

Volunteer Project Matching

Final Report

Submitted for the BSc in
Computer Science

May 2016

By

Robert Laurence Smith

Word Count: 12,271

Acknowledgements

I would like to thank Dr. John Rayner for the proposal of this project, and the support and guidance he has given throughout.

Furthermore, I would like to thank my fiancée Amber McLaren who has supported me through the best and worst times of my three years at university.

Table of Contents

| | |
|--|----|
| Acknowledgements | 2 |
| 1 Introduction | 5 |
| 1.1 Initial Brief | 5 |
| 2 Aim and Objectives | 6 |
| 3 Background..... | 9 |
| 3.1 Alternative Solutions | 9 |
| 3.2 Technologies | 10 |
| 3.2.1 Database Design | 10 |
| 3.2.2 Programming Languages..... | 14 |
| 3.2.3 System Architecture..... | 16 |
| 3.3 Security Considerations | 17 |
| 4 Technical Development..... | 20 |
| 4.1 System Design & Implementation | 20 |
| 4.1.1 Use Case Analysis..... | 20 |
| 4.1.2 Database Development | 20 |
| 4.1.3 Website Development..... | 23 |
| 4.2 Matching Algorithm | 25 |
| 4.2.1 Skills, Interest, and Location Matching | 25 |
| 4.2.2 Availability Matching | 25 |
| 4.2.3 Algorithm Refinement – Hidden Projects..... | 27 |
| 4.2.4 Partial Matching..... | 27 |
| 4.2.5 Strict Matching..... | 27 |
| 4.2.6 Testing..... | 28 |
| 4.3 UI Design..... | 28 |
| 4.4 Data Dictionary Performance | 30 |
| 5 Evaluation | 32 |
| 5.1 Project Achievements | 32 |
| 5.2 Further Development | 36 |
| 5.2.1 Strict Matching..... | 36 |
| 5.2.2 Availability Issues (<i>DayX-All</i>) | 36 |
| 5.2.3 Potential reach..... | 36 |
| 5.2.4 Improvements to administration console | 36 |
| 5.2.5 General User Interface Improvements | 37 |
| 5.3 Personal Reflection..... | 37 |
| 6 Conclusion | 38 |
| Appendix A: Database Entity Relationship Diagram | 39 |
| Appendix B: Database Attributes..... | 40 |

| | | |
|-------------|-------------------------------------|----|
| Appendix C: | Use Case Diagram | 46 |
| Appendix D: | Algorithm Flow Diagram Design | 47 |
| Appendix E: | Database Efficiency Testing | 48 |
| Appendix F: | Initial Time Plan..... | 53 |
| Appendix G: | Revised Time Plan | 58 |
| References | | 62 |

1 Introduction

The purpose of this document is to expand on the reports submitted for this project and to discuss the project as the development comes to a finish including an evaluation of the project.

The project title is 'Volunteer Project Matching' and the goal of this was to create a volunteer matching system that would be made available through the World Wide Web, with the system being subject to suitable security and privacy provisions.

The intention for the system would be for volunteers to sign-up to the website as a voluntary user, select a number of skill, interest and location-based attributes, and the availabilities that applied to them. Charities would use their side of this of the system to add a project which has attributes in the same way.

Where these attributes match the volunteer's, the volunteer would be shown relevant projects, ranked based on the number of matches.

The system would be used by a local volunteering bureau to enable the listing of projects local to the area, of which charities can list their projects to local people. The benefit of this is that when someone wants to volunteer for a project they can visit their local volunteering bureau and find projects near them and be matched to suitable ones.

The volunteering matching system is to be developed using PHP along with SQL for a Microsoft SQL Database backend. Additional functionalities within the website use AJAX, JQuery and JavaScript programming languages. Bootstrap would be used as the website development platform to enable cross-browser and device compatibility.

During the lifecycle of the project, there has been a number of systems research, planning ideas and implementation strategies.

This report will discuss the development of the overall project including; Aims and objectives, Background, Technical Development, Evaluation and a Conclusion, all of which critically evaluate the progress of the project and the steps in-between.

1.1 Initial Brief

"Local community volunteer support organisations usually provide a service to local charities to attract people who can help out with projects. The skills required and the location of the voluntary work in the local area are matched with the skills, interests and availability of volunteers, in a variety of different activity categories. A computer database and skills matching system would potentially allow prospective volunteers to pre-select relevant projects before discussing details with volunteer support workers. The system could ideally be made available through WWW, subject to necessary security and confidentiality provisions."
(Rayner, 2016)

2 Aim and Objectives

The overall aim of the project is to create a system whereby registered volunteers can be matched to available projects based on the; skills, interests, availability and location of the volunteer.

The system would be made available via the WWW (a website) with volunteer and project data stored in a database. The system is to take into account security and confidentiality provisions.

In order to be successful in the achievement of the project aim, the following objectives will have to be met;

Objective 1 – Volunteer Matching Algorithm

The project is based on volunteer project matching – the system needs to be able to match volunteers to projects, and potentially projects to volunteers.

In order to achieve this, an algorithm will be developed to enable the matching process. The user would be matched to projects and the output of the matching would be displayed to the user, with 'closer matching' i.e. matches with more criteria, listed further towards the top of the list.

This would be evaluated by creating a number of different users with different skills and availabilities – whom would be matched against a set of projects. The suitability of the project to the user would be evaluated to determine the success of the match.

Deliverables for objective:

- Matching algorithm for users to projects
- Listed output of suitable projects for users to select

Objective 2 – Suitable Database Creation

All the data about projects, users, charities, and details of projects and users (e.g. Skills had/required, Availability, Locations ...) are stored in the database, which the algorithm would use to query.

In order to achieve this, the design of the database would need to be suitably planned. Once planned the data would need to be suitably normalised.

The suitability of the database would be evaluated by the ease of storing of data and the efficiency of this i.e. the referential integrity of the data, and the dependencies.

Deliverables for objective:

- Design for the database (Entity Relationship Diagram)
- Creation of database

Objective 3 – Web User Interface

The volunteer matching system is to be used via the World Wide Web. The website should be easy to use for the different users to achieve their purpose of use (e.g. Volunteering, Project Management, Site Administration ...)

In order to achieve this, a design of the website would be created, and then the design would be made into the live version. Using a tool such as Bootstrap will provide an easy platform to create a website. This is discussed further in the web development section.

The suitability of the web interface would be evaluated by ensuring the different user classes are able to use the website for their purpose.

Objective 4 – Security Considerations

Part of the scope of this project is taking security considerations into account. Security also falls within the Data Protection Act, BCS 'Code of Ethics' and user trust.

Therefore, data that is stored about users should be kept secure, additionally, security provisions should be implemented onto the website to ensure users are authenticated and authorized to the relevant parts of the system.

In order to achieve this, different forms of security will be implemented to protect the data and safeguard the system from unauthorized access and use. Examples of intended implementations include; a login mechanism, parameterization of queries, access restrictions, and levels of access.

Security of the system would be evaluated by trying to misuse the system and ensuring the security implementation prevents such activities.

The deliverable for this objective is the security implementation and testing of security.

Objective 5 – Administrative Functions

Volunteer/Project matching will use interests, skills, location and availability as part of the algorithm. The 'dictionary' of the skills, interests and locations shall be manageable via an 'administration console' and stored within the database. Other admin functions include; user management and charity management.

To achieve this, an administration area is to be created where administrators can manage the dictionaries.

This will be evaluated by testing the administrative functions and if; new items can be added, items can be edited, or item removed.

Objective 6 – Managing of project listings

Volunteers are matched to projects, these projects are added by a charity user who can create and edit projects. Projects should also be able to be hidden and removed from the website listings.

This would be achieved by creating an area specifically for charities to use where they are able to create, edit, enable/disable, and delete projects

The management of projects would be tested by creating and managing projects which would then be used for the testing of matching algorithm.

Objective 7 – Event Book On

On a project listing there will be an option to register interest for that project. Charity users would have the option to approve the volunteer, enabling the volunteer to be booked onto the project. This objective would enable two functions; the charity to have a list of expected volunteers, and for the volunteers to have a history of projects they have participated in.

This would be achieved by having a table of applications for projects, and having a button on a project listing to apply for that project, with charities users having the option to accept or reject that volunteer.

Objective 8 – XML Import / Export (Secondary)

Some charities may operate in a wider area and may want to upload the same project to different volunteering bureaus that use this volunteering system. Therefore projects should be able to be exported and imported.

This would be achieved by having the option to export project data on the listing and on the creation of projects the ability to import via XML.

XML export and import abilities would be tested by using the functionality which would be on the charities area of the website.

Objective 9 – Notification Messaging (Secondary)

Within the system, there should be a mechanism where users can be informed of actions that have taken place, this notification/messaging functionality could be a replacement to sending emails directly, and would be used to inform users of system events.

e.g. 'User: John Doe (ID: 000) – Has expressed interest in your project 'Park Cleanup'

This would be achieved by having a table for messages to be stored including; who it is for, who it is to, message, 'is read' flag, and a timestamp. Actions within the website can then create notifications or messages to send to the user. This can be tested as the system is developed.

3 Background

The aim of the project is to create a system which would enable volunteers to be matched to projects based on skills, interests, availability, and location.

Results from the Citizen and Community Life Survey 2014/15 show an estimated 14.2 million people formally volunteered at least once a month in 2014/15, with 21.8 million volunteering at least once a year, if the gathered results were extrapolated to the UK population. (National Council for Voluntary Organisations, 2016)

Based on the 2011 census of the UK population of 63.2 million (Office for National Statistics, 2012) and the extrapolated volunteering data; of around 14.2 million people volunteering, this means around 22.5% of people in the UK volunteer at least once a month. As the aim of the project is creating a volunteer matching solution that is local, potentially to a volunteering bureau, a potential user base size can be calculated based on target location population.

| Location | Approximate Population | Potential User base |
|--------------------|------------------------|---------------------|
| Kingston Upon Hull | 260,000 | 58,000 |
| Leeds | 750,000 | 168,750 |
| Middleborough | 142,000 | 31,950 |
| London | 8,500,000 | 1,912,500 |

Table 1. Potential user bases for different locations using the volunteering system.

As the volunteering system intends to be used locally, based on the statics sourced by the National Council for Voluntary Organisations, Table 1 shows the number of potential users if implemented in that location.

Volunteering matching to projects has existed for a while, an example of one is the Hull Community and Voluntary Service (Hull CVS), and they established in 1981 and provide a drop-in service to find suitable opportunities for people who wish to volunteer. (Hull CVS, 2015). As well as using do-it.org, which will be discussed further on, they offer a drop-in service to find volunteering opportunities.

Given this Hull CVS or any service who seeks to find volunteering opportunities could use and administrate this system, allowing the users to find opportunities, or vice-versa.

According to the 2013/14 annual report from the Hull CVS, over the reporting period over 380 members of the public used their physical centre to be supported into volunteering – with 406 people being matched to a volunteering project who are deemed to have extra support needs. (Hull CVS, 2014).

3.1 Alternative Solutions

One of the largest solutions that provide a volunteer to project match is do-it. “*Do-it is the national database of volunteering opportunities*”. (National Council for Voluntary Organisation, 2015). The website will be useful in terms of guidance for the scope of this project.

Do-it.org is used across the UK for volunteering activities; Hull CVS also use do-it as part of their services to charities who are recruiting volunteers.

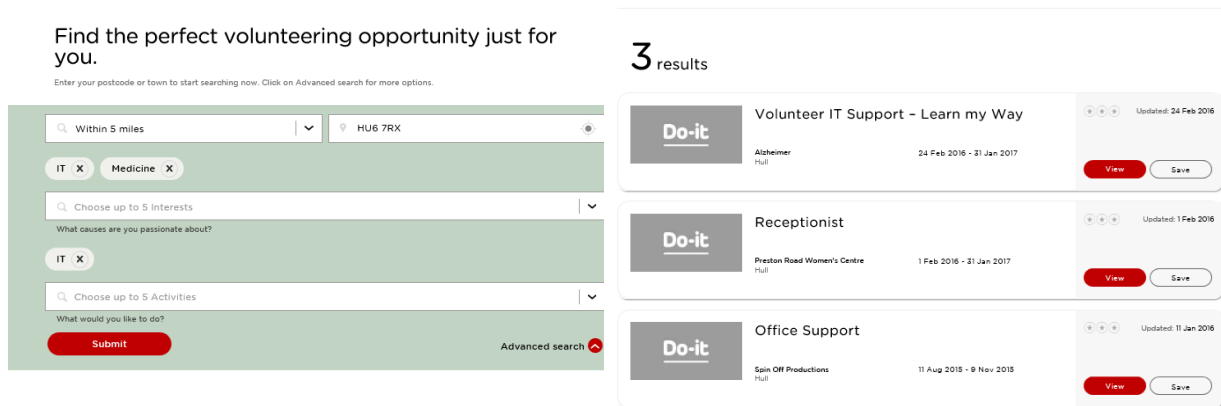


Figure 1: Screenshot of do-it.org matching to suitable projects. (Do-it Trust, 2016).
Criteria Used (Left) – Results (Right)

Hull CVS also have their own offline alternative solution, they provide a drop-in service twice a week where their team of advisors offers drop-in sessions, where their interview process is explained and suitable opportunities are found. (Hull CVS, 2016)

Matching solutions also exist in different forms other such examples are; job search websites, such as Intern Avenue. Intern Avenue provide a database of graduate opportunities where graduates are matched based on degree, location and opportunity preferences e.g. field of work or pay expectations.

Outside of the work paradigm, there exists a website called 'Suggest Me Movie' that matches films based on user criteria such as; Genre, IMDb score, and release year. However, the website displays film results at random, compared to showing a list of films with the user's entered criteria.

3.2 Technologies

3.2.1 Database Design

Data used for the system would be held within a database. There are different database platforms that can be used to store the data. Examples of database platforms include; MySQL, MSSQL, Oracle and MS Access (W3Schools, 2016).

There are many different strengths when it comes to the use of a database:

- Stored data can be queried. The querying of data is essential for the matching process.
- Scalability – as the system grows, i.e. more; users, projects, and charities. The data volume increases, and a database would be suitable for this.
- Relations – with a relational database, records can be related to another with the use of foreign keys, as an example. Allowing 1:1 or 1: Many relations between different entities, such as 1 user : Many skills. This helps prevent duplicate data values, by relating data via foreign keys.
- *Database transactions are processed reliably, they are processed; Atomicity, Consistently, in Isolation, and Durably.* (Haerder & Reuter, 1983, pp. 287-317). Though not used within this system, it could be used within transactional databases. As ACID protects the integrity of data.
- Database access is concurrent, enabling multiple users to access stored data. Limiting the risk of corrupting data, compared with a data file.

Furthermore content within the website doesn't need to be static, project listings are created and then dynamically displayed from the database record of the project. This means the website's content is managed and changes to the code are not required.

A relational database management system (RDBMS) in the form of Microsoft's SQL server has been chosen as the database platform. Alternative ways of the storing of data could be to use flat files. The latter option would not be suitable due to the strengths of use of a database listed above.

Infrastructure restrictions have too resulted in the chosen database platform to use MS SQL. However MSSQL and MySQL are within the top 3 of database technologies used (DB-Engines, 2016) and the developer has experience of using both RDBMS.

3.2.1.1 Object Orientated VS Relational Database Models

Other forms of database types exist which include; Relation Databases, Hierarchical, Network, and Object Orientated (OO).

An Object Orientated Database contains data in the form of objects and can be accessed and manipulated by object orientated programming languages; such as Java or C++. The data being used by the system is about a user which is seldom edited and attributes which are stored as a foreign key on user ID or project ID, along with a foreign key to a skill or interest. As such an OO approach could be argued as excessive. An OO approach is more suitable when high performance is a factor along with complex data. (Barry, 2016)

A Relational Database is a collection of related information that is organised into tables. Each table stores data as row a row in the table; where the data is arranged into columns. Tables are stored within database schemas in the RDBMS where a user can access their tables or grant permissions for others to access the tables. (Price, 2014)

3.2.1.2 Available Database Technologies

The system would make use of data storage in order to hold data and to be used in the matching process. There are different database platforms that exist that are capable of holding data, examples include;

MySQL

MySQL is a popular Open Source SQL DBMS, which is developed and supported by Oracle. The databases provided by MySQL are of the relation model, where data is stored in multiple related tables, instead of one large data warehouse. MySQL was initially developed to handle large databases much quicker than existing solutions. (Oracle Corporation, 2016)

MS SQL SERVER

Microsoft SQL Server is another server-based database management platform available to use for the project, which is developed by Microsoft.

ORACLE

Oracle is another relational DBMS, developed by Oracle and released in 1977. (Beynon-Davies, 2000, p. 195). Although Oracle and MS SQL are both relational SQL databases which are suitable to use within the project, there are some slight differences in regards to the SQL. For example; MS SQL uses Transact-SQL (T-SQL) which is easier to use, while Oracle databases use Procedural Language SQL. Each SQL type is a different 'dialects' and use different syntaxes, and have different capabilities. (Stansfield, 2014).

