# System and Software Architecture Description (SSAD)

**SpyFall – The Android Mobile Game**

**John Sun (Product Manager),**

**Kaya Ota (UX Designer),**

**Frank Li (Tech Lead),**

**Eric Vuu (Tester)**

**5/15/2017**

# Version History

| Date | Author | Version | hanges made | Rationale |
| --- | --- | --- | --- | --- |
| 05/15/2017 | JS | 2 | * Some proposed features were cut and the documentation reflects that * Changed from SQL to NoSQL * Numerous updated artifacts | * Document updated to reflect finished application; which is much different from what we imagined it would be. |
| 03/01/2017 | JS | 1 | * Original document | * Initial draft for semester project |

# Table of Contents

System and Software Architecture Description (SSAD) i

Version History ii

Table of Contents iii

Table of Tables iv

Table of Figures v

1. Introduction 1

1.1 Purpose of the SSAD 1

1.2 Status of the SSAD 1

2. System Analysis 2

2.1 System Analysis Overview 2

2.2 System Analysis Rationale 5

3. Technology-Specific System Design 9

4.1 Design Overview 9

4.2 Design Rationale 10

4. Architectural Styles, Patterns and Frameworks 11

# Table of Tables

[Table 1: Actors Summary 8](#_Toc463312298)

[Table 2: Artifacts and Information Summary 9](#_Toc463312299)

[Table 3: Process Description 9](#_Toc463312300)

[Table 4: Typical Course of Action 10](#_Toc463312301)

[Table 5: Alternate Course of Action 10](#_Toc463312302)

[Table 6: Exceptional Course of Action 10](#_Toc463312303)

[Table 7: Hardware Component Description 13](#_Toc463312304)

[Table 8: Software Component Description 13](#_Toc463312305)

[Table 9: Design Class Description 15](#_Toc463312307)

[Table 10: Architectural Styles, Patterns, and Frameworks 18](#_Toc463312312)

# Table of Figures

[Figure 1: System Context Diagram 7](#_Toc463312318)

[Figure 2: Artifacts and Information Diagram 8](#_Toc463312319)

[Figure 3: Process Diagram 9](#_Toc463312320)

[Figure 4: Conceptual Domain Model 12](#_Toc463312321)

[Figure 5: Hardware Component Class Diagram 12](#_Toc463312322)

[Figure 6: Deployment Diagram 13](#_Toc463312324)

[Figure 7: Design Class Diagram 14](#_Toc463312326)

[Figure 8: Robustness Diagram 16](#_Toc463312327)

[Figure 9: Sequence Diagram 16](#_Toc463312328)

### Introduction

SpyFall – a mobile game designed for Android – is a game of intrigue, and deception, where players must figure out who is trust worthy and who isn’t. At the start of every game, players are randomly assigned location and role. Everyone who isn’t a spy, has a job at this random location. The spy does not know what the location is and it is her job to figure out the location, based on the questions the other players are asking each other. Players ask each other questions to determine who is the spy. If someone arouses enough suspicion – to be voted as a spy by most players – then that player must guess the location. If she is wrong, then she loses the game and the other players win. Therefore, it is in the spy’s best interest to stall the game. (This is another way the spy can win, stalling the game until the timer runs out without being caught.)

#### Purpose of the SSAD

The objectives of this document are to: Help our team formulate a concrete implementation plan for our game, and to describe how the game is designed; so that it can be reproduced and built upon by other developers.

#### Status of the SSAD

This is the updated version of our SSAD and there have been a few architectural changes. Originally, we planned to use JDBC drivers to connect our app to a SQL backend. We found that to be impractical so we moved our system to a Firebase NoSQL backend. Furthermore, due to time constraints we had to remove several player features from the app. Core features are untouched and the game still works, but there are no rich multiplayer features.

### System Analysis

#### System Analysis Overview

The purpose of this mobile game is to bring Spy Fall – a beloved card and web application based game – to Android. This way, players do not have to use their browsers to play the game, and can access it right on their cell phones. By making a mobile optimized app, our game will be able to better represent the experience of playing the physical card game compared to any web app.

