Table of contentes

Introduction:

The inspiration behind this project stems from my personal experience working as a bartender in London. In the fast-paced bartending industry, there is a high turnover rate, with some bartenders changing jobs every month or even every week. As a result, the consistency of knowledge and skills related to cocktail specifications among bartenders can vary greatly. This inconsistency has led to significant losses in time and resources for businesses.

To address this issue, I suggest creating an application designed to facilitate remote training for bartenders. Exploit the power of technology, the app will offer accessible training sessions that bartenders can complete from anywhere, equipping them with the necessary knowledge and skills to prepare cocktails according to the required specifications. Moreover, the app will include a managerial dashboard, allowing company managers to

track their bartenders' progress and provide feedback. This feature ensures that the entire team is well-trained and prepared to deliver the desired results.

Design:

The Cocktail Training app is designed to provide remote training to bartenders and managers. The app is built on the Android platform using Java programming language. The app features a login screen that enables existing users to login and new users to create an account.

Upon successful login, the user is redirected to the quiz section of the app. The quiz section features multiple-choice questions related to cocktail recipes, and the user's progress is tracked and stored in an SQLite database on a remote server. The app also provides a leaderboard to enable managers to track their team's performance and evaluate their progress.

The app's user interface features a simple and intuitive design. The login screen features two input fields for the user's username and password and three buttons for logging in, creating a new user account, and deleting an existing user account. The quiz section of the app features four buttons for each multiple-choice question and an image of the cocktail recipe.

The app is built using Android fragments and Intents, which enable seamless transitions between various activities. The app's functionality is powered by a QuizClass class, which stores the questions, options, and correct answers for the quiz. The app utilizes a BottomSheetDialog to display the user's score at the end of the quiz.

The app's database is stored on a remote server, which enables managers to access their team's progress from anywhere. The app sends user scores to the server using a POST request, and the server stores the data in an SQL database. The server also provides a PHP script for inserting user scores into the database.

Software Architecture

The bartender training app's software architecture is composed of the following key components:

User Interface Layer:

This layer encompasses the app's graphical user interface, which allows users to engage with the application. Components include the login screen, quiz screen, and score screen.

Application Logic Layer:

Responsible for the app's business logic, this layer contains the code for user login and registration, quiz question retrieval from the database, user answer verification, and user score calculation.

Data Storage Layer:

This layer houses the database and code for data storage and retrieval. The database stores user information, quiz questions, and user scores, utilizing SQLite for on-device data storage.

Network Communication Layer:

This layer contains the code for server communication to store user scores, using HTTP for data transmission to and from the server.

The software architecture adheres to the Model-View-Controller (MVC) pattern. The user interface layer represents the view, displaying information and receiving user input. The application logic layer, the controller, manages user input and interacts with the model layer. The data storage layer, the model, stores and retrieves data.

Additionally, the software architecture follows the SOLID principles of software design:

Single Responsibility Principle:

Each application component has one responsibility, simplifying understanding, modification, and maintenance.

Open-Closed Principle:

The app is designed for extension while remaining closed to modification, allowing new features without altering existing code.

Liskov Substitution Principle:

The app allows for interchangeable components that implement the same interface, promoting flexibility and modularity.

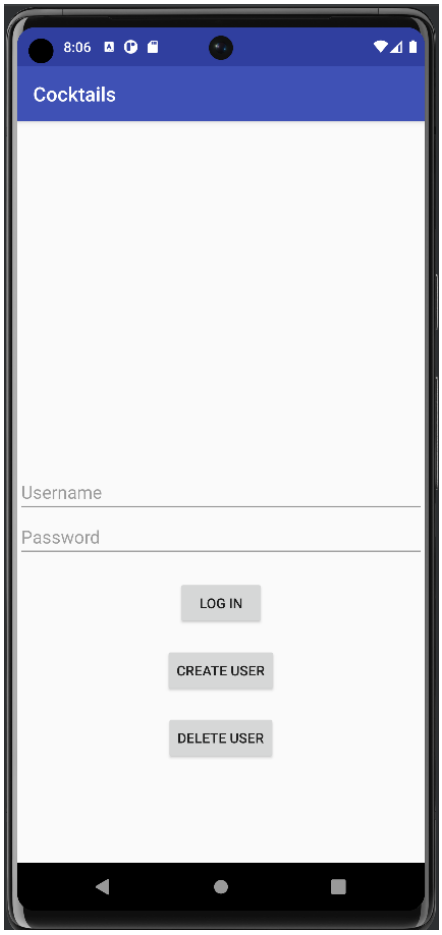
Interface Segregation Principle:

Designed with small, focused interfaces, the app is easier to use and maintain.

