

**Computer Science and Engineering**

**DotaMatch**

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

**Version 1.1**

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 |
| Evans Yeung | Author/Reviewer | December 2, 2016 | Evans Yeung |
| Corey Chong | Author/Reviewer | December 2, 2016 | Corey Chong |
| Albert Su | Author/Reviewer | December 2, 2016 | Albert Su |
|  |  |  |  |
|  |  |  |  |

**REVISION LEVEL**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision Number** | **Purpose** |
| October 8, 2016 | Version 1.0 | Initial Release |
| December 2, 2016 | Version 1.1 | Update/Fix |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. INTRODUCTION 6](#_Toc468749863)

[1.1 Purpose 6](#_Toc468749864)

[1.2 Scope 6](#_Toc468749865)

[1.3 Identification 6](#_Toc468749866)

[1.4 Document Summary 6](#_Toc468749867)

[1.5 System Overview 7](#_Toc468749868)

[1.6 Document Overview 7](#_Toc468749869)

[2. REFERENCE DOCUMENTS 8](#_Toc468749870)

[3. SYSTEM WIDE DESIGN DECISIONS 8](#_Toc468749871)

[3.1 SOFTWARE COMPONENT ARCHITECTURAL DESIGN 8](#_Toc468749872)

[3.2 SOFTWARE ARCHITECTURE GENERAL DESCRIPTION 11](#_Toc468749873)

[3.3 SOFTWARE ITEM COMPONENTS 11](#_Toc468749874)

[3.4 COMPONENT INTERFACE IDENTIFICATION 12](#_Toc468749875)

[3.5 SOFTWARE COMPONENT CONCEPT OF EXECUTION 16](#_Toc468749876)

[4. SOFTWARE ITEM DETAILED DESIGN 18](#_Toc468749877)

[4.1 STRUCTURE 18](#_Toc468749878)

[4.1.1 Software Unit Detailed Design 18](#_Toc468749879)

[4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT 20](#_Toc468749880)

[4.2.1 Run-time Object Instances 21](#_Toc468749881)

[4.3 BEHAVIOR 36](#_Toc468749882)

[4.3.1 Sequence Diagrams 36](#_Toc468749883)

[4.3.2 Collaboration Diagram 43](#_Toc468749884)

[4.3.3 Activity Diagrams 43](#_Toc468749885)

[4.4 CONCEPT OF EXECUTION 44](#_Toc468749886)

[4.5 INTERFACE DESIGN 45](#_Toc468749887)

[4.5.1 Unique Identifier of Interface 45](#_Toc468749888)

[4.5.2 Interface Identification and Diagrams 45](#_Toc468749889)

[5. IMPLEMENTATION ARCHITECTURE OF (NOT REQUIRED) 46](#_Toc468749890)

[5.1 ALL ACTIVE AND PASSIVE CLASSES ASSIGNED TO COMPONENTS 46](#_Toc468749891)

[5.2 DIAGRAM OF PHYSICAL PACKAGING OF LOGICAL COMPONENTS 46](#_Toc468749892)

[6. DEPLOYMENT ARCHITECTURE 47](#_Toc468749893)

[6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM 47](#_Toc468749894)

[7. CODE (PSEUDO) 49](#_Toc468749895)

[8. DICTIONARIES 55](#_Toc468749896)

[9. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION 70](#_Toc468749897)

[10. REQUIREMENTS TRACEABILITY 70](#_Toc468749898)

[10.1 SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY 70](#_Toc468749899)

[11. SYSTEM DESIGN TESTING 70](#_Toc468749900)

[12. RATIONALE 70](#_Toc468749901)

[13. NOTES 71](#_Toc468749902)

[14. APPENDICES 71](#_Toc468749903)

[14.1 DICTIONARIES 71](#_Toc468749904)

[14.2 UML DIAGRAMS 71](#_Toc468749905)

[14.3. SCHEDULE TRACKING 71](#_Toc468749906)

[14.4 DEFECT TRACKING 73](#_Toc468749907)

[14.5 GANTT CHART 76](#_Toc468749908)

# 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 1.1, December 2, 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

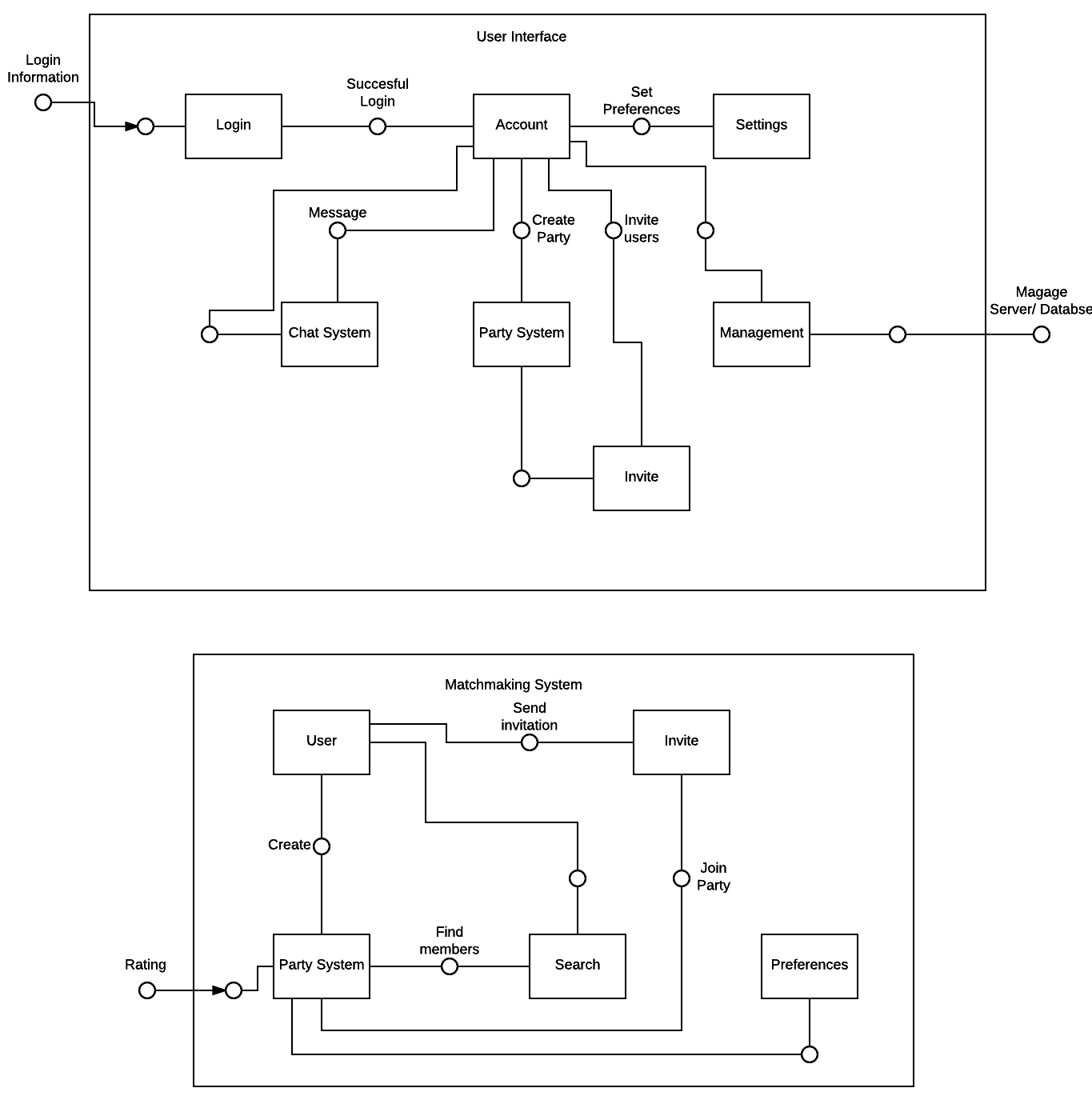
All documents are electronically submitted to NYU Classes and can be referenced through NYU Classes.

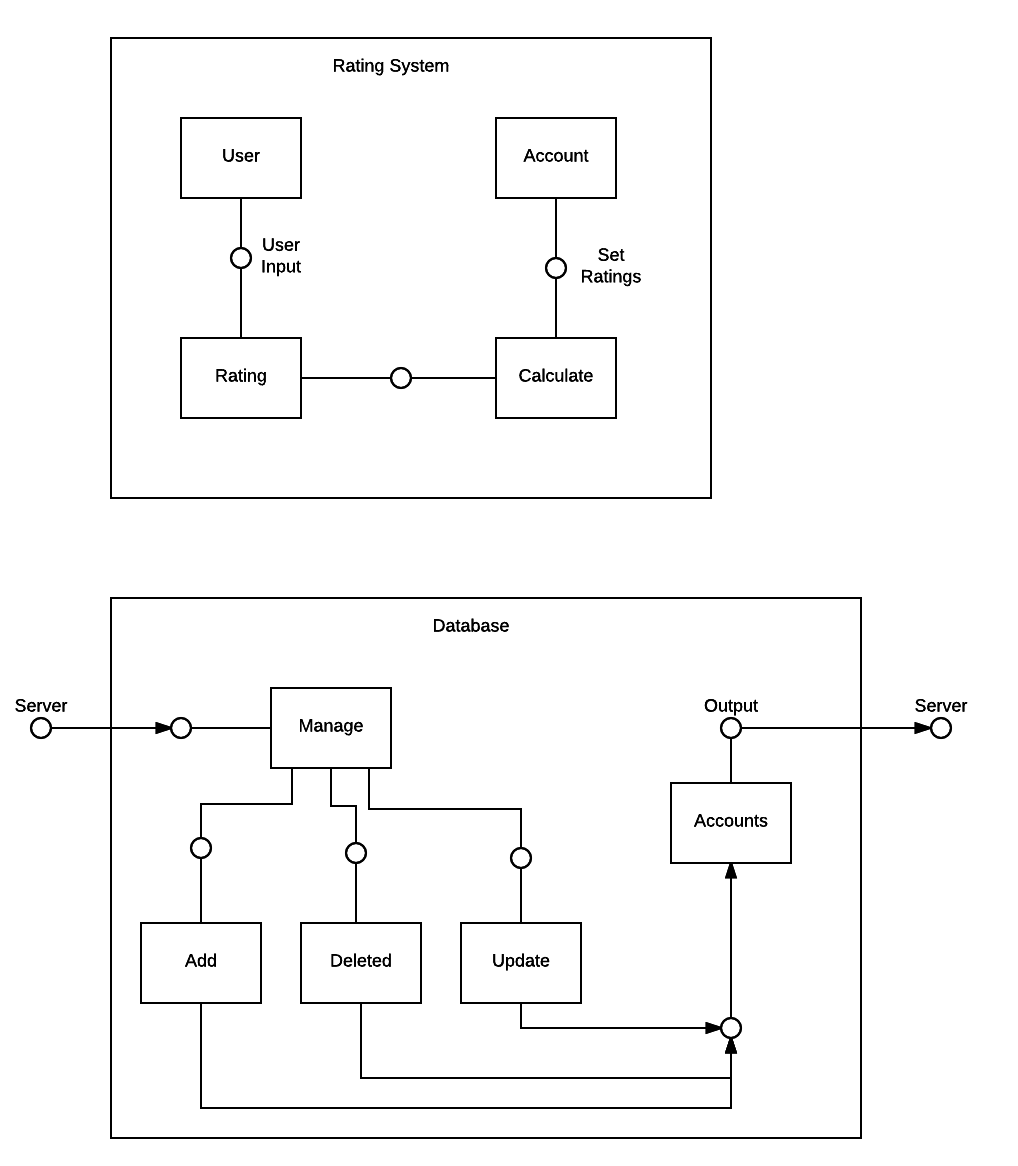
- Project Proposal was electronically submitted to NYU Classes on February 17, 2016.

* 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 was electronically submitted to NYU Classes on October 25, 2016.
* Software Design Description Version 1.1 will be submitted to NYU Classes on December 6, 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

**Startup**:

- User starts up their Android phone.

- User installs DotaMatch application.

- User taps DotaMatch application to start it up.

- User is brought to login page.

**Registration**:

- User is at the login page.

- If user does not have an existing DotaMatch account user clicks the registrations button.

- User will be prompted to enter their email and create a password.

- Once user finalizes all inputs, the user account will be created and added to the database.

