ADVANCED ANDROID APP DEVELOPMENT

(Project Semester January-May 2024)

Recipe App

Submitted by

Bana Yaswanth Reddy

Registration No 12012200

Programme and Section KO203

Course Code CSE227

Under the Guidance of

Dr.Subhita (20260)

Discipline of CSE/IT

Lovely School of Computer Science of Engineering

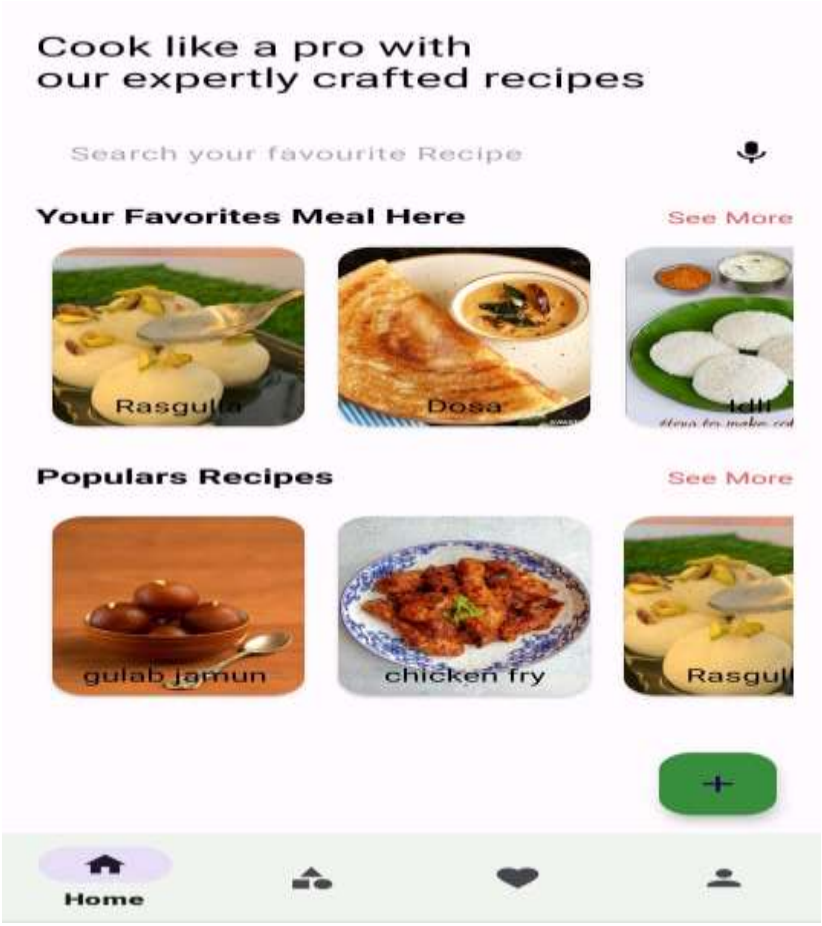
Lovely Professional University, Phagw

CONTENTS:

INTRODUCTION:

In the bustling world of gastronomy, where flavors dance and ingredients sing, finding the perfect recipe can be a daunting task. Enter "RecipesApp" – your trusty sidekick in the kitchen, designed to revolutionize your culinary journey.

With its intuitive interface and extensive database of delectable recipes, RecipesApp is more than just a cooking app – it's a culinary wizardry at your fingertips. Whether you're a novice cook or a seasoned chef, RecipesApp caters to your every need, offering a plethora of recipes ranging from quick and easy meals to gourmet delights.



OBJECTIVES:

The main objective of our recipe application is to empower users to explore, discover, and create culinary delights effortlessly. We aim to provide a user-friendly platform where individuals can access a diverse range of recipes, cooking videos, and culinary inspiration, enhancing their cooking experience and fostering a sense of culinary adventure and creativity.

1) Enhanced User Experience: Ensure that users have a seamless and enjoyable experience browsing, searching, and accessing recipes within the application.User can read the trending.

2) Increased Engagement: Encourage users to engage with the application by regularly exploring new recipes, watching cooking videos, and interacting with the content.

3) Content Variety and Quality: Curate a diverse range of recipes across different categories such as breakfast, lunch, dinner, and vegetarian options to cater to various dietary preferences and occasions.

4) Personalization: Allow users to personalize their experience by adding recipes to their favorites list, enabling them to easily access their preferred recipes for future reference.

