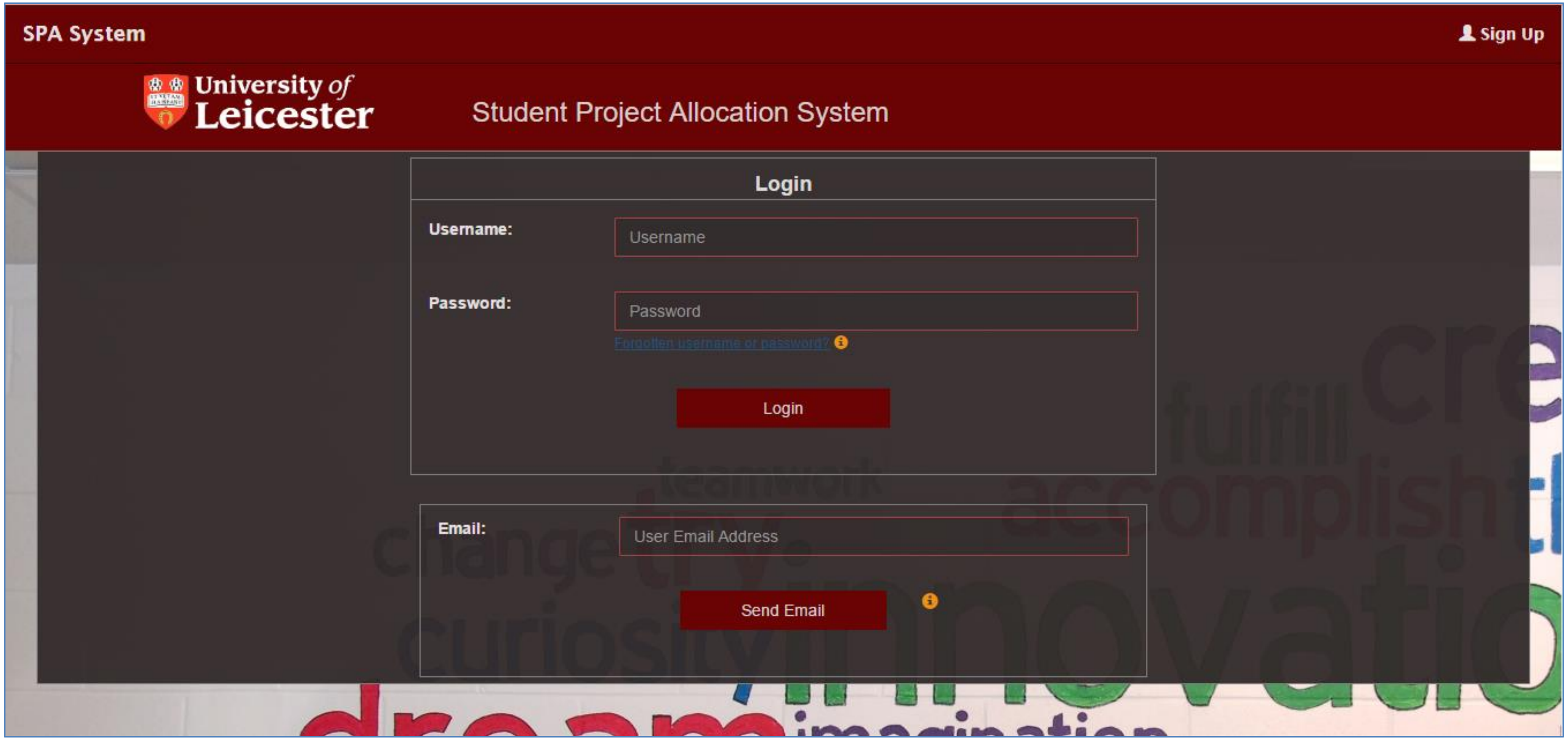


Figure 7.3-1: Home/Login Page (appendices 20)

7.3.2. Admin Home Page:

On Successful login, administrators are directed to the admin homepage (illustrated figure-7.3-2), where they can access tasks such as, Deadline Management, Project Allocation, User Management, Account Settings and Logout from current/all devices. Each of the functionalities is implemented according to the requirements specified at Ch-4.1.1 and 4.1.4.

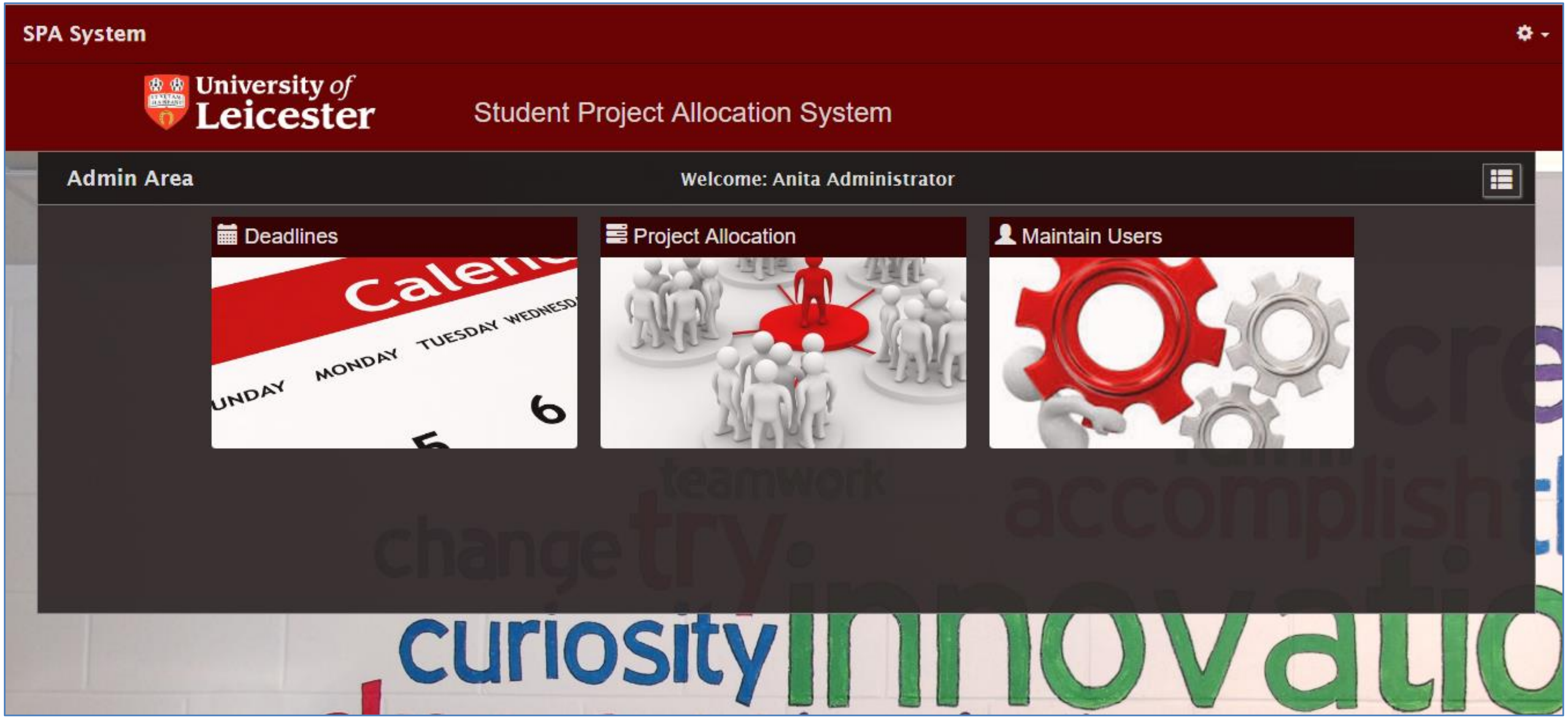


Figure 7.3-2: Admin Homepage (appendices 21)

7.3.3. Allocation Section:

One of the main functionalities of the system is project allocation which carried out by administrators. The allocation section is divided into four sections:

1. *Advance Algorithm parameters*
   This section allows users to set various algorithm parameters such as, Number of generation, Mutation rate, Population size, Stopping criteria and Fitness approach for proposer project distribution.
   A screenshot is available in: appendices_22.

2. *Derived Suggestions*
   When an administrator requests for suggestions, the system will provide suggestions with various trades-offs and display a chart along with a data table. Colour scheme for non-dominated fronts – 1: green, 2: blue, 3: pink and others: white.

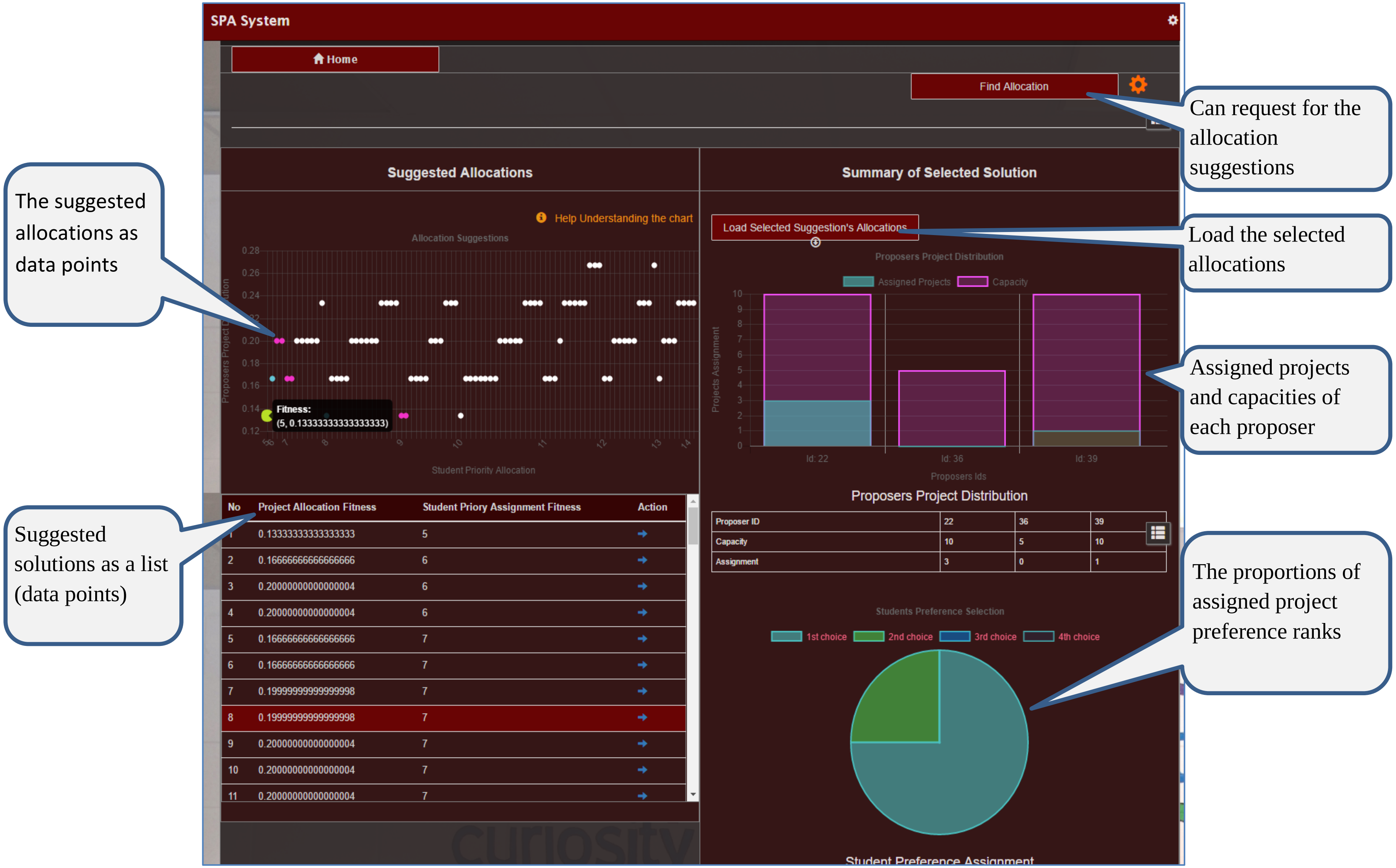
3. *Selected Suggestion/ Solution*
   A user can either click on the data point on the chart or list item from the data-table, to view more details on the selected suggestion, this will update on the right. A column chart will display projects capacity and assignment for each proposer and a pie chart will display proportions of assigned priorities. This visual presentation with data tables helps users to understand each trade-off in detail.

