**NYU Polytechnic School of Engineering**

**Computer Science and Engineering**

**eHarbinger**

**Software Design Documentation**

**Version 1.1**

**Document Number: SDD-002**

Project Team B3

Project Team Members:

Meghan Clark mc4677

Brian Marks bm1549

Priyam Nidhi pn613

Benson Tsai bt773

**REVIEWS AND APPROVALS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Printed Name and Title** | **Function (Author, Reviewer, Approval)** | **Date** | **Signature** |
| Brian Marks | Author | October 25, 2015 | Signature on File |
| Meghan Clark | Author | October 25, 2015 | Signature on File |
| Priyam Nidhi | Author | October 25, 2015 | Signature on File |
| Benson Tsai | Author | October 25, 2015 | Signature on File |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision Level**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision Number** | **Purpose** |
| October 25, 2015 | Version 1.1 | Initial Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. INTRODUCTION 6](#_Toc434227233)

[1.1 PURPOSE 6](#_Toc434227234)

[1.2 SCOPE 6](#_Toc434227235)

[1.3 IDENTIFICATION 6](#_Toc434227236)

[1.4 DOCUMENT SUMMARY 6](#_Toc434227237)

[1.5 SYSTEM OVERVIEW 7](#_Toc434227238)

[1.6 DOCUMENT OVERVIEW 7](#_Toc434227239)

[2. REFERENCE DOCUMENTS 9](#_Toc434227240)

[3. SYSTEM WIDE DESIGN DECISIONS 10](#_Toc434227241)

[3.1 SOFTWARE COMPONENT ARCHITECTURAL DESIGN 10](#_Toc434227242)

[3.2 SOFTWARE ARCHITECTURE GENERAL DESCRIPTION 10](#_Toc434227243)

[3.3 SOFTWARE ITEM COMPONENTS 10](#_Toc434227244)

[3.4 COMPONENT INTERFACE IDENTIFICATION 11](#_Toc434227245)

[3.5 SOFTWARE COMPONENT CONCEPT OF EXECUTION 12](#_Toc434227246)

[4. SOFTWARE ITEM DETAILED DESIGN 12](#_Toc434227247)

[4.1 STRUCTURE 12](#_Toc434227248)

[4.1.1 Software Unit Detailed Design 12](#_Toc434227249)

[4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT 14](#_Toc434227250)

[4.2.1 Run-time Object Instances 14](#_Toc434227251)

[4.3 BEHAVIOR 15](#_Toc434227252)

[4.3.1 Interaction Diagrams 15](#_Toc434227253)

[4.3.2 Collaboration Diagrams 16](#_Toc434227254)

[4.3.3 Activity Diagrams 17](#_Toc434227255)

[4.4 CONCEPT OF EXECUTION 17](#_Toc434227256)

[4.5 INTERFACE DESIGN 18](#_Toc434227257)

[4.5.1 Interface Identification and Diagrams 18](#_Toc434227258)

[4.5.2 Unique identifier of Interface 19](#_Toc434227259)

[6. DEPLOYMENT ARCHITECTURE 19](#_Toc434227260)

[6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM 19](#_Toc434227261)

[7. DICTIONARIES 20](#_Toc434227262)

[8. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION 20](#_Toc434227263)

[9. REQUIREMENTS TRACEABILITY 21](#_Toc434227264)

[9.1 SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY 21](#_Toc434227265)

[10. SYSTEM DESIGN TESTING 22](#_Toc434227266)

[11. RATIONALE 23](#_Toc434227267)

[12. NOTES 24](#_Toc434227268)

[13. APPENDICES 24](#_Toc434227269)

[13.1 DICTIONARIES 24](#_Toc434227270)

[13.2 UML DIAGRAMS 31](#_Toc434227271)

[13.3 SCHEDULE TRACKING 31](#_Toc434227272)

# 1. INTRODUCTION

## 1.1 PURPOSE

The purpose of this Software Design Description is to define the design contents of eHarbinger’s software system. The intended audience of this document is developers of the product where the architecture of the software project is discussed. This information is used to communicate overall quantitative and qualitative system characteristics to operations management, technical support, training, and operators.

