QUIZZERR

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

VIMAL KUMAR MISHRA

12111630

CSE 226

ANDROID APP DEPLOYMENT

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month - November Year – 2024

INTRODUCTION

I have developed an engaging quiz application using Android Studio, integrating Firebase for authentication and data management. The app provides a seamless and interactive platform for users to test their knowledge on various topics through a series of well-structured quiz questions.

Firebase Authentication ensures a secure and user-friendly login experience, allowing users to sign up or log in with their credentials. For question storage and management, Firebase Realtime Database (or Firestore) serves as a reliable backend, ensuring efficient retrieval and dynamic updates of quiz content.

With a clean and intuitive interface, this app delivers an enjoyable learning experience while leveraging the power of modern Android development practices and cloud-based solutions.

TECHNICAL REQUIREMENTS

Development Tools:

1.Android Studio (latest stable version).

2.Kotlin as the programming language.

Firebase Integration:

1.Firebase Authentication for user login and signup.

2.Firebase Realtime Database or Firestore for storing and retrieving quiz questions and user data.

Dependencies:

1.Google Firebase SDKs for Android.

2.Required libraries for network operations, such as Retrofit or Volley (if additional APIs are used).

3.UI libraries such as Material Design Components.

Device Requirements

1.Android device or emulator running Android 8.0 or higher.

2.Minimum 2GB RAM for optimal performance.

Functional Requirements

1.User Authentication

2.Users must be able to create an account, log in, and log out securely using Firebase Authentication.

Question Management:

1.Questions and answers must be stored in Firebase.

2.Dynamic fetching of questions for quizzes based on categories.

Quiz Functionality:

1.Display one question at a time with options.

2.Allow users to select answers and navigate between questions.

Scoring and Results:

1.Keep track of the user's score and display results at the end of the quiz.

Non-Functional Requirements

Performance:

1.Quick data retrieval from Firebase for a seamless user experience.

Security:

1.Ensure data transmission is secure using Firebase rules and HTTPS.

User Interface:

1.A clean, intuitive, and visually appealing UI following Material Design guidelines.

Scalability:

1.The app should be able to handle a growing number of users and quiz content without performance degradation.

2.By meeting these requirements, the app ensures a robust, user-friendly, and secure quiz-taking experience.

MODULES EXPLANATION

LOGIN ACTIVITY

Initialization

- View Binding:

    - The activity uses ActivityLoginBinding for binding the UI elements defined in activity_login.xml.
    - This approach eliminates the need for findViewById.
- Firebase Services:
    - FirebaseAuth is initialized to handle user authentication.
    - FirebaseFirestore is initialized to interact with Firestore, enabling user data validation (e.g., email existence).
- ProgressDialog:
    - A loading dialog is created to provide feedback during login attempts or password resets.

---

2. Checking for Active Session

- Auto Login:
    - If a user is already logged in (auth.currentUser != null), they are redirected to the MainActivity without needing to log in again.

---

3. Login Process

- Input Validation:
    - Checks if the email is empty or invalid using Patterns.EMAIL_ADDRESS.
    - Ensures the password field is not empty.
    - Displays appropriate error messages if validation fails.
- Authentication:
    - The auth.signInWithEmailAndPassword method is used to log in users.
    - If the login is successful, the user is redirected to the MainActivity.
    - If unsuccessful, a toast message notifies the user of the failure.

---

4. Password Reset

- Email Verification:
    - The entered email is validated and checked against the Firestore users collection.
    - If the email exists in the database, the auth.sendPasswordResetEmail method sends a reset link to the user’s email.
    - If the email is invalid or doesn't exist, a failure message is displayed.

---

5. Navigation to Signup

- Account Creation:
    - When the "Create New" button is clicked, the app redirects the user to the SignupActivity to create a new account.

---

6. User Experience Features

- Progress Feedback:
    - The ProgressDialog provides a better user experience during potentially long-running tasks like authentication.
- Toast Messages:
    - Used to notify the user about the success or failure of login and password reset actions.

Overall Flow:

1. Login Check: If logged in, redirect to MainActivity.
2. Login Attempt: Validate credentials and authenticate with Firebase.
3. Password Reset: Check email existence in Firestore and send a reset link.
4. Signup Navigation: Redirect to account creation if needed.

This activity is a key component for ensuring secure and user-friendly access to the app.

SIGNUP ACTIVITY

1. Initialization

- View Binding:
    - ActivitySignupBinding links the activity to its corresponding XML layout (activity_signup.xml), ensuring seamless interaction with UI components.
- Firebase Services:
    - FirebaseAuth: Used for authentication, enabling account creation.
    - FirebaseFirestore: Used to store user details like name, email, avatar URL, and referral code.
- ProgressDialog:
    - Displays a loading dialog with the message "We're creating new account..." during the account creation process to improve user experience.

---

2. Input Validation

- Name, Email, and Password Validation:
    - Checks that all fields are filled.
    - Validates the email format using Patterns.EMAIL_ADDRESS.
    - Displays appropriate error messages for invalid or empty fields.

---

3. User Registration

- Creating the User:
    - A User object is created, containing:
        - Name.
        - Email.
        - Default profile picture URL.
        - Referral code (if provided).

- Firebase Authentication:
    - The auth.createUserWithEmailAndPassword method is used to create the user's account in Firebase Authentication.
    - On successful registration, the user's uid is retrieved for storing additional details in Firestore.
- Storing User Data in Firestore:
    - The database.collection("users").document(uid).set(user) method saves the User object in Firestore.
    - On success, the app navigates to the MainActivity.
    - If Firestore storage fails:
        - The user's authentication account is deleted to ensure data consistency.
        - An error message is displayed.

---

4. Navigation to Login

- Login Option:
    - A "Login" button allows users to navigate to LoginActivity if they already have an account.

---

5. Privacy Policy

- Opening Privacy Policy:
    - Clicking on the "Privacy Policy" link triggers an Intent.ACTION_VIEW to open the URL: https://vimalkmgithub.github.io/QuizzerAppPrivacyPolicy/
    - Displays a toast message: "Opening browser..." before launching the browser.

---

6. User Experience Features

- Default Avatar:

    - New users are assigned a default profile picture stored at a public URL.
    - This can later be updated by the user if desired.
- Progress Feedback:
    - The ProgressDialog ensures the user is aware of ongoing background operations.
- Error Handling:
    - Ensures robust handling of failures, including retry options for both account creation and Firestore updates.

Overall Flow

1. Input Validation: Ensures fields are correctly filled.
2. User Creation: Authenticates the user in Firebase and stores their details in Firestore.
3. Error Handling: Deletes partial accounts if any failure occurs.
4. Navigation Options: Includes links to the Login screen and Privacy Policy.

This activity ensures a smooth and user-friendly onboarding process for new users in your Quiz application.

HOME ACTIVITY

1. Initialization

- View Binding:
    - The fragment uses FragmentHomeBinding for a seamless connection between the layout (fragment_home.xml) and the code.
    - A nullable reference (hbinding) ensures proper lifecycle management and avoids memory leaks.
- Firestore Database:

  - The FirebaseFirestore instance is initialized to fetch the quiz categories from the categories collection.

---

2. onCreateView Method

- Inflates the layout for the fragment and assigns it to the binding instance (hbinding).
- Returns the root view of the layout.

---

3. onViewCreated Method

This is where the main functionality is implemented:

- Fetching Data from Firestore:
  - The Firestore addSnapshotListener listens for real-time updates in the categories collection.
  - Each document is mapped to a CategoryModel object using the toObject method.
  - The categoryId is explicitly set using the document ID for further usage (e.g., identifying a category).
- Updating the Adapter:
  - The categories list is cleared and repopulated with the fetched data.
  - The adapter.notifyDataSetChanged() call ensures the UI is updated dynamically.
- Setting Up the RecyclerView:
  - A GridLayoutManager with 2 columns is used to display categories in a grid layout.
  - The CategoryAdapter is initialized with the list of categories and set as the adapter for the RecyclerView.

---

4. onDestroyView Method

- Ensures proper cleanup by setting the binding reference (hbinding) to null when the view is destroyed. This avoids memory leaks associated with fragments.

---

5. Key Components

- Firestore Integration:
  - Retrieves data in real-time using the addSnapshotListener method.
  - Ensures that the app reflects any changes in the categories collection instantly.
- RecyclerView with GridLayoutManager:
  - Provides a visually appealing and structured grid layout for displaying categories.
- Adapter (CategoryAdapter):
  - Manages the data-to-UI binding for each category item.
  - Dynamically updates the UI whenever the category list changes.

---

6. Overall Flow

1. Fragment Initialization: Inflates the UI and initializes Firestore.
2. Data Fetching: Listens for real-time updates in the categories Firestore collection.
3. UI Update: Updates the RecyclerView whenever data changes.
4. Resource Cleanup: Ensures the binding reference is cleared to prevent memory leaks.

Example Use Case

The HomeFragment could display categories like Science, History, Math, etc. Users can tap on a category to navigate to a quiz screen containing questions specific to that category. This dynamic and real-time approach ensures the app stays updated with minimal manual effort.

LEADERBOARD ACTIVITY

The LeaderboardsFragment is responsible for displaying a dynamic leaderboard that ranks users based on their coin count in descending order, creating a competitive experience within the app. It uses FragmentLeaderboardsBinding for seamless interaction with the associated layout, ensuring efficient resource management by setting the binding reference to null in onDestroyView to prevent memory leaks. The fragment fetches user data from the Firestore users collection, utilizing a query that orders the records by the coins field in descending order. The retrieved data is mapped to a list of User objects, which are then passed to a custom LeaderboardsAdapter for display in a RecyclerView. The RecyclerView is configured with a vertical LinearLayoutManager, providing a smooth scrolling experience for the leaderboard. Once the data is successfully loaded, the adapter is notified to refresh the UI, ensuring the leaderboard remains up-to-date. This functionality enables users to view and compare their rankings, fostering engagement and motivation.

<u>SPLASH ACTIVITY</u>

The Splash activity serves as the splash screen of the Quiz application, providing a brief introductory experience to users before navigating to the login screen. On activity creation, it sets the layout using activity_splash.xml and employs a Handler with Looper.getMainLooper() to introduce a delay of 1.5 seconds. After the delay, the app transitions to the LoginActivity using an Intent, ensuring a smooth user experience. The finish() method is called to remove the Splash activity from the back stack, preventing users from navigating back to it. This setup creates a seamless transition from the splash screen to the login process.

<u>PROFILE ACTIVITY</u>

The ProfileFragment is designed to display and manage the user's profile information while providing an option to sign out of the application. It uses FragmentProfileBinding for view binding, ensuring smooth interaction with the UI elements and efficient memory management by nullifying the binding reference in onDestroyView.

Upon view creation, the fragment disables editing for the emailBox, passBox, and nameBox fields, ensuring that these fields are read-only. It retrieves the current user's UID from Firebase Authentication and uses Firestore to fetch the user's details from the users collection. Once the document is retrieved, it populates the emailBox and nameBox fields with the user's email and name, respectively.

The fragment also includes a "Sign Out" button, which, when clicked, signs the user out using Firebase Authentication and redirects them to the LoginActivity. The requireActivity().finish() call ensures that the current activity is closed, preventing users from navigating back to the profile after signing out. This fragment provides a secure and user-friendly way to view and manage profile information while offering seamless sign-out functionality.

QUIZ ACTIVITY

The QuizActivity is the core component of the quiz functionality, responsible for managing the quiz process, including fetching questions from Firestore, displaying them to the user, handling user responses, and calculating the final score. Upon creation, it retrieves a category ID passed via Intent and queries the Firestore database for five random questions from the selected category. If fewer than five questions are available, it fetches additional ones.

A CountDownTimer is used to display a countdown for each question, set to 30 seconds per question. If the timer finishes, it automatically moves to the next question. The user's answers are compared with the correct answer stored in the question object, and feedback is provided by changing the background color of the selected answer to indicate whether it's right or wrong. After each answer, the question counter is updated, and the next question is displayed.

