**Software Design Specification**

**for**

Social Engineering Training Game

**Version 1.0**

**Prepared by: Frederic Stresau, Simon Chummar, Alexander C. Bodoh,   
Quy Pham, Jackson Harper**

**Florida Polytechnic University   
Secure Software Engineering Group 7**

**4/24/2024**

**Table of Contents**

[**1. Introduction 1**](#_30j0zll)

[1.1 Purpose 1](#_1fob9te)

[1.2 System Overview 1](#_3znysh7)

[**2. Design Considerations 1**](#_2et92p0)

[2.1 Assumptions 1](#)

[2.2 General Constraints 1](#)

[**3. System Architecture 2**](#_1t3h5sf)

[4. Detailed System Design 3](#)

[4.1 Class Diagram 3](#)

[4.2 Activity Diagram 6](#)

[4.3 Sequence Diagrams 6](#)

[4.3.1 Scenario 1: Account Creation 6](#_pzejzww9oss8)

[4.3.2 Scenario 2: Log in to Existing Account 7](#_lysevgxd5lv)

[4.3.3 Scenario 3: Retrieve User’s Score History after Logging in 8](#_q18qicqcp6zz)

[4.3.4 Scenario 4: Example of a Single Game Round 8](#_juwp5v6o1vna)

[4.4 State Chart Diagram 10](#)

[4.5 Algorithms for Components/Methods 10](#)

[4.6 Database Design 13](#)

[**5. User Interface Design 13**](#_2jxsxqh)

[Common Behaviors: 13](#_q17vvhl9hovk)

# **Introduction**

## **Purpose**

The software requirements specified in this document pertain to the development of a Social Engineering training software in the form of a fun entertaining game. The purpose of this software is to enhance users' awareness of social engineering attacks in an engaging and entertaining format. By simulating various social engineering scenarios and providing feedback on user responses, the software aims to educate users about different types of social engineering attacks and basic protection techniques.

## **System Overview**

The system will perform a Software Engineering quiz game where the user can learn about social engineering tactics and prevention. The system will include a user interface so the user can interact with the game in an intuitive manner. User authentication will be included, as well as secure hashing of the user’s password and encryption of the user’s past score data. The game itself will consist of 5 questions, whose content relating to question, answer and feedback are loaded at random from an on-device question bank JSON file before being displayed to the user.

# **Design Considerations**

## **Assumptions**

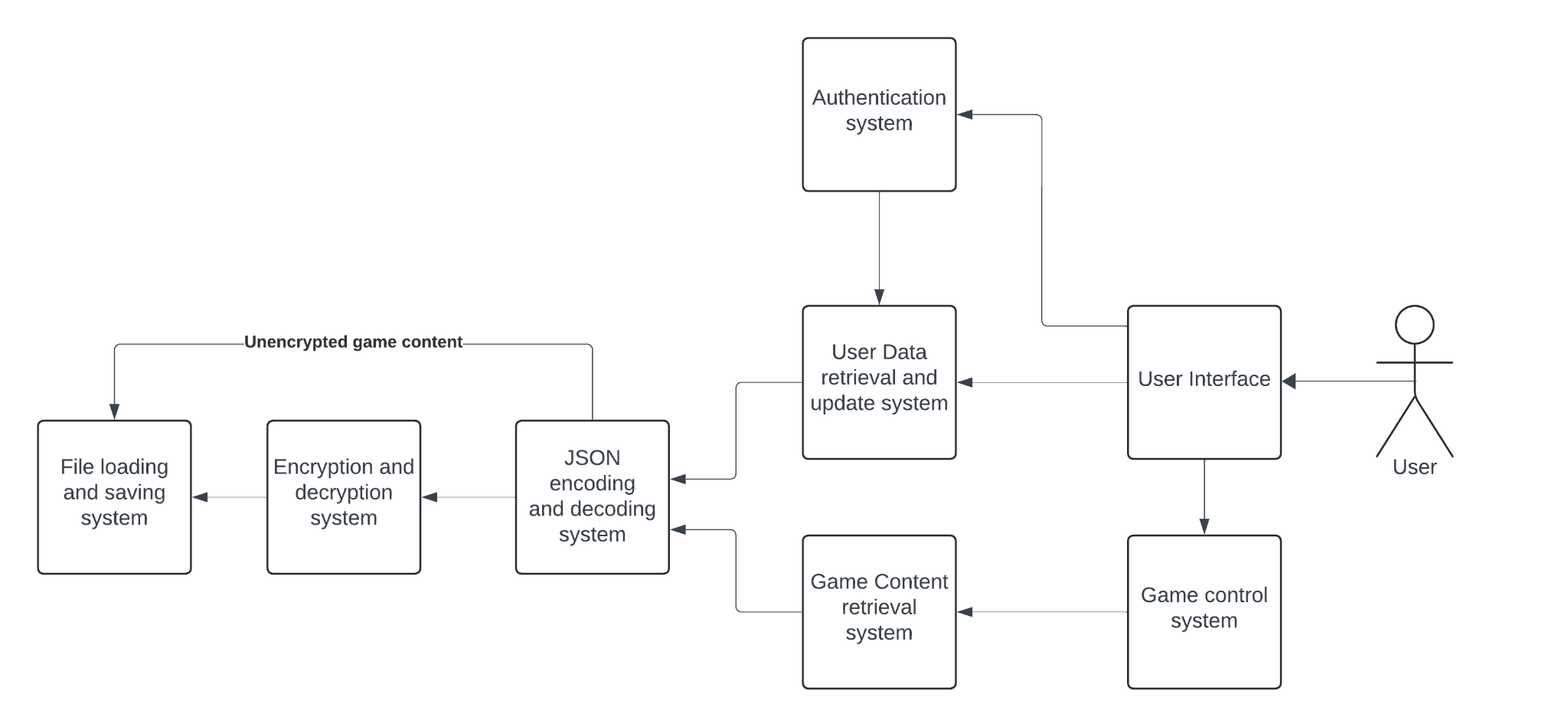
* The software is designed for use on the Android OS platform.
* The software will be played and tested on an Android OS phone.
* Users are expected to have a basic understanding of touchscreen and cellphone technology.
* Possible changes may include altering the number of questions presented to the user.
* The availability and reliability of internet connectivity for account creation and accessing online resources.
* Users will provide accurate information during account creation and gameplay.
* The software will be developed using standard development tools and frameworks compatible with Android platforms.