Dependency Inversion Principle:

The app depends on abstractions rather than concrete implementations, enhancing flexibility and modularity.

Log in design:



It can be observed in this first activity there are 2 fields for the username and password and 3 buttons: Log in, Create user and Delete user.

Xml code for the log in interface:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".LoginActivity">

    <!-- Username input field -->
    <EditText
        android:id="@+id/usernameEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:hint="Username" />

    <!-- Password input field -->
    <EditText
        android:id="@+id/passwordEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/usernameEditText"
        android:hint="Password"
        android:inputType="textPassword" />

    <!-- Log In button -->
    <Button
        android:id="@+id/loginButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/passwordEditText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="Log In" />

    <!-- Create User button -->
    <Button
        android:id="@+id/createUserButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/loginButton"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="Create User" />

    <!-- Delete User button -->
    <Button
        android:id="@+id/deleteUserButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/createUserButton"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="Delete User" />



</RelativeLayout>
```

It defines a RelativeLayout with four child views: two EditText views for the user to input their username and password, and three Button views to log in, create a new user, and delete an existing user.

The EditText views have unique IDs assigned to them, which can be used to reference them in the Java code of the application. The loginButton, createUserButton, and deleteUserButton also have unique IDs assigned to them, which can be used to handle the button click events in the Java code.

Additionally, the layout attributes for each view are defined, including their size, position, and text. For example, the loginButton is centered horizontally in the layout, below the passwordEditText and with a margin of 20dp from the top. Its text is set to "Log In".

Java class for the log in activity:

```
package com.example.cocktails;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class LoginActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // Initialize views
        EditText usernameEditText = findViewById(R.id.usernameEditText);
        EditText passwordEditText = findViewById(R.id.passwordEditText);
        Button loginButton = findViewById(R.id.loginButton);

        // Set login button click listener
        loginButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = usernameEditText.getText().toString();
                String password = passwordEditText.getText().toString();

                getCurrentUsername(username);

                // Retrieve stored password for the given username
                SharedPreferences sharedPreferences =
getSharedPreferences("user_credentials", MODE_PRIVATE);
                String storedPassword =
sharedPreferences.getString(username, null);

                // Check if stored password matches the input password
                if (storedPassword != null &&
storedPassword.equals(password)) {
                    // Open MainActivity and pass the username
                    Intent intent = new Intent(LoginActivity.this,
MainActivity.class);
                    intent.putExtra("username", username);
```

```
                    startActivity(intent);
                    finish();
                } else {
                    // Show login failed message
                    Toast.makeText(LoginActivity.this, "Invalid username or
password", Toast.LENGTH_SHORT).show();
                }
            }

            public String getCurrentUsername(String username) {
                return username;
            }
        });

        // Initialize create user button
        Button createUserButton = findViewById(R.id.createUserButton);

        // Set create user button click listener
        createUserButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = usernameEditText.getText().toString();
                String password = passwordEditText.getText().toString();

                // Check if input fields are empty
                if (username.isEmpty() || password.isEmpty()) {
                    Toast.makeText(LoginActivity.this, "Username and
password cannot be empty", Toast.LENGTH_SHORT).show();
                }

                // Check if user already exists
                SharedPreferences sharedPreferences =
getSharedPreferences("user_credentials", MODE_PRIVATE);
                String existingPassword =
sharedPreferences.getString(username, null);

                // Create new user if it does not exist
                if (existingPassword == null) {
                    SharedPreferences.Editor editor =
sharedPreferences.edit();
                    editor.putString(username, password);
                    editor.apply();

                    Toast.makeText(LoginActivity.this, "User created
successfully", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(LoginActivity.this, "User already
exists", Toast.LENGTH_SHORT).show();
                }
            }
        });

        // Initialize delete user button
        Button deleteUserButton = findViewById(R.id.deleteUserButton);

        // Set delete user button click listener
        deleteUserButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = usernameEditText.getText().toString();
```

```
                // Check if username field is empty
                if (username.isEmpty()) {
                    Toast.makeText(LoginActivity.this, "Username cannot be
empty", Toast.LENGTH_SHORT).show();
                } else {
                    SharedPreferences sharedPreferences =
getSharedPreferences("user_credentials", MODE_PRIVATE);
                    String existingPassword =
sharedPreferences.getString(username, null);