When the user selects an answer, the checkAnswer() method compares the selected answer with the correct one, and the appropriate visual feedback is shown. After all questions have been answered or the user clicks the "Next" button, the total score is calculated, and the app navigates to the ResultActivity to display the score.

Additionally, the QuizActivity provides a "Back to Quiz List" button (quizBtn) that exits the quiz and returns the user to the previous screen, ensuring a smooth user experience.

RESULT ACTIVITY

The ResultActivity displays the quiz results to the user, showing how many questions they answered correctly and the total coins they earned based on their performance. When the activity is created, it retrieves the number of correct answers and the total number of questions from the intent. It then calculates the total coins earned by multiplying the correct answers by a predefined point value (10 coins per correct answer).

The score is displayed in the format "correct/total" on the UI, and the total coins earned are shown as well. The activity also updates the user's coin balance in Firestore by using the FieldValue.increment() method to add the earned coins to their existing coin balance in the database.

A "Restart" button is provided to allow the user to start a new quiz session. When clicked, the app navigates back to the MainActivity and clears the activity stack using finishAffinity() to prevent returning to the result screen. This completes the quiz flow and offers users a seamless transition back to the app's main interface.

WALLET ACTIVITY

The WalletFragment provides a functionality where users can view their current coin balance and make a withdrawal request if they meet the required criteria.

When the fragment is created, it retrieves the user's data from Firestore using their UID (from FirebaseAuth). The User object is fetched and used to display the current coin balance on the UI. The balance is shown in the currentCoins TextView.

The fragment also provides a button, sendRequest, allowing users to initiate a withdrawal request. If the user has more than 50,000 coins, they can input their PayPal email and submit a request for withdrawal. The request is sent to Firestore's withdraws collection, where a WithdrawRequest object is created with the user's PayPal email and name. If the withdrawal request is successful, a Toast message notifies the user.

If the user has fewer than 50,000 coins, a Toast message informs them that they need more coins to be eligible for withdrawal.

This functionality ties into user account management, allowing them to interact with their earnings within the app.

LOCATION ACTIVITY

The Location activity is designed to display a Google Map interface, allowing users to search for locations, view their current location, and interact with the map by clicking on it.

Key Features:

1. **Location Search**:
    - The user can input a location name in the editLocation EditText field. When the search button is pressed, the app attempts to convert the location name into geographical coordinates (latitude and longitude) using Geocoder.
    - If a valid location is found, it displays a marker on the map and zooms into the location. The latitude, longitude, and full address are shown on the UI.
2. **Current Location**:

  - The app can access the user's current location via FusedLocationProviderClient. When the user presses the "Current Location" button, it checks if location services are enabled and requests the necessary permissions.
  - If permissions are granted, the app retrieves the last known location and updates the UI with the user's coordinates and a marker for their current position.
  - If the location is not found or the device’s last known location is unavailable, it requests a new location update.

3. **Map Interaction**:
    - When the user clicks on the map, the app captures the clicked position (LatLng), retrieves the address corresponding to that position, and adds a marker on the map. The address, latitude, and longitude are displayed on the UI.
4. **Map Type Options**:
    - The app offers options to change the map’s appearance. The user can toggle between Satellite, Terrain, Hybrid, and Normal map types using corresponding buttons. The map’s view is updated according to the selected map type.
5. **UI Updates**:
    - The latitude and longitude are displayed in text1 and text2 respectively. The address of the location (whether searched or clicked) is displayed in the addressEditText.
6. **Location Permissions**:
    - The app checks for location permissions (ACCESS_FINE_LOCATION) before accessing the device's location. If permissions are missing, it requests them.
7. **Location Services Check**:
    - Before attempting to access the user's location, the app ensures that the location services (GPS or network provider) are enabled on the device.

Structure and Flow:

- The app initializes a map using SupportMapFragment and attaches an OnMapClickListener to handle user interactions on the map.

- It includes buttons for searching locations, getting the current location, and toggling map types.
- The app uses Geocoder for converting between addresses and geographical coordinates.

This implementation allows for a rich, interactive map experience where users can search for places, view their current location, and interact with the map to get information about specific locations.

CODE

CategoryAdapter

```
package com.example.quizpr

import android.content.Context
import android.content.Intent
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide

class CategoryAdapter(
    private val context: Context, private val categoryModels: ArrayList<CategoryModel>
) : RecyclerView.Adapter<CategoryAdapter.CategoryViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CategoryViewHolder {
        val view = LayoutInflater.from(context).inflate(R.layout.item_category, parent, false)
        return CategoryViewHolder(view)
    }

    override fun onBindViewHolder(holder: CategoryViewHolder, position: Int) {
        if (categoryModels.isEmpty()) return
        val model = categoryModels[position]
        holder.textView.text = model.categoryName
        Glide.with(context).load(model.categoryImage).into(holder.imageView)
        holder.itemView.setOnClickListener {
            val intent = Intent(context, QuizActivity::class.java)
            intent.putExtra("catId", model.categoryId)
            context.startActivity(intent)
        }
    }

    override fun getItemCount(): Int {
        return categoryModels.size
    }
    inner class CategoryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val imageView: ImageView = itemView.findViewById(R.id.image)
        val textView: TextView = itemView.findViewById(R.id.category)
    }
}
```

HomeFragment

```
package com.example.quizpr

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.GridLayoutManager
import com.example.quizpr.databinding.FragmentHomeBinding
import com.google.firebase.firestore.FirebaseFirestore

class HomeFragment : Fragment() {
    private var hbinding: FragmentHomeBinding? = null
    private val binding get() = hbinding!!
    private lateinit var database: FirebaseFirestore
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        hbinding = FragmentHomeBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        database = FirebaseFirestore.getInstance()
        val categories = ArrayList<CategoryModel>()
        val adapter = CategoryAdapter(requireContext(), categories)
        database.collection("categories").addSnapshotListener { value, _ ->
                categories.clear()
                value?.let {
                    for (snapshot in it.documents) {
                        val model = snapshot.toObject(CategoryModel::class.java)
                        model?.categoryId = snapshot.id
                        model?.let { categoryModel -> categories.add(categoryModel) }
                    }
                    adapter.notifyDataSetChanged()
                }
            }
        binding.categoryList.layoutManager = GridLayoutManager(requireContext(), 2)
        binding.categoryList.adapter = adapter
    }

    override fun onDestroyView() {
        super.onDestroyView()
        hbinding = null
    }
}
```

LeaderboardsAdapter

```
package com.example.quizpr

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.quizpr.databinding.RowLeaderboardsBinding

class LeaderboardsAdapter(private val context: Context, private val users: ArrayList<User>) :
  RecyclerView.Adapter<LeaderboardsAdapter.LeaderboardViewHolder>() {
  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LeaderboardViewHolder {
    val view = LayoutInflater.from(context).inflate(R.layout.row_leaderboards, parent, false)
    return LeaderboardViewHolder(view)
  }

  override fun onBindViewHolder(holder: LeaderboardViewHolder, position: Int) {
    val user = users[position]
    holder.binding.name.text = user.name
    holder.binding.coins.text = user.coins.toString()
    "#${position + 1}".also { holder.binding.index.text = it }
    Glide.with(context).load(user.profile).into(holder.binding.imageView7)
  }

  override fun getItemCount(): Int {
    return users.size
  }

  inner class LeaderboardViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val binding: RowLeaderboardsBinding = RowLeaderboardsBinding.bind(itemView)
  }
}
```

LeaderboardsFragment

```
package com.example.quizpr

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.quizpr.databinding.FragmentLeaderboardsBinding
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.Query

class LeaderboardsFragment : Fragment() {
  private var lbinding: FragmentLeaderboardsBinding? = null
  private val binding get() = lbinding!!
  override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        lbinding = FragmentLeaderboardsBinding.inflate(inflater, container, false)
        val database = FirebaseFirestore.getInstance()
        val users = ArrayList<User>()
        val adapter = LeaderboardsAdapter(requireContext(), users)
        binding.recyclerView.adapter = adapter
        binding.recyclerView.layoutManager = LinearLayoutManager(requireContext())
        database.collection("users").orderBy("coins", Query.Direction.DESCENDING).get()
            .addOnSuccessListener { queryDocumentSnapshots ->
                for (snapshot in queryDocumentSnapshots) {
                    val user = snapshot.toObject(User::class.java)
                    users.add(user)
                }
                adapter.notifyDataSetChanged()
            }
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        lbinding = null
    }
}
```

Location

```
package com.example.quizpr

import android.Manifest
import android.content.pm.PackageManager
import android.location.Address
import android.location.Geocoder
import android.location.LocationManager
import android.os.Bundle
import android.os.Looper
import android.view.View
import android.widget.EditText
import android.widget.ImageButton
import android.widget.LinearLayout
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import com.google.android.gms.location.*
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import com.google.android.material.floatingactionbutton.FloatingActionButton

class Location : AppCompatActivity(), GoogleMap.OnMapClickListener {
    private lateinit var googleMap: GoogleMap
    private lateinit var editLocation: EditText
    private lateinit var addressEditText: EditText
    private lateinit var text1: TextView
    private lateinit var text2: TextView
    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private lateinit var locationCallback: LocationCallback
    private var currentZoomLevel: Float = 17f
    private lateinit var satelliteButton: ImageButton
    private lateinit var terrainButton: ImageButton
    private lateinit var hybridButton: ImageButton
    private lateinit var normalButton: ImageButton
    override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_location)
       fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
       val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as SupportMapFragment
       mapFragment.getMapAsync { googleMap ->
          this.googleMap = googleMap
          googleMap.setOnMapClickListener(this)
       }
       editLocation = findViewById(R.id.editTextText)
       addressEditText = findViewById(R.id.addressEditText)
       text1 = findViewById(R.id.textView)
       text2 = findViewById(R.id.textView2)
       val searchButton: ImageButton = findViewById(R.id.button)
       searchButton.setOnClickListener {
          val locationName = editLocation.text.toString().trim()
          if (locationName.isEmpty()) {
             Toast.makeText(this, "Enter a location to search", Toast.LENGTH_SHORT).show()
          } else {
             addressEditText.text.clear()
             getLocationFromAddress(locationName)
          }
       }
       val getLocationButton: ImageButton = findViewById(R.id.currentLocationIcon)
       getLocationButton.setOnClickListener {
          getCurrentLocation()
       }
       val fab: FloatingActionButton = findViewById(R.id.floatingActionButton)
       val mapOptionsLayout: LinearLayout = findViewById(R.id.mapOptionsLayout)
       fab.setOnClickListener {
          mapOptionsLayout.visibility =
             if (mapOptionsLayout.visibility == View.GONE) View.VISIBLE else View.GONE
       }
       satelliteButton = findViewById(R.id.satelliteButton)
       satelliteButton.setOnClickListener {
          googleMap.mapType = GoogleMap.MAP_TYPE_SATELLITE
       }
       terrainButton = findViewById(R.id.terrainButton)
       terrainButton.setOnClickListener {
          googleMap.mapType = GoogleMap.MAP_TYPE_TERRAIN
       }
       hybridButton = findViewById(R.id.hybridButton)
       hybridButton.setOnClickListener {
          googleMap.mapType = GoogleMap.MAP_TYPE_HYBRID
       }
```

```
    normalButton = findViewById(R.id.normalButton)
    normalButton.setOnClickListener {
        googleMap.mapType = GoogleMap.MAP_TYPE_NORMAL
    }
}

override fun onMapClick(latLng: LatLng) {
    currentZoomLevel = googleMap.cameraPosition.zoom
    getAddressFromLatLng(latLng)
}

private fun getLocationFromAddress(location: String) {
    val geocoder = Geocoder(this)
    try {
        val addresses: List<Address> = geocoder.getFromLocationName(location, 1) ?: emptyList()
        if (addresses.isNotEmpty()) {
            val address = addresses[0]
            val latLng = LatLng(address.latitude, address.longitude)
            googleMap.clear()
            googleMap.addMarker(MarkerOptions().position(latLng).title(location))
            val cameraUpdate = CameraUpdateFactory.newLatLngZoom(latLng, 17f)
            googleMap.animateCamera(cameraUpdate, 1500, null) // Duration: 1.5 seconds
            text1.text = "Latitude: ${address.latitude}"
            text2.text = "Longitude: ${address.longitude}"
            addressEditText.setText(address.getAddressLine(0))
        } else {
            Toast.makeText(this, "Address not found", Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception) {
        Toast.makeText(this, "Error fetching location", Toast.LENGTH_SHORT).show()
    }
}

private fun getAddressFromLatLng(latLng: LatLng) {
    val geocoder = Geocoder(this)
    try {
        val addresses = geocoder.getFromLocation(latLng.latitude, latLng.longitude, 1)
        if (addresses != null && addresses.isNotEmpty()) {
            val address = addresses[0]
            addressEditText.setText(address.getAddressLine(0))
            text1.text = "Latitude: ${latLng.latitude}"
            text2.text = "Longitude: ${latLng.longitude}"
            googleMap.clear()
            googleMap.addMarker(
                MarkerOptions().position(latLng).title(address.getAddressLine(0))
            )
            googleMap.animateCamera(
                CameraUpdateFactory.newLatLngZoom(
                    latLng,
                    currentZoomLevel
                )
            )
        } else {
            Toast.makeText(this, "No address found", Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception) {
        Toast.makeText(this, "Error getting address", Toast.LENGTH_SHORT).show()
    }
}

private fun getCurrentLocation() {
    if (!isLocationEnabled()) {
        Toast.makeText(this, "Enable location services", Toast.LENGTH_SHORT).show()
        return
    }
    if (ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            1
        )
        return
    }
    fusedLocationClient.lastLocation.addOnSuccessListener { location ->
        if (location != null) {
            val latLng = LatLng(location.latitude, location.longitude)
            updateUI(latLng)
        } else {
            requestNewLocation()
        }
    }
}

private fun requestNewLocation() {
    val locationRequest = LocationRequest.create().apply {
        interval = 10000
        fastestInterval = 5000
        priority = LocationRequest.PRIORITY_HIGH_ACCURACY
    }
    locationCallback = object : LocationCallback() {
        override fun onLocationResult(locationResult: LocationResult) {
            val location = locationResult.lastLocation
            if (location != null) {
                val latLng = LatLng(location.latitude, location.longitude)
                updateUI(latLng)
            }
        }
    }
    if (ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
            return
        }
        fusedLocationClient.requestLocationUpdates(
            locationRequest,
            locationCallback,
            Looper.getMainLooper()
        )
    }

    private fun updateUI(latLng: LatLng) {
        googleMap.clear()
        googleMap.addMarker(MarkerOptions().position(latLng).title("Current Location"))
        val cameraUpdate = CameraUpdateFactory.newLatLngZoom(latLng, 17f)
        googleMap.animateCamera(cameraUpdate, 1500, null)
        text1.text = "Latitude: ${latLng.latitude}"
        text2.text = "Longitude: ${latLng.longitude}"
        val geocoder = Geocoder(this)
        val addresses = geocoder.getFromLocation(latLng.latitude, latLng.longitude, 1)
        if (addresses?.isNotEmpty() == true) {
            addressEditText.setText(addresses[0].getAddressLine(0))
        } else {
            Toast.makeText(this, "No address found", Toast.LENGTH_SHORT).show()
        }
    }

    private fun isLocationEnabled(): Boolean {
        val locationManager = getSystemService(LOCATION_SERVICE) as LocationManager
        return locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER) ||
                locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER)
    }
}
```

