<u>Software Requirement Document</u>

Casino Game Application

<u>03/13/2021</u>

<u>Status</u>: Draft

<u>Prepared For</u>:

Ike Quigley

Software Engineering class

CSC 340-02

<u>Prepared By</u>:

The Four Lokos!

★ Serena Wisnewski

★ Alana Traylor

★ Keshawn Posey

★ Srushti Honnenahalli

Stakeholders/Company - Honor Code:

We have abided by the UNCG honor code, the Academic Integrity policy, and syllabus requirements which states the following:

"All work (including assignments and tests) is subject to the UNCG Academic Integrity Policy. When you submit your work and exams, you are implicitly agreeing to this policy. Academic dishonesty includes submitting for credit any software that you (or your team, in case of a team project) did not write yourself/yourselves. Violations of the Academic Integrity Policy will be reported to university administration."

Name: Alana Traylor	**Signature**: Alana Traylor	**Date**: 3/17/21
Name: Keshawn Posey	**Signature**: Ke'Shawn Posey	**Date**: 3-17-21
Name: Serena Wisnewski	**Signature**: Serena Wisnewski H.	**Date**: 03/17/2021
Name: Srushti Honnenahalli	**Signature**: Srushti.H.A.	**Date**: 03/13/2021

Table of Contents:

1. **<u>Introduction</u>:**

The Software Requirement Document (SRD) expresses a deck of cards, which summarises the War card game and uses features such as shuffling, drawing, listing, adding cards, creating a new deck, calculating the coins or points gained or lost by the players, and Declaring War on the opponent. This app is created by implementing a Deck of Card API and creating our own features.

<u>References</u>:

- Deck of Cards API

1.1. Purpose:

The purpose of this Software Requirement Document (SRD) is to describe the User-view and Developer-view requirements and features for the card games in this Casino Application. The user-oriented requirements focus on the User perspective of the system, such as, detailed description on product features, etc. The developer-oriented requirements focus on the Developer's perspective of the system, such as, detailed description on functional requirements, performance, technical requirements, and other important requirements. We created a fun and functional application whose purpose is to allow the user to win points and reach leaderboard status by playing a simple card game "*I Declare War*". This app has no "in app purchases" nor microtransactions.

1.2. Document Conventions:

This Software Requirement Document (SRD) is structured as follows:

- ❖ Section 1 contains the above mentioned introduction and purpose.
- ❖ Section 2 contains the description of our application, including it's features, characteristics, and constraints.

- ❖ Section 3 contains functional requirements.
- ❖ Section 4 contains technical requirements, including compatibility and interface requirements.
- ❖ Section 5 contains additional nonfunctional requirements including performance, safety, and security information.

1.3. Intended Audience

The intended audience is Ike Quigley. Besides Ike Quigley, the app would be used by the general population, especially gamers and people who are interested in card games. Plus, this application contains card games, like War, which are rated **E**, i.e., for everyone since the content is suitable for all ages.

1.4. Definitions/Jargons:

- ★ SRD: An **S**oftware **R**equirement **D**ocumentation explains how the software operates and/ how the software is to be used. The documentation will consist of functional and technical requirements, design structure, end-user manuals, and non-functional requirements.
  - ○ Reference:

    Wikipedia contributors. (2021, January 24). *Software documentation*. Wikipedia. https://en.wikipedia.org/wiki/Software_documentation#Composing_user_documentation.
- ★ API: **A**pplication **P**rogramming **I**nterface helps to deliver a user response to a system and vice versa.
  - ○ Reference:

What is an API? (Application Programming Interface). MuleSoft. https://www.mulesoft.com/resources/api/what-is-an-api#:%7E:text=API%20is%20the%20acronym%20for,you're%20using%20an%20API.

- ★ Java IDE: **I**ntegrated **D**evelopment **E**nvironment is a software for building applications which combines developer tools into a single graphical user interface. An Java IDE allows users to easily write and debug Java language programs.
    - Reference:

      Contributors to Wikimedia projects. (2019, December 26). *Java Programming/Java IDEs*. WikiBooks. https://en.wikibooks.org/wiki/Java_Programming/Java_IDEs.
- ★ GUI: **G**raphical **U**ser **I**nterface "is a type of user interface through which users interact with electronic devices via visual indicator representations."
    - Reference:

      What is a Graphical User Interface? Definition and FAQs | OmniSci. OmniSci. https://www.omnisci.com/technical-glossary/graphical-user-interface.
- ★ **Note**: Will define more jargons as we continue. These are just a few for now.

1.5. Project Scope:

The project purpose is to build an application that implements an API and is written in Java language. In our application we are creating a Cassino Application that has the

War card game within it. The application will be user friendly and calculate the points earned by users accurately.

1.6. Technical Challenges:

A technical challenge would be we have less experience with external APIs and making the text file connect and edit the stored data.

1.7. References

- Deck of Cards API
- Java IDE
- Use Case Diagram

2. <u>Overall Description</u>:

War is a simple card game which is typically played by two players using a standard playing card deck and the objective of the game is for a player to win all the cards in the match. The two players must deal the deck of cards. Each player must play a card and is compared in a "battle" match. The higher card, either by value or rank,wins the battle, and the winner takes both cards, adding them to the bottom of their deck. If there is a tie of points in a round then WAR is declared. This process is repeated until a player loses the game which is if they run out of cards during the War. The cards are represented as Integer values from 2 to 14 and the suits as strings: Diamond, Clubs, Spade, and Heart. Also, the higher card is not decided by suits because in this game the string holds no specific value to determine the bigger card so the suits are irrelevant to decide.

Note: If the winner isn't declared by the 1000th battle in the game of war, then the game is declared as a Draw.

2.1. Product Features:

The product features for this app are:

- ❏ A Game Play where users play "I Declare War" card game against a computer.
- ❏ A Leader Board where users can see top players with the most wins
- ❏ User Profile where users can view personal game statistics and update personal information.
- ❏ Login/Sign Up where users create and access a personal account which saves personal information and player scores.
- ❏ Implements Deck of Cards API.

