Programming Project 4: State capitals quiz

Due: October 28, 2019

In this project, you and your partner (you will work in pairs) will create a simple quiz application, called the State Capitals Quiz, testing the user's knowledge of state capital cities in the US. As in the previous project, you will work with fragments, in order to make your app more flexible. In addition, you will work with saving and restoring the state of your app. You will also learn how to work with Comma-Separated Values (CSV) files, asynchronous tasks, and an SQLite database. Finally, you will further develop your skills in developing apps and utilizing new widgets, as needed.

An initial CSV file with states and their capitals, `state_capitals.csv`, is available on `eLC` in the Projects folder. You should edit the file and provide 2 additional cities for each state, besides the already listed capital. The 2 additional cities should be the largest (or well known and significant) cities in the state to serve as *plausible* capitals in the quiz. You can use Microsoft Excel or LibreOffice Calc to edit the CSV file.

Here is how your application should work:

1. At the initialization, the app should read the CSV file with the 50 states and their capitals (the file should be provided as a raw resource or in the assets folder) and store its data in a suitable SQLite database. The main UI thread should not be adversely affected by IO operations, so you *must* perform this operation within an `AsyncTask`. Initially, the SQLite database does not exist, and the app should create a suitable SQLite database to store the states and cities, quizzes, and their results. On all subsequent executions, your app should check if the SQLite database already exists, and if so, reuse it.
2. The initial (splash) screen should display the purpose of the quiz and explain the basic rules (you may also offer a help screen). From the initial screen, the user should be able to start a new quiz or view the results of the past quizzes.
3. When the user wants to start a new quiz, the app should create a new quiz by randomly selecting 6 states from the database as the quiz questions (use `Math.random`). It should store the quiz (with the selected states) in the database (a quiz should also have a date, which would be set at the end of the quiz). Also, it should populate a suitable data structure in the app to store the states, their capitals, the 2 additional cities, and the user answer. Be clever about selecting the states randomly, as the states in a quiz should not have duplicates. The user will be quizzed on the selected states.
4. Then, the app should quiz the user, asking about the capital of each of the selected states. Each question should display the 3 cities in a given state *in random order* and ask the user which one is the capital. The choices should be labeled 1 through 3 (or A through C). The quiz question should be clearly stated ("What is the capitol of XXX?", or similarly), and the choices should be shown below as RadioButtons (only one choice will be accepted).
5. The user should be able to swipe left to go to the next question. Consequently, when the user swipes left, the system should record the user's answer as correct or incorrect and display the next question. You may also offer an additional way to move to the next question (for example, using a "Next question" button) but swiping left *must* be implemented.
6. Once the user answers all of the questions, the app should display the result of the current quiz to the user, which should be right after the final swipe left, following the final question. At that point, the app should store the date of the quiz and its result in the SQLite database. Again, this should be done in an `AsyncTask`.

7. From the splash screen, or from the current quiz results screen the user should be able to go to his/her prior quizzes and results; the results should include the date of each quiz. Database retrieval of past results should be done using an `AsyncTask`, as well.
8. You may incorporate a navigation drawer in your app, if you would like.
9. You *must* implement application state saving and restoring and provide for orientation changes. A simple way to do this would be to save the number already answered questions. If the partial results are stored, the user should be able to resume the quiz after, for example, answering a phone call.

Additional requirements

- **IMPORTANT**: Both students on each team are expected to contribute equally to the project. You should meet and work on the project together, preferably engaging in *pair programming*. You will be asked to submit a form with detailed contributions of both members.
- Select a suitable name for your project. **You must use Android Studio version 3.5 or 3.4.2 (preferably, 3.5.1, the most recent one). When creating your project, you must select `Phone and Tablet`, and API 16: Android 4.1 (Jelly Bean)**. **You must code your app in Java.**
- Application checkpoints:
    - The app must use fragments, swiping, radio buttons, and other views/widgets, as needed.
    - You must also use a CSV file with the initial data and an SQLite database to store quizzes and their results.
    - I will accept apps, which do not initially use a given CSV file, but use a pre-loaded database file, also as a raw resource or an asset file. At the app initialization, the database should be copied to the proper directory, using an `AsyncTask`.
    - You must use `AsyncTask` to perform IO operations with the CSV file and the database.
    - Your app must adjust the layout when switching orientations, as needed.
    - Your app should be able to save and restore the partial quiz results, as the quiz may be interrupted by an incoming call, or by the user switching to another app for a moment, or just by changing the orientation of the device.
    - Whenever possible, you should use `ConstraintLayout`. However, you *must not* use RelativeLayout in your app.
- Test your app in a Pixel XL, API 27 AVD and eliminate any errors.
- Create a ZIP file with your entire project directory and submit it your zipped project to the **Project 4 Assignment Submission** Folder on eLC. As before, if you don't have a fast Internet access at home, plan on submitting your project while on campus, as an expected size of your ZIP file may well exceed 12-15 MB.
- Follow good Java coding style (suitable variable names, indentation, etc).
- Include suitable comments in JavaDoc style.

Additional explanations will be provided in class and on eLC.