LoginActivity

```
package com.example.quizpr

import android.app.ProgressDialog
import android.content.Intent
import android.os.Bundle
import android.util.Patterns
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.quizpr.databinding.ActivityLoginBinding
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore

class LoginActivity : AppCompatActivity() {
    private lateinit var binding: ActivityLoginBinding
    private lateinit var auth: FirebaseAuth
    private lateinit var dialog: ProgressDialog
    private lateinit var firestore: FirebaseFirestore
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)
```

```
auth = FirebaseAuth.getInstance()
firestore = FirebaseFirestore.getInstance()
dialog = ProgressDialog(this)
dialog.setMessage("Logging in...")
dialog.setCancelable(false)
if (auth.currentUser != null) {
    startActivity(Intent(this@LoginActivity, MainActivity::class.java))
    finish()
}
binding.submitBtn.setOnClickListener {
    val email = binding.emailBox.text.toString().trim()
    val pass = binding.passwordBox.text.toString()
    if (email.isEmpty()) {
        binding.emailBox.error = "Enter email..."
    } else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        binding.emailBox.error = "Enter a valid email..."
    } else if (pass.isEmpty()) {
        binding.passwordBox.error = "Enter password..."
    } else {
        dialog.show()
        auth.signInWithEmailAndPassword(email, pass).addOnCompleteListener { task ->
            dialog.dismiss()
            if (task.isSuccessful) {
                startActivity(Intent(this@LoginActivity, MainActivity::class.java))
                finish()
            } else {
                Toast.makeText(
                    this@LoginActivity, "Failed try again...", Toast.LENGTH_SHORT
                ).show()
            }
        }
    }
}
binding.resetPasswd.setOnClickListener {
    val email = binding.emailBox.text.toString().trim()
    if (email.isEmpty()) {
        binding.emailBox.error = "Enter email..."
    } else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        binding.emailBox.error = "Enter a valid email..."
    } else {
        firestore.collection("users").whereEqualTo("email", email).get()
            .addOnSuccessListener { documents ->
                if (!documents.isEmpty) {
                    auth.sendPasswordResetEmail(email).addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            Toast.makeText(
                                this@LoginActivity,
                                "Password reset link sent...",
                                Toast.LENGTH_SHORT
                            ).show()
                        } else {
                            Toast.makeText(
                                this@LoginActivity,
                                "Failed try again...",
                                Toast.LENGTH_SHORT
                    ).show()
                  }
                }
              } else {
                dialog.dismiss()
                Toast.makeText(
                  this@LoginActivity, "Failed try again...", Toast.LENGTH_SHORT
                ).show()
              }
            }
        }
      }
      binding.createNewBtn.setOnClickListener {
        startActivity(Intent(this@LoginActivity, SignupActivity::class.java))
        finish()
      }
    }
}
```

MainActivity

```
package com.example.quizpr

import android.content.Intent
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.FragmentTransaction
import com.example.quizpr.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
  private lateinit var binding: ActivityMainBinding

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    // Setup Toolbar
    setSupportActionBar(binding.toolbar)

    // Setup Navigation Drawer
    val toggle = ActionBarDrawerToggle(
      this,
      binding.drawerLayout,
      binding.toolbar,
      R.string.navigation_drawer_open,
      R.string.navigation_drawer_close
    )
    binding.drawerLayout.addDrawerListener(toggle)
    toggle.drawerArrowDrawable.color = resources.getColor(android.R.color.white, theme)
    toggle.syncState()
```

```
        // Load Default Fragment
        supportFragmentManager.beginTransaction().replace(R.id.content, HomeFragment()).commit()

        // Bottom Navigation Bar Handling
        binding.bottomBar.setOnItemSelectedListener {
            val transaction = supportFragmentManager.beginTransaction()
            when (it) {
                0 -> transaction.replace(R.id.content, HomeFragment())
                1 -> transaction.replace(R.id.content, LeaderboardsFragment())
                2 -> transaction.replace(R.id.content, WalletFragment())
                3 -> transaction.replace(R.id.content, ProfileFragment())
            }
            transaction.commit()
        }

        // Drawer Navigation Handling
        binding.navigationView.setNavigationItemSelectedListener { menuItem ->
            val transaction = supportFragmentManager.beginTransaction()
            when (menuItem.itemId) {
                R.id.nav_location -> startActivity(Intent(this@MainActivity, Location::class.java))
            }
            transaction.commit()
            binding.drawerLayout.closeDrawer(binding.navigationView)
            true
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.home_menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.wallet) {
            val transaction = supportFragmentManager.beginTransaction()
            transaction.replace(R.id.content, WalletFragment())
            transaction.commit()
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onBackPressed() {
        if (binding.drawerLayout.isDrawerOpen(binding.navigationView)) {
            binding.drawerLayout.closeDrawer(binding.navigationView)
        } else {
            super.onBackPressed()
        }
    }
}
```

ProfileFragment

```
package com.example.quizpr

import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.example.quizpr.databinding.FragmentProfileBinding
import com.google.firebase.Firebase
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.auth
import com.google.firebase.firestore.FirebaseFirestore

class ProfileFragment : Fragment() {
    private var pbinding: FragmentProfileBinding? = null
    private val binding get() = pbinding!!
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        pbinding = FragmentProfileBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        binding.emailBox.isEnabled = false
        binding.passBox.isEnabled = false
        binding.nameBox.isEnabled = false

        super.onViewCreated(view, savedInstanceState)
        val userUid = FirebaseAuth.getInstance().currentUser?.uid
        val users = FirebaseFirestore.getInstance().collection("users")
        userUid?.let { uid ->
            users.document(uid).get()
                .addOnSuccessListener { documentSnapshot ->
                    if (documentSnapshot.exists()) {
                        binding.emailBox.setText(documentSnapshot["email"].toString())
                        binding.nameBox.setText(documentSnapshot["name"].toString())
                    }
                }
        }
        binding.signOutBtn.setOnClickListener {
            Firebase.auth.signOut()
            val intent = Intent(requireContext(), LoginActivity::class.java)
            startActivity(intent)
            requireActivity().finish()
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        pbinding = null
    }
}
```

<u>QuizActivity</u>

```
package com.example.quizpr

import android.content.Intent
import android.os.Bundle
import android.os.CountDownTimer
import android.view.View
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import com.example.quizpr.databinding.ActivityQuizBinding
import com.google.firebase.firestore.FirebaseFirestore
import java.util.*
import kotlin.collections.ArrayList

class QuizActivity : AppCompatActivity() {
    private lateinit var binding: ActivityQuizBinding
    private var questions = ArrayList<Question>()
    private lateinit var question: Question
    private lateinit var timer: CountDownTimer
    private lateinit var database: FirebaseFirestore
    private var correctAnswers = 0
    private var index = 0
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityQuizBinding.inflate(layoutInflater)
        setContentView(binding.root)
        database = FirebaseFirestore.getInstance()
        val catId = intent.getStringExtra("catId")
        val random = Random()
        val rand = random.nextInt(5)
        database.collection("categories").document(catId!!).collection("questions")
            .whereGreaterThanOrEqualTo("index", rand).orderBy("index").limit(5).get()
            .addOnSuccessListener { queryDocumentSnapshots ->
                if (queryDocumentSnapshots.documents.size < 5) {
                    database.collection("categories").document(catId).collection("questions")
                        .whereLessThanOrEqualTo("index", rand).orderBy("index").limit(5).get()
                        .addOnSuccessListener { querySnapshot ->
                            for (snapshot in querySnapshot) {
                                val question = snapshot.toObject(Question::class.java)
                                questions.add(question)
                            }
                            setNextQuestion()
                        }
                } else {
                    for (snapshot in queryDocumentSnapshots) {
                        val question = snapshot.toObject(Question::class.java)
                        questions.add(question)
                    }
                    setNextQuestion()
                }
            }
        resetTimer()
    }

    private fun resetTimer() {
        timer = object : CountDownTimer(30000, 1000) {
```

```
        override fun onTick(millisUntilFinished: Long) {
            binding.timer.text = (millisUntilFinished / 1000).toString()
        }

        override fun onFinish() {
            index++
            if (index < questions.size) {
                setNextQuestion()
            } else {
                timer.cancel()
                val intent = Intent(this@QuizActivity, ResultActivity::class.java)
                intent.putExtra("correct", correctAnswers)
                intent.putExtra("total", questions.size)
                startActivity(intent)
                finish()
            }
        }
    }
}

private fun showAnswer() {
    when (question.answer) {
        binding.option1.text -> binding.option1.setBackgroundResource(R.drawable.option_right)
        binding.option2.text -> binding.option2.setBackgroundResource(R.drawable.option_right)
        binding.option3.text -> binding.option3.setBackgroundResource(R.drawable.option_right)
        binding.option4.text -> binding.option4.setBackgroundResource(R.drawable.option_right)
    }
}

private fun setNextQuestion() {
    if (::timer.isInitialized) timer.cancel()
    timer.start()
    if (index < questions.size) {
        binding.questionCounter.text = String.format("%d/%d", index + 1, questions.size)
        question = questions[index]
        binding.question.text = question.question
        binding.option1.text = question.option1
        binding.option2.text = question.option2
        binding.option3.text = question.option3
        binding.option4.text = question.option4
    } else {
        timer.cancel()
        val intent = Intent(this@QuizActivity, ResultActivity::class.java)
        intent.putExtra("correct", correctAnswers)
        intent.putExtra("total", questions.size)
        startActivity(intent)
        finish()
    }
}

private fun checkAnswer(textView: TextView) {
    val selectedAnswer = textView.text.toString()
    if (selectedAnswer == question.answer) {
        correctAnswers++
        textView.setBackgroundResource(R.drawable.option_right)
        } else {
            showAnswer()
            textView.setBackgroundResource(R.drawable.option_wrong)
        }
    }

    private fun reset() {
        binding.option1.setBackgroundResource(R.drawable.option_unselected)
        binding.option2.setBackgroundResource(R.drawable.option_unselected)
        binding.option3.setBackgroundResource(R.drawable.option_unselected)
        binding.option4.setBackgroundResource(R.drawable.option_unselected)
    }

    fun onClick(view: View) {
        when (view.id) {
            R.id.option_1, R.id.option_2, R.id.option_3, R.id.option_4 -> {
                val selected = view as TextView
                checkAnswer(selected)
            }

            R.id.nextBtn -> {
                reset()
                if (index < questions.size - 1) {
                    index++
                    setNextQuestion()
                } else {
                    timer.cancel()
                    val intent = Intent(this@QuizActivity, ResultActivity::class.java)
                    intent.putExtra("correct", correctAnswers)
                    intent.putExtra("total", questions.size)
                    startActivity(intent)
                    finish()
                }
            }

            R.id.quizBtn -> {
                finish()
            }
        }
    }
}
```