4. *Project Allocation Management*
   Once the administrator is satisfied with a suggestion, they can load the allocation by clicking "Load Allocation" button. The Project allocations section will open where allocations will be divided into three sections:

   1. Suggested allocations: Students, who have no allocation is assigned and all four preferences are submitted, are suggested by algorithm.

   2. Completed Allocations: Students, who have been already allocated projects due to self-proposals, will be displayed but not to be amended.

   3. Pending Allocations: Algorithm does not provide suggestions for students who have not been allocated a project and not completed all preferences. In this situation, system will provide a field where administrator can manually provide project id for the allocation.
   The figure-7.3-3 displays the project allocation section.



No	Project Allocation Fitness	Student Priory Assignment Fitness	Action
1	0.13333333333333333	5	→
2	0.16666666666666666	6	→
3	0.20000000000000004	6	→
4	0.20000000000000004	6	→
5	0.16666666666666666	7	→
6	0.16666666666666666	7	→
7	0.19999999999999998	7	→
8	0.19999999999999998	7	→
9	0.20000000000000004	7	→
10	0.20000000000000004	7	→
11	0.20000000000000004	7	→

Proposer ID	22	36	39
Capacity	10	5	10
Assignment	3	0	1

The suggested allocations as data points

Suggested solutions as a list (data points)

Can request for the allocation suggestions

Load the selected allocations

Assigned projects and capacities of each proposer

The proportions of assigned project preference ranks

Figure 7.3-3: Allocation Suggestions (appendices 23)

7.3.4. Proposer Home Page:

On Successful login, proposers are directed to the Proposer homepage (illustrated in figure-7.3-4) where they can add/edit/delete projects, view their project allocations and search/view/load projects. (Requirements coverage according to: Ch-4.1.2 and 4.1.4.)

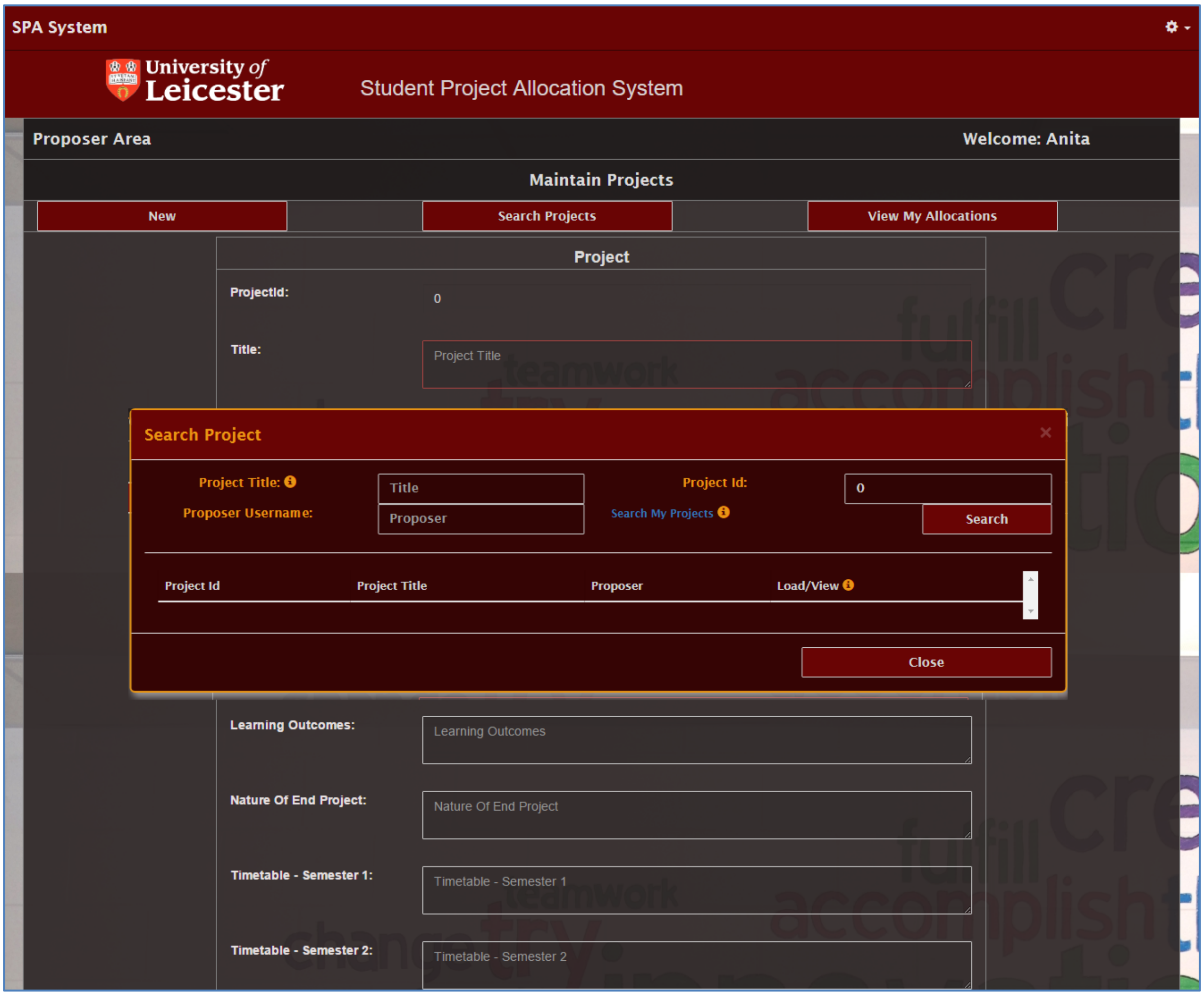


Figure 7.3-4: Proposer Homepage (appendices 24)

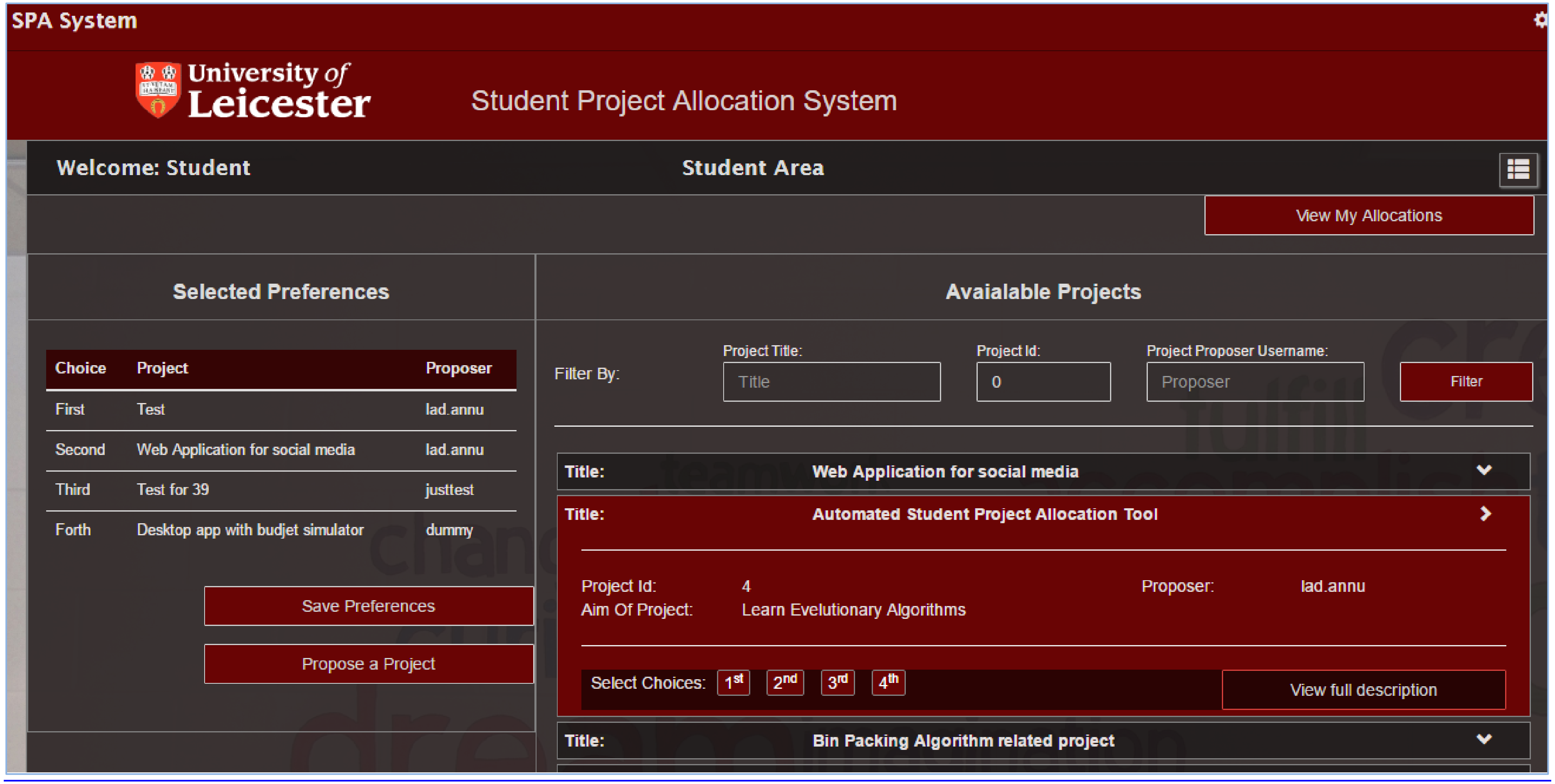
7.3.5. Student Home Page:

On successful login, Students are directed to the Student homepage (illustrated in figure-7.3-5) where they can manage preferences from the available projects.

Students can propose their own projects. On proposal, a notification email will be sent to the chosen lecturer for approval. The lecturer can then reject or accept the request. Acceptance will result into final allocation and rejection will result into removal of the project along with student's preference. On any of occasions above, a notification email will be sent to both lecturer and student.

Students can also view projects' descriptions in separate windows.

(Requirements are implemented according to: Ch-4.1.3 and 4.1.4.)



SPA System

University of Leicester

Student Project Allocation System

Welcome: Student

Student Area

View My Allocations

Selected Preferences

Choice	Project	Proposer
First	Test	lad.annu
Second	Web Application for social media	lad.annu
Third	Test for 39	justtest
Forth	Desktop app with budjet simulator	dummy

Save Preferences

Propose a Project

Avaialable Projects

Filter By: Project Title: Title Project Id: 0 Project Proposer Username: Proposer Filter

Title: Web Application for social media

Title: Automated Student Project Allocation Tool

Project Id: 4
Aim Of Project: Learn Evelutionary Algorithms
Proposer: lad.annu

Select Choices: 1st 2nd 3rd 4th

View full description

Title: Bin Packing Algorithm related project

Figure 7.3-5: Student Homepage

8. Evolutionary Algorithm Implementation (Objective 6)

One of the main aims of the project is to suggest a good allocation of students to projects considering the available projects and the students' chosen preferences.

