

# **AmazingStoke: A Facebook game for the Not-For-Profit sector**

**Joanna Pinto**

**MSc Computer Science Project Report  
Department of Computer Science and Information Systems,  
Birkbeck College, University of London 2011**

**This proposal is substantially the result of my own work,  
expressed in my own words, except where explicitly  
indicated in the text. I give my permission for it to be  
submitted to the JISC Plagiarism Detection Service.**

**The proposal may be freely copied and distributed  
provided the source is explicitly acknowledged.**

## Abstract

In this report I describe the design and development of the online game *AmazingStoke* for the not-for-profit sector. The game aims to engage players in real-world actions for not-for-profit organisations. The game is played via the social networking platform Facebook, and utilises recent research in the fields of online and alternate reality games (ARGs). The topic is first introduced with a review of the background and findings of the initial project proposal. Then follows *AmazingStoke* systems analysis and design using UML diagramming techniques. This report includes a discussion of the implementation of the various system components, followed by a walkthrough of the system. The *AmazingStoke* system is evaluated in terms of initial goals, and possible future extensions to the system are discussed. The conclusion of this report is that a useful prototype system has been developed, with potential for development into a genuinely valuable tool for not-for-profit organisation.

# **Table of Contents**

<b>Chapter 1: Introduction .....</b>	<b>1</b>
1.1 Why Tackle This Problem?.....	1
1.2 The Approach.....	2
1.3 Assumed Knowledge.....	2
1.4 Terminology .....	3
1.5 Roadmap Of Remaining Chapters.....	3
<b>Chapter 2: Background.....</b>	<b>5</b>
2.1 Not-For-Profit Organisations And Online Gaming.....	5
2.2 The Facebook Platform .....	6
2.3 The Client-Server Model And Facebook Apps.....	6
2.4 Authentication Of Challenges By <i>AmazingStoke</i> Players .....	7
<b>Chapter 3: System Analysis and Design .....</b>	<b>8</b>
3.1 System Requirements .....	8
3.1.1 High-Level Goals .....	9
3.1.2 Requirements From Research Questionnaire .....	11
3.1.3 FURPS Requirements Specification.....	12
3.1.4 Use Case Models .....	15
3.2 System Design.....	17
3.2.1 Identifying Conceptual Classes .....	17
3.2.2 Domain Model.....	17
3.2.3 Model, View, Controller.....	18
3.2.4 Identifying Different Views.....	19
3.2.5 Identifying Core Functionality .....	20
3.2.6 Graphics.....	21
3.3 NFP Site Design .....	22
3.3.1 Use Of A Database .....	22

3.3.2 Use Of Sessions In NFP Site .....	22
3.3.3 Adding A Challenge .....	22
3.3.4 Design Class Diagram (NFP Site) .....	24
3.4 Game Design .....	25
3.4.1 Use of Sessions In Game .....	25
3.4.2 Logging In To Game .....	25
3.4.3 Buying Buildings .....	27
3.4.4 Authenticating A Challenge .....	29
3.4.5 Viewing Charity Information & Viewing Tasks .....	30
3.4.6 Design Class Diagram (Game) .....	32
3.5 Database Design .....	33
<b>Chapter 4: Implementation.....</b>	<b>36</b>
4.1 Technologies .....	37
4.1.1 Database: SQL / MySQL Database .....	37
4.1.2 PHP .....	37
4.1.3 JavaScript.....	39
4.1.4 WAMP Development Stack .....	40
4.1.5 Facebook API .....	40
4.1.6 Hosting .....	42
4.2 First Iteration Of The System.....	44
4.2.1 First Iteration Of The MySQL Database .....	44
4.2.2 First Iteration Of NFP Site.....	46
4.2.3 First Iteration of Game .....	54
4.3 Second Iteration Of The System .....	64
4.3.1 Second Iteration Of NFP Site .....	64
4.3.2 Second Iteration Of Game .....	68
4.4 Third Iteration Of The System .....	72

<b>Chapter 5: Demonstration .....</b>	<b>73</b>
5.1 NFP Site .....	74
5.1.1 Login.....	74
5.1.2 Update Details .....	76
5.1.3 Add A Challenge .....	77
5.1.4 View Voucher Codes.....	78
5.1.5 Generate More Voucher Codes .....	80
5.1.6 Logout.....	81
5.2 Game .....	82
5.2.1 Create Account Via Facebook.....	82
5.2.2 Log Back In Via Facebook.....	84
5.2.3 Buy A Building.....	85
5.2.4 View And Accept A New Challenge.....	87
5.2.5 View NFP Information .....	88
5.2.6 Authenticate A Challenge.....	89
5.2.7 Update A Player's MarvelMoney On Login .....	91
<b>Chapter 6: Areas For Exploration .....</b>	<b>92</b>
6.1 Development Technologies.....	92
6.2 System Functionality.....	94
6.3 Further Developments .....	95
<b>Chapter 7: Conclusions .....</b>	<b>97</b>

**References**

**Appendix A: Use Cases**

**Appendix B: Code (as attached CD)**

# **Chapter 1: Introduction**

This report describes the design and development of *AmazingStoke*, an online game to aid not-for-profit organisations such as charities in engaging with new and existing supporters. *AmazingStoke* is played via the social networking site Facebook (Facebook Inc., 2011) with player information saved between sessions of the game. Alongside the game interface for players, the system includes a separate website for not-for-profit organisations to update their game information regarding their organisations.

Using the *AmazingStoke* system, not-for-profit organisations are able to list specific real-world challenges for their supporters, such as:

- donating money to the organisation
- using the player social networks to publicise the organisation
- completing volunteering activities.

Organisations can reward *AmazingStoke* players with a customisable amount of in-game credits, known as **MarvelMoney**. From the players' perspective, these MarvelMoney credits can be used within a city-building game (following the model of popular online games such as *CityVille* (Zynga Game Network Inc., 2011)). Points are used to purchase virtual houses, schools, shops and similar to create a unique "virtual town". *AmazingStoke* also features ways to earn points without necessarily taking on new challenges for not-for-profit organisations. The overall concept of "gamification" of real-world experiences has been given the name of alternate reality gaming (McGonigal, 2011) and was discussed extensively in the original project proposal.

## **1.1 Why Tackle This Problem?**

The *AmazingStoke* system was developed in response to research carried out with the not-for-profit sector specifically for this project and included in the original project proposal. Twelve not-for-profit organisations completed an online questionnaire about their current and anticipated future use of social media technologies to engage with their supporters. Organisations that completed the questionnaire included Anti-Slavery International, Greenpeace, the British Association for Adoption & Fostering (BAAF) and Marie Curie Cancer Care.

The research, together with other reports and studies, found that whilst many organisations within the not-for-profit sector were using social media to engage with supporters, people working within the sector felt that there was scope for more diverse forms of engagement than currently being used. The results were discussed in-depth in original project proposal.

## 1.2 The Approach

The *AmazingStoke* system is a cross-organisational platform, which can be used by any registered UK charity. By allowing many organisations to use one system, *AmazingStoke* allows players to discover and learn more about other charitable organisations using *AmazingStoke*.

To create both the game and the not-for-profit organisations' website, it was necessary to incorporate technologies allowing dynamic user interaction within a web browser, and a system for storing information between sessions. Decisions related to the selection of the appropriate technologies are described in **Chapter 4**.

This project followed an iterative Rapid Application Development (RAD) approach to the development, with prototype systems developed early in the implementation. This is important in a web-based application, particularly one accessed via Facebook, as there are a number of issues relating to compatibility with different web-browsers, and unique constraints of a web application accessed via the Facebook platform. It was necessary to utilise a design methodology which could be incremental (i.e., not all system requirements known in advance) and involved prototyping, as RAD does (Avison & Fitzgerald, 2002).

## 1.3 Assumed Knowledge

It is assumed that the reader has an understanding of:

- the concept of the World Wide Web as a collection of interactive pages, and of web browser technologies, in particular Internet Explorer, Firefox and Chrome (listed by the W3Schools website as the most commonly used browsers at the time of writing) (W3C, 2011)
- the HTML mark-up language for the formatting of web pages (as covered in the module **Internet and Web Technologies**) (Moller & Schwartzbach, 2006)
- HTML forms and the use of GET and POST HTTP request methods to pass information from a client-side page to a server-side program (as covered in the module **Internet and Web Technologies**) (Comer, 2004)
- the JavaScript language's Document Object Model (DOM) (as covered in the module **Internet and Web Technologies**) (Jacobs, 2006)
- object-oriented systems design and programming concepts, and the UML modelling framework (as covered in the module **Object-Oriented Design and Programming**) (Larman, 2004)
- relational databases including normalisation of databases, and the SQL language for definition and manipulation of relations (as covered in the module **Data and Knowledge Management**) (Ramakrishna & Gehrke, 2003)

The Facebook platform may be unfamiliar to readers, so a brief summary is included in **Chapter 2**.

## 1.4 Terminology

The term “not-for-profit organisation” is used to describe organisations using the *AmazingStoke* system. The system is initially envisaged for use only by organisations which are UK charities registered with either the Charity Commission for England and Wales (Charity Commission, 2011) or the Office of the Scottish Charity Regulator (Office of the Scottish Charity Regular, 2011). From this point forward the term not-for-profit organisation will be replaced with **NFP** or **NFPs**, which are commonly used acronyms within the sector.

The following formatting conventions are used within this report.

For sections of code given as examples, the following formatting is used:

```
1.  <?php
2.  echo "Hello World!"
3.  ?>
```

Names of parts of the system identifies as conceptual classes in UML modelling are identified by the following formatting:

- **Players**, or
- **NFPs**

References to functions or variables, or to file names, are identified by the following formatting:

- `mysql_fetch_row`, or
- `$_POST`, or
- `index.php`

## 1.5 Roadmap Of Remaining Chapters

### Chapter 2: Background

Provides an introduction to the aims of the *AmazingStoke* system. Goals of the system defined in **Section 2.1**. Looks in greater detail at the Facebook platform and at the technologies used to build an online game integrated into Facebook. Outlines the main technologies for dynamic web-sites and data storage used in the design and implementation of the system.



### **Chapter 3: System Analysis and Design**

Works through the design process in clear stages, using UML modelling techniques to illustrate the development of the proposed system. Outlines the design of the three main components of the system – the underlying data storage system, the website for NFPs to manage their accounts including adding challenges and updating organisational information, and the game played via Facebook.

### **Chapter 4: Implementation**

Describes the implementation of the *AmazingStoke* system. Describes the iterative development of the system with layers of complexity and functionality being added in with successive iterations. Complications and challenges are described and solutions presented, with representative examples of code where appropriate.

### **Chapter 5: Evaluation**

Gives walk-throughs of the key activities of the system, presenting screenshots and descriptions of the processes used by the distinct user groups (players and NFPs) to achieve their goals within the system.

### **Chapter 6: Areas For Exploration**

Discusses areas for further development of the system beyond the scope of the project.

### **Chapter 7: Conclusions**

Reviews the overall project, comparing against initial objectives with final system. Review of personal goals. Evaluation of the approach.

## **Chapter 2: Background**

### **2.1 Not-For-Profit Organisations And Online Gaming**

Before undertaking this MSc in Computer Science at Birkbeck College, University of London, I spent four years working in the fundraising and communications team of a national UK charity, the British Association for Adoption & Fostering (BAAF). During that time, I was struck by two observations.

Firstly, that NFPs often used social and (non-computer) gaming events to raise money and build awareness of the organisation. Such events include fun runs, quiz evenings, fancy-dress events (such as the annual “Moonwalk” for charity where supporters decorate bras and walk through London to raise money for Breast Cancer Awareness).

Secondly, I was aware that NFP organisations were exploring ways to use web-based technologies to create greater engagement with supporters. I was personally involved with the setting up of BAAF’s own Facebook page, with the updating of BAAF’s account on micro-blogging site Twitter (Twitter, 2011), with handling moneys raised via online donations sties JustGiving (JustGiving, 2011) and Virgin Money Giving (Virgin Money, 2011), and charity video website See The Difference (See The Difference, 2011) (all discussed in the original project proposal).

Whilst studying for this MSc in Computer Science, I became aware of the body of work looking at overlapping online gaming with real-world social engagement. I discussed in the original project proposal the work of game designer Jane McGonigal, who created online games such as World Without Oil (Eklund & McGonigal, 2007). I also discussed games such as FreeRice (UN World Food Programme, 2009), UK charity Save The Children’s involvement with online game Second Life (Gibson, 2006), and other overlapping of real-world engagement with online gaming.

Although the concept of using online gaming by NFPs to create greater engagement with supports is not novel, there seemed to be a lack of an easy-to-use online game suitable for use by multiple charities on a large scale, in the way that Virgin Money Giving and JustGiving delivered online solutions to sponsorship of fundraising challenges for individuals. The *AmazingStoke* system was therefore been developed to meet the specific goals of:

- A single online game that can be used by multiple organisations
- Easily accessible via standard web browsers
- Each NFP has the ability to load customised tasks for supporters to complete into the game
- Rewards real-world actions with online in-game credits

## 2.2 The Facebook Platform

Facebook is a free-to-use site where individuals register to create a profile of personal information (such as name, date of birth, current place of residence, relationship status, and photo) which is displayed as a web-page within the site. It is commonly referred to as a social networking site, although it describes itself as a “social utility”. Once a user has created a profile, they are able to send requests to link other registered users to become “friends” on the site, meaning they are able to view one another’s profiles and other shared information. Users can post public or private messages to other users, share media such as videos or pictures. Organisations can set up pages of information for users to access in-site.

Facebook also has a wide range of optional mini-programs that users can choose to access, known as apps, which run within frames in the main Facebook site (Facebook Developers, 2011). The technology used to display these apps is the `iframe`, discussed in the original project proposal.

The decision to use the Facebook platform was discussed in the original project proposal.

## 2.3 The Client-Server Model And Facebook Apps

The client-server model of a web-based application was discussed in the original project proposal. It was also explained that in the case of a Facebook app, the client-server architecture is slightly more complicated. The Facebook site use `iframes` to deliver apps, meaning that in in-line frame within the Facebook site displays the page hosted by the application server. Although it appears to the player that they are viewing a single page, the view is actually made up of web page loaded within a web page. The overview of how the client makes calls to the two separate servers (Facebook site server and the app server), and the way in which client-side scripting (in this illustration, JavaScript within the browser) makes calls to Facebook server, are illustrated by **Figure 2.1** from the Facebook Developers site (Facebook Developers, 2011).

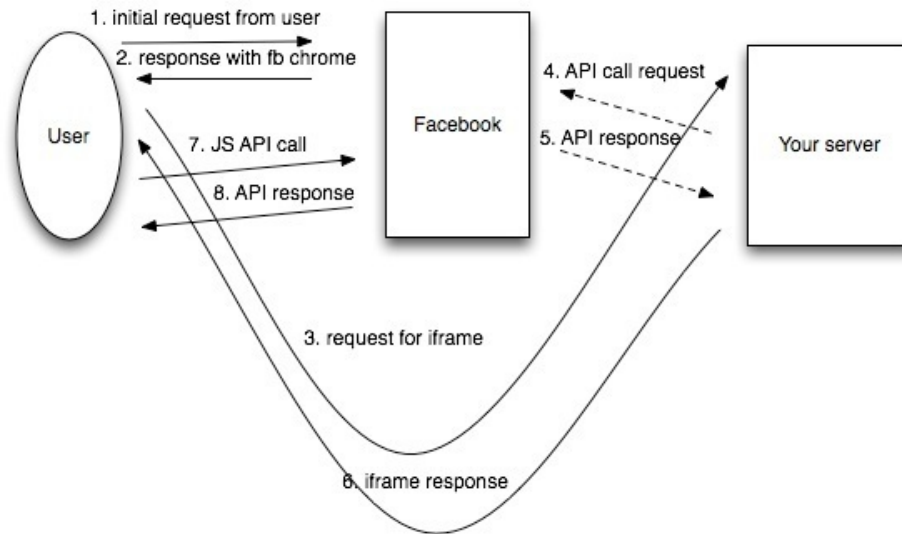


Figure 2.1: A diagram showing calls from the AmazingStoke player (“User”) to the Facebook server and the AmazingStoke server (“Your server”). JS denotes JavaScript. Taken from Facebook Developers (Facebook Developers, 2011).

## 2.4 Authentication Of Challenges By AmazingStoke Players

It became necessary fairly early in the system design to make a decision regarding the method by which a Player will authenticate that they have completed a challenge that they have taken on.

In the original project proposal, I had suggested that NFPs would need to manually check any tasks that a Player had stated that they had completed. This has the potential to be extremely time-consuming for NFPs.

Therefore, I made the decision that Challenges would be authenticated by **Voucher Codes**, which are six-digit integers between 100000 and 999999 which the NFP generates at the time of adding the task, with the functionality to generate further Voucher Codes at a later point.

My chosen method is that NFPs will be able to generate either a single authentication codes that can be used by multiple Players for the same Challenge (which would be useful when Players are completing major Challenges for an NFP, such as completing a marathon or becoming a media volunteer for an organisation), or multiple authentication codes which are single-use (i.e., deleted from persistent data structure after use) which would be suitable for, for example, mailing out to any supporter who gave a donation over a certain size, or anyone who changed their Facebook profile picture to a charity’s logo.

## **Chapter 3: System Analysis and Design**

### **3.1 System Requirements**

From initial project proposal, it was clear that the *AmazingStoke* system will be used by two main groups with separate (although related) goals. These groups are players and NFPs. From this point onwards, these user groups will be referred to as **Players** and **NFPs** respectively.

Initial ideas also suggested that the system would comprise three main components – the player game, the NFP administration website, and the underlying persistent data structures. Outside of this core design decision, the use of various UML techniques assisted in the design of the conceptual “to-be” system. This chapter explores the conceptual system via a series of modelling techniques and incorporates design patterns to solve certain problems that arose in the design process.

Given the nature of this project – a web-based project being developed in conjunction with a highly changeable social networking interface – it seemed appropriate to use an agile and iterative development process for system development. As noted in **Chapter 1**, various RAD principles will be used.

Additionally, some (although not all) of the principles of Extreme Programming (XP) such as continuous testing, simple coding of frequent “releases” to prototypes of parts of the system to users quickly, were adopted in the development of the *AmazingStoke* system (Larman, 2004). Given the fact that a specific user group had been identified and involved at an early stage of the project, ongoing user testing would be both important and practical to carry out. The modularity of the system, combined with the fact that programming for the web can offer a broad range of solutions to design decisions, meant that a flexible and iterative approach seemed likely to lead to the best possible end system.

### 3.1.1 High-Level Goals

The background research questionnaire and the initial project proposal explored many of the key ideas of the requirements of the *AmazingStoke* proposed system, and included two use cases defining steps taken by the two groups of users to achieve their goals using the system. From these initial documents (questionnaire results, initial project proposal) lists of key user goals were defined as below.

#### *NFP User Tasks*

1. Upload new **Challenges** to *AmazingStoke* system
2. Generate random (or pseudorandom) authentication **Voucher Codes** for distribution to **Players**
3. Update organisational information
4. Send messages to **Players**
5. Obtain statistics relating to numbers of **Players** registering for and completing **Challenges**

#### *Player User Tasks*

1. Create **Game** account via Facebook
2. Log in to **Game** via Facebook
3. Buy **Buildings** for in-game **Map**
4. Find out more about **NFP** users
5. Take on new **Challenges**
6. Use **Voucher Codes** to confirm completion of **Challenges** and receive **MarvelMoney** rewards
7. Suggest **Challenges** for other **Players** with whom they are linked in-game (potentially via the Facebook Social Graph tools)
8. Collect in-game earnings from **Buildings** placed on in-game **Map**

There are additional tasks which may be required within the system, such as the setting up of organisations on the system, which may be referred to under a third heading of *System Administrator Tasks*. The original project proposal included a discussion on the feasibility of a full-time site moderator to ensure that the system was being not being abused by NFPs. It was stated in the original specification that the nature of the proposed *AmazingStoke* system would require a human moderator, both for the creation of new **NFP** user accounts and for the approval of **Challenges** listed on the system. Therefore, the following tasks are those that

would be required to take place by an external moderator. I have decided that it would be outside of the scope to build a separate interface for the moderation of the system at this stage – at this stage, it should be assumed that the following *System Administrator Tasks* are to be carried out via a direct access to the underlying persistent data structures.

I also decided that, although the system would include potential for **Challenges** to be marked as “Live” or not “Live”, the system developed for this project would initially list all **Challenges** as “Live.

### *System Administrator Tasks*

1. Set up new **NFP** accounts for the *AmazingStoke* system
2. Check that **Challenges** listed conform to guidelines of what can and cannot be asked of **Players** (i.e. not asking **Players** to commit potentially illegal acts such as destruction of property)
3. Remove expired **Challenges** from the system

### 3.1.2 Requirements From Research Questionnaire

As part of the original research questionnaire, the twelve NFP organisations were asked to rate fifteen different requirements of the proposed *AmazingStoke* system in terms of importance.

NFPs gave each system requirement an importance out of five, with five being “most important” and 1 being “least important”. NFPs were not constrained on how often they could award each rating.

Although included in the original project proposal, the combined ratings of the importance of different system aspect are reproduced in Table 1.

<b>Requirement</b>	<b>Importance</b>
User-friendly design	4.33
Easy for users to find out more about organisation	4.33
Fun for users	4.17
Low cost to the organisation	4.08
Information security and data protection	4.08
Easy for users to sign up or cancel account	4.08
Large number of participating players	3.92
Easy for organisation to sign up or cancel account	3.91
Appropriate content, i.e. no material inappropriate for young people	3.83
Good technical support	3.58
Input into the development of the site, i.e. where it is marketed, what audiences are targeted	3.33
Easy for charities to learn how to use	3.18
Low involvement of the organisation, i.e. not needing to check site regularly	3.00
Which other organisations are using the technology	3.00
Ability to control number of people contacting organisation	2.92

*Table 3.1: Importance of different AmazingStoke system requirements out of 5.0, as rated by twelve different NFP organisations in April 2011*



From these results, it is possible to see that the most important requirements, as requested by NFPs, are:

- User-friendly design
- Easy for users to find out more about organisation
- Fun for users
- Low cost to the organisation
- Information security and data protection
- Easy for users to sign up or cancel account

### 3.1.3 FURPS Requirements Specification

According to the Unified Process, system requirements should be categorised according to the FURPS+ model (Larman, 2004). This divides system requirements into the categories of Functionality, Usability, Reliability, Performance, Supportability, with “+” denoting the additional optional requirement factors of Implementation, Interface, Operations, Packaging, Legal.

#### Functional Requirements

The system must allow new **Players** to sign up to the **Game** via Facebook (as part of requirements “Easy for users to sign up or cancel account”).

The system must allow **Players** to login to the **Game** using their Facebook login information.

The system must write all changes confirmed by users (**Players** and **NFPs**) to underlying persistent data structure such as a **Database** stored on the server computer.

The system must keep track of the passage of time and update both **Game** information and **NFP Site** information accordingly.

The system must require users of either part of the system (**Game** and **NFP Site**) to login securely (as part of requirements of “Information security and data protection”)

It must be easy for **Players** to find out more about organisations using the *AmazingStoke* system.

## **Usability Requirements**

### *Browser Support*

Both parts of the system (**Game** and **NFP Site**) should work on the three most commonly used web browsers – Internet Explorer, Firefox and Chrome (W3C, 2011)

Browsers used for accessing the **Game** must have enabled JavaScript to run within the browser

### *Human Accessibility*

Both parts of the system should take into account issues of web accessibility for users of differing levels of ability (W3C, 2011)

## **Reliability Requirements**

The system must make writes to the persistent data structure with sufficient frequency that in the event of a crash of browser, all data is not lost.

## **Performance**

Both parts of the system (**Game** and **NFP Site**) must ensure an appropriate split of the workload between client and server parts of the system, to ensure that the user experience is as smooth as possible whilst making as few calls to the server as possible whilst maintaining reliability. Fewer calls to the server mean greater speed for all users (as part of requirements “User-friendly design” and “Fun for users”).

The system must ensure that both parts of the system – but in particular the **Game** – respond swiftly to user interaction (as part of requirements “User-friendly design” and “Fun for users”).

## **Supportability**

### *Facebook Support*

The **Game** should work properly within the Facebook site, and also take advantage of the social networking features offered as part of the Facebook API and Social Graph (Facebook Developers, 2011).

### *Browser Support*

As mentioned under Usability, both parts of the system should work on at least Internet Explorer, Firefox and Chrome.

## **Adaptability**

Different users of the **NFP Site** will have different requirements in terms of the challenges they would like to load onto the system and methods of authentication. The system should therefore include sufficient flexibility to accommodate different types of tasks and authentication methods.

### **+: Implementation**

The system should incorporate both client-side and server-side scripting, and a permanent underlying persistent data structure such as a relational **Database**.

The **Game** side of the system should be integrated into the main Facebook site, accessed as an app using the Facebook `iframe` app support.

### **+: Interface**

The system must include one interface for NFPs (**NFP Site**) and one interface for Players (**Game**).

The **Game** must be accessible via Facebook.

### **+: Legal**

The system should include manual checking of all new **Challenges** loaded into the system, to ensure that **Players** are not being asked to undertake unsafe or illegal activities.

### 3.1.4 Use Case Models

Having identified the key goals of the key, it becomes helpful at this stage to create set of use cases, to better understand the processes and potential conceptual classes present within the system. Two sets of Use Cases and accompanying Use Case Models, one for Players and one for NFPs, can be developed from **Section 3.1.1**.

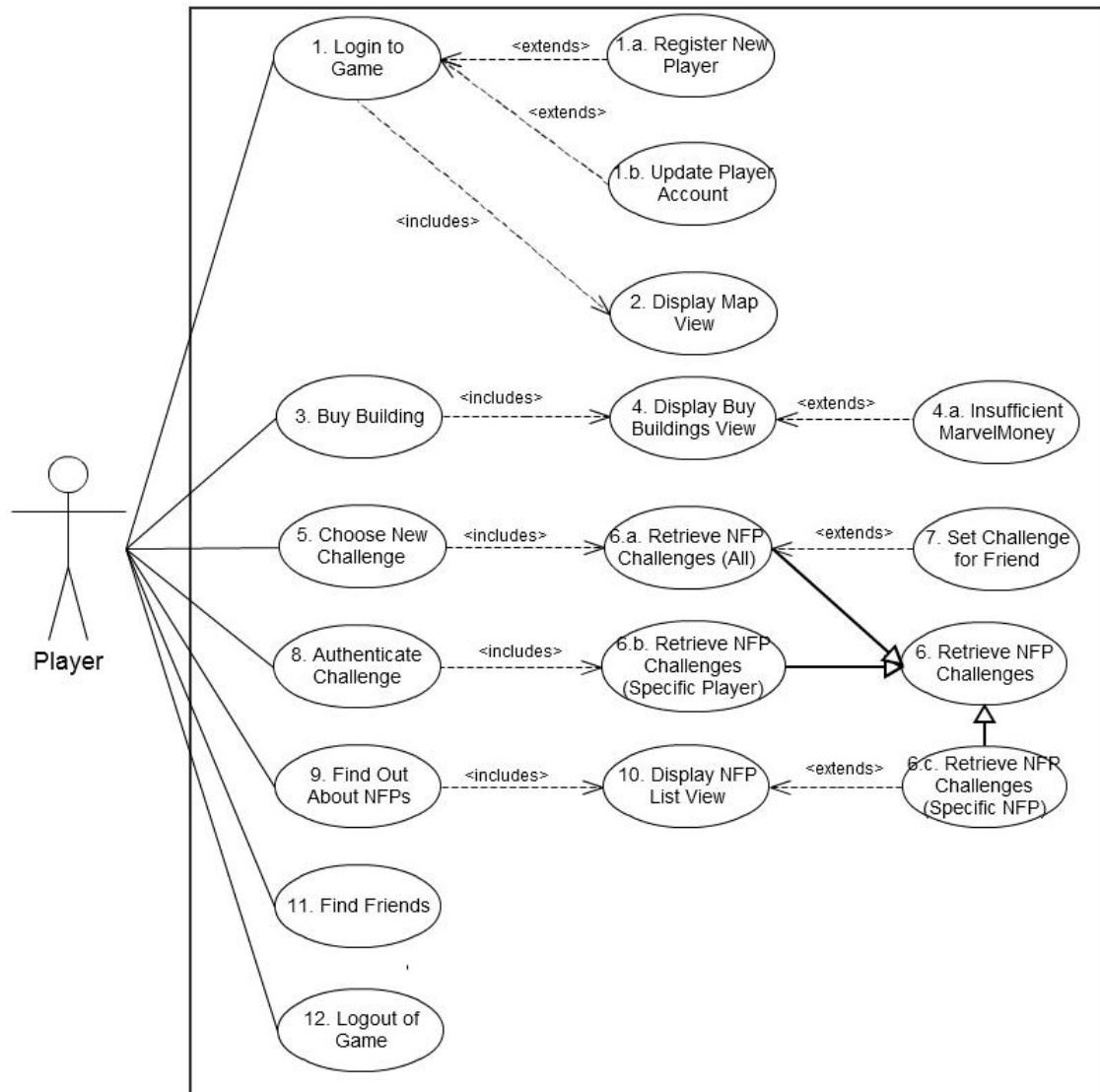


Figure 3.1: Use Case Model for Players

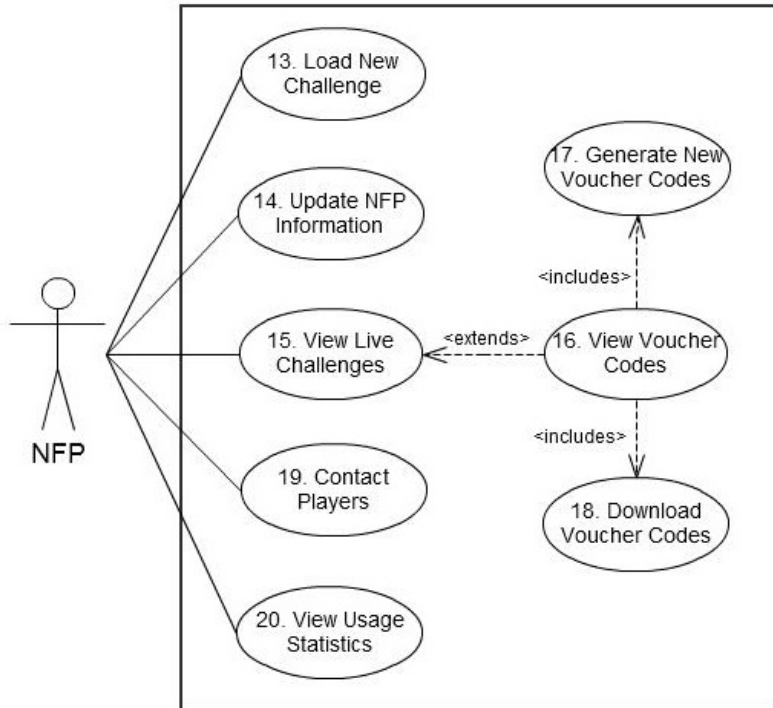


Figure 3.2: Use Case Model for NFPs

The set of major Use Cases can be found within **Appendix A**.

