

**Computer Science and Engineering**

**DotaMatch**

**CS-UY-4523-Software Design Description (SDD)**

**Version 1.0**

Document Number: SDD-001

Project Team Number B11

Project Team Members:

Corey Chong cc3899

Albert Su as7353

Evans Yeung eby210

**REVIEW AND APPROVALS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Printed Name and Title** | **Function (Author, Reviewer, Approval)** | **Date** | **Signature** |
| Evans Yeung | Author | October 8, 2016 | Evans Yeung |
| Corey Chong | Author | October 8, 2016 | Corey Chong |
| Albert Su | Author | October 8, 2016 | Albert Su |
| Evans Yeung | Reviewer | October 24, 2016 | Evans Yeung |
| Corey Chong | Reviewer | October 24, 2016 | Corey Chong |
| Albert Su | Reviewer | October 24, 2016 | Albert Su |
|  |  |  |  |

**REVISION LEVEL**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision Number** | **Purpose** |
| October 8, 2016 | Version 1.0 | Initial Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. INTRODUCTION 6](#_Toc465154349)

[1.1 Purpose 6](#_Toc465154350)

[1.2 Scope 6](#_Toc465154351)

[1.3 Identification 6](#_Toc465154352)

[1.4 Document Summary 6](#_Toc465154353)

[1.5 System Overview 7](#_Toc465154354)

[1.6 Document Overview 7](#_Toc465154355)

[2. REFERENCE DOCUMENTS 8](#_Toc465154356)

[3. SYSTEM WIDE DESIGN DECISIONS 8](#_Toc465154357)

[3.1 SOFTWARE COMPONENT ARCHITECTURAL DESIGN 8](#_Toc465154358)

[3.2 SOFTWARE ARCHITECTURE GENERAL DESCRIPTION 11](#_Toc465154359)

[3.3 SOFTWARE ITEM COMPONENTS 11](#_Toc465154360)

[3.4 COMPONENT INTERFACE IDENTIFICATION 12](#_Toc465154361)

[3.5 SOFTWARE COMPONENT CONCEPT OF EXECUTION 16](#_Toc465154362)

[4. SOFTWARE ITEM DETAILED DESIGN 17](#_Toc465154363)

[4.1 STRUCTURE 17](#_Toc465154364)

[4.1.1 Software Unit Detailed Design 17](#_Toc465154365)

[4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT 19](#_Toc465154366)

[4.2.1 Run-time Object Instances 20](#_Toc465154367)

[4.3 BEHAVIOR 20](#_Toc465154368)

[4.3.1 Sequence Diagrams 20](#_Toc465154369)

[4.3.2 Collaboration Diagram 20](#_Toc465154370)

[4.3.3 Activity Diagrams 20](#_Toc465154371)

[4.4 CONCEPT OF EXECUTION 22](#_Toc465154372)

[4.5 INTERFACE DESIGN 23](#_Toc465154373)

[4.5.1 Unique Identifier of Interface 23](#_Toc465154374)

[4.5.2 Interface Identification and Diagrams 23](#_Toc465154375)

[5. IMPLEMENTATION ARCHITECTURE OF (NOT REQUIRED) 24](#_Toc465154376)

[5.1 ALL ACTIVE AND PASSIVE CLASSES ASSIGNED TO COMPONENTS 24](#_Toc465154377)

[5.2 DIAGRAM OF PHYSICAL PACKAGING OF LOGICAL COMPONENTS 24](#_Toc465154378)

[6. DEPLOYMENT ARCHITECTURE 24](#_Toc465154379)

[6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM 24](#_Toc465154380)

[7. CODE (PSEUDO) 24](#_Toc465154381)

[8. DICTIONARIES 25](#_Toc465154382)

[9. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION 27](#_Toc465154383)

[10. REQUIREMENTS TRACEABILITY 28](#_Toc465154384)

[10.1 SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY 28](#_Toc465154385)

[11. SYSTEM DESIGN TESTING 28](#_Toc465154386)

[12. RATIONALE 28](#_Toc465154387)

[13. NOTES 28](#_Toc465154388)

[14. APPENDICES 29](#_Toc465154389)

[14.1 DICTIONARIES 29](#_Toc465154390)

[14.2 UML DIAGRAMS 29](#_Toc465154391)

[14.3. SCHEDULE TRACKING 29](#_Toc465154392)

[14.4 DEFECT TRACKING 31](#_Toc465154393)

[14.5 GANTT CHART 33](#_Toc465154394)

# 1. INTRODUCTION

## 1.1 Purpose

DotaMatch is a matchmaking application that will make the process of finding fellow players a more fun and enjoyable experience by introducing other factors besides skill into the equation. Users will be able to set their own preferences for the type of players they would like to be matched with.

The purpose of this document is to design the features of the system. This includes how the system and its features work. This document is intended for the developers so they will understand how to design the application. Architecture diagrams and specification of function will aid the software development team.

## 1.2 Scope

DotaMatch is a matchmaking application that will make the process of finding fellow players a more fun and enjoyable experience by introducing other factors besides skill into the equation. Users will be able to set their own preferences for the type of players they would like to be matched with. Our algorithm takes into account these preferences and other factors to decisively match players in the database. Once matchmaking is complete, the user will be presented with a matched player’s profile which includes: in-game statistics, preferred role, favorite characters, etc. A player rating system is implemented so that the user will be able to rate the players they have played with. Overall this application is designed with the user’s freedom of choice in mind in order to create a positive experience for all of our users.