## 1.2 SCOPE

This document is used to define the architecture of software-only products. Since eHarbinger is a web based service, there are no hardware constraints to consider in this release. The architectural layout and the infrastructural support required to develop the application are discussed.

The project is aimed at communities of gamers who want to play with other like-minded gamers. This product will be aimed at users who play video games but are not a part of a gaming community. Since the proposed target group is vast, the designers must make sure that the software we create is easy to use and applicable to many groups of people.

## 1.3 IDENTIFICATION

This document is the Software Design Description (SDD-002, Initial Release) with a single revision at this stage.

## 1.4 DOCUMENT SUMMARY

The eHarbinger system aims to bring users together who have similar interests in gaming so that the gaming experience can be more enjoyable. This product will be aimed at users who play video games but are not a part of a gaming community. The project priority is high and agile lifecycle would be used to meet these user-driven functionalities.

In this document, the basic structure of the user profiles, questionnaire, chat and their home page is discussed. The framework for these will allow the developers to smoothly translate these architectural designs into code.

## 1.5 SYSTEM OVERVIEW

The product function is to match users with other users based on their compatibility in various video games. The primary software components discussed in this document are: the product will contain a user profile with publicly available information about the users, for users that have been matched together, a chat will be available for the users to communicate, and there will be a section private to each user where he/she may answer questions. The responses to these questions will determine how the users are matched together.

The users answer a questionnaire upon joining and information obtained from the questionnaire is used to match them with others. The algorithm used to match users is laid out to retrieve the correct values from the database. This document will aid the developers in understanding the architectural layout to do so.

The majority of the product will be coded modularly. By doing so, changes can easily be made without having a negative shift in performance in other areas. Dependencies would include linking profiles with external sources, such as Facebook and Steam. If the API changes for these sites, a fix would need to be applied. Furthermore, we need to keep our list of games up to date so that users can find other users for new games.

## 1.6 DOCUMENT OVERVIEW

The project follows agile lifecycle and the data dictionaries, collaboration and activity diagrams are object oriented. The subsequent portions of this document describe the project break down:

Section 2 provides references to Proposal, SPMP, RAS, and SDD(with Code).

Section 3 discusses the system wide design decisions which include the software components design and execution for this project.

Section 4 lays out the software item detailed design interaction, collaboration and activity diagrams. The interface design is also discussed in the section.

Section 5 is not required in this document.

Section 6 covers the deployment architecture with the required diagram.

Section 7 has the system dictionary for the components.

Section 8 covers the software item used and their resource utilization on the computer.

Section 9 describes how every requirement is accounted for in the documentation and

system implementation.

Section 10 covers the mechanism for system design testing.

Section 11 describes the rationale behind the project.

Section 12 provides additional notes.

Section 13 is the appendix with schedule and defect tracking.

# 2. REFERENCE DOCUMENTS

|  |  |  |  |
| --- | --- | --- | --- |
| Title | Author | Version | Date |
| Project proposal | Project Team | v1.1 | 09/16/2015 |
| Software Project Management Plan (SPMP) | Project Team | v1.1 | 09/23/2015 |
| Requirements/Analysis Documentation (RAS) | Project Team | v2.0 | 10/10/2015 |
| Design Description (SDD) - Initial | Project Team | v1.1 | 10/25/2015 |
| Design Document Final (with Code) | Project Team | v1.2 | 11/18/2015 |

# 3. SYSTEM WIDE DESIGN DECISIONS

## 3.1 SOFTWARE COMPONENT ARCHITECTURAL DESIGN

Customer

Accesses Website

Back-End of Website

Database

Access Info

Returns Info

Front-End of Website

Populates

Sends Information to Browser

## 3.2 SOFTWARE ARCHITECTURE GENERAL DESCRIPTION

When a customer accesses our website using the URL, the customer will get a response from the website containing the web page they requested. The way this works is as follows. Back-End of website will get the request from the customer and will access any necessary information from the Database. The Database will return the information to the Back-End and the Back-End will populate information into the Front-End of the website. Then, the Front-End of the website will send the entire web page to the customer through their browser.