MS ACCESS

Microsoft Access (MS Access) is a desktop based DBMS, which is developed by Microsoft as is part of the Microsoft Office Suite or can be bought on its own separate instance. Access databases tend to be run on standalone, however, it can be used with a multi-user system, albeit with issues for example transaction management is non-existent. Access tends to be used for low volume, small to medium complexities. Beyond this Access can be used as a front end, with data being stored in another database solution, such as Microsoft SQL Server. (Beynon-Davies, 2000)

NOSQL

NOSQL stands for Not-Only-SQL and was developed to solve the issues that Relational Databases face. (MongoDB, 2016). One of the advantages to NOSQL is the ability to run on clusters, as relational databases are not designed to run efficiently on a clustered setup, and there are four types of NOSQL databases which are; Key-Value, Document, Column Family Stores, and Graph Databases. (Sadalage, 2014).

FLAT FILE

A flat file database structure is designed around a one single table structure. Whereby data is saved in a single table (or list) with fields representing all the attributes. (DatatabaseDev, 2015). Issues with a flat-file database would be the potential for data redundancy, this is because the lack of relational capabilities and there is potential for duplication. For example, if a customer order database was implemented using a flat file model, elements such as customer name and address may be duplicated if they have ordered many products.

Compared to a relation model, the customer would have a 1-to-optional many link on a customer table which would hold the customer ID, product IDs, and other necessary attributes for the system. There is a less duplication, for example, the customer table would hold names and addresses once, unlike a flat file.

Due to the complexity of the system in terms of relationships between entities, as discussed in the database development section, along with the potential of data replication, a flat file structure is not suitable.

3.2.1.3 Entity Relationship Model

An entity-relation model (ER model) consists of entities and the relationships between entities. The ER model describes the requirements of the database. (Mannila & R  ih  , 1992).

There are two types of entities – ‘strong entity’ which is one that can exist independently i.e. has no dependencies. While a ‘weak entity’ requires a link to another entity i.e. dependent on another. Further to this, an entity can be described as Physical, (e.g. Volunteer, Charity, Project, etc...) or Conceptual (Booking, Availability, Message, etc...)

The entity relationship diagram would show the relationship between different entities in the database whereas the schema describes the structure of the database. Figure 2 (below) shows an example of a relationship within the system, where a charity has many projects, and a project can have many volunteer bookings.

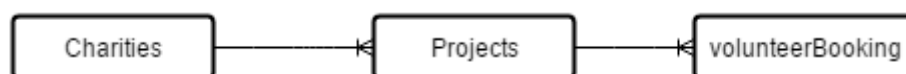


Figure 2. Example of an Entity Relationship

The entity relation diagram of the system's database is in Appendix A. The structure of the database in regards to attributes is in Appendix B.

From Figure 2, the 'volunteerBooking' table would hold the user ID of the volunteer along with the project ID of which the volunteer has booked onto. The project would hold the ID of the charity whom the project belongs to.

Within the ER model there are different types of joins available to denote a relationship between attributes, these are; one-to-one (1:1), one-to-many (1:M), and many-to-many (M:M). With a one-to-one relationship, tables are associated together using the primary key in both tables. In a one-to-many relationship, the primary key in the one must be an attribute in the many table, which would be the foreign key to link back to the one.

"In a many-to-many, a new table is always created with the primary key stored in the bridge with the originally primary key used to form the primary key or compound key." (University of Washington, 2001).

3.2.1.4 Normalisation

A relational database has been proposed for the project, therefore, normalisation would be required. *"Normalisation identifies a set of suitable relations [,] the benefits of which is that it is easier for the user to access and maintain data, and minimal storage take up"* (Connolly & Begg, 2005, pp. 388-389).

Normalisation involves the creation of new tables and creating the relationships in between, reducing; data redundancy i.e. duplication of data, inconsistent dependencies (Microsoft, 2013) and issues on data replication; such as update and delete.

There are many forms in the normalisation process; starting with UNF (Unnormalised Form), each normal form stage builds on the last and addresses and fixes issues. *"In most cases Third Normal Form is sufficient enough to ensure a high degree of data integrity"* (Taylor, 2006, p. 116)

However, there are multiple steps in the normalisation process, where each rule builds on top of each other in order to further ensure data integrity and up to fifth normal form are;

UNF to First Normal Form (1NF)

1NF requires data within a table does not repeat and that *"attributes are atomic, i.e. the attribute cannot be broken into down further meaningful components"*. (McFadyen, 2006)

1NF to Second Normal Form (2NF)

Data would be in 1NF and the partial dependencies on the primary key would need to be eliminated and moved to a new table. Meaning the non-key attributes are fully functional dependent on the primary key. (Beynon-Davies, 2000, p. 217).

2NF to Third Normal Form (3NF)

Data would meet 2NF and there are no transitive dependencies. This would involve the removal of attributes that depend on non-key attributes. All attributes would depend on the primary key and the primary key alone. (Fleming & Von Halle, 1989, p. 187)

Boyce-Codd Normal Form (BCNF)

BCNF could be compared to a stronger version 3NF, if a relation is in BCNF it would also meet the rules of 3NF. *"A relation is only in BCNF if every determinant is a candidate key, a determinate is any attribute on which another attribute is fully functionally dependent"* (Russell, 2008)

There are other higher normal forms that can be completed which address other issues, typically 3NF or BCNF tend to be sufficient by removing functional and transitive dependencies within the database.

BCNF to Fourth Normal Form (4NF)

4NF requires that the relation is in BCNF and non-trivial multivalued dependencies are removed. (Connolly & Begg, 2005, pp. 428-429)

Fifth Normal Form (5NF)

A table that meets 4NF can be in 5NF so long as there are no join dependencies and that the table cannot be non-loss decomposed into a series of smaller tables. (Beynon-Davies, 2000, p. 228).

3.2.1.5 Attributes

Suitable attributes were fully decided upon once the relationships between entities were finalised and normalisation to the sufficient normal form. The design of the attributes used can be seen in Appendix B.

3.2.2 Programming Languages

3.2.2.1 Scripting Languages

A scripting language is that type of programming language used to create programs of limited capability, called scripts. Unlike other programming languages where code can be compiled and executed, these scripts are interpreted on the server on execution. (Microsoft MSDN, 2016)

There are two types of scripting languages for the web; these are client-side scripting and server-side scripting.

A server-side scripting language is a programming language which executes on the server and performs an action. Examples of suitable scripting programming languages include; PHP, JSP, and ASP. (Connolly & Begg, 2005, p. 1011).

PHP

PHP stands for Hypertext Pre-processor. It is *“a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.”* (Connolly & Begg, 2005, p. 1014). One of the advantages of using PHP is that scripts can be included within HTML content, allowing the execution of scripts on any page, upon load– this allows the creation of dynamic content with only a HTML template being created. Further to this PHP can be used to interact with different database technologies such as MySQL and Microsoft SQL Server. With PHP, it supports both IIS and Apache web server platforms. (W3Schools, 2016).

ASP.Net

ASP.Net is a server-side programming language developed by Microsoft. ASP.Net is another scripting language, which can be written in any .NET language such as C# or VB.Net.

3.2.2.2 Client Side Scripting

Unlike server-side scripts, client-side scripts are interpreted and run by the browser, suitable programming languages include; JavaScript, AJAX and jQuery. These scripts are downloaded and interpreted by the browser. This means the client-side scripts are

accessible to the user if viewing the page source resulting in the potential for security issues; for example, removing validation, though input checking should be done on both sides. Furthermore, if a user has JavaScript disabled, the scripts will not run, resulting in functions which rely on the enablement of JavaScript, not running.

3.2.2.3 Choice of development language

Based on the three main server-side scripting languages discussed above, a repertory grid analysis has been used to further influence a suitable choice of language.

The following measures would be carried out when looking into a language;

Developers Knowledge looks at whether the developer has used or had experience in the programming language before, and whether the previous experience will be useful to the project.

Support – During development, there could be times where further research is needed in order to aid the development process. The support aspect looks at documentation behind the programming language, and if guides or libraries are easily accessible, additionally if online communities are present such as ‘Stackoverflow’ to name one.

Compatibility – Does the programming language support the given environment? The provided environment is hosting on a web server running Internet Information Services with a Microsoft SQL 2008 database.

Scoring – For every attribute there is a trait which is what the best outcome is for a given attribute which scores a 3. Alternatively, the opposite is the least preferred which scores 1. A score of 2 indicates an in-between.

| Attribute: | Developer's Knowledge of Language | Support (e.g. communities, frameworks & documentation) | Compatibility with given infrastructure | |
|-----------------------|--|---|--|------------|
| <i>Trait Opposite</i> | <i>High (3) Low (1)</i> | <i>High (3) Low (1)</i> | <i>Full Support (3) Not Supported(1)</i> | Sum |
| PHP | 3 | 3 | 3 | 9 |
| ASP.Net | 2 | 3 | 3 | 8 |
| JSP | 1 | 2 | 2 | 5 |

Table 2. Repertory Grid Analysis (RGA) of server-side programming languages

“The REP grid technique is derived from George Kelly’s personal construct theory and is based on an object-attribute matrix, containing a Likert-type scale. It is suitable for statistical analysis allowing an interpretation of results.” (Kent & Williams, 2001).

Table 2 shows the analysis of choosing a programming language for development.

From the use of Repertory Grid Analysis as seen in Table 2, PHP has the higher number of positive factors, compared with other suitable programming languages.

On the documentation and support front, there is high-level support of the language through the PHP language website. The documentation also supports previous versions of PHP, which will be useful as the infrastructure has been limited to PHP version 5.3.10 whereas the latest version is PHP 7.0.5. (The PHP Group, 2016).

As for developer's experience, PHP has been used more often than ASP.Net where the experience is from the use of the .net language, notably C# with minimal VB, which are both programming languages supported by ASP. The developer has had no experience of JSP but is aware of its presence and capabilities.

PHP is also fully compatible with the given infrastructure which is Windows based using IIS. Further to this, if the website was to be ported over to Apache (Linux based web servers) the system would still be functional, compared to development in ASP.Net where additional modules may be required such as <mod_mono> (Mono Project, 2016). This means overall PHP is fully compatible with the given infrastructures, as is ASP, JSP would be also with an additional add-on, however if the project was to be expanded – if PHP was used, it would be supported by many different web servers.

There are other measures that can be considered when choosing a programming language. One example is time-to-develop, whereby a task could be performed in fewer lines of code in one language, and more in another. For example PHP vs ASP, in some cases, PHP can be quicker to code in. Additionally, ASP is compiled, whereas PHP is interpreted at runtime.

As such the chosen development language for the system's server-side functionalities would be using PHP.

3.2.2.4 Database Querying

SQL stands for Structured Query Language and is the de facto language for accessing databases, from standardization by the American National Standards Institute in 1986, which happened as a result of IBM using SQL in their first RDBMS product, with smaller DBMS companies modeling their languages in order to be compatible with IBM's DBMS. (Taylor, 2006)

Using SQL would enable to querying of data within the database, however, it can also be used for database management such as insertion, deletion, or modification; of data, databases and database structures.

There are some alternatives to using SQL as a language could include LINQ, which stands for Language Integrated Query and is a programming model that introduces the ability to use queries into any Microsoft .NET language. (Russo & Pialorsi, 2007, p. 2)

Alternatives to the relational database model can include NoSQL.

3.2.3 System Architecture

As stated in the aims and objectives of the project, the system would make use of the World Wide Web, which uses the HTTP protocol. This approach is client-server whereby a client (or multiple) can use the system which is held on a web server with the database stored on an SQL server. The advantage of this method is the enablement of concurrent access to the system; compared to an all-in-one system that could use a Microsoft Access front end, which is held on a computer within a volunteer bureau.

With a client/server approach; there are two options. The system could be hosted internally (intranet based) however, this would mean the bureau would need to self-host the system. However, this could be possible using a dedicated computer running WAMP (Windows Apache MySQL PHP) or IIS (Internet Information Services). Via this method the system would not be accessible via the WWW unless access was made by some form of an extranet. Intranets can be thought of as internal websites, extranets can allow external access to intranet systems.

The other client/server option is having the website accessible on the web, with this approach there could be greater potential for a larger user base, however, there is the increased risk of attack, and responsibility to ensure data is kept secure and complies with the data protection act.

3.3 Security Considerations

Any websites hosted online or hosted internally (intranet) can be at risk from attack. Especially if security is not considered, the website could be used maliciously or even worse, the data with the database; edited deleted, stolen or accessed by an unauthorised party.

Security considerations are important to implement, especially with the high-profile breaches that have happened in the past. Table 3 (below), shows a list of high-profile breaches, which were a result of an online attack.

| Website / Provider | Breached Accounts |
|--------------------|-------------------|
| Adobe | 152,445,165 |
| Ashley Madison | 30,811,934 |
| 000webhost | 13,545,468 |
| Vtech | 4,833,678 |
| Snapchat | 4,609,615 |
| Forbes | 1,057,819 |
| Yahoo | 453,427 |
| TalkTalk | 157,000 |

*Table 3. High profile account breaches, ranging from 2012 – 2016.
(Hunt, 2015) Talk-Talk figure by (BBC, 2015).*

Attacks on websites are always possible with a reported 30,000 websites on average are attacked a day (Lyne, 2013). Therefore, there is a need to safeguard the data held by the system to prevent unauthorized access or use. Though the websites listed in Table 3 have a higher user-base, thus are more likely to be victims of attack, as they have a higher user base it is likely there are implemented safeguards to prevent such attacks. The table highlights the need to ensure there are security provisions in place.

There is also a legal aspect to protecting the personal information of users. Principle 7 of the Data Protection Act states:

*“Appropriate technical and organisational measures shall be taken against unauthorised or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.”
(Information Commissioner's Office, 2016)*

In order to ensure the system has safeguards towards security, one could argue that there are three principles; Authentication, Authorisation and Accounting/Auditing (AAA). AAA is a security protocol used on networks (The Internet Engineering Task Force, 2007); this could form the basis for implementing security.

An example of how following the AAA standard on this system could be
Authentication: Users are authenticated and have their own accounts.
Authorisation: Users have access rights depending on their use class.
Auditing: Use tracking and error logging.

Further security implementations include:

Input Validation

Input validation could be used to determine what the user has entered is valid or not, this helps ensure the user is not entering an 'illegal' value either deliberately or purposefully. If an input is determined to be invalid, it could then be simply rejected by returning false.

Parameterized Queries

A parametrized query is a query where placeholders in the form of question marks are used for parameters, with the parameter values being supplied when the SQL statement is executed. (Swan, 2008). Using parametrized queries any input would be treated as a value (such as text), and therefore help prevent SQL injections.

Consider the example:

If a user was to enter the following on the 'new customer' page

```
"          BadSQLCode' ); DROP TABLE Customer; "
```

This would result in the following SQL statement(s)

```
INSERT INTO Customer VALUES ( 'BadSQLCode' ); DROP TABLE Customer;
```

There is the `INSERT INTO Customer` statement which is followed by a semicolon for the next statement which is to `DROP` the table. (Will, 2014)

Other issues present within SQL injections are tautologies, e.g. "1=1", which will always return true. This could be used to exploit a login script.

A valid SQL login script could be;

```
SELECT * FROM Users WHERE username = 'Rob' AND password = 'Password'
```

Using a tautology as an SQL injection results in the same statement, with the following SQL being executed, assuming "OR '1'='1'" has been used as the injection;

```
SELECT * FROM Users WHERE username = 'Rob'  
AND password = 'Password' OR '1'='1';
```

Scrambling Passwords

Passwords should not be stored as plain text, if someone was to access the database, they would see the password, and there is never really a need to see a password. There are different ways of hashing password values within the PHP library, these include; MD5, SH1, and Crypt (Guzel, 2012).

Compared to encryption, hashing would be the method of choice and can be more secure in the fact that it is one-way, with the hashed version of a password is stored in the database. When a user logs in the password entered is hashed and compared with the version in the database. If the system's users table was compromised along with email and password combinations, such as the attacks in Table 3, the hashed passwords may not be of much use without knowing how it was hashed or the different factors such as cost.

Bcrypt would be the preferred method of the hashing of passwords, this would, however, be supported in more recent versions of PHP (version 5.5.0+), however, Bcrypt is not supported

on the PHP version on the given infrastructure. One of the notable attributes of Bcrypt is that it is slow; this means it is harder to brute force passwords.

