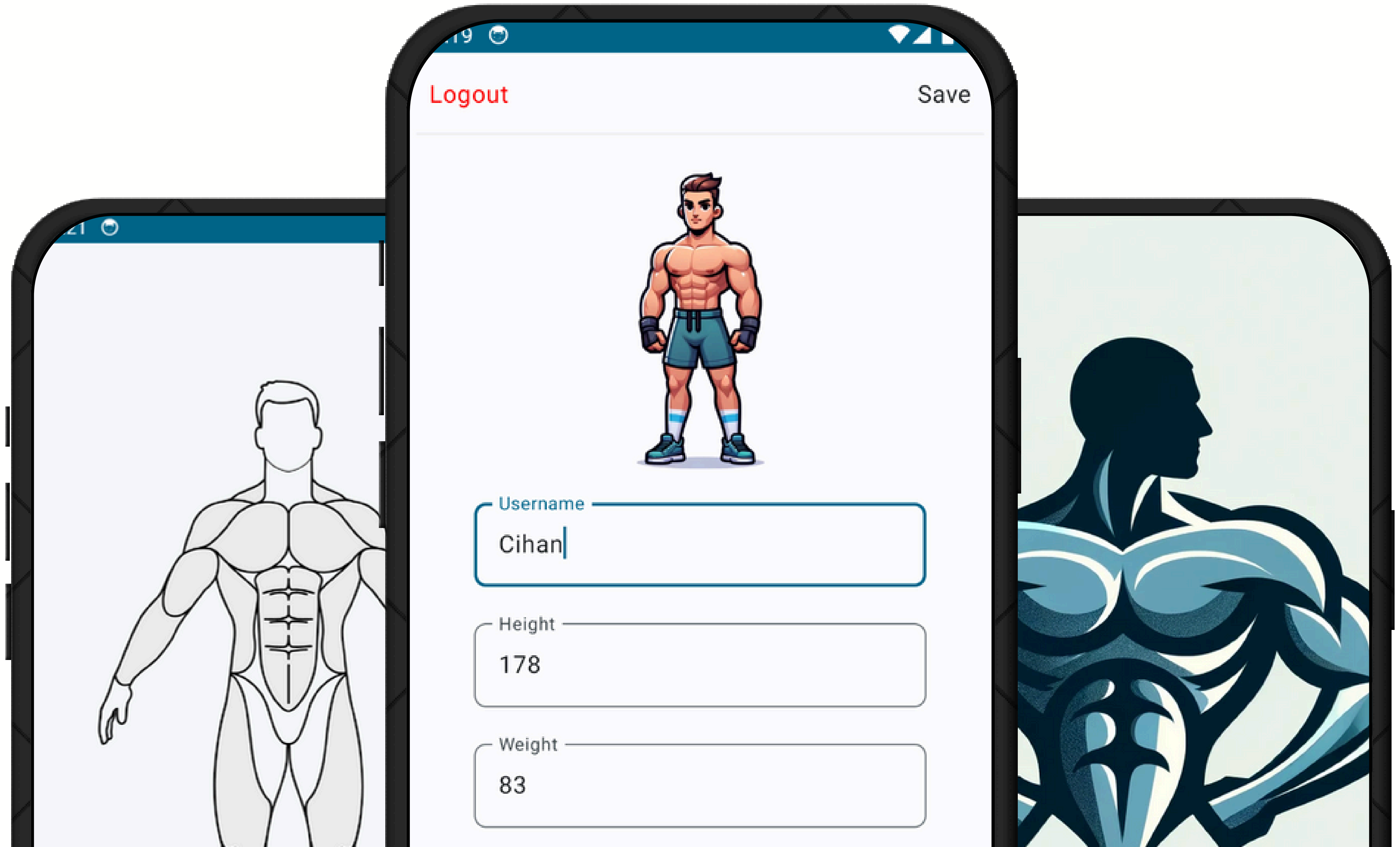
Arda Altınörs

ACM-431.3

https://github.com/ardaaltinors/MuscleMasterApp

Fitness Helper App

MuscleMaster



Logout

Save

Username

Cihan

Height

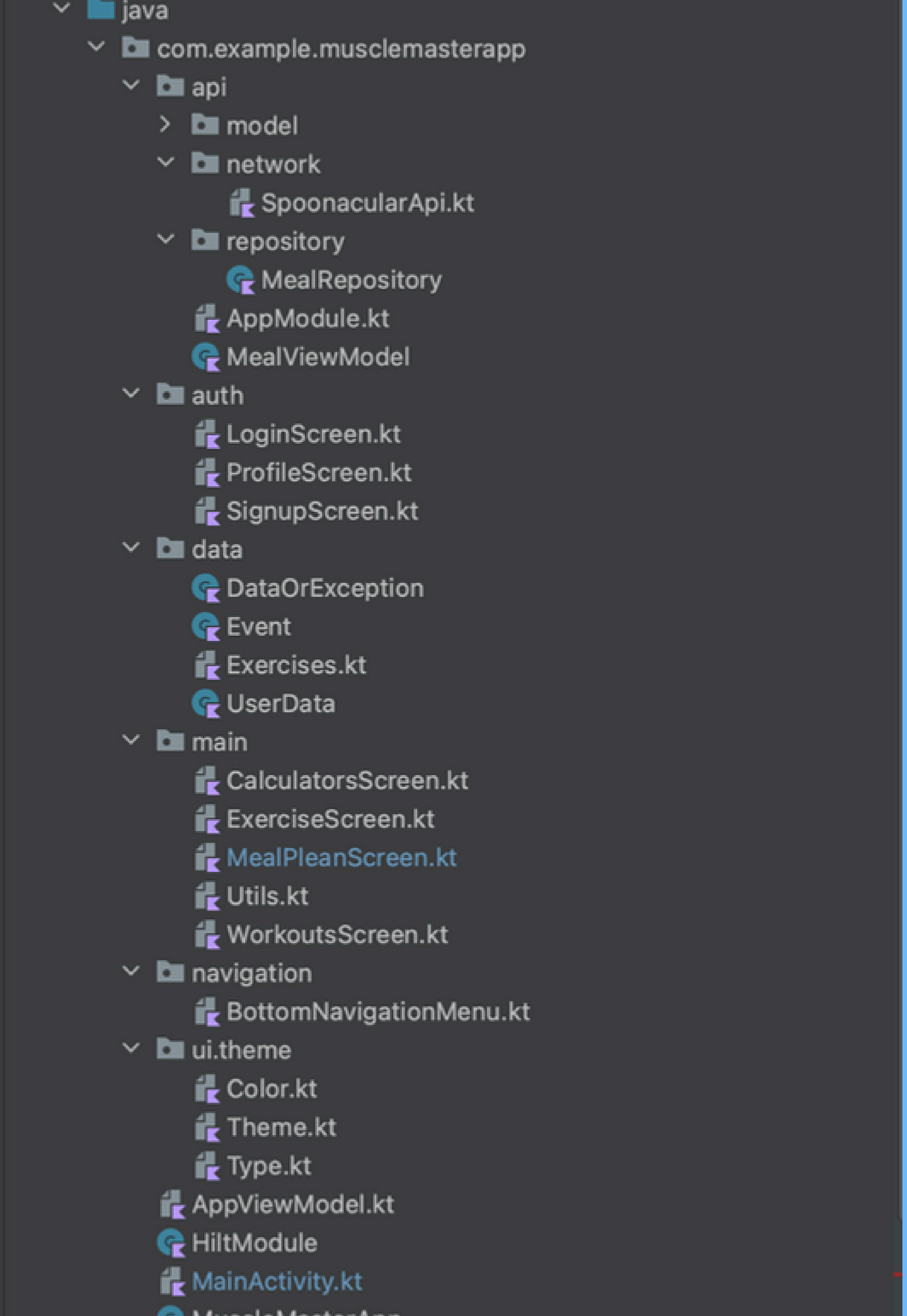
178

Weight

83

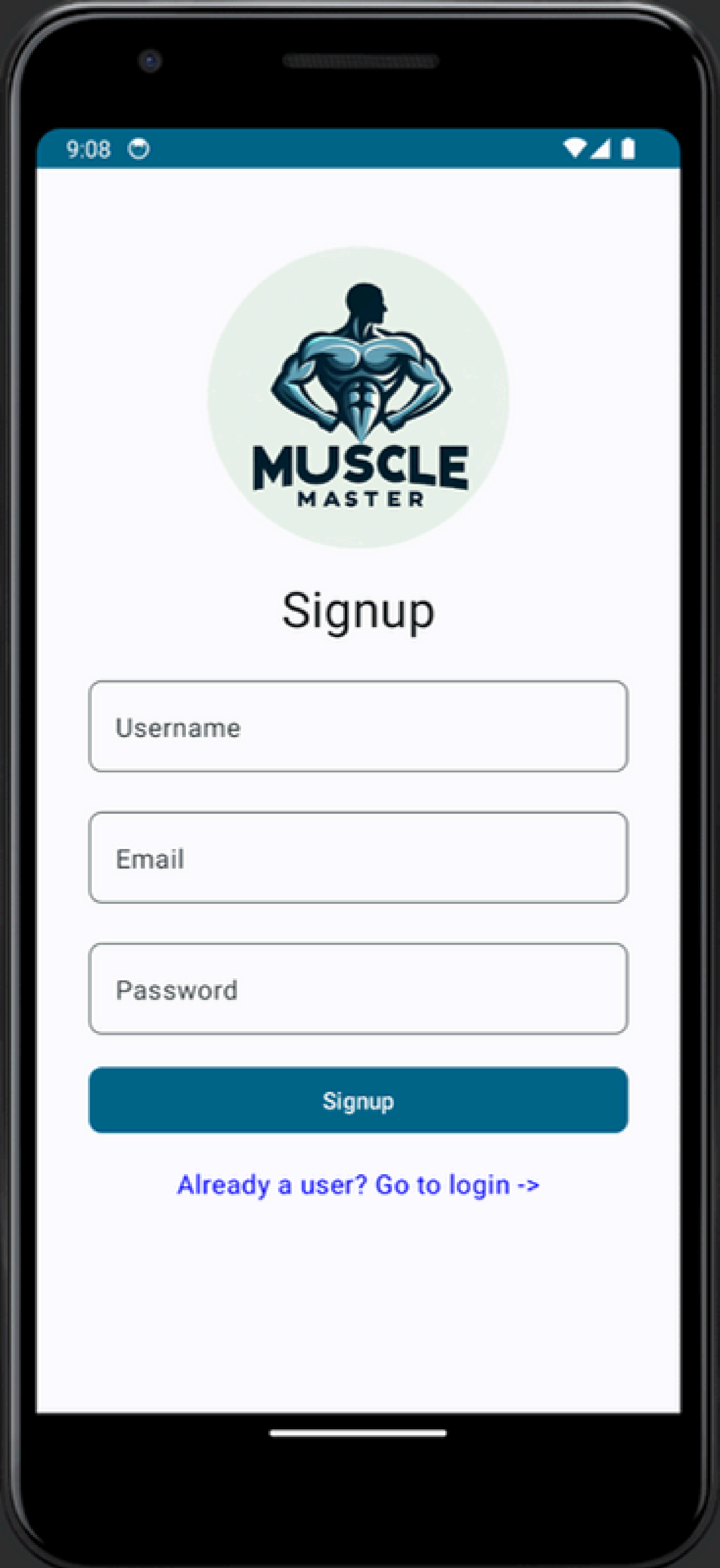
App Features:

- MuscleMaster is a fitness assistant application.
- Create a profile and enter information such as height and weight.
- Select the muscle group you want to work on and list the best exercises for it.
- View your Body Mass Index and daily calorie needs.
- Use the nutrition program we’ve created specially for you.



```
java
  com.example.musclemasterapp
    api
      model
      network
        SpoonacularApi.kt
      repository
        MealRepository
      AppModule.kt
      MealViewModel
    auth
      LoginScreen.kt
      ProfileScreen.kt
      SignupScreen.kt
    data
      DataOrException
      Event
      Exercises.kt
      UserData
    main
      CalculatorsScreen.kt
      ExerciseScreen.kt
      MealPleanScreen.kt
      Utils.kt
      WorkoutsScreen.kt
    navigation
      BottomNavigationMenu.kt
    ui.theme
      Color.kt
      Theme.kt
      Type.kt
    AppViewModel.kt
    HiltModule
    MainActivity.kt
```

We adopted the
MVVM architecture
to ensure a
modular and maintainable
codebase
for our application.



Signup

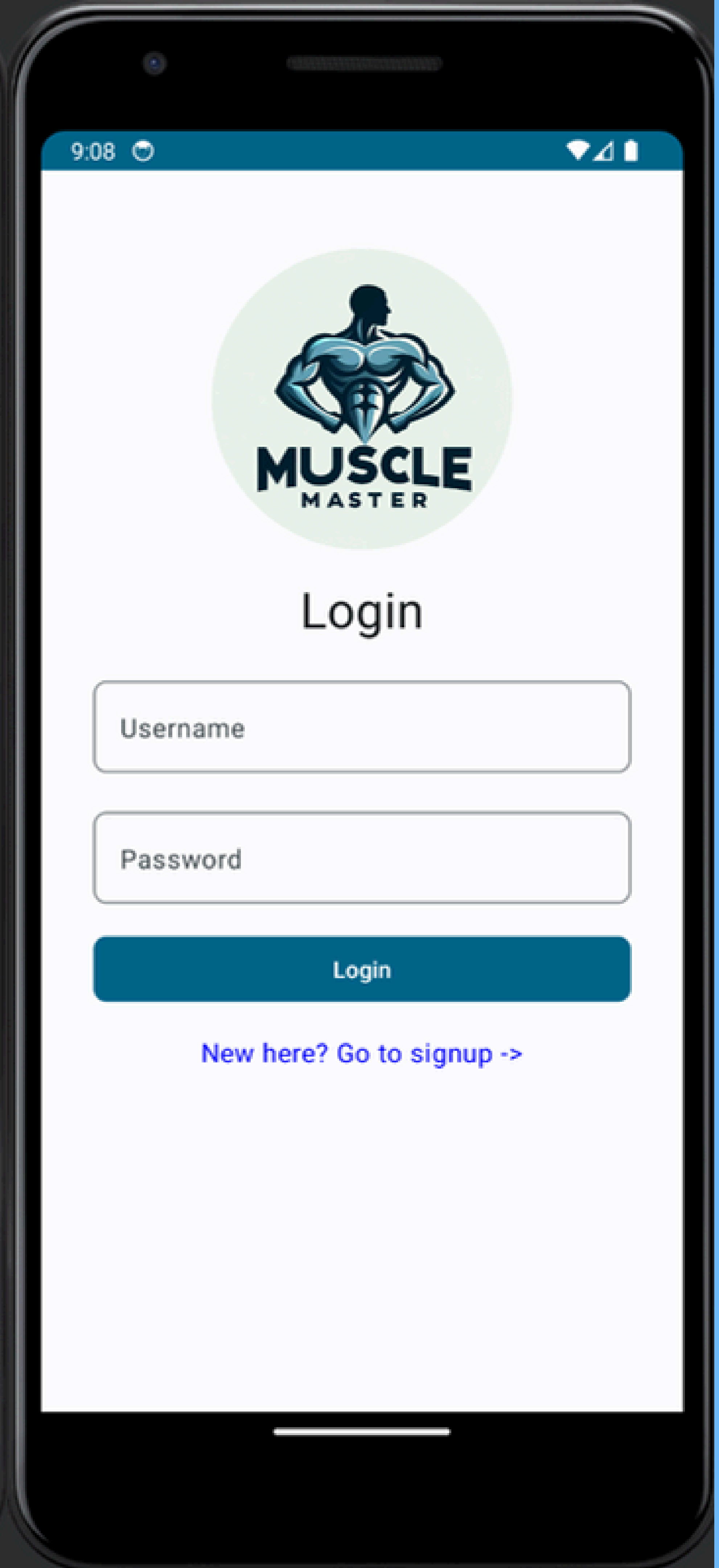
Username

Email

Password

Signup

Already a user? Go to login ->



Login

Username

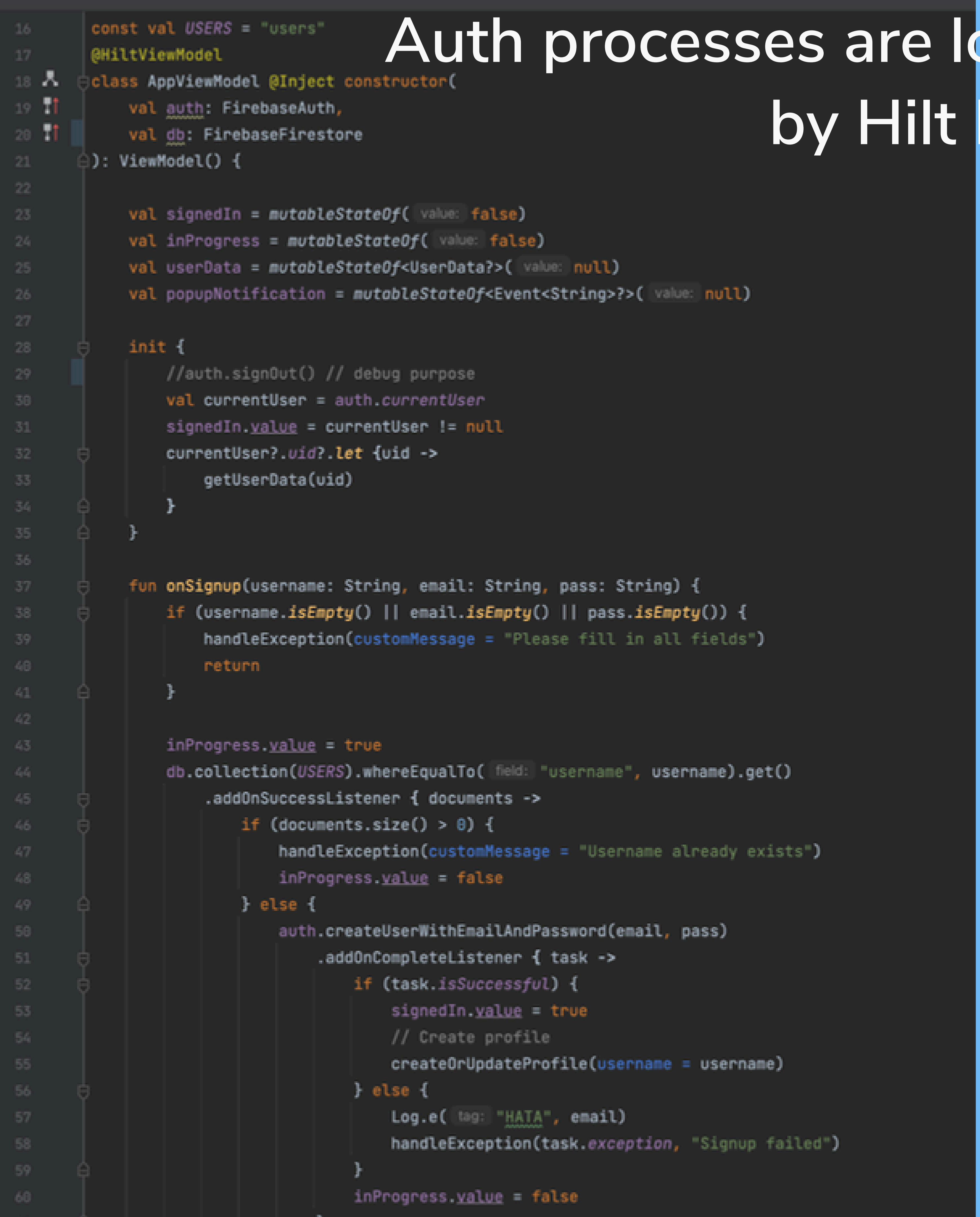
Password

Login

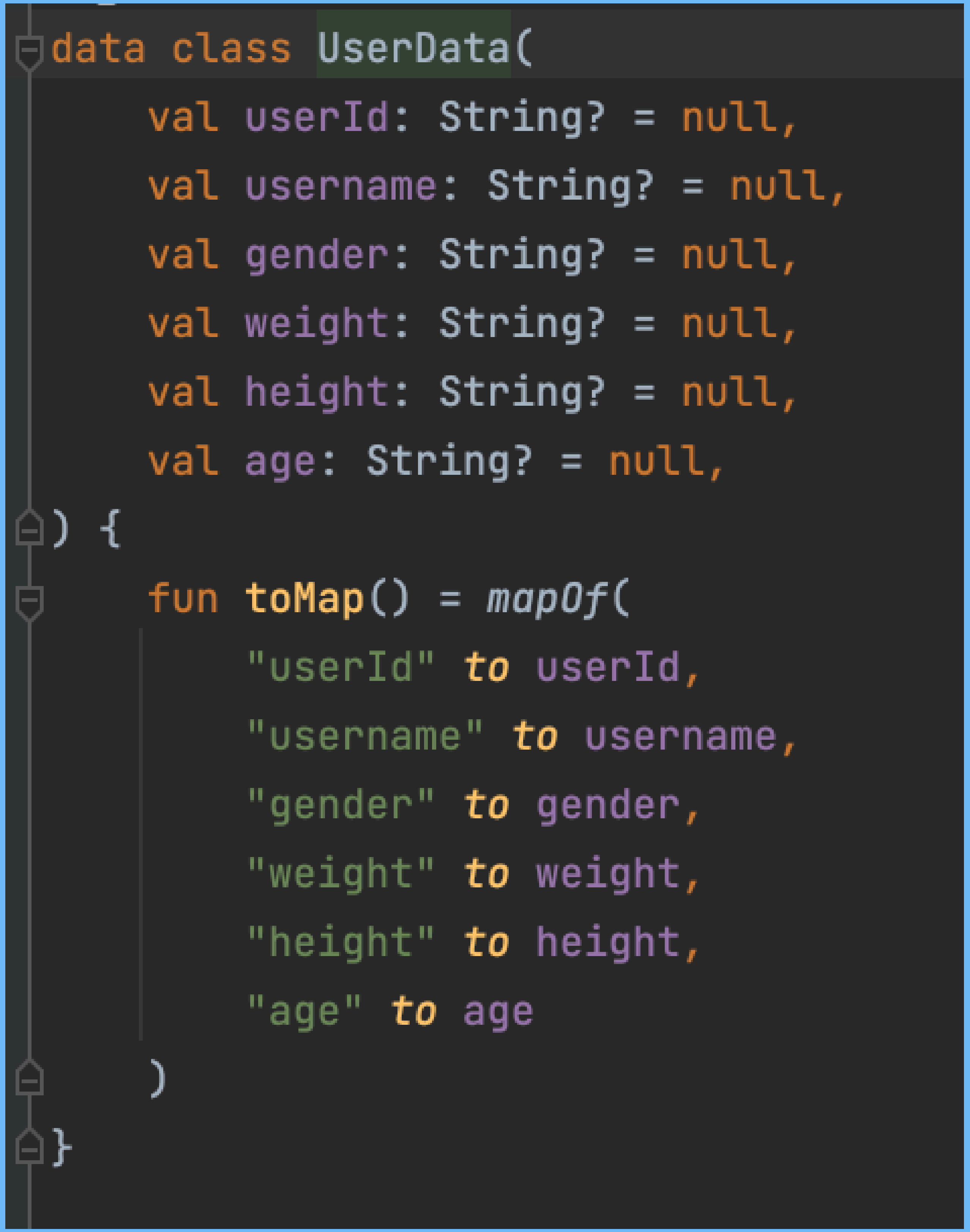
New here? Go to signup ->

Google Firebase
has been used in this project
to save userdata

Auth processes are located in ViewModel by Hilt Module

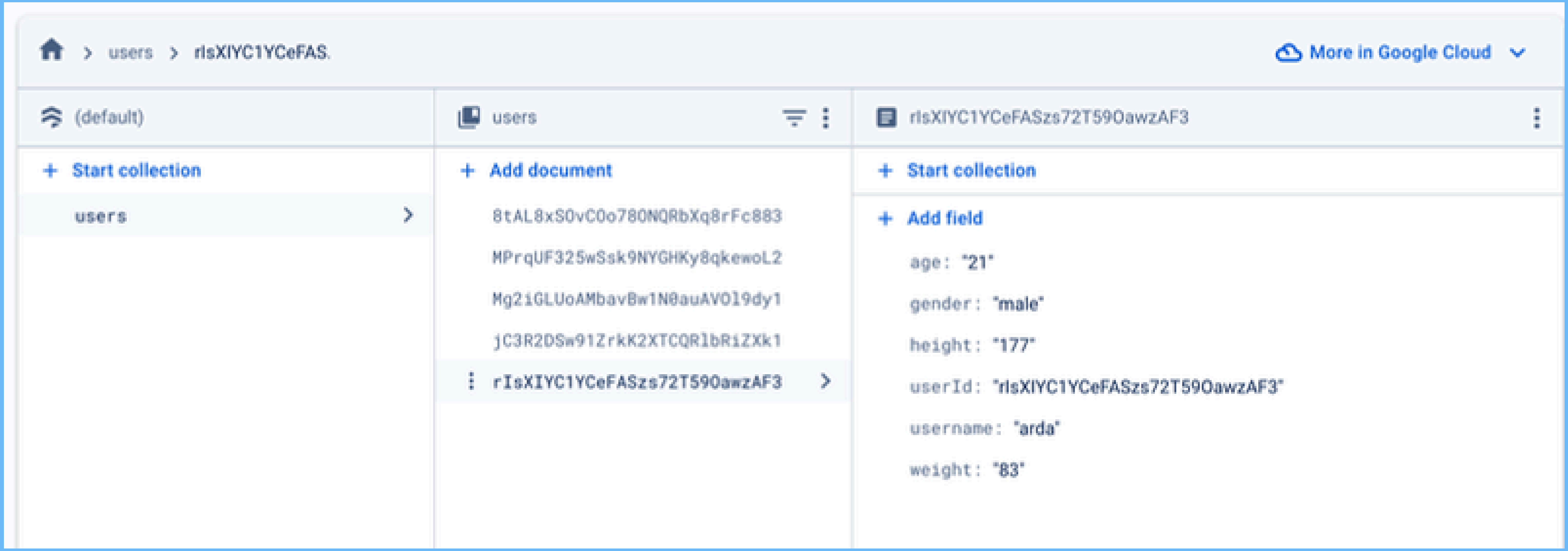


```
const val USERS = "users"
@HiltViewModel
class AppViewModel @Inject constructor(
    val auth: FirebaseAuth,
    val db: FirebaseFirestore
): ViewModel() {

    val signedIn = mutableStateOf( value: false)
    val inProgress = mutableStateOf( value: false)
    val userData = mutableStateOf<UserData?>( value: null)
    val popupNotification = mutableStateOf<Event<String>?>( value: null)

    init {