## 1.3 Identification

Software Design Description, SDD-001, Version 1.0, October 8, 2016

## 1.4 Document Summary

The purpose of the Software Design Document (SDD) is a written description of a software product. The SDD provides the software development team an overall guidance to the architecture of the software project. The SDD will contain multiple architecture diagrams and specifications of functions.

## 1.5 System Overview

DotaMatch is a matchmaking application that will make the process of finding fellow players a more fun and enjoyable experience by introducing other factors besides skill into the equation. Users will be able to set their own preferences for the type of players they would like to be matched with. Our algorithm takes into account these preferences and other factors to decisively match players in the database. Once matchmaking is complete, the user will be presented with a matched player’s profile which includes: in-game statistics, preferred role, favorite characters, etc. A player rating system is implemented so that the user will be able to rate the players they have played with. Overall this application is designed with the user’s freedom of choice in mind in order to create a positive experience for all of our users.

## 1.6 Document Overview

* Section 1 contains the purpose of the project and this document. It also contains a summary of the document and the system.
* Section 2 contains the scope of the product’s lifecycle including the document milestones.
* Section 3 contains the reference documents.
* Section 4 contains business requirements, including market considerations, risks, and resources.
* Section 5 contains the implementation architecture. This includes the active and passive classes and diagrams of physical packaging for each component.
* Section 6 contains the deployment architecture.
* Section 7 contains the pseudocode for the application.
* Section 8 contains the dictionary of terms needed to understand this document.
* Section 9 contains the software item computer resource utilization.
* Section 10 contains the requirements traceability for information provided in this document and previous documents.
* Section 11 contains the plans for system design testing.
* Section 12 contains the rationale.
* Section 13 contains notes that aid in understanding this document.
* Section 14 contains dictionaries, UML diagrams, schedule tracking, defect tracking, and Gantt charts.

# 2. REFERENCE DOCUMENTS

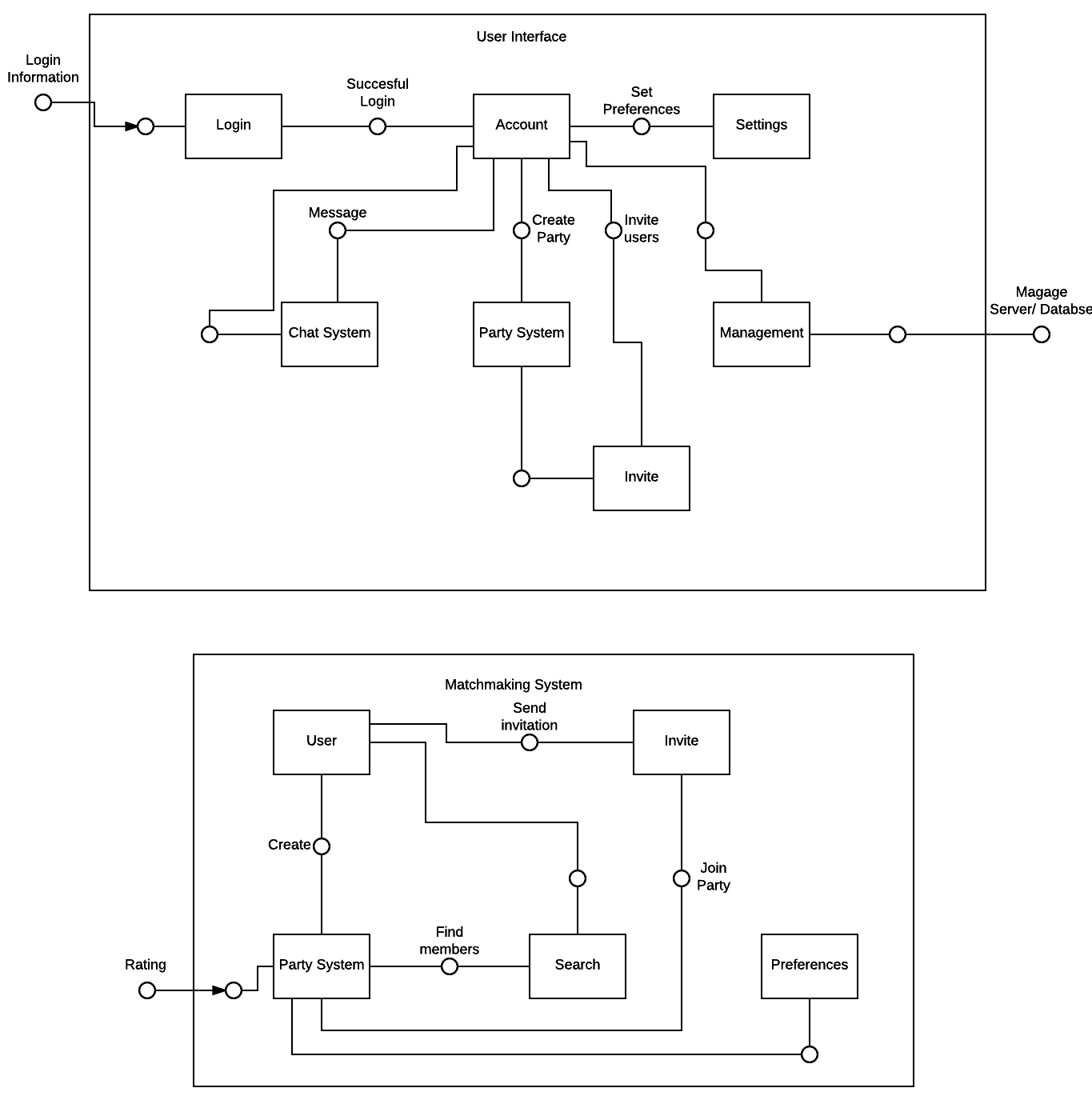
All documents are electronically submitted to NYU Classes and can be referenced through NYU Classes.