**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.

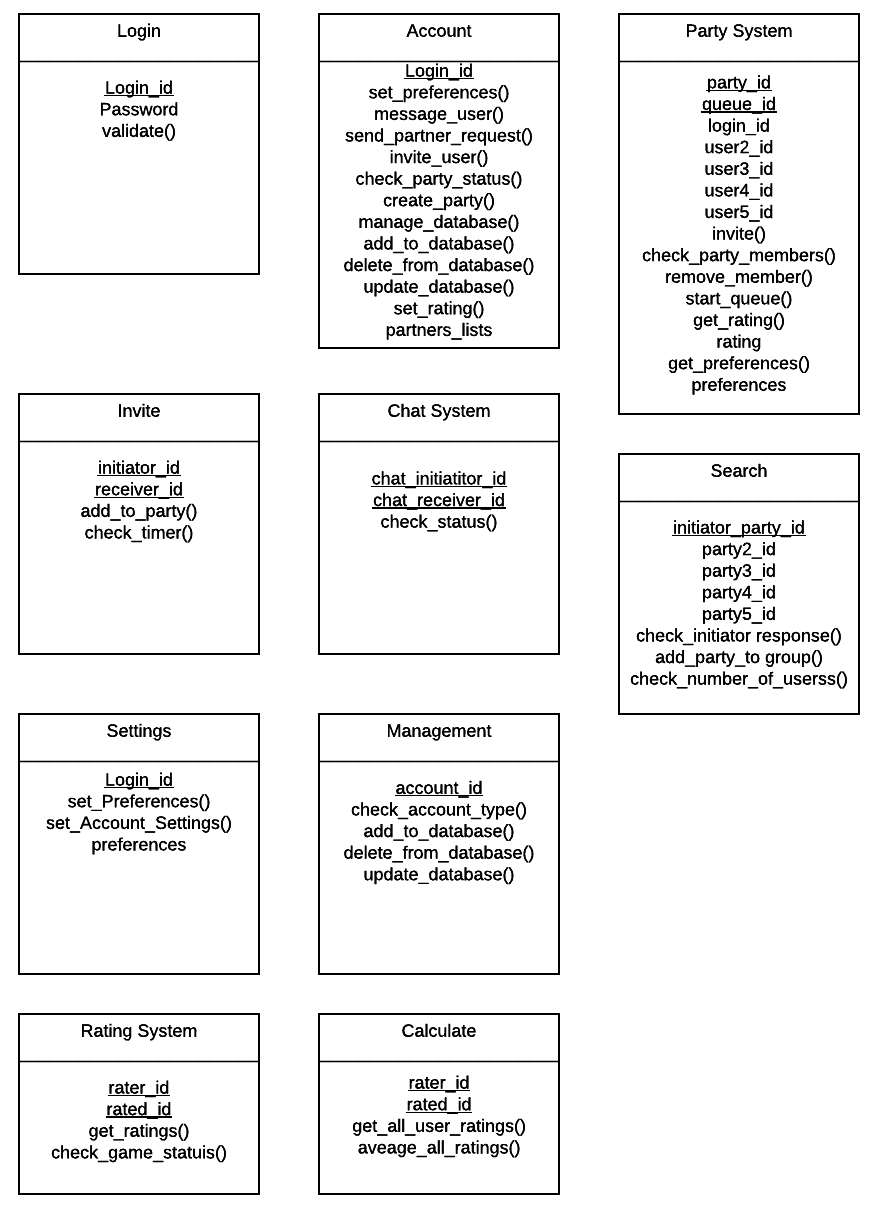
-Invitation is extended.

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

# 4. SOFTWARE ITEM DETAILED DESIGN

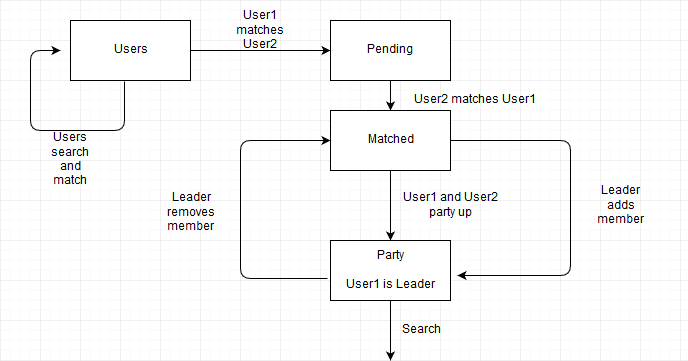
## 4.1 STRUCTURE

### 4.1.1 Software Unit Detailed Design

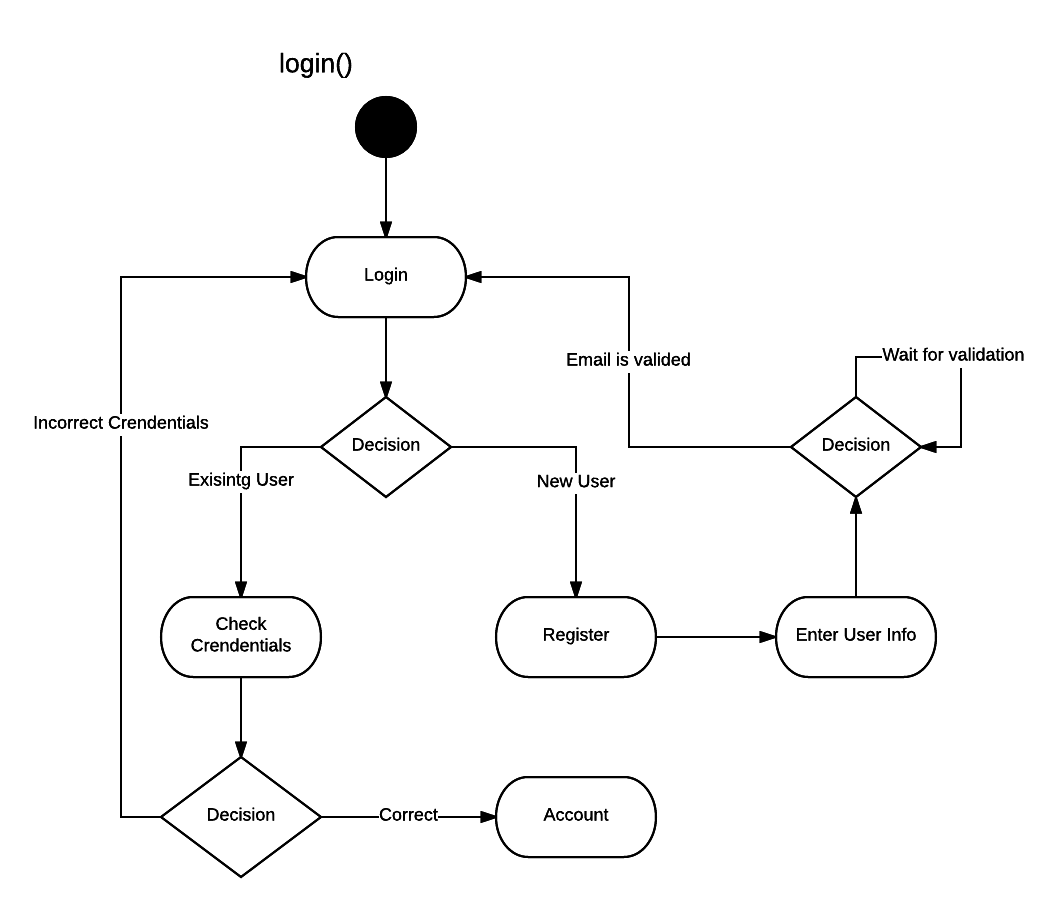


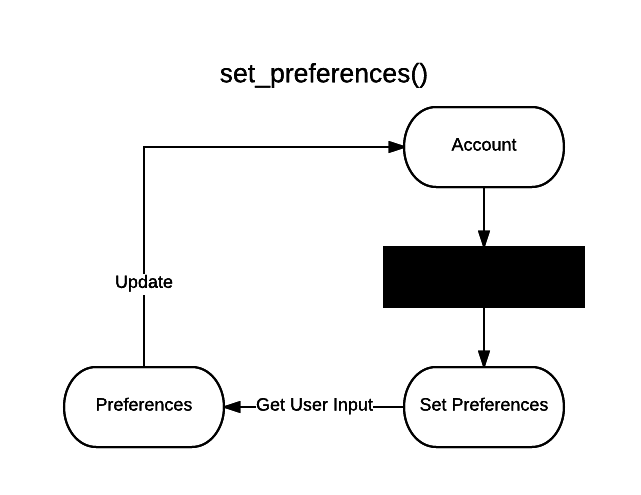
## 4.2 STATIC RELATIONSHIP OF SOFTWARE UNIT