        //auth.signOut() // debug purpose
        val currentUser = auth.currentUser
        signedIn.value = currentUser != null
        currentUser?.uid?.let {uid ->
            getUserData(uid)
        }
    }

    fun onSignup(username: String, email: String, pass: String) {
        if (username.isEmpty() || email.isEmpty() || pass.isEmpty()) {
            handleException(customMessage = "Please fill in all fields")
            return
        }

        inProgress.value = true
        db.collection(USERS).whereEqualTo( field: "username", username).get()
            .addOnSuccessListener { documents ->
                if (documents.size() > 0) {
                    handleException(customMessage = "Username already exists")
                    inProgress.value = false
                } else {
                    auth.createUserWithEmailAndPassword(email, pass)
                        .addOnCompleteListener { task ->
                            if (task.isSuccessful) {
                                signedIn.value = true
                                // Create profile
                                createOrUpdateProfile(username = username)
                            } else {
                                Log.e( tag: "HATA", email)
                                handleException(task.exception, "Signup failed")
                            }
                            inProgress.value = false
```



```
data class UserData(
    val userId: String? = null,
    val username: String? = null,
    val gender: String? = null,
    val weight: String? = null,
    val height: String? = null,
    val age: String? = null,
) {
    fun toMap() = mapOf(
        "userId" to userId,
        "username" to username,
        "gender" to gender,
        "weight" to weight,
        "height" to height,
        "age" to age
    )
}
```

Firebase Firestore (Database):

