Final Year Project Report

Project Title: PartyUp

Author: Munaib Hussain

Acknowledgements

I would also like to thank the many users involved throughout the projects development and testing phases who spent a large amount of time testing and completing surveys for the purpose of verifying the applications quality.

Abstract

The absence of a “perfect” matchmaking system for the online community has resulted in the distribution of an inaccurate and flawed system within many games; impacting the quality of matchmaking greatly. Therein lies a major issue as these systems provide players with limited control over this process, resulting in unpredictable outcomes such as: being matched with players from different regions/locations making team oriented play difficult due to connectivity and communication issues.

The existence of these issues confirm the necessity for an alternate matchmaking system that handles this process in a stronger and more robust manner regardless of game or platform, whilst ultimately leaving players in control. This report will cover the background research and development stages of the solution to create a semi-auto matchmaking system for impromptu and user-friendly matchmaking.

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1. Introduction

The existence of an imperfect matchmaking system and the solution of a semi-automated and impromptu matchmaking system will be outlined within this chapter with the various aims and objectives along with the scope of the project being covered.

1.2. Aims and Objectives

Aims:	1. The aim of this project will be to create a web application aimed at providing a semi-automated matchmaking experience to gamers; allowing them to find their ideal partner for their chosen game and location.
Objectives:	1. Propose a new method of matchmaking online gamers using a web application surpasses the current alternative. 2. Complete background research on the proposed system by analysing any existing systems for their strengths and weaknesses. 3. Plan and design the new matchmaking application. 4. Implement the system to the tackle the issue of poor matchmaking and become a solution to this issue. 5. Test and evaluate the system against the outlined requirements and usability aspects to judge its completeness and any improvements needed.

Table 1: Aims and Objectives

1.3. Scope (Limitations and Assumptions)

Scope:	1. The system will record the name of the game the player wishes to find a partner within. 2. The system will retrieve a list of ideal gaming partners. 3. The system will display a list of matching partners filtered based on the player's parameters. 4. The system has the ability to keep track of the players preferences 5. The system has the ability manage a gamer's preferences. 6. The system will be accessed by using a username and password. 7. The system will allow gamers to access their account alter their preferences.
Limitations:	1. The system will only be of use to gamers who wish to change from the existing matchmaking solution. 2. The system is required to have a constant connection to the internet; without it accessing and using the system will not be possible. 3. The development is constrained by time with the system having to be completed within 60 days. 4. There will only be a single developer working on the project.
Assumptions:	1. The required materials and software can be acquired on schedule. 2. That the equipment needed will be available when needed. 3. The chosen framework utilises the expected database version and has not migrated to a different version than I am accustomed to. 4. I have acquired the relevant knowledge to create the proposed solution in its entirety.

	5. The individuals conducting the testing phase will be available when the testing phase is reached.

Table 2: Scope (Limitations and Assumptions)

Chapter 2: Requirements

2.1. Problem Statement

The native matchmaking systems present in the majority of current games should aim to match players with their ideal partners by refining their search criteria; achieved using parameters such as: region, current game and the players' preferences. The process of refining search criteria during matchmaking would result in far more accurate and pleasing results for gamers, thus leading to a more "ideal" matchmaking experience.

The process of matchmaking should be optimized for efficiency, but should also take user control, satisfaction and the accuracy of results into account, thus ensuring the user is satisfied with the results the system provides. However, according to an online survey completed by a group of fifty gamers in 2016 (present within Appendix C) only "10.2%" of participants have never "felt dissatisfied" with a games native matchmaking system", this connotes that a broad range of developers have not satisfied their users' with their systems functionality. By wasting gamers' time with a flawed matchmaking system results in games becoming less likely to be enjoyed, hence creating an unfavourable situation for both gamers and developers alike.

Gamers stand to substantially benefit from the adoption of a new solution to rectify the existing issues, this conclusion is further enhanced through the results of the online survey within Appendix C stating when gamers were asked "Would you find use in a service tailored to find you the ideal individual/team to play with?" "53.1%" of the gamers replied "Yes" to the statement, whilst another "36.1%" replied "maybe". The mentioned statistics clearly highlights that the need for a solution has been recognised by the main demographic it is aimed towards and the adoption of a new, more effective matchmaking source would aid gamers from various platforms. The use of a specialised and tailored application would create a norm that could be applied across various games and systems offering a consistent experience that is currently not available in the existing market and as such could solve a number of issues experienced by a number of gamers in a more centralized manner.

In this project, the alternative matchmaking system and solution will make use of a semi-automated system to retrieve gamers and requests that match by preferences outlined by the user (mainly location, game and gaming tags) only showing incomplete requests. A messaging platform will also be provided to allow gamers to further interact with their matched partners.

2.2. Context Diagram

A context diagram (DFD 0) has been constructed as shown in Figure. 1 below to identify the boundaries of the system within its environment aiding the process of identifying inflows, outflows and events between the system and its external entities along with the manner in which they interact. There were many advantages to creating this diagram, the most impactful being able to get a clear idea of the scope and boundaries of my system.

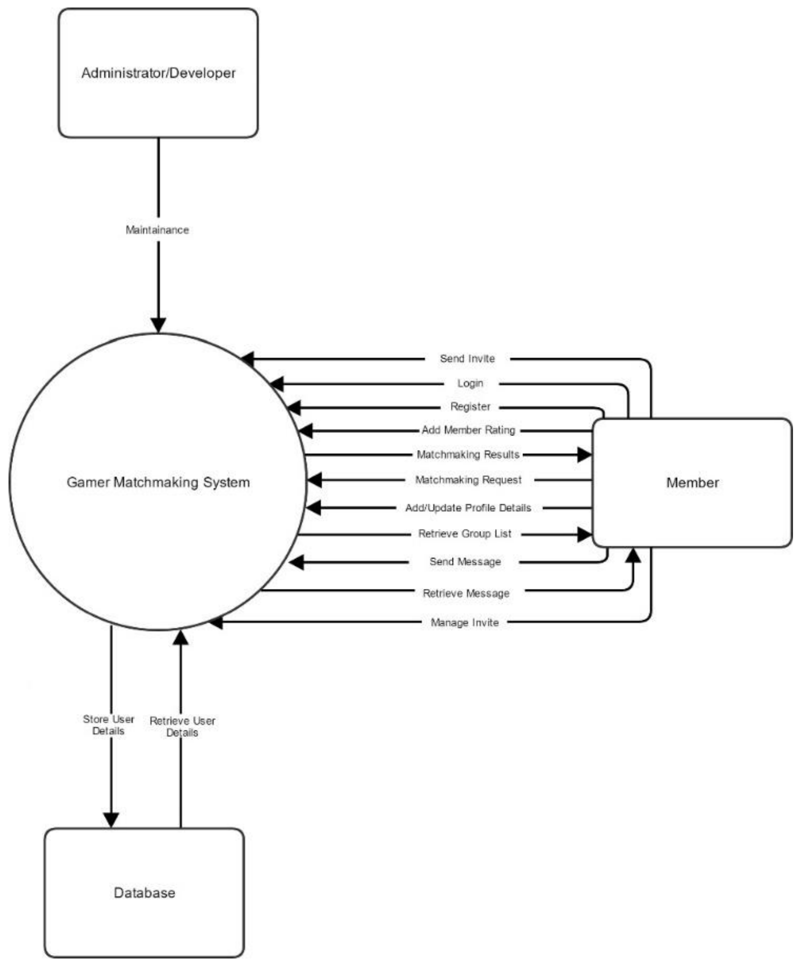


Figure 1: Context Diagram

The context diagram shown in Figure.1 depicts that the main external entities that interact with the system are the '"Administrator/Developer", "Member" and the "Database". These entities along with the inflows and outflows are what the matchmaking system (represented by the circle labelled "Gamer Matchmaking System") needs in order to function, therefore, the boundaries, interfaces and the events for our system have been defined allowing myself as the developer and the stakeholders of the system to understand how the system will function and what data will be needed to achieve the outlined functionality.

2.3. Requirements Elicitation Techniques

The requirements gathering process featured the use of a number of elicitation techniques that shed light on both the functional and non-functional requirements needed as part of this solution, these techniques were purposefully chosen in order to provide the specific data/information needed to derive requirements.

The first adopted elicitation technique was surveys, the use of this technique was chosen due its simplicity and robust nature making the process of distributing and retrieving responses relatively inexpensive and time efficient. The advantages of this method became more apparent when compared to other methods such as interviews which require more time and manpower from both parties to produce results. Some advantages held by surveys include: the fact that they are inexpensive to distribute globally which is especially helpful when the respondents trying to being reached are spread over a large geographical area (as they are within my project) and the fact that they consequently provide the opportunity to receive a large range of targeted responses in a short time frame which is a huge advantage when considering the time limitations of the project.

The survey constructed made use of both open and closed ended questions to elicit the needed responses from the participants. The closed ended questions made use of predefined responses that could numerically represented making the process of evaluating, analysing and drawing conclusions from the data easier. Alternatively, open-ended questions provided a greater level of detail and insight allowing further issues to be brought forward and depicted more clearly.

An excerpt drawn from the survey (shown in Appendix C) that resulted in both a new functional and non-functional requirement being derived was “Have you ever felt dissatisfied with a games inbuilt matchmaking system”, this was a combination of an open and closed ended question with a respondent being able to further specify the reasons for their chosen answer this question not only allowed me to elicit requirements, but it also provided the opportunity to solidify the need for my project. A response to this question stated that they respondent/stakeholder had been “automatically paired with people from different regions” resulting in “server connection issues which in turn causes lag ruining the online gaming experience.” This response was used to identify the need for a functional requirement that allowed members of the system to be able to control the process of matchmaking to avoid the issue of being arbitrarily and “automatically paired” with other users (references in FR1 and FR2). A further response stating “I have also been forced to wait for extended periods of time as matchmaking was unavailable to me” also drawn from the same question within the survey aided in the elicitation of a non-functional requirement (NFR3) which states: “The website will be available 95% of the time”, this allowed me to ensure that the system created as part of my project avoids the errors present in the current solution.

Document analysis was the second means of requirement elicitation adopted, this technique unlike the previously mentioned uses existing documentation from already established systems to gather and elicit requirements rather than conducting independent research as per the previous technique. The main advantage of this method was that since existing data/information was being used I was not forced to start from a blank slate and instead was able to learn from these systems and adopt their strengths and norms, whilst avoiding their mistakes which was extremely advantageous and less time consuming.

When considering potential luxury requirements, the process of analysing existing documentation proved extremely useful. I analysed the documentation of established matchmaking systems by studying their algorithmic approaches to the matchmaking process (as shown in Appendix B).This process allowed me to gain an understanding of how the matchmaking process was handled by revealing: the data being used and how it is used. This was then applied to my project in the form of functional requirements in order to ensure the matchmaking process is accurate and flexible and can be added to in the future. As such I analysed the online matchmaking dating system “Okcupid” present in Appendix B which gave me the opportunity to discover a number of strengths associated within its system that are not present in the current gaming matchmaking systems. These strengths were used to elicit luxury functional requirements (FR13).

The analysis of the mainstream algorithmic processes present within “Okcupid” (shown in Appendix B) highlighted the fact that the matchmaking process was based on “level of importance and what users want their match to answer”, this emphasised the fact that they provided a tailored and up to date matchmaking. This discovery aided in further eliciting the functional requirements that became “The member shall be able to enter their personal profile details into the matchmaking system” and the luxury requirement “The matchmaking system make use automated search algorithm to match the members with suitable partners using their profile details order by closest matching to farthest.” Furthermore, it is evident that current gaming matchmaking systems are devoid of this functionality, therefore this fact was a cornerstone to eliciting these requirements within my project. This highlights an example of how the document analysis elicitation technique aided in identifying functional requirements during the requirements gathering process.

When considering the analysis of existing systems, interface analysis was also a technique used to aid the elicitation of a plethora of requirements as it revealed the data needed by the system, (input and output) along with additional features needed. Furthermore, the fact that they were also lacking functionality revealed a number of requirements also. An advantage of this technique is that it provided an opportunity to quickly and painlessly understand what interface and features would be needed within a matchmaking system. This process was highly efficient when compared to manually conducting this research from users, as such we are able to gain information from a platform already used and take the role of a user to analyse its strengths and weaknesses.

An example of this technique being used within my project is shown within Appendix A when analysing an existing gaming matchmaking website “Destiny LFG”. Whilst analysing this systems user interface and inputs during matchmaking I came across that it is lacking the feature of allowing a user to login and save their details making the process or rerunning the matchmaking process “extremely tedious and time consuming due to the fact the search parameters need to be rendered each time they visit the website” (as stated within Appendix A), this conclusion was used to elicit the functional requirement stating “Members shall be able to login to the system using a username and password (FR5), therefore counteracting this issue revealed during the interface analysis. A further requirement derived from this technique was the functional requirement present stating that “The matchmaking system should allow the member to further filter the results by: game and tags”, this was inspired by the fact the interface of the system (within Appendix A) was extremely intuitive allowing the users of the system to further filter their matchmaking results by gaming mode, mic and location whether they are looking for a single gamer or a party” (DestinyLFG.com, 2015) for a greater degree of accuracy as such this aided in eliciting the functional requirement mentioned above to improve the system in my project overall.

Lastly, within Appendix A it is stated that a system made use of an “integrated chat client, thus allowing users to chat with their fellow gamers” and also provided the “ability to save gamers”, the discovery of these features that provided greater functionality are reflected through FR08 to FR-11.

In conclusion, a range of elicitation techniques have been used during the requirements gathering process to derive and reveal both a series of functional and non-functional requirements, the techniques used where chosen specifically for the advantages they brought forward to the process.

2.4. Use Case Diagram and Descriptions

According to Alexander and Beus-Dukic (2009) “a use case diagram list in outline all the things that a product will have to do”, consequently, utilising this diagram provided a high-level view of all the major processes within my system, this was especially helpful during the requirements gathering/elicitation process as it provided myself as the developer and any stakeholders a clear understanding of what functionality the matchmaking system should provide. Furthermore, the inclusion of a use case descriptions present within Appendix D allowed me to go further than just the models within the use case diagram, allowing me to delve into the details of specific use cases.

The task of capturing functional requirements has the greatest significance as these outline what the system should and do and allow the users to do. The process of identifying the “Basic Flow”, “Alternative Flow” and “Exceptions” (within the use case description) allowed me to not only capture the narrative of the system, but also revealed the various “What if?” scenarios and how the system should handle them. Furthermore, the constant revision of the use cases after identifying these aspects lead me to closer to an ‘ideal’ process by instigating the removal of actions that were not needed by the system or the users. These actions not only lead to the identification and extraction of a concise set of functional requirements, but also ensured the system would be more efficient when developed and designed.

The construction of these diagrams and descriptions allowed me to differentiate between the core and extended functionality of the system, achieved through the analysis of the “Basic Flow”, “Alternative Flow” and “Exceptions” also allowing features outside the projects scope to come to light. This process forced me prioritize the development of certain requirements to ensure I create a usable solution by the deadline. Identifying the core and extra functionality based on the outline derived from the use cases I was able to further refine my catalogue of requirements by ordering them by priority, this was conveyed by labelling requirements as “Essential”, “Luxury” and “Desired”.

Finally, since I have chosen to adopt an iterative life cycle model by developing and implementing modules/parts of the application in phases based on priority; identifying what was within and outside my projects scope and the priority of each requirement aided in ensuring I can adhere to this mythology choice during the development process.

2.5. Use Case Diagrams

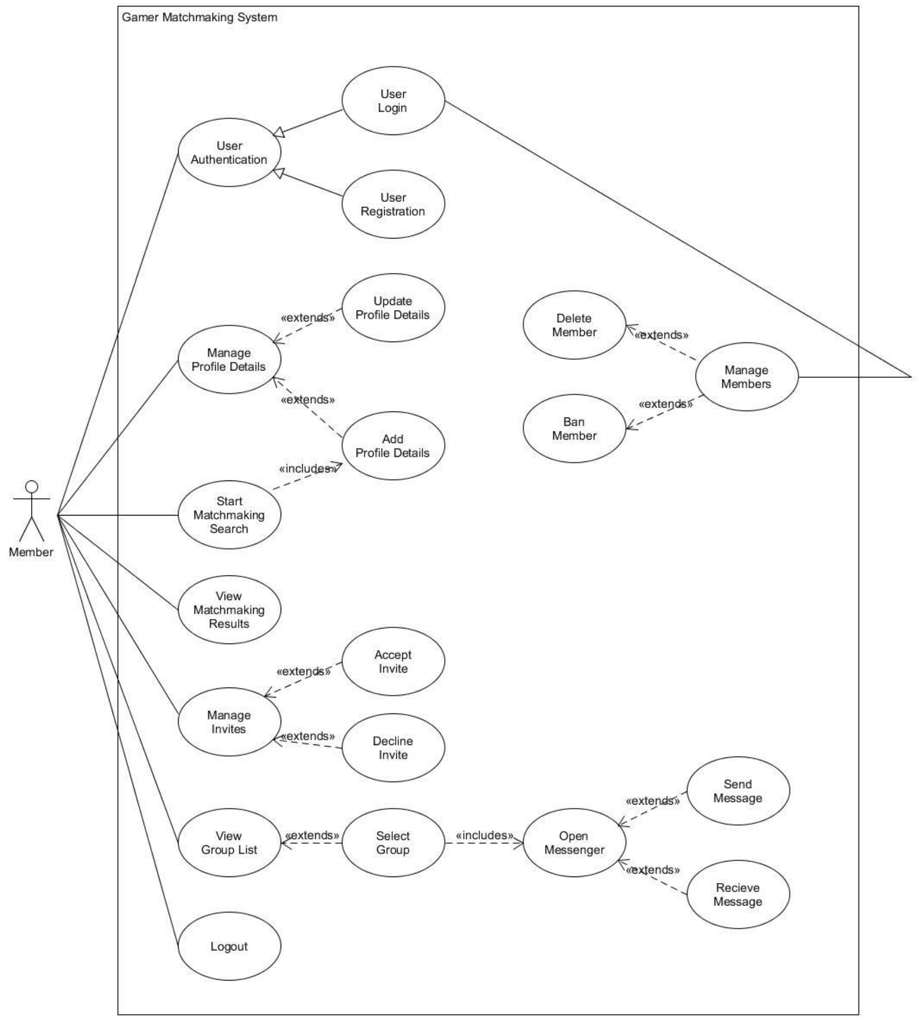


Figure 2: Use Case Diagram 1

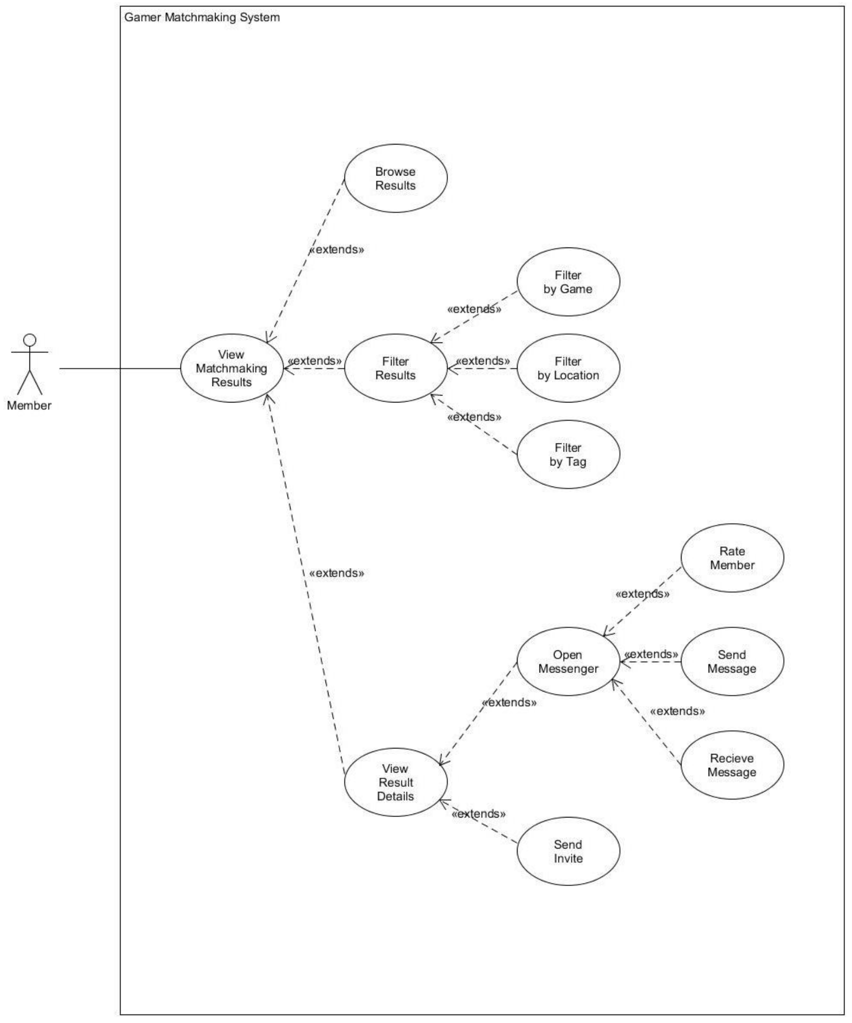


Figure 3: Use Case Diagram 2

2.6. Use Case Justification

The matchmaking system as outlined in both Figures 2 and 3 will contain two types of users, these are the “Administrator” and “Member”. Whilst a the member actor focuses on using the system for its intended purpose of conducting a matchmaking search, the Administrator on the other hand is granted a separate set of features by being tasked with managing members by being able to ban them. The main rationale behind my decision to not simply combine these actors into a single entity was that they both have distinctly separate sets of functionalities within the system and as such they cannot use each other features and can only use the use cases (functionality/features) associated them.

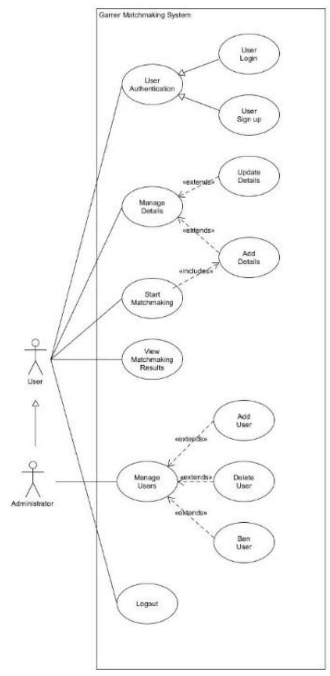


Figure 4: Initial Use Case Diagram

I decided to iterate upon my previous use case diagram in which actor inheritance was used between the two actors as shown by Figure 4, the previous layout allowed the Administrator to inherit all the use cases associated with the Member actor in addition to its own, consequently resulting in the administrator being a member with extra functionality and heightened privileges. However, this idea was iterated upon to ensure a more succinct and efficient layout was created for my system allowing the Administrator to focus solely on the tasks assigned to them without facing a cluttered interface in the future, thus allowing tasks to be completed with greater efficiency. As a result of this action I was able to create a clear user hierarchy within the matchmaking system and also aid the usability of the system overall when implemented.

