Project Documentation

Grand Slam Baseball Allstars 2: Electric Boogaloo

COP 4331, Spring, 2018

Team Name: Team MLB Card Game 2

Team Members:

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

Contents of this Document

Preface

This document is a concatenation of every paper deliverable requested during the development of this project. Artifacts are presented in the order they were produced.

Concept of Operations

Baseball Card Collection Software

COP 4331, Spring, 2018

Team Name: Group 1

Team Members:

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

NOTE: This ConOps was written prior to our project being changed to a baseball card database and game. Safa stated this ConOps did not need to be rewritten for our final documentation.

Contents of this Document

The Current System

Many online sports card collection services exist today on all platforms. Some of these services are offered for free with ad support, while some offer paid subscription models. Most offer similar features for maintaining a collection and want list, such as custom inventory queries and relevant card information. The most popular of these services, sportscardalbum, functions like a social media website, giving collectors a live feed of other collectors' recent acquisitions and profile updates, as well as the ability to follow users, and like and share uploaded cards.

The Proposed System

Motivation

The current system is unsatisfactory for the customer as the breadth of features in most sports card collection software would negatively impact the quality of the user experience. The customer's association is dedicated exclusively to the collection of baseball cards, so features

that aid in the collection of other popular sports cards and social media features would serve only to obfusticate the members' goal. Not only would the inclusion of such features clutter the user interface with information that members of the association simply don't care about, it would also bloat the repository of required card information, which makes the time to fetch any information substantially longer.

Additionally, third party card collection services must support themselves monetarily, which can lead to issues that would further degrade the quality of the user experience. The first being that the system uses a 'paying user' model, in which free users are offered a version of the service with limited features and are frequently prompted to upgrade to the full featured, paid version. The pitfall of this monetization scheme is obvious, as it gives a poor first impression of the system to prospective users. The second issue arises from the 'ad-supported' model, in which all users have full access to the system at the cost of some of the user's screen space being taken up by paid advertisements. This would inevitably annoy some users, which might turn them away from the clients association.

The motivation of our system would be to alleviate the issue of a bloated repository by simply doing away with features that aren't related to baseball card collection. Removing these features would streamline both the card repository and the user interface look-and-feel. Since the customer's association wishes to offer a card collection service to their members as a perk of membership, the proposed system shall not employ one of these invasive monetization schemes.

Users and Modes of Operation

User Class 1: Members - Members of the customer's association must have full access to all features of the proposed system. This includes the ability to edit their collections and want lists, as well as the ability to run queries on their collection, want list, or list of all cards.

User Class 2: Non-members - Non-members of the customer's association must have access to all features of the proposed system, however none of the data related to the non-members should remain in the proposed system's repository, as this would bloat the repository resulting in slower searches for members and act as a deterrent to joining the association. To achieve this goal, the proposed system must allow non members to sign in as a guest, giving them access to a pre-built temporary collection and want list so that they may evaluate the association's system.

Mode of Operation 1: Collection - The proposed system shall provide members and non-members with an inventory of all of their currently owned cards. Users must be able to view their collection, add a card to their collection, remove cards from their collection, and view information related to each card.

Mode of Operation 2: Trade list - The proposed system shall provide members and non-members with a list of cards that they are currently seeking. The trade list will be structured similarly to the collection, with the removal of information that does not pertain to unowned cards (such as date obtained or condition).

Mode of Operation 3: Cards list (all cards) - The proposed system shall provide members and non-members with a list of all cards currently contained in the card repository. Users must have the ability to query cards based on information such as team, price, year/range of years, and card number.

Operational Scenarios

When the user first opens the proposed system they will be directed to a login page where they will have the ability to enter login info, sign up for membership, or retrieve a forgotten password. Additionally, there will be an option to sign in as a guest for non-members. An invalid login will result in a failed login prompt to the user. If an invalid login is detected three times in a row a password reset will be required for that user.

After validating their login, the user will then be directed to their collection page that allows them to explore their card collection. There will be an option to search their card collection for any particular card. If the card is found then it will be shown to the user with all of that card's information, if the card is not found the user will see a message stating that information and as well as an image of the card. There will also be an option for editing the collection, which will allow the user to add or delete cards that are in their collection. Additionally, there will be an option that allows the user to see their wantlist. The wantlist will have the same functionality as the card collection page with the removal of features such as 'date acquired' that don't apply to unowned cards.

To prevent data loss in the event of an internet connection interruption or if a fatal error occurs, the proposed system must save a snapshot of the customer's collection upon an edit taking place. The user will be prompted upon the loss of connection or application error with options for attempting a reconnection or exiting the application. Additionally, the proposed system will open to the last edited page when restoring from a application crash.

Operational Features

Must Have:

1. Collection
2. Want list
3. Robust filter options (by team, player, price, number, year/range of years etc.)
4. Card information (team, condition, price, etc.)

Would Like to Have:

1. Custom search filters (allowing the user to create their own filters) 2. Notifications for when a cards value drastically increases/decreases

Analysis

The language our team will be using to develop the proposed system is Java. The

reasoning for this choice is twofold; first, each member of our team has years of experience with Java, and second, Java's automatic memory management will greatly expedite the process of debugging.

The proposed system shall be developed as a mobile application for android devices, since mobile devices are all but ubiquitous in today's market. Additionally, the development team has a majority of android users. Due to this, a large portion of the team is familiar with how a properly developed project should look on the Android platform. This experience will help keep the team on track as we develop the proposed system. Some members of the development team have little experience with mobile development, so there will be a small learning period to understand its minutia.

The proposed system must have a card repository capable of storing, adding, and searching cards. Both member users and non-member users must be able to search this repository for cards specifically by card number or by using a variety of filters. These filters must include 'cards by year or range of years', 'cards by player', 'cards by team', and the total value of the collection by year or range of years.

Since the proposed system shall be developed on a mobile platform, the card repository will only be accessible through wifi or mobile networks. An alternative to not allowing users to observe their collection without internet connection would be to also have a local collection updated on their android device. The user would not be able update anything for other users to see, but they would be able to observe their own collection. The tradeoff of this method is the storage space required on the android device. The proposed system does not require their collection to be saved to their phone, while the hypothetical local-system would.

The proposed system uses the best possible selections for our platform due to the fact that most users will likely be using an Android mobile phone device. While phones have been evolving over time, one of their biggest weaknesses is still storage space. Our selection is built to take up as little storage space as possible, which is an appealing feature for Android projects.

Project Management Plan

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

Contents of this Document

Project Overview

The objective of our project is to create a Unity based game that utilizes a database of baseball cards to let players manage their own dream team of players and battle against others through asynchronous online play. Players will create their team by opening card packs that contain players and items with unique stats, and then construct a 10-man roster (1 pitcher, 1 catcher, 3 base men, 1 shortstop, 3 outfield, 1 bench) from their collection of cards. Once a team is created, the player will be able to battle teams previously created by other players. The gameplay will focus on team management and item collection, with battles being incredibly

quick innings of baseball based on team stats and random number generation. Since the gameplay is built around team management, stat management, and random number generation, there will be no need for both players to be online to play; one player will simply initiate a battle and then be matched with another team stored on the database, similar to the popular NBA 2K17 mobile game.

Applicable Standards[1]

Our team will be strictly following the Unity C# coding guidelines, as it will allow for ease of code management for anyone familiar with unity, were the game to be sold to another entity at some point. These guidelines are as follows:

- Braces are mandatory for all control flow statements and must begin on the line following the control flow statement and end the line immediately after the block has concluded. The only exception to this is single line statements, where open and closed braces can appear on the same line.
- Indentation for each level of a series of control flow statements is mandatory (including for cases in switch statements). Indentation must be a single tab character.
- Commenting is mandatory for each class and function and should describe what a class is used for or what a function's inputs, purpose, and outputs are. Commenting is encouraged throughout code to explain logical flow. Comments are preferred to be above code rather than to the right of it.
- In function declarations and calls, a single space must be placed after each comma to separate parameters and nowhere else. This means omitting spaces after the function name but before the parentheses and spaces within the parentheses before or after all parameters are declared.
- A single space must be used before parentheses in all control flow statements, as well as a single space before and after comparison operators.
- For all member variables, parameters, and local variables, camelCasing is required. Functions, properties, events, and class names must use PascalCasing. Interface names must be preceded with the capital letter "I".
- Source files should contain only one public type, and the file and type should share a name. Class members should be organized alphabetically and placed in groups based on what they are (a group for methods, a group for constructors, etc.)

Our team's artifact size metric standards are threefold, focusing on number of classes, number of assets, and number of deliverable reports. The breakdown of these artifacts is as follows:

- Number of Classes - The expected number of classes for our project is 13, with 5 classes being produced for the front end, and 8 classes being produced for the backend. For the front end, we expect to make a class for the main menu, the primary in-game interface, the pause menu, the pack store (for purchasing new players and items with in-game currency), and one for the option menu. For the backend, one class will be made to give functionality to each of the front end classes. Additionally, there will be a

back end class for the in-game logic, a class to handle the various kinds of random number generation in game (for handling probability of success on a hit, probability of getting rare cards, etc.), and a class to interface the game with the database containing player information.

- Number of assets - Assets for the game are broken up into 3 categories; card assets, sound assets, and animation assets. The total number of expected card assets, including cards for both players and items is expected to be around 100 at a minimum, going up to around 200 if time allows. At least 1 sound asset and at least 1 animation asset must be present in the final project, but this number may go up to around 5 if time allows.
- Number of deliverable reports - The number of deliverable reports for the project is expected to reach 68, including those listed in the deliverables section, as well as 10 progress reports from each of the 6 team members that will be delivered to both the project manager (Dan) and the CTO (Safa) on a weekly basis.

Project Team Organization

In order to increase productivity, we have divided the group members listed above into two groups. The project manager Dan Carmen, along with John DeSimone and Ronald Marrero will work on the backend logic for the deliverable game, while Malik Henriquez, Brian Wengier, and Kaleb Yangson will focus on front end development.