5) User Retention and Loyalty: Implement features such as the ability to add recipes as favorites to encourage users to return to the application regularly and build long-term loyalty.

Topics Covered in this project:

- Scroll View
- Rating Bar
- Intents
- Progress bar
- Fragments
- Camera Intent
- Splash screen
- Firebase
- Creating Swipe Views with Tabs
- Creating Bottom Navigation Drawer
- Card View
- Speech to Text converter
- Floating Action Button
- Maps
- Animations(slide_from_bottom ,pulse, fade_out)

Functionalities:

Splash Screen:

Topics Used:

Xml Code:

The XML layout (activity_splash.xml) utilizes ConstraintLayout and includes ImageView and TextView elements to create the visual components of the splash screen.

Kotlin Code:

In the "RecipesApp" Android application, a Handler is set up, linked to the main thread's Looper, facilitating delayed execution managed by a defined Runnable. The loading progress of the splash screen is visualized through a configured ProgressBar. Firebase services initialization, triggered by FirebaseApp.initializeApp(), enables seamless background tasks execution, including user authentication checks via FirebaseAuth.getInstance().currentUser. Depending on the authentication status, the application smoothly transitions to either the MainActivity or the LoginActivity using Intent and startActivity(). Finally, to ensure a streamlined user experience, the finish() method is employed to terminate the current activity, preventing inadvertent return to the splash screen upon navigation.

```
@SuppressLint("CustomSplashScreen")
class SplashActivity : AppCompatActivity() {
    private val splashScreenTime = 1000 // 3 seconds
    private val timeInterval = 100 // 0.1 seconds
    private var progress = 0 // 0 to 100 for progress bar
    private var runnable: Runnable? = null
    private var handler: Handler? = null
    private var binding: ActivitySplashBinding? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySplashBinding.inflate(layoutInflater) // V
        setContentView(binding!!.root)
        binding!!.progressBar.max = splashScreenTime // set max valu
        binding!!.progressBar.progress = progress // set initial val
        handler = Handler(Looper.getMainLooper()) // create handler
        runnable = Runnable {
            // This code will check splash screen time completed or
            if (progress < splashScreenTime) {
                progress += timeInterval
                binding!!.progressBar.progress = progress
                handler!!.postDelayed(runnable!!, timeInterval.toLon
            } else {
                // This code will check user is logged in or not
                FirebaseApp.initializeApp( context: this)
                val user = FirebaseAuth.getInstance().currentUser
                // if user is logged in
```

Login Page:

The Kotlin code, contained in the LoginActivity class, initializes the activity layout and defines the functionality for user login. Upon user interaction with the login button, the login() function is invoked, retrieving the email and password entered by the user and performing validation checks. If the input fields are empty or the email format is invalid, corresponding toast messages are displayed. Otherwise, Firebase authentication is utilized to authenticate the user's credentials. Upon successful authentication, the user is navigated to the MainActivity. The XML layout, defined in activity_login.xml, employs ConstraintLayout to arrange visual components such as ImageView, EditText, Button, and TextView elements. These elements collectively create a visually appealing login interface, complete with input fields for user email and password, a "Forgot password?" option, a "Log In" button, and a "Sign Up" link. The layout design aims to enhance user interaction and facilitate seamless navigation within the application.


```
        // Our code works fine, but we can improve it.
        FirebaseApp.initializeApp( context: this)
        val auth = FirebaseAuth.getInstance()
        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener { task: Task<AuthResult?> ->
                if (task.isSuccessful) {
                    Toast.makeText( context: this,  text: "Login successful", Toast.LENGTH_SHORT).show()
                    startActivity(
                        Intent(
                            packageContext: this,
                            MainActivity::class.java
                        )
                    ) // Navigate to main activity
                    finish()
                } else {
                    Toast.makeText(
                        context: this,
                        text: "Login failed: " + Objects.requireNonNull(task.exception)!!.message,
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
    }
}
```