- ❏ Java language is used for the code.

2.2. User Characteristics:

Users are anyone interested in playing simple card games. It's open for everyone, there are no restrictions for age, gender, education level nor cultural background.

2.3. Operating Environment:

The operating environment includes a graphical user interface where players interact with decks of cards in order to play “I Declare War.” The application was created at home with the use of materials from class, zoom meetings, Java IDE, gitHub, Deck of Cards API, and help of Lucid chart to create the Use-case diagram.

2.4. Design and Implementation Constraints:

Being mobile-friendly would be a design and implementation constraint.

2.5. Assumptions and Dependencies:

<u>Assumption</u>:

- The card API remains active and does not change its current functions.
- Approximately, the players must get the same number of turns over the course of this game.
- Players can't undo or redo their moves.
- Players understand the rules of the game.
- The suits (diamond, heart, club, spade) have no effect on the calculations of points/coins.
- Login site is easily accessible and user friendly.

<u>Dependencies</u>: Creating and reading through the users.txt file.

3. Functional Requirements:

3.1. Primary:

Primarily this Card game app should be able run the WAR card game and divide the deck evenly among the players, user and computer.

3.2. Secondary:

The application should receive the users log in data and should be stored in a text file based database so that it could be used for various analytic dashboards or reports.

4. <u>Technical requirements</u>:

4.1. Operating Systems/Compatibility:

It can run on Windows, Mac OS, and Linux operating systems.

4.2. Interface Requirements:

4.2.1. User Interface:

- Front-end software: Java Netbeans IDE
- Back-end software: Text file under database folder called as users.txt

4.2.2. Hardware Interface:

- Windows, Mac OS, Linux operating systems.
- DeckOfCards API, Netbeans IDE, Github.

4.2.3. Software Interface:

Software Used	Description
Operating system	The api and application is supported on Windows, Mac OS, and Linux operating systems. So, it provides good support and provides user-friendliness.
Database	To just store the login and signup information, we have chosen to use a simple .txt file to keep track of the information.
Java Netbeans IDE	To implement the project we have chosen Java Netbeans IDE language for more interactive support.

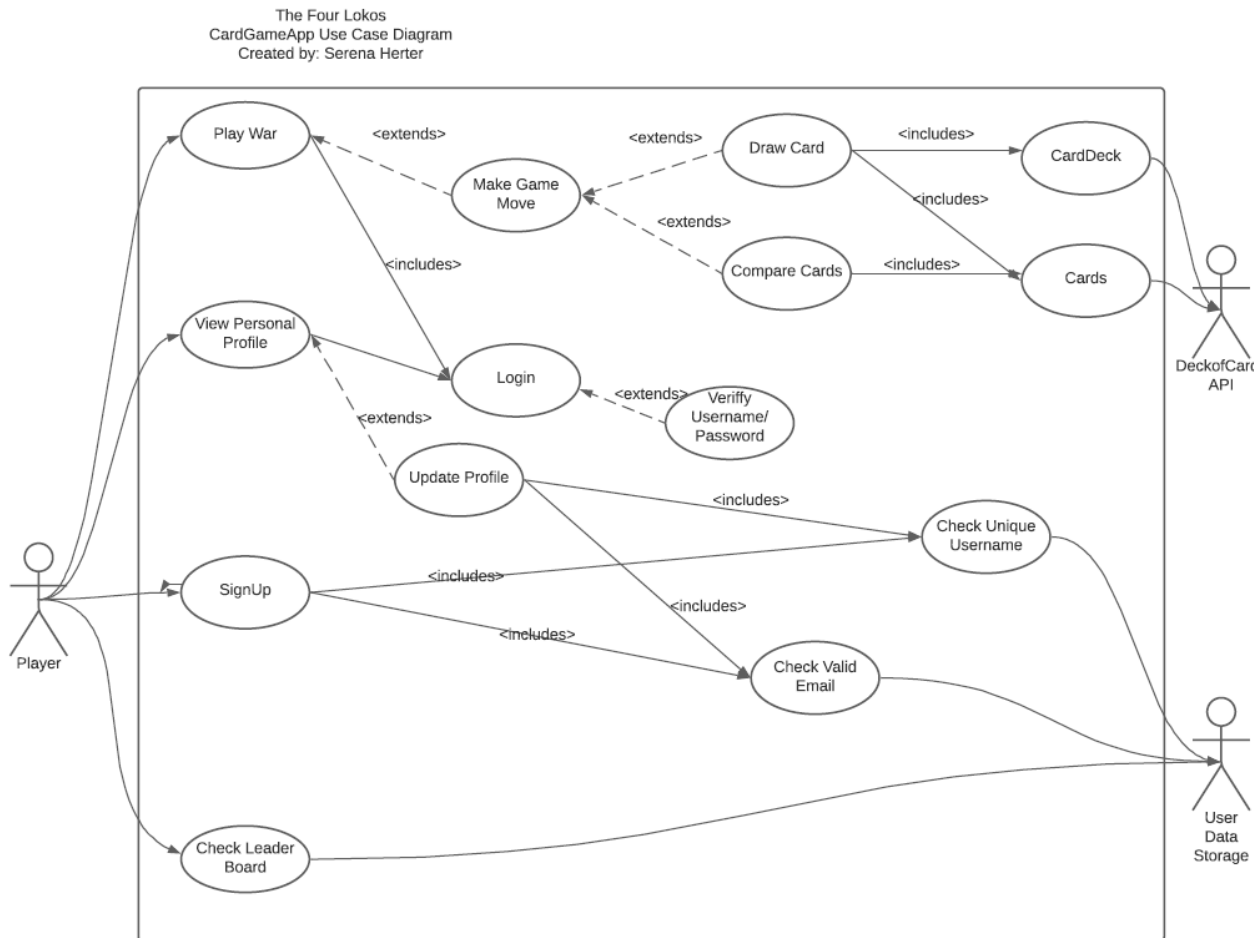
4.2.4. Communication Interface:

A simple text file was created so that it reads through the stored data. Also, the DeckOfCards API allows the back end of software and services to communicate with one another.