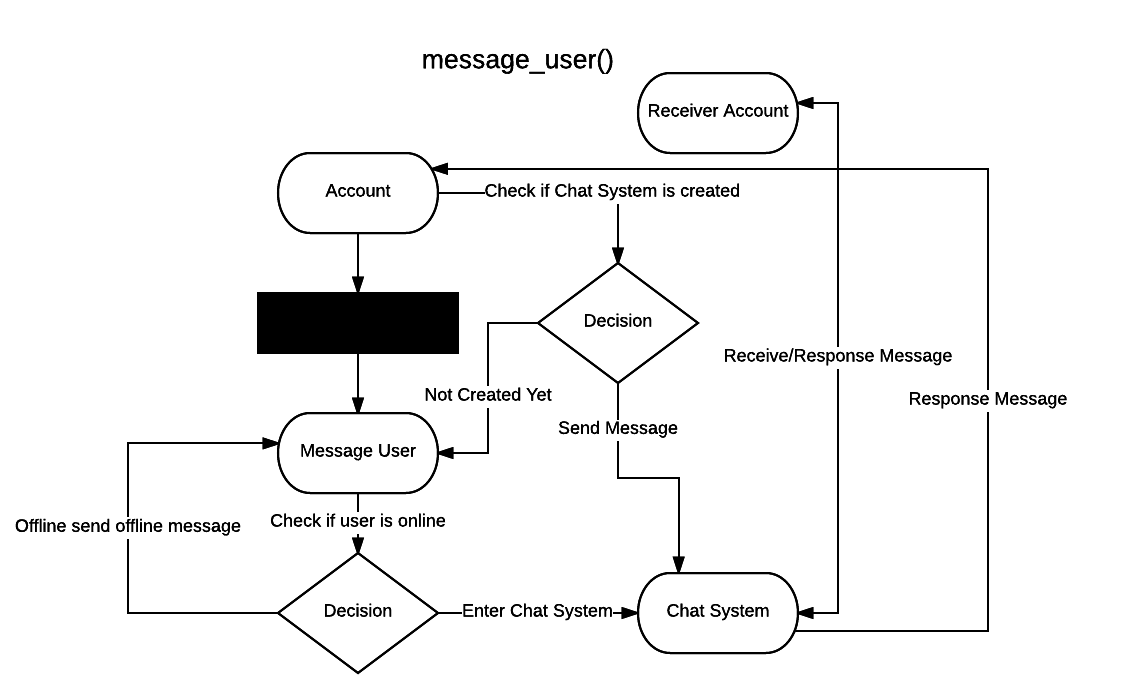
### 4.2.1 Run-time Object Instances

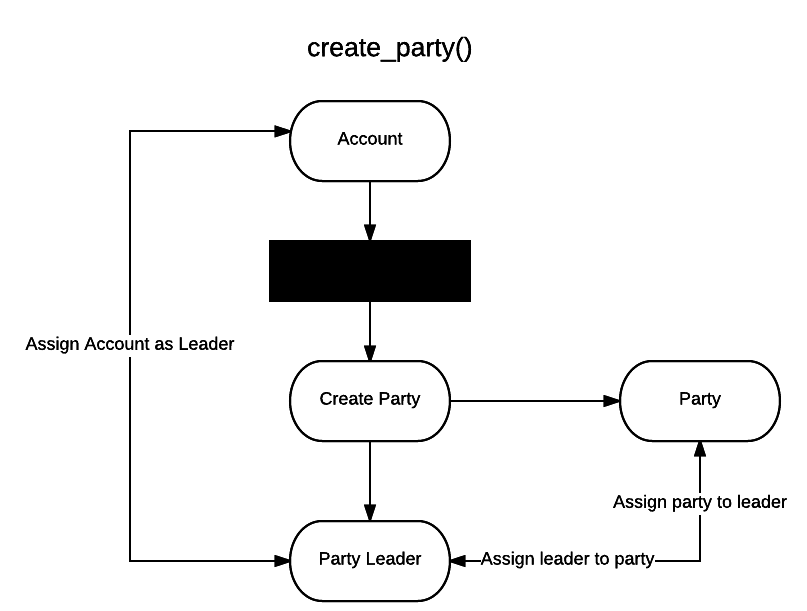


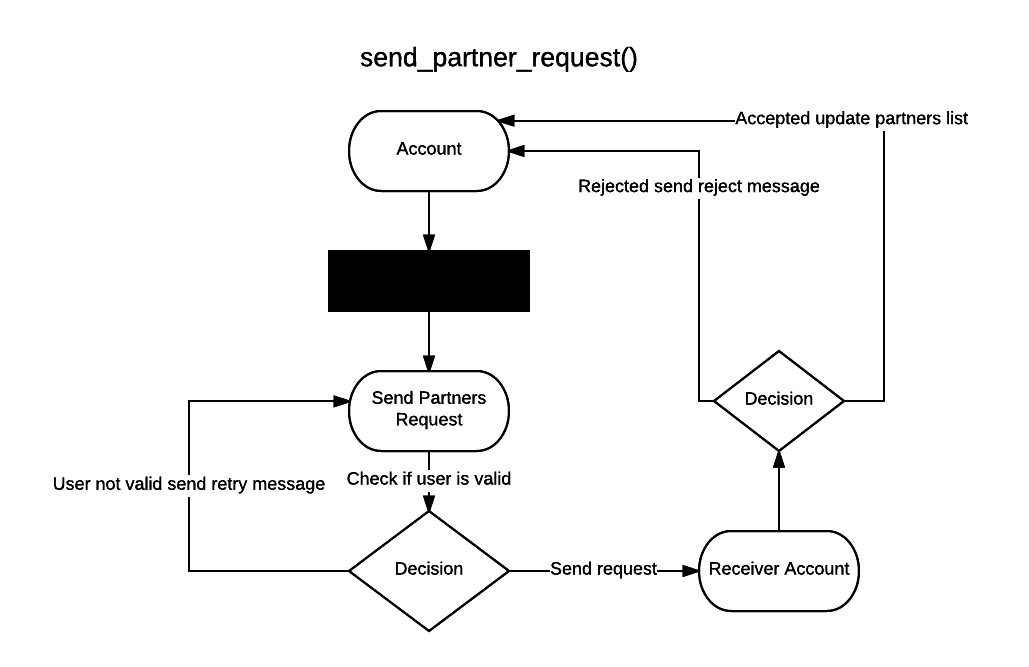
**Class Method State Diagrams**

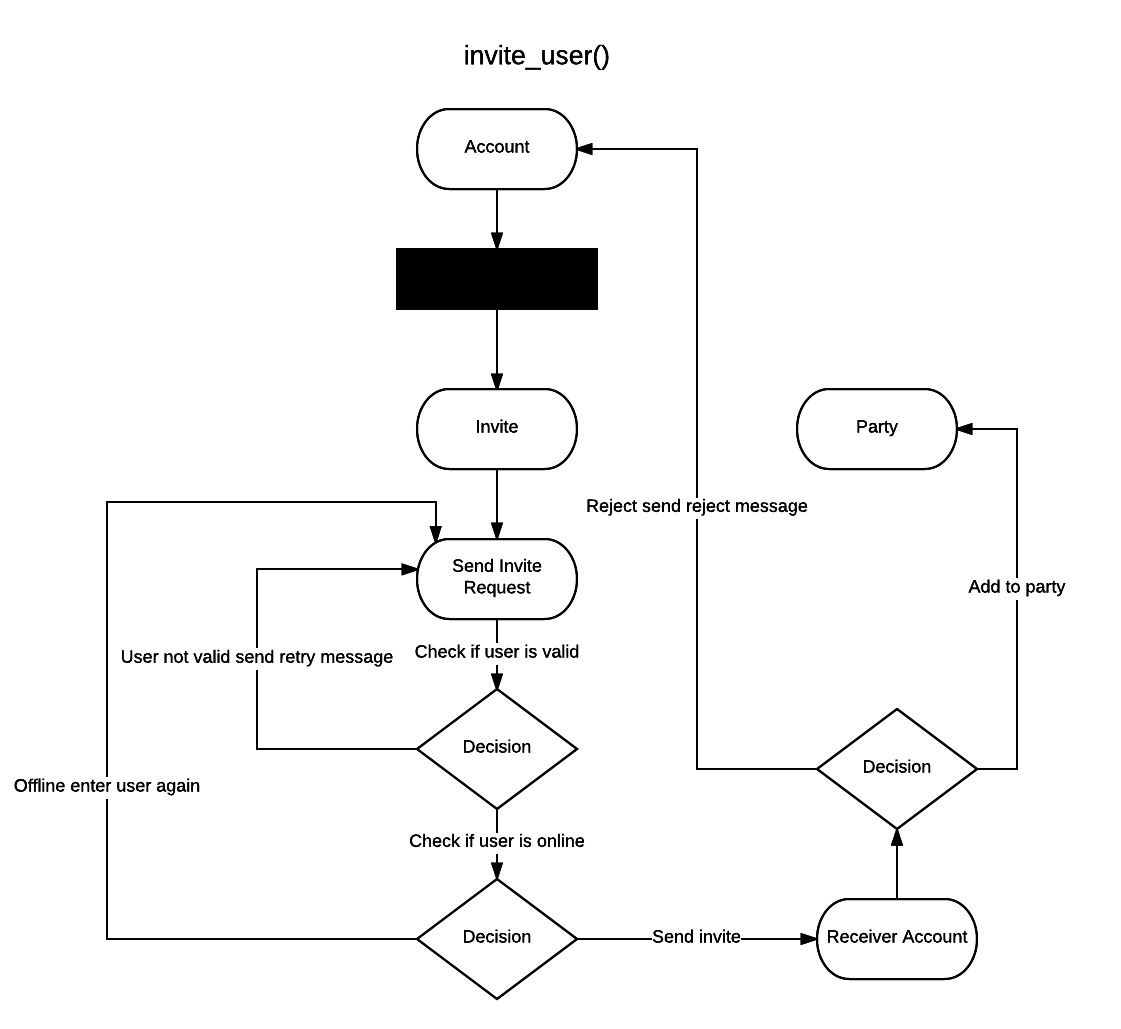
**Class User**

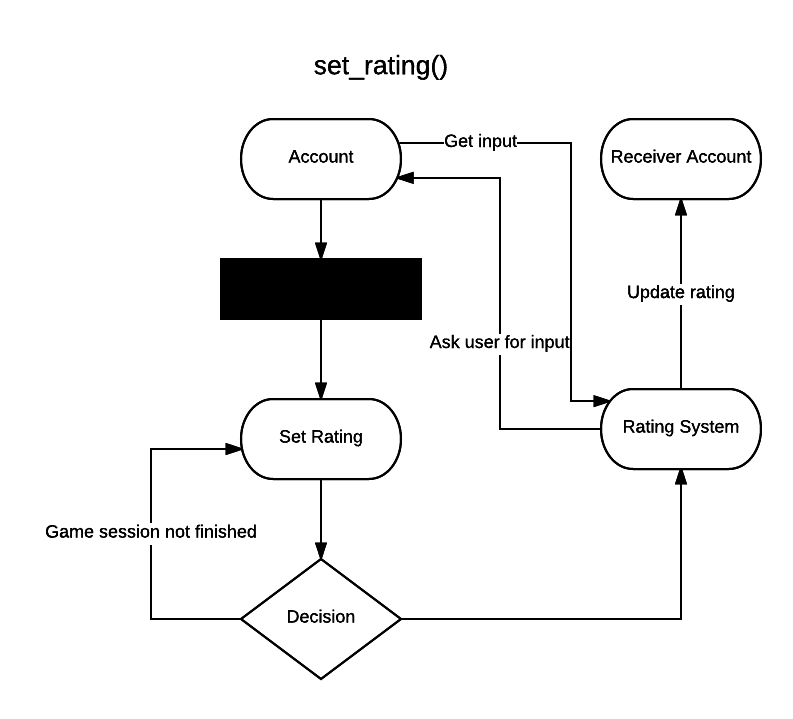


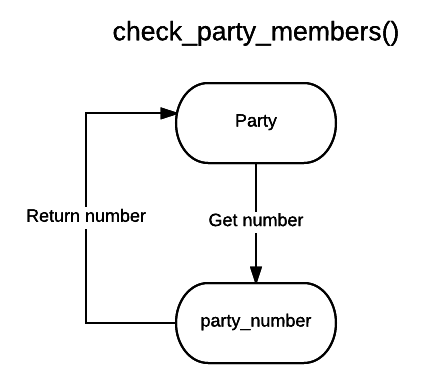




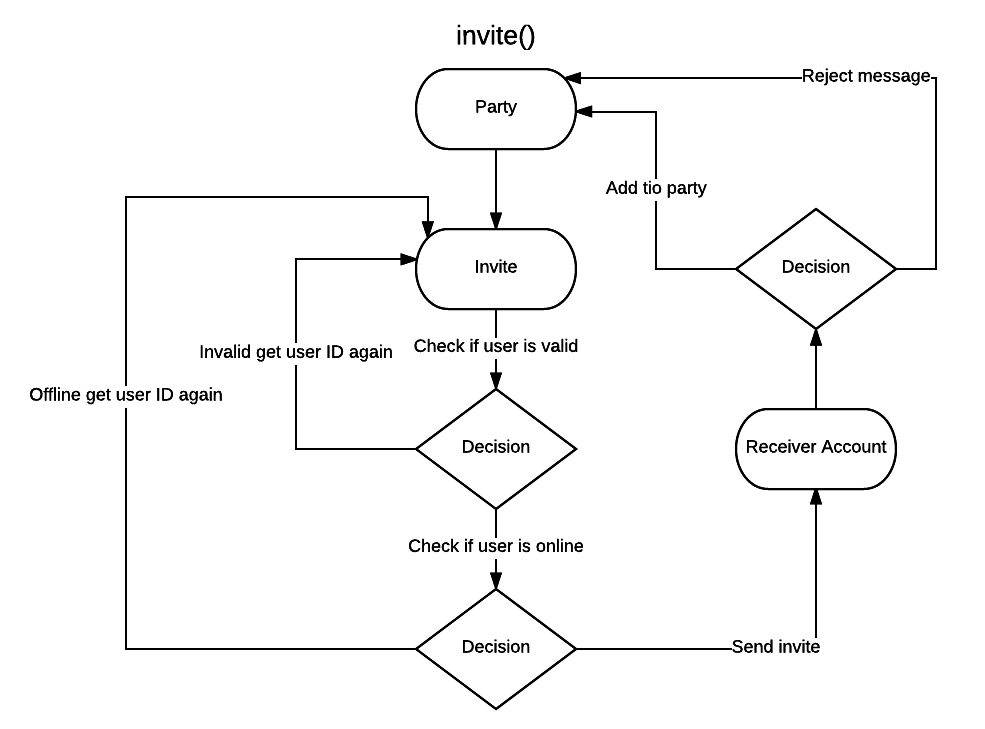


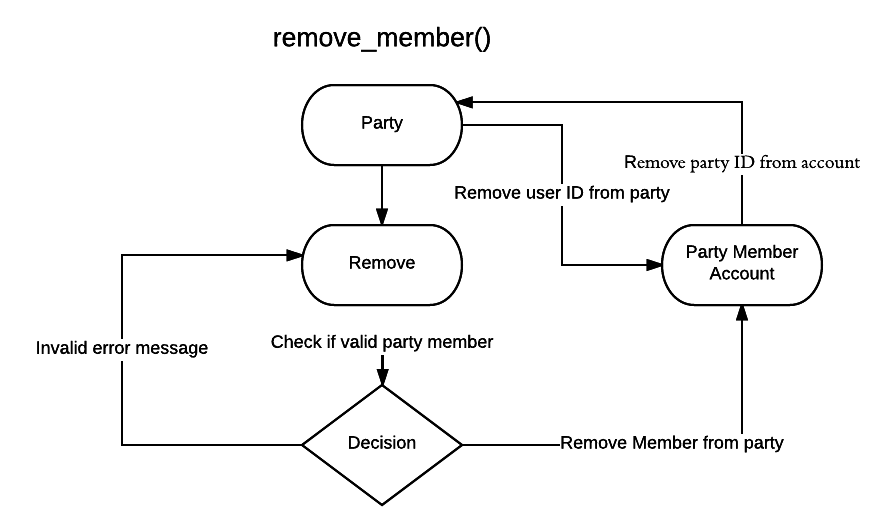