                    // Delete user if it exists
                    if (existingPassword != null) {
                        SharedPreferences.Editor editor =
sharedPreferences.edit();
                        editor.remove(username);
                        editor.apply();

                        Toast.makeText(LoginActivity.this, "User deleted
successfully", Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(LoginActivity.this, "User not
found", Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });

    }

}
```

It handles user login, user creation, and user deletion functionalities. The activity has three main buttons: login, create user, and delete user. Users can input their username and password in the provided fields.

When the login button is clicked, the app checks if the input username and password match the stored credentials. If they match, the user is successfully logged in and navigated to the MainActivity. If not, an error message is displayed.

When the create user button is clicked, the app checks if the username and password fields are not empty. If the input username does not exist, the app creates a new user by storing the username and password. If the user already exists, an error message is displayed.

When the delete user button is clicked, the app checks if the username field is not empty. If the input username exists, the app deletes the user by removing the stored username and password. If the user is not found, an error message is displayed.

Main Activity design:

Xml code:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!-- Display the number of questions attempted -->
    <TextView
        android:id="@+id/NumberQuestionsAttemptedTV"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:gravity="center"
        android:text="Nr. of Questions"
        android:textAlignment="center"
        android:textAllCaps="false"
        android:textColor="@color/black"
        android:textSize="20dp" />

    <!-- Display the question text -->
```

```
    <TextView
        android:id="@+id/QuestionsTV"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/NumberQuestionsAttemptedTV"
        android:layout_marginTop="20dp"
        android:gravity="center"
        android:text="Questions"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="25sp" />


    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/QuestionsTV"
        android:orientation="vertical" />

    <!-- Button for option 1 -->
    <Button
        android:id="@+id/btOption1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/QuestionsTV"
        android:layout_marginTop="325dp"
        android:layout_marginBottom="10dp"
        android:background="@color/black"
        android:padding="4dp"
        android:text="Option 1"
        android:textColor="@android:color/white"
        android:textAllCaps="false" />


    <!-- Button for option 2 -->
    <Button
        android:id="@+id/btOption2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btOption1"
        android:layout_marginTop="15dp"
        android:layout_marginBottom="10dp"
        android:background="@color/black"
        android:padding="4dp"
        android:text="Option 2"
        android:textColor="@android:color/white"
        android:textAllCaps="false" />


    <!-- Button for option 3 -->
    <Button
        android:id="@+id/btOption3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btOption2"
        android:layout_marginTop="15dp"
        android:layout_marginBottom="10dp"
        android:background="@color/black"
        android:padding="4dp"
        android:text="Option 3"
        android:textColor="@android:color/white"
```

```
        android:textAllCaps="false" />


    <!-- Button for option 4 -->
    <Button
        android:id="@+id/btOption4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btOption3"
        android:layout_marginTop="15dp"
        android:layout_marginBottom="10dp"
        android:background="@color/black"
        android:padding="4dp"
        android:text="Option 4"
        android:textColor="@android:color/white"
        android:textAllCaps="false" />

    <!-- Image of the cocktail -->
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="316dp"
        android:layout_below="@+id/QuestionsTV"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="0dp"
        android:layout_marginEnd="0dp" />


</RelativeLayout>
```

The layout contains various UI elements, such as TextViews, Buttons, and an ImageView, which are arranged within a RelativeLayout.

- TextView (NumberQuestionsAttemptedTV): Displays the number of questions attempted so far.
- TextView (QuestionsTV): Displays the text of the current quiz question.
- LinearLayout: A placeholder that is not used in the current layout.
- Button (btOption1): Represents the first answer option for the quiz question.
- Button (btOption2): Represents the second answer option for the quiz question.
- Button (btOption3): Represents the third answer option for the quiz question.
- Button (btOption4): Represents the fourth answer option for the quiz question.
- ImageView (imageView): Displays an image related to the current quiz question.

The layout is designed to present a quiz question along with four answer options. The user can select one of the options by clicking the corresponding button.