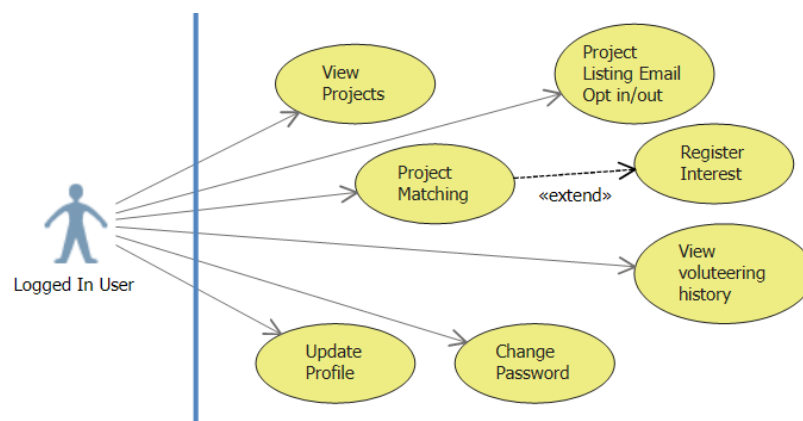
4 Technical Development

The product being developed is a system that enables volunteers to be matched to projects which are listed by charities, which is accessible via the web. In order to complete this achievement, the system needs to be planned before development to ensure all requirements are met. This section covers the planning, development, and implementation of the project.

4.1 System Design & Implementation

4.1.1 Use Case Analysis

The system being developed offers different functionalities and serves different types of users. In order to map out the top-level requirements, a use case diagram has been developed.



*Figure 3. Sample of UML Diagram showing logged in volunteer use case
The full UML diagram is in appendix item C*

Figure 3 illustrates the use case diagram for a logged in volunteer user of the system. As stated the full diagram is included in Appendix item C. The system has three main classes of users; Volunteers, Charities and System Administrators. Within these user classes are subsets of users. Charities have charity administrators and charity users, the administrators have slightly more rights than a user. System Administrators have three different levels; each higher level possessing elevated rights, for security purposes and ensuring only certain users can do certain actions.

With the different types of user, the system would need to know who they are and what type of user they are enabling different interaction of the system.

The use cases within the use case diagram derive from the aim and objectives as set in section 2.

4.1.2 Database Development

A database has been developed to store data that would be used by the system. For the reasons discussed in the Database Design section during background research, a relational database was designed, developed and implemented.

In order to successfully create a database the scope of the project was analysed and broken down into individual elements and interactions in order to decide upon the required entities and relations.

4.1.2.1 Users Table

A users table was created to hold details about a user, each registered user would start off as type 'volunteer user' which was stored in a user type attributed with a value of 1-5. The user table only holds details about the user such as; name, email, password (hashed), date of birth, and other factors such as; if they hold a valid CRB, or do they have a driving license. The purpose of which is that some projects may not be suitable if a CRB is not held, or if a potential volunteer doesn't have a driving license. Users would have to be registered to the system before being able to be matched or to apply to projects. The ID of the user is then used in the relational elements of the database.

With further consideration during development, the user roles were changed to volunteer, charity, and administrator [levels 1-3]. The justification of this was the charity role in the user table initially existed as charity user and charity admin, however a conflict arises due to the ambiguity of a user is a charity admin, but the user belongs to many charities (potently), meaning knowing which of charities the user belongs the one the user administrates is unknown. This was refined by the inclusion of the charity user table, and only holding charity levels in that table.

Further to this, with the user role, there is a potential for a conflict of interest between users who have multiple roles such as system administrator and any of the lower user classes. The conflict exists as the user has multiple levels of access, and therefore has more control over the system. Though this is more of a client issue, oppose to a development issue. One solution could be system administrative account only exist in that context and cannot hold any other role, i.e. volunteer or charity roles.

4.1.2.2 Charity Users Table

A charity user table was created as a user could work for many different charities and have different roles for different charities, thus, a many-to-many relationship potently exists between users and charities. Therefore, this table breaks this relationship. The table holds two foreign keys; one from users and another from charities.

| |
|---|
| charityUser (<u>id</u> , userid, charityid, charityrole) |
|---|

Figure 4 – Design of charity user table showing attributes

4.1.2.3 Charities Table

Charities are stored within a table which holds details about charities such as; name, charity number, and a point of contact. As projects are linked to a charity i.e. a charity has many projects.

| |
|---|
| Charities (<u>id</u> , chairtyName, charityNumber, userID, userRole, telephone, addressLine1, addressLine2, postcode, website, accredited, approvedBy, rejectionReason, timestampCreated, timestampUpdated, description) |
|---|

Figure 5 – Design of charities table showing attributes

Within the charities table is a flag to determine whether the charity has been accredited, and therefore allowed to add projects to the system, this is for ethical and safety reasons, so invalid charities cannot use the system. The accreditation is a role done by an administrator user, a role that is fulfilled by the volunteer bureau. The table would also hold who approved the charity and the user ID who created it, which if a foreign key from the user table. All of the other attributes are about the charity which would be displayed on a project listing and for the administrator to see as part of the charity request process.

4.1.2.4 Volunteer and Project; skills, interests, and locations.

A volunteer may have many skills, interests, and locations to where they can commute. Projects would look for these skills and interests, and may operate in one or multiple locations. These attributes for a project or volunteer are saved in separate tables. Meaning there is a separate table for a volunteer's; skills, interests, and locations. Likewise for projects. Each entry would have a primary key, with a foreign key for the user ID (or project ID) and a foreign key for that particular skill, interest, or location which are stored in common shared dictionary tables.

These entities break down the many-to-many relationships that live between users and projects to skills, interests, and locations.

```
volSkill (id, volID, skillID)
volInterests (id, volID, interestID)
volDistricts (id, volID, districtID)
```

Figure 6 – Design of volunteer skills, interests, and location tables.

4.1.2.5 Project and User Availability

During the volunteer sign up process, a new row is created in the table for volunteer availability using the user ID as a foreign key, the process is replicated for new project creations, however, an entry is created in a project availability table using the project ID. Included within the entity are the timeslot values with a default value of 0 (representing false) which would be updated to 1 (true) if the project or volunteer selects that time value on the web interface.

```
volAvail (id, userID,
[mon-all], [mon-am], [mon-pm], [mon-eve], [mon-night],
[tue-all], [tue-am], [tue-pm], [tue-eve], [tue-night],
[wed-all], [wed-am], [wed-pm], [wed-eve], [wed-night],
[thr-all], [thr-am], [thr-pm], [thr-eve], [thr-night],
[fri-all], [fri-am], [fri-pm], [fri-eve], [fri-night],
[sat-all], [sat-am], [sat-pm], [sat-eve], [sat-night],
[sun-all], [sun-am], [sun-pm], [sun-eve], [sun-night])
```

Figure 7 – Design of volunteer availability table

Figure 7 (above) shows the table design of volunteer availability. Each attribute represents a time value. For example; mon-am represents Monday AM (morning), mon-all represents all day Monday, and mon-eve represents Monday evening. The project availability follows the same pattern of data storage, however, the table uses project ID as a foreign key, instead of user ID.

4.1.2.6 Messages

A messaging system which is used for notifications uses a table to hold the messages. Though this hasn't been implemented in the way the development project would have liked, which is discussed in the evaluation. It has been used for notifications instead. An entry would hold who the message is from and to, which is a foreign key from the users table, along with the message content, timestamp, and a message read flag.

The exception to the sender is a system message – which is used by the system to notify a user, if for example – their charity request has been accepted or rejected. A sender with an ID of 0 is used, as an ID in a database starts at 0, therefore, wouldn't conflict with an actual user of the system.

4.1.2.7 Registering of interest

Upon viewing a project a volunteer can register their interest, this is stored within an 'apply' table to store the applications. A charity user of that project's charity can see the volunteers who have expressed interest in the project, to either accept them, or decline. Volunteers who are accepted on the project are then considered booked on, this would have been stored within a volunteer booking table. The purpose of the application system is that charities are able to administrate who is booking onto a project.

4.1.2.8 Volunteer Booking

As previously explained in the registering of interest, if a volunteer is accepted onto the project, this is then inserted into the volunteer booking table, which consists of three foreign keys; user ID, project ID, charityuser ID (which shows who accepted the user), the fourth attribute of 'attendance' is used which determines if a volunteer attended or not; which can be used to show the volunteer a history of project participation.

4.1.3 Website Development

4.1.3.1 Development

As discussed previously in the scripting languages section of comparison of technologies, this website has been produced using a mix of PHP on the server-side, and HTML as the markup language for the website layout. Bootstrap has been used as the platform the website will be developed with, as per objective 2 of the aims, *'the website should be user-friendly'*. Bootstrap is *"the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web."* (Bootstrap, 2013). Further to this interaction with the database is carried out by using SQL, with some SQL initiated by using AJAX.

4.1.3.2 Page Structure

The website serves different many different functions for the volunteer, charities, and administrator users. Along with this, there are separate pages of about the system, and volunteer and charity sign ups. For the website, a sitemap was created ensuring pages are not missed – detailing what is required. Figure 11 shows the sitemap from which pages were developed.

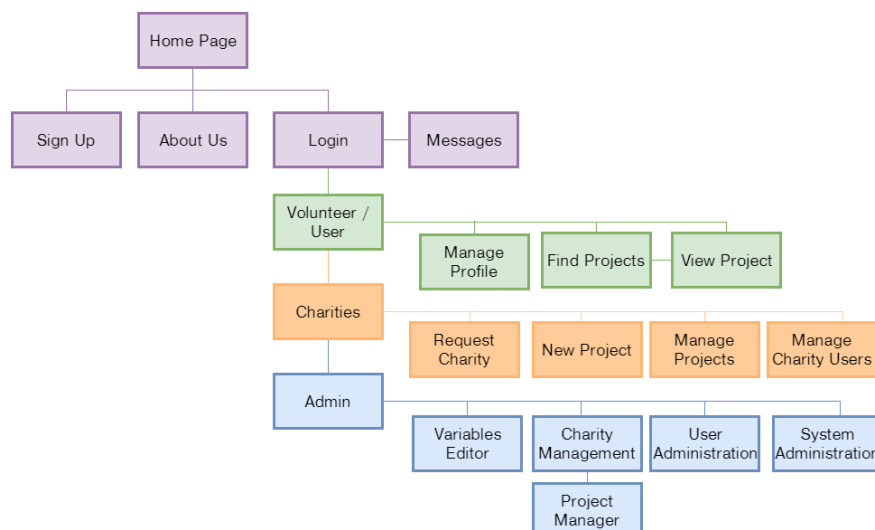


Figure 8 – Website Layout (excluding data processing pages)

4.1.3.1 Security

There have been elements of security implemented to help prevent attacks and misuse of the system.

Passwords

Upon registering with the system, passwords are hashed compared to saving them as plain text. An external library has been used for the hashing of passwords due to the development system using an older version of PHP, meaning a lack of support to use Bcrypt. (Ferrara, 2016). With the code, the cost factor can be amended which makes the hashing process slower. On one hand this causing more processing time on the server end however, there is more time involved when trying to brute force the system.

Project Management

Projects can only be deleted and set hidden/public by a charity admin user of that charity. On the user interface, a charity user would not have the option, and users who don't belong to that charity would not see the project listed to manage. Additionally, the script that processes the deletion checks the project ID and ensures the user who has processed the deletion is of type charity admin for the charity the project belongs.

Administration Console

Only users who have a user level of type 'admin' can access the administration console, and the user who is not authorized to access the administrative section of the website may have their account locked. The code behind the user test allows a toggle to automatically disable a user. However, an access denied page is always displayed.

There are different levels of administrative access, for example, bronze users are unable to delete items from the dictionaries and thus, can only add, modify, or delete items. All admin users have the ability to edit users, including levels – however, they cannot promote users to a level higher than themselves.

SQL Querying

As discussed in the security considerations, the SQL statements that are used in the system are parameterized in order to help prevent SQL injections, by use of parameters which is supplied when executing the query.

4.1.3.2 Notifications

The system has been designed to send notifications to a user. This is in replacement of sending automated emails due to infrastructure restrictions, with there being no phpmail() functionality, and the native mail function (if supported) not fully supporting authenticated SMTP in the current setup.

When the system performs an action in SQL, the result would either return true or false. The statement would be tested, to see if it was successful or not, and execute another SQL statement to insert a record in the 'notifications' table, which would show up as a message on the user were the statement was executed.

4.2 Matching Algorithm

4.2.1 Skills, Interest, and Location Matching

Each element of matching is done separately at this point. Appendix D shows the process flow diagram that was created prior to the development of the algorithm.

For instance to begin the matching process for skills; a temporary table is created which would hold the skills of the volunteer, the skills required of the project, and the project ID. A join is implemented between both sets of skills allowing the matching process. The number of distinct volunteer skills is counted and grouped with the project id, showing a sum total of skill matches for this project.

The purpose of this is that a volunteer can have many skills. Likewise, a project may require many skills – and a volunteer could have many of the skills the project looks for. Figure 12 (below) shows the first iteration of matching code for volunteer skills to projects.

```
WITH TmpSkillMatch (volSkill, ProjID, VolID)
AS (
  SELECT volSkills.skillID AS volSkill, proSkills.projectID AS
  projID, volSkills.volID AS VolID
  FROM proSkills INNER JOIN volSkills ON proSkills.skillID =
  volSkills.skillID
)
SELECT
  COUNT (DISTINCT volSkill) as SkillCount, ProjID, VolID
  FROM TmpSkillMatch WHERE VolID = 4 GROUP BY ProjID, VolID ORDER BY
  skillcount DESC
```

Figure 9 – SQL code for skill matching. Testing with user ID 4.

The interest and location matching works in the same way as the skill matching, that is the SQL queries the database for the matching attribute of the volunteer and skills, along with project ID to return how many hits of matching that volunteer has with a project. This matching would be completed for every project that is set as public.

4.2.2 Availability Matching

There have been two approaches to the implementation of matching by availability. The initial idea was to follow the same design pattern as skill, interest, and location matching. However unlike the previous implementation, the attribute table only stores an instance ID, which is not used, a user ID, and project ID - meaning only data values are stored, whereas with availability all the different time alternative are stored as columns, meaning the matching has more complexity.

Instead of comparing entries in a skill table, and counting the skill IDs for all projects limited to a user ID, then showing how many hits there are for a given project ID, i.e. number of skill matches for that project.

The availability matching would match the times a project runs and a volunteer's availability. One of the issues is that all the different time values are under separate column IDs.

```
WITH tmptimematch
AS (SELECT volavail.[mon-all] AS monall,
          proavail.projectid AS projID,
          volavail.userid AS VolID
  FROM proavail

  INNER JOIN volavail
```

```

        ON proavail. [mon-all] = volavail.[mon-all]
        WHERE ( volavail.[mon-all] = 1 ))
SELECT COUNT(DISTINCT monall) AS timematchcount,
       projid,
       valid
FROM   tmptimematch
WHERE  ( valid = 4 )
GROUP BY projid,
         valid
ORDER BY timematchcount

```

Figure 10 – SQL code for Monday-All matching Testing with user ID 4.

Figure 10 shows an SQL statement for matching of projects for a volunteer if they are available on mon-all, which is used as ‘Monday All’. There are two issues with this implementation.

The issue with the implementation as seen in Figure 13 is that it is a one-to-one matching, and would need to be replicated for every element of time matching. The system has five times of day of matching, over a week, this would equate to 35 statements if implemented in the same way.

Another method of availability matching was implemented by querying the database for the current volunteer’s availability and querying the availability of all active projects. PHP was then used to carry out the matching. Figure 14 (below) shows the process of the matching.

```

Get availability of all projects
for every project
{
    Store all time values in variables
    Get availability of the current user
    {
        Compare project availability with user availability
        If match or user availability is all-day on day of project
        {
            Increase availability hit counter for given project
        }
        If counter is > 0
        {
            Insert counter value into array using project ID as key
        }
    }
}
Sort array by counter value (high-to-low)
For every item in array
{
    add project to the table
}
Display match result

```

Figure 11 – Pseudocode for matching using PHP

Figure 11 (above) uses SQL and PHP for the matching, one of the advantages is that only two SQL statements are ever executed, which could be more efficient than N queries being executed at the RDBMS; in this example only two queries are executed (highlighted in blue) compared to a potential 36 statements being executed with further processing on the database if matching was to be implemented in the statement.

4.2.3 Algorithm Refinement – Hidden Projects

When a project is created by a charity, by default the project is not set public to allow the proofing of the listing. Additionally, charities may want to hide their project if they get full of members, expire, or canceled. The first iteration of the algorithm did not take this into consideration and still match

The algorithm was further developed to only match projects which are listed as publically available. Furthermore, with a join on charity ID to link to the charities table, allowed the system to show details about the charity, such as the name of the charity.

This was made possible by including `project.name` and `charities.charityName` within the second SELECT statement, with an additional inner join on the charities table on the `project.charityID` attribute to the ID of the charities table.

To ensure volunteers are not matched to projects that are not visible to the public, i.e. hidden or expired. Within the second WHERE condition is a check for the project's publicity. The status of the public flag is stored as an integer, with 1 equaling true. 'Public' is a keyword used in SQL, as such to implicitly tell SQL that it is an attribute, square brackets are used. The code used to ensure projects are public is: `(project.[public] = 1)`. This was then included in all the different matching algorithms.