## 3.3 SOFTWARE ITEM COMPONENTS

Looking at the software item components, the breakdown will done in 3 layers:

**1. Front End/User Interface**

Type: Web page

Description: This is the initial home page presented to the user upon connection with eHarbinger’s server. There is a grid to list out the recent games they are interested in playing and the desired skill level which would lead them to another web page.

Name: Select(), Display()

Pre-condition: Connected to site and is signed in.

Post-condition: On another page showing the matched gamers.

Exceptions: If the user is not registered, they are prompted to sign in instead.

Flow of Events:

1. User is presented with home page

2. User updates his gaming profile.

3. User is then listed out other gamers who match his specifications.

**2. Server Hosting Web Application**

Type: Apache

Descriptions: The eHarbinger web handler will be Apache because it is a well-known versatile web handler.

Attributes: Allows connection to database and handling for incoming web requests.

Operations: It helps manage the incoming connections and respond to the requests made by the gamers.

**3. Database/Storage of Gamers Information**

Type: Database

Descriptions: The eHarbinger database will reside on our server. The database will be managed in PostgreSQL.

Attributes: Name, Password, Games Played, Skill Level, Users Connected to Gamer, Chat Messages, Archived messages

Operations: New() to add new records to the database

Replace() to change an existing record.

Search() where the eHarbinger server searches the Database with a particular record and presents it to the client.

The team would use LAMP stack to run the dynamic web site and server. Since PHP, which is being used for the user interface, is consistent with LAMP, this would yield high performance and a reliable platform for the software components.

## 3.4 COMPONENT INTERFACE IDENTIFICATION

Customer interacts directly with the front end of the website, which formats and provides actions for customer to complete. Front end invokes methods in the back end.

The back end acts as the middle agent between the front end and database. When a method is invoked, the back end either: processes the data to a format accepted by the database and feeds that data into appropriate procedure in the database for information entry, or call appropriate procedure to retrieve the data requested by the method, format the raw data, and return it to the front end.

The database stores all user information. When a procedure is called, it accepts the input and enter it in a corresponding schema in database, or returns information requested by the back end.

## 3.5 SOFTWARE COMPONENT CONCEPT OF EXECUTION

User profile, presented as a web page, is a front-end component and interacts with the user. Information is presented to the user through text and images. Content of the information is dependant on previous request received by the back-end server. User requests are relayed by user profile, through buttons and links, by calling methods in the back-end modules. User profile primarily initiates a request.

The server handles and processes information to and from user profile and database. When a method in the back-end module is called, back-end server processes data provided by the user and relays it to the suitable database procedure. If return data is requested, the server also processes the data to a user-friendly form and relays it back to the user profile. Server primarily accepts requests from user profile only and initiates requests to database only.

The database stores all information of the users. Input is accepted through procedures, which specifies which schema to be accessed, and what values to return to the caller. Database primarily accepts requests from back-end only.

# 4. SOFTWARE ITEM DETAILED DESIGN

## 4.1 STRUCTURE

### 4.1.1 Software Unit Detailed Design

|  |  |
| --- | --- |
| Name: | Directory |
| Attributes: | userLoc - Pointer Variable to User information in Database |
| Methods: | getUserName - gets the user’s real name and username  getUserGames - gets user’s game preferences  getUserSkill - gets the user’s skill level in gaming  chatWithUser- alerts user that another user wishes to chat  updateUser - updates any information needed in the database from the user class  deleteUser - deletes user profile  createUser - creates user profile |