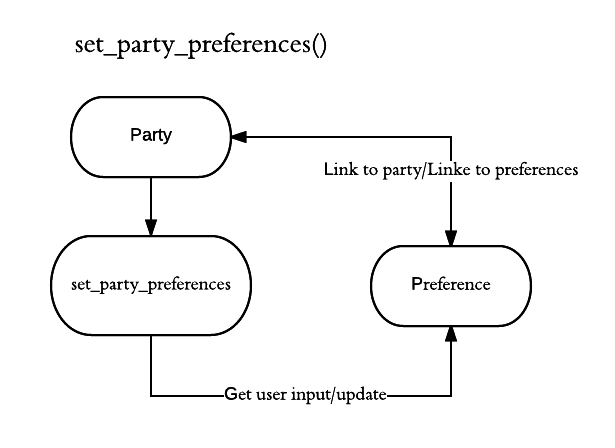


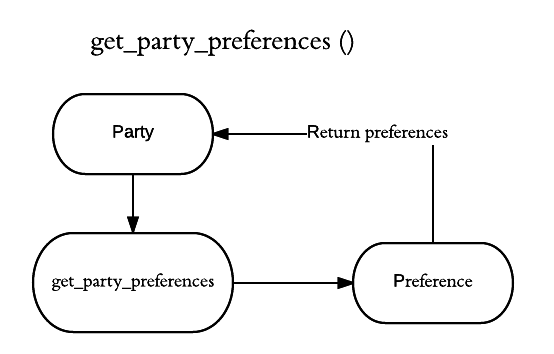


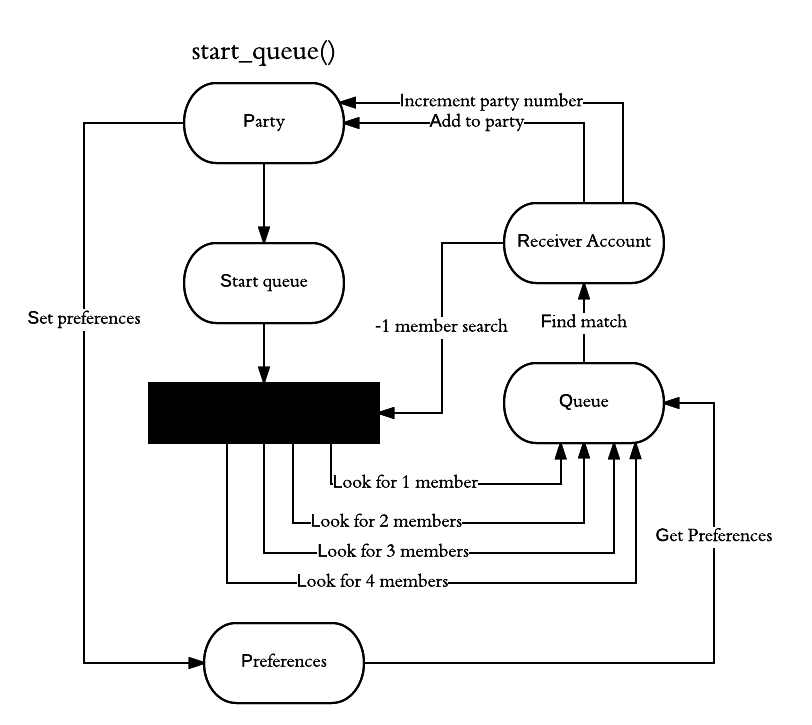
**Class Party**



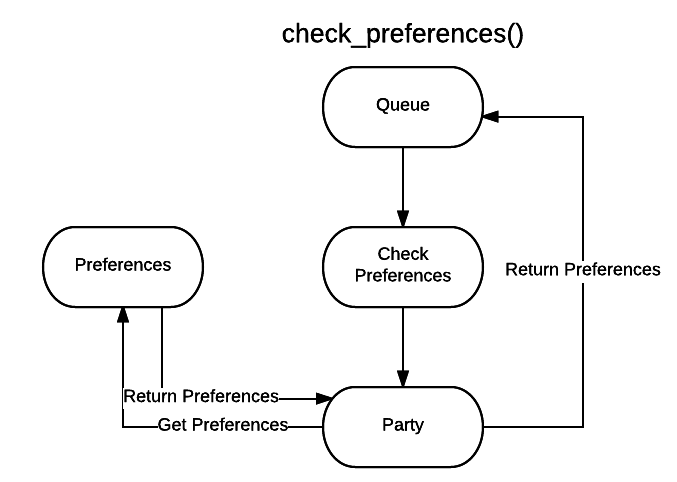


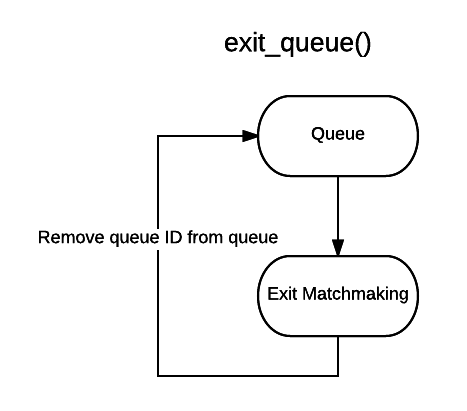


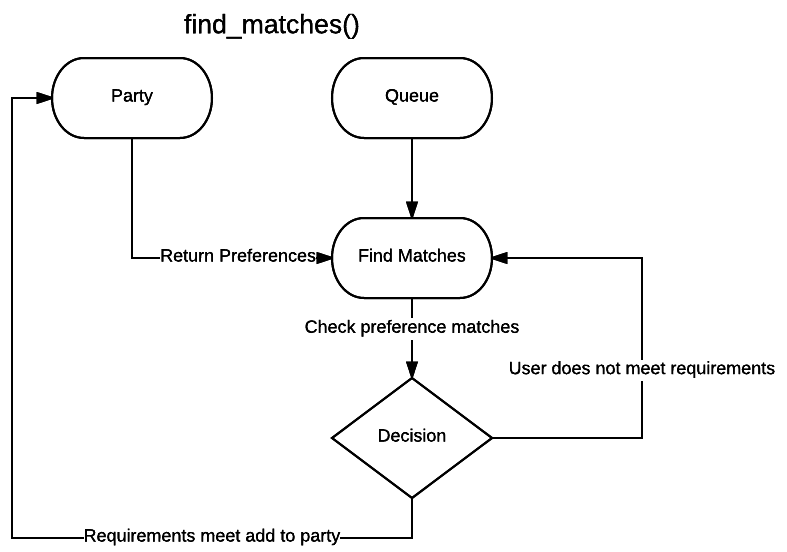




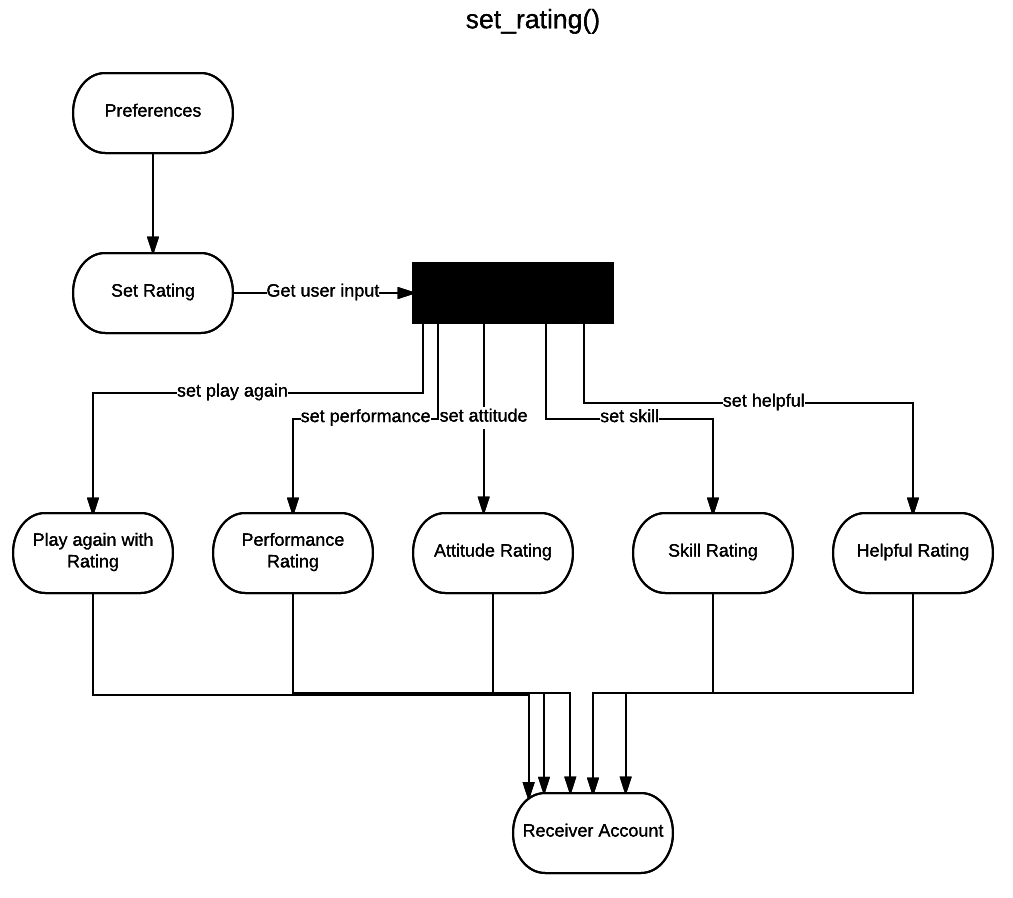
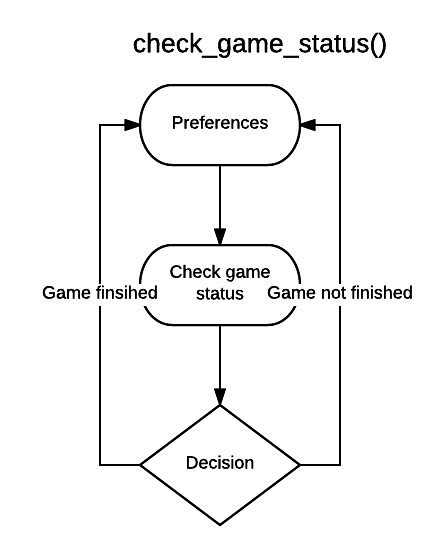
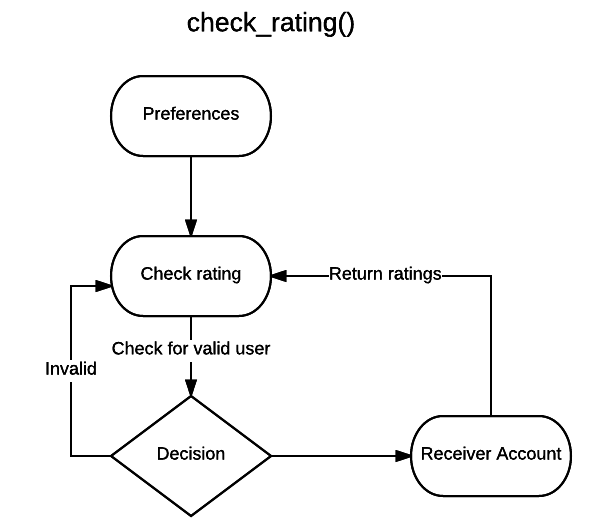
**Class Queue**