4.2.4 Partial Matching

Partial matching uses the individual matching algorithms and sums the totals for each different measure of matching. When a volunteer has been matched, the algorithm tests to see if the project exists in the array of suitably matched projects; if it is not – the project is added into the array, else the total count is added to the current count in the matched projects array.

4.2.5 Strict Matching

Thought strict matching was not implemented, which will be discussed in the evaluation, the implementation of strict matching was planned following partial matching. The difference between partial and strict matching would have been; for each element of matching (availability, skill, interest, or location) for a given project if there are no matches (hits), that project would not be listed to the volunteer.

The plan would have been to use a multidimensional array, a multidimensional array is an array which contains arrays.

The approach would allow rows of matched projects with columns holding the number of matches. The system would display all the projects the volunteer matched in a tabular format, similar to the single or partial matching. However, it would display the individual counts, allowing the volunteer to see how much of a close hit the project is.

Figure 12 (below) shows how this could be implemented.

```
$suitableProjects = array ( array($projectID, $sCount, $iCount, $lCount, $aCount));
```

Figure 12 – Example of multidimensional array

4.2.6 Testing

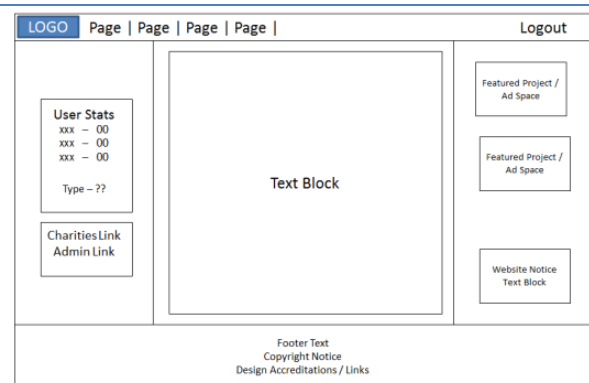
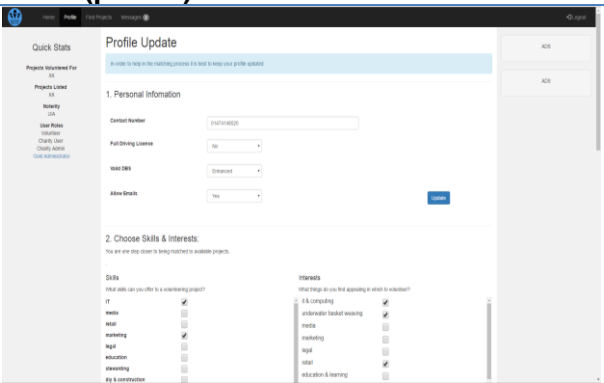
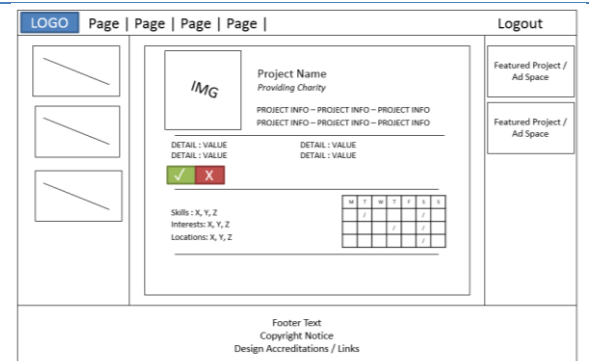
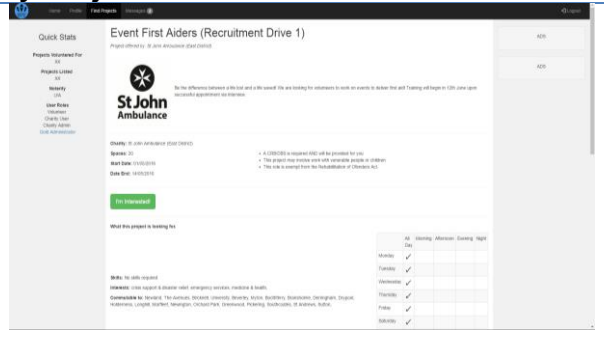
During the development process test data has been used in the system to create dummy-users and dummy-projects in order to test the algorithms during development, but also to test functions within the website; such as creation and management of projects, or creation of charities. The purpose of testing during development of the algorithm is to ensure it works as expected.

This was carried out by working on individual matching, such as skills, then amending a user's skills to suit the skills of created projects, or existing projects, where skills may vary to ensure the matching works as it should.

4.3 UI Design

Different user classes fulfil different roles on the website and there are different interfaces for each of these roles (volunteer, charities, and administrations) each of which was designed and implemented.

As the aim of the website is creating the matching system viewable via the WWW, it seems appropriate to use a web framework which helps conform to browser standards and cross-device compatibility, ensuring all users get a good user experience. This is why Bootstrap was chosen to aid the development of the website front end, due to limited experience and skills in web design.

| Wireframe Design | Implementation |
|---|--|
| Volunteer Interface (profile) | |
|  |  |
| View Charity Project | |
|  |  |
| Project Management (project listings) | |

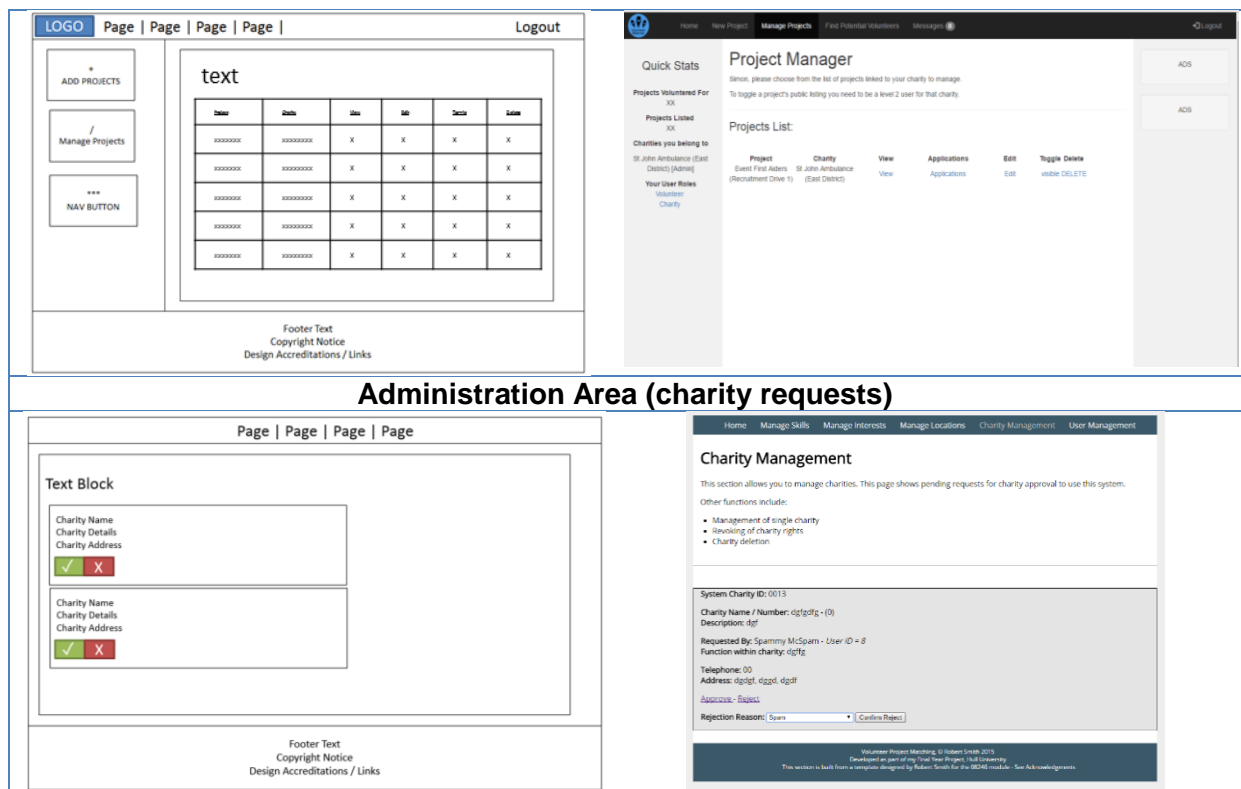


Table 4 – UI designs of different user views

Table 4 show the first iteration of the user interface using a basic bootstrap theme, however, the administration area uses a basic HTML template created manually, in order to differentiate between user types. In practice, this has worked, however, the screenshots show a full-screen width, the advantage with using Bootstrap is the scalability for smaller-screen devices.

4.4 Data Dictionary Performance

The three entities of skills, interests and districts are used by the system in the matching process. Volunteers would have a set of skills, interests, and districts (places that are commutable) and projects would look for them in a volunteer.

The design of the database is that those entities are stored in separate tables, with the data split into logical collections, compared to having one master table. The justification of this is that it may be theoretically easier to tell what data is what. For example:

`“SELECT skill FROM skills”`

This would get all the skills from a skills table.

Compared to a one table dictionary, where the statement may be

`“SELECT name FROM dictionary WHERE type = 1”`

Another aspect is that as the data in the dictionary grows, there is a potential for statements taking longer to execute.

| Table | No of Records | Attributes | Execution Time (milliseconds) | | | | Query |
|-----------|---------------|----------------------------------|-------------------------------|-------|-------|------|--------------------------------|
| | | | Run 1 | Run 2 | Run 3 | Mean | |
| Skills | 25 | 3 (id, skill, description) | 0 | 0 | 0 | 0 | SELECT skill FROM skills |
| Interests | 37 | 3 (id, interest, description) | 0 | 0 | 0 | 0 | SELECT interest from interests |
| Districts | 21 | 3 (id, district, description) | 0 | 0 | 0 | 0 | SELECT district FROM district |

Table 5 – Execution of SQL statements based on current database values.

Though table 3 has limited data, a C# script was developed to insert data into a temporary table to test whether it is quicker and therefore more efficient to use a single table data dictionary or three entities.

To create a single table solution in order to test this, a script was created which would enter data into the database. The structure of which was:

`dataDictionary (id, item, type, description)`

The difference between the location, skills and interest tables and the data dictionary is the additional 'type' attribute which would be a 1-3 value to represent what type of data it is.

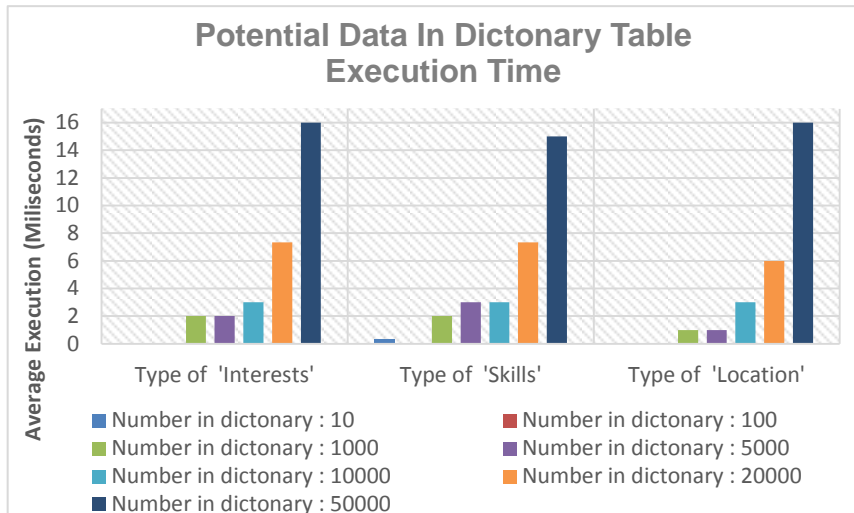


Figure 13 – Execution time of a single table solution

a skill, interest or location table. The execution of which can be seen in figure 14.

From figure 6, if there were 16668 entries in a location table, for example - it would have taken 12ms to generate a list of values.

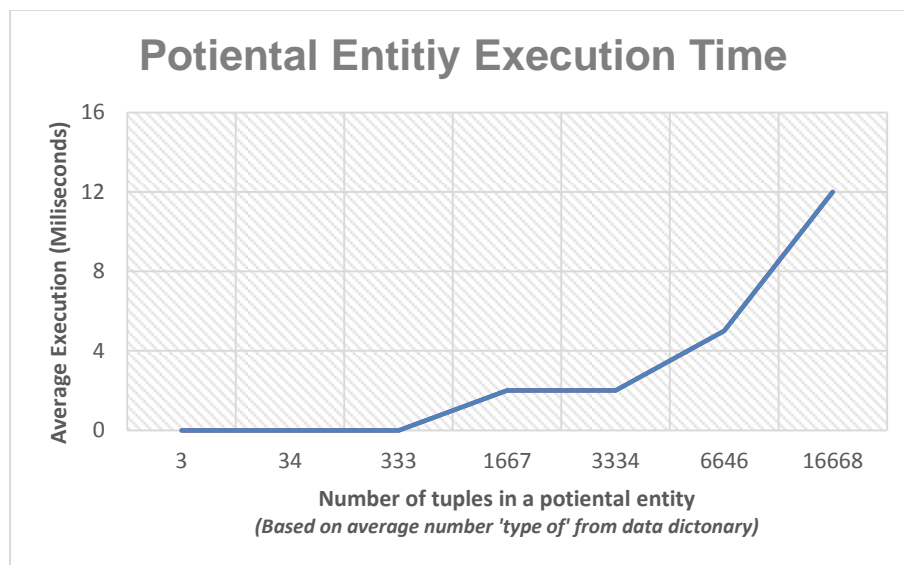


Figure 14 – Execution time a single type attribute

1000 entries in a data dictionary to split the data into logical entities (assuming an average of 333 tuples) of a particular type. Appendix E shows the full results from the experimentation, including code.

Figure 13 shows the mean time it took an SQL statement which returns a list of values. This was tested with different numbers in the data dictionary.

Entries and a value type were randomly generated and inserted into the dictionary up to 50,000 values.

The average of the value type was used to generate a table which would represent either

Compared to a data dictionary which may have 50,000 entries of which 16549 were skills which took an average of 16ms to generate a list.

Though a location table of 16549 could be argued as excessive, the experiment has shown that it is more efficient in terms of execution time once there are

5 Evaluation

5.1 Project Achievements

There was a total of seven primary objectives and two secondary objectives. The subsection will evaluate whether the objectives were achieved in order to meet the overall aim of the project. Table X below shows a visual representation of the project objective overall, with each objective discussed individually in subsections.

| Objective | Achieved? | Notes |
|-----------------------------|-----------|--------------------------------------|
| 1 | Partial | Achieved most elements. |
| 2 | Yes | |
| 3 | Yes | |
| 4 | Yes | |
| 5 | Partial | Most functions implemented |
| 6 | Yes | Pictures cannot be changed |
| 7 | Yes | |
| Secondary Objectives | | |
| 8 | No | Secondary objective, not implemented |
| 9 | Yes | Now used as a notification centre |

Table 6 – Overall project achievements

Objective 1

Objective 1 was to create the volunteer matching algorithm. The deliverables for was to develop a matching algorithm and to display a list of suitable projects to the volunteer. In order to achieve these different elements of matching were created along with a 'generic match' which counts the total number of hits, with a 'strict matching' where there must be at least one hit for every element of matching for a volunteer to a project.

| Objective | Achieved? | Notes |
|-----------------------|-----------|---|
| Skills Matching | Yes | Volunteers are matched to projects based on skills, showing a total count of matches |
| Interest Matching | Yes | Volunteers are matched to projects based on interests, showing a total count of matches |
| Location Matching | Yes | Volunteers are matched to projects based on locations they can commute to |
| Availability Matching | Yes | <ul style="list-style-type: none"> - Volunteers are matched to projects based on their location availability - Matching is done using PHP instead of SQL. |
| Overall Matching | Yes | Volunteers are matched to projects with A total count of hits for; skills, interests, locations, and availabilities. |
| Strict Matching | Partial | Due to time constraints strict matching was not completed, though the overall matching forms a very good basis for strict matching. |

| | | |
|----------------|-----|---|
| Project Output | Yes | Where there is matching the volunteer is shown projects applicable for the matching done. |
|----------------|-----|---|

Table 7 – Objective 1 Achievements

Unfortunately, strict matching was not fully implemented into the system due to time constraints. More time than predicted was used for availability matching due to the design of the database, which stored the data for better efficiently instead of the matching process, along with restrictions in the SQL programming language.

In order for the strict matching to be completed, the array that is used in ‘overall matching’ which stores the project ID as a key and the value as the number of hits, should instead store separate values for each element of matching using a multidimensional array, and throw any project away that has a value of 0 indicating no match for that element.

Objective 2

Objective 2 was to create a database which would store all data that would be used by the system. This objective was fully implemented and suitably normalized. There was an issue with getting date values to store in the database which for dates used the DATE format. This issue was resolved by using the “createFormFormat” function in PHP.