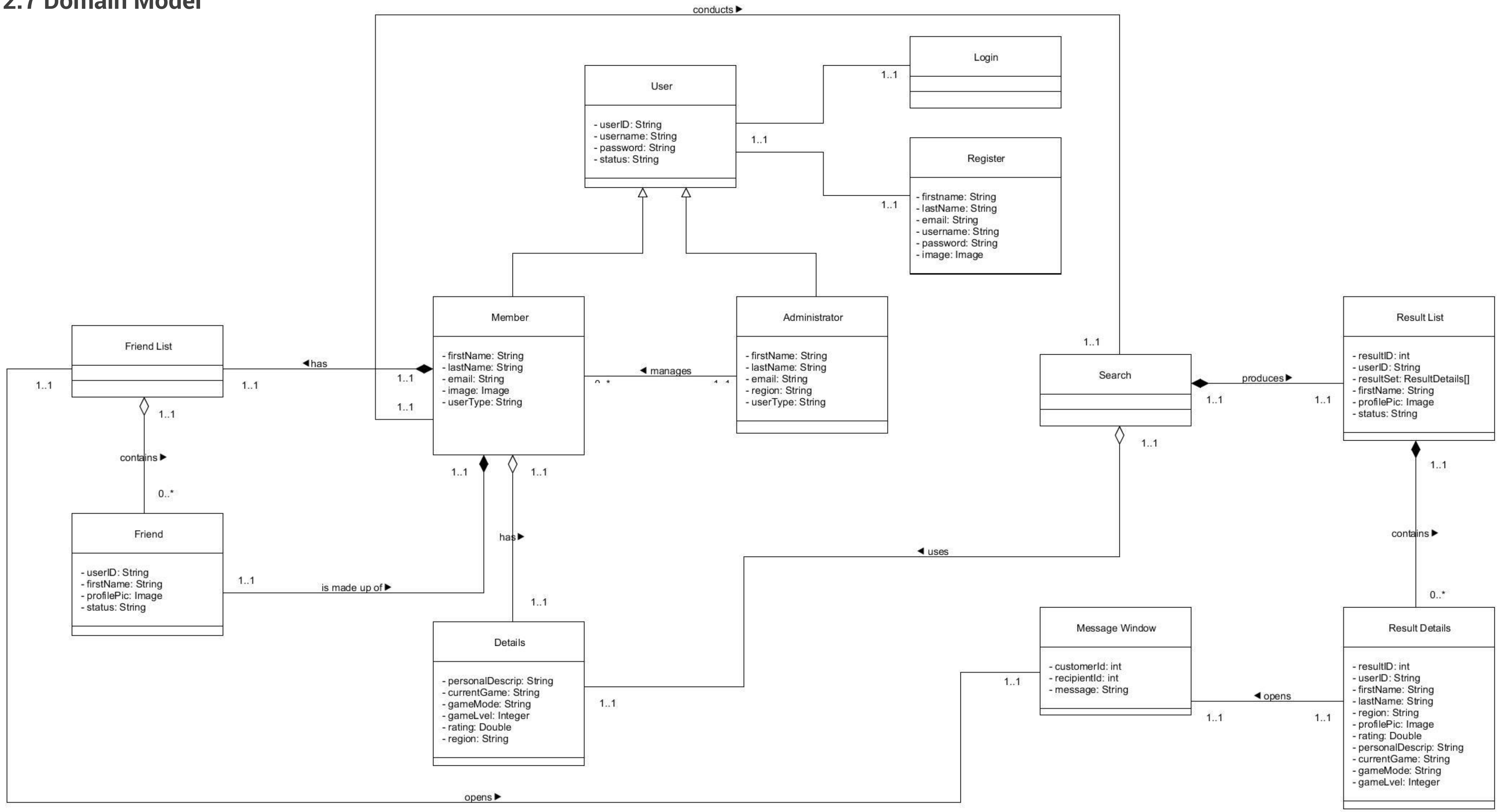
The inclusion of include relationships have been used to indicate mandatory steps, most noticeably shown between by the use case “Add Profile Details” and “Start Matchmaking Search” since a matchmaking search cannot be executed without first adding the users details which is used to power the search as such this relationship has been used to enforce this functionality. Furthermore, these include relationships have been used to elicit pre-conditions within the use case descriptions, in terms of this example a post-condition would be “The user must have added their details to their profile” present within the “Start Matchmaking Search” use case description (present in Appendix D), by analysing the include relationship I was able to extract that this was needed to follow the ideal flow and continue to the next use cases.

Within the use case diagrams a common occurrence is the use of the extend relationships between various use cases which is used to indicate optional behaviour extending the use case, an example of this within my use case diagram is “View Matchmaking Results” as shown in Figure. 2 which contains a number of extend relationship to show optional functionality. A further example would be the messaging use case extending both “Send Message” and “Receive Message” which are also both optional steps within the system as the user is not required to perform these actions. However, unlike the use cases “User Login” and “User Registration” which are treated as two separate use cases within the use case descriptions, the extended use cases within “View Matchmaking Results” were treated as “Alternative Flows” within the use case description as they all are features that will be available when completing the single task and are not entirely separate from one another like login and register as such they all work towards a common success post-condition and have been grouped for this very reason.

Lastly, the construction of use cases lead “Post-Conditions” being revealed, thus revealing what the result of running use cases would be, an example derived from my use case diagram as shown in Figure. 2 is the “Viewing Matchmaking Results” I noticed that the use case could not function without the “Start Matchmaking” use case being run before as such I included this as a pre-condition within “View Matchmaking Results” as it would not function without this being done.

This clearly indicates how the construction of both the use case diagrams and description aided me in further understanding the systems functionality and what is needed for these functions to work.

2.7 Domain Model



Use Case Elements	Classes Involved	Cross Reference/ Description
User Login, User Register	User, Login, Register	The use cases “User Login” and “User Register” have been reflected within the domain model by the class’ named “Login” and “Register” furthermore their association with the user’s class indicate that a user may both login and register into the system.
Administrator and Member Actors	User, Administrator, Member	The two separate actors (Administrator and Member) shown in the use case diagram (Figure 1) have been reflected within the domain model by creating two separate class’ each appropriately named “Administrator” and “Member”. In addition, since they are both users they have inherited from the superclass “User” (indicated by the non-filled solid arrow) and contain a “userType” attribute which is mentioned with UC-001 to be distinguished from one another.
Mange Profile Details, Add Profile Details, Update Profile Details	User, Administrator, Member	A member should be able to add and update their profile details as indicated by the use cases extending “Manage Profile Details”. This has been reflected within the domain model through the inclusion of the class “Details”. The fact that use case description (UC-003) states the details should include: “current game, description, level, gaming mode and region”, this has been reflected through the attribute that the “Description” class holds. Lastly, the multiplicity on the association between the “Member” and “Details” class (one to one) indicates a user can only have a single set of profile details.

Start Matchmaking Search	Member, Search, Details	As presented within the use case "Start Matchmaking Search" a member will able to conduct a matchmaking search, consequently, this use case was used to derive the "Search" class within the domain model. This class holds "1..1" to "1..1" multiplicity and the association ("uses") with the "Details" class indicates that fact as the matchmaking search "will be using stored profile details as parameters for the search" (UC-004), furthermore, this class' one to one association with the "Member" class denotes the fact that only one user can run the search and the fact that a member cannot concurrently run multiple searches.
View Matchmaking Results, View Result Details	Result List, Result Details	The use case "View Matchmaking Results" has been reflected within the domain model through the class "Result List", it is stated within the use case description that this will contain "rating, first name and image" this has been achieved from the inclusion of the attributes of this class. The one to "0…* relationship this class holds with "Result Details", indicates that a single results list will be made up on a number of individual results indicated by the class. Finally, the "Result Details" table also indicates the functionality of being able to "View Result Details, these details contains more in depth information by the user as shown through the various attributes present within the "Result Details" class.
Open Messenger	Message Window	The use case of "Open Messenger" has been reflected via the class of "Message Window" within the domain model its association ("opens") with the "Group List" and "Result Details" class' indicate the

		fact that the message window can be opened from within the details or friends list as indicated within the alternative flows of UC-005 and UC-006. Furthermore, the fact that it can receive and send messages with another member has been identified by the attributes it holds.
View Friend List	Friend List, Friend	The "View Group List" use case (present within Figure 1 and UC-009) was used to derive the class "Friend List", this class will contain a collection of friends stored within the list. Since the friend's list will be compromised of a collection of member/friends, this collection has been represented through the inclusion of the "Friend" class and its association with its parent class "Friend List" with it holding a multiplicity of "1..1" to "0..*" to indicate a single friend list can contain a range of friends. Furthermore, the aggregate relationship utilised indicates without the child ("Friend") the "Friend List" class would remain empty but would still exist.
Manage Members	Administrator, Member	The association between the Administrator and Member classes indicates that an Administrator has a "1..1" to "0..* multiplicity with the Member showing that the Administrator is able to manage the users as indicated within the use case diagram by the use case "Manage Users". This is further enhance by the naming of the association being "manages", thus showing the relevant use case been used to correctly derive this class.

During the implementation conducted at a later stage it became apparent that the domain model present in the figure above would not be able to satisfy the needs for the system. Therefore, a number of iterations were made to the structure of the classes through the adoption of a number of extra classes in addition to a model view controller methodology being utilised throughout the system.

2.8. Functional and Non-Functional Requirements

ID	Statement	Priority
FR1	Members shall be able to search for matching players based on parameters defined by them.	Essential
FR2	The member shall be able to enter their personal profile details into the matchmaking system	Essential
FR3	The matchmaking system shall allow users to edit and manage their entered profile details.	Essential
FR4	The system shall conduct and display the results of a matchmaking search by showing potential partners based on their completion status, location and user preferences.	Essential
FR5	Members shall be able to login to the system using a username and password	Essential
FR6	New users shall be able to create login and profile details to gain access to the matchmaking system.	Essential
FR7	The matchmaking system should allow the member to further filter the results by: game and tags.	Desirable
FR8	The matchmaking system should allow members to add users from the matchmaking results to a matchmaking group.	Desirable
FR9	The matchmaking system should allow the members to browse through their groups.	Desirable

FR10	The system should allow members to manage invite requests by accepting or declining.	Desirable
FR11	Members should be able to message other users within a group using a text based messaging client.	Luxury
FR12	A member should be able to cancel a pending invite for a matchmaking request.	Luxury
FR13	The matchmaking system make use automated search algorithm to match the members with suitable partners using their profile details order by closest matching to farthest.	Luxury
FR14	The matchmaking system should optionally enable members to rate other members.	Luxury
FR15	The matchmaking system should differentiate between user types (Administrator or Member).	Luxury
FR16	The matchmaking system should allow Administrators to manage member by being able to ban and delete them.	Luxury
NFR1	The matchmaking system shall make use of the secure HTTPS protocol to ensure security of the users' details.	Desirable
NFR2	The matchmaking system shall be able to run multiple activities at the same without affecting the performance of other tasks (e.g. search and messaging should be able to run at the same time)	Essential
NFR3	The matchmaking system should be accessible 95% of time.	Desirable

Table 3: Functional and Non-Functional Requirements

Chapter 3: Design

3.1. Introduction

The stage following the requirements gathering process is the design phase, this chapter will cover the design of: data storage and database along with wireframes and user flow. This chapter will also justify various design choices in terms of usability when considering the usability framework discussed in Appendix F along with Jakob Nielsen's (1999) principles as usability is a key component within my application and heavily dictates its success as such these principles will be used to judge the systems effectiveness during this phase and the testing phase.

3.2. Entity Relationship Diagram

The data for my implementation will be stored within a SQL Database accessed using the PHP content-management system 'PhpMyAdmin' in order to make interfacing with the database an easier task. A number of tables, rows and relationships have been defined in order to implement the functionality outlined within the requirements. Consequently as shown below in Figure. 5 is an entity relation diagram that has been created to visually depict the relationships between the identified tables, these entities identify the data stored, attributes and relationships.

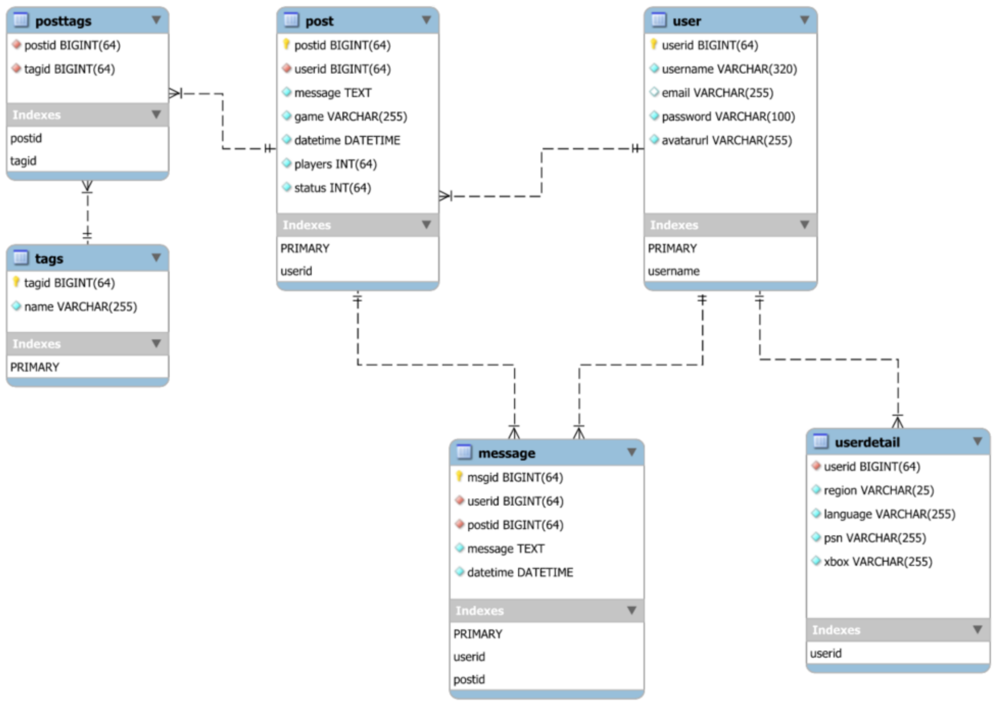


Figure 5: Entity Relationship Diagram

“Members shall be able to login to the system using a username and password” (FR5) is a requirement of the system, as such the system should be secured by both a username and password to reject unauthorized access. This requirement has been achieved through the implementation of the “User” table present within Figure 5, this tables contains a number of columns which enable the required functionality. The ‘User’ table contains an auto-incrementing primary key present within the column named ‘userid’, this primary makes use of the datatype BIGINT to support its purpose of uniquely identifying each user present within the system and also enables the database to scale due to the higher limit provided by the data type, therefore resulting in the elimination of multiple instances of the same user existing within the table. The existence of the primary key “userid” ensures the integrity of the data as it ensures when the details for a particular user are edited or added only the intended row is being targeted, thus avoiding cases where a single user possess multiple sets of credentials or their details being altered by mistake.

Due to the fact this previously mentioned functionality relies on a username and password to authenticate users, the columns “username” and “password” (respectively) have been included within “User” table. These fields have been assigned the datatype of varchar allowing the user to have the freedom of choosing between a mix between both integers and character within both their username and passwords. The length of the “username” field has been limited to 320 (as shown through varchar(320), the decision behind implementing this data type and length is enhanced by the fact that a combination of both numbers makes the username less predictable and harder to guess thus increasing its effectiveness from a security standpoint. On the other hand, the password column has been assigned a length of 100 (“varchar(100)”) to support the hashed password that will be stored within this field. Whilst a simplistic system would store the passwords as plain text, this approach will be avoided within my system to sidestep a number of security flaws, as a result, a secure hash algorithm (“crypt”) will be utilised on the data that will be within the “password” fields by assigning them a sequence of bytes and the length of the field matches the output hash string. Furthermore, the data type of varchar will be used to support the mix of characters (both numbers and letters) that are created as a result of hashing the password.

The existence of profile details is something that is present within a number of requirements as they allow the user to further personalize the system. Therefore, in order to implement the functionalities the table “UserDetail” was created. This table has been tasked with storing the users’ location, language, and xbox and psn gamer tags; achieved through the implementation of the columns ‘region’, ‘language’, ‘xbox’ and ‘psn’ columns respectively. These columns have all been assigned the datatype of varchar due to the fact they will all contain strings. In addition, they have all been assigned a maximum length of 255 due to the fact the contained values may vary and the addition of these details aid in future expansions.

A functional requirement (FR4) states the “system shall conduct and display the results of a matchmaking search by showing potential partners based on their completion status, location and user preferences”, this requirement has been satisfied through the creation of the “post” table which stores the matchmaking requests so they can be retrieved at a later time. This table identifies the player making the request through the one to many relationship it maintains with the “User” table, this has been achieved with the “post” table containing the foreign key “userid” which is also present within as a primary key within the “User” table helping to solidify this relationship. A one to “0..*” relationship has been used since each user within the “Users” table can at minimum manage have no posts. The column ‘message’ has also been included and given a datatype of text to accommodate a message entered by a player when their request is made. Furthermore, to identify if a post has been successfully completed a column named status has been added and will contain a numerical value of either 0 or 1 (depicting complete or incomplete respectively), hence a datatype of integer to support this data. Lastly, the columns of ‘game’ have been used to identify the users current request and has been given the datatype of varchar with a max length of 255 due to the fact that games name can vary I length and can contain a mix between both integer and characters. Lastly, a ‘datetime’ column has been added which will contain the date a request was created.

In order to support the requirement stating the “user should be able to further filter the results” a table named “tags” has been added to provide further context for posts/requests. Firstly, this table includes an auto-incrementing integer named ‘tagid’ acting as a primary key to uniquely identify each tag. The column ‘tagname’ also has been created with a varchar with a max length of 255 to accommodate the name of a tag and its varying lengths, ttable maintains a one to 0..* relationship with the table ‘posttag’ table which stores two foreign keys creating a link between the tag and post table. The columns that make this relation possible are the ‘postid’ column which is a foreign key which maps to the ‘postid’ primary key also present within the ‘post’ table as such a one to 0…* relationship is formed between the ‘post’ table and the ‘posttag’ table and the ‘tagid’ is also present within both tables creating a primary and foreign key relationship. As a result, a post can contain multiple tags which can be used to further filter the results once implemented.

The tables and columns implemented serve to satisfy the various functional requirements outlined, therefore I should be able to use the data stored in the database to create the proposed application allowing users to conduct and control their matchmaking process.

3.3. Wireframes/User Flow Diagram

As shown below a number of prototypes and wireframes were created, these were then combined into a user-flow diagram allowing me to further discern the overall structure of the system along with how the intended users' would navigate the system. The fact that the structure was considered resulted in many aspects of usability being applied in a more thorough manner, thus improving the usability of the application and greatly contributing to the overall solution, achieved by incorporating Nielsen's Heuristics and Usability Principles.

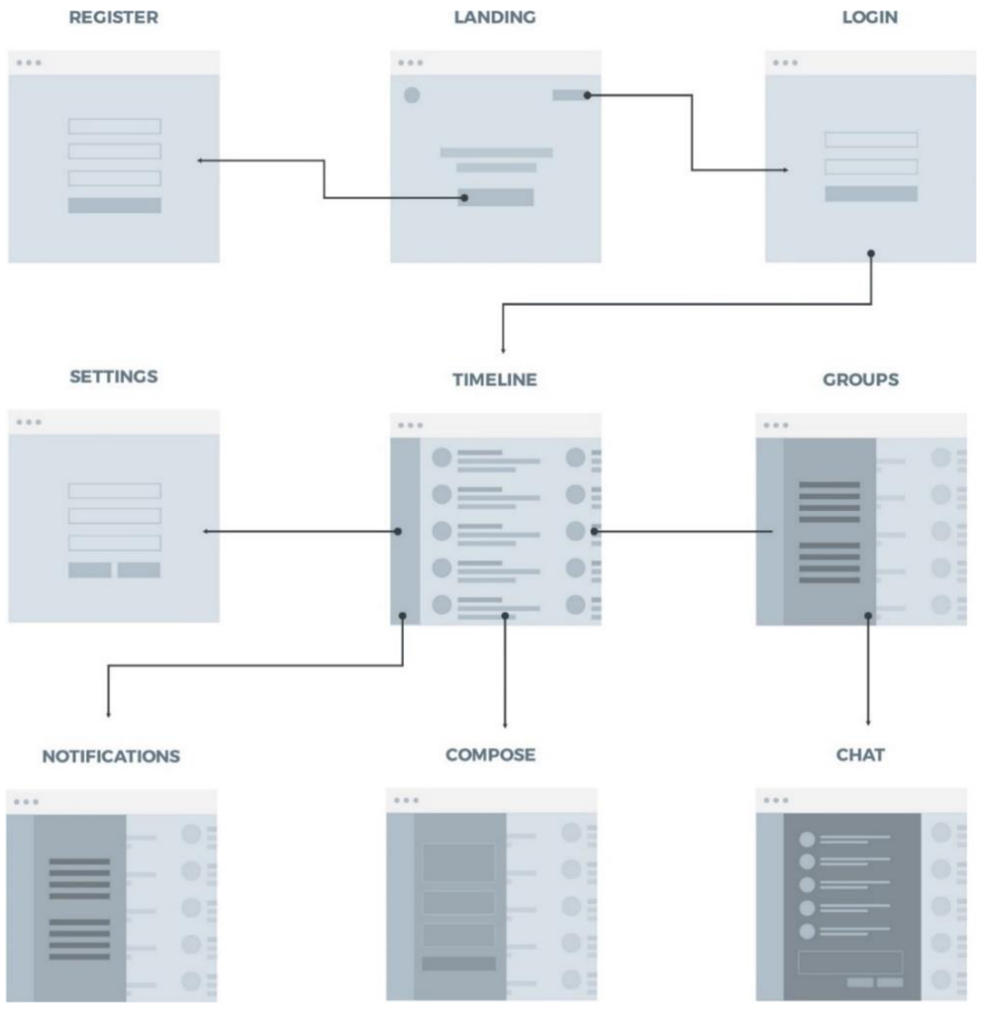


Figure 6: Wieframe/Userflow Diagram

The user-flow diagram above has been constructed to outline the core functionality required of the proposed matchmaking system, the screens/views housing the functionality and their relationship to one another have also been depicted.