|  |  |
| --- | --- |
| Name: | User |
| Attributes: | fullName - user’s first and last name  userName- user’s public username for contacting  password- User’s password for profile  gamingPrefrence- platforms/games user prefers to play  gamingSkills - skill level of user  feedbackLevel - number of thumbs up and thumbs down user has received |
| Methods: | getFullName - gets user’s full name  getUserName - gets user’s username  getUserGames - gets user’s game preferences  getUserSkill - gets the user’s skill level in gaming  updateFullName - changes full name of user  updateUserName - changes username  updatePassword - changes password  updatePreferences - change game preferences  updateSkills - change skill level  updateFeedback - updates feedback of user  setFullName - sets full name of user  setEmail - sets email of user  setUserName - sets username  setPassword - sets password  setPreferences - sets preferences of user  setSkills - sets skill level of user |

|  |  |
| --- | --- |
| Name: | Customer |
| Attributes: | userLoc - Pointer Variable to user profile |
| Methods: | createProfile - allows customer to create Profile  viewProfile - allows customer to view other user’s profiles  giveFeedback - allows customer to give feedback to other users  chatUser - allows customer to chat with other user  updateProfile - updates profile information  deleteProfile - deletes user’s profile |

## 4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT

### 4.2.1 Run-time Object Instances

At this stage, we will only be using a single-thread mechanism. Apache can handle multiple requests, but that is outside of the scope of our project. The requests will be handled by Apache in a FIFO fashion.

Apache

PostGres

Query

Returns

HTML/CSS/JS

PHP

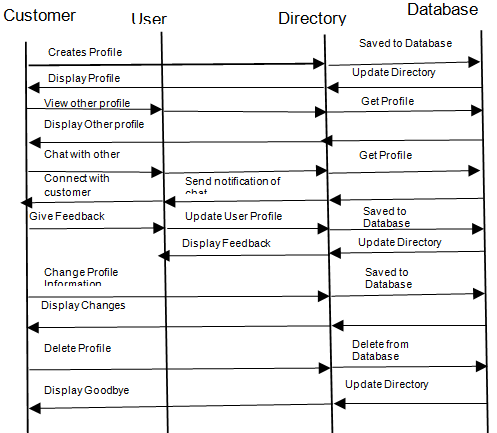
Sends Information to Browser

Browser (External to product)

URL

## 4.3 BEHAVIOR

### 4.3.1 Interaction Diagrams



### 4.3.2 Collaboration Diagrams

Customer

User

Directory

getUserName(); getUserGames(); getUserSkill(); updateUser(); deleteUser(); createUser()

chatWithUser();

getUserName(); getUserGames(); getUserSkill(); updateUser(); deleteUser(); createUser()

createProfile();

viewProfile();

updateProfile();

deleteProfile();

giveFeedback()

chatUser();

giverFeedback();

chatUser();

getFullName();

getUserName();

getUserSkill();

updateFullName();

updateUserName();

updatePassword();

updatePrederences()

updateSkills();

updateFeedback();

setEmail();

setFullName();

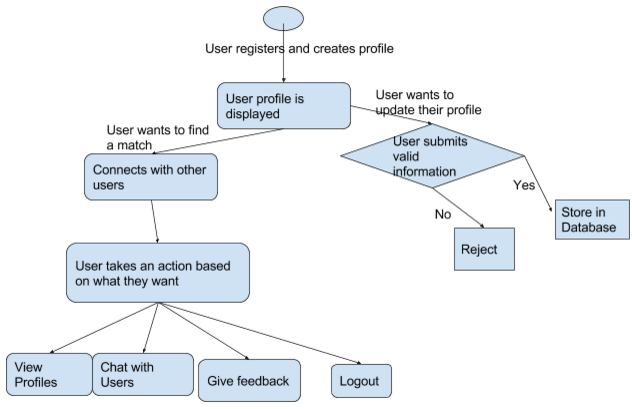
setUserName();

setPassword();

setPreferences();

setSkills();

### 4.3.3 Activity Diagrams



## 4.4 CONCEPT OF EXECUTION

Customer-User: Customer is the user that is currently logged in to the server. Between two users, two functions are available; chat and feedback.

Customer-Directory: This is the main form of communication of the website. Customers click on links and buttons to view and update their profiles, view other users’ profiles, skills and games, and add games to their profiles.

Directory-User: This part of execution is essential but hidden behind the scene. Directory communicate with user to gather information for gamer matching, as well as return data from requests in Customer-Directory communication.

