Link to Github Master Branch

https://github.gatech.edu/VIP-ITS/QuizApp-Spring-2022

Setup

App: We used Android Studio to develop the app. After downloading Android Studio and cloning the repo, load the QuizApp folder into Android Studio as a project. You likely will need to "Clear Caches and Restart" at this time. Add a virtual device of your choosing, using R as the Android version for the device. An excellent tutorial on getting started with Android Studio can be found here: https://youtu.be/jqB3r_16WqA

Database: We used a MongoDB Atlas cluster to store the information related to the classes in the UML diagram. To access the database, download MongoDB Compass here. Since this is a cloud-based database, there is no need to download mongo itself. Connect to the database with the connection string:
mongodb+srv://quizAppMobileUser:xkLaBBaa1dTsZuX8@quizappdb.7ltdg.mongodb.net/test
Each class is represented as a Collection and each object of that class is represented as a Document within that collection.

MainActivity.java

This class powers the home page of the app. The corresponding layout file is res/layout/activity_main.xml. The main activity is launched when the user launches the app. If the global CurrentUser object is not instantiated, then the user is redirected to the login page. If the CurrentUser object is instantiated, then the Home page is shown with three buttons: Browse Quizzes, Analytics, and Log Out.

ResultsActivity.java

This class powers the Results page of the app. The corresponding layout file is res/layout/activity_results.xml. At the end of a quiz, the user is directed to the Results page. This is when the app would make API requests to submit the quiz attempt and fetch skill recommendations from the knowledge tracing (KT) API. The User is meant to be shown how many questions they got right out of the total questions in the quiz. They are also shown recommended skills to study next, based on the results of the KT API calls.

User.java

This class stores user information. It is instantiated and populated upon login. When the user is currently working on a quiz, the currentQuiz holds that quiz. At that time, the currentQuizAttempt holds the users answers to that quiz. currentQuizAttempt is not yet implemented.

Quiz.java

This class holds a quiz object, which consists of a list of Question objects. The Quiz object is loaded by the Controller class from the MongoDB backend. The getCurrentQuestion method is meant to get the current question so it can be displayed. The peekNextQuestion method is meant to peek at the next question in order to determine its type so that the appropriate screen can be shown to the User.

Question.java

The Question class is a superclass for each of the question types. It has information such as question ID and question text.

Flashcard.java

The Flashcard class is a subclass of Question. It has a front and back of the card. It also requires the user to self-report if they got the flashcard right or wrong.

ShortAnswerQuestion.java

The ShortAnswerQuestion class is a subclass of Question. Grading an answer would be done by comparing the text to the correct answer.

MultipleChoiceQuestion.java

The MultipleChoiceQuestion class is a subclass of Question. In addition to the correct answer, it must have 3 incorrect answers. Future implementation details would include randomizing the order of the answers for display.

QuizAttempt.java

The QuizAttempt class is meant to hold the user's current answers for the quiz they are taking. As the user proceeds through a quiz, UserAnswer objects should be created and appended to the User's currentQuizAttempt. It has not been implemented yet.

UserAnswer.java

The UserAnswer class is meant to hold a User's response to a specific question and whether they got the answer right or wrong. The Knowledge Tracing team requires a timestamp for each question, so it also holds some metadata such as timestamp. This class has not been implemented yet.

Controller.java

The Controller class is meant to interface with the MongoDB cloud database. It can query any collection, such as the Question collection, returning a list of questions in JSON string format.

Converter.java

The Converter class is meant to convert data retrieved from the Controller class to usable formats, such as Question objects, Flashcard objects, etc..

FlashCardFrontActivity.java

This class powers the flashcard question screen of the app and it represents the front side of the flashcard. The corresponding layout file is activity_ flashcard_front.xml. Upon entering this screen, either a new question is created and populated using the getNextQuestion() method, or the data from the current flashcard question is populated, depending on whether the user is flipping to the front of the flashcard or if they have moved on to the next question in the quiz. The user can flip to the back of the flashcard by pressing on the Flip button, which will take them to the FlashCardBackActivity screen. They can also mark the question as incorrect or correct, or skip to the next question using the corresponding buttons on the screen.

FlashCardBackActivity.java

This class represents the back of the flashcard. The corresponding layout file is activity_flashcard_back.xml. When the user presses the button that allows them to flip to the front of the flashcard, the data of the current question is passed to the FlashCardFrontActivity screen in the form of an Intent object. They can also mark the question as incorrect or correct, or go to the next question in the quiz using the corresponding buttons on the screen.

MultipleChoiceActivity.java

This class represents the multiple choice question screen of the app. The corresponding layout file is activity_multiplechoice_question_xml. The user can select any of the radio buttons to answer the question. The Submit button will provide instant answer validation (for future implementation). The Next button takes the user to the next question screen.

ShortAnswerActivity.java

This class represents the Short Answer question screen of the app. The corresponding layout file is activity_short_answer.xml. The user can select any of the radio buttons to answer the question. The Submit button will provide instant answer validation (for future implementation). The Next button takes the user to the next question screen. It features an image with a corresponding question, and asks the user a question based on that image. There is a textbox where the user can enter their answer.   It allows variations of the correct answer to be inputted based on the values specified within a hashmap(in the example demo, if you input ‘sine’ or ‘sine graph’ you will get the question correct). It also provides feedback letting you know whether or not you answered the question correctly, and what the correct answer was if you did not answer it correctly.