

EMPOWER WORKOUT

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING

By

VELUGU NAVEEN

REG NO: - 12205426

CSE 226

ANDROID APP DEVELOPMENT



School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

November 2024

TABLE OF CONTENTS

Sl. No	CONTENTS	PAGE NO.
1.	Introduction	
2.	Modules or Activity Explanation & Screenshots	
	❖ AndroidManifest.xml	
	❖ Main Activity	
	❖ Login Activity	
	❖ Signup Activity	
	❖ Home Fragment Activity	
	❖ Diet Meal Plan Fragment	
	❖ WorkoutPlanFragment	
	❖ DietPlanAdapter	
3.	Activity Wise Xml & design	
	1. activity_diet_meal_plan_fragment.xml	
	2. activity_login.xml	
	3. activity-main.xml	
	4. activity_sign_up.xml	
	5. ragment_home.xml	
	6. ragment_workout_plan.xml	
	7. item_diet_plan.xml	
	8. item_schedule.xml	
4.	Emulator Screenshots	
5.	Conclusion	
6.	Project GitHub link	

INTRODUCTION

In the modern era, where sedentary lifestyles and unhealthy eating habits are prevalent, the importance of fitness and nutrition has become paramount. The rise of mobile technology offers a unique opportunity to address these challenges by providing users with accessible, personalized, and interactive solutions for maintaining a healthy lifestyle. This paper introduces the development of the **Empower Workout App**, a comprehensive fitness and nutrition application designed to promote holistic well-being.

The **Empower Workout App** caters to users seeking structured workout routines and balanced dietary plans. It features an intuitive interface powered by Android, integrating fragments for modularity and enhanced user interaction. The app emphasizes a personalized experience by dynamically presenting workout and diet schedules based on the day of the week. Users can easily navigate between home, workout plans, and diet meal plans using a bottom navigation bar, ensuring a seamless experience.

Key features include:

- **Dynamic Workout Plans:** Tailored routines for each day, focusing on muscle groups and rest days to ensure optimal recovery.
- **Dietary Flexibility:** Separate vegetarian and non-vegetarian meal plans, catering to diverse user preferences.
- **Interactive User Interface:** A RecyclerView-based weekly schedule overview and image-based descriptions to make navigation and understanding effortless.
- **Ease of Access:** Modular fragments like Home Fragment, WorkoutPlanFragment, and DietMealPlanFragment improve navigation and maintainability.

This project leverages Android's powerful development framework, employing RecyclerView for schedule management, and user-friendly layouts to present complex data efficiently. The app's architecture ensures scalability, making it

adaptable for future enhancements like user login, progress tracking, and calorie calculation.

By combining effective workout regimens and diet recommendations, the **Empower Workout App** aims to empower individuals to take charge of their health in an engaging and structured manner. The subsequent sections of this paper delve into the design, implementation, and evaluation of the application, highlighting its potential to bridge the gap between technology and health-conscious living.

ALL MODULES WITH IT'S ACTIVITY

The Android Manifest file (AndroidManifest.xml) is a core component of every Android application. It provides essential information to the Android operating system about the app's structure, components, permissions, and configurations. This file acts as a blueprint, guiding the system in managing the app's behaviour.

Key Components

1. Root Element:

The <manifest> tag defines the app's namespace and package name. It also includes attributes for Android-specific (xmlns: android) and development tools (xmlns: tools).

2. Application Tag:

The <application> tag contains global attributes for the app, such as:

- **android: allowBackup:** Enables automatic data backup.
- **android: icon:** Sets the app's launcher icon.
- **android: theme:** Defines the app's appearance.
It also includes tools: target Api for compatibility with API level 31.

3. Activities:

Activities represent the app's screens.

- **MainActivity:** The app's entry point, defined with an <intent-filter> for launcher access.
- **LoginActivity** and **SignUpActivity:** Handle user authentication.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    xmlns:tools="http://schemas.android.com/tools">
4
5    <application
6        android:allowBackup="true"
7        android:dataExtractionRules="@xml/data_extraction_rules"
8        android:fullBackupContent="@xml/backup_rules"
9        android:icon="@drawable/appicon"
10       android:label="@string/app_name"
11       android:roundIcon="@drawable/appicon"
12       android:supportsRtl="true"
13       android:theme="@style/Theme.EmpowerWorkoutApp"
14       tools:targetApi="31">
15
16        <activity
17            android:name=".MainActivity"
18            android:exported="true">
19            <intent-filter>
20                <action android:name="android.intent.action.MAIN" />
21                <category android:name="android.intent.category.LAUNCHER" />
22            </intent-filter>
23        </activity>
24
25        <activity
26            android:name=".LoginActivity"
27            android:exported="true" />
28
29        <activity
30            android:name=".SignUpActivity"
31            android:exported="true" />
32
33    </application>
34
35 </manifest>
36
```

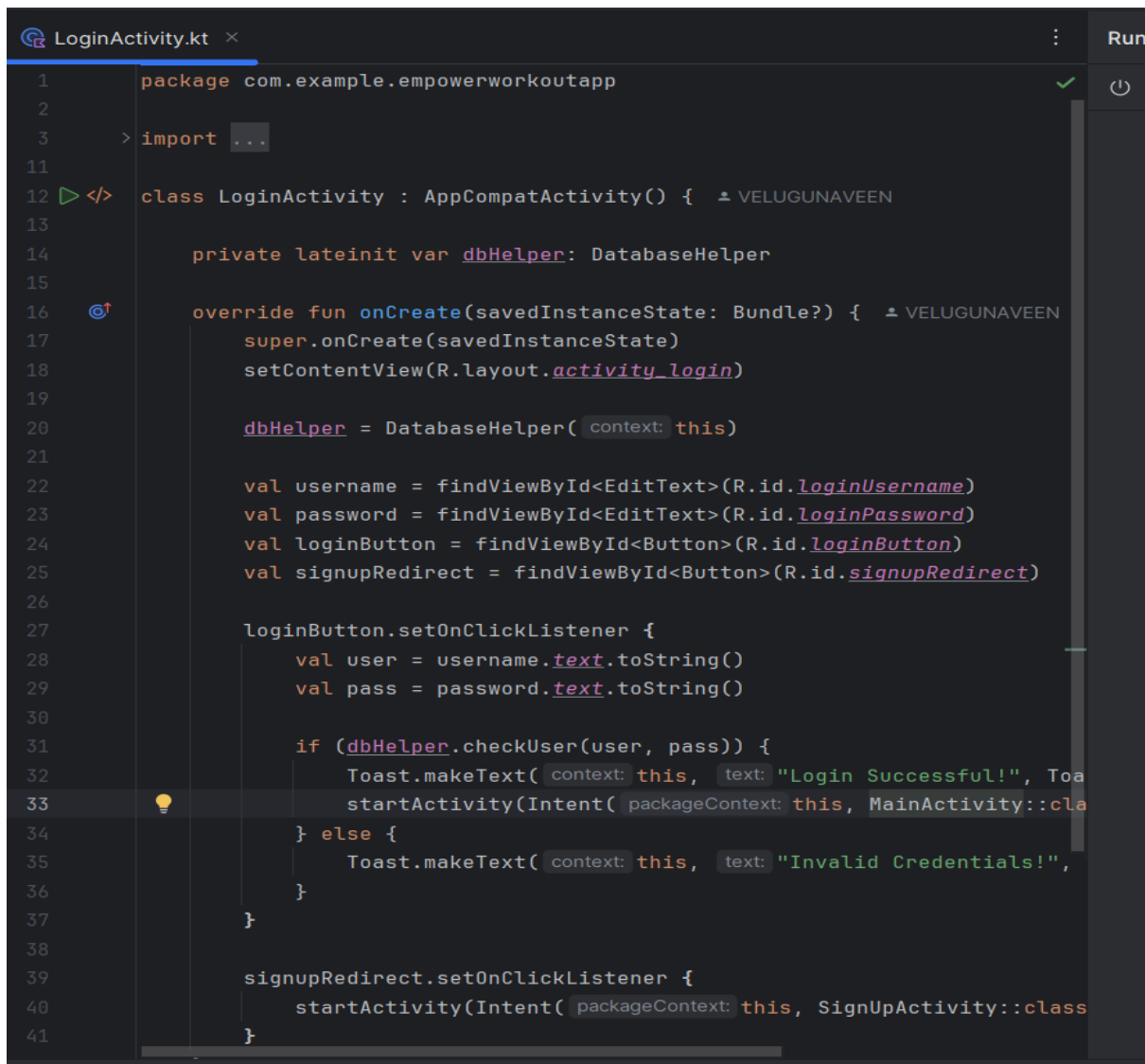
Main Activity

The MainActivity class is the main entry point of the application. It extends AppCompatActivity and sets the layout using setContentView. A Toolbar is initialized and set as the app's action bar. A BottomNavigationView is used to navigate between three fragments: HomeFragment, WorkoutPlanFragment, and DietMealPlanFragment. The setOnNavigationItemSelectedListener handles navigation based on the selected menu item, loading the corresponding fragment using the loadFragment method. The loadFragment method replaces the current fragment in the fragmentManager using FragmentManager. Initially, HomeFragment is loaded as the default screen. This structure ensures seamless fragment navigation.