Main Activity java code:

```
package com.example.cocktails;

import androidx.appcompat.app.AppCompatActivity;
```

```java
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.google.android.material.bottomsheet.BottomSheetDialog;

import java.util.ArrayList;
import java.util.Locale;
import java.util.Random;

// MainActivity class for the cocktail quiz app
public class MainActivity extends AppCompatActivity {
    // Declare UI elements and variables
    private Button option1Bt, option2Bt, option3Bt, option4Bt;
    private ArrayList<QuizClass> QArrayList;
    Random random;
    int CurrentScore = 0;
    int QuestionAttempted = 0;
    int CurrentPosition;
    private TextView QuestionTV, QuestionNumberTV;
    private ImageView QuestionImage;
    private String username;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initialize UI elements
        QuestionTV = findViewById(R.id.QuestionsTV);
        QuestionNumberTV = findViewById(R.id.NumberQuestionsAttemptedTV);
        option1Bt = findViewById(R.id.btOption1);
        option2Bt = findViewById(R.id.btOption2);
        option3Bt = findViewById(R.id.btOption3);
        option4Bt = findViewById(R.id.btOption4);
        QuestionImage = findViewById(R.id.imageView);

        // Initialize quiz data and random number generator
        QArrayList = new ArrayList<>();
        random = new Random();

        // Populate quiz questions
        getQuizQuestions(QArrayList);

        // Set the initial question and options
        CurrentPosition = random.nextInt(QArrayList.size());
        setDataToViews(CurrentPosition);

        // Set onClick listeners for each of the four option buttons

        // Option 1
        option1Bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if the selected answer is correct and update the
```

```
score
                if
(QArrayList.get(CurrentPosition).getAnswer().trim().toLowerCase().equals(op
tion1Bt.getText().toString().trim().toLowerCase())) {
                    CurrentScore++;
                }
                // Update the number of questions attempted and display a
new question
                QuestionAttempted++;
                CurrentPosition = random.nextInt(QArrayList.size());
                setDataToViews(CurrentPosition);
            }
        });

        // Option 2
        option2Bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if the selected answer is correct and update the
score
                if
(QArrayList.get(CurrentPosition).getAnswer().trim().toLowerCase().equals(op
tion2Bt.getText().toString().trim().toLowerCase())) {
                    CurrentScore++;
                }
                // Update the number of questions attempted and display a
new question
                QuestionAttempted++;
                CurrentPosition = random.nextInt(QArrayList.size());
                setDataToViews(CurrentPosition);
            }
        });

        // Option 3
        option3Bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Check if the selected answer is correct and update the
score
                if
(QArrayList.get(CurrentPosition).getAnswer().trim().toLowerCase().equals(op
tion3Bt.getText().toString().trim().toLowerCase())) {
                    CurrentScore++;
                }
                // Update the number of questions attempted and display a
new question
                QuestionAttempted++;
                CurrentPosition = random.nextInt(QArrayList.size());
                setDataToViews(CurrentPosition);
            }
        });

        // Option 4
        option4Bt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
// Check if the selected answer is correct and update the score
                if
(QArrayList.get(CurrentPosition).getAnswer().trim().toLowerCase().equals(op
tion4Bt.getText().toString().trim().toLowerCase())) {
                    CurrentScore++;
```

```
                }
// Update the number of questions attempted and display a new question
                QuestionAttempted++;
                CurrentPosition = random.nextInt(QArrayList.size());
                setDataToViews(CurrentPosition);
            }
        });
    }

    // Method to show the score in a bottom sheet dialog
    private void ShowScore() {
        BottomSheetDialog BSD = new BottomSheetDialog(MainActivity.this);
        View BSV =
LayoutInflater.from(getApplicationContext()).inflate(R.layout.score,
(LinearLayout) findViewById(R.id.LLid));
        TextView scoreTV = BSV.findViewById(R.id.TVscore);
        Button restartQuizBt = BSV.findViewById(R.id.ButtonRestart);
        scoreTV.setText("Your score \n" + CurrentScore + "/4");

        // Send the user's score to the server
        QuizResultsActivity score = new QuizResultsActivity();
        // Retrieve the username from the Intent
        Intent intent = getIntent();
        username = intent.getStringExtra("username");
        score.sendUserScoreToServer(username, CurrentScore);

        // Set the bottom sheet as non-cancellable and show it
        BSD.setCancelable(false);
        BSD.setContentView(BSV);
        BSD.show();

        // Set onClick listener for the restart quiz button
        restartQuizBt.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Reset the quiz and dismiss the bottom sheet
                CurrentPosition = random.nextInt(QArrayList.size());
                setDataToViews(CurrentPosition);
                CurrentScore = 0;
                BSD.dismiss();
            }
        });
    }

    // Method to set data to UI elements
    private void setDataToViews(int currentPosition) {
        // Update the number of questions attempted
        QuestionNumberTV.setText("Questions Attempted: " +
QuestionAttempted + "/4");

        // If all questions have been attempted, show the score
        if (QuestionAttempted == 4) {
            QuestionAttempted = 0;
            ShowScore();
        } else {
            // Set question and options to the UI elements

QuestionTV.setText(QArrayList.get(CurrentPosition).getQuestion());

option1Bt.setText(QArrayList.get(CurrentPosition).getOption1());
```

```
option2Bt.setText(QArrayList.get(CurrentPosition).getOption2());

option3Bt.setText(QArrayList.get(CurrentPosition).getOption3());

option4Bt.setText(QArrayList.get(CurrentPosition).getOption4());

QuestionImage.setImageResource(QArrayList.get(currentPosition).getImageReso
urceId());
        }
    }

    // Method to populate quiz questions
    private void getQuizQuestions(ArrayList<QuizClass> quizClassArrayList)
{
        quizClassArrayList.add(new QuizClass("Which Spirit is in Negroni?",
"Vodka", "Gin", "Rum", "Tequila", "Gin", R.drawable.negroni));
        quizClassArrayList.add(new QuizClass("Which Spirit is in Espresso
Martini?", "Vodka", "Gin", "Rum", "Tequila", "Vodka",
R.drawable.espressomartini));
        quizClassArrayList.add(new QuizClass("Which Spirit is in Bramble?",
"Vodka", "Gin", "Rum", "Tequila", "Gin", R.drawable.bramble));
        quizClassArrayList.add(new QuizClass("Which Spirit is in Pina
Colada?", "Vodka", "Gin", "Rum", "Tequila", "Rum", R.drawable.pina));
    }

}
```