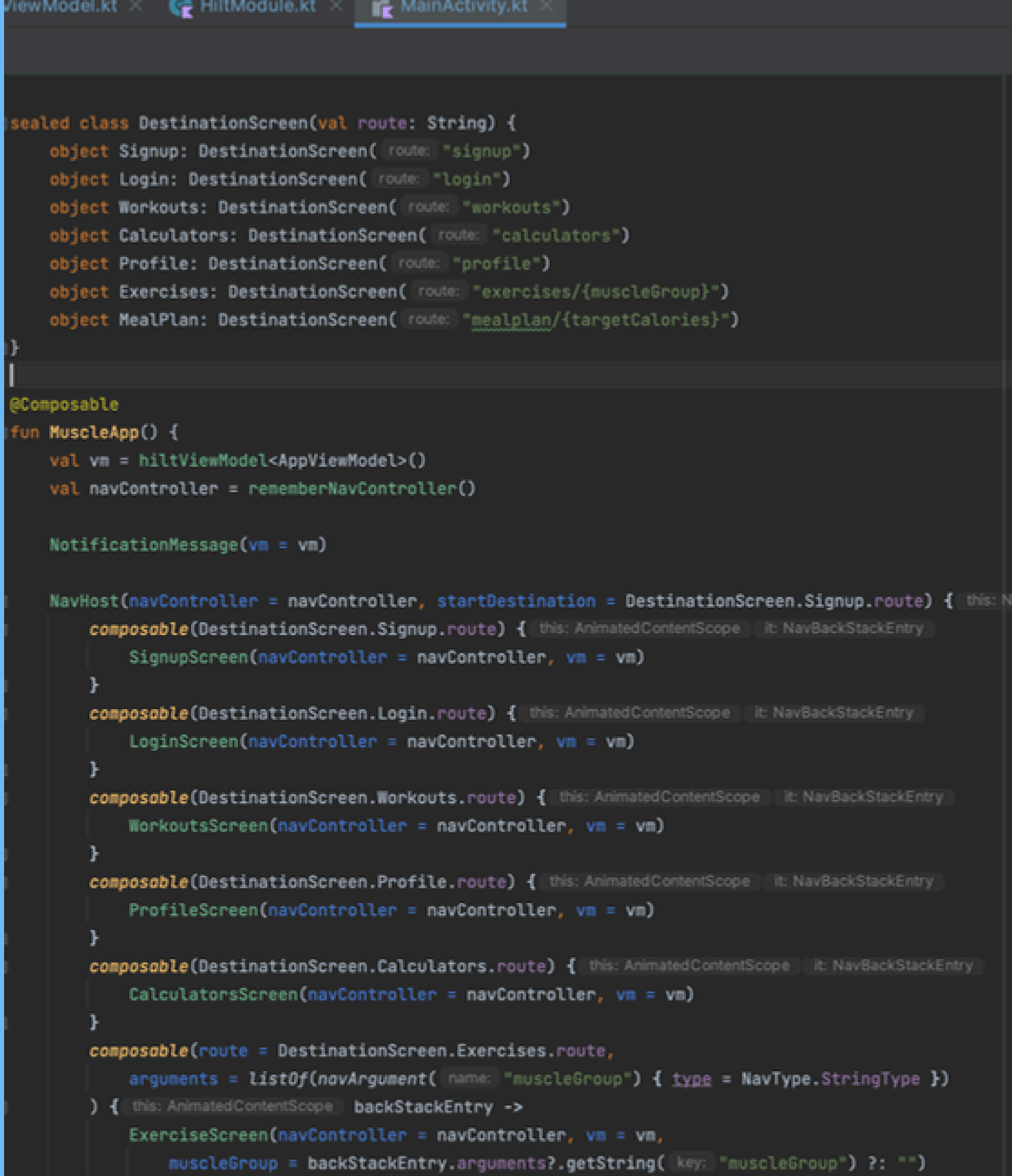


users

8tAL8xSOvCOo78ONQRbXq8rFc883
MPrqUF325wSsk9NYGHKy8qkewoL2
Mg2iGLUoAMbavBw1N0auAVOl9dy1
jC3R2DSw91ZrkK2XTCQRlbRiZXk1
rIsXIYC1YCeFASzs72T59OawzAF3

age: "21"
gender: "male"
height: "177"
userId: "rlsXIYC1YCeFASzs72T59OawzAF3"
username: "arda"
weight: "83"

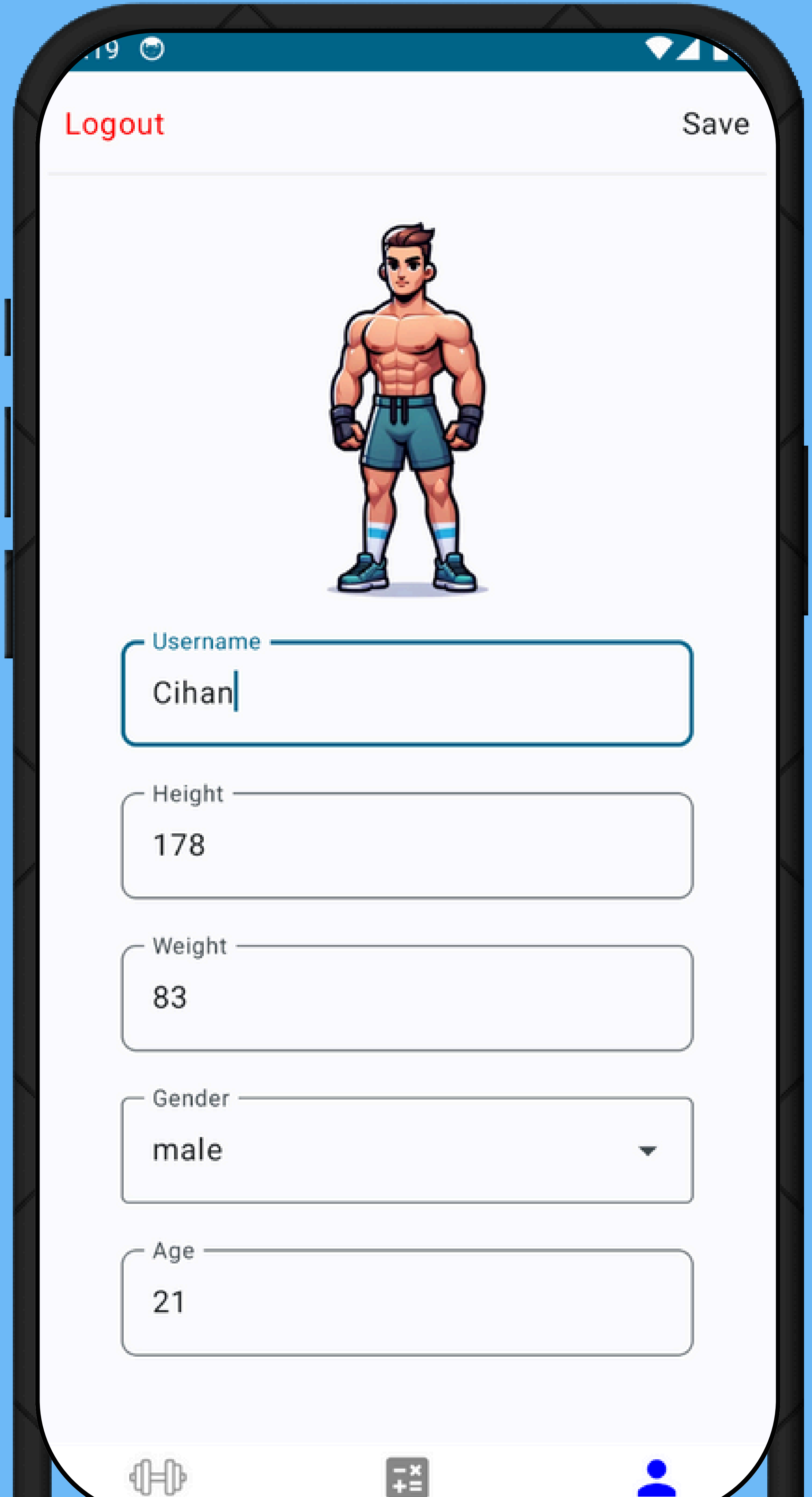
We used the NavHost component for screen transitions.



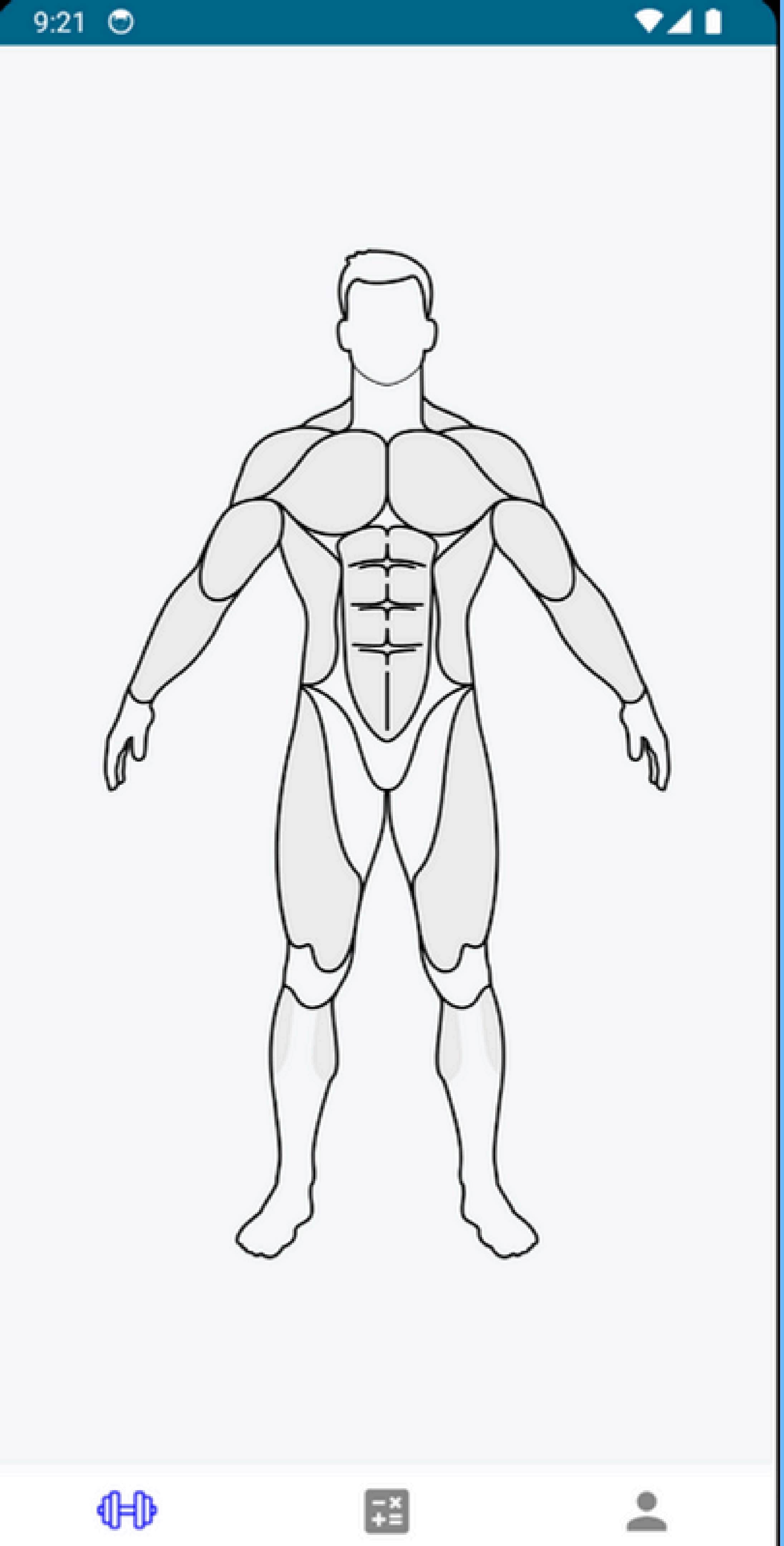
```kotlin
sealed class DestinationScreen(val route: String) {
    object Signup: DestinationScreen( route: "signup")
    object Login: DestinationScreen( route: "login")
    object Workouts: DestinationScreen( route: "workouts")
    object Calculators: DestinationScreen( route: "calculators")
    object Profile: DestinationScreen( route: "profile")
    object Exercises: DestinationScreen( route: "exercises/{muscleGroup}")
    object MealPlan: DestinationScreen( route: "mealplan/{targetCalories}")
}

@Composable
fun MuscleApp() {
    val vm = hiltViewModel<AppViewModel>()
    val navController = rememberNavController()

    NotificationMessage(vm = vm)

    NavHost(navController = navController, startDestination = DestinationScreen.Signup.route) { this: N
        composable(DestinationScreen.Signup.route) { this: AnimatedContentScope  it: NavBackStackEntry
            SignupScreen(navController = navController, vm = vm)
        }
        composable(DestinationScreen.Login.route) { this: AnimatedContentScope  it: NavBackStackEntry
            LoginScreen(navController = navController, vm = vm)
        }
        composable(DestinationScreen.Workouts.route) { this: AnimatedContentScope  it: NavBackStackEntry
            WorkoutsScreen(navController = navController, vm = vm)
        }
        composable(DestinationScreen.Profile.route) { this: AnimatedContentScope  it: NavBackStackEntry
            ProfileScreen(navController = navController, vm = vm)
        }
        composable(DestinationScreen.Calculators.route) { this: AnimatedContentScope  it: NavBackStackEntry
            CalculatorsScreen(navController = navController, vm = vm)
        }
        composable(route = DestinationScreen.Exercises.route,
            arguments = listOf(navArgument( name: "muscleGroup") { type = NavType.StringType })
        ) { this: AnimatedContentScope  backStackEntry ->
            ExerciseScreen(navController = navController, vm = vm,
                muscleGroup = backStackEntry.arguments?.getString( key: "muscleGroup") ?: "")
```