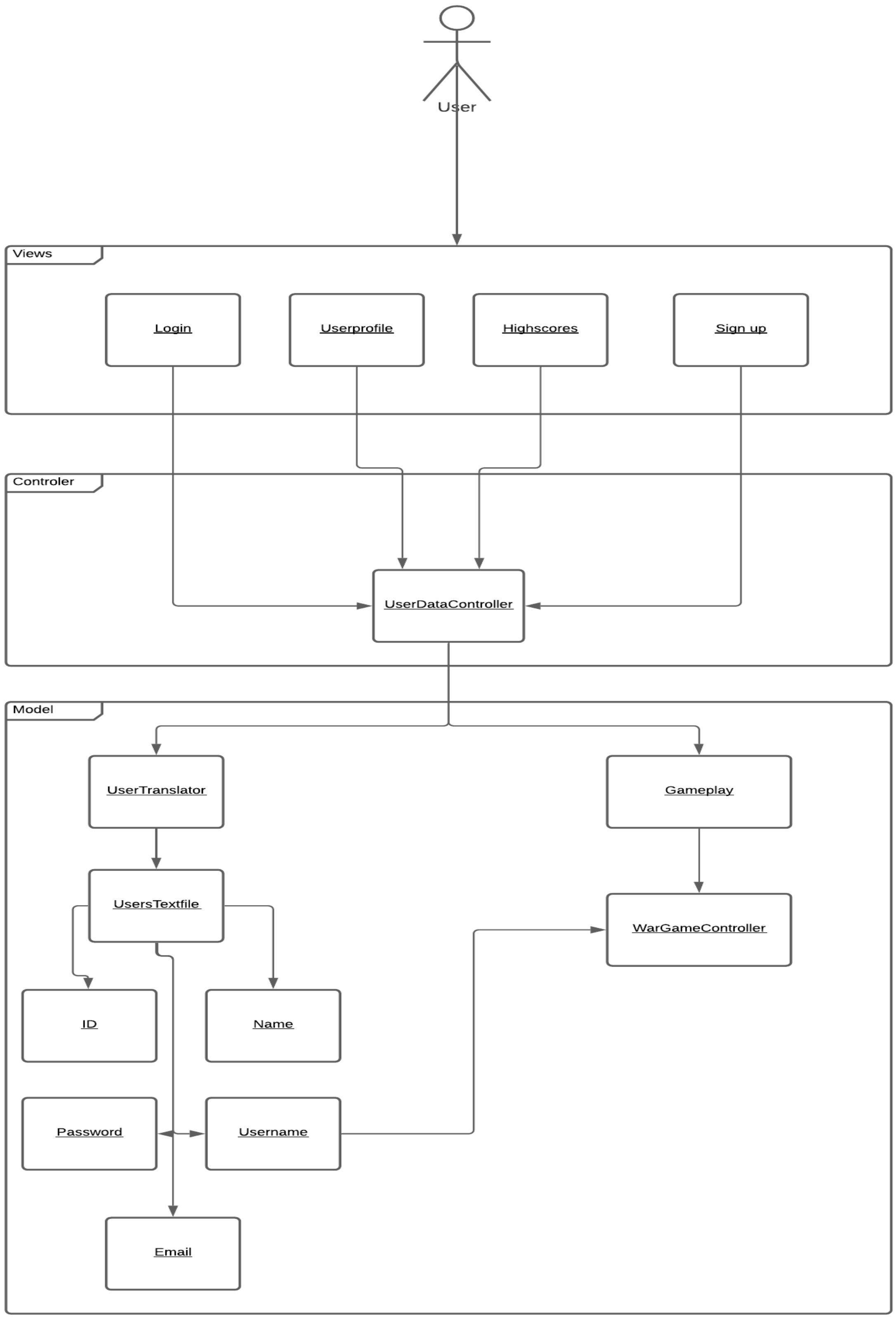
5. Non-functional Requirements:

5.1:Performance Requirements: The non-functional steps involved to create the implementation of this War Card Game application are as listed below.