```
MainActivity.kt x
1  package com.example.empowerworkoutapp
2
3  > import ...
4
5
6
7
8
9  class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13         val toolbar: Toolbar = findViewById(R.id.toolbar)
14         setSupportActionBar(toolbar)
15
16         val bottomNav = findViewById<BottomNavigationView>(R.id.bottomNavigationView)
17         bottomNav.setOnNavigationItemSelectedListener {
18             when (it.itemId) {
19                 R.id.nav_home -> loadFragment(HomeFragment())
20                 R.id.nav_workout -> loadFragment(WorkoutPlanFragment())
21                 R.id.nav_dietschedule -> loadFragment(DietMealPlanFragment())
22             }
23             true
24         }
25
26         loadFragment(HomeFragment()) // Default screen
27     }
28
29     private fun loadFragment(fragment: Fragment) {
30         supportFragmentManager.beginTransaction().replace(R.id.fragmentContainer, fragment).commit()
31     }
32 }
33
```

Login Activity

The LoginActivity class is responsible for user authentication in the app. It initializes email, password fields, and login/signup buttons. Upon clicking the **Login** button, the app retrieves user input and validates it against saved credentials in Shared Preferences. If the credentials match, the user is redirected to the MainActivity. Otherwise, a toast message informs them of invalid credentials, and the **Sign-Up** button is made visible for registration. The **Sign-Up** button navigates to SignUpActivity for user registration. This activity ensures secure and efficient user login handling, using local storage for credential management and providing a seamless navigation experience.

A screenshot of an IDE showing the LoginActivity.kt file. The code is in Kotlin and implements the LoginActivity class, which inherits from AppCompatActivity. It includes imports for DatabaseHelper, Toast, and Intent. The onCreate method initializes the dbHelper, finds the login and signup buttons, and sets their click listeners. The login button listener checks the user's credentials against the dbHelper and either starts MainActivity on success or shows a toast message on failure. The signup button listener starts the SignUpActivity.

```
1 package com.example.empowerworkoutapp
2
3 > import ...
4
11
12 class LoginActivity : AppCompatActivity() {
13
14     private lateinit var dbHelper: DatabaseHelper
15
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         setContentView(R.layout.activity_login)
19
20         dbHelper = DatabaseHelper(context = this)
21
22         val username = findViewById<EditText>(R.id.loginUsername)
23         val password = findViewById<EditText>(R.id.loginPassword)
24         val loginButton = findViewById<Button>(R.id.loginButton)
25         val signupRedirect = findViewById<Button>(R.id.signupRedirect)
26
27         loginButton.setOnClickListener {
28             val user = username.text.toString()
29             val pass = password.text.toString()
30
31             if (dbHelper.checkUser(user, pass)) {
32                 Toast.makeText(context = this, text = "Login Successful!", Toa
33                 startActivity(Intent(context = this, MainActivity::cla
34             } else {
35                 Toast.makeText(context = this, text = "Invalid Credentials!",
36             }
37         }
38
39         signupRedirect.setOnClickListener {
40             startActivity(Intent(context = this, SignUpActivity::class
41         }
```

Signup Activity

The `SignUpActivity` class facilitates user registration. It initializes input fields for name, email, password, and confirmation password, along with a **Sign Up** button. Upon clicking the button, the app validates the inputs to ensure completeness and matching passwords. If valid, the credentials are stored securely in `SharedPreferences`. After successful registration, the user is redirected to the `MainActivity`, and a success message is displayed. Input validation includes checking for empty fields and matching passwords. The `saveUserCredentials` function ensures persistent storage of user data, providing a secure and seamless onboarding process for new users in the app.

```
1 package com.example.empowerworkoutapp
2
3 > import ...
10
11 class SignUpActivity : AppCompatActivity() {
12
13     private lateinit var etName: EditText
14     private lateinit var etEmail: EditText
15     private lateinit var etPassword: EditText
16     private lateinit var etConfirmPassword: EditText
17     private lateinit var btnSignUp: Button
18
19     override fun onCreate(savedInstanceState: Bundle?) {
20         super.onCreate(savedInstanceState)
21         setContentView(R.layout.activity_sign_up)
22
23         // Initialize views
24         etName = findViewById(R.id.etName)
25         etEmail = findViewById(R.id.etEmail)
26         etPassword = findViewById(R.id.etPassword)
27         etConfirmPassword = findViewById(R.id.etConfirmPassword)
28         btnSignUp = findViewById(R.id.btnSignUp)
29
30         // Set up sign-up button click listener
31         btnSignUp.setOnClickListener {
32             val name = etName.text.toString().trim()
33             val email = etEmail.text.toString().trim()
34             val password = etPassword.text.toString().trim()
35             val confirmPassword = etConfirmPassword.text.toString().trim()
36
37             if (validateInputs(name, email, password, confirmPassword)) {
38                 // Save the user credentials
39                 saveUserCredentials(email, password)
40
41                 // Navigate to MainActivity after successful sign-up
42                 val intent = Intent(packageContext, MainActivity::class.java)
43                 startActivity(intent)
44                 finish()
45             }
46         }
47     }
48
49     // Function to validate input fields
50     private fun validateInputs(name: String, email: String, password: String, confirmPassword: String): Boolean {
```


Home Fragment Activity

The HomeFragment class displays a weekly workout schedule using a RecyclerView. It inflates its layout using FragmentHomeBinding for view binding. A sample scheduleList containing workout activities for each day, along with images, is created. The RecyclerView is initialized with a LinearLayoutManager for vertical scrolling and a custom ScheduleAdapter to bind the schedule data to the UI. This setup ensures a clean and user-friendly display of the workout plan for each day of the week. The fragment provides an organized and interactive way to navigate through workout schedules, enhancing the user experience within the app.

```
1 package com.example.empowerworkoutapp
2
3 > import ...
11
12 class HomeFragment : Fragment() {
13
14     private lateinit var binding: FragmentHomeBinding
15
16     override fun onCreateView(
17         inflater: LayoutInflater, container: ViewGroup?,
18         savedInstanceState: Bundle?
19     ): View? {
20         binding = FragmentHomeBinding.inflate(inflater, container, attachToParent: false)
21
22         // Sample data for the week
23         val scheduleList = listOf(
24             DaySchedule(day: "Monday", workout: "Chest + Triceps", R.drawable.chest),
25             DaySchedule(day: "Tuesday", workout: "Back + Biceps + Abs", R.drawable.back),
26             DaySchedule(day: "Wednesday", workout: "Shoulder + Legs", R.drawable.shoulder),
27             DaySchedule(day: "Thursday", workout: "Chest + Triceps", R.drawable.chest),
28             DaySchedule(day: "Friday", workout: "Back + Biceps + Abs", R.drawable.back),
29             DaySchedule(day: "Saturday", workout: "Shoulder + Legs", R.drawable.shoulder),
30             DaySchedule(day: "Sunday", workout: "Rest", R.drawable.rest)
31         )
32
33         // Set up RecyclerView
34         binding.recyclerView.layoutManager = LinearLayoutManager(requireContext())
35         binding.recyclerView.adapter = ScheduleAdapter(scheduleList)
36
37         return binding.root
38     }
39
40 }
```

Diet Meal Plan Fragment

The DietMealPlanFragment class displays vegetarian and non-vegetarian diet plans using a RecyclerView. It initializes two lists, vegDietPlan and nonVegDietPlan, containing meal details for each day with associated images. The fragment inflates its layout, setting up a RecyclerView with a LinearLayoutManager and a DietPlanAdapter (defaulting to vegetarian meals). Two buttons, vegButton and nonVegButton, allow users to toggle between the diet plans. Clicking a button updates the adapter's data with the corresponding diet plan. This interactive setup provides a dynamic and user-friendly way to explore meal options, enhancing the app's functionality for personalized diet planning.

```
1 package com.example.empowerworkoutapp
2
3 > import ...
11
12 <> class DietMealPlanFragment : Fragment() {  ± VELUGUNAVEEN
13
14     private val vegDietPlan = listOf(
15         DietPlan( day: "Monday", meal: "Vegetable Salad with Quinoa", R.drawable.veg_monday),
16         DietPlan( day: "Tuesday", meal: "Grilled Paneer with Brown Rice", R.drawable.veg_tuesday),
17         DietPlan( day: "Wednesday", meal: "Vegetable Stir-fry with Tofu", R.drawable.veg_wednesday),
18         DietPlan( day: "Thursday", meal: "Chickpea Salad", R.drawable.veg_thursday),
19         DietPlan( day: "Friday", meal: "Lentil Soup with Bread", R.drawable.veg_friday),
20         DietPlan( day: "Saturday", meal: "Spinach and Cheese Wrap", R.drawable.veg_saturday),
21         DietPlan( day: "Sunday", meal: "Fruit Smoothie Bowl", R.drawable.veg_sunday)
22     )
23
24     private val nonVegDietPlan = listOf(
25         DietPlan( day: "Monday", meal: "Grilled Chicken with Vegetables", R.drawable.nonveg_monday),
26         DietPlan( day: "Tuesday", meal: "Fish Curry with Rice", R.drawable.nonveg_tuesday),
27         DietPlan( day: "Wednesday", meal: "Egg Salad Sandwich", R.drawable.nonveg_wednesday),
28         DietPlan( day: "Thursday", meal: "Turkey Wrap with Veggies", R.drawable.nonveg_thursday),
29         DietPlan( day: "Friday", meal: "Chicken Soup with Bread", R.drawable.nonveg_friday),
30         DietPlan( day: "Saturday", meal: "Shrimp Stir-fry with Rice", R.drawable.nonveg_saturday),
31         DietPlan( day: "Sunday", meal: "Grilled Salmon with Salad", R.drawable.nonveg_sunday)
32     )
33
34     override fun onCreateView(  ± VELUGUNAVEEN
35         inflater: LayoutInflater, container: ViewGroup?,
36         savedInstanceState: Bundle?
37     ): View? {
38         val view = inflater.inflate(R.layout.activity_diet_meal_plan_fragment, container, attachToRoot: false)
39
40         val recyclerView = view.findViewById<RecyclerView>(R.id.dietPlanRecyclerView)
41         val vegButton = view.findViewById<Button>(R.id.vegButton)
```

> com > example > empowerworkoutapp > DietMealPlanFragment > nonVegDietPlan 26:45 LF UTF-8 4 spaces

WorkoutPlanFragment

The WorkoutPlanFragment dynamically displays a daily workout and corresponding diet plans based on the current day. It uses predefined maps (workoutData, vegDietData, and nonVegDietData) to associate each day of the week with a workout routine, vegetarian meal, and non-vegetarian meal, along with their respective images. The onCreateView method inflates the fragment's layout and initializes UI elements, such as TextView and ImageView. The getCurrentDay function determines the current day of the week using the Calendar class. The workout and diet details for the current day are fetched from the maps and displayed, creating a personalized user experience.

