Game Frame

Networked multiplayer games made easy.

Table of Contents

Meet the Team

Shaun Toomey (toomey1) - Android Client Lead

Hussain Alzeera (alzeera1) - Networking Lead

Nathan Hibner (hibner1) - Server Lead

Chris Jeong (jeong10) - Testing Lead

Paul Zhang (zhang248) - Game Developer

Alan Xia (alanxia1) - Game Developer

Project Overview

The goal of Game Frame is to empower developers to build high-quality, real-time multiplayer games across a variety of platforms and devices. Using Game Frame, we want to make it downright simple for developers to implement common features such as user registration, game lobbies, real-time multiplayer, cross-platform play (i.e. Android and web, currently), and "second-screen" capabilities (i.e. display part of the game through a Chromecast connected to a TV). Developers should be able to focus on the gameplay, graphics, and logic in their games: we'll handle the rest.

Project Links

- Online Documentation
- gameframe-server repo
- gameframe-android repo
- poker-android repo
- poker-js repo

Development Process

While developing this project, we followed a modified version of scrum.

Conforming to the 2 week iteration meetings required by the class, our aim was to implement a couple of user stories (complete with tests) every iteration. User stories were written with the various areas of our project in mind (server, Android client SDK, Poker sample game) and were taken care of by those assigned to each area (rather than a bidding process/planning game).

Refactoring was performed whenever we decided to change the functionality, mechanism, or

layout of our code. Occasionally, if a spike solution was produced to quickly remove the development block on another team, the spike code would later be refactored to conform with the coding standards specified by our group at the beginning of the semester (i.e. the Android Code Style Guidelines for Contributors for our Java code, found here, and the Node Style Guide for our Javascript code, seen here).

In contrast to the “test driven development” of XP, we began with code that fulfilled our required functionality and wrote unit tests afterward as a monitoring tool to verify the correct functionality of our code (independant to the specific implementation). This method cut down on the somewhat tedious process of committing tests that naturally break code and following up with the correct code. We also made use of various resources such as Mockito or Google Espresso to accomplish the necessary testing for our Android code and behavior driven tests using Mocha for our server code.

Our group met regularly every week to discuss progress made on user stories, discuss overarching design choices and request action on any blocks to development. Collaborative development mainly took place within the subgroups of each area of development, though every team member was free to ask and discuss design choices and “best practice” questions. Within each sub team, effort was made to keep development tasks as orthogonal as possible to facilitate independent and concurrent development. Code was reviewed intra-team during development and cross-team when code was pulled from the development branch to the master branch to keep every subteam updated on the progress and design choices of every other subteam.

Requirements & Specifications

Device Requirements

- Android 4.0 (Ice Cream Sandwich) or above
- Chromecast (optional but recommended)
- Decent internet connection for all devices

User Stories

- Iteration 2
    - As a developer, I want to easily register players within my game.
    - As a developer, I want to easily login players within my game.
    - As a player, I want to be able to register within the Poker game.
    - As a player, I want to be able to login within the Poker game.
- Iteration 3
    - As a developer, I want to be able to retrieve a list of current lobbies for my game from the server.
    - As a developer, I want to be able to join a lobby from the list of available lobbies.

    - As a developer, I want to be able to create a lobby from my game.
    - As a developer, I want to receive real-time updates whenever a user joins/leaves a lobby.
    - As a player, I want to be able to view a list of available poker gamers to join.
- Iteration 4
    - As a developer, I want to easily integrate chat functionality into my games
    - As a developer, I want my users to be able to start a match.
- Iteration 5
    - As a developer, I want to be able to synchronize the game state across devices using an easy-to-use API.
    - As a developer, I want to be able to upload an HTML file to be accessed through the web/Chromecast.
    - As a player, I want to be able to view my poker hand and act accordingly.
- Iteration 6
    - As a player, I would like to be able to view the poker match on a secondary screen as it progresses.
    - As a player, I would like to be able to cast the poker match in-progress to my Chromecast device for viewing on a TV.
    - As a player, I would like to be able to play a full game of poker.