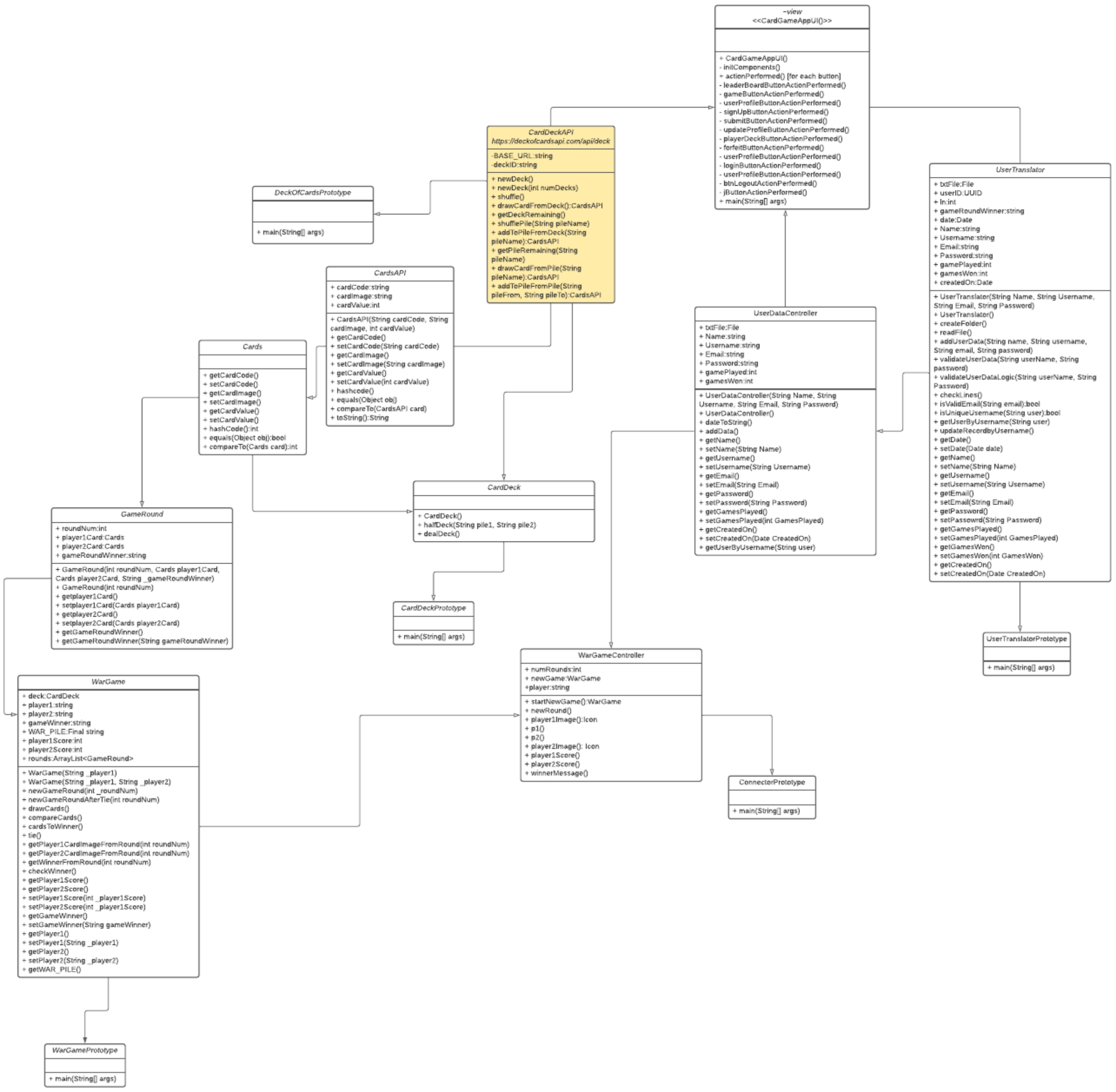
1. Use Case Diagram:



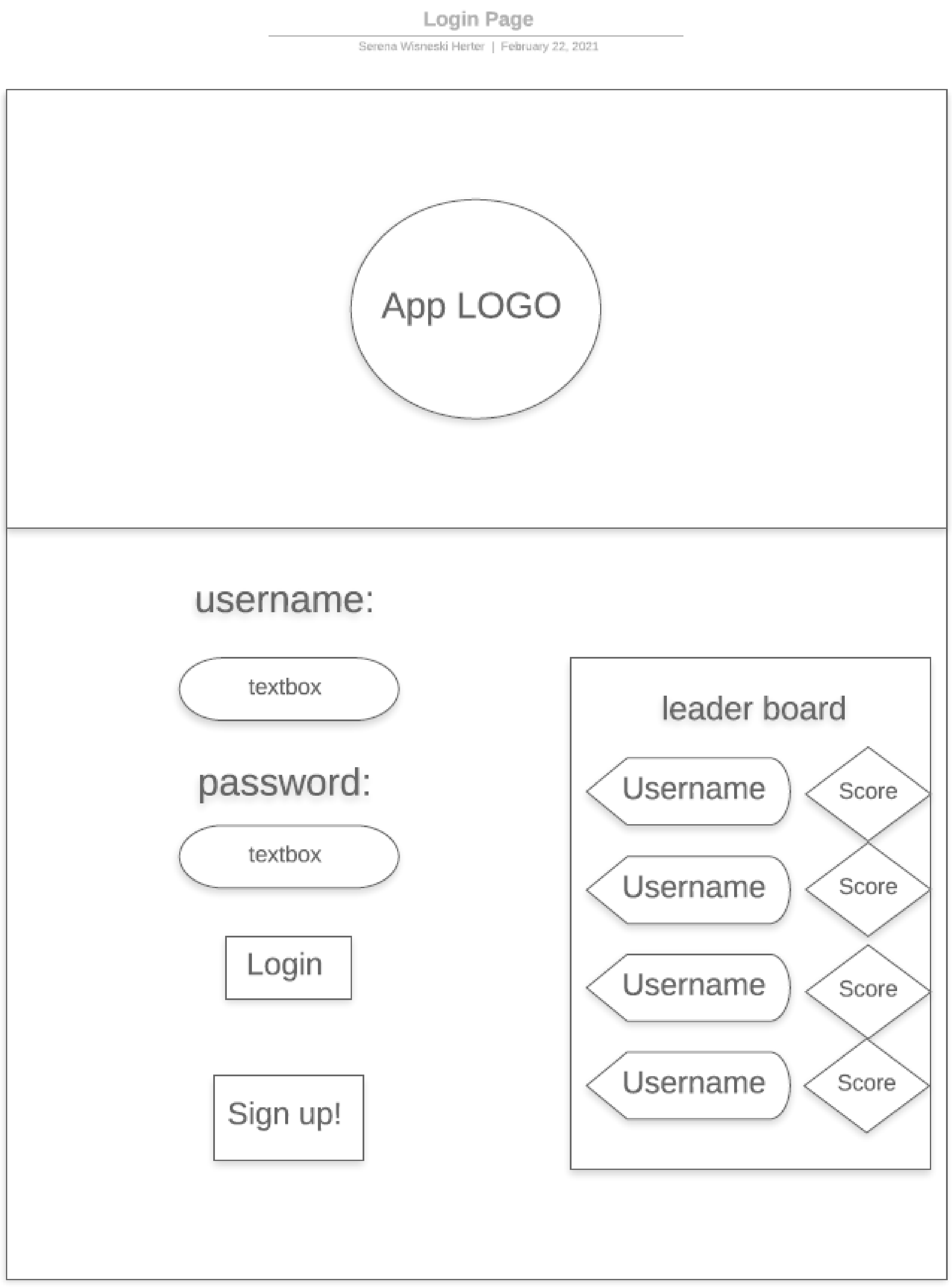
2. Architectural Diagram:[https://lucid.app/documents/edit/45a78744-2000-4f8b-bf26-3baa19914b2f/0?callback=close&name=docs&callback_type=back&v=1035&s=612]