```
package com.example.empowerworkoutapp

import ...

class WorkoutPlanFragment : Fragment() {
    private val workoutData = mapOf(
        "Monday" to Pair("Chest + Triceps", R.drawable.chest),
        "Tuesday" to Pair("Back + Biceps + Abs", R.drawable.back),
        "Wednesday" to Pair("Shoulder + Legs", R.drawable.shoulder),
        "Thursday" to Pair("Chest + Triceps", R.drawable.chest),
        "Friday" to Pair("Back + Biceps + Abs", R.drawable.back),
        "Saturday" to Pair("Shoulder + Legs", R.drawable.shoulder),
        "Sunday" to Pair("Rest", R.drawable.rest)
    )

    private val vegDietData = mapOf(
        "Monday" to Pair("Vegetable Salad with Quinoa", R.drawable.veg_monday),
        "Tuesday" to Pair("Grilled Paneer with Brown Rice", R.drawable.veg_tuesday),
        "Wednesday" to Pair("Vegetable Stir-fry with Tofu", R.drawable.veg_wednesday),
        "Thursday" to Pair("Chickpea Salad", R.drawable.veg_thursday),
        "Friday" to Pair("Lentil Soup with Bread", R.drawable.veg_friday),
        "Saturday" to Pair("Spinach and Cheese Wrap", R.drawable.veg_saturday),
        "Sunday" to Pair("Fruit Smoothie Bowl", R.drawable.veg_sunday)
    )

    private val nonVegDietData = mapOf(
        "Monday" to Pair("Grilled Chicken with Vegetables", R.drawable.nonveg_monday),
        "Tuesday" to Pair("Fish Curry with Rice", R.drawable.nonveg_tuesday),
        "Wednesday" to Pair("Egg Salad Sandwich", R.drawable.nonveg_wednesday),
        "Thursday" to Pair("Turkey Wrap with Veggies", R.drawable.nonveg_thursday),
        "Friday" to Pair("Chicken Soup with Bread", R.drawable.nonveg_friday),
        "Saturday" to Pair("Shrimp Stir-fry with Rice", R.drawable.nonveg_saturday),
    )
}
```

DietPlanAdapter

The `DietPlanAdapter` is a custom adapter for populating a `RecyclerView` with diet plans. It manages a list of `DietPlan` objects, each containing a day, meal description, and an image resource. The `DietPlanViewHolder` class binds views (day, meal, and image) to their respective layout elements. The `onCreateViewHolder` inflates the item layout, while `onBindViewHolder` sets the data for each item in the list based on its position. The `getItemCount` method returns the size of the list. The `updateData` function allows updating the dataset dynamically, notifying the `RecyclerView` to refresh and reflect changes in the UI.