Explanation of what the Java code does:

- Declare and initialize UI elements: The code declares and initializes the TextViews, Buttons, and ImageView that were defined in the XML layout. It also declares variables for the current score, the number of questions attempted, and the current position in the quiz.

- Create an ArrayList of QuizClass objects: This ArrayList stores the quiz questions, answer options, correct answer, and image resource IDs. The getQuizQuestions() method is called to populate this ArrayList with quiz data.

- Generate a random question: The random object is used to generate a random index for selecting a question from the ArrayList. The setDataToViews() method is called to display the selected question and its answer options.

- Set up onClickListeners for the option buttons: The code sets up click listeners for each answer option button (option1Bt, option2Bt, option3Bt, option4Bt). When a user clicks on an option, the app checks whether the selected option is correct by comparing it with the correct answer stored in the QuizClass object. If the answer is correct, the user's score is incremented. The app then selects another random question and updates the UI.

- Update the UI: The setDataToViews() method is called to update the UI elements, such as the question text, answer options, and question image. It also displays the user's progress (number of questions attempted) and, after the user has attempted all questions, calls the ShowScore() method to display the final score.

- Show the final score: The ShowScore() method displays the user's final score in a BottomSheetDialog. It also sends the user's score to the server (using the sendUserScoreToServer() method in the QuizResultsActivity class) and provides a "Restart" button for the user to start the quiz again.

QuizClass java code:

```
// Class that defines the Quiz object
public class QuizClass {

    // Fields that define the Quiz object
    private String question;
    private String option1;
    private String option2;
    private String option3;
    private String option4;
    private String answer;
    private int imageResourceId;

    // Constructor that initializes the Quiz object with the passed values
    public QuizClass(String question, String option1, String option2,
String option3, String option4, String answer,int imageResourceId) {
        this.question = question;
        this.option1 = option1;
        this.option2 = option2;
        this.option3 = option3;
        this.option4 = option4;
        this.answer = answer;
        this.imageResourceId = imageResourceId;
    }

    // Getter method for the question field
    public String getQuestion() {
        return question;
    }

    // Setter method for the question field
    public void setQuestion(String question) {
        this.question = question;
    }

    // Getter method for the option1 field
    public String getOption1() {
        return option1;
    }

    // Setter method for the option1 field
```

```
    public void setOption1(String option1) {
        this.option1 = option1;
    }

    // Getter method for the option2 field
    public String getOption2() {
        return option2;
    }

    // Setter method for the option2 field
    public void setOption2(String option2) {
        this.option2 = option2;
    }

    // Getter method for the option3 field
    public String getOption3() {
        return option3;
    }

    // Setter method for the option3 field
    public void setOption3(String option3) {
        this.option3 = option3;
    }

    // Getter method for the option4 field
    public String getOption4() {
        return option4;
    }

    // Setter method for the option4 field
    public void setOption4(String option4) {
        this.option4 = option4;
    }

    // Getter method for the answer field
    public String getAnswer() {
        return answer;
    }

    // Setter method for the answer field
    public void setAnswer(String answer) {
        this.answer = answer;
    }

    // Getter method for the imageResourceId field
    public int getImageResourceId() {
        return imageResourceId;
    }

    // Setter method for the imageResourceId field
    public void setImageResourceId(int imageResourceId) {
        this.imageResourceId = imageResourceId;
    }
}
```

This class is used to store and manage the data related to each question in the quiz app. The class has the following components:

- Fields: The QuizClass object has several fields that store information about a quiz question, such as the question text, answer options, correct answer, and the image resource ID associated with the question.