Sub-section-8.1 briefly describes the problem and sub-section-8.2 formulates the problem components. Sub-section-8.3 provides reasoning for choosing the NSGA-II approach and 8.4 describes the implementation of the algorithm.

8.1. Description of the problem:

Each of the N students has four distinct project preferences according to their priorities (Pri_i) between 1 and 4.
Each proposer proposes 0-to-many projects.
A project can be assigned to 0-to-many students.
Student cannot select more than two projects from same proposer.
Each proposer has their maximum supervision capacity $P_{MaxSupCapacity}$.

The optimisation algorithm should take in account two objectives:

1. Allocate projects of higher priorities
2. Distribute allocation of the projects evenly throughout the proposers in accordance with their capacity

8.2. Problem formulation:

8.2.1. Design variable:

Integer representational: vector v ∈ $\{1, 2, 3, 4\}^1$, where each number represents the rank of a project priority. The vector is of size N providing one position for each student.

8.2.2. Objective Functions:

8.2.2.1. **Student Preference Allocation Fitness:**

The student preference allocation is a sum of total preferences. The smaller sum (fitness) indicates that students have been assigned higher priorities. The formulation is shown in figure-8.2-1.

Student Preference Allocation Fitness

$$(to\ be\ minimised) = \sum_{i=1}^{N} Pri_i$$

- Where N is population size, Pri_i is the priority of the project.

Figure 8.2-1: Student_Preference_Allocation_Fitness

8.2.2.2. Proposer Project Distribution Fitness:

There are two approaches available for the objective functions: Entropy and Standard Deviation. Both objectives are set to be minimised.

Proposers' project distribution requires an in-depth understanding. An algorithm is required to provide reasonable distribution of the projects amongst proposers according to their supervision capacities.

Let's say we have 6 students and 2 proposers having capacities 8 and 4 respectively. The algorithm will try to assign 8->4 and 4->2. This way, proposers with higher capacity will be assigned more projects than lower capacity proposers.

When proposers have lower capacities than the requirement, the algorithms will still emphasis on the capacity. For example, we have 9 students and 2 proposers, each with capacities 4 and 2 respectively. Algorithm will aim for the assignment of 4->6 and 2->3. In a real world scenario, proposers' capacities should have been managed prior.

Entropy vs Standard Deviation

Entropy focuses on the overall distribution, if 70% of the distribution is equal then it indicates higher equality. Standard deviation focuses on the distribution of each element, so outliers can affect the fitness significantly. If 70% of the distribution is even and if one number has a very large/small assignment then the fitness will be a lot worse than entropy.

There is a significate correlation between both entropy and standard deviation; however the degree of change is very different.

The usage:

Entropy can be used when the goal is set for majority of equal distribution and minority of inequality is acceptable.

Standard Deviation can be used when distribution for each proposer is taken in account and outliers are considered bad because each proposer has equal importance thus ignorance of minority is unacceptable.

Formulation-Entropy: Shown in figure-8.2-2

Entropy: Proposer Project Distribution Fitness

EntropyInverse:

$$(to\ be\ minimised) = 1 - EntropySol$$

EntropySol:

$$= \min(1, SumOfEntropy)$$

SumOfEntropy :

$$= \sum_{i=1}^{N} Nor(p_i) * \left(\log_N \frac{1}{Nor(p_i)}\right)$$

-N is total proposers and P_i is a proposer if $P_{MaxSupCapacity} > 0$

Nor(P):

$$= \left(\frac{\frac{P_{AssignedProjects}}{P_{MaxSupCapacity}}}{\sum_{i=1}^{N} \frac{P_{i\_AssignedProjects}}{P_{i\_MaxSupCapacity}}} + 0.00000001 \right)$$

-$P_{AssignedProjects}$ is a number of assigned projects to the proposer P

-N is total proposers and P_i is a proposer if $P_{MaxSupCapacity} > 0$

Figure 8.2-2:Entropy_Project_Distribution_Fitness

Formulation - Standard Deviation:

Standard deviation: Proposer Project Distribution Fitness

StdDev: $(to\ be\ minimised) = sqrt(Variance)$

Variance:

$$= \frac{\sum_{i=1}^{N}\left(\left(\frac{P_{i\_AssignedProjects}}{P_{i\_MaxSupCapacity}}\right) - \left(\sum_{i=1}^{N}\left(\frac{P_{i\_AssignedProjects}}{P_{i\_MaxSupCapacity}}\right)\Big/_{N}\right)\right)^{2}}{N}$$

- N is total proposers if Proposer $P_{i\_MaxSupCapacity} > 0$
-
- $P_{i\_AssignedProjects}$ are projects assigned to the proposer P_i and $P_{i\_MaxSupCapacity}$ is the maximum number of project proposer can supervise

Figure 8.2-3:Standard Deviation Project Distribtuon Fitness

8.2.3. Constraints:

There are two main constraints:

1) Students' can only get assigned the project from selected preferences- (Dealt within representation)

2) A proposer should not be assigned more projects than their maximum capacities- (Dealt with algorithm fitness calculation)

8.3. Selection of Algorithm approach:

According to the evaluation carried out in the Background chapter, I decided to adopt NSGA-II approach. The detailed process of NSGA-II is described in Explanation of Algorithm section. The algorithm pseudocode is explained in Figure-8.3-1.

```
    Algorithm
    t              = Iteration
    $P_t$          = Population of iteration t
    N              = Population Size
    $t_{Max}$      = Maximum iterations (Generation)
    C              = A Set of Offspring
    $F_1$ .. $F_N$ = Non-dominated Fronts
    S              = A union of Parent and Offspring of an iteration
    $R_c$          = Repeated fitness counter
    $R_{max}$      = Maximum repeated fitness runs
    $t_{fit}$      = Fitness of each individual within the t

Process:
1) Set t = 0 (current generation)

2) Initialise population $P_t$ with size N
   -  initializePopulation()

3) Sort $P_t$ into different non-dominated fronts
   -  fastNonDominatedFrontsSorting()

4) Determine crowding distance of each individual in $P_t$
   -  crowdingDistanceAssginment()

5) While t < $t_{max}$ && $R_c$ <= $R_{max}$

   1.Select parents from $P_t$ using 2 - tournament selection based
     on non-dominated fronts and crowding distance
   - tournamentParentSelection()

   2.Apply crossover to individuals in C- Uniform Crossover
   - applyCrossover()

   3.Apply mutation to each individual C – Creep Mutation
   - applyMutation()

   4.Combine Parent and offspring S <- $P_t$ ∪ C

   5.Sort S to different non-dominated front according to their
     dominance
   - fastNonDominatedFrontsSorting()

   6.Determine crowding distance for each individual in S
   - crowdingDistanceAssginment()

   7.Select Survivals from S based on non-dominated fronts and
     crowding distance(Elitist Survival Selection)
     survivalSelection()

   8.If($t_{fit}$ == t-$1_{fit}$)then $R_c$ <- $R_c$+1 else $R_c$ <- 0
     compareSolFitnesses()
   9.Increase the iteration counter t <- t+1
```

Figure 8.3-1:NAGA-II Algorithm

8.4. NSGA-II Implementation:

8.4.1. Initialization of Population:

Initialisation process involves initiating the first population.

The process first eliminates already allocated students from the list. After that, it removes the students with incomplete preference submissions to avoid system errors. It also calculates both objectives' fitness values for each solution.

The code sample can be viewed at appendices 14.

8.4.2. Fast Non-dominated Front Sorting:

The algorithm identifies dominated front for each individual solution within the population.

First the algorithm goes through each solution-(p) and finds any solutions that are dominated by p-(q) and also a number of solutions that dominates the p-(np). The solutions that have np=0 means they are at the pareto front 1, then it goes through each solutions that are dominated by the pareto front's solutions (q) and decreases their domination count (np) by one. It will sort all the solutions until no solutions are left with rank > 0.

Figure-8.4-1 explains the process in pseudocode. Code sample is available at appendices 15.

Algorithm

```
    Pop             = Population
    p, q            = Individual solution within population
    $S_p$              = A set of solutions dominated by p
    $N_p$, $Q_p$          = A number of solutions dominates p
    $P_{rank}$, $Q_{rank}$    = A rank/front number
    $F_1$ .. $F_N$        = A set of Non-dominated Fronts
    $F_i$, Q           = A set of solutions within each front

Process:
For each p ∈ Pop {
    $S_p$ <- $\phi$
    $N_p$ <- 0
    For each q ∈ Pop{
        If(p < q) then $S_p$ <- $S_p$ ∪ {q}
        Else if (q < p) then $N_p$ <- $N_p$ + 1
    }
    If $N_p$ == 0 then $P_{rank}$ <- 1
    $F_1$ <- $F_1$ ∪ {p}
}

i <- 1
While($F_i$ ≠ $\phi$){
  Q <- $\phi$
    For each p ∈ $F_i$ {
      For each q ∈ $S_p${
        $N_q$ <- $N_q$ - 1
        If($N_q$ == 0) then $Q_{rank}$ <- $Q_{rank}$ + 1; Q <- Q ∪ {q}
      }
    }
    i <- i + 1; $F_i$ <- Q
}
```

Figure 8.4-1: Fast non-dominated front sorting

8.4.3. Crowding Distance

As mentioned in the Background chapter briefly, crowding distance is a technique which ensures diversity in the solution by determining the distance of neighbouring solutions.

The crowding distance is used on two occasions:

1) Tournament parent selection, when two solutions are from the same front,

2) Survivor selection.

In both cases we surely place selection pressure towards highest crowding distance solutions for the improvement in solutions' fitness.

Original approach- (Cuboid Measure)

The approach is proposed by K. Deb (2002) [17]. The calculation (pseudocode is explained in 8.4-2) requires sorting for each objective and it can be at most N if entire population contains the same front then time taken can be **O(MN log N)** where (N logN) is a complexity to sort N and M is a number of objectives. Code sample is available at appendices 16.

Algorithm

```
    |I|, N      = number of solutions in a Pareto front I
    M           = A set of Objectives
    ∞           = Infinite positive number
    $f_{max\_m}$     = maximum value of the objective m
    $f_{min\_m}$     = minimum value of the objective m
    I[L]        = Last element in I
    c           = counter

  Process:

  For each m ∈ M {
    I <- sort(I,m)
    I[1]distance <- ∞
    I[L]distance <- ∞
    $f_{max\_m} =$  I[L].m
    $f_{min\_m} =$  I[1].m

    For c = 2 .. N-1{
        I[c].distance = I[c].distance
            + ((I[c-1].m - I[c+1].m)/($f_{max\_m}$ - $f_{min\_m}$))
    }
  }
```

Figure 8.4-2: Crowding distance- Cuboid

Alternative Approach – (Harmonic Average Distance):

The alternative approach for determining crowding area is a Harmonic Average Distance (HaD). This approach overcomes a limitation of crowding distance as explained in the example below:

Sol	f1	f2
1	10	6
2	**20**	**5**
3	30	4
4	**35**	**3.5**
5	50	2
6	60	1

Figure 8.4-3: Distance Data table

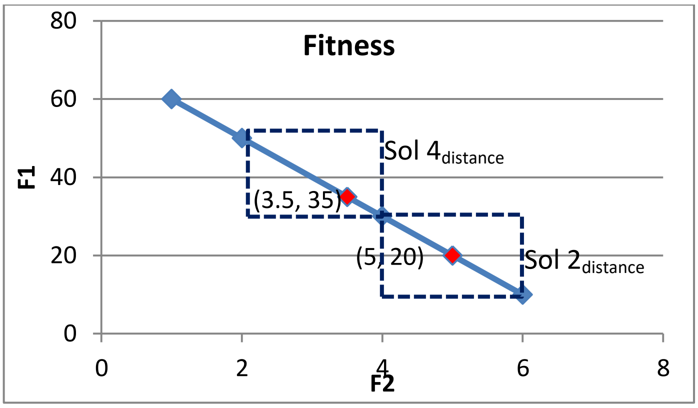


Figure 8.4-4: Crowding distance - Cuboid

The crowding distance calculated with cuboid approach will give the same crowding distance to sol2 and sol4 as they have overall same distance. However, we can clearly see that sol4 is more crowded than sol2 as it's closer to sol3. The HaD can eliminate this limitation – observed by V. L. Huang, et. al, (2005)(p-10)[20].