ResultActivity

```
package com.example.quizpr

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.example.quizpr.databinding.ActivityResultBinding
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FieldValue
import com.google.firebase.firestore.FirebaseFirestore

class ResultActivity : AppCompatActivity() {
  private lateinit var binding: ActivityResultBinding
  private val points = 10
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityResultBinding.inflate(layoutInflater)
    setContentView(binding.root)
    val correctAnswers = intent.getIntExtra("correct", 0)
    val totalQuestions = intent.getIntExtra("total", 0)
    val points = correctAnswers * points.toLong()
    binding.score.text = String.format("%d/%d", correctAnswers, totalQuestions)
    binding.earnedCoins.text = points.toString()
    val database = FirebaseFirestore.getInstance()
    database.collection("users").document(FirebaseAuth.getInstance().uid!!)
      .update("coins", FieldValue.increment(points))
    binding.restartBtn.setOnClickListener {
      startActivity(Intent(this@ResultActivity, MainActivity::class.java))
      finishAffinity()
    }
  }
}
```

SignupActivity

```
package com.example.quizpr

import android.app.ProgressDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.util.Patterns
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.quizpr.databinding.ActivitySignupBinding
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore

class SignupActivity : AppCompatActivity() {
  private lateinit var binding: ActivitySignupBinding
  private lateinit var auth: FirebaseAuth
  private lateinit var database: FirebaseFirestore
  private lateinit var dialog: ProgressDialog
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivitySignupBinding.inflate(layoutInflater)
    setContentView(binding.root)
    auth = FirebaseAuth.getInstance()
    database = FirebaseFirestore.getInstance()
    dialog = ProgressDialog(this)
    dialog.setMessage("We're creating new account...")
    dialog.setCancelable(false)
    binding.createNewBtn.setOnClickListener {
      val email = binding.emailBox.text.toString().trim()
      val pass = binding.passwordBox.text.toString()
      val name = binding.nameBox.text.toString()
      val referCode = binding.referBox.text.toString()
```

```
if (name.isEmpty()) {
    binding.nameBox.error = "Enter name..."
} else if (email.isEmpty()) {
    binding.emailBox.error = "Enter email..."
} else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
    binding.emailBox.error = "Enter a valid email..."
} else if (pass.isEmpty()) {
    binding.passwordBox.error = "Enter password..."
} else {
    val user = User(
        name,
        email,
        "https://cdn.vectorstock.com/i/500p/53/42/user-member-avatar-face-profile-icon-vector-22965342.jpg",
        referCode
    )
    dialog.show()
    auth.createUserWithEmailAndPassword(email, pass).addOnCompleteListener { task ->
        if (task.isSuccessful) {
            val uid = task.result?.user?.uid
            uid?.let {
                database.collection("users").document(it).set(user)
                    .addOnCompleteListener { dbTask ->
                        if (dbTask.isSuccessful) {
                            dialog.dismiss()
                            startActivity(
                                Intent(
                                    this@SignupActivity, MainActivity::class.java
                                )
                            )
                            finish()
                        } else {
                            dialog.dismiss()
                            auth.currentUser?.delete()
                            Toast.makeText(
                                this@SignupActivity,
                                "Failed try again...",
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }.addOnFailureListener {
                        dialog.dismiss()
                        auth.currentUser?.delete()
                        Toast.makeText(
                            this@SignupActivity,
                            "Failed try again...",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
            }
        } else {
            dialog.dismiss()
            Toast.makeText(
                this@SignupActivity, "Failed try again...", Toast.LENGTH_SHORT
            ).show()
        }
```

```
                }
            }
        }
        binding.loginBtn.setOnClickListener {
            startActivity(Intent(this@SignupActivity, LoginActivity::class.java))
            finish()
        }
        binding.policy.setOnClickListener {
            Toast.makeText(
                this@SignupActivity, "Opening browser...", Toast.LENGTH_SHORT
            ).show()
            val intent = Intent(
                Intent.ACTION_VIEW,
                Uri.parse("https://vimalkmgithub.github.io/QuizzerAppPrivacyPolicy/")
            )
            startActivity(intent)
        }
    }
}
```

Splash

```
package com.example.quizpr

import android.content.Intent
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import androidx.appcompat.app.AppCompatActivity

class Splash : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)
        Handler(Looper.getMainLooper()).postDelayed({
            startActivity(Intent(this@Splash, LoginActivity::class.java))
            finish()
        }, 1500)
    }
}
```

WalletFragment

```
package com.example.quizpr

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.quizpr.databinding.FragmentWalletBinding
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
```

```
class WalletFragment : Fragment() {
   private lateinit var binding: FragmentWalletBinding
   private lateinit var database: FirebaseFirestore
   private lateinit var user: User
   override fun onCreateView(
      inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
   ): View {
      binding = FragmentWalletBinding.inflate(inflater, container, false)
      database = FirebaseFirestore.getInstance()
      database.collection("users").document(FirebaseAuth.getInstance().uid!!).get()
         .addOnSuccessListener { documentSnapshot ->
            user = documentSnapshot.toObject(User::class.java)!!
            binding.currentCoins.text = user.coins.toString()
         }
      binding.sendRequest.setOnClickListener {
         if (user.coins > 50000) {
            val uid = FirebaseAuth.getInstance().uid
            val payPal = binding.emailBox.text.toString()
            val request = WithdrawRequest(payPal, user.name)
            if (uid != null) {
               database.collection("withdraws").document(uid).set(request)
                  .addOnSuccessListener {
                     Toast.makeText(
                        context, "Request sent successfully...", Toast.LENGTH_SHORT
                     ).show()
                  }
            }
         } else {
            Toast.makeText(context, "You need more coins to withdraw...", Toast.LENGTH_SHORT)
               .show()
         }
      }
      return binding.root
   }
}
```