## 4.5 INTERFACE DESIGN

### 4.5.1 Interface Identification and Diagrams

Database

Customer

(Front-End of Website)

User/Directory

(Back-End of Website)

Links

Methods

Procedures

Webpages

giveFeedback()

chatUser()

getFullName()

getUserName()

getUserSkill()

updateFullName()

updateUserName()

updatePassword()

updatePrederences()

updateSkills()

updateFeedback()

setEmail()

setFullName()

setUserName()

setPassword()

setPreferences()

setSkills()

getUserGames() updateUser()

deleteUser() createUser()

createProfile()

viewProfile()

updateProfile()

deleteProfile()

add()

delete()

edit()

get()

### 4.5.2 Unique identifier of Interface

***Directory***

get(): takes string specifying the user and field to access in the database. Returns field value, a string or integer, to the caller.

update(): takes string specifying the user and field to update in the database. Returns status code, indicating successful update or not, to the caller.

create():takes string specifying the type of profile entry to create in the database. Returns status code, a single integer, indicating successful creation or not, to the caller.

delete():takes string specifying the entry to delete in the database. Returns status code, a single integer, indicating successful deletion or not, to the caller.

chat(): takes string specifying the user to chat with. Returns status code, a single integer, indicating successful transmission of message to the user, to the caller.

Database

add(): takes string specifying the schema to add entry to. Returns status code, a single integer, indicating success addition of the entry to the caller.

edit():takes string specifying the schema and entry to edit. Returns status code, a single integer, indicating success addition of the entry to the caller.

delete():takes string specifying the schema to delete entry from. Returns status code, a single integer, indicating success addition of the entry to the caller.

get():takes string specifying the schema to access entry from. Returns the entry to the caller.

# 6. DEPLOYMENT ARCHITECTURE

## 6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM

This diagram describes how this web-based architecture would provide the required functionalities to the user. It models the physical deployment of artifacts on the nodes.

<<device>>

Web Server

<<artifact>>

Website

<<device>>

DB Server

<<device>>

User Client

<<device>>

Browser

<<artifact>>

HTML5

<<artifact>>

PostgreSQL

HTTP: Port 80

# 7. DICTIONARIES

The dictionaries include all classes, methods, attributes, messages, events and their association. Section 13.1 contains a detailed breakdown of the classes and attributes required at this stage of eHarbinger design.

# 8. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION

This section of the document will be used to emphasize how each software aspect will be critical towards the product.

Each part of the system connects to one another to ensure that everything runs smoothly together. However each part individually holds a key part that is utilized by the system. The server will be hosting the website and information, which is key to the project because, without the server, nothing would be available for the users.

Another key factor in this project is the database. The database itself will hold all the information about each user’s profile that they put into the website as well as any history of interaction between users. This history will be important for knowing which users have played or chatted with other users as users can only give feedback to other users that they have previously played or chatted with. We will also hold each person’s profile if they choose to deactivate the account for a set period of time in case a user decided in the future that they still wanted to use the website.

The last key part of the system is the design of the website. The design of the website must be simple enough to be able to be easily navigable by the user while also having an eye-catching design to make users want to continue using the service. The website is also the only aspect of the system that the user will actually be seeing while the rest of the key parts of the system are working behind the scenes. Therefore the website design must be created with the customer in mind, thinking what a gamer would want to see in this “match making” website and testing the project on actual gamers who would want to use the product.

# 9. REQUIREMENTS TRACEABILITY

## 9.1 SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY

This section is to provide information about the reasoning for certain requirements in relation to the artifacts the project contains. Included is a description of the requirements and related dependencies to be able to track software changes to the requirements.

1. Since this project will be focused around an online market, all operating systems will be supported as long as they have a current web browser. The connections will be established through the standard port number 80 for http connections.

2. For the system to provide up to date information on the newest games and user matches, the database will be updated weekly and will be cross checked with the list of popular multi-player games found online.