Use Cases

- Priority 1
    - The server provides RESTful API to the client.
    - The server provides real-time event messaging to clients.
    - The server provides database access to store information.
    - The server supports Cross-platform play.
- Priority 2
    - The developer adds simplified user registration functionality.
    - The developer adds simplified user login functionality.
    - The developer adds functionality that allows a retrieval of a list of available games.
    - The developer adds lobby functionality to the games.
    - The developer provides chat capability.
    - The developer extends synchronized game state with game-specific details.
- Priority 3
    - The developer adds waiting rooms for users to join before a game starts.
    - The developer adds game settings functionality.
    - The developer adds Chromecast support.

Architecture & Design

Server

Overview

The server is the key component of the Game Frame platform. All messages pass through the server, and all data must ultimately be stored on the server before being synchronized between clients connected to the server. The server is open-source, well-documented, and well-tested.

Implementation

The server is built on the Sails.js framework, itself which runs on top of a Node.js server. It utilizes the Waterline ORM built into Sails.js to interface with MongoDB for data storage and retrieval.

From a top-down perspective, the server consists of several controllers, each of which corresponds to a different set of features within the Game Frame platform. These controllers are:

- *Auth Controller* - authenticates users (i.e. user registration or login)
- *Dev Controller* - handles the Developer Console interface and associated methods
- *Game Controller*- manages the list of games and their corresponding lobbies
- *Lobby Controller* - contains all endpoints for lobby management and chat
- *Main Controller* - server functions (i.e. update on GitHub push and cookie-fetching)
- *Match Controller* - comprises all endpoints for handling the flow of a match
- *Public Controller* - consists of all endpoints that should be accessible without logging in
- *User Controller* - helpful endpoints to fetch user profile information

For much more detailed information about each controller, their specific endpoints, their parameters, and their error messages, please see the online documentation here.

Database Models

The server stores all information in a MongoDB collection as one of the following models:

- *User* - user profile information
- *Dev* - developer profile information (notably, a list of their games)
- *Game* - information about a given game (i.e. title, URL, associated lobbies, etc.)
- *Lobby* - contains all data regarding a lobby (i.e. users, messages, the game, etc.)
- *Message* - the content and metadata for a message object sent in a lobby
- *Match* - the current state of a match (i.e. the common state and each player state)
- *Common State* - a developer-defined JSON object representing all common information shared between all players in a match
- *Player State* - a developer-defined JSON object representing the individual, non-public information for a given player (i.e. a player's private hand of cards in a card game)

- *Event* - contains data sent between players and the server during a match

For much more detailed information about these models and their specific attributes, please see the online documentation here.

Why Sails.js and MongoDB?

The decision to build the Game Frame server on top of the Sails.js framework was significant in its consequences for the design of the rest of the project. Sails.js offers us an easy way to structure our server code using controllers, services, and policies, along with a simple means to test our code. Another vital feature of Sails.js is its native support for Web Sockets. We use this technology to enable our clients to communicate in real-time with our server, an important feature when designing real-time game systems. Lastly, Sails.js uses a database layer coined Waterline to handle data storage. Waterline allows us to switch which database we use without changing our code, which is definitely a nice feature. We ended up choosing MongoDB for our database, as it has great JSON support and worked well with how we wanted to structure our data.

UML Diagrams

See the UML Diagrams under the Android Client SDK architecture description (sequence diagrams reference communication with the Game Frame server).

Android Client SDK

Overview

The Android client SDK provides methods to interact with a GameFrame server through any Android application. The client will use these functions to send and receive data to and from other players, as well as for user authentication and account and game data storage.

Implementation

The Client SDK exists as an Android project, written entirely in Java. An Android application using GameFrame will communicate with the server through a series of managers. Each of these managers is meant to step a player through a stage of the game process and make sure their data is correctly synced. The managers create requests and handle responses as detailed by the server specifications. The managers are:

- Authentication Manager - Register and authenticate users
- Lobby List Manager - Fetch a list of ongoing lobbies and notifications of lobby updates
- Lobby Manager - Join and leave lobbies, track other users in a given lobby
- Match Manager - Send and receive updates to the current game and player states
- Message Manager - Send and receive messages in a match or lobby
- User Manager - Fetch information about users in a match or lobby

For much more detailed information about the managers and their specific functions, please see the online documentation here.