##### System Context

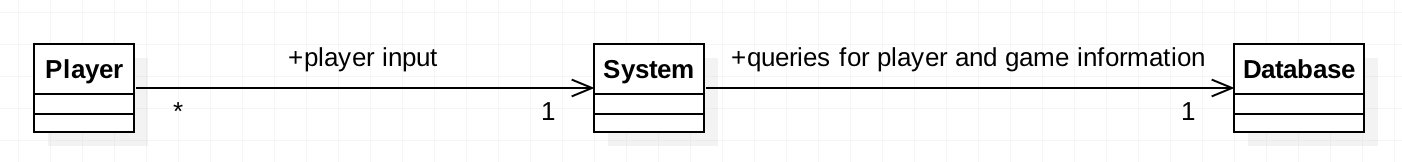


Figure 1: System Context Diagram

Player

The player is the actor that interacts directly with the system, and causes the system to query the database. Player responsibilities are very limited, as most of the work is done by the system/db. When the user logs into the game, that will cause a query against the db. Later, when the player creates, joins or plays a game of *SpyFall*, those will cause more queries against the db.

System

The system is comprised of the Android application, along with all the activities, resource files, libraries, and so on that make it up. Our system will help hide all the complexity of the game’s backend and the database so that the user experience is accessible. It will achieve this by automating a lot of the work of creating, joining, and starting a game.

Database

Our application’s backbone is the NoSQL Firebase database, which hosts all our player and game data. The database acts as a “middle man” for data between players and the app.

Table 1: Actors Summary

| **Actor** | **Description** | **Responsibilities** |
| --- | --- | --- |
| Player | The person playing the game | Play the game |
| System | The Android application | Hide complexity from user and query database for appropriate information |
| Database | Google Firebase | System with asynchronous access to player and game data |

##### Artifacts & Information

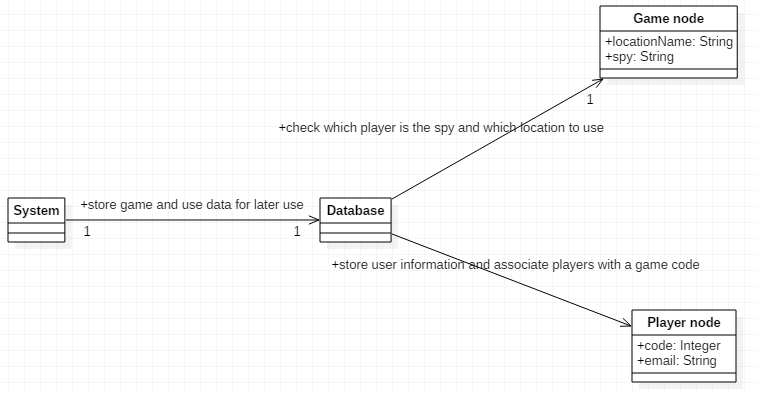


Figure 2: Artifacts and Information Diagram

Table 2: Artifacts and Information Summary

|  |  |
| --- | --- |
| **Artifact** | **Purpose** |
| Player node | Player nodes keep track of user information during the game. Information stored in these nodes is pulled from the FirebaseUser object. Nodes are named after user ids to make it easier to know which node to remove, if a person leaves a game lobby. There are up to 4 player nodes per game. |
| Game node | Game node keeps track of 2 vital pieces of information: the location where the game is taking place, and the identity of the spy. When a client joins the game, it will be his responsibility to pull that information from Firebase. |

##### Behavior

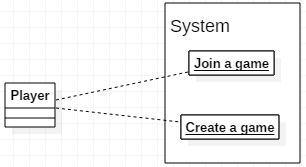


Figure 3: Player-System Interactions (After User Logs In)

###### Player-System Interactions (After User Logs In)

Create/Join a Game

Table 3: Process Description

|  |  |
| --- | --- |
| **Identifier** | Create a Game/Join a Game |
| **Purpose** | Allows player to create or join a customizable game lobby. |
| **Requirements** | Activity front end (within the app), and a database back end. |
| **Development Risks** | Using one database for backend is not a scalable solution. |
| **Pre-conditions** | The user must already have logged into their account. |
| **Post-conditions** | The database must be up and running. |