3. On the user interface, there will be considerable changes after user feedback is received for their profiles. Some users would suggest different avatars that they would like for their profiles which would have to be hosted on the server so that when a request is made for these, an appropriate response retrieves the given software component.

# 10. SYSTEM DESIGN TESTING

The system design testing would provide knowledge on how closely the requirements map out with the data retrieved and its representation on the front end. We would include graphical user interface testing to assure that the web layout matches the user stories. Since this is a user-centered design, we would also include usability testing in the final stages of this project.

The UI of this project will mostly be tested through our group of potential customers as we want to ensure that our product will not only be easy to use but also be pleasing to look at. Through testing the UI, in turn we will also end up testing the UX. Our testing group will determine if it is possible to complete any action on the website that was said to be able to done by our team and tell us if there are any additional features they might find helpful for the website.

In the back-end, because of its special status of a middle agent, a three-step testing will be used.

First, the interface between front-end and back-end will be tested to ensure correct methods in the back-end are called by the front-end. This will be validated by examining the input provided to the back-end, and checking data displayed in the front-end.

Second, the interface between back-end and database will be tested to ensure correct procedure is called in the database. This will be validated by examining the database entries, and data returned to the front-end.

Finally, testing will be done in a full flow from the front-end to back-end, validated by user input in the front-end and resulting data entry in the back-end.

To test the database schema and stored procedures, a three-part testing scheme will be used.

In part one, the schema will be closely evaluated by the team to ensure that all needed fields are present and that no erroneous information is stored in the database. We will use Boyce Codd normal form in order to keep the database size small and to allow queries to run quickly.

In part two, the stored procedures will be tested individually against a set of test data where the output is known. Each procedure must run within a reasonable time and return the correct result. For any procedures that return unexpected information, the team as a whole will analyze the procedure for faults.

In part three, the database schema and stored procedures will be tested in a complete product. Each of the use cases will serve as a particular test. If all use cases work as expected, the database schema and stored procedure testing will be considered complete and ready to move on to production. If any use cases do not work as expected, the team as a whole will analyze the schema, stored procedures, as well as the back-end and front-end code to find where the fault(s) exist.

# 11. RATIONALE

The design of the product was completed with the customer in mind. Some constraints have also been analyzed in more detail in this document. As laid out in the project proposal, many gamers wish to play with other like-minded gamers and our product fills that particular niche. A user can answer questions such as how competitive they are, how competitive they want their teammates to be, as well as their skill level and the skill level they expect of their teammates. These questions, along with other similar questions, helps us pair users up with other users that will get along together. We will accomplish this goal using a PostGres relational database, as described earlier.

The product will be created through object-oriented Programming. In order to have all the workflows for the product, each object will need to be put in a class. For example, to allow a gamer to look through our database of user profiles, we must have an object in the code that is able to perform his actions. These methods will be based on the context diagram and will change if either the requirements are changed from the client or the team discovers an addition is required. The best choice for an object-oriented web friendly language would be PHP, so much of our back-end design will be done using PHP.

Many times, people deciding to play together randomly doesn’t go very well. There is a lot of trash talking and there aren’t many expectations of each other. eHarbinger gives the users a framework to communicate and collaborate and be able to meet not only gaming buddies, but possibly life-long friends based on their compatibility. The compatibility will be generated using information stored in our database related to their ratings and answers to questions, information all stored in our database.

We will update our list of games daily in order to not exclude any new games and we will have users continue to answer more and more questions to keep their matches as relevant as possible. As mentioned above, the more questions a user asks, the easier it is for us to match users together. We will implement an easy to use interface that everyone can figure out upon a glance. This is important since our product is a user-centered product. We will accomplish this using a daily task using PHP in a Crontab.

After thorough analysis, as well as feedback from potential customers, we believe there exists great demand for such a system, and the need has not yet been fulfilled. We are confident this project is feasible and will be well-received by the market when it comes to fruition.

# 12. NOTES

None in this release.

# 13. APPENDICES

## 13.1 DICTIONARIES

Dictionaries for each class, method, and attribute from Section 6.11 are a part of this document.