Individual tasks will be assigned through Trello by Dan Carmen. Daily scrum participation on the group Slack channel is required by group members. In addition, a weekly big picture meeting which will take place in an online voice chat or in a face-to-face meeting on Friday in order to give status updates on big picture tasks, keep track of timetables for the project, and to more effectively allocate human resources as the project continues.

Deliverables

Artifact	**Due Dates**
Individual Scrum Reports	Daily
Concept of Operations	2/9/2018
Project Management Plan	2/18/2018
Software Requirements Specification	2/25/2018
Software design Description	3/25/2018

Test Plan	4/5/2018
Test Results	4/9/2018
Source, Executable, Build Instructions	4/20/2018

Software Life Cycle Process

For this project, the development team has decided to use the Scrum software development methodology, which is a combination of incremental, agile software development framework . This development methodology divides work amongst the respective programming teams into timed milestones called ‘sprints.’ Progress is tracked and milestones are updated in short, daily meetings called scrums. The development team chose this methodology because it was designed specifically for a programming team of our size (typically three to nine members) and allows the team flexibility in allocating resources and accomplishing the overall programming deliverable by in the assigned development time.

Tools and Computing Environment

The programming team is developing this deliverable for release on Android supported devices. Development will take place on a mix of PCs and Mac laptops. In addition, the Unity engine will be used in development along with C# and UnityScript as scripting languages. Product testing will be performed on Android emulators and Android phone systems.

Configuration Management

Version Control will be accomplished through a private Github repository. Only members of this project group will have access to view the project or create pull requests. There will only be two branches to track our development: QA and Master.

The QA branch will not have strict acceptance requirements. If it builds, it will be merged. This branch will be the one that all pull requests go through. This will house any revisions with untested features that will not affect the master branch.

The Master branch will store only stable builds. These builds will only come from the latest stable build in the QA Branch. Once a QA build has been thoroughly tested, it will enter a change control process to determine if it will be merged to master. The entire team will participate in the change control process to decide if the build should be merged. If the decision is made, the QA version will simply be merged up to Master as well.

Ronald Marrero will serve as the GitHub administrator and will be responsible for merging pull requests as well as restoring previous versions in the event of some failure. Ronald

will act in accordance with the group's decisions instead of making autonomous decisions.

Quality Assurance

Quality Assurance will take place at multiple points during development. As code is completed it will be pushed to GitHub. Then, a developer who did not write the code will be chosen from a rotation and review the code that has been submitted for peer review purposes. If problems are found, the code will be bounded back to the original developer along with notes detailing necessary edits to the code.

As the project gets further into development, a table in our documentation will be created to keep track of tests that need to be executed. Successful tests will be marked with the word 'Pass', while failures will be marked with the word "Fail". Failed tests must rerun and successfully complete the test before they can be marked with 'Pass'. This testing suite should be run for each new release of the application.

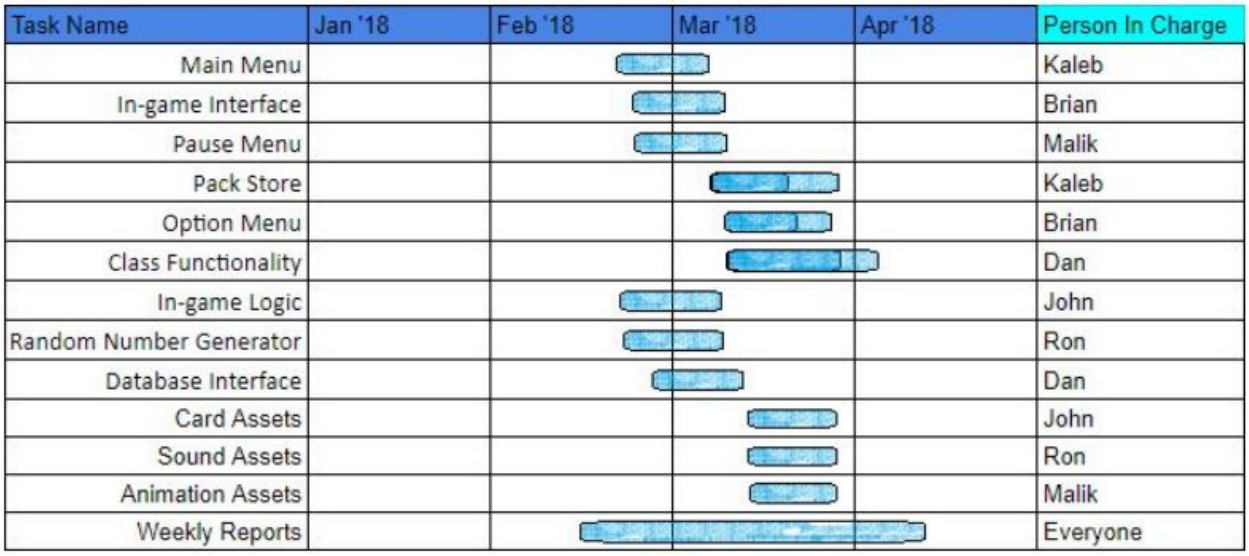
Risk Management

The following are our team's currently expected risks and our plans to mitigate them. The probability of each risk is given as a percentage from 0 to 1 (with 0 being impossible and 1 being guaranteed), and a risk factor ranging from 1 to 5 (1 being minimally impacting, 5 being catastrophic)

- A team member leaves the group - Weather it's due to a team member's non participation or unforeseen circumstances, it is possible that a team member could be removed from the project group. To lessen the impact of this risk, our team shall work diligently to complete deliverables ahead of time and strictly adhere to the project's coding standards so that in the event a group member does leave, there will be less work to do and more information to go off of for the team members that have to pick up where they left off.
    - Probability: 0.1, Risk Factor: 3
- A new team member joins the group - If another team loses too many members, there is a possibility that another person could be reassigned to our project group if their previous project is deemed unviable. To lessen the impact of this risk, thorough commenting shall be peppered throughout all code artifacts, and all paper artifacts shall adhere to the same document formatting.
    - Probability: 0.05, Risk Factor: 4
- Library Deprecation - In the event that a library becomes deprecated, code refactoring will be added to the list of assignments needed to be completed. The workload caused by this refactoring shall then be distributed evenly across the group. Aside from that, library depreciation is an accepted risk for the project, as our team has no way of predicting which libraries will go out of use.
    - Probability: 0.2, Risk Factor: 2
- Compatibility issues with team code - Since this project is being developed by multiple

people, it is extremely likely that there will be compatibility issues when it comes time to packaging each team member's artifacts together. In order to lessen the impact of this risk, all team members shall strictly adhere to the coding standards described above. Additionally, each artifact shall undergo peer review whenever a new version is uploaded to the git repository.
    - Probability: 1, Risk Factor: 1
- Github repository crash - Since we have no control over the stability of the Github repository being used to house all code deliverables for the project, there is a small chance that this repository could crash or become faulty. In order to lessen the impact of this catastrophic risk in the event it occurs, each group member will clone a copy of the master branch to their local machine whenever it is updated.
    - Probability: 0.05, Risk Factor: 5

Table of Work Packages, Time Estimates, and Assignments



The GANTT chart shows the 13 different artifacts that the group will be working on. Task Name is the name of the tasks assigned, the dates represent the month and year, and Person In Charge represents what group member is responsible for making sure that the task gets completed. The blue bars within the graph represent the estimated amount of time that each task will take.

Technical Progress Metrics

Requirements Phase

- Number of meetings done with group - This metric will benefit the team because throughout the project it will be necessary for the team to meet, face to face or conference call, to ensure everyone is on the same page.
- Number of requirements created/completed - This metric is important because the team

will need to create requirements throughout the project.
- Number of requirements changed - This metric is important because as we progress in the project it will be necessary to look back and modify some of the team's previous requirements.
- Individual Progress Reports - This metric is important because it allows the project manager to keep track of the work of each project member.

Design Phase

- Number of Classes designed - This metric will be important because during the design phase the team will be constantly creating new classes.
- Number of uml diagrams - This metric will be important because if design is going smoothly our team will be consistently making new diagrams for the system.
- Number of revisions completed - This metric is important because throughout the design phases there will be a need to make a revision in order to improve the system.
- Implementation Phase:
- Lines of Code Per Day - This metric will be beneficial because it will ensure that project team members are actively working everyday on implementing the project.
- Number of classes completed - This metric will be important because it will ensure that the project manager has a general idea of the team's progress and makes sure no classes are missed during the implementation phase.

Testing Phase

- Number of test cases - This metric is important because if testing phase is conducted correctly the project team will be testing multiple variables multiple times to make sure that the final product is reliable and of good quality.
- Passes/fails for tests - This metric is important because it will allow the team to keep track of the results of each test case throughout the testing phase of the project.
- Bugs found/fixed - This metric is important because during if testing is done correctly bugs will be found occasionally and the team will need to fix the problem.

Packaging Phase

- Number of builds - This metric is important because during the packaging phase the team will need to conduct multiple builds to ensure a reliable final product.

Maintenance Phase

- Number of updates - This metric is important because as Operating Systems are updated it is important to update the program to ensure it remains a quality product.

Plan for tracking, control, and reporting of progress

Each day during development, all team members will congregate for a 10 minute scrum meeting to discuss current progress and pitfalls. After each week, the progress reported from these scrums shall be analysed to determine areas of the project that are struggling or that need more focus. Additionally, technical progress metrics and test results will be used to determine

the project’s progress and direction.

Software Requirements Specification

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

Contents of this Document

- Resource Requirements
- Security Requirements
- Quality Assurance Requirements (AKA nonfunctional, Quality Requirements)

Supporting Material

Introduction

Software to be Produced

There are two main software products that are to be produced as part of this project. More information on each deliverable is available in the rest of this document.