## 3.2 System Design

This section covers the development from the requirements of the system through to initial design of the system. Further UML modelling techniques are used to clarify and record system design considerations.

### 3.2.1 Identifying Conceptual Classes

Using the information gathered in the **Section 3.1**, it was possible to identify high-level conceptual classes. These included:

- **Player**
- **NFP**
- **Challenge**
- **Map**
- **Building**
- **Voucher Codes**
- **NFP Site**
- **Game**
- **Database**

### 3.2.2 Domain Model

Using the high-level conceptual classes identified in **Section 3.2.1** and the set of Use Cases in **Appendix A**, it was possible to construct a high-level Domain Model showing relationships between the conceptual classes. This is shown as **Figure 3.3**. This representation was used to assist to design the structure of the underlying persistent data model (**Section 3.5**).

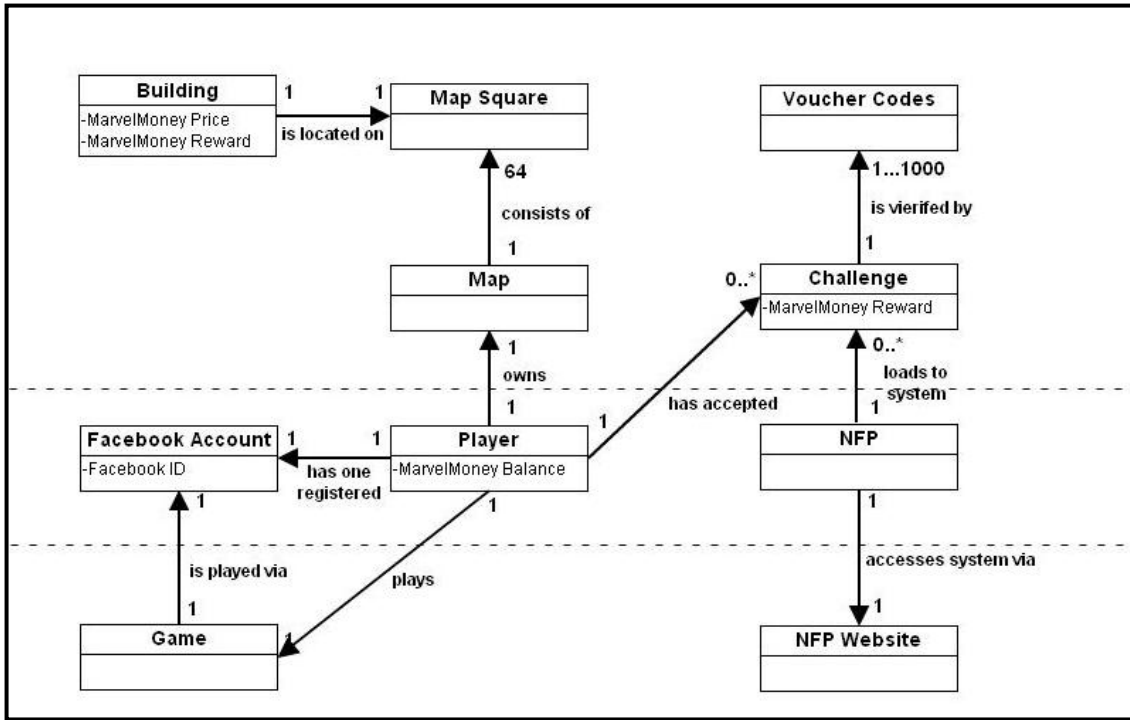


Figure 3.3: AmazingStoke System Domain Model

### 3.2.3 Model, View, Controller

It was useful to consider existing design patterns in relation to the proposed system architecture. Design patterns present solutions to problems that occur repeatedly within different forms of systems designs and development. A design pattern is a generalised set of communicating objects and classes which can be customised to solve a problem in a specific system (Gamma, Helm, Johnson, & Vlissides, 1995).

The architectural pattern Model-View-Controller, first envisioned by the creators of the Smalltalk language in the early 1970s (Dennis, Wixom, & Tegarden, 2010), is often applied to aid understanding of web applications.

The key concept is the separation of the *application* logic of the system from the logic of the *user interface* (input and presentation). The elements may be described as follows:

**Models** implement the application logic

**Views** implement the presentation of the underlying Models. Multiple views can represent a single underlying Model object.

The **Controller** collects input from users, and implements the logic on the underlying model

Figure 3.4 presents a representation of the Model-View-Controller design pattern.

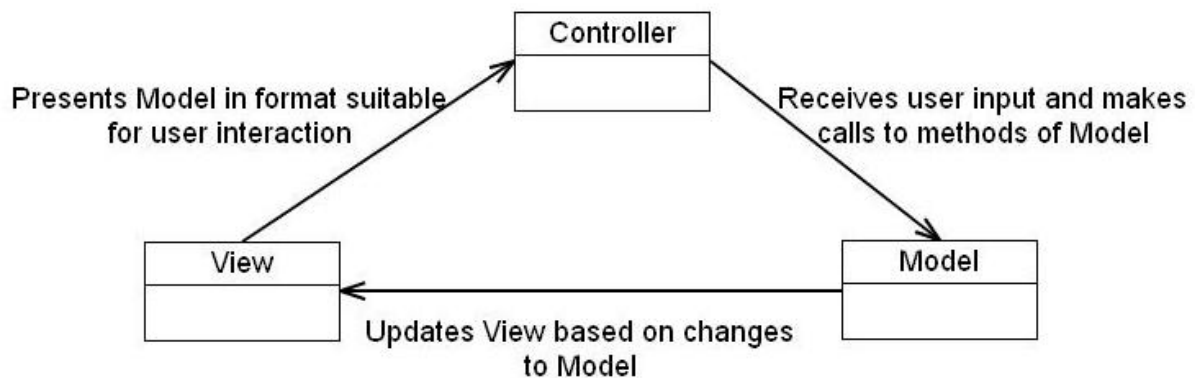


Figure 3.4 Model-View-Controller Design Pattern

The problem solved by this design pattern is that changes to the underlying Model can update to any number of Views, without changed Views needing to have knowledge of one another. The Model-View-Controller pattern decouples of the presentation of the information from the information itself, a concept which was important to the development of the *AmazingStoke* web-based system with multiple users accessing a single shared persistent data structure.

### 3.2.4 Identifying Different Views

In applying the Model-View-Controller pattern to the *AmazingStoke* proposed system, it is clear to see that there are two main Views of the underlying data Model. There is the **NFP Site** and the **Game**, both providing separate Views (and incorporating different Controller mechanisms) of the underlying data Model.

It is also possible to see from the Use Cases that furthermore there are a number of different Views of the underlying persistent data model within the **Game** part of the system, including:

- **Map View**
- **Buy Buildings View**
- **NFP List View**
- **Challenges List View**

Finally, it can also be seen that **Players** can also access three different variations of the **Challenges** objects in the system. These three different variations of Views on **Challenge** objects held in the Model are:

- **Challenges List (All)**
- **Challenges List (Specific NFP)**
- **Challenges List (Player)**



### 3.2.5 Identifying Core Functionality

In the development of a modular system, it was helpful for design purposes to identify the key functionality which should be developed first, with the potential for further functionality to be added in according to time available.

Identification of the key components of the propose *AmazingStoke* system was carried out in the initial project proposal, and restated below.

- Database of **Players** and their **Maps**, **NFPs**, **Building** types and supporter **Challenges**
- A view of the proposed system for **Players** as a Facebook app
- A stand-alone site for **NFPs** that allows the addition of new supporter **Challenges** to the **Database** and validation of supporter **Challenges** completed by **Players**
- An ability for **Players** to link to each other as friends
- An in-game messaging system and **Challenge**-setting system
- A “leader board” as part of Facebook app for **Players**
- Potentially, pop-out browser-based games accessed by clicking on the buildings in a town map – for example, clicking on a “School” building placed on a map could launch a word puzzle game, or clicking on “Bank” building on a map could launch a maths puzzle game, in new windows.

The most important modules are:

- the **Database** module
- the **NFP Site** accessing the **Database**
- the **Player** Facebook app accessing the **Database**.

Modules of lesser importance include:

- linking and messaging facilities
- leader board
- pop-out games

### 3.2.6 Graphics

As stated in the original proposal, given that the aim of this project was to develop software design and development skills. As I am not an expert in graphic design, the graphics for this project were fairly rudimentary as spending a large amount of time on design would have taken time away from programming and development of functionality.

A set of five basic **Building** images was created using and were used to illustrate the functionality of the system. However, the *AmazingStoke* system was designed in such a way that more aesthetically pleasing visuals could be easily “added in” to the system by designers. This issue will be further addressed in **Chapter 6**.

## 3.3 NFP Site Design

This section of the Analysis and Design relates to the creation of the detailed design model for the **NFP site**. To this end, a Use Case developed in **Section 3.2** was analysed in greater detail to determine necessary functionality and interactions.

The Domain Model created in **Section 3.2.2** allowed the identification of the conceptual classes which used by the **NFP site** section of the proposed *AmazingStoke* system. These are:

- **NFP, NFP Site, Challenge, Voucher Codes**

### 3.3.1 Use Of A Database

As discussed in earlier sections, a persistent underlying data structure was necessary to store information between sessions. From this point onwards, it was assumed that this some form of **Database**, with exact specifications fixed at a later point (see **Section 3.5**).

This design section also assumed that the **NFP** has a permanent record already stored within the system by the time of logging in. As discussed in the initial project proposal and in earlier sections, the addition of new **NFPs** to the system falls under the heading of **System Administration Tasks**, i.e. managed by a site moderator and outside the scope of this project.

### 3.3.2 Use Of Sessions In NFP Site

It was also decided at this point that the **NFP site** should not only include password-protected login, but should also keep **NFPs** logged in to the site between pages. There are numerous ways of ensuring that login information is stored as users navigate pages within a website, which are be explored in further detail in **Section 3.4.1**. However, it was assumed at this stage of design that an appropriate method of storing user information between different pages of the site (referred to from now on as a Session) would be built into the system.

### 3.3.3 Adding A Challenge

Many of the Use Cases identified in **Section 3.1.4** involved simple CRUD (Create, Read, Update, Delete) calls to the underlying **Database**. However, the adding of a new **Challenge** to the **Database** (**Use Case 13**) required a more complex series of events. A System Sequence Diagram (**Figure 3.5**) was been created to show the events required for an **NFP** to log in to the **NFP site**, navigate to the Load a Challenge Form, enter the details, have details checked in-browser, and then written to the **Database**.

By drawing a System Sequence Diagram for this more complex Use Case, it was possible to extrapolate similar sequences of events for other CRUD operations on the **Database** by an **NFP** via the **NFP Site**.

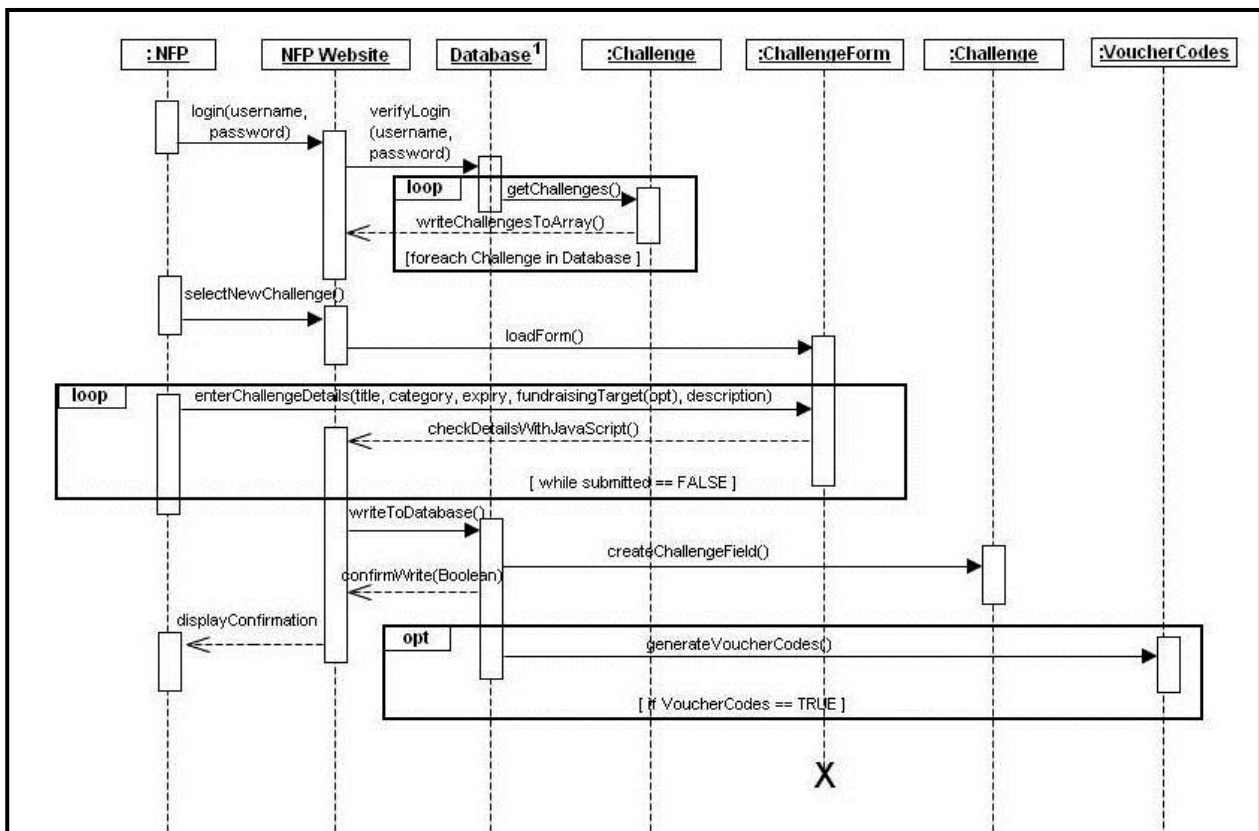


Figure 3.5: System Sequence Diagram for Use Case 13: Add New Challenge

### 3.3.4 Design Class Diagram (NFP Site)

Figure 3.6 shows a Design Class Diagram of the classes identified and developed in Section 3.3.3. Necessary attributes and operations for classes begin to emerge as the model develops. The attributes of the classes also began to suggest a structure for the underlying persistent data model (see Section 3.5 for more on this).

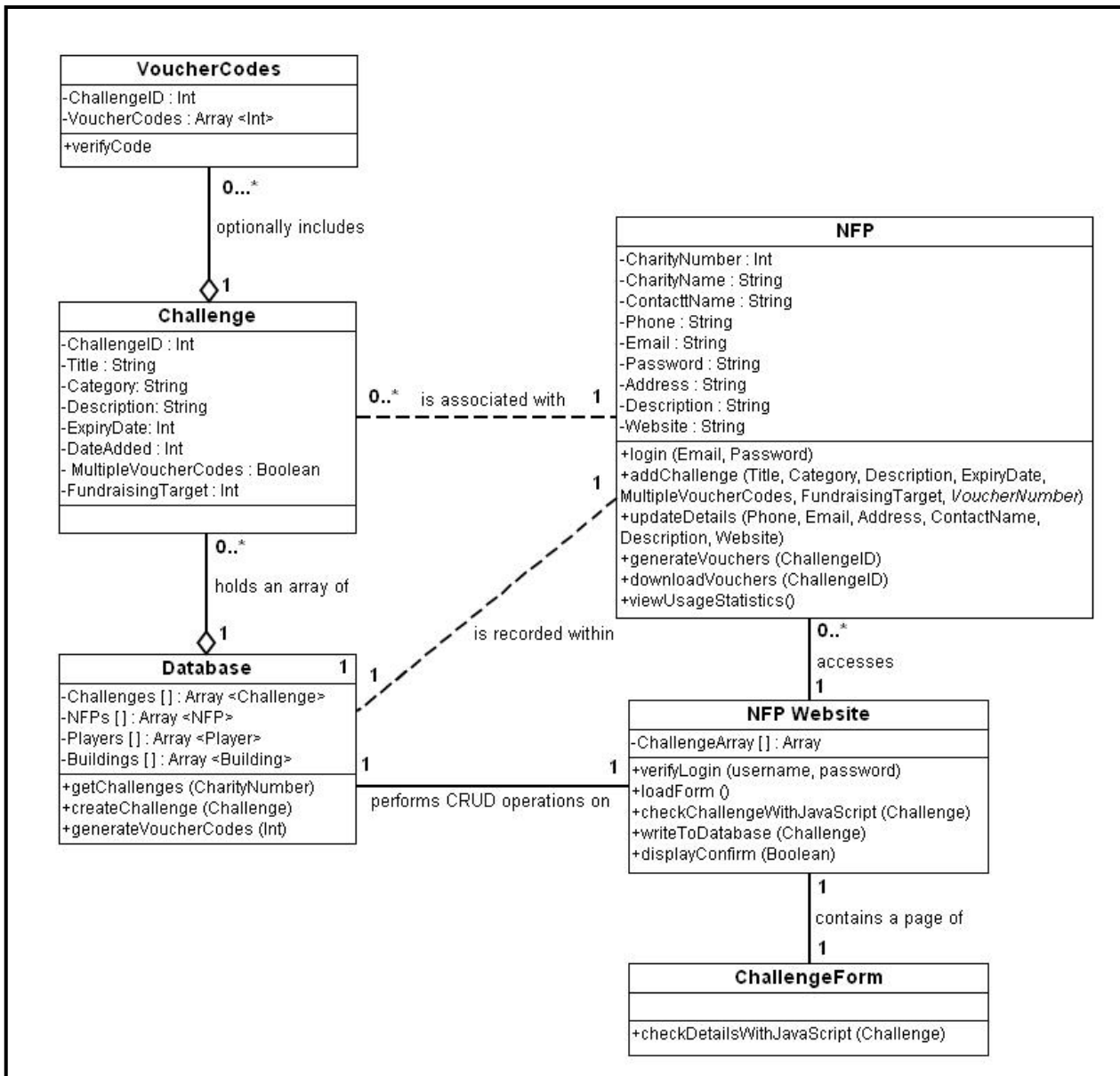


Figure 3.6: Design Class Diagram for NFP Site

## 3.4 Game Design

This section of the Analysis and Design relates to the creation of the detailed design model for the **Game** interface. As before, several of the Use Cases developed in **Section 3.1.4** were analysed in greater detail to determine necessary functionality and interactions, and the Domain Model created in **Section 3.2.2** allowed identification of the conceptual classes used by the **Game** section of the *AmazingStoke* system.

Conceptual classes involved in the **Game** section are:

- **Player, Game, Facebook Account, Map, MapSquare, Building**

As stated in **Section 3.3.1**, it was assumed that some form of **Database** provided a persistent underlying data structure accessed by the **Game**.

### 3.4.1 Use of Sessions In Game

It is also assumed that, as discussed in **Section 3.3.2**, the website hosting the **Game** interface viewed via Facebook uses sessions technology to keep **Players** logged in as they navigate to different parts of the **Game**.

Although this will be discussed in more detail later in the report, a key part of the authentication will be that it is done using information from a **Player**'s Facebook account, rather than requiring a separate username and password as necessitated by the **NFP Site**.

### 3.4.2 Logging In To Game

**Figure 3.7** shows a System Sequence Diagram for **Use Case 1: Login To Game**. It is assumed that the **Player** has a Facebook account. Although some successful Facebook games do also offer alternative version of the game via stand-alone sites, such as the game *FarmVille* (Zynga Games Inc., 2011), this would be beyond the scope of the current project. As an area for further development, please see **Chapter 6**.

This System Sequence Diagram also shows the first decision about whether to host information client-side or server-side. In this design diagram, a **MapArray** object is created within the browser using server-side scripting, and the information regarding the **Buildings** on **MapSquares** is stored within this array. In practical terms, this meant that rather than reloading of the page when a change was required, the **MapArray** object could be dynamically modified in-browser, creating a more seamless user experience and fewer time-consuming calls to the remote server.

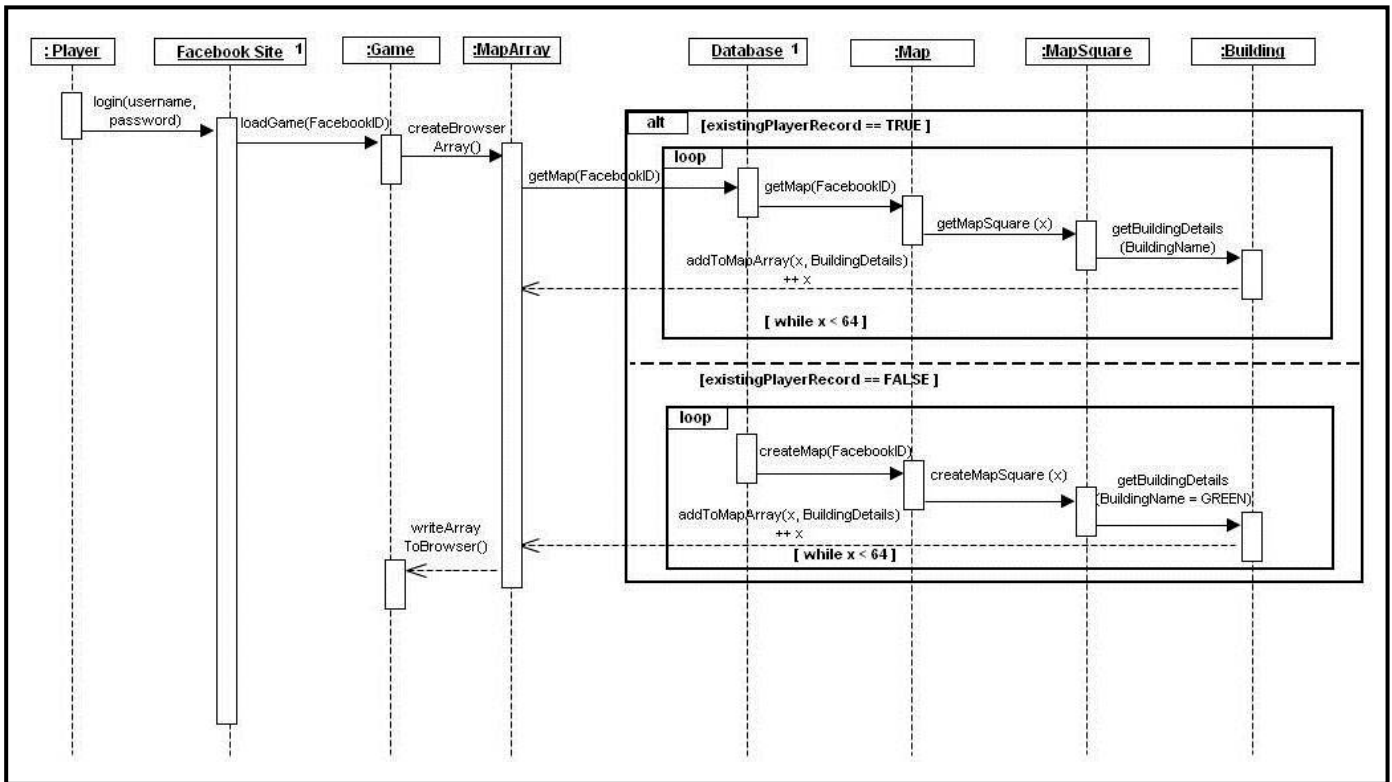


Figure 3.7: System Sequence Diagram for Use Case 1: Login To Game

### 3.4.3 Buying Buildings

**Figure 3.8** shows a System Sequence Diagram for **Use Case 3: Buy Building**. In this example, it was assumed that the **Player** had already successfully logged in to the **Game** as described in **Section 3.4.2**.

This System Sequence Diagram incorporates a similar design decision to the one discussed in **Section 3.4.2** – namely, what functionality should be handled client-side and what should be handled server-side.

To purchase a **Building** for an *AmazingStoke* town map, it was necessary to display a list of the **Buildings** stored in the underlying data structure, along with a mechanism for the **Player** to select one of these **Buildings** (such as submitting data via a form or by clicking on a specific image). As discussed in the Use Cases, I had decided that this would bring up a new **View** of the data – that is, the **Map** will not be visible whilst the **Buildings** are displayed.

However, the two different types of scripting – client-side and server-side – presented different possible solutions for this.

*Server-side scripting* would require a call from the client to the remote server to download a complete new page, complete with call to the database, each time the **Player** instructed the **Game** to switch to the **Buying Buildings View**. In this **Use Case 3: Buy Building**, two calls to the server would be required once the initial page has loaded – one call to load the **Buying Buildings View**, and one call to load the **Map View** once the **Player** has completed this sequence of events.

*Client-side scripting* provided a different approach to this. By loading a greater amount of information from the server in the initial loading of the page, then using client-side scripting to dynamically rewrite the page within the browser, it would appear to the **Player** that different pages are being displayed when in fact no further calls to the server were being made. Generally speaking, networking operations are the most time-consuming operations of a system (Yahoo! Inc., 2011). Therefore, reducing the calls to the server from two to zero would speed up the **Player** experience and reduce workload on the server. It was hoped that this would work towards two of the three most important **NFP** user requirements (see **Section 3.1.2**) of “User-friendly design” (rated 4.33 out of 5.0) and “Fun for users” (rated 4.17 out of 5.0).

There are different methods by which the View could be changed through the user of client-side scripting are further discussed in **Chapter 4**. At this point, however, it is assumed that the **Game** switches between the **Map View** and **Buying Building Views** using client-side scripting.