**Classes**:

|  |  |  |  |
| --- | --- | --- | --- |
| Directory | A collection of user profiles to access by other users. | userLoc; | getUserName();  getUserGames();  getUserSkill();  chatWithUser();  updateUser();  deleteUser();  createUser(); |

|  |  |  |  |
| --- | --- | --- | --- |
| User | Controls actions of users who aren’t the current user on the device. | fullName;  userName;  password;  gamingPreferences;  gamingSkills;  feedbackLevel; | getFullName();  getUserName();  getUserSkill();  updateFullName();  updateUserName();  updatePassword();  updatePreferences();  updateSkills();  updateFeedback();  setFullName();  setEmail();  setUserName();  setPassword();  setPreferences();  setSkills(); |

|  |  |  |  |
| --- | --- | --- | --- |
| Customer | The holding place for the customer’s information while they navigate through the site. | userLoc; | createProfile();  viewProfile();  giveFeedback();  chatUser();  updateProfile();  deleteProfile(); |

**Methods**:

|  |  |  |  |
| --- | --- | --- | --- |
| getUserName | Gets the username the user has created for account. | email(needs email to connect to username) | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| getUserGames | Gets the user’s game preferences and platforms | userName(user that needs game preferences to display) | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| getUserSkill | Gets the user’s skill level | userName(user that needs game skill displayed) | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| chatWithUser | Alerts user a customer wishes to chat with them | user.userName(username of who to alert)  customer.userName(username of customer who wishes to chat | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| updateUser | Updates information customer wishes to update | userName(user that wants to update information) | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| deleteUser | Deletes user from database | userName(user that wishes to delete profile) | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| createUser | Creates new profile of user | email,userName, password, fullName, gamePreferences, skills | Directory |

|  |  |  |  |
| --- | --- | --- | --- |
| getFullName | Displays full name. | userName(user who needs full name displayed) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| getUserName | Displays username. | email(user who needs username displayed( | User |

|  |  |  |  |
| --- | --- | --- | --- |
| getUserSkill | Displays skill level | userName(user who needs skill level displayed) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updateFullName | Changes first and last name | userName(user who needs to change name) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updateUserName | Changes username | email(user who needs to change username) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updatePassword | Changes password | userName(user who needs to change password) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updatePreferences | Changes game and platform preferences | userName(user who needs to change preferences) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updateSkills | Changes skill level | userName(user who needs to change skill level) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| updateFeedback | Changes feedback level | userName(user who needs to change feedback) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| setFullName | Sets first and last name | email(user who needs to set name) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| setUserName | Sets username of user | email(user who needs to set username) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| setPassword | Sets password | email(user who needs to set password) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| setPreferences | Sets game and platform preferences | email(user who needs to set preferences) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| setSkills | Sets skill level | email(user who needs to set skill level) | User |

|  |  |  |  |
| --- | --- | --- | --- |
| createProfile | Creates profile through User class methods | email(user who needs to create profile) | Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| viewProfile | Displays profile information through user and directory class | userName(user who needs to display information) | Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| giveFeedback | Changes feedback level of another user | userName(user whose feedback will be changed)  feedbackLevelChange(whether the feedback will go up or down) | Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| chatUser | Creates chat with another user | userName(wanted user to chat with) | Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| updateProfile | Updates profile through User class methods | userName(user who needs to change profile) | Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| deleteProfile | Deletes profile | userName(user who needs to delete profile) | Customer |

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| userLoc | Pointer Variable to current user’s information in directory | Simple | Directory  Customer |

|  |  |  |  |
| --- | --- | --- | --- |
| fullName | Pointer Variable to current user’s full name | Simple | User |

|  |  |  |  |
| --- | --- | --- | --- |
| userName | Pointer Variable to current user’s username | Simple | User |

|  |  |  |  |
| --- | --- | --- | --- |
| password | Pointer Variable to current user’s password | Simple | User |

|  |  |  |  |
| --- | --- | --- | --- |
| gamingPreferences | Array of games and platforms current user prefers | Complex | User |

|  |  |  |  |
| --- | --- | --- | --- |
| gamingSkills | Array of Skills in different games | Complex | User |

|  |  |  |  |
| --- | --- | --- | --- |
| feedbackLevel | Level of positive or negative feedback | Simple | User |

## 13.2 UML DIAGRAMS

All UML diagrams are included in their respective sections in this document except for the code. The code will be separately included in the document in the next version of this document.