The above scenario, Cuboid distance: sol2=sol4=0.8. However, HaD: sol2=0.28 and sol4=0.21. Therefore, HaD has better crowding degree than Cuboid. I have implemented the harmonic distance as a crowding measure to achieve better diversity. Figure-8.4-5 illustrates the process. Code sample is available in appendices 17.

Algorithm

```
|I|, N        = number of solutions in a Pareto front I
∞             = Infinite positive number
$f_{max\_m}$        = maximum value of the objective m
$f_{min\_m}$        = minimum value of the objective m
I[L]          = Last element in I
c             = counter
Process:
I <- sort(I,m1)
I[1]distance <- ∞;    I[L]distance <- ∞
$f_{max\_m1}$ =  I[L].m1;        $f_{max\_m2}$ =  I[L].m2
$f_{min\_m1}$ =  I[1].m1;        $f_{min\_m2}$ =  I[1].m2
For c = 2 .. N-1{
   I[c].distance <- 2/((1/EDist(c,c-1))+(1/EDist(c,c+1)))
}
EDist(x,y)= sqrt((Nor(x.$m_1$)– Nor(y.$m_1$))$^2$ +
                    (Nor(x.$m_2$)– Nor(y.$m_2$ ))$^2$)
Nor(x.m) = (x.m – $f_{min\_m}$)/($f_{max\_m}$ – $f_{min\_m}$)
```

Figure 8.4-5: Crowding distance - Harmonic

In addition to the better diversity, harmonic distance running time is **O(N log N)** which better that the cuboid distance.

8.4.4. Tournament Selection

Tournament selection is a technique to select parent solutions to produce initial population. I have used 2-tournament selection process. The algorithm randomly selects two individuals and takes the solution which has lower non-dominated front or highest crowding distance. If both solutions are equal in fitness then it will repeat the selection process until the condition is satisfied.

Algorithm

```
P              = A set of selected solutions
I              = A set of available solutions to choose from
N              = Population size
$X_{rank}$          = Non-dominated front of the solution X
$X_{distance}$      = Crowding distance of the solution x
c              = counter

Process:

P <- {}
c <- 1

while c < N do{
   c <- c+1

   i <- select random solution(I)
   j <- select random solution(I)

   if($i_{rank}$ < $j_{rank}$) then P <- P ∪ {i}
   else if($j_{rank}$ < $i_{rank}$) then P <- P ∪ {j}
   else{
      if($i_{distance}$ > $j_{distance}$) then P <- P ∪ {i}
      else if($j_{distance}$ > $i_{distance}$) then P <- P ∪ {j}
      else c <- c-1
   }
}
```

Figure 8.4-6: 2-Tournament Selection

8.4.5. Crossover

I have applied a uniform crossover approach for generating children solutions from parents. The approach assigns equal probability for gene (an allocation of student-project) to be selected from either parent-1 or parent-2. The randomly generated number indicates whether the gene will be taken from parent-1 or parent-2.

The code sample is available at appendices 18.

Algorithm: Corssover

```
C1, C2      = children to be returned after crossover
P1, P2      = Parents
N           = Total genes in an individual
For i <- 1 ..N do
   r <- select_random_number(0..99)
   if(r < 50)then
      C1.allocation <- C1.allocation ∪ {P1.allocation$_i$}
      C2.allocation <- C2.allocation ∪ {P2.allocation$_i$}
   else
      C1.allocation <- C1.allocation ∪ {P2.allocation$_i$}
      C2.allocation <- C2.allocation ∪ {P1.allocation$_i$}
```

Algorithm: Mutation

```
C           = child to mutate
Cm          = Mutated child
For i <- 1 ..N{
   r <- select_random_number(0..99)

   if(r < mutationRate)then
      C1.allocation <- C1.allocation ∪
                          {BinomialRandomAllocation}
   else
      Cm.allocation <- Cm.allocation ∪ {C.allocation}
}
```

Figure 8.4-7: Crossover and Mutation

8.4.6. Mutation

The mutation introduces unexpected changes within the gene. I have applied creep mutation to generate mutated solutions. The mutation rate can be assigned by administrator. I have used TRandom library to generate binomial random number that assigns higher priority to smaller changes than the bigger changes. For example, if current priority is 2 then mutation from 2->3 or 2->1 is higher than 2->4.

Algorithm will keep generating random binomial numbers until valid priority is achieved.

8.4.7. Elitist Survival Selection

As mentioned in the Background chapter, Elitist combines parent and children solutions based on nomination fronts and crowding distance. The code **sample** is available at appendice_20. Figure 8.4-8 displays the pseudocode.

```
Algorithm
      C      = Combination of parents and children
      P      = selected solutions
      $F_i$     = Solutions within the front i
      N      = Population size
  Process:
      Set P <- $\phi$
      i <- 1
      $F_i$ <- select $front_i$ solutions(C)
      while (|P| + |$F_i$| <= N) {
            P <- P ∪ $F_i$
            i <- i+1
      }
      Sort $F_i$ by crowding distance in descending order
      l <- N - |P|
      P <- P ∪ take {$F_i$[1], .. ,$F_i$[l]}
```

Figure 8.4-8: Elitist_Survival_Selection

8.4.8. Stopping Criteria

The NSGA-II algorithm's main loop has two stopping criteria:

1. Iteration reaches the maximum generation point
2. Student preference assignment and proposer project distribution fitness are unchanged for specified number of runs.

The second criteria ensure that the algorithm generates different fitness on each run. The process keeps track of the numbers of consecutive runs where fitness values are unchanged.

If fitness doesn't change for specified consecutive runs, the algorithm assumes that the optimal solution has been achieved thus not possible to obtain a better solution and stops.

In order to compare the fitness between runs, I have used *minimal comparison strategy*. In an ideal case, I would have to compare each solution within the population thus N (the population size) solutions for each objective M and for maximum iterations G. The at most comparison would be **O(NMG)**. In default scenario, this will be **100***1000*2=**200000** comparisons. Instead of comparing each solution, I compared solutions on uniquely spaced intervals such as 0, 10, 20, 30… 100. This reduces the overall comparison requirement to **11***1000*2=**22000** and the space complexity to (N/10+1)=**11**.

8.4.9. Constraints Handling

I restricted the representation of the project preferences to numbers from 1 to 4 to avoid invalid entries.

On the calculation of proposers' projects distribution, I have placed a check that ensures that proposers are not overloaded with projects. If so, for each overloaded project, I add a large positive constant (penalty) to the distribution fitness to mark that particular solution least acceptable for the selection. The penalty factor guides the algorithm to select non-overloaded solutions versus overloaded ones and the distribution fitness factor guides the algorithm to choose evenly distributed solutions versus poorly distributed ones.

Formulation:

Constraints:

Entropy: = EntropyFitnessVal + *OverallPenalties*

StandardDeviation: = StandardDeviationFitnessVal + *OverallPenalties*

OverallPenalties: $= \prod_{i=1}^{N} C * OverloadedProjects(P_i)$

- Where N is number of proposers
- C is a large predefined positive constant

OverloadedProjects(P) =

$$\begin{cases} P_{AssignedProjects} - P_{MaxSupCapacity} \mid if\ P_{AssignedProjects} > P_{MaxSupCapacity} \\ 0 \mid otherwise \end{cases}$$

Figure 8.4-9:Constraints

9. Testing

I have implemented a test project to test functionalities of various components. The implemented tests are mentioned in 9.1 and 9.2. Sub-section-9.3 describes the performance tests of the algorithm approaches.

9.1. Unit Tests

Unit Tests with Mock Objects

Mock objects allow defining database states and running controller actions with mock database context without executing queries to the actual database. These have been used in conjunction with unit tests.

`EntityFrameworkMockHelper.cs` provides `GetMockContext<T>()` method which accepts the dbContext and derives all tables and their properties and any defined rules, so that any SQL injections can be performed on temporary objects.

Benefits of mocks

MockHelper directory within the test project provides dbContext mocking. Mocked context can be prepared with pre-defined database entity values for testing (Assertions). An action may change the database state, however action with mocked context can only change mocked context not the real database.

If any changes are made within the system, the overall functionalities of the system can be tested by running predefined unit tests.

I have managed to cover each non-trivial part of the system with unit tests. All tests have been passed successfully and the information on running the tests is available in the SPASystem.SolutionTests folder (readme.txt).

9.2. Integration Tests

Unit tests within the project, tests the behaviour of the repositories. However, it cannot cover the validations within the controllers before calling the repository. In order to cover those areas, I have placed minimal integration tests which allow test cases to be connected with real database and a server instance, so that authentication, authorisation and validity checks can be tested.

The sample integration test is included in appendices 13.

9.3. Algorithm Performance Tests

The approach:

I carried out six tests to evaluate the performance of the algorithm by testing proposers' supervision capacity when it is “equal”, “less” and “greater” than the requirement of the number of projects to be allocated with entropy and standard deviation approaches. The chosen test parameter, the number of generations, is referred from the literatures [17], [18] and [19]. I observed test results using various generation sizes and mutation rates, and decided upon taking the values described in Figure 9.3-1. The obtained test results are available in Appendices_30. The number of students, projects and proposers are based on a real-life scenario.

Algorithm Parameters:

Number of Students:	100
Number of Proposers:	20
Number of available Projects:	100
Uniform Crossover rate:	0.5
Creep Mutation rate:	0.50
Generation:	250
Population size:	100

Overall supervision capacity is Less than the number of project assignments:
Each proposer's capacity: 1
Total: 20 (1 x 20); Required: 100

Overall supervision capacity is Equal to the number of project assignments:
Each proposer's capacity: 5
Total: 100 (5 x 20); Required: 100

Overall supervision capacity is Greater than the number of project assignments:
Each proposer's capacity: 100
Total: 2000 (100 x 20); Required: 100

Figure 9.3-1: Algorithm Parameters

Multi-objective algorithm's performance measure has two goals: 1) Convergence (the quality of the solutions) and 2) Diversity (the spread of the solutions) as described by K.Deb, et al. (2002) [17].

Diversity and Convergence performance metrics are placed as explained in [17]. In addition, I introduced some enhancements to the convergence metric-

1. The "true" pareto behaves as a "threshold" thus gives more emphasis on better solutions.
2. The fitness values for the "threshold" pareto solutions and derived non-dominated front values are normalised to eliminate the limitation of different scales.
3. A use of key-pair value data structure can facilitate element search in 1 constant time therefore improves the search operation time during convergence calculation.

The calculation process is described in Figure-9.3-3.

Calculation of Convergence:

Convergence is calculated from threshold points that are calculated upon the best possible fitness (min) and the desirable worst fitness (max). The used parameters are described in figure-9.3.2.