## **General Constraints**

* Our chosen environment of smartphone hardware running the Android operating system constrains the design of the user interface and gameplay mechanics.
  + The small touchscreen interface of a smartphone requires a user interface that has large, easily tappable buttons as well as clearly readable text in a limited space.
  + Furthermore, focusing on the Android operating system significantly constrains the libraries and programming languages available.
  + Focusing on Android allows the application to be tested in the Android Studio device emulator, which significantly increases the number of (simulated) hardware devices on which the implementation can be tested.
* Security constraints:
  + Since the system must hash the user’s password to determine if their login credentials are valid, a secure hashing algorithm that has not been cracked must be used. Thus, for instance, this constraint eliminates the possibility of using the MD5 algorithm which has been cracked.
  + Since the user’s past history is private, it must be encrypted with a secure, non-cracked encryption algorithm. The key used for the encryption should be secure against reverse-engineering.
* Testing constraints: Since verification and validation will be performed primarily using the Android Studio device emulator, the computers used for testing must have adequate hardware to realistically emulate the virtual devices.

# **System Architecture**

This section presents the high level system architecture of the application, with descriptions of each of the components presented in the diagram.



The Authentication system allows the user to create a new account given a username and password as well as to log in to an existing account given a valid username and password combination. The Authentication system also ensures that the user's desired password and username name match the requirements. Once logged in, the Authentication system provides credentials to access the user's password-protected data.

The User Data retrieval and update system is responsible for taking the authentication parameters from the Authentication system, and appropriately instructing the JSON encoding and decoding system to load the user's previous score history. It also needs to instruct the JSON Encoding and decoding system to save new scores to the user's history when instructed to do so by the user interface.

The user interface allows the user to interact with the application, and forwards appropriate information or commands to the Authentication system, User Data retrieval and update system, and the Game control system. The user interface is also responsible for taking appropriate information from these subsystems and displaying it to the user so they can make a decision, view their previous history, and so on.

The Game control system is responsible for coordinating a single round of five distinct questions about social engineering presented to the user. This system will, in sequence, display each of the five questions as returned by the Game content retrieval system, as well as the answers and then later feedback for the user to see in the user interface. Finally, once the game is finished, the Game control system will return the user's score for that round so it can be displayed in the user interface.

The File Loading and Saving System manages the JSON and encrypted data, ensuring smooth loading and saving of game data.

The Encryption System secures the sensitive user data.

The JSON Handling System either sends or gets data from the encrypt/decrypt system, or directly from the file loading/saving system.

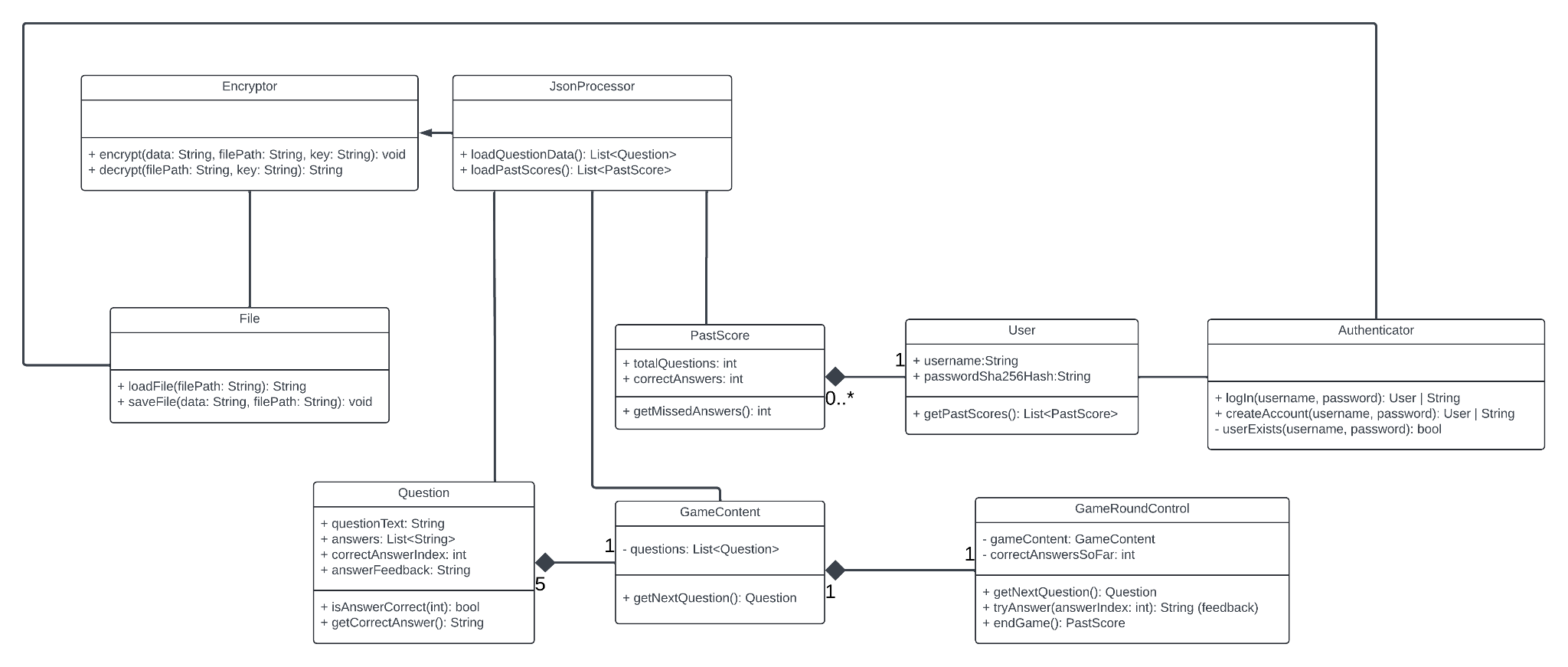
The Game Content Retrieval System fetches the game content—questions, answers, and feedback—from the JSON encode/decode system.