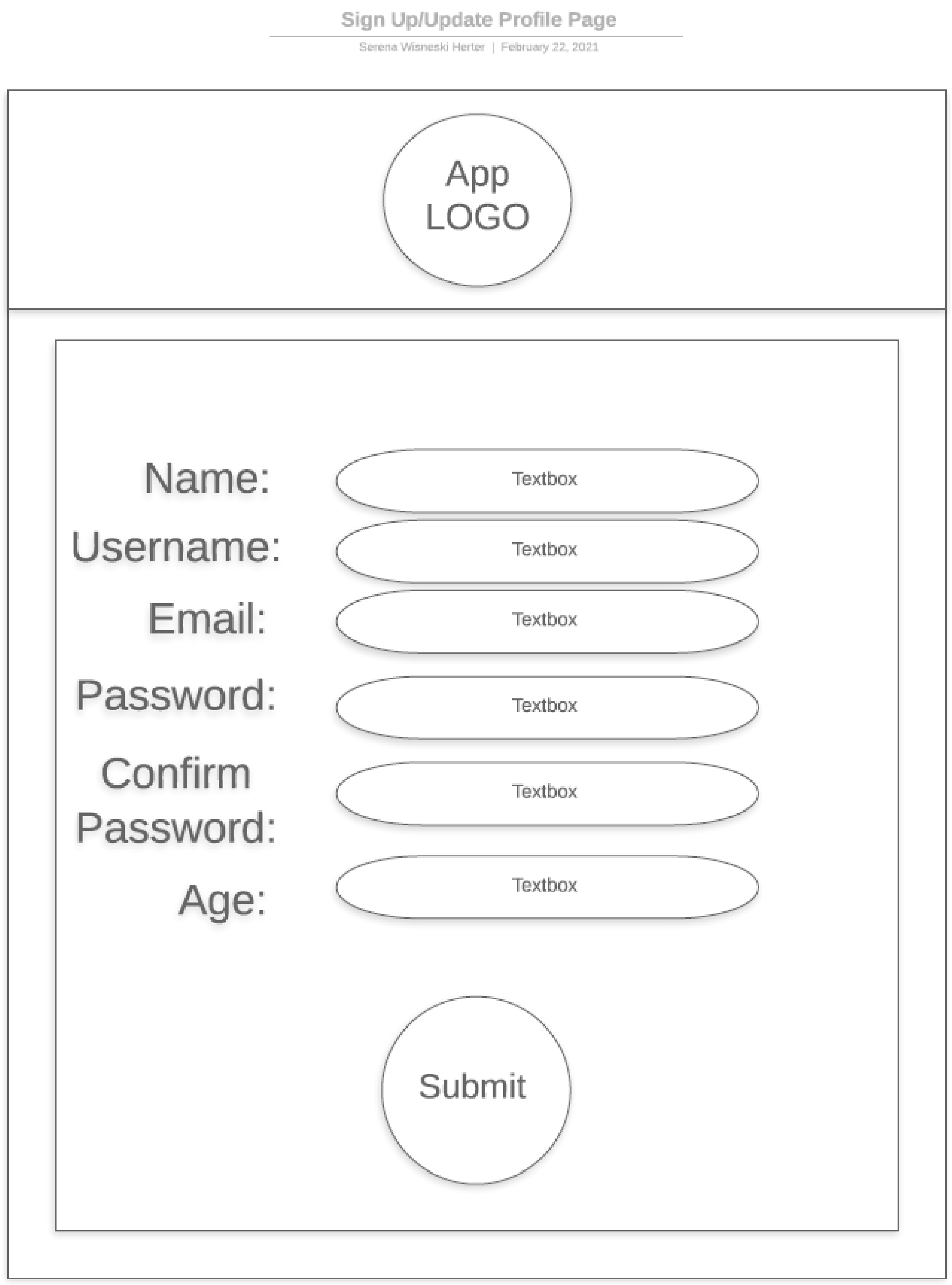


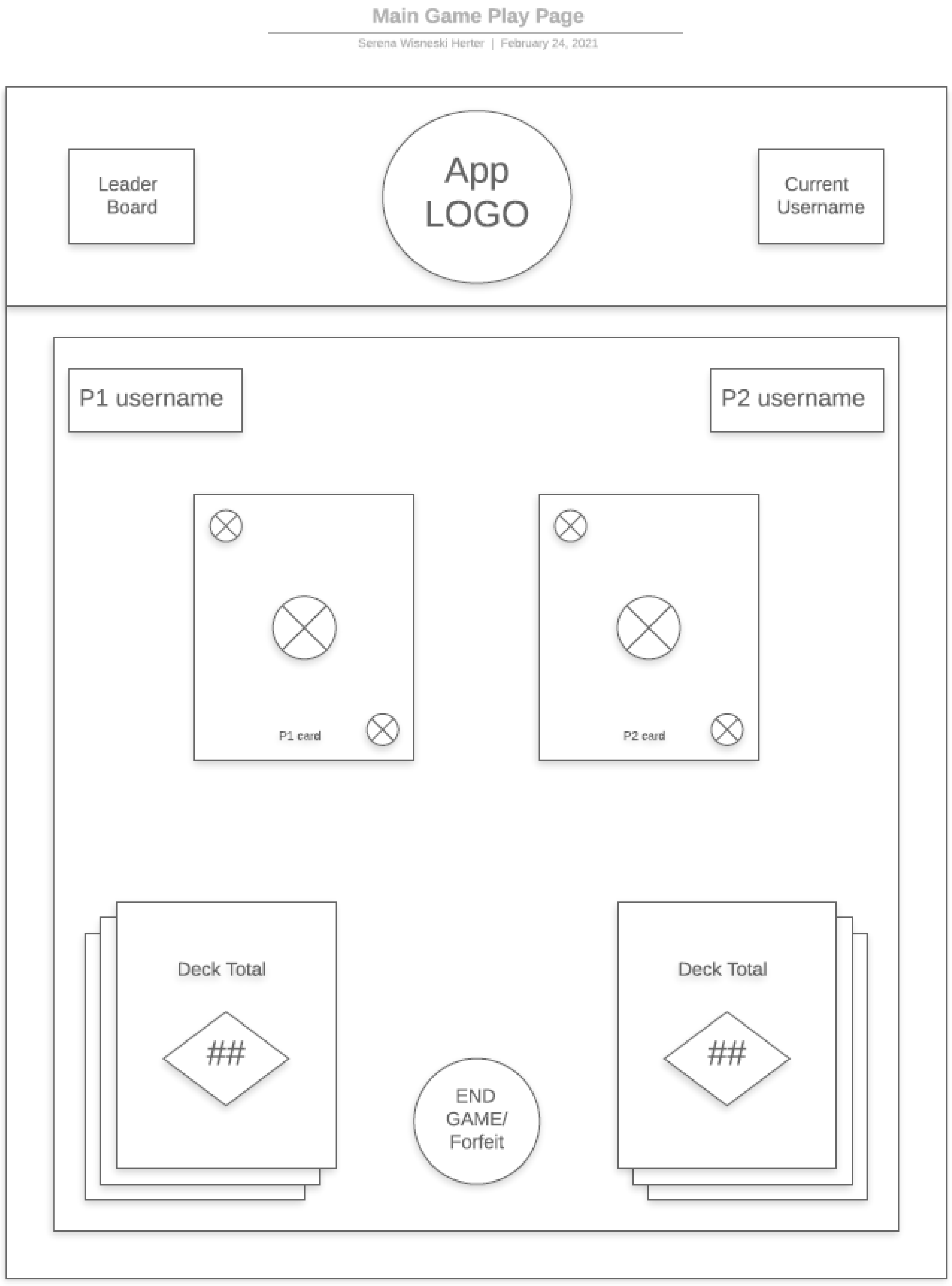
3. UML Class Diagram: [https://lucid.app/lucidchart/invitations/accept/inv_8abdc93d-224a-4e96-9b61-ca2e6abcb6fb]