Test Scenario: Capacity	**X Axes:** **Student's Project Priority Assignment**		**Y Axes:** **Proposers' Project Distribution**	
Overall supervision capacity in regards to the number of projects to be assigned	Min	Max	Min (Best possible)	Max (Desirable)
Greater	100	400	0	1
Equal	100	400	0	1
Less	100	400	8000	8001

Figure 9.3-2:Convergence Pareto Calculation Parameters

Algorithm

```
T          = A set of threshold pareto
S(s,p)     = A key value pair of Student preference fitness s
             and Project Distribution p
S(ns,np)   = A solution with normalised fitness
F          = A set of solutions within the front 1
totalConv  = convergence of the derived pareto
conv       = final average convergence
Diversity  = final Diversity

 Process: Convergence
 -------------------------------------------
    For each (p in T)do
         x <- p-1.prDistFitness –
         ((maxProjDistFit - minProjDistFit)/|T|);
         p(s,p)| s <- 100..400,p <- x))

    For each (t in T)do
         t(ns,np) <- t(ns <- normalise(s),
                    np <- 1-(sqrt(1-(pow((normalise(p)),2)))
    For each (p in F) do
         p(ns,np) <-p(ns <- normalise(s)with min/max of the
                     threshold pareto,
                     np <- normalise(p)with min/max of the
                     threshold pareto)

    For each (f(ns,np) in F){
      minDist <- Euclidean distance from d to any nearest
                 point in T

      if(np < (np1 <- T(ns, np1))) then
          totalConv <- totalConv – minDist
      else
          totalConv <- totalConv + minDist
    }
    conv <- totalConv/|F|

 Process: Diversity
 -------------------------------------------
    For each (p in F){
      p(ns,np) <- p(ns <- normalise(s)with min/max of the
      derived pareto, np <- normalise(p) with min/max of the
      derived pareto)
    }
    For F <- 1,..(|F|-1) do
        Fi.ecluDist <- EuclideanDistance(Fi, Fi+1)

    Diversity <- calculate standard Deviation of all Fi ∈ F
```

Figure_9.3-3:Convergence and Diversity metrics

Results of Simulation:

I have observed the algorithm's behaviour in six different scenarios: capacities are equal, less and greater than the requirements with entropy and standard deviation as described in figure-9.3-1.

According to the obtained results displayed in figures 9.3.4 and 9.3.5, we can observe that-

When capacity scenarios are "greater" and "less", entropy seemed more robust against the difficulty of the problem instance in regards to convergence. Its convergence was similar across problem instances, whereas standard deviation struggled with more difficult problem instances such as "less" capacity scenario where convergence was very high than the "greater" scenario.

The convergence is heavily affected by the non-trivial decision factors of the problem such as a number of projects proposed by lecturers, the preference selection of students and collective and individual capacities of proposers, etc.

When capacity is "equal", the convergence values are more sensitive to the non-trivial decision factors. As a result, both standard deviation and entropy have struggled to keep a good level of pareto optimality.

Entropy and standard deviation have preserved a good diversity spread across all problem instances.

The test data table can be found in appendix_26. I also have included derived median solutions of entropy in each problem instance. Please see figures 9.3-6 to 9.3-10 for further information.

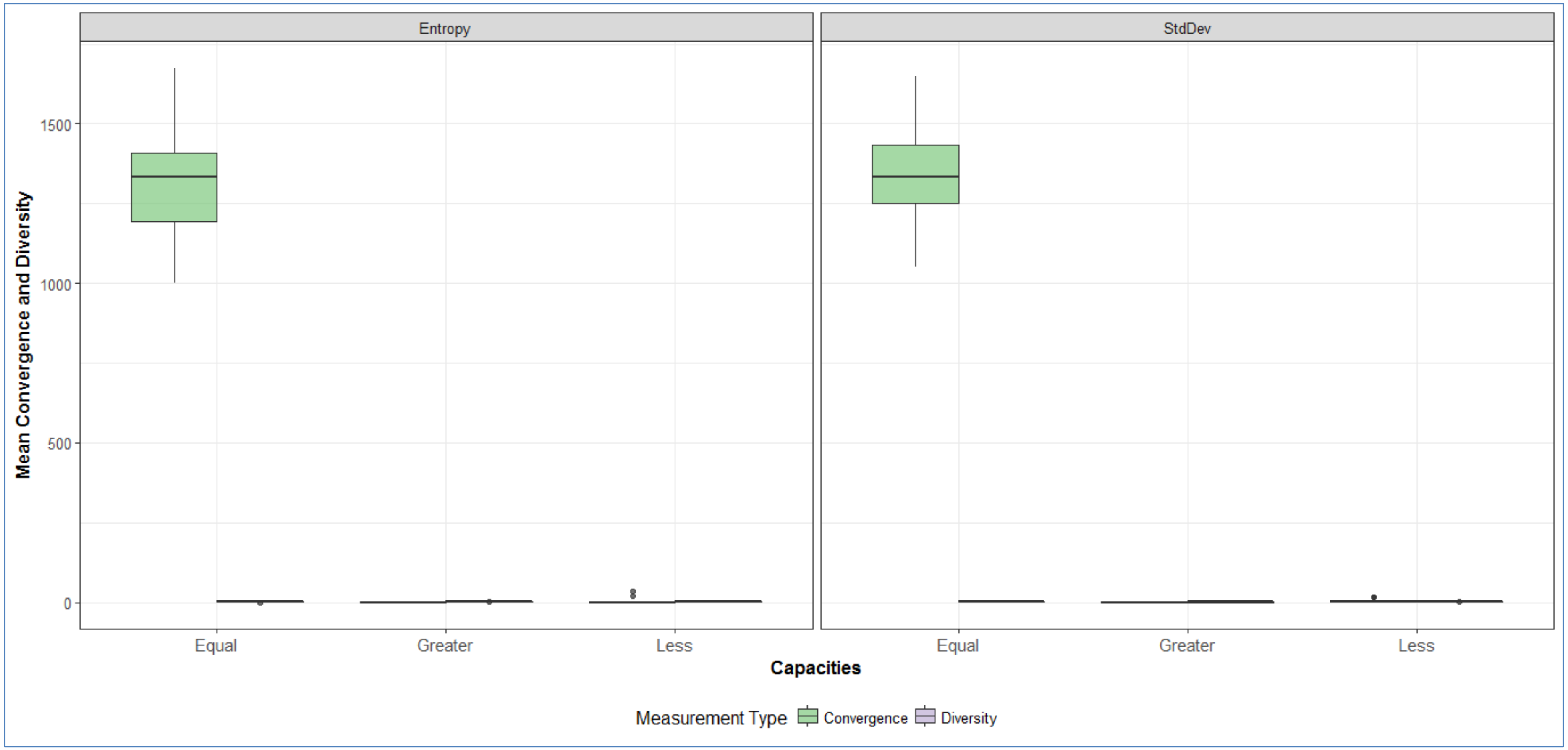


Figure 9.3-4: Test results for 30 runs on problem instances where supervision capacity is greater/equal/less than the number of projects to be allocated. The pareto fronts used to calculate convergence were different (Parameters are explained at figure 9.3.2), so that the scale of the performance measures is similar for the different problem instances (See appendices 28)- (Full Data)

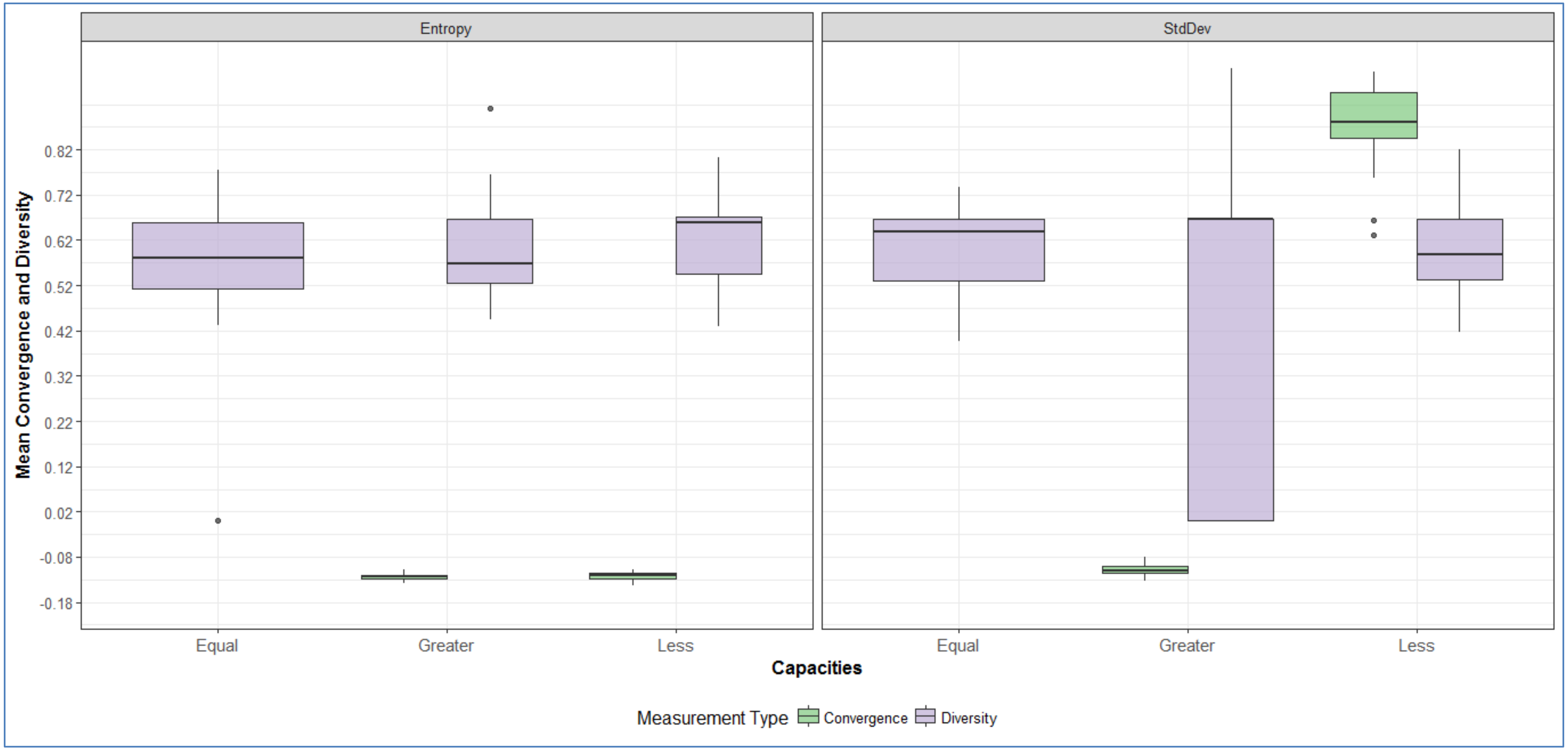


Figure 9.3-5: Test results for 30 runs on problem instances where supervision capacity is greater/equal/less than the number of projects to be allocated. The pareto fronts used to calculate convergence were different (Parameters are explained at figure 9.3.2), so that the scale of the performance measures is similar for the different problem instances. (Zoomed in parts for better evaluation)- (See appendices 27)