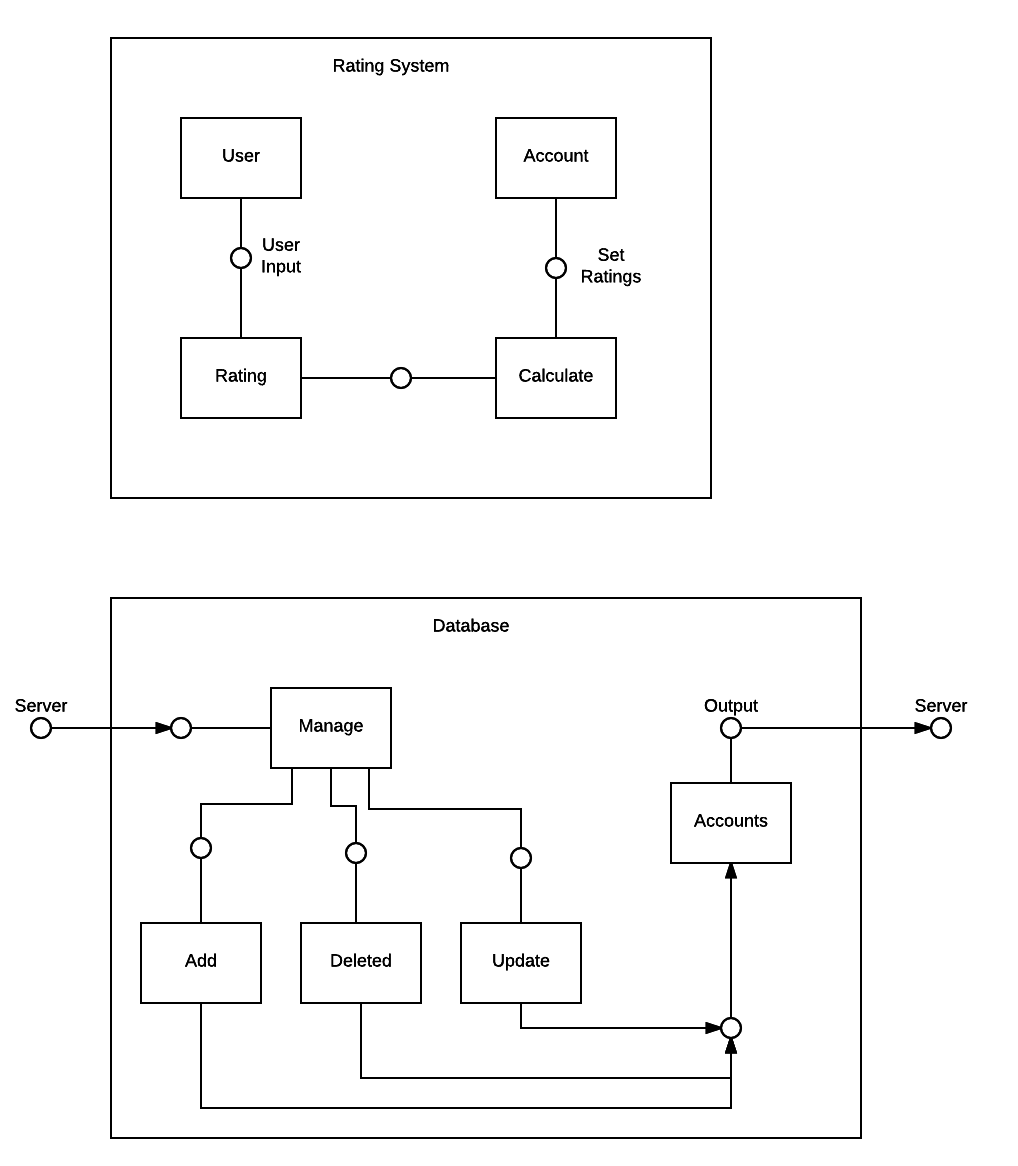
- Project Proposal was electronically submitted to NYU Classes on February 17, 2016.

* Software Requirements Specifications (SRS) Version 1.0 was electronically submitted to NYU Classes on March 7, 2016.
* Software Requirements Specifications (SRS) Version 2.0 was electronically submitted to NYU Classes on March 21, 2016.
* Software Project Management Plan (SPMP) Version 1.0 was electronically submitted to NYU Classes on April 6, 2016.
* Software Analysis Specification (SAS) Version 1.0 was electronically submitted to NYU Classes on April 18, 2016.
* Software Project Management Plan (SPMP) Version 2.0 was electronically submitted to NYU Classes on September 27, 2016.
* Requirements/Analysis Specification (RAS) Version 1.0 was electronically submitted to NYU Classes on October 7, 2016.
* Software Design Description Version 1.0 will be submitted to NYU Classes on October 25, 2016. Once submitted it can be referenced through NYU Classes. It can currently be referenced through Google Doc.