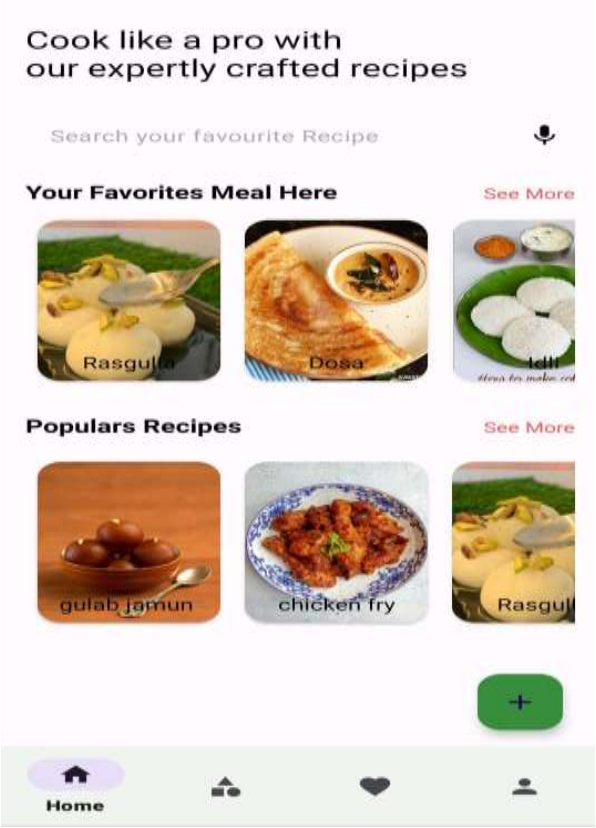
Signup Page:

The Kotlin logic, encapsulated within the SignUpActivity class, orchestrates user registration by capturing input data such as name, email, and password. Upon user interaction with the signup button, validation checks ensure that all required fields are filled and that the email format is valid. If validation passes, a ProgressDialog is displayed to indicate the user creation process. Firebase authentication is then utilized to create a new user account with the provided credentials. Upon successful account creation, the user's name and email are saved to the Firebase Realtime Database as part of the user profile. Toast messages provide feedback to the user regarding the success or failure of the signup process. The XML layout, defined in activity_signup.xml, employs ConstraintLayout and includes visual elements such as ImageView, EditText, Button, and TextView, arranged to create an intuitive and visually appealing signup interface. Overall, the signup functionality aims to enhance user engagement and facilitate seamless registration within the "RecipesApp" application.

```
private fun createNewUser(name: String, email: String, password: String) {

    dialog = ProgressDialog( context: this)
    dialog!!.setMessage("Creating user...")
    dialog!!.setCancelable(false)
    dialog!!.show()
    FirebaseApp.initializeApp( context: this)
    val auth = FirebaseAuth.getInstance()
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener { task: Task<AuthResult?> ->
            if (task.isSuccessful) {
                // user account created successfully
                saveName(name, email)
            } else {
                // account creation failed
                dialog!!.dismiss()
                Toast.makeText( context: this,  text: "Account creation failed", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Main Activity:

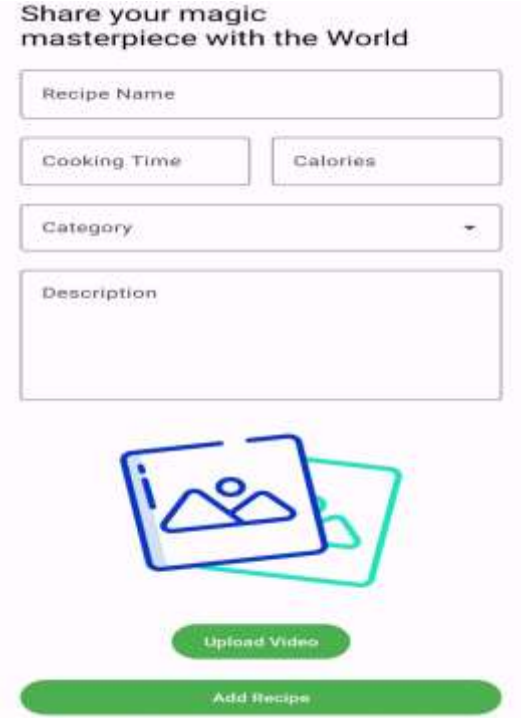


In the MainActivity, the BottomNavigationView serves as a centralized hub for navigating various app sections, providing users with intuitive access to different functionalities. Meanwhile, the FloatingActionButton enhances user engagement by facilitating the addition of new recipes with a single tap. The fragment management system, powered by NavController, efficiently handles the navigation flow within the app, with the HomeFragment serving as a core component. Within the HomeFragment, users encounter a visually appealing and functional interface designed to facilitate recipe exploration. Elements like TextViews for titles, an EditText for seamless search functionality, and RecyclerViews for displaying popular and favorite recipes ensure a rich user experience. Additionally, the integration of voice search via SpeechRecognizer adds a layer of convenience, allowing users to search for recipes hands-free. The inclusion of "see more" TextViews encourages further exploration, fostering user engagement and retention. Overall, the strategic layout design utilizing ConstraintLayout ensures consistent and responsive presentation across various screen sizes, enhancing usability and accessibility. This comprehensive setup ultimately empowers users to effortlessly discover, explore, and interact with the diverse array of recipe content offered by the application.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val binding: ActivityMainBinding
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        val navController = findNavController( activity: this, R.id.nav_host_fragment_activity_main)
        setupWithNavController(binding.navView, navController)
        binding.floatingActionButton.setOnClickListener { view: View? ->
            if (FirebaseAuth.getInstance().currentUser == null) Toast.makeText(
                 context: this,
                 text: "Please login to add recipe",
                Toast.LENGTH_SHORT
            ).show() else startActivity(Intent( packageContext: this@MainActivity, AddRecipeActivity::class.java))
        }
    }
}
```