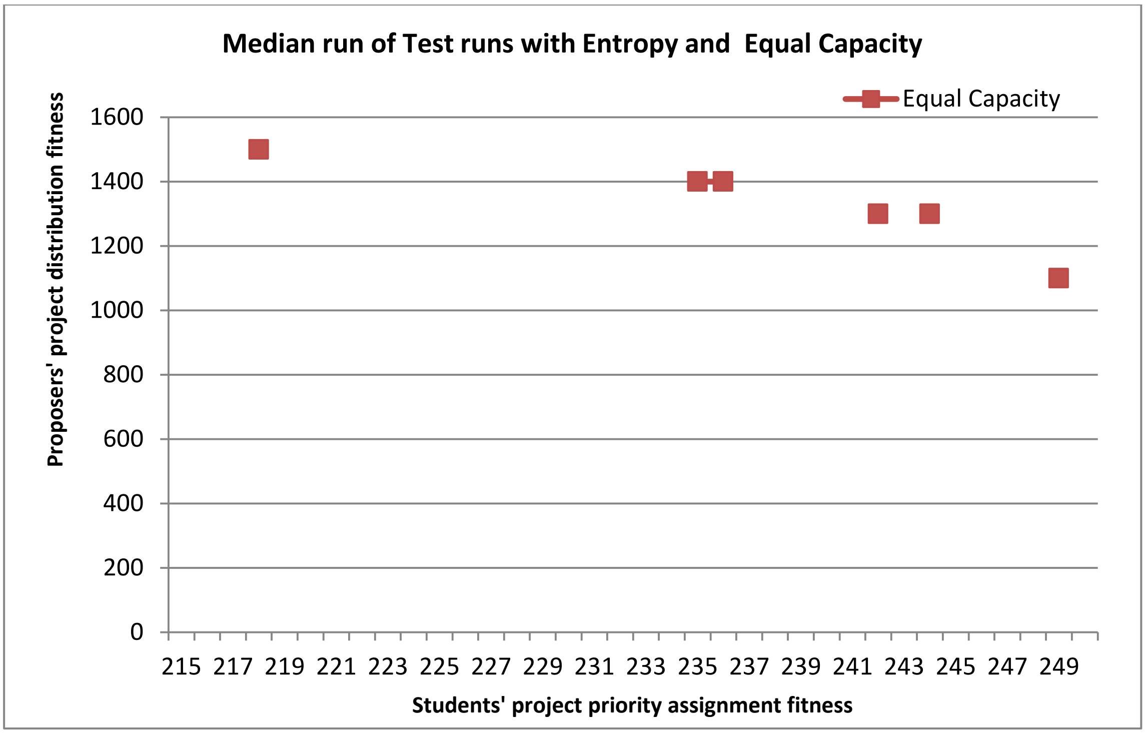


Figure 9.3-6: Solutions within pareto front 1: Entropy: Equal

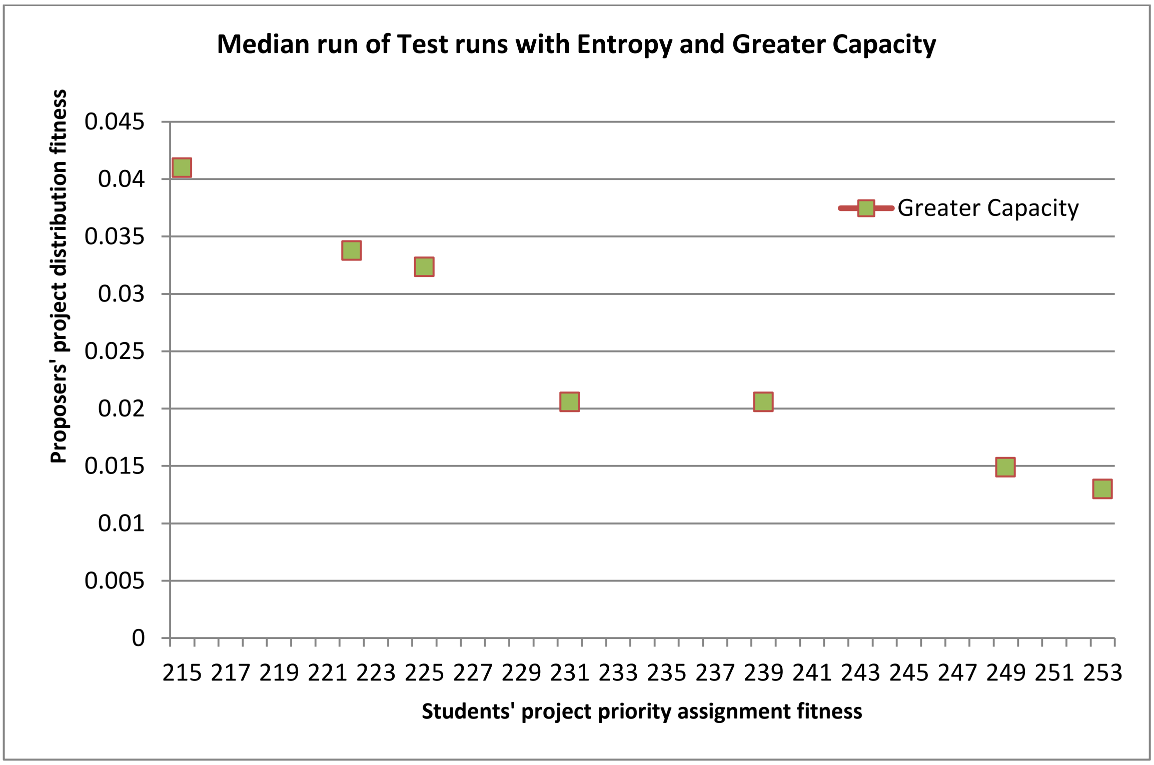


Figure_9.3-7: Solutions of pareto front 1: Entropy: Greater

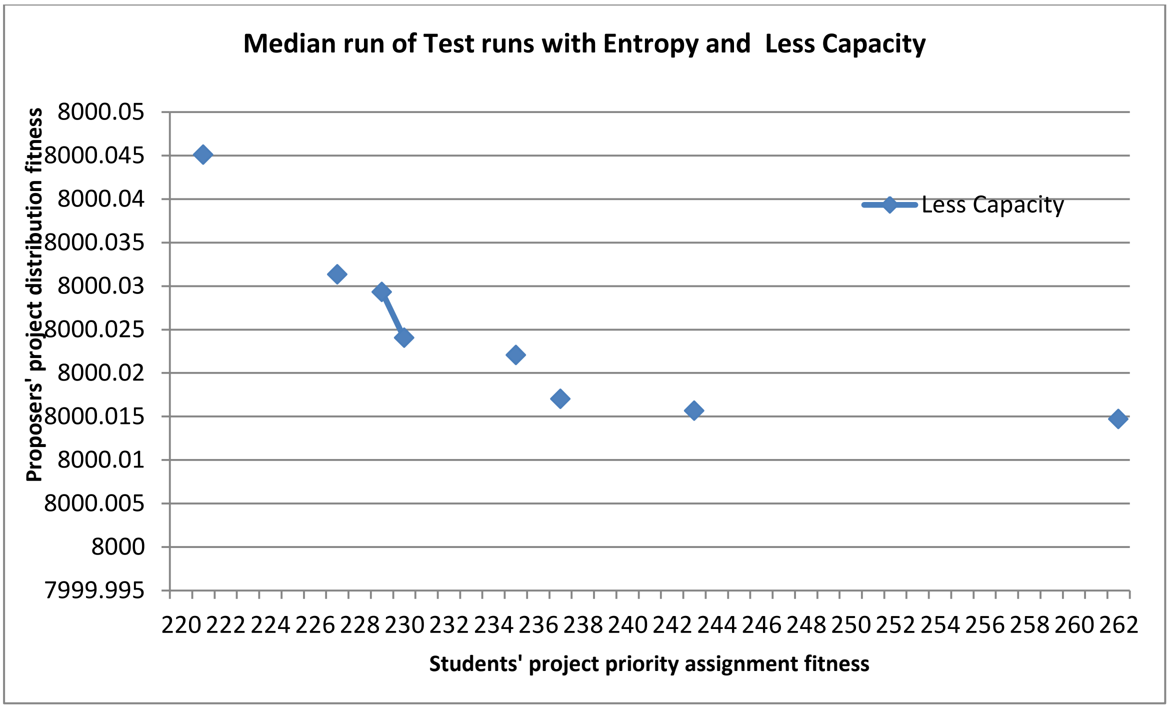


Figure 9.3-8: Solutions of pareto front 1: Entropy: Less

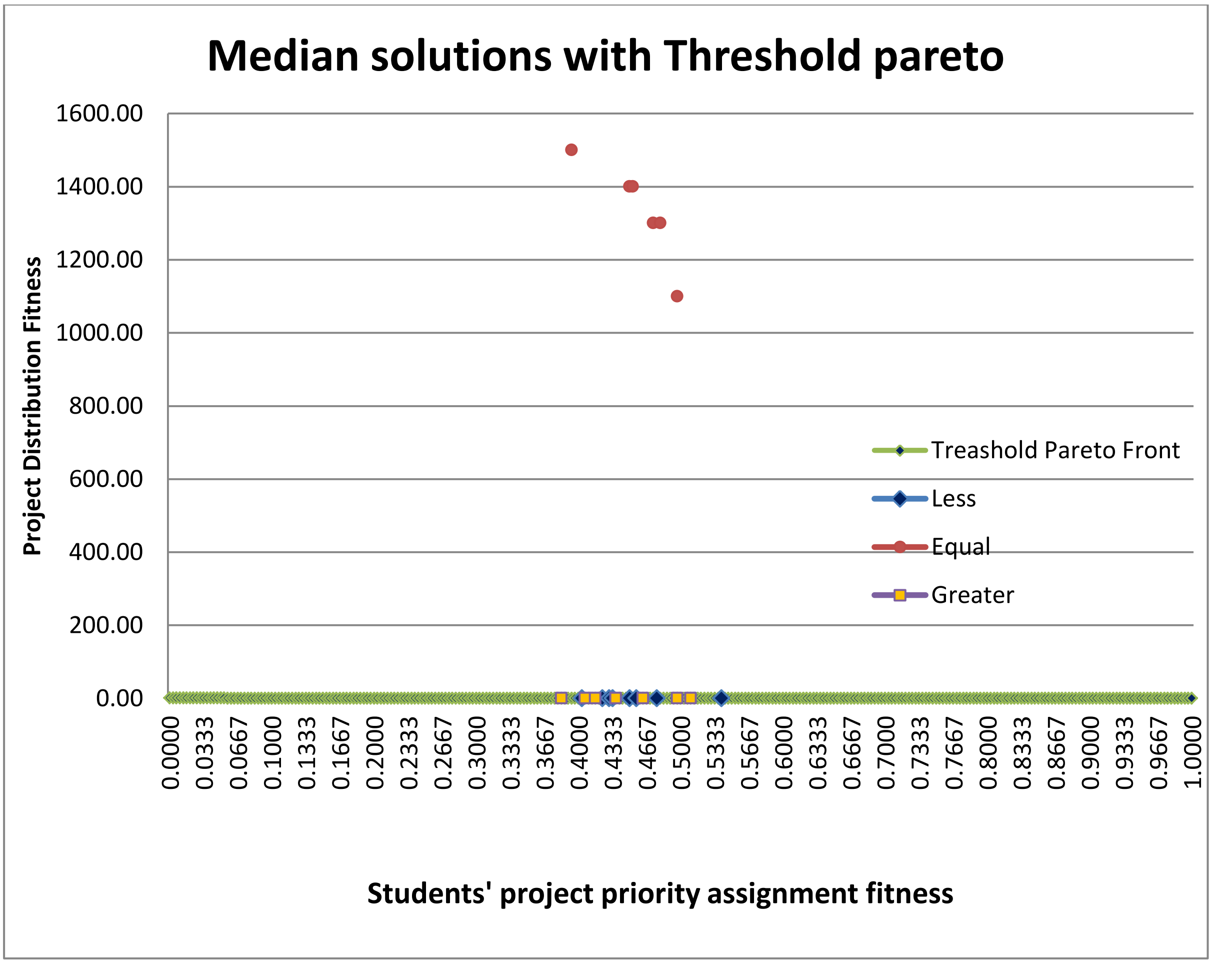


Figure 9.3-9: Solutions of pareto front 1: with Threshold Pareto

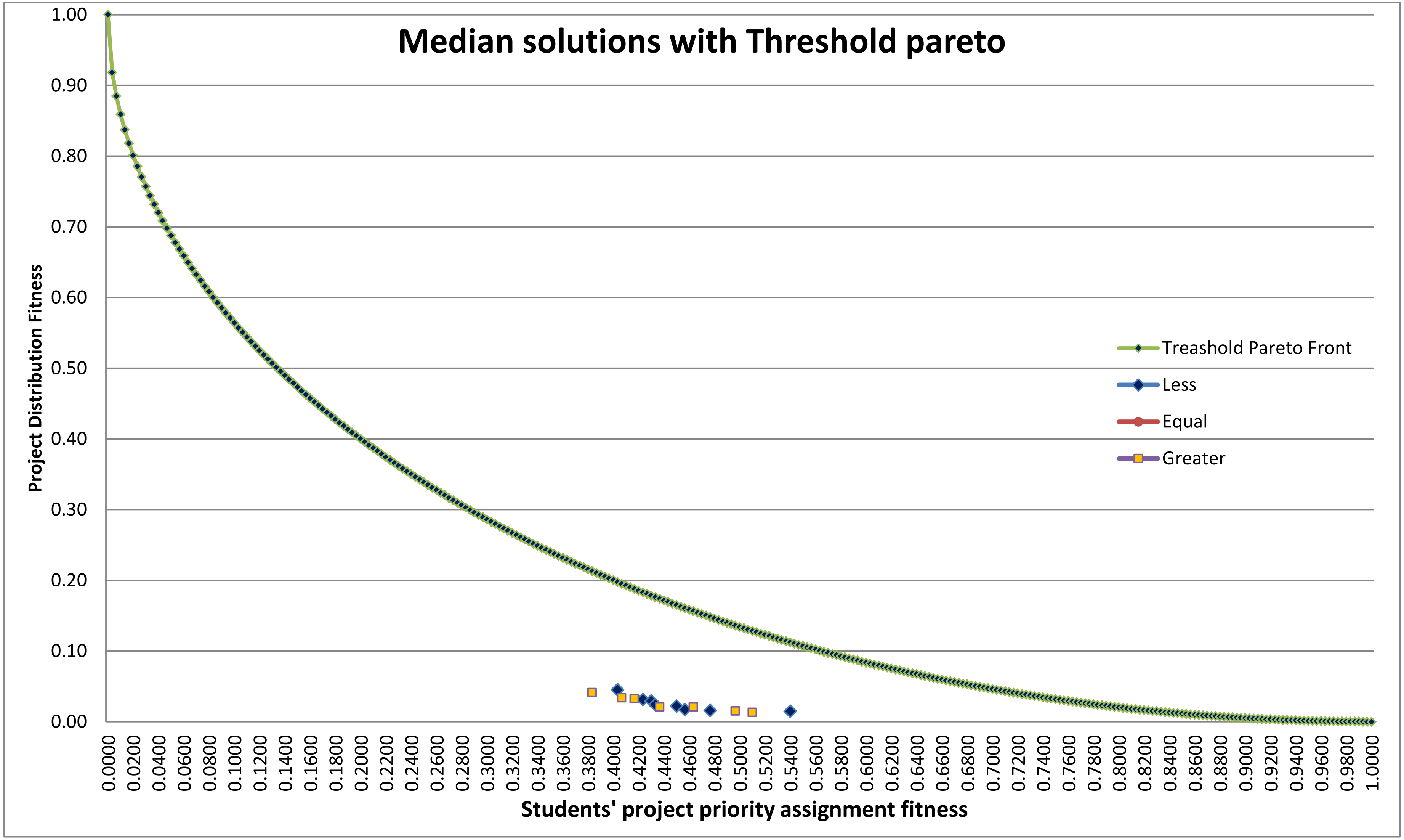


Figure 9.3-10: Median Solutions in regards with Threshold Pareto

10. Critical Appraisal

10.1. Critical Summary

The original aim was to develop a web application which can replace the current system and provide better user experience. I have used modern Modal View Controller (MVC) approach to develop the web application. The application is supported with advanced frameworks such as AngularJS and bootstrap to give a slick look and feel. It is designed as a responsive application which auto-adjusts according to the device screen size and rotation. The application is also able to identify errors and display relevant messages with reasons and possible suggestions to avoid them. In order to provide help on various features, info tags are placed so that the user can gain information about the feature. I integrated chart.js with the client-side AngularJS to display active charts which can handle click events on data points and display relevant detailed information for the chosen point.

I applied the concept of design patterns by following articles and tutorials within the .Net environment. The articles also helped me understand and implement very important aspects of the web development such as authentication, authorization, privacy and safety and discovered different techniques to strengthen the password security such as salting.

The system is capable of sending automated email notifications that saved administrators' time, efforts and also reduces the possibilities for human errors.

One of the main aim of the project was to implement an algorithm which provides allocation suggestions with different trade-offs. With the help of research papers, I managed to evaluate the various evolutionary algorithms in regards to their suitable usage, advantages and disadvantages. The application is supported with NSGA-II algorithm to solve the allocation problem.

I have placed various validations checks both on client side and server side to reduce the risks of system failure.

There are enhancements which are not the part of objectives, however would have been beneficial.
I was deciding upon possible options to develop the system in .NET Core environment. The current environment can only run on windows environment via visual studio however, .NET Core can support various operating systems such as Linux. I decided not to use this approach

as it has only been released in July and there wasn't much help and guidance available at that time regarding this new feature.

The system could be implemented to support multiple departments; however this enhancement can be introduced in future upon success of the current system.

Due to limited time constraints to finish the project, I could not undertake additional rigorous integration tests.

Algorithm's behaviour can be tested further to find out its performance with various crossover approaches and mutation rates in regards to various capacity levels.

I would also gather multiple users' feedback for further enhancement.

I have successfully accomplished the desired aims of the project by producing a high quality application and implementing an advanced, fast and elitist evolutionary algorithm. As an additional feature, the user can adjust the algorithm parameters as they require.

Overall, I have implemented all essential, recommended and optional requirements of the project.

10.2. Discussion of Context

The system is specifically designed for university's informatics department for handling the student project allocation process and could replace the old application for good. The system can be suitably used for other departments' allocation processes as well. However, it can only support single department on one deployment. If many departments wish to use the same application then it needs to be deployed separately for each department.

The system can also be used within any other allocation process where proposers and preferences are involved. For example, in the context of commercial aspects, the system can be used or adapted for employee task allocation process. Managers propose different tasks and employees choose their preference of tasks. The automated allocation process can then be carried out to gather optimal allocation suggestions for employees' to tasks allocations.

From my point of view, the system will not pose any risks or negative impacts on the society or university's processes as it is a standalone system that doesn't interfere with other university's systems.

In fact, as mentioned in the abstract, the system will have positive impact on the department processing as it will save a lot time and effort invested by administrators on manual allocation arrangement.