# **Detailed System Design**

This section provides our detailed system design for the software engineering training game, including a discussion of the classes involved, the games activities, sequences of actions in nontrivial scenarios, state transitions for the system overall, and finally pseudocode for some of the non-trivial algorithms in the game.

## **Class Diagram**

*A detailed class diagram is developed to depict the relationships between classes as well as the attributes and operations of each class. Provide detailed description as well.*



In the File class there are two methods: loadFile() and SaveFile(). The loadFile method is used to read and return the contents inside a file. This will be useful for loading the user login data. This method will use the parameter ‘filePath’ to get the location of the file. Then, it will open the file and return the content inside the file. The saveFile method is used for saving data into a file at the specified file location. This will be used for saving user data such as username and the hashed password.

In the Encryptor class there are two methods: encrypt() and decrypt(). The encrypt method is used for encrypting data using a key. The encrypted data will then be sent to a specified file path. The decrypt method will be used for decrypting data from a specified file path. The method will first read the encrypted data from the file. It will then use the provided key to decrypt the encrypted data. The decrypted data will then be returned as the original plaintext as a string.

The JSONProcessor class is used for handling data that is in JSON format. This class has the methods loadQuestionData() and loadPastScores(). The loadQuestionData method is used for loading and parsing the data which contains the questions, possible answers, and the correct answer. The loadPastScores method is used for loading the users score from the previous match that the user played. When the data from these methods are loaded they will be parsed so the data can be used correctly.

The user class has the attributes username and passwordSha256Hash. The username attribute will store the user’s chosen username. The username will be a part of the user’s login process. The attribute passwordSha256Hash will contain the user's password hashed using SHA-256. The getPastScores() method is used for retrieving the scores of past games the user has played.

The Authenticator class is used for creating and verifying user’s accounts. The authenticator class has the methods login(), createAccount(), and userExists(). The login method will take in the parameter’s username and password. The createAccount() method will allow the user to create an account using a unique username and password which must satisfy the username and password requirements. The userExists() method will check if the username already exists when a user is creating an account and it will also check if the user exists when a user is trying to log in.

In the class Question, we have 4 attributes: questionsText, answers, correctAnswerIndex, and answerFeedback. The questionText attribute is a String text that is a question asked in the game. The answers attribute is a list of answers for the questions asked. The correctAnswerIndex attribute tells how many questions are correct from the answer list. The answerFeedback attribute gives feedback to the user after answering a question. The two operations in the question class are isAnswerCorrect, and getCorrectAnswer. The isAnswerCorrect operation is a Boolean that sends a true or false if the answer is correct. The getCorrectAnswer operation returns the correct answers.

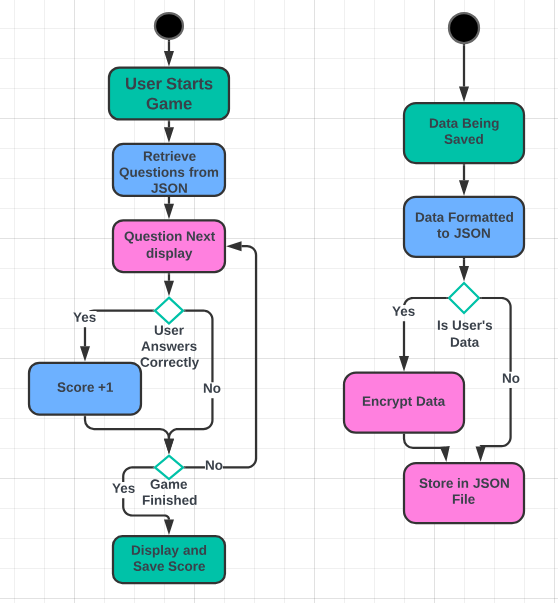
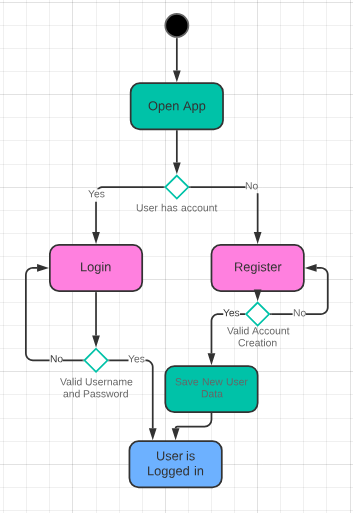
In the class GameContent, we have an attribute question: List<Question> that is a list of Question objects that form the content of the game. We also have a getNextQuestion operation that gets a new question from the game content, this question is randomly picked from the question list.

In the class PastScores, we have two attributes which are TotalQuestions and CorrectAnswers. The TotalQuestion attribute is an int that tells the number of questions attempted from the previous games. The CorrectAnswers attribute is an int that tells the number of correct answers from the previous games. The pastScore class has one operation, getMissedQuestions which calculates and returns the missed questions from the previous games.

In the class GameRoundControl, we have two attributes which are gameContent and correctAnswersSoFar. The gameContent attribute contains the content and questions in the game. The correctAnswerSoFar attribute shows the correct answers the user has in the current game they are playing. The GameRoundControl has three operations, getNextQuestion, tryAnswer, and endGame. The getNextQuestion operation gets the next question from the gameContent. The tryAnswer operation allows the user to attempt to answer the current question, if the user gets it correct, it increments the correctAnswerSoFar attribute. The last operation is the endGame, which ends the game and generates the PastScore.