- Core game application - The primary deliverable is the game application itself. This application is a mobile game that will contain all of the necessary assets to run the game, except for the user and card data, which will be stored on the database.
- Database - The secondary deliverable is the database that will store user and card data for all of the users and cards that exist in the game. This data will be stored on the server to prevent players from altering their user data in malicious ways, such as editing their card collection or game statistics (such as their win/loss ratio). This data will be fetched from the server when the user logs in and will be updated to the server after each card pack acquisition, after each change to the player's teams, and after each game concludes.

Applicable Standards

- All requirements shall be listed in the same format. Each requirement will be given a unique number, description,

Definitions, Acronyms, and Abbreviations

- User - any player with an account stored on the user database.

- Active user - the user that is currently playing the game.

- User data - the data that is associated with each user. This data consists of the user's current teams, the user's win/loss ratio, the win/loss ratio for each of their currently built teams, the user's card collection, and the user's current amount of in-game currency.

- Database - the database that houses user data for each user and card.

Product Overview

Assumptions

- Users are assumed to have an android device with at least a 1 GHz processor and 1 GB of RAM.
- Users are assumed to have an internet connection capable of the minimal transfers required to communicate with the database.
- Users are assumed to have basic ‘smartphone literacy’, aka the ability to operate standard smartphone applications.
- The hosting service being used for the user database is assumed to provide us with at least 99% uptime.

Stakeholders

- Users - This class of stakeholder consists of the players that shall use the system to play the described game. The user’s main interest is in the core functionality of the game, as well as its responsiveness and visual appeal.
- Admins - This class of stakeholder consists of the administrators that will oversee application and database maintenance. Their primary concerns is ease of maintainability, as well as the software’s immunity to errors.
- Customer - This stakeholder is the customer that proposed the development of the application. Their main concerns are the functionality, robustness, immunity to errors, and visual appeal of the final system.

Event Table

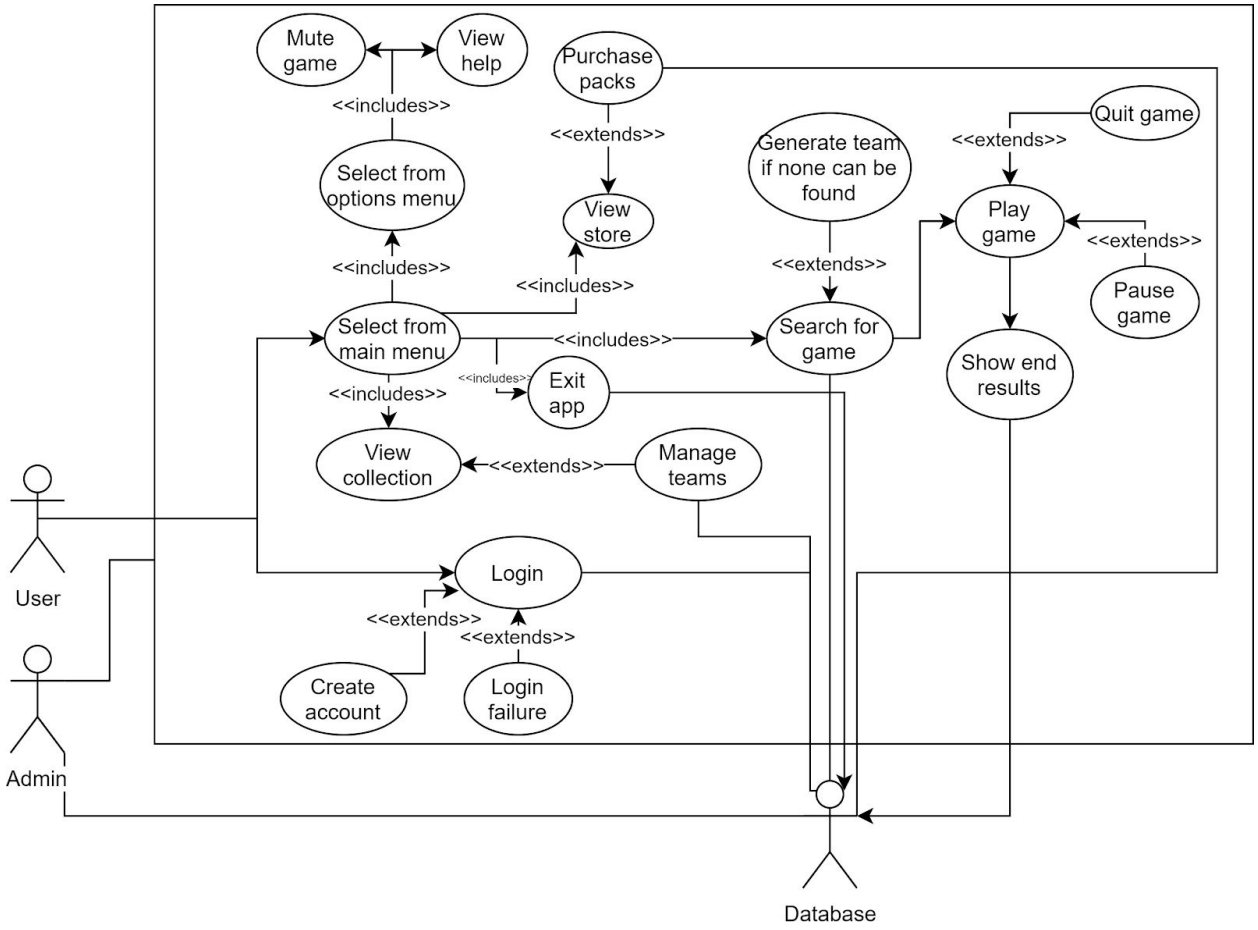
Event Name	**External Stimuli**	**External Responses**	**Internal data and state**
Startup	User launches the application	App shows Unity logo and starts up, then prompts the user to login	No internal data yet, system state goes to login page
Login	User enters valid login information, user data sent from server	Prompts the user with a welcome message, then loads the main menu	User login info sent to database, user and card data fetched from database, state change to main menu

Failed Login	User enters invalid login information, Null reference sent from server	Prompts the user with a message saying “Invalid email or password.”	Invalid login info sent to database, no user data fetched, no state change
Main Menu	User makes a selection	User is sent to submenu depending on selection (session search if Play game is selected, scripted tutorial game if tutorial is selected, store if card shop is selected, options menu if options is selected)	User data, current state is main menu, state change depending on user selection
Game Search	N/A	User is sent to a multiplayer lobby where the user will be paired to another player seeking a game (if available)	User data, opposing user data, state change to active game
Card Collection Menu	User input (scrolling, card selection)	User is shown their current card collection and can sort based on drop-down criteria	User data, state change to selected card (if applicable)
Options Menu	User makes a selection (Sound settings, network settings, help screen)	User is shown a list of options where they can change volume level, brightness, and graphic quality	User is shown options to change based on role (administrator or user)
Shop Menu	User makes a selection (which kind of card pack to purchase)	User is shown a variety of card packs and a price associated depending on the pack type	Card pack data, state change to selected pack
Start Turn	Active user presses start turn button	“TURN START!” message displayed on screen	Active user data, opposing user data, game results initialized, state change to main step
Main Step	N/A	Simulated inning will play out on screen with card graphics and animations,	Active user data, opposing user data, state change to

		turn results screen will be displayed after completion, along with “Next turn” button	game results based on turn results
Game End	N/A	After 9 innings if score is tied, simulated audience will choose which team is the victor (based on things like showboating and home field advantage). Otherwise the team with the highest score is shown the “ALL-STAR WINNER” screen, and the other player is show the “ALL-STAR LOSER” screen	Active user data, opposing user data, state change to game results based on current game results, active user data and opposing user data updated with completed game results
Pause Menu	User presses the pause button	User is shown options menu, with additional option for “Quit game”	Active user data, opposing user data, state change depending on user input
Team Creation Menu	User selection	User is shown controls and options for creating a team based on available cards	Active user data, state change depending on user input
Concede	User selects “Quit Game” from pause menu	User is shown a confirmation page to verify requires to leave	Active user data, opposing user data, Game state changes depending on user selection, active user data and opposing user data updated to database
After Match Report	User selection	User is shown match stats and options for “Main Menu” and “New Game”(game search)	Active user data, game report, state change depending on user selection, active user data and opposing user data updated to database

Exit Application	User closes app through native phone controls	Database connections are closed, default view is set to main menu for next application use	User data updated to database, application closes

Use Case Diagram



Use Case Descriptions

- Select from main menu - This use case describes the user's ability to select an option from the main menu. These options include "collection", "options", "store", "find game", and "exit".

- View collection - This use case describes the user's ability to view their current card collection. Within their collection, they will also have the option to manage their teams.

- Manage teams - This use case describes the user's ability to create and manage new teams using their currently acquired cards.

- Exit app - This use case describes the user's ability to safely exit the application from the main menu.

- Select from options menu - This use case describes the user's ability to select one of several options available in the games options menu, which is accessible from the main menu and from the game's pause screen.

- Pause game - This use case describes the active user's ability to pause the game at any point during play. Pausing the game will display the options menu.

- Mute game - This use case describes the user's ability to mute all game audio.

- View help - This use case describes the user's ability to view the provided in-game help documentation, which is written to provide new users with a basic understanding of the game's mechanics.

- View store - This use case describes the user's ability to view the card pack shop, which will show the user a list of the types of packs currently available for purchase with their in-game currency.

- Purchase packs - This use case extends the functionality of view store, allowing users to select an available card pack and purchase it with their in-game currency.

- Create account - This use case describes the system's ability to allow a new user to create an account.

- Login - This use case describes the user's ability to enter their username and password in order to log in to the game.

- Login failure - This use case extends the functionality of the login use case, allowing the system to handle situations where the user provides invalid login credentials.

- Search for game - This use case describes the user's ability to search the user database for an opposing team.

- Generate team if none can be found - This use case extends the functionality of search for game, allowing the system to handle situations where no opposing team can be found.

