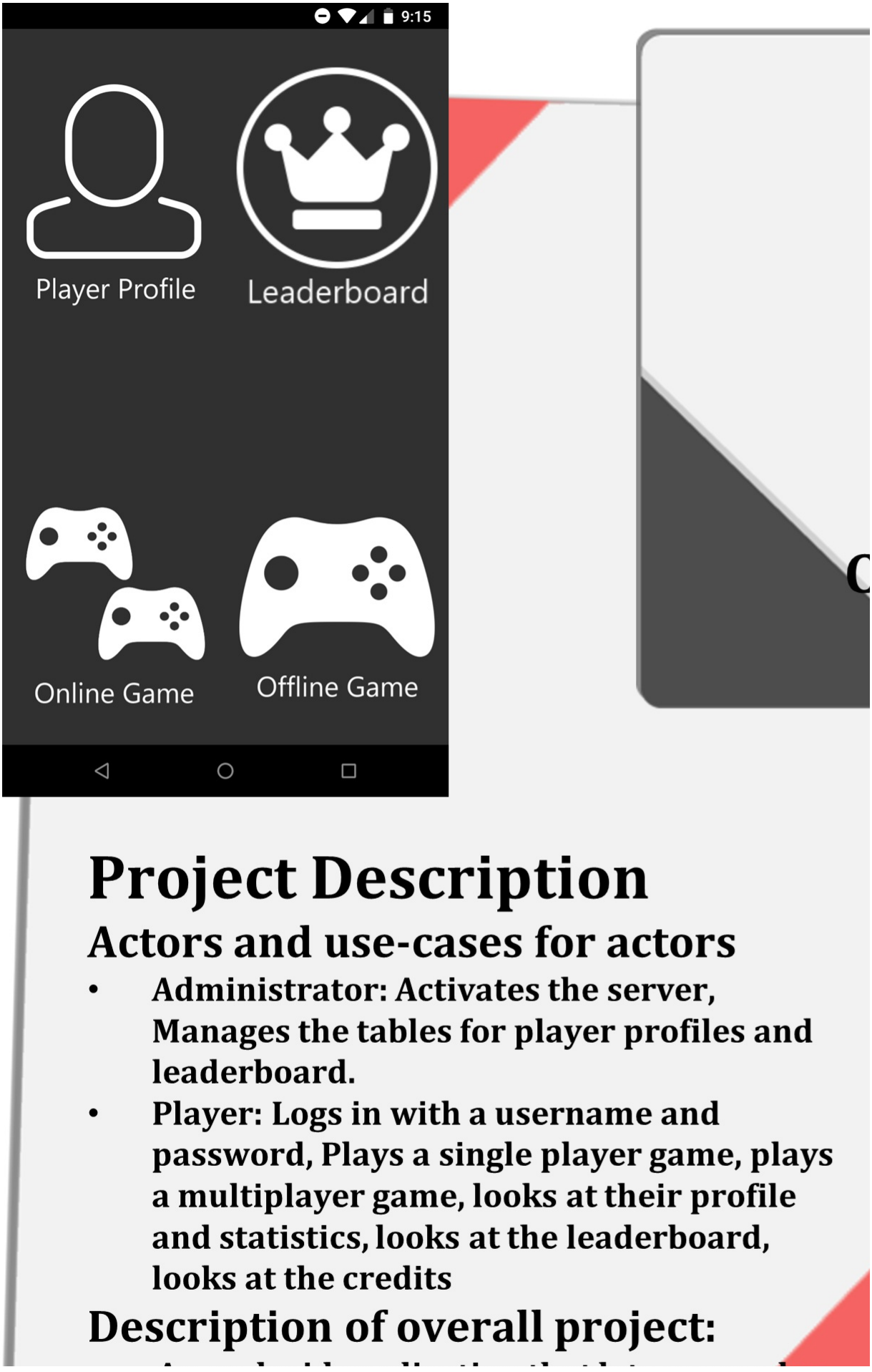
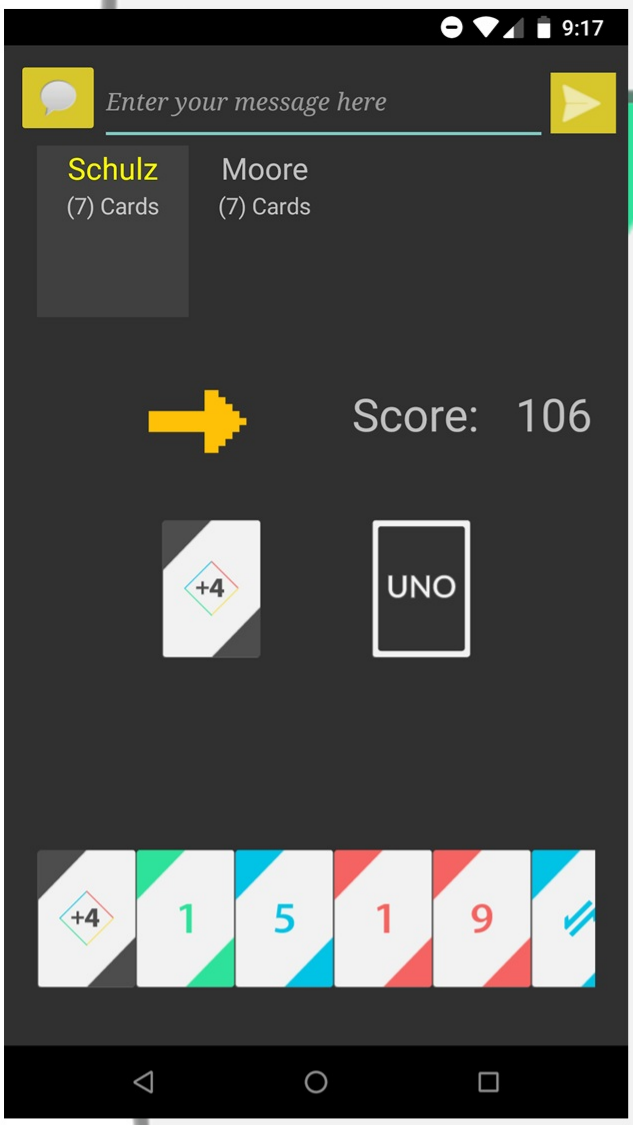
Multiplayer UNO Android Game Application

