

# PROJECT Design Documentation

## Team Information

- Team name: Resistance
- Team members
  - Justin Lam
  - Alan Tan
  - Jesse Chen
  - Elijah Cantella
  - Jay Gogri

## Executive Summary

This is the documentation for our project after Sprint 1. What has been implemented is creating the backend for the Sign In and also displaying the Checker board with the pieces on its respective sides.

## Purpose

To allow users to sign in, sign out, and enter a game with another player by clicking on the opponent's name in the home page.

## Glossary and Acronyms

Provide a table of terms and acronyms.

Term	Definition
VO	Value Object

## Requirements

During this sprint, we were given two main tasks to complete

- Implement a Sign In interface so that users can sign in and sign out
  - Name restrictions
    - \* No duplicate names
    - \* No null names such as double quotes (“
  - List out all the users currently online only if the user is sign in
- Implement a Game interface so that users can start a game with each other
  - Utilizing that list a user can click on another player and start a game with him/her
    - \* If selected player is in a game than, a message should show that a game cannot be started
  - The pieces on the board should be displayed properly, with challenger as red and challenged as white, with respective pieces on the bottom of each user's board
  - Pieces should be draggable and droppable on valid places on the board

## Definition of MVP

The Minimum Viable Product should be a product that can sign a user in and out (if such user is already signed in), and start a game with properly aligned pieces.

## **MVP Features**

- Sign In
  - Player Sign-In
  - Player Sign-Out
- Game Play
  - Player Setup
  - Disc Placement

## **Roadmap of Enhancements**

- Player Turn
  - Player Movement
  - Player Capture
  - Player King
- End Game
  - Player Win
  - Player Lose
  - Player Quit
- Logging System
  - Pause
  - Resume
- Chatting System
  - Game Chat

## **Application Domain**

This section describes the application domain.

### **Overview of Major Domain Areas**

The main domain areas are the Player, CheckersGame, Board, Squares, and Pieces. These define the checkers game that two players will play utilizing pieces on a board.

### **Details of each Domain Area**

- Pieces are utilized to represent either Player on the board, which is either a red or white piece.
- Players play a CheckersGame which is played on a Board represented by Squares.

## **Architecture**

This section describes the application architecture.

### **Summary**

The WebCheckers webapp will use a Java-based web server, this is done using the Spark web micro framework and the FreeMarker template engine to handle HTTP requests and generate HTTP responses.

## Overview of User Interface

The application's user interface consists of two parts, the server UI and the Client UI. In the Server UI it is made up of the various routes we implemented so that the Spark framework and the FreeMarker template engine's are able to handle HTTP request and HTTP responses. We also alter and made new ftl files in order to create the Client UI, which is a combinations of HTML and CSS. Together these two parts made up the UI views and UI controllers.

### UI Tier

The user of the application interacts with the UI tier, then the UI tier interacts with the Application and Model tiers. We implemented various routes in order display proper information back to the user. \* `GetGameRoute` \* This class deals with initializing a game, where a player had clicked on an opponent in the home screen to face against. \* If the the player had clicked on an opponent already in a game then he/she will be sent back to the home page with a error message and may click on another opponent. \* If it is a valid game, then the player will be the red piece and the opponent will be the white piece with respective pieces on the bottom of whoever's screen. \* `GetHomeRoute` \* This is the home screen of the whole application. This is what the player will see as the first thing he/she opens up the web browser. \* In the navigation bar the player may sign in. \* Only when he/she is signed may he/she see all the players online and click on someone to face against. \* `GetSignInRoute` \* When a player decides to sign in, this class allows the player to sign in with a username \* `PostSignInRoute` \* When the player enters a username of their choice, this handles if the username is a valid username or not. \* Returns a error message if the username is taken or if it is a empty username. \* `GetSignOutRoute` \* Signs a player out and returns the player to the home page.