AppViewModel:

```
private fun createOrUpdateProfile(
    username: String? = null,
    gender: String? = null,
    weight: String? = null,
    height: String? = null,
    age: String? = null,
  ) {
    val uid = auth.currentUser?.uid
    val userData = UserData(
      userId = uid,
      username = username ?: userData.value?.username,
      gender = gender ?: userData.value?.gender,
      weight = weight ?: userData.value?.weight,
      height = height ?: userData.value?.height,
      age = age ?: userData.value?.age
    )

    uid?.let { uid ->
      inProgress.value = true
      db.collection(USERS).document(uid).get()
        .addOnSuccessListener {
          if (it.exists()) {
            it.reference.update(userData.toMap())
              .addOnSuccessListener {
                this.userData.value = userData
                inProgress.value = false
              }
              .addOnFailureListener {
                handleException(it, "Can not update user")
                inProgress.value = false
              }
          } else {
            db.collection(USERS).document(uid).set(userData)
            getUserData(uid)
          }
```

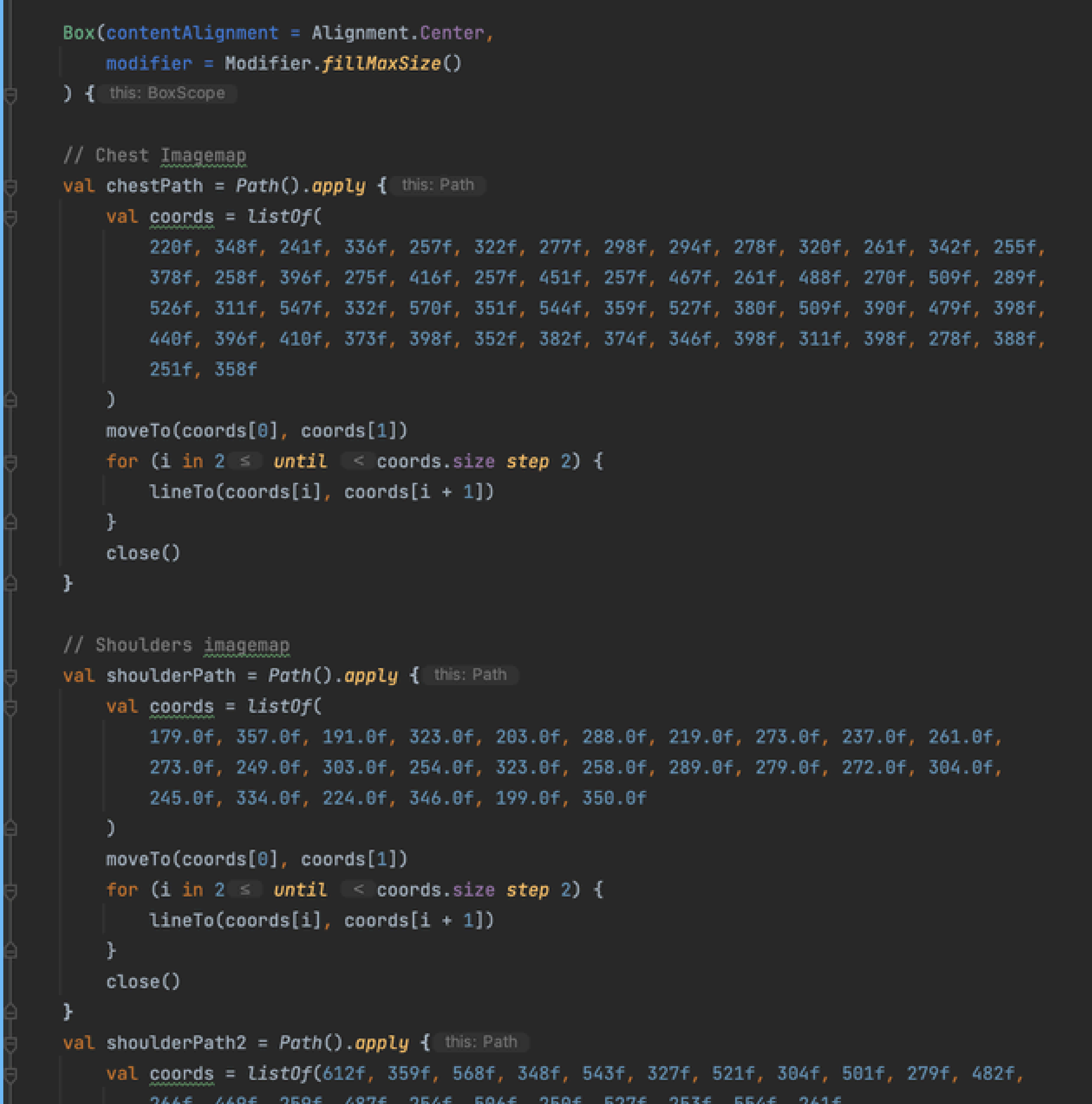

Logout
Save

Username: Cihan
Height: 178
Weight: 83
Gender: male
Age: 21



Select Target Muscle that you want to work!

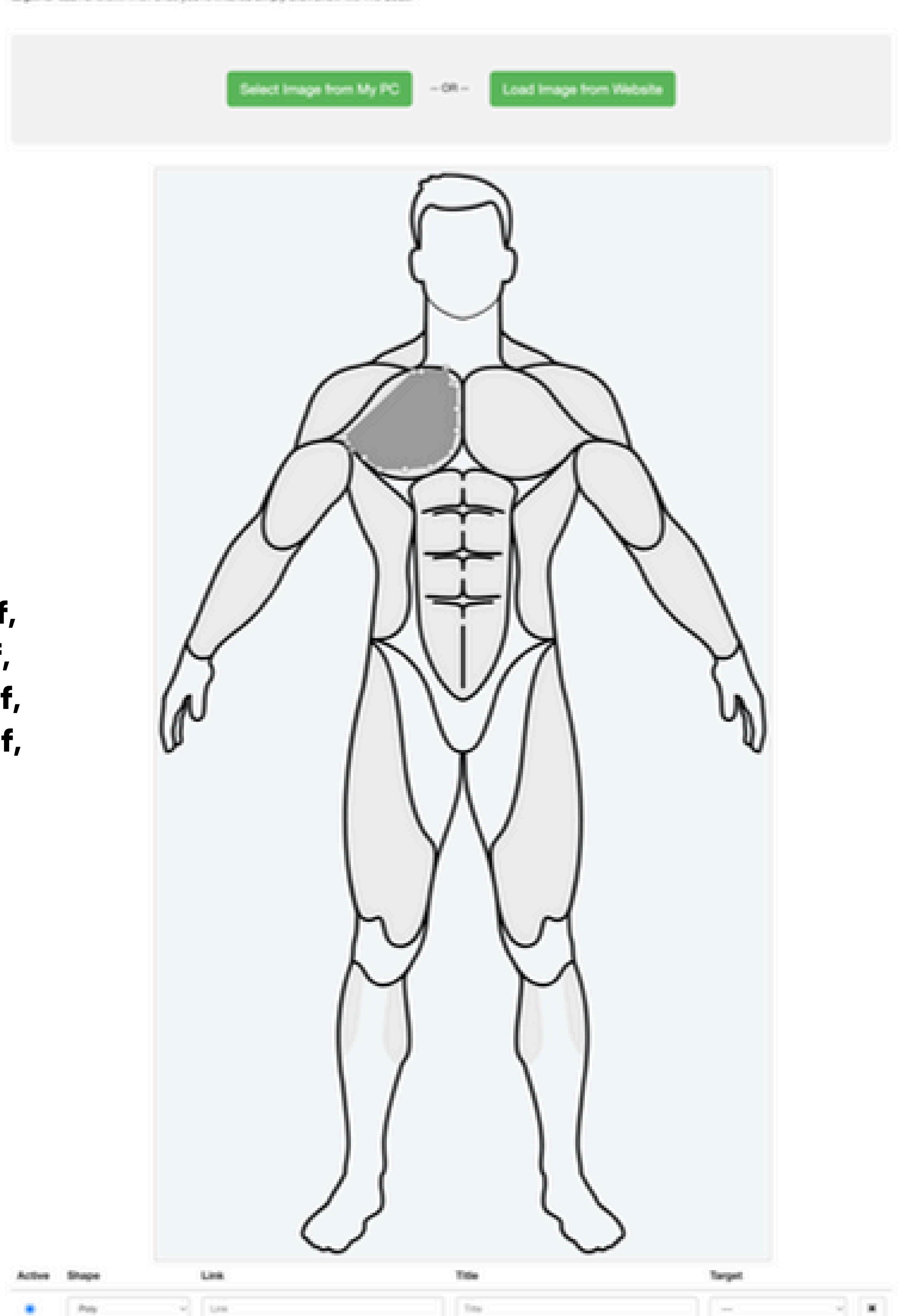
We used **Image Mapping** for the model hit-boxes.