All interactions with the server happen on a socket level, and none of these calls will ever block on the executing thread. Since the responses to all of these interactions must be managed by the application, each message to the client is put on an event bus that the application can respond to. A Java method with the @Subscribe annotation, the “public” and “void” signature, and one argument will be called by the event bus. The argument must be the event the application wishes to respond to. These events are:

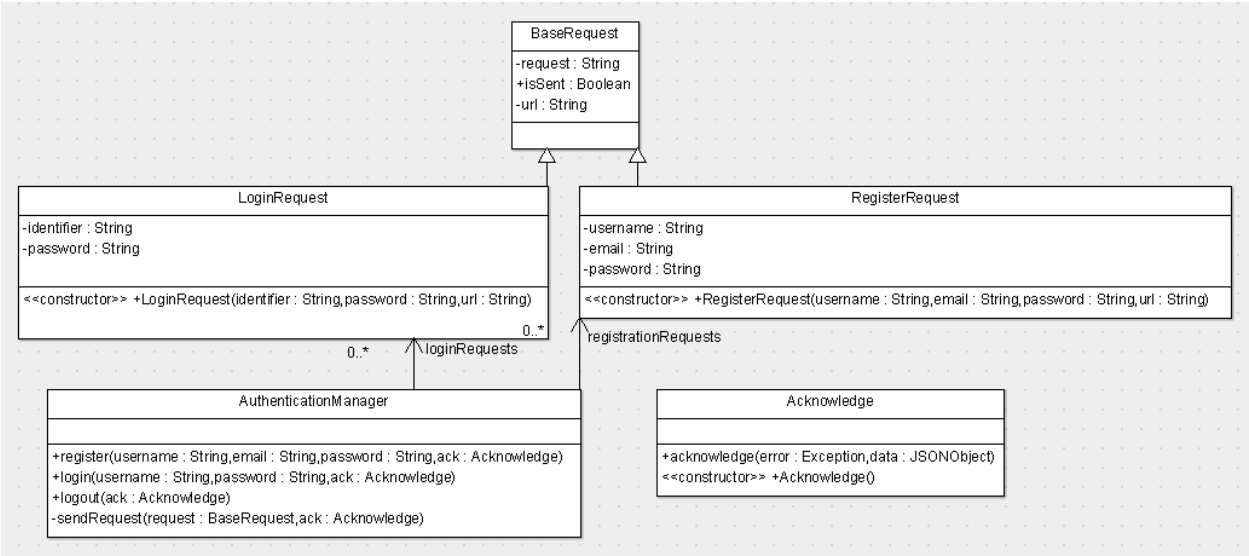
- Broadcast Event - A player has sent game information to one or more other players
- Lobby Add Event - A lobby has been added to the list of available lobbies
- Lobby Create Event - Your requested lobby has been created and added to the list
- Jobby Join Event - The local player has been added to a lobby
- Lobby Leave Event - The local player has been removed from a lobby
- Lobby List Event - The current list of ongoing lobbies
- Lobby Remove Event - A lobby has been removed from the list of available lobbies
- Local Host Event - The local player is now the host of a lobby
- Match End Event - The match has ended
- Match Started Event - The match has started
- New Message Event - A new chat message has been received
- Remote Host Event - A player other than the local one is now the host
- Request Event - A request from another player has been received
- Response Event - A response from another player has been received
- Turnover Event - The current player’s turn has ended
- Update Common State Event - The common state has changed
- User Add Event - A user has been added to the lobby
- User Login Event - The local user has been logged in
- User Logout Event - The local player has been logged out
- User RegisterEvent - The requested user has been successfully registered
- User Remove Event - A user has been removed from the lobby
- Users Fetched Event - The information of the requested users

For much more detailed information about the events and their content, please see the online documentation here.