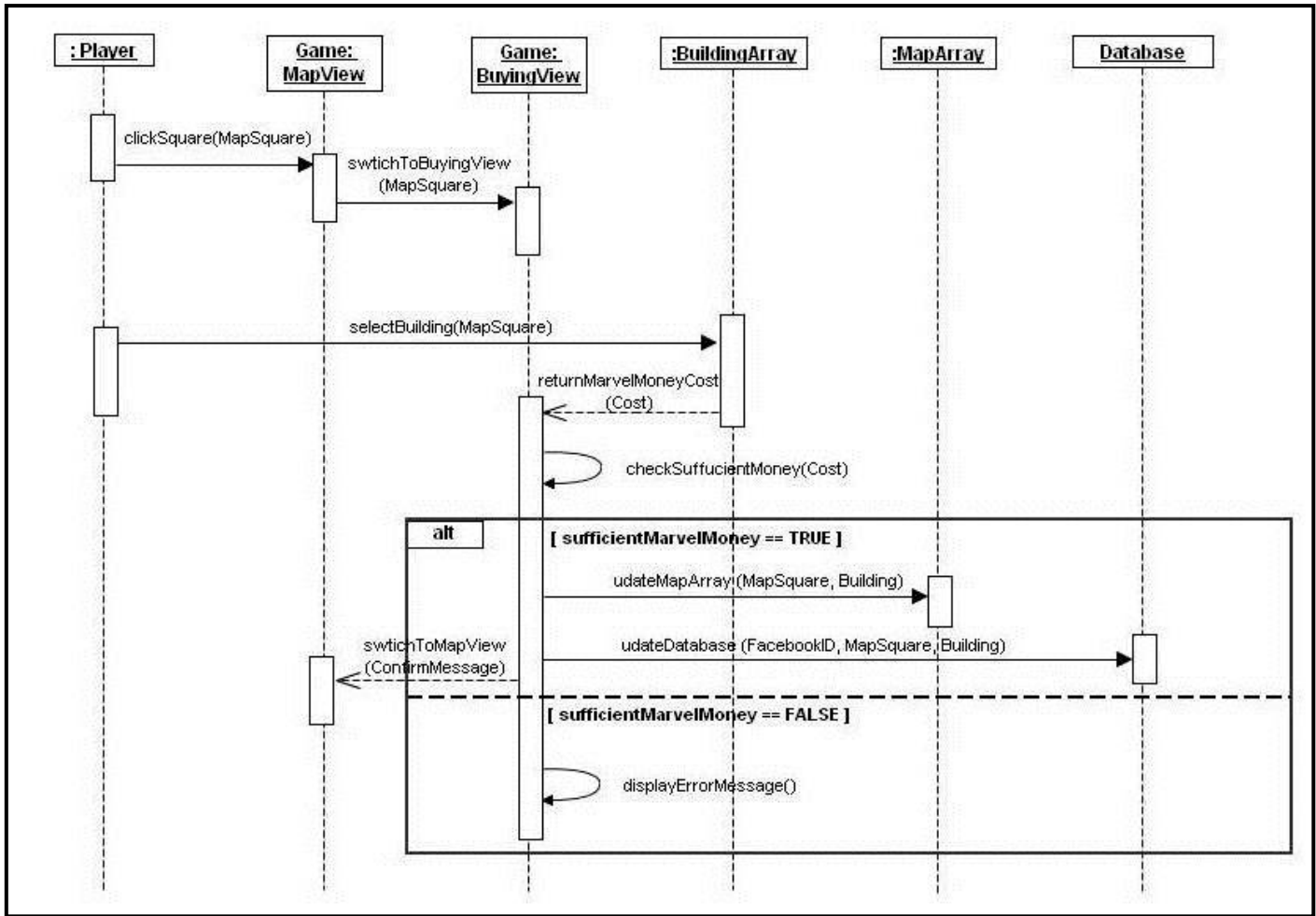


Figure 3.8: System Sequence Diagram for Use Case 3:Buy A Building

The use of server-side scripting to switch between the Map View and the Buying Building View would mean that whilst it was possible to make the changes to the MapArray object in the browser, this would not result in the change being written to the Database. For this, the inclusion of additional Ajax technology would be necessary (see **Section 4.3.2**).

### 3.4.4 Authenticating A Challenge

Figure 3.9 shows a System Sequence Diagram for Use Case 7: Authenticate Challenge. In this example, it was assumed that:

- The **Player** had already successfully logged in to the **Game** as described in Section 3.4.2
- The **Player** had previously successfully completed Use Case 5: Choose New Challenge
- The **Player** had received a **Voucher Code** from the **NFP** that added the **Challenge** to the *AmazingStoke* system

It was assumed that the **Challenge List View** shown in the System Sequence Diagram included an HTML form which allowed a **Player** to enter the **Voucher Code** as a string.

This Use Case required further design decisions related to what functionality would be handled client-side and what should be handled server-side.

As in Use Case 3: Buy Building, it would be theoretically possible to load the information relating to all tasks and their authentication codes into the browser, and then use an Ajax call to the server if the correct information is entered.

However, this has security issues as all possible voucher codes would need to be loaded as information into the browser on the client-side. This would be in contravention of the high-importance user requirement “Information security and data protection” (rated 4.08 out of 5.0 importance in research questionnaire, see Section 3.1.2). Therefore, in this case, it is necessary to send the **Voucher Code** to the remote server for checking against the **Database**, rather than attempting to authenticate using information stored client-side.

This would require an increased number of calls to the server. However, to minimise calls to the server, the workload could be split as follows:

- Information about **Challenges** undertaken loaded into data structures on client-side, to allow swift and fluid viewing of **Player** information with the **Game**
- Once a **Voucher Code** is entered and submitted via a form from the **Challenge List View** would a call be made to the **Database** on the remote server to verify the **Voucher Code**

In this way, the maximum amount of processing of information would take place on the client-side, to reduce calls to the remote server and thus speed up the user experience. However, for security reasons, in this case a call to the remote server was considered essential.

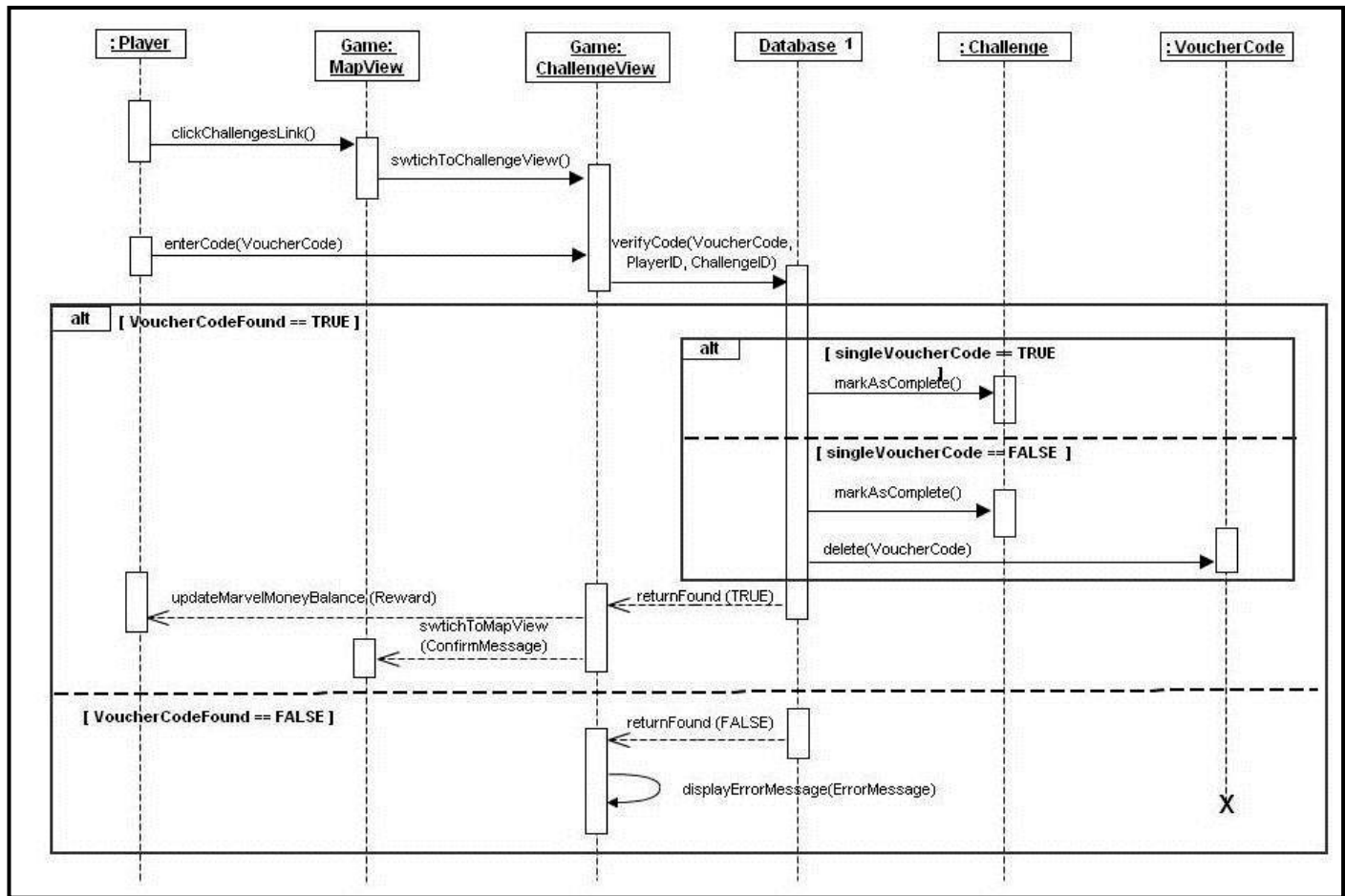


Figure 3.9: System Sequence Diagram for Use Case 7:Authenticate Challenge

### 3.4.5 Viewing Charity Information & Viewing Tasks

Two other key Use Cases identified for Players in Section 3.1.4 were Use Case 5: Choose New Challenge and Use Case 9: Find Out More NFPs.

Both of these Use Cases deliver key functionality to the **Game**. However, from the detailed assessment of Use Case 3: Buy Building and Use Case 7: Authenticate Challenge, it was possible to see that Use Case 5: Choose New Challenge and Use Case 9: Find Out More NFPs both follow similar steps to those already explored in more details.

That is, when the **Game** is loaded via the **Facebook Site**, a call is made to the **Database** to retrieve the initial information about **NFPs** and their **Challenges**. This information is then held in data structures or objects within the browser on the client side.

Alternatively, this information could be retrieved from the **Database** each time the **Player** switches to this **View (Challenge List, NFP List)** as a separate page called from the server. This greater use of server calls potentially makes the gaming experience slower and thus less enjoyable for **Players** of the **Game**.

A proposed breakdown of information and functionality managed the browser evolved as follows:

#### *Client-Side*

Loaded into the browser at the time of the initial call with client-side scripting to switch between views of this information:

- Switching between different views of information, i.e. a single page within the side using client-side scripting to show or hide different sections of the page
- **Map** information (with behind-the-scenes calls to the server when new **Buildings** are bought)
- **Building** information
- **NFPs** information (i.e. NFP Users of the *AmazingStoke* system)
- **Challenges**, both that a **Player** has accepted and **Challenges** listed by **NFPs** (with behind-the-scenes calls to the server when new **Challenges** are taken on)

#### *Server-Side*

- Server call when **Game** is first loaded via the **Facebook Site**
- Ajax calls to the server when a new **Building** is purchased
- Ajax calls to the server when a new **Challenge** is accepted
- Call to the server using HTML form when a Challenge is authenticated using a **Voucher Code**

### 3.4.6 Design Class Diagram (Game)

Figure 3.10 shows a Design Class Diagram of the classes identified and developed in Section 3.4.

As with the Design Class Diagram in Section 3.3.4, necessary attributes and operations for classes began to emerge as the model developed. The attributes of the classes also began to suggest a structure for the underlying persistent data model (see Section 3.5 for more on this).

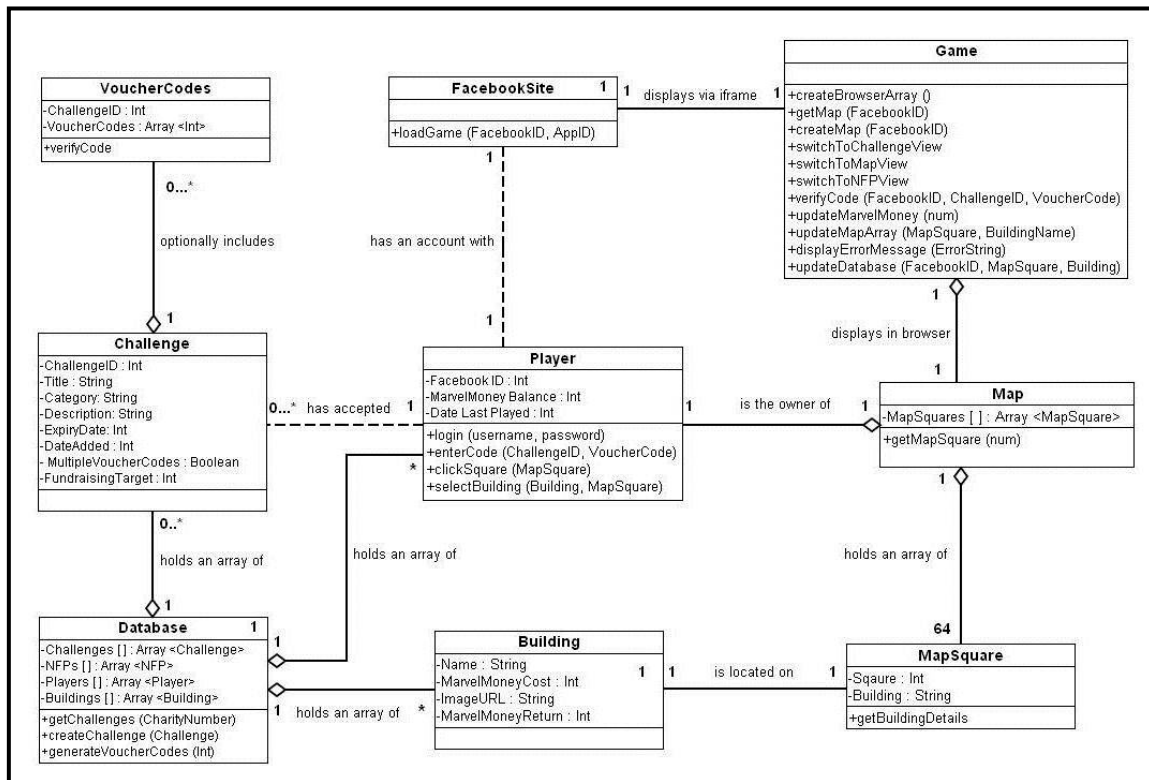


Figure 3.10: Design Class Diagram for Game

## 3.5 Database Design

As discussed previously, it was necessary to store the underlying data in a persistent format. I decided to use a relational database, and this section discusses the relations created for the system.

The previous design activities had identified the information that would need to be held by the system on various conceptual classes. The data fields needed for a **Player** included the following:

- **Facebook ID, MarvelMoney Balance, Date Last Played, Challenge IDs** (multiple repeated data), **Map Square Contents 1-64**

The data fields needed for an **NFP** included the following:

- **Charity Number, Charity Name, Contact First Name, Contact Surname, Phone Number, Email Address, Password, Address Line 1, Address Line 2, Address Line 3, Town / City, County, Postcode, Charity Description, Charity Website**

The data fields needed for a **Building** included the following:

- **Name, MarvelMoney Cost, Image Name, MarvelMoney Earnings**

The data fields needed for the **Challenges** included the following:

- **Challenge ID, Charity Number, Date Listed, Date Expires, Title, Category, Description, Proof, MarvelMoney Reward, Voucher Codes** (multiple repeated data), **Live** (Boolean value)

At this point, it became necessary to make a design decision about whether a **Challenge** that uses only one multiple-use **Voucher Code** would use a **Database** field for a separate **Voucher Code**, would use a separate relation to hold the single **Voucher Code**, or whether some other system would be used.

I decided that, in the main **Challenge** record, one field would hold a Boolean value, recording TRUE for **Challenges** which use multiple **Voucher Codes**, and FALSE for **Challenges** authenticated by a single **Voucher Code**.

In the situation where a **Challenge** could be authenticated by a single **Voucher Code**, it made greater sense in terms of space efficiency to re-use an existing piece of data held about the **Challenge**. In this case, I made the design decision that where multiple **Voucher Codes** were *not* used, the **Challenge ID** would be used for authentication, and a secondary relation of voucher codes would *not* be created for these **Challenges**.

Relational databases may be optimised for space efficiency via a process known as normalisation (Ramakrishna & Gehrke, 2003). Once the data had been converted to 3rd Normal Form (3NF), the database relations were as follows:

### **Players**

Facebook ID
MarvelMoney Balance
Date Last Played
MapSquare1
...
MapSquare64

### **PlayerChallenges** (*one per Player*)

Facebook ID
Challenge IDs 0..*

### **Challenges**

Challenge ID
Charity Number
Date Listed
Date Expires
Title
Category
Description
Proof
MarvelMoney Reward
MultipleVoucherCodes
Live

## NFPs

CharityNumber  
CharityName  
ContactFirstName  
ContactSurname  
Phone  
Email  
Password  
Address1  
Address2  
Address3  
TownCity  
County  
PostCode  
Charity Description  
Website

## Buildings

Name  
MarvelMoney Cost  
Image Name  
MarvelMoney Earnings

## VoucherCodes (*one per Challenge, certain Challenges only*)

Challenge ID  
VoucherCode1  
...  
VoucherCode\*



## **Chapter 4: Implementation**

This chapter covers the iterative process of implementing the design ideas of **Chapter 3** to create the *AmazingStoke* system. The three main components of this system, each of which underwent an iterative development cycle, were:

- **Database** of **NFP**, **Player** and system data
- Web interface for NFPs to manage their *AmazingStoke* accounts (referred to from this point on as **NFP Site**)
- Web-based game for players accessed as an app via Facebook (referred to from this point on as **Game**)

As explained in the **Chapter 3**, an agile and iterative methodology was employed in the development of the *AmazingStoke* system, meaning that this chapter will discuss how various simple sections of code were written and tested, building up to greater complexity with user testing on a live site taking place throughout the process.

## 4.1 Technologies

As discussed in the initial project proposal, three key types of technology were employed in the creation of the initial version of the *AmazingStoke* interactive web-based system. The three key technologies required were:

- Persistent data storage mechanism, in this case a **Database**
- Server-side scripting language
- Client-side scripting language

The specific technologies chosen for each of these categories are described briefly below.

### 4.1.1 Database: SQL / MySQL Database

For persistent data storage, I chose to use a relational database management system (RDBMS). Having studied them extensively throughout the course, this seemed the logical choice for data storage.

The MySQL RDBMS is an open-source database used by major web-based products such as Facebook, Google and Adobe (MySQL, 2011). It is one of the key components of the LAMP (Linux, Apache, MySQL, PHP / Python / Perl) and WAMP (Windows, Apache, MySQL, PHP / Python / Perl) development stacks, which were technologies I was keen to explore in this project.

It is a widely-used technology for websites of which I already had a sound understanding, and it seemed an obvious choice for the development of the *AmazingStoke* system.

### 4.1.2 PHP

Server-side scripting refers code run on a web server when a user requests a page, which results in dynamically-generated web pages. There are a variety of programming languages which can be used for this, including but not limited to Ruby on Rails (Heinemeier Hansson, 2003), Java via JavaServerPages (JSP) technology (Sun Developer Network (SDN), Oracle, 2010), Active Server Pages (ASP) (Microsoft, 2011) and more.

I chose early on in the process to use PHP (PHP Hypertext Preprocessor). PHP is a web-scripting server side language, which runs on multiple (Argerich, et al., 2002). It is an open source language with broad functionality for databases, strings, Java, XML and other web technologies (Argerich, et al., 2002). As someone with prior programming experience of C++, the syntax of PHP seemed relatively simple to learn.

PHP programs use the file extension .php, and PHP instructions are enclosed in delimiters so that browsers will run the PHP scripts. PHP instructions are enclosed within:

```
1. <?php
```

```
2. ...
3. ?>
```

The `echo` command in PHP can be used to output text from the PHP program within delimiters to the browser. For example, the following code segment:

```
4. <?php
5. echo "Hello World!"
6. ?>
```

will write the text “Hello World!” in the browser window. PHP also utilises the `<<<` operator, known as *heredoc* for output to browser of string literals preserving line breaks and whitespace. For example:

```
1. <?php
2. echo <<<_END
3. This is the start of the heredoc output.
4. Hello World!
5. This is the end of the heredoc output.
6. _END;
7. ?>
```

Note that `_END` is a tag chosen by the developer. Convention is to use an underscore preceding the heredoc section name, but this is not essential.

Different developers use different conventions for the embedding of HTML and PHP within web pages. Some developers prefer to use a minimum of PHP within a file, dropping in and out as required, whilst others prefer to simply open and close the `<?php...?>` tag once. For example, both of the following are valid:

```
1. <?php
2. $username = "Joanna Pinto";
3. echo "Hello $username!"
4. ?>
```

```
1. Hello <?php $username = "Joanna Pinto";?>
2. $username!
```

In this project, I chosen mostly to use the opening and closing `<?php...?>` tag once, as this seemed simpler to me as a novice PHP user.

Variables in PHP are identified using the \$ (dollar) sign, and must always include this at the beginning of any reference to the variable. PHP is a loosely typed language, and variables do

not need be declared before use. PHP supports both functions and object-oriented programming.

PHP has a library of functions relating to the MySQL database system. An initial call to connect to a database from a PHP program is made using:

```
1. mysql_connect(servername, username, password);
```

A PHP string can then be passed as an SQL query to the currently active database using the following function:

```
1. mysql_query(querystring);
```

If a return variable is required, such as a set of rows resulting from a SELECT query, an SQL resource is returned to the browser. This can then be manipulated using further functions from the PHP MySQL library. For example:

```
1. $queryString = "SELECT * FROM cats";  
2. $returnedTable = mysql_query($queryString);
```

Information can then be extracted using functions such as:

```
1. $numberOfRows = mysql_num_rows($returnedTable);  
2. for ($i = 0; $i < $numberOfRows; ++ $i)  
3. {   $currentRow = mysql_fetch_row($returnedTable);  
4.     echo "$currentRow[0] <br />";  
5. }
```

PHP was a new language to me at the start of this project, and therefore it was necessary to allocate time to learn to use this language. A book that was highly useful in the learning of PHP and MySQL (as well as JavaScript) was *Learning PHP, MySQL and JavaScript* (Nixon, 2009).

### 4.1.3 JavaScript

JavaScript is an object-oriented client-side scripting language, first seen in Netscape Navigator browser in 1995 (Nixon, 2009). It is an implementation of the ECMAScript language standard. Despite its name, it is not related to the Java language. It allows for changes to be made to a browser document without a call to the server to reload the page. It runs within the HTML of a web page within `<script>` and `</script>` tags.

At the heart of JavaScript is the Document Object Model, or JavaScript DOM. The DOM was covered during the teaching of the MSc course, and as stated in **Section 1.3**, it is assumed that the reader has an understanding of this technology (Jacobs, 2006). It should be stated at this point, however, that there are several difficulties which frequently arise in the use of JavaScript.

One of these is the problems that the JavaScript DOM is implemented differently in Internet Explorer from other browsers such as Firefox or Chrome. When there are specific issues related to this in the development of the *AmazingStoke* system, this will be explicitly described in this report.

A second problem is that not all users chose to allow JavaScript to run on their computers. There are different approaches to dealing with this potential situation, and programmers may choose to include alternative methods of generating results when JavaScript is not allowed. Given the nature of the **Game**, it seems impractical to develop an alternate version for those users who have disabled their JavaScript, as JavaScript will be used to implement core functionality. However, a potentially useful consideration would be a JavaScript-free version of the **NFP Site**, where JavaScript is intended that JavaScript creates a more seamless user experience but is not essential to the site.

#### **4.1.4 WAMP Development Stack**

As outlined in **Section 4.1.1**, the selection of PHP and MySQL as development tools for the *AmazingStoke* system allowed for the use of a WAMP (Windows, Apache, MySQL, PHP / Python / Perl) development stack. The use of a solution stack was discussed in the initial project proposal, however in summary a WAMP package binds the named programs together as a development environment on one machine so that they do not need to be set up separately (Nixon, 2009). Use of a WAMP package can significantly simplify and speed up the building of a system that uses a range of different components.

I chose to use the EasyPHP (EasyPHP, 2011), a free-to-download WAMP utilising PHP 5 and incorporating the database manager phpMyAdmin and the debugger Xdebug. phpMyAdmin allows browser-based manipulation of SQL databases, and significantly simplifies the creation of the system's relational Database.

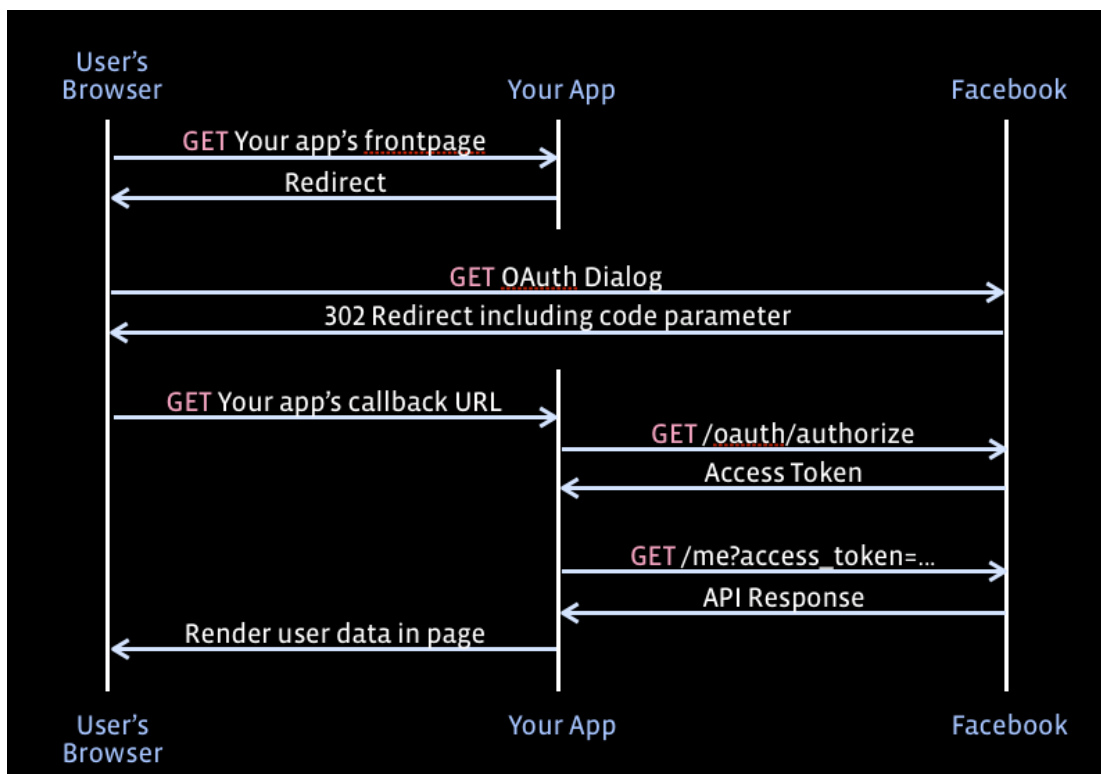
#### **4.1.5 Facebook API**

As the Game section of the *AmazingStoke* will be hosted within the Facebook site, it is relevant to include a discussion of the Facebook API.

The Facebook Developers Group is a site hosted within Facebook that gives extensive instructions on creating a web-based application to be hosted within Facebook (Facebook Developers, 2011). As explained in the initial project proposal, Facebook apps are loaded into a Canvas Page. The app developer provides a Canvas URL that contains the web-based application hosted on an external site. When a Facebook user requests the Canvas Page, Facebook loads the Canvas URL within an `iframe`, resulting in the app being displayed within the Facebook chrome. Once a user has authorised an app, a bookmark (small redirect icon) to it is added to that user's Facebook homepage.

The Facebook Developers site lists the three main concepts of using Facebook as an integrated host for an app are the authentication via Facebook, the Graph API, and the Social Channels (Facebook Developers, 2011).

Authentication uses the OAuth 2.0 protocol to authenticate users accessing the app, and to authorise the app. When a Facebook user authorises an app, the default settings means that app can access basic information that is available publicly such as the user’s name and the Facebook ID number (either a 9-digit or 15-digit unique numerical identifier). **Figure 4.1**, taken from Facebook Developers, shows the flow of information when a server-side call is made to access or authorise the app.



*Figure 4.1: The flow of data in authentication of a Facebook user for an app when a server call is made (Facebook Developers, 2011)*

The Graph API refers to what Facebook terms its “social graph”. Facebook Developers states: “The Graph API presents a simple, consistent view of the Facebook social graph, uniformly representing objects in the graph (e.g., people, photos, events, and pages) and the connections between them (e.g., friend relationships, shared content, and photo tags)” (Facebook Developers, 2011). Every object in this “social graph” has a unique identification number. By using the Graph API to access the other objects within the Facebook site linked to the user, it is possible to create a Game with complex social networking functionality enabled by existing Facebook technology. For example, it should be possible to utilise the Graph API to connect Players to other Players who are already linked as “Friends” on Facebook, without needing to develop a new system for this. Significant parts of the *AmazingStoke* proposed system could be implemented using standard Facebook functionality.

Finally, the Social Channels allow Facebook users to share information regarding their app usage with other Facebook users. These include:

- publishing information about a user via the *News Feed* (the page that a user sees when they initially log in to Facebook) using the *Feed Dialog* (which suggests text the users may wish to publish on their Walls as news stories) and the *Like* button (which allows a user to click a single button to share information from an app on their Wall)
- using *Requests*, which can enable users to notify their Facebook friends to take a specific actions within an app
- the *Automatic Channels*, which Facebook generates regarding app usage including *Bookmarks, Notifications, Usage Stories* and *App Profiles & Search*.

