Software Design Description (SDD)

Grand Slam Baseball All-Stars 2: Electric Boogaloo

COP4331C, Spring, 2018

Team Name: Team MLB Card Game 2

Team Members:

- Daniel Carman
- Malik Henriquez
- Ronald Marrero
- Brian Wengier
- Kaleb Yangson

1. Introduction

1.1. Project Overview

Our client is the administrator of a baseball card collectors' association. The client's association comes together to discuss their hobby, to trade cards, and to meet like minded hobbyists in their area.

The market for trading cards has become increasingly diluted in recent decades with the advent of new trading card games such as Magic: The Gathering and Yugioh. As such, our client is focused on retaining and expanding their association in the face of this dilution. In order to compete with newer, more interactive trading card games, our client has proposed that we create an online baseball trading card game in the Unity game engine that will utilize a baseball trading cards as game pieces. Our client gave us great freedom regarding the genre of the game and the content in it, so the direction we decided to take the idea of a baseball card game is essentially a "Baseball Manager Simulator". The gameplay will consist of players creating teams with currently owned player and item cards, pitting their teams against others in asynchronous online play, and opening card packs to acquire new players and items.

Our system will have 2 expected user classes: players and admins. Players will be able to create an account, log in, change various settings on their local device, build teams, purchase

packs, and play games. Admins will be able to view all data on the user/card data repository. The role of the admins is to oversee the maintenance and proper functioning of the core game and the data repository.

1.2. Project Scope

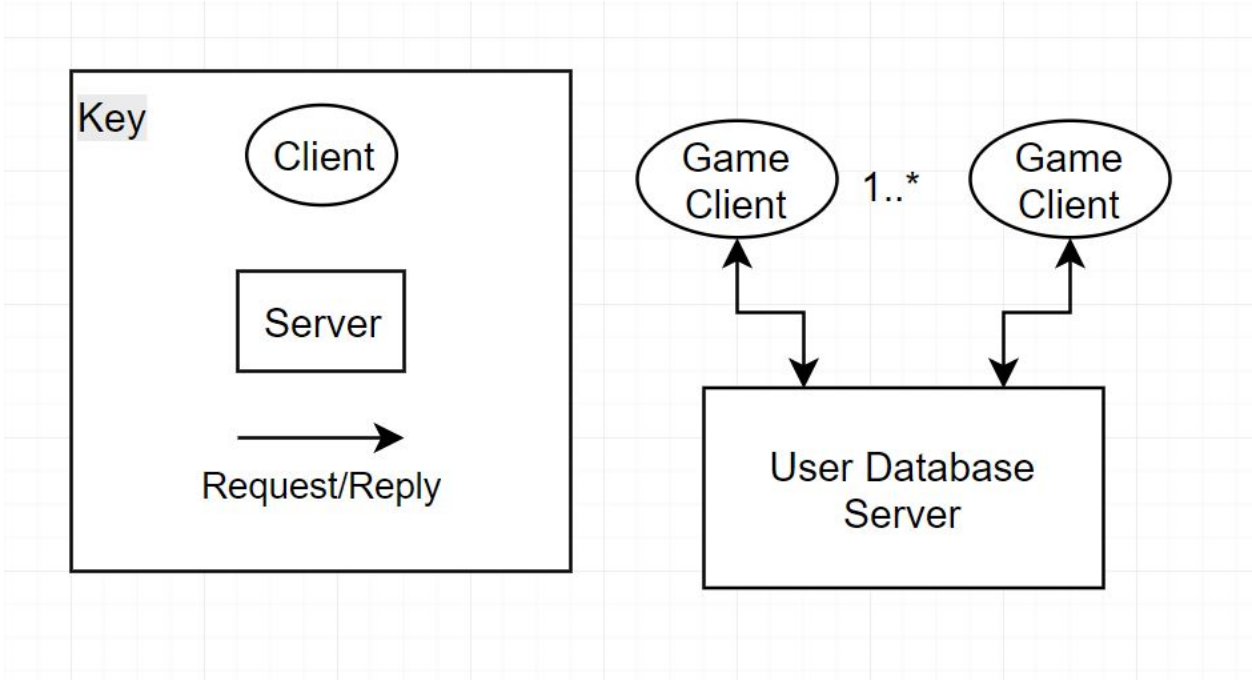
Our system is composed of 3 major components; the game frontend, the game backend, and the data repository, each with their own required features. The front end of the game must allow users to navigate through menus, display game assets (such as cards or scenery), and interact with the system using their phone's touch screen. The backend of the system must give the system the ability to simulate games of baseball using player and item stats, randomly generate packs of cards or in game outcomes, and communicate with the data repository. The data repository must store all user information (such as username, password, and amount of in game currency), as well as all information pertaining to each in game card. The purpose of storing cards on this repository rather than locally on the system is so that the association will easily be able to add cards without requiring a new Google Play release.