Table 4: Typical Course of Action – User Joins a Game

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player selects Join a Game. | System displays activity with Text (Enter a game code), EditText and Button (Enter) views. |
| **2** | Player enters in the code associated with the game lobby they wish to join, then clicks Join Game. | System searches the Lobby table for games that have a matching code. If found, the user is automatically ushered into the desired game lobby. |

Table 5: Alternate Course of Action – User Enters Incorrect Code

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player selects Join a Game | System displays activity with Text (Enter a game code), EditText and Button (Enter) views. |
| **2** | Player enters in the code associated with the game lobby they wish to join, then clicks Enter. | After searching the Lobby table for the appropriate game, the system determines that the user entered an incorrect game code. An error message is displayed to the user, informing them that they have entered an incorrect game code; and they should try again. |

Table 6: Exceptional Course of Action – Database is Down

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Player selects Join a Game | System displays activity with Text (Enter a game code), EditText and Button (Enter) views. |
| **2** | Player enters in the code associated with the game lobby they wish to join, then clicks Enter. | The database is down, and the application shows a user-friendly error message that the server is down. |

### Technology-Specific System Design

#### Design Overview

##### System Structure

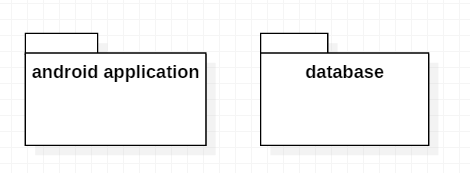


Figure 4: Conceptual Domain Model

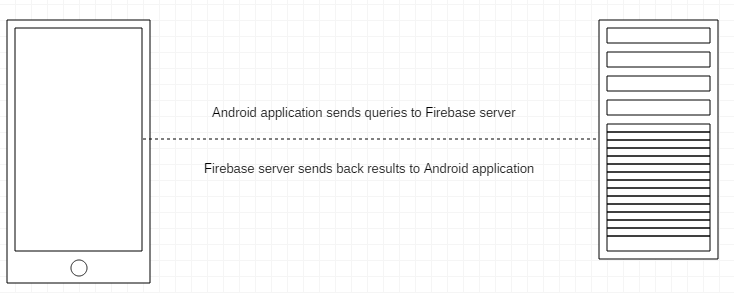


Figure 5: Hardware Component Class Diagram

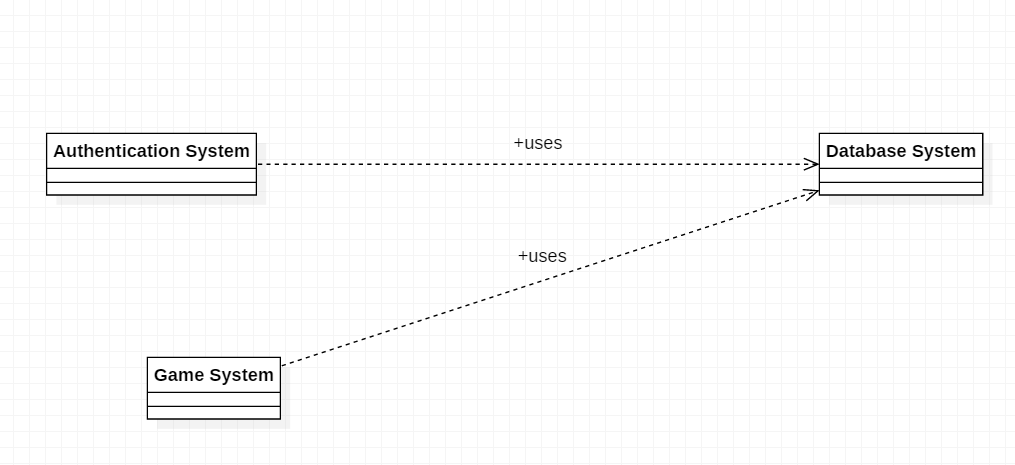


Figure 6: Software Component Class Diagram

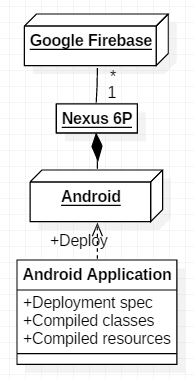


Figure 6: Deployment Diagram

Table 7: Hardware Component Description

|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Android Device | Any Android phone that runs Android Nougat |
| Database Server | Google Firebase |