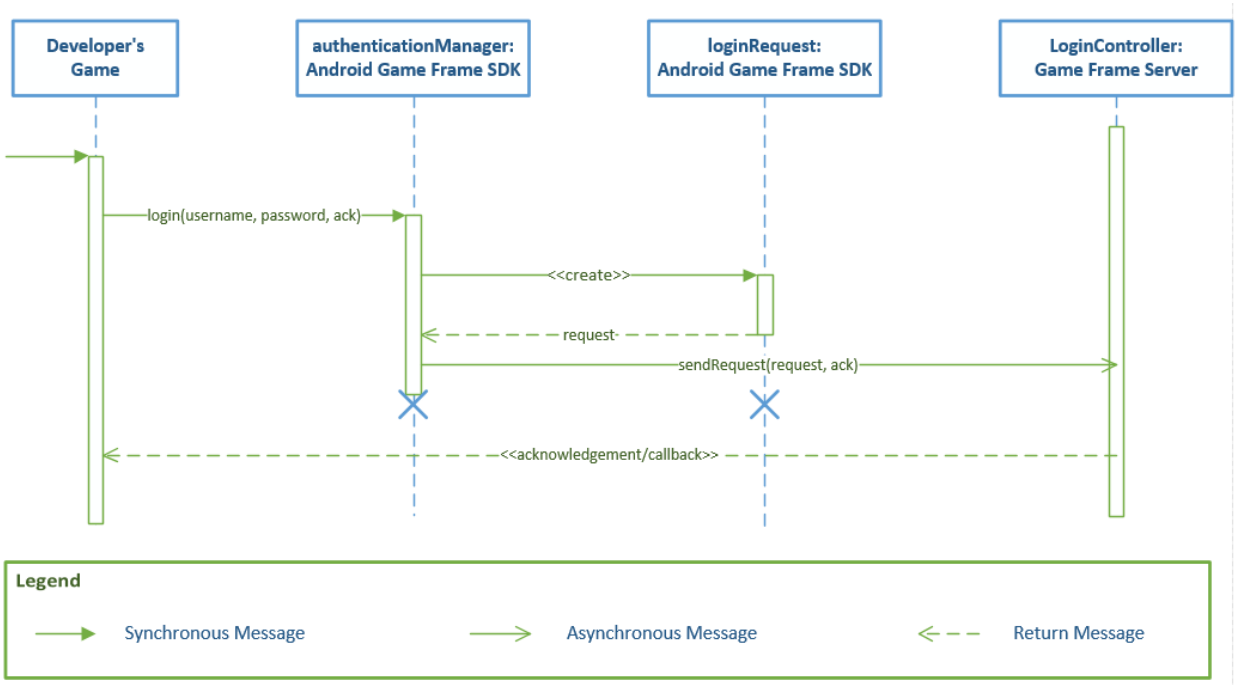
The Android SDK parses the JSON objects from the server, formatting them into one of these events. Developers can use these events and the Java versions of the models described in the server section to manipulate their application logic.

UML Diagrams

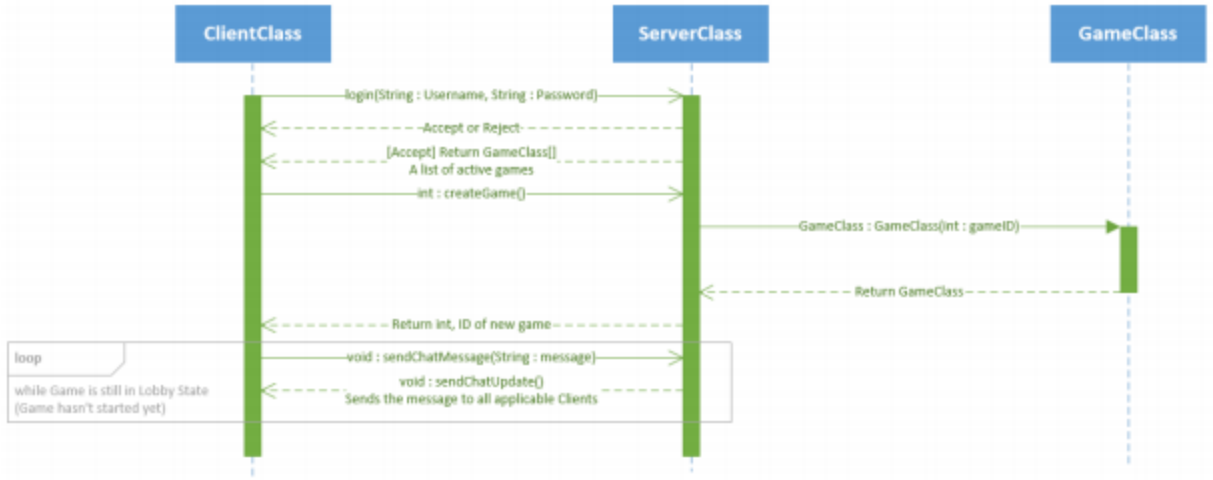
User Registration and Login - Class Diagram