```

1 package com.example.empowerworkoutapp
2
3 > import ...
4
5
6
7
8
9
10 class DietPlanAdapter(private var dietPlanList: List<DietPlan>) : RecyclerView.Adapter<DietPlanAdapter.DietPlanViewHolder>() {
11     RecyclerView.Adapter<DietPlanAdapter.DietPlanViewHolder>() {
12
13     class DietPlanViewHolder(view: View) : RecyclerView.ViewHolder(view) {
14         val dayTextView: TextView = view.findViewById(R.id.dayTextView)
15         val mealTextView: TextView = view.findViewById(R.id.mealTextView)
16         val mealImageView: ImageView = view.findViewById(R.id.mealImageView)
17     }
18
19     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DietPlanViewHolder {
20         val view = LayoutInflater.from(parent.context)
21             .inflate(R.layout.item_diet_plan, parent, attachToRoot = false)
22         return DietPlanViewHolder(view)
23     }
24
25     override fun onBindViewHolder(holder: DietPlanViewHolder, position: Int) {
26         val dietPlan = dietPlanList[position]
27         holder.dayTextView.text = dietPlan.day
28         holder.mealTextView.text = dietPlan.meal
29         holder.mealImageView.setImageResource(dietPlan.imageResId)
30     }
31
32     override fun getItemCount() = dietPlanList.size
33
34     // Update data and notify RecyclerView
35     fun updateData(newDietPlanList: List<DietPlan>) {
36         dietPlanList = newDietPlanList
37         notifyDataSetChanged()
38     }
39 }

```

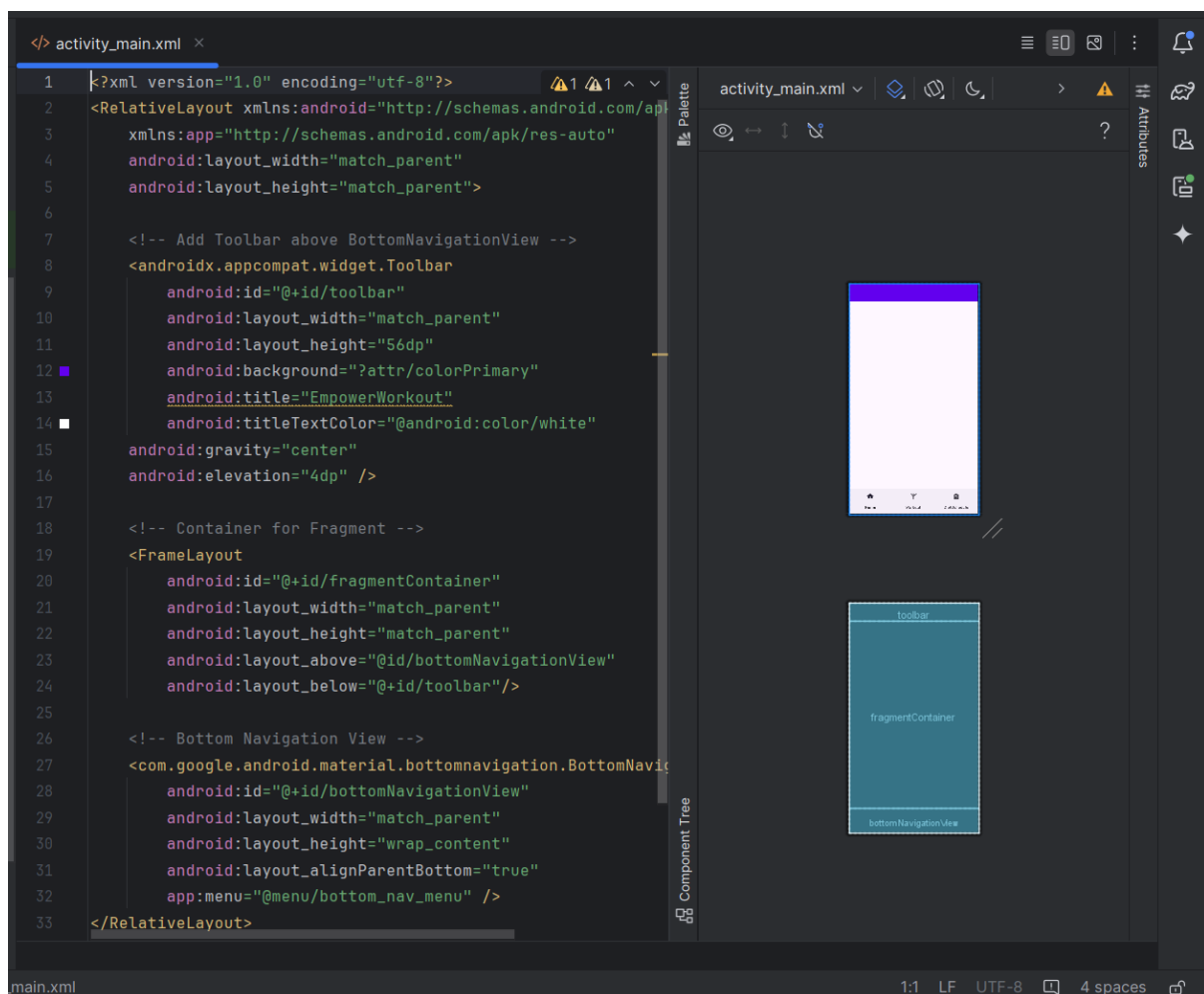
Activity Wise Xml & design

activity-main.xml

This XML layout defines a screen using a RelativeLayout with three main components: a Toolbar, a FrameLayout, and a BottomNavigationView.

The Toolbar at the top serves as the app's title bar, styled with a primary color and centered text. Below it, the FrameLayout acts as a container for displaying fragments dynamically, allowing different screens to be loaded based on user interaction. At the bottom, the BottomNavigationView provides a navigation bar for switching between fragments, linked to a menu resource (bottom_nav_menu). The layout ensures the Toolbar and navigation bar are fixed, with the fragment content dynamically displayed between them.

40

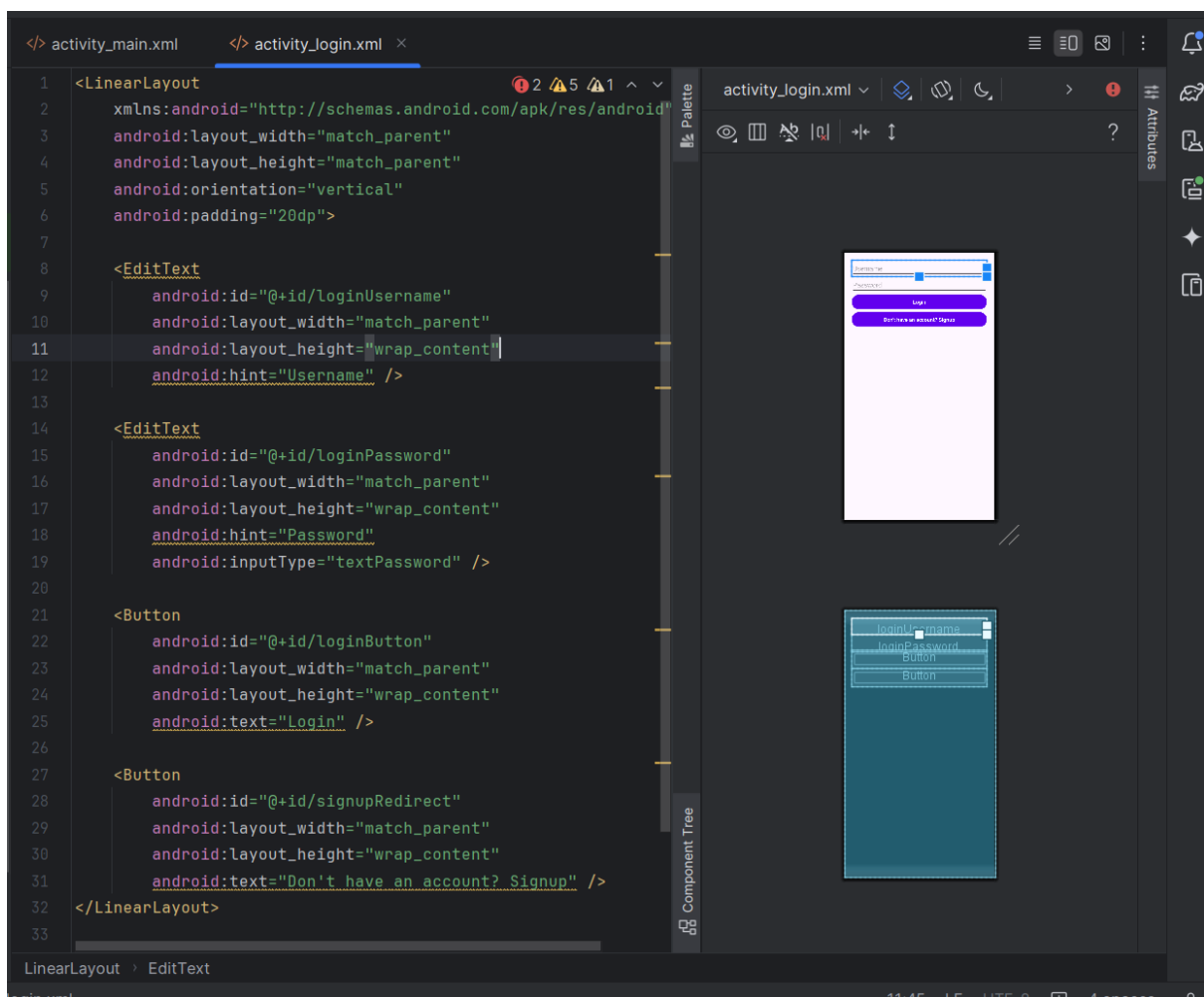


activity_login.xml

This XML layout uses a `LinearLayout` with vertical orientation to create a simple login screen. It includes:

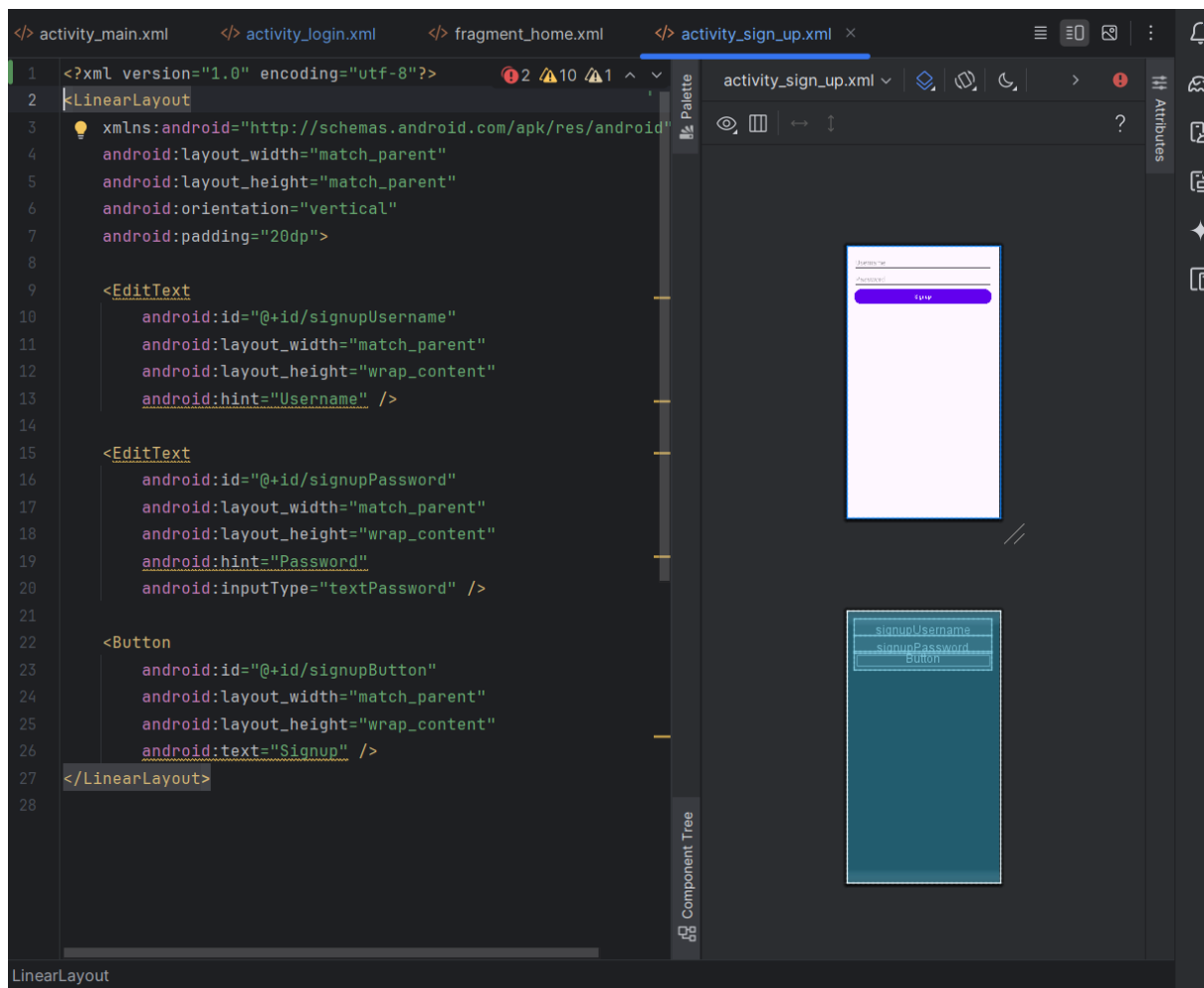
1. **Username Field** (`EditText` with `id=loginUsername`): Allows users to input their username, styled with a placeholder text ("Username").
2. **Password Field** (`EditText` with `id=loginPassword`): Collects user passwords, with input type set to `textPassword` to mask input for privacy.
3. **Login Button** (`Button` with `id=loginButton`): Enables users to log in after entering credentials.
4. **Signup Button** (`Button` with `id=signupRedirect`): Redirects users to a sign-up screen if they don't have an account.

The layout is padded (20dp) for spacing and ensures components align vertically.



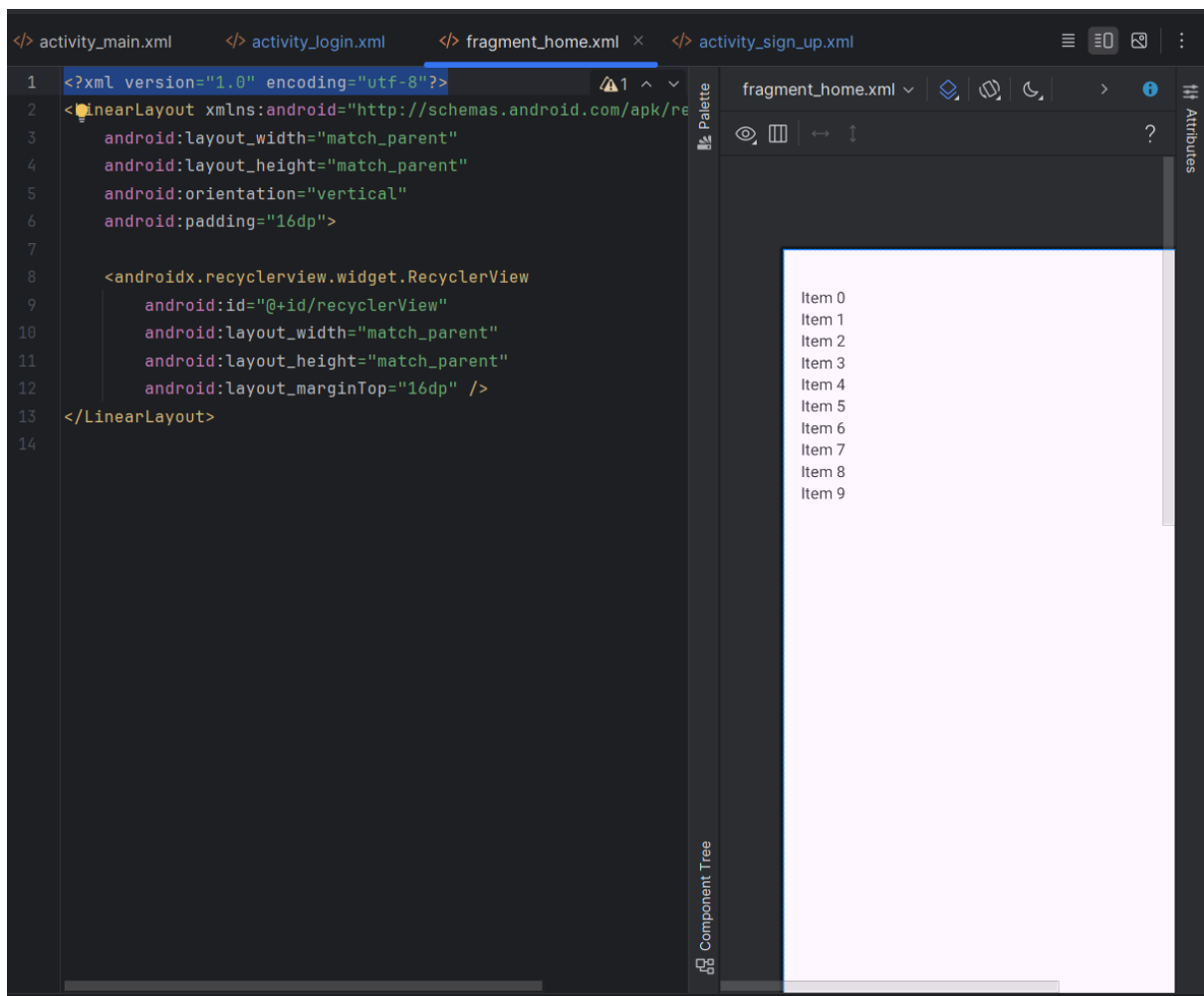
activity_sign_up.xml

This XML layout defines a basic sign-up screen using a `LinearLayout` with vertical orientation and padding for spacing. It contains two `EditText` fields and a `Button`. The first `EditText` (`id=signupUsername`) allows users to input their username, with a placeholder hint for guidance. The second `EditText` (`id=signupPassword`) is for password input, with `inputType="textPassword"` ensuring secure text entry. Below these fields, a `Button` (`id=signupButton`) is provided for the user to submit their sign-up information. The layout is simple and user-friendly, making it ideal for a minimalistic sign-up interface in an Android application.



Fragment_home_xml

This XML layout defines a screen using a `LinearLayout` with vertical orientation and padding of 16dp for spacing. It contains a single `RecyclerView` widget, identified by `id=recyclerView`. The `RecyclerView` occupies the entire width and height of the parent layout (`match_parent` for both dimensions) and has a `layout_marginTop` of 16dp to provide spacing from the top. This layout is ideal for displaying scrollable lists or grids of data dynamically using adapters. The simple structure ensures flexibility for various content types, making it suitable for creating list-based UI components in Android applications.

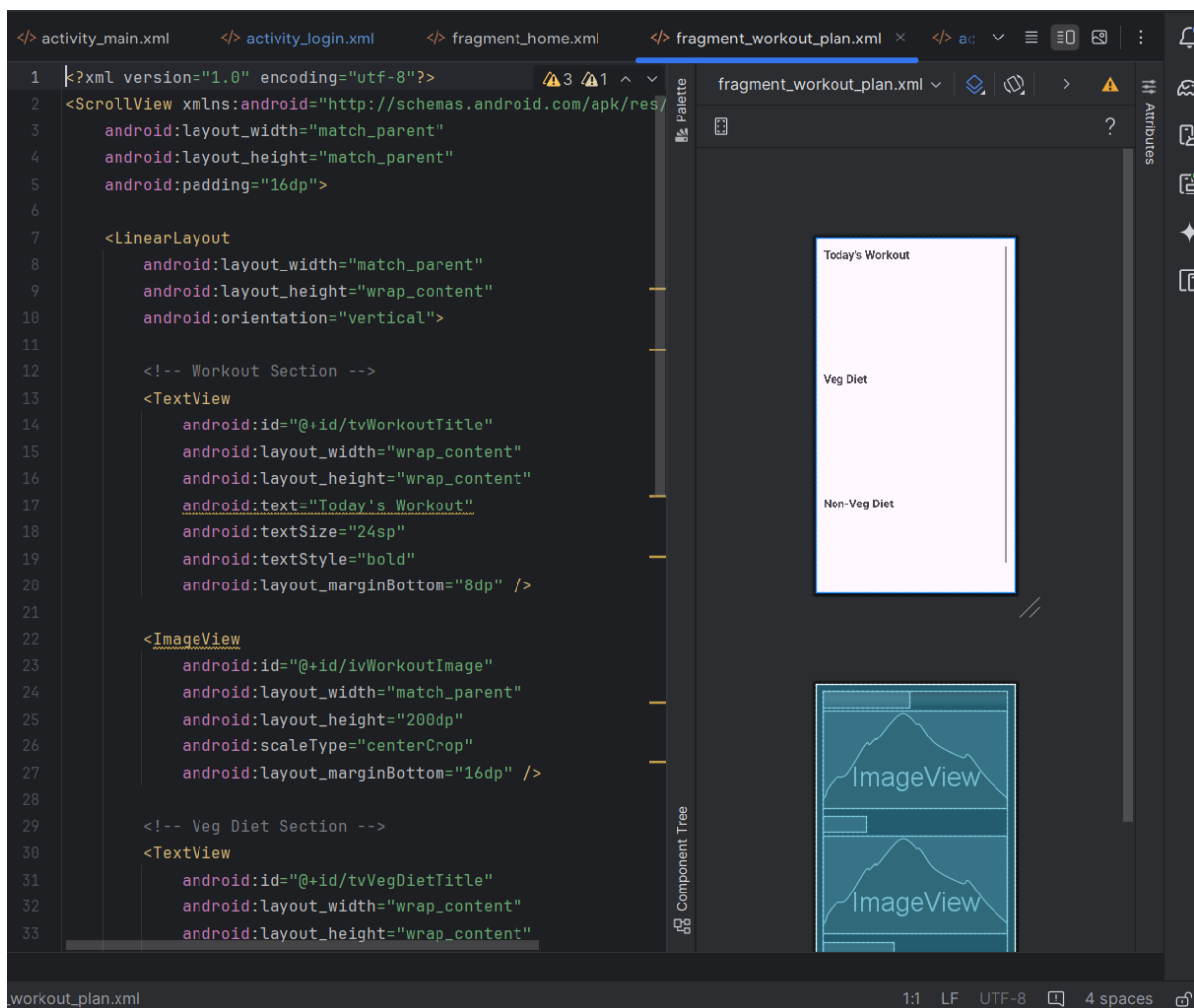


fragment_workout_plan.xml

This XML code defines a **ScrollView** that contains a **LinearLayout** to display a structured UI for a fitness app. Inside the layout:

1. **Workout Section:** Includes a **TextView** titled "Today's Workout" and an **ImageView** for a workout image.