- This class focused on using standard software development practices in small teams to develop some software application. Our team decided to develop an Android Application that allowed a user to play the UNO card game against either a computer or other players over a network connection.
- Our team followed an Agile development cycle with 2 week sprints where we'd meet and demonstrate the progress on the project, gain feedback, and discuss future tasks.
- The back-end was developed using Java, Android Studio, Spring boot, and MySQL.
- The front-end was developed using Java and Android Studio.
- I have included a copy of our final poster for the application that includes all the relevant information regarding the development of the application. >



An android application that lets users play the card game UNO online with friends or offline against AI. It also lets players create profiles and climb the leaderboard rankings.



Module Interfac

Server/Database

ChatRepository Interface
- Creates a repository to interact with the chat table from the c the JpaRepository

ChatController
- A web controller class that maps weblinks after /chat and ad data from the chat table using the ChatRepository.

LeaderboardRepository Interface
- This interface manages the repository to interact with the Le

LeaderboardController
- A web controller class that maps weblinks after /leaderboar returns data from the leaderboard table via LeaderboardRep

ServerSocketApplication
- Static void checkForWin(UnoPlayer player)
  Checks for a win for the current player
- Static void handleAction(UnoCard card, UnoPlayer currentPl
  Deals with the actio cards dealt
- Static void runSocket()
  Starts the socket for communication between the serv
- Static void setUpGame()
  Sets up the multiplayer game and broadcasts it to all tl
- Static void simulateTurn(UnoCard card)
  Simulates a turn in the Uno Game

Client

Void launchLobbyActivity(java.lang.String username)
- Launch the HUB activity, carrying username intent.

io.socket.client.Socket getSocket()
- Getter method for getting the socket in activities throughout

Design Decisions:

Designs we ended up going with (Single lobby vs multi-lobby, no ranked matchmaking, sockets vs turn-based, Player and Leaderboard databases

- Used sockets for instant gameplay for multiplayer
- Modulated the leaderboard, profiles, and gameplay to simplify working on seperate parts.
- Single player games with CPUs and multiplayer games without CPUs

multiplayer games without CPUs

- Dual-threaded, one for gameplay and one for database management.
- Database of player profiles and leaderboard players.