Adding Recipe:

The AddRecipeActivity allows users to contribute their recipes to the app's database, enhancing the platform's content diversity. Users can input recipe details such as name, description, cooking time, category, and calorie count via intuitive text input fields. These details help categorize and organize recipes within the app, making them easily discoverable by other users. Additionally, the activity facilitates the selection and upload of both images and videos, enriching the recipe presentation with multimedia content.



The UI layout employs a ScrollView wrapped around a ConstraintLayout to ensure smooth scrolling and flexible arrangement of UI components. TextInputLayouts with TextInputEditTexts provide a modern and user-friendly input experience, complete with hints and validation. An AutoCompleteTextView for selecting recipe categories streamlines the categorization process by offering predefined options.

Image and video selection functionalities are seamlessly integrated, allowing users to pick media files from their device's gallery. Upon selection, the chosen image/video is displayed in an ImageView, providing users with visual feedback. The "Add Recipe" button triggers the process of uploading recipe details along with the selected image and video to Firebase Realtime Database and Storage

```
class AddRecipeActivity : AppCompatActivity() {

    private var binding: ActivityAddRecipeBinding? = null
    private var isImageSelected = false
    private var isVideoSelected = false
    private var dialog: ProgressDialog? = null
    private var imageUrl: String? = null
    private var videoUrl: String? = null

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

        val categories = listOf(
            "Breakfast", "Lunch", "Dinner", "Dessert",
            "Vegetarian", "Italian", "Mexican",
        )

        loadCategories(categories)

        binding!!.btnAddRecipe.setOnClickListener { it: View!
            addRecipe()
        }

        binding!!.imgRecipe.setOnClickListener { it: View!
            pickImage()
        }
```

Category Fragment:

The CategoriesFragment displays a list of recipe categories fetched from the Firebase Realtime Database. It serves as a centralized hub for users to explore various culinary genres and discover recipes tailored to their preferences.

The fragment utilizes a RecyclerView with a LinearLayoutManager to present the categories in a vertical list format, ensuring efficient scrolling and navigation. Each category is represented as an item in the RecyclerView, allowing users to easily browse through the available options.

Upon initialization, the fragment establishes a connection to the Firebase Realtime Database and retrieves the list of unique categories associated with the stored recipes. These categories are then sorted alphabetically and mapped to Category objects for seamless integration with the RecyclerView.

```
class CategoriesFragment : Fragment() {

    private var _binding: FragmentCategoryBinding? = null
    private val binding get() = _binding!!
    private lateinit var categoryAdapter: CategoryAdapter
    private lateinit var databaseReference: DatabaseReference

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentCategoryBinding.inflate(inflater, container,  attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        categoryAdapter = CategoryAdapter()
        binding.rvCategories.apply { this: RecyclerView
            adapter = categoryAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }
        databaseReference = FirebaseDatabase.getInstance().getReference( path: "Recipes")
        loadCategoriesFromFirebase()
    }
```

The XML layout file defines the visual structure of the fragment, featuring a ConstraintLayout as the root view. A TextView at the top provides a welcoming message, inviting users to embark on a flavor journey. Below the TextView, the RecyclerView occupies the remaining space to accommodate the list of categories.

Favourites Fragment:

The FavouritesFragment is responsible for displaying a list of favorite recipes stored in the app. It offers users a convenient way to access and revisit their preferred recipes with just a tap.