# 3. SYSTEM WIDE DESIGN DECISIONS

## 3.1 SOFTWARE COMPONENT ARCHITECTURAL DESIGN





## 3.2 SOFTWARE ARCHITECTURE GENERAL DESCRIPTION

The user will utilize the account (player profile) view as an entry point to access all other components of the application once their credentials have been validated. This will allow the user to interact with as few components as possible to access the desired utility or function. Should the user have a party assembled already, this architecture expedites the matchmaking experience (as intended).

There are three component architectures: User Interface, Server, and Database. The user interface contains seven lower level component architectures. The User Interface component contains all of the user functionality components. Users have to login to their accounts. From the account component, users can set preferences, message, create party, invite, and manage server and database the user is logged into a developer account. The second main component architecture is the server which includes the matchmaking and rating system. The matchmaking system has five lower level components. These include the user who has the ability to create a party and invite members. The party system can search for members using preferences and rating information. The rating system has four lower level components. This includes the user who has the ability to rate another user. This rating is then calculated and set into the user account data. The final main component is the database which has five lower level components. Input is received from the server and output goes back to the server. Users can manage the account database with add, delete, and update commands.

## 3.3 SOFTWARE ITEM COMPONENTS

**Login**: Fields to input credentials.

**Accoun**t: Interface that allows the user to view their profile in addition to giving access to most components of the application.

**Settings**: Group of menus that allow the user to modify their preferences.

**Chat System**: Object comprised of text fields for messaging members of the party or individual users.

**Party System**: Party management interface for the party members to mute, ignore other members, and an invitation/kick interface for the party leader.

**Management**: Server interface for the developers to access the database, allowing the developer to allocate or rescind account permissions regarding party/group accessibility.

**Invite**: A subcomponent of the Party System, extends the invitation function to the owner of groups and lobbies.

## 3.4 COMPONENT INTERFACE IDENTIFICATION

**User Interface**

**Login → Account**

* Name: “Successful Description”
* Description: Validation of user credentials from database
* Component 1: Login
* Component 2: Account

**Account → Settings**

* Name: “Account Settings”
* Description: Allow users to set preferences of prefer party members based on a series of questions. Also allow users to set account preferences.
* Component 1: Account
* Component 2: Setting

**Account → Chat System**

* Name: “Message Buddies”
* Description: Allow users to send a message to another user who is on their partner’s list.
* Component 1: Account
* Component 2: Chat System

**Account → Party System**

* Name: “Create Party”
* Description: Allow users to create a party to look for potential party members. The party creator becomes the party leader.
* Component 1: Account
* Component 2: Party System

**Party System → Invite**

* Name: “Invite Users”
* Description: Allow the party leader to invite users to the party.
* Component 1: Party System
* Component 2: Invite

**Account → Invite**

* Name: “Invite Partner”
* Description: Allow users to create a party along with their partners.
* Component 1: Account
* Component 2: Invite

**Account → Management**

* Name: “Manage Account Information”
* Description: Allow developers to manage user accounts.
* Component 1: Account
* Component 2: Management

**Matchmaking System**

**User → Party System**

* Name: “Create Party”
* Description: Allow users to create a party to look for potential party members. The party creator becomes the party leader.
* Component 1: User
* Component 2: Party System

**User → Search**

* Name: “Start Queue”
* Description: Party leader enters party into matchmaking queue. Looks for potential members based off of ratings and preset preferences.
* Component 1: User
* Component 2: Party System

**User → Invite**

* Name: “Invite Partner”
* Description: Allow users to invite their partners to the party.
* Component 1: User
* Component 2: Invite

**Party System → Preferences**

* Name: “Consider Preferences”
* Description: Party system will look at preset preferences to determine potential party members.
* Component 1: Party System
* Component 2: Preferences

**Rating System → Party System**

* Name: “Input Rating Data”
* Description: Rating system data will be imputed to the Party System. The Party System takes this data to determine potential party members.
* Component 1: Rating System
* Component 2: Party System

**Rating System**

**User → Rating**

* Name: “Rate Other User”
* Description: User will input rating data based off of a scale of 1-5 for other users. Rating will only be accepted for users inside the party system and once game has ended.
* Component 1: User
* Component 2: Rating System

**Rating → Calculated**

* Name: “Calculate Ratings”
* Description: The rating system will take the user inputted data and calculate the average from all party members.
* Component 1: Rating System
* Component 2: Calculate

**Calculate → Account**

* Name: “Update Account Ratings”
* Description: All calculated ratings will be added to an account’s existing rating. Update the account’s rating.
* Component 1: Calculate
* Component 2: Account

**Database**

**Server → Manage**

* Name: “Get Server Inputs”
* Description: Server inputs are received. Decide action based on input.
* Component 1: Server
* Component 2: Manage

**Manage → Add**

* Name: “Receive Additional User Input”
* Description: Take input from server for new user account information.
* Component 1: Manage
* Component 2: Add

**Add → Accounts**

* Name: “Add Account to Database”
* Description: Add new user account information to Accounts database.
* Component 1: Add
* Component 2: Accounts

**Manage → Delete**

* Name: “Receive Deletion User Input”
* Description: Take input from server for user account deletion.
* Component 1: Manage
* Component 2: Add

**Delete → Accounts**

* Name: “Delete User from Database”
* Description: Upon request of the user, delete user account information from the database. Or clean database after a set period of time. Old accounts which have not been used for a period of time will be deleted.
* Component 1: Delete
* Component 2: Accounts

**Manage → Update**

* Name: “Receive New User Input”
* Description: Take input from server for new user account information.
* Component 1: Manage
* Component 2: Update

**Update → Accounts**

* Name: “Update User Account”
* Description: Update user account based on inputted information from server.
* Component 1: Update
* Component 2: Accounts

## 3.5 SOFTWARE COMPONENT CONCEPT OF EXECUTION

**Login**:

- User wants access their account.