The inputs to our system will be comprised solely of player touch screen input. After launching the application, players will be able to use their device's touch screen to interact with various elements on screen. The outputs of the system are the graphics and sound produced by the game. A single data store will be used for this project, which will house all data related to users and cards.

The only major constraint of this project comes from the target platform of the game; Android mobile phones. Android phones have increased in computational ability exponentially in the past years, however the resources that are available on modern smartphones (especially when considering a graphically intensive application such as a game) are less than that of a desktop. Additionally, the size of mobile phones will force us to accommodate to a platform where screen real estate is sparse.

2. Architectural Design (reference Chapter 5)

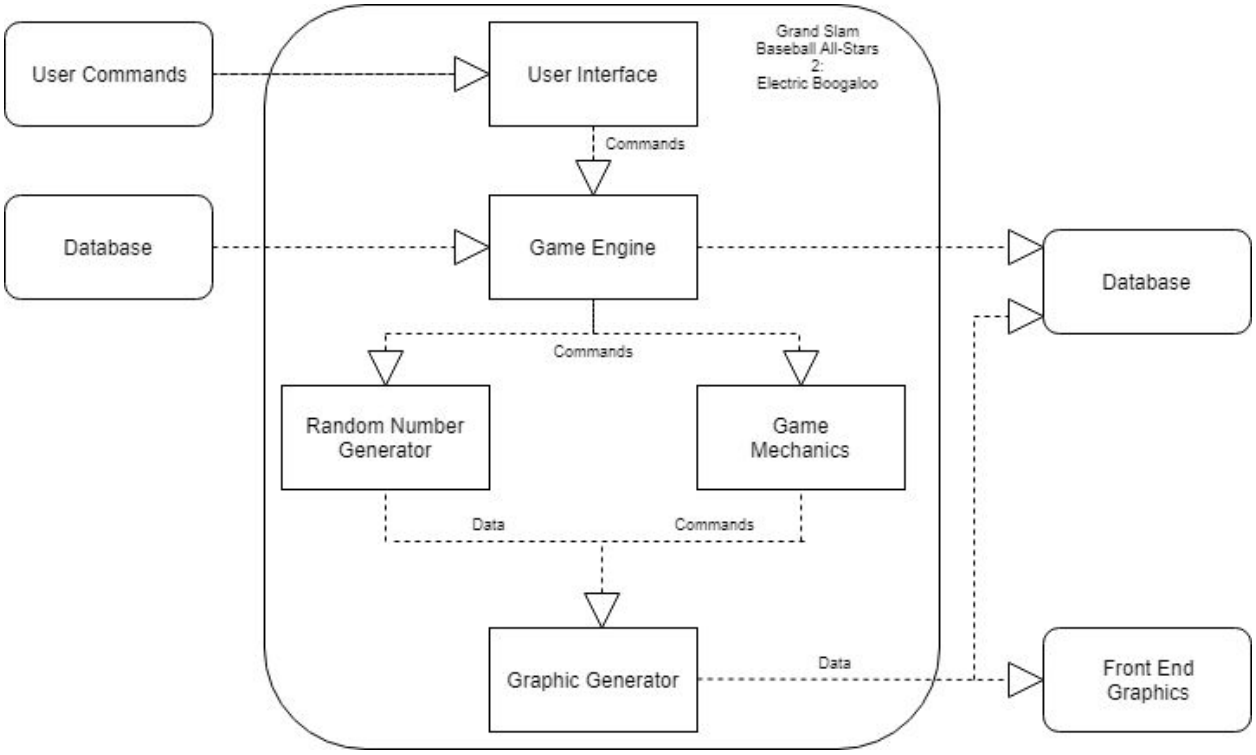
2.1. High-level Architecture.



The game client is the primary component of our system. It is comprised of all of the data and services required for the system to function, with the caveat of user data. One or more game clients can send requests to the user database repository, the other primary component of our system. The user database repository will then send an appropriate reply to the game clients' requests.

When starting out with the design of this project, we were torn over whether we should consider this a client/server style system or a repository system. In the end, our software largely falls under the repository model, as data changes on one part of the repository are always reflected system wide. Additionally, there isn't much of a need to distribute services across several servers, as the only thing that cannot be provided by the client application is user data and card data.