2. **Veg Diet Section:** A **TextView** for "Veg Diet" and an **ImageView** to display related content.
3. **Non-Veg Diet Section:** Similar structure with a "Non-Veg Diet" **TextView** and **ImageView**.

The **ScrollView** ensures the content is scrollable if it exceeds the screen height. Margins and padding provide spacing for better UI readability.

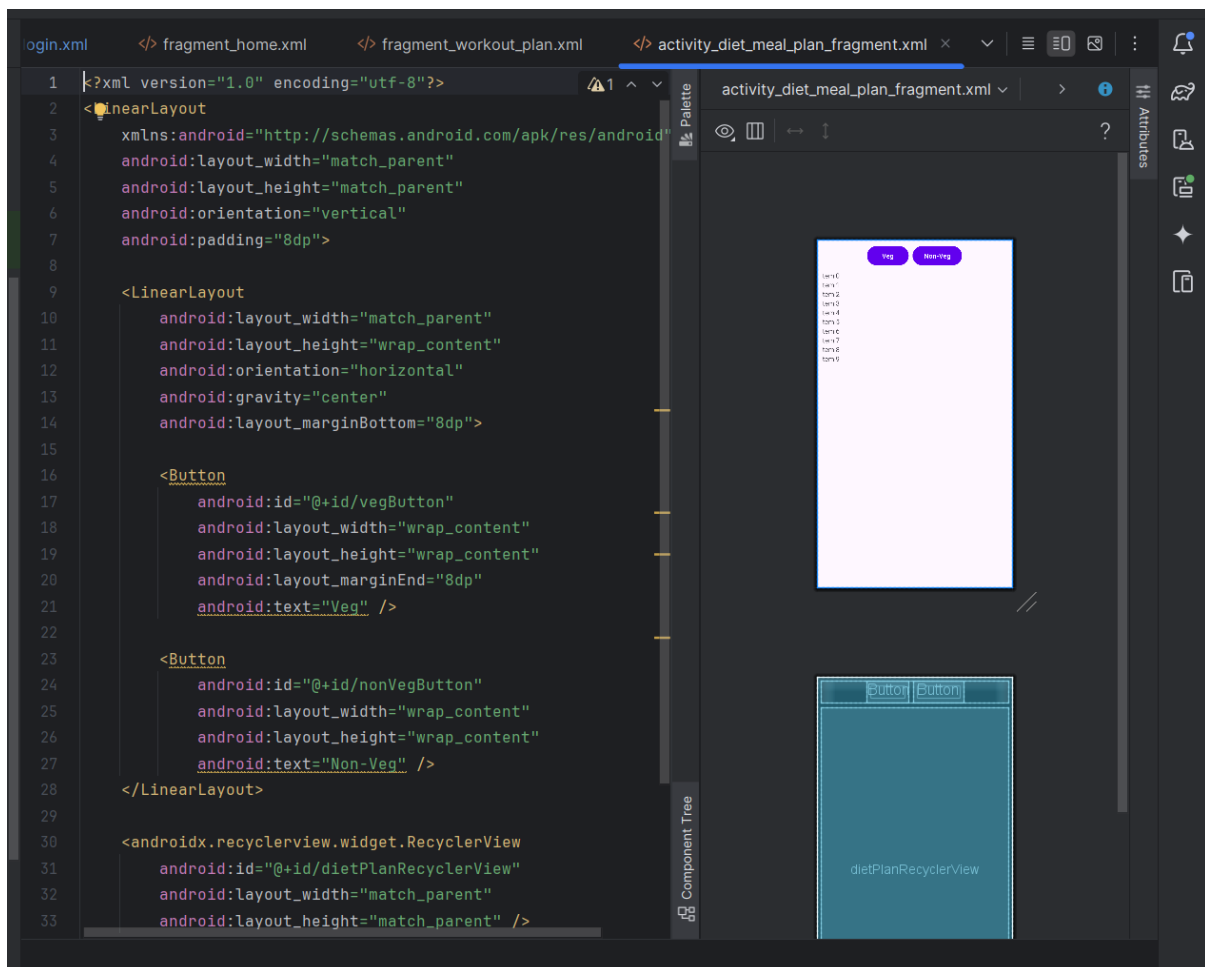


activity_diet_meal_plan_fragment.xml

This XML layout defines a **LinearLayout** for a diet plan app, with a vertical orientation. It contains:

1. **Button Section:** A horizontal **LinearLayout** with two buttons ("Veg" and "Non-Veg"), allowing users to toggle between diet plans. The buttons are centered and have spacing via `layout_marginEnd`.
2. **RecyclerView Section:** An `androidx.recyclerview.widget.RecyclerView` below the buttons displays a scrollable list of diet plan items, dynamically populated at runtime.

The outer **LinearLayout** uses padding for spacing, and the **RecyclerView** takes up the remaining screen space for a seamless scrollable experience. This layout is efficient for displaying filterable list content.

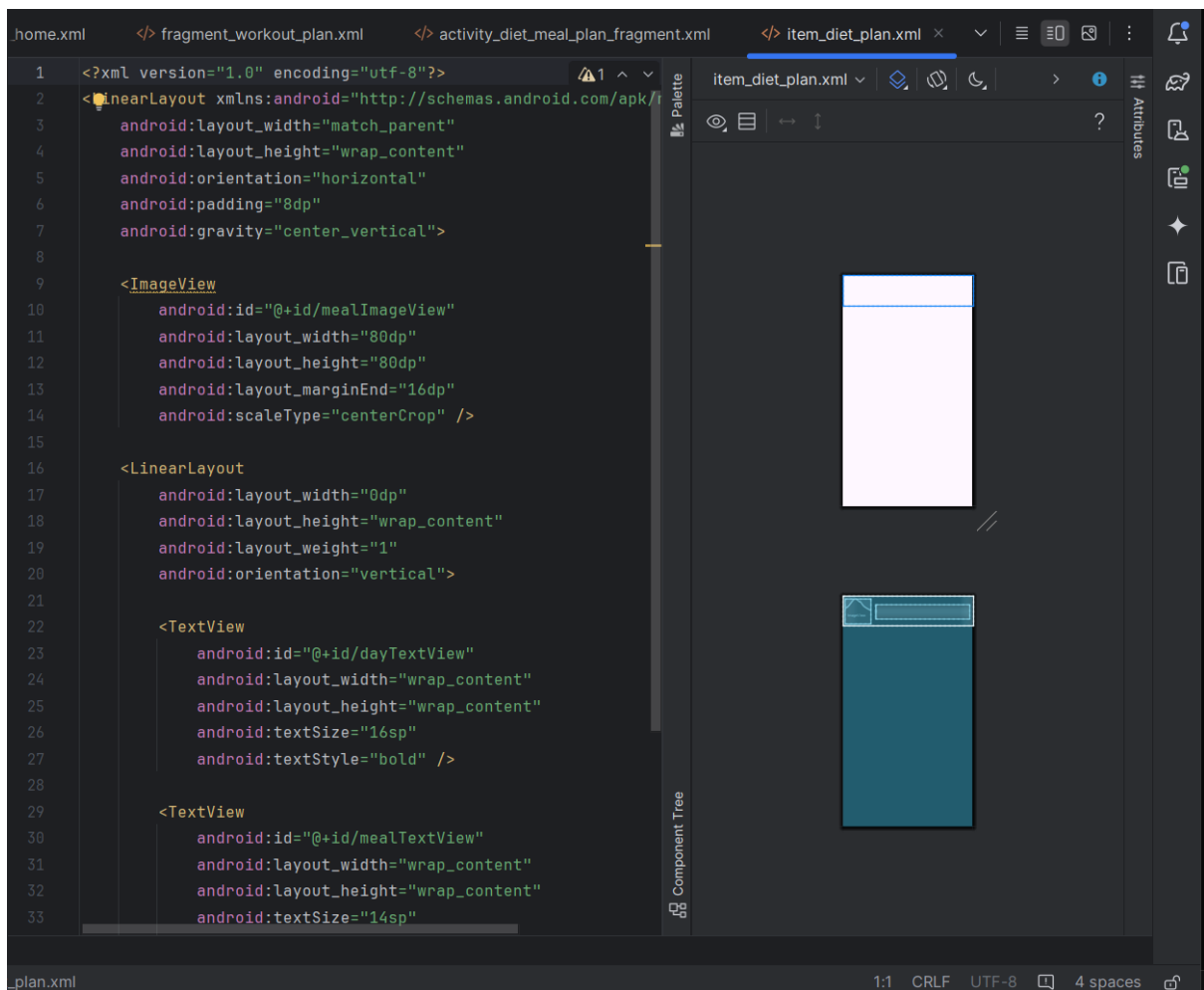


item_diet_plan.xml

This XML defines a **LinearLayout** for a horizontal list item, ideal for a diet plan or meal app. It includes:

1. **ImageView**: Displays an 80x80 dp image (e.g., meal photo) with a margin to separate it from the text. The `scaleType="centerCrop"` ensures the image fits proportionally.
2. **Text Section**: A vertical **LinearLayout** (`layout_weight="1"`) contains:
 - **dayTextView**: Displays bold text for the day or category, styled with 16sp.
 - **mealTextView**: Displays descriptive text about the meal, styled with 14sp and a light gray color.

The layout is optimized for readability and space efficiency in horizontal list designs.

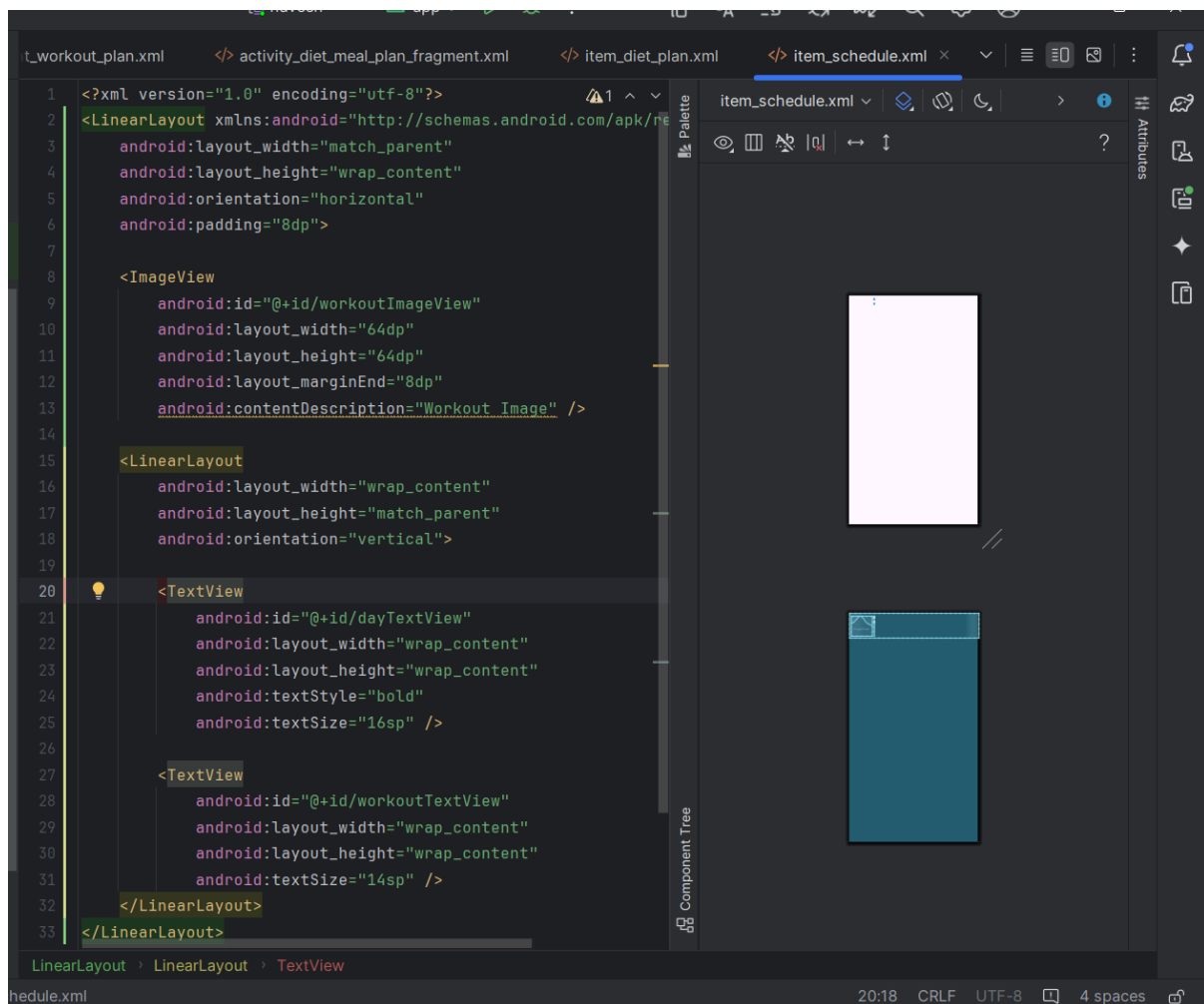


item_schedule.xml

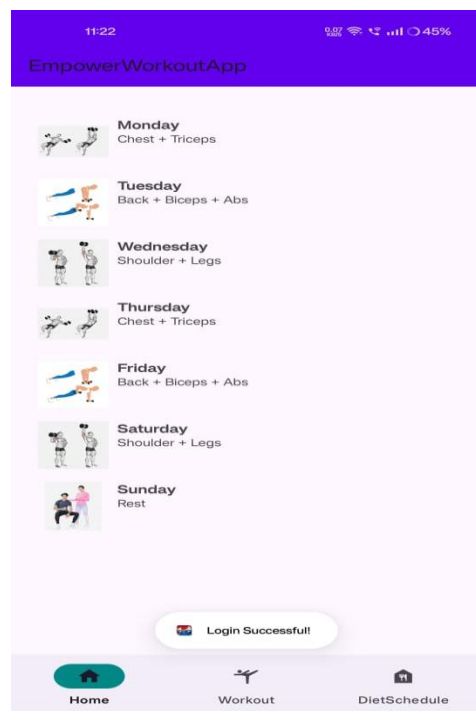
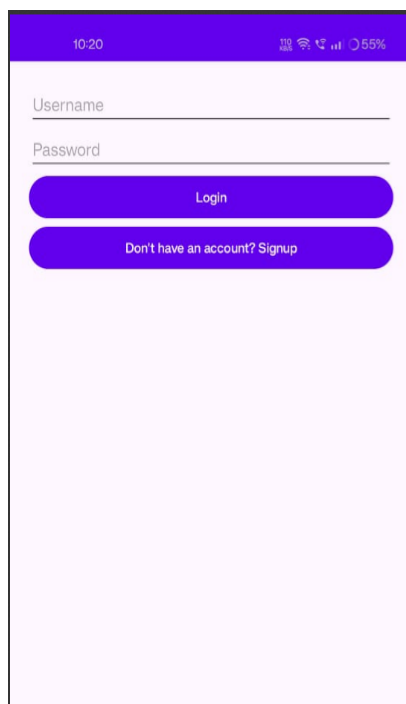
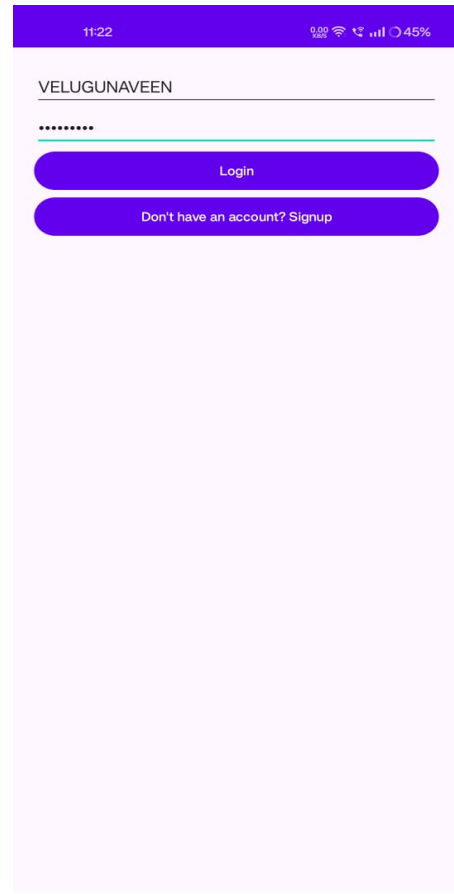
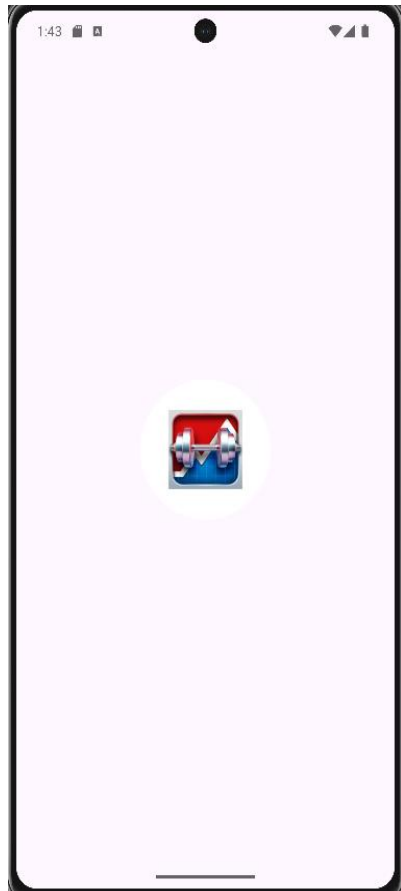
This XML defines a **LinearLayout** for a horizontal list item, commonly used in workout or fitness apps. Key components:

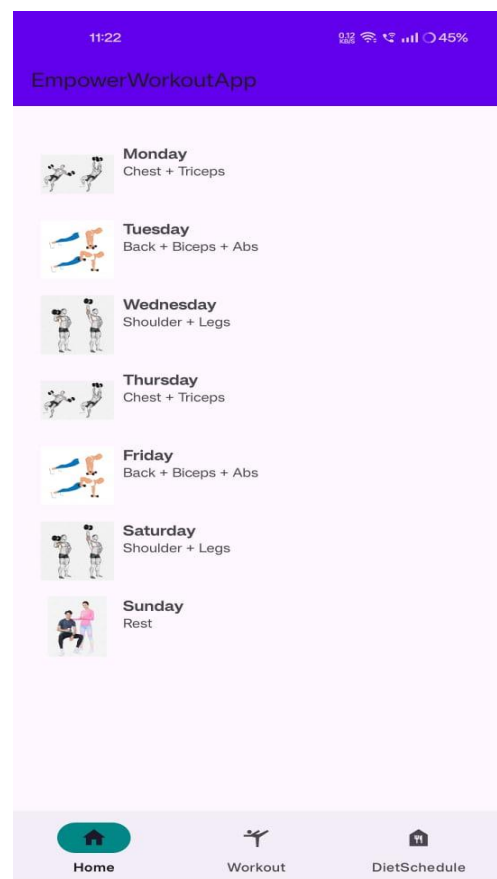
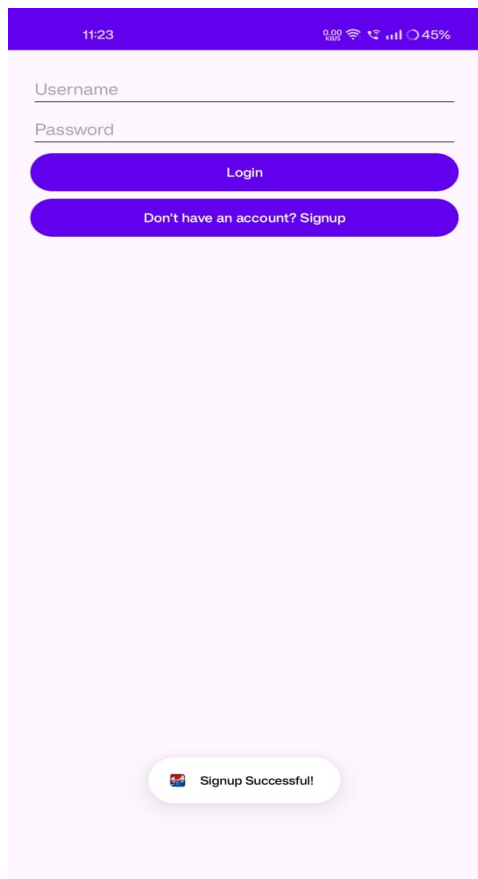
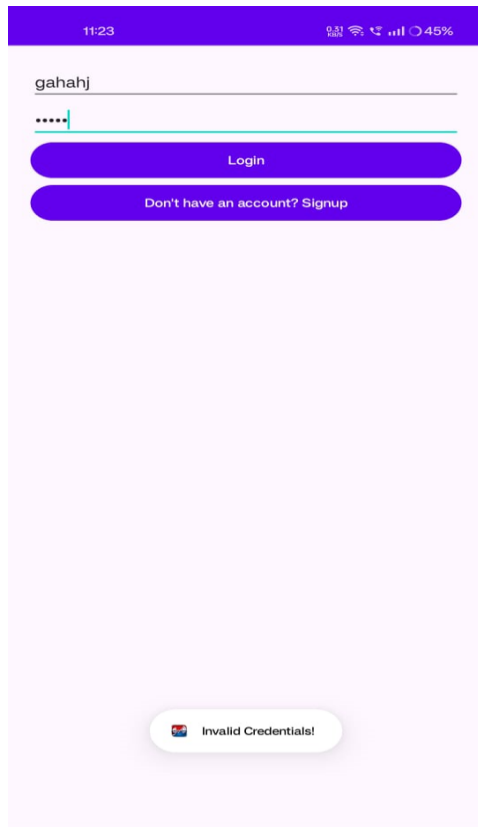
