DYNAMIC QUIZ APP

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 225

ANDROID APP DEVELOPMENT

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month - April Year - 2024

INTRODUCTION

The advent of mobile applications has revolutionized the way we interact with technology, offering seamless access to information, entertainment, and services at our fingertips. In this digital age, the development of robust and user-friendly mobile applications has become paramount in meeting the evolving needs of users worldwide.

This report documents the creation of an Android application designed to provide an engaging and interactive experience for users. The application encompasses various features including authentication, dynamic content delivery, and user data management, all powered by Firebase services.

At its core, the application aims to offer users an enriching experience through a combination of quizzes, personalized profiles, and seamless navigation. Leveraging Firebase for authentication, Firestore for real-time database management, and Firebase Storage for efficient storage solutions, the application ensures scalability, reliability, and security.

The report provides a comprehensive overview of the application architecture, design considerations, implementation details, and the utilization of Firebase services. Additionally, it delves into the challenges encountered during development, the strategies employed to overcome them, and insights gained throughout the process.

Through this report, readers will gain valuable insights into the development lifecycle of a modern Android application, as well as the integration of Firebase services to enhance functionality, performance, and user experience.

ACTIVITIES EXPLANATION

SPLASH ACTIVITY

The Splash screen, represented by the `Splash` activity, serves as the introductory interface of the QuizPR application, providing users with a visually appealing entry point while essential components initialize in the background. Let's delve into its functionality to understand its purpose.

Activity Definition: `Splash` is an activity subclassing `AppCompatActivity`, indicating its role as a screen within the application's navigation flow.

onCreate() Method: Upon creation, the `onCreate()` method sets the stage for the Splash screen's presentation. It inflates the layout defined in `activity_splash.xml`, initializing the visual components displayed to the user during the splash screen transition.

Delayed Transition: A `Handler` with a delay of 1500 milliseconds (1.5 seconds) is employed to orchestrate the transition from the Splash screen to the `LoginActivity`. This delay enhances the user experience, allowing sufficient time for the splash screen to be displayed before proceeding to the next activity.

Intent Navigation: After the delay, an explicit intent is used to navigate from the Splash activity to the LoginActivity. This transition marks the completion of the splash screen's purpose, guiding users to the login interface seamlessly.

Finish Activity: Upon initiating the intent navigation, the `finish()` method is called to ensure that the Splash activity is terminated, optimizing memory usage and ensuring a smooth transition to the subsequent activity.

In essence, the Splash screen serves as the welcoming gateway to the QuizPR application, offering users a brief yet captivating glimpse into the app's interface before embarking on their interactive journey. Its implementation reflects the application's commitment to providing a polished and engaging user experience from the moment of entry.

MAIN ACTIVITY

In the realm of Android app development, the MainActivity serves as the entry point and orchestrator of various functionalities within the application. It sets the tone for user interaction, navigation, and menu options, offering a cohesive experience.

Package and Imports: The package declaration and import statements lay the foundation for accessing necessary classes and resources throughout the activity's lifecycle.

Class Definition: MainActivity is at the helm, inheriting from AppCompatActivity, a crucial component for modern Android applications that ensures compatibility with older versions of Android while offering the latest features.

Variable Declaration: Within the MainActivity, the declaration of the 'binding' variable of type ActivityMainBinding exemplifies the utilization of View Binding, a contemporary approach to accessing UI elements in Android development.

onCreate() Method: This method serves as the genesis, triggered when the activity is instantiated. It initializes the layout, configures the toolbar, and seamlessly integrates the HomeFragment into the UI using a FragmentTransaction, ensuring a smooth transition into the app's core functionality.

onCreateOptionsMenu() Method: As users navigate through the app, their interaction extends to menu options. This method orchestrates the creation of the options menu, inflating its layout from home_menu.xml, thus empowering users with additional functionality right from the toolbar.

onOptionsItemSelected() Method: When users interact with menu items, their actions resonate within this method. Here, the app reacts dynamically to user input, responding to the selection of the 'wallet' menu item by smoothly transitioning to the WalletFragment, enriching the user experience.

Bottom Navigation Item Selection: In the pursuit of user-centric design, the MainActivity embraces bottom navigation as a means of intuitive exploration. Through setOnItemSelectedListener, users are empowered to seamlessly navigate between different fragments, from home to leaderboards, wallets, and profiles, fostering a fluid and engaging user journey.

In essence, the MainActivity epitomizes the nexus of user interaction and app functionality, laying the groundwork for a seamless and immersive user experience within the Android application ecosystem.

LOGIN ACTIVITY

In the vibrant landscape of Android app development, the LoginActivity stands as a gateway, facilitating user authentication and access to the application's features. Let's dissect its components and elucidate its functionality.

Package and Imports: The foundation is laid with the package declaration and necessary imports, harnessing the power of external libraries and resources to streamline the authentication process.

Class Definition: LoginActivity emerges as a pivotal subclass of AppCompatActivity, embodying the essence of user authentication within the application.

Variable Declaration: Within the LoginActivity, key variables are initialized to orchestrate the authentication process seamlessly. Instances of FirebaseAuth and FirebaseFirestore are created to harness the capabilities of Firebase Authentication and Firestore database services. Additionally, a ProgressDialog instance is instantiated to provide visual feedback to users during the authentication process.

onCreate() Method: The genesis of the LoginActivity, onCreate() method sets the stage for user authentication. It inflates the layout defined in activity_login.xml, initializes Firebase authentication and Firestore instances, and configures a ProgressDialog to indicate login progress. Furthermore, it checks if a user is already authenticated. If so, it directs them to the MainActivity, bypassing the login screen.

Handling User Login: The login process is facilitated by the submitBtn click listener. It retrieves the user-entered email and password, validates them, and attempts to authenticate the user using FirebaseAuth. Upon successful authentication, the user is redirected to the MainActivity. In case of authentication failure, a Toast message notifies the user to try again.

Password Reset: Users are empowered to reset their passwords by clicking on the resetPasswd button. Upon validation of the entered email, the application queries the Firestore database to verify the existence of the user. If the user exists, a password reset email is sent via FirebaseAuth. Appropriate Toast messages inform the user of the outcome.

Creating a New Account: The LoginActivity extends a warm invitation to new users, directing them to the SignupActivity upon clicking the createNewBtn. This fosters a seamless onboarding experience, enabling users to effortlessly transition from exploration to engagement within the application.

In essence, the LoginActivity serves as the cornerstone of user authentication, embodying the application's commitment to security, accessibility, and user-centric design.

SIGNUP ACTIVITY

Within the dynamic realm of Android application development, the SignupActivity emerges as a pivotal component, facilitating user registration and account creation. Let's delve into its intricacies to understand its functionality.

Package and Imports: The foundation is laid with the package declaration and necessary imports, harnessing the power of external libraries and resources to streamline the user registration process.

Class Definition: SignupActivity epitomizes user onboarding within the application, serving as a crucial subclass of AppCompatActivity.

Variable Declaration: Key variables are initialized to orchestrate the user registration process seamlessly. Instances of FirebaseAuth and FirebaseFirestore are created to leverage the capabilities of Firebase Authentication and Firestore database services. Additionally, a ProgressDialog instance is instantiated to provide visual feedback to users during the account creation process.

onCreate() Method: The genesis of the SignupActivity, onCreate() method sets the stage for user registration. It inflates the layout defined in activity_signup.xml, initializes Firebase authentication and Firestore instances, and configures a ProgressDialog to indicate account creation progress.

User Registration: The createNewBtn click listener initiates the user registration process. It retrieves the user-entered name, email, password, and referral code, validates them, and attempts to create a new user account using FirebaseAuth. Upon successful account creation, user details are stored in the Firestore database. Appropriate Toast messages inform the user of the outcome.

Login Redirection: Users are provided with a seamless transition to the LoginActivity upon clicking the loginBtn, fostering a cohesive user journey within the application.

Privacy Policy: Users are empowered with transparency and informed decision-making through access to the privacy policy. The policy click listener opens the device's browser, directing users to the specified URL for detailed policy information.