The database development was done using Microsoft SQL Server 2008, purely due to infrastructure restrictions, given this is a platform where experience was limited, there was a successful implementation following suitable research.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|-------------------------|------------------|--|
| Database Design | Yes | The database was suitably planned by finding requirements and creating an ERD. |
| Normalisation | Yes | The database has been normalized to a suitable normal form, this has been carried out throughout the planning of the database structure. |
| Database Implementation | Yes | The database has been implemented and is used by the system successfully. |

Table 8 – Objective 2 Achievements

Objective 3

Objective 3 was to implement a user interface to ensure the system was usable via the World Wide Web. The website would be suitably planned.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|-----------------------------|------------------|--|
| Website Interface Designs | Yes | The layout for the different types of users has forms of design paperwork |
| Bootstrap Implementation | Yes | Bootstrap has been used for the development of the website layout with a basic template used and sourced to ensure website usability |
| Can users fulfil their role | Yes | Each user class has the options that ensure they can fulfil their role |

Table 9 – Objective 3 Achievements

Objective 4

The focus of objective 4 is on security as the aim of the projects states “*Taking into account security provisions*” In order to achieve the security element a number of different tasks were involved which are evaluated below.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|---------------------------|------------------|---|
| Safeguarding of passwords | Yes | Passwords are hashed using BCrypt |
| Authentication to system | Yes | Users are required to log in to the system to access different areas |
| Authorisation | Yes | There are 5 levels of access ranging from volunteer to gold administrator (super-admin). Different sections of the website or functions require a higher level of access. Thus, would be unavailable to someone less than the required level. |
| Query parameterization | Yes | Queries are written using parameters in order to prevent injections and to keep data and the integrity of data safe. |

Table 10 – Objective 4 Achievements

Objective 5

The initial idea was to create the system as a package whereby a volunteer bureau could install the system package and manage volunteers, charities, and the system themselves. Unfortunately, this was not implemented, however, most administrative functions were implemented into the system, allowing for administration.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|-------------------------|------------------|---|
| Management of skills | Yes | Skills can be edited, added, and removed. |
| Management of interests | Yes | Interests can be edited, added, and removed. |
| Management of locations | Yes | Locations can be edited, added, and removed. |
| Charity Management | Partial | Charities can be approved or rejected. However, they cannot be edited, deleted, or have accreditation revoked |
| User Management | Yes | User levels can be changed, along with the disabling and enabling of accounts, and password resets. |

Table 11 – Objective 5 Achievements

Objective 6

Projects that are listed would have the option to be managed by a user or administrator of that charity who listed the project.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|----------------------|------------------|---|
| Editing of project | Yes | Project attributes can be edited and saved |
| Change visibility | Yes | Projects can be hidden from public listing and therefore excluded from matching |
| Deletion | Yes | Projects can be deleted |

Table 12 – Objective 6 Achievements

Objective 7

Once a volunteer is matched, or has found a project which they would like to participate in, there is an option on the project listing to register interest. From this, a charity user would be able to view all interested volunteers and view their profile, which shows their details and past participations. The charity user then has the option to accept or turn down the volunteer, who is then notified.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|-----------------------------------|------------------|--|
| Registering interest in a project | Yes | Volunteers can register and remove their interest in a project. |
| Expected Volunteers | Yes | Within the project management options, there is an option to view volunteers who are booked onto the project; with an option to remove volunteers. |
| Project History | Yes | Volunteers can view projects that they have been approved to participate in on their profile. |

Table 13 – Objective 7 Achievements

Objective 8 (Secondary)

Due to time constraints, this objective was not implemented. The idea was for a charity to download the project data in the form of XML, with the idea of if another party used this system the charity could import the project using XML to save having to manually list another project.

Objective 9 (Secondary)

The objective has changed from a messaging functionality in the sense of messaging from one user to another (like an email) and changed into notification function. There are two main reasons for this;

1. Time constraints, due to development time running out and with this being a secondary objective, the development of this objective took another slant, which was further justified on the basis of infrastructure limitations of the given PHP version meaning the mail() function, which allows the sending of emails, is not supported.
2. Privacy and spam prevention. Some users may not appreciate unsolicited messages from other users, however, this could be avoided by only allowing emails to be sent by volunteers to charity staff of whose projects they are interested in.

| <u>Sub-Objective</u> | <u>Achieved?</u> | <u>Notes</u> |
|-------------------------------------|------------------|---|
| Notifications Centre Implementation | Yes | The system can send notifications to all of users of system based on actions such as; <ul style="list-style-type: none">- Adding / Removing to projects- Account level changes- Charity approval / rejection. |

Table 14 – Objective 9 Achievements

5.2 Further Development

There always exists some scope for further enhancement of the project, within further adaptations and modifications will be discussed which would make the project more superior.

5.2.1 Strict Matching

Due to time restrictions, strict project matching was not fully implemented. Given more time, with this feature implemented volunteers would find projects that exactly matches their given skills, interests, locations, and availability. Instead of 'Close Match' which matches volunteer to projects on an 'OR' basis- considering if a project matched volunteers skills, locations, and interests exactly, yet the project ran on a day they are not available, then the volunteer may not be able to take part. Although they could rearrange their schedules, offering a strict matching allows an explicit match. This would be the first priority given refinements.

5.2.2 Availability Issues (DayX-All)

For each day, one of the times-of-day options is 'all' which implies available/runs all day. For example, volunteers have the ability to select 'Monday-All' as an availability option, this was a system design issue and was stored in the database. This is an example consequence of how the user interface operates is affecting how the system operates, which caused additional work in creating then refining the availability matching algorithm when this problem came to light.

Given additional time to work on the project, the storing of *Day-All* would be removed, although the user interface option to select 'all day' would remain to a volunteer or the 'add project screen', instead upon selection the user interface would select all options time-of-day options for that day.

The extra work that the algorithm entailed is an extra test to see if all-day is selected along with each time-of-day element. Consequently, there is extra processing happening on the server end by the PHP, and furthermore, for each project and for each user there are seven additional data values being held in the database.

5.2.3 Potential reach

One of the initial ideas was to match projects to volunteers, which would allow charities to see who match projects. However, this was not implemented because of time, moreover the issue of privacy concerns and whether volunteers would want notifications for every project they are a match for, although there is an option for this (if such feature was implemented), a volunteer could have issues with details coming up to the charity about themselves. On way around this could have been to implement a 'potential reach' count, allowing the charity to see their potential success, allowing them to advertise or promote their project their own way.

5.2.4 Improvements to administration console

There is a lot of doing back and forth between the page of users which shows their user ID and email which can be used for changing details such as; user type, password resets, or disabling / enabling of accounts. This could be improved by allowing an administrator to search for users instead, then being able to click on the user to then managing the user from a single page.

Further to this, although the tables were created, security logs were not implemented. Given extra time, they would be implemented and any logs created would be shown to an appropriate level administrator with actions; such as disable a user, or even ban an IP address. The banning of an IP address altogether could be done using PHP, or if the web server runs apache, it could be done by altering the .htaccess file.

5.2.5 General User Interface Improvements

Most development time has been focused on enabling the matching process, whether this is in the form of algorithm development, data gathering, or providing functionality. Some interface elements have been overlooked and could be made more user-friendly.

5.3 Personal Reflection

Most features were implemented for the project and therefore it could be argued this project was a partial success, however if the strict matching was implemented the project would have been successful, albeit some enhancements could have been implemented.

Never before has a project of this size had been undertaken and the successes of most objectives were met through planning what tasks needed to be completed and giving sufficient time to complete them, including adding contingency time.

During this project lessons were learned in time planning, and that some tasks may take longer or during development new problems may be found; as a result of this the initial time plan did change due to underestimation of task lengths, but the planned contingency time from the changes has allowed for a project with suitable deliverables. Both of these time plans are available in Appendices E (Initial) and F (Revised).

During development, one such problem did arise with the matching of availability with regards to how the data is stored in the database. It is stored in a logical fashion i.e. captured as the user enters their availability and stored in one table. Which had been problematic when trying to match using SQL, however, this was overcome by use of PHP.

Security is always something to be aware of, especially in the development of a web system, the experience of ensuring elements are kept secure, especially in the use of parametrized queries – and finding out what would happen if they were not used, resulting in SQL injections. It has been interesting and exciting in testing and securing the system.

6 Conclusion

Overall the project to create a volunteering matching system has been regarded as a partial success in regards to the aims and objectives set out. One of the downfalls of the project is not fully implementing the 'strict matching' of volunteers to projects.

Most primary objectives were achieved, with two partially achieved. For the secondary objectives one was not achieved and the other was achieved.

A user is able to sign up to the system, and find a project – based on skills, interests, locations, and their availability. However if one of them is not true, e.g. for a given project the volunteer cannot commute to that location, instead of being thrown away, it will still be listed as a suitable project. Therefore there is partial matching and individual matching.

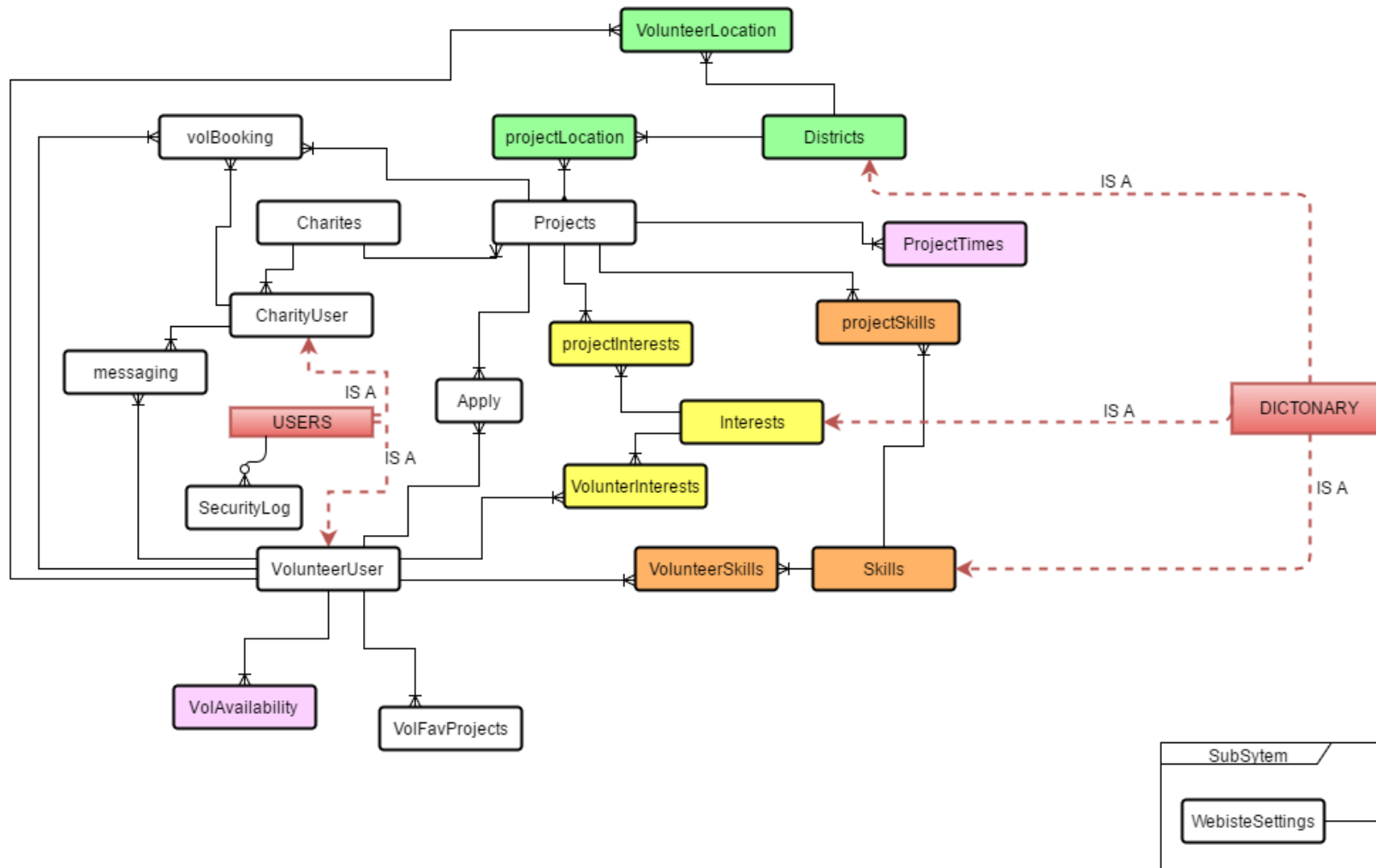
For a charity, a new charity can be requested which is then approved by an administrator; who would be the customer of this system, such as a volunteering bureau, once approved the charity can begin adding projects which can be found by volunteers.

The overall aim of this project is a system that is capable of matching, which this system is almost there. The final step would be refining the strict match algorithm.

Part of the reason for the non-implementation is time, however from undertaking this project – lessons have been learned which can be taken on in future projects regarding time management and scope creep.

If a project was to be undertaken again, focus would be on ensuring all objectives are SMART (Specific, Measurable, Attainable, Relevant, Timely) the purpose of doing that is to ensure the project doesn't run into scope creep, and when the objectives are broken down into tasks, they are not underestimated as things can go wrong.

Appendix A: Database Entity Relationship Diagram



Appendix B: Database Attributes

| Apply Table to hold project applications | | |
|--|-------------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| userID | int | Foreign Key to Users.ID |
| projectID | int | Foreign Key to Project.ID |

| Charities Table to hold information about charities | | |
|---|---------------|-------------------------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| charityName | varchar(50) | |
| charityNumber | int | |
| userID | int | Foreign Key to Users.ID |
| userRole | varchar(50) | Applicant job role in charity |
| telephone | bigint | |
| charityAL1 | varchar(50) | |
| charityAL2 | varchar(50) | |
| postcode | varchar(8) | |
| website | varchar(100) | |
| accredited | int | |
| approvedBy | int | Foreign key on user.id |
| rejectionReason | varchar(50) | (Nulls allowed) |
| timestampcreated | smalldatetime | |
| timestampupdated | smalldatetime | |
| description | text | |

| Users Table to hold information about a user | | |
|--|--------------|--------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| fname | varchar(50) | |
| sname | varchar(50) | |
| password | varchar(100) | |
| accountType | int | |
| email | varchar(100) | |
| gender | varchar(7) | |
| dob | date | |
| contactNo | varchar(25) | |
| accStatus | int | |
| drivingLicence | int | |
| validDBS | int | |
| allowEmails | int | |
| timestamp | datetime | |

| charityUsers Table to hold users who work for charities (M:M breakdown) | | |
|---|------|-----------------------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| userid | int | Foreign Key to Users.ID |
| charityid | int | Foreign Key to Charities.id |
| charityrole | int | |

| District Table to hold dictionary of locations | | |
|--|--------------|-----------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| district | varchar(50) | |
| description | varchar(100) | (Nulls allowed) |

| Skills Table to hold dictionary of skills | | |
|---|-------------|-----------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| skill | varchar(50) | |
| description | varchar(75) | (Nulls allowed) |

| Interests Table to hold dictionary of interests | | |
|---|--------------|-----------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| interest | varchar(50) | |
| description | varchar(100) | (Nulls allowed) |

| Messages Table to hold messages sent in the system | | |
|--|--------------|------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| recipient | int | Foreign key on user.id |
| sender | int | Foreign key on user.id |
| message | varchar(200) | |
| msgread | int | |
| timestamp | datetime | |

| Project Table to hold projects | | |
|--|--------------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| charityID | int | Foreign key on charity.id |
| name | varchar(100) | |
| details | text | |
| nopers | int | |
| DBSCheck | int | |
| Vulnerable | int | |

| | | |
|----------------|--------------|--------------------|
| DrivingLicence | int | |
| ROAExempt | int | |
| pstart | date | |
| pend | date | |
| img | varchar(200) | |
| addedby | int | Foreign on user.id |
| [public] | int | |
| featured | int | |

| | | |
|------------------------------|-------------|---------------------------|
| proAvail | | |
| Availability data of project | | |
| Attribute | Type | Notes |
| Id | int | Primary Key |
| projectID | int | Foreign key on project.id |
| [mon-all] | int | |
| [mon-am] | int | |
| [mon-pm] | int | |
| [mon-eve] | int | |
| [mon-night] | int | |
| [tue-all] | int | |
| [tue-am] | int | |
| [tue-pm] | int | |
| [tue-eve] | int | |
| [tue-night] | int | |
| [wed-all] | int | |
| [wed-am] | int | |
| [wed-pm] | int | |
| [wed-eve] | int | |
| [wed-night] | int | |
| [thr-all] | int | |
| [thr-am] | int | |
| [thr-pm] | int | |
| [thr-eve] | int | |
| [thr-night] | int | |
| [fri-all] | int | |
| [fri-am] | int | |
| [fri-pm] | int | |
| [fri-eve] | int | |
| [fri-night] | int | |
| [sat-all] | int | |
| [sat-am] | int | |
| [sat-pm] | int | |
| [sat-eve] | int | |
| [sat-night] | int | |
| [sun-all] | int | |
| [sun-am] | int | |
| [sun-pm] | int | |
| [sun-eve] | int | |
| [sun-night] | int | |

| proDistrict Table to hold where projects run | | |
|--|------|------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| projectID | int | Foreign on project.id |
| districtID | int | Foreign on district.id |

| proInts Table to hold project interests | | |
|---|------|---------------------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| projectID | int | Foreign key on project.id |
| interestID | int | Foreign on interest.id |

| proSkills Table to hold project skills | | |
|--|------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| projectID | int | Foreign key on project.id |
| skillID | int | Foreign key on skill.id |

| securityLog Table to hold security messages | | |
|---|---------------|------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| userID | int | Foreign on user.id |
| IPAddress | varchar(100) | |
| level | int | |
| alertMsg | varchar(50) | |
| timestamp | smalldatetime | |
| actioned | int | Foreign key on user.id |

| volDistrict Table to hold where volunteers can commute | | |
|--|------|----------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| volID | int | Foreign on user.id |
| districtID | int | Foreign key on district.id |

| volInts Table to hold volunteer interests | | |
|---|------|----------------------------|
| Attribute | Type | Notes |
| id | int | Primary Key |
| volID | int | Foreign key on user.id |
| interestID | int | Foreign key on interest.id |

| volSkills Table to hold volunteer skills | | |
|--|------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| projectID | int | Foreign key on project.id |
| skillID | int | Foreign key on skill.id |