- User inputs their credentials.

- Credentials are either validated and the user is given access, or they are denied.

**Account**:

- User needs to view profile statistics and access other components.

- User will interact with UI elements to view information or access the function of components.

- Information is displayed and utility is exposed.

**Settings**:

- User wants to modify their preferences.

- User will interact with UI elements to modify value of preferences.

- Changes are pushed to the database and are reflected in the client.

**Chat System**:

- User wants to communicate with party or individuals.

- User provides input to component.

- Input is transmitted to recipient(s).

**Party System**:

- User(s) would like to join or create a party

- User(s) send invitation requests to other clients

- Requests are either accepted and a party is formed or they are denied.

**Management:**

- Developers want to make changes to the database.

- Developers create user policies or modify individual accounts.

-Modifications are pushed to clients.

**Invite:**

-Group owner would like to invite other users, or they are requested by users.

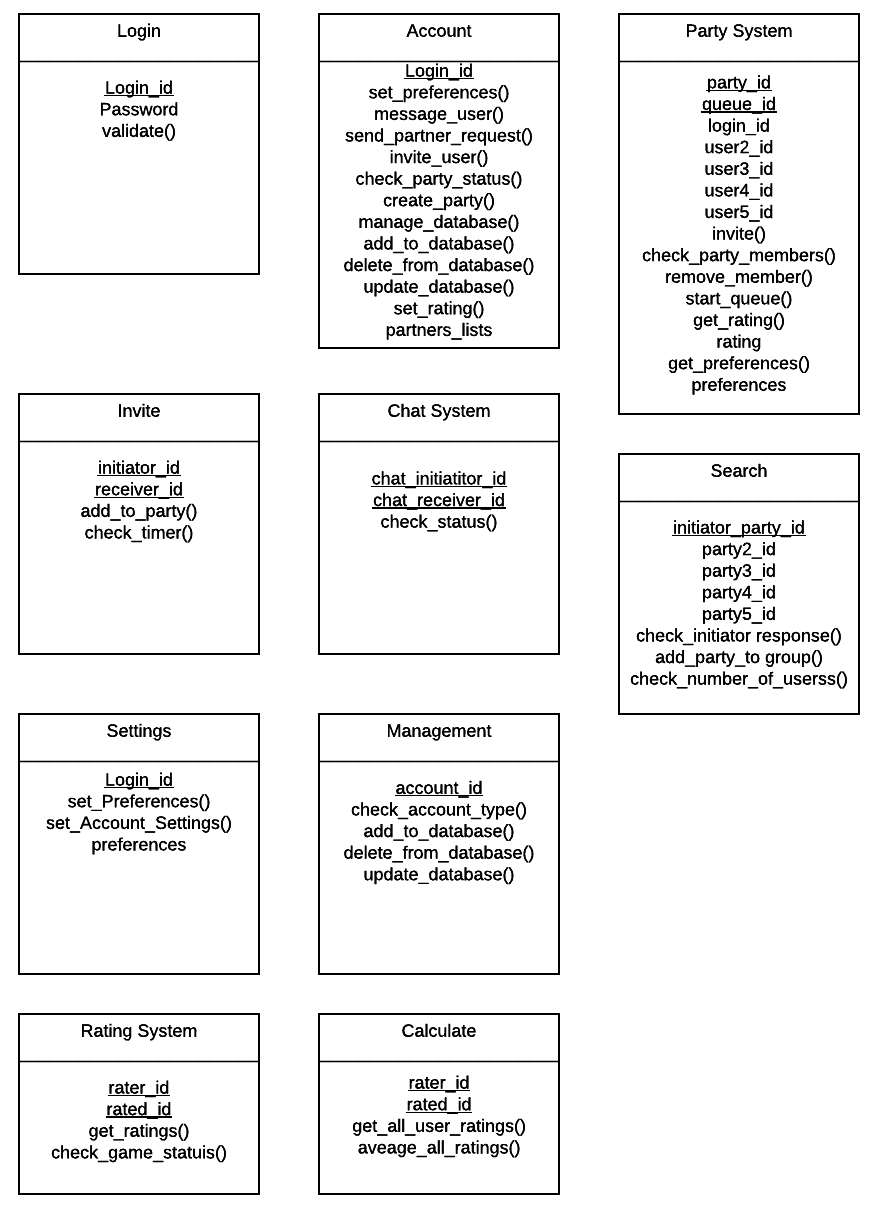
-Invitation is extended.

-Invite is either accepted and membership is allocated or it is denied.