The first view a prospect or member is met with is the ‘landing’ view, this page has been designed to convince these users to either re-enter of register to the system; a number of usability principles have been applied to ensure the user is persuaded to use the application by selecting one of the two options available which would in turn load their respective views (of ‘Register’ and ‘Login’). Therefore, two call-to-actions buttons have been used to satisfy the requirements of FR5 and FR6 which handle the process of signing in or registering to the system. Nielsen (2016) states that a system must promote “Recognition rather than recall” to adhere to this principle the call-to-actions present throughout the web application will be designed to minimize the users memory load by making themselves clearly visible (through the use of contrasting colours and larger sizes) allowing users to instantly distinguish how they can perform these tasks and thus fulfil the requirements. Furthermore, the labels accompanying these buttons also depict the functionalities e.g. “get started now” and “sign in” along with the slogan “find a team” create emphasis towards these requirements. A deliberate choice of wording has been used to incorporate the usability principle which states “Match between system and real world” and “system should speak the users’ language” (Nielsen, 2016) allowing users to make decisions and use the system with greater speed and efficiency, thus aiding the usability of the application as a whole. As a result, this design will aim to promote the completion of the requirements of the system clearly and coherently by applying the mentioned usability principles.

Both the ‘login’ and ‘register’ views within Figure. 6 contain input fields and buttons connoting they will require the user to input data into the system, therefore a number of design choices have been made to support this process. The fact the entered data will be crucial to systems operation means error prevention and the usability aspects attached should be considered. It is apparent that the forms within these views along with other forms present within the system should employ error prevention techniques as a “design which prevents a problem from occurring in the first place” (Nielsen, 2016) is an effective design in terms of usability. Therefore, during the design phase it was decided that per-field validation would be used to counteract possible errors before the user has a chance to submit them, this decision meant that users also needed to be prompted on how to correct issues as stated also previously within Appendix D’s use case descriptions. This decision correlates to the principle stating that a system should be designed to “Help users recognize, diagnose, and recover from errors” (Nielsen, 2016) as such error messages shall appear above resulting fields to aid error prevention and usability. Lastly, when expressing errors it is stated that “Error messages should be expressed in plain language (no codes), precisely indicate the problem” (Nielson, 2016) based on this fact a design choice was to use simplistic language when conveying errors that are clear and coherent.

Nngroup.com (2016) define usability as a “quality attribute that assesses how easy user interfaces are to use” which are judged based on the following quality attributes: learnability, memorability, satisfaction and efficiency. They also state that “usability is a necessary conditional for survival” as such these quality attributes have been applied to the core design of my proposed system to ensure the good usability and customer satisfaction stems from my system.

Once logged into the system the user will be able to access the main functionality of the system from the ‘timeline’ view (as depicted in Figure. 6), the user will be able to navigate to other views to complete functionality such as: managing invites, sending messages, viewing groups, managing profile details and conducting matchmaking searches. The various views which make up this application will be accessed via a persistent navigation to provide a fluid experience.

In accordance with Nielsen’s Heuristics it is stated that a user should be aware of their location and status within the application at all times by answering the questions “Where am I? Where have I been? Where can I go?” (Nielsen, 1999) to answer the questions posed by this principle it is stated by Galitz (2007) that a “navigation must be consistent throughout a graphical system or website” and this has been considered greatly throughout my design process to aid the usability. A depicted within Figure 6 a persistent navigation will be to the left of the content and maintain the same aesthetic design across all pages ensuring users are able to access functions with efficiency (minimal clicks) and know how to act, this navigation has been added to answer the questions posed above and aid the usability of the system by ensuring the efficiency and memorability for the users actions. A further design choice when considering the navigation was the use of graphical icons to depict functionality, the rationale behind this decision was to add to the usability by increasing the ease of use of the system and the learnability. In regards to usability the Nngroup.com, (2016) refer to icons as “a visual representation of an object, action, or idea”, the fact that I have used icons that are considered the norm poses a number of usability advantages, such as: reducing the need to translate, improve the aesthetics whilst also clearly depicting the functionality of the system, thus creating emphasis on the usability principle of memorability.

In conclusion, during the design phase a number of usability principles were applied to ensure users was able to complete the requirements of the system with the upmost efficiency and ease. The applied usability principles will also be evaluated during the testing phase to judge the effectiveness of the implementation.

Chapter 4: Implementation

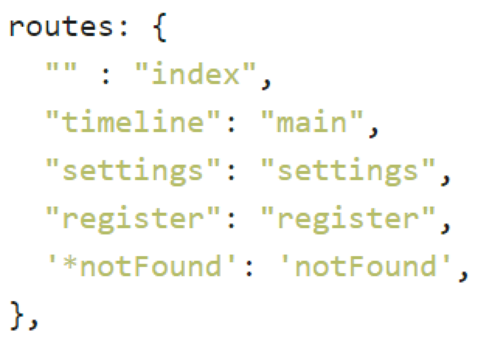
4.1. Introduction

This section covers the specifics of the implementation phase of the system, snippets of code are provided to support discussion on topics such as: code critical to the system, innovative ways of tackling unforeseen issues and how the current implementation rectifies these issues.

The implementation is a single page application making use of the front-end framework Backbone.js to provide an MVC structure and the features required of a SPA including Models and Collections to manage the data and views which contain the user interface. This application communicates with a Rest API built using PHP to retrieve and send data. The chosen technologies have been discussed and their use has been justified within Appendix E with the results of the implementation being shown in Appendix M.

4.2. Authentication

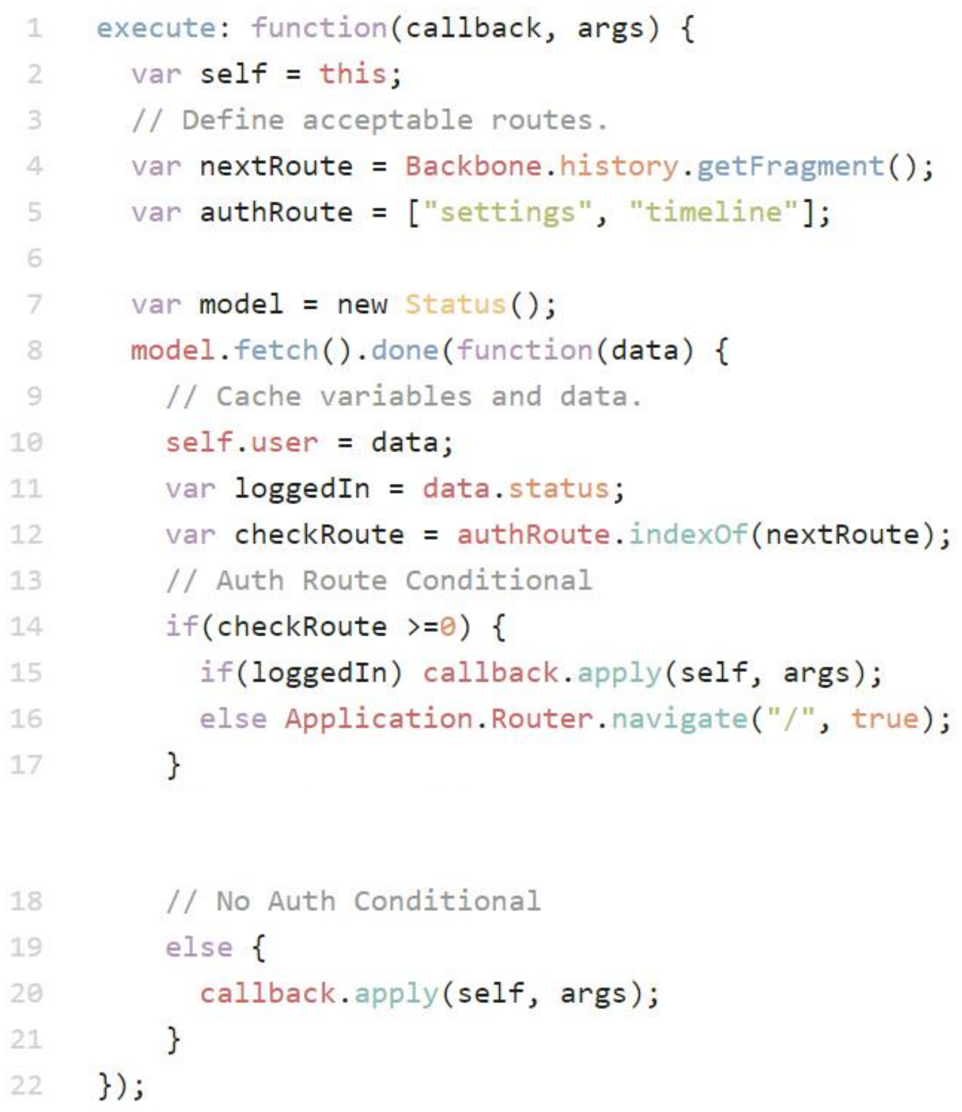
As stated within FR5 the system should be secured by a username and password, however due to the single page and asynchronous nature of the system the process of authenticating a user and acting accordingly became a more difficult task than initially anticipated when considering my prior experience. A typical PHP application incorporating authentication is tightly coupled with the backend service and has files containing pages, meaning if an user attempts to access a page via a URL the browser would determine their authorization status by first checking if a ‘session’ exists and act accordingly based on that (redirect or page load). The fact that my application made use of a REST API meant the front-end was decoupled from the backend and the data was received via AJAX using the API GET requests meaning that sessions did not exist in the same location as the application, furthermore, the system having a single page made the process of authentication a further complex issue.



```
routes: {
  "" : "index",
  "timeline": "main",
  "settings": "settings",
  "register": "register",
  '*notFound': 'notFound',
},
```

Figure 7: Authentication Code Example 1

To combat the lack of multi-page functionality ‘routing’ was used in which a single page application is able to map segments of the URL proceeding a hash (#) to a specific function which in turn would load a new view (containing the UI), thus replacing the content on the page with new content. As a result, I was still able to create an interface in which the various views could be navigated in a similar manner to multiple pages, but maintain the speed and efficiency offered by a SPA. An example as shown in Figure 7 above would be ‘/#/timeline/’ calling the “main” function which in turn would load the view containing the logged in user interface for a particular user.



```
execute: function(callback, args) {
  var self = this;
  // Define acceptable routes.
  var nextRoute = Backbone.history.getFragment();
  var authRoute = ["settings", "timeline"];

  var model = new Status();
  model.fetch().done(function(data) {
    // Cache variables and data.
    self.user = data;
    var loggedIn = data.status;
    var checkRoute = authRoute.indexOf(nextRoute);
    // Auth Route Conditional
    if(checkRoute >=0) {
      if(loggedIn) callback.apply(self, args);
      else Application.Router.navigate("/", true);
    }


    // No Auth Conditional
    else {
      callback.apply(self, args);
    }
});
```

Figure 8: Authentication Code Example 2

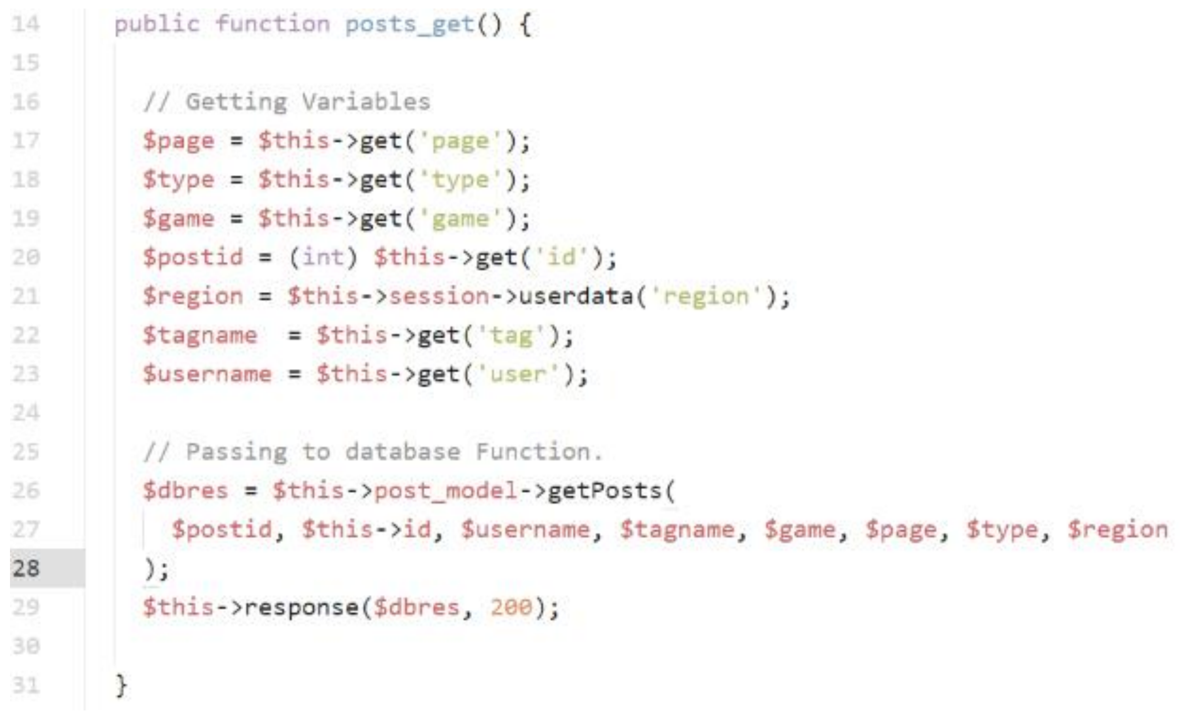
As shown by Figure 8 a function named ‘execute’ was implemented to handle authentication, this function was bound to the loading state of each route, as such when any route was called or initialized before the UI could be loaded the logic within the ‘execute’ function would be run. This function whenever called first initializes the model needed to communicate with authentication restful route in this case the ‘auth’ route accessed via the restful route ‘/auth/’, once fetched the status would then be returned with the value either being a JavaScript object containing the value ‘{status: false}’ if the user is not logged in and their details if they are. This data is then used within conditionals to decide whether to redirect the use to a particular route (e.g. the landing page (“/” if not logged in). However if they are logged in the routing would continue as usual and the function would be passed to the original call-back loading the correct code, views as a result.

However, an issue that arose was that this function would fire regardless of route and that fact that were two routes that did not require the user to be authenticated namely the ‘landing’, ‘login’ and ‘register’ meant extra logic (present on lines 4, 5 and 14 of Figure 8) needed to be added to stop the redirection of these routes to the landing page. I achieved this by first creating an array stored in a variable named ‘authRoute’ which contained the names of the routes that needed to be secured by the system, these included: ‘settings’ and ‘timeline’. Next, I needed a way to identify the current route for comparison this was achieved by using the frameworks ability to retrieve the value preceding the hash achieved using the function ‘Backbone.history.getFragment’ which was then stored in a variable named ‘nextRoute’. Lastly I created a conditional surrounding the previous login condition (present on line 14) which used JavaScript’s ‘.indexOf’ method to check if the ‘nextRoute’ is stored within ‘authRoute’. If the negative integer was returned (-1) the else statement would be triggered calling a call-back function mapping to the original function routing as normal. However if a positive number was returned the conditional would amount to true and then either load a new view or redirect the user based on their login/authentication status.

4.3. Retrieving and Filtering Requests

The essential requirement stating: “the matchmaking system should allow the member to further filter the results by: game and tags” meant the API needed to be flexible enough to handle different requests for different data and combinations of data which would then be returned to the front-end framework to be processed and displayed.

The excerpts present in Figure. 9 and 10 below are critical to this functionality as they hold the logic of not only retrieving data pertaining to a players request, but also filtering the day a by a number of parameters such as: the game, date, location, user, tag and page. Due to the dynamic nature of this request and the fact a large amount of information would be computed, producing this code proved to be a difficult task when considering the efficiency, speed and flexibility of the computation. Building this functional requirement required the use of PHP along with the framework CodeIgniter which provided a number of helper methods and enforced a MVC structure needed for the backend rest API.



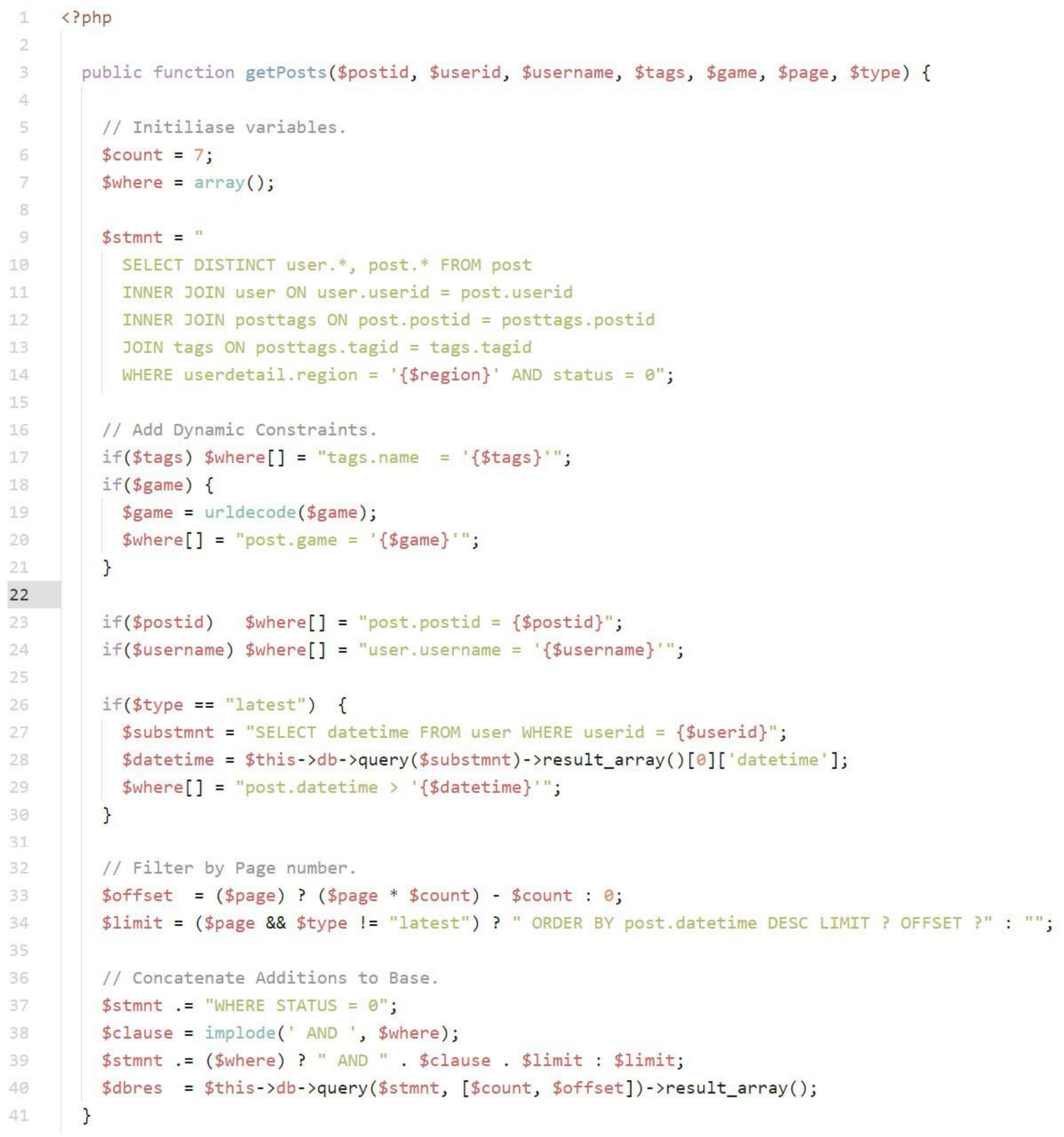
```php
public function posts_get() {

  // Getting Variables
  $page = $this->get('page');
  $type = $this->get('type');
  $game = $this->get('game');
  $postid = (int) $this->get('id');
  $region = $this->session->userdata('region');
  $tagname  = $this->get('tag');
  $username = $this->get('user');

  // Passing to database Function.
  $dbres = $this->post_model->getPosts(
    $postid, $this->id, $username, $tagname, $game, $page, $type, $region
  );
  $this->response($dbres, 200);

}
```

The above code as shown in Figure. 9 depicts the controller of the REST API which is called when the front-end framework performs a GET request for the post route. This function is tasked with retrieving the various parameters which will be used to filter the results. This is achieved by the parameters being contained in the URL key value pairs allowing them to easily be identified and stored in their designated variables for later use, an example of this within my application would be retrieving posts for a certain game with a particular tag through the use of the following url: “api/post/game/Pokemon/tags/Mic/”, this would result in the value ‘Pokemon’ being stored in the $game variable and the values ‘Mic’ being stored in the $tagname variable and these variables and any ignored containing the value of NULL. These values were then passed to the model function named getPosts() within the post_model which retrieved and returned the corresponding data

Figure 9: Retrieving Requests Code Example 1

from the database using the various passed parameters to filter the results. Once these parameters had been passed they were used to filter the data returned to the controller and consequently returned to the application, the complexity of efficiently retrieving data based on user preferences resulted in a number of issues arising.

Firstly, whilst my framework of choice (CodeIgniter) provided abstracted database classes offering simplistic syntax, it had a major drawback as when retrieving data from the database a major factor leading to inefficiency was the fact the query was passed to a separate class before it be could be executed, thus adding a needless layer resulting in the execution of the query being slowed down and impacting the computation and speed of the system overall. Furthermore, the database class' simplistic syntax made producing complex queries using the parameters passed a difficult task thus impacting the functions flexibility. However, the lack of flexibility in addition to inefficiency was resolved by the code below (Figure. 10) by bypassing the frameworks innate database class I was able to create my own method of creating dynamic queries and that fact that this method did not use an abstracted class lead to a speed improvement.



```php
1  <?php
2
3    public function getPosts($postid, $userid, $username, $tags, $game, $page, $type) {
4
5      // Initiliase variables.
6      $count = 7;
7      $where = array();
8
9      $stmnt = "
10       SELECT DISTINCT user.*, post.* FROM post
11       INNER JOIN user ON user.userid = post.userid
12       INNER JOIN posttags ON post.postid = posttags.postid
13       JOIN tags ON posttags.tagid = tags.tagid
14       WHERE userdetail.region = '{$region}' AND status = 0";
15
16     // Add Dynamic Constraints.
17     if($tags) $where[] = "tags.name  = '{$tags}'";
18     if($game) {
19       $game = urldecode($game);
20       $where[] = "post.game = '{$game}'";
21     }
22
23     if($postid)   $where[] = "post.postid = {$postid}";
24     if($username) $where[] = "user.username = '{$username}'";
25
26     if($type == "latest")  {
27       $substmnt = "SELECT datetime FROM user WHERE userid = {$userid}";
28       $datetime = $this->db->query($substmnt)->result_array()[0]['datetime'];
29       $where[] = "post.datetime > '{$datetime}'";
30     }
31
32     // Filter by Page number.
33     $offset  = ($page) ? ($page * $count) - $count : 0;
34     $limit = ($page && $type != "latest") ? " ORDER BY post.datetime DESC LIMIT ? OFFSET ?" : "";
35
36     // Concatenate Additions to Base.
37     $stmnt .= "WHERE STATUS = 0";
38     $clause = implode(' AND ', $where);
39     $stmnt .= ($where) ? " AND " . $clause . $limit : $limit;
40     $dbres  = $this->db->query($stmnt, [$count, $offset])->result_array();
41   }
```

Figure 10: Retrieveing Request Code Example 2

As shown above this was achieved by first specifying a base SQL query string containing the table dependencies identified by using the ‘FROM’ and ‘INNER JOIN’ using the foreign key relationships within the ‘ON’ clause to retrieve the correct information, this has been shown on lines 9-13 of Figure. 10, this was then stored in a variable named $stmnt to be dynamically built upon using the parameters passed to further constrain the request to the database through a number of clauses (where, and).