### Application Tier

The Application tier holds the logic that controls the flow of the application. \* `GameCenter` \* This class initializes a `CheckersGame` instance. \* If there is an existing game instance, it will return that instead, so a game wouldn't be lost. \* `PlayerLobby` \* Holds all the online players so that they can be displayed in the home page. \* `PlayerServices` \* Each player needs to be part of a `GameCenter`, where they will be part of a `CheckersGame`

### Model Tier

The Model holds all the of the core domain logic. \* `CheckersGame` \* Is part of the `GameCenter`. \* Holds a Board. \* `Board` \* Initializes the board by passing in white or black tiles to the `Row` class. \* `Row` \* Creates each row on the board using the `Square` class. \* Places all the pawn pieces in the rows on the board \* `Square` \* Contains information on each individual square on the board. \* If the square is valid, if it has a Piece. \* `Piece` \* Contains information on a singular piece, the color and type. \* `Player` \* Contains information on each user of the application as long as they are signed in. \* Holds his/her username.

## Sub-system GameCenter

This section describes the detail design of sub-system `GameCenter`.

### Purpose of the sub-system

Essentially this the brains of the game part of the application, it is responsible for assigning responsibilities onto other classes. The game hiearchy is basically a `GameCenter` returning a game instance either new or existing, then the game has a Board, which has a Row containing Squares with Pieces on it. Then all of these is handled with by the `GetGameRoute` which displays everything.

## Static models

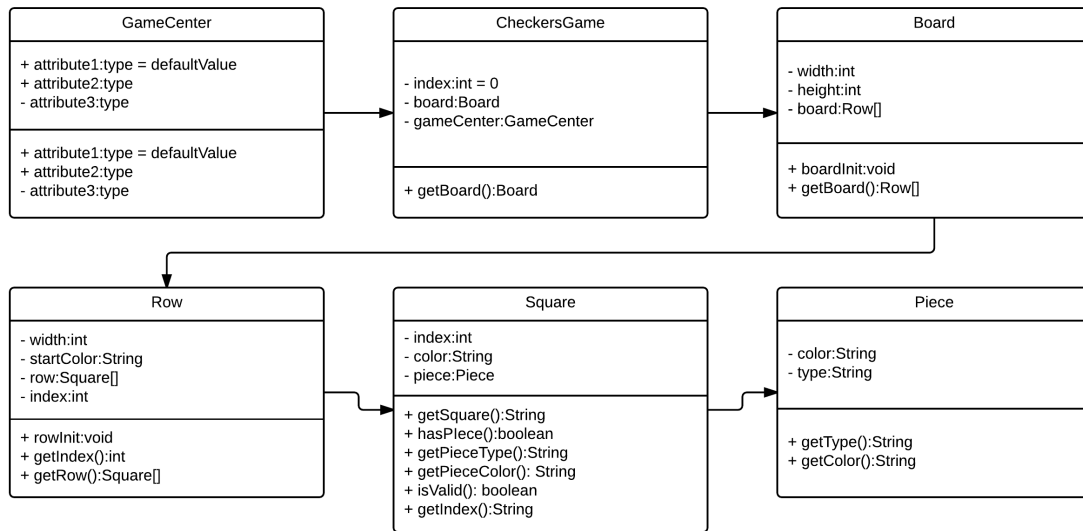


Figure 1: UML for the Sub-system GameCenter

## Dynamic models

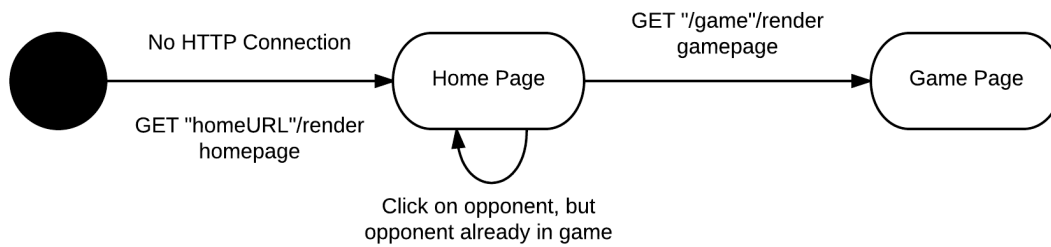


Figure 2: State Diagram