Design Document for Assassins

Group#LA-05

MemberName: Tyler Fenton

MemberName: Paul Martinson

MemberName: Nathan Sorlien

MemberName: Scott Rowekamp

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Change** |
| 0.1 | 10/08/16 | SM | Initial Document |
| 1 | 10/22/16 | LA-05 | 05\_Design\_Interfaces |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 3](#_Toc84977455)

[1.1 Purpose 3](#_Toc84977456)

[1.2 Scope 3](#_Toc84977457)

[1.3 Definitions, Acronyms, Abbreviations 3](#_Toc84977458)

[1.4 Design Goals 3](#_Toc84977459)

[2 References 4](#_Toc84977460)

[3 Decomposition Description 5](#_Toc84977461)

[3.1 Module Decomposition 5](#_Toc84977462)

[3.2 Concurrent Process 5](#_Toc84977463)

[3.3 Data Decomposition 5](#_Toc84977464)

[3.4 STATES 5](#_Toc84977465)

[4 Dependency Description 6](#_Toc84977466)

[4.1 Intermodule Dependencies 6](#_Toc84977467)

[4.2 InterProcess Dependencies 6](#_Toc84977468)

[4.3 Data Dependencies 6](#_Toc84977469)

[5 Interface Description 7](#_Toc84977470)

[5.1 Module Interface 7](#_Toc84977471)

[5.2 Process Interface 7](#_Toc84977472)

[6 Detailed Design 8](#_Toc84977473)

[7 Design Rationale 9](#_Toc84977474)

[7.1 Design Issues 9](#_Toc84977475)

[7.2 <Issue 1> 9](#_Toc84977476)

[7.3 <Issue 1> 9](#_Toc84977477)

[8 Traceability 11](#_Toc84977478)

# Introduction

## Purpose

[NONE]

## Scope

[NONE]

## Definitions, Acronyms, Abbreviations

|  |  |
| --- | --- |
| Term | Description |
|  |  |
|  |  |

## Design Goals

1. Response Time   
   -Server must update each client’s location quickly in order for the player to effectively track down his target. Goal time is to have location information distributed every three seconds.
2. Reliability  
   - How much the game can be counted upon to work well. Expectations is that the game functions well and doesn’t crash lose connection.
3. Capacity

- The server should be able to run 10 games with a max of 10 players per game.