The dynamic additions of constraints to the existing base query stored within $stmnt was achieved using an array stored within a variable named $where which contained the additional clauses that would further filter the data, the clauses to be added were decided using conditionals. This was achieved by checking if the parameter passed resolved to null and if it did not (meaning that this parameter was requested) the value stored within the parameter would be concatenated to the additional where clauses and the additional clauses would be added as a new index in the $where array. As a result of using this method I was able to build an array of additional constraints for each passed parameter that is to be added to the base SQL query stored within $stmn. Once the appropriate clauses for each parameter had been added to the array, it was then imploded into a string with the delimiter being "AND" as such each index containing a string would be separated by an AND forming an properly structured where clause e.g. "WHERE STATUS = '0' AND USER='munaibh' AND tag='Mic'" as shown this output SQL string meets the specifications if the requesting url was '/api/post/user/munaibh/tag/mic'. This addition was then concatenated to the existing base SQL query stored within $stmnt resulting in a highly specific query that would retrieve only the data needed forum the database and return it to the front-end framework with speed and efficiency.

Lastly, when considering the data returned could contain a large amount of items despite being filtered caused further issues when considering loading time as these would have be to be returned and then added into the user interface, thus impacting loading times greatly. Therefore, in a similar manner to the previous parameters a page number was passed via the url e.g. "?page=2" this page number was decided by the front-end (which would be incremented each time a user reached the end of the available requests) this would then be used to contain the returned results by creating an offset and limit (shown in line 34) based on the number of items allowed per request as defined by the $count variable present within Figure. 10 as such I was able to limit the amount of results available per request thus further increasing loading times and efficiency of the system.

4.4. Database Restructure

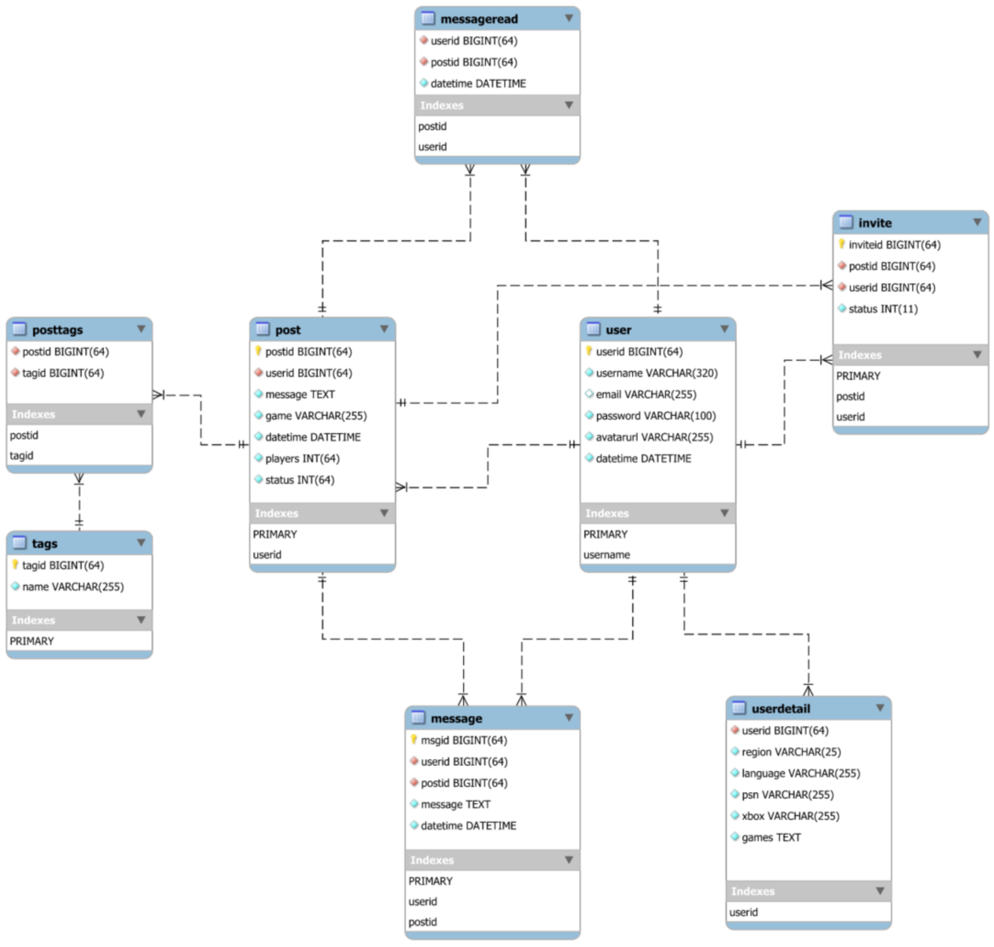


Figure 11: Revised Entity Relationship Diagram

During the implementation phase it became apparent that my database design would not be able to support my requirements fully, therefore alterations were made to accommodate the various functionalities of the system. The databases' size increased with an additional three tables being added along with an assortment of columns, the revised ERD is shown in Figure 11 above.

The 'datetime' column is an additional column added to the "User" table with a datetime data type, after great difficulty retrieving the latest requests for users who were logged in through the use of a live feed, this column was added to resolve this issue as it allowed the system to query the database to retrieve all posts at or after the date and time outlined within this field, thus aiding in the implementation of one of the core functionalities of displaying matchmaking requests.

The column named “games” was added to the “UserDetail” table to allow users’ to personalise their experience within the application to optionally have a filtered view containing only posts matching the game names entered. Furthermore, another table that was implemented was named ‘messageread’, this table greatly supported the messaging functionality embedded within the system as it allowed the system to keep track of the last messages viewed by the user for a particular post/group, this was achieved through the use of the foreign key constraint identify the group of messages and a ‘datetime’ field containing the last time the user had access/read a message, this was strengthened by the parent child relationship between the tables helped emphasis this relationship showing the data was tied to the correct group and user.

The last addition was a table named ‘invite’ this table supported the functionality of sending invites to players and the functionality of accepting or declining invites. This table has a number of relationships with the other tables within the database; a one to many relationship was held between the post and invite table with once post being able to have multiple invitations, this relationship was created between the common column named ‘postid; with it being the primary key in the ‘post’ table and the foreign key in the ‘invite’ table. A similar relationship is held between the ‘user’ and ‘invite’ table with the column ‘userid’ being a primary key in the ‘user’ table it being the foreign key in the ‘invite’ table allowing the user requesting an invite to be identified and as such their various details can be accessed. Lastly, the column status which contains a numerical status of the invite status which can then be relayed to the user when the status is retrieved as such this fields within this column are an integer with a max length of 11. The numbers within this column that represent the states of invites are 0 being undecided, 1 being accepted and 2 being declined.

As we can see from the diagram a number of additions have been made to the structure of the database, this was identified during the implementation process as a number of features could not be implemented and as such these alterations were made to resolve these issues.

Chapter 5: Testing

5.1. Introduction

This chapter covers the various testing procedures carried out during the course of the project in order to ensure the implementation met the essential requirements when regarding functionality and usability. As discussed above, the iterative methodology was adopted due to the fact that core requirements of the system were clearly defined and these details were able to be altered over the course of the project, consequently this choice influenced the testing methods used throughout the project. This approach also allowed a requirement to be implemented and tested before the next requirement was implemented.

Unit testing was a pivotal testing method used within my project, it featured white box testing using pre-planned test cases which were completed once an implementation for a specific requirement was complete. Once complete, acceptance using the defined requirements and Larisa Thomason’s (2004) usability checklist (present within the Appendix G) as base were conducted by 9 potential end-users, thus allowing me to gauge the effectiveness of the application overall.

5.2. White-box testing

The result of choosing an iterative based development resulted in the project being implemented in segments with a requirement being created, tested and revised before the next requirement was tackled. As a result, I was able to ensure that a requirement was fully functional before moving on to the next. The testing method to achieve this desired effect was created various test cases for each module outlining the test data, scenario and the steps required to perform this test case, the purpose of this method was to verify the provided functionality against a specific requirement. As such white-box testing was used, this was efficient in identifying errors and helping in optimizing the code. This approach of test driven development aided the final product greatly as it ensured the core requirements functioned correctly.

As shown in Appendix I these test cases were filled out after each modules completion by using the planned data input and following the planned step, whilst recording the actual outcome. The white-box testing of most modules whilst most positives identified a number issues which resulted in some requirements not being fully functional, these were identified by the expected result not matching the actual result, consequently these issues were logged and defects and then resolved and retested. Some of the identified issues included the system not being properly secured by username and password, the system not handling the messaging functionality correctly (more details are outlined within the appendices).

This testing method was useful in identifying and rectifying the mistakes. The full test case document is shown in Appendix I which outlines the various scenarios white-box testing was used under and the results of those test cases.

5.3. User Acceptance/Black-Box Testing

Once the all the pre-planned test cases were complete (i.e. the essential functionality of the system was implemented), the system was then handed to a series potential end-users who completed acceptance tests based on their expectations of the systems functionality. The procedure allowed me not only to judge the completeness of the application in its intended environment, but also gauge the effectiveness in terms of usability and receive feedback on how to improve areas of my app from the audience I intended to target.

User acceptance testing was completed in the form of “Beta Testing” were a select few users were asked to use the application in its proposed environment. Whilst the core functionality was accepted a number of improvements were suggested and could be added in future iterations (this will be discussed in detail in evaluation section), some these improvements these include: the addition of an autocomplete for game entrances and greater functionality when considering messaging, such as which users are online and typing etc. When considering the aspect of usability, the beta/black box testing methods used as part of user acceptance tests make it apparent the applications core functionality was functioning and easy to become accustomed to mainly due to the “good use of icons” which depicted the various functionality. A number of usability issues were also risen these included the fact that the colour scheme seems “too vibrant” and thus resulted in “distracting users from buttons” and “the navigation bar was not easily noticeable due to the background colour”, these suggestions lead to a number of improvements being placed upon the application resulting in a much more polished prototype being produced. The results of the usability testing are discussed in detail within Appendix G.

5.4. Conclusion

When considering the aspect of usability, the testing made it apparent the applications core functionality was fairly easy to use and became much easier the more users used the website more and became more familiar with the functionality. A number of usability issues were also risen, these comments resulted in a number of changes and improvements being made to the application resulting in a much more polished prototype being produced.

Chapter 6: Evaluation

6.1. What has been achieved

The project resulted in the creation of a semi-automated matchmaking system that allows users to look for a potential partner and/or team for a particular game, post their request for all local users to see and submit invites to requests. As such the user can then view their invites and accept and decline invites to the system and interact with these users using a text based medium to arrange a game to play. The construction of this application made use of a number of requirements essential, desirable and luxury and implemented them successfully.

Throughout the course of the project a number of processes were completed, the first of these was the requirements gathering process, this stage included the analysis of the current state of the market to ensure that my solution was needed, once identified as needed, the next step was to analyse existing systems to discover major flaws and advantages that could be applied to my solution. This process allowed me to identify a number of features, such as the ability to: identify game, use a chat based text client and add tags to requests, whilst these features were initially discovered they were further built upon during the requirements process to make the system more robust. Furthermore, this process allowed me to iterative upon my initial idea and make a system better suited to the problem it was aiming to solve.

During the requirements gathering stages, I constructed both a use case diagram and description however when researching potential design during the next phase it became apparent that the design was not efficient enough when it came to usability, as such it resulted in a number of iterations when considering use case diagram. Similarly, the design phases a number of alterations were made to the design of both the system and the database in order to ensure the correct functionality could be implemented in a usable manner.

Lastly, during the implementation I implemented the various requirements of the system, whilst also filling gaps in my knowledge in parallel allowing me to efficiently learn and complete the solution resulting in the project being completed in a swift manner. The features implemented included all the planned essential and desirable requirements and the majority of the luxury requirements.

In conclusion, I believe the various stages of the project were completed as needed and as such I learnt a number of skills such as how to handle errors and setbacks as well as how to handle the creation of projects on a larger scale that I was not previously accustomed to.

6.2. Limitations

During the course of the project many limitations arose which were not foreseen or accounted for during the earlier stages of the project as outlined within Chapter 1. This was mainly due to the new technologies used within the projects and my lack experience with them.

Due to the fact it was decided that a single-page application would be created it became apparent that a REST API needed to access the data needed via a URL, when considering the fact I have basic knowledge of PHP along with minimal knowledge of creating complex REST APIs, the creation of a fully functional REST API for my application provided to be a difficult and time consuming task due to the various intricacies involved (e.g. serializing the information to the correct format and returning the correct http headers to be used by the front-end framework). As a result, the creation of this backend API resulted in less time being spent on the other sections due to its importance and my difficulty with it.

A further limitation was my lack of SQL knowledge which was needed when constructing the rest API to retrieve data from the database using the shortest and most efficient method. However my lack of practical knowledge using PHP and SQL (e.g. using joins) together within a complex database further caused a number of issues, therefore resulting in time being spent to gather the necessary knowledge and correct the various mistakes made, which had resulted in the REST API not getting the data needed from the database, thus disrupting the planned schedule further.

Lastly, in regards to my personal experience when using the JavaScript framework Backbone.js my lack of JavaScript knowledge became apparent in many instances some of the most notable problems that arose was problems when binding events, as events would either not fire when needed, fire multiple times or overlap with other elements events as such this caused a number of issues when considering the functionality and a number of extra measures and steps had to made to prevent this measure the most important of this being learning about techniques such as ‘’delegating events”.

The outlined issues along were some of the limitations present within the project and as such they slowed down the progress of the project overall so they could be resolves. These limitations resulted in some functionality (e.g. luxury requirements) not being implemented as initially planned.

6.3. Future Improvements

There are many aspects of my solution that could benefit from future enhancements, some of these enhancements include: making the system multi-platform, the integration of external APIs and the ability to automate the search further.

Firstly, the system would benefit from the integration of an external API such as the “Giant Bomb API” which would provide a retrievable and up to date list of games. This addition would be able to provide further functionality to users when making requests, such as autocompleting game entries thus ensuring players are submitting requests for their intended games, unlike the current implementation which requires players to manually check both the spelling and name of the entered game beforehand which can prove tedious and if mistaken matchmaking will not be conducted for their expected game.

The fact that my application is limited to desktop web browsers due to its complex nature makes the implementation of responsive web design a difficult task meaning its reach has become limited which is a major flaw. A future improvement that I would like to implement is making the application available on multiple platforms such as Android, iOS and Windows Phone, thus increasing its reach and making it more portable allowing for users to conduct matchmaking searches and the various other functionalities provided by the system regardless of location. I would have also liked to implement the ability to users to close groups they had created, in addition to creating online documentation for my application making it easier for novice users to get acquainted with the application.

Lastly, introducing an algorithm which matches players with greater accuracy when compared to the existing matchmaking was an luxury requirement I was unable to implement due to time constraints, therefore, this functionality is an aspect that I would like to add as a future enhancement to allow players the option of conducting more specialized matchmaking searches with even less effort thus making the application more convenient to some of the systems players.

6.4. Conclusion

Throughout the project I believe I have created a web application which exhibits the traits needed to solve the problem originally defined and has been implemented to the best of abilities when considering my skill level and time constraints. A number of techniques and skills were learnt as a consequence of this project and resulted in a number of features that would have not been implemented otherwise being included, furthermore skills in terms of time management and problem solving were also acquired. This application has been constructed to the best of my current ability and I hope to improve it in the future.

Bibliography

1. Alexander, I. and Beus-Dukic, L. (2009). *Discovering requirements*. Chichester, England: Wiley.
2. Hussain, M. (2015). *Gaming Survey.*. [online] Google Docs. Available at: https://docs.google.com/forms/d/1XMxayTTePo1RMD5BzVWZhp97twkBT7RMlKW0yyH6-lU/edit [Accessed 23 Nov. 2015].
3. Okcupid.com, (2015). *OkCupid | Support*. [online] Available at: https://www.okcupid.com/help/match-percentages [Accessed 24 Nov. 2015].
4. Evolve, (2015). *Social Gaming Platform - Evolve*. [online] Available at: https://www.evolvehq.com/welcome [Accessed 24 Nov. 2015].
5. DestinyLFG.com, (2015). *The Bungie Featured Destiny Looking for Group Site*. [online] Available at: http://www.destinylfg.com/ [Accessed 24 Nov. 2015].
6. Dumas, J. and Redish, J. (1993). *A practical guide to usability testing*. Norwood, N.J.: Ablex Pub. Corp.

7. Measuringu.com, (2015). *10 Essential Usability Metrics: MeasuringU*. [online] Available at: http://www.measuringu.com/blog/essential-metrics.php [Accessed 8 Dec. 2015].

8. Galitz, W. (2007). The essential guide to user interface design. Indianapolis, IN: Wiley Pub.

9. Nielsen, J. (1999). Designing web usability. Indianapolis, IN: New Riders.

10. Nielsen, J. (2016). Memory Recognition and Recall in User Interfaces. [online] Nngroup.com. Available at: https://www.nngroup.com/articles/recognition-and-recall/ [Accessed 26 Apr. 2016].

11. Nngroup.com. (2016). Icon Usability. [online] Available at: https://www.nngroup.com/articles/icon-usability/ [Accessed 26 Apr. 2016].

12. Nngroup.com. (2016). Usability 101: Introduction to Usability. [online] Available at: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ [Accessed 26 Apr. 2016].

13. Nngroup.com. (2016). 10 Heuristics for User Interface Design: Article by Jakob Nielsen. [online] Available at: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 26 Apr. 2016].

14. Thomason, L. (2004). *Web Site Usability Checklist*. 1st ed. [ebook] Available at: http://www.uobabylon.edu.iq/eprints/publication_4_8707_26.pdf [Accessed 27 Apr. 2016].

15. Wikipedia. (2016). *Single-page application*. [online] Available at: https://en.wikipedia.org/wiki/Single-page_application [Accessed 28 Apr. 2016].

Appendices

Appendix A: Literature Review

"Evolve" created by Evolve Labs is a software that has been created to also tackle the issue of matchmaking for gamers, "Evolve" is a LAN (Local Area Network) based matchmaking client; offering an extensive amount of features to its user base, one of which includes the ability to conduct "Universal matchmaking" (Evolve, 2015) allowing users "to search both in and out of gameplay" (Evolve, 2015) environments for other gamers to play with, this search can span multiple games that are available to the user on the machine conducting the search. According to Evolve (2015) some of the other features this application provides to its users include the ability to: “Join a game”, “Start a party”, “Chat with friends”, “Surf the web”, “Take screenshots”, “Record video” and “Broadcast with Twitch”. Whilst, this service provides a broad range of features, my solution will contain a smaller amount of features allowing the users to complete their designated task with the most efficiency, some of the features I will include are: matchmaking, the ability to chat to gamers and the ability to start a party (i.e. add to group list).

When it comes to matchmaking “Evolve” allows users to search for gamers using a strict set of search parameters, however this feature only works with games containing local area network capabilities, my solution will disregard this parameter as it will limit the results available and instead the search shall span across multiple games regardless of features within them. Furthermore, “Evolve” is not limited to searching for gamers in a single game during a search, the service can search across multiple games making it highly efficient. However, whilst “Evolve” offers matchmaking capabilities it does not rely on this feature and instead enhances it with the use of an integrated chat client, thus allowing users to chat to their fellow gamers, chatting can be achieved via two methods: peer-to-peer VoIP or text based chat. Lastly, this service provides the user with the ability to save gamers they have met via matchmaking to a friend’s list so they may immediately utilise the above features without the need to once again run a matchmaking search to find a player once again. The feature pertaining to adding users to a friend’s list is an extended feature I wish to add within my solution in the form of creating groups.

“Evolve” utilises the following languages as its base: C++, C# and Ruby, whilst the backend data is stored using PostgreSQL, thus allowing data to be retrieved when needed in real-time. Evolve (2015) states that its program functions at “high-performance”, this is made possible by the fact that it uses the language C# and C++ denoting that this is a native Windows application, as such it cannot be used on other devices such as: Windows, Macintoshes, Mobiles, Linux and many other platforms. As a result, “Evolve” has limited its user base to a specific demographic which is PC gamers. On the other hand, I wish to make a web application for the specific reason that it should be usable across multiple devices regardless of the user's device preference, thus allowing my solution to reach as many users as possible. The search algorithm most likely uses Ruby to search for currently logged in users and checks their current game to find if there are any free slots available. However, this algorithm does not consider the parameter of a user's location, which my solution aims to include as Evolve mainly aims to match users into a free party slot regardless of location.

A further website tackling the issue of problematic matchmaking is the website "DestinyLFG", this website provides the users with the facilities to find gaming partners based on a number of search parameters. "DestinyLFG" offers a number of search parameters that the results can be filtered by, these include: "users' current console, gaming mode, character level and mic" (DestinyLFG.com, 2015). Whilst, this website is rich in features it has a number of weaknesses, the largest being that it is tailored towards a single game making its audience reach severely limited as only gamers playing this specific game can utilise the matchmaking capabilities, this is an issue I wish to rectify within my solution by making features available across multiple games. Furthermore, unlike the proposed solution which will store the users preference by utilising a login system within "DestinyLFG" the user is forced to continually and manually enter their search preferences each time they wish to find another matching gamer, this can be extremely tedious and time consuming due to the fact they will need to renter their search parameters each time they visit the website.

Another main feature that carries this service is the ability for users to add themselves to a database of users who can be searched, this is achieved by users by clicking a button named "List My Guardian" (DestinyLFG.com, 2015) which will allow them to enter various information that shall be used as criteria when another user is searching for players. Some of the data the user can enter to reflect themselves and their gaming character include: language (this fields allows users to enter their speaking language), the gaming mode they want to embark on and their characters level, as a result the users will be met with other players matching their search preferences. It is apparent that this websites searching process is used to match the parameters specified with users currently stored within the database with matching tags. The result of this search provides very little flexibility as it will only show complete matches to the user's preferences and will not recommend any other gamers if none are found, this limits users to a small boundary of players to meet. My solution aims to overcome this issue by giving the user the flexibility to change their parameters to expand their search such as choosing a different game or changing their filter tags.

This service is built using the LAMP stack which is a highly popular open source web platform used to develop and run dynamic websites and services, the technologies include and used within "Destiny LFG" include Linux, Apache, MySQL and PHP, this choice of technology aids the development and deployment of this website as these technology choices aid with performance and stability. "Destiny LFG" is most likely utilising PHP conditionals to power its searching and filtering mechanism and algorithm allowing for robust and fast results with minimal amount of server-side scripting.