In essence, the SignupActivity serves as a gateway to user engagement, embodying the application's commitment to user-centric design, security, and transparency in user interactions.

QUIZ ACTIVITY

In the realm of educational engagement and user interactivity, the QuizActivity takes center stage, providing users with an immersive quiz experience. Let's dissect its components and elucidate its functionality.

Package and Imports: The groundwork is laid with the package declaration and necessary imports, harnessing external resources to bolster the quiz functionality.

Class Definition: QuizActivity emerges as the epicenter of quiz engagement, standing as a crucial subclass of AppCompatActivity.

Variable Declaration: Crucial variables are initialized to orchestrate the quiz process seamlessly. Instances of FirebaseFirestore are created to leverage the capabilities of Firestore database services. Additional variables like 'questions', 'question', 'timer', 'correctAnswers', and 'index' serve as the backbone of the quiz mechanics.

onCreate() Method: The inception of the QuizActivity, onCreate() method sets the stage for quiz engagement. It inflates the layout defined in activity_quiz.xml, initializes Firestore instances, and dynamically fetches quiz questions based on the selected category. Furthermore, it resets the timer for each question.

Reset Timer: A countdown timer is initiated for each question, ensuring timely progression within the quiz.

Show Answer: In case of an incorrect answer, the correct answer is highlighted, providing users with immediate feedback.

setNextQuestion(): This method facilitates the transition to the next question within the quiz. It updates the UI with the next question's details and options, ensuring a seamless quiz experience.

checkAnswer(): User-selected answers are validated against the correct answer, updating the score and providing visual feedback.

Reset(): Option backgrounds are reset to their default state after each question, ensuring a clean UI for the next question.

onClick() Method: User interactions with quiz options and buttons are handled dynamically through this method. Whether selecting an option, moving to the next question, or exiting the quiz, user actions are seamlessly interpreted and processed.

In essence, the QuizActivity serves as the pinnacle of user engagement, fostering interactive learning experiences and providing users with a platform to test their knowledge in a dynamic and immersive manner.

RESULT ACTIVITY

In the culminating moments of the user's quiz journey, the ResultActivity emerges as the harbinger of feedback and reward, providing users with insights into their performance and tangible rewards for their efforts. Let's delve into its intricacies to understand its functionality.

Package and Imports: The groundwork is laid with the package declaration and necessary imports, harnessing external resources to bolster the result display and reward functionality.

Class Definition: ResultActivity emerges as the final chapter of the quiz engagement, standing as a crucial subclass of AppCompatActivity.

Variable Declaration: Key variables are initialized to orchestrate the result display and reward distribution seamlessly. Instances of FirebaseAuth and FirebaseFirestore are created to leverage the capabilities of Firebase Authentication and Firestore database services. Additionally, 'points' serve as the currency of reward for correct answers.

onCreate() Method: The inception of the ResultActivity, onCreate() method sets the stage for result display and reward distribution. It inflates the layout defined in activity_result.xml, retrieves the user's performance data from the intent extras, calculates the earned points, updates the UI with the score and earned coins, and updates the user's coin balance in the Firestore database.

Restart Button Click Listener: User interactions with the restart button are handled dynamically through this click listener. Upon clicking the button, users are redirected to the MainActivity, signaling the end of the quiz journey and fostering continuity within the application.

In essence, the ResultActivity serves as the culmination of the user's quiz experience, providing insightful feedback on their performance and rewarding their efforts with tangible virtual currency. It embodies the application's commitment to user engagement, feedback, and gamification, fostering a dynamic and immersive learning environment.

FRAGMENTS EXPLANATION

HOME FRAGMENT

Within the immersive landscape of the application's home interface, the HomeFragment emerges as the beacon of exploration and engagement, providing users with access to diverse quiz categories. Let's unravel its intricacies to understand its functionality.

Package and Imports: The groundwork is laid with the package declaration and necessary imports, harnessing external resources to bolster the home interface's functionality.

Class Definition: HomeFragment stands as a pivotal fragment within the application's navigation hierarchy, embodying the essence of user engagement and exploration.

Variable Declaration: Crucial variables are initialized to orchestrate the presentation of quiz categories seamlessly. An instance of FirebaseFirestore is created to leverage the capabilities of Firestore database services. Additionally, 'categories' serve as containers for category data, while 'adapter' facilitates the dynamic rendering of category items.

onCreateView() Method: The genesis of the HomeFragment, onCreateView() method sets the stage for category display. It inflates the layout defined in fragment_home.xml, initializing the binding to access UI elements within the fragment.

onViewCreated() Method: Once the view hierarchy is created, onViewCreated() method further configures the fragment's UI elements and fetches quiz category data from the Firestore database. A snapshot listener is employed to dynamically update the UI when changes occur in the database.

Category Data Retrieval: The fragment dynamically fetches quiz category data from the Firestore database, populating the 'categories' list with CategoryModel objects. These objects encapsulate category information, facilitating seamless rendering within the RecyclerView adapter.

RecyclerView Configuration: The RecyclerView displaying quiz categories is configured with a GridLayoutManager, providing a visually appealing grid layout. The adapter is set to populate category items within the RecyclerView, ensuring a smooth and immersive browsing experience for users.

onDestroyView() Method: Upon view destruction, onDestroyView() method releases the binding instance, preventing memory leaks and ensuring efficient resource management.

In essence, the HomeFragment serves as the gateway to user exploration and engagement within the application, offering a curated selection of quiz categories and fostering a dynamic and immersive learning environment.

LEADERBOARD FRAGMENT

In the immersive realm of user competition and achievement recognition, the LeaderboardsFragment emerges as the epitome of user engagement and competition, providing users with insights into top performers within the application. Let's delve into its intricacies to understand its functionality.

Package and Imports: The groundwork is laid with the package declaration and necessary imports, harnessing external resources to bolster the leaderboards interface's functionality.

Class Definition: LeaderboardsFragment stands as a crucial fragment within the application's navigation hierarchy, embodying the essence of user competition and achievement recognition.

Variable Declaration: Crucial variables are initialized to orchestrate the presentation of user data seamlessly. An instance of FirebaseFirestore is created to leverage the capabilities of Firestore database services. Additionally, 'users' serve as containers for user data, while 'adapter' facilitates the dynamic rendering of user items within the RecyclerView.

onCreateView() Method: The genesis of the LeaderboardsFragment, onCreateView() method sets the stage for user data display. It inflates the layout defined in fragment_leaderboards.xml, initializing the binding to access UI elements within the fragment.

RecyclerView Configuration: Once the view hierarchy is created, the RecyclerView displaying user data is configured with a LinearLayoutManager, providing a vertically scrolling list layout. The adapter is set to populate user items within the RecyclerView, ensuring a smooth and immersive browsing experience for users.

User Data Retrieval: The fragment dynamically fetches user data from the Firestore database, sorting users based on their coin balance in descending order. User objects retrieved from the database are added to the 'users' list, facilitating seamless rendering within the RecyclerView adapter.

onDestroyView() Method: Upon view destruction, onDestroyView() method releases the binding instance, preventing memory leaks and ensuring efficient resource management.

In essence, the LeaderboardsFragment serves as the beacon of competition and achievement within the application, offering users insights into top performers and fostering a competitive and engaging user experience.

PROFILE FRAGMENT

In the immersive realm of user profiles and settings, the ProfileFragment emerges as the hub of user information and interaction, providing users with insights into their account details and options for customization. Let's unravel its intricacies to understand its functionality.

Package and Imports: The groundwork is laid with the package declaration and necessary imports, harnessing external resources to bolster the profile interface's functionality.

Class Definition: ProfileFragment stands as a crucial fragment within the application's navigation hierarchy, embodying the essence of user account management and customization.

Variable Declaration: Key variables are initialized to orchestrate the presentation of user data seamlessly. Instances of FirebaseAuth and FirebaseFirestore are created to leverage the capabilities of Firebase Authentication and Firestore database services.

onCreateView() Method: The genesis of the ProfileFragment, onCreateView() method sets the stage for user data display. It inflates the layout defined in fragment_profile.xml, initializing the binding to access UI elements within the fragment.

onViewCreated() Method: Once the view hierarchy is created, onViewCreated() method further configures the fragment's UI elements and fetches user data from the Firestore database. User details such as email and name are retrieved and displayed in corresponding UI elements.