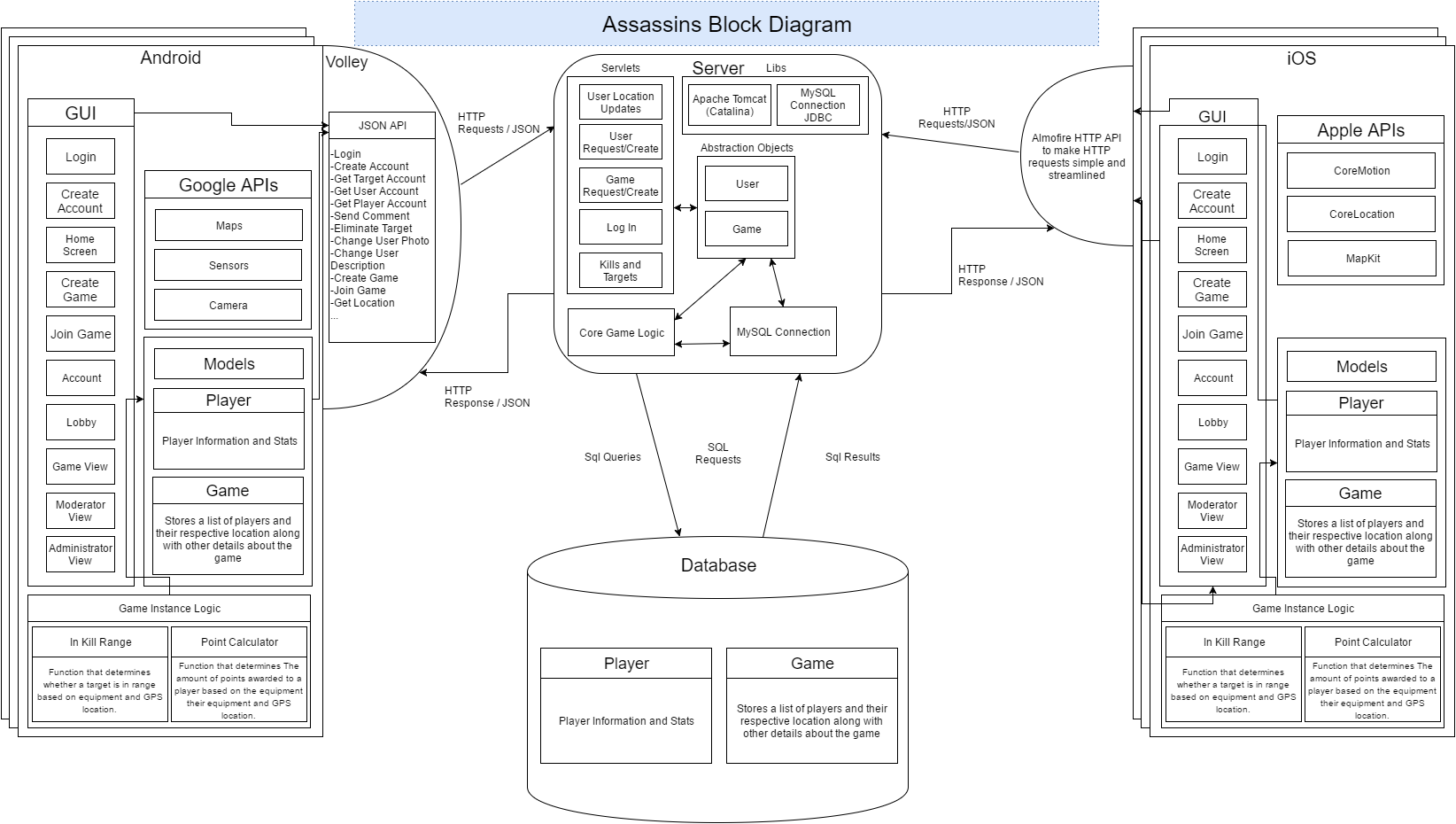
1. Throughput  
   - The server must to be able to have a single game allow a maximum of 10 players.
2. Maintainability  
   - We want to be able to make the app work for every sequential release of the Android and iOS operating systems.
3. Extensibility   
   - We want to be able to add new game features and modes in the future.

# References

[NONE]

# Decomposition Description

## Module Decomposition



### Android/iOS User GUI All user interface actions will happen on a single asynchronous thread.  HTTP requests and calculations that don’t happen instantly will be processed on a separate thread so that user interface does not get stuck.  The main thing that will be happening graphically on the client side is updating the compass to point in the correct direction of the target with respect to heading of the phone. The user will interact with the GUI by tapping obvious buttons to navigate its screens and menus.

### Game Instance Logic To avoid server stress, the game will have instances of players and the game on the client side, this way it can gather information and perform calculations on screen. It also allows for the server to only be requesting and pushing out information rather than constantly calculating functions.

### Android Communication Will be using Volley to manage requests between the Server and the front end of Android. The data will be sent as HTTP requests.

### iOS Communication The Alamofire api will make HTTP requests to the server and process the returned JSON data which is then passed to the game for processing.

### Apache Tomcat Server An Apache Tomcat server will run various servlet applications to handle data from the clients and interact with the database. There will be servlets for the basic network functions of the app including creating/accessing user accounts, creating/accessing games, and sending/receiving kill and location data. The servlets will create and access abstracted java objects that will be stored in the database and processed by the core game logic. The game logic will assign targets to each player and update the game state when players join and leave. It will also update the individual players’ stats.

### SQL Database Creates tables and stores values for Player info as well as dynamically holding information and updating it while a game is in progress.