1. **ImageView**: A 64x64 dp image, placed on the left with `layout_marginEnd="8dp"` for spacing. It serves as a placeholder for workout-related visuals. The `contentDescription` improves accessibility.
2. **Text Section**: A vertical **LinearLayout** containing:
 - **dayTextView**: Displays bold text (e.g., the day of the workout), styled with 16sp.
 - **workoutTextView**: Displays supporting text (e.g., workout details) with 14sp for secondary emphasis.

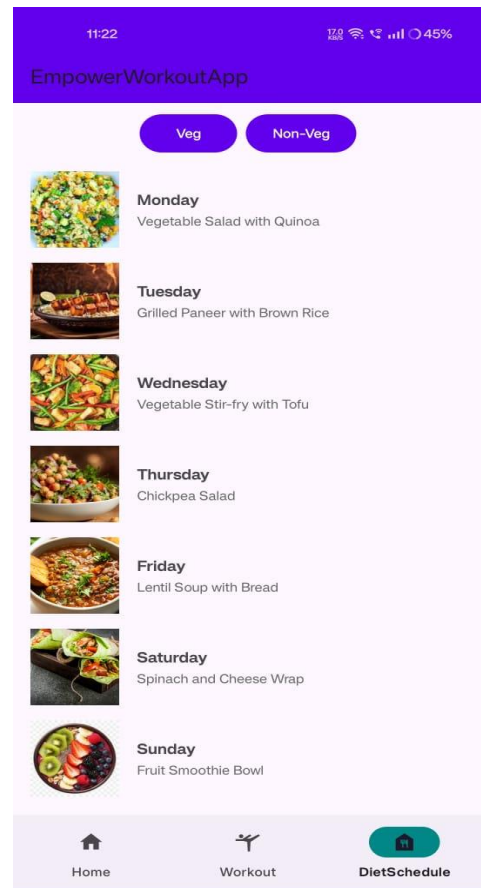
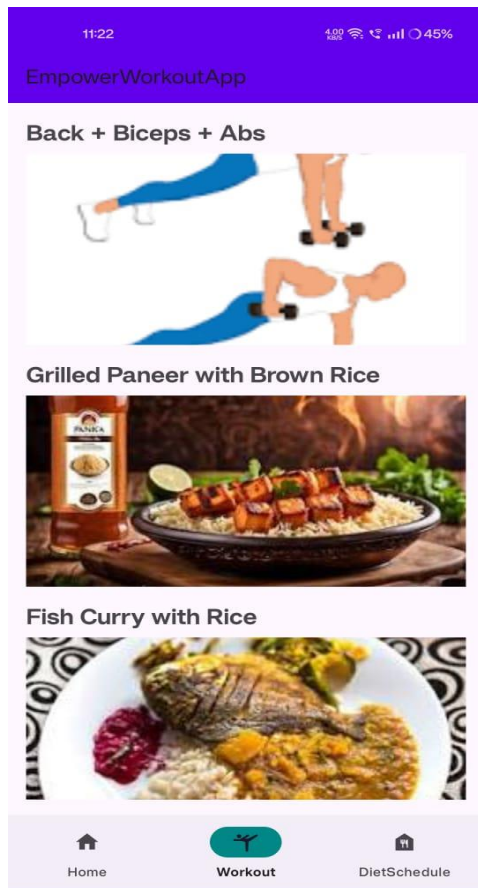
The layout is compact and well-organized, making it suitable for list views or RecyclerView items in fitness apps.



Emulator Screenshots







Conclusion

Empower Workout is a well-rounded fitness and lifestyle app designed to help users maintain a healthy and disciplined routine. With its intuitive design and versatile functionality, it effectively caters to workout planning, diet management, and goal tracking for both beginners and fitness enthusiasts.

The app integrates features like a login and sign-up system to personalize user experiences. New users can create accounts through the **Sign-Up module**, while existing users can securely log in to their accounts, ensuring privacy and data safety through the use of SharedPreferences. These credentials provide seamless access to workout plans and diet schedules.

The app's core functionality revolves around the **Home, Workout Plan, and Diet Meal Plan fragments**. The **HomeFragment** offers users a weekly overview of their exercise regimen, neatly presented in a RecyclerView, ensuring ease of navigation. The **WorkoutPlanFragment** dynamically adjusts to display daily exercise routines tailored for specific muscle groups or rest days, integrating visually appealing images for better user engagement.

Similarly, the **DietMealPlanFragment** provides both vegetarian and non-vegetarian diet options. Users can effortlessly switch between plans with the click of a button. The integration of RecyclerView with adaptable data makes it highly interactive, offering meal visuals and descriptions to inspire healthier eating habits.

The user interface is designed for clarity and usability. The **BottomNavigationView** ensures easy access to major features, while the **Toolbar** adds branding and a professional touch. Key layout elements like buttons and input fields in login and sign-up screens are intuitively positioned, enhancing usability.