Appendix B: Mainstream Algorithmic Approaches

There are a number of existing mainstream algorithmic approaches to recommendation and matchmaking systems, whilst not all pertaining to the topic of gamer matchmaking they all tackle the same challenge of matching individuals that would work well together based on a strict set of criteria.

A common approach that recommendation systems take when tackling the matchmaking process is associating text based values such as answers with numerical values to generate percentages which are then used to drive the matchmaking process. An example of this approach is used within the online dating, matchmaking and recommendation system "Okcupid" (2015) which generates recommendations by using a strict algorithm to calculate a "similarity percentage" (Okcupid, 2015). This algorithm utilises three key values: "Your answer, How you'd like someone else to answer, and How important the question is to you" (Okcupid.com, 2015) which are then converted into numerical values and used to calculate the similarity percentage, with the numerical value of importance of the question being used to calculate the total points available and two individual's matching answers being used to create score against the total points available which are then used to calculate the percentage.

The online dating system "Okcupid" recommends a partner to match a user with based on the results of this formula which calculates similarity percentages. The two users with the highest percentages based on the results of a strict set of questions are recommended to one another and matched together, as stated above this service learns from: "Your answer, How you'd like someone else to answer, and How important the question is to you" (Okcupid.com, 2015). "Okcupid" creates a total by assigning numerical values based on the users answer to the question "How important is this question to you?" as stated by "Okcupid.com" (2015) the levels of importance' and their point values are shown in table below.

Level of Importance	Point Value
Irrelevant	0
A little important	1
Somewhat important	10
Very important	250

Figure. 12: Importance Value (Okcipid.com, 2015)

An example taking this algorithm into consideration to generate a recommendation would include two individuals (referred to as person X and person Y in this study) each completing two questions to test their compatibility and whether they would be recommended to one another. Considering the questions “How clean are you?” and “Do you like gaming?” both being answered in the format of: users answer, level of importance and what they want their match to answer. With Person X’s answer to the first question being “Very Clean”, “Very Important”, “Average” and person Ys answer to the same question being: “Average”, “Average”, “Very Important”.

Person Ys similarity percentage to person X would be calculated by first taking the important of the first question in to consideration, since it has been given the importance level of “Very Important” the total level will be set to 250, since the second answers importance was set to “A little important it was given” value as one. As a result the overall total is 251. Since the first answers of both person X and Y match they were given 250 points. However, since the second answers didn’t match they were not given the addition 1 point. As such the similarity percentage is calculated by “(250/251)*100” showing that there is a 99.6% match for person Y to X. Next, we need to calculate how person X matches up to person Ys preferences. Since person Y set the first answers importance to “A little important” a value of 1 has been assigned and since the second questions importance level was set to “Somewhat Important” it has been assigned a value of 10, thus making the total 11. Since, the person As first answer matches person Ys they have scored 10 points, but the second questions did not match as such they have also lost a single point. As a result, their similarity percentage (calculated by “(10/11)*100”) is 90.9%. Finally to calculate to calculate the final match percentage the two percentages are multiplied together and square rooted in our case giving a value of 95%.

Whilst, “Okcupid” was used as an example there are a plethora of other matchmaking and recommendation systems available that utilise a similar numerical value based system to power their recommendations systems. An example in terms of gaming systems would be to use the numerical values associated with skills e.g. “Level 4” to find players also in the same skill level, consequently matching users with someone they could play on even terms with. Furthermore, a number of online games algorithm focus on locating the fastest ranking server with a free slot that the gamer can join. However, the large drawback associated with this method is that it produces unfair teams which contain unbalanced numbers of not only players, but gamers of a certain level, thus usually giving a certain team a clear advantage resulting in a subpar experience.

Appendix C: Requirements Survey Responses

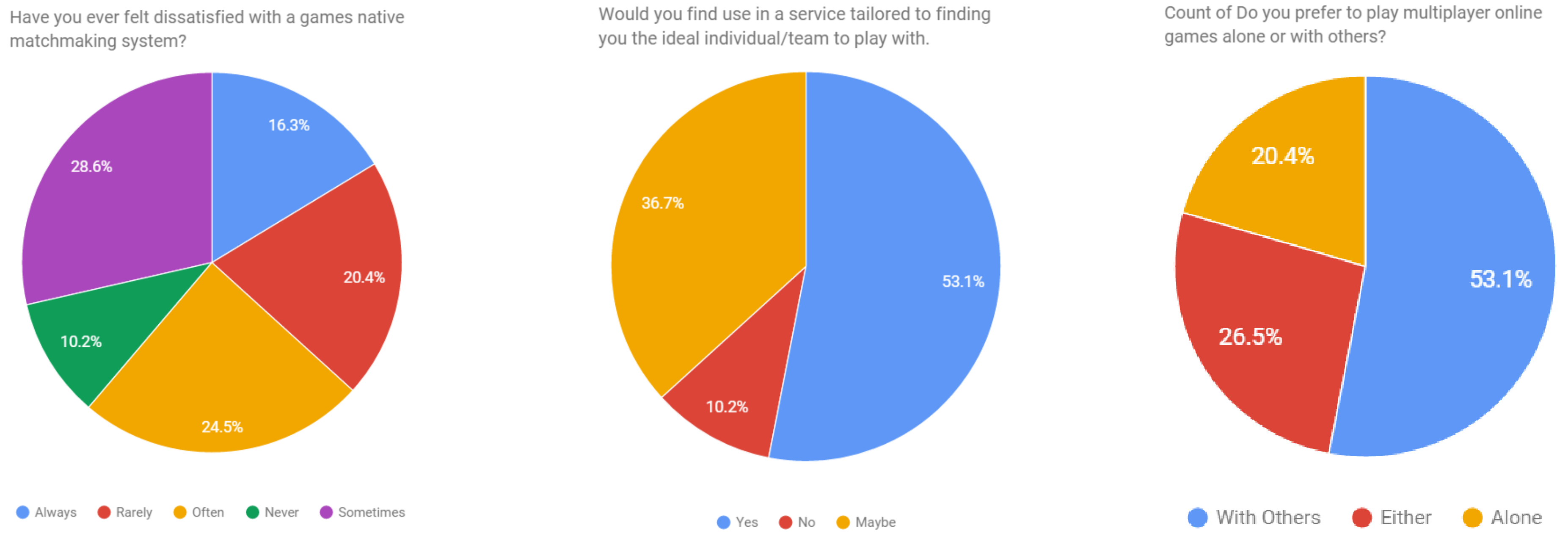


Figure 12: Requirements Survey Chart

If you answered yes to the above Question 1, What made you feel this way?
Only when it lags like hell. But overall I don't complain.
I have repeatedly been paired up with younger individuals, who show very little respect when playing with others and would have preferred to have been matched up with someone more mature.
I often get teamed up with people who don't share my language, so working as a team towards a common goal is impossible and frustrating.
Finding people of the same level as me in games can be somewhat difficult using a games native matchmaking. They don't provide enough info.
Players are rarely matched by skill, causing frustration to the individual player.
Latency may also be an issue if players are matched with varying regions.
Experienced players being matched with novices.
Taking way too long to connect you to a game which leads to players leaving the connection in hopes of it going faster on another try.
Being automatically paired with people from different regions can cause sever connection issues with in turn causes lag ruining the online gaming experience.
I used to play Lord of the Rings Online. I hated its PUG matchmaking system - it worked well for level-cap instances and raids, but on lower levels, you'd either have to sit in the queue for hours, waiting for the game's idea of an "ideal" party to happen, or risk a random assortment of companions, and end up with a level 40 DPS, a level 20 tank, and a level 25 healer trying to push through a level 25 encounter. You end up with the DPS drawing aggro, the tank not being able to take the beating and/or continually dropping aggro, and an aggravated healer trying to keep everyone alive and preferably not draw anything.
Getting paired with people of different ages can make it hard to communicate.
It sometimes takes too long.

It didn't make intelligent choices
Sometimes the matchmaking seems to gauge skill levels poorly.
Servers lagging, other people lagging, non-competitive matchmaking (no skill based mm)
Poor lobby structure or P2P. I have also been forced to wait for extended periods of time as matchmaking was unavailable to me.
It is frustrating when I play someone better than I am, I'd like to talk to someone before I play with them and have access to a friend's list for that specific game instantly.
Sometimes I get matched up with people who are significantly better or worse than me, or just plain old mean.
I disapprove of any matchmaking system that doesn't utilize dedicated servers where each player connects to a host server independent of the players themselves. Peer-to-peer systems never work well in my experience.
Player ranks being too far apart, making the game frustrating/unfair. Mostly encountered this in MOBAs.
People who don't speak my language ruins the matchmaking systems for me.
Being matched with opponents who are far away or have poor internet connections.
It is frustrating when I play someone better than I am, I'd like to talk to someone before I play with them to know their skill levels and have access to a friend's list for that specific game instantly to know who I'm playing with
Occasional mismatch in skill level.

Appendix D: Use Case Descriptions

Name	User Login
Identifier	UC-001
Brief Description	A user (member or administrator) of the matchmaking system wishes to login to the system.
Actor(s)	Member, Administrator
Flow of Events	
Basic Flow	

This use case begins when a logged-out user selects the login option from the landing page.

1. The system prompts the user to enter their username and password into the provided fields.
2. The system generates and displays the fields to the user.
3. The user enters their username and password into the provided fields.
4. The user submits their credentials.
    a. The system authenticates the entered username and password to ensure that the credentials are valid in the system storage.
    b. The system checks the user’s type.
    c. The system identifies the user type is member.
5. The user is logged in successfully and redirected to the member home page.
6. The use case ends successfully.

Alternate Flows	
Title	**Description**

A1. Invalid User	If at step 4a of the Basic Flow the users entered details are invalid as a result of the systems authentication, the following occurs: 1. The system describes the reason the user failed authentication. 2. The system presents possible solutions to resolve the authentication issue. 3. The system prompts the user to re-enter their invalid username or password. 4. The use case resumes from step 3 of the Basic Flow.
A2. User type is Administrator	If at step 4c of the Basic flow the system identifies the user's type as an Administrator, the following occurs: 1. The Administrator is taken to the Administrator dashboard and is granted the allocated additional privileges. 2. The use case ends successfully.
Exceptions	
Title	**Description**
E2. Abort login procedure	If at step 3 of the Basic Flow the user cancels the login procedure by clicking the "Cancel" option the use case is terminated with a failure condition.
Pre-Conditions	
Title	**Description**
1. The user must have completed the signup process ("User Signup" use case) or the Administrator must have been added to system by the developer.	
Post-Conditions	
Title	**Description**
Success	The user is authenticated by the system and the system displays a page based on the user type.
Failure	The user is unable to log in for one or more reasons and is taken back to the landing page.

Name	User Registration
Identifier	UC-002
Brief Description	A user of the matchmaking system wishes to gain access to the system by registering to create an account.
Actor(s)	Member
Flow of Events	
Basic Flow	
This use case begins when the new user selects the signup/register option from the landing page. 1. The system prompts the user to enter their registration information, which includes: username, password, region, language, game interests and a profile picture. 2. The system issues the provided fields. 3. The user enters their registration information into the provided fields. 4. The user submits their entered information. a. The system validates the entered information to check for its correctness by evaluating its formatting. 5. The use case ends successfully.	
Alternate Flows	
Title	**Description**
A1. Invalid Information	If at step 4a of the Basic Flow the users entered registration information is deemed as invalid by the system the following occurs: 1. The system indicates the fields with containing the errors and offers possible solutions to the errors. 2. The system prompts the users to re-enter the incorrect information. 3. The use case resumes from step 3 of the Basic Flow.
Exceptions	
Title	**Description**

E2. Abort sign-up procedure	If at step 3 of the Basic Flow the user cancels the registration process by clicking the "Cancel" option the use case is terminated with a failure condition.
Pre-Conditions	
Title	**Description**
N/A	
Post-Conditions	
Title	**Description**
Success	The users entered information successfully stored in the systems database and their username and password may now be used to login to the system.
Failure	The user is unable to sign up for one or more reasons and is returned to the landing page.

Name	Manage Profile Details
Identifier	UC-003
Brief Description	A user of the matchmaking system wishes to manage their profile by adding or changes the stored details.
Actor(s)	Member
Flow of Events	
Basic Flow	

This use case begins when the user accesses the profile section of the matchmaking system. 1. The user selects option “add details” option. 2. The system prompts the user to enter their: gaming interests, Xbox gamer-tag and/or PlayStation gamer-tag. 3. The system generates the required fields. 4. The user completes the relevant fields by adding the required information. 5. The user submits their entered information. a. The system validates the entered profile information to check for its correctness and validity. 6. The system adds the details entered to the user’s profile. 7. The use case ends successfully.	
Alternate Flows	
Title	**Description**
A1. Invalid information	If at step 5a of the Basic Flow the user’s entered profile information is deemed as invalid after validation occurs, the following occurs: 1. The system indicates the fields containing the invalid information. 2. The system offers possible solutions to rectify the validation errors. 3. The system prompts the user to re-enter the incorrect information. 4. The use case resumes from step 4 of the Basic Flow.
A1. User selects update	If at step 1 of the Basic Flow the user has existing details stored within the system the following occurs: 1. The system generates the required fields. 2. The system retrieves the stored existing profile information and fills out the generated fields using the retrieved information. 3. The user modifies their chosen fields with new information. 4. The use case resumes at step 5 of the Basic Flow.
Exceptions	
Title	**Description**

E2. Abort add/update profile details procedure	If at step 4 of the Basic Flow and step 3 of A1 in the Alternative Flows the user chooses to cancel the add details or update details procedure by choosing the "Cancel" option the use case is terminated with a failure condition.
Pre-Conditions	
Title	**Description**
1. The user must have registered to the matchmaking system. 2. The user must be logged into the matchmaking system.	
Post-Conditions	
Title	**Description**
Success	The user's new profile are added to the system and stored within the database for later use.
Failure	The user's profile details are not altered and the user is returned to the home page.

Name	Start Matchmaking Search
Identifier	UC-004
Brief Description	A user of the matchmaking system wishes to start a matchmaking search to find players wishing to play a game.
Actor(s)	Member
Flow of Events	
Basic Flow	

This use case begins when a user accesses the matchmaking system. 1. The system uses the users stored profile details as a parameters for the matchmaking search. 2. The system conducts the search. a. The system calculates the number of returned results and returns most recent seven results. 3. The use case ends successfully.	
Alternate Flows	
N/A	N/A
Exceptions	
Title	**Description**
E1. No results	If at step 2a of the Basic Flow the system calculates the number of returned results is equal to zero the following occurs: 1. The system prompts the user that there are no results using their details as parameters. 2. The use case terminates with a failure condition.
Pre-Conditions	
Title	**Description**
1. The user must have registered to the matchmaking system. 2. The user must be logged into the matchmaking system. 3. The user must have added their details to their profile.	
Post-Conditions	
Title	**Description**
Success	The search is correctly run, thus generating the results and the system then allows the user to view the results.
Failure	The search is unable to complete for one or more reasons and the user is redirected to the manage details page.

Name	View Matchmaking Results
Identifier	UC-005
Brief Description	A user wishes to the see the results matchmaking search.
Actor(s)	Member
Flow of Events	
Basic Flow	
This use case begins once the “Start Matchmaking” search use case has been completed by a user and the results are to be is displayed. 1. The system retrieves the results of the matchmaking search. 2. The system displays the results of the matchmaking search by displaying any of the resulting matching users: username, message, and game. 3. The user browses through the results of the matchmaking search. 4. The use case ends successfully.	
Alternate Flows	
Title	**Description**
A1. More details	If at step 3 of the Basic Flow the user selects a particular result to view more details on the matched player/member, the following occurs: 1. The system retrieves and displays more details pertaining to the result the user selected by displaying an invite button and the any users already added to the matchmaking request. 2. The user views the detailed information on the matching user/member/player. 3. The use resumes from step 3 of the Basic Flow.

A2. Filter results	If at step 3 of the Basic Flow the user chooses enters a value into the provided the filter field after the search has completed the following occurs: 1. The system conducts a further search retrieving the next set of matchmaking results. 2. The system retrieves the appropriate results and appends them to the existing result. 3. The use case resumes from step 3 of the Basic Flow.
A3. Send invite	If at step 2 of A1 of the Alternative Flow the user selects the invite button, the following occurs: 1. The system adds the user invite user to hosts notifications. 2. The use case resumes from step 2 of A1 of the Alternative Flows.
A4. Loading more results.	If at step 3 of the Basic Flow the user reaches the end of the provided matchmaking results the follow occurs: 1. The system conducts a further search retrieving the next set of matchmaking results. 2. The system retrieves the appropriate results and appends them to the existing result. 3. The use case resumes at step 3.
Exceptions	
Title	**Description**
E1. No results	If at step 2 of the Alternative Flow no results are available when retrieving additional results, the following occurs: 1. The system prompts the user that there are no matching requests. 2. The use case terminates with a failure condition.
Pre-Conditions	

Title	Description
1. The user must be logged into the matchmaking system. 2. The user must have previously conducted a matchmaking search.	
Post-Conditions	
Title	**Description**
Success	The user views the results of the matchmaking search and is able to complete a range of actions, such as: sending an invite other users and viewing a request in further detail.
Failure	The system is unable to display the results of the matchmaking search for one or more reasons and the user is notified.

Name	View Group List
Identifier	UC-006
Brief Description	A user wishes see the other users/members they have added to their friends list.
Actor(s)	Member
Flow of Events	
Basic Flow	

This use case begins when the user selects the “View Group List” option from within the system. 1. The system retrieves all the groups and users that have been added previously. 2. The system displays the details for each of the retrieved groups within list by displaying their: matchmaking requests message. 3. The user browses through the group list. 4. The user selects a group from the retrieved list. 5. The system displays a chat window. a. The user sends messages b. The user retrieves messages.	
6. The member exits the chat window. 7. The use case ends successfully.	
Alternate Flows	
Title	**Description**
A1. Remove group.	If at step 4 of the Basic Flow the user chooses the “delete group” option, the following occurs: 1. The system prompts the user that they will be disband the currently selected group. 2. The user confirms they wish to delete the group from the list. 3. The system removes the group from the friends list. 4. The use case resumes from step 3 of the Basic Flow.
Exception	
Title	**Description**
E1. No Group	If at step 1 of the Basic Flow no groups are available when retrieving the friends list, the following occurs: 1. The system prompts the user that there are no friends stored within the list. 2. The use case terminates with a failure condition.

Pre-Conditions	
Title	**Description**
1. The user must be logged into the matchmaking system. 2. The user must have added another member to the friend's list.	
Post-Conditions	
Title	**Description**
Success	The user views the list of their available friends from the results of the matchmaking search and are able to select the friend to exchange messages
Failure	The system is unable to display the friends list from one or more reasons and the user is redirected to the homepage.

Name	Manage Invites
Identifier	UC-007
Brief Description	A user wishes manage the invites they have received from other user as part of their matching request.
Actor(s)	Member
Flow of Events	
Basic Flow	
This use case begins when the user selects the "Notifications" option from within the system. 1. The system displays notification panel. 2. The system retrieves all pending invites pertaining to all the users matching requests. 3. The system displays the details for each of the retrieved requests, with an invite request displaying user's image, username and accept/decline buttons. 4. The user browses through the notification list. 5. The user selects an invite. a. The user selects accept button.	

b. The system adds the accepted user to the group and logs this in the database. 6. The use case ends with a success condition.	
Alternate Flows	
Title	**Description**
A1. Decline Invite.	If at step 6a of the Basic flow the user instead clicks the decline button the following occurs; 1. The system logged the declined user in the database to prevent further requests being made from the same user. 2. The accepted user is added to the users group for the matchmaking request. 3. The user is removed from the notification panel. 4. The use case resumes from step 4 of the basic flow.
Exception	
Title	**Description**
E1. No Invites	If at step 2 of the Basic Flow no invites are found the user is notified and the use case ends with a failure condition.
Pre-Conditions	
Title	**Description**
1. The user must be logged into the matchmaking system.	
Post-Conditions	
Title	**Description**
Success	The user is able to manage their invites by declining or accepting invites to add them to their group.
Failure	The system is unable to display the invites for one or more reasons and the user is notified.

Name	Manage Members

Identifier	UC-008
Brief Description	The Administrator of the system wishes to manage the users currently registered into the system.
Actor(s)	Administrator
Flow of Events	
Basic Flow	
The use case starts when the Administrator selects the manage users option from the administrator home page: 1. The administrator chooses the delete member feature. 2. The system retrieves a list of all the current members'. 3. The administrator selects the member the wish to remove from the system	
4. The system displays a confirmation message to prompt the Administrator for their final decision (either "yes" or "no") 5. The Administrator selects "yes" to delete the member. 6. The system removes member from the system. 7. The use case ends successfully.	
Alternate Flows	
Title	**Description**
A2. Ban User	If at step 1 of the Basic Flow the administrator chooses the "Ban Members" feature, the following occurs. 1. The system retrieves a list of all the currently signed up members. 2. The Administrator selects to ban a specific member. 3. The system displays a confirmation message to prompt the Administrator for their final decision (either "yes" or "no") 4. The Administrator selects "yes" to ban the member. 5. The system proceeds to revoke the member's access to the system. 6. The use case ends successfully.

Exceptions	
Title	**Description**
E1. Cancel Delete	If at step 4 of the Basic Flow the Administrator wants to cancel deleting a member, the following occurs: 1. The administrator chooses “no” to not delete the member. 2. The system prompts the administrator that the member has not been deleted. 3. The use case ends with a failure condition.
E2. Cancel Ban	If at step 3 of the A2. The Administrator wants to cancel banning the member, the following occurs: 1. The administrator chooses “no” to not ban the member. 2. The system prompts the administrator that the member has not been banned. 3. The use case ends with a failure condition.
Pre-Conditions	
Title	**Description**
1. The administrator must be logged in. 2. The user type must be an administrator. 3. There must at least one user stored within the system.	
Post-Conditions	
Title	**Description**
Success	The Administrators chosen member is either banned or deleted successfully and they are returned to the Administrator dashboard.
Failure	The system is unable to display the update the details for one or more reasons and the administrator is returned to the administrator dashboard.

Name	Logout
Identifier	UC-009

Brief Description	A user of the matchmaking system (either Administrator or Member) wishes to logout the system.
Actor(s)	Member, Administrator
Flow of Events	
Basic Flow	
This use case begins when a user wishes to logout from within the system. 1. The user selects the logout option. 2. The system logs the user out of the system. 3. The system redirects the user to the landing page 4. The use case ends successfully.	
Pre-Conditions	
Title	**Description**
1. The user must be logged in to the matchmaking system. 2. The user no longer wants to be logged in and use the system.	
Post-Conditions	
Title	**Description**
Success	The user is logged out of the system and is redirected to the landing page by the system.
Failure	The user is unable to logout and the system prompts the user with the reason and they are returned to the home page.