Table 8: Software Component Description

|  |  |
| --- | --- |
| **Software Component** | **Description** |
| SpyFall Mobile Application | The game that we are developing |
| Google Firebase | Google’s NoSQL service |

##### Design Classes

###### Overview of Classes

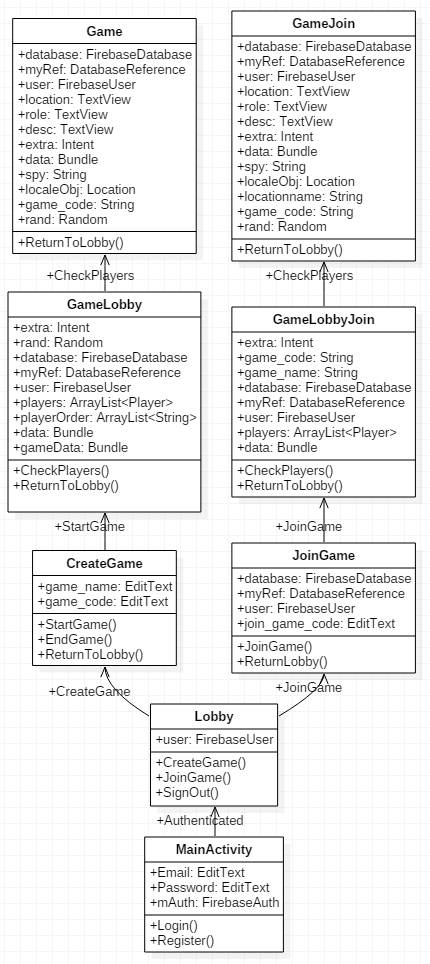


Figure 7: Design Class Diagram

Table 9: Design Class Description

|  |  |  |
| --- | --- | --- |
| **Class** | **Type** | **Description** |
| MainActivity | Class | Authenticates user and allows her to associate game play data with her profile. Creates a new user account if user does not have an account. |
| Lobby | Class | Displays a menu that asks the user what she would like to do: Join or create a game. |
| CreateGame | Class | Simply asks the user for a game name, and code then passes that information to Game |
| Game | Class | Picks a random location and a random spy, then stores that information in Firebase. This class is the main game activity for player 1. |
| GameJoin | Class | The main game activity for players 2,3, and 4. Handles retrieving game information from Firebase using listeners. |
| GameLobby | Class | Displays the emails of all users who are waiting in the current game lobby. This particular activity is meant for player 1. |
| GameLobbyJoin | Class | Like GameLobby, but meant for players 2, 3 and 4. The reason we have multiple activities for the same thing, is because the host player does different things from the client player. The host sets up the game location and spy identity, whereas the client does not do that. |
| JoinGame | Class | Simple activity that asks for a game code. |
| Location | Class | Custom data type that holds game data. |
| Player | Class | Custom data type that holds player data. |