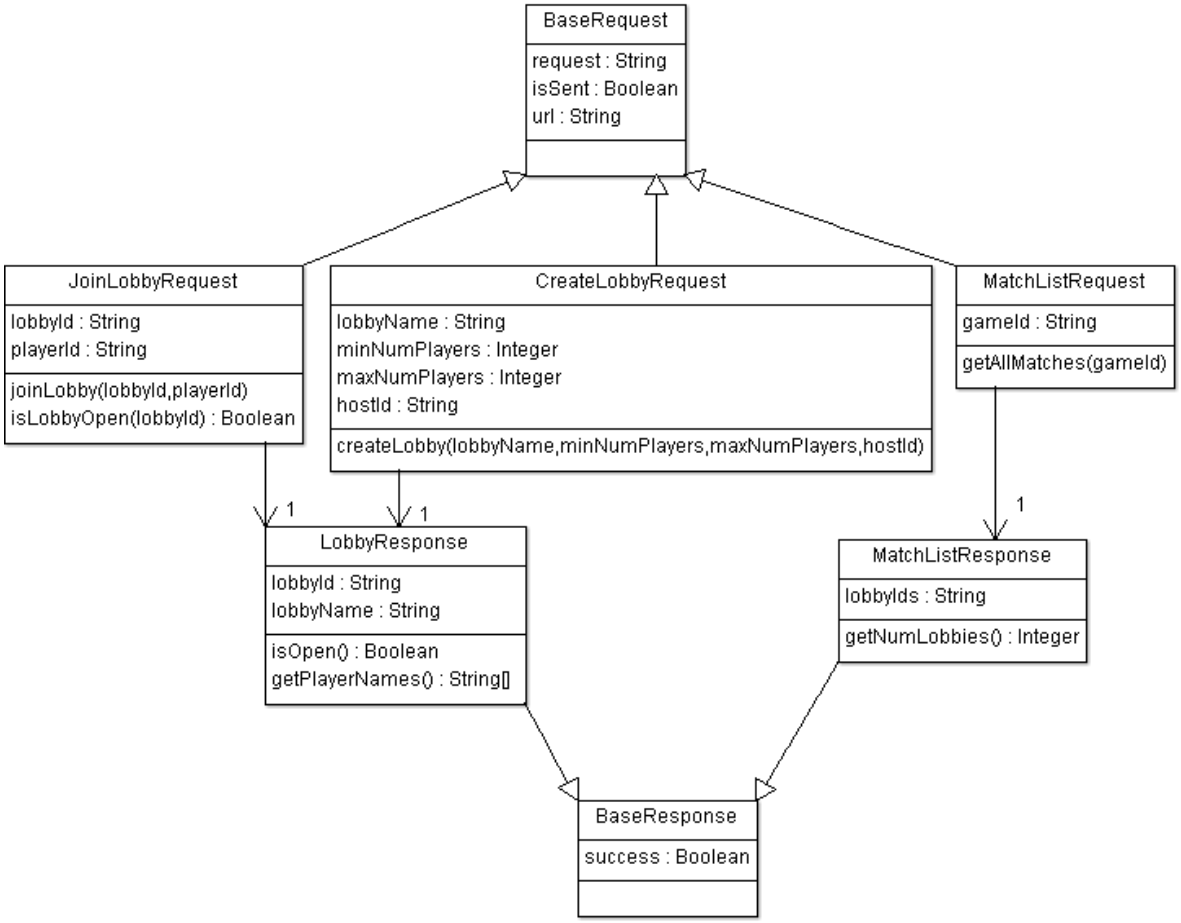
User Login - Sequence Diagram



Chat System - Sequence Diagram



Requests and Responses for Lobbies - Class Diagram



Web/Chromecast

Overview

The design of Game Frame allows any client that implements the proper API to communicate with our server. For our project, we focused on creating an Android client SDK to communicate with our server. However, we did began to implement a Javascript client. Currently, this is not a full Game Frame client: rather, it is designed to solely receive updates from the server and update its display according to the data received (i.e. it does not send data back to the server).