User Data Retrieval: The fragment dynamically fetches user data from the Firestore database based on the current user's UID. Email and name information retrieved from the database are displayed in the respective text fields, providing users with insights into their account details.

Sign Out Button Click Listener: User interactions with the sign-out button are handled dynamically through this click listener. Upon clicking the button, users are signed out of their current session, and redirected to the LoginActivity, fostering a seamless transition between user sessions.

onDestroyView() Method: Upon view destruction, onDestroyView() method releases the binding instance, preventing memory leaks and ensuring efficient resource management.

In essence, the ProfileFragment serves as the nexus of user account management and customization within the application, offering users insights into their account details and options for interaction and customization.

WALLET FRAGMENT

In the domain of virtual currency management and transactions, the WalletFragment stands as the hub of financial interactions, providing users with insights into their coin balance and options for withdrawal. Let's unravel its intricacies to understand its functionality.

Package and Imports: The foundation is laid with the package declaration and necessary imports, harnessing external resources to bolster the wallet interface's functionality.

Class Definition: WalletFragment emerges as a pivotal fragment within the application's navigation hierarchy, embodying the essence of virtual currency management and transactions.

Variable Declaration: Key variables are initialized to orchestrate the presentation of user wallet data seamlessly. Instances of FirebaseAuth and FirebaseFirestore are created to leverage the capabilities of Firebase Authentication and Firestore database services. Additionally, 'user' serves as a container for user data, encapsulating details such as coin balance.

onCreateView() Method: The inception of the WalletFragment, onCreateView() method sets the stage for wallet data display and interaction. It inflates the layout defined in fragment_wallet.xml, initializing the binding to access UI elements within the fragment.

User Data Retrieval: The fragment dynamically fetches user wallet data from the Firestore database based on the current user's UID. The user's coin balance is retrieved and displayed in the UI, providing users with insights into their financial status.

Send Request Button Click Listener: User interactions with the send request button are handled dynamically through this click listener. Upon clicking the button, a withdrawal request is initiated if the user's coin balance exceeds 50,000 coins. The withdrawal request details, including the user's PayPal email and name, are stored in the Firestore database.

Toast Messages: Toast messages are employed to provide users with feedback on their withdrawal request status, informing them of successful request submission or insufficient coins for withdrawal.

In essence, the WalletFragment serves as the focal point of user financial management within the application, offering users insights into their coin balance and facilitating withdrawal transactions with seamless interaction and feedback mechanisms.

CODE

SPLASH ACTIVITY

```
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)
    }
}
```

MAIN ACTIVITY

```
package com.example.quizpr

import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
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)
        setSupportActionBar(binding.toolbar)
        supportFragmentManager.beginTransaction().replace(R.id.content,
HomeFragment()).commit()
        binding.bottomBar.setOnItemSelectedListener {
            val transaction: FragmentTransaction =
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()
        }
    }

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

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

LOGIN ACTIVITY

```
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()
        }
    }
}
```

SIGNUP ACTIVITY

```
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://firebasestorage.googleapis.com/v0/b/quiz-
5c7fb.appspot.com/o/avatar.png?alt=media&token=c4c1d3ed-541a-4d39-8e8a-9f005fae9ce2",
                    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)
        }
    }
}
```

QUIZ ACTIVITY

```
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()
            }
        }
    }
}
```

RESULT ACTIVITY

```
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()
        }
    }
}
```

HOME FRAGMENT

```
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
    }
}
```

LEADERBOARD FRAGMENT

```
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)
    }
}
```

PROFILE FRAGMENT

```
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
    }
}
```

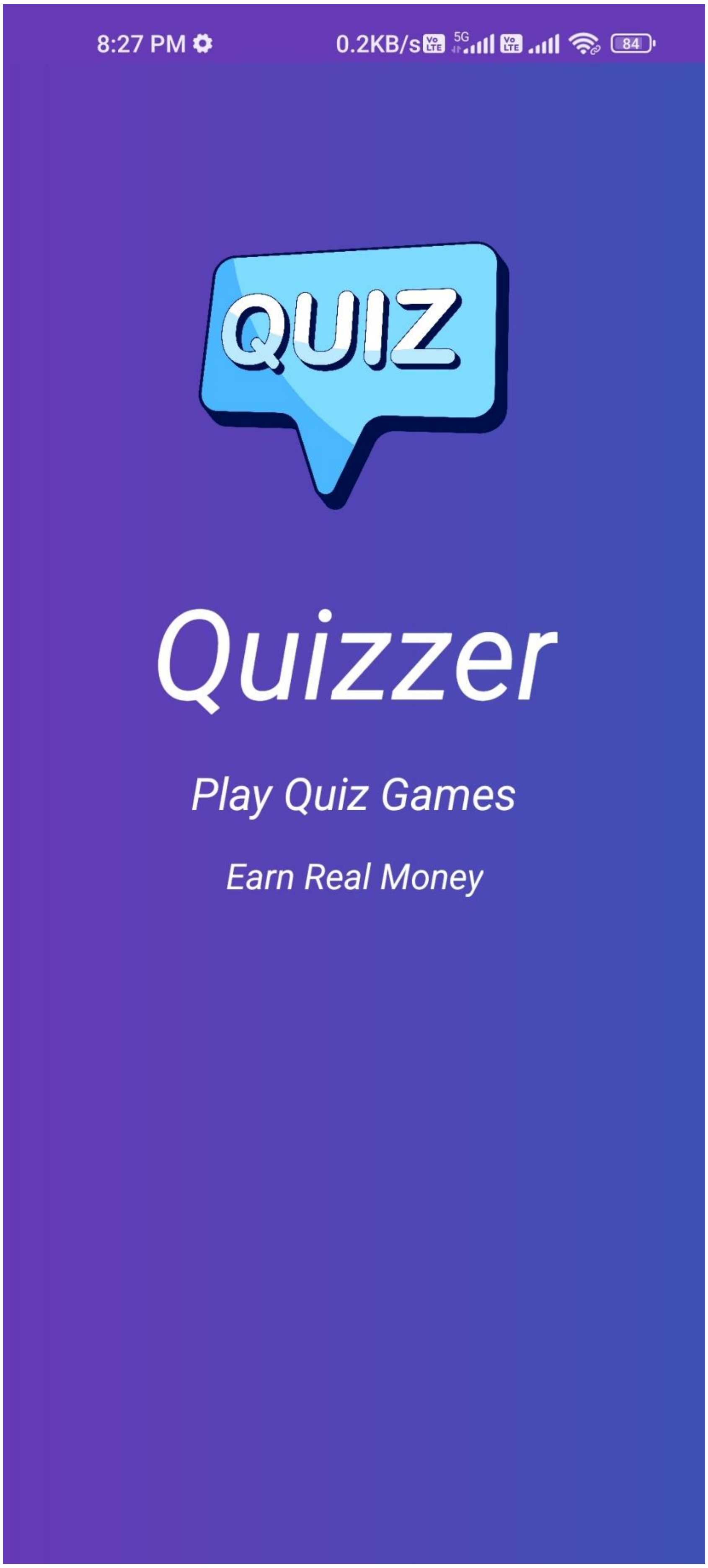
WALLET FRAGMENT

