CENG 453 - Pişti the Game
Spring 2021
General Documentation

Varol, Rabia
rabia.varol@metu.edu.tr

Gerçel, Doruk
doruk.gercel@metu.edu.tr

July 5, 2021

---

1 Components

In this project, we have grouped all of our components in to two main groups: **server** and **client**.

Components that are responsible for dealing with the user interactions and rendering of the user interface graphics are grouped under **client** components. Therefore we can safely state that **client** components are mostly responsible for dealing with the *view* of the project. These components can be divided into subgroups. There are *controllers* which are responsible for changing and controlling the view of the application. All of the scenes are defined as an *FXML file* and these components are their view controllers. When an UI event is triggered, these components perform the necessary actions or dynamically adjust the view. Moreover *controllers* can be subdivided into two categories as-well *base controllers* and *network base controllers*. Network base controllers are inherited from the base controllers. Their only difference is, network base controllers are able to send requests to the back-end. Also there are several *manager* components which are responsible for performing background tasks. For example they deal with navigation between the scenes, initializing card photos or creating a request, response connection. (These components can be find under *definitions* directory.)

Rest of the components can be grouped under **server** components. These components can actually be divided in to subgroups under the **server** category. There are *repository and model* components which are responsible for creating and managing data-tables. Also they are responsible for dealing with the transactional functionalities as-well: like addition or update of data in the tables or retrieval of the data from them. Another subgroup of components, is *API*s. These components are actually REST controllers that provide API services and communicate with the **client** side of the project. They retrieve the user interactions from the *client* side of the project and send back necessary information to be displayed or to be used in the front-end. Also in the **server** side of the project there are stand alone *classes* that can be named as *definitions*. These components can actually be considered as fill-the-gap type of components. Their main purpose is to represent data types that are used by other components and reflect the concept of the project (in this sense, reflects the pişti card game). Last group of components are named as *services*. These components are the glue that holds the all other components together. They are responsible for managing the interactions between the components and (therefore) to control the main programming logic. Also security functionality is implemented in the server components as-well. The text are not stored as plain text. The password typed by the user is encrypted by using *BCrypt Password Encoder*. This encoder is especially chosen as it produces different results in each call. According to the description it is also stated that "BCrypt will internally generate a random salt ".

2 Multiplayer Game Protocol

In the implementation of the multiplayer game, there is a need of collaboration between the client and the server components. In the server side there is a multiplayer game map (similar to the single player game map), which holds the records of the active multiplayer games. When client interact with the server, client sends the game id as-well therefore directly interacts with the corresponding game instance. The multiplayer game instance is a thread safe component, it stores the id's of the both players and allows them to perform moves in-order accordingly. This component is thread safety strategy is as follows: It allows one of the user to perform the move and change the game state. After changing the state this thread signals the reader thread (the player that doesn't make the move but observe the changes in the game state), and gets blocked until reader thread finishes reading (reader thread was also blocked until changes in the game state were made). After reading this thread signal the move performing thread, and they send the necessary messages back to the clients. In the client side a logic is implemented so that, the clients may allow the user to perform actions or just observe the changes in the game state. Therefore although the main intelligence and the thread safety mechanisms are in the server-side, a certain amount of intelligence is added to the client side as-well in-order to act according to the messages.

3 Technologies

- **Spring Boot** is used to develop the back-end components. Spring Core, Spring Web, Spring Data JPA and Spring Security are used.
- **JavaFX** is used for the development of the front-end of the project.
- **MariaDB** is selected to be the DBMS technology of the project.
- **Apache Tomcat** is used as the servlet container and for serving the back-end of the project.

4 Package Structure

The project is divided in to two main packages: **server** and **client**. In the **server** side, the packages are created according to their group of functionalities, therefore the name of the sub-packages are like: api, service, model etc. Also under the packages project is subdivided according to it's context and use-cases to clearly demonstrate the functionalities. The same approach is used for the **client** side as-well. There are sub-packages like: controller, definitions, model etc. Apart from these packages, **client** side includes several resource packages as-well. The FXML files, CSS file for styling and the images that are used in the application are placed under resource packages.