## **Activity Diagram**

*#1 #2 #3*

**

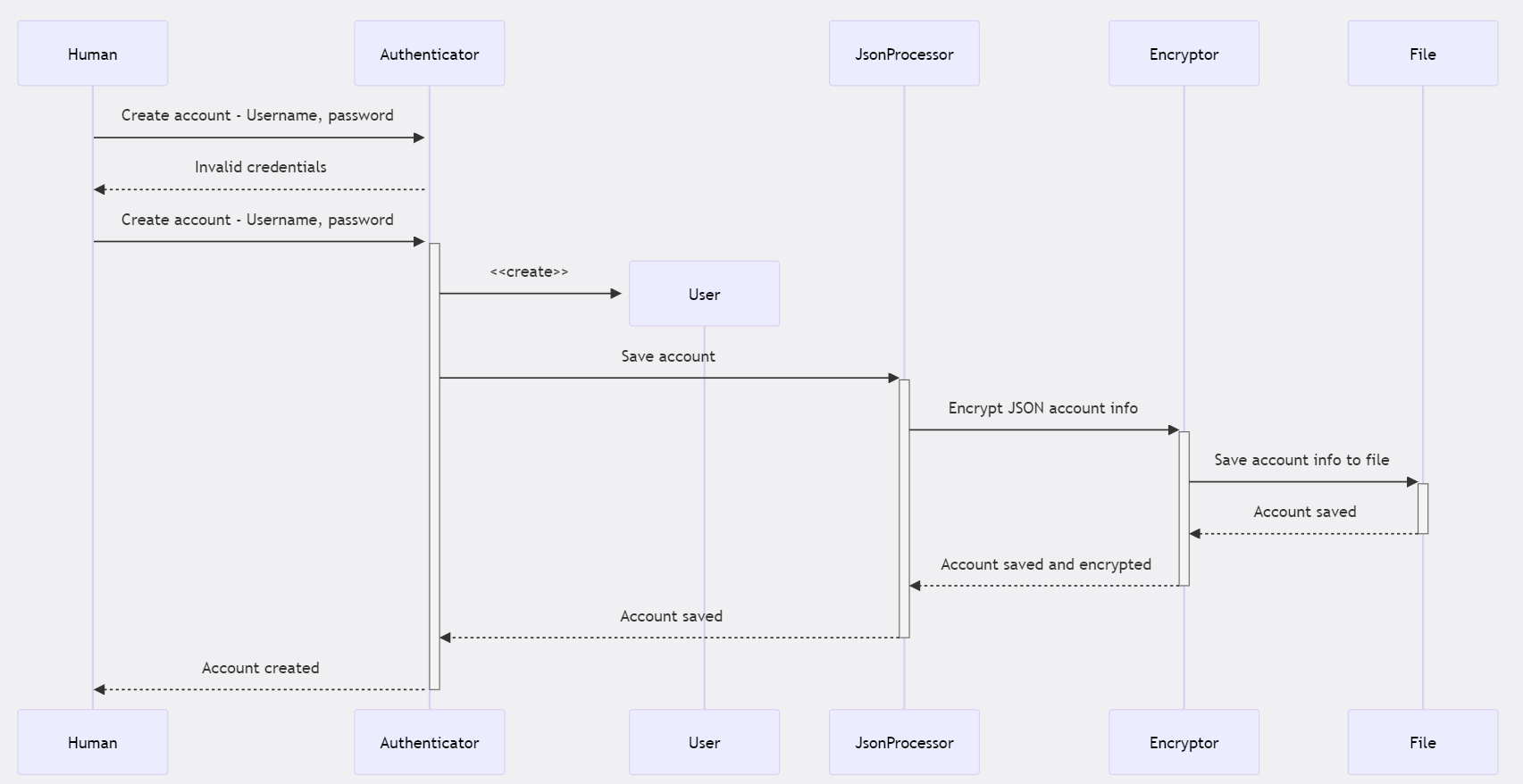
1. This is the Activity Diagram for when the user first opens the app. They start at the login screen where they’ll either need to log in or register. If the user registers, their data will be encrypted and saved. Finally, both of these options will log the user in.
2. This Activity Diagram starts when the game is started. It gets the questions needed from the JSON file and then displays the first one. If the user answers correctly a point is added to their score. If the game is finished display and save the score, otherwise loop up to display the next question.
3. This Activity Diagram starts when data is being saved. It’s first formatted to go into a JSON file, then if it’s the user’s login data it’s encrypted and stored. Otherwise, the data is stored directly in the JSON file.

## **Sequence Diagrams**

This section presents the sequence diagrams for some of the more complex scenarios in the application.