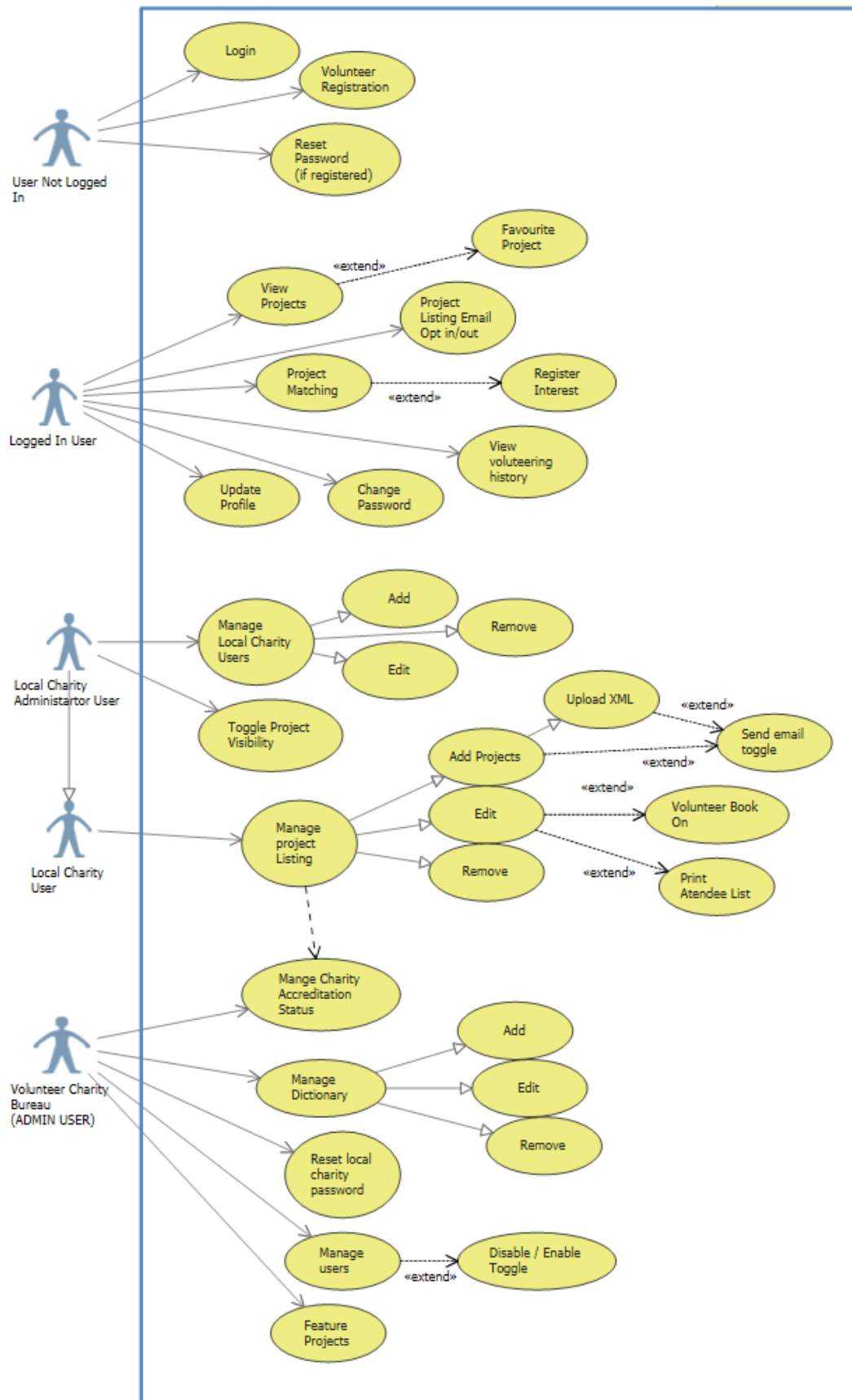
| volBooking Table to hold volunteers that are booked on projects | | |
|---|------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| userID | int | Foreign key on user.id |
| projectID | int | Foreign key on project.id |
| approvedBy | int | Foreign key on user.id |

| volFavProjects Table to hold projects volunteers have favorited | | |
|---|------|---------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| userID | int | Foreign key on user.id |
| projectID | int | Foreign Key on project.id |

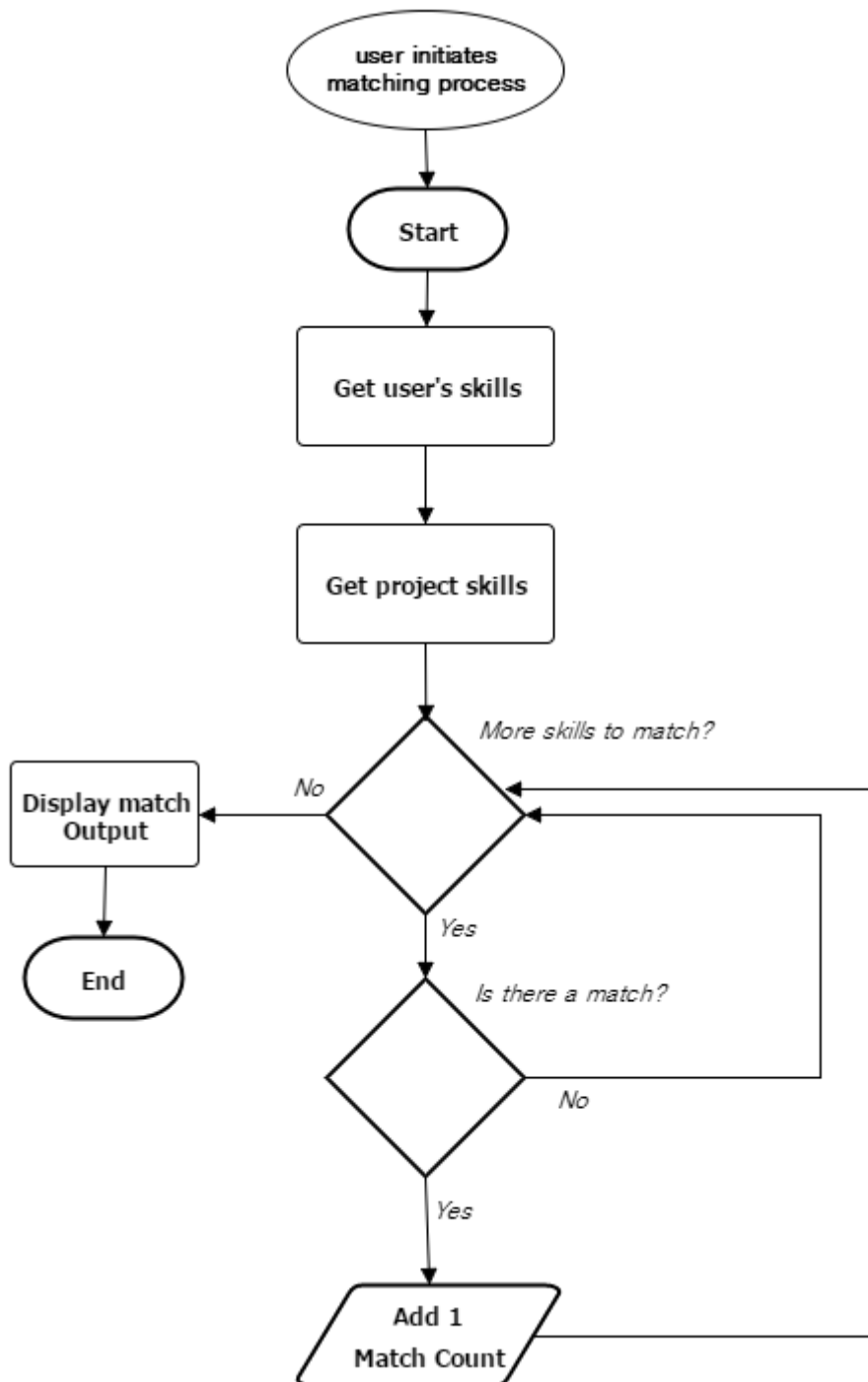
| volAvail Availability data of volunteer | | |
|---|------|------------------------|
| Attribute | Type | Notes |
| Id | int | Primary Key |
| userID | int | Foreign key on user.id |
| [mon-all] | int | |
| [mon-am] | int | |
| [mon-pm] | int | |
| [mon-eve] | int | |
| [mon-night] | int | |
| [tue-all] | int | |
| [tue-am] | int | |
| [tue-pm] | int | |
| [tue-eve] | int | |
| [tue-night] | int | |
| [wed-all] | int | |
| [wed-am] | int | |
| [wed-pm] | int | |
| [wed-eve] | int | |
| [wed-night] | int | |
| [thr-all] | int | |
| [thr-am] | int | |
| [thr-pm] | int | |
| [thr-eve] | int | |
| [thr-night] | int | |
| [fri-all] | int | |
| [fri-am] | int | |

| | | |
|-------------|-----|--|
| [fri-pm] | int | |
| [fri-eve] | int | |
| [fri-night] | int | |
| [sat-all] | int | |
| [sat-am] | int | |
| [sat-pm] | int | |
| [sat-eve] | int | |
| [sat-night] | int | |
| [sun-all] | int | |
| [sun-am] | int | |
| [sun-pm] | int | |
| [sun-eve] | int | |
| [sun-night] | int | |

Appendix C: Use Case Diagram



Appendix D: Algorithm Flow Diagram Design



Appendix D shows the process flow for the individual elements matching algorithm.

The algorithm designed shows how skills would be matched for a given project; this would be replicated for each project in the system.

Although the diagram above shows as 'skills', this is also how the algorithm works in the matching of interests and locations.

Appendix E: Database Efficiency Testing

Single Table Dictionary VS Logical Entity Split

| <u>Type of 'Interests'</u> | | | <u>Type of 'Skills'</u> | | | <u>Type of 'Location'</u> | | | <u>Using AVG No of tuples</u> | | | |
|---|---------------------|------|-------------------------|------------------|------|---------------------------|--------------------|------|-------------------------------|--------------|------|------|
| <u>Number in dictionary : 10</u> | | | | | | | | | | | | |
| Average No of tuples 3 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 3 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 0 | | 3 | 0.333333333 | | 3 | 0 | | 4 | 0 | | |

| | | | | | | | | | | | | |
|--|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| <u>Number in dictionary : 100</u> | | | | | | | | | | | | |
| Average No of tuples 34 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 34 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 0 | | 37 | 0 | | 33 | 0 | | 33 | 0 | | |

| | | | | | | | | | | | | |
|---|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| <u>Number in dictionary : 1000</u> | | | | | | | | | | | | |
| Average No of tuples 333 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 333 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 3 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 0 | 0 | 0 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 2 | | 329 | 2 | | 323 | 1 | | 348 | 0 | | |

Number in dictionary : 5000

| | | | | | | | | | | | | |
|------------------------------|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| Average No of tuples 1667 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 1667 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 3 | 0 | 3 | 3 | 3 | 3 | 0 | 3 | 0 | 3 | 3 | 0 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 2 | | 1671 | 3 | | 1614 | 1 | | 1715 | 2 | | |

Number in dictionary : 10000

| | | | | | | | | | | | | |
|------------------------------|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| Average No of tuples 3334 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 3334 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 3 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 3 | | 3366 | 3 | | 3300 | 3 | | 3337 | 2 | | |

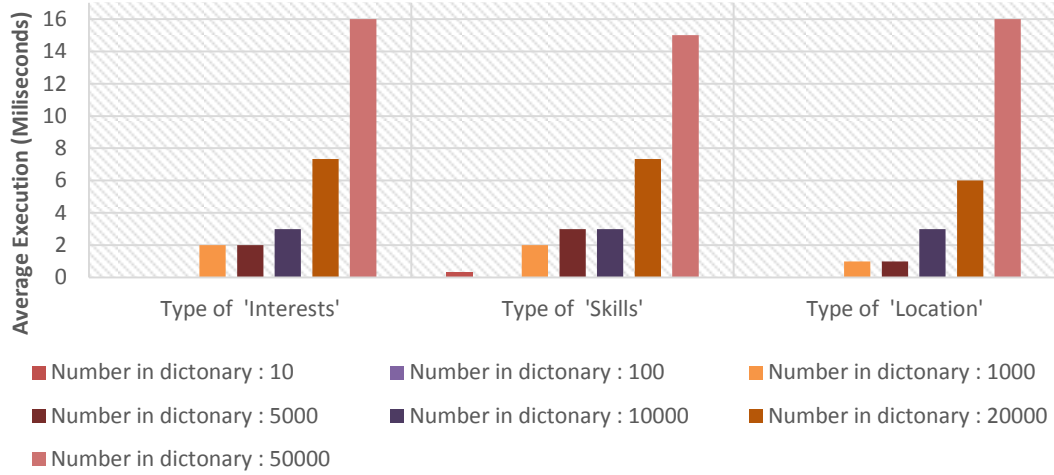
Number in dictionary : 20000

| | | | | | | | | | | | | |
|------------------------------|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| Average No of tuples 6646 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 6646 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 6 | 10 | 6 | 6 | 10 | 6 | 6 | 6 | 6 | 6 | 3 | 6 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 7.3 | | 6749 | 7.3 | | 6621 | 6 | | 6567 | 5 | | |

Number in dictionary : 50000

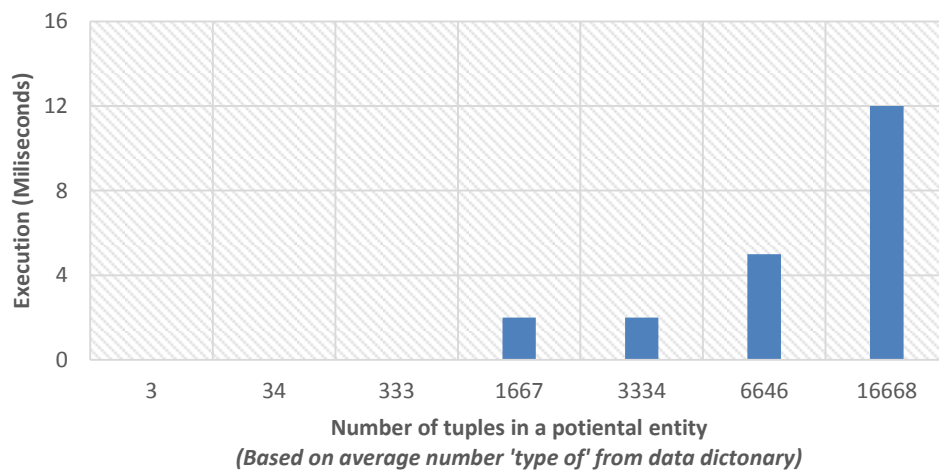
| | | | | | | | | | | | | |
|-------------------------------|---------------------|------|---------|------------------|------|---------|--------------------|------|---------|--------------|------|------|
| Average No of tuples 16668 | Type of 'Interests' | | | Type of 'Skills' | | | Type of 'Location' | | | 16668 | | |
| | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 | run1 | run2 | run3 |
| | 16 | 16 | 16 | 16 | 16 | 13 | 16 | 16 | 16 | 13 | 13 | 10 |
| | Average (ms) | | results | Average (ms) | | results | Average (ms) | | results | Average (ms) | | |
| | 16 | | 16735 | 15 | | 16719 | 16 | | 16549 | 12 | | |

Potential Data Dictionary Table Execution Time



| Number of tuples in data dictionary | Type Of | | |
|--|---------|----------|------------|
| | Skill | Interest | Dictionary |
| 10 | 3 | 3 | 4 |
| 100 | 37 | 33 | 33 |
| 1000 | 329 | 323 | 348 |
| 5000 | 1671 | 1614 | 1715 |
| 10000 | 3366 | 3300 | 3337 |
| 20000 | 6749 | 6621 | 6567 |
| 50000 | 16735 | 16719 | 16549 |

Relational Database Execution Time



Code used for generation of dummy records.