****

****

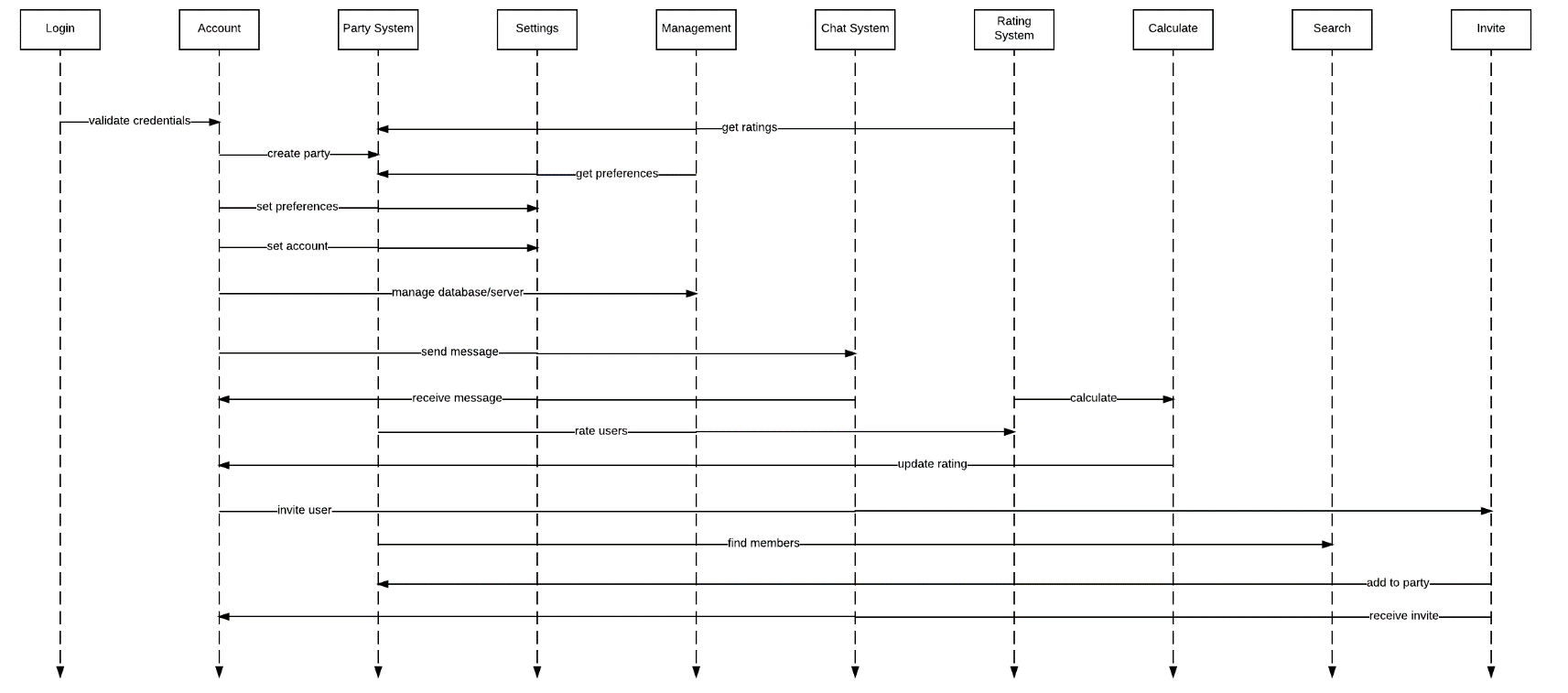
****

**Class Rating**

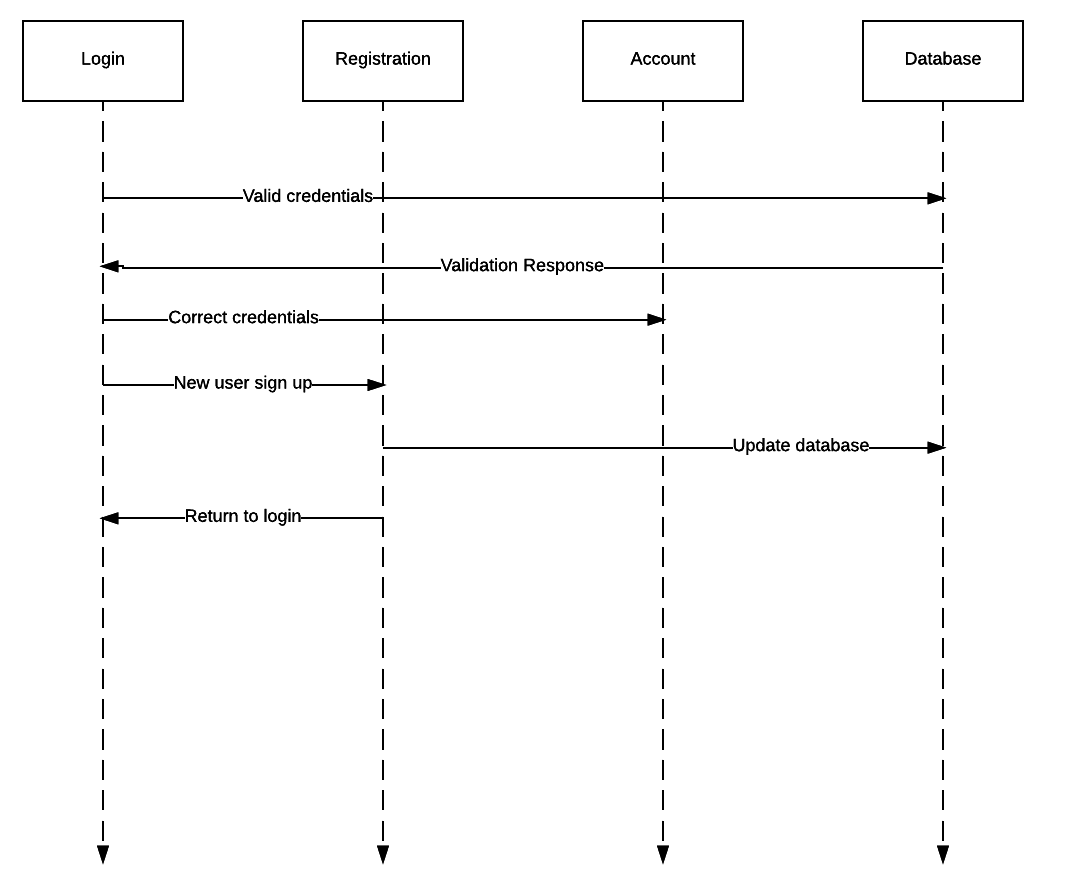
****

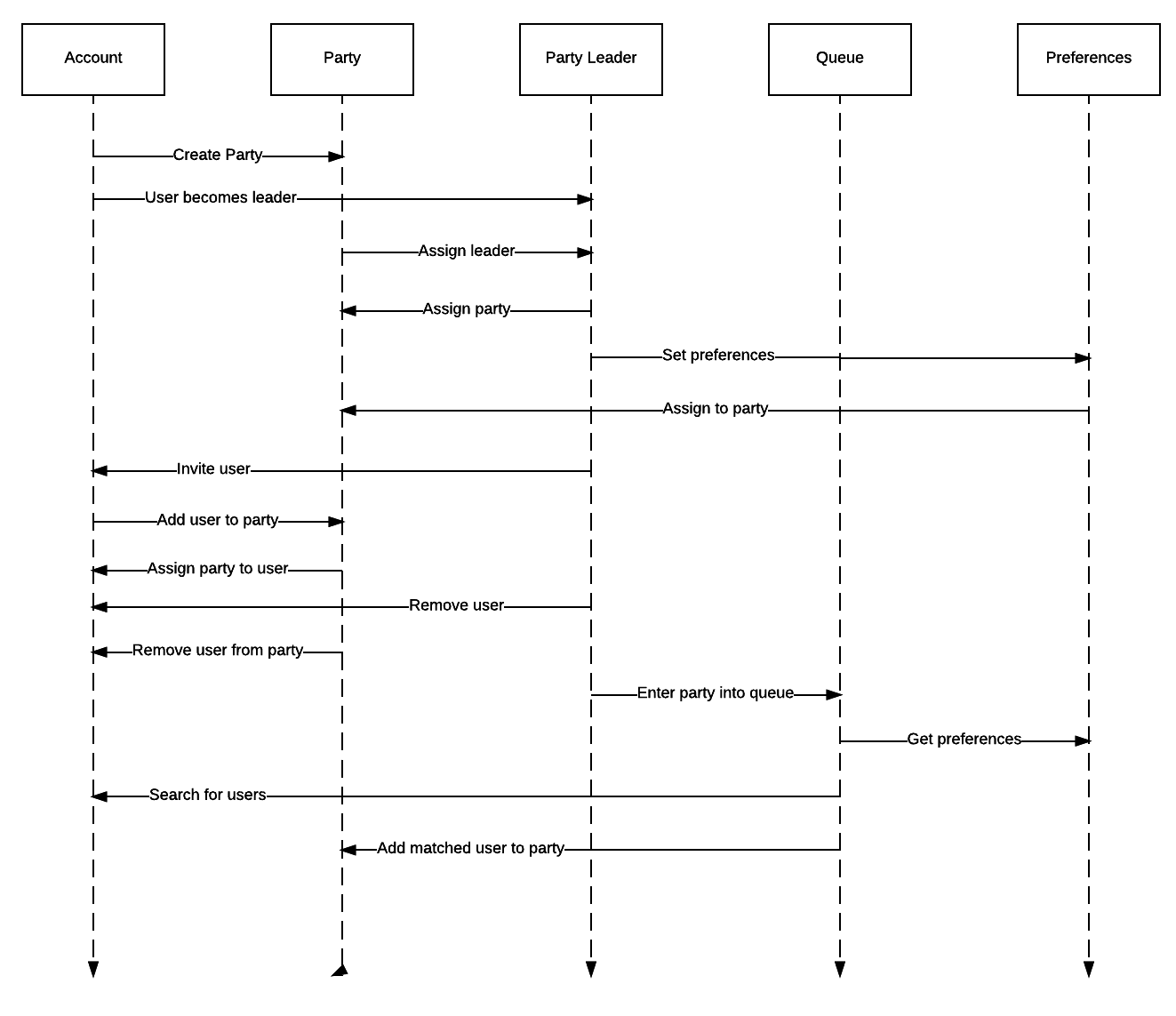
## 4.3 BEHAVIOR

### 4.3.1 Sequence Diagrams

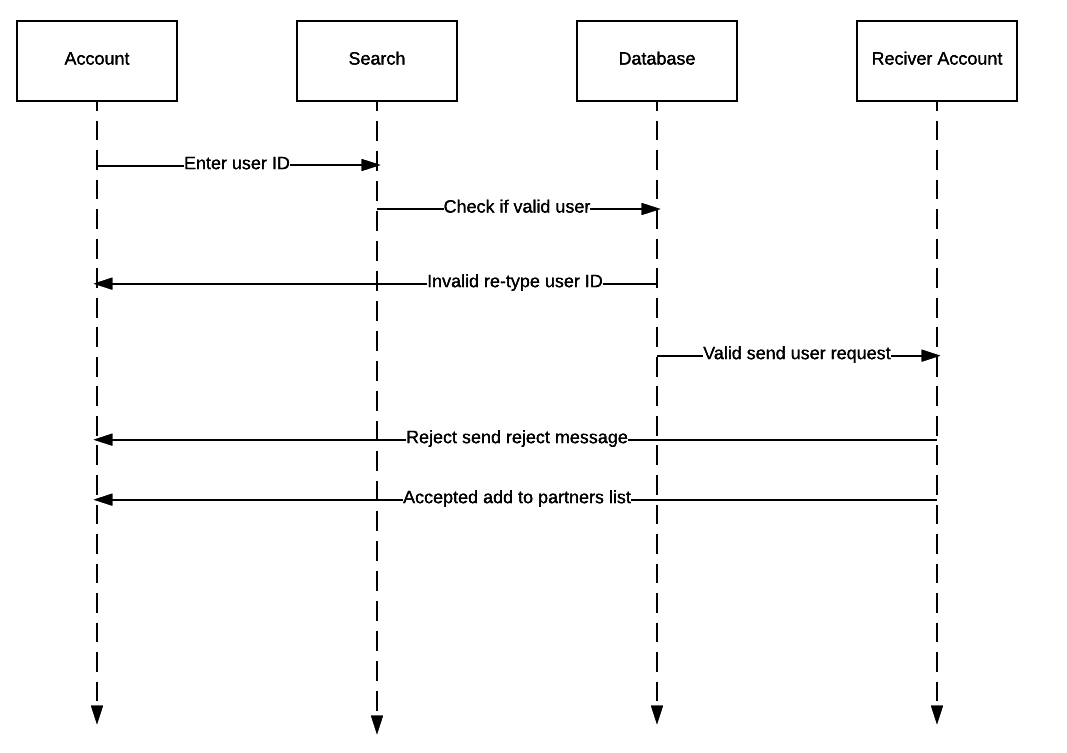
**Components Sequence Diagram**

**Functional Requirements Sequence Diagram**

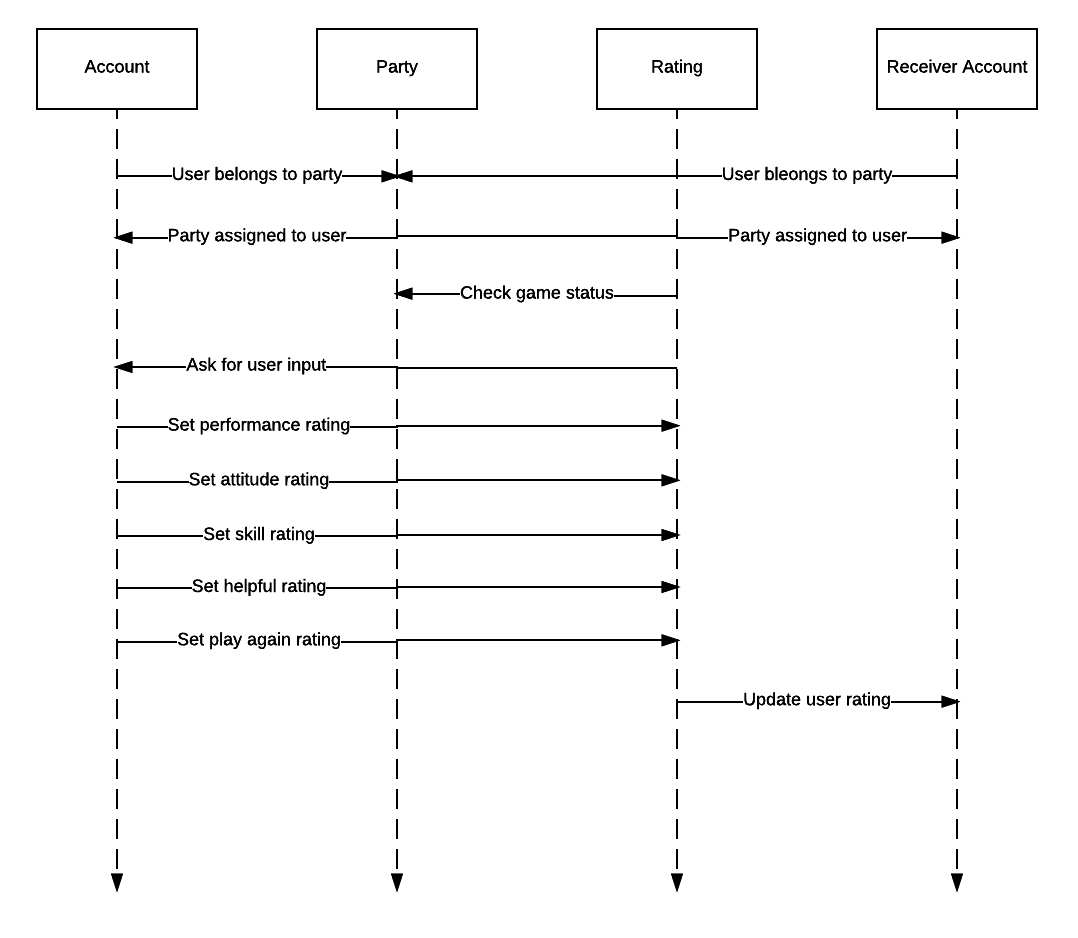
**Login/Registration**

**Enter Queue (Matchmaking)**

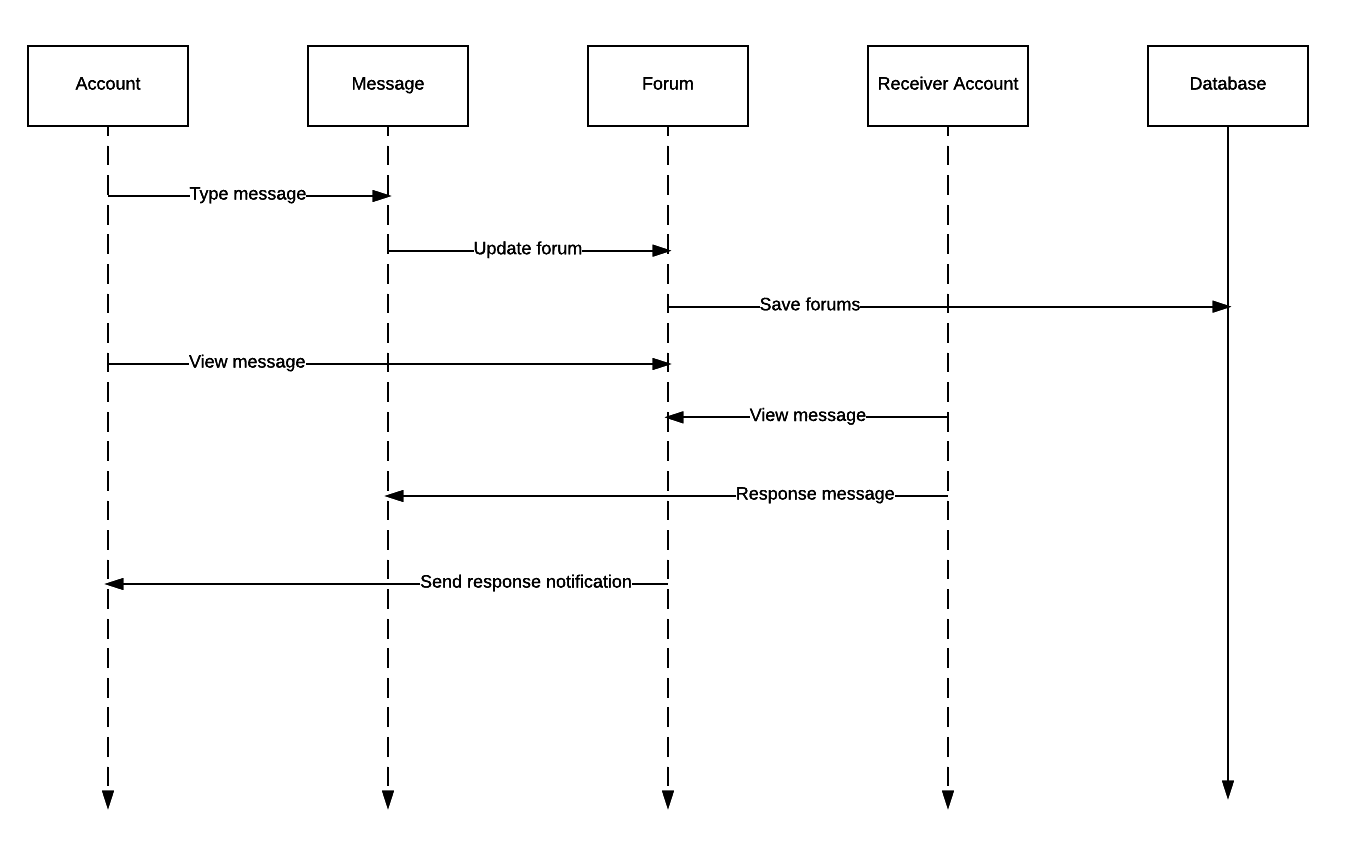
**Add Partner (Friend)**



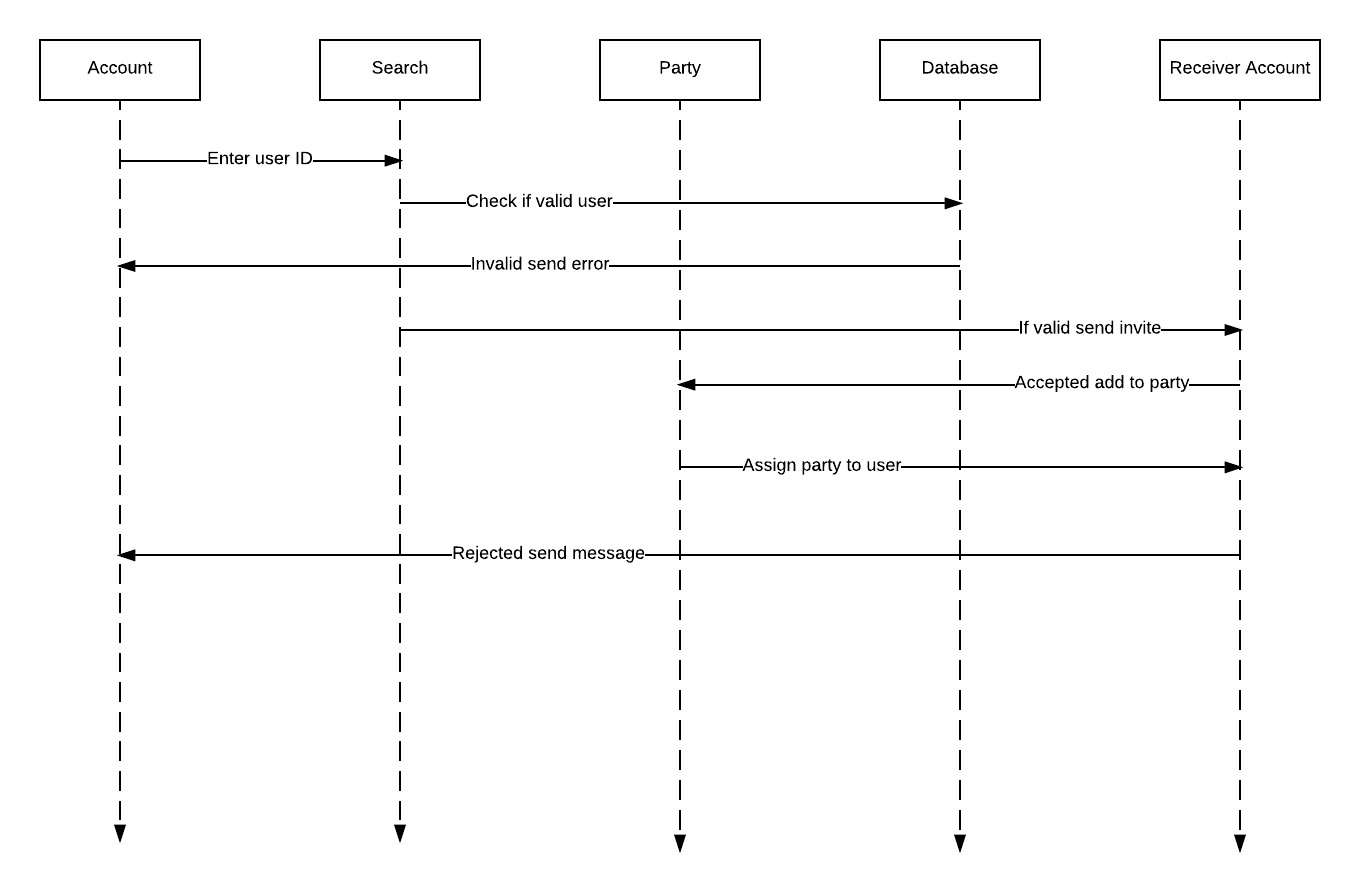
**Rating System**



**Public Posting**



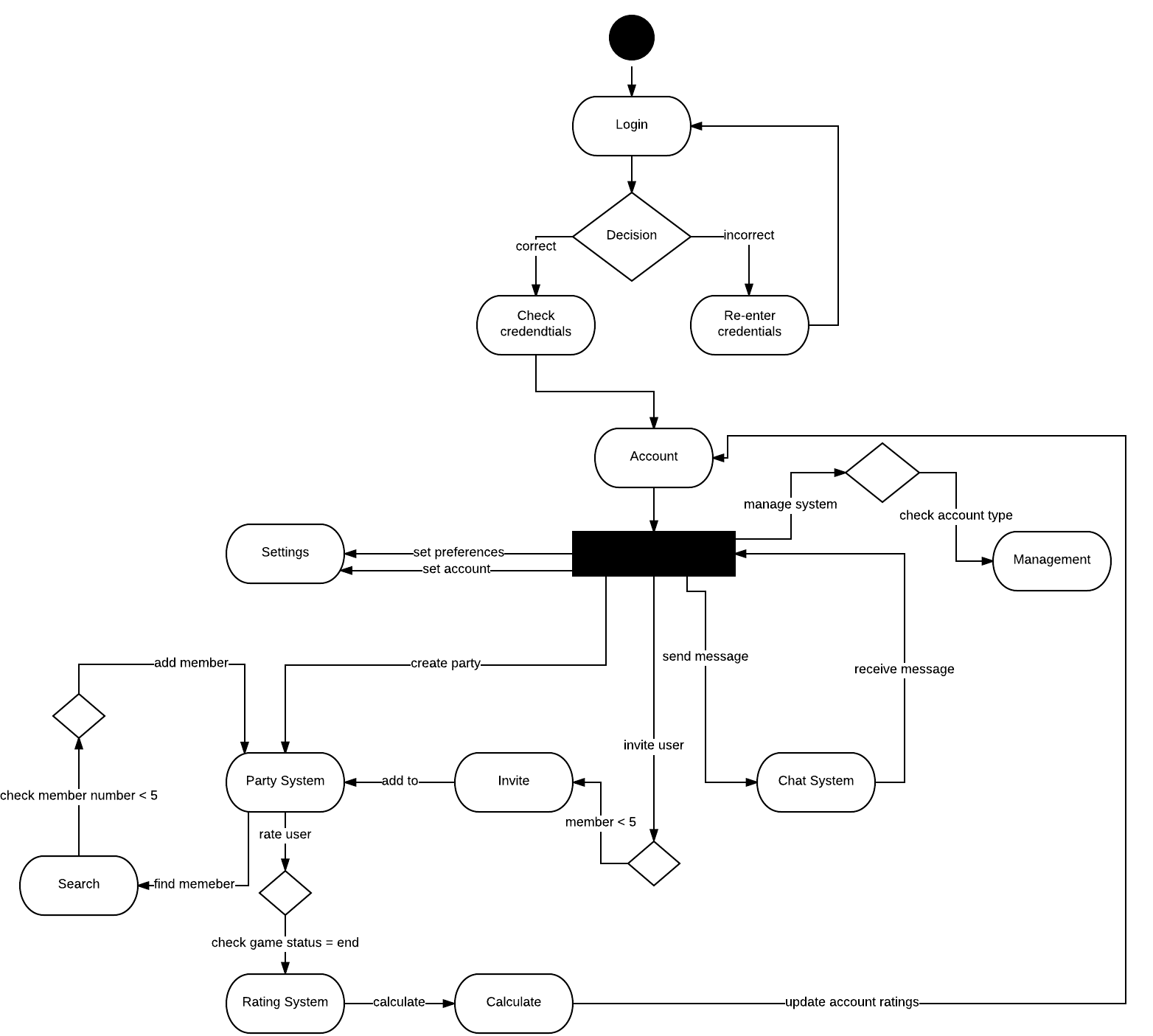
**Invite User**



### 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

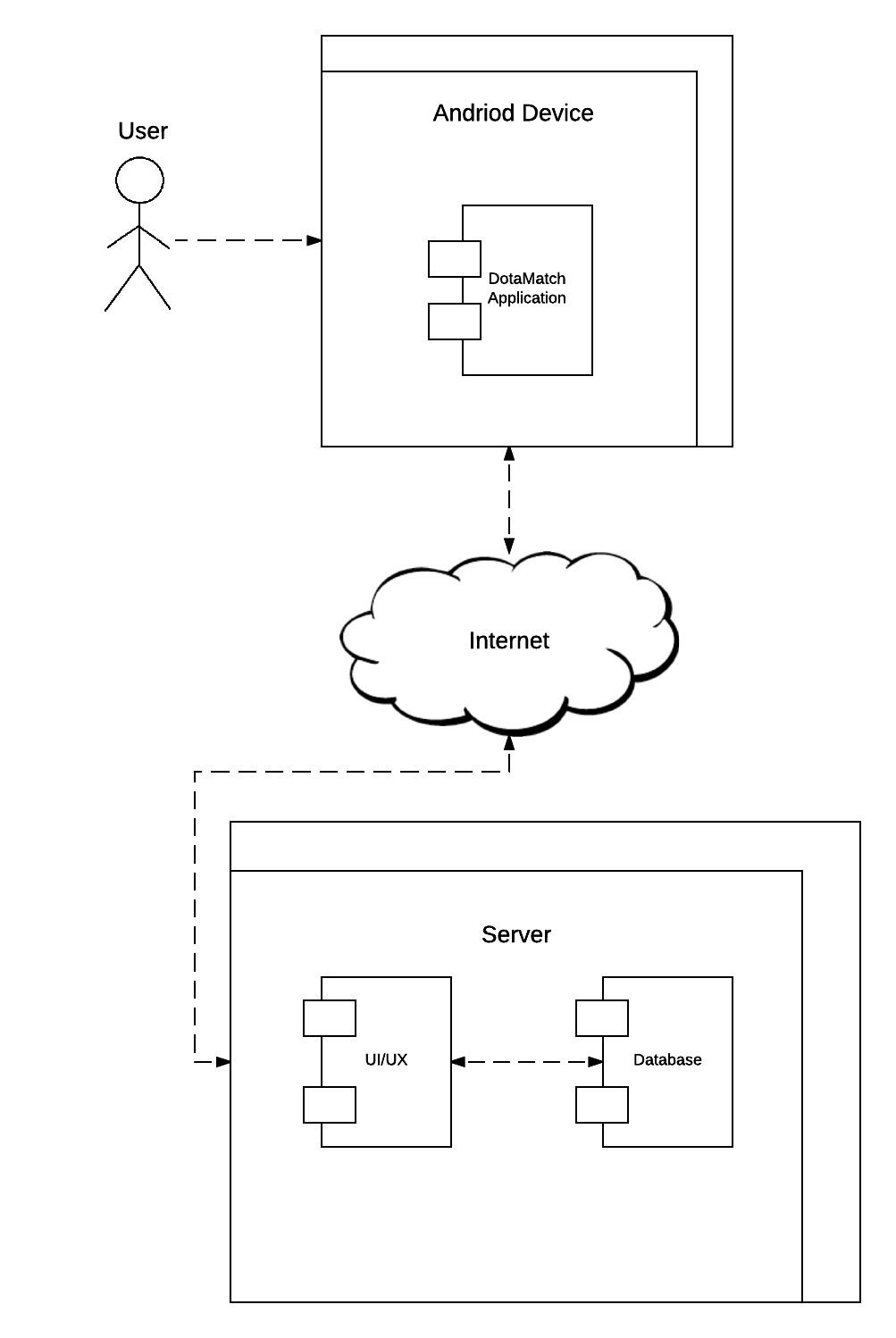
Not required.

## 5.2 DIAGRAM OF PHYSICAL PACKAGING OF LOGICAL COMPONENTS

Not required.

# 6. DEPLOYMENT ARCHITECTURE

## 6.1 PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM



# 7. CODE (PSEUDO)

LoginActivity:

attemptLogin():

email = emailField.getText

password = passField.getText  
 If password field is empty and not valid:

Throw Exception

If email field is empty and not valid:

Throw Exception

private final String mEmail;

private final String mPassword;

UserLoginTask(String email, String password) {

  mEmail = email;

  mPassword = password;

}

class Account

public:

int Login\_id

string status

string preferences[5]

int rating[5]

int partners\_list[]

Chat\* chat\_system

Party party

void set\_preferences() {

  Ask user the first question

Provide user with possible answers for question

Set preferences[0] = user input

Ask user the second question

Provide user with possible answers for question

Set preferences[1] = user input

Ask user the third question

Provide user with possible answers for question

Set preferences[2] = user input

Ask user the fourth question

Provide user with possible answers for question

Set preferences[3] = user input

Ask user the fifth question

Provide user with possible answers for question

Set preferences[4] = user input

}

void message\_user(int sender\_id, int receiver\_id) {  
 Check with database for valid receiver\_id

If not valid throw exception

If valid check receiver\_id’s status with chat\_system->check\_status()

If status is equal to “Offline”

Tell sender that user is offline

Else

Create chat system object

Set chat\_initiator\_id to sender\_id

Set chat\_receiver\_id to receiver\_id

Both sender\_id and receiver\_id enter chat room

Receive user input for message

Send message to receiver\_id including sender\_id

}

void send\_partner\_request(int sender\_id, int partner\_id) {  
 Check with database for valid receiver\_id

If not valid throw exception

If valid send partner\_id a message

Wait for partner\_id’s response

If partner\_id declines partner request

Break

Else partner\_id accepts

Update sender’s partners\_list by adding new partner\_id

Update new partner’s partners\_list by adding sender\_id

}

void create\_party() {

Create new object party

Set party’s login\_id to creator’s login\_id

Set party’s party\_numer to 1

}

void invite\_user(Party party, int user\_id) {

Check with database for valid receiver\_id

If not valid throw exception

Send invite to user\_id

Wait for user\_id’s response

If decline

Tell sender that user\_id declined

Break

Else

Check which party’s userX\_id slot is open