The automation process has been implemented with fast and elitist Non-dominated Genetic Evolutionary Algorithm – II which has been proved efficient in comparison to other available approaches as mentioned by K. Deb. et al.(2002)[17].

10.3. Personal Development

I have developed many valuable analytical and technical skills during this project and they will be very helpful within my future career establishment.

I have learned to prioritise and manage tasks in timely manner thus helped me strengthen my time management skills.

Regular enhancements and corrections in the project have helped me adapt with changes to the requirements. I have enhanced my communication skills throughout regular meetings, presentations and interviews. I worked with my own initiative, which improved my decision making and problem solving skills.

One of the main technical achievements was understanding and implementation of the optimised algorithms with that I developed my research and experimental skills.

The experience of configuring Server, Network, SMTP, Database and managing project throughout full life-cycle will be very helpful in future as an application developer role as well as project management role.

11. Conclusion

Overall, I have managed to fulfil the desired aims of the project by developing the application and also implementing the NSGA-II optimised algorithm. I have managed to meet all the desired objectives within the time scale and learnt many skills and experience along the way. The application may require further testing and feedback before the final deployment and I would be more than happy to help the university in the deployment process.

There is a potential for the project to be extended further by introducing new features such as -

1. Adding support for the second marker by implementing another algorithm to allocate second supervision to the projects in accordance with proposers' assigned project load.
2. The system can support the feedback process after allocation.
3. System can be extended to support multiple departments via single system.

12. Bibliography and Citations

[1] Oracle Corporation (2016) "*Why is virtualization useful?"*, Available at: https://www.virtualbox.org/manual/ch01.html,(Accessed:13/09/2016).

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

[3] Microsoft Corporation (2016). "*UML Use Case Diagrams: Guidelines*". Available at: https://msdn.microsoft.com/en-us/library/dd409432.aspx. (Accessed:03-10-2016).

[4] 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:04/10/2016).

[5] Brandenburg L.(2016) *"How to Create an Entity Relationship Diagram"*. Available at: http://www.bridging-the-gap.com/erd-entity-relationship-diagram/. (Accessed:04/10/2016).

[6] Ambler S. W. (2003). *"UML 2 Class Diagrams: An Agile Introduction"*. Available at: http://agilemodeling.com/artifacts/classDiagram.htm. (Accessed: 05-10-2016).

[7] 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:07/10/2016).

[8] Microsoft Corporation (2016). "*OData in ASP.NET Web API*". Available at: https://docs.microsoft.com/en-us/aspnet/web-api/overview/odata-support-in-aspnet-web-api/odata-v4/entity-relations-in-odata-v4. (Accessed:20/10/2016)

[9] Microsoft Corporation (2016). "*Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application*". Available at: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application. (Accessed:30/10/2016)

[10] Mittal A. (2015). "*Security in Web APIs-Basic Authentication and Token based custom Authorization in Web APIs using Action Filters*". Available at: https://www.codeproject.com/articles/1005485/restful-day-sharp-security-in-web-apis-basic (Accessed:15/11/2016)

[11] Microsoft Corporation (2016). "*Authentication Filters in ASP.NET Web API 2*". Available at: https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-filters. (Accessed:07/11/2016)

[12] Microsoft Corporation (2016). "*Password Hashing*". Available at: https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing. (Accessed:10/11/2016)

[13] Defuse Security (2016). "*Salted Password Hashing - Doing it Right*". Available at: https://crackstation.net/hashing-security.htm. (Accessed:10/11/2016)

[14] Defuse Security (2016). "*password-hashing*". Available at: https://github.com/defuse/password-hashing/blob/master/PasswordStorage.cs. (Accessed:10/11/2016)

[15] Zeeshan, A. (2014). "*Hashing Passwords using ASP.NET's Crypto Class*". Available at: https://www.codeproject.com/articles/844722/hashing-passwords-using-asp-nets-crypto-class. (Accessed:10/11/2016)

[16] Orchardproject, (No Date). "*Setting up SSL made easy*". Available at: https://blogs.iis.net/thomad/setting-up-ssl-made-easy. (Accessed:10/12/2016)

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm":
NSGA-II: *IEEE Transactions on Evolutionary computation*, April 2002, vol. 6, no. 2, pp- 182-196.

[18] F. Fortin, M. Parizeau "Revisiting the NSGA-II crowding-distance computation": *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation GECCO '13*, 2013, pp. 623-630.

[19] L.L. Minku, (2016). "Multi-Objective Evolutionary Algorithms", CO3091: Computational Intelligence and Software Engineering. Available at: https://campus.cs.le.ac.uk/teaching/resources/CO3091/lecture_notes/12-NSGA-II.pdf (Accessed:01/02/2017).