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;

namespace testAdd
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("SQL Test Data Adding");
            Console.Write("How Many Entries? : ");
            int values = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("\n\n");

            string[] letters = new string[] { "a", "b", "c", "d", "e", "f", "g", "h",
            "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y",
            "z", };

            string word = string.Empty;
            Random rnd = new Random();
            for (int i = 0; i < values; i++) //SQL statement
            {
                int length = rnd.Next(5, 15); //random word length

                //ATTRIBUTE WORD BUILDER
                for (int j = 0; j < length; j++) //for the word length
                {
                    int random = rnd.Next(0, 25); //random number 0-25 (alphabet
array)

                    word = word + letters[random]; //add random letter to word
                }

                int type = rnd.Next(1, 4); // random number value for DB.

                Console.WriteLine("INSERT INTO masterDic (item, type) VALUES ('" +
word + "', '" + type + "')");

                /*
                Start SQL
                EXTERNAL SOURCE FROM:
                Woo, J. (2012). How to insert data into SQL Server. [online]
Stackoverflow.com.
                Available at: http://stackoverflow.com/a/12241118 [Accessed 17 Apr.
2016].

                Modifications:
                - Storing data variables specific to project
                - Message shown if successful execution
                */

                using (SqlConnection connection = new SqlConnection("Data
Source=sql.net.dcs.hull.ac.uk;Integrated Security=True;database=rde_468566"))
                {
                    using (SqlCommand command = new SqlCommand())
                    {
                        command.Connection = connection;
                    }
                }
            }
        }
    }
}
```

```

VALUES (@item, @type)";
        command.CommandText = "INSERT into masterDic (item, type)
VALUES (@item, @type)";
        command.Parameters.AddWithValue("@item", word);
        command.Parameters.AddWithValue("@type", type);
        try
        {
            connection.Open();
            int recordsAffected = command.ExecuteNonQuery();
            if (recordsAffected == 1)
            {
                Console.WriteLine("Execution success!");
                Console.Beep();
            }
        }
        catch (SqlException)
        {
            Console.WriteLine("FAILED!");
        }
        finally
        {
            connection.Close();
        }
    }
}
/* END OF SOURCED CONTENT */