Set free userX\_id to user\_id

Set user\_id’s party\_id to current party ID

Increase party\_number by 1

}

void set\_rating(Party party, int user\_id) {

Check if user\_id matches one of the userX\_ids

Get time since party was created

Get party status

If party status does not equal “Game Finish” or time is not greater than 45 minutes

Tell rater that game has not finished

Else

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[0] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[1] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[2] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[3] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[4] = user input

}

};

class Party

public:

int party\_id

int queue\_id

int login\_id

int user2\_id

int user3\_id

int user4\_id

int user5\_id

int party\_number

int rating

string party\_preferences[5]

bool check\_party\_member() {

Check if party\_number is equal to 5.

If so return true

Else

Return false

}

void invite(int receiver\_id) {

Check for valid user id.

If valid send user an invite.

add\_member\_to\_group()

Else

Throw exception

}

void add\_member\_to\_group(int\* user\_id) {

Check current party\_number

If party\_number is less than 5

Check if user2\_id is used

Check if user3\_id is used

Check if user4\_id is used

Check if user5\_id is used

Assign user\_id to unused userX\_id

Assign user\_id to party\_id

Change user\_id’s party field to true

Else

Throw exception

}

void remove\_member(int user\_id) {

Check which userX\_id corresponds to user\_id

Set userX\_id to NULL

Set user\_id’s party\_id to NULL

Remove user\_id from party lobby

}

void start\_queue() {  
 Get party’s current number of members by using function check\_party\_member()

Create an int spots to hold 5 - check\_party\_member()

Set search classes’ initiator\_party\_id to this party\_id

Search for other parties in queue

If other party’s total number of members is equal or less than spots

Add party\_id to partyX\_id

Else if other party’s total number of members is greater than spots

Continue searching

If spots is equal to 5

Tell all party members to start game

}

void set\_party\_preferences() {

Ask user the first question

Provide user with possible answers for question

Set party\_preferences[0] = user input

Ask user the second user input

Provide user with possible answers for question

Set preferparty\_preferencesences[1] = answer

Ask user the third user input

Provide user with possible answers for question

Set party\_preferences[2] = answer

Ask user the fourth user input

Provide user with possible answers for question

Set party\_preferences[3] = answer

Ask user the fifth user input

Provide user with possible answers for question

Set party\_preferences[4] = user input

}

string get\_party\_preferences() {  
 Check if preferences array is filled

If not

Throw Exception

Else

Return party\_preferences array

}

void set\_rating(Party party, int user\_id) {

Check if user\_id matches one of the userX\_ids

Get time since party was created

Get party status

If party status does not equal “Game Finish” or time is not greater than 45 minutes

Tell rater that game has not finished

Else

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[0] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[1] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[2] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[3] = user input