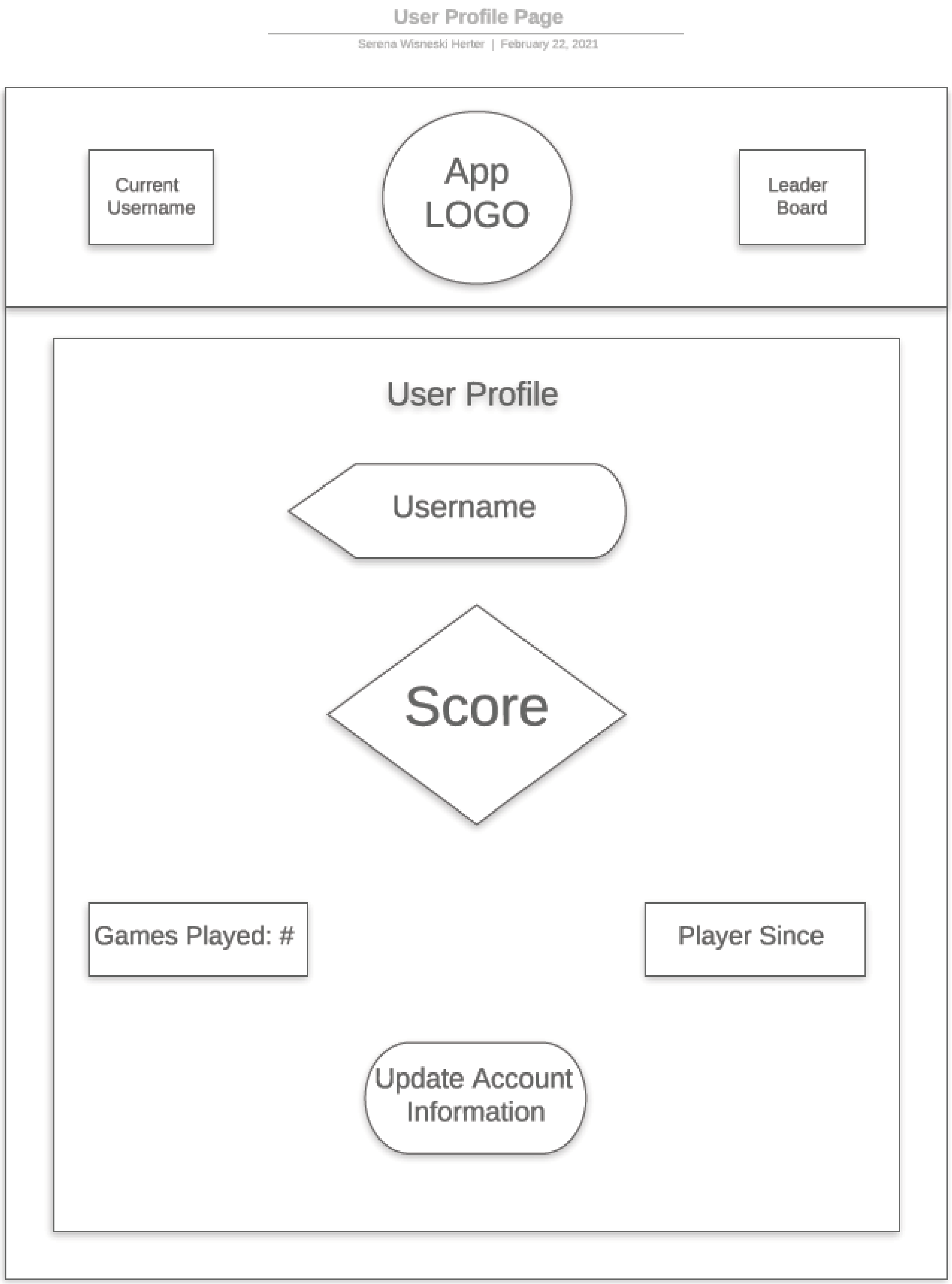

~view
<<CardGameAppUI()>>
+ CardGameAppUI()
- initComponents()
+ actionPerformed() [for each button]
- leaderBoardButtonActionPerformed()
- gameButtonActionPerformed()
- userProfileButtonActionPerformed()
- signUpButtonActionPerformed()
- submitButtonActionPerformed()
- updateProfileButtonActionPerformed()
- playerDeckButtonActionPerformed()
- forfeitButtonActionPerformed()
- userProfileButtonActionPerformed()
- loginButtonActionPerformed()
- userProfileButtonActionPerformed()
- btnLogoutActionPerformed()
- jButtonActionPerformed()
+ main(String[] args)
CardDeckAPI
https://deckofcardsapi.com/api/deck
-BASE_URL:string
-deckID:string
+ newDeck()
+ newDeck(int numDecks)
+ shuffle()
+ drawCardFromDeck():CardsAPI
+ getDeckRemaining()
+ shufflePile(String pileName)
+ addToPileFromDeck(String pileName):CardsAPI
+ getPileRemaining(String pileName)
+ drawCardFromPile(String pileName):CardsAPI
+ addToPileFromPile(String pileFrom, String pileTo):CardsAPI
DeckOfCardsPrototype
+ main(String[] args)
CardsAPI
+ cardCode:string
+ cardImage:string
+ cardValue:int
+ CardsAPI(String cardCode, String cardImage, int cardValue)
+ getCardCode()
+ setCardCode(String cardCode)
+ getCardImage()
+ setCardImage(String cardImage)
+ getCardValue()
+ setCardValue(int cardValue)
+ hashcode()
+ equals(Object obj)
+ compareTo(CardsAPI card)
+ toString():String
Cards
+ getCardCode()
+ setCardCode()
+ getCardImage()
+ setCardImage()
+ getCardValue()
+ setCardValue()
+ hashCode():int
+ equals(Object obj):bool
+ compareTo(Cards card):int
UserTranslator
+ txtFile:File
+ userID:UUID
+ ln:int
+ gameRoundWinner:string
+ date:Date
+ Name:string
+ Username:string
+ Email:string
+ Password:string
+ gamePlayed:int
+ gamesWon:int
+ createdOn:Date
+ UserTranslator(String Name, String Username, String Email, String Password)
+ UserTranslator()
+ createFolder()
+ readFile()
+ addUserData(String name, String username, String email, String password)
+ validateUserData(String userName, String password)
+ validateUserDataLogic(String userName, String Password)
+ checkLines()
+ isValidEmail(String email):bool
+ isUniqueUsername(String user):bool
+ getUserByUsername(String user)
+ updateRecordbyUsername()
+ getDate()
+ setDate(Date date)
+ getName()
+ setName(String Name)
+ getUsername()
+ setUsername(String Username)
+ getEmail()
+ setEmail(String Email)