Table 4: Use Case Descriptions

Appendix E: Technologies Chosen

E.1. Introduction

There a number of web technologies available that could be used to implemented the proposed solution for the semi-automated matchmaking system. This chapter covers the comparison of various technologies and justifications of choosing a particular technology to be used during the implementation process.

E.2. Web Server Technologies

When considering web server technologies to be used during the implementation process the two most notable choices are 'Apache HTTP Web Server' and the 'NGINX Web Server'. The most impactful differences between these platforms in relation to my implementation is the scripting languages they utilise, in addition to the support available e.g. documentation. These difference greatly influenced my choice in technologies.

The Apace HTTP server was originally created in 1995 and has continually been developed by the Apache Software Foundation, the fact that Apache has existed longer than its competition holds a number of advantages which are brought forward by its longevity including: widespread support for the platform, maintained and large amount document making setup, maintenance and updates an easily manageable task. Furthermore a number of database Management Systems such as 'PhpMyAdmin' have been paired with this webserver making the management of the server an easier task. Because of these advantages this webserver is an ideal choice when considering the time constraints this implementation is under when considering solving common errors. On the other hand, NGINX which was released after Apache was able to resolve a number of issues which Apache faced due to its later release date. The major advantage of NGINX handles the scaling of websites with greater ease and efficiency as it is able to handle a large number of connections at single time. As a result when considering the future of this implementation this web server technology would be an ideal choice to ensure scalability and consistency when used by a large number of users resulting in the system being put under heavy load.

Due to the ease of use and compatibility with PHP I believe for this implementation the use of the Apache HTTP Web Server would aid the implementation in terms of speed as it provides a number of useful documentation that would help resolve a number of issues I may run into.

E.3. Single Page Application or Multi-Page Application

It is stated by Nielson that “Usability is a quality attribute that assesses how easy user interfaces are to use” and “also refers to methods for improving ease-of-use during the design process” (Nngroup.com, 2016). Therefore it is imperative that the correct method of developing the implementation of the matchmaking web application is chosen and it is constructed in a manner that leverages and enhances its usability. The two methods of implementing a website are creating a multi-page application or a single-page application, these methods offer different advantages in the aspects of usability and present different challenges during the development process.

The most common method of implementing a website is through multiple page, this is due to the ease of development and the fact that a website is able to become modular allowing users to access the webpages directly, thus making some content easily accessible to the user aiding in usability. This method functions through a website being constructed of multiple pages and having these pages connected using links, once a user selects a particular link a request is sent by the browser to the server to retrieve the new page and show it user, this methodology has become the norm.

An alternative to this is the creation of a single-page application (SPA) which offers a number of advantages when considering the aspect of usability. According to Wikipedia (2016) “A single page application (SPA) also known as a single page interface (SPI), is a web application or website that fits on a single webpage with the goal of providing a more fluid user experience akin to a desktop application”. Rather than the method of loading multiple pages present in multi-paged websites, single-page applications load a single HTML page into the browser with the content being updated dynamically resulting in a more fluid user experience. Due to the fact webpages have a number of repeated segments e.g. the footer, navigation and a number of images with only a small segment of content changing from page to page, a major disadvantage of multi-page websites that a SPA resolved is the number of browser refreshes to retrieve small segments of data which is not efficient in both performance and users time SPA method is highly advantageous as it rectifies this issue by only requiring a single page to be loaded and updates only the content needed. However, a major disadvantage that makes it a harder choice to use within my implementation is it lacks when considering ease of development.

When considering Nielsen (2016) principle of efficiency which regards users and “how quickly they can perform tasks” and also the principle of satisfaction stating “How pleasant is the design to use?” SPA add to these principles greatly when compared to the alternative of multi-page application. Due to the fact that when using a SPA a more fluid and faster user experience is given due to fact content does not have to be repeatedly downloaded and can be instead requested by AJAX, resulting in a more satisfying experience when considering the design as only a portion of content that needs to be changed is swapped behind the scenes. As a result, this results in users having to waste less time and they are able to perform tasks with greater efficiency and speed as less synchronous requests are needed resulting in less time spent waiting resulting in faster execution.

In conclusion, despite that difficultly coupled with creating a single-page application I have decided to use this within my implementation to improve the usability of my application by increasing its ease of use, loading times and fluidity. As a result, this would increase the effectiveness of the implementation overall.

E.4. Back-end Framework

Due to the decision to make a SPA (single-page application) a number of frameworks have become available to choose from to power the front-end of the system (e.g. user interface and data retrieval) as well as the backend-end of the system which communicates to the database and will be accessed via AJAX.

Due to the fact I have had some experience using the language of PHP this will be used as my chosen service side language. Furthermore, rather than using simplistic creating PHP files I have opted to use a MVC framework, this provides a number of options however due to their learning curve the two most appealing choices are the frameworks CodeIgniter and Laravel. The advantage of using a PHP framework is that it allows me to become abstracted from the low level details and as such aids in increasing the efficiency of the implementation process. Furthermore, a number of frameworks provide a MVC structure allowing which will allow me to correctly separate the logic within the backend of my code into controllers and models making the application much more organised and scalable. Furthermore the addition of the use of the framework and i’s modularity will aid the process of testing, maintenance and debugging.

Whilst comparing the frameworks of CodeIgniter and Laravel I believe choosing the former will benefit the implementation process. Due to its simplicity and its lightweight nature I believe this framework is the ideal choice as it provides the best performance and maintains the edge when considering the difficulty required during the setup process. As such I have chosen CodeIgniter as my backend service for my implementation process.

E.5. Front-end Framework

Due to the fact it was decided that the implementation would be a SPA (single-page application) a need for a front-end framework that provides the ability to effectively connect to the abstracted back-end service via AJAX as well as the other various features required of SPA has arisen. There are a number of JavaScript frameworks available, with the most popular being Angular and Backbone. Both Backbone and Angular differ in a number of aspects, these include: syntax, speed of development, code maintainability, and speed, therefore the use cases for each framework must be compared before a decision is made.

Angular is a highly popular framework that is used to build complex single page applications, this framework was created in 2009 and is an open source framework meaning it has a large community supporting it with a large amount of documentation available. The main advantage of Angular itself is that it is an extension to the mark-up language HTML being used within this implementation, furthermore, its unique functionality of two way data binding allows for a number of functionality that would require a number of lines of JavaScript to be performed without any additional code being written.

Backbone.js is another framework that provides the tools needed to build a SPA whilst avoiding needless complexity, however this framework is more lightweight and does not force new syntax upon the user allowing the users current knowledge to be transferred with ease. Backbone.js is a simplistic JavaScript which has existed since 2010. The major advantage brought forward by this framework is that it does not contain the complexity held by competitors such as AngularJS. Backbone.js provides the developer with a MVC framework, providing models and collections to handle the storage of key-value data which are also able to access a backend service via AJAX, views to contain user interfaces which are able to handle events and data changes. In addition a Router is also provided to aid in the handling of URL allowing states to be created within this SPA.

Due to the fact these frameworks will be loaded as dependencies means that they can greatly impact the loading time of pages which is a crucial factor for users when considering usability. As such it is imperative functions efficiency. The main factor that can impact this is the frameworks size and when comparing the size it becomes apparent the Backbone is a much more lightweight framework with it being 6.5kb compared to Angular size of is 39.5kb.

As Backbone is more lightweight compared to its competitor and allows previous knowledge to easily be transferred over, furthermore due to the fact that Backbone can be easily paired with other libraries means it can easily be extended with further functionality unlike Angular. Lastly, when considering the use case of Angular being focused towards building more complex application which require a single framework, whereas backbone is more extensible and focuses on smaller web application. As such I believe Backbone's advantages and use cases make it the ideal choice to be used within my implementation as such this will be used in conjunction with CodeIgniter to construct my application.

Appendix F: Chosen Usability Framework

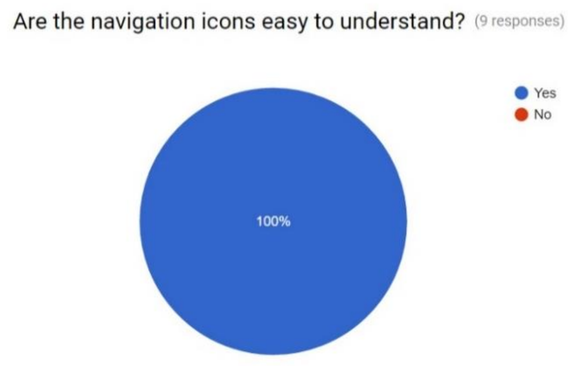
There are number of techniques that are employed to quantify and measure a web applications or software's usability. Usability testing is a method employed to discover bottle-necks and issues within a specific design, the most adopted method is to track a users' interaction with a user interface and to also record the user's satisfaction with the said interface that is currently being tested. The most common techniques adopt this methodology when testing usability, whilst a select few methods utilise alternative techniques to achieve a similar result. A method of recording users' feeling and in turn the effectiveness of usability is through the use of surveys/questionnaires, therefore, surveys have once again been employed during the user acceptance testing phase to judge the effectiveness of the websites usability.

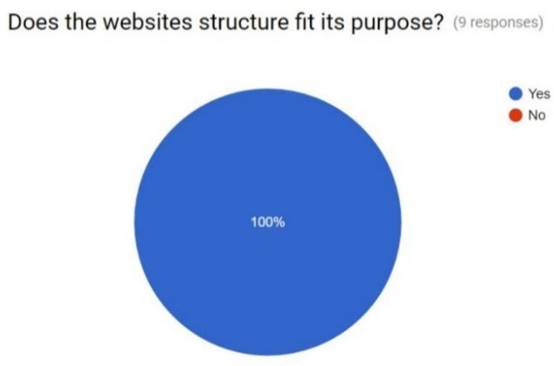
The questions within this survey will be based on the usability checklist/framework by Larisa Thomason (2004), this framework judges the effectiveness of usability through a number of question which judge the: branding, the ease of navigation, overall design. Furthermore, due to the fact this framework incorporates many usability principles brought forward by Jakob Nielsen makes it the ideal choice to judge the effectiveness of my system.

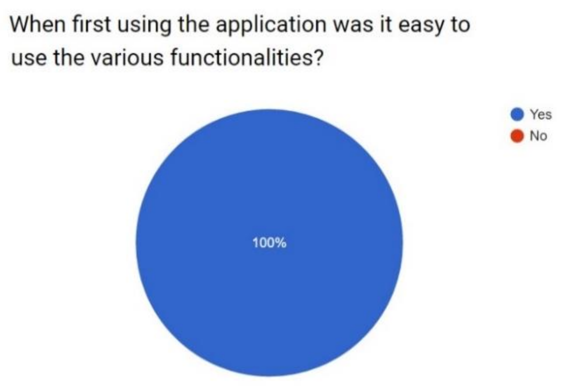
The results of the checklist shall be represented using simplistic metrics that can easily be analysed to judge the effectiveness of the usability, the checklist in question that has been filled out by a number of users during the user acceptance/black-box testing phase and is shown in Appendix H.

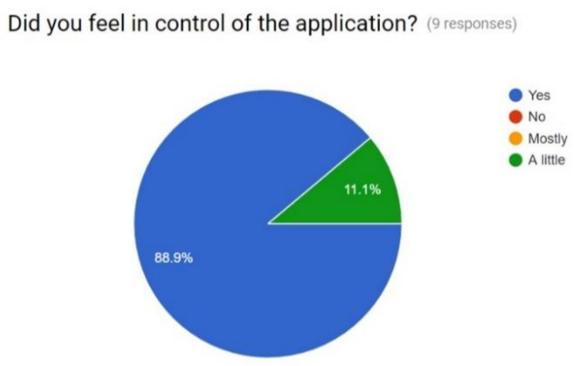
Appendix G: Usability Testing

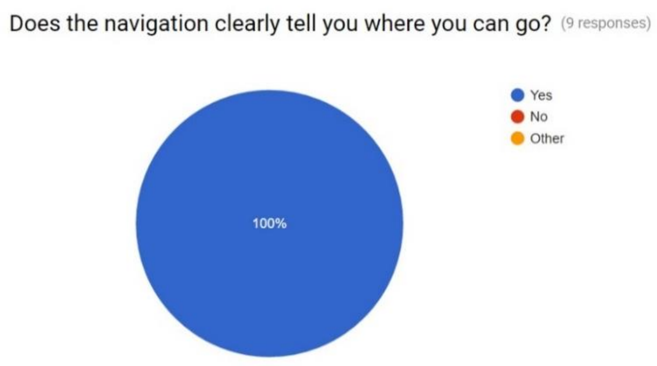
The figure shown below outlines the questions and the results of the survey conducted during this phase, this survey was designed to evaluate the system overall along with its effectiveness in terms of usability. As a result, the questions stored within this document (as shown in Appendix H) were constructed and derived from the website usability checklist provided by Larisa Thomason (2004) which also incorporates a number of Jakob Nielsen's principles which can be used to accurately determine effective use of usability principles.

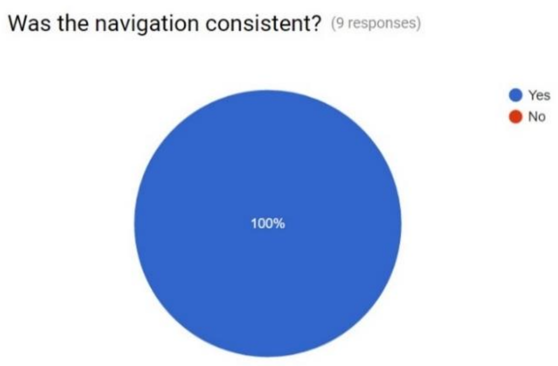


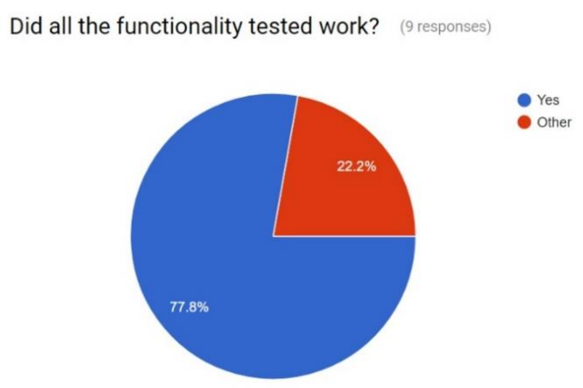


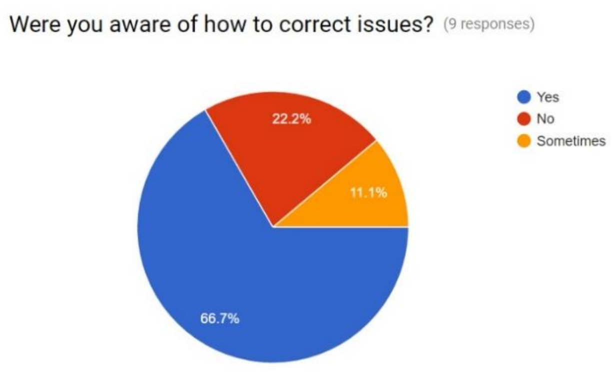


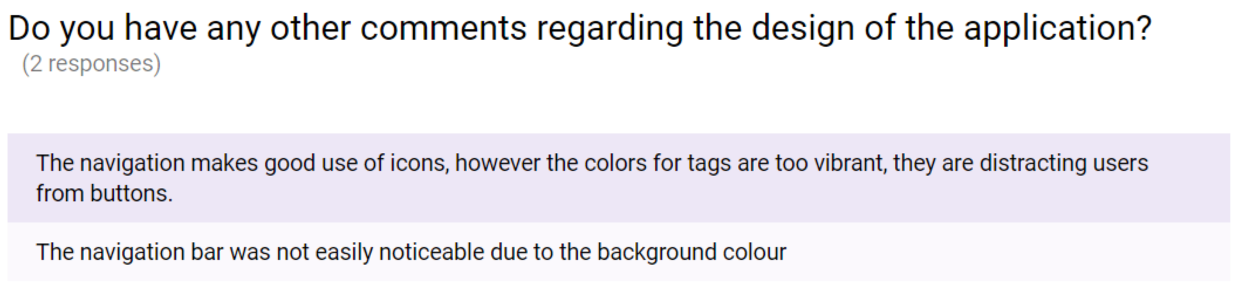












Do you have any other comments regarding the design of the application?
(2 responses)

The navigation makes good use of icons, however the colors for tags are too vibrant, they are distracting users from buttons.

The navigation bar was not easily noticeable due to the background colour

Figure 13: Usability Testing Results

The results of the survey above show the effectiveness of the application in terms of usability, these aspects that were considered at length during the design stage were judged during this testing phase to ensure their effectiveness within the solution.

A key usability issue which is pivotal in ensuring the success of my project was learnability, it is stated that learnability regards the question: “How easy is it for users to accomplish basic tasks the first time they encounter the design?” (Nngroup.com, 2016). During the process of black-box testing this principle was evaluated through the question “When first using the application was it easy to use the various functionalities?” with 100% of users saying replying “Yes” denotes the fact that my system has successfully applied this aspect of usability. A further aspect when considering Nielsen’s principles is that of efficiency which regards “how quickly can they perform tasks” (Nngroup.com, 2016) and the fact the results to the question “was it easy to use the various functionalities?" was an unanimous 'yes' by all participants highlight the fact that the structure of the application made the process of fulfilling the requirements easy or the users and as such this statistic also highlights the effectiveness of the application when evaluated by the user in terms of usability.

A major usability aspect brought up within Larisa Thomason's (2004) website usability framework is that of navigation, it is stated that the navigation should be "in the same place on every page and have the same format" to avoid users becoming "confused and frustrated". This usability aspect was carefully considered and applied through the use of a persistent navigation placed to the left of the content (as outlined within the design phase) and when considering the importance brought forward by this checklist in regards to this aspect I believe it has been successfully achieved within my application. Through usability testing it has become apparent that all users felt that my application maintained consistent use of the navigation as shown through 100% of usability testers answering "Yes" to the question "Was the navigation consistent?", this in turn resulting in users feeling in control of the website and its functionality with 88.9% agreeing that they felt in control of the application as depicted by the question "Did you feel in control of the application?", However, the remaining 11.11% replied they felt "a little" in control highlighting in terms of usability this an aspect that can be improved upon, but overall it achieves the desired effect in terms of usability.

Whilst, I was not able to measure the usability principle of "Memorability" within Nielsen's Heuristics due to the limited timeframe that testing could be conducted in, through the questions completed by testers I am able to gauge the effectiveness of the application in terms of this principle. The design choice of using bold navigation icons was accepted by the testers as all stated they were easy to understand, this combined with the previous metric of all users being able to use the functionality of the application with ease on their first try highlights the fact that my application is easy to understand and use. Consequently, it is likely that usability principle would be achieved by application.

The penultimate usability aspect tested using the frameworks applied was that of errors, Larissa Thomason (2004) poses the question “Can they recover from errors?”, this ability is key to my application as the system requires data input and through testing I was able to discern that 66.7% of users were aware of how to correct issues encountered due to the implementation of error messages. However, this statistic highlights the fact that this aspect is a usability principle that needs further improving as a small number of users encountered errors they were unable to solve. The final most principle to be tested against the framework used was that of satisfaction, this principle is important in gauging if users found the application useful and likely to return, through the use of a linear scale within the application I was able to judge out of the 8 testers the average rating was 8.8 out of 10 emphasising the fact the end-users testing the application felt the application was above average and thus increasing their likelihood of reusing the application.

A core method of testing usability is to “test the site on real users” (Thomason, 2004), to satisfy this method of testing the website was made public for short amount of time between the 22rd and 29th April 2016 after the user acceptance testing was complete to gain further statistics from actual users and gauge the effectiveness of the application as a whole. The analytics results present within Appendix J depict the usability and adoption of the application during this time frame, the application picked up “24” users and a total of “150” page views, thus creating emphasis on the fact that the intended audience of gamers are adopting the application. Furthermore, the fact the bounce rate was 34.48% (which is a negative statistic and should remain as low as possible) highlights the fact that users went on to visit other pages of the website, such as the timeline by creating an account, connoting the effectiveness of the applied usability techniques as the application was able to convert prospects into a members who completed the various functionalities provided by the system.

In conclusion the usability framework in the form of Larisa Thomason's (2004) "Web Site Usability Checklist" was used to form a survey which be used to gauge proficiency of my web application in terms of usability. Whilst the websites analytics statistics gathered over the course of a course of a week emphasis the success of the application in addition to the effectiveness of the usability. These results highlighted the fact that my application was proficient when considering many aspects of usability and user adoption, however a number of flaws and improvements were also brought to light during these processes.

Appendix H: User Acceptance (Usability) Questionnaire

Navigation (Learnability and Efficiency)

1. **Does the websites structure fit its purpose?**
   Mark only one oval.

   ◯ Yes
   ◯ No

2. **Does the navigation clearly tell you where you can go?**
   Mark only one oval.

   ◯ Yes
   ◯ No
   ◯ Other: ....

3. **Did you feel in control of the application?**
   Mark only one oval.

   ◯ Yes
   ◯ No
   ◯ Mostly

4. **Was the navigation consistent?**
   Mark only one oval.

   ◯ Yes
   ◯ No

5. **When first using the application was it easy to use the various functionalities?**
   Mark only one oval.

   ◯ Yes
   ◯ No

6. **Are the navigation icons easy to understand?**
   Mark only one oval.

   ◯ Yes
   ◯ No

Layout

7. **How quickly were you able to perform tasks?**
Mark only one oval.

- Fast
- Medium
- Slow

8. **Was the websites a layout consistent?**
Mark only one oval.

- Yes
- No
- Other: ____________________

9. **Do you have any other comments regarding the design of the application?**

____

____

____

____

____

Error Tolerance

10. **Did you encounter any errors?**
Mark only one oval.

- Yes
- No

11. **Were you aware of how to correct issues?**
Mark only one oval.

- Yes
- No
- Sometimes

12. **How do you feel about the error handling within the website?**

....

....

....

....

....

Platform and Implementation

13. **Was the loading time efficient? (below 20 seconds)**
Mark only one oval.

◯ Yes
◯ No
◯ Other: ....

14. **Did all the functionality tested work? If no what didn't work?**
Mark only one oval.

◯ Yes
◯ Other: ....

Where you satisfied with the website overall?

15. *Mark only one oval.*

1	2	3	4	5	6	7	8	9	10
◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

Appendix I: Test Cases

Test Case ID	T-001
Test Priority	High
Module Name	Authentication
Test Title	Verify login functionality with valid username and password.
Description	A test to see if a user can use their stored login credentials to gain access to the matchmaking system.
Preconditions	1. User must have valid username and password.
Post-conditions	1. User's session details are logged and user is redirected to the systems main view (timeline).