Ask rater the first rating question

Get user input on a scale of 1-5

Set user\_id.rating[4] = user input

}

};

# 8. DICTIONARIES

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| User | Users of DotaMatch must register an user account before using our services. Upon registration and logging in, the user will have access to parties, chat system, friends list, and rating. | set\_preferences() | login\_id |
| message\_user() | dota\_game\_name |
| send\_partner\_request() | status |
| create\_party() | preferences |
| invite\_user() | rating |
| set\_rating() | partners\_list |
|  | chat\_system |
|  | party |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| set\_preferences() | Allows user to set preferences for players the user wants to play with. Preferences are taken into account while searching for party members. | User | N/A |
| message\_user() | Allows the user to send messages to a receiver\_id. Allows a receiver to send response messages to the sender\_id. Messages are sent through the chat\_id. | User | Chat chat\_id  int sender\_id  int receiver\_id |
| send\_partner\_request() | Sends a request to another user asking to add them to the user’s partners\_list. | User | int sender\_id  int partner\_id |
| create\_party() | Creates a party with the creator’s user\_id set as the party leader. | User | N/A |
| invite\_user() | Allows for the party leader to invite other users into the party if party number is equal or greater than 5. | User | Party party  int user\_id |
| set\_rating() | Allows user to rate another users once gaming sessions has concluded. | User | Party party  int user\_id |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| login\_id | Unique account identifier  Login name credential | Simple | int |
| dota\_game\_name | Unique Dota account identifier | Simple | string |
| password | Login password credential | Simple | string |
| partners\_list | Friends list | Simple | int array |
| party | Party contains a maximum of 5 users. | Complex | Class Party |
| chat\_system | Chat system between two or more usres. Allows for sending and receiving of messages between users. | Complex | Class Chat |
| rating | Rating is calculated through feedback from other users. Identifies the type of player this user is based on average of ratings. Ratings will be taken in account while looking for party members. | Complex | Class Rating |
| preferences | Preferences for the type of players the user is looking to play with. Preferences are taken into account while searching for party members. | Complex | Class Perferences |

**Relationship**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Set preferences | User must set preferences | User | Preferences | Optional | 2 |
| Rating | Users have their own ratings based on a feedback system used by other users | Ratin | User | Mandatory | 2 |
| Login | Users must have login credentials to access their account and its functions | Database | User | Mandatory | 3 |
| Chat | Users can send and receive messages from other userse | Users | Chat\_System | Optional | 3 |
| Partners Request | Users can send other users a friend request | User | User | Optional | 2 |
| Party | Users can create a party | User | Party | Optional | 3 |