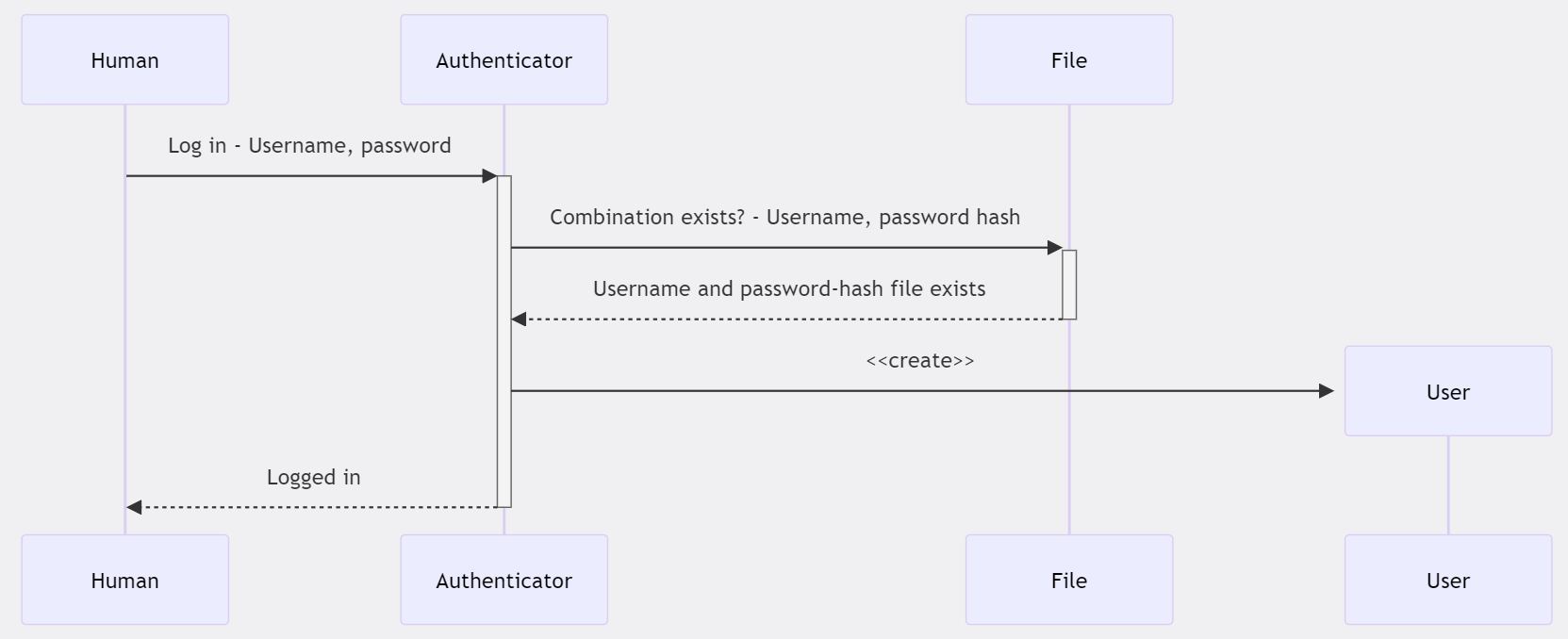
### 4.3.1 Scenario 1: Account Creation

In this scenario, the user attempts to create an account, first with invalid credentials, and then with valid credentials. When the user supplies valid credentials, the authenticator class coordinates actions with the other classes in order to save the user's new account information, and then inform the user that their account has been created.



### 4.3.2 Scenario 2: Log in to Existing Account

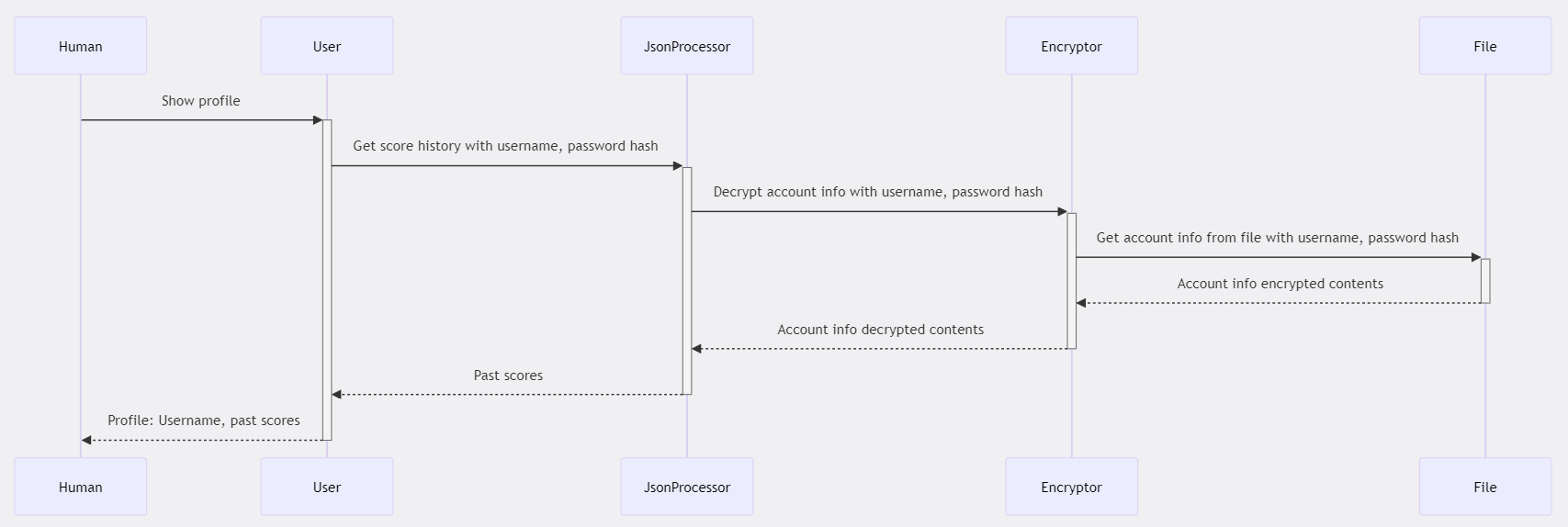
Here, the user attempts to log in with an existing username and password. In this case, the Authenticator class communicates with the File class to determine that the user's combination of username and password hash does exist. Then, the Authenticator class creates an instance of the User class to store the user's credentials for later use in the application, and then informs the human user that their login was successful.