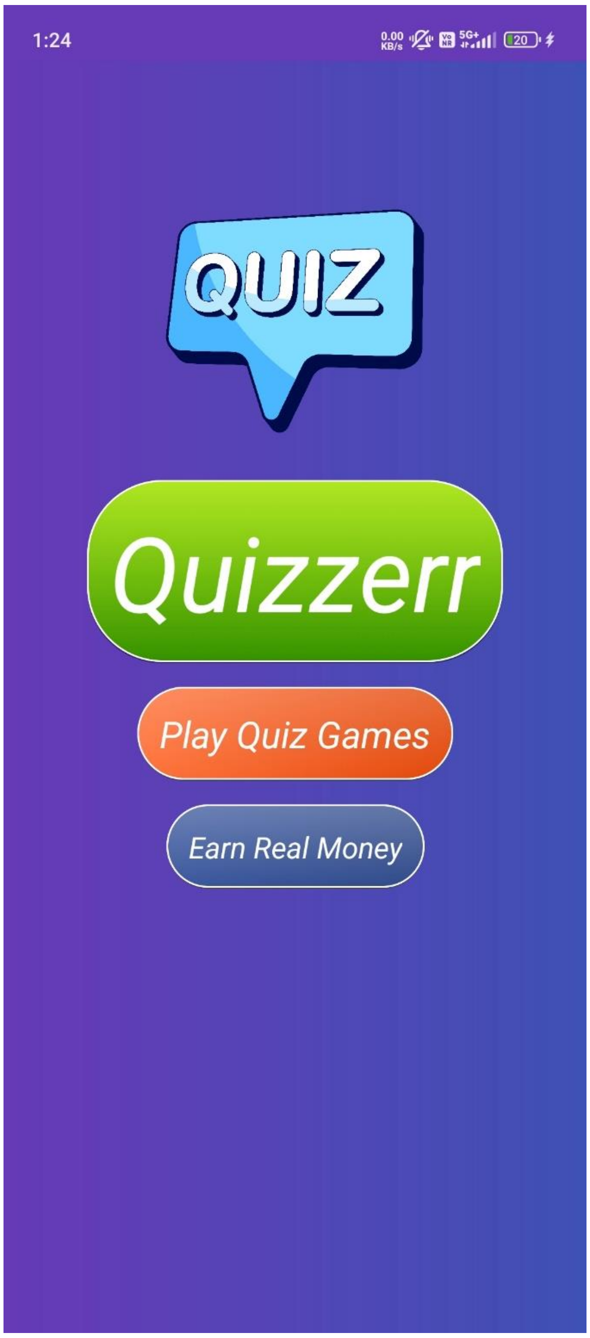
MODEL FILES

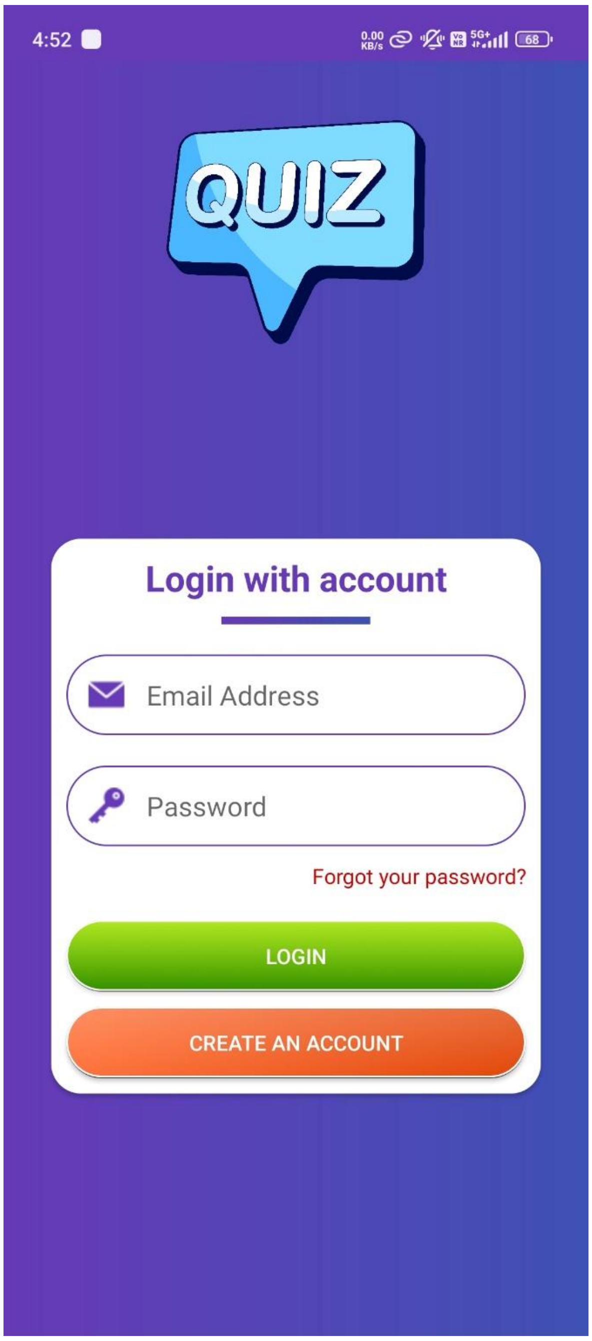
```
package com.example.quizpr

data class CategoryModel(
   var categoryId: String = "", var categoryName: String = "", var categoryImage: String = ""
)
```

```
package com.example.quizpr

data class Question(
   var question: String = "",
   var option1: String = "",
   var option2: String = "",
   var option3: String = "",
   var option4: String = "",
    var answer: String = ""
)



package com.example.quizpr

data class User(
    var name: String = "",
    var email: String = "",
    var profile: String = "",
    var referCode: String = "",
    var coins: Long = 25
)



package com.example.quizpr

import com.google.firebase.firestore.ServerTimestamp
import java.util.Date

data class WithdrawRequest(
    var emailAddress: String = "", var requestedBy: String = ""
) {
    @ServerTimestamp
    var createdAt: Date? = null
}
```

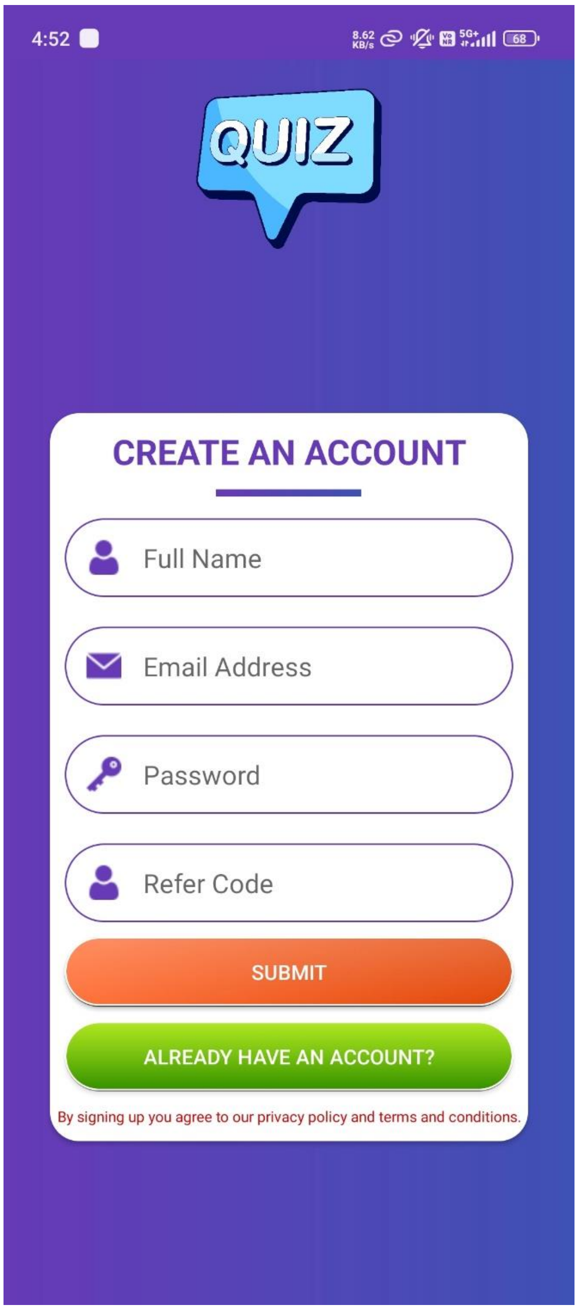
SCREENSHOTS

1. Splash Screen ***2. Login Screen***





3. Registration Screen



4:52

CREATE AN ACCOUNT

Full Name

Email Address

Password

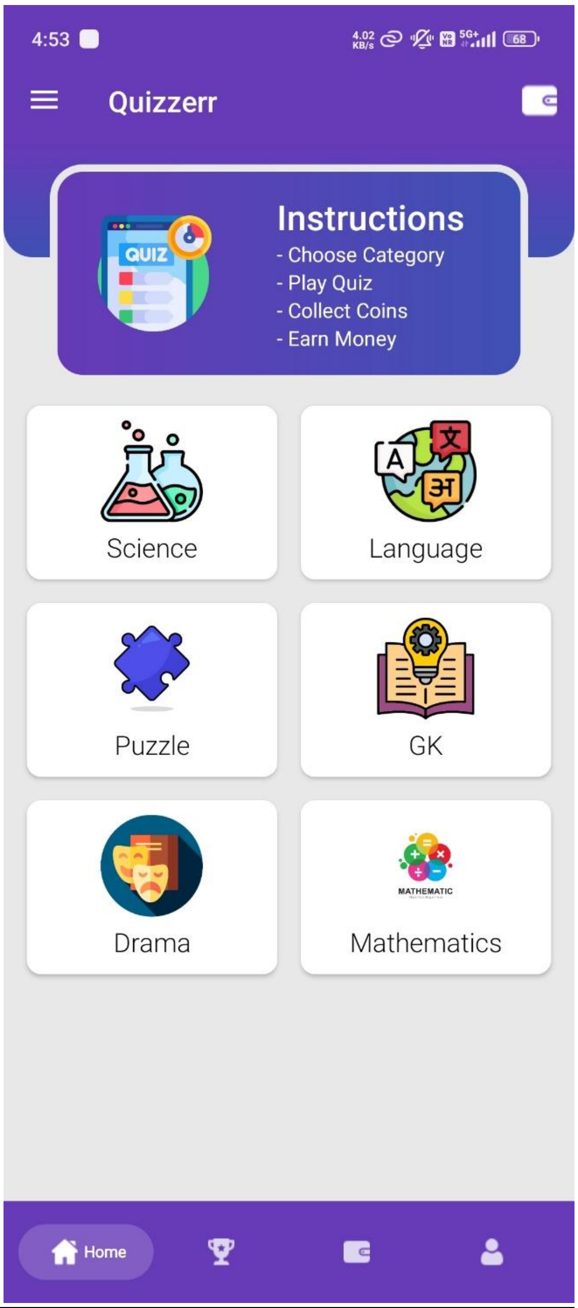
Refer Code

SUBMIT

ALREADY HAVE AN ACCOUNT?

By signing up you agree to our privacy policy and terms and conditions.