**Key Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Create Party | User creates a party | | Search for other users to play a gaming session with. | | Create party object. Set user’s party\_id to party object. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Check if user is already in a party. | | Enter party lobby | | User enters party lobby | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Party Leader | User that created a party becomes the party leader | | Have a leader for each party who can invite, remove, and start the queue. | | Create party leader object and set user\_id to it. Set party\_id to party object. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Check if user is already in a party. | | Set party leader object to user. | | User enters party lobby.  User obtains party leader symbol. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Login | User must enter valid credentials in order to gain access to account an account and its functionalities. | | Prevent unauthorized access | | Check input crendentials. Valid if credentials match stored login and passwords of an account. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Input credentials much match credenetials saved in the database. | | If valid grant user access to account and functionatlies.  If invalid ask user to retry. | | Change from login page to user account page. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Message | Allow users to send or receive messages from another user. | | Allow for communication between users. | | Create chat system object. Have two or more users be linked to the chat system object. Messages can then be transferred through the chat system. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must be valid user. | | Messages will be sent to users. | | Users will enter chat system’s message page. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Partner Request | Send a friend invite to antoher user | | Allow for a user to make friends they can play with on a consistent basis. | | User will send a partner request to another user. Receiver can either accept or reject the invite. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must be valid user | | If receiver accepted the invite, both users update their partner’s list.  Else respond with rejection message. | | If accepted partner’s list is updated.  Else no state change. | |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| party | Party is a waiting lobby for users who would like to play a game session together. Party contains a maximum of 5 users. Party is placed into the matchmaking queue to search for potential members. Party leader can set preferences for the type of player to search for. | check\_party\_members() | party\_id |
| invite() | queue\_id |
| remove\_member() | login\_id |
| start\_queue() | user2\_id |
| set\_party\_preference() | user3\_id |
| get\_party\_preferences() | user4\_id |
| set\_rating() | user5\_id |
|  | party\_number |
|  | rating |
|  | party\_preferences |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| check\_party\_members() | Returns current number of party members | Party | N/A |
| invite() | Send an invite request to user\_id | Party | int user\_id |
| remove\_member() | Remove a party member from the party | Party | int user\_id |
| start\_queue() | Enter queue into matchmaking system. Search for other party members. | Party | N/A |
| set\_party\_preference() | Allow party leader to set preferences for the type of user they would like to play with. | Party | N/A |
| get\_party\_preferences() | Return preferences | Party | Preference preferences |
| set\_rating() | Allow player members to rate other party members once game session has ended. | Party | Rating rating |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| party\_id | Identifier for the party | Simple | int |
| queue\_id | Identifier used while inside matchmaking queue | Simple | int |
| party\_leader\_id | Identifier for the user who created the party | Complex | Class Party\_Leader |
| user2\_id | Identifier for second party member | Simple | int |
| user3\_id | Identifier for third party member | Simple | int |
| user4\_id | Identifier for fourth party member | Simple | int |
| user5\_id | Identifier for fifth party member | Simple | int |
| party\_number | Member of members in party | Simple | int |
| rating | Users rate fellow party members after a gaming session. Ratings are stored in the rating class object. | Complex | Class Rating |
| party\_preferences | Preferences taken into account while searching for other party members. | Complex | Class Preference |

**Relationship**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Set preferences | Party leader must set preferences | Party Leader | Preferences | Optional | 3 |
| Remove member | Party leader can remove a party member from the party | Party Leader | User | Optional | 3 |
| Invite Member | Party members can invite other users to the party | Party | User | Optional | 2 |
| Rate | User rates another party member | User | Rate | Mandatory | 2 |

**Key** **Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Matchmaking Queue | Enter party\_id into queue to start for party members | | Add additional users to party | | Search for other party members by check how many members is needed |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Check current number of party members.  Take into account preferences before looking for party members | | Add new user to party and set their party\_id to this party. | | Number of members increase by amount of new users. Take party out of matchmaking queue once party is complete. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | **Action** | |
| Invite | Add user to party | | Add additional users to party | Send invite to targeted user. Check for user’s response. If accepted add user to party. Else receive decline message. | |
| **Pre-conditions** | | **Post conditions** | | | **State Change** |
| Check current number of party members.  Check if user is a valid user.  Check is user is already in a party | | Wait for response.  If accept increase party member number by one and add user to party. Else receive decline message. | | | Number of members increase by amount one. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | **Action** | |
| Rate | User rates another party member | | Feedback system for users | Allow user to answer specific questions. Get user input. Update rating object’s ratings for targeted user. | |
| **Pre-conditions** | | **Post conditions** | | | **State Change** |
| Game session must have ended. | | Save user inputs for targeted user’s rating. | | | Take input from user object and update rating object’s variables. |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| Party Leader | User who creates a party will be the designed party leader. Party leader has the ability to invite, remove, and enter the party into the queue. | invite\_member() | login\_id |
| remove\_party\_member() | party\_id |
| start\_queue() |  |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| invite\_member() | Send an invite request to user\_id | Party | int user\_id |
| remove\_party\_member() | Remove a party member from the party | Party | int user\_id |
| start\_queue() | Enter queue into matchmaking system. Search for other party members. | Party | N/A |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| login\_id | Identifier for the user | Simple | int |
| party\_id | Identifier for the party | Simple | int |

**Relationship**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Assign Party Leader | Assign party leader to user who created party | User | Party Leader | Mandatory | 2 |
| Assign Party | Assign party leader to party | Party Leader | Party | Mandatory | 2 |

**Key** **Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Assign Party Leader | Assign party leader to user who created party | | Every party needs a leader to start, invite, and remove members | | Party leader object is assigned user\_id  User object is creates party leader object |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Check if user is a valid user.  Check is user is already in a party | | Assign party leader to user | | N/A | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Assign Party leader to party | Assign party leader to party | | Every party needs a leader to start, invite, and remove members | | Party leader object is assigned party  Party object is assigned party leader |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Check if user is a valid user.  Check is user is already in a party | | Assign party leader to party | | N/A | |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| Queue | Party leader enters the party into the queue. While in the queue, it searches for other player members to join the party. Once party slots have filled up remove the party from the queue. | check\_preferences() | queue\_id |
| find\_matches() | first\_party\_id |
| exit\_queues() | second\_party\_id |
|  | third\_party\_id |
|  | fourth\_party\_id |
|  | fifth\_party\_id |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| check\_preferences() | Get preferences from party. Searching for other members takes into account the set preferences. | Queue | int party\_id  Preference preferences |
| find\_matches() | Search for other parties in the queue | Queue | N/A |
| exit\_queue() | Leave queue once all slots are filled | Queue | N/A |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| queue\_id | Identifier for the queue | Simple | int |
| first\_party\_id | Identifier for the first party | Simple | int |
| second\_party\_id | Identifier for the second party | Simple | int |
| third\_party\_id | Identifier for the third party | Simple | int |
| fourth\_party\_id | Identifier for the fourth party | Simple | int |
| fifth\_party\_id | Identifier for the fifth party | Simple | int |

**Relationships**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Queue Slots | Each queue should have a maximum of 5 players. Thus all total number of players in all party\_ids must be equal to 5 or less. | Party | Queue | Mandatory | 2 |

**Key** **Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Check number of players | For each party\_id get the number of players for that party. Total all number of players in the current queue. If number of players equals 5 leave queue. | | Makes sure party total is 5 or less. | | Look at the number of players in each party. Return total number. Leave queue if its equal to 5. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must have at least 1 person for party to be in queue. | | If there is 5 total players leave queue. | | Enter queue to search for player. Once player maximum has been reached leave queue. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Leave Queue | Leave the queue (stop searching for players). | | Makes sure party total is 5 or less. | | Remove queue\_id form the search. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must have 5 players total in queue | | Invite each party member in Dota to start a game | | Exit queue into party lobby | |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| Preferences | Contains attributes the user can set. These attributes are the type of players the user wants to play with. Attributes are takening into account during matchmaking | N/A | log\_in\_name |
|  | party\_id |
|  | skill\_rating |
|  | role |
|  | attitude |
|  |  |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| N/A | N/A | N/A | N/A |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| log\_in\_name | Identifier for the User | Simple | int |
| party\_id | Identifier for the party | Simple | int |
| skill\_rating | Identifier for the second party | Simple | int |
| role | Identifier for the third party | Simple | int |
| attitude | Identifier for the fourth party | Simple | int |

**Relationships**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Account Preferences | Allow users to set personal preferences | User | Preferences | Optional | 2 |
| Party Preferences | Allow party leader to set party preferences | Party Leader | Preferences | Optional | 3 |

**Key** **Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Party Preferences | Party leader sets preferences for the type of player to search for while in queue. | | Add parameters to queue search so players matched will be the close to the preferences set. | | Queue was take into account the preferences when looking for other parties. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must be in a party | | Matched player will be added to the party | | N/A | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| User Preferences | User sets preferences for themselves. | | Add parameters search so players matched will be the close to the preferences set. | | Queue was take into account each user’s preferences when looking for other parties. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Must be a valid account | | Matched player will be added to the party | | N/A | |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| Rating | Contains different attributes with ratings on a scale of 0-5. Other players rate fellow party members after a game session is completed. | set\_rating() | rater\_id |
| check\_rating() | rated\_id |
| check\_game status() | performance[] |
|  | attitude[] |
|  | skill[] |
|  | helpful[] |
|  | matchagain[] |

**Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| set\_rating() | Allows user to set rating for fellow party member | Rating | int user\_id |
| check\_rating() | Queue system retrieves the ratings of matched player | Rating | int user\_id |
| check\_game status() | Checks if game has been finished | Rating | N/A |

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **C/S** | **Type** |
| rater\_id | Identifier for the user that is rating the targeted user. | Simple | int |
| rated\_id | Identifier for the user that is being rated. | Simple | int |
| performance[] | Attribute for how well the user performed in game | Simple | int array |
| attitude[] | Attribute for the attitude of the player while in game | Simple | int array |
| skill[] | Attribute for how well the user’s skill matched the user’s advertised skill level | Simple | int array |
| helpful[] | Attribute for how helpful the user was in game | Simple | int array |
| match\_again[] | Attribute for if the user would like to play with this user again | Simple | int array |

**Relationships**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From class** | **To class** | **Optional/Mandatory** | **Cadrinality** |
| Set Rating | Allows user to set rating for fellow party member | User | Rating | Mandatory | 2 |
| Check Rating | Get the average rating of a user | Rating | User | Mandatory | 3 |
| Game Status | Check game status for whether game has concluded or not | Party | Rating | Mandatory | 2 |

**Key** **Events**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Set Rating | Allows user to set rating for fellow party member once the game has concluded | | Feedback system | | Allow user to input on a scale of 0-5 values for each attribute. Update targeted user’s ratings with new values. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| Game must have concluded.  Targeted user must be in the same party and game. | | Update target user’s rating with new values. | | User rating is updated. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | | **Motive** | | **Action** |
| Check Rating | Get the average rating of a user. Use this rating to find a close to perfect match for a party. | | Find the most optimal players for each party’s preferences. | | Check rating of user. If it matches the party’s preferences add the user to the party. |
| **Pre-conditions** | | **Post conditions** | | **State Change** | |
| User must be inside the queue search for parties. | | If party is matched add user to party. | | If party is matched exit queue and join party lobby. | |

# 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 |
| Software Design Document (SDD) Version 1.1 | Corey Chong | 26 | 42 | 16 |
| Software Design Document (SDD) Version 1.1 | Evans Yeung | 24 | 39 | 15 |
| Software Design Document (SDD) Version 1.1 | Albert | 20 | 38 | 18 |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated | Actual | Difference |
| Corey Chong | 54 | 72 | 18 |
| Albert Su | 48 | 69 | 21 |
| Evans Yeung | 53 | 74 | 21 |

## 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 |
| Software Design Document (SDD) Version 1.1 | Corey Chong | 110 | 73 | 37 |
| Software Design Document (SDD) Version 1.1 | Evans Yeung | 84 | 54 | 30 |
| Software Design Document (SDD) Version 1.1 | Albert | 80 | 71 | 9 |

**Cumulative**

|  |  |  |  |
| --- | --- | --- | --- |
| Who (individual or Team) | Estimated | Actual | Difference |
| Corey Chong | 186 | 155 | 31 |
| Albert Su | 166 | 171 | 5 |
| Evans Yeung | 161 | 141 | 20 |

## 14.5 GANTT CHART