The XML layout file defines the visual structure of the fragment, featuring a ConstraintLayout as the root view. It includes a TextView at the top, providing a welcoming message to users, and a RecyclerView below it to display the favorite recipes. Additionally, an ImageView with an empty state image is included to indicate when there are no favorite recipes to display.

Within the fragment's code, the loadFavorites() function is called onResume to fetch and populate the list of favorite recipes. It retrieves the favorites from a local SQLite database using the RecipeRepository class and checks if there are any favorites available. If no favorites are found, a toast


```
class FavouritesFragment : Fragment() {
    var binding: FragmentFavouritesBinding? = null
    var recipeRepository: RecipeRepository? = null
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentFavouritesBinding.inflate(inflater, container, attachToParent: false)
        return binding!!.root
    }

    override fun onResume() {
        super.onResume()
        loadFavorites()
    }

    private fun loadFavorites() {
        recipeRepository = RecipeRepository(requireActivity().application)
        val favouriteRecipes = recipeRepository!!.allFavourites
        if (favouriteRecipes!!.isEmpty()) {
            Toast.makeText(requireContext(), text: "No Favourites", Toast.LENGTH_SHORT).show()
            binding!!.rvFavourites.visibility = View.GONE
            binding!!.noFavourites.visibility = View.VISIBLE
        } else {
```

message is displayed, and the RecyclerView is hidden while the empty state ImageView is shown. Otherwise, the RecyclerView is configured with a GridLayoutManager for a two-column layout, and the favorite recipes are fetched from the Firebase Realtime Database based on their unique IDs. The fetched recipes are then added to a list and passed to the RecyclerView adapter for display.

Profile Fragment:

The ProfileFragment manages the user profile screen within the Recipes App. It allows users to view and edit their profile details, including their name, email, profile picture, cover image, and favorite recipes.

In the XML layout file, a ConstraintLayout is used to organize the visual elements of the profile screen. It includes:

ImageView for the profile picture (img_profile) and cover image (img_cover), along with edit icons (img_edit_profile and img_edit_cover) to enable users to change these images.

TextView for displaying the user's name (tv_user_name) and email (tv_email).

RecyclerView (rv_profile) to display a grid of the user's favorite recipes.

ImageView for the settings button (btn_setting) to navigate to the settings screen.

Within the ProfileFragment class, the onViewCreated method initializes the profile screen, checking if the user is logged in. If not logged in, it prompts the user to log in. Otherwise, it loads the user's profile details and sets up the UI elements.

The init method sets up click listeners for the edit icons (img_edit_profile and img_edit_cover) to allow users to pick images from the gallery and upload them to Firebase Storage as their profile picture and cover image, respectively. It also sets up a click listener for the settings button

(btn_setting) to navigate to the settings screen.

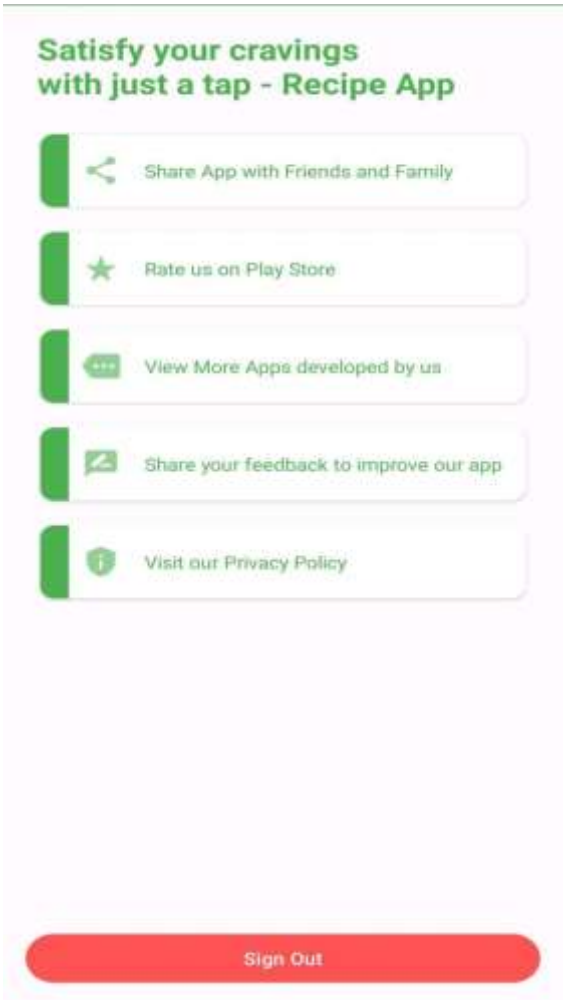
The loadProfile method fetches the user's profile details from the Firebase Realtime Database and populates the UI elements with the retrieved data. Additionally, it sets default values for the profile details if no data is available.