2.2. System-interface Architecture

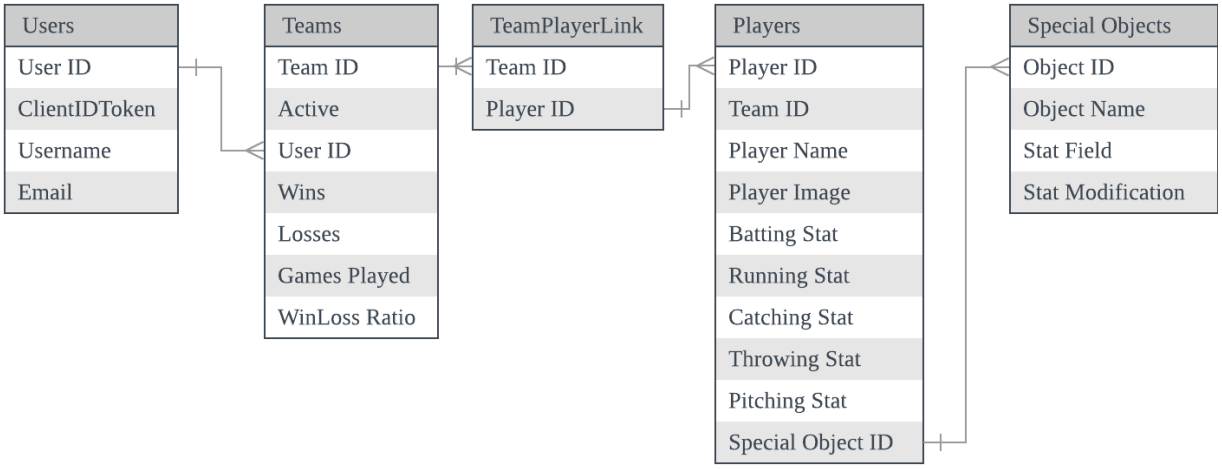


The User Interface represents the graphical interface that the User can see and interact with. With a User interacts with the User Interface, the UI sends commands to the Game Engine which exists to interpret what to do with User commands. The Game Engine, at the same time, is taking in data from the Database that it will use alongside the commands from the UI to send out commands to the Random Number Generator, Game Mechanics, and Database. The Random Number Generator will send data to the Graphic Generator. The Game Mechanics Interface will take the commands from Game Engine and use the game logic that we have given it to decide what happens within the game, then send commands to the Graphic Generator. The Graphic Generator then takes in the commands and data that are sent to it and generates graphics for the Front End Graphics and Database.

2.3. Data Design

Externally, there will be one database to maintain titled UMPIRE_DB which will contain multiple tables to store the user/team information and will be linked with foreign key constraints. This database is stored on an Azure Cloud server. The Android application will interface with a Web Service that queries the database and retrieves the requested information.

ER Database Diagram



Internally, the application will maintain a cache of the latest database information pertaining to that user. Only when database changes are made will the cache be updated. (By default, the application starts with no cache and first gets the latest data from the database according to the logged in user id.)

Additionally, the application will store the sound clips and images for the teams, players, and application functions. Each card that will get rendered to the user is a player that is stored directly on the database. All sound clips will be stored in the mp3 file format and all images will be stored in the png file format for space and consistency. These resources will only get updated on each new version of the application.

2.4. Alternatives Considered

Microsoft SQL Server was chosen for our backend architecture due to its proven efficiency and ease of use. Since it easily integrates with Microsoft Azure, the decision was made to move forward with this platform. One major issue in selecting MSSQL is Android feasibility. Google's real-time database, Firebase, has made app authentication and data integration very simple. However our application would work well on multiple platforms such as iOS and Web, instead of being restricted to Android. Thus for longevity, MSSQL was chosen as it is being interacted through a web service that any platform can call.

3. Detailed Design (reference Chapter 6)

3.1. Design Issues

For this project, it was decided to use Microsoft SQL as the backend database for the program. The alternative considered was Google's Firebase Realtime Database. During the discovery phase, two prototypes were created to perform simple lookups to databases housed on both systems. Using these prototypes, a comparison was made to determine the better selection.

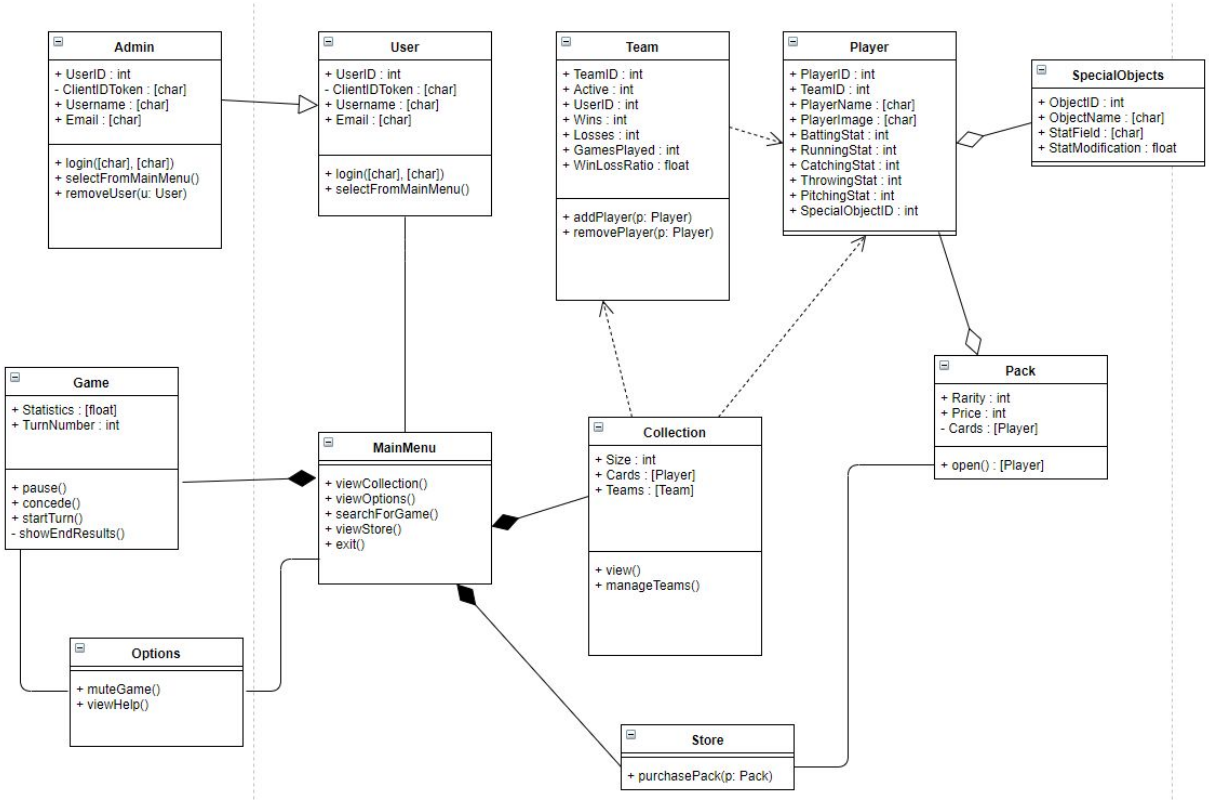
Since MSSQL allows for foreign keys, logically separated tables were created to store the information for users, teams and players. Queries performed in our first prototype were consistent and easy to interpret. Additionally, since the Android application interacts to the database through a web service, code reusability was maximized.

Conversely from MSSQL, Firebase sends requests using one single JSON tree. In the Firebase prototype, fewer code was needed to implement the database functionality as a result. The testing behaved similarly to the MSSQL tests with no noticeable differences.

Since both prototypes behaved similarly, the ultimate deciding factor was in programmability. With MSSQL, we can take advantage of creating stored procedures to execute SQL statements and return values that make sense for our needs. One good example of this is a stored procedure to return the number of teams saved without having to redo the computation each time. On small databases, calculating the code on the client-side would not be an issue but as our data scales, performing the computation on the server-side would provide the greatest efficiency. Additionally, we are able to take advantage of database triggers which are important in re-calculating the win/loss ratio percentage each time a team's wins or losses gets updated.

In conclusion, we decided on MSSQL as our database choice for the freedom it gives us and the efficiency it provides as our data scales. Firebase is still a relatively new database that does not completely fit into the team's design goals as MSSQL does.

3.2. Component *n* detailed design



3.3. Requirements Traceability Matrix

Req ID.	Requirement Description	Architecture Reference	Design Reference
1	The system shall allow the user to login to the system	User interface, Database	User, Database Model
1.1	The system shall notify the user in the event of a failed login	User interface, Database	User, Database Model
2	The system shall allow the user to create a new account	User interface, Database	User, Database Model
3	The system shall allow the user to search for a team to play a game against	User interface, Game Engine, Database	Main Menu, DatabaseModel
3.1	If no other teams are	Game Engine, Random	Game, Team, Player

	available, the system shall generate a fake team	Number Generator	
4	The system shall allow the user to play a game	Game Engine, Random Number Generator, Game Mechanics, Front End Graphics	Team, Game, Player, SpecialObjects
4.1	The system shall allow the user to choose when to start a turn	Game Engine	Game
4.2	The system shall allow the user to end a game via completion	Game Engine	Game, DatabaseModel
4.3	The system shall allow the user to concede a game before its conclusion	Game Engine	Game, DatabaseModel
5	The system shall allow a user to view their collection of cards	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel
6	The system shall allow the user to create and manage teams	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel
7	The system shall allow the user to view and purchase available packs with acquired in game currency	Game Engine, Front End Graphics, User Interface	DatabaseModel, Store
8	The system shall allow the user to exit the application	User Interface	MainMenu
9	The system shall be able to retrieve user information from the database	Database	DatabaseModel
9.1	The system shall be able to retrieve team information from the	Database	DatabaseModel

	database		
9.2	The system will be able to retrieve the score of a game.	Database	DatabaseModel
9.3	The system will be able to retrieve store information.	Database, Game Mechanics	DatabaseModel, Store
9.4	The system will be able to retrieve game information.	Database	DatabaseModel
10	The system will operate on an Android version above version 4.4	Game Engine	Game, DatabaseModel
11	The system will have an active data connection with the database.	Database	DatabaseModel
12	The system will be able to adjust text size.	Database, User Interface, Front End Graphics	DatabaseModel, Options
13	The system will be able to retrieve forgotten user info for users.	Database, User Interface	DatabaseModel
14	The system will allow users to view the starting manual.	Database, User Interface, Graphic Generator	DatabaseModel
15	The system will determine who will have access to the game documentation.	Game Engine, Database	DatabaseModel
16	The system will have a central documentation repository	Database	DatabaseModel
17	The system will use calculations to play games.	Game Mechanics, Database, Random Number Generator	DatabaseModel, Game
17.1	The system will gather player stats before games.	Database, Game Engine	DatabaseModel, Game

17.2	The system will use calculations to determine the success of certain actions.	Game Mechanics, Database, Random Number Generator	DatabaseModel, Game, Team, Player, Special Objects
17.3	The system will decide who the winner is if the two teams end up with the same score.	Game Mechanics, Database	DatabaseModel, Game, Team, Player, Special Objects
18	The system will display a winner to the user using a graphic after the game.	Game Mechanics, Graphic Generator, Database	DatabaseModel, Game
19	The database shall be built by 5 individuals with some database experience	Database	DatabaseModel
20	The core game application shall be built by 5 individuals using the Unity engine, C#, and Visual Studio	User Interface, Graphic Generator, Game Mechanics, Random Number Generator, Game Engine	All Classes
21	Product Testing shall be performed on Android emulators and Android phones	NA	NA
22	The system shall be accessed only by authenticated users	User Interface	User, Admin
23	The system shall use cryptographic protocols such as SSL and HTTPS for network communications	NA	NA
24	The system shall end the session automatically when an open session is not used for 10 minutes	NA	NA
25	The system shall have an uptime of 99%	NA	NA

26	The system shall present information to the user with a visually pleasing interface	User Interface, Graphics Generator	All Classes
27	The system shall be available to both customers and admins 24/7	NA	NA
28	The system shall be restarted within 5 minutes of system failure	NA	NA
29	The system shall alert an admin via email when the system has a failure	NA	DatabaseModel
30	The system shall protect customer's information using SSP and HTTPS	NA	NA

4. **Conclusion**

The intention of this document is to provide our project team with a development roadmap that ensures each member knows how their classes will interface with others and what each class should do. The 'cowboy coding' philosophy in which developers run straight into the implementation phase, guns blazing, fails with the vast majority of projects with a team size greater than 1, since a lack of a concrete plan leaves implementation of classes up to the whim of each developer. If each developer has their own separate idea of what the project is or how it should be implemented, the system will inevitably fall apart when it comes time to stitch all of the classes together.

After completing this document, our project team has a greater understanding of the composition of classes and the system interface. We discovered that our project would benefit greatly from having a single database model class to interface with the database, as many of the calls being made require similar information. Having this class will cut down on the tedium of rewriting the same code in different locations and will allow us to easily change the format of our database or core application without affecting the other. Additionally, we realised that our project will need heavy integration testing, as the amount of 'moving pieces' in the core app provides ample opportunity for system faults.