Implementation

Our current Javascript implementation of a Game Frame client for spectating a match can be viewed while using the Chromecast feature of the Poker sample game. Here is how the Game Frame Android client SDK sets up the Chromecast for viewing a match:

1. The user's device detects a Chromecast on the same WiFi network and displays the Cast button in the Action Bar at the top of the screen.
2. The user taps the Cast button and selects the Chromecast device to cast the match to.
3. After connecting with the selected Chromecast device, the device tells the Chromecast to open a web browser pointing to our web app, located here.
4. After the website is open, the user's device transmits the lobby ID to the Chromecast.
5. The Chromecast subscribes to all events for the given lobby through the server.
6. Every time a player updates the common state or broadcasts a message, the Chromecast will also receive the data and can update its display accordingly.

Note: The Chromecast is just directed to display a web page. As such, our implementation also allows any web browser to visit our Poker web app, input the lobby ID, and spectate on the match as well. There is nothing particularly specific about using a Chromecast, other than it allows players an easy way to view the game on their TVs.

Integrating Game Frame in Your Application

Game Frame provides developers with easy-access to a variety of features, allowing them to build high-quality games in less time. In order to integrate Game Frame into your app and gain access to its features, follow these steps:

1. Find or create a Game Frame server to host your game.
    a. The developers of Game Frame host an open server, available for all developers to use, at: http://gameframe.dyndns.org
    b. Otherwise, host your own server. To do this, please see the following section on hosting your own Game Frame server, then return to Step 2 below.
2. Once a server is decided on, register as a developer. You may do this by visiting the address of your server in a web browser, which will open the Developer Console.
3. After registering as a developer and logging into the Developer Console, you may add a

game to your developer account. Once you do this, be sure to note the game ID listed for the game you created (it is used later when configuring Game Frame in your game).

4. Clone the Game Frame Android client SDK repo using:
   git clone https://github.com/shauntoomey25/gameframe-android.git
5. Add the newly-created “gameframe-android” as a library dependency for your Android project. This differs depending on the IDE you are using for Android development. In Eclipse specifically, right-click on your project, go to Properties, Android, and add it as a library dependency by clicking the “Add” button for the “Library” section. For other IDEs, consult other online documentation and tutorials as necessary.
6. Initialize the Game Frame SDK with the proper configuration options in the Application object instance (i.e. one of the first objects created when an Android app is started) within your game. Specifically, you will need to set the URL of your server and the game ID. This is best demonstrated through example. For reference, please see the application object of our Poker sample app here.
7. Your game is now ready to use all of Game Frame’s features. Congrats!

Hosting a Game Frame Server

Here are the steps to host your own Game Frame server. For more detailed and up-to-date instructions, consult the README file from the repo here.

1. Install Node.js, MongoDB, and Git on your server. This process varies widely between platforms and releases. As such, consult other online documentation and tutorials pertaining to your specific server configuration as necessary.
2. Clone the Game Frame server repo using:
   git clone https://github.com/shauntoomey25/gameframe-server.git
3. Run the command “npm install” inside the root of the “gameframe-server” directory to install all project dependencies.
4. Configure the server for use by creating a file at the path ‘\config\local.js’ with the proper formatting and parameters (more information on this can be found in the README here).
5. Start the server with the command: sails lift

Modifying the Game Frame Source

Server

To modify the code behind the Game Frame server:

1. Get a local instance of a Game Frame server up and running (see previous section of this document for instructions).
2. Ensure that all tests pass by running the command: npm test
   Note: See the README here for instructions on configuring testing and debugging for

the Game Frame server.
3. Modify the code as you like. Contributions are welcomed as long as they follow the Node Style Guide (link). Writing tests for all changes to the code is highly recommended.

Android Client SDK

To modify the code behind the Game Frame Android client SDK:

1. Clone the Game Frame Android client SDK repo using:
   git clone https://github.com/shauntoomey25/gameframe-android.git
2. Import the project into your favorite IDE.
3. Ensure that the imported project has the proper dependencies properly setup:
   a. android-support-v7-appcompat
   b. android-support-v7-mediarouter
   c. CastCompanionLibrary-android
   d. google-play-services_lib
4. You may test the code by running the tests in the “test” folder as jUnit 4 tests.
5. Modify the code as you like. Contributions are welcomed as long as they follow the Android Code Style Guidelines for Contributors (link). Writing tests is highly encouraged.