Empower Workout emphasizes simplicity, functionality, and flexibility in promoting healthier lifestyles. It acts as a personal fitness guide, helping users achieve their fitness and nutritional goals efficiently.

Empower Workout is a well-rounded fitness and lifestyle app designed to help users maintain a healthy and disciplined routine. With its intuitive design and versatile functionality, it effectively caters to workout planning, diet management, and goal tracking for both beginners and fitness enthusiasts.

The app integrates features like a login and sign-up system to personalize user experiences. New users can create accounts through the **Sign-Up module**, while existing users can securely log in to their accounts, ensuring privacy and data safety through the use of SharedPreferences. These credentials provide seamless access to workout plans and diet schedules.

The app's core functionality revolves around the **Home, Workout Plan, and Diet Meal Plan fragments**. The **HomeFragment** offers users a weekly overview of their exercise regimen, neatly presented in a RecyclerView, ensuring ease of navigation. The **WorkoutPlanFragment** dynamically adjusts to display daily exercise routines tailored for specific muscle groups or rest days, integrating visually appealing images for better user engagement.

Similarly, the **DietMealPlanFragment** provides both vegetarian and non-vegetarian diet options. Users can effortlessly switch between plans with the click of a button. The integration of RecyclerView with adaptable data makes it highly interactive, offering meal visuals and descriptions to inspire healthier eating habits.

The user interface is designed for clarity and usability. The **BottomNavigationView** ensures easy access to major features, while the **Toolbar** adds branding and a professional touch. Key layout elements like buttons and input fields in login and sign-up screens are intuitively positioned, enhancing usability.

Empower Workout emphasizes simplicity, functionality, and flexibility in promoting healthier lifestyles. It acts as a personal fitness guide, helping users achieve their fitness and nutritional goals efficiently.

Project GitHub link: -

<https://github.com/VELUGUNAVEEN/empowerworkoutapp.git>