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        if (FirebaseAuth.getInstance().currentUser == null) {
            AlertDialog.Builder(context) AlertDialog.Builder
                .setTitle("Login Required") AlertDialog.Builder!
                .setMessage("You need to login to view your profile")
                .show()
        } else {
            loadProfile()
            init()
        }
    }
```

Setting Activity:

The XML layout file defines the UI for the Settings screen (SettingActivity) of the Recipes App. The layout consists of a ConstraintLayout containing various CardView elements organized within a vertical LinearLayout. Here's a breakdown of the components:



TextView (textView): Displays a title for the settings screen.

LinearLayout (linearLayoutShare): Represents a clickable area for sharing the app. It contains an icon (ic_share) and a text label (share_app).

LinearLayout (linearLayoutRate): Represents a clickable area for rating the app. It contains an icon (ic_rate) and a text label (rate_app).

LinearLayout (linearLayoutApps): Represents a clickable area for viewing more apps by the developer. It contains an icon (ic_more_apps) and a text label (view_more_apps).

LinearLayout (linearLayoutFeedback): Represents a clickable area for sending feedback about the app. It contains an icon (ic_feedback) and a text label (send_feedback).

LinearLayout (linearLayoutPrivacy): Represents a clickable area for viewing the privacy policy of the app. It contains an icon (ic_policy) and a text label (privacy_policy).

Button (btn_signout): Allows the user to sign out of the app.

The SettingActivity class handles the functionality of the Settings screen. It sets up click listeners for each option, such as sharing the app, rating it, sending feedback, viewing more apps, and accessing the privacy policy. Additionally, it provides a sign-out functionality for the user. When the user clicks on any of the options, corresponding actions are triggered, such as opening links, sending emails, or launching other apps.

```
class SettingActivity : AppCompatActivity() {
    var binding: ActivitySettingBinding? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySettingBinding.inflate(layoutInflater)
        setContentView(binding!!.root)
        binding!!.linearLayoutShare.setOnClickListener { view: View? -> shareApp() }
        binding!!.linearLayoutRate.setOnClickListener { view: View? -> rateApp() }
        binding!!.linearLayoutFeedback.setOnClickListener { view: View? -> sendFeedback() }
        binding!!.linearLayoutApps.setOnClickListener { view: View? -> moreApps() }
        binding!!.linearLayoutPrivacy.setOnClickListener { view: View? -> privacyPolicy() }
        binding!!.btnSignout.setOnClickListener { view: View? -> signOut() }
    }