### 

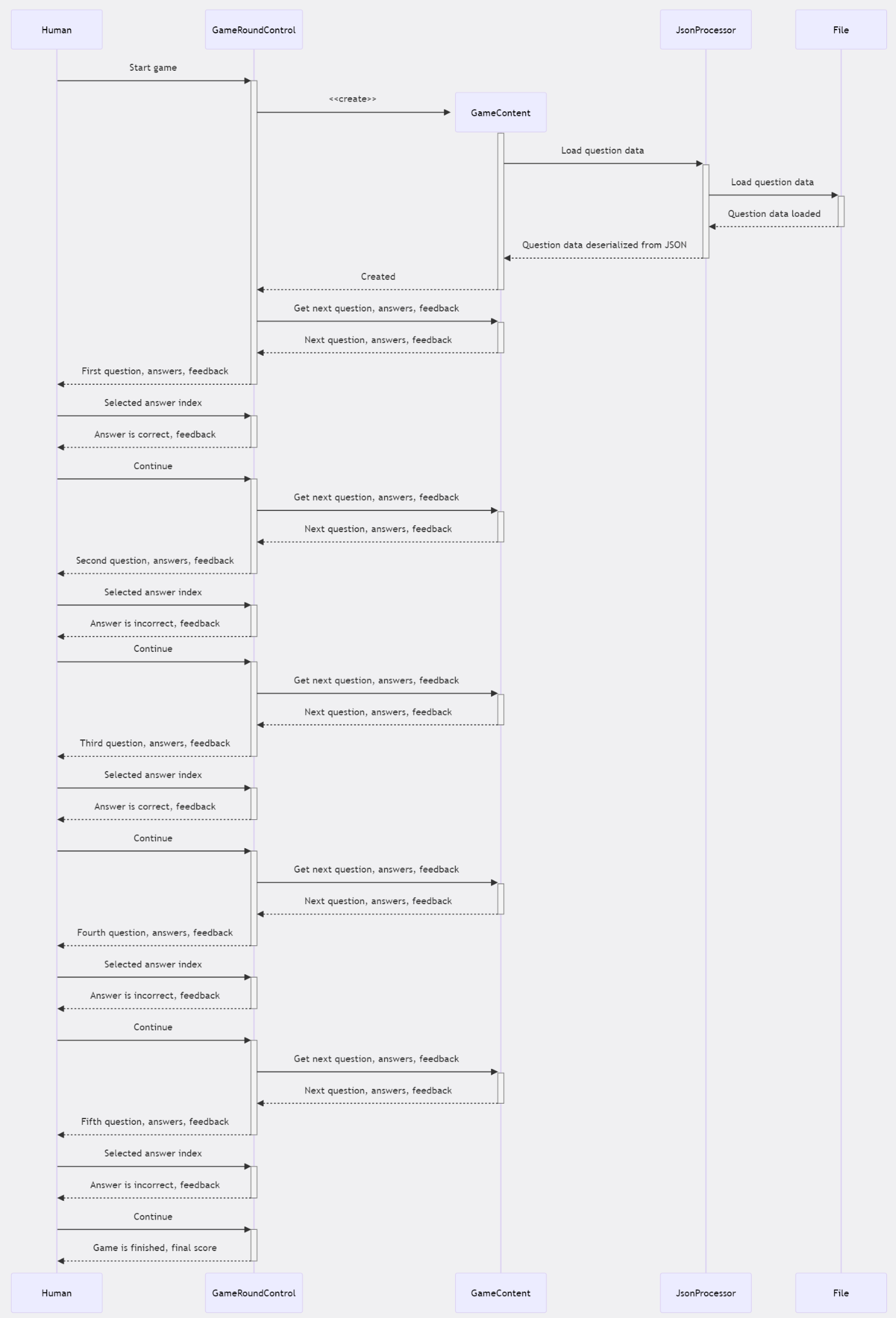
### 4.3.3 Scenario 3: Retrieve User’s Score History after Logging in

In this scenario, the human instructs the User class to retrieve the profile information. When this happens, the User class requests the score history from the JsonProcessor, which in turn requests the decrypted file contents from the Encryptor, which itself asks the File class for the actual encrypted file contents, corresponding to the users’ username and password hash combination.



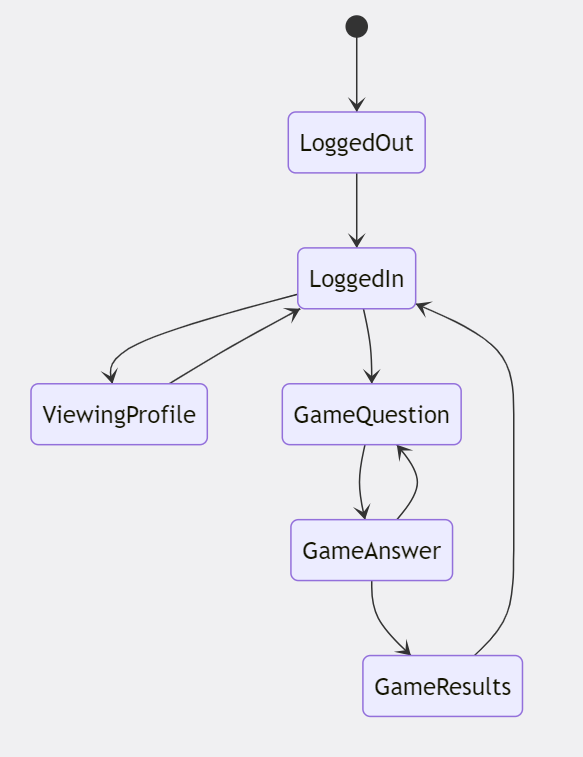
### 4.3.4 Scenario 4: Example of a Single Game Round

In this example, the human instructs the GameRoundControl class to start a game. The GameRoundControl class creates a new GameContent object, which loads unencrypted question data with the help of the JsonProcessor and File classes. Once the game content has been retrieved, the GameRoundControl coordinates the game with the user. The user inputs their choices for answers and then sees what they got correct and what they got wrong. Finally, once the round is finished, the GameRoundControl object presents the user’s final score.



## **State Chart Diagram**

Here, the state chart for the overall system is displayed.The game starts in the LoggedOut state, where the user needs to provide valid credentials or valid new account information in order to be logged in. From the LoggedIn state, the user can either view their, or start a game which brings them into the two game states, GameQuestion and GameAnswer. Once the user has exhausted all of the five questions in the game, they are brought to the GameResults stage where they can see their score, and then they are brought back to the home screen where they can begin again, or view their profile as before.