```
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
    }
}
```

EMULATOR SCREENSHOTS



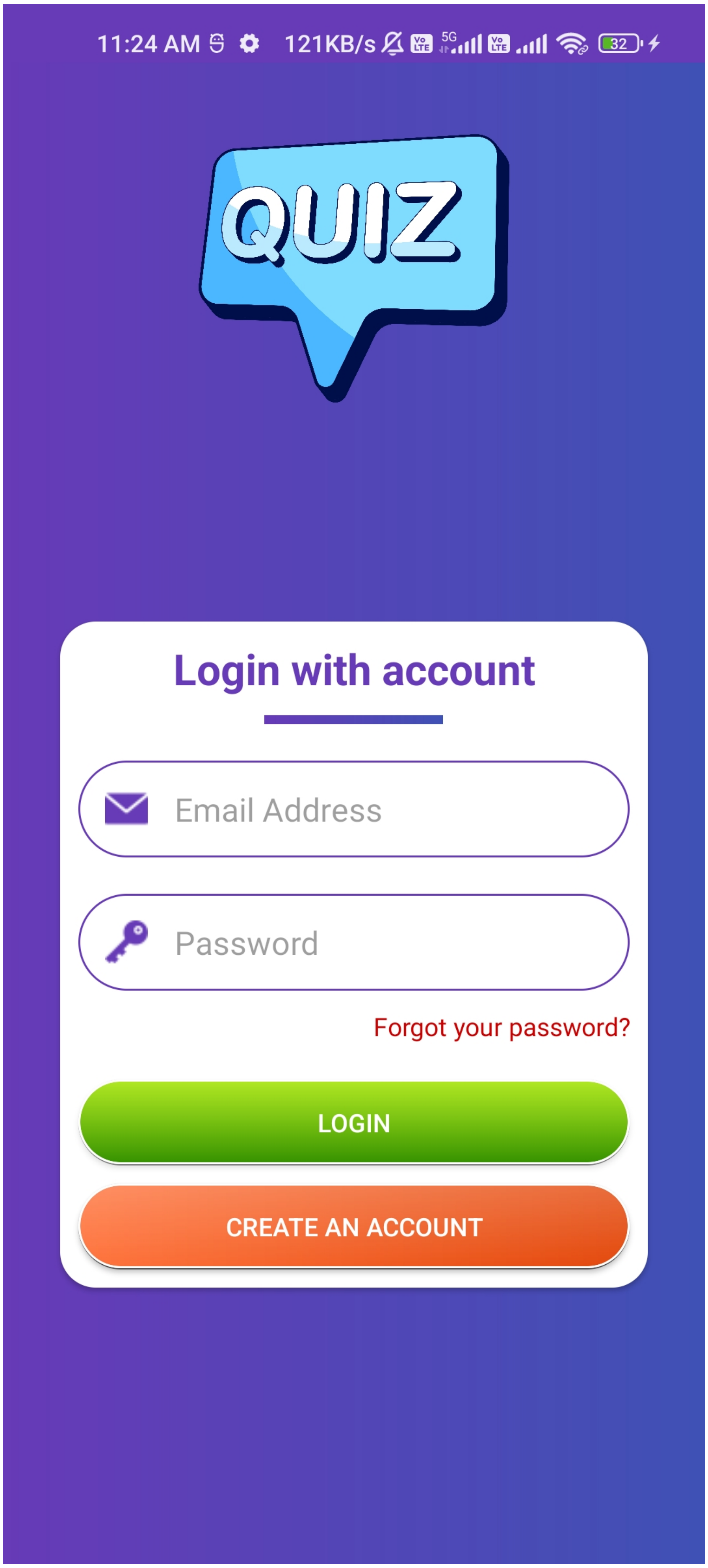
8:27 PM

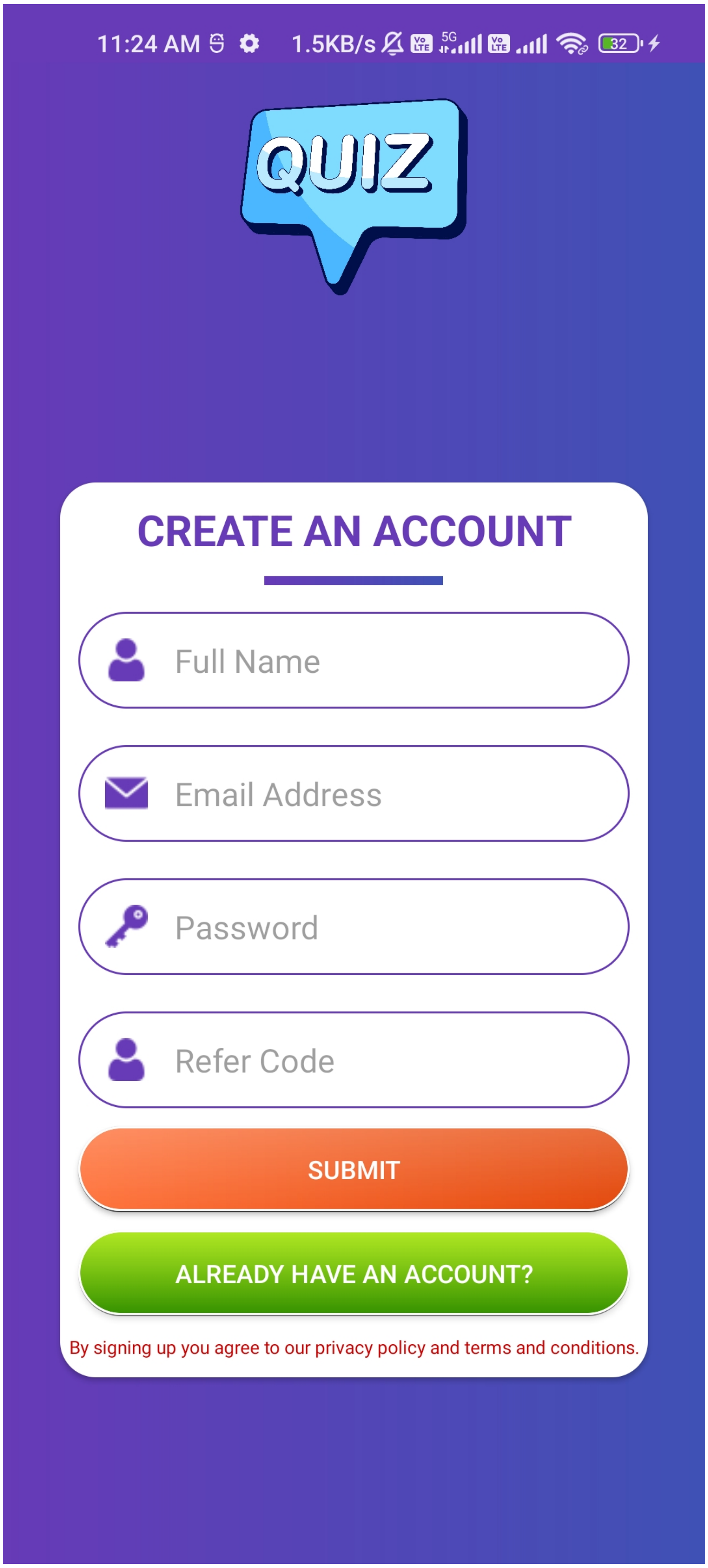
0.2KB/s 84

Quizzer

Play Quiz Games

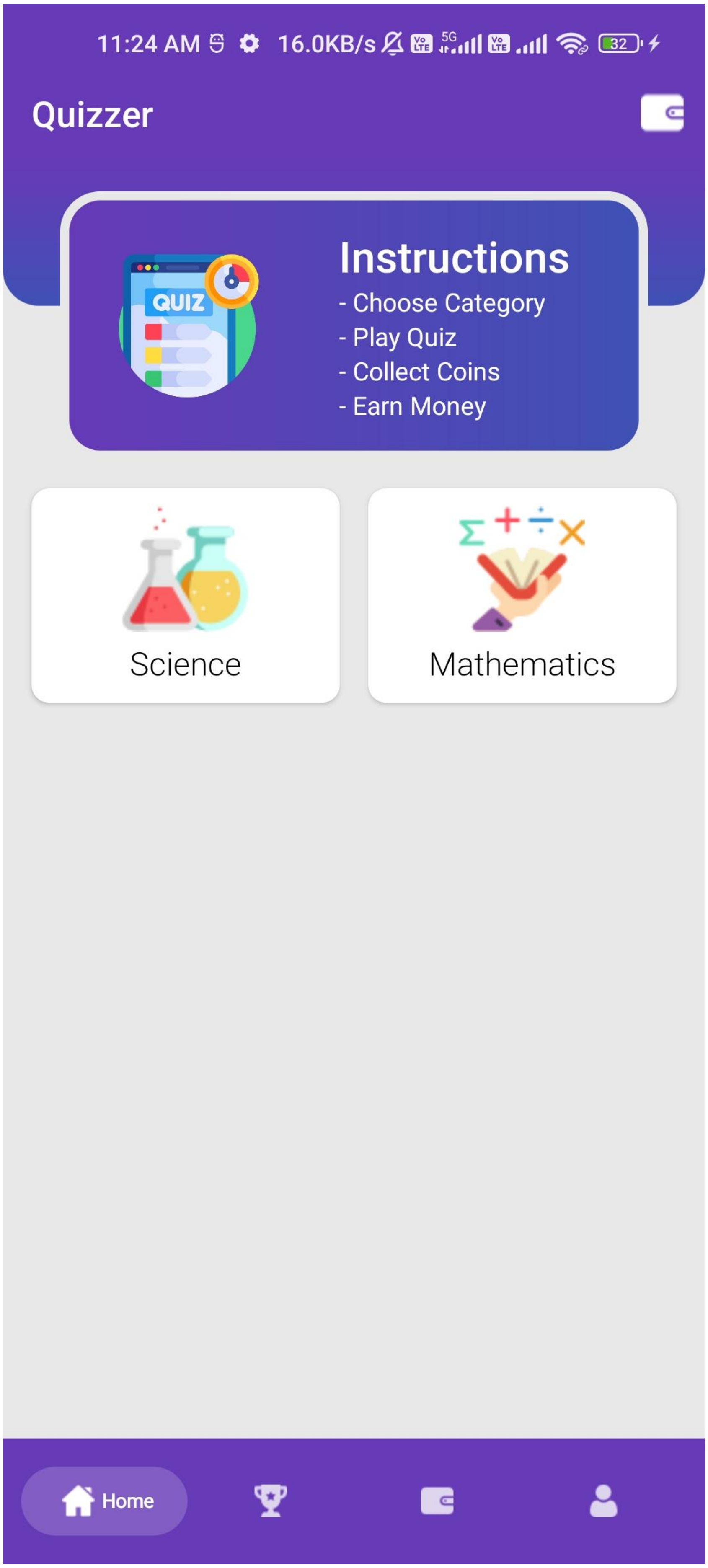
Earn Real Money


11:24 AM
121KB/s
QUIZ
Login with account
Email Address
Password
Forgot your password?
LOGIN
CREATE AN ACCOUNT



CREATE AN ACCOUNT

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

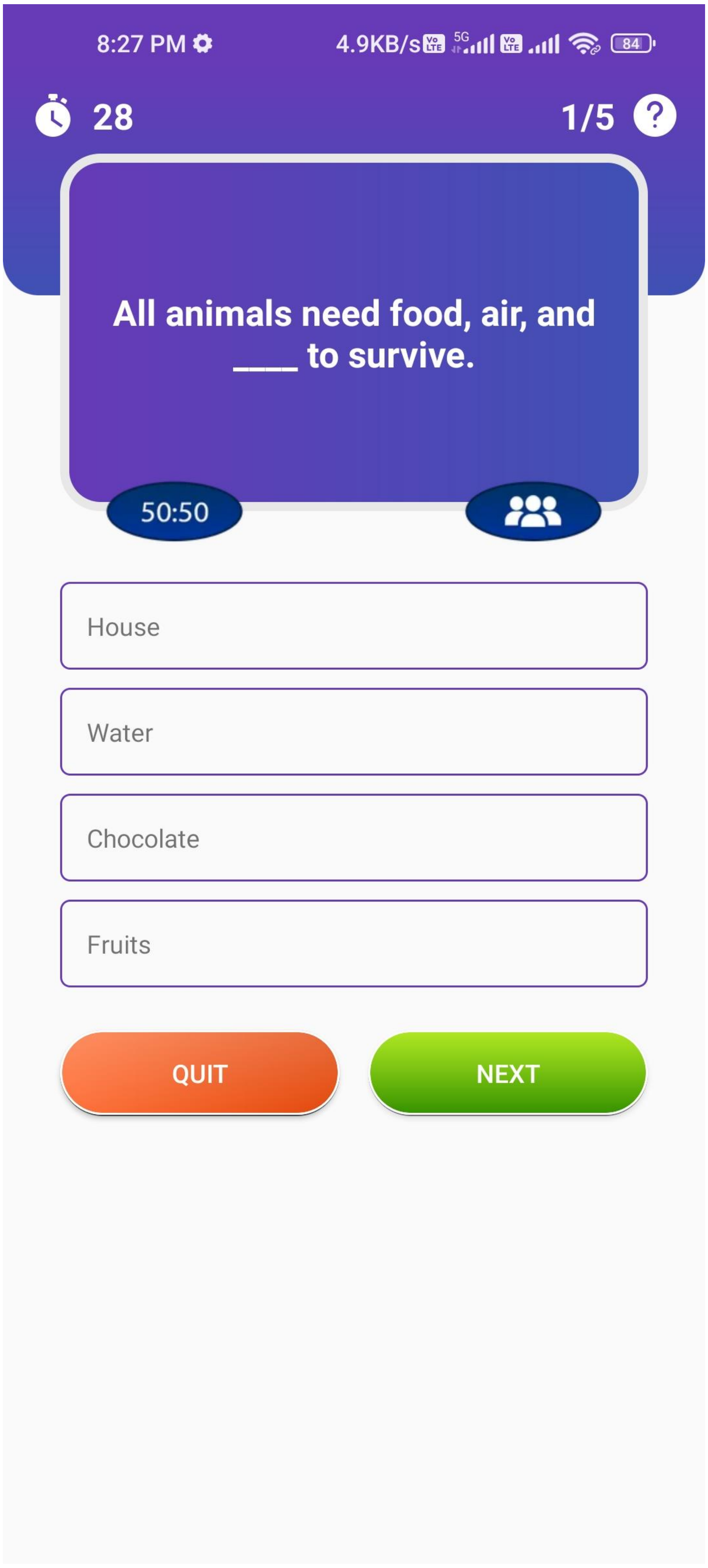

11:24 AM 16.0KB/s 32

Quizzer

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

Science

Mathematics



8:27 PM 4.9KB/s 84

28 1/5

All animals need food, air, and ____ to survive.

50:50

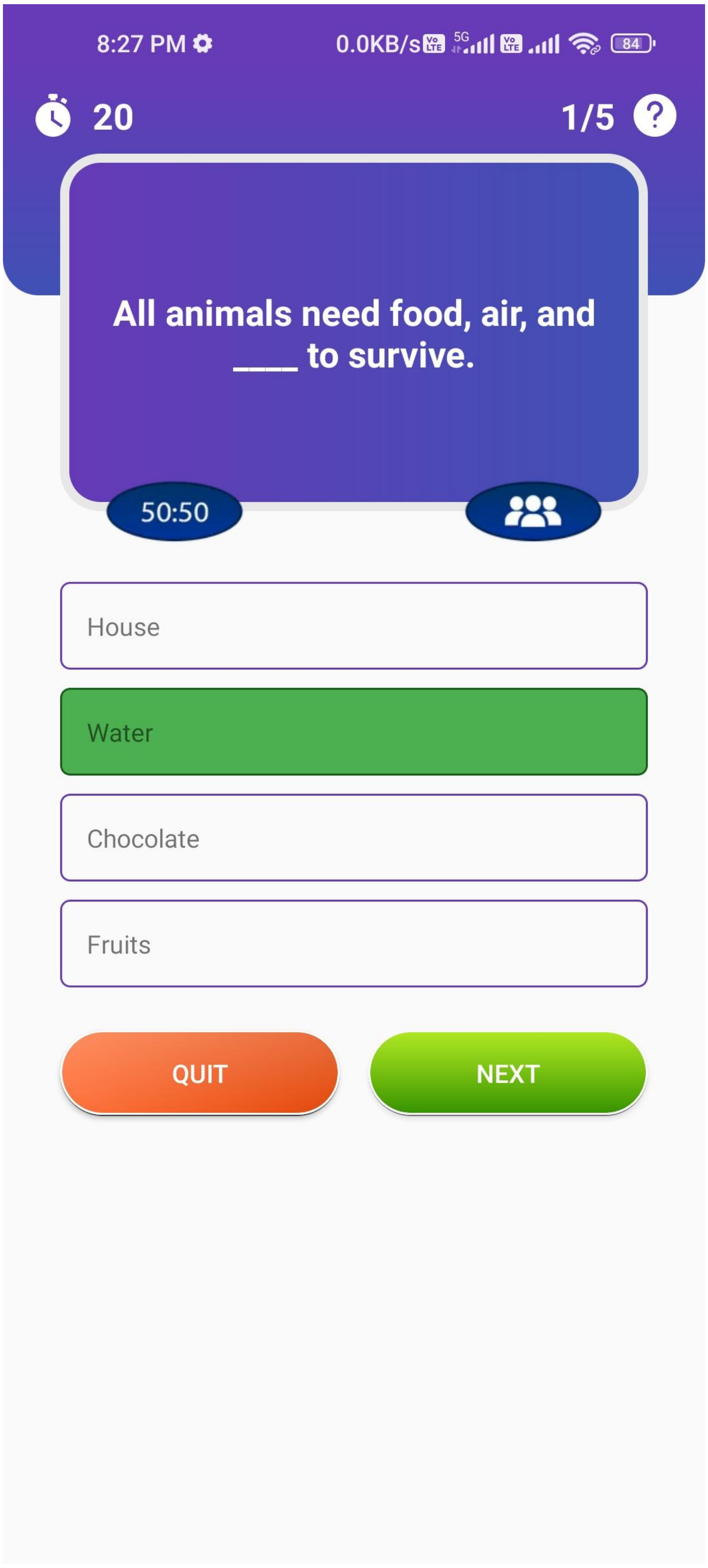
House

Water

Chocolate

Fruits

QUIT NEXT


8:27 PM

0.0KB/s 84

20

1/5

All animals need food, air, and ____ to survive.

50:50

House

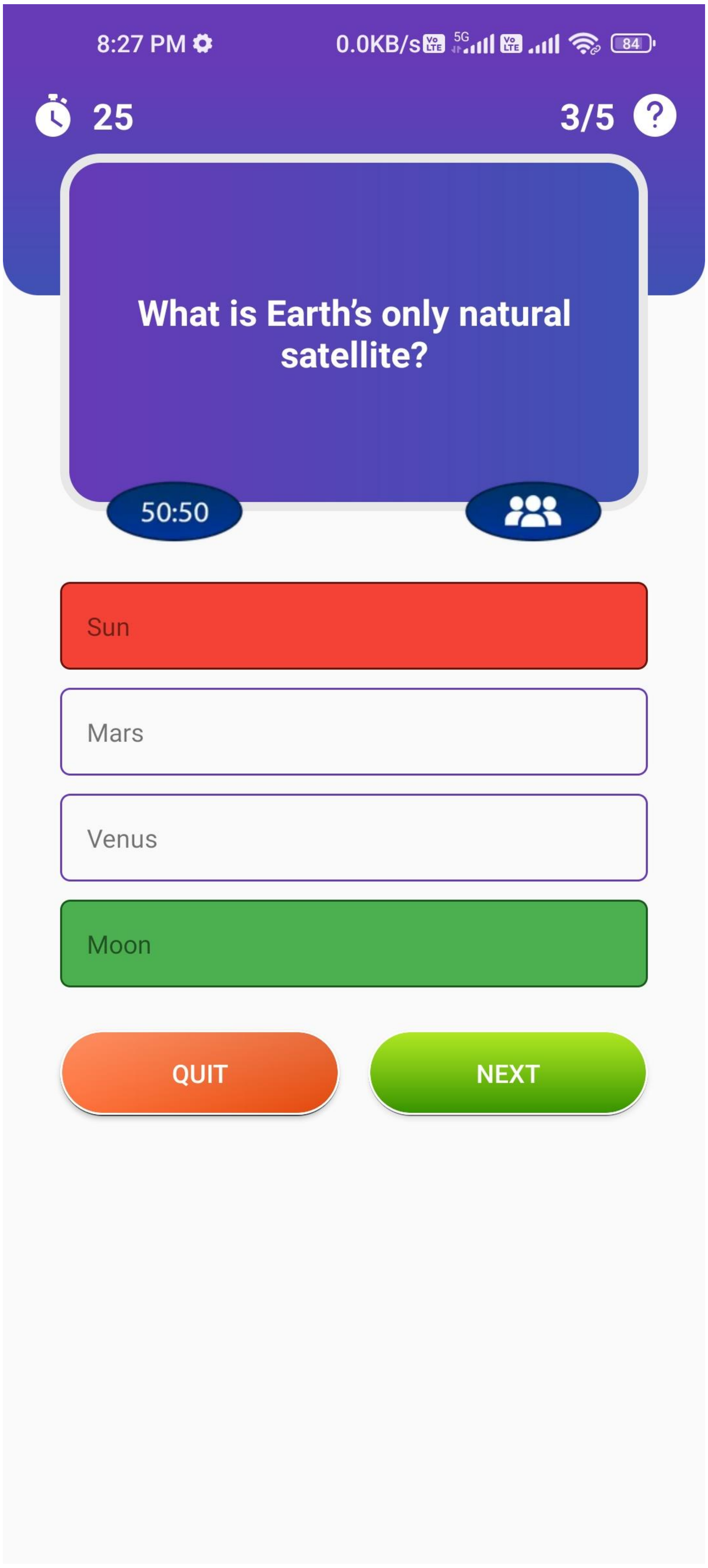
Water

Chocolate

Fruits

QUIT

NEXT



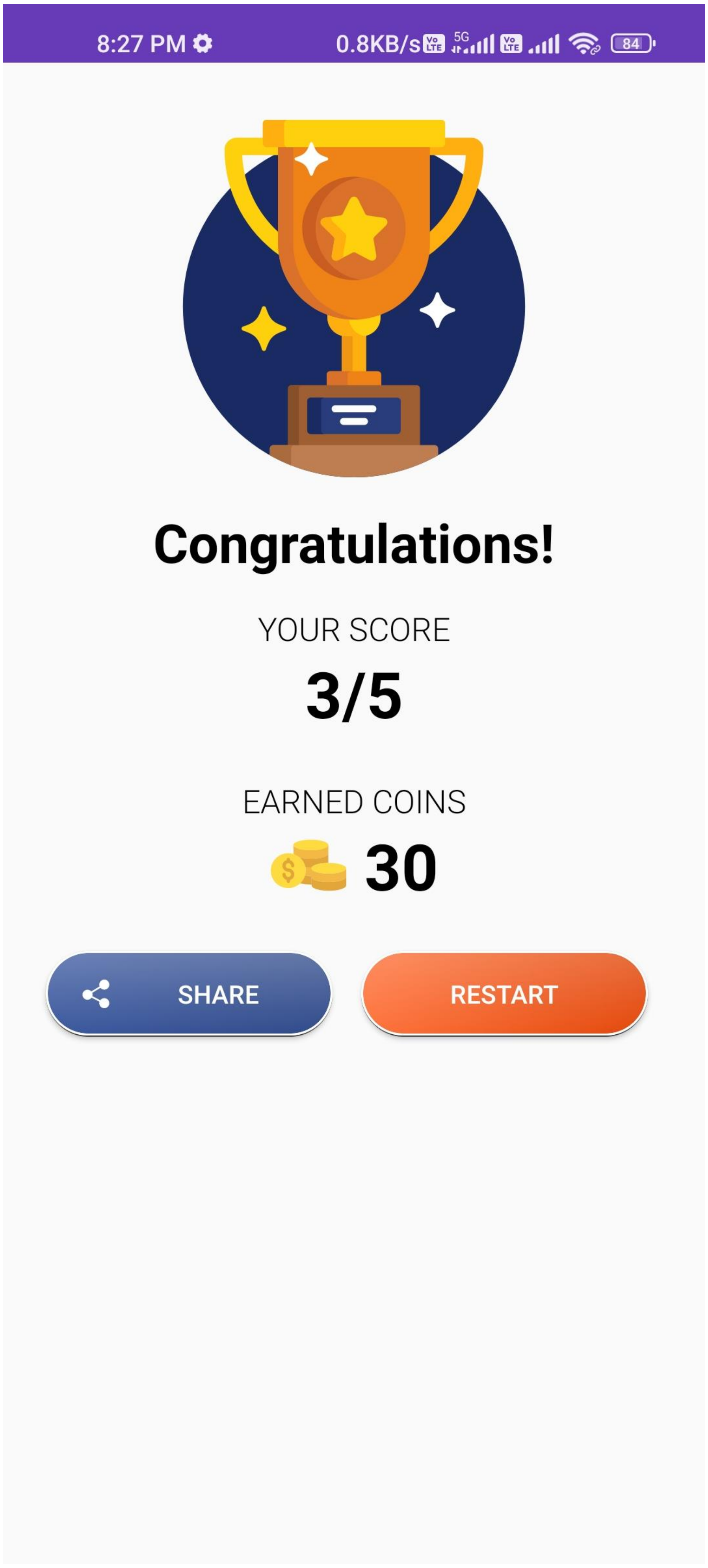
What is Earth’s only natural satellite?

Sun

Mars

Venus

Moon



8:27 PM 0.8KB/s 84

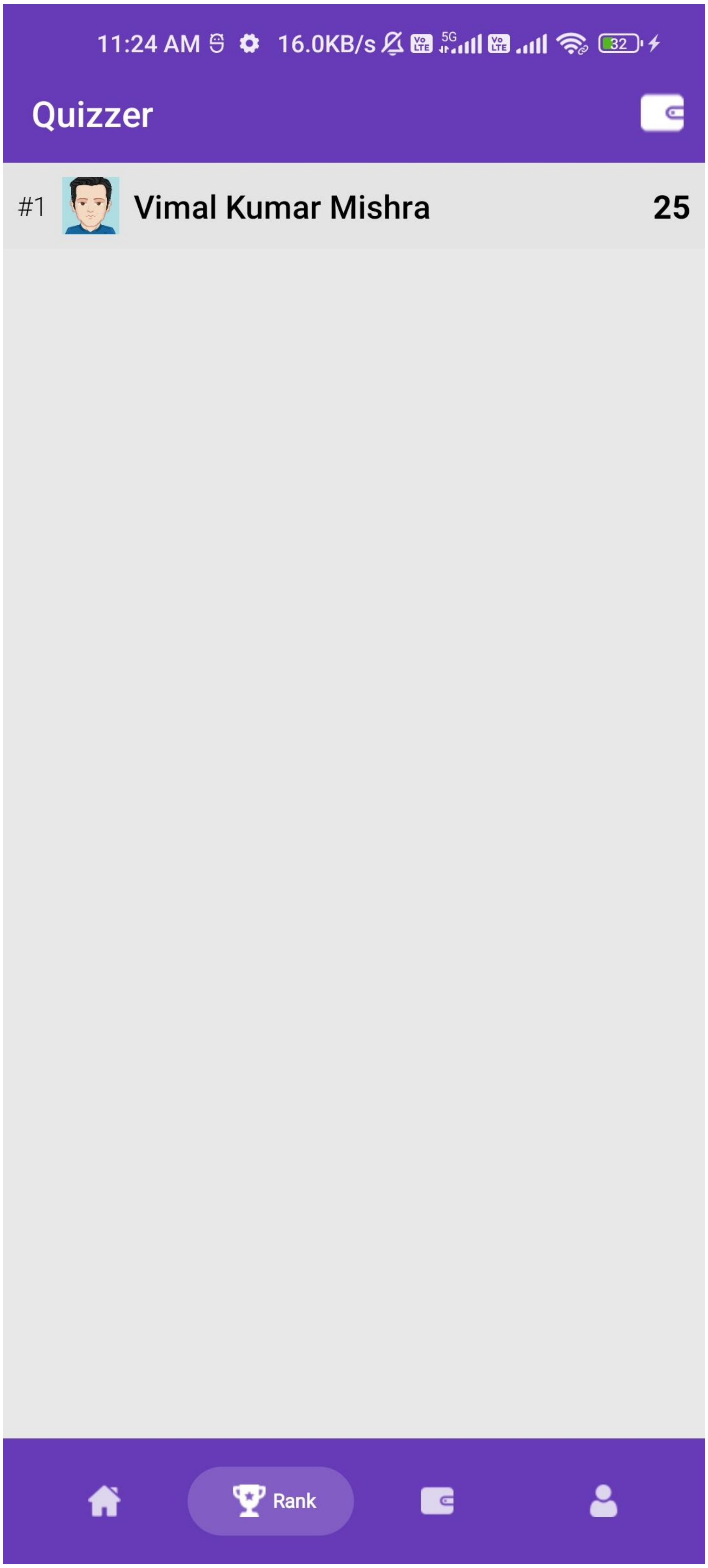
Congratulations!

YOUR SCORE

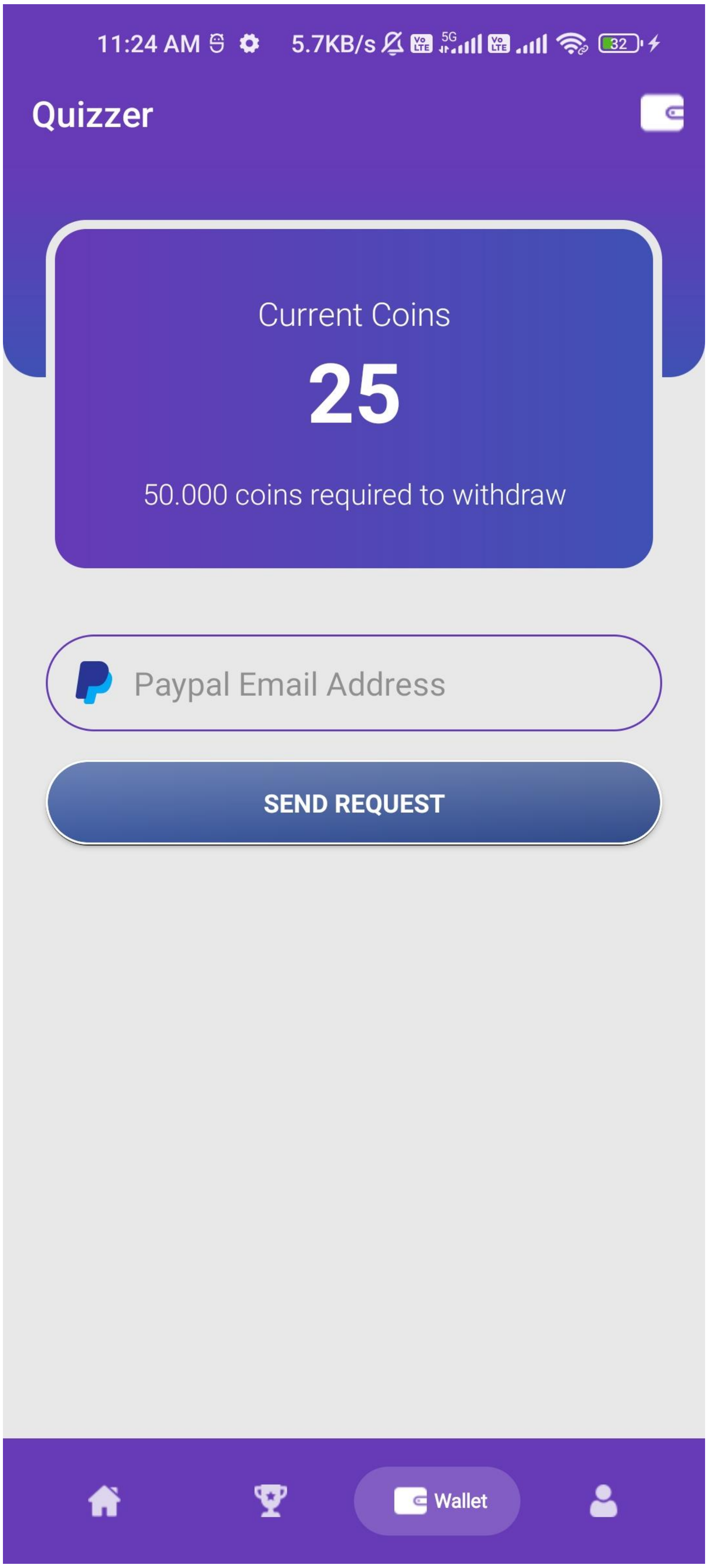