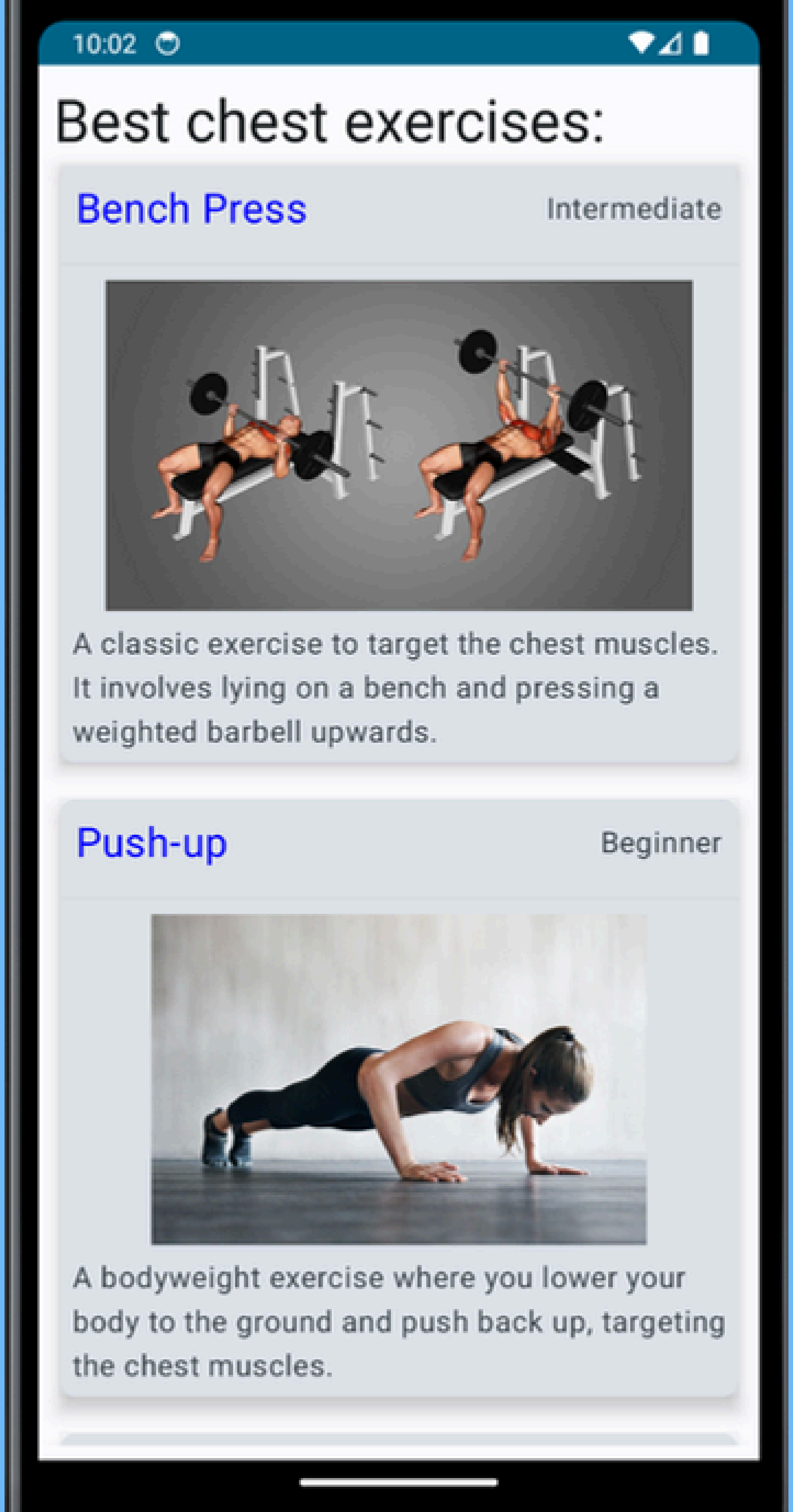


```kotlin
Box(contentAlignment = Alignment.Center,
    modifier = Modifier.fillMaxSize()
) {

// Chest Imagemap
val chestPath = Path().apply {
    val coords = listOf(
        220f, 348f, 241f, 336f, 257f, 322f, 277f, 298f, 294f, 278f, 320f, 261f, 342f, 255f,
        378f, 258f, 396f, 275f, 416f, 257f, 451f, 257f, 467f, 261f, 488f, 270f, 509f, 289f,
        526f, 311f, 547f, 332f, 570f, 351f, 544f, 359f, 527f, 380f, 509f, 390f, 479f, 398f,
        440f, 396f, 410f, 373f, 398f, 352f, 382f, 374f, 346f, 398f, 311f, 398f, 278f, 388f,
        251f, 358f
    )
    moveTo(coords[0], coords[1])
    for (i in 2 until coords.size step 2) {
        lineTo(coords[i], coords[i + 1])
    }
    close()
}

// Shoulders imagemap
val shoulderPath = Path().apply {
    val coords = listOf(
        179.0f, 357.0f, 191.0f, 323.0f, 203.0f, 288.0f, 219.0f, 273.0f, 237.0f, 261.0f,
        273.0f, 249.0f, 303.0f, 254.0f, 323.0f, 258.0f, 289.0f, 279.0f, 272.0f, 304.0f,
        245.0f, 334.0f, 224.0f, 346.0f, 199.0f, 350.0f
    )
    moveTo(coords[0], coords[1])
    for (i in 2 until coords.size step 2) {
        lineTo(coords[i], coords[i + 1])
    }
    close()
}
val shoulderPath2 = Path().apply {
    val coords = listOf(612f, 359f, 568f, 348f, 543f, 327f, 521f, 304f, 501f, 279f, 482f,
```

image-map.net has been used to get the coordinates of the selected area

```
val chestPath = Path().apply {
        val coords = listOf(
            220f, 348f, 241f, 336f, 257f, 322f, 277f, 298f, 294f, 278f, 320f, 261f, 342f, 255f,
            378f, 258f, 396f, 275f, 416f, 257f, 451f, 257f, 467f, 261f, 488f, 270f, 509f, 289f,
            526f, 311f, 547f, 332f, 570f, 351f, 544f, 359f, 527f, 380f, 509f, 390f, 479f, 398f,
            440f, 396f, 410f, 373f, 398f, 352f, 382f, 374f, 346f, 398f, 311f, 398f, 278f, 388f,
            251f, 358f
        )
        moveTo(coords[0], coords[1])
        for (i in 2 until coords.size step 2) {
            lineTo(coords[i], coords[i + 1])
        }
        close()
    }
```




10:02

Best chest exercises:

Bench Press Intermediate

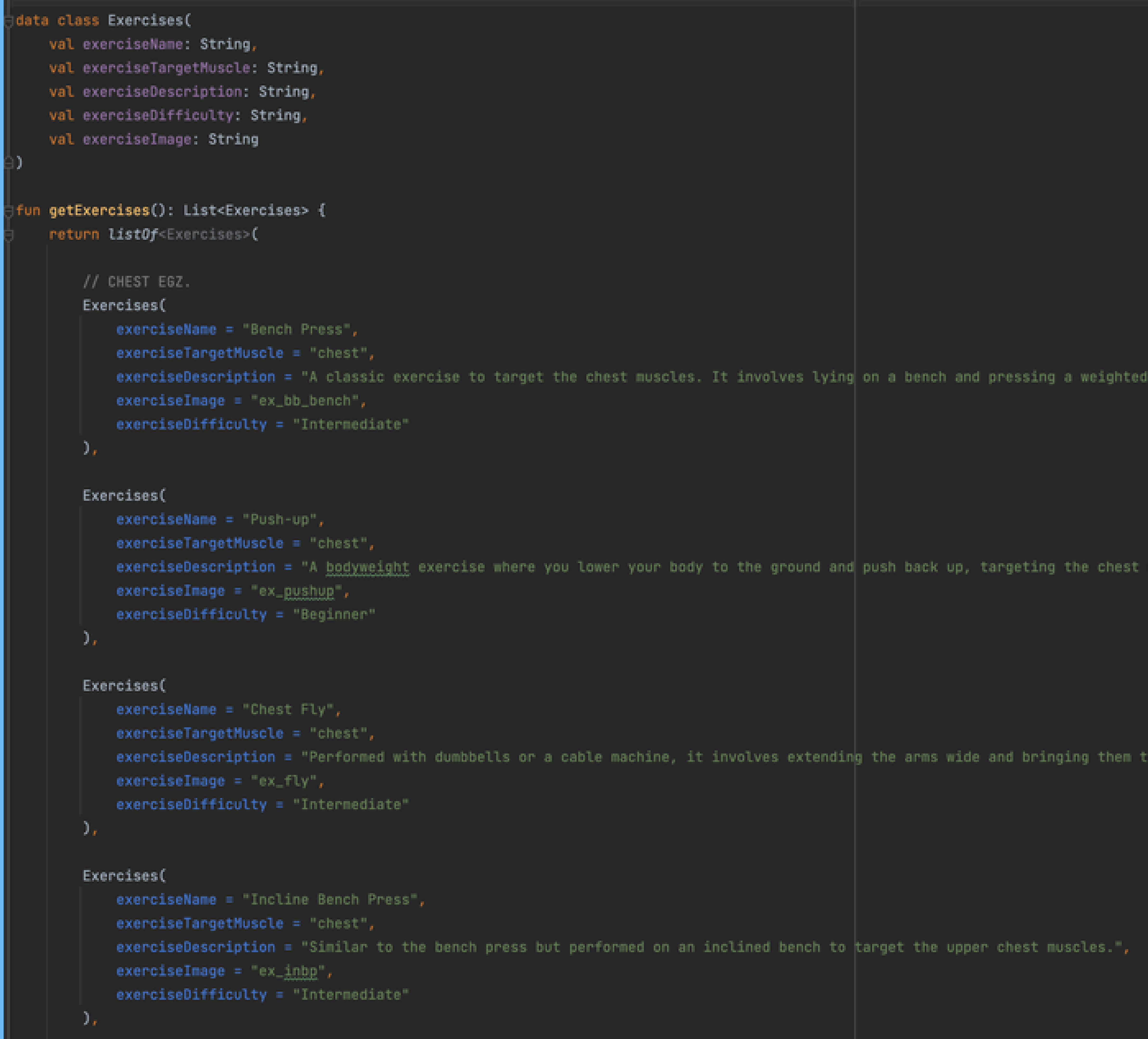
A classic exercise to target the chest muscles. It involves lying on a bench and pressing a weighted barbell upwards.