- Constructor: The constructor is called when creating a new QuizClass object. It initializes the object with the values passed as arguments for the question, answer options, correct answer, and image resource ID.

- Getter methods: These methods allow other parts of the code to retrieve the values of the fields in a QuizClass object. For example, getQuestion() returns the question text, and getOption1() returns the text for the first answer option.

- Setter methods: These methods enable other parts of the code to modify the values of the fields in a QuizClass object. For instance, setQuestion() updates the question text, and setOption1() updates the text for the first answer option.

By using the QuizClass, the quiz app can store and manage the data for each question, making it easier to work with the quiz questions and answers throughout the app.

Xml score layout code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LLid"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <!-- TextView to display the user's score -->
    <TextView
        android:id="@+id/TVscore"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:gravity="center"
        android:padding="3dp"
        android:text="Your Score: "
        android:textAlignment="center"
        android:textAllCaps="false"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="normal" />

    <!-- Button to restart the quiz -->
    <Button
        android:id="@+id/ButtonRestart"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_below="@+id/TVscore"
        android:layout_margin="20dp"
        android:text="Restart Quiz"
        android:textAllCaps="false" />

</LinearLayout>
```

The layout is used to display the user's score and allow them to restart the quiz.

Breakdown of the elements and their attributes:

- LinearLayout: This is the root layout element with a vertical orientation, which means that its child elements will be arranged one below the other.

- TextView (ID: TVscore): This TextView is used to display the user's score. It has various attributes such as layout width, height, margin, gravity, padding, text alignment, text color, and text size, which control its appearance and position within the layout.

- Button (ID: ButtonRestart): This Button is used to restart the quiz when clicked. It is placed below the TextView and has attributes such as layout width, height, margin, and text, which control its appearance and position within the layout.

The purpose of this layout is to show the user's score after they have completed the quiz and provide an option to restart the quiz by clicking the "Restart Quiz" button.

Score layout:

Quiz Results Activity java code:

```
package com.example.cocktails;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
```

```
import okhttp3.Call;
import okhttp3.Callback;
import okhttp3.FormBody;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;

public class QuizResultsActivity extends AppCompatActivity {

    // This method sends user's score to the server
    public void sendUserScoreToServer(final String username, final int
score) {
        // Run the network operations on a separate thread
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    // Create URL object
                    URL url = new
URL("http://10.0.2.2/cocktail_training/insert_user_score.php");
                    // Open a connection to the server
                    HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                    // Set the request method to POST
                    connection.setRequestMethod("POST");
                    // Enable output from the connection
                    connection.setDoOutput(true);
                    // Set the content type of the request
                    connection.setRequestProperty("Content-Type",
"application/json; charset=UTF-8");

                    // Create a JSON object with the user's score and
username
                    JSONObject jsonObject = new JSONObject();
                    jsonObject.put("username", username);
                    jsonObject.put("score", score);

                    // Get the output stream from the connection and write
the JSON object to it
                    OutputStream outputStream =
connection.getOutputStream();
                    outputStream.write(jsonObject.toString().getBytes("UTF-
8"));
                    outputStream.close();

                    // Get the response code from the server
                    int responseCode = connection.getResponseCode();
                    if (responseCode == HttpURLConnection.HTTP_OK) {
                        // If the response code is OK, log a success
message
                        Log.d("QuizResultsActivity", "User score saved
successfully.");
                    } else {
                        // If the response code is not OK, log an error
message
                        Log.e("QuizResultsActivity", "Error saving user
score, response code: " + responseCode);
                    }
```

```
                    // Disconnect the connection
                    connection.disconnect();
                } catch (IOException | JSONException e) {
                    // If an exception is caught, print the stack trace
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

The activity has one method, sendUserScoreToServer, which takes the user's username and score as input arguments.

sendUserScoreToServer method:

- Run the network operations on a separate thread to avoid blocking the main UI thread.
- Create a URL object pointing to the server-side script that processes the user scores (insert_user_score.php).
- Open an HttpURLConnection to establish a connection with the server.
- Set the request method to "POST" to send data to the server.
- Enable output from the connection to send data.
- Set the content type of the request to "application/json; charset=UTF-8".
- Create a JSONObject with the user's score and username.
- Get the output stream from the connection and write the JSON object to it, then close the output stream.
- Get the response code from the server to check whether the request was successful.
    - If the response code is HTTP_OK, log a success message indicating that the user's score was saved successfully.
    - If the response code is not HTTP_OK, log an error message with the response code.
- Disconnect the connection.
- If any exceptions occur (such as IOException or JSONException), catch them and print the stack trace.

This code is responsible for sending the user's quiz score to a server using an HttpURLConnection. It creates a JSON object containing the user's username and score and sends it as a POST request to the server-side script (insert_user_score.php). The script then processes the user's score and stores it in the server's database.

insert_user_score.php code:

<?php

```
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");

// Database credentials
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "cocktail_training";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$data = json_decode(file_get_contents("php://input"));

if (!empty($data->username) && !empty($data->score)) {
    $username = $data->username;
    $score = $data->score;

    $stmt = $conn->prepare("INSERT INTO users (username, score) VALUES (?, ?)");
    $stmt->bind_param("si", $username, $score);

    if ($stmt->execute()) {
```

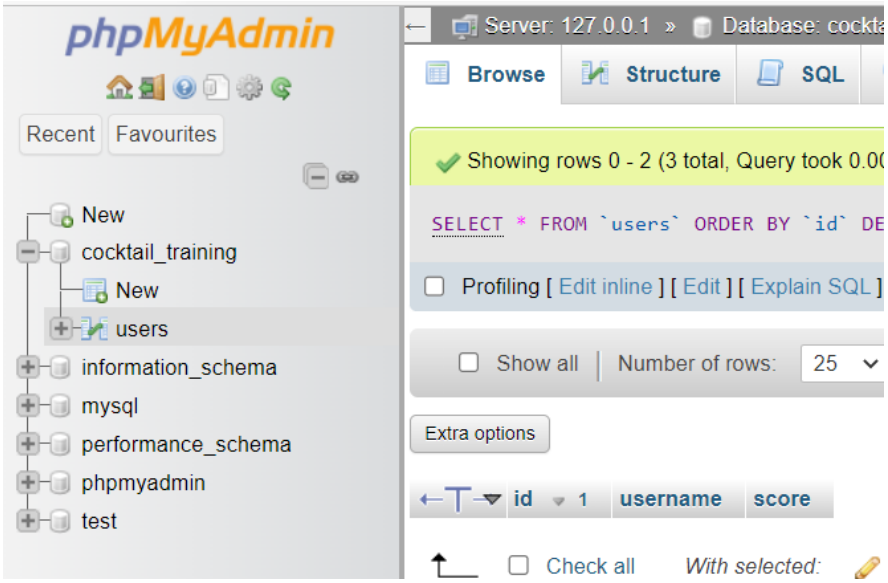
```
        file_put_contents('log.txt', "Data inserted successfully\n", FILE_APPEND);
    } else {
        file_put_contents('log.txt', "Error inserting data: " . $stmt->error . "\n", FILE_APPEND);
    }

    $stmt->close();
} else {
    http_response_code(400);
    echo json_encode(array("message" => "Unable to save user score. Data is incomplete."));
}

$conn->close();
file_put_contents('log.txt', "Data received: " . json_encode($data) . "\n", FILE_APPEND);

?>
```

This PHP script receives JSON data containing a user's username and score via a POST request, inserts the data into the users table in the MySQL database, and logs the result to a text file. If the data is incomplete, it returns a 400 Bad Request status and a JSON message indicating that the data is incomplete.

get_user_scores.php code:

This PHP script connects to the cocktail_training MySQL database, retrieves user data (id, username, and score) from the users table, and returns the data as a JSON-encoded string. If no records are found, the script returns a JSON message indicating that no records were found.

Database:



We can observe that there's a database called cocktail_training and a table in it called users which stores the id, username and score of the user.

The primary key is the id which will be auto incremented by the database at each insertion of a new user.