The details of precisely how to make an app accessible via Facebook will be covered in **Section 4.2.3**. The Facebook Developers site gives useful code examples for this in PHP, which are used and accredited accordingly.

It is worthy of mention that an issue that arose with this project was that Facebook changed the way that apps were hosted within the site during the course of this MSc. Prior to March 2011, interactions between the Facebook site and apps had been handled using the Facebook Markup Language (FBML). After March 2011, however, new apps could not use FBML and instead were loaded into Facebook using `iframe` technology and the Facebook API. Unfortunately, this meant that by the time of implementation of the *AmazingStoke* proposed system, many of the resources available regarding the creation of Facebook apps (such as the book *How To Do Everything: Facebook Applications* (Feiler, 2008)) and information on various Facebook app development forums (including some of the advice on the Facebook Developers site itself) was obsolete.

Whilst this did not make the project impossible, having limited resources to draw on in learning how to develop a Facebook app did make this project more complicated than originally envisioned. It was an important learning point that there can be significant risks as well as benefits in designing a web-based application integrated into a larger external system.

## 4.1.6 Hosting

As noted in **Section 4.1.2**, I intended to use EasyPHP for the development of the *AmazingStoke* system. However, it became apparent very early on that whilst this was feasible for the **NFP Site** and the **Database** could be developed and tested on the local development server, the **Game** needed to be hosted on a live site in order to test the integration and functionality within Facebook.

I therefore decided to purchase web hosting from the company One.com, which offered web hosting including PHP 5 support and an integrated MySQL database accessed via the same phpMyAdmin interface as my chosen WAMP (One.com, 2011).

My approach in this project was therefore to use the WAMP as a development and testing environment for the **Database** and **NFP Site**, together with some aspects of the **Game**, before migrating these to the live site hosted by One.com, with the URL `www.astoke.co.uk`



## 4.2 First Iteration Of The System

This section of the report concerns the development of the first iteration of the *AmazingStoke* proposed system. Various “proof-of-concept” systems were developed using simple methods, to test ideas and to learn various features of the PHP and JavaScript languages, as well as to explore the Facebook API.

### 4.2.1 First Iteration Of The MySQL Database

The design of the **Database** was described in detail in **Section 3.5**. The creation of the necessary relations utilising phpMyAdmin saved significant time over using manual SQL CREATE statements to define the necessary relations.

It should be noted that the TINYINT data type is used to record a Boolean value (0 for FALSE, 1 for TRUE).

In the first iteration, the relations were created as follows:

#### **Users1**

```
playerID varchar(32), sq1 varchar(32), sq2 varchar(32), sq3  
varchar(32), sq4 varchar(32), sq5 varchar(32), sq6  
varchar(32), sq7 varchar(32), sq8 varchar(32), sq9  
varchar(32), sq10 varchar(32), sq11 varchar(32), sq12  
varchar(32), sq13 varchar(32), sq14 varchar(32), sq15  
varchar(32), sq16 varchar(32), sq17 varchar(32), sq18  
varchar(32), sq19 varchar(32), sq20 varchar(32), sq21  
varchar(32), sq22 varchar(32), sq23 varchar(32), sq24  
varchar(32), sq25 varchar(32), sq26 varchar(32), sq27  
varchar(32), sq28 varchar(32), sq29 varchar(32), sq30  
varchar(32), sq31 varchar(32), sq32 varchar(32), sq33  
varchar(32), sq34 varchar(32), sq35 varchar(32), sq36  
varchar(32), sq37 varchar(32), sq38 varchar(32), sq39  
varchar(32), sq40 varchar(32), sq41 varchar(32), sq42  
varchar(32), sq43 varchar(32), sq44 varchar(32), sq45  
varchar(32), sq46 varchar(32), sq47 varchar(32), sq48  
varchar(32), sq49 varchar(32), sq50 varchar(32), sq51  
varchar(32), sq52 varchar(32), sq53 varchar(32), sq54  
varchar(32), sq55 varchar(32), sq56 varchar(32), sq57  
varchar(32), sq58 varchar(32), sq59 varchar(32), sq60  
varchar(32), sq61 varchar(32), sq62 varchar(32), sq63  
varchar(32), sq64 varchar(32), money int(13), lastPlayed  
int(8)
```

## AsCharities

```
CharityNumber int(6), CharityName varchar(128),  
ContactFirstName varchar(45), ContactSurname varchar(45),  
Phone varchar(32), Email varchar(128), Password varchar(16),  
Address1 varchar(4096), Address2 varchar(4096), Address3  
varchar(4096), TownCity varchar(64), County varchar(64),  
PostCode varchar(12), Blurb varchar(500), Image tinyint(1),  
Website varchar(512)
```

## AsCharityTasks

```
TaskID int(8), CharityNumber int(8), DateListed int(8),  
DateExpires int(8), Title varchar(32), Category varchar(32),  
DescriptionBlurb varchar(4096), ProofBlurb varchar(4096),  
MarvelMoney int(9), VoucherCodes tinyint(4), Live tinyint(1)
```

## Buildings

```
Name varchar(32), Cost int(7), Image varchar(256), Earnings  
varchar(128)
```

Relations for the individual **Player Challenges** and **Challenge Voucher Codes** were be generated dynamically as **Players** register for the game, and as **NFPs** add **Challenges** via the **NFP Site**. These manual creations will be explored in greater detail in the later sections.

An example of a **Player Challenges** relation, for a **Player** with playerID of 762451966, would be:

### PlayerTasks762451966

```
TaskID varchar(255), Completed tinyint(1)
```

An example of **Charity Voucher Codes** relation, for a **Challenge** with a ChallengeID of 123472, would be

### Task123472

```
Voucher varchar(255)
```

## 4.2.2 First Iteration Of NFP Site

This section covers the development of the **NFP Site** where **NFPs** could add **Challenges**, view existing **Challenges**, view **Voucher Codes** for existing **Challenges**, and change their organisational information.

Throughout the development of this section, examples of developing a social networking site are drawn significantly from *Learning PHP, MySQL and JavaScript* (Nixon, 2009). Any code adapted from that source text is clearly marked as such in both the code and this report.

### 4.2.2.1 Key Functionality

As described above, the key tasks an NFP would carry out via the NFP Site are:

- Add a **Challenge**
- View existing **Challenges**
- View **Voucher Codes** available for existing **Challenges**
- Change their organisational information.

I decided at this stage that the ability to generate additional **Voucher Codes** relating to an existing **Challenge** would be added in at a later iteration of the *AmazingStoke* system.

On arriving at the **NFP Site**, the **NFP** sees an information page, and a link to login to the site (`index.php`). On clicking the link, they are taken to an HTML form asking for an email address and password (`aslogin.php`).

On entering correct details, they are taken to a homepage (`ashome.php`) showing their live **Challenges** with links to view the **Voucher Codes** if multiple **Voucher Codes** have been generated. Additionally there are links to the forms to Add a **Challenge**, Update Organisational Information, and Logout. A site overview could be sketched as **Figure 4.2**. Rectangles with square corners represent statically generated pages, and rectangles with rounded corners represent pages generated dynamically based on the **NFP**'s data.

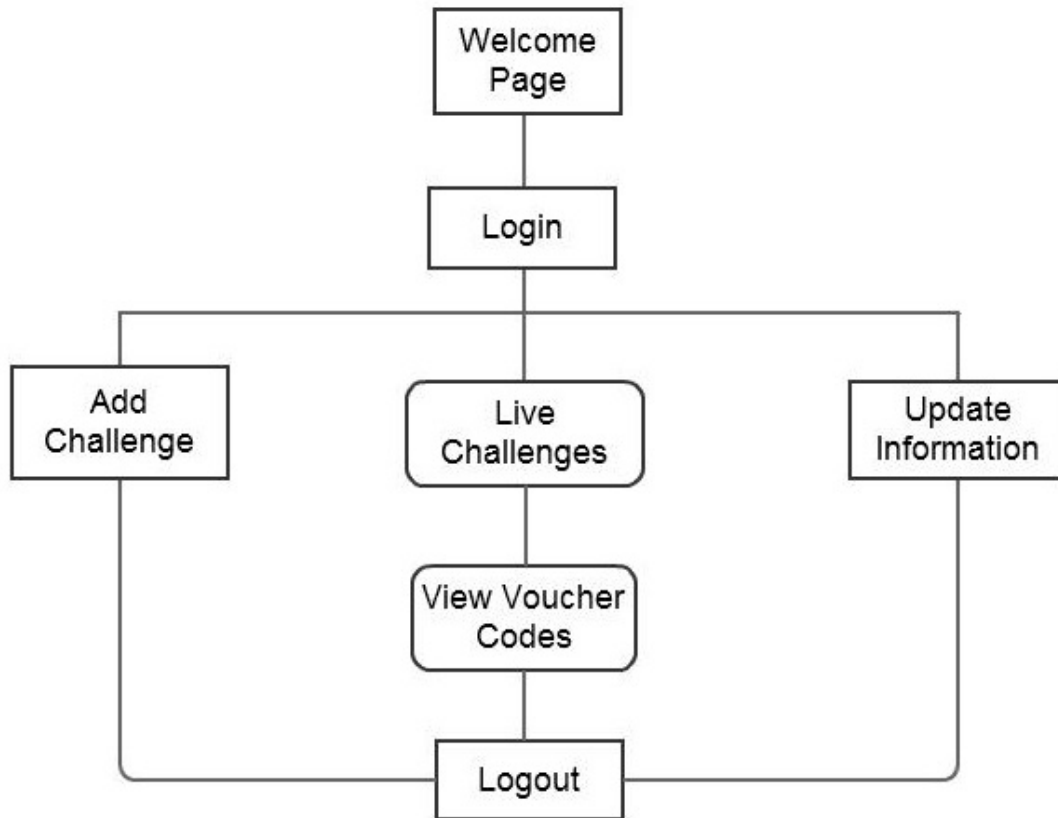


Figure 4.2: Site Overview, NFP Site

As discussed in **Section 3.3.2**, one of the features of the **NFP Site** would be that **NFPs** log in to the site using a username and password, then stayed logged in throughout their time on the site. Theoretically, it would be possible to pass information about the currently logged in NFP using GET and POST methods via the server, but PHP provides a more efficient solution.

To store information about users as they access different pages, PHP contains the ability to create sessions, which are collections of variables stored on the server about the current user. A cookie is saved in the client's web browser to identify the NFP's server-side saved information. If a user has cookies turned off, PHP will identify this and place the identifying information in the GET portion of URL requests instead (Nixon, 2009).

Session information is stored as a PHP superglobal variable, as an associative array. This array is always named `$_SESSION`, and values are set and retrieved using `[ ]` notation, for example:

```
1. $_SESSION['username'] = "Joanna Pinto";
```

To access information in session, the PHP function `session_start` must be called before any HTML is output. The existence of specific keys within the `$_SESSION` array can be checked using the `isset` function. For example:

```
1. session_start();
2. $_SESSION['username'] = "Joanna Pinto";
3. if(isset($_SESSION['username']))
4.     echo "Hello $_SESSION['username']!";
5. else echo "Oh dear, no username has been set.";
```

Variables within a session usually persist whilst the browser remains open. However, PHP contains the `session_destroy` function which, combined with the deletion of automatically stored information such as a timestamp and a user identifier, means that a user can “log out” of a site without having to close their browser (see **Section 4.2.2.7**).

#### 4.2.2.2 NFP Login Page With PHP And MySQL

To allow **NFPs** to log in to the **NFP Site**, a simple HTML form using the POST method was created, asking for the registered user’s email address and password. The PHP program (`aslogin.php`) was written so that once the form was submitted by the user, the login information provided was passed back to the same program within the `$_POST` superglobal array.

In this program, simple error checking is performed using the PHP `isset` function to ensure that both a username and password have been entered – if not, the login page is reloaded with an error message informing the user of this.

Once a username and password have been entered, the information is checked against registered **NFPs** using a call to the **Database**. For the sake of efficiency, a simple PHP program to establish a connection to the Database was written and entitled `db_login.php`. Also included in this program is a PHP function `queryMySQL`, which takes in a string, checks it against the Database, and returns the MySQL error message if there is an error in the SQL. This function is adapted from Nixon, 2009.

```
1. function queryMySQL($query)
2. { $result = mysql_query($query) or
3.     die(mysql_error());
4.     return $result;
5. }
```

This login file was included in each page of the site using the following PHP code at the start of each program:

```
1. <?php
2. include 'db_login.php';
3. ...
```

Once an **NFP** has submitted their email address and password, a query string is passed to the **Database**. If the email address and password match information held in the Database, the

information is set to the `$_SESSION` array and the user redirected to the View Current Challenges (`ashome.php`) page as follows:

```
1.  $query = "SELECT CharityNumber, Email, Password FROM
2.  Ascharities WHERE Email='$user' AND
3.  Password='$pass'";
4.
5.  $query_result = queryMysql($query);
6.  if (mysql_num_rows($query_result) == 0)
7.  { $error = "Email / Password invalid<br />";
8.  }
9.
10. else
11. { $row = mysql_fetch_row($query_result);
12.   $charityID = $row[0];
13.
14.   $_SESSION['user'] = $mailing;
15.   $_SESSION['pass'] = $pass;
16.   $_SESSION['charityID'] = $charityID;
17.
18.   header('Location:
19.   http://astoke.co.uk/charities/ashome.php');
20. }
```

#### 4.2.2.3 View Challenges Page

Upon entering valid login information, **NFPs** are directed to a welcome page giving further information about the **NFP Site**, showing details of the current logged in **NFP** and listing their current live **Challenges**. Links are also provided to the Add A Challenge, Update Information and Logout pages. This PHP program is saved as *ashome.php* and, as before, the file *db\_login.php* is included.

If the `$_SESSION` information has not been set, for example if a user has entered the URL of *ashome.php* directly into their browser without first logging in, this page simply displays an error message.

Otherwise, a call to the **Database** retrieves the relevant information for the specific charity. Information regarding **Challenges** which have been loaded to the system is displayed in HTML table format.

The table also contains form buttons for Challenges with multiple Voucher Codes. For each Challenge with **VoucherCodes** set to 1 (TRUE), a form is created within the table to pass the ChallengeID to the View Vouchers page (see **Section 4.2.2.6**). The following code demonstrates this:

```

1.   $vouchers = $currentRow[9];
2.   if ($vouchers=='1')
3.   { $vouchers = 'Yes';
4.     $viewVouchers = "<form action='viewvouchers.php'
5.       method='post'><input type='hidden' name='taskID'
6.       value='$taskID' /><input type='hidden'
7.       name='title' value='$title' /><input type='submit'
8.       value='View voucher codes' /></form>";
9.   }
10.
11.  else
12.  { $vouchers = 'No';
13.    $viewVouchers = "n/a";
14.  }
15.
16.  echo
17.  "<tr>
18.    <td>$title<td />
19.    <td>$description<td />
20.    <td>$marvelMoney<td />
21.    <td>$vouchers<td />
22.    <td>$viewVouchers<td />
23.  </tr>";

```

#### 4.2.2.4 Add Challenge Form

In the first iteration of the NFP Site, the Add Challenge Form was a simple PHP program incorporating an HTML form, with all areas of the form visible and error checking handled by PHP and error message PHP variables.

As discussed in the original project proposal, specific **Challenges** that an **NFP** could load would be worth specific amounts of **MarvelMoney** (in-game reward for **Players**). Therefore, within this Add Challenge page, the **MarvelMoney** value for the Challenge would be ascertained from the Category selected, and written to the **Database** by the system without the NFP selecting this. The exception to this would be if a **Challenge** was a Fundraising **Challenge**, in which case the **NFP** enters an amount via the form, and the **MarvelMoney** reward is set at the rate of 4 **MarvelMoney** per £1 raised.

The new **Challenge** is then inserted to the **Database** relation **AsCharityTasks** using the `queryMysql` function described in **Section 4.2.2.2**.

The creation of a relation of multiple **Voucher Codes** is achieved by first issuing a `CREATE` statement to the **Database**. However, as **Challenges** are assigned **ChallengeIDs** in ascending numerical order, a call must then be issued to the **Database** to find the **ChallengeID** of the **Challenge** just added:

```
1. $taskIDQuery = "SELECT TaskID FROM ascharitytasks
2.   WHERE Title='$title' AND CharityNumber='$charityID'
3.   AND DateListed='$datelisted'";
```

Once the ChallengeID has been ascertained using the result object returned from the above query string, a relation of **Voucher Codes** can be created as follows:

```
1. $tableName = "Task" . $taskID;
2. $voucherQuery = "CREATE TABLE $tableName (Voucher
3.   VARCHAR(255) )";
4. queryMysql($voucherQuery);
```

The **Voucher Codes** relation must then be populated with codes. I decided to create 6-digit verification codes, and to start each sequence from a pseudorandom number between 100,000 and 899,999.

The maximum number of initial **Voucher Codes** generated per **Challenge** is capped at 1,000. This was capped so as to ensure that **NFPs** did not use up large amounts of **Database** space by listing excessive number of **Voucher Codes**, most of which would be unused. However, by using an initial starting point starts guarantees at least 100,000 **Voucher Codes** can be generated, **NFPs** can generate a large number of additional **Voucher Codes** (as described later) whilst ensuring that no **Voucher Code** will be greater than maximum decimal 6-digit integer (999999).

Using a pseudorandom number to begin the sequence, rather than starting simply at 100000, minimises the ease of obvious potential “cheating” by **Players** by brute-force entry of **Voucher Codes** starting at 100000 for each **Challenge**.

A pseudorandom number between 100,000 and 899,999 is generated by the following PHP code:

```
1. $voucherStart = rand(100000, 899999);
```

The **Voucher Codes** relation is then populated by means of a simple for loop:

```
1. $voucherEnd = $voucherStart + $vouchers;
2.
3. for ($i = $voucherStart; $i < $voucherEnd; ++$i)
4. { $insertQuery = "INSERT INTO $tableName VALUES
5.   ($i)";
6.   queryMysql($insertQuery);
7. }
```



Finally, on successful insertion of the new challenge into the **Database**, and optionally the creation of a new **Voucher Codes** relation in the **Database**, the **NFP** is redirected to a confirmation page `asthanks.php`.

```
1.    if($approval) header('Location:
2.    http://astoke.co.uk/charities/asthanks.php');
```

#### 4.2.2.5 Update Details Form

The Update Details page provides a simple HTML form to perform an update on the **Database**. The PHP program for this was created as `aseditinfo.php`.

The code for this will not be described in detail as it required only a simple HTML form using an HTTP POST request to the same PHP program and a call to the **Database**. The code ensures that if a blank field is submitted in the form, no information is written for that field to the **Database**. As with previous pages, the login details for the **Database** are included at the beginning of the PHP program using:

```
1.    <?php
2.    include 'db_login.php';
3.    ...
```

On successful updating of the **Database**, the **NFP** is redirected to the confirmation page `asthanks.php`.

#### 4.2.2.6 View Voucher Codes

The program to display onscreen the available **Voucher Codes** for a specific **Challenge**, as accessed from the View Challenges page (*ashome.php*) was created as `viewvouchers.php`.

As described in **Section 4.2.2.3**, the `ChallengeID` is passed to this program by means of the HTTP POST, and accessed by the `viewvouchers.php` program using the PHP superglobal array `$_POST`.

**Voucher Codes** are then retrieved from the **Database** and displayed as an on-screen list using a single call to retrieve the data from the **Database**, then a `for` loop to display the returned **Voucher Codes** onscreen.

The code for this will not be described in detail. As with previous pages, the login file `db_login.php` is included in the `viewvoucher.php` program.

#### 4.2.2.7 Logout

The final PHP program required as part of the first iteration of the **NFP Site** is a simple program to log the **NFP** out by destroying the information set for the current session.

Adapted from Nixon, 2009, the `aslogout.php` is as follows:

```
1. <?php
2.
3. include 'db_login.php';
4. session_start();
5.
6. function destroySession()
7. { $_SESSION = array();
8.   if (session_id() != "" ||
9.       isset($_COOKIE[session_name()]))
10.      setcookie(session_name(), '', time()-2592000,
11.                '/');
12.   session_destroy();
13. }
14.
15. if(isset($_SESSION['user']))
16. { destroySession();
17.   echo
18.   "...
19.   Thank you. You are now logged out of the site.
20.   ...";
21. }
22. ?>
```

## 4.2.3 First Iteration of Game

This section covers the development of the **Game** where **Players** can add view their *AmazingStoke* town **Map**, new on new **Challenges**, authenticate completed **Challenges** and find out more about **NFPs**. All of this takes place within the social networking site Facebook, and uses that site's authentication system to log **Players** in to the **Game**.

In this first iteration of the **Game**, the integration into Facebook and overall game-related concepts were explored by creating a series of PHP programs which did not, at this stage, include the use of JavaScript and Ajax described in **Chapter 3**. This iterative first approach meant that concepts and code for the system could be explored whilst at the same time continuing the learning and research associated with learning JavaScript (which, although covered briefly in the teaching of this MSc, required further learning on my part to create the desired functionality).

Throughout the development of this section, code from the Facebook Developers site was used to integrate the externally-hosted game into the Facebook as an app. Any code adapted from that online resource is clearly marked as such in both the code and this report.

### 4.2.3.1 Key Functionality

As described above, the key tasks a **Player** carries out in the **Game** are:

- View their *AmazingStoke* **Map**
- Buy a **Building**
- Take on new **Challenges**
- Authenticate **Challenges**
- Find out more about **NFPs** using the *AmazingStoke* system

### 4.2.3.2 Basic Facebook API

Having decided to host the **Game** module of the *AmazingStoke* system within the social networking site, it was necessary to explore how an app is loaded into Facebook, and how Facebook could be used to authenticate user information.

To create an app in Facebook, a Facebook user account must be set up for the developer, and the user must join the Facebook Developers Group via the site. From there, a link is provided on the site to "Create New App" (see **Figure 4.3**).

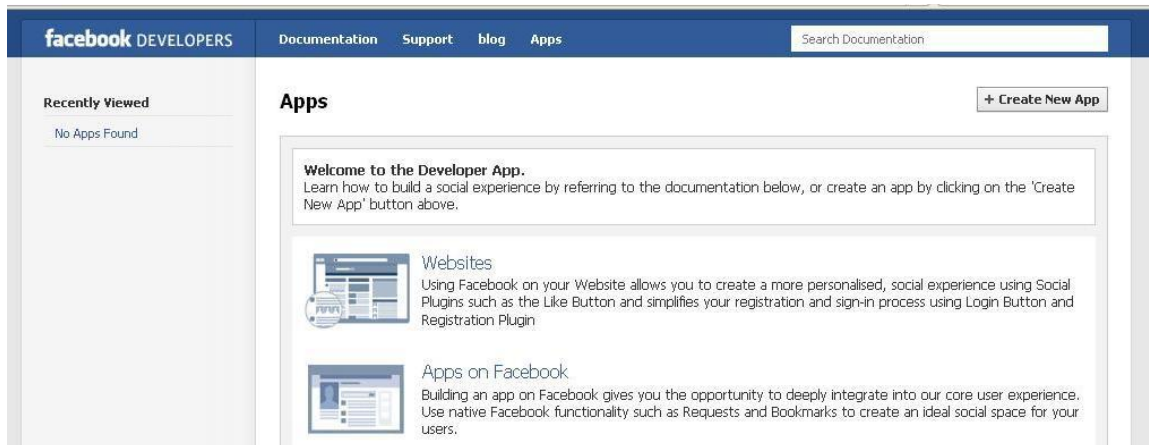


Figure 4.3: Facebook Developers site, with link to Create New App in top right-hand corner

After choosing a name, agreeing to Terms & Conditions and entering authentication details, the new App is created within the Facebook site, as shown in **Figure 4.4**.

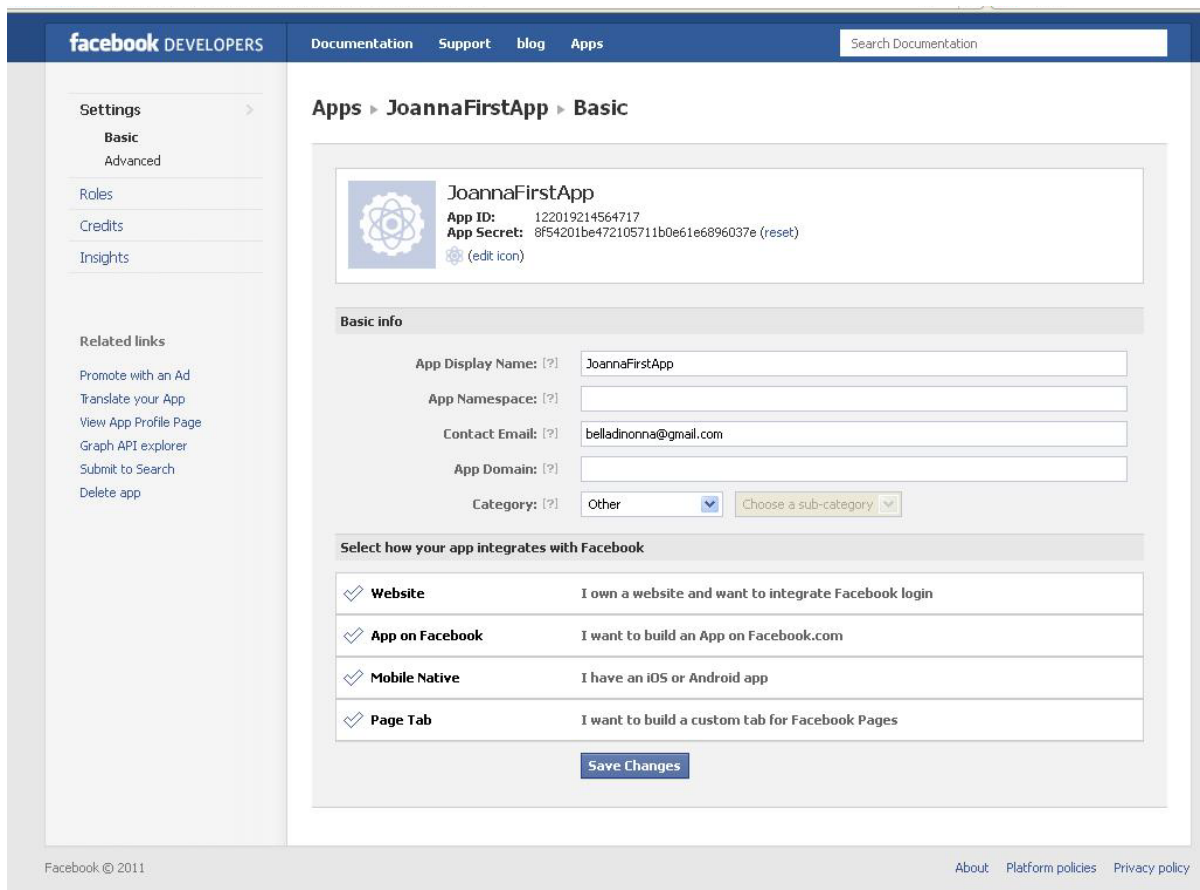


Figure 4.4: App overview within Facebook Developers

By selecting the option “App on Facebook”, a form is revealed to enter the URL of the Canvas page (that is, the URL of the external site which will be viewed via an `iframe` within Facebook). Once this has been done, the app is now loaded into Facebook, and can be accessed via the Canvas Page URL listed on the app information (as shown in **Figure 4.5**).

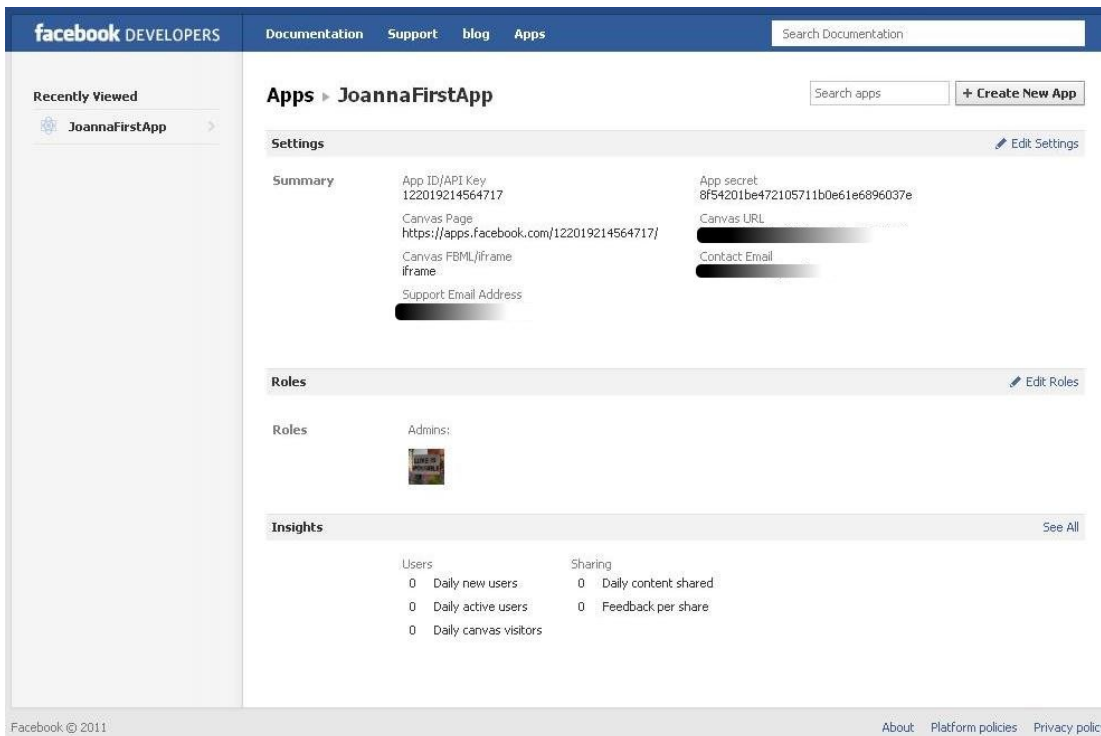


Figure 4.5: App information within Facebook Developers

Upon loading the Canvas Page URL, the site hosted at the Canvas URL is visible as an app within Facebook, as shown in **Figure 4.6**.