#	Action	Expected Result	Status	Comments
1	User accesses the landing page.	The system should display the landing page for the website displaying.	Pass	-
2	Click the login button.	The login fields should be displayed containing the appropriate input fields for username and password.	Pass	-
3	Enter username "munaibh".	-	Pass	-
4	Enter password "pwd123".	-	Pass	-
5	Submit button is clicked.	See post-condition 1.	Pass	-

Test Case ID	T-002

Test Priority	High
Module Name	Authentication
Test Title	Verify login does not accept invalid username and password.
Description	A test to see if the login screen recognises an invalid username and/or password and informs the user of their error.
Preconditions	1. User must have accessed the landing page. 2. User must have access the login screen by clicking the 'login' button.
Post-conditions	1. System will display error message to the user labelled 'your username or password are incorrect' and the user will not be redirected.
Defect ID	D-001

#	Action	Expected Result	Status	Comments
1	Enter username 'munaibh'	-	Pass	-
2	Enter invalid password "pass332"	-	Pass	-
3	Submit button is clicked.	See post-condition 1.	Fail	The error message was displayed, but the user was redirected to the main view with an error of '401 Unauthorized' being displayed and no content being loaded.

Test Case ID	T-003
Test Priority	Medium
Module Name	Validation

Test Title	Verify login form validation.
Description	A test to see if the login form validates the data entered correctly and prevents in accurate information being submitted.
Preconditions	1. User must have accessed the landing page. 2. User must have access the login screen by clicking the 'login' button.
Post-conditions	1. The user's credentials should not be submitted.

#	Action	Expected Result	Status	Comments
1	Leave username blank	An error message should be displayed above the input field saying "this field is required".	Pass	-
2	Enter invalid password "pass332"	This field should be accepted.	Pass	-
3	Submit button is clicked.	See post-condition 1.	Pass	-

Test Case ID	T-004
Test Priority	High
Module Name	Authentication
Test Title	Verify Register Functionality.
Description	A test to see if register screen/form correctly validates user information and submits user information.
Preconditions	1. User must have accessed the landing page.
Post-conditions	1. The user's credentials are added to the database and the user is redirected to the landing page.

#	Action	Expected Result	Status	Comments
1	The username “munaibh” is entered.	The username already exists within the database so the user should be notified.	Pass	-
2	The username is changed to “munaibh117”	The username should be accepted as it does not exist.	Pass	-
3	The password “hussain447” is entered twice.	These passwords should be compared and be accepted due to them being the same.	Pass	-
4	Location left unselected and the form is submitted	An error should be displayed above the empty location field saying it is required.	Pass	-
6	The optional fields of gamer-tags and interests are left blank.	-	Pass	-
5	The submit button is clicked.	See post-condition 1	Pass	-

Test Case ID	T-005
Test Priority	High
Module Name	Matchmaking
Test Title	Verify matchmaking request retrieval.
Description	A test to see if they system retrieves and displays the correct matchmaking results based on the users preferences (location and incomplete)
Preconditions	1. User must be logged into the system.

	2. The database must contain matchmaking requests.
Post-conditions	1. The system should display the first 7 matching matchmaking requests to the user outlining the user, message and the game. 2. The system should retrieve the next set of matching matchmaking requests from the database and append to the existing list.

#	Action	Expected Result	Status	Comments
1	The timeline view is loaded.	See post condition 1	Pass	The system retrieves only the incomplete posts and they were all within the "United Kingdom" same as logged in user.
2	The last matchmaking result is scrolled to.	See post condition 2	Pass	-

Test Case ID	T-006
Test Priority	High
Module Name	Matchmaking
Test Title	Verifying sending an invite to a request functions.
Description	A test to determine whether a user is able to send an invite to a posted matchmaking request and the system validates multiple requests to by the same user.
Preconditions	1. User must be logged into the system. 2. The user must have at least a single matchmaking request available to them.
Post-conditions	1. The system logs the invite for the request within the database and awaits the requester's action. 2. The system recognises the user has already sent an invite to the selected post and ignore the invite request.

#	Action	Expected Result	Status	Comments
1	The timeline view is loaded.	The system should display the first seven matching matchmaking requests.	Pass	-
2	A matchmaking request is selected.	The system expands the request revealing added players along with an invite button.	Pass	-
3	The invite button is clicked	See Post Condition 1	Pass	-
4	The same invite button is clicked a second time.	See Post Condition 2	Pass	-

Test Case ID	T-007
Test Priority	Low
Module Name	Matchmaking
Test Title	Very a player cannot send an invite to their own request.
Description	A test to see if the invite functionality within prevents a player from sending an invite to their own matchmaking request.
Preconditions	1. The user must be logged into the system. 2. A matchmaking request must have been submitted and be incomplete. 3. The player's matchmaking request should be displayed within the timeline.
Post-conditions	1. The user should not be able to send an invite and the request will be ignored by the system.
Defect ID	D-002

#	Action	Expected Result	Status	Comments
1	The player's matchmaking request is selected.	The request is expanded revealing any accepted players, but the invite button should not be displayed.	Fail	The invite button was displayed, thus allowing a user to send an invite.
2	The invite button is clicked.	See Post Condition 1	Pass	-

Test Case ID	T-008
Test Priority	Medium
Module Name	Matchmaking
Test Title	Verify matchmaking requests can be further filtered by tags
Description	Test the matchmaking systems capability to further filter matchmaking results by the tags attached to the posts.
Preconditions	1. The user must be logged into the system. 2. The system must contain more than 3 matchmaking containing varying tags including "Mic", "BossFight" and "1v1".
Post-conditions	1. The user should only be able to view results containing the entered tag name.

#	Action	Expected Result	Status	Comments
1	The timeline view is viewed.	The system should display the first seven results for all tags.	Pass	-
2	The tag name "Mic" is entered into the filter field.	The system should replacing the previous results with only results containing the tag "Mic".	Pass	-

3	The fist tag is removed and the tag “BossFight” is entered.	The system should remove the previously filtered result and replace it with the single result containing the specified “BossFight” tag.	Pass	-
4	The final filter tag “1v1” is entered.	The post containing the “1v1” tag should only be displayed to the user replacing all other results.	Pass	-

Test Case ID	T-009
Test Priority	High
Module Name	Notification
Test Title	Verify notification display.
Description	Test to see if the notifications functionality displays correctly by displaying received invites for matchmaking requests.
Preconditions	1. The user must be logged into the system.
Post-conditions	1. The user will be able to view pending invites and complete actions on them such as “accepting” or “declining”.

#	Action	Expected Result	Status	Comments
1	The timeline view is viewed.	The matchmaking requests along with the user interface should be loaded (including the navigation)	Pass	-
2	The notification option is selected from the navigation.	The notification section is loaded allowing the user to see their pending requests and allows them to act accordingly (post-condition 1)	Pass	-

Test Case ID	T-010
Test Priority	Low
Module Name	Notification
Test Title	Verify a user can cancel pending invite.
Description	A test to see if a user can cancel a pending invite they have made to a posted matchmaking request.
Preconditions	1. The user must be logged into the system. 2. The user must have submitted an invite to a posted matchmaking request and it should not have been accepted of declined.
Post-conditions	1. The users invite will be removed from the database.

#	Action	Expected Result	Status	Comments
1	The notification option is selected from the navigation bar.	The notifications (invites) along with the pending invites and the user interface should be displayed.	Pass	-
2	The provided 'cancel button beside the invite is clicked.	The invite from to matchmaking request is cancelled the post is removed the notification panel and the post-condition occurs.	Pass	-

Test Case ID	T-011
Test Priority	High
Module Name	Notification
Test Title	Verify a user can accept an invite.

Description	A test to see if a player is able to correctly accept an invite made to their submitted matchmaking request.
Preconditions	1. The user must be logged into the system. 2. The user must have received an invite from a posted matchmaking request.
Post-conditions	1. The user will be to matchmaking group and this will be logged within the database.

#	Action	Expected Result	Status	Comments
1	The notification option is selected from the navigation bar.	The notifications (invites) along with the user interface should be displayed.	Pass	-
2	The provided 'accept' button is clicked.	The matchmaking request is removed from the notifications section and the post-condition occurs.	Pass	-

Test Case ID	T-012
Test Priority	High
Module Name	Notification
Test Title	Verify a user can decline an invite.
Description	A test to see if a player is able to correctly decline an invite made to their submitted matchmaking request.
Preconditions	1. The user must be logged into the system. 2. The user must have received an invite from a posted matchmaking request.
Post-conditions	1. The user will be rejected and this will be recorded within the database to prevent future requests being made to this post.

#	Action	Expected Result	Status	Comments
1	The notification option is selected from the navigation bar.	The notifications (invites) along with the user interface should be displayed.	Pass	-
2	The provided 'decline' button is clicked.	The invite should be removed from notifications and the database should be updated (post-condition 1).	Pass	-

Test Case ID	T-013
Test Priority	Medium
Module Name	Messaging
Test Title	Verify user can view groups
Description	Test to see if the system displays groups with more than 1 active member and if they are separated by joined groups and hosted groups.
Preconditions	1. The user must be logged into the system. 2. User must have submitted a matchmaking request and an invite must have be accepted.
Post-conditions	1. The user will be able to access and chat to members of their joined matchmaking groups.

#	Action	Expected Result	Status	Comments
1	The message icon is clicked from the navigation.	The system should display a view containing the groups separated by both hosted and joined.	Pass	-

2	View available groups.	The system should display a single item under the heading my group depicting a user has joined a matchmaking request. Furthermore a single item should listed under listen group.	Pass	-

Test Case ID	T-014
Test Priority	Medium
Module Name	Messaging
Test Title	Test message retrieval capabilities.
Description	Test to see if system is able to load and display messages for a particular matchmaking group.
Preconditions	1. The user must be logged into the system.
Post-conditions	2. The user will be able to view and send messages to other users within the selected group.

#	Action	Expected Result	Status	Comments
1	The message icon is clicked from the navigation.	The system should display a view containing the groups separated by both hosted and joined.	Pass	-
2	A displayed group is selected.	A new section is displayed containing all the messages associated and sent within the selected group along with a form to send more messages.	Pass	-

Test Case ID	T-015
Test Priority	Medium

Module Name	Messaging
Test Title	Test message sending capabilities.
Description	A test to see if a user is able to send messages within a group along with their gamer-tag.
Preconditions	1. The user must be logged into the system. 2. The user must have selected a group and be able to view its messages.
Post-conditions	1. The user will be able to access and chat to members of their joined matchmaking groups.
Defect ID	D-003

#	Action	Expected Result	Status	Comments
1	The message “Hello!” is entered into the text area.	The field should be validated to ensure field is not empty.	Pass	-
2	The send button is clicked.	The message should be sent and all the new messages including the one just sent should be retrieved and appended to the existing messages.	Fail	Whilst the messages were retrieved correctly, the submitted message was sent multiple times.
3	The gamer-tag button is clicked.	The stored gamertag pertaining to the correct console is retrieved from the database and sent as a message.	Pass	-

Test Case ID	T-016
Test Priority	Medium
Module Name	Messaging

Test Title	Test message retrieval capabilities.
Description	Test to see if a user is able to retrieve messages in real-time allowing them to send and view up to date messages.
Preconditions	1. The user must be logged into the system. 2. A group must be selected in which more than a single user is part of. 3. Two process must be active have complete the steps above and send messages.
Post-conditions	2. The user will be able to receive send messages in real time.

#	Action	Expected Result	Status	Comments
1	User 1 sends a message containing “Message1”.	The message is sent and stored within the database.	Pass	-
2	User 2 sends a message containing “Message 2”.	The second message is sent and stored within the database and the two messages sent recently by user 1 and 2 are retrieved and appended to the message list.	Pass	-
3	User 1 waits 60 seconds.	User 2 message is retrieved and appended to the list of messages.	Pass	-

Test Case ID	T-017
Test Priority	High
Module Name	Profile Details/Settings
Test Title	Test to open the profile details management section.
Description	Test to see whether a user is able to gain access to their profile details from within the settings menu.

Preconditions	1. The user must be logged into the system.
Post-conditions	1. The user will be able to edit their entered profile details.

#	Action	Expected Result	Status	Comments
1	The profile option is selected from the navigation bar.	The system loads a new view containing the fields needed filled with the existing data from within the database to be altered.	Pass	-

Test Case ID	T-018
Test Priority	High
Module Name	Profile Details/Settings
Test Title	Test to edit the profile details.
Description	Test the editing of the three available options, these include pinned games, and the gamer tags for both and psn and xbox from within the settings menu.
Preconditions	1. The user must be logged into the system.
Post-conditions	1. The user’s details will be updated and stored within the database.

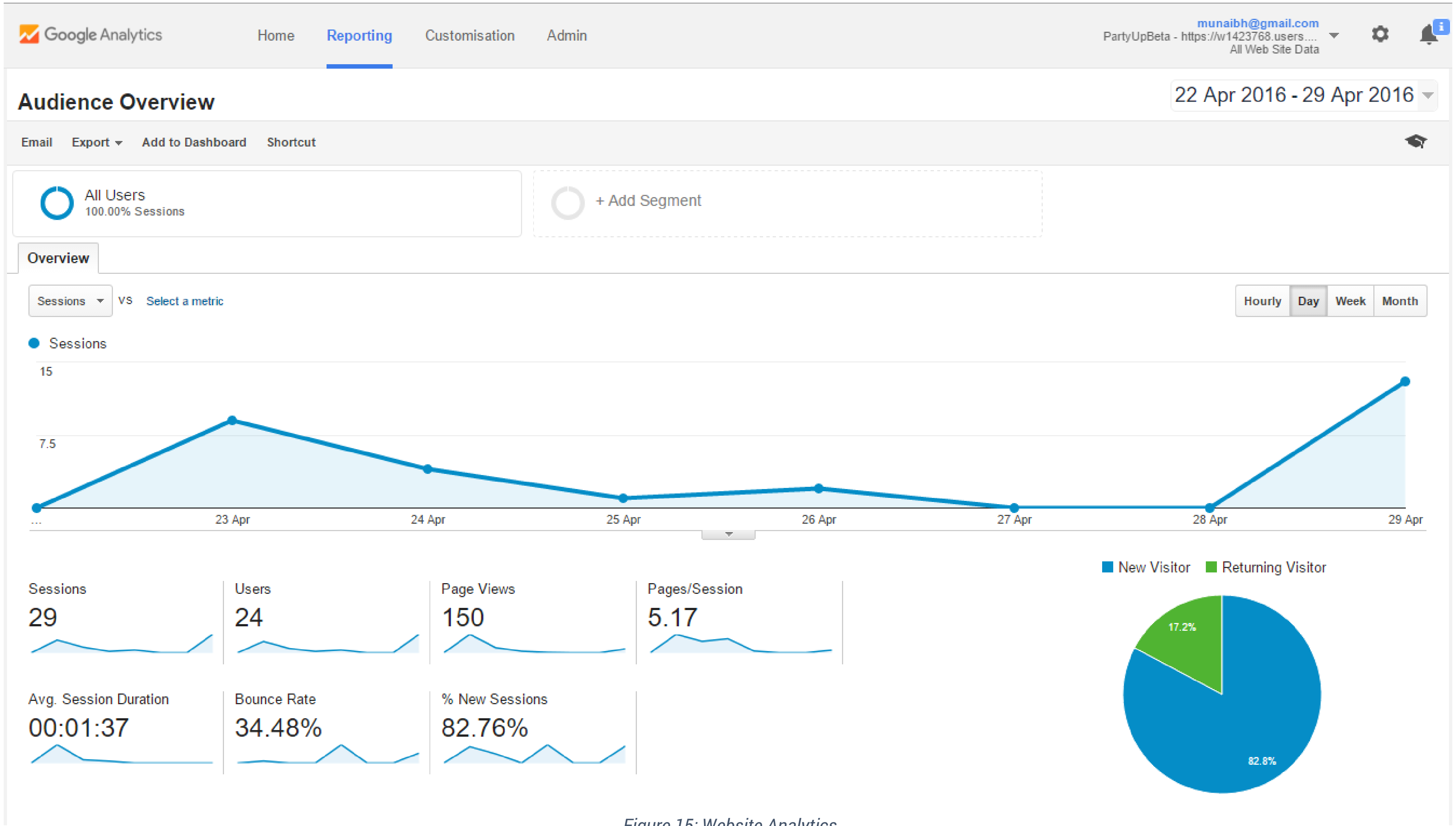
#	Action	Expected Result	Status	Comments
1	The xbox fields information is changed from “munz” to “munz_box”	-	Pass	-

2	The psn gamertag fields content is altered from “munz” to “munz_psn”	-	Pass	-
3	“Pokemon. Destiny” is added to the empty interest field.	-	Pass	-
4	The submit button is clicked.	The user should be redirected to the timeline view and the new values should be updated within the database and the user interface this should be apparent by pinned filtered requested being present within the timeline view.	Pass	-

Defect ID	Resolved	Expected Response	Actual Response
D-001	True	The system will display an error message when username or password are not recognised and the user will not be redirected.	As expected
D-002	True	The matchmaking request is expanded revealing any accepted players, but the invite button should not be displayed.	As expected
D-003	True	The message should be sent and all the new messages including the one just sent should be retrieved and appended to the existing messages.	As expected.

Figure 14: Test Cases

Appendix J: Real World User Analytics



Audience Overview

22 Apr 2016 - 29 Apr 2016

All Users
100.00% Sessions

Sessions	Users	Page Views	Pages/Session
29	24	150	5.17

Avg. Session Duration	Bounce Rate	% New Sessions
00:01:37	34.48%	82.76%

Figure 15: Website Analytics

Appendix K: Milestones

1. Complete Literature Search and Review (Complete by 8th November)
2. Polish Introduction, Scope and Literature overview (9th November – 14th November)
3. Analysis and Modelling (15th November – 5th December)
    a. Context Diagram (Complete by 18th November)
    b. User Case Diagram and Spec (Complete by Use 25th November)
    c. Domain Model (Complete by 2nd December)
    d. Functional and Non-functional Requirements (5th December)
4. Improve Preliminary Report (Complete by 14th December)
5. System Design (14th December – 25th December)
6. Acquire relevant extra knowledge and code simple prototypes (25th December – 31st December)
7. Implementation (1st January – 29th February)
8. System testing and Evaluation (1st March – 30th March)
9. Work on Final Report (1st April - 30th April 2015)

Appendix L: Documentation

To create an account:

1. Visit the website at http://localhost/PartyUpBeta/ , or go directly to the register page http://localhost/PartyUpBeta/#/register.
2. Enter your desired username, password, repeat your password and click next.
3. Choose you location, language and optionally enter your xbox gamertag, playstation gamertag and your preferred games (separated by commas) and click next.
4. Optionally add an avatar and click signup.
5. You will be redirected to the landing page ready to login.

To access an account:

1. Visit the website at http://localhost/PartyUpBeta/.
2. Click the button labelled "sign in".
3. Enter your username and password into the provided fields.
4. Click "Sign in" button and you will be redirected to the timeline.

To make a matchmaking request:

1. Click the pen icon ✎ and the compose panel will appear.
2. In the fields provided add a message, number of players and chosen game.
3. Choose a selection of tags that relate to your request making it easier for people to find you.
4. Click the "send" button and you request will be sent to people in your location.

Manage invites:

1. Click the bell icon and the notifications panel will appear.
2. Your notifications will be displayed here with "accept" and "decline" buttons beside.
3. Click the "accept" button to accept a user and they will be able to message you in the chat window accessed using the @ icon.
4. Click the "decline" button to decline a user and will be removed and stopped from sending any more invites.

Cancel Invite.

1. Click the bell icon and the notifications panel will appear.
2. Under the heading “Pending invites” you will see the game you sent an invite for along with a “cancel button”
3. Click the “cancel” button beside the game name and the invite will be removed and cancelled.

Sending Invite:

1. Click the “expand” link on a post you want to send an invite too, the added players’ icons along with an invite button will be displayed.
2. Click the invite + icon and your invite will be sent.

Filtering Posts:

1. Enter a tag name in the input field above each column.
2. The requests will be filtered to show only the posts with the chosen tag.
3. Optionally to pin a game to the dashboard click the avatar icon and you will be redirected to the settings page, or access it directly through http://localhost/PartyUpBeta/#/settings and add a game to the input field named “interests”. Multiple games can be added by separated the names with a comma.
4. Click the “update” button and you will return to the timeline and will see a pinned column with a heading of you chosen game containing a filtered list.

Add/Change gamer tags:

1. Click the avatar icon or go directly to the settings page using http://localhost/PartyUpBeta/#/settings
2. Enter you chosen xbox or psn gamertag in the correct labelled field and click “update”.
3. You will be redirected to the settings page with you gamertags changed.

Sending Messages:

1. Click the message icon @ and a panel will be displayed containing your groups separated by your “hosted groups” and “joined groups”.
2. Click the name of a group to open the messenger.
3. Enter you message into the provided field and click the “send” button to send it.
4. To send a gamertag click the buttons labelled with the chosen gamertag and it will automatically be sent.

Appendix M: Results

The results of the implementation are shown below with a number of screenshots of the application being shown, these screenshots are taken from the various sections of the application and many of the features have been simulated with two users to power the various functionality required such as: accepting an invite and conducting a chat.