3/5

EARNED COINS

30



#1 Vimal Kumar Mishra 25


11:24 AM 5.7KB/s 5G 32

Quizzer

Current Coins

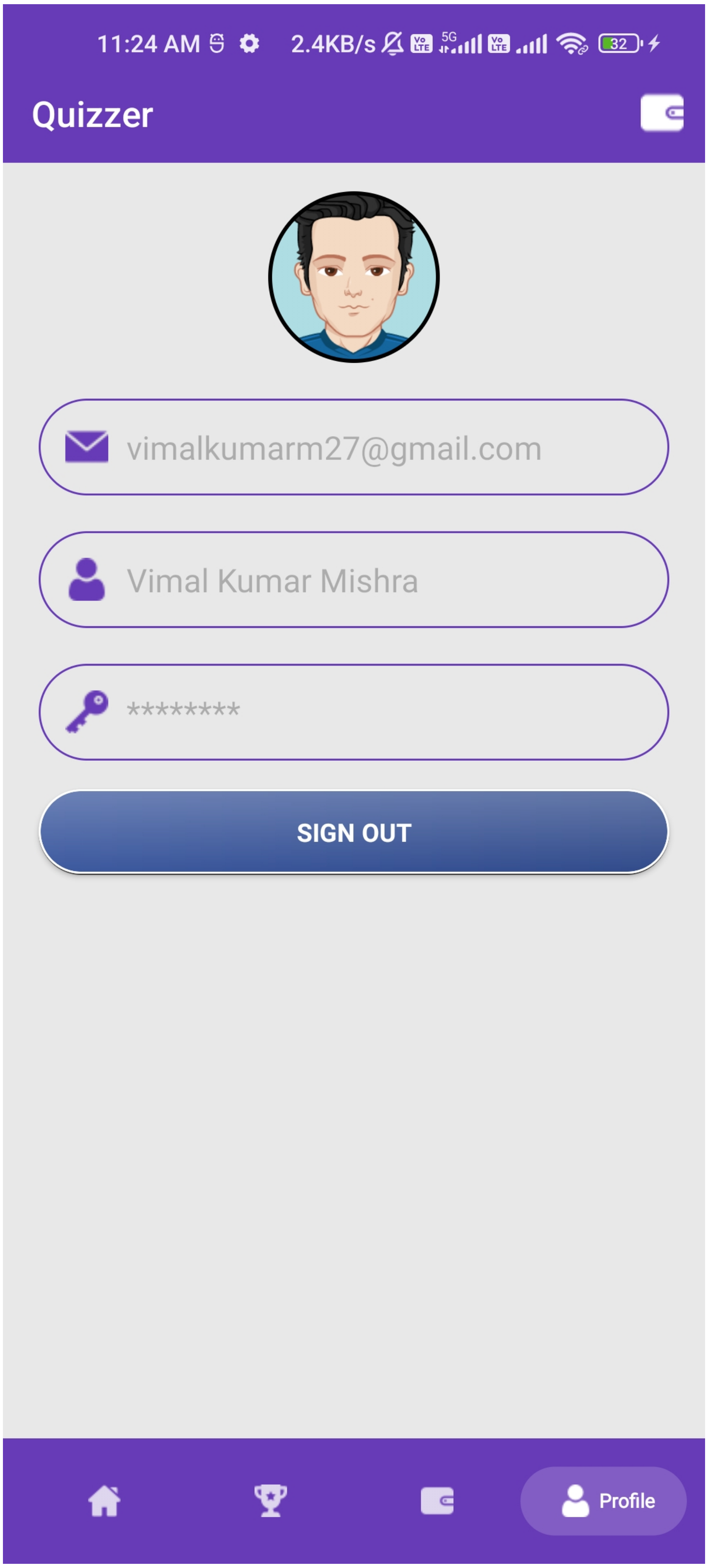
25

50.000 coins required to withdraw

Paypal Email Address

SEND REQUEST

Wallet



11:24 AM 2.4KB/s

Quizzer

vimalkumarm27@gmail.com

Vimal Kumar Mishra

****

SIGN OUT

CONCLUSION & FUTURE SCOPE

CONCLUSION

In the dynamic landscape of educational engagement and user interaction, the QuizPR application stands as a testament to the fusion of technology and learning. With its array of features and seamless functionality, QuizPR offers users a captivating platform to enhance their knowledge while enjoying a rewarding and interactive experience.

Throughout this application, users embark on a journey filled with exploration, learning, and competition. The HomeFragment greets them with a curated selection of quiz categories, inviting them to delve into topics of their interest. The QuizActivity immerses users in challenging quizzes, testing their knowledge and providing immediate feedback on their performance. LeaderboardsFragment showcases top performers, fostering healthy competition and motivation among users. The ProfileFragment and WalletFragment offer users personalized experiences, allowing them to manage their accounts and finances with ease.