+ getPassword()
+ setPassowrd(String Password)
+ getGamesPlayed()
+ setGamesPlayed(int GamesPlayed)
+ getGamesWon()
+ setGamesWon(int GamesWon)
+ getCreatedOn()
+ setCreatedOn(Date CreatedOn)
UserDataController
+ txtFile:File
+ Name:string
+ Username:string
+ Email:string
+ Password:string
+ gamePlayed:int
+ gamesWon:int
+ UserDataController(String Name, String Username, String Email, String Password)
+ UserDataController()
+ dateToString()
+ addData()
+ getName()
+ setName(String Name)
+ getUsername()
+ setUsername(String Username)
+ getEmail()
+ setEmail(String Email)
+ getPassword()
+ setPassword(String Password)
+ getGamesPlayed()
+ setGamesPlayed(int GamesPlayed)
+ getCreatedOn()
+ setCreatedOn(Date CreatedOn)
+ getUserByUsername(String user)
CardDeck
+ CardDeck()
+ halfDeck(String pile1, String pile2)
+ dealDeck()
GameRound
+ roundNum:int
+ player1Card:Cards
+ player2Card:Cards
+ gameRoundWinner:string
+ GameRound(int roundNum, Cards player1Card, Cards player2Card, String _gameRoundWinner)
+ GameRound(int roundNum)
+ getplayer1Card()
+ setplayer1Card(Cards player1Card)
+ getplayer2Card()
+ setplayer2Card(Cards player2Card)
+ getGameRoundWinner()
+ getGameRoundWinner(String gameRoundWinner)
CardDeckPrototype
+ main(String[] args)
UserTranslatorPrototype
+ main(String[] args)
WarGame
+ deck:CardDeck
+ player1:string
+ player2:string
+ gameWinner:string
+ WAR_PILE:Final string
+ player1Score:int
+ player2Score:int
+ rounds:ArrayList<GameRound>
+ WarGame(String _player1)
+ WarGame(String _player1, String _player2)
+ newGameRound(int _roundNum)
+ newGameRoundAfterTie(int roundNum)
+ drawCards()
+ compareCards()
+ cardsToWinner()
+ tie()
+ getPlayer1CardImageFromRound(int roundNum)
+ getPlayer2CardImageFromRound(int roundNum)
+ getWinnerFromRound(int roundNum)
+ checkWinner()
+ getPlayer1Score()
+ getPlayer2Score()
+ setPlayer1Score(int _player1Score)
+ setPlayer2Score(int _player1Score)
+ getGameWinner()
+ setGameWinner(String gameWinner)
+ getPlayer1()
+ setPlayer1(String _player1)
+ getPlayer2()
+ setPlayer2(String _player2)
+ getWAR_PILE()
WarGameController
+ numRounds:int
+ newGame:WarGame
+player:string
+ startNewGame():WarGame
+ newRound()
+ player1Image():Icon
+ p1()
+ p2()
+ player2Image(): Icon
+ player1Score()
+ player2Score()
+ winnerMessage()
ConnectorPrototype
+ main(String[] args)
WarGamePrototype
+ main(String[] args)