# 4. SOFTWARE ITEM DETAILED DESIGN

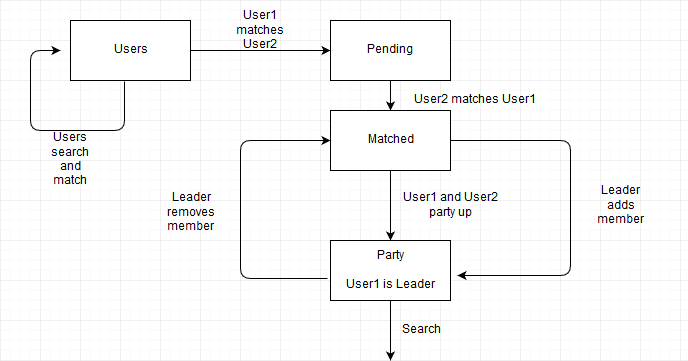
## 4.1 STRUCTURE

### 4.1.1 Software Unit Detailed Design



## 4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT

### 4.2.1 Run-time Object Instances



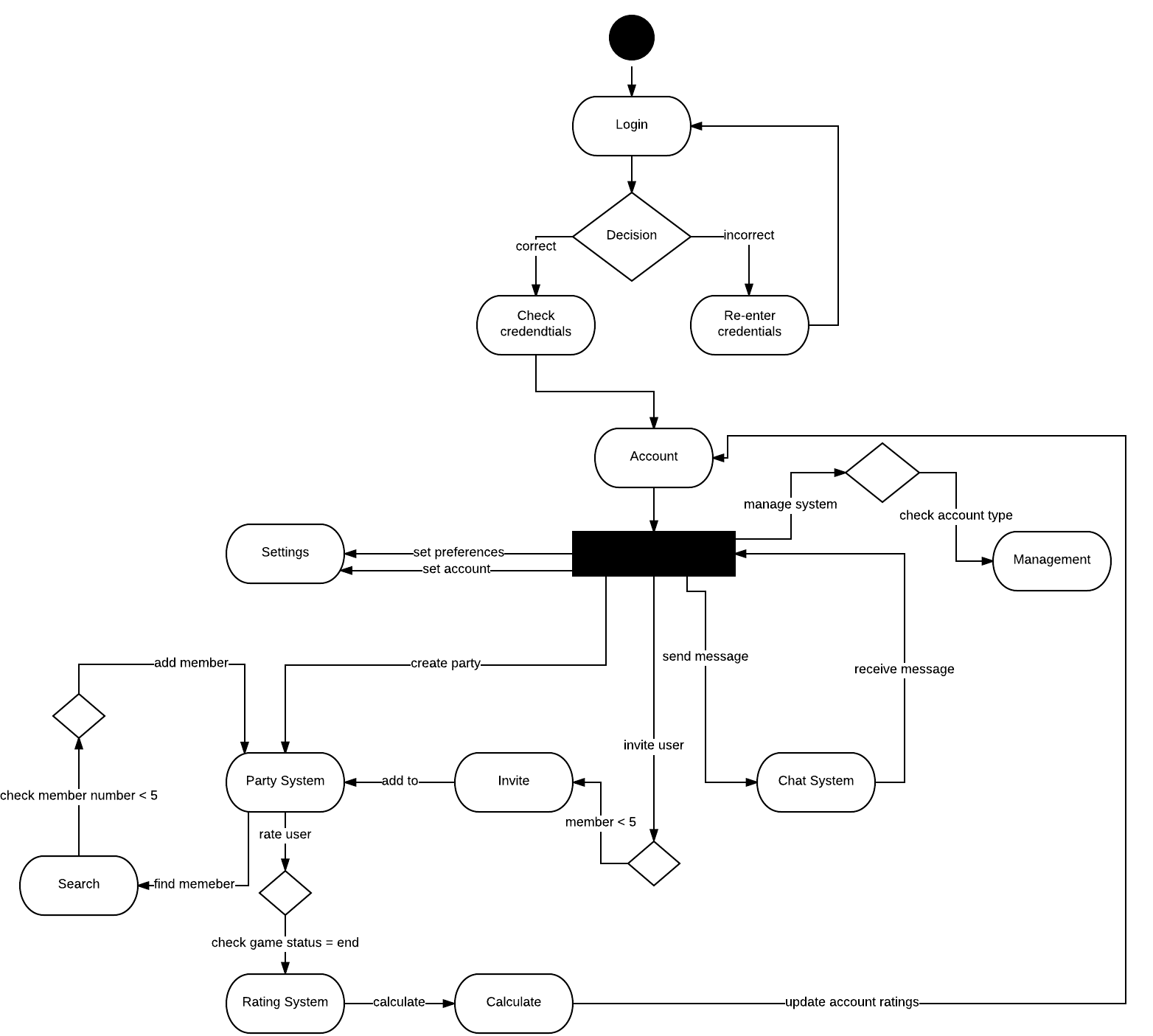
## 4.3 BEHAVIOR

### 4.3.1 Sequence Diagrams

### 4.3.2 Collaboration Diagram

See sequence diagrams.

### 4.3.3 Activity Diagrams



## 4.4 CONCEPT OF EXECUTION

**Login**:

- User wants access their account.

- User inputs their credentials.

- Credentials are either validated and the user is given access, or they are denied.

**Account**:

- User needs to view profile statistics and access other components.

- User will interact with UI elements to view information or access the function of components.

- Information is displayed and utility is exposed.

**Settings**:

- User wants to modify their preferences.

- User will interact with UI elements to modify value of preferences.

- Changes are pushed to the database and are reflected in the client.

**Chat System**:

- User wants to communicate with party or individuals.

- User provides input to component.

- Input is transmitted to recipient(s).

**Party System**:

- User(s) would like to join or create a party

- User(s) send invitation requests to other clients

- Requests are either accepted and a party is formed or they are denied.

**Management:**

- Developers want to make changes to the database.

- Developers create user policies or modify individual accounts.

- Modifications are pushed to clients.