Push-up Beginner

A bodyweight exercise where you lower your body to the ground and push back up, targeting the chest muscles.

Here are the exercises you can do for the selected muscle!

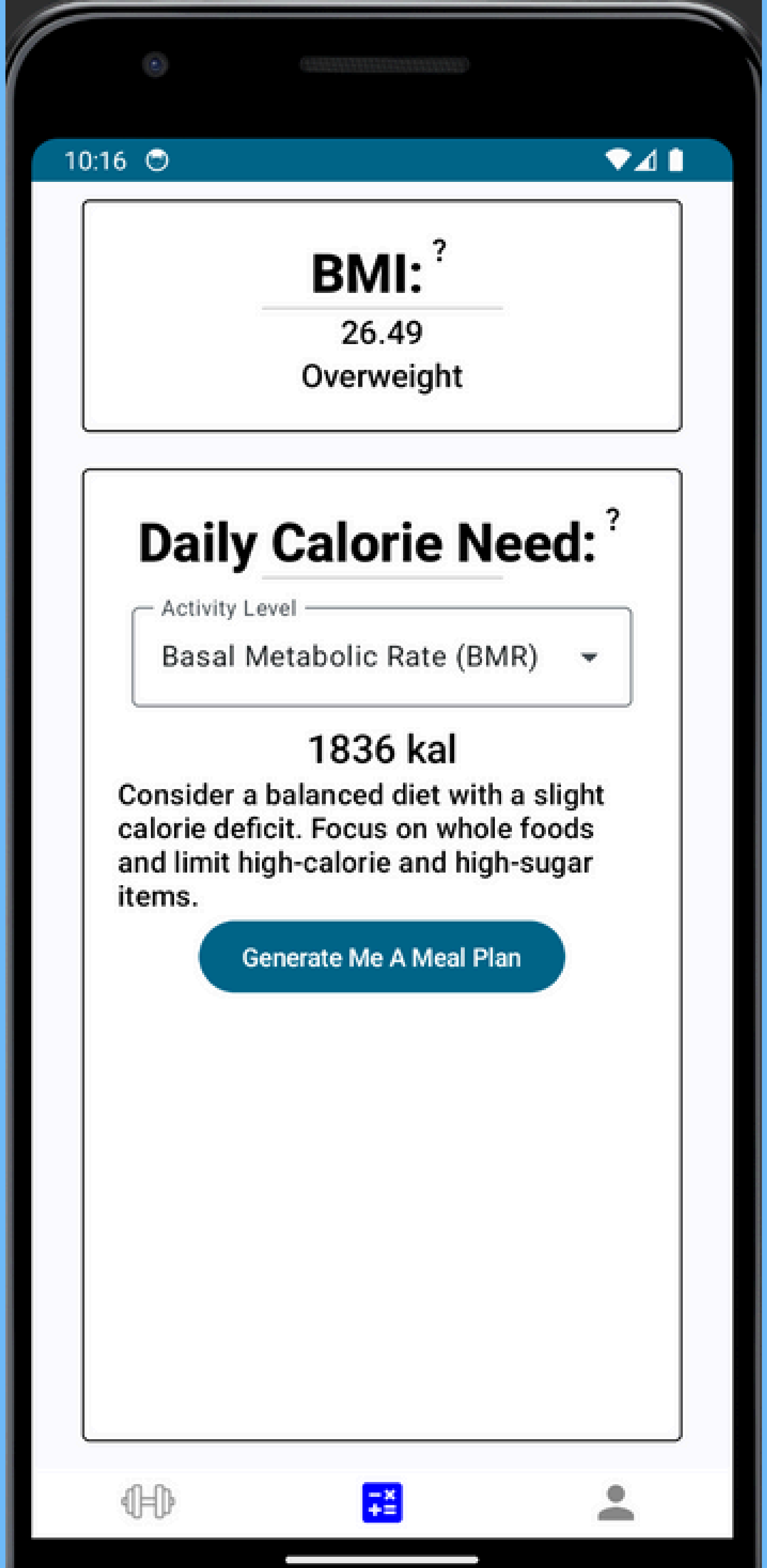
We store the exercise data in a **data class**



```kotlin
data class Exercises(
    val exerciseName: String,
    val exerciseTargetMuscle: String,
    val exerciseDescription: String,
    val exerciseDifficulty: String,
    val exerciseImage: String
)

fun getExercises(): List<Exercises> {
    return listOf<Exercises>(

        // CHEST EGZ.
        Exercises(
            exerciseName = "Bench Press",
            exerciseTargetMuscle = "chest",
            exerciseDescription = "A classic exercise to target the chest muscles. It involves lying on a bench and pressing a weighted
            exerciseImage = "ex_bb_bench",
            exerciseDifficulty = "Intermediate"
        ),

        Exercises(
            exerciseName = "Push-up",
            exerciseTargetMuscle = "chest",
            exerciseDescription = "A bodyweight exercise where you lower your body to the ground and push back up, targeting the chest
            exerciseImage = "ex_pushup",
            exerciseDifficulty = "Beginner"
        ),

        Exercises(
            exerciseName = "Chest Fly",
            exerciseTargetMuscle = "chest",
            exerciseDescription = "Performed with dumbbells or a cable machine, it involves extending the arms wide and bringing them t
            exerciseImage = "ex_fly",
            exerciseDifficulty = "Intermediate"
        ),

        Exercises(
            exerciseName = "Incline Bench Press",
            exerciseTargetMuscle = "chest",
            exerciseDescription = "Similar to the bench press but performed on an inclined bench to target the upper chest muscles.",
            exerciseImage = "ex_inbp",
            exerciseDifficulty = "Intermediate"
        ),
```

No need to use database, since:

- **Quick to Set Up and Easy:** Using data that is coded directly into the app is fast and easy, especially when starting or for small projects.
- **Independence:** It works without needing an external database server, so it doesn't need an internet connection and has fewer dependencies.
- **Reliability:** Keeping your data inside the app means it won't be affected by problems like internet errors or database server issues.



10:16

BMI: ?

26.49

Overweight

Daily Calorie Need: ?

Activity Level

Basal Metabolic Rate (BMR)

1836 kal

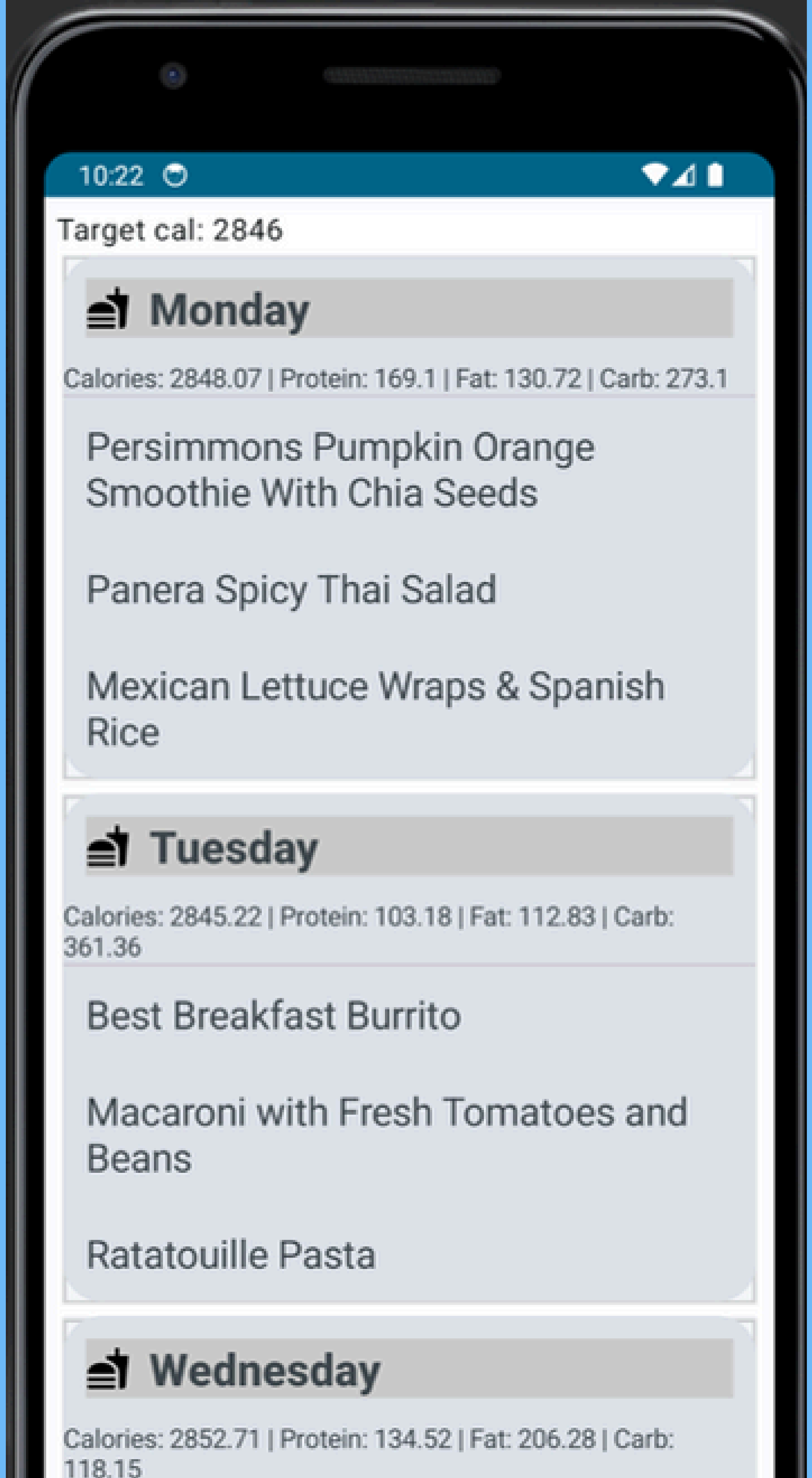
Consider a balanced diet with a slight calorie deficit. Focus on whole foods and limit high-calorie and high-sugar items.