word = string.Empty;
}
Console.WriteLine("\n*** End of execution ***");
Console.Beep();
Console.Read();
}
}
}
}

```

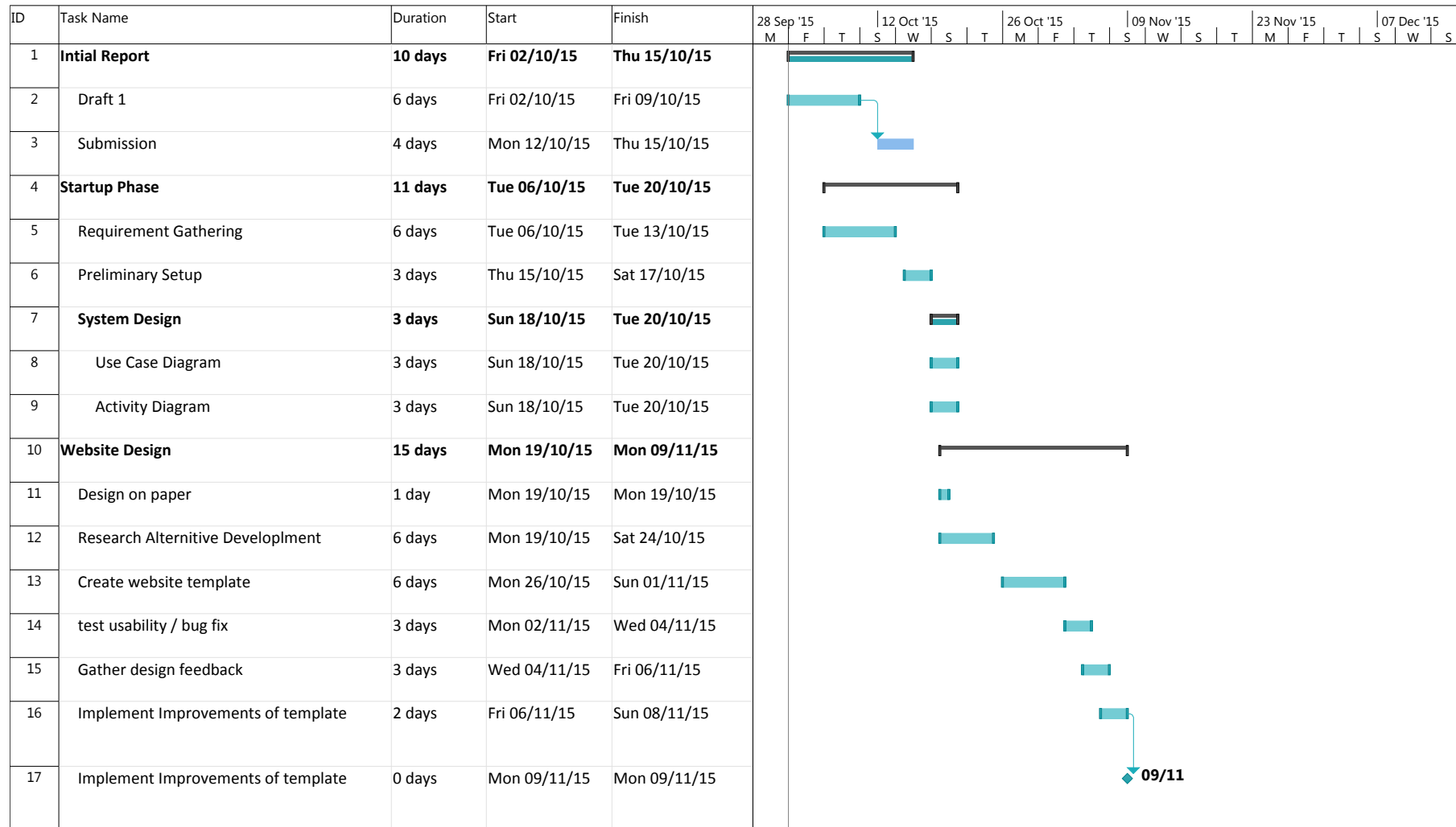
The code above is used to generate an item to insert a randomly generated item into the data dictionary.

A random number is generated between one and three to determine of what type it belongs, i.e. skill, interest, or location.

The above code is written in C# and uses the LINQ library to enable the modification and access to the Microsoft SQL Database.

Adding of the generated data values to the data was sourced externally. (Woo, 2012).

Appendix F: Initial Time Plan














(Continued)

| ID | Task Name | Duration | Start | Finish | <div> <div>26 Oct '15</div> <div>09 Nov '15</div> <div>23 Nov '15</div> <div>07 Dec '15</div> <div>21 Dec '15</div> <div>04 Jan '16</div> <div>18 Jan '16</div> </div> |
|----|---|----------------|---------------------|---------------------|--|
| 18 | Database Creation | 12 days | Mon 26/10/15 | Tue 10/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 19 | List Entities & Relations | 5 days | Mon 26/10/15 | Fri 30/10/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 20 | Design Database ERD | 3 days | Fri 30/10/15 | Tue 03/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 21 | Normalisation | 1 wk | Tue 03/11/15 | Mon 09/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 22 | Create Database SQL Schema | 1 day | Tue 10/11/15 | Tue 10/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 23 | <i>Database Created</i> | 0 days | Tue 10/11/15 | Tue 10/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 24 | Security Considerations / Research | 7 days | Mon 09/11/15 | Tue 17/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 25 | Resarch Securty | 4 days | Mon 09/11/15 | Thu 12/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 26 | Website Protection | 4 days | Tue 10/11/15 | Fri 13/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 27 | User Access Rights | 4 days | Wed 11/11/15 | Mon 16/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 28 | Website Functionailty Development | 2 wks | Wed 11/11/15 | Tue 24/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 29 | User Registration | 1 day | Wed 11/11/15 | Wed 11/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 30 | Logon Mechanism | 1 day | Wed 11/11/15 | Wed 11/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 31 | User Accounts | 2 days | Wed 11/11/15 | Thu 12/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 32 | Volunteer Profile | 2 days | Thu 12/11/15 | Fri 13/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |
| 33 | Local Charity Registration | 1 day | Fri 13/11/15 | Fri 13/11/15 | <div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> <div>T</div> <div>S</div> <div>W</div> <div>S</div> <div>T</div> <div>M</div> <div>F</div> </div> |

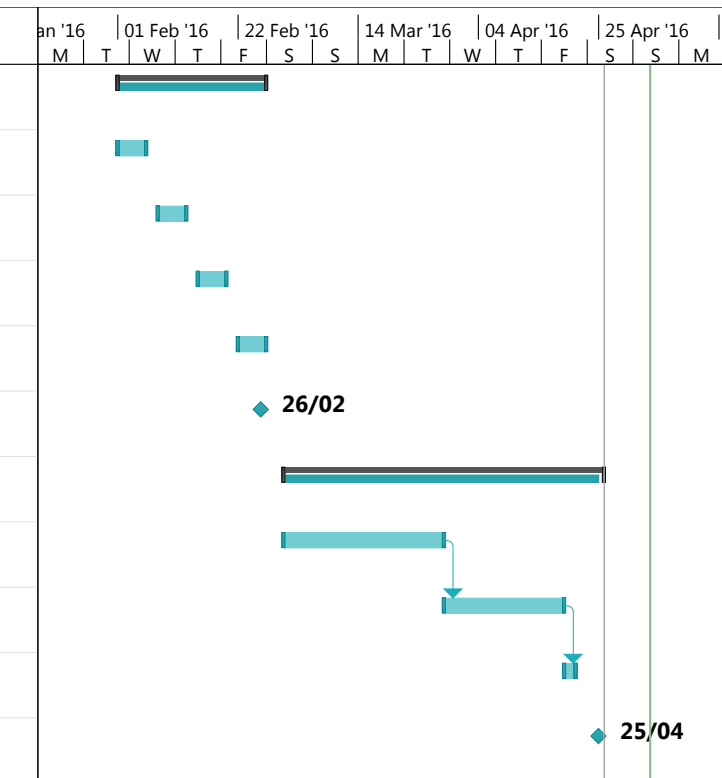
(Continued)

| ID | Task Name | Duration | Start | Finish | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----------------------------------|----------|--------------|--------------|---------|---|------------|---|---|---|------------|---|---|---|------------|---|---|---|------------|---|---|---|------------|---|---|---|------------|---|--|--|--|--|
| | | | | | Oct '15 | | 09 Nov '15 | | | | 23 Nov '15 | | | | 07 Dec '15 | | | | 21 Dec '15 | | | | 04 Jan '16 | | | | 18 Jan '16 | | | | | |
| | | | | | F | T | S | W | S | T | M | F | T | S | W | S | T | M | F | T | S | W | S | T | M | F | T | S | | | | |
| 34 | Implement Secuirty | 1 day | Wed 11/11/15 | Wed 11/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 35 | Volunteer Project Listing | 2 days | Mon 16/11/15 | Tue 17/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 36 | Volunteer Project Upload | 2 days | Tue 17/11/15 | Wed 18/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 37 | Allow Editing of Projects | 1 day | Wed 18/11/15 | Wed 18/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38 | Create Test Projects | 2 days | Thu 19/11/15 | Fri 20/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 39 | Admin User Account Actions | 2 days | Fri 20/11/15 | Mon 23/11/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 40 | Algorithm Design and Development | 2 mons | Mon 23/11/15 | Fri 15/01/16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 41 | Research into matching Alogrithm | 3 wks | Mon 23/11/15 | Fri 11/12/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 42 | Develop Prototype | 2 wks | Wed 02/12/15 | Tue 15/12/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 43 | Analyse Algorithm Prototype | 2 days | Tue 15/12/15 | Wed 16/12/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 44 | Refine | 1 wk | Wed 16/12/15 | Tue 22/12/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 45 | Test algorithm | 3 days | Mon 28/12/15 | Wed 30/12/15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 46 | Implement onto website | 2 days | Sat 02/01/16 | Mon 04/01/16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 47 | Algorithm Implemented | 0 days | Mon 04/01/16 | Mon 04/01/16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

(Continued)






















































| ID | Task Name | Duration | Start | Finish | '15 F | 09 Nov '15 S | 30 Nov '15 S | 21 Dec '15 M | 11 Jan '16 T | 01 Feb '16 W | 22 Feb '16 T | 14 Mar '16 F | 04 Apr '16 S | 25 Apr '16 S | | |
|----|----------------------------|----------------|---------------------|---------------------|---|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|--|--|
| 48 | Interim Report ACW | 53 days | Thu 05/11/15 | Sat 16/01/16 |  | | | | | | | | | | | |
| 49 | Draft 1 | 46 days | Thu 05/11/15 | Thu 07/01/16 |  | | | | | | | | | | | |
| 50 | Submission | 7 days | Fri 08/01/16 | Sat 16/01/16 |  | | | | | | | | | | | |
| 51 | Showing Interest | 4 days | Mon 25/01/16 | Thu 28/01/16 |  | | | | | | | | | | | |
| 52 | Show interest in project | 1 day | Mon 25/01/16 | Mon 25/01/16 |  | | | | | | | | | | | |
| 53 | Email potential volunteers | 4 days | Mon 25/01/16 | Thu 28/01/16 |  | | | | | | | | | | | |
| 54 | Event Book On | 1 wk | Thu 28/01/16 | Wed 03/02/16 |  | | | | | | | | | | | |
| 55 | Book volunteer on project | 1 day | Thu 28/01/16 | Thu 28/01/16 |  | | | | | | | | | | | |
| 56 | Remove volunteers | 1 day | Fri 29/01/16 | Fri 29/01/16 |  | | | | | | | | | | | |
| 57 | Printable List | 1 day | Fri 29/01/16 | Fri 29/01/16 |  | | | | | | | | | | | |
| 58 | Volunteering History | 1 day | Sat 30/01/16 | Sat 30/01/16 |  | | | | | | | | | | | |

(Continued)











































| ID | Task Name | Duration | Start | Finish |  | | | | | | | | | | | | | | | | | | |
|----|---|----------|--------------|--------------|---|---|------------|---|------------|---|---|---|------------|---|---|---|------------|---|---|------------|---|---|---|
| | | | | | an '16 | T | 01 Feb '16 | T | 22 Feb '16 | F | S | S | 14 Mar '16 | M | T | W | 04 Apr '16 | T | F | 25 Apr '16 | S | S | M |
| 59 | User Testing | 4 wks | Mon 01/02/16 | Fri 26/02/16 | | | | | | | | | | | | | | | | | | | |
| 60 | Get User Feedback | 1 wk | Mon 01/02/16 | Fri 05/02/16 | | | | | | | | | | | | | | | | | | | |
| 61 | Refinements from feedback | 1 wk | Mon 08/02/16 | Fri 12/02/16 | | | | | | | | | | | | | | | | | | | |
| 62 | Regather Feedback | 1 wk | Mon 15/02/16 | Fri 19/02/16 | | | | | | | | | | | | | | | | | | | |
| 63 | Final Refinements | 1 wk | Mon 22/02/16 | Fri 26/02/16 | | | | | | | | | | | | | | | | | | | |
| 64 | Final Refinements | 0 days | Fri 26/02/16 | Fri 26/02/16 | | | | | | | | | | | | | | | | | | | |
| 65 | Final Report ACW | 2 mons | Tue 01/03/16 | Mon 25/04/16 | | | | | | | | | | | | | | | | | | | |
| 66 | Draft 1 | 1 mon | Tue 01/03/16 | Mon 28/03/16 | | | | | | | | | | | | | | | | | | | |
| 67 | Draft 2 | 3 wks | Tue 29/03/16 | Mon 18/04/16 | | | | | | | | | | | | | | | | | | | |
| 68 | Submission | 2 days | Tue 19/04/16 | Wed 20/04/16 | | | | | | | | | | | | | | | | | | | |
| 69 | Submission of final year project report | 0 days | Mon 25/04/16 | Mon 25/04/16 | | | | | | | | | | | | | | | | | | | |

(End)













































Appendix G: Revised Time Plan

| ID |  | Task Mode | Task Name | Duration | Start | Finish | 01 September | 11 October | 21 November | 01 January | 11 February |
|----|---|---|----------------------------|----------------|---------------------|---------------------|--------------|---|-------------|------------|-------------|
| 1 |  |  | Initial Report | 10 days | Fri 02/10/15 | Thu 15/10/15 | |  | | | |
| 2 |  |  | Draft 1 | 6 days | Fri 02/10/15 | Fri 09/10/15 | |  | | | |
| 3 |  |  | Submission | 4 days | Mon 12/10/15 | Thu 15/10/15 | |  | | | |
| 4 |  |  | Startup Phase | 11 days | Tue 06/10/15 | Tue 20/10/15 | |  | | | |
| 5 |  |  | Requirement Gathering | 6 days | Tue 06/10/15 | Tue 13/10/15 | |  | | | |
| 6 |  |  | Preliminary Setup | 3 days | Thu 15/10/15 | Sat 17/10/15 | |  | | | |
| 7 |  |  | System Design | 3 days | Sun 18/10/15 | Tue 20/10/15 | |  | | | |
| 8 |  |  | Use Case Diagram | 3 days | Sun 18/10/15 | Tue 20/10/15 | |  | | | |
| 9 |  |  | Activity Diagram | 3 days | Sun 18/10/15 | Tue 20/10/15 | |  | | | |
| 10 |  |  | Website Design | 10 days | Mon 19/10/15 | Sun 01/11/15 | |  | | | |
| 11 |  |  | Design on paper | 1 day | Mon 19/10/15 | Mon 19/10/15 | |  | | | |
| 12 |  |  | Create website template | 6 days | Mon 26/10/15 | Sun 01/11/15 | |  | | | |
| 13 |  |  | Database Creation | 12 days | Mon 26/10/15 | Tue 10/11/15 | |  | | | |
| 14 |  |  | List Entities & Relations | 5 days | Mon 26/10/15 | Fri 30/10/15 | |  | | | |
| 15 |  |  | Design Database ERD | 3 days | Fri 30/10/15 | Tue 03/11/15 | |  | | | |
| 16 |  |  | Normalisation | 1 wk | Tue 03/11/15 | Mon 09/11/15 | |  | | | |
| 17 |  |  | Create Database SQL Schema | 1 day | Tue 10/11/15 | Tue 10/11/15 | |  | | | |
| 18 |  |  | Database Created | 0 days | Tue 10/11/15 | Tue 10/11/15 | |  | | | |


(Continued)

| ID |  | Task Mode | Task Name | Duration | Start | Finish | 11 October | 21 November | 01 January | 11 February | 21 March |
|----|---|---|---|----------------|---------------------|---------------------|---|---|---|---|----------|
| 19 |  |  | Security Considerations / Research | 7 days | Mon 09/11/15 | Tue 17/11/15 |  | | | | |
| 20 |  |  | Resarch Securty | 4 days | Mon 09/11/15 | Thu 12/11/15 |  | | | | |
| 21 |  |  | Website Protection | 4 days | Tue 10/11/15 | Fri 13/11/15 |  | | | | |
| 22 |  |  | User Access Rights | 4 days | Wed 11/11/15 | Mon 16/11/15 |  | | | | |
| 23 | |  | Website Functionailty Development | 49 days | Thu 10/12/15 | Tue 16/02/16 | |  | | | |
| 24 |  |  | User Registration | 2 days | Thu 10/12/15 | Fri 11/12/15 | |  | | | |
| 25 |  |  | Logon Mechanism | 1 day | Thu 10/12/15 | Thu 10/12/15 | |  | | | |
| 26 | |  | Admin User Account Actions | 25 days | Mon 28/12/15 | Fri 29/01/16 | | |  | | |
| 27 |  |  | Manage User Rights | 20 days | Mon 28/12/15 | Fri 22/01/16 | | |  | | |
| 28 |  |  | Manage Skills | 3 days | Mon 28/12/15 | Wed 30/12/15 | |  | | | |
| 29 |  |  | Manage Interests | 1 day | Mon 25/01/16 | Mon 25/01/16 | | |  | | |
| 30 |  |  | Manage Location Districts | 1 day | Tue 26/01/16 | Tue 26/01/16 | | |  | | |
| 31 | |  | Manage Additonal Elements | 3 days | Wed 27/01/16 | Fri 29/01/16 | | |  | | |
| 32 | |  | Create a bunch of user accounts | 2 days | Sat 02/01/16 | Mon 04/01/16 | |  | | | |
| 33 |  |  | Volunteer Profile Page | 6 days | Sun 31/01/16 | Fri 05/02/16 | | | |  | |

(Continued)

| ID |  | Task Mode | Task Name | Duration | Start | Finish | 21 November | 01 January | 11 February | 21 March | 01 May |
|----|---|---|----------------------------------|----------|--------------|--------------|---|---|---|----------|--------|
| 34 |  |  | Local Charity Registration | 5 days | Fri 05/02/16 | Thu 11/02/16 | | |  | | |
| 35 | |  | Volunteer Project Listing | 3 days | Thu 11/02/16 | Tue 16/02/16 | | |  | | |
| 36 | |  | Create Test Projects | 2 days | Thu 11/02/16 | Fri 12/02/16 | | |  | | |
| 37 | |  | Volunteer Project Upload | 3 days | Thu 11/02/16 | Mon 15/02/16 | | |  | | |
| 38 |  |  | Allow Editing of Projects | 0 days | Tue 16/02/16 | Tue 16/02/16 | | |  16/02 | | |
| 39 | |  | Algorithm Design and Development | 2.5 wks | Wed 17/02/16 | Fri 04/03/16 | | |  | | |
| 40 |  |  | Research into matching Alogrithm | 1 wk | Wed 17/02/16 | Tue 23/02/16 | | |  | | |
| 41 |  |  | Develop Prototype | 1 wk | Thu 18/02/16 | Wed 24/02/16 | | |  | | |
| 42 |  |  | Analyse Algorithm Prototype | 1 day | Tue 23/02/16 | Tue 23/02/16 | | |  | | |
| 43 |  |  | Refine | 1 wk | Tue 23/02/16 | Mon 29/02/16 | | |  | | |
| 44 |  |  | Test algorithm | 3 days | Mon 29/02/16 | Wed 02/03/16 | | |  | | |
| 45 |  |  | Implement onto website | 2 days | Wed 02/03/16 | Thu 03/03/16 | | |  | | |
| 46 | |  | Algorithm Implemented | 0 days | Thu 03/03/16 | Thu 03/03/16 | | |  03/03 | | |
| 47 |  |  | Interim Report ACW | 3 wks | Sun 03/01/16 | Thu 21/01/16 |  | | | | |
| 48 |  |  | Draft 1 | 3 wks | Sun 03/01/16 | Thu 21/01/16 |  | | | | |
| 49 |  |  | Submission | 0 days | Thu 21/01/16 | Thu 21/01/16 | |  21/01 | | | |

(Continued)

| ID |  | Task Mode | Task Name | Duration | Start | Finish | 21 November | 01 January | 11 February | 21 March | 01 May |
|----|---|---|-----------------------------|----------|--------------|--------------|-------------|------------|---|----------|--------|
| 50 | |  | Showing interest in project | 5 days | Wed 17/02/16 | Tue 23/02/16 | | |  | | |
| 51 |  |  | Show interest in project | 1 day | Wed 17/02/16 | Wed 17/02/16 | | |  | | |
| 52 | |  | Email potential volunteers | 4 days | Thu 18/02/16 | Tue 23/02/16 | | |  | | |
| 53 | |  | Event Book On | 1 wk | Wed 17/02/16 | Tue 23/02/16 | | |  | | |
| 54 |  |  | Book volunteer on project | 1 day | Wed 17/02/16 | Wed 17/02/16 | | |  | | |
| 55 |  |  | Remove volunteers | 1 day | Wed 17/02/16 | Wed 17/02/16 | | |  | | |
| 56 | |  | Printable List | 1 day | Thu 18/02/16 | Thu 18/02/16 | | |  | | |
| 57 |  |  | Volunteering History | 1 day | Thu 18/02/16 | Thu 18/02/16 | | |  | | |
| 58 | |  | User Testing | 4 wks | Sat 05/03/16 | Thu 31/03/16 | | |  | | |
| 64 | |  | Final Report ACW | 2 mons | Tue 01/03/16 | Mon 25/04/16 | | |  | | |

(End)

References

- Barry, D. K., 2016. *When an Object Database Should Be Used*. [Online]
Available at: http://www.service-architecture.com/articles/object-oriented-databases/when_an_object_database_should_be_used.html
[Accessed 28 April 2016].
- BBC, 2015. *TalkTalk hack 'affected 157,000 customers'*. [Online]
Available at: <http://www.bbc.co.uk/news/business-34743185>
[Accessed 15 April 2016].
- Beynon-Davies, P., 2000. *Database Systems*. 2nd ed. Basingstoke: Palgrave.
- Bootstrap, 2013. *Bootstrap - The world's most popular mobile-first and responsive front-end framework*. [Online]
Available at: <http://getbootstrap.com/>
[Accessed 19 January 2016].
- Connolly, T. M. & Begg, C. E., 2005. *Database Systems: A Practical Approach to Design, Implementation, and Management*. 4th ed. Harlow, England: Pearson Education.
- DatatabaseDev, 2015. *Flat File Database Design vs. Relational Database Design*. [Online]
Available at: <http://www.databasedev.co.uk/flatfile-vs-rdbms.html>
[Accessed 3 May 2016].
- DB-Engines, 2016. *DB-Engines Ranking*. [Online]
Available at: <http://db-engines.com/en/ranking>
[Accessed 18 January 2016].
- Do-it Trust, 2016. *Find the perfect volunteering opportunity just for you*. [Online]
Available at: <https://do-it.org/opportunities/search?miles=5&interests=52bd46a8-63d2-47d8-b4a1-785f8215da59,8832e673-44fc-4771-a66d-40dfb9d038da&lat=53.7696&lng=-0.367165&location=HU6%20RX&sort=updated&order=dsc&page=1&activities=329be453-05b1-4ab6-a2f4-eee9d33cb38d>
[Accessed 29 April 2016].
- Ferrara, A., 2016. *password_compat/password.php at master*. [Online]
Available at: https://github.com/ircmaxell/password_compat/blob/master/lib/password.php
[Accessed 17 April 2016].
- Fleming, C. C. & Von Halle, B., 1989. *Handbook of relational database design*. 1st ed. Reading: Addison-Wesley.
- Guzel, B., 2012. *Understanding Hash Functions and Keeping Passwords Safe*. [Online]
Available at: <http://code.tutsplus.com/tutorials/understanding-hash-functions-and-keeping-passwords-safe--net-17577>
[Accessed 21 January 2016].
- Haerder, T. & Reuter, A., 1983. Principles of transaction-oriented database recovery. *CSUR*, 15(4), pp. 287-317.
- Hull CVS, 2014. *Annual Report of the Trustees and Financial*. [Online]
Available at: <http://hullcvs.org.uk/documents/hull-cvs-annual-report-and-accounts.pdf>
[Accessed 29 April 2016].

- Hull CVS, 2015. *Drop In Service*. [Online]
Available at: <http://hullcvs.org.uk/volunteer-centre-hull/drop-in-service/>
[Accessed 13 October 2015].
- Hull CVS, 2016. *Drop In Service*. [Online]
Available at: <http://hullcvs.org.uk/volunteer-centre-hull/drop-in-service/>
[Accessed 7 March 2016].
- Hunt, T., 2015. *Pwned websites*. [Online]
Available at: <https://haveibeenpwned.com/PwnedWebsites>
[Accessed 20 January 2016].
- Information Commissioner's Office, 2016. *Information security (Principle 7)*. [Online]
Available at: <https://ico.org.uk/for-organisations/guide-to-data-protection/principle-7-security/>
[Accessed 20 January 2016].
- Kent, A. & Williams, J. G., 2001. *Encyclopedia of Computer Science and Technology: Supplement 29 Volume 44*. 1st ed. New York: Marcel Dekker, Inc..
- Lyne, J., 2013. *30,000 Web Sites Hacked A Day. How Do You Host Yours?*. [Online]
Available at: <http://www.forbes.com/sites/jameslyne/2013/09/06/30000-web-sites-hacked-a-day-how-do-you-host-yours/>
[Accessed 21 April 2016].
- Mannila, H. & R  ih  , K.-J., 1992. *The design of relational databases*. 1st ed. Reading: Addison-Wesley.
- McFadyen, R., 2006. *Atomic Attributes*. [Online]
Available at: <http://ion.uwinnipeg.ca/~rmcfadye/2914/hypergraph/atomic.html>
[Accessed 13 April 2016].
- Microsoft MSDN, 2016. *Working with Scripting Languages*. [Online]
Available at: <https://msdn.microsoft.com/en-us/library/ms525153%28v=vs.90%29.aspx?f=255&MSPPErr=-2147217396>
[Accessed 10 April 2016].
- Microsoft, 2013. *Description of the database normalization basics*. [Online]
Available at: <https://support.microsoft.com/en-us/kb/283878>
[Accessed 13 April 2016].
- MongoDB, 2016. *What is NoSQL?*. [Online]
Available at: <https://www.mongodb.com/nosql-explained>
[Accessed 2 May 2016].
- Mono Project, 2016. *Cross platform, open source .NET framework*. [Online]
Available at: <http://www.mono-project.com/>
[Accessed 2 May 2016].
- National Council for Voluntary Organisation, 2015. *I want to volunteer*. [Online]
Available at: <https://www.ncvo.org.uk/ncvo-volunteering/i-want-to-volunteer>
[Accessed 13 October 2015].
- National Council for Voluntary Organisations, 2016. *Volunteering overview*. [Online]
Available at: <https://data.ncvo.org.uk/a/almanac16/volunteer-overview/>
[Accessed 28 April 2016].
- Office for National Statistics, 2012. *2011 Census: Population Estimates for the United Kingdom, 27 March 2011*. [Online]

Available at: http://www.ons.gov.uk/ons/dcp171778_292378.pdf
[Accessed 28 April 2016].

Oracle Corporation, 2016. *What is MySQL?*. [Online]
Available at: <http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html>
[Accessed 30 March 2016].

Price, J., 2014. *Oracle Database 12c SQL*. 1st ed. New York: McGraw-Hill Education.

Rayner, J., 2016. *Project Catalogue: Volunteer Project Matching*. [Online]
Available at:
<http://intra.dcs.hull.ac.uk/student/modules/08341/Selectable%20Projects/DispForm.aspx?ID=771&Source=http%3A%2F%2Fintra%2Edcs%2Ehull%2Eac%2Euk%2Fstudent%2Fmodules%2F08341%2FSelectable%2520Projects%2FNewDef%2Easpx>
[Accessed 2 May 2016].

Russell, G., 2008. *Normalisation - BCNF*. [Online]
Available at: <http://db.grussell.org.uk/section009.html>
[Accessed 13 April 2016].

Russo, M. & Pialorsi, P., 2007. *Introducing Microsoft SQL*. 1st ed. Redmond: Microsoft Press.

Sadalage, P., 2014. *NoSQL Databases: An Overview*. [Online]
Available at: <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>
[Accessed 2 May 2016].

Stansfield, J., 2014. *Microsoft SQL Server vs. Oracle: The Same, But Different?*. [Online]
Available at: <http://www.seguetech.com/blog/2014/03/13/Microsoft-SQL-Server-versus-oracle>
[Accessed 2 May 2016].

Swan, B., 2008. *How and Why to Use Parameterized Queries*. [Online]
Available at: <https://blogs.msdn.microsoft.com/sqlphp/2008/09/30/how-and-why-to-use-parameterized-queries/>
[Accessed 15 April 2016].

Taylor, A. G., 2006. *SQL for dummies*. 1st ed. Hoboken, NJ: John Wiley.

The Internet Engineering Task Force, 2007. *Guidance for Authentication, Authorization, and Accounting (AAA)*. [Online]
Available at: <https://tools.ietf.org/html/rfc4962>
[Accessed 15 April 2016].

The PHP Group, 2016. *Downloads*. [Online]
Available at: <http://php.net/downloads.php>
[Accessed 11 April 2016].

University of Washington, 2001. *Relationships between Tables and Entity Relationship Diagrams*. [Online]
Available at: <https://courses.cs.washington.edu/courses/cse100/01au/lectures/lecture25.pdf>
[Accessed 2016 April 28].

W3Schools, 2016. *PHP 5 Introduction*. [Online]
Available at: http://www.w3schools.com/php/php_intro.asp
[Accessed 27 04 2016].

W3Schools, 2016. *Web Hosting Database Technologies*. [Online]
Available at: http://www.w3schools.com/website/web_host_databases.asp
[Accessed 18 January 2016].

Will, 2014. *How does the SQL injection from the "Bobby Tables" XKCD comic work?*. [Online]
Available at: <http://stackoverflow.com/a/332367>
[Accessed 21 January 2016].

Woo, J., 2012. *How to insert data into SQL Server*. [Online]
Available at: <http://stackoverflow.com/a/12241118>
[Accessed 17 April 2016].