Figure 4.6: An example app loaded into Facebook and accessed via the site

To allow **Players** to use their Facebook login information to access an app, the Facebook Developers site gave a simple code snippet in PHP, which is reproduced below. The values of

`$app_id` and `$canvas_page` were taken from the app information page illustrated by **Figure 4.5**

```
1.  $app_id = "122019214564717";
2.
3.  $canvas_page =
4.  "http://www.dcs.bbk.ac.uk/~jpinto02/";
5.
6.  $auth_url =
7.  "https://www.facebook.com/dialog/oauth?client_id=" .
8.  $app_id . "&redirect_uri="
9.  . urlencode($canvas_page);
10.
11. $signed_request = $_REQUEST["signed_request"];
12.
13. list($encoded_sig, $payload) = explode('.',
14.     $signed_request, 2);
15.
16. $data = json_decode(base64_decode(strtr($payload, '-
17.     _', '+/')), true);
18.
19. if (empty($data["user_id"]))
20. { echo("<script> top.location.href='" . $auth_url .
21.   "'</script>");
22. }
23.
24. // App code goes in else clause
25. else
26. { ...
27. }
```

This simple code section manages the user login to the app, and creates the `$data` array holding information relating to the user's Facebook account. The PHP `$_REQUEST` superglobal array called in this code section contains by default the contents of arrays `$_GET`, `$_POST` and `$_COOKIE`.

In this example, the information relating to a Facebook user account is encoded in a JSON (JavaScript Object Notation) object. JSON is a text-based standard for language-independent transfer of information in a human-readable format, based on JavaScript. As with XML, JSON is a standard for containing information. The JSON object encoded in the `signed_request` contains the information listed in Table 4.1 (taken from Facebook Developers, 2011).

Name	Description
<code>user</code>	A JSON object containing the <code>locale</code> string, <code>country</code> string and the <code>age</code> object (containing the <code>min</code> and <code>max</code> number range of the age) for the current user.
<code>algorithm</code>	A JSON string containing the mechanism used to sign the request.
<code>issued_at</code>	A JSON number containing the Unix timestamp when the request was signed.
<code>user_id</code>	A JSON string containing the Facebook user identifier (UID) of the current user.
<code>oauth_token</code>	A JSON string that you can pass to the Graph API or the Legacy REST API.
<code>expires</code>	A JSON number containing the Unix timestamp when the <code>oauth_token</code> expires.
<code>app_data</code>	A JSON string containing the content of a query string parameter also called <code>app_data</code> . Usually specified when the application built the link to pass some data to itself. Only available if your app is an iframe loaded in a Page tab.
<code>page</code>	A JSON object containing the <code>page_id</code> string, the <code>liked</code> boolean if the user has liked the page, the <code>admin</code> boolean if the user is an admin. Only available if your app is an iframe loaded in a Page tab.
<code>profile_id</code>	A JSON number containing the Page ID if your app is loaded within. Only available if your app is written in FBML and loaded in a Page tab.

*Table 4.1: Contents of the Facebook signed\_request (Facebook Developers, 2011)*

Alongside the initial code fragment to log the user in to the app using the Facebook authentication procedure, it was necessary to include other code at the beginning of the main program of the app related to connection to the Database and using the PHP `$_SESSION` superglobal array.

```
1. header('P3P: CP="CAO PSA OUR"');
2. include 'db_login.php';
3. session_start();
```

The second and third lines of code above have been explained previously, relating to logging in to the Database on the server computer and accessing the `$_SESSION` superglobal array.

The first line related to an issue that arises when PHP sessions are used in conjunction with a site accessed via an `iframe` in Internet Explorer. The **Game** within Facebook runs from two server machines, as illustrated in by **Figure 2.1**. Standard security settings in Internet Explorer will block the setting of cookies by the **Game** because, as it is running on a separate server from the Facebook site, cookies from there are considered as 3rd party cookies. The setting of 3rd party cookies is prevented when Internet Explorer privacy settings are set at Medium or above (Whitt, 2005).

The following line of PHP code resolves this (Whitt, 2005):

```
1. header('P3P: CP="CAO PSA OUR"');
```

This manually sets the browser privacy policy to allow the cookie required for PHP session usage within Facebook. This is a general issue related to the use of `iframes` (which were noted to be problematic in the original project proposal) and is not exclusive to Facebook.

### 4.2.3.3 Loading of the Game

Upon loading the **Game** into Facebook using the Facebook login information, the system first needs to check whether this is a new **Player**. If so, the system should create a new record of the **Player**'s details to the **Users1** relation in the **Database**, and create their **PlayerChallenges** relation.

As the main starting page for the *AmazingStoke Game*, this PHP program was named `index.php`.

After storing the Facebook ID within the `$_SESSION` superglobal array, and using the PHP `getdate` function to ascertain the current date before converting it to an 8-digit integer (of the format `YYYYMMDD`), a query of the Database returns the Player's record. If the number of rows returned is 0, then the Player has not been found in the Database, and an `INSERT SQL` statement should be performed. The code below demonstrates this (with `$currentDate` set as described above):

```
1.  $map_result = queryMysql($map_query);
2.  $rows_returned = mysql_num_rows($map_result);
3.
4.  if ($rows_returned == 0)
5.  { $newPlayer = "INSERT INTO users1
6.    VALUES('$userIDnumber', 'green', 'green', 'green',
7.    'green', 'green', 'green', 'green', 'green',
8.    'green', 'green', 'green', 'green', 'green',
9.    'green', 'green', 'green', 'green', 'green',
10.   'green', 'green', 'green', 'green', 'green',
11.   'green', 'green', 'green', 'green', 'green',
12.   'green', 'green', 'green', 'green', 'green',
13.   'green', 'green', 'green', 'green', 'green',
14.   'green', 'green', 'green', 'green', 'green',
15.   'green', 'green', 'green', 'green', 'green',
16.   'green', 'green', 'green', 'green', 'green',
17.   'green', 'green', 'green', 'green', 'green',
18.   'green', 'green', 'green', 'green', 'green',
19.   'green', '500', '$currentDate')";
20.   queryMysql($newPlayer);
21.
22.   $tasksTable = "PlayerTasks" . $userIDnumber;
23.   $newTasks = "CREATE TABLE $tasksTable(TaskID
24.   VARCHAR(255) NOTNULL, Completed TINYINT(1)NOTNULL
25.   DEFAULT '0')";
26.   queryMysql($newTasks);
27. }
```



As can be seen above, the program creates a new **Player** with each square of their **Map** set to “Green” and a starting **MarvelMoney** balance of 500.

Once the system has checked whether the **Player** is already registered in the **Database**, and created a new record for them if they are logging in for the first time, the **Player**’s record from **User1** can be used to create a graphic representation of the **Player**’s *AmazingStoke Map* and other information. The **Player**’s record from the **Database** is set as a PHP array `$getMapRow`.

In this first iteration of the system, the building names from the `$getMapRow` PHP array are simply used to dynamically generate a grid of images of eight square by eight squares. These images are saved in the `../drawing` directory under the same directory as `index.php` file, and named so that a simple string concatenation uses the Name of a building as a reference to the image representing it. The code for this is shown below.

```
1.  for ( $i=1; $i < 65; ++$i)
2.  { $square = "sq" . $i;
3.    $building = $getMapRow[$i];
4.    $img_string = "../drawing/" . $building . ".jpg";
5.
6.    if ($i % 8 == 0)
7.      echo "<img src=$img_string /><br />";
8.
9.    else echo "<img src=$img_string />";
10.   }
```

The HTML output by the PHP program also includes:

- The **Player**’s **MarvelMoney** balance
- Hyperlinks to the PHP programs to “Buy a Building”, “Take on new Challenges”, “Authenticate Challenges”, and “Find out more about NFP using the *AmazingStoke* system”.

#### 4.2.3.4 Buying A Building

The first iteration of the program to “Buy A Building” is a simple PHP program utilising **Player** input via an HTML form and an HTTP POST. The **Player**’s Facebook ID is accessed via the `$_SESSION` superglobal array. An SQL query is sent to the **Database** to retrieve the **Player**’s **MarvelMoney** balance which is stored as the PHP variable `$marvelmoney_balance`. The **Buildings** are downloaded from the **Database** and details loaded into the HTML form. The program to create this page is called `buildings.php`.

If a **Player** attempts to purchase a **Building** for which they do not have enough **MarvelMoney**, a message to this effect is output to the browser with a link for the **Player**

to return to their *AmazingStoke* map. Otherwise, an UPDATE SQL query is sent to the **Database**, and the **Player** is redirected to `index.php`.

In this first iteration of the **Game**, the information relating to **Buildings** is initially hard-coded into the PHP. It is intended that subsequent iterations of the system will access the **Buildings** information from the **Database** when the page is loaded, using an array of JavaScript objects

I had originally attempted to create a grid of images in `index.php` as an HTML form which would pass the image location as a hidden value to the `buildings.php` program via an HTTP POST request. However, this proved to be impossible with Internet Explorer, as it was not possible to pass the reference to the square as a hidden field when using an image within a form using the POST method (due to the nature of the POST array passed within Internet Explorer).

#### 4.2.3.5 Viewing And Accepting New Challenges

The first iteration of the program to View And Accept New Challenges is a simple PHP program entitled `alltasks.php`.

The **Player**'s Facebook ID is accessed via the `$_SESSION` superglobal array. All **Challenges** which have been approved (Live == '1') are retrieved from the Database, and output to an HTML form. **Players** select a **Challenge** by the use of radio buttons, which allow only one value to be select from the form. The **Player** input is passed via an HTTP POST request to the program `taskadded.php`.

`taskadded.php` is a short PHP program which simply accesses the **Player**'s Facebook ID via the `$_SESSION` superglobal array and the ChallengeID via the `$_POST` array, and adds the information to the relevant **PlayerTasks** relation.

```
1. $userIDnumber = $_SESSION['userIDnumber'];
2. $taskSelected = $_POST['taskSelected'];
3. $tasksTable = "PlayerTasks" . $userIDnumber;
4. $newTaskQuery = "INSERT INTO $tasksTable VALUES
5.     ('$taskSelected', '0')";
6. queryMysql($newTaskQuery);
```

It then writes confirmation to the browser, with a link for a **Player** to return to `index.php`.

Whilst other forms have returned the data input by the **Player** to the same program, this particular functionality is handled by a separate program. This is because the `taskadded.php` provides a functionality required by several different displays of the underlying **Challenges** information stored in the **Database**. A single PHP program which can be reused is more efficient both in terms of coding time and usage of resources. An example of this reuse will be shown in **Section 4.2.3.7**.

### 4.2.3.6 Authenticating Challenges

To authenticate currently accepted **Challenges**, a logged-in **Player** clicks on a hyperlink to launch the program `yourtasks.php`.

This retrieves from the **Database** all the **Player**'s information from **Users1** and **Challenges** in a **Player's Challenges** relation where `Completed == '0'`.

```
1.  "SELECT * FROM $tasksTable, ascharitytasks,
2.  ascharities
3.  WHERE $tasksTable.TaskID=ascharitytasks.TaskID
4.  AND ascharities.CharityNumber =
5.  ascharitytasks.CharityNumber AND Completed='0'";
```

This information is then output to the user as an HTML form, with text boxes for the entry of **Voucher Codes** and which submits by means of an HTTP POST request to the PHP program `verify.php`.

The program `verify.php` executes a call to the **Database** to retrieve a single row containing the **Challenge** information from the **AsCharityTasks** relation.

The program must first ascertain whether the **Voucher Code** entered relates to a **Challenge** with a single **Voucher Code** that can be used multiple times, or to a **Challenge** with multiple single-user **Voucher Codes** (which should be deleted from the relation after one use).

If `VoucherCodes == '1'` (i.e. there exists a separate relation of **Voucher Codes**), a second call to the **Database** retrieves the set of **Voucher Codes**, and checks whether the value entered by the **Player** in `yourtasks.php` matches a value in the relation. If so, that row is deleted from the **Voucher Codes** relation. If `VoucherCodes == '0'`, then the program checks whether the value entered by the **Player** in `yourtasks.php` matches a **ChallengeID**.

In either case, if `verify.php` matches a **Voucher Code**, a call to the **Database** marks the **Challenge** as **Completed** in the **Player's PlayerChallenges** relation and updates their **MarvelMoney** balance in **Users1**. The **Game** then outputs confirmation to the browser. Else, an error message is written to the browser. In either case, the **Player** is presented with a hyperlink to return to `index.php` once `verify.php` has completed.

### 4.2.3.7 NFP Information

To display information regarding the **NFPs** using *AmazingStoke*, two PHP programs were written.

The first, `orglist.php`, retrieves all the **CharityName** information from the **AsCharities** relation, sorted alphabetically. It then writes this information to the browser,

with each NFP assigned a simple HTML form that will submit the CharityName to orginfo.php via an HTTP POST request when the **Player** presses submit.

The orginfo.php program then issues a call to the **Database** to retrieve the information on the **NFP** from **AsCharities**. A second call is then issued to the **Database** to retrieve that **NFP's Challenges** from **AsCharityTasks**.

```
1. $organisation_query = "SELECT CharityNumber,
2. CharityName, Address1, Address2, Address3, TownCity,
3. County, PostCode, Blurb, Website FROM ascharities
4. WHERE CharityName='$charity_name'";
5.
6. $tasks_query = "SELECT * FROM ascharitytasks WHERE
7. CharityNumber='$charitynumber'";
```

The information retrieved from **AsCharities** is written as formatted text to the browser.

The information retrieved from **AsCharityTasks** is written to the browser as an HTML form incorporating radio buttons, allowing a **Player** to select **Challenges** as described in **Section 4.2.3.5**.

Upon selecting tasks from the form produced by orginfo.php, the **Game** calls taskadded.php as also described in **Section 4.2.3.5**.

## 4.3 Second Iteration Of The System

This section of the report concerns the development of the second iteration of the *AmazingStoke* proposed system.

Having developed an initial “proof-of-concept” system, the intention of this iteration was to improve the user experience by more effective use of client-server workload division discussed in **Chapter 3**. Client-side scripting technology begins to be significantly implemented by use of JavaScript in the browser with Ajax technologies used to make necessary changes server-side.

At this stage, additional non-essential functionality begins to be introduced to the system, as well as greater instructions for users.

### 4.3.1 Second Iteration Of NFP Site

This section describes the addition of increased functionality to the *AmazingStoke* **NFP Site**.

#### 4.3.1.2 Add Challenge Form

To reduce workload of checking potentially incomplete forms via a call to a PHP program on the server, the second iteration of the Add A Challenge form (*asaddtask.php*) incorporated the following JavaScript functionality:

A JavaScript function called `validateForm` created to check that all required fields in the form are been completed, and that the expiry date listed is at least one day after the current date, before the information is passed to the server. If all sections of the form are not been complete, error messages are displayed next to the incomplete or incorrect sections through use of the `<span>` HTML elements and using the JavaScript `getElementById('...').innerHTML` to rewrite the document in the browser.

JavaScript is then used in combination with Cascading Style Sheets (CSS) technology to show or hide certain areas of the form, depending on **NFP** choices relating to Category and the use of multiple **Voucher Codes**. CSS was defined by the World Wide Web Consortium (W3C) to apply formatting information to HTML (W3C, 2011). CSS formatting information can be saved as separate `.css` file, or can be used embedded within the HTML. To show and hide information in an HTML document, the following CSS rule can be used:

```
1. <span id='myID' style='display:none'>
```

The `id` value can then be used in conjunction with a JavaScript function to rewrite the value of `display`, as follows:

```
1. document.getElementById('myID').style.display = ''
```

Functions using JavaScript and CSS were written to show or hide the input area for the amount of real-world money if Fundraising were selected as Category, and to show or hide the input area for the number of **Voucher Codes** required when listing a **Challenge** that use multiple codes.

#### 4.3.1.3 View Voucher Codes

Functionality was added in to this iteration of the system to allow an **NFP** to download **Voucher Codes** as a Comma Separated Values (CSV) file. This functionality was added in to the PHP program `viewvouchers.php`.

A program to create a CSV file using PHP from a MySQL database was found online on the website WebCheatSheet.com (WebCheatSheet.com, 2009). It is included as the PHP program `downloadvouchers.php`. A link was included in `viewvouchers.php` to run `downloadvouchers.php`, passing in the name of the relation using an HTTP GET request. It should be noted that this function has been used in its entirety from WebCheatSheet.com, 2009, and is not the work of myself.

Additionally, functionality was added in to `viewvouchers.php` to allow **NFPs** to show or hide the Voucher Codes onscreen, using the same JavaScript and CSS rewriting as described in **Section 4.3.1.2**.

#### 4.3.1.4 Generate More Voucher Codes

The next addition in functionality to the second iteration of the **NFP Site** was the functionality to allow an **NFP** to generate more **Voucher Codes** for an existing **Challenge**. This functionality was added in to the PHP program `viewvouchers.php`.

A simple HTML form within the page allows an **NFP** to specify a number of additional **Voucher Codes** to generate. On submission of the form, a JavaScript function checks that the **NFP** has entered a valid integer value of 1000 or less. If not, an error message is written to the browser document using client-side scripting, else the form passes the necessary information back to `viewvouchers.php` using an HTTP POST request. Additionally, if the creation of the **Voucher Codes** would take the last number in the sequence to more than 999,999, an error message is written to the document using JavaScript.

Otherwise, the requested number of **Voucher Codes** are then generated at the end of the existing sequence stored in the **Database**.

The above functionality required the final number in the existing Voucher Code sequence. Code is added in to `viewvouchers.php` to set this as a PHP variable `$maximumCode` when the page is initially generated.

```
1.     $maximumCode = 0;
2.     for ($i = 0; $i < $returnVouchersRows; ++$i)
3.     { $currentRow =
4.         mysql_fetch_row($returnVouchersResult);
```

```

5.     $currentCode = $currentRow[0];
6.     $voucherArray[] = $currentCode;
7.     $maximumCode = $currentCode;
8.     }

```

This value is then used as a starting point in a for loop to insert more **Voucher Codes** to the relevant relation.

Once the insertion has taken place on the Database, a confirmation message is written to the reloaded `viewvouchers.php` document in the browser.

### 4.3.1.5 Cascading Style Sheets

As described in **Section 4.3.1**, Cascading Style Sheets (CSS) allow presentation information regarding the HTML to be stored as a separate `.css` file known as an external style sheet. Every page within the site that applies the formatting information must include within the head section the following code:

```

1. <link rel='stylesheet' type='text/css' href='...css' />

```

For example, for the **NFP Site**, a simple style sheet to modify the font used throughout the `<body>` sections of the HTML was created as follows as saved as `amazingstokestyle.css`.

```

1. body
2. { font-family:Arial, Helvetica, sans-serif;
3. }

```

With the application of this stylesheet, the appearance of the page is changed from **Figure 4.7** to **Figure 4.8**.



Figure 4.7: NFP Site Home page before use of CSS

**AmazingStoke: Gaming for Good**

Logged in as: johnsmith@mynewcharity.org.uk  
 Charity Numer: 123456  
[Home](#) [Add A Challenge](#) [Edit Your Description](#) [Log out](#)

Welcome to your administrator page for **AmazingStoke** - the Facebook game that allows you to reward supporters for real-world actions.

Players log in to the game via Facebook, then can choose to undertake any challenges that you set them via this website. Generate voucher codes to give to your supporters once they complete their tasks.

You can generate voucher codes to be used multiple times (useful for if only a few people are completing a challenge for you - say, running a marathon or being a media volunteer.

Or, you can generate lists of one-use-only voucher codes (useful for giving out to donors, online supporters, and so on) and download these as CSV (Excel) files.

**Your current Challenges:**

Title	Description	MarvelMoney	Multiple Vouchers?	Voucher Codes
Donate £5 to My New Charity	Please donate £5 to My New Charity. On receipt of the money, we will send you a thank-you email containing a Voucher Code to redeem your MarvelMoney.	20	Yes	<a href="#">View voucher codes</a>

*Figure 4.8: NFP Site Home page after use of CSS*

It was also the intention to make greater use CSS technology to improve the overall appearance of both the **NFP Site** and the **Game**. However, unfortunately this was something that there was not sufficient time to undertake on this project, and so only these minor changes were made. Design issues will be explored in greater depth in **Chapter 6**.



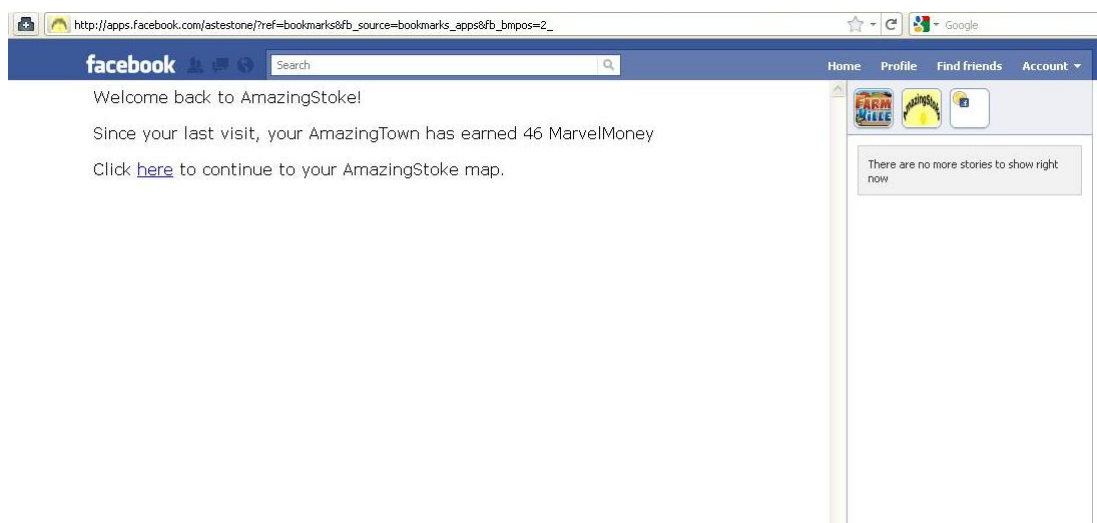
## 4.3.2 Second Iteration Of Game

This section describes the addition of increased functionality to the *AmazingStoke Game* via Facebook, and of the introduction of Ajax technology into the system.

### 4.3.2.1 Updating Player's MarvelMoney On Login

One functionality noted in both the original project proposal and in **Section 3: System Analysis and Design** is that the **Game** should reward **Players** with **MarvelMoney** without the need to take on **Challenges** for **NFPs**.

One way to do this is to reward the **Player** each time they login based on the contents of their **Map**, using the Earnings value of each **Building**. Functionality was added to the `index.php` program to check the date of the **Player**'s last login against the current date. If the **Player** had not logged in that day, they were redirected to the program `updater.php`. This program updates the **MarvelMoney** and `DateLastPlayed` of **Users1**. A message is then written out to the browser, informing the **Player** with the increase in the **MarvelMoney**, and with a link to reload the `index.php` program (which will this time not trigger the `updater.php` program).



*Figure 4.9: Message informing the Player that their AmazingStoke Town has earned MarvelMoney when logging in*

### 4.3.2.2 JavaScript And CSS To Display Map / Buy Buildings Form Within The Browser

As discussed previously, one feature of a web applications offering good user experience is the appropriate division of client-side and server-side programming.

The first iteration of the **Game** used separate PHP programs to display the **Map** and Buy Buildings form (`index.php` and `buildings.php` respectively). In the second iteration of the **Game**, these two functions are combined as originally envisioned into one program,

which uses JavaScript to makes changes to the in-browser document and uses Ajax calls to update the **Database**.

**Section 4.3.1.2** already described how JavaScript and CSS can be used to display or hide certain area of a browser document. It was therefore relatively straightforward to write a JavaScript function to display the Buy A Building Form and hide the **Map**, and vice versa, when the **Player** clicks on their **Map** or clicks the “Click Here To Return To Your Map” button respectively.

To make changes within the browser to the **Player**’s **MarvelMoney** balance and to the **Buildings** on their **Map** without a call to the server, it was necessary for this data to be accessible as variables that could be modified within the browser. Therefore, it was necessary to convert the PHP `$MapArray` to a JSON string (as described in **Section 4.2.3.2**) using the PHP function `json_encode`. If the **Player** does not have enough **MarvelMoney** to purchase the building, an error message is written to the browser using `<span>` elements and JavaScript `innerHTML`. Otherwise, the JSON `MapArray` and the **MarvelMoney** amount are updated using JavaScript.

```
1. $playerArray = $getMapRow;
2. $playerMapAsArray = json_encode($playerArray);
```

Now that the buying of a **Building** is being handled within the browser on the client-side, it is necessary to write a program to carry out the behind-the-scenes call to update the **Database**.

### 4.3.2.3 Rewriting Map Using JavaScript And Ajax

As described in the original project proposal, Ajax (Asynchronous JavaScript And XML) is a collection of technologies to run behind-the-scenes server scripts. In this case, Ajax is used to update the **Database** when a **Player** buys a new **Building** as described below.

The key JavaScript functionality used in Ajax to run server-side scripts without reloading the page is the `XMLHttpRequest` object. As Flanagan explains in his book *JavaScript: The Definitive Guide* (Flanagan, 2011), an HTTP request consists of four parts:

- the HTTP request method (such as GET or POST)
- the URL being requested on the server-side
- an (optional) body of information in the form of request headers
- an (optional) request body

(Flanagan, 2011). This information must be included within the attributes of the JavaScript `XMLHttpRequest` object.

Once the `XMLHttpRequest` object has been created within the client-side script, it must be opened specifying the request method, the subject of the request, and whether the request

should be handled asynchronously. Other optional information which can be included in the open method includes a username and password. Finally, once the XMLHttpRequest object has been created and opened, it can be sent as request to the server to run the specified program.

The section of code below shows the creation of the XMLHttpRequest object entitled dataSendObject. This creation must be handled differently for versions of Internet Explorer before version 7. A function is written to ensure confirmation is written to the browser when the status of the dataSendObject changes, for example when the open function is called upon it. This is followed by the rewriting of the document in the browser, followed opening of the dataSendObject, followed the sending of this XMLHttpRequest object to the server-side program openAjax.php with the variables to be used in the **Database** update included in the URL.

The following function is based on an example from W3Schools (W3C, 2011).

```
1. function sendData()  
2. {  
3. // Creates object in IE7+, Firefox, Chrome, Opera,  
4. // Safari  
5.  
6.   if (window.XMLHttpRequest)  
7. { dataSendObject = new XMLHttpRequest();  
8. }  
9.  
10. // Creates object in IE6, IE5  
11. else  
12. dataSendObject = new  
13. ActiveXObject('Microsoft.XMLHTTP');  
  
14. dataSendObject.onreadystatechange=function()  
15. { if (dataSendObject.readyState==4 &&  
16.     dataSendObject.status==200)  
17.   {document.getElementById('sentToServer').innerHTML  
18.     = 'Congratulations on your new AmazingStoke  
19.     purchase!<br />';  
20.   }  
21. }  
22.  
23. buildingBought = '';  
24. for ( i = 0 ; i <  
25. document.getElementById('buyingForm').building.  
26. length; i++)  
27. { if  
28. (document.getElementById('buyingForm').building[i].
```

```

29.  checked)
30.  buildingBought = document.getElementById
31.  ('buyingForm').building[i].value;
32.  }
33.
34.  squarePlaced = document.getElementById('buyingForm')
35.  .squareValue.value;
36.  mapArrayIndex = squarePlaced.substring(2);
37.  sendAjaxString = 'openAjax.php?square=' +
38.  mapArrayIndex + '&building=' + buildingBought +
39.  '&balance=' + currentBalance + '&player=' +
40.  playerID;

41.  dataSendObject.open('GET', sendAjaxString, true);
42.  dataSendObject.send();
43.  }

```

The program `openAjax.php` is a PHP program which extracts information regarding changes to the **MarvelMoney** amount and placement of **Buildings** on the **Map** using the `$_GET` superglobal array, and makes the necessary changes to **Database**.

```

1.  $sentSquare = $_GET['square'];
2.  $sentBuilding = $_GET['building'];
3.  $sentBalance = $_GET['balance'];
4.  $sentPlayer = $_GET['player'];
5.  $squareString = 'sq' . $sentSquare;
6.
7.  $updateMapQuery = "UPDATE users1 SET
8.  $squareString='$sentBuilding', money='$sentBalance'
9.  WHERE playerID='$sentPlayer'";
10. queryMysql($updateMapQuery);

```

#### 4.3.2.4 Cascading Style Sheets

As described in **Section 4.3.1.5**, the external style sheet `amazingstokestyle.css` was created and applied to the **NFP Site**. In the second iteration of the **Game**, this style sheet was also applied to the **Game**.