4. Home Screen



4:53

Quizzerr

Instructions
- Choose Category
- Play Quiz
- Collect Coins
- Earn Money

Science

Language

Puzzle

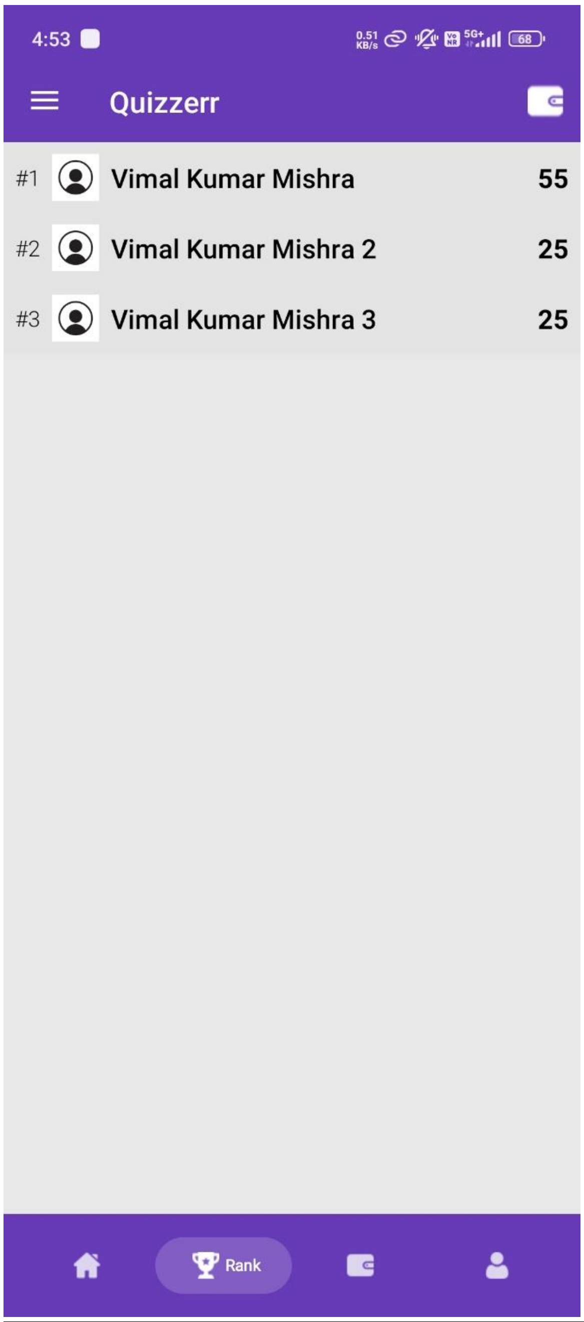
GK

Drama

Mathematics

Home

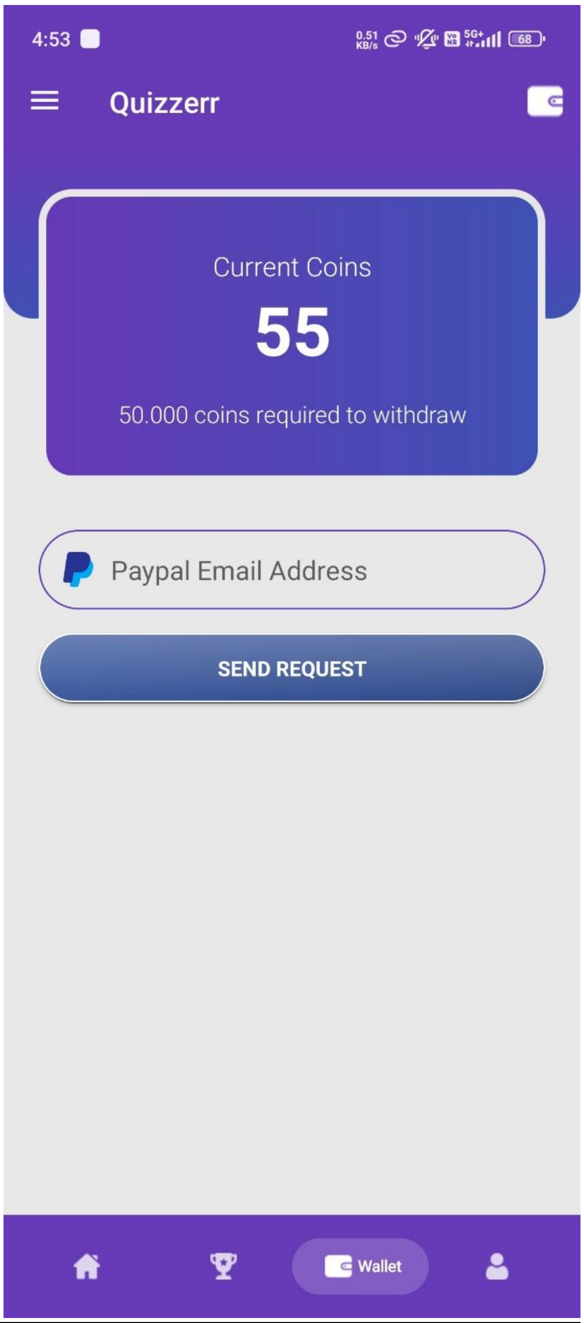
5. Rank Screen


4:53

Quizzerr

#1	Vimal Kumar Mishra	55
#2	Vimal Kumar Mishra 2	25
#3	Vimal Kumar Mishra 3	25

6. Wallet Screen


4:53

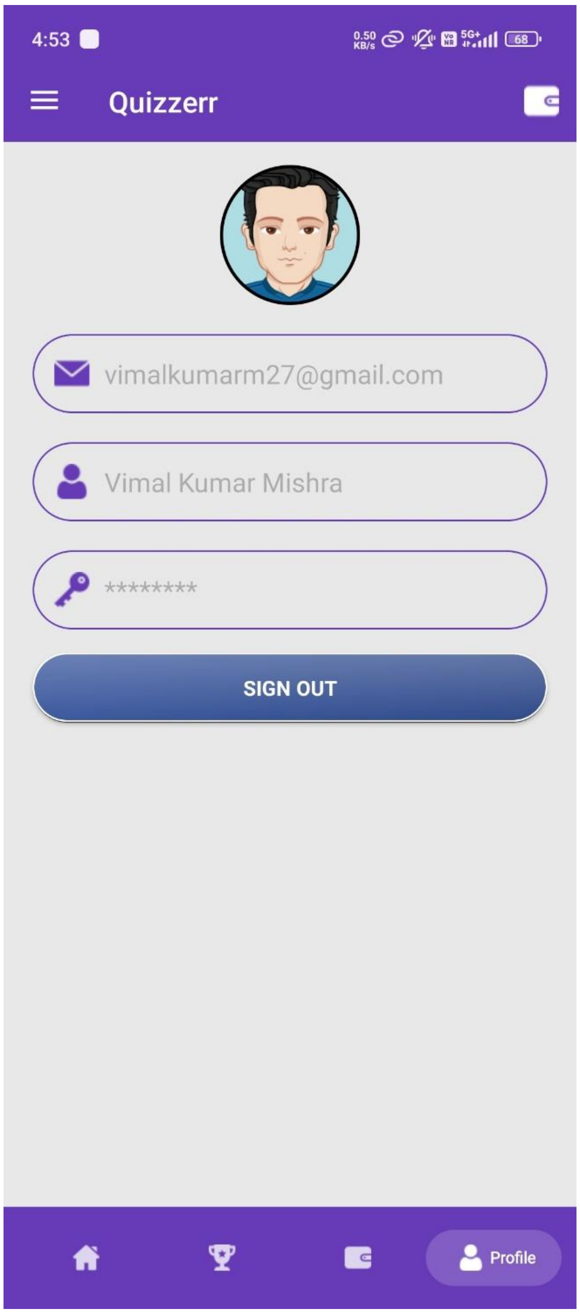
Quizzerr

Current Coins

55

50.000 coins required to withdraw

7. Profile Screen



4:53

Quizzerr

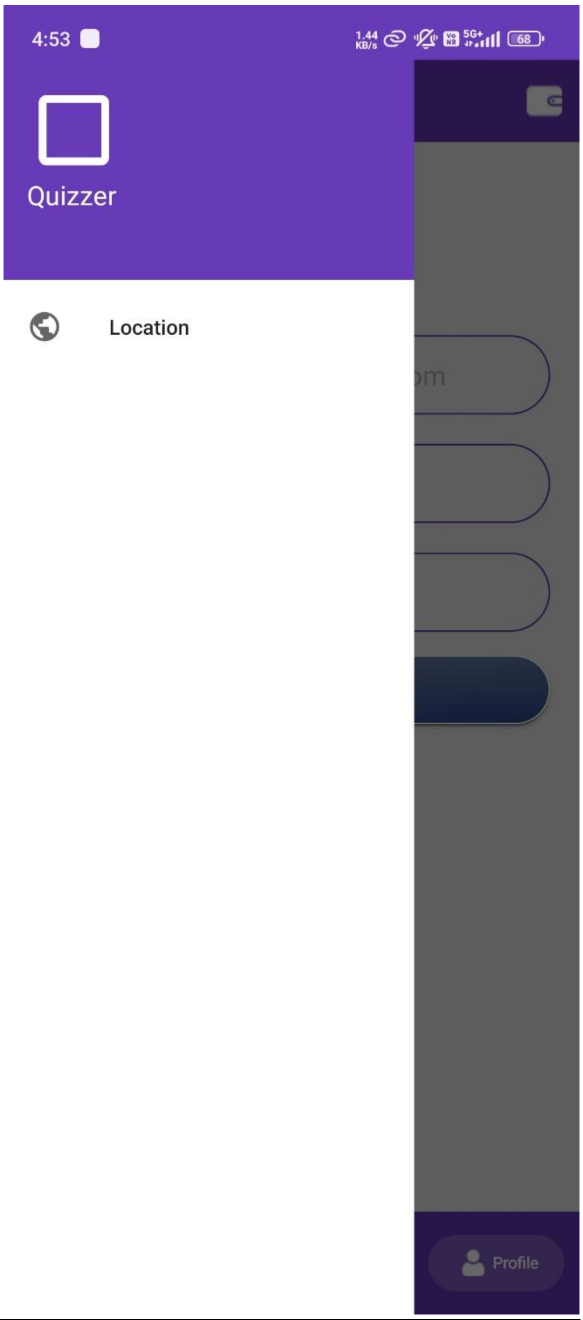
vimalkumarm27@gmail.com

Vimal Kumar Mishra

****

SIGN OUT

8. Navigation Drawer Screen

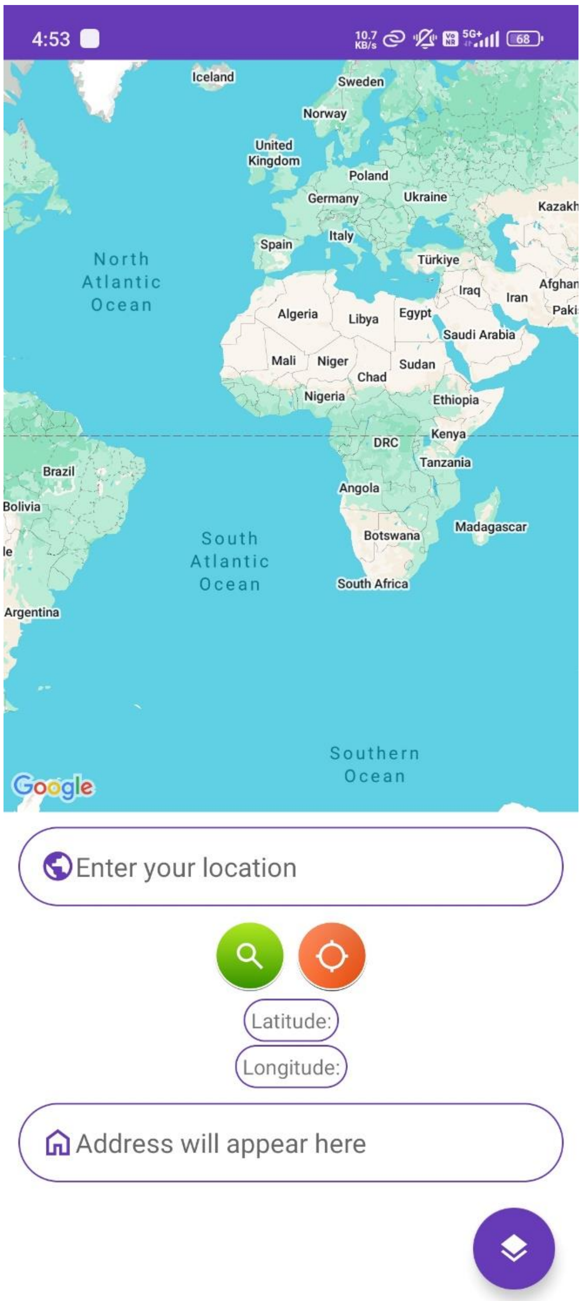


4:53

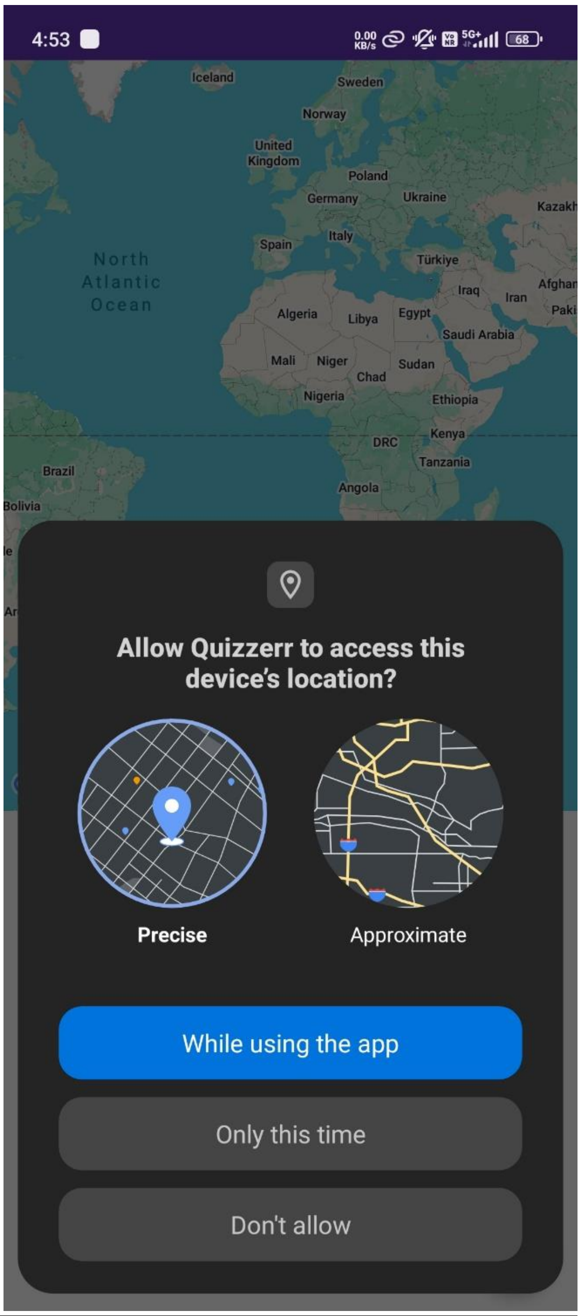
Quizzer

Location

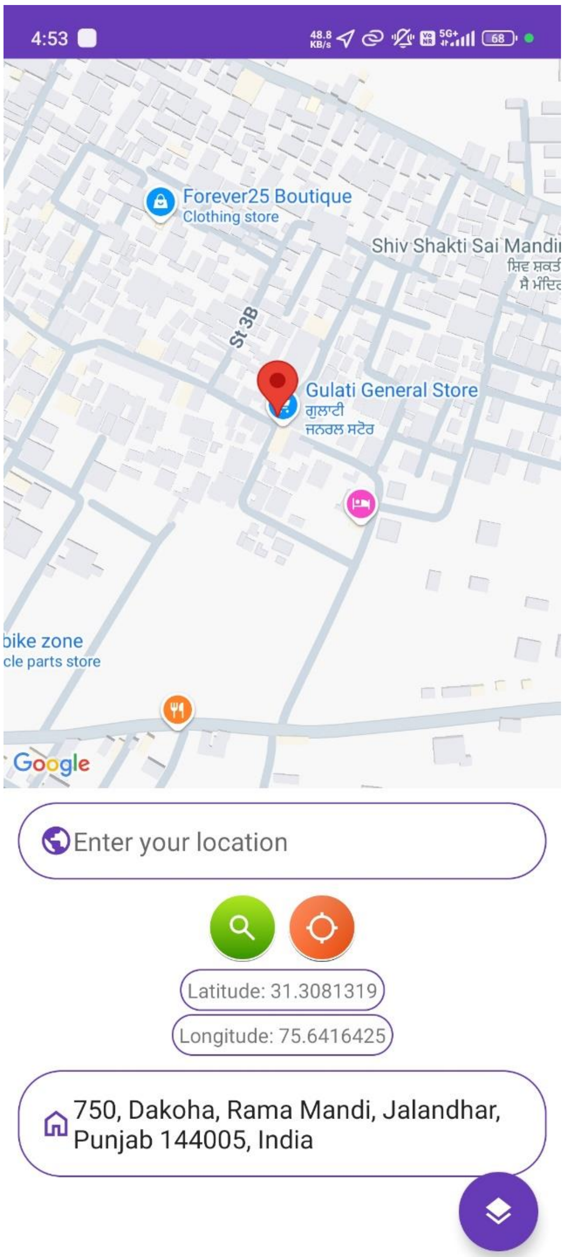
9. Map landing Screen



10. Asking for permission



11. Fetching location from GPS default view