Powered by Firebase services, QuizPR ensures seamless authentication, real-time database updates, and secure data storage, enhancing user experience and ensuring data integrity.

As users navigate through QuizPR, they not only expand their knowledge but also foster a sense of community and achievement. Whether it's climbing the leaderboard ranks, mastering challenging quizzes, or managing their virtual currency, users find themselves engaged and empowered.

In conclusion, QuizPR transcends the traditional boundaries of learning applications, offering users a dynamic and immersive platform to enrich their knowledge, compete with peers, and embark on a journey of continuous growth and discovery. With its user-centric design, robust functionality, and seamless integration of technology, QuizPR redefines the learning experience, making education both engaging and rewarding.

FUTURE SCOPE

Looking ahead, QuizPR holds vast potential for expansion and enhancement, with several avenues for future development. One promising direction is the integration of advertisement features to generate revenue and further enrich the user experience.

Advertisement Integration: By incorporating strategically placed advertisements within the application, QuizPR can tap into a lucrative revenue stream. These advertisements can be seamlessly integrated into various sections of the app, such as between quiz questions, on leaderboard screens, or within the profile and wallet sections. Implementing non-intrusive ad formats ensures a positive user experience while maximizing revenue potential.

Monetization Models: QuizPR can explore diverse monetization models, including pay-per-click (PPC), cost-per-mile (CPM), or cost-per-action (CPA) advertising. Additionally, partnerships with relevant brands and sponsors can offer targeted advertising opportunities, aligning advertisements with users' interests and preferences.

User Incentives: To incentivize user engagement with advertisements, QuizPR can implement reward-based ad viewing mechanisms. Users could earn virtual currency, power-ups, or exclusive in-app benefits by voluntarily engaging with advertisements. This not only drives ad revenue but also enhances user engagement and retention.

Personalized Ad Targeting: Leveraging user data and preferences collected through Firebase Analytics, QuizPR can offer personalized ad targeting capabilities. By delivering relevant and contextual advertisements tailored to each user's interests, QuizPR enhances the effectiveness of advertising campaigns while providing users with value-added content.

Expansion to Premium Features: In addition to advertisement integration, QuizPR can explore the introduction of premium features or subscription-based models. Premium features could include ad-free experiences, access to exclusive quizzes or content, advanced analytics insights, or enhanced customization options. This diversification of revenue streams ensures sustainable growth and profitability for QuizPR.

Incorporating advertisement features into QuizPR not only enables revenue generation but also fosters the continuous improvement and expansion of the application. By embracing innovative monetization strategies and prioritizing user experience, QuizPR remains poised for success in the competitive landscape of educational and quiz-based applications.

PROJECT GITHUB LINK

https://github.com/VimalKMGithub/DynamicQuizApp