## **Algorithms for Components/Methods**

**Authenticator.login**

Check if the username and password credentials are empty

If username and or password are empty then return error

Return “username or password is incorrect”

Check if the user exists

If user does not exist return error

If user exists compare password with the stored hash value

compareHash()

If hash does not match return error

If the hash matches the user will successfully log in

**Authenticator. createAccount**

Check if the username and password fields are empty

If empty

Return “Please enter a username and password”

Check if the password meets the requirements for a strong password

The username must be alphanumeric, including underscores and hyphens, start with a letter, and contain 1-12 characters.

If !isStrongPassword()

Return “Password does not meet the strength requirements”

Check if user already exists

If user already exists then return “User already exists”

If user does not already exist

HashPassword()

Save user information in database

**GameRoundControl Constructor**

Create Attributes

gameContent

correctAnswersSoFar

Constructor(gameContent) = gameContent

this.gameContent = gameContent

this.correctAnswersSoFar = 0

**GameRoundControl.tryAnswer**

Retrieve next question data from gameContent //such as question, possible answers, and correct answer

ansiwerIsCorrect = nextQuestion.isAnswerCorrect(answer)

If the user is incorrect return “Incorrect”, correct answer, feedback

If the user is correct:

this.correctAnswersSoFar++

return “Correct”, correct answer, feedback

**Function JsonProcessor.LoadPastScores()**

scoresEncrypted = Call File.loadFile(path to past scores)

scoresDecrypted = Call Encryptor.decrypt(scoresEncrypted)

pastScores = Parse Json PastScore data;

If Json is parsed

return pastScores;

Else

print Error;

return NULL;

**Function JsonProcessor.LoadQuestions()**

questionFileContents = Call File.loadFile(path to past scores)

questions = Parse Json question data;

If Json is parsed

return questions;

Else

print Error;

return NULL;

**GameContent constructor()**

questionBank = JsonProcessor.LoadQuestions();

this.questions = choose 5 random questions from questionBank

## **Database Design**

Not applicable.