    private fun signOut() {
        if (FirebaseAuth.getInstance().currentUser == null) {
            return
        }
        AlertDialog.Builder( context: this) AlertDialog.Builder
            .setTitle("Sign Out") AlertDialog.Builder
            .setMessage("Are you sure you want to sign out?")
            .setPositiveButton( text: "Sign Out") { dialogInterface: DialogInterface?, i: Int ->
                FirebaseAuth.getInstance().signOut()
                startActivity(Intent( packageContext: this@SettingActivity, LoginActivity::class.java))
                finishAffinity()
            }
```

Recipe Activity Details:

The XML layout file defines the UI for displaying details of a recipe in the Recipe Details screen (RecipeDetailsActivity) of the Recipes App. The layout is wrapped inside a ScrollView to allow scrolling through the content if it exceeds the screen size. Here's a breakdown of the components:

ImageView (img_recipe): Displays the image of the recipe.

TextView (tv_name): Displays the name or title of the recipe.

TextView (tc_category): Displays the category of the recipe.

TextView (tv_calories): Displays the calorie count of the recipe.

ImageView (img_fvrt): Allows the user to mark the recipe as a favorite.

ImageView (img_edit): Allows the user to edit the recipe (visible only if the user is the author of the recipe).

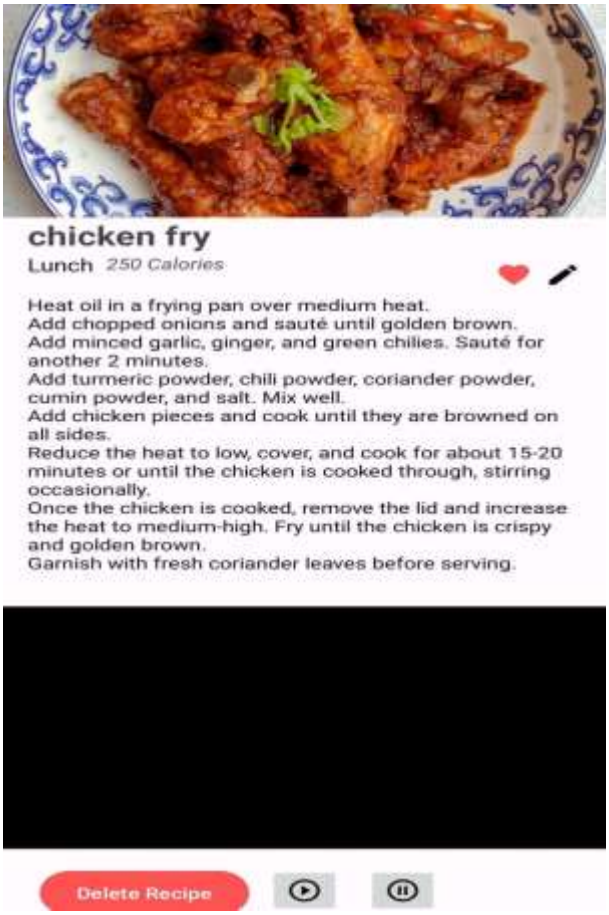
TextView (tv_description): Displays the description or instructions of the recipe.

VideoView (video_view): Displays a video related to the recipe, if available. This is set to visibility="gone" by default and becomes visible if there is a video associated with the recipe.

Button (btn_delete): Allows the user to delete the recipe (visible only if the user is the author of the recipe).

ImageButton (btnPlay): Button to play the video associated with the recipe.

ImageButton (btnPause): Button to pause the video associated with the recipe.




```
class RecipeDetailsActivity : AppCompatActivity() {
    private var binding: ActivityRecipeDetailsBinding? = null
    private var recipe: Recipe? = null
    private var videoUri: Uri? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityRecipeDetailsBinding.inflate(layoutInflater)
        val view = binding!!.root
        setContentView(view)
        init()
    }

    private fun init() {
        recipe = intent.getSerializableExtra( name: "recipe") as Recipe?
        recipe?.let { displayRecipeDetails(it) }
        setupUI()
    }

    private fun displayRecipeDetails(recipe: Recipe) {
        Log.d( tag: "TAG", recipe.toString())
        binding?.apply { this: ActivityRecipeDetailsBinding
            tvName.text = recipe.name
```

References

[1] Scroll View: https://developer.android.com/reference/android/widget/ScrollView

[2] Rating Bar: https://developer.android.com/reference/android/widget/RatingBar

[3] Intents: https://developer.android.com/guide/components/intents-filters

[4] Progress Bar: https://developer.android.com/reference/android/widget/ProgressBar

[5] Fragments: https://developer.android.com/guide/fragments

[6] Camera Intent: https://developer.android.com/training/camera/photobasics

[7] Splash Screen: https://developer.android.com/guide/topics/ui/declaring-layout

[8] Firebase: https://firebase.google.com/

[9] Creating Swipe Views with Tabs: https://developer.android.com/training/animation/screen-slide

[10] Creating Bottom Navigation Drawer: https://developer.android.com/guide/navigation/navigation-ui#bottom_nav

[11] Card View: https://developer.android.com/guide/topics/ui/layout/cardview

[12] Speech to Text Converter: https://developer.android.com/reference/android/speech/SpeechRecognizer

[13] Floating Action Button: https://developer.android.com/guide/topics/ui/floating-action-button

Bibliography

1. ANDROID APPLICATION DEVELOPMENT ALL-IN-ONE FOR DUMMIES by BARRY BURD, JOHN PAUL MUELLER, WILEY
2. ANDROID APPLICATION DEVELOPMENT by BARRY BURD, WILEY
3. ANDROID APPLICATION DEVELOPMENT by PRADEEP KOTHARI, DREAMTECH PRESS