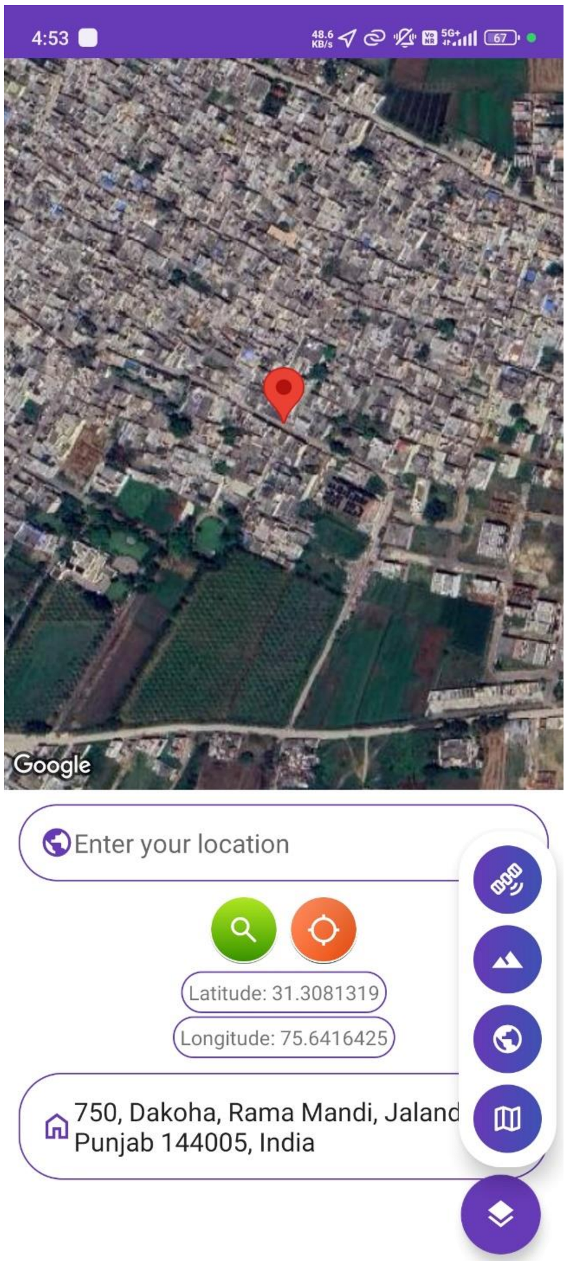


Latitude: 31.3081319

Longitude: 75.6416425

750, Dakoha, Rama Mandi, Jalandhar, Punjab 144005, India

12. Change to satellite view using floating action button

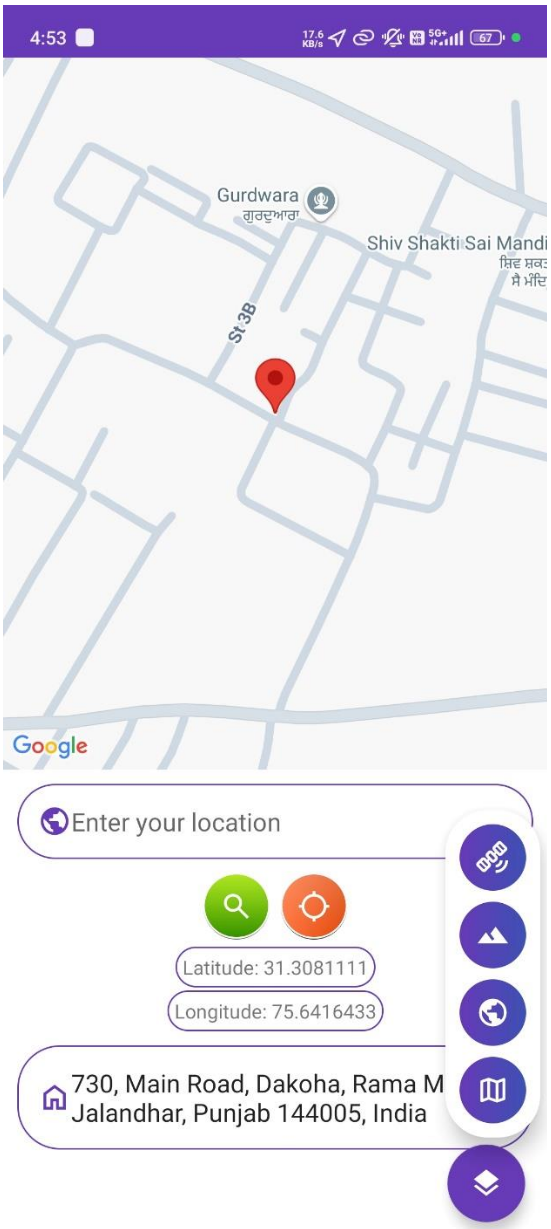


Latitude: 31.3081319

Longitude: 75.6416425

750, Dakoha, Rama Mandi, Jaland
Punjab 144005, India

13. Terrain view

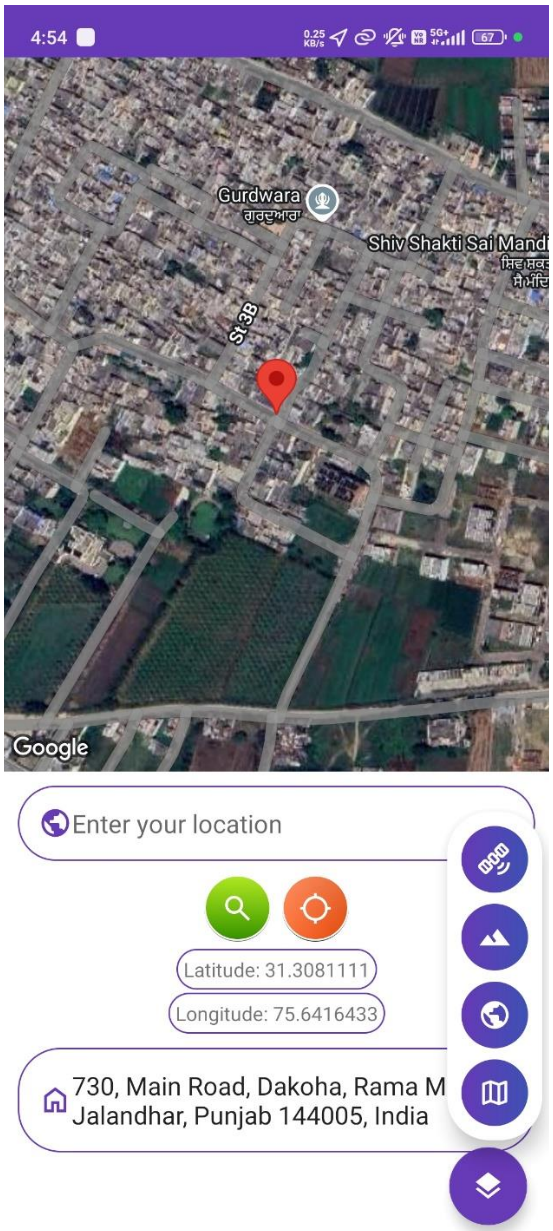


Latitude: 31.3081111

Longitude: 75.6416433

730, Main Road, Dakoha, Rama M
Jalandhar, Punjab 144005, India

14. Mixed view

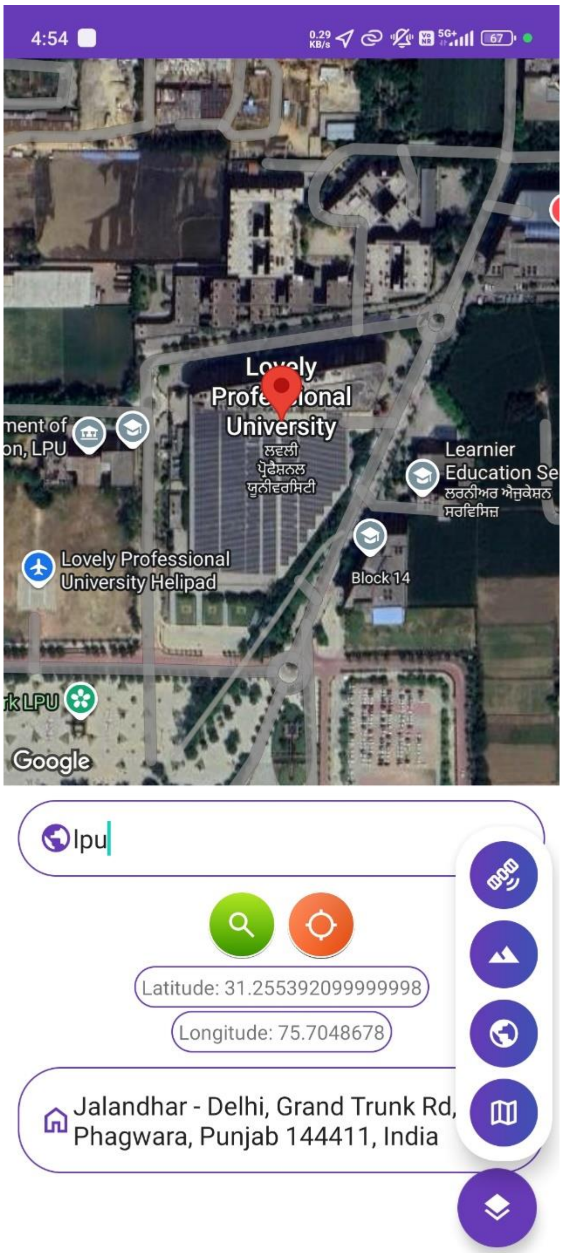


Latitude: 31.3081111

Longitude: 75.6416433

730, Main Road, Dakoha, Rama M
Jalandhar, Punjab 144005, India

15. Searching address



4:54
Lovely Professional University
ਲਵਲੀ ਪ੍ਰੋਫੈਸ਼ਨਲ ਯੂਨੀਵਰਸਿਟੀ
ment of on, LPU
Learnier Education Se
ਲਰਨੀਅਰ ਐਜੂਕੇਸ਼ਨ ਸਰਵਿਸਿਜ਼
Lovely Professional University Helipad
Block 14
k LPU
Google