## 13.3 SCHEDULE TRACKING

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated  (Hours) | Actual | Difference |
| SPMP | Brian | 5 | 4 | 1 |
| SPMP | Priyam | 2 | 4 | 2 |
| SPMP | Meghan | 3 | 2 | 1 |
| SPMP | Benson | 3 | 4 | 1 |
| SPMP | Team | 13 | 14 | 5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated  (Hours) | Actual | Difference |
| RAS | Brian | 3 | 3 | 0 |
| RAS | Priyam | 3 | 3 | 0 |
| RAS | Meghan | 5 | 5 | 0 |
| RAS | Benson | 3 | 3 | 0 |
| RAS | Team | 14 | 14 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated  (Hours) | Actual | Difference |
| SDD (INITIAL) | Brian | 3 | 2 | 1 |
| SDD (INITIAL) | Priyam | 3 | 3 | 0 |
| SDD (INITIAL) | Meghan | 3 | 4 | 1 |
| SDD (INITIAL) | Benson | 3 | 3 | 0 |
| SDD (INITIAL) | Team | 12 | 12 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated  (Hours) | Actual | Difference |
| SDD (FINAL) | Brian | 2 | N/A | N/A |
| SDD (FINAL) | Priyam | 2 | N/A | N/A |
| SDD (FINAL) | Meghan | 2 | N/A | N/A |
| SDD (FINAL) | Benson | 2 | N/A | N/A |
| SDD (FINAL) | Team | 2 | N/A | N/A |

Cumulative\*

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated  (Hours) | Actual | Difference |
| Brian | 11 | 9 | 2 |
| Priyam | 8 | 10 | 2 |
| Meghan | 11 | 11 | 0 |
| Benson | 9 | 10 | 1 |
| Team | 29 | 40 | 5 |

\*Excludes SDD Final

13.4 DEFECT TRACKING

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated | Actual | Difference |
| SPMP | Brian | 3 | 3 | 0 |
| SPMP | Priyam | 4 | 4 | 0 |
| SPMP | Meghan | 7 | 7 | 0 |
| SPMP | Benson | 5 | 5 | 0 |
| SPMP | Team | 19 | 19 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated | Actual | Difference |
| RAS | Brian | 2 | 2 | 0 |
| RAS | Priyam | 4 | 4 | 0 |
| RAS | Meghan | 6 | 6 | 0 |
| RAS | Benson | 4 | 4 | 0 |
| RAS | Team | 16 | 16 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or  Deliverable | Who (individual  or Team) | Estimated | Actual | Difference |
| SDD (INITIAL) | Brian | 3 | 2 | 1 |
| SDD (INITIAL) | Priyam | 2 | 1 | 1 |
| SDD (INITIAL) | Meghan | 5 | 2 | 3 |
| SDD (INITIAL) | Benson | 3 | 4 | 1 |
| SDD (INITIAL) | Team | 13 | 9 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact 1or  Deliverable | Who (individual  or Team) | Estimated | Actual | Difference |
| SDD (FINAL) | Brian | 5 | N/A | N/A |
| SDD (FINAL) | Priyam | 3 | N/A | N/A |
| SDD (FINAL) | Meghan | 5 | N/A | N/A |
| SDD (FINAL) | Benson | 2 | N/A | N/A |
| SDD (FINAL) | Team | 15 | N/A | N/A |

Cumulative\*

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated | Actual | Difference |
| Brian | 8 | 7 | 1 |
| Priyam | 10 | 9 | 1 |
| Meghan | 13 | 15 | 2 |
| Benson | 13 | 17 | 4 |
| Team | 44 | 48 | 6 |

\*Excludes SDD Final