Sample Code

Hello, World!

This sample is included within the Android client SDK under the “sample-project” folder. It serves as a basic example to demonstrate user registration, user login, lobby functionality, the chat system, and simple game state communication between devices.

Poker

To further demonstrate the use of GameFrame, a simple poker game was built in Android that takes advantage of most of the features of our framework. All of the network and administrative tasks are handled by GameFrame, including account registration and login, allowing users to organize into lobbies and chat, and the transmission of game data to and from individual clients. Furthermore, our Poker game includes Chromecast support, allowing players to view the poker table on a TV using a cheap Chromecast device.

To view the code, you can find the Android code here and the web code here.

Future Plans

Over the summer, our team plans to continue refining Game Frame. We have several new features and areas of improvement that we have identified:

- New user interface components in the Android client SDK to make it even easier to implement Game Frame into your game
- Improved Developer Console
- Creation of a full Game Frame SDK for both Javascript and iOS

Near the end of the summer, we plan on releasing Game Frame to the public. We will be open-sourcing all of our repositories on GitHub under a permissive license to encourage adoption. Releasing Game Frame will be as simple as making our repos public, updating our documentation website (link) to reflect the release, and announcing it to the public.

Reflections

Shaun:

I enjoyed working on this project quite a bit. I'm actually quite proud of the product we ended up with after only a semester. The most important success for me is that we ended up with a very clean design and internal implementation. Being able to step through the entire process of creating something like this was very interesting. It showed me how much time and thought work a good design backed by good code can require, but also how much that pays off; we were able to implement an entire game quite easily once the framework was working as intended. I also got to discover some new technologies I was interested in and flesh out my development skills with new design patterns and architectures.

Hussain:

I enjoyed this semester very much and I'm happy with what we have accomplished in this time. I had my hands in both the Android and server code, which helped me keep the big picture of the framework in mind the entire time. I have learned much about web development by using libraries like Sails. I have also had a lot of fun implementing the chromcast functionality on both sides. I think we can easily make a card game like Poker within a week, which is what we set out to do, so I am content with the work we have done on this project so far.

Nathan:

I really enjoyed my time this semester working on the project. As I was in charge of server development, I had the opportunity to learn a lot about Node.js development, MongoDB, and in particular, BDD-style testing. In the past, I only wrote tests when required for classes. However, this project has helped me understand how a comprehensive set of tests can be highly valuable

for a project. Multiple times, I was able to use the tests to quickly narrow my search for bugs, which was incredibly useful. But overall, I had a great time working with my team as we iteratively added new features to Game Frame on a weekly basis throughout the semester. Without contest, I am more proud of this school project than any other I have worked on.

Chris:

I think this class was more fun than CS 427. I was able to enjoy working on the project that was more meaningful for me, and I am glad that other members were enjoying it as well. It is not an easy opportunity, especially in classroom settings, for anyone to join the development team consisting of hardworking and kind people. On the programming side, this class provided me a slight taste on how merging the code across the team actually looks like. If I were to work alone, then I could have fixed or left any part of the code until I felt like doing it, which is definitely not the case when working together. Overall, I am happy to see that my team's project became successful, thanks to everyone's hard work.

Paul:

Overall, I had fun working on this project. Android was a completely foreign topic to me at the beginning and even the idea of writing the behavioral code and defining the layouts in xml were quite new. Working with git over svn was also quite a learning experience. Working on this project proved to be quite insightful on many topics including Android development and the networking aspect of coding. I felt that the process we followed was quite effective, we were able to make a significant progress while minimizing blockage. I learned quite a lot by working on this project and I feel like the end product is something that is not only fulfills the requirements we set out to complete, but also something that is truly useful.

Alan:

This project held a huge opportunity to learn and explore, ranging from Android to the Web as well as the backend with the server. I ended up dealing with the Poker application on the Android. Although I have experience with Android, my area of expertise is limited and I learned ways to play around with Android, such as extending view classes. Additionally, this was the first time I used Git, so I made multiple mistakes and caused trouble for others. However, my teammates were kind enough to tolerate them and inform me of what I did wrong, I learned how to avoid such mistakes. I'm glad to have been working with my teammates over this project, that everyone has been putting their best effort to make this project successful and that this experience and project is something that we can put into great use in the future.