**Invite:**

- Group owner would like to invite other users, or they are requested by users.

- Invitation is extended.

- Invite is either accepted and membership is allocated or it is denied.

## 4.5 INTERFACE DESIGN

### 4.5.1 Unique Identifier of Interface

View (Android):

Interface component responsible for drawing and event handling.

Components:

- Credential Authentication

- Preferences

- Queue Directory (Matchmaking)

- Public Posting

- Party Management Interface

- Private Messaging Interface

- Teammate Rating

### 4.5.2 Interface Identification and Diagrams

Credential Authentication:

- Text fields for the user to input their username and password.

Preferences:

- Contains drop down and context menus, in addition to radio buttons for the user to adjust their preferences.

Queue Directory:

- Button(s) that allows the party members to initiate the matchmaking process, as well as accepting a match if found.

Public Posting:

- Text field to allow other users to post messages on player profiles or group pages.

Party Management Interface:

- Buttons that allow party members to mute, ignore, invite or request invitation from other users (not limited to the party leader).

Private Messaging Interface:

- Text input and field that allow users to message the party or individual users. This interface should be accessible from all views in the application.

Teammate Rating:

- A rating interface to rate members of the party to rate each other. Shall only be accessible through the post-match phase.

# 5. IMPLEMENTATION ARCHITECTURE OF (NOT REQUIRED)

## 5.1 ALL ACTIVE AND PASSIVE CLASSES ASSIGNED TO COMPONENTS

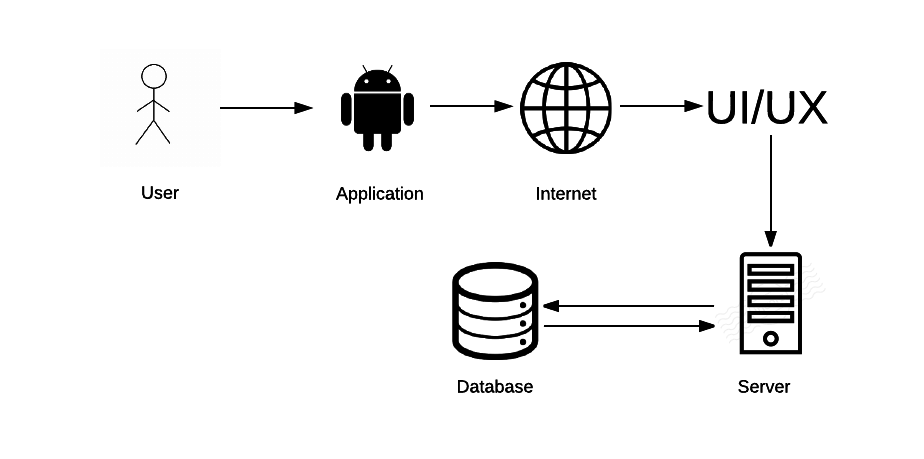
Not required.

## 5.2 DIAGRAM OF PHYSICAL PACKAGING OF LOGICAL COMPONENTS

Not required.

# 6. DEPLOYMENT ARCHITECTURE

## 6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM



# 7. CODE (PSEUDO)

Not required in current version.

Required in final version of SDD due November 22, 2016.

# 8. DICTIONARIES

**Classes**

* **User** - either general user of developer level user.
  + log\_in\_name - user name used for credential validation.
  + Dota\_game\_name - Dota game name used to identify in game name.
  + password - password used for credential validation.
  + party\_id - used as a pointer to point to party.
  + rating - rating is calculated through feedback from other users. Identifies the type of player this user is.
  + partner\_list - friend list.
* set\_user-preference() - allows user to set preferences to for future party members.
* send\_partner\_request() - sends a request to another user asking to add them to friends list.
* send\_message() - user can send messages to other users
* remove\_from\_party() - if the user is the party leader, they can remove party members from the party.
* exit\_party() - if the user is the party leader, they can disband the party.
* **Party** - contains a maximum of 5 users.
  + party\_id - identifier for the party.
  + queue\_id - identifier used while inside matchmaking queue.
  + member\_name\_2 - second party member name.
  + member\_name\_3 - third party member name.
  + member\_name\_4 - fourth party member name.
  + member\_name\_5 - fifth party member name.
  + number\_of\_member - the number of members already in the party. Cannot exceed 5.
* set\_party\_preferences() - variables conditions are set to look for users who meet the criteria.
* **Party Leader** - user who created and has control over party function
  + in\_game\_name - pointer to the user who created the party
  + party\_id - pointer to the party. Used to identify who the party leader is for the party.
* invite\_member() - send invites to partners to join the party.
* remove\_party\_member() - removes a user from the party.
* start\_queue() - place party into matchmaking queue
* **Queue** - matchmaking queue to find party members.
  + queue\_id - identifier for the queue while in the matchmaking system.
  + first\_party\_id - identifier for the first party.
  + second\_party\_id - identifier for the second party.
  + third\_party\_id - identifier for the third party.
  + fourth\_party\_id - identifier for the fourth party.
  + fifth\_party\_id - identifier for the fifth party.
* check\_preferences() - use variable conditions to look for party members
* find\_matches() - find party members.
* exit\_queues() - take party out of queue.
* **Preferences** - variables that are preset by users for future party members.
  + log\_in\_name - pointer used to identify which user account
  + Dota\_game\_name - pointer used to identify which user account
  + party\_id - pointer used to identify party.
  + skill\_rating - used to determine potential party members.
  + role - used to determine potential party members.
  + attitude - used to determine potential party members.