Find Your **Body/Mass Index** and Calculate **Calorie Need**

```
val userData = vm.userData.value
        var gender by rememberSaveable { mutableStateOf(userData?.gender ?: "") }
        var weight by rememberSaveable { mutableStateOf(userData?.weight ?: "") }
        var height by rememberSaveable { mutableStateOf(userData?.height ?: "") }
        var age by rememberSaveable { mutableStateOf(userData?.age ?: "") }
```

A **diet plan** prepared exclusively for you.

We use **Spoonacular API** to get you best meal plan ever on earth!



Target cal: 2846

Monday

Calories: 2848.07 | Protein: 169.1 | Fat: 130.72 | Carb: 273.1

Persimmons Pumpkin Orange Smoothie With Chia Seeds

Panera Spicy Thai Salad

Mexican Lettuce Wraps & Spanish Rice

Tuesday

Calories: 2845.22 | Protein: 103.18 | Fat: 112.83 | Carb: 361.36

Best Breakfast Burrito

Macaroni with Fresh Tomatoes and Beans

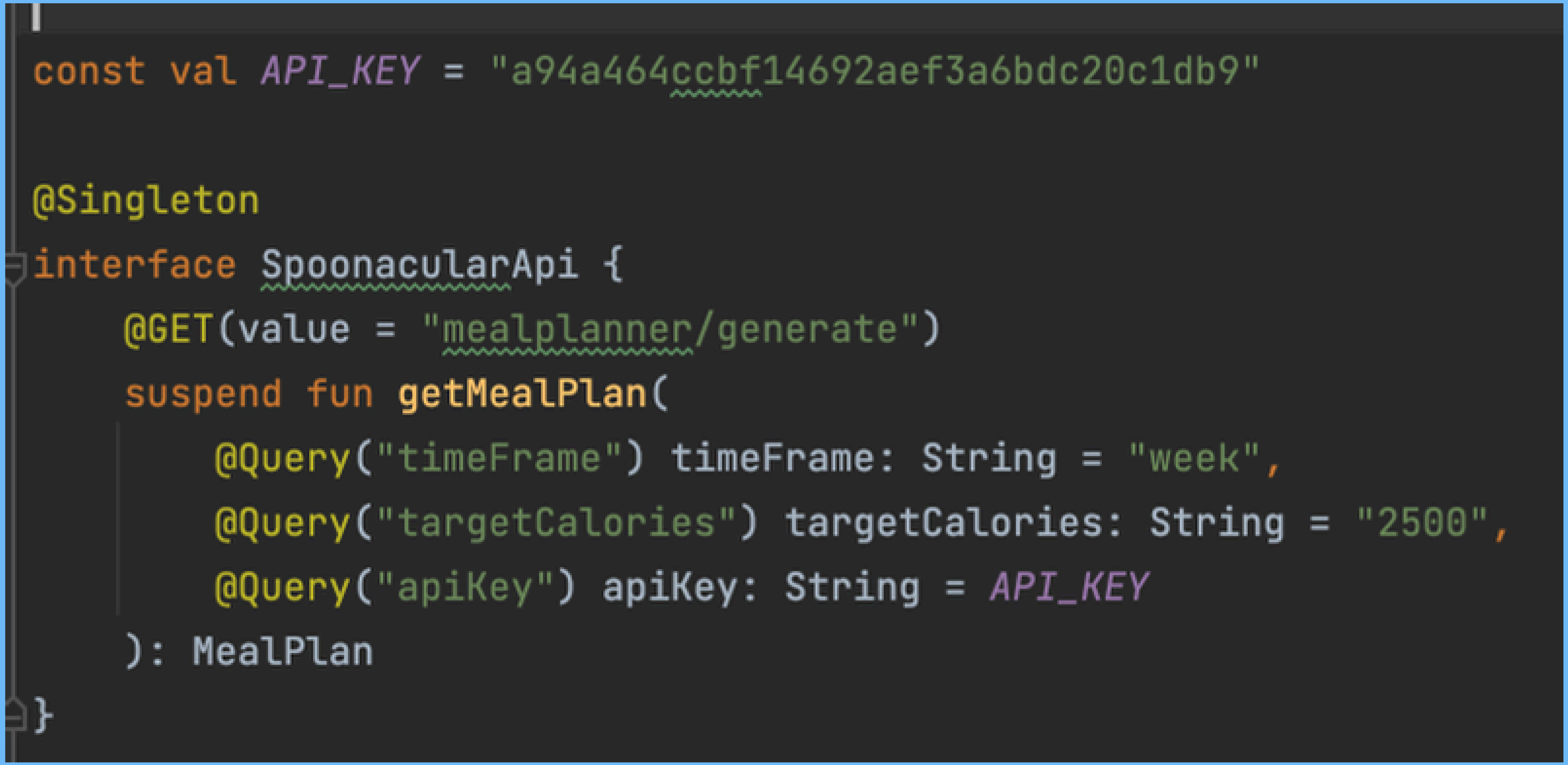
Ratatouille Pasta

Wednesday

Calories: 2852.71 | Protein: 134.52 | Fat: 206.28 | Carb: 118.15

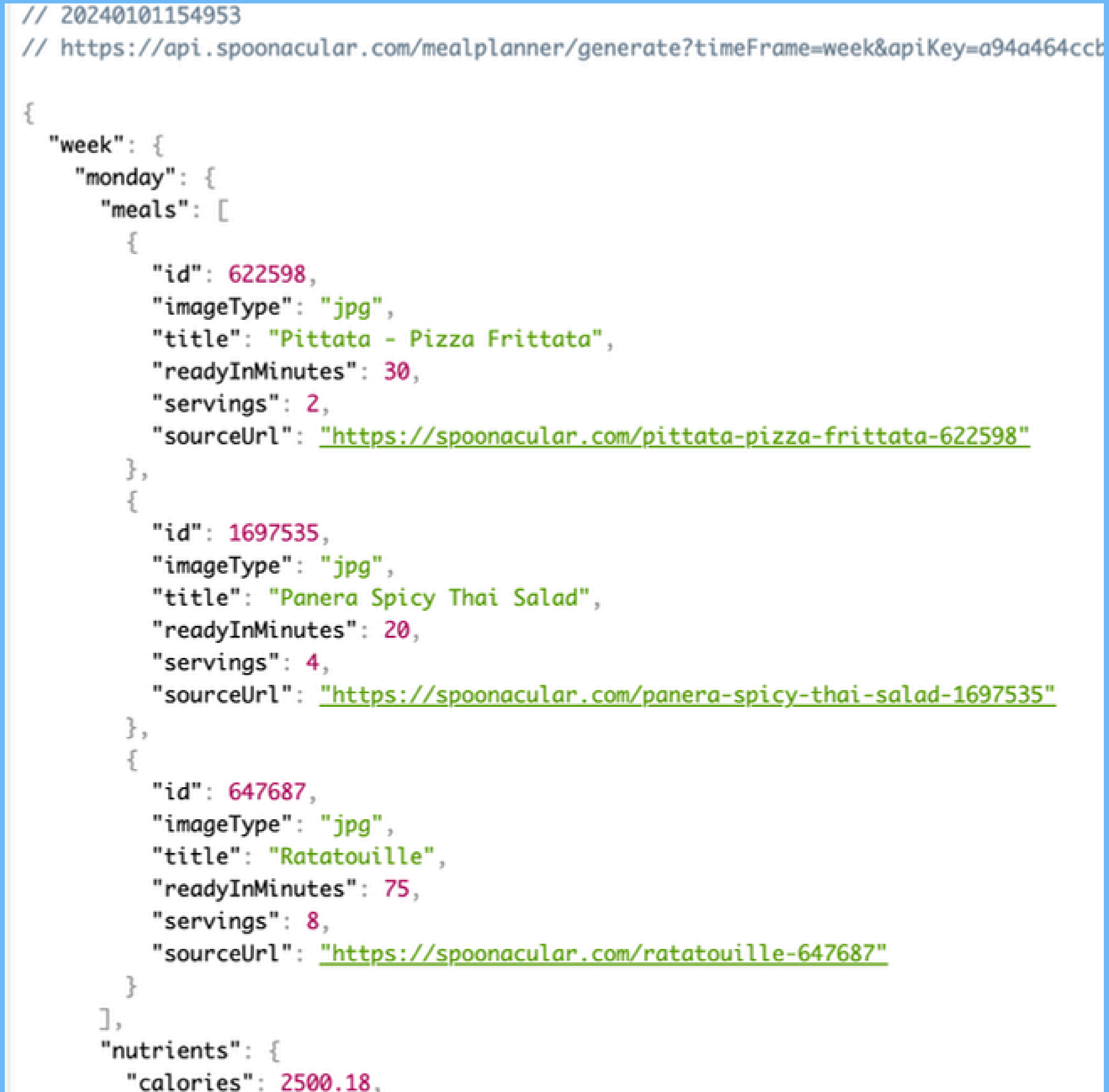
Query that we sent to the API:

https://api.spoonacular.com/mealplanner/generate?
timeFrame=week&apiKey=apiKey&targetCalories=**YourData**