# **User Interface Design**

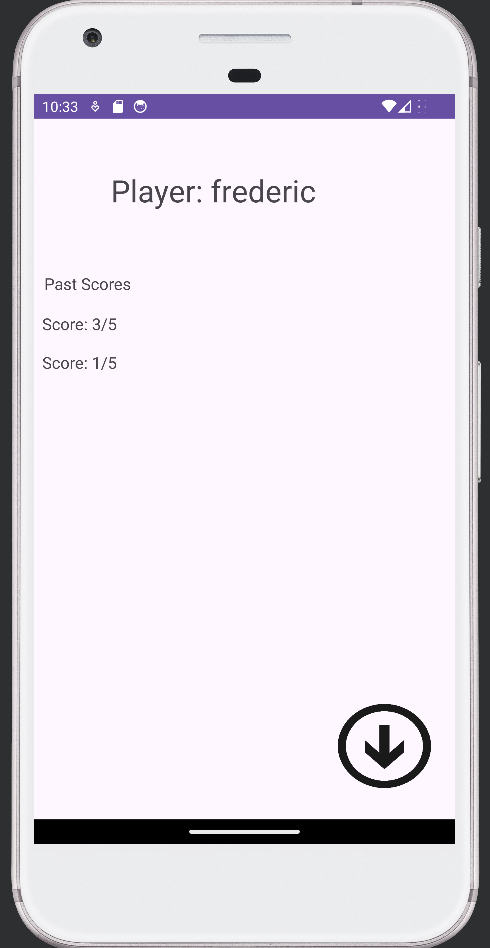
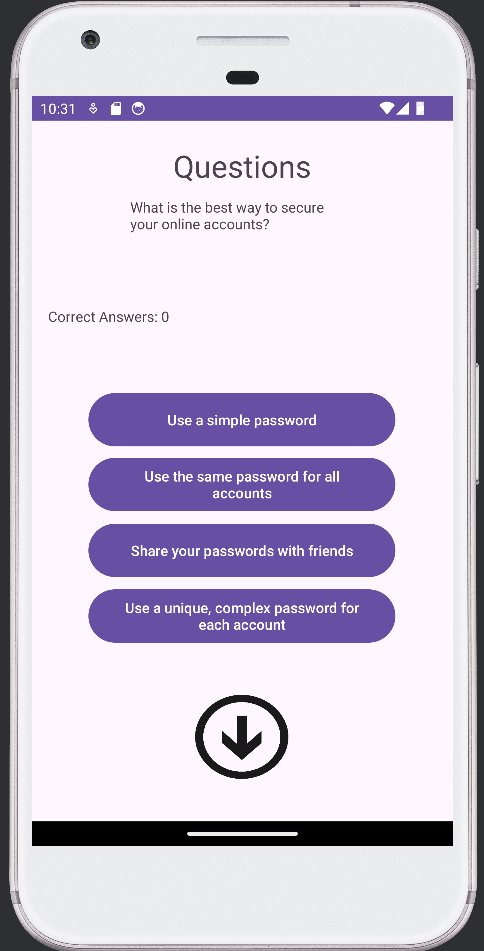
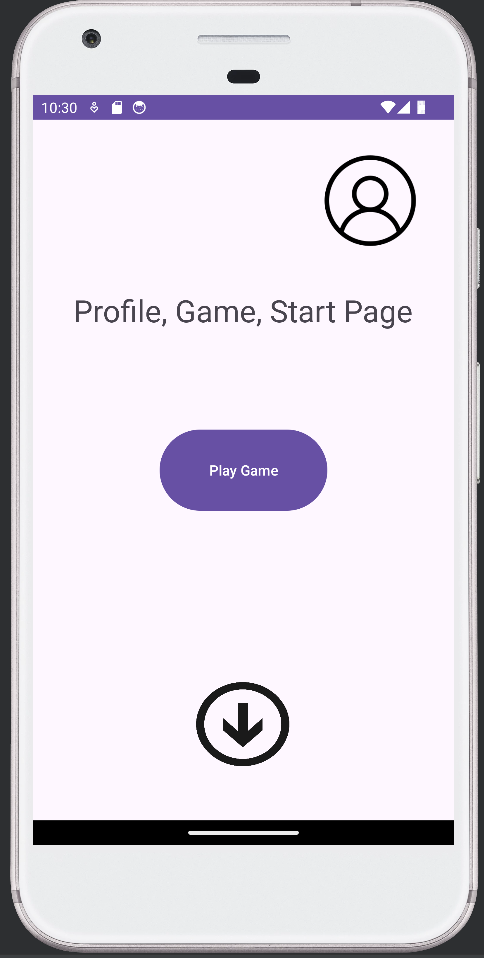
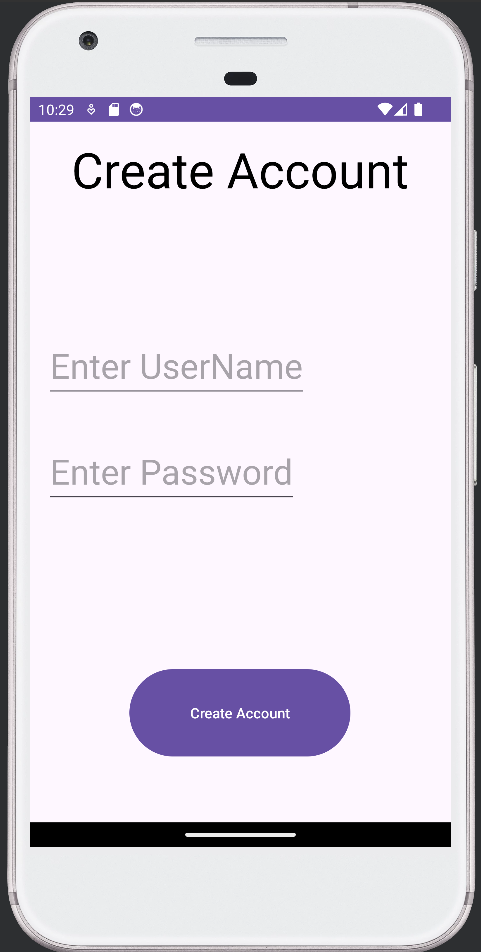
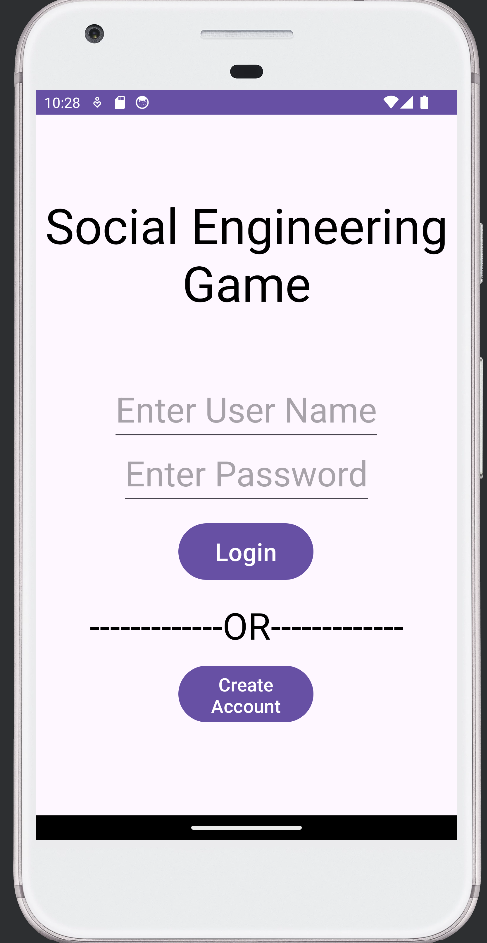
The User Interface of the application is designed to keep the user engaged on different screens. Each screen has a well organized layout with different components like buttons, text fields and icons which are positioned in a way to ensure ease of access.

The application has five screens with each screen being very descriptive.

* The “Start Screen” has a welcoming layout as it displays the name of the game, two text fields for the username and the password, and it has a login button as well as a button to create an account.
* If we click the “Create Account” button, it will take us to the screen where we create our account by providing some information.
* Once we login the home screen has a big “Play Game” button in the middle of the screen, a “Stats” button and a “Profile” button at the top right corner.
* The Profile button brings us to our profile displaying our name and our scores from our previous attempts.
* Our game screen has a friendly layout with a Kahoot theme to it. The question is displayed close to the top and under that there are four buttons each being different colors and it will display the four options. When you are done with the quiz there is a submit button above the question.

## Common Behaviors:

1. Status Bar:
   1. The status bar will be located at the top and it will display system information like time, battery percentage and network. It will be shown on all screens. It is not on the prototype
2. Home Button:
   1. We have a home button which will be on the bottom of the screen. It is displayed as a home icon which is simple and self explanatory.
3. Action Bar:
   1. We can include an action bar for each screen, with a back button that will bring you to the previous screen and it will include the home button. Also on the quiz screen it should display which question the user is on and how many remaining.
4. Menu and Popup Menus:
   1. Since this application is simple it does not necessarily need a menu option.
5. Profile Icon:
   1. We have a profile button on the top right corner on each screen which leads to the profile screen. It is not on the quiz screen.



**Appendix A: Glossary**

* JSON: A human- and machine-readable format for storing organized data.
* Android: A popular operating system for smartphones and other devices.
* Android Studio: A development environment for applications running on the Android Operating System.
* Android Studio device emulator: A simulator built into Android Studio which emulates characteristics of real smartphone hardware for application testing.
* Java: A popular general-purpose programming language used by Android devices.
* Moshi: A Java software library for parsing and encoding JSON data.
* MD5: A widely used but insecure hash function that produces a 128-bit hash.
* SHA256: A secure hash function that produces a 256-bit hash.