### Player:

### The player table stores login information which contains the username and password of a player. Also stores an image of the player to be used in game for identification purposes. It will store how many games the player has played as well as his wins and total experience value. It can be used to implement another parameter when searching for players in Join Game.

### Game:

### The Game table creates a table that holds necessary data upon creation of a game. It will hold player data that can be used to reference their current location using GPS. The table also contains the image provided by the player so that it can be seen on the game menu. It will contain the game key which can be used by players to join the game directly. This table will also contain a boolean that decides if the player is alive or dead. The game logic can then reassign targets. Finally, it will contain the game duration, and size of the game.

## Concurrent Process

[NONE]

## Data Decomposition

[NONE]

## STATES

### [NONE]

# Dependency Description

## Intermodule Dependencies

[NONE]

## InterProcess Dependencies

## [NONE]

## Data Dependencies

## [NONE]

# Interface Description

## Module Interface

### Authentication Interface

#### JSONObject login(String username, String password) - Returns a JSONObject containing the result of the login and also a JSONObject representing the user if successful

#### JSONObject createAccount(String username, String password, String realName, String Base64Image) - Returns a JSONObject containing the result of the create account operation and also a JSONObject representing the user if successful

#### JSONObject changePassword(String username, String oldPassword, String newPassword) this function will return an object that specifies whether or not the password was able to be changed.

### Create Game Interface

#### createGame(String gameID, String password, float xCenter, float yCenter, int radius, int hostID, int durationS) - createGame will call the game initialization servlet which will create a game table with the given information.

#### joinGame(string gameID, String password) - attempts to add the user to the gameID’s player list. This method will then return the game settings and information to the client.

### Lobby Interface

#### JSONObject getPlayers(float xLocation, float yLocation, String gameID) - returns the current list of players and their locations from the server to display on the client’s device. Also sends the current location of the client to the database. If the game has started, it returns the target and start time.

#### JSONObject gameStart(String currentTime, String gameID) - Returns a JSONObject containing the host’s target and the end time.

### Game Interface

#### JSONObject updateGame(String gameID, int playerID, float xLocation, float yLocation) - returns a JSONObject containing the user’s target and if the user was killed. All living players will make this request every 3 seconds.

#### kill(String gameID, int playerID) - Removes the killed player from the linked list of players.

#### finishGame(gameID) - Called by the last user alive or when the end time is reached. Updates player stats and clears the game’s data from the database.

### Database Communication - We are using the JDBC API to execute queries and updates on the provided database server.

#### Users Table Columns:

|  |  |
| --- | --- |
| id | int(11) AI PK |
| username | varchar(45) |
| password | varchar(45) |
| real\_name | varchar(45) |
| image\_filename | varchar(64) |
| total\_kills | int(11) |
| games\_played | int(11) |

5.1.5.2 Games Instance Table Column:

|  |  |
| --- | --- |
| id | int(11) AI PK |
| gameID | varchar(45) |
| playersList | varchar(45) |
| playersAlive | varchar(45) |
| password | varchar(45) |
| xCenter | float |
| yCenter | float |
| radius | int(11) |
| endTime | varchar(45) |

5.1.5.3 Games Table Columns:

|  |  |
| --- | --- |
| id | int(11) AI PK |
| gameID | varchar(45) |

## Process Interface

### HTTP Interface

#### Handled by Tomcat – Clients will connect to the server using specific URLs for each function the server provides. Once the server gets a request, it will send the requested data from the database to the client.

# Detailed Design

NOT REQUIRED

# Design Rationale

## Design Issues

[NONE]

## [NONE]

### Description

### Factors affecting Issue

### Alternatives and their pros and cons

### Resolution of Issue

# Traceability

|  |  |  |
| --- | --- | --- |
| **No** | **Use Case/ Non-functional Description** | **Subsystem/Module/classes that handles it** |
| 1 |  |  |
| 2 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

FEEL FREE TO ADD APPENDICES AS NEEDED. UPDATE TOC BEFORE SUBMITTING