HTML webpage code:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>User Scores</title>

    <style>

        table {
```

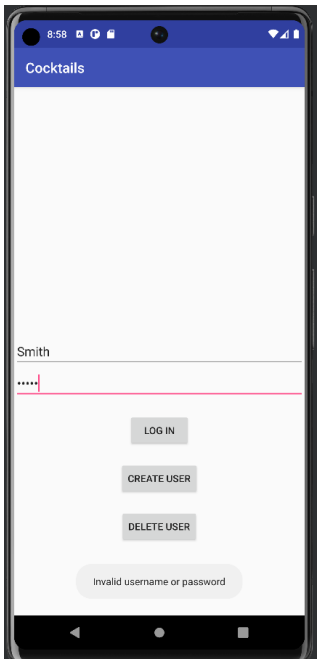
```
            border-collapse: collapse;
            width: 100%;
        }

        th, td {
            border: 1px solid black;
            padding: 8px;
            text-align: left;
        }

        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>
    <h1>User Scores</h1>
    <table id="userScoresTable">
        <thead>
            <tr>
                <th>ID</th>
                <th>Username</th>
                <th>Score</th>
            </tr>
        </thead>
        <tbody>
        </tbody>
    </table>

    <script>
        function fetchUserScores() {
```

```
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4 && xhr.status === 200) {
                var scores = JSON.parse(xhr.responseText);
                displayUserScores(scores);
            }
        };
        xhr.open("GET", "http://localhost/cocktail_training/get_user_scores.php", true);
        xhr.send();
    }

    function displayUserScores(scores) {
        var tableBody =
document.getElementById("userScoresTable").getElementsByTagName("tbody")[0];
        tableBody.innerHTML = "";

        for (var i = 0; i < scores.length; i++) {
            var newRow = tableBody.insertRow(i);
            newRow.insertCell(0).innerHTML = scores[i].id;
            newRow.insertCell(1).innerHTML = scores[i].username;
            newRow.insertCell(2).innerHTML = scores[i].score;
        }
    }

    fetchUserScores();
    setInterval(fetchUserScores, 5000); // Update the table every 5 seconds
  </script>
</body>
</html>
```

This HTML file displays the table of user scores fetched from the server. The table is populated using JavaScript to make an AJAX request to the server for the data and it's updated every 5 sec.

Testing

First let’s try to log in with the username:Smith and password:Smith :



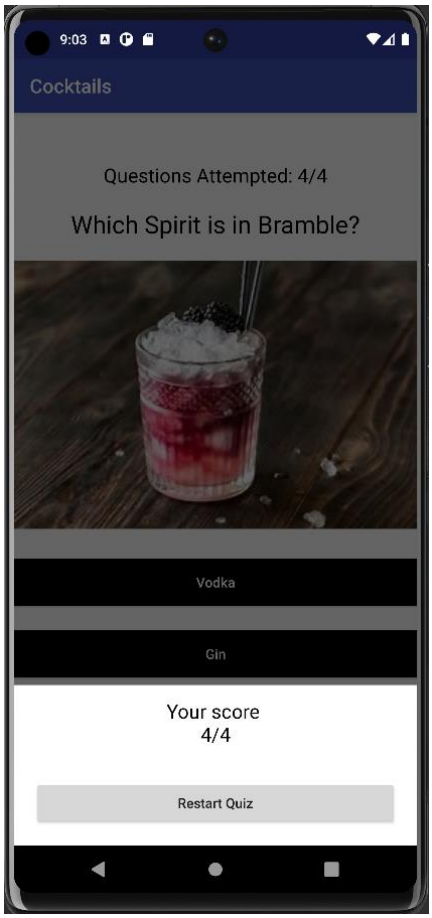
We can observe that we get an error because there is no user with these credentials, if we first press the create user button it will create the user with these credentials and the we can log in and start the training.

Training session:

Now we can choose between 4 options between the spirits that should contain the cocktail in the image.

After completing the training session of 4 cocktails random generated, we get our score.

Score:



When we press the restart quiz button it will restart the quiz and also the username and the score will be sent to the database where it will be inserted in the table, then the table is visualized on the web page.

Web page:

Conclusion:

In conclusion, this application serves as an effective solution for training bartenders remotely by providing an interactive quiz game. By implementing a user-friendly interface, game logic, and server-side functionality to save user scores, the application helps bartenders learn and reinforce their knowledge about cocktails. The combination of Java for Android development, PHP for server-side processing, and HTML, CSS, and JavaScript for web-based user score display ensures a great experience for both the trainees and trainers. This application can be further enhanced by adding more features, such as personalized learning paths, a wider range of questions, or multimedia content to enrich the learning experience.

Overall, this application offers a solid foundation for a comprehensive remote bartender training platform.