**Components**

* **User Interface**
  + Login Information - user inputs login credentials
  + Login - receives user login inputs
  + Account - checks for successful user login. Once logged in, user can select multiple actions.
  + Chat System - messaging system a user can use to contact another user.
  + Party System - users can create a party in order to look for potential party members.
  + Invite - users can invite friends before creating a party or party leaders can invite users to the party.
  + Management - used by developers to manage the server or database.
  + Settings - users can set preferences for their account or potential party members.
* **Matchmaking System**
  + User - users have multiple actions to select from.
  + Party System - can receive ratings from the rating system or use preset preferences to determine potential party members.
  + Search - look for potential party members.
  + Invite - users or party leaders can invite users to the party.
* **Rating System**
  + User - users are only allowed to rate another user from their party once game has ended.
  + Rating - receive user inputs.
  + Calculate - take received inputs and take the average of inputs from party members.
  + Account -update user rating information.
* **Database**
  + Server - database receives inputs from the server and outputs requested data to the server.
  + Manage - developers can manage the database.
  + Add - receive new user information.
  + Delete - receive user information to delete.
  + Update - receive new user information to update existing user information.
  + Accounts - contains all user account data.

# 9. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION

No resources required.

# 10. REQUIREMENTS TRACEABILITY

## 10.1 SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY

All Software Component-Level documents should have forward (to all artifacts spawned by this document) and backward (to previous stages of development) traceability. Forward traceability will allow tracing in a forward direction tracing from requirements to post-delivery workflows. While backward traceability allows tracing in the direction of post-delivery to requirements workflows.

All component documents are accessible through Google Docs and each developer's hard drive. Requirements Specification Analysis Version 1.0 and other previous documents were used to update and implement component documents. All previous documents are accessible through NYU Classes and Google Docs.

# 11. SYSTEM DESIGN TESTING

* System test plans will be developed and conducted by the Software Quality Group (SQA).
* SQA will also run the product test.
* Product will be tested at the end of each workflow
* Product will be tested with a small sample of users to determine that the implemented functions work as intended.
* After implementation testing is successful, a larger user base will be introduced to test the endurance of the system.
* Acceptance test will be conducted by the client. The client can ask the SQA to assist them but it is the responsibility of the client to approve the product.

# 12. RATIONALE

No additional rationales.

# 13. NOTES

No additional notes.

# 14. APPENDICES

## 14.1 DICTIONARIES

Can be found in Section 8.

## 14.2 UML DIAGRAMS

Included in body of document.

## 14.3. SCHEDULE TRACKING

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or Deliverable | Who (Individual or Team) | Estimated | Actual | Difference |
| Software Requirements Specification (SRS) Version 1.0 | Corey Chong | 6 | 7 | 1 |
| Software Requirements Specification (SRS) Version 1.0 | Albert Su | 6 | 7 | 1 |
| Software Requirements Specification (SRS) Version 1.0 | Evans Yeung | 6 | 7 | 1 |
| Software Requirements Specification (SRS) Version 2.0 | Corey Chong | 5 | 4 | 1 |
| Software Requirements Specification (SRS) Version 2.0 | Albert Su | 5 | 5 | 0 |
| Software Requirements Specification (SRS) Version 2.0 | Evans Yeung | 6 | 8 | 2 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Corey Chong | 2 | 2 | 0 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Evans Yeung | 4 | 3 | 1 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Albert | 1 | 2 | 1 |
| Software Design Document (SDD) Version 1.0 | Corey Chong | 15 | 17 | 2 |
| Software Design Document (SDD) Version 1.0 | Evans Yeung | 13 | 17 | 4 |
| Software Design Document (SDD) Version 1.0 | Albert | 16 | 17 | 1 |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated | Actual | Difference |
| Corey Chong | 52 | 51 | 1 |
| Albert Su | 49 | 48 | 1 |
| Evans Yeung | 57 | 57 | 0 |

## 14.4 DEFECT TRACKING

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or Deliverable | Who (Individual or Team) | Estimated | Actual | Difference |
| Software Requirements Specification (SRS) Version 1.0 | Corey Chong | 15 | 20 | 5 |
| Software Requirements Specification (SRS) Version 1.0 | Albert Su | 14 | 17 | 3 |
| Software Requirements Specification (SRS) Version 1.0 | Evans Yeung | 18 | 25 | 7 |
| Software Requirements Specification (SRS) Version 2.0 | Corey Chong | 15 | 22 | 6 |
| Software Requirements Specification (SRS) Version 2.0 | Albert Su | 20 | 24 | 4 |
| Software Requirements Specification (SRS) Version 2.0 | Evans Yeung | 14 | 15 | 1 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Corey Chong | 10 | 8 | 2 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Albert Su | 7 | 8 | 1 |
| Requirements and Analysis Specification (RAS) Version 1.0 | Evans Yeung | 15 | 6 | 9 |
| Software Design Document (SDD) Version 1.0 | Corey Chong | 36 | 32 | 8 |
| Software Design Document (SDD) Version 1.0 | Evans Yeung | 30 | 41 | 4 |
| Software Design Document (SDD) Version 1.0 | Albert | 45 | 51 | 6 |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated | Actual | Difference |
| Corey Chong | 146 | 155 | 9 |
| Albert Su | 163 | 167 | 4 |
| Evans Yeung | 150 | 168 | 18 |

## 14.5 GANTT CHART