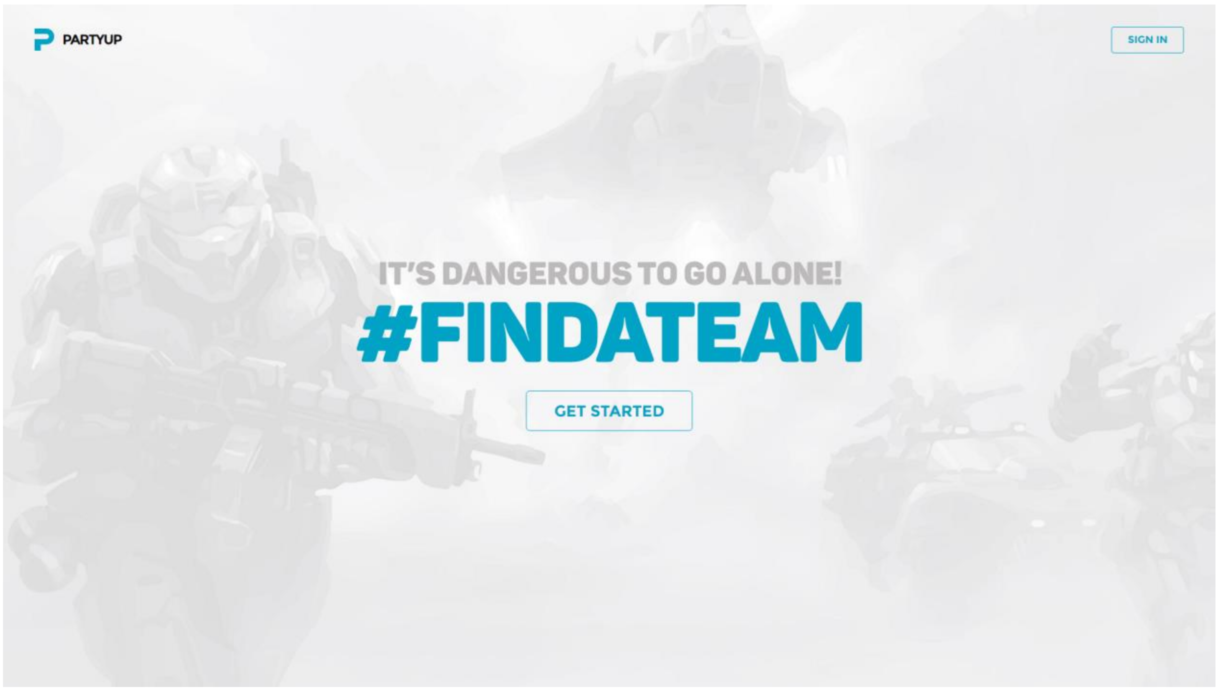


Figure 16: Landing Page View

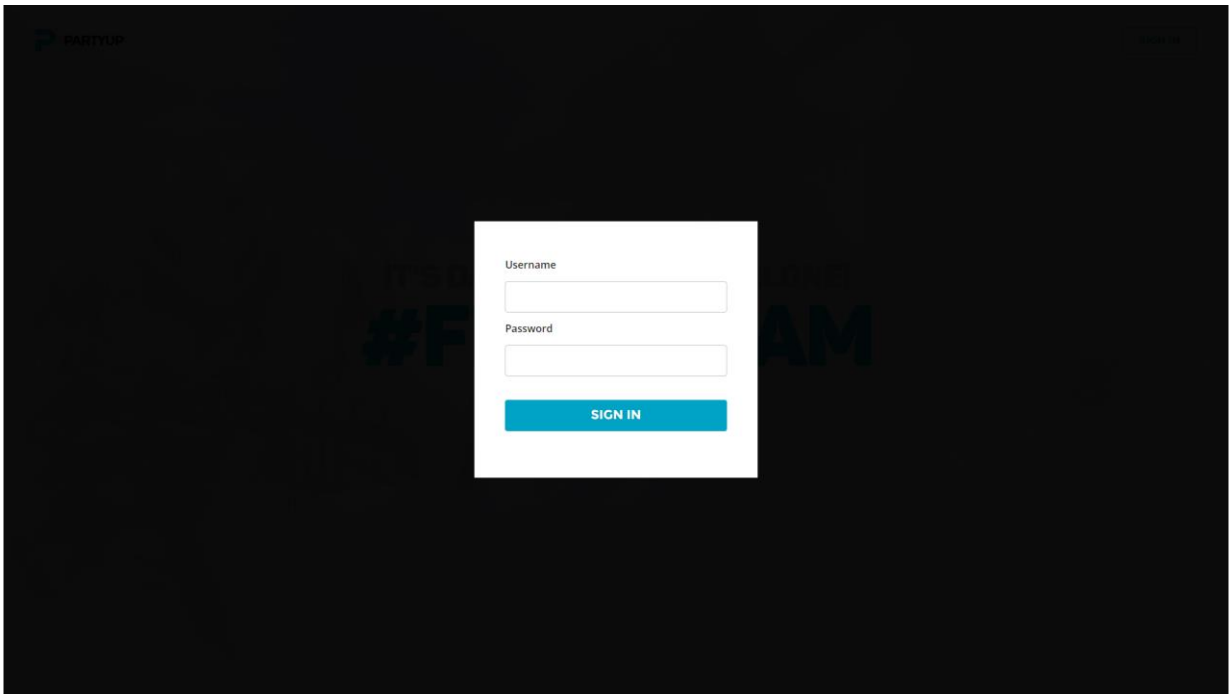


Figure 17: Login Page View

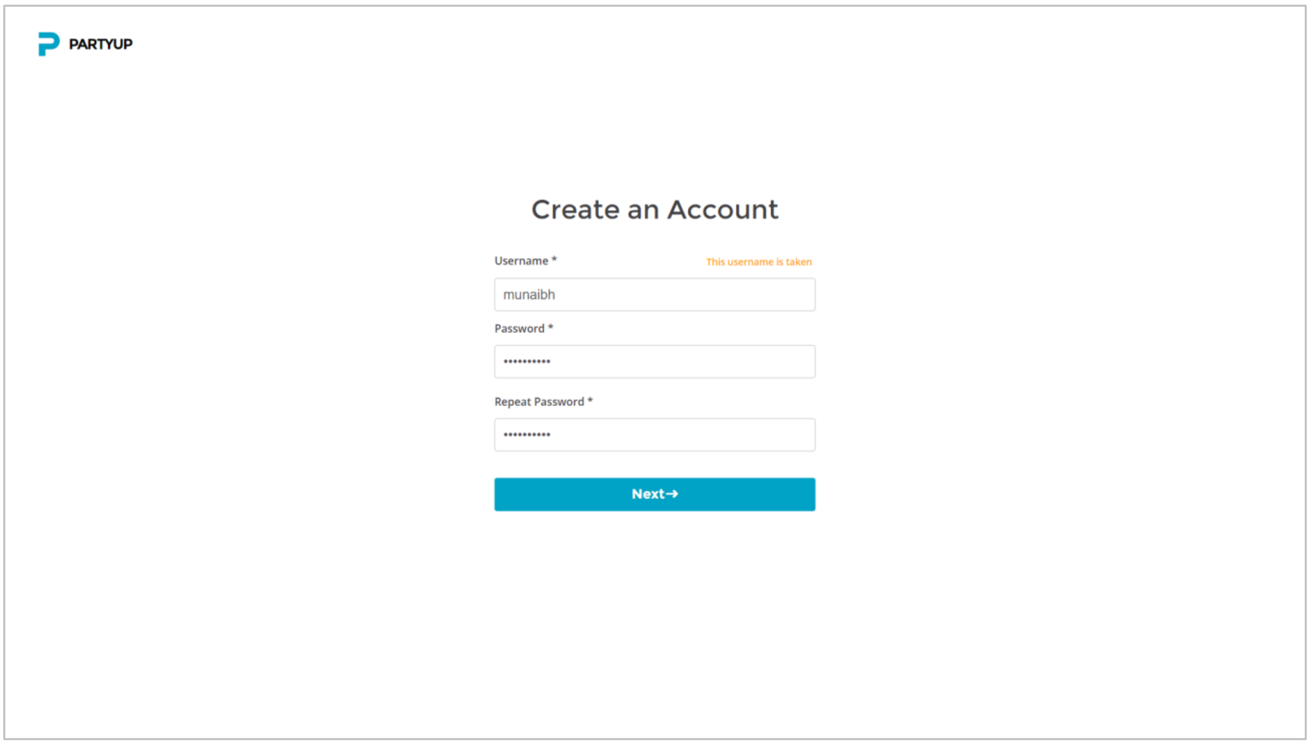
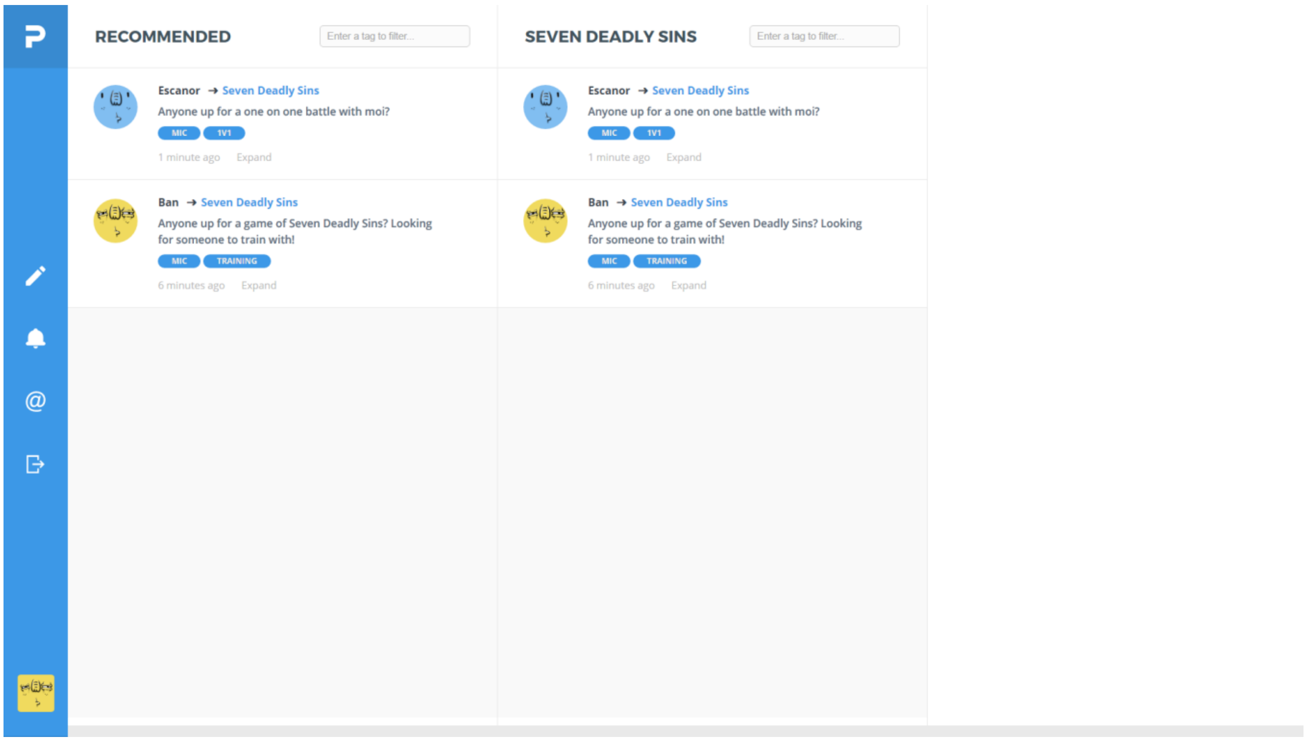


Figure 18: Register View



Escanor → Seven Deadly Sins
Anyone up for a one on one battle with moi?
1 minute ago

Ban → Seven Deadly Sins
Anyone up for a game of Seven Deadly Sins? Looking for someone to train with!
6 minutes ago

Figure 19: Timeline View

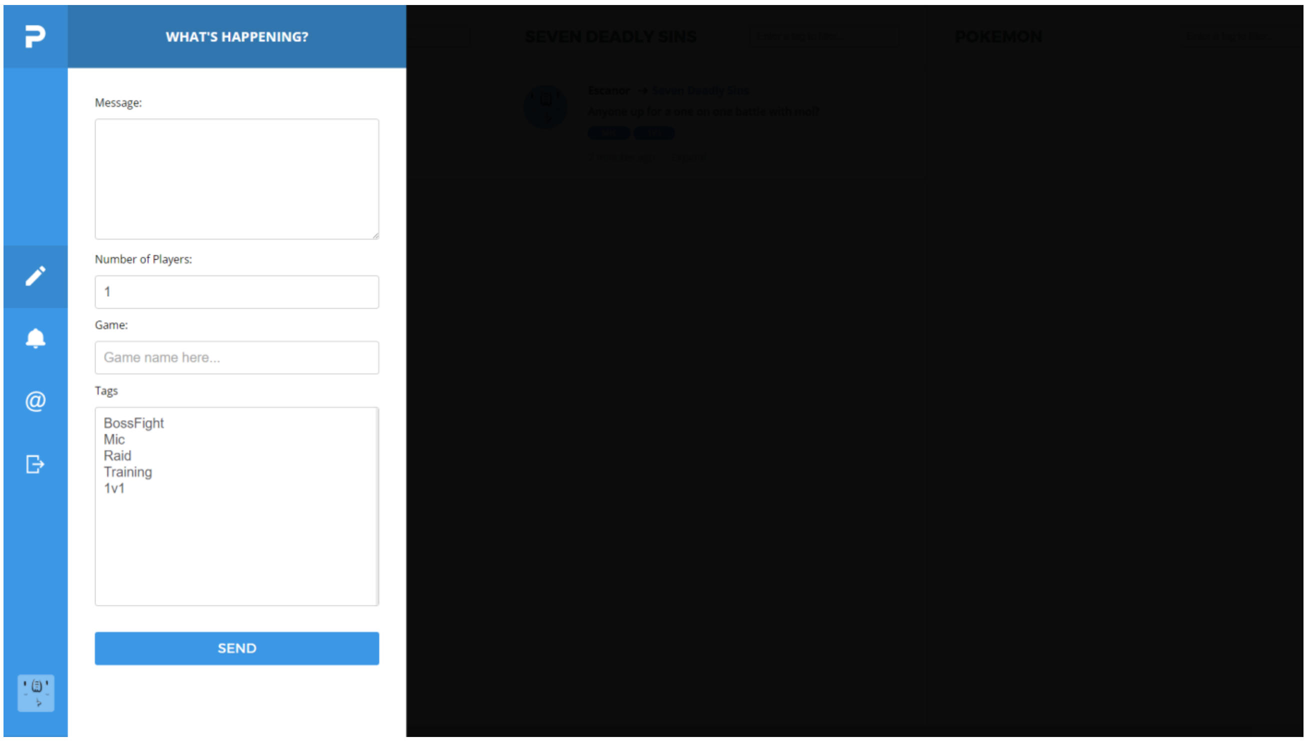


Figure 20: Compose Panel

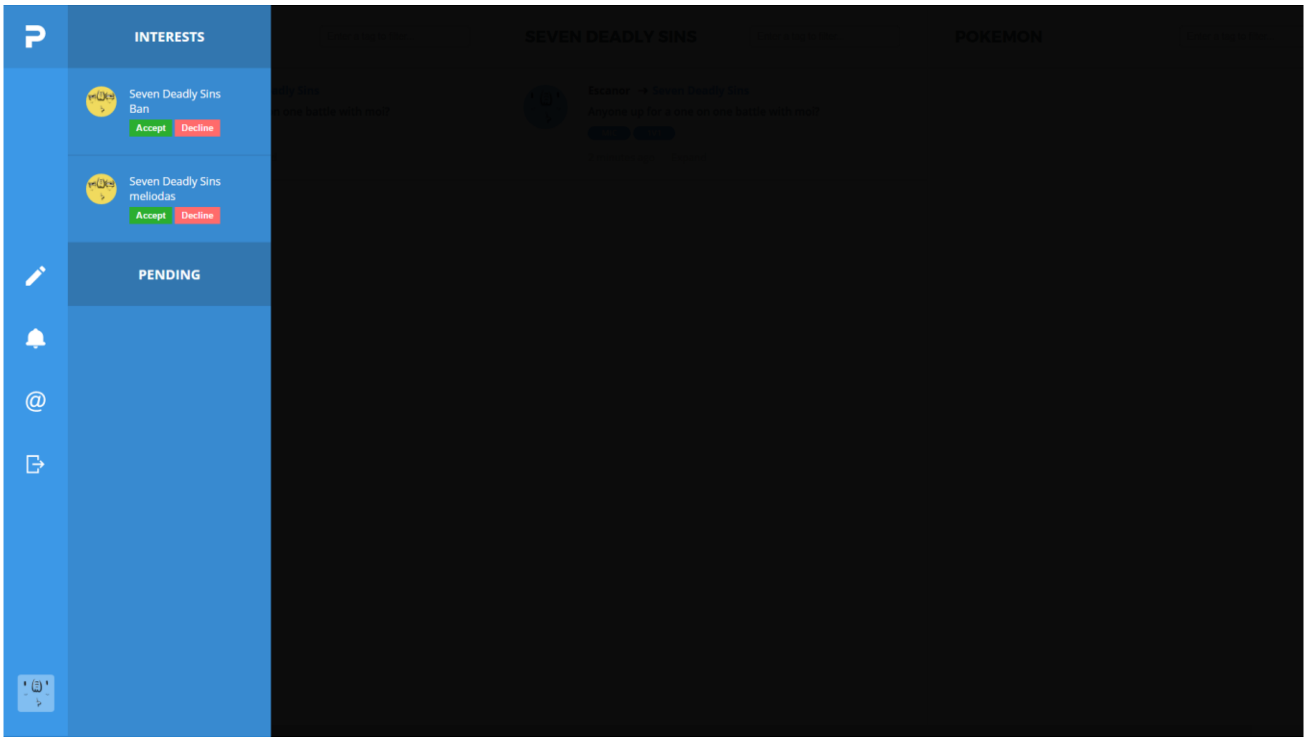
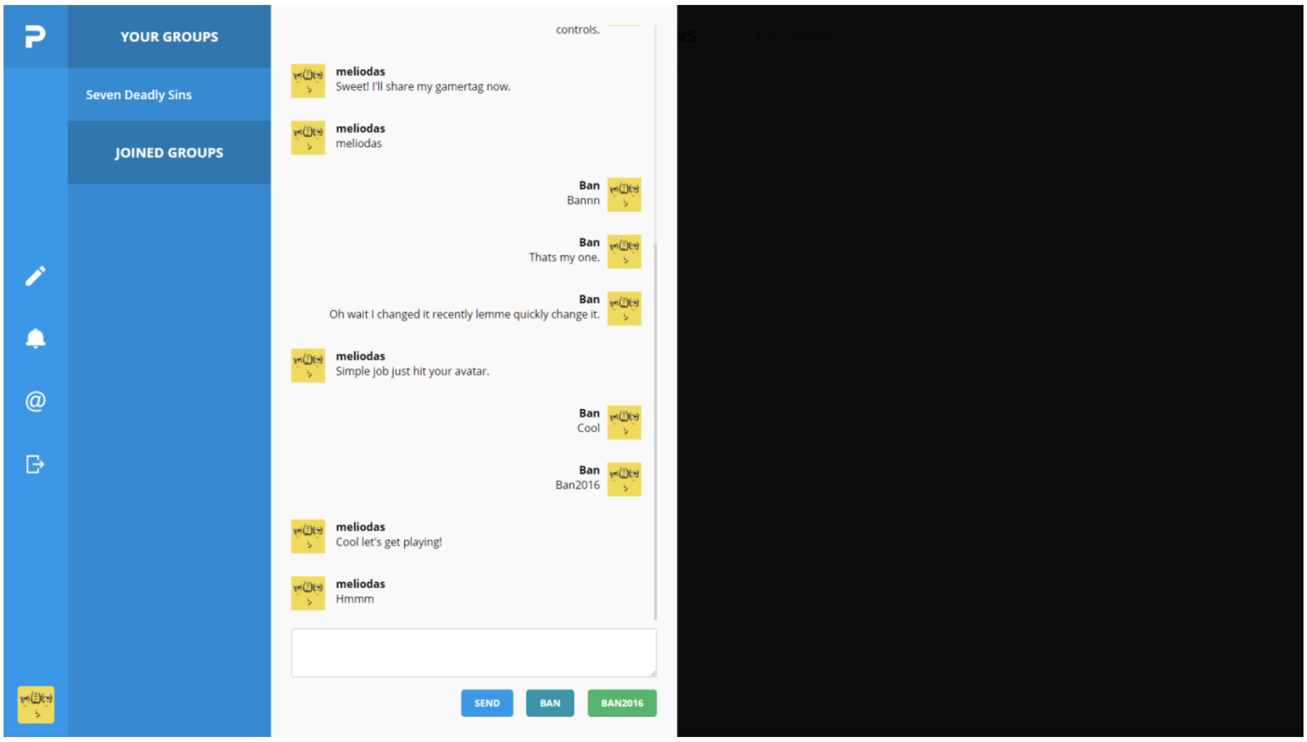


Figure 21: Notifications Panel View



controls.

meliodas
Sweet! I'll share my gamertag now.

meliodas
meliodas

Ban
Bannn

Ban
Thats my one.

Ban
Oh wait I changed it recently lemme quickly change it.

meliodas
Simple job just hit your avatar.

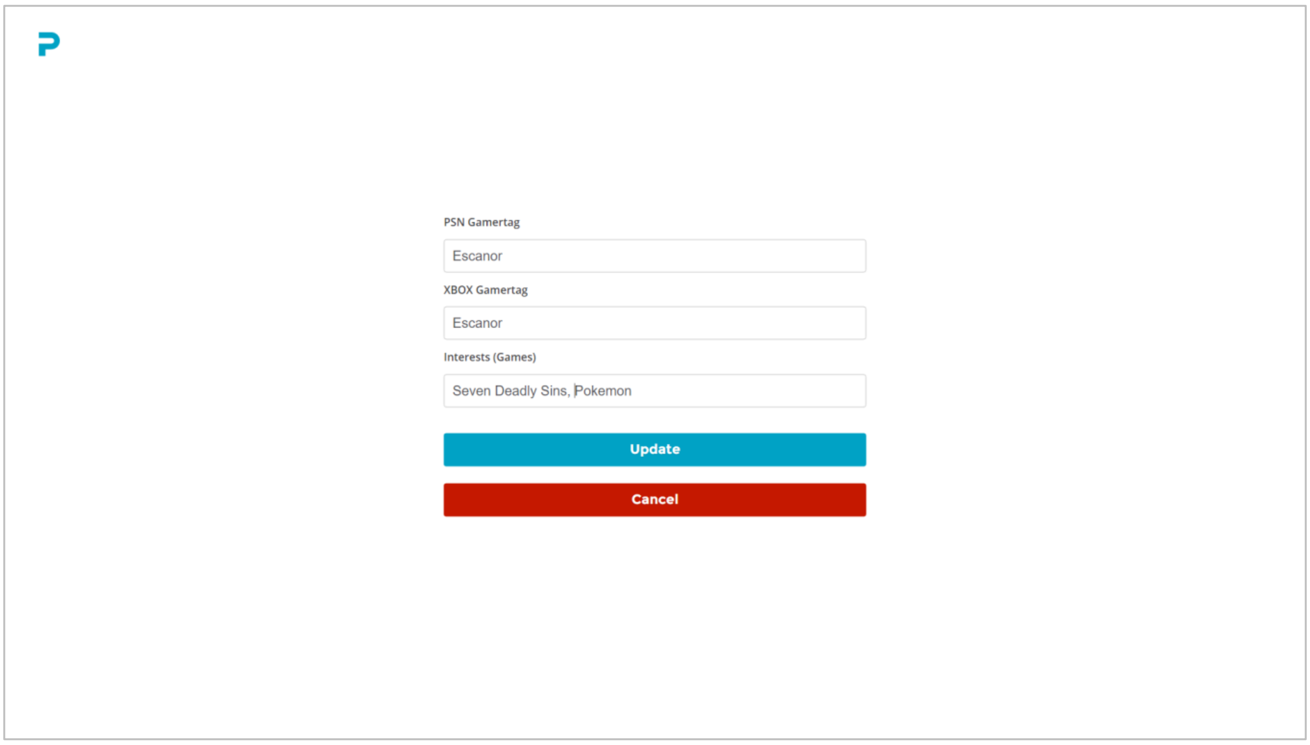
Ban
Cool

Ban
Ban2016

meliodas
Cool let's get playing!

meliodas
Hmmm

Figure 22: Message Window View


PSN Gamertag

Escanor

XBOX Gamertag

Escanor

Interests (Games)

Seven Deadly Sins, Pokemon

Figure 23: Edit Profile View