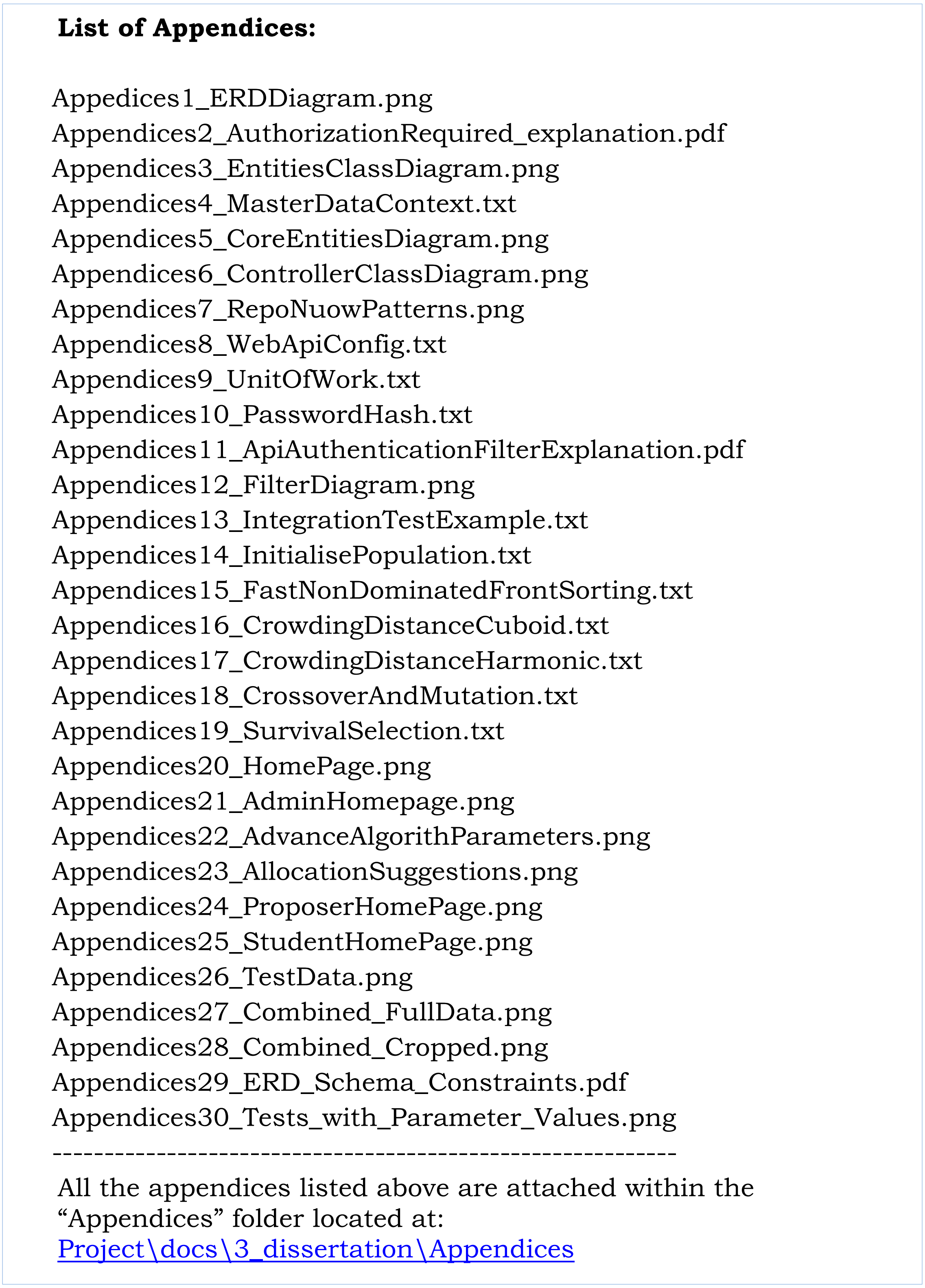
[20] V. L. Huang, P. N. Suganthan, A. K. Qin and S. Baskar, "Multiobjective differential evolution with external archive and harmonic distance-based diversity measure", Technical Report, Nanyang Technological University, 2005.

[21] L.L. Minku, (2016). "Energy Consumption Optimisation", CO3091: Computational Intelligence and Software Engineering. Available at: https://campus.cs.le.ac.uk/teaching/resources/CO3091/lecture_notes/13-energy-consumption-optimisation.pdf (Accessed:10/04/2017).

[22] ProjectList,(no date). “Online Project/Dissertation Allocation System”, Available at:https://projectlist.lboro.ac.uk/info/?p=overview (Accessed:29/04/2017)

[23] D. J. Abraham, R. W. Irving and D. F. Manlove, "The student-project allocation problem," Proc. ISAAC 2003, LNCS 2906, pp. 474-484, 2003

13. Appendices



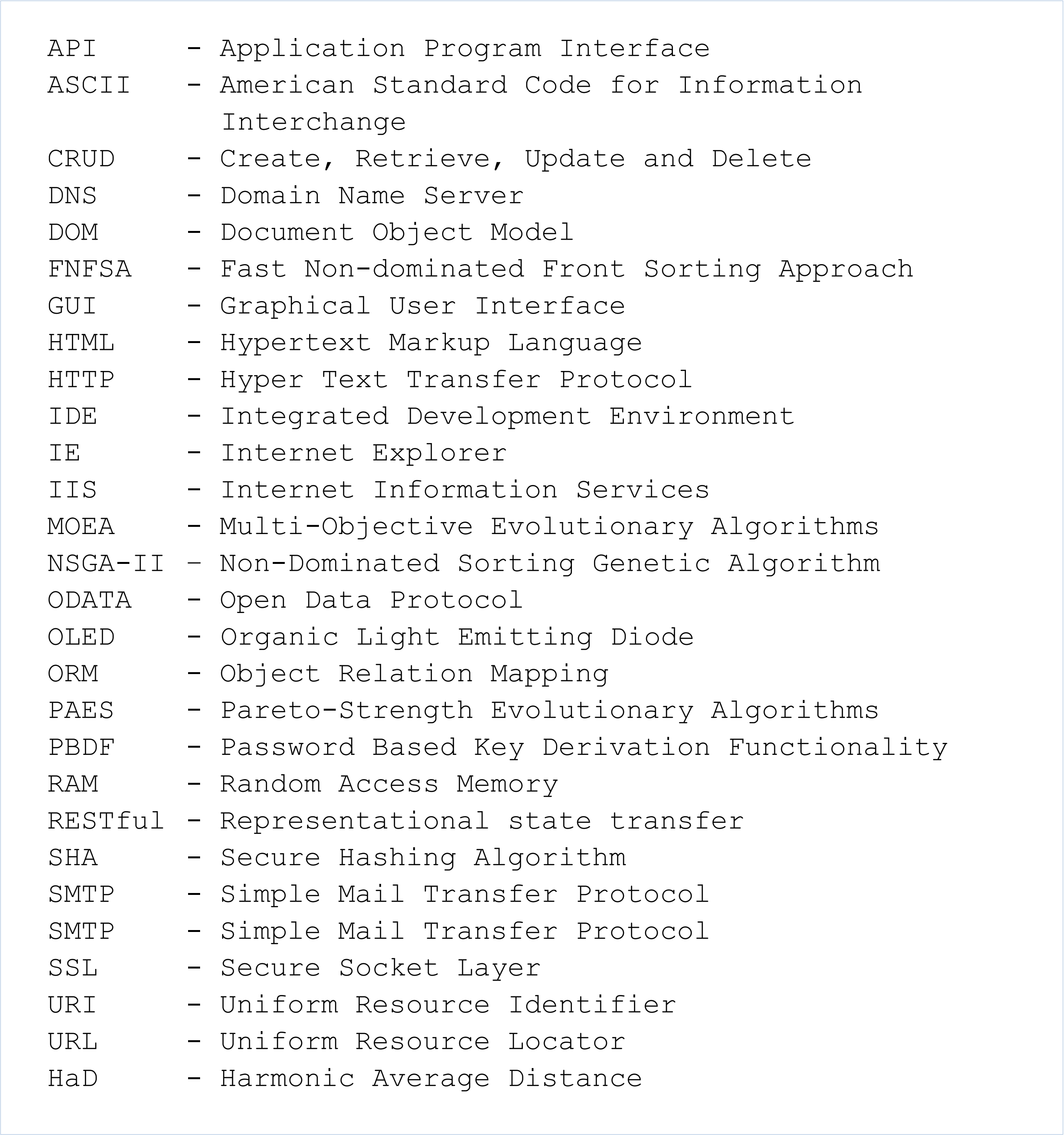
List of Appendices:

Appedices1_ERDDiagram.png
Appendices2_AuthorizationRequired_explanation.pdf
Appendices3_EntitiesClassDiagram.png
Appendices4_MasterDataContext.txt
Appendices5_CoreEntitiesDiagram.png
Appendices6_ControllerClassDiagram.png
Appendices7_RepoNuowPatterns.png
Appendices8_WebApiConfig.txt
Appendices9_UnitOfWork.txt
Appendices10_PasswordHash.txt
Appendices11_ApiAuthenticationFilterExplanation.pdf
Appendices12_FilterDiagram.png
Appendices13_IntegrationTestExample.txt
Appendices14_InitialisePopulation.txt
Appendices15_FastNonDominatedFrontSorting.txt
Appendices16_CrowdingDistanceCuboid.txt
Appendices17_CrowdingDistanceHarmonic.txt
Appendices18_CrossoverAndMutation.txt
Appendices19_SurvivalSelection.txt
Appendices20_HomePage.png
Appendices21_AdminHomepage.png
Appendices22_AdvanceAlgorithParameters.png
Appendices23_AllocationSuggestions.png
Appendices24_ProposerHomePage.png
Appendices25_StudentHomePage.png
Appendices26_TestData.png
Appendices27_Combined_FullData.png
Appendices28_Combined_Cropped.png
Appendices29_ERD_Schema_Constraints.pdf
Appendices30_Tests_with_Parameter_Values.png

All the appendices listed above are attached within the "Appendices" folder located at:
Project\docs\3_dissertation\Appendices

Figure 10.3-1: A List of Appendices

14. Glossary



API	-	Application Program Interface
ASCII	-	American Standard Code for Information Interchange
CRUD	-	Create, Retrieve, Update and Delete
DNS	-	Domain Name Server
DOM	-	Document Object Model
FNFSA	-	Fast Non-dominated Front Sorting Approach
GUI	-	Graphical User Interface
HTML	-	Hypertext Markup Language
HTTP	-	Hyper Text Transfer Protocol
IDE	-	Integrated Development Environment
IE	-	Internet Explorer
IIS	-	Internet Information Services
MOEA	-	Multi-Objective Evolutionary Algorithms
NSGA-II	-	Non-Dominated Sorting Genetic Algorithm
ODATA	-	Open Data Protocol
OLED	-	Organic Light Emitting Diode
ORM	-	Object Relation Mapping
PAES	-	Pareto-Strength Evolutionary Algorithms
PBDF	-	Password Based Key Derivation Functionality
RAM	-	Random Access Memory
RESTful	-	Representational state transfer
SHA	-	Secure Hashing Algorithm
SMTP	-	Simple Mail Transfer Protocol
SMTP	-	Simple Mail Transfer Protocol
SSL	-	Secure Socket Layer
URI	-	Uniform Resource Identifier
URL	-	Uniform Resource Locator
HaD	-	Harmonic Average Distance

Figure_10.3-2:_Glossary