## 4.4 Third Iteration Of The System

It was the original intention of this project to implement a third iteration of the *AmazingStoke* system, to include further refinements and additional functionality.

Unfortunately, I had underestimated the amount of time that would be needed to learn PHP and the additional JavaScript needed for this system, and the iterative approach described meant that more complex functionality was left until later in the development of the system.

Therefore, although I had intended to implement more of the *AmazingStoke* system, all that I was able to complete by the project deadline was the two iterations described in **Section 4.2**.

The key functionality that I had hoped to implement but was unable due to time was:

- Storing all **NFP** information in the **Game** client-side in the browser, to speed up **Player** experience once the initial loading has completed
- Storing all **Challenge** information in the **Game** client-side in the browser, to speed up **Player** experience once the initial loading has completed

It was an important learning point that whilst the basics of new languages (such as PHP and JavaScript) may be learnt quickly, it takes time and practice to be able to develop more complex system using these tools. In hindsight, it might have been more efficient to use a server-side programming framework such as Grails (SpringSource, 2009), which is built upon the Groovy language studied in-depth in the **Object-Oriented Design and Programming** module of the MSc Computer Science.

In addition, there were a number of development frameworks that could have assisted with the implementation of this project, which as a novice to these technologies I did not investigate fully. These key learning points will be addressed in **Chapter 7**.

Although lack of remaining time meant that it was not possible to complete the writing of the development of other features of the *AmazingStoke* system, the planned further functionality is described in detail in **Chapter 6**.

## **Chapter 5: Demonstration**

This chapter describes, via a series of screenshots, the key activities of the system, presenting screenshots and descriptions of the processes used by the distinct user groups (Players and NFPs) to achieve their goals within the system.

As discussed previously in this report, this system is hosted live via provider One . com.

The NFP site is hosted at:

`www.astoke.co.uk/charities`

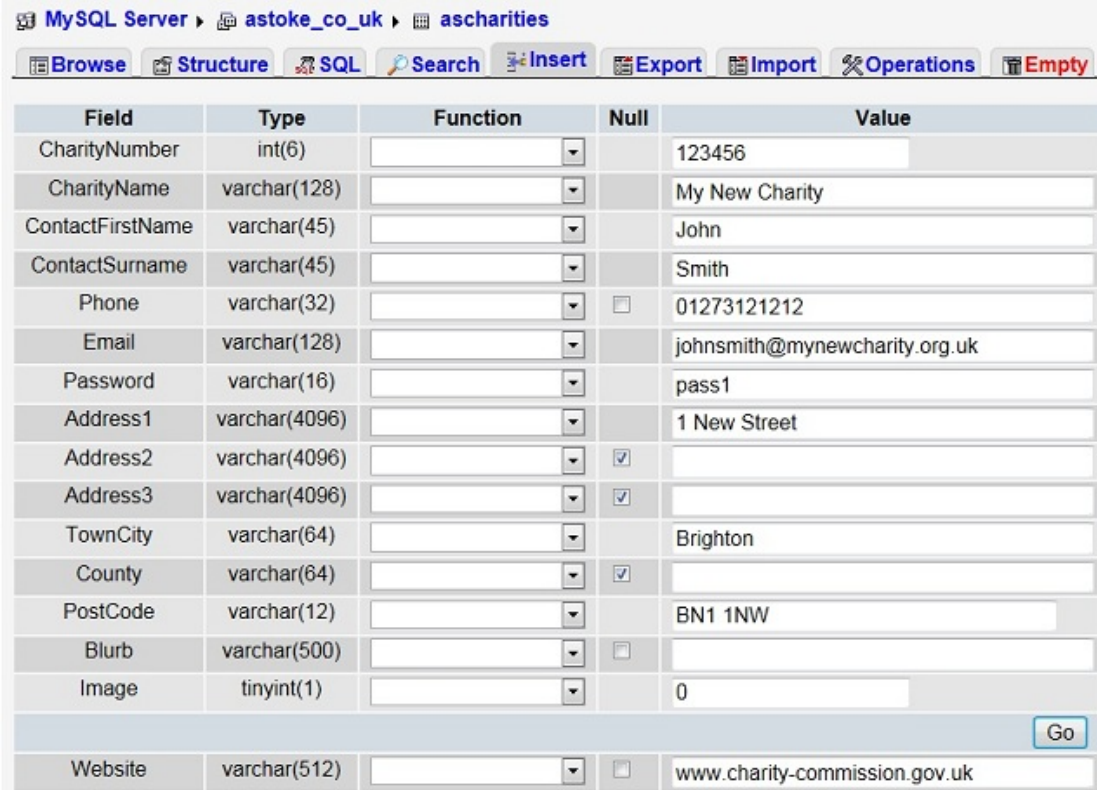
The Game is hosted at:

`www.astoke.co.uk`

All aspects of the system were tested in the two most frequently used browsers, Internet Explorer (Version 9) and Mozilla Firefox (Version 5).

## 5.1 NFP Site

To demonstrate the functionality of the NFP site, a mock NFP organisation was set up on the Database using phpMyAdmin. The organisation was given the name My New Charity, and set up as shown in **Figure 5.1**.



Field	Type	Function	Null	Value
CharityNumber	int(6)			123456
CharityName	varchar(128)			My New Charity
ContactFirstName	varchar(45)			John
ContactSurname	varchar(45)			Smith
Phone	varchar(32)		<input type="checkbox"/>	01273121212
Email	varchar(128)			johnsmith@mynewcharity.org.uk
Password	varchar(16)			pass1
Address1	varchar(4096)			1 New Street
Address2	varchar(4096)		<input checked="" type="checkbox"/>	
Address3	varchar(4096)		<input checked="" type="checkbox"/>	
TownCity	varchar(64)			Brighton
County	varchar(64)		<input checked="" type="checkbox"/>	
PostCode	varchar(12)			BN1 1NW
Blurb	varchar(500)		<input type="checkbox"/>	
Image	tinyint(1)			0
				<input type="button" value="Go"/>
Website	varchar(512)		<input type="checkbox"/>	www.charity-commission.gov.uk

*Figure 5.1: phpMyAdmin display of NFP account set up for My New Charity*

### 5.1.1 Login

As seen in Figure 5.1, the email address assigned to the NFP is **johnsmith@mynewcharity.org.uk**, and the password assigned is **pass1**. Login was attempted using these details.

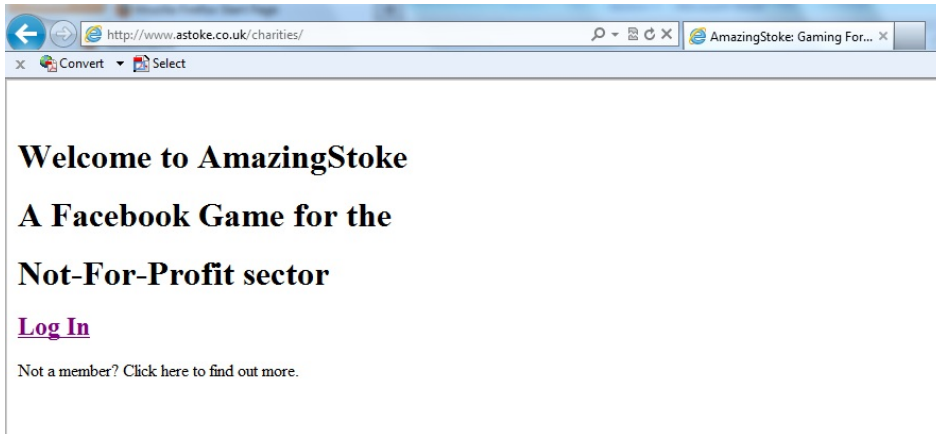


Figure 5.2: NFP Site Welcome Page

First, the error checking was tested by entering an invalid email address (see **Figure 5.3**).

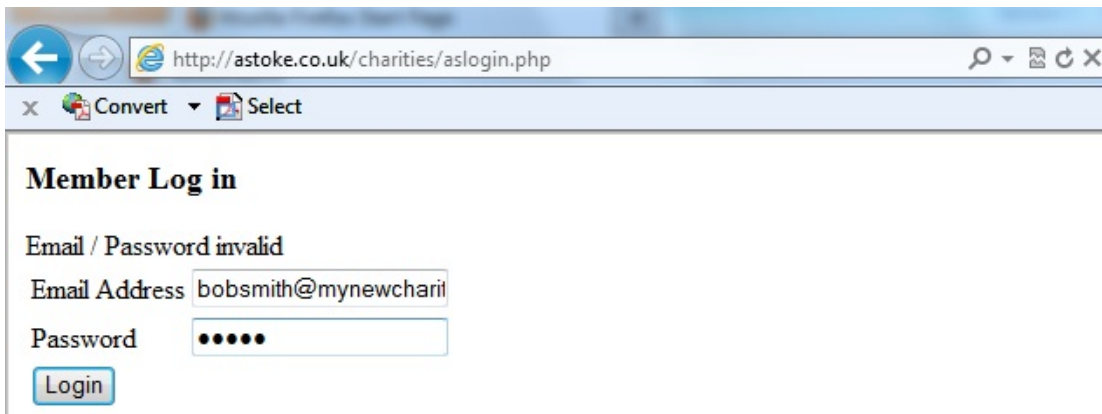


Figure 5.3: NFP Site Login page after an invalid (i.e. not in the Database) email address was submitted

Using correct login details as described above, the NFP then successfully logged in (see **Figure 5.4**). No Challenges are currently listed for the NFP, so the initial page first displays only instructions relating to the system.

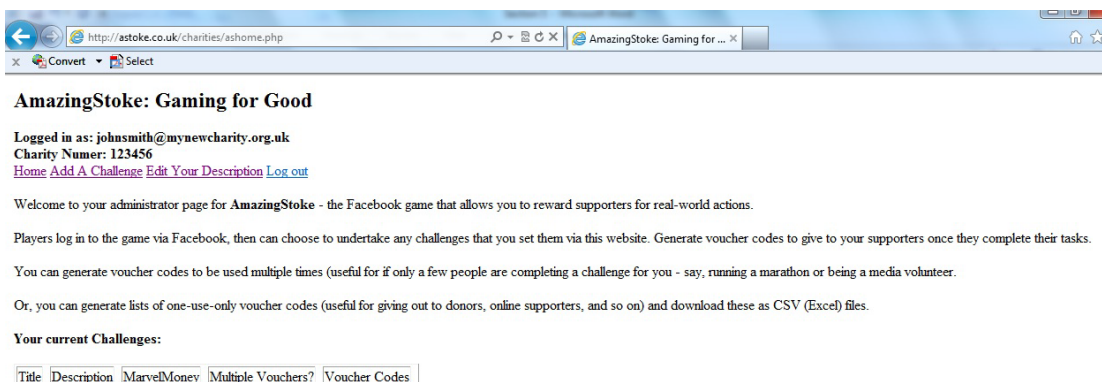


Figure 5.4: NFP Site Home page for logged in NFP My New Charity



## 5.1.2 Update Details

As seen in **Figure 5.1**, this NFP was originally created on the Database with no description of the organisation. Therefore, a description is added in using the Edit Your Description (`aseditinfo.php`) page.

Logged in as: johnsmith@mynewcharity.org.uk  
[Home](#) | [Add A Challenge](#) | [Edit Your Description](#) | [Log out](#)

### Edit Information About Your Organisation

About Your Organisation (max. 500 characters)  
 My New Charity is a new charity, set up on 12 September 2011. It exists to do charitable things in a charitable way.

Contact First Name  
 Contact Surname  
 Password

*Figure 5.5: The Edit Your Description form, ready to be submitted*

Using phpMyAdmin, the change to the Database is then confirmed. The changed information is highlighted in green in **Figure 5.6**.

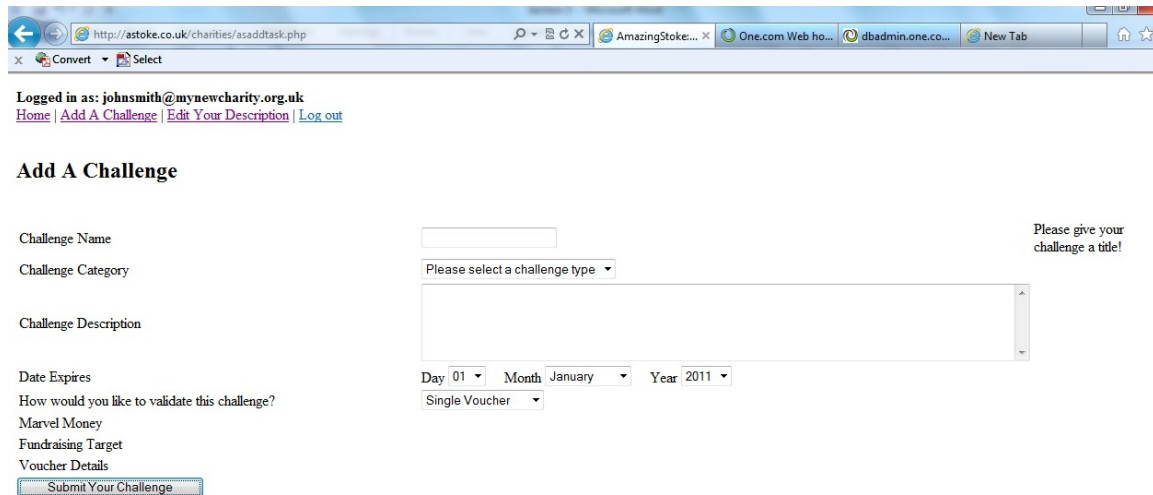
Field	Type	Function	Null	Value
CharityNumber	int(6)			123456
CharityName	varchar(128)			My New Charity
ContactFirstName	varchar(45)			John
ContactSurname	varchar(45)			Smith
Phone	varchar(32)		<input type="checkbox"/>	01273121212
Email	varchar(128)			johnsmith@mynewcharity.org.uk
Password	varchar(16)			pass1
Address1	varchar(4096)			1 New Street
Address2	varchar(4096)		<input checked="" type="checkbox"/>	
Address3	varchar(4096)		<input checked="" type="checkbox"/>	
TownCity	varchar(64)			Brighton
County	varchar(64)		<input checked="" type="checkbox"/>	
PostCode	varchar(12)			BN1 1NW
Blurb	varchar(500)		<input type="checkbox"/>	My New Charity is a new charity, set up on 12 September 2011. It exists to do charitable things in a charitable way.
Image	tinyint(1)			0
Website	varchar(512)		<input type="checkbox"/>	www.charity-commission.gov.uk

*Figure 5.6: phpMyAdmin display of NFP account for My New Charity after `aseditinfo.php`*

### 5.1.3 Add A Challenge

To test the program `asaddtask.php`, I attempted to add a simple Fundraising challenge, asking Players to make a £5 donation to My New Charity, with 500 Voucher Codes initially generated.

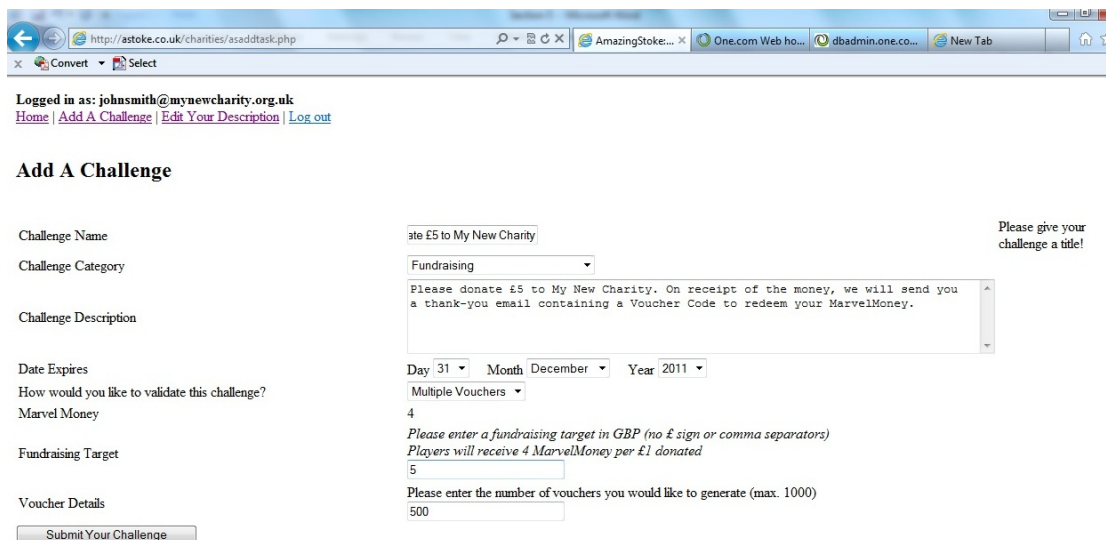
First, the error checking was tested by submitting an incomplete form (see **Figure 5.7**).



The screenshot shows a web browser window with the URL `http://astoke.co.uk/charities/asaddtask.php`. The user is logged in as `johnsmith@mynewcharity.org.uk`. The page title is "Add A Challenge". The form fields are: Challenge Name (empty), Challenge Category (Please select a challenge type), Challenge Description (empty), Date Expires (Day 01, Month January, Year 2011), How would you like to validate this challenge? (Single Voucher), Marvel Money, Fundraising Target, and Voucher Details. A "Submit Your Challenge" button is at the bottom. An error message "Please give your challenge a title!" is displayed on the right side of the form.

*Figure 5.7: NFP Site Add A Challenge page after a blank form was submitted – note error message “Please give your challenge a title!”*

Then correct details were entered onto the form, with the JavaScript / CSS displaying of certain initially “hidden” elements of the page.



The screenshot shows the same web browser window. The form fields are now filled out: Challenge Name (ate £5 to My New Charity), Challenge Category (Fundraising), Challenge Description (Please donate £5 to My New Charity. On receipt of the money, we will send you a thank-you email containing a Voucher Code to redeem your MarvelMoney.), Date Expires (Day 31, Month December, Year 2011), How would you like to validate this challenge? (Multiple Vouchers), Marvel Money (4), Fundraising Target (Please enter a fundraising target in GBP (no £ sign or comma separators) Players will receive 4 MarvelMoney per £1 donated, 5), and Voucher Details (Please enter the number of vouchers you would like to generate (max. 1000) 500). The "Submit Your Challenge" button is still present.

*Figure 5.8: NFP Site Add A Challenge page with hidden text areas “Fundraising Target” and “Voucher Details” displayed*

After submission of the completed form, returning to the homepage now shows the updated information on NFP Challenges (**Figure 5.9**).

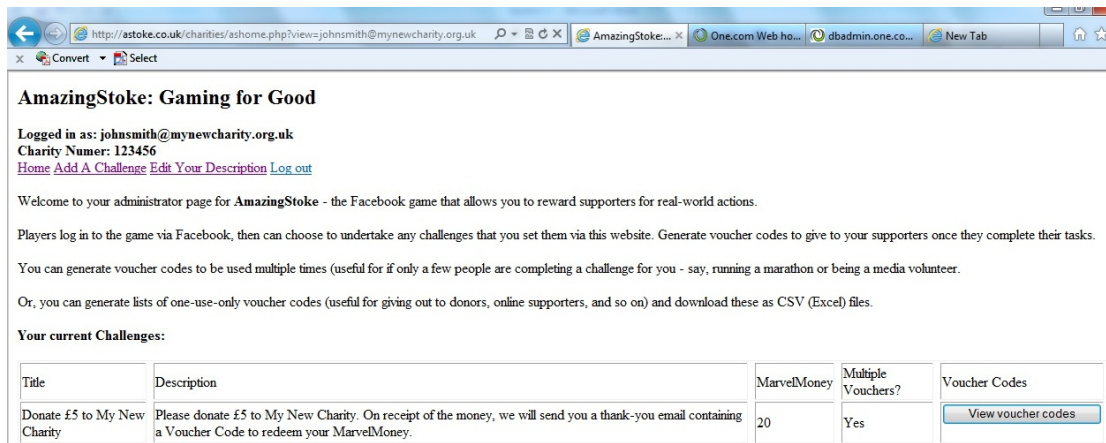


Figure 5.9: NFP Site Homepage for logged in NFP My New Charity, now showing the Challenge added in Section 5.1.3

By using phpMyAdmin to view the Database, it is also possible to see the successful creation of a relation of Voucher Codes for this Challenge, entitled **Task123499**.

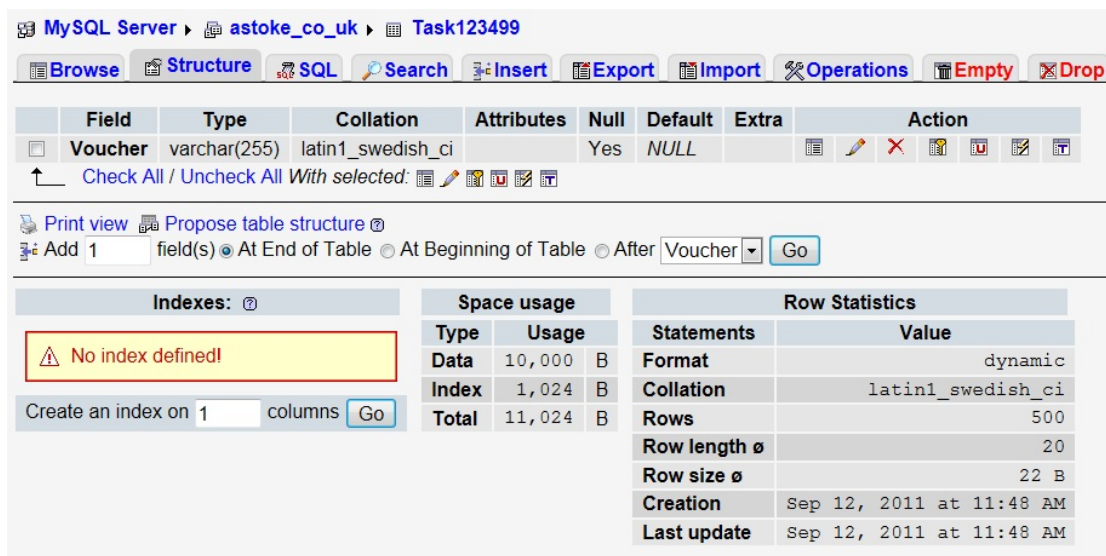


Figure 5.10: phpMyAdmin display of relation Task123499

It can be seen from **Figure 5.10** that 500 rows have also been successfully inserted into this relation containing Voucher Codes.

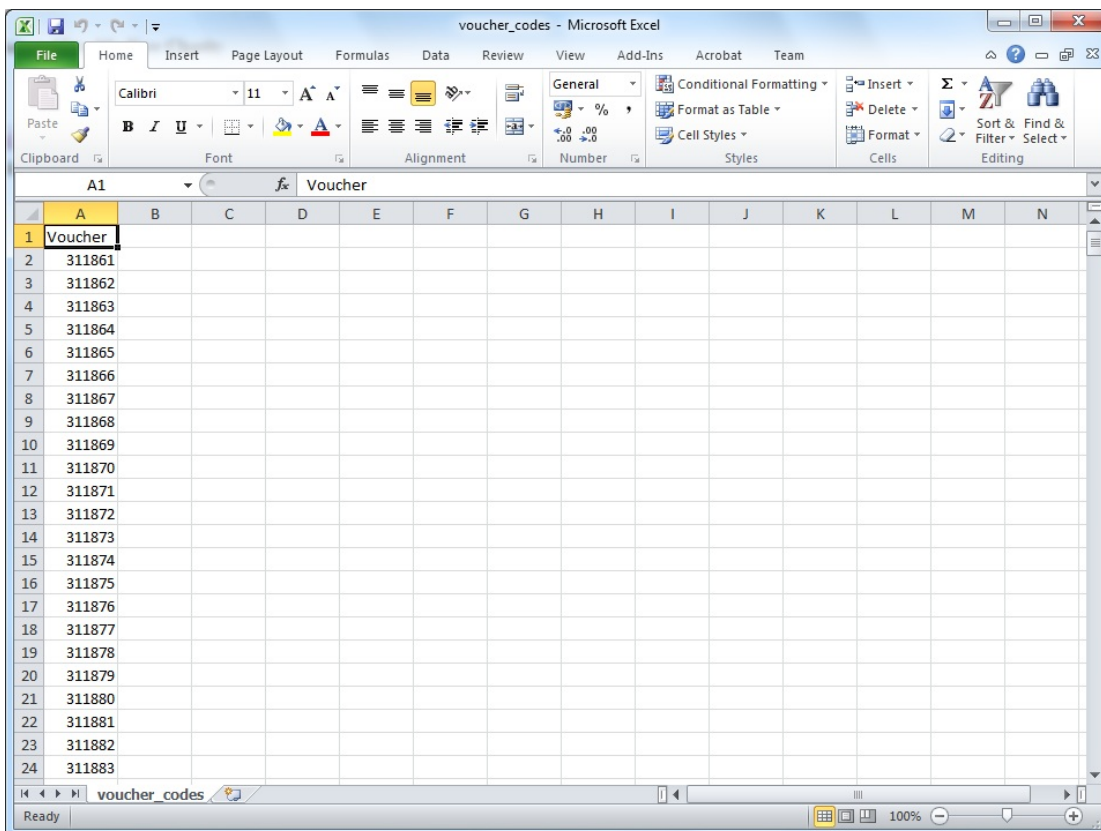
## 5.1.4 View Voucher Codes

By returning to the NFP Site homepage and click on the button “View Voucher Codes” for the Challenge added in **Section 5.1.3**, it is possible to see the link to download these Voucher Codes as a CSV file.



*Figure 5.11: NFP Site View Vouchers for logged in NFP My New Charity, now showing the Vouchers Codes for Challenge added in Section 5.1.3*

By clicking the link “Click here to download these codes as a CSV file”, a CSV file entitled voucher\_codes.csv is created and opens automatically using (in this instance) Excel.



*Figure 5.12: CSV file of Vouchers Codes for Challenge added in Section 5.1.3*



## 5.1.5 Generate More Voucher Codes

Next, the functionality to create more **Voucher Codes** was tested. First, the string “Hundred” was entered to check the error checking of the form. The result is shown in **Figure 5.13**.



The screenshot shows a web browser window with the URL <http://astoke.co.uk/charities/viewvouchers.php>. The user is logged in as [johnsmith@mynewcharity.org.uk](mailto:johnsmith@mynewcharity.org.uk). The page title is "Voucher codes for Donate £5 to My New Charity". The total number of vouchers is 500. A text input field contains "Hundred" and a "Generate More Voucher Codes" button is visible. Below the input field, a message reads: "Please enter a valid number of vouchers for your challenge, without commas!". A link "Click here to download these codes as a CSV file" is provided. At the bottom, there is a "Display Codes Below" button.

*Figure 5.13: Error message for generating more Vouchers Codes*

Then, the number “100” was entered and the form submitted. The result is shown in **Figure 5.14**, with the confirmation message written to the browser and the number of **Voucher Codes** updated to 600.

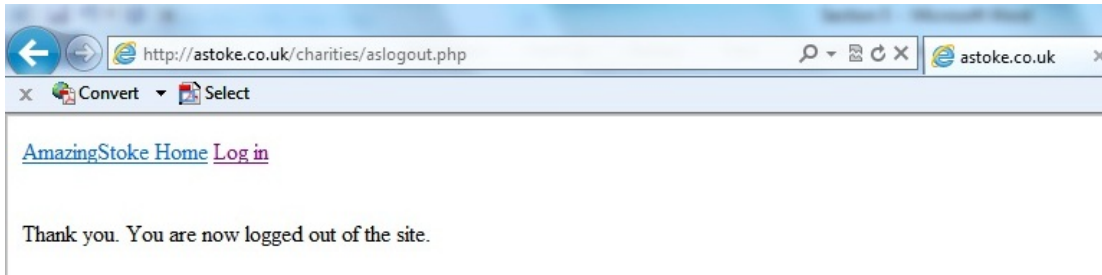


The screenshot shows the same web browser window. The user is still logged in as [johnsmith@mynewcharity.org.uk](mailto:johnsmith@mynewcharity.org.uk). The page title is "Voucher codes for Donate £5 to My New Charity". The total number of vouchers is now 600. A message reads: "Thank you. 100 more Voucher Codes have now been generated for this Challenge." The text input field now contains "0" and the "Generate More Voucher Codes" button is visible. Below the input field, the link "Click here to download these codes as a CSV file" is provided. At the bottom, there is a "Display Codes Below" button.