##### Process Realization

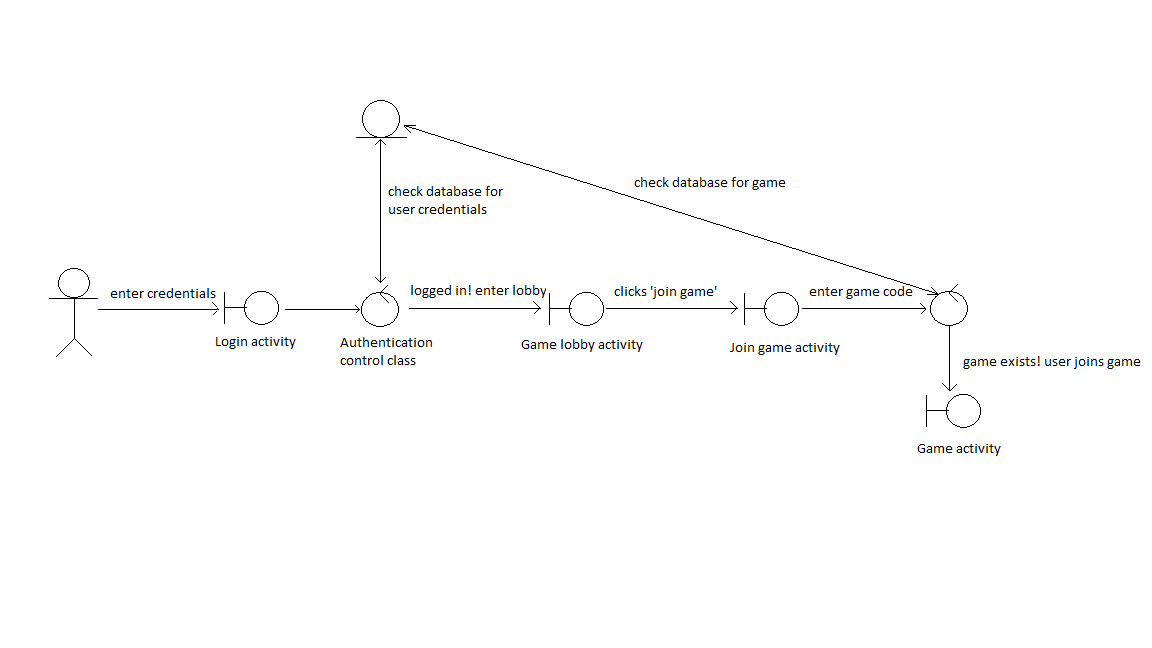


Figure 8: Robustness Diagram

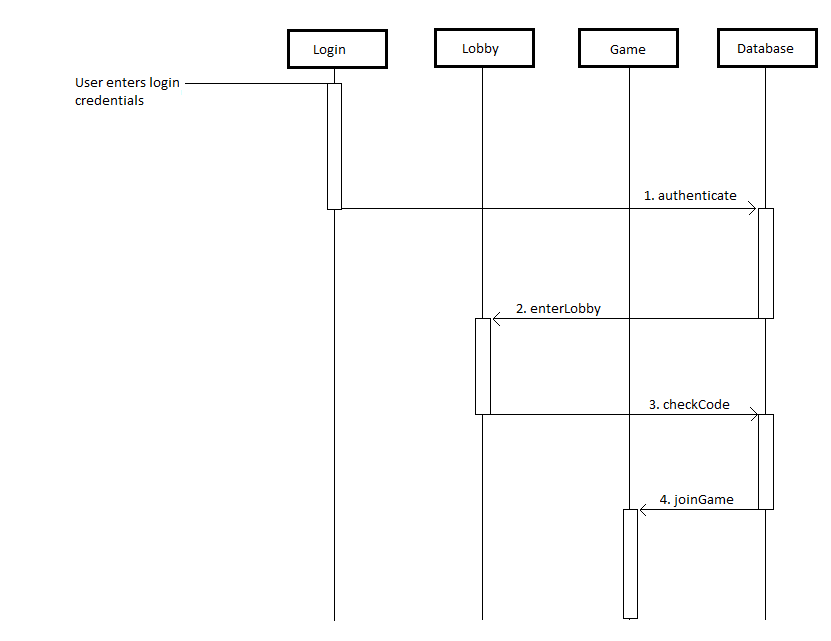


Figure 9: Sequence Diagram

#### Design Rationale

Originally, we chose the design of an Android device communicating with a MySQL database server because all our team members are familiar with SQL. We then realized that JDBC does not work on mobile platforms, so we switched to Google Firebase. This proved to be a challenge because no one on the team had worked with NoSQL before. We eventually worked out how to make the core features of our app work, but we had to scope down and remove several features. Features such as the store or player points had to be removed so we could finish the project.

Table 10: Architectural Styles, Patterns, and Frameworks

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Benefits, Costs, and Limitations** |
| Java | Oracle’s very own language | Everyone on the team knows Java, and it is the language for developing Android applications with Firebase. |