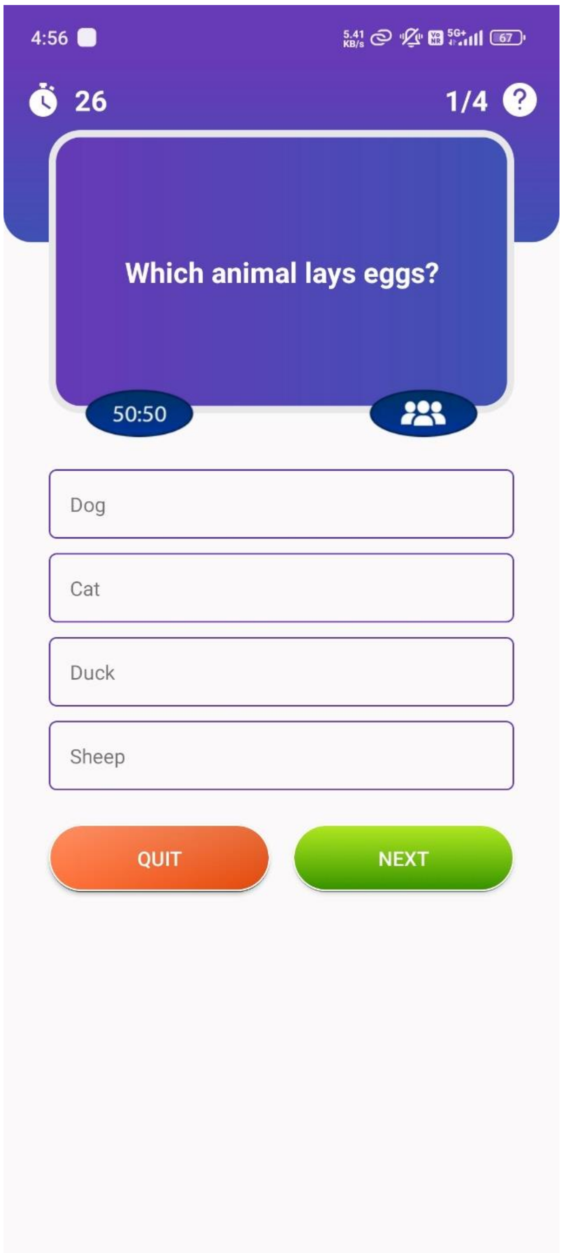
lpu

Latitude: 31.255392099999998

Longitude: 75.7048678

Jalandhar - Delhi, Grand Trunk Rd, Phagwara, Punjab 144411, India

16. Question activity



Which animal lays eggs?

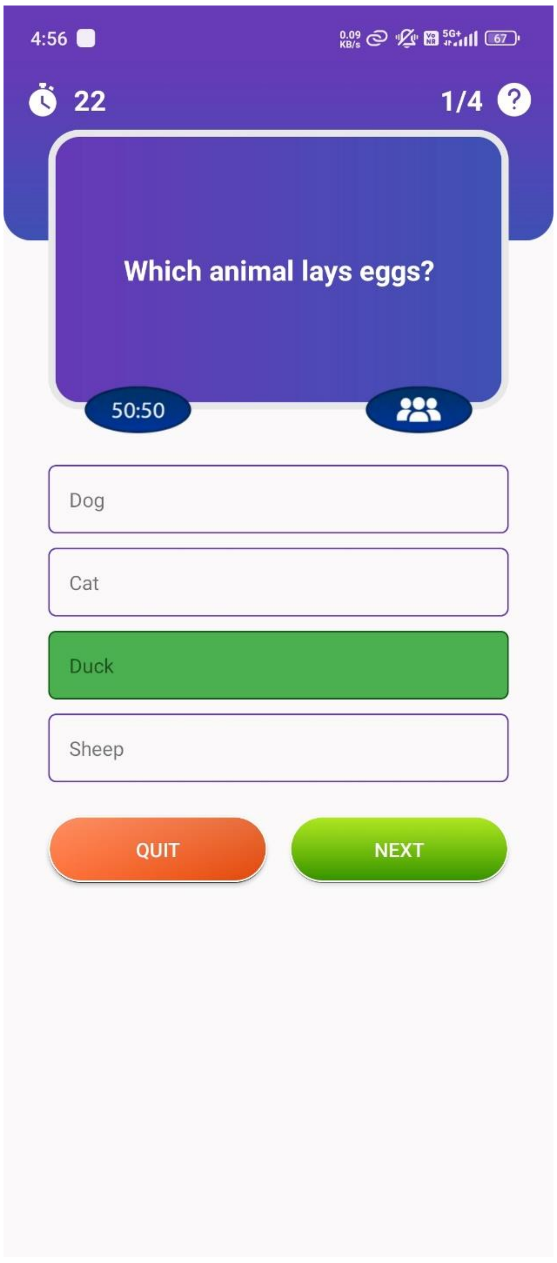
Dog

Cat

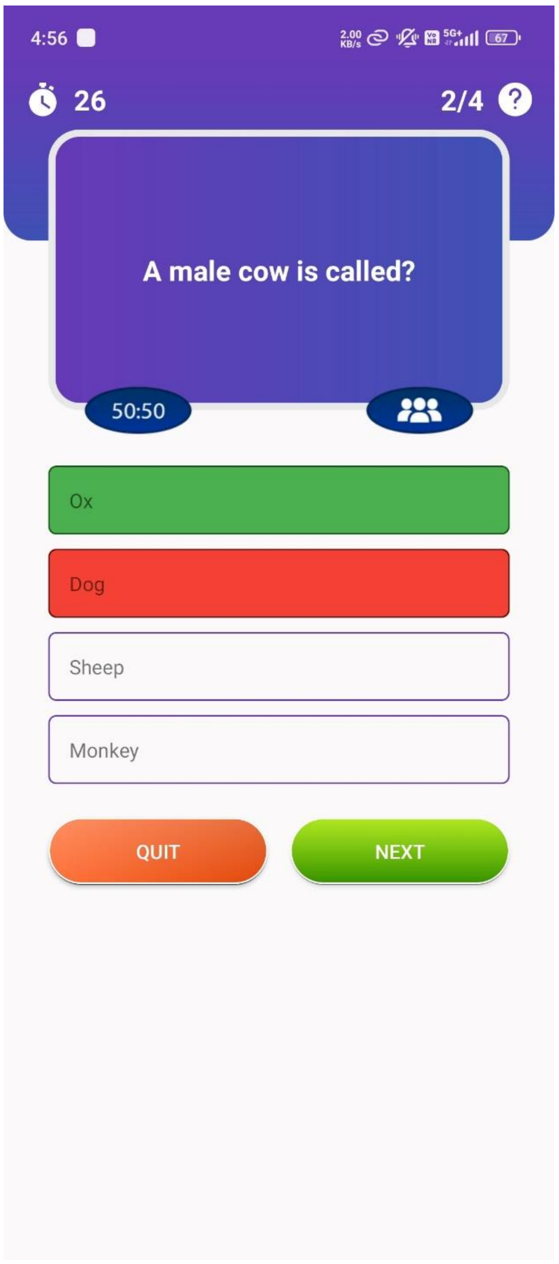
Duck

Sheep

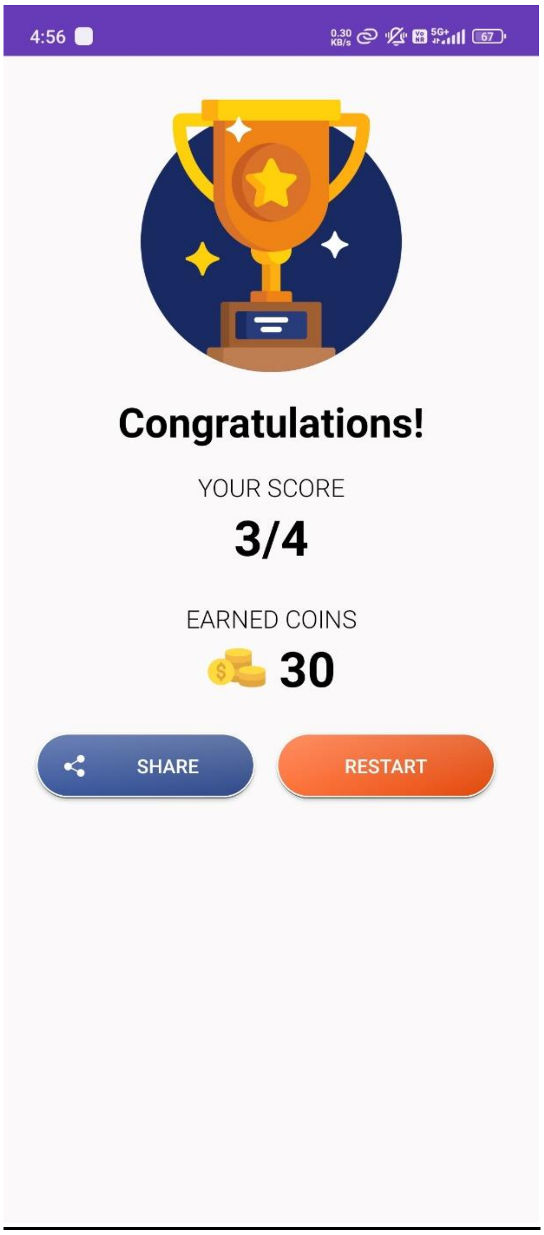
15. Right answer



16. Wrong answer along with right answer



15. Result activity



Conclusion & Future Scope

The Location activity successfully integrates Google Maps to provide a dynamic and interactive experience for users. Key functionalities include the ability to search for locations, display the user's current location, and allow users to interact with the map by clicking on specific locations. With the integration of various map types (Satellite, Terrain, Hybrid, and Normal), users can view the map according to their preference. The app uses the Geocoder class to convert between geographical coordinates and addresses, ensuring that both the user and the map can operate seamlessly. Furthermore, the app ensures that location permissions are requested and location services are enabled, providing a smooth user experience.

Key Takeaways:

- Location Search: The app allows users to search for a specific location by name and displays the corresponding coordinates and address on the map.
- Current Location: Users can access their real-time location, view it on the map, and see relevant details like latitude, longitude, and address.
- Map Interaction: Users can click anywhere on the map to view information related to that location, including latitude, longitude, and address.
- Map Customization: Users can toggle between different map types, enhancing the flexibility and user experience.

Future Scope:

1. Enhanced Search Functionality:
    - Implement autocomplete functionality for the location search, suggesting places as the user types, making it more user-friendly and faster.
    - Provide more filters in the search results, such as distance-based sorting or adding different categories (e.g., restaurants, hospitals).
2. User Location Sharing:
    - Add functionality to share the user's current location or a searched location via social media, email, or messaging apps.
    - Enable users to share the location coordinates or address with others, allowing for collaborative map usage.
3. Location-Based Services:

    - Integrate location-based services such as nearest restaurants, ATMs, gas stations, or popular landmarks, leveraging the Google Places API or other location-based services.
    - Implement real-time traffic information and route planning to improve the navigation experience.
4. Enhanced Map Features:
    - Implement a feature that allows users to save favorite locations, create custom maps, or mark locations for later reference.
    - Allow users to measure distances between two points on the map or calculate estimated travel times based on different transportation modes.
5. Augmented Reality (AR) Integration:
    - Integrate augmented reality (AR) to show real-time location-based data overlaid on the user's camera view, providing a more immersive map experience.
6. Offline Support:
    - Implement offline map functionality, where users can download maps for offline use, making the app useful in areas with poor or no internet connectivity.
    - Store previously searched locations and enable users to access them even without an internet connection.
7. User Data Integration:
    - Integrate with a user account system (e.g., Firebase or Google authentication) to store user preferences, location history, and saved places across devices.
    - Provide personalized location recommendations based on the user’s past searches or preferences.
8. Improved UI/UX:
    - Enhance the app’s UI/UX design, making it more intuitive and visually appealing by adding smooth animations, responsive layouts, and modern UI elements.
    - Optimize the layout for different screen sizes and devices, ensuring that the app works well on phones, tablets, and potentially wearables.

Technological Enhancements:

- Explore integrating Machine Learning models to provide smart location recommendations based on user behavior or context (e.g., predicting the next location they might want to search based on historical data).
- Utilize Cloud-Based Mapping Services to enhance the performance of map features, especially when dealing with large amounts of geographic data.

By implementing these improvements, the app can become even more feature-rich and versatile, providing an excellent tool for location-based tasks while ensuring a seamless and intuitive user experience.

GITHUB LINK

https://github.com/VimalKMGithub/Quizzerr