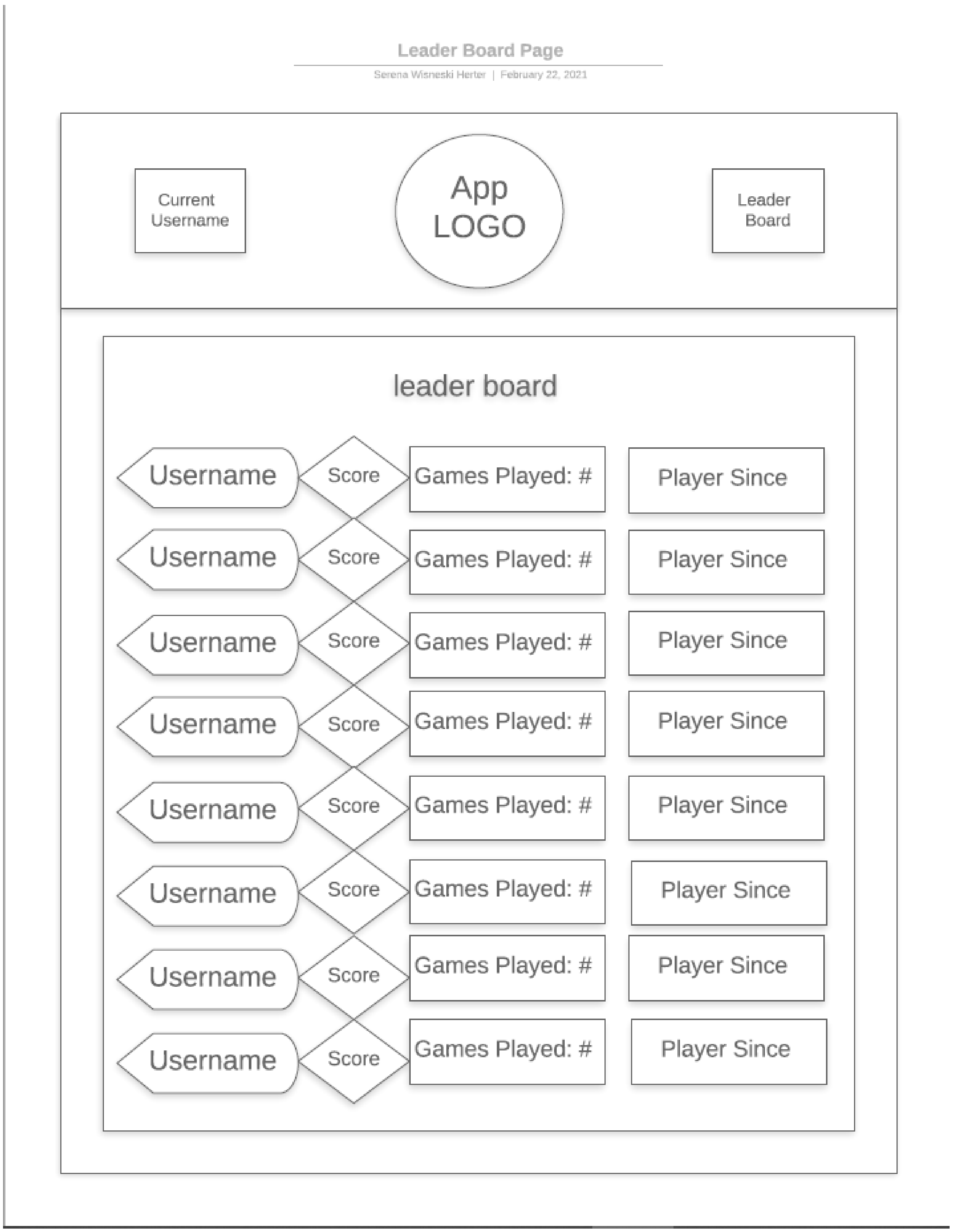
4. Wireframe Diagram:




Sign Up/Update Profile Page
Serena Wisneski Herter | February 22, 2021
App LOGO
Name:
Textbox
Username:
Textbox
Email:
Textbox
Password:
Textbox
Confirm Password:
Textbox
Age:
Textbox
Submit


Main Game Play Page
Serena Wisneski Herter | February 24, 2021
Leader Board
App LOGO
Current Username
P1 username
P2 username
P1 card
P2 card
Deck Total
##
END GAME/ Forfeit
Deck Total
##


User Profile Page
Serena Wisneski Herter | February 22, 2021
Current Username
App LOGO
Leader Board
User Profile
Username
Score
Games Played: #
Player Since
Update Account Information


Leader Board Page
Serena Wisneski Herter | February 22, 2021
Current Username
App LOGO
Leader Board
leader board
Username
Score
Games Played: #
Player Since

5.2: Safety/Recovery Requirements:

All user information and preferences are stored and dynamically updated as the user interacts with the application. There are no “forgot user/password” functions as this app is free and a user can make several accounts with a single email address.

5.3: Security Requirements:

The API is secured for all operating systems and since the text file handles the role of a database, it is more secure than the implementation of an external database.

5.4: Policy Requirements:

We have followed the universities policy requirements and we have used open-source, secured, and free API in our application.

5.5: Software Quality Attributes:

5.5.1. Availability:

Once downloaded, one user may begin the application and play till either they or the computer wins the WAR match.

5.5.2. Correctness:

Once the application runs, it should allow easy login and play the game till either the user (player) or the computer (another player) wins or ties the match.

5.5.3. Maintainability:

The admins should be able to make sure that the game must run properly for anyone who downloads the application.

5.5.4. Reusability:

The packages in the CardGame application can be used in other Card games and when needed can be further updated.

5.5.5. Portability:

Has been pushed on GitHub so it allows easy .zip download of the application and can be used on any operating system that has JAVA Netbeans IDE.

5.6: Process Requirements:

5.6.1. Development Process Used:

- JAVA Netbeans IDE
- GitHub
- DeckOfCard API
- Manual testing
- External JSON library
- Lucid Charts

5.6.2. Time Constraints:

By the end of semester: Spring 2021.

5.6.3. Cost and Delivery Date:

Free and is expected to be due on April 30, 2021.