- Play game - This use case describes the user's ability to play through a full simulated game of baseball with their team. (need to add quit game use case)

- Show end results - This use case describes the system's ability to provide the user with an end of game statistics screen, as well as update the user database with these statistics.

- NOTE: The role of the Admin actor is to oversee maintenance of the system and the database.

Specific Requirements

Functional Requirements

No: 1
Statement: The system shall allow the user to open the application and login with their username and password
Source: Events Table
Dependency: Requirement 1.1, 2
Conflicts: None
Supporting Materials: Events Table, UML Use Case diagram (above)
Evaluation Method: A successful login will return user data and allow the user to enter the game's main menu
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 1.1
Statement: The system shall prompt the user to login again if a login attempt fails to fetch user data
Source: Events Table

Dependency: None
Conflicts: None
Supporting Materials: UML Use Case Diagram 1 (above)
Evaluation Method: If a login attempt fails, the user will be prompted with a message informing them of the failure and asking them to try again
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 2
Statement: The system shall allow a user to make a new account on the login page
Source: Requirement 1
Dependency: None
Conflicts: Requirement 1 (if the chosen account information is already taken)
Supporting Materials: UML Use Case diagram
Evaluation Method: If the account is successfully created, its information will be available on the user database
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 3
Statement: The system shall allow the user to search for a new game
Source: Events Table, UML Use Case Diagram (above)
Dependency: Requirements 5

Conflicts: Requirement 3.1
Supporting Materials: None
Evaluation Method: This requirement shall be met if the game search function successfully places the active user into a game
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 3.1
Statement: If a game cannot be found, the system shall generate a team for the active user to play against
Source: Requirement 3
Dependency: None
Conflicts: None
Supporting Materials: None
Evaluation Method: A game will be searched for when the database has no user data besides that of the active user to see if a team is properly generated
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 4
Statement: The system shall use the active user’s team stats and the other user’s team stats to simulate a game of baseball
Source: Events Table
Dependency: Requirements 3, 3.1, 4.1, 4.2

Conflicts: None
Supporting Materials: UML Use Case Diagram
Evaluation Method: This requirement is met if the completed system can play through an entire game from the starting turn until the game end screen is shown
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 4.1
Statement: The system shall allow the active user to control the flow of the game by letting them start each turn (inning)
Source: Events table
Dependency: None
Conflicts: None
Supporting Materials: UML Use Case Diagram
Evaluation Method: This requirement is met if the active user is able to make the turns go by at their leisure through use of the next turn button
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 4.2
Statement: The system shall end the game and tally up the game statistics after the 9th turn (inning)
Source: Events Table
Dependency: None

Conflicts: None
Supporting Materials: Data requirements
Evaluation Method: This requirement will be satisfied if the end game statistics screen matches up with manual, by hand calculations done for a single game
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 4.3
Statement: The system shall allow the active user to concede a game they are currently playing, resulting in a loss for them and a perfect win for the other use (a win as if the other user had no opposition)
Source: Events Table
Dependency: None
Conflicts: None
Supporting Materials: Data requirements, UML Use Case diagram
Evaluation Method: This requirement will be met if the active user's data is updated to result a total defeat in the user database and the other user's data is updated to reflect a perfect win in the user database
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 5
Statement: The system shall allow the user to view their current card collection
Source: Events Table, Project Management Plan
Dependency: None

Conflicts: None
Supporting Materials: UML Use Case diagram
Evaluation Method: This requirement shall be met if the displayed collection matches up perfectly with the user's card collection data in the user database
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 6
Statement: The system shall allow users to create and manage their teams with their currently collected cards
Source: <source of the requirement>
Dependency: Requirement 5
Conflicts: None
Supporting Materials: UML Use Case diagram, UML Class diagram
Evaluation Method: This requirement shall be met if users can create, edit, and delete their teams and have these changes show up in the user database
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 7
Statement: The system shall allow users to view packs available for purchase and to purchase said card packs using their acquired in game currency
Source: Events Table
Dependency: None

Conflicts: None
Supporting Materials: UML Use Case diagram, UML Class diagram
Evaluation Method: This requirement shall be met if users are able to purchase a pack and have the newly acquired cards show up in the user database
Revision History: Daniel Carman, 2/24/2018, ver. 01

No: 8
Statement: The system shall allow the user to exit the application via an option on the main menu
Source: Events Table
Dependency: None
Conflicts: None
Supporting Materials: UML Use Case diagram
Evaluation Method: This requirement shall be met if
Revision History: Daniel Carman, 2/24/2018, ver. 01

Interface Requirements

No: 9
Statement: The system shall retrieve user information with synchronous calls.
Source: Events table, UML Use Case Diagram
Dependency: Requirement 1

Conflicts: None
Supporting Materials: Data requirements
Evaluation Method: The system will output correct user information.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 9.1
Statement: The system shall retrieve team information with synchronous calls.
Source: Events table, UML Use Case Diagram
Dependency: Requirement 4
Conflicts: None
Supporting Materials: Data requirements
Evaluation Method: The system will output correct and current team stats within a range of [0%, 100%].
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 9.2
Statement: The system shall retrieve player scores with asynchronous calls every 30 seconds.
Source: Events table, UML Use Case Diagram
Dependency: Requirement 4
Conflicts: None
Supporting Materials: Data requirements

Evaluation Method: The system will reflect successful updates to all player scores within a positive range of integers.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 9.3
Statement: The system shall retrieve store information with synchronous calls.
Source: Events table, UML Use Case Diagram
Dependency: Requirement 7
Conflicts: None
Supporting Materials: Data requirements
Evaluation Method: The system will output all current store items and will accept successful input from the user on purchases. The price will also be within a positive range of floats where accuracy is crucial up to the hundredths place.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 9.4
Statement: The system shall retrieve active game information with asynchronous calls at 5 second intervals in an active game until game completion.
Source: Events table, UML Use Case Diagram
Dependency: Requirement 4
Conflicts: None
Supporting Materials: Data requirements

Evaluation Method: A success message from the database call will indicate that this requirement is satisfied.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

Physical Environment Requirements

No: 10
Statement: The system shall only run on android devices running Android 4.4 and above.
Source: Startup
Dependency: None
Conflicts: None
Supporting Materials: Events Table
Evaluation Method: A successful installation of the application will show that the application succeeded on supported hardware.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 11
Statement: The system shall only run on devices with an active data connection.
Source: Startup
Dependency: Requirement 10
Conflicts: None
Supporting Materials: Events Table

Evaluation Method: The application will switch between menus successfully if there is an active data connection.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

User and Human Factors Requirements

No: 12
Statement: Any user shall be able to adjust the text size in their option menu
Source: Options Menu
Dependency: None
Conflicts: None
Supporting Materials: Project Management Plan
Evaluation Method: During the testing phase, we will have a user try and adjust the text size in the option menu.
Revision History: Malik Henriquez, 2/24/2018, ver.01

No: 13
Statement: Any user shall be able to retrieve a forgotten username and/or password
Source: Failed Login event
Dependency: None
Conflicts: None
Supporting Materials:

Evaluation Method: During the testing phase, we will have a user try and retrieve their username and password
Revision History: Malik Henriquez, 2/24/2018, ver.01

Documentation Requirements

No: 14
Statement: The users shall be able to view a manual containing the game features and rules
Source: Events Table
Dependency: None
Conflicts: None
Supporting Materials: Project Management Plan
Evaluation Method: An inexperienced user will evaluate the manual for aiding in learning gameplay
Revision History: Malik Henriquez, 2/24/2018, ver. 01

No: 15
Statement: Future software developers shall be able to view information in regards to managing and updating the system.
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials: PMP

Evaluation Method: Software Developers will review document and evaluate on understanding and detail. It is assumed that they know the basics of software development.
Revision History: Malik Henriquez, 2/24/2018, ver. 01

No: 16
Statement: All documentation shall be available to future developers via a central documentation repository
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials: PMP
Evaluation Method: The existence of this repository is sufficient for this requirement evaluation
Revision History: Malik Henriquez, 2/24/2018, ver. 01

Data Requirements

No: 17
Statement: The user shall be able to start a new multiplayer game where the winner will be determined by calculations.
Source: Game Search Event
Dependency: Requirement 4
Conflicts: None
Supporting Materials: UML Use Case Diagram

Evaluation Method: A successful pairing will occur and the active user will be entered into a game with another user.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 17.1
Statement: The system shall gather a list of each player's stats for each team.
Source: Start Turn event
Dependency: Requirement 17
Conflicts: None
Supporting Materials: Events Table, UML Use Case diagram
Evaluation Method: A success message from the database call will indicate that this requirement is satisfied.
Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 17.2
Statement: The system shall make data calculations to determine successful actions.
Source: Start Turn event
Dependency: Requirement 17.1
Conflicts: None
Supporting Materials: UML Use Case Diagram
Evaluation Method: A successful change of game state by both players will indicate success.