*Figure 5.14: Confirmation of generating more Vouchers Codes*

## 5.1.6 Logout

The final functionality to test was the program `aslogout.php`. **Figure 5.15** shows the confirmation screen after the NFP clicks the “Log Out” link.



*Figure 5.15 NFP Site confirmation of successful logout by NFP*

## 5.2 Game

To demonstrate the functionality of the Game, I used my real-world Facebook account, with Facebook ID of 762451966 (an older 9-digit account number).

### 5.2.1 Create Account Via Facebook

To create a new Player account in the *AmazingStoke* system, I first logged in to my Facebook account and searched for the *AmazingStoke* in Facebook's top search bar. As the app had been loaded into Facebook as described in 4.2.3.2, it appeared in the search results as shown in **Figure 5.16**.

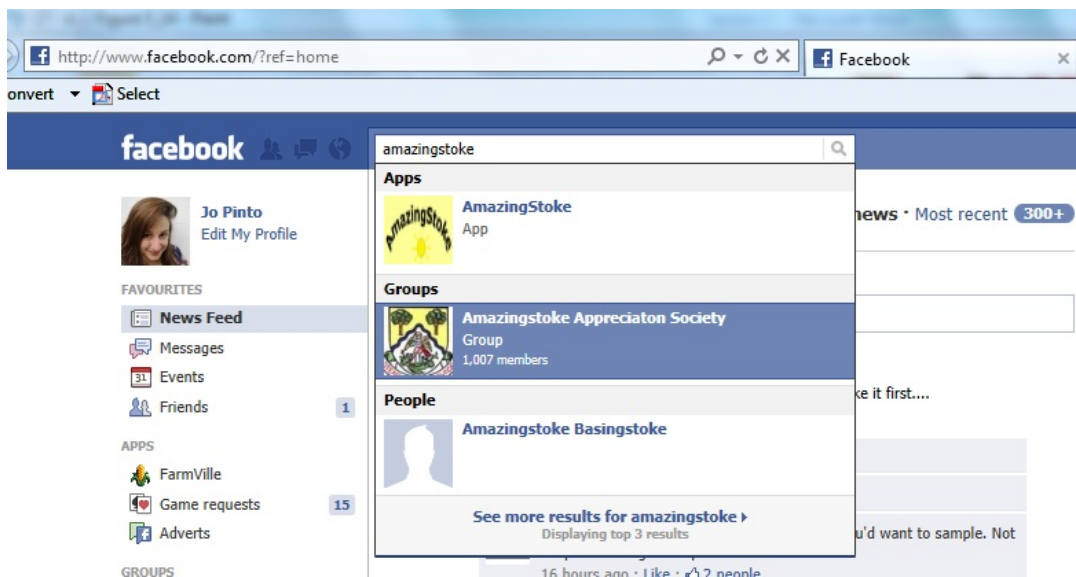


Figure 5.16: Facebook Search results for “AmazingStoke”

By clicking on the link to the *AmazingStoke* app, Facebook displayed the authentication page for a potential Player to authorise the app to access their basic Facebook information, as shown in **Figure 5.17**.

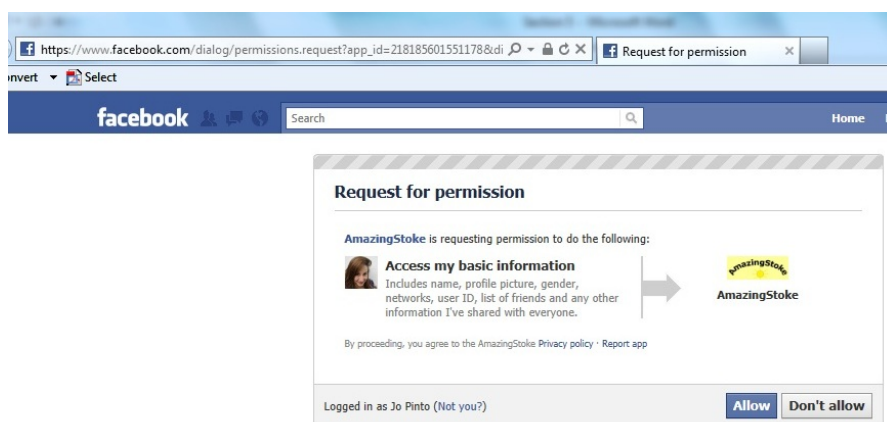
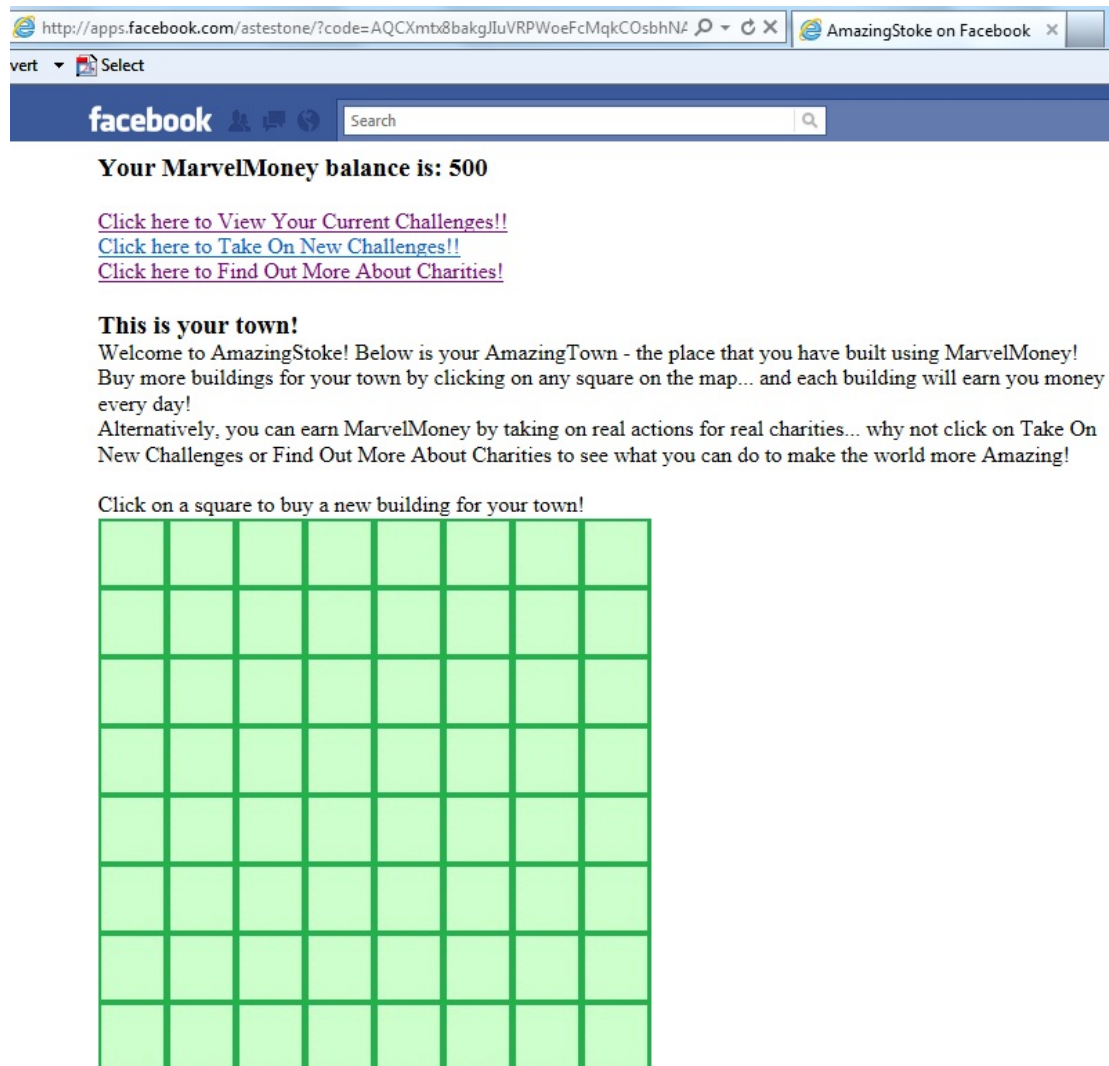


Figure 5.17: Authorisation of access to Facebook information for AmazingStoke app

Clicking on the link to “Allow” the app then redirects to the Game home, showing a blank Map and a **MarvelMoney** balance of 500, as shown in **Figure 5.18**.



*Figure 5.18: Game Home within Facebook showing Map and MarvelMoney balance of 500*

By using phpMyAdmin to view the Database, it is also possible to see the successful insertion of a new Player in the **Users1** relation (see **Figure 5.19**), and the creation of a Player Challenges relation (see **Figure 5.20**).



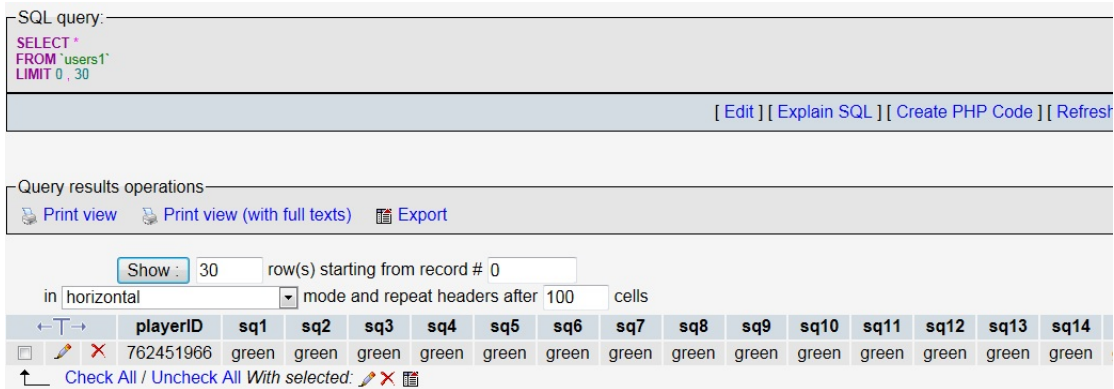


Figure 5.19: phpMyAdmin display of Player account for Facebook User 762451966 after `index.php`

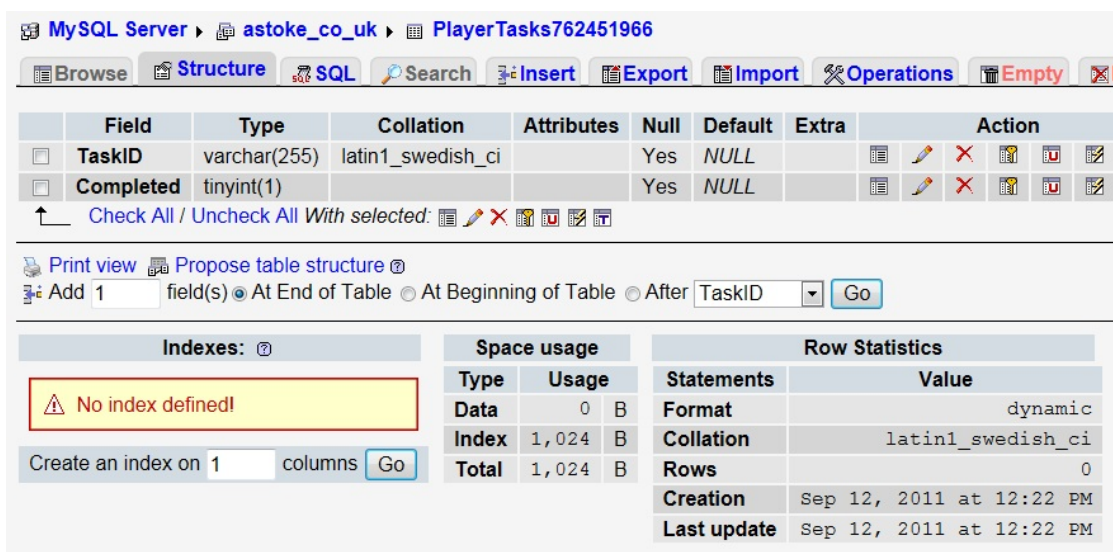


Figure 5.20: phpMyAdmin display of Player Challenges relation for Facebook User 762451966 after `index.php`

## 5.2.2 Log Back In Via Facebook

Once the Player has authorised to *AmazingStoke* app, a link to the app is automatically added to the Player's main Facebook home page (this is built-in functionality of Facebook, shown in **Figure 5.21**).



*Figure 5.21: Facebook homepage with a link to the authorised AmazingStoke app*

To return to the Game, it is then a simple matter of clicking on the small *AmazingStoke* icon displayed on the Player's Facebook homepage.

### **5.2.3 Buy A Building**

Having returned to the Game as described in **Section 5.2.2**, the initial Map is displayed as shown in **Figure 5.18**. As instructed on the screen, the Buy A Building View is accessed by clicking on the Square of the Map on which the Building is to be placed.

Initially in this case, the Square selected is the top right-hand Square. Clicking on this displays the Buy A Building form as shown in **Figure 5.22**.

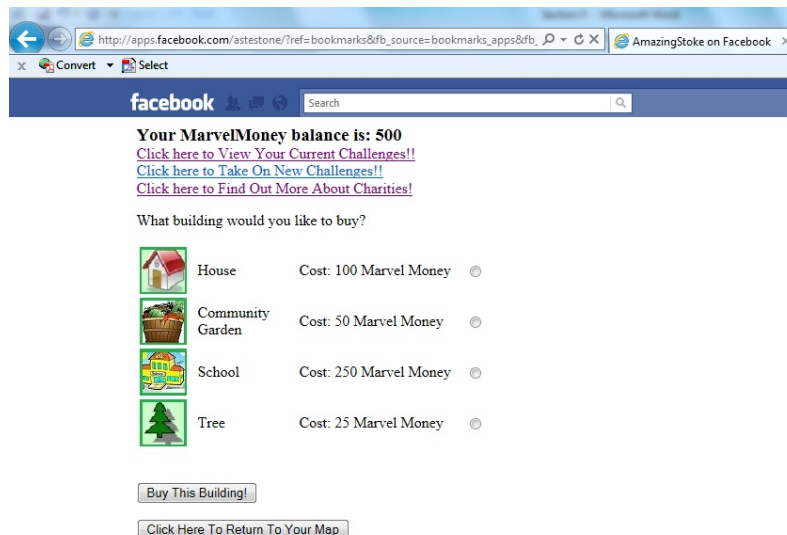


Figure 5.22: Buy A Building View within Facebook

By selecting a Tree from the form, and clicking the “Buy This Building!” button, the Player’s information is updated both in the Browser and on the Database, and the view changes back to the Map view as shown in **Figure 5.23**. This demonstrates that both the image in the top right-hand square and the **MarvelMoney** amount have changed.



Figure 5.23: Map View after Buying A Building

**Figure 5.24** shows the use phpMyAdmin to view the Database to see the successful update of the **Users1** relation via a behind-the-scenes Ajax call from the Game. As can be seen, the value of **sq8** is “tree”, whereas previously it had been “green”.

Although not shown, at this point various other Buildings were purchased to demonstrate the functionality in **Section 5.2.7**.

	playerID	sq1	sq2	sq3	sq4	sq5	sq6	sq7	sq8	sq9	sq10	sq11	sq12	sq13	sq14	sq15
<input type="checkbox"/>	762451966	green	green	green	green	green	green	green	tree	green	green	green	green	green	green	gre

Check All / Uncheck All With selected:

Figure 5.24: phpMyAdmin display of Users1 relation after index.php

## 5.2.4 View And Accept A New Challenge

Testing now took place to Accept the Challenge posted by mock NFP organisation My New Charity in **Section 5.1.3**. From the Map view, the link “Click here to Take On New Challenges!!” was followed to launch `alltasks.php`, as shown in **Figure 5.25**.

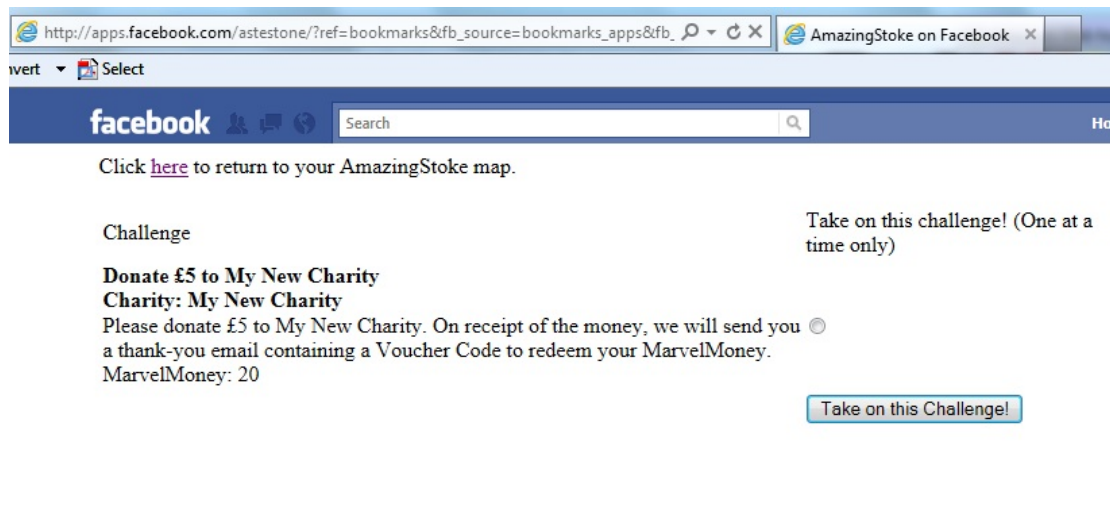


Figure 5.25: View Challenges within Facebook

**Figure 5.25** shows that the Challenge added in **Section 5.1.3** has successfully been retrieved from the Database by the Game. At this point, it is the only Challenge listed on the Database, but a more populated `AsCharityTasks` relation would result in a longer list of available Challenges.

Upon clicking the radio button and “Take on this Challenge!”, the Player is taken to the confirmation screen shown in **Figure 5.26**.

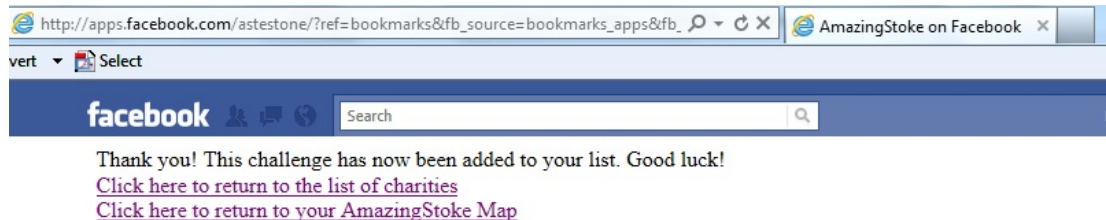


Figure 5.26: Challenge acceptance confirmation within Facebook

Figure

## 5.2.5 View NFP Information

To view information on the NFP My New Charity, the link “Click here to Find Out More About Charities!” was clicked from the Map view to launch `orglist.php` as shown in **Figure 5.27** (note that a second mock NFP, Fans Of Cats, was added to the Database for testing purposes).



Figure 5.27: List of NFPs using AmazingStoke within Facebook

By clicking on the button “Find out more!” next to My New Charity, the program `orginfo.php` is launched using the details of My New Charity passed via an HTTP POST request. The resultant page is shown in **Figure 5.28**.



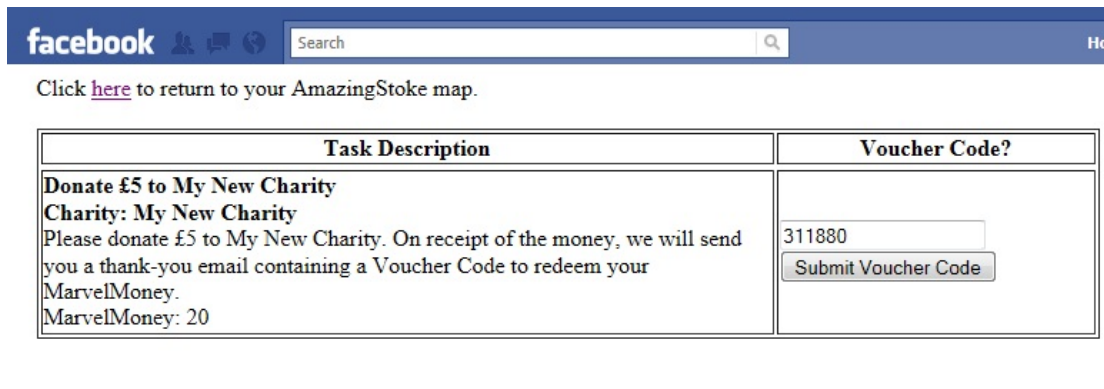
Figure 5.28: Information held on My New Charity on AmazingStoke within Facebook



It can be seen from **Figure 5.28** that the changes to information made in **Section 5.1.2** and **Section 5.1.3** are reflected within the Game.

## 5.2.6 Authenticate A Challenge


To authenticate an accepted Challenge, the link “Click here to View Your Current Challenges!!” was clicked from the Map view to launch `yourtasks.php` as shown in **Figure 5.29**.



Task Description	Voucher Code?
<b>Donate £5 to My New Charity</b> <b>Charity: My New Charity</b> Please donate £5 to My New Charity. On receipt of the money, we will send you a thank-you email containing a Voucher Code to redeem your MarvelMoney. MarvelMoney: 20	311880 <input type="button" value="Submit Voucher Code"/>

*Figure 5.29: Accepted Challenges for AmazingStoke within Facebook*

For the purposes of testing, on the Voucher Codes (311880) shown in **Figure 5.12** was entered on the form and “Submit Voucher Code” clicked to launch `verify.php`. The confirmation of this Challenge authentication is shown in **Figure 5.30**.



facebook Search

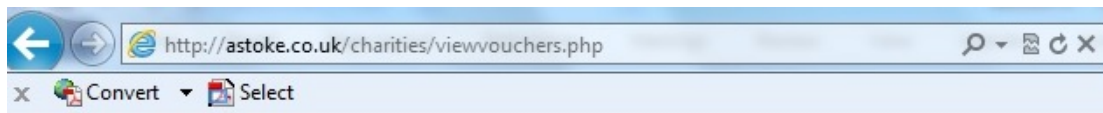
Congratulations! That Challenge is now marked as completed, and you have earned 20 MarvelMoney!

Click [here](#) to return to your AmazingStoke map.

*Figure 5.30: Confirmation of correct Voucher Code for AmazingStoke within Facebook*

On returning to the Map view, **MarvelMoney** balance has been updated, and the Challenge no longer shows when “Click here to View Your Current Challenges!!” is clicked.

The changes to the Database, that is, the removal of Voucher Code 311880 from the list of available codes, can be verified by use of phpMyAdmin, or by logging back in to the NFP Site and using the JavaScript functionality of **Section 5.1.4** to display the remaining Voucher Codes onscreen.



Logged in as: johnsmith@mynewcharity.org.uk  
[Home](#) | [Add A Challenge](#) [Edit Your Description](#) [Log out](#)

### Voucher codes for Donate £5 to My New Charity

Total vouchers = 499

[Click here to download these codes as a CSV file](#)

Or click the button to display the voucher codes below

Display Codes Below

311861  
311862  
311863  
311864  
311865  
311866  
311867  
311868  
311869  
311870  
311871  
311872  
311873  
311874  
311875  
311876  
311877  
311878  
311879  
311881  
311882  
311883  
.....

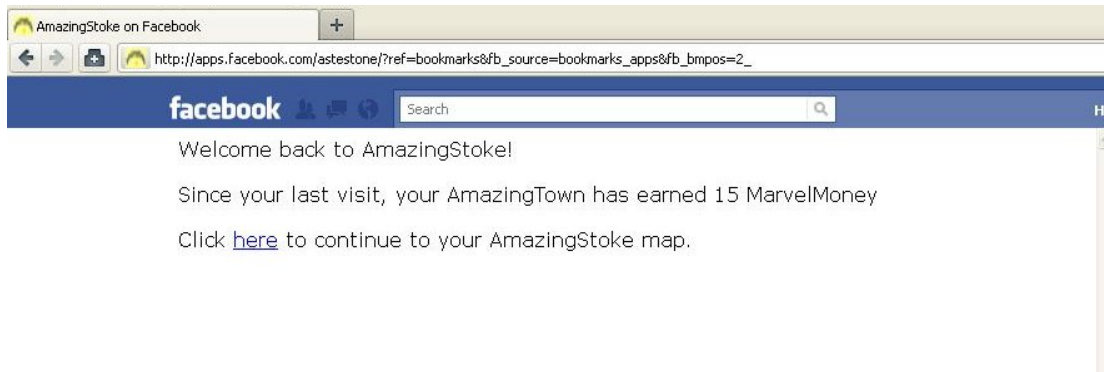
*Figure 5.31: NFP Site View Vouchers for logged in NFP My New Charity, with Voucher Codes 311880 no longer shown in list*

As can be seen in **Figure 5.31**, the Voucher Code 311880 is now no longer listed as available.

## 5.2.7 Update A Player's MarvelMoney On Login

One day after the initial Player account was created as described in **Section 5.2.1**, the Game was logged back in to as described in **Section 5.2.2**.

On this occasion, on logging in, before reaching the Map View, notification was displayed that the **MarvelMoney** balance had been updated based on the current Map. This notification is shown in **Figure 5.32**.



*Figure 5.32: Confirmation of updated MarvelMoney balance when logging in to AmazingStoke on unique day within Facebook*

These changes are also reflected in the **Database**.



## **Chapter 6: Areas For Exploration**

As stated in **Section 4.4**, it was not possible to complete all the intended functionality of the *AmazingStoke* system, as I underestimated the amount of time involved in learning the necessary amount of PHP and JavaScript, and implementing the first two iterations of the system.

This chapter therefore talks through the functionality that was intended for inclusion in the system and which would be included in the development of *AmazingStoke* as a fully-fledged system (as opposed to the prototype that it was possible to build during this project).

### **6.1 Development Technologies**

#### **Templating**

It was the intention to use a PHP templating system, in order to separate the logic of the PHP programs, from the layout of that page (i.e. the HTML output) (Lerdorf, Tatro, & MacIntyre, 2006). This would allow the programming and the design of the web pages to be separate, increasing flexibility and reusability within the system. It should be noted that templating can incur a performance penalty to the user as it requires more calls to the server, and its advantage is to the program creator rather than the user.

It was the intention of this project to use the popular PHP templating engine *Smarty*, which turns templates into PHP code and caches them, replacing only when the template is changed.

However, having built the initial iterations of the system and looking into *Smarty* in more detail, it became apparent that extending this across the site would require more time than had been allowed for this. The use of PHP templating would be a priority addition for the next iteration of the *AmazingStoke* system.

#### **JavaScript And PHP Objects**

**Chapter 3** made significant use of an object-oriented design methodology in creating a design for the system. However, in the implementation of this project using PHP and JavaScript – both of which support object-oriented programming – only the built-in objects of the programming languages were included. Greater use of object-oriented programming – as opposed to the mostly functional programming implemented in the first two iterations of the system – could have led to a more reusable, adaptable and robust system.

As an example, aspects of the system such as **Maps** and **Buildings** were designed as objects with methods and attributes in **Chapter 3**, but were not implemented as objects in the current system.

Implementing these a JavaScript objects on the client-side could have led to simpler, more robust code, and could have made it easier to implement a greater amount of client-side functionality, and therefore future iterations of the system should involve a shift towards

greater use of objects in the code. This would have made it easier for the program creator to implement a more seamless user experience.

## The jQuery Library

As noted in **Section 4.1.3**, one of the difficulties with the JavaScript API is the major differences in the way that JavaScript is implemented in different browsers (as seen, for example, in the creation of XMLHttpRequest object in **Section 4.3.2.3**). One system to assist in the creation of JavaScript is jQuery, a free library to make common JavaScript tasks easier and manage the differences between browsers (Flanagan, 2011). jQuery typically uses a CSS selector to specify a set of elements within the document, then returns an objects representing the set. The object's methods can then be used to modify all the elements represented by the object.

It was my intention to learn to use the jQuery library – both as a personal learning objective and to assist in the creation of JavaScript for this system. However, once I had explored the basics of JavaScript as implemented in this system, insufficient time remained to then learn jQuery as a tool for creating and managing JavaScript functionality. In hindsight, earlier adoption of the jQuery library could have meant greater time for implementing more of the extensions to the system detailed in this chapter.

## Graphics And Cascading Style Sheets

As described in **Section 4.3.1**, a PHP templating engine such as *Smarty* allows for the separation of programming logic from presentation. In addition to this, Cascading Style Sheets (CSS) which were described briefly in **Section 4.3.1**, and allow presentation information regarding the HTML to be stored as a separate `.css` file.

It was also the intention to use CSS technology to improve the overall appearance of both the NFP Site and the Game. However, unfortunately this was again something that there was not sufficient time to undertake on this project. Creation of more aesthetically pleasing visuals is foreseen as an addition for a later iteration of the *AmazingStoke* system.

## 6.2 System Functionality

### Social Network Integration

As noted in the original project proposal and throughout this report, one of the expected advantages of implementing the **Game** as an app within Facebook was the planned use of the Facebook API for social networking within the **Game**. The Graph API was discussed in **Section 4.1.5**, and researching the API suggested that it could be used for finding and adding Facebook friends as in-**Game** contacts, messaging, and setting challenges for other **Players**.

As explained in the Facebook Developers site (Facebook Developers, 2011), the object to represent a Facebook account holders in the Graph API is a `User` object. This object holds various attributes of the Facebook account holders. In addition, the `read_friendlists` method on a `User` object returns the Facebook account holder's friends as an array of names and Facebook User IDs (Facebook Developers, 2011).

Due to the issues already discussed regarding the amount of time spent learning the basic PHP and JavaScript tools, and the issues regarding changes to the hosting of Facebook apps as discussed in **Section 4.1.5**, the intended social networking functionality was not added in to this version of the *AmazingStoke* system. However, this would be a priority addition to the next iteration of the *AmazingStoke* system.

### Pop-Out Games

It was suggested in the original project proposal that clicking on the Buildings of an *AmazingStoke* Map could launch separate mini-games (such as Noughts and Crosses), either in separate pop-out windows or within the main `iframe`. It was noted from the original project proposal onwards that these additional games would not be a key part of the system, and this is functionality that could be added in at a much later iteration. It could also be added in for certain **Buildings** and not others, and is envisioned as a series of flexible plug-ins for the main *AmazingStoke* system.

### System Security

For the **Game** site, user login information is stored and managed by Facebook, and additional security is not required. In addition, the *AmazingStoke* **Database** does not hold any information on the **Players** other than their Facebook User IDs at this stage. However, a greater amount of information is held about **NFPs** using the *AmazingStoke* system. In launching a full version of this system, it would be necessary to significantly improve the security of the system, both in terms of access to **NFP** data and potential malicious attacks on the **NFP Site**.

Nixon notes in his book *Learning PHP, MySQL and JavaScript* that a programmer should also ensure that data sent using a POST or GET request should be processed to ensure

malicious code has not been inserted (Nixon, 2009). He discusses various options of “sanitising” user input – that is, ensuring that it does not contain malicious code. He also presents a series of functions to prevent escape characters being injected into a string for MySQL, getting rid of unwanted slashes, and removing any HTML from user input. A priority task for the next iteration of the *AmazingStoke* system would be the creation and utilisation of the necessary functions to sanitise all user input to the system from the various HTML forms.

In addition to this, the password security for **NFPs** needs to be improved in the next iteration of the system. A plausible way of doing this would be to use a one-way function (such as PHP md5 or sha1 functions, which convert strings to 32-character and 40-character hexadecimal numbers respectively). In addition, adding additional text to strings before the one-way function is carried out would help prevent brute-force attacks on the **NFP Site**.

## 6.3 Further Developments

### Standalone Game

It was noted in **Section 3.4.2** that some popular Facebook apps such as *FarmVille* (Zynga Games Inc., 2011) are mirrored in a stand-alone version that does not include the Facebook functionality but are still played casually online. Although it falls beyond the scope of this project, a potential future iteration of the *AmazingStoke* system could be the inclusion of an alternate **Game View / Controller**, hosted on a separate site and connecting to the same relations of **NFPs** and **Challenges** as the Facebook version of the **Game**.

### Mobile Gaming

As stated in the original project proposal, the very earliest idea for the *AmazingStoke* game was as a game for an internet-enabled phone such as the iPhone or an Android phone. This idea was not implemented for various reasons discussed in the original project proposal. However, it is possible that a version of the Game could be developed for mobile phones, and although it falls outside the scope of this project, this alternate version could include pop-up games and different volumes of information downloaded at login.

## Peer-To-Peer Challenge Creation And Rewarding

One suggestion that was put forward when discussing the *AmazingStoke* system informally with **NFPs** was the suggestion that the system could include mechanisms for **Players** to add **Challenges** to the **Database** themselves. That is, rather than **NFP** organisations taking a top-down approach to social change using gaming, individuals would be able to challenge and reward each other.

The value of peer-to-peer challenge setting in gaming was discussed in the original project proposal. Online games already using such a concept include *Chore Wars*, an online game in which members of a household create avatars and award each other in-game points depending on the amount of household tasks they complete (Davis, 2007).

It could be possible to create separate versions of the *AmazingStoke Game* in which **Players** can create as well as set **Challenges** for each other. These versions could be downloadable standalone versions, which do not overlap with the moderated **Game** described in this project report. This idea would take the **Game** outside the scope of a System Administrator and see it being used as tool by which community groups, organisations or individuals could “gamify” (McGonigal, 2011) any aspect of social engagement that they choose. As with the mobile version described above, this game could be enhanced by the addition of pop-out games.

Although this idea fell outside the scope of the project, it holds interesting potential for development of the existing *AmazingStoke* system.

## **Chapter 7: Conclusions**

The initial proposal was to create an online game for the NFP sector, which would allow NFPs to reward supporters for real-world actions via a Facebook game. It was intended that use of the Facebook platform would not only make it easier for players to sign up and return to the game, but that the system would leverage social networking features of Facebook such as linking other players as friends or sending in-game requests to friends. At the start of the project, I had some theoretical knowledge of designing and implementing a web-based application, but little practical experience of building web-based systems.

The system described in this report implements some of the basic functionality listed in **Section 3.1**. A basic NFP site was built, allowing NFPs to log in, add challenges to the system, view available voucher codes, generate more codes, and update their organisational information. A basic game was built and successfully hosted within the Facebook site, allowing players to register for the game, buy in-game buildings using the credits known as MarvelMoney, take on challenges, and find out more about NFPs using the system. In certain places, the game used JavaScript and Ajax to make the player experience fast and seamless whilst carrying out behind-the-scenes calls to the server to make changes to a database.

I underestimated the amount of time involved in learning PHP and JavaScript, and this meant that many features intended in the original proposal were not implemented in this version of the system. In particular, the social networking features of Facebook were not fully exploited in this version of the application. Issues such as scaling Facebook applications, pop-up games and additional security were not explored as hoped in this project. Although user testing was carried out informally throughout the development of the system, and a walkthrough included in **Chapter 5**, a formal set of user tests with NFPs and players was not carried out as originally intended.

In addition, my plan to learn the basics of the tools and then “add in” at a later point technologies such as PHP templating and jQuery meant that the *AmazingStoke* system at this point lacks the sophistication and modularity for which I had hoped.

The design methodology used – an object-oriented approach using UML – was reasonably thorough and provided a solid framework for the implementation of the system. However, my actual approach to the implementation did not incorporate many object-oriented features, and so parts of the design work were unused in the implementation. This, in addition to the time spent learning PHP, leads me to the conclusion that although I gained a lot personally by learning PHP, the system might have been more developed and elegant had I used Grails (SpringSource, 2009) as the server-side language.

Overall, I believe that the *AmazingStoke* system described in this report stands as an interesting prototype for the intended system. I was able to carry out solid background research relating to the idea for the initial project proposal, and implement enough functionality to create the basis for a full working system. I also learned a huge amount over the course of this project, both in terms of programming skills (PHP and JavaScript) and also

regarding the challenges of developing web-based systems, such as browser compatibility issues and issues relating to integrating application into an existing system such as Facebook.

The *AmazingStoke* system described in this system did not live up to all my initial hopes and goals, due to my inexperience as a programmer and some of the planning decisions (such as leaving investigation of jQuery to the end of the project). However, the learning from this project in the development of web-based applications will be of great benefit to me in the future. It is also my hope that the research undertaken and the prototype developed means that development of the *AmazingStoke* system could carry on after this project, to one day create a genuinely valuable tool for charities and other not-for-profit organisations.

## References

- Argerich, L., Lea, C., Egervari, K., Anton, M., Hubbard, C., Fuller, J., et al. (2002). *Professional PHP4 XML*. Wrox Press.
- Avison, D., & Fitzgerald, G. (2002). *Information Systems Developer: Methodologies, Techniques & Tools. 4th Edition*. Maidenhead, Berkshire: McGraw-Hill Education.
- Charity Commission. (2011). Retrieved September 15, 2011, from The Charity Commission for England and Wales: <http://www.charity-commission.gov.uk/>
- Comer, D. (2004). *Computer Networks and Internets. 4th Edition*. Pearson.
- Davis, K. 2007. Chore Wars (2007). Retrieved 7 April 2011, from Chore Wars. <http://www.chorewars.com>
- Dennis, A., Wixom, B., & Tegarden, D. (2010). *Systems Analysis and Design with UML: An Object-Oriented Approach, 3rd Edition*. Hoboken, New Jersey: Wiley.
- EasyPHP. (2011). Retrieved September 15, 2011, from EasyPHP: <http://www.easyphp.org>
- Eklund, K., & McGonigal, J. (2007). Retrieved April 2, 2011, from World Without Oil: <http://www.worldwithoutoil.org>
- Facebook Developers. (2011). Retrieved September 15, 2011, from Facebook Developers: <http://developers.facebook.com>
- Facebook Inc. (2011). Retrieved September 15, 2011, from Facebook: <http://www.facebook.com>
- Feiler, J. (2008). *How To Do Everything: Facebook Application*. New York: McGraw Hill.
- Flanagan, D. (2011). *JavaScript: The Definitive Guide. 6th Edition*. Sebastopol, California: O'Reilly Media Inc.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis: Addison-Wesley.
- Gibson, O. (2006, December 5). Charity hopes to make real point in virtual game world. *The Guardian*, p. 5.
- Heinemeier Hansson, D. (2003). Retrieved September 19, 2011, from Ruby on Rails: <http://rubyonrails.org/>
- Jacobs, S. (2006). *Beginning XML with DOM and AJAX*. Apress.
- JustGiving. (2011). Retrieved September 15, 2011, from JustGiving: <http://www.justgiving.com>
- Larman, C. (2004). *Applying UML and Patterns: An introduction to object-oriented analysis and design and iterative development*. Upper Saddle River, New Jersey: Prentice Hall PTR.
- Lerdorf, R., Tatroe, K., & MacIntyre, P. (2006). *Programming PHP*. Sebastopol, California: O'Reilly Media Inc.



- McGonigal, J. (2011). *Reality Is Broken: Why Games Make Us Better and How They Can Change The World*. London: Jonathan Cope.
- Microsoft. (2011). Retrieved September 19, 2011, from Active Server Pages:  
<http://msdn.microsoft.com/en-us/library/aa286483.aspx>
- Moller, A., & Schwartzbach, M. (2006). *An Introduction to XML and Web Technologies*. Indianapolis: Addison-Wesley.
- MySQL. (2011). *About MySQL*. Retrieved September 15, 2011, from MySQL:  
<http://www.mysql.com/about/>
- Nixon, R. (2009). *Learning PHP, MySQL and JavaScript*. Sebastopol, California: O'Reilly Media Inc.
- Office of the Scottish Charity Regular. (2011). Retrieved September 15, 2011, from Office of the Scottish Charity Regular: <http://www.oscr.org.uk>
- One.com. (2011). Retrieved September 15, 2011, from One.com: <http://www.one.com>
- Ramakrishna, R., & Gehrke, J. (2003). *Database Management Systems. 3rd Edition*. Maidenhead, Berkshire: McGraw-Hill Education.
- See The Difference. (2011). Retrieved September 15, 2011, from See The Difference :  
<http://www.seethedifference.org>
- SpringSource. (2009). Retrieved September 15, 2011, from Grails: <http://grails.org>
- Sun Developer Network (SDN), Oracle. (2010). Retrieved September 19, 2011, from JavaServer Pages Technology: <http://java.sun.com/products/jsp/docs.html>
- Twitter. (2011). Retrieved September 15, 2011, from Twitter: [www.twitter.com](http://www.twitter.com)
- UN World Food Programme. (2009). Retrieved April 2, 2011, from FreeRice:  
<http://www.freerice.com>
- Virgin Money. (2011). Retrieved September 2011, 2011, from Virgin Money Giving:  
<http://www.virginmoneygiving.com>
- W3C. (2011). *Browser Statistics*. Retrieved September 15, 2011, from W3Schools:  
[http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)
- W3C. (2011). *CSS Introduction*. Retrieved September 15, 2011, from W3Schools:  
[http://www.w3schools.com/css/css\\_intro.asp](http://www.w3schools.com/css/css_intro.asp)
- W3C. (2011). *Standards*. Retrieved September 15, 2011, from W3C: <http://www.w3.org/standards/>
- WebCheatSheet.com. (2009). *Creating Word, Excel and CSV files with PHP*. Retrieved September 15, 2011, from WebCheatSheet.com:  
[http://www.webcheatsheet.com/php/create\\_word\\_excel\\_csv\\_files\\_with\\_php.php#csvheader](http://www.webcheatsheet.com/php/create_word_excel_csv_files_with_php.php#csvheader)

Whitt, J. (2005, November 18). *PHP Session / Cookie in Frames Using Internet Explorer*. Retrieved September 15, 2011, from James Whitt's Blog:  
<http://james.jamesandkristin.net/2005/11/18/php-session-cookie-in-frames-using-internet-explorer>

Yahoo! Inc. (2011). *Best Practices for Speeding Up Your Web Site*. Retrieved September 15, 2011, from Yahoo! Developer Network: <http://developer.yahoo.com/performance/rules.html>

Zynga Game Network Inc. (2011). Retrieved September 15, 2011, from CityVille:  
<http://apps.facebook.com/cityville/>

Zynga Games Inc. (2011). Retrieved September 15, 2011, from FarmVille:  
<http://apps.facebook.com/onthefarm>

# Appendix A: Use Cases

## Use Case 1: Login To Game

<b>Use Case</b>	1: Login To Game
<b>Description</b>	Use Case for Player login to Game
<b>Actors</b>	Player, Game, Facebook Site, Database
<b>Assumptions</b>	Player has Facebook account
<b>Steps</b>	<ol style="list-style-type: none"> <li>1. Player logs in to Facebook</li> <li>2. Player select <i>AmazingStoke</i> app from bookmark or search results</li> <li>3. Facebook performs authentication process for apps</li> <li>4. Game retrieves system \$CurrentDate</li> <li>5. Game retrieves Player information from Database</li> </ol> <p>IF Player not in Database</p> <ol style="list-style-type: none"> <li>6. EXTENDS USE CASE 1.A. Register New User</li> <li>ELSE IF DateLastPlayed != CurrentDate</li> <li>7. EXTENDS USE CASE 1.B. Update Player Account</li> </ol> <p>END IF</p> <ol style="list-style-type: none"> <li>8. Game sets \$MapArray</li> <li>9. Game sets \$BuildingArray</li> <li>10. Game retrieves Challenges information from Database and sets \$ChallengeArray</li> <li>11. PERFORM USE CASE 2 Display Map View</li> </ol>

## Use Case 1.a: Register New Player

<b>Use Case</b>	1.a: Register New Player
<b>Description</b>	Use Case for registering new Game Player Extends Use Case 1: Login to Game
<b>Actors</b>	Game, Database
<b>Assumptions</b>	Use Case 1: Login to Game
<b>Steps</b>	<ol style="list-style-type: none"> <li>1. Game sends query to Database (INSERT INTO Users1 VALUES...)</li> <li>2. Game sends query to Database (CREATE TABLE PlayerChallenges\$playerID)</li> </ol>

### Use Case 1.b: Update Player Account

<b>Use Case</b>	1.b: Update Player Account
<b>Description</b>	Use Case for updating Player login on unique date Extends Use Case 1: Login to Game
<b>Actors</b>	Game, Database
<b>Assumptions</b>	Use Case 1. Login to Game
<b>Steps</b>	FOR (Building in \$MapArray) 1. (MarvelMoney = MarvelMoney + Building.Reward) END FOR 2. Game sends query to Database (SET MarvelMoney='MarvelMoney', DateLastPlayed= '\$CurrentDate')

### Use Case 2: Display Map View

<b>Use Case</b>	2: Display Map View
<b>Description</b>	Use Case for displaying Map View in Game
<b>Actors</b>	Game
<b>Assumptions</b>	Use Case 1. Login to Game
<b>Steps</b>	1. Game write header information to browser FOR (Building in \$BuildingArray) 2. Game displays Building information IF (BuildingsDisplayed % 8 == 0) 3. Game writes   to browser END IF END FOR

### Use Case 3: Buy Building

<b>Use Case</b>	3: Buy Building
<b>Description</b>	Use Case for buying Building in Game
<b>Actors</b>	Player, Game, Database
<b>Assumptions</b>	Use Case 4: Display Buy Buildings View Player has submitted form from Buy Buildings View
<b>Steps</b>	1. Game updates \$MapArray 2. Game updates \$MarvelMoney 3. Game sends Ajax write to Database (SET sq..=., MarvelMoney= MarvelMoney - Building.Cost)

#### Use Case 4: Display Buy Buildings View

<b>Use Case</b>	4: Display Buy Buildings View
<b>Description</b>	Use Case for displaying Buying View in Game
<b>Actors</b>	Player, Game, Database
<b>Assumptions</b>	Use Case 1: Login to Game Use Case 2: Display Map View
<b>Steps</b>	1. Player clicks on square in browser 2. Game writes form to browser FOR (Building in\$BuildingArray) 3. Game displays Building information IF (Player submits form) IF (Building.Cost > MarvelMoneyBalance) 4. PERFORM USE CASE 4.A. Alert Insufficient MarvelMoney ELSE 5. PERFORM USE CASE 3. Buy Building 6. PERFORM USE CASE 2. Display Map View END IF END IF

#### Use Case 4.a: Alert Insufficient MarvelMoney

<b>Use Case</b>	4.a: Alert Insufficient MarvelMoney
<b>Description</b>	Use Case for dealing with error in Use Case 3: Buy Building
<b>Actors</b>	Game
<b>Assumptions</b>	Use Case 3: Buy Building
<b>Steps</b>	1. Game writes to <div> area in browser "Sorry! You don't have enough MarvelMoney to purchase that Building."

#### Use Case 5: Choose New Challenge

<b>Use Case</b>	5: Choose New Challenge
<b>Description</b>	Use Case for adding Challenge to Player Challenges in Game
<b>Actors</b>	Player, Game, Database
<b>Assumptions</b>	Use Case 6: Retrieve NFP Challenges
<b>Steps</b>	1. Player selects Challenge 2. Player submits form 3. Game sends query to Database (INSERT INTO PlayerChallenges\$playerID VALUES (\$taskID, '0') 4. Game writes confirmation message to browser

### Use Case 6: Retrieve NFP Challenges

<b>Use Case</b>	6: Retrieve NFP Challenges
<b>Description</b>	Use Case for displaying Challenge List View in Game
<b>Actors</b>	Player, Game, Challenge
<b>Assumptions</b>	Challenges have been added to the system by NFPs
<b>Steps</b>	1. Player clicks on link to display information 2. Game writes form to browser FOR (Challenge in \$ChallengeArray) 3. Game displays Challenge information END FOR
<b>Extensions</b>	<b>Use Case 6.a: Retrieve NFP Challenges (All)</b> 3.a. FOR Challenge in \$ChallengeArray IF (Challenge.NFP == *) ... END IF END FOR  <b>Use Case 6.b: Retrieve NFP Challenges (SpecificNFP)</b> 3.b. FOR Challenge in \$ChallengeArray IF (Challenge.NFP == SpecificNFP) ... END IF END FOR  <b>Use Case 6.c: Retrieve NFP Challenges (SpecificPlayer)</b> 3.c. Game sends query to Database (SELECT taskID FROM PlayerChallenges\$playerID AND Completed='0') FOR Challenge in \$ChallengeArray FOR (TaskID in Player.Challenges) ... END FOR END FOR

### Use Case 8: Authenticate Challenge

<b>Use Case</b>	8: Authenticate Challenge
<b>Description</b>	Use Case for authenticating Challenge with Voucher Code
<b>Actors</b>	Player, Game, Database, Voucher Code
<b>Assumptions</b>	Use Case 6.c: Retrieve NFP Challenges (SpecificPlayer)
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Player enters \$VoucherCode in form</li><li>2. Player submits form</li><li>3. Game sends query to Database</li></ol> IF (Voucher Code !valid) <ol style="list-style-type: none"><li>4. Game writes error message to browser</li></ol> ELSE IF (Multiple Voucher Codes used) <ol style="list-style-type: none"><li>5. Game sends query to Database (DELETE FROM Task\$taskID WHERE VoucherCode=\$VoucherCode')</li></ol> END IF <ol style="list-style-type: none"><li>6. Game sends query to Database (UPDATE PlayerChallenges\$playerID SET Completed='1' WHERE taskID=\$taskID')</li><li>7. Game sends query to Database (UPDATE PlayerChallenges\$playerID SET Completed='1' WHERE taskID=\$taskID')</li><li>8. Game sends query to Database (UPDATE Users1 SET MarvelMoney = MarvelMoney + Challenge.Reward WHERE playerID=\$playerID')</li><li>9. Game writes confirmation message to browser</li></ol> END IF

### Use Case 9: Find Out About NFPs

<b>Use Case</b>	9: Find Out About NFPs
<b>Description</b>	Use Case for displaying Specific NFP List information in Game
<b>Actors</b>	Player, Game
<b>Assumptions</b>	Use Case 10: Display NFP List View
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Player clicks on specific NFP from NFP List View</li><li>2. Game writes NFP information to browser</li><li>3. Game PERFORMS USE CASE 6.B: Retrieve NFP Challenges (SpecificNFP)</li></ol>

### Use Case 10: Display NFP List View

<b>Use Case</b>	10: Display NFP List View
<b>Description</b>	Use Case for displaying NFP List View in Game
<b>Actors</b>	Player, Game, Database,
<b>Assumptions</b>	Use Case 1: Login to Game
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Player clicks on link to display NFP List View</li><li>2. Game sends query to Database (SELECT * FROM AsCharities) FOR (NFP returned from AsCharities)</li><li>3. Game writes NFP name and link to information</li></ol> END FOR

### Use Case 12: Logout of Game

<b>Use Case</b>	12: Logout of Game
<b>Description</b>	Use Case for Player logout of Game
<b>Actors</b>	Player, Facebook Site
<b>Assumptions</b>	Use Case 1: Login to Game
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Player clicks on Facebook logo from Facebook browser chrome</li><li>2. Facebook loads main Facebook site</li></ol>



### Use Case 13: Load New Challenge

<b>Use Case</b>	13: Load New Challenge
<b>Description</b>	Use Case for NFP to add new Challenge to the Database
<b>Actors</b>	NFP, NFP Site, Database, Challenge
<b>Assumptions</b>	Use Case 12: Login to NFP Site
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP clicks on link from any page to Add New Challenge</li><li>2. NFP Site displays Add New Challenge form</li><li>3. NFP completes form</li><li>4. NFP submits form</li><li>5. NFP Site checks all parts of form complete using JavaScript in the browser</li></ol> <p>IF (form incomplete)</p> <ol style="list-style-type: none"><li>6. NFP writes error message to current page using JavaScript in the browser</li></ol> <p>ELSE</p> <ol style="list-style-type: none"><li>7. NFP Site sends query to Database (INSERT INTO AsCharityTasks VALUES(...))</li></ol> <p>IF (MultipleVoucherCodes = "1")</p> <ol style="list-style-type: none"><li>8. NFP Site generates pseudorandom \$VoucherCodeStart</li><li>9. NFP Site sends query to Database (CREATE TABLE Tasks\$taskID..) FOR (\$i = 0; \$i &lt; NumberVoucherCodes; ++\$i)</li><li>10. NFP Site sends query to Database (INSERT INTO Tasks\$taskID VALUES (\$VoucherCodeStart + \$i) ) END FOR</li></ol> <p>END IF</p> <ol style="list-style-type: none"><li>11. NFP Site redirects NFP to confirmation page</li></ol> <p>END IF</p>

#### Use Case 14: Update NFP Information

<b>Use Case</b>	14: Update NFP Information
<b>Description</b>	Use Case for NFP to make changes to their organisation information in the Database
<b>Actors</b>	NFP, NFP Site, Database
<b>Assumptions</b>	Use Case 12: Login to NFP Site
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP clicks on link from any page to Edit Information</li><li>2. NFP Site displays Edit Information form</li><li>3. NFP completes form</li><li>4. NFP submits form</li><li>5. NFP Site checks if any part of the form contains information IF (text area != "")</li><li>6. NFP Site sends query to Database (UPDATE AsCharities SET...) END IF</li><li>7. NFP Site redirects NFP to confirmation page</li></ol>

#### Use Case 15: View Live Challenges

<b>Use Case</b>	15: View Live Challenges
<b>Description</b>	Use Case for NFP Site to display Live Challenges for NFP
<b>Actors</b>	NFP Site, Database, Challenge
<b>Assumptions</b>	Use Case 12: Login to NFP Site
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP Site sends query to Database (SELECT * FROM AsCharityTasks WHERE CharityNumber = "\$CharityID") FOR (NFP returned from AsCharities)</li><li>2. NFP Site writes Challenge details to browser IF (Challenge.MultipleVouchers = "1")</li><li>3. NFP site writes "View Voucher Codes" link for Challenge END IF</li></ol> END FOR

### Use Case 16: View Voucher Codes

<b>Use Case</b>	16: View Voucher Codes
<b>Description</b>	Use Case for NFP to view Voucher Codes for an existing Challenge
<b>Actors</b>	NFP, NFP Site, Database, Voucher Code
<b>Assumptions</b>	Use Case 13: Load New Challenge Use Case 15: View Live Challenges
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP clicks on “View Voucher Codes” from Live Challenges List</li><li>2. NFP Site redirects NFP to new page in browser</li><li>3. NFP Site sends query to Database (SELECT * FROM Tasks\$taskID)</li><li>4. NFP Site writes COUNT(Voucher Codes) to current page in browser</li><li>5. NFP Site writes “Download as CSV file” link to current page in browser</li><li>FOR (Voucher Code returned from Tasks\$taskID)</li><li>6. NFP Site writes Voucher Code to hidden area of current page in browser</li><li>END FOR</li><li>IF (NFP clicks “Display Voucher Codes”)</li><li>7. NFP Site displays hidden are of Voucher Codes</li><li>END IF</li></ol>

### Use Case 17: Generate New Voucher Codes

<b>Use Case</b>	17: Use Case Generate New Voucher Codes
<b>Description</b>	Use Case for NFP to generate more Voucher Codes for an existing Challenge
<b>Actors</b>	NFP, NFP Site, Database, Challenge, Voucher Code
<b>Assumptions</b>	Use Case 12: Login to NFP Site Use Case 13: Load New Challenge Use Case 16: View Voucher Codes
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP clicks link to “Generate More Voucher Codes” from View Voucher Codes page</li><li>2. NFP Site displays form with text box in current page in browser</li><li>3. NFP enters number of additional vouchers in text box</li><li>4. NFP Site checks form information using JavaScript in browser IF (ValueEntered is NaN or “”)</li><li>5. NFP Site writes error message to current page in browser ELSE</li><li>6. NFP Site generates new pseudorandom Voucher Code start FOR (\$i = 0; \$i &lt; NumberVoucherCodes; ++\$i)</li><li>7. NFP Site sends query to Database (INSERT INTO Tasks\$taskID VALUES (\$VoucherCodeStart + \$i) ) END FOR</li><li>8. NFP Site reloads View Voucher Codes page with confirmation message written to browser END IF</li></ol>
<b>Issues</b>	NFP Site must ensure no duplicate codes generated (as relations cannot contain duplicate rows)

### Use Case 18: Download Voucher Codes

<b>Use Case</b>	18: Download Voucher Codes
<b>Description</b>	Use Case for NFP to download Voucher Codes for an existing Challenge as CSV file
<b>Actors</b>	NFP, NFP Site, Database, Challenge, Voucher Code
<b>Assumptions</b>	Use Case 17: View Voucher Codes
<b>Steps</b>	<ol style="list-style-type: none"><li>1. NFP clicks on “Download as CSV file” link from View Voucher Codes</li><li>2. NFP Site creates new CSV file entitled voucher_codes.csv</li><li>3. NFP Site write row header information to <i>voucher_codes.csv</i> FOR (Voucher Code returned from Tasks\$taskID)</li><li>4. NFP Site writes Voucher Code to <i>voucher_codes.csv</i> END FOR</li><li>5. NFP Site asks NFP to confirm next steps with <i>voucher_codes.csv</i> (dependent on browser)</li></ol>