Revision History: Ronald Marrero, 2/24/2018, ver. 01

No: 17.3
Statement: In the event of a tie, the system shall determine the winner based on a calculation.
Source: Events Table
Dependency: Requirement 17
Conflicts: Requirement 18
Supporting Materials: UML Use Case Diagram
Evaluation Method: A valid winner in the case of a tied game after 9 turns (innings) will indicate success.
Revision History: Kaleb Yangson, 02/24/2018, ver. 01

No: 18
Statement: The system shall declare a winner and update both players' win/loss ratio in the database.
Source: Events Table
Dependency: Requirement 17
Conflicts: Requirement 17.3
Supporting Materials: Events Table, UML Use Case Diagram
Evaluation Method: A success message from the database call will indicate that this requirement is satisfied.
Revision History: Kaleb Yangson, 02/24/2018, ver. 01

Resource Requirements

No: 19
Statement: The user database shall be built by 6 individuals with some database experience
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials: PMP
Evaluation Method: The functioning user database will be sufficient for the evaluation of this requirement
Revision History: Brian Wengier, 02/24/2018, ver. 01

No: 20
Statement: The core game application shall be built by 6 individuals using the Unity engine, C#, and Visual Studios
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials: PMP
Evaluation Method: The functioning game will be sufficient for the evaluation of this requirement
Revision History: Brian Wengier, 02/24/2018, ver. 01

No: 21
Statement: Product testing shall be performed on Android emulators and Android phones
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials:PMP
Evaluation Method: During the testing phase, logs will be kept to prove testing was done on Android emulators and phones
Revision History: Brian Wengier, 02/24/2018, ver. 01

Security Requirements

No: 22
Statement: The system shall be accessed only by authenticated users.
Source: Events Table
Dependency: Access to the rest of the system beyond the login page
Conflicts: None
Supporting Materials: UML Use Case diagram
Evaluation Method: The system satisfies this requirement when a user cannot access the system until they enter their username and password
Revision History: Brian Wengier, 02/24/2018, ver. 01

No: 23
Statement: The system shall use cryptographic protocols such as SSL and HTTPS for network communications
Source: PMP
Dependency: None
Conflicts: None
Supporting Materials: UML Use Case Diagram
Evaluation Method: The use of these protocols is sufficient for the evaluation of this requirement
Revision History: Brian Wengier, 02/24/2018, ver. 01

No: 24
Statement: The system shall end the session automatically when an open session is not used for 10 minutes
Source: Login event
Dependency: None
Conflicts: None
Supporting Materials: UML Use Case Diagram
Evaluation Method: The system satisfies this requirement when the user is logged out after being inactive for 10 minutes
Revision History: Brian Wengier, 02/24/2018, ver. 01

Quality Assurance Requirements (AKA nonfunctional, Quality Requirements)

No: 25
Statement: The system shall have an uptime of 99%
Source: User’s needs
Dependency: None
Conflicts: 28
Supporting Materials: PMP
Evaluation Method: Logs of uptime/downtime history
Revision History: Kaleb Yangson, 02/24/2018

No: 26
Statement: The system shall present information to the user with a visually pleasing interface
Source: User’s needs
Dependency: None
Conflicts: None
Supporting Materials: PMP
Evaluation Method: Game testers, customer feedback
Revision History: Kaleb Yangson, 02/24/2018

No: 27

Statement: The system shall be available to both customers and admins 24/7
Source: Requirement 25
Dependency: 25
Conflicts: 28
Supporting Materials: PMP
Evaluation Method: Logs of uptime/downtime history
Revision History: Kaleb Yangson, 02/24/2018

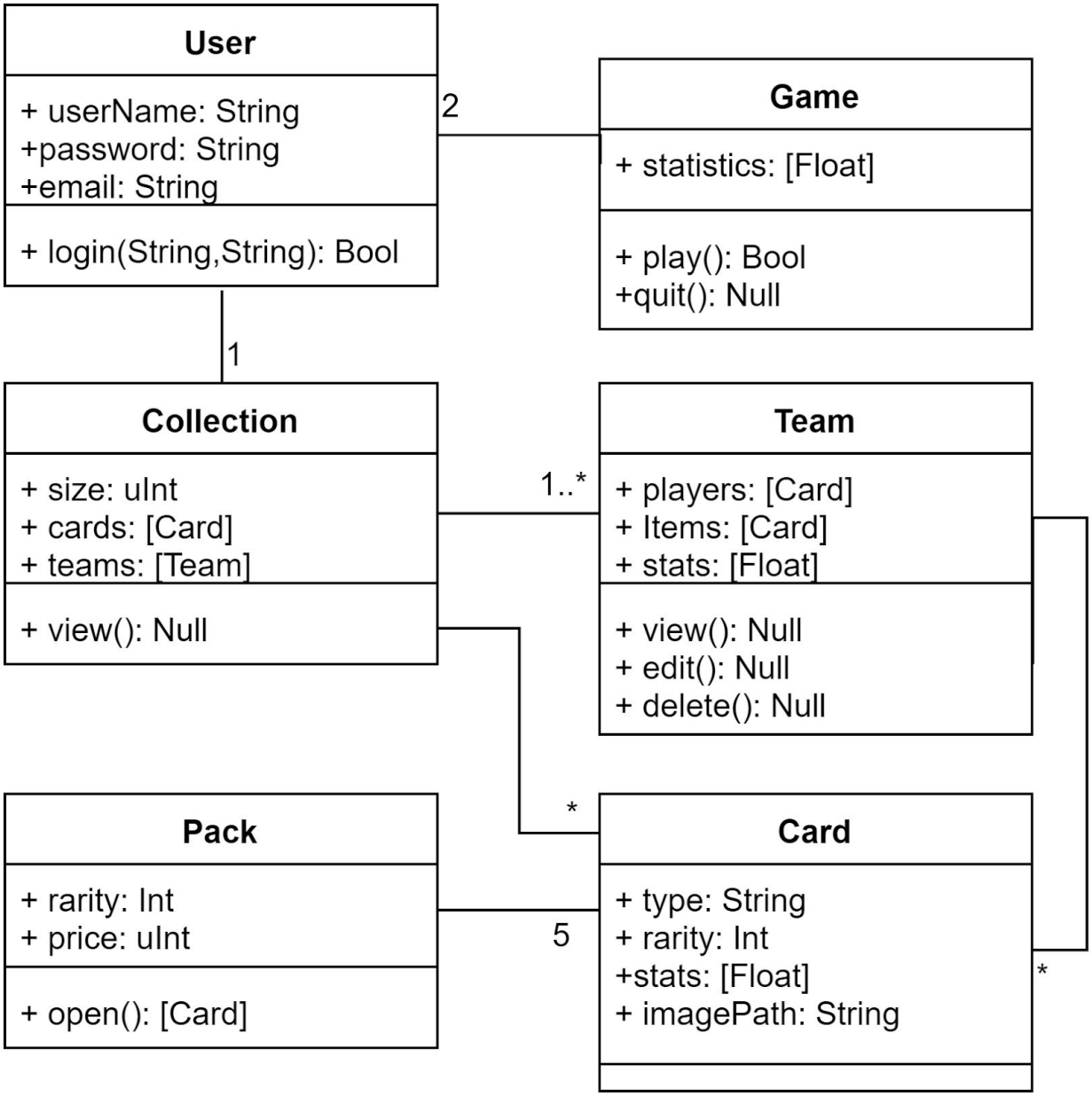
No: 28
Statement: The system shall be restarted within 5 minutes of system failure
Source: Requirement 25
Dependency: None
Conflicts: Requirements 25, 27
Supporting Materials: PMP
Evaluation Method: Logs of uptime/downtime history
Revision History: Kaleb Yangson, 02/24/2018

No: 29
Statement: The system shall alert an admin via email when the system has a failure.
Source: Requirement 28

Dependency: Requirement 28
Conflicts: None
Supporting Materials: PMP
Evaluation Method: Testing the system by simulating a failure and checking for an email.
Revision History: Kaleb Yangson, 02/24/2018

No: 30
Statement: The system shall protect customer’s information using SSP and HTTPS
Source: Requirement 23
Dependency: Requirement 23
Conflicts: None
Supporting Materials: PMP, UML Use Case Diagram
Evaluation Method: The use of these protocols is sufficient for the evaluation of this requirement
Revision History: Kaleb Yangson, 02/24/2018

Section 4: Supporting Material



Software Design Description

Grand Slam Baseball All-Stars 2: Electric Boogaloo

COP 4331, Spring, 2018

Team Name: Team MLB Card Game 2

Team Members:

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

Contents of this Document

Introduction

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.

Project Scope

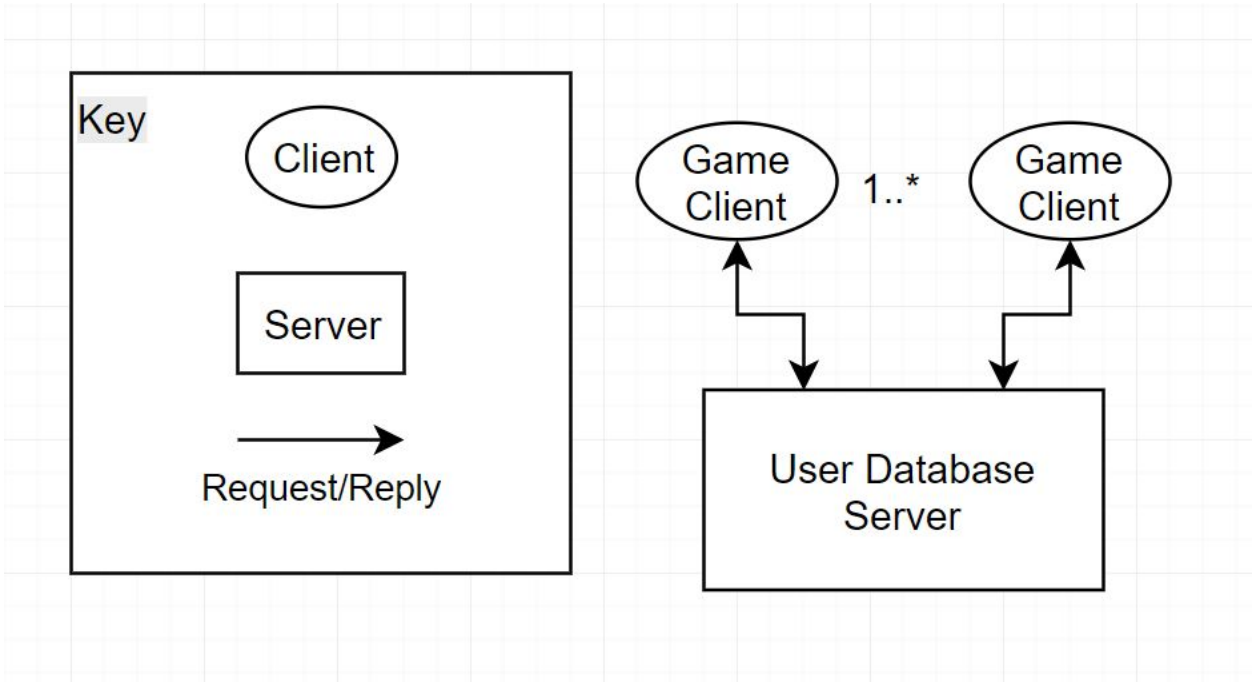
Our system is composed of 3 major components; the game frontend, the game backend, and the data repository, each with their own required features. The front end of the game must allow users to navigate through menus, display game assets (such as cards or scenery), and interact with the system using their 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.

Architectural Design

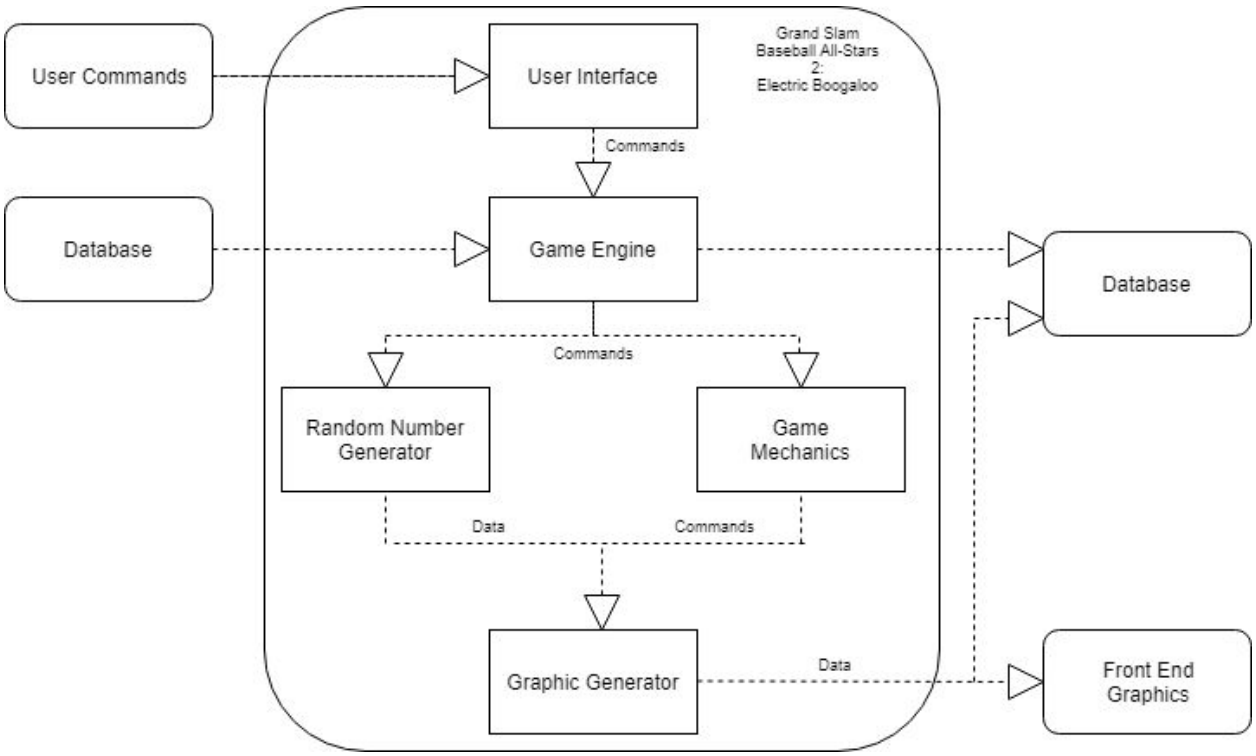
High-level Architecture



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.

System-interface Architecture

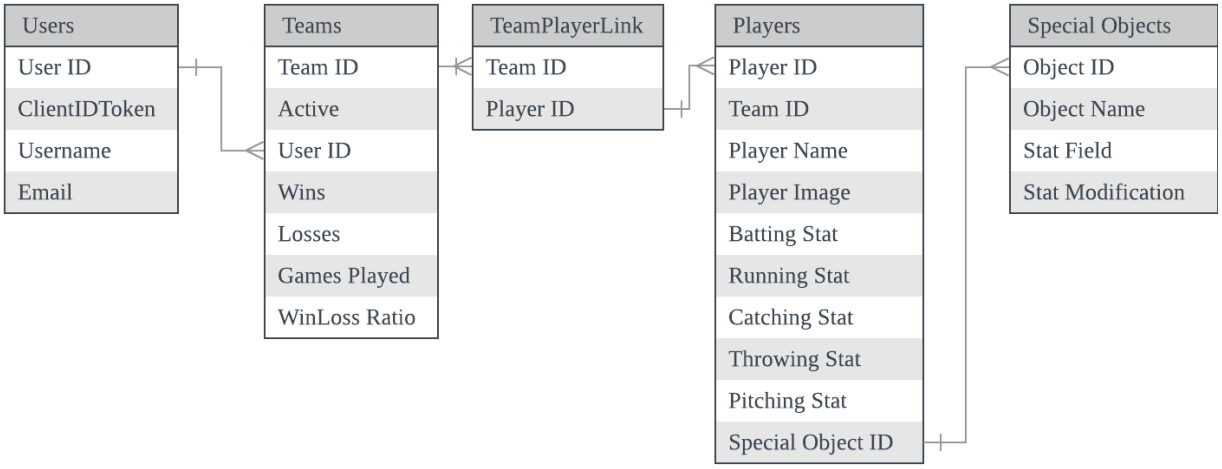


The User Interface represents the graphical interface that the User can see and interact with. With a User interacts with the User Interface, the UI sends commands to the Game Engine which exists to interpret what to do with User commands. The Game Engine, at the same time, is taking in data from the 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.

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.

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.

Detailed Design

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.

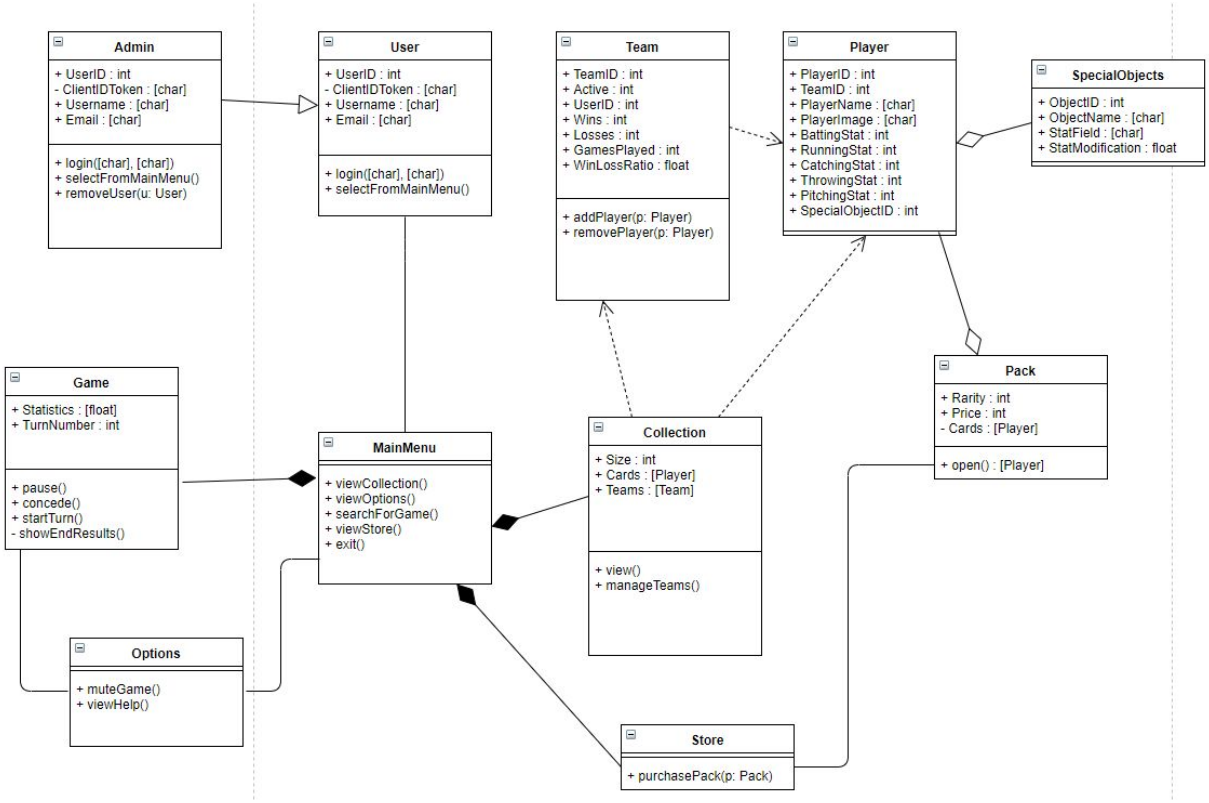
Since MSSQL allows for foreign keys, logically separated tables were created to store the information for users, teams and players. Queries performed in our first prototype were consistent and easy to interpret. Additionally, since the Android application interacts to the database through a web service,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

Component *n* Detailed Design



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

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.

Test Plan

Grand Slam Baseball Allstars 2: Electric Boogaloo

COP 4331, Spring, 2018

Team Name: Team MLB Card Game 2

Team Members:

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

Contents of this Document

Test Plan

Overall Objective for Software Test Activity

The goal of this software test plan is to provide our team with a framework for ensuring the reliability of our system and for assessing its adherence to our requirements. The tests outlined in this plan will assist us in finding faults with our system so that they may be corrected before the system's delivery. Additionally, the tests are designed in a way such that they can be mapped backwards to each requirement in order to demonstrate a requirement implementation.

Description of Test Environment

The majority of unit and system testing will be done by the developers of the system on android emulators run on windows machines. The emulators will be running Android version 6.0, API 23, as it offers relatively good compatibility with modern devices and our team is familiar with it. Each team member will create a different Android Virtual Device (AVD),

corresponding to a different popular phone running Android 6.0. This will allow us to gain some insights as to how the system might run on a range of hardware systems.

Running the unit testing in emulators rather than on physical devices will allow us to perform some automated testing that would be incredibly tedious to do by hand. One the system has a working build, system testing will first be performed on emulators and then will be performed on physical devices, to ensure the system remains functional in its final installation environment.

Stopping Criteria

Testing will be conducted in blocks, with each block being focused on one major module of the system and lasting 1-2 hours in duration. During each block, a single test case explicitly concerning that module will be repeated to ensure each component in that module is functioning as expected. These test cases are designed to require the proper functioning of all components in their module, meaning that they will not properly conclude if even 1 component in the module is not functioning correctly. If any faults are discovered during testing, they will be noted with details regarding the fault and, if possible, reproduction steps for the fault.

If a fatal issue is encountered that prevents the proper functioning of the system past a certain point, the block will conclude prematurely and the team will work on correcting this fatal fault. Once it is corrected, the team will attempt to reproduce the fault, and then resume with the current testing block if the fault cannot be reproduced and the proper documentation has been created.

After a block has concluded, everyone on the team will be assigned one or several faults to investigate. Their goal is to find our what is causing the faults to occur and how to prevent them from occurring, either by means of data guarding, error handling, or logic debugging. Once the fault is corrected, that team member will document what changes were made both in code and in documentation to assist future system maintenance specialists. After all faults have been corrected, the block will be run once again to ensure the proper functioning of the system. The system will be deemed good enough to deliver after it successfully passes all blocks one after another.

Description of Individual System Test Cases

Test Case No.	**Test Objective**	**Test Description**	**Test Conditions**	**Expected Results**
Case 1	This test shall demonstrate the proper functioning of the login	An automated test will be used to test this module. Each correct pair of user and password strings will be input into the login	See test environment	The system will allow login for each account on the first pass, and not allow

	module.	screen, and an attempt will be made to log in to the system. The process will be completed with each correct user name with an incorrect password to ensure the system is properly authenticating login.		login on the 2nd pass.
Case 2	This test shall demonstrate the ability to create a new user.	Manual testing shall be used to test this functionality. First, the tester shall attempt to make a new user account with a previously unused username. After this, the tester shall attempt to create a new user with that same username in order to test how the system handles an attempt at a duplicate name. To conclude the test, the tester shall attempt to recover their username and password using their email account.	See test environment	A new user account will successfully be created on the first try, but the attempt will be denied the 2nd try. The tester shall receive email instructions containing their username and a link to reset their password.
Case 3	This test shall demonstrate the ability to play a full, simulated game of baseball.	Manual testing shall be used to test this functionality. First, the tester shall attempt to find a game when there is no other user information on the server to test if the system can generate a fake team. This test will be passed if the tester can get through the game with only minor cosmetic issues. Next, the tester will repeat the test when the database is populated with other user data to test if it can retrieve another user's team.	Pass 1: The test shall be conducted on an empty database. Pass 2: The test shall be conducted on a populated database.	The results of the game will be reflected in the user database for both games.

Case 4	This test shall test the system's ability to let a user view their collection and edit their teams.	This test shall be conducted manually. In this test, the tester shall go into the collection screen, view their collection, and attempt to create a new team from their collection. Then, they shall attempt to edit their team after creation. If the tester can do all of this without faults, this test shall be passed.	The collection of the tester will be populated with enough cards to form a team.	The new team and the changes made to it will be visible in the database.
Case 5	This test shall test the system’s ability to let the user purchase packs with in game currency.	In this test, the tester shall first attempt to purchase a pack with the requisite currency on their account. After this, they will attempt to purchase a pack without the requisite currency.	The tester account will have exactly as much currency as it takes to get a pack	The first pack should be purchased successfully, with the cards in it showing up in the database. The second pack should be unsuccessful.
Case 6	This test shall test if the system allows the user to smoothly exit the application.	In this test, the tester will attempt to exit the application via the exit option on the main menu. If the system updates any changes to the database and then closes the application without fault, this test will be passed.	See test environment	Any changes made to teams should be updated to the database before close.
Case 7	This test shall examine the functionality of the options menu.	This test case shall be performed manually. In this test, the tester shall examine each option within the options menu and determine if it is functioning properly.	See test environment	Each option should either have a clear effect on the settings of the app or a clear visual output in the case of the help screen.
Case 8	This case shall test quality	This test case will be done manually.This test is broken up into 7 subtests, which	This test must be performed extensively on	This test will be evaluated on a per-subtest

	requirements.	each test particular quality requirements (requirement IDs in parentheses): 1. Must run on a version of android above 4.4 (10) Output: Pass 2. The database will be built by our team with MySQL, the app will be created with unity and C# (19, 20) Output: Pass 3. The app will be tested on android emulators and phones (21) Output: Pass 4. Connections and user data will be secured with SSL and HTTPS (23, 30) Output: Pass 5. Must display a game's winner via a graphic (18) Output: Pass, Fail 6. The system shall have an uptime of 99%, shall be available to customers 24/7, shall email admins on a fatal fault, and shall be restarted within 5 minutes of failure. (25, 27, 28, 29) Output: Pass, Fail 7. The system must be	actual phones; a real hardware environment has a higher chance of faulty behavior when compared to an emulated environment.	basis. For example, the system can pass for subtest 1, but fail for the rest of them, or vice versa. For time based requirements, the system should show high robustness, staying active without fatal faults for a long period of time. Qualitative judgements given should be as objective as possible.

		visually pleasing (26) Output: Pass, Fail Subtests 1-4 (and also their requirements) are self sufficient and thus will never impede this test's success. Test 5 is a binary decision; pass if there is a winner's graphic, fail if there isn't. Test 6 will be evaluated as passed if the system is capable of running without failure for the entire testing block. Test 7 will either pass or fail based solely on the tester's decision.		
Case 9	This test case shall test all components of the system that connect to the database for functionality.	This test will be performed manually.This test is broken up into 6 subtests, which each test particular database related requirements (requirement IDs in parentheses): 1. Ability to login (1, 1.1, 22) Output: Pass, Fail 2. Ability to make new account (2) Output: Pass, Fail 3. Ability to search for team on the database (3) Output: Pass, Fail 4. Ability to update game stats to the database (4.2, 4.3) Output: Pass, Fail 5. Ability to retrieve user data (9, 9.1,	This test must be performed extensively on actual phones; mobile data connections can be much more unreliable when compared to LAN or WiFi.	This test will be evaluated on a per-subtest basis. For example, the system can pass for subtest 1, but fail for the rest of them, or vice versa.

		9.2, 9.3, 9.4) Output: Pass, Fail 6. Ability to maintain an active connection until user is inactive for 10 min. (11, 24) Output: Pass, Fail Any non fatal faults found during testing should be noted and placed in the pool of need-to-fix bugs.		

Trace of Individual Test Cases to the Requirements Traceability Matrix

Req ID.	Requirement Description	Architecture Reference	Design Reference	Test Case Reference
1	The system shall allow the user to login to the system	User interface, Database	User, Database Model	Cases 1, 9
1.1	The system shall notify the user in the event of a failed login	User interface, Database	User, Database Model	Cases 1, 9
2	The system shall allow the user to create a new account	User interface, Database	User, Database Model	Cases 2, 9
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	Cases 3. 9
3.1	If no other teams are available, the system shall generate a fake team	Game Engine, Random Number Generator	Game, Team, Player	Case 3
4	The system shall allow the user to play a game	Game Engine, Random Number Generator, Game Mechanics, Front	Team, Game, Player, SpecialObjects	Case 3

		End Graphics		
4.1	The system shall allow the user to choose when to start a turn	Game Engine	Game	Case 3
4.2	The system shall allow the user to end a game via completion	Game Engine	Game, DatabaseModel	Case 3
4.3	The system shall allow the user to concede a game before its conclusion	Game Engine	Game, DatabaseModel	Cases 3, 9
5	The system shall allow a user to view their collection of cards	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel	Cases 4, 9
6	The system shall allow the user to create and manage teams	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel	Cases 4, 9
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	Cases 5, 9
8	The system shall allow the user to exit the application	User Interface	MainMenu	Case 6
9	The system shall be able to retrieve user information from the database	Database	DatabaseModel	Cases 3, 4, 9
9.1	The system shall be able to retrieve team information from the database	Database	DatabaseModel	Cases 3, 9
9.2	The system will be able to retrieve the score of a game.	Database	DatabaseModel	Cases 3, 9
9.3	The system will be able	Database, Game	DatabaseModel,	Cases 5, 9

	to retrieve store information.	Mechanics	Store	
9.4	The system will be able to retrieve game information.	Database	DatabaseModel	Cases 3, 9
10	The system will operate on an Android version above version 4.4	Game Engine	Game, DatabaseModel	All Cases
11	The system will have an active data connection with the database.	Database	DatabaseModel	Cases 1, 2, 3, 4, 5, 9
12	The system will be able to adjust text size.	Database, User Interface, Front End Graphics	DatabaseModel, Options	Case 7
13	The system will be able to retrieve forgotten user info for users.	Database, User Interface	DatabaseModel	Cases 2, 9
14	The system will allow users to view the starting manual.	Database, User Interface, Graphic Generator	DatabaseModel	Case 7
15	The system will determine who will have access to the game documentation.	Game Engine, Database	DatabaseModel	Case 9
16	The system will have a central documentation repository	Database	DatabaseModel	Cases 8, 9
17	The system will use calculations to play games.	Game Mechanics, Database, Random Number Generator	DatabaseModel, Game	Case 3
17.1	The system will gather player stats before games.	Database, Game Engine	DatabaseModel, Game	Cases 3, 9
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	Case 3

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	Case 3
18	The system will display a winner to the user using a graphic after the game.	Game Mechanics, Graphic Generator, Database	DatabaseModel, Game	Case 3
19	The database shall be built by 5 individuals with some database experience	Database	DatabaseModel	Cases 8, 9
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	All Cases
21	Product Testing shall be performed on Android emulators and Android phones	NA	NA	All Cases
22	The system shall be accessed only by authenticated users	User Interface	User, Admin	Cases 1, 9
23	The system shall use cryptographic protocols such as SSL and HTTPS for network communications	NA	NA	Cases 8, 9
24	The system shall end the session automatically when an open session is not used for 10 minutes	NA	NA	Case 6
25	The system shall have an uptime of 99%	NA	NA	Case 8

26	The system shall present information to the user with a visually pleasing interface	User Interface, Graphics Generator	All Classes	Case 8
27	The system shall be available to both customers and admins 24/7	NA	NA	Case 8
28	The system shall be restarted within 5 minutes of system failure	NA	NA	Case 8
29	The system shall alert an admin via email when the system has a failure	NA	DatabaseModel	Case 8
30	The system shall protect customer's information using SSP and HTTPS	NA	NA	Case 8, 9

Test Results

Description of the Actual Test Environment

The actual testing environment didn't differ much from what was expected, however we did ignore one tool in our initial description that was going to be extremely helpful during testing; the Unity Editor. Besides use of the editor to handle dependencies between objects and translations in the game world, the actual testing was still performed on android emulators during each partial build.

Results of Testing

Unit Tests

Throughout implementation of our system, our team has been performing unit testing on the components that they have been developing. Below is a short synopsis of some of those tests.

Daniel Carman - Core Game

My primary development task has been creating the core game application. Currently, the framework behind the game's functionality, consisting of teams, players, items, and the ball are implemented and functional, all that needs to be added is the logic that utilizes the stats

along with some RNG to produce the math that drives matches.

Currently the system meets game related requirements 3.1, 4, 4.1, 17, 17.1, and 17.2. This means that the system is capable of generating a fake team to play against, starting up a new game with the active player team and an opposing team, and letting the user start their turn, thus causing the game to progress according to the currently implemented game logic. These requirements were tested by running code in the Unity editor throughout development, documenting which requirements were satisfied, and occasionally checking my work by running the full system on an emulator.

Currently, requirements, 3, 4.2, 4.3, 17.3, and 18 have not been implemented. This means that the system currently doesn't have any end conditions implemented for the game, which also implies that it has no winner's graphic, and the team search function is not currently implemented.

Moving forward, my top priority is to finish implementing the game logic and end conditions so that I may work with Ron to get the team search function properly working.

Ronald Marrero - Database Connection

My primary development task has been establishing the data connections from the app to the database. Connections to the database are only made with users after they have logged in. After that, each connection is made over HTTPS to the Firebase database. Testing ensured full functionality of the previously mentioned components as well as reading/writing to the realtime database. Further development is required to test game logic against the database as only standard CRUD (Create Read Update Delete) operations have been tested.

Kaleb Yangson - Options Menu/Common Assets

My primary development task has been building a visually pleasing interface that allows both the customization of the game (Options Menu) and the currency shop that allows the purchase of crates using in game currency (Common Assets). Testing ensured functionality of menu buttons and navigation. Further development is required for the actual in game currency system as well as universal variables that will be able to be edited from the options menu.

Brian Wengier - Card Collection/Main Menu

My primary development task has been building an interface that allows players to both navigate from the main menu to various other interfaces (such as Options Menu, Play Menu, and Quit) and to view the player's card collection. Testing ensured the functionality of menu buttons and navigation. It also ensured the ability to pull up a player's cards and view their stats. Further development is required on the storing and calling of cards with the server and a more visually pleasing interface for the main menu.

Malik Henriquez - Pause Menu / Animation Assets

My primary development task has been building a pause menu that would allow a user to interrupt a game in progress as well as adding the animation assets needed while the game is being played. For the pause menu, testing ensured that any background game processes would be put to sleep while the user decided which option to select on the pause menu. Testing of the animation assets in unity ensured that there were no glitches in how the animations were occurring and also ensured that the animations were occurring at the proper times.

This testing has not yet been done in conjunction with the database connection but has been confined to local debugging. The next and final step for my piece of integration testing is to ensure that there are no faults from pausing a game in progress and that once the pause menu is closed, the connection re-opens and the game resumes.

System Tests

Since our system did not manage to implement all of its features, the system tests obviously showed some failures. To reflect this, requirements are considered passed if they pass their unit tests and if they pass their portion of their system tests. The features that we're not able to be implemented were the in-game store, the team search, game concession, view a starting manual in app, and sending an email warning on system failure. The reason we were not able to implement the in-game store, game concession, starting manual and team search was due to the unforeseen event of losing our primary unity knowledge holder and the time required to pick up the slack. The reason for omitting the email warning is that our team realized that there is a paradox that arises from telling a failed system to perform a task properly; if the system has already failed it would not be able to email an administrator.

Requirements Traceability Matrix (With Status)

Req ID.	Requirement Description	Architecture Reference	Design Reference	Test Case Reference	Status
1	The system shall allow the user to login to the system	User interface, Database	User, Database Model	Cases 1, 9	Pass
1.1	The system shall notify the user in the event of a failed login	User interface, Database	User, Database Model	Cases 1, 9	Pass
2	The system shall allow the user to create a	User interface, Database	User, Database Model	Cases 2, 9	Pass

	new account				
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	Cases 3. 9	Fail
3.1	If no other teams are available, the system shall generate a fake team	Game Engine, Random Number Generator	Game, Team, Player	Case 3	Pass
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	Case 3	Pass
4.1	The system shall allow the user to choose when to start a turn	Game Engine	Game	Case 3	Pass
4.2	The system shall allow the user to end a game via completion	Game Engine	Game, DatabaseModel	Case 3	Pass
4.3	The system shall allow the user to concede a game before its conclusion	Game Engine	Game, DatabaseModel	Cases 3, 9	Fail
5	The system shall allow a user to view their collection	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel	Cases 4, 9	Pass

	of cards				
6	The system shall allow the user to create and manage teams	Front End Graphics, User Interface	User, Team, Player, Collection, DatabaseModel	Cases 4, 9	Pass
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	Cases 5, 9	Fail
8	The system shall allow the user to exit the application	User Interface	MainMenu	Case 6	Pass
9	The system shall be able to retrieve user information from the database	Database	DatabaseModel	Cases 3, 4, 9	Pass
9.1	The system shall be able to retrieve team information from the database	Database	DatabaseModel	Cases 3, 9	Pass
9.2	The system will be able to retrieve the score of a game.	Database	DatabaseModel	Cases 3, 9	Pass
9.3	The system will be able to retrieve store information.	Database, Game Mechanics	DatabaseModel, Store	Cases 5, 9	Fail
9.4	The system will be able to	Database	DatabaseModel	Cases 3, 9	Pass

	retrieve game information.				
10	The system will operate on an Android version above version 4.4	Game Engine	Game, DatabaseModel	All Cases	Pass (self sufficient)
11	The system will have an active data connection with the database.	Database	DatabaseModel	Cases 1, 2, 3, 4, 5, 9	Pass
12	The system will be able to adjust text size.	Database, User Interface, Front End Graphics	DatabaseModel, Options	Case 7	Pass
13	The system will be able to retrieve forgotten user info for users.	Database, User Interface	DatabaseModel	Cases 2, 9	Pass
14	The system will allow users to view the starting manual.	Database, User Interface, Graphic Generator	DatabaseModel	Case 7	Fail
15	The system will determine who will have access to the game documentation.	Game Engine, Database	DatabaseModel	Case 9	Pass (self sufficient)
16	The system will have a central documentation repository	Database	DatabaseModel	Cases 8, 9	Pass (self sufficient)
17	The system will use calculations to play games.	Game Mechanics, Database, Random Number	DatabaseModel, Game	Case 3	Pass

		Generator			
17.1	The system will gather player stats before games.	Database, Game Engine	DatabaseModel, Game	Cases 3, 9	Pass
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	Case 3	Pass
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	Case 3	Pass
18	The system will display a winner to the user using a graphic after the game.	Game Mechanics, Graphic Generator, Database	DatabaseModel, Game	Case 3	Pass
19	The database shall be built by 5 individuals with some database experience	Database	DatabaseModel	Cases 8, 9	Pass (self sufficient)
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	All Cases	Pass (self sufficient)
21	Product Testing shall be performed	NA	NA	All Cases	Pass (self sufficient)

	on Android emulators and Android phones				
22	The system shall be accessed only by authenticated users	User Interface	User, Admin	Cases 1, 9	Pass
23	The system shall use cryptographic protocols such as SSL and HTTPS for network communications	NA	NA	Cases 8, 9	Pass (self sufficient)
24	The system shall end the session automatically when an open session is not used for 10 minutes	NA	NA	Case 6	Pass (self sufficient)
25	The system shall have an uptime of 99%	NA	NA	Case 8	Pass
26	The system shall present information to the user with a visually pleasing interface	User Interface, Graphics Generator	All Classes	Case 8	Pass
27	The system shall be available to both customers and admins	NA	NA	Case 8	Pass (as long as database remains up)

	24/7				
28	The system shall be restarted within 5 minutes of system failure	NA	NA	Case 8	Pass
29	The system shall alert an admin via email when the system has a failure	NA	DatabaseModel	Case 8	Fail
30	The system shall protect customer's information using SSP and HTTPS	NA	NA	Case 8, 9	Pass (self sufficient)

Conclusion

Our conclusion from testing is that the system is in a mostly functional state, however there are some minor cosmetic features missing, as well as the major store feature. These features could surely be implemented with more time, as most of the difficulties with implementation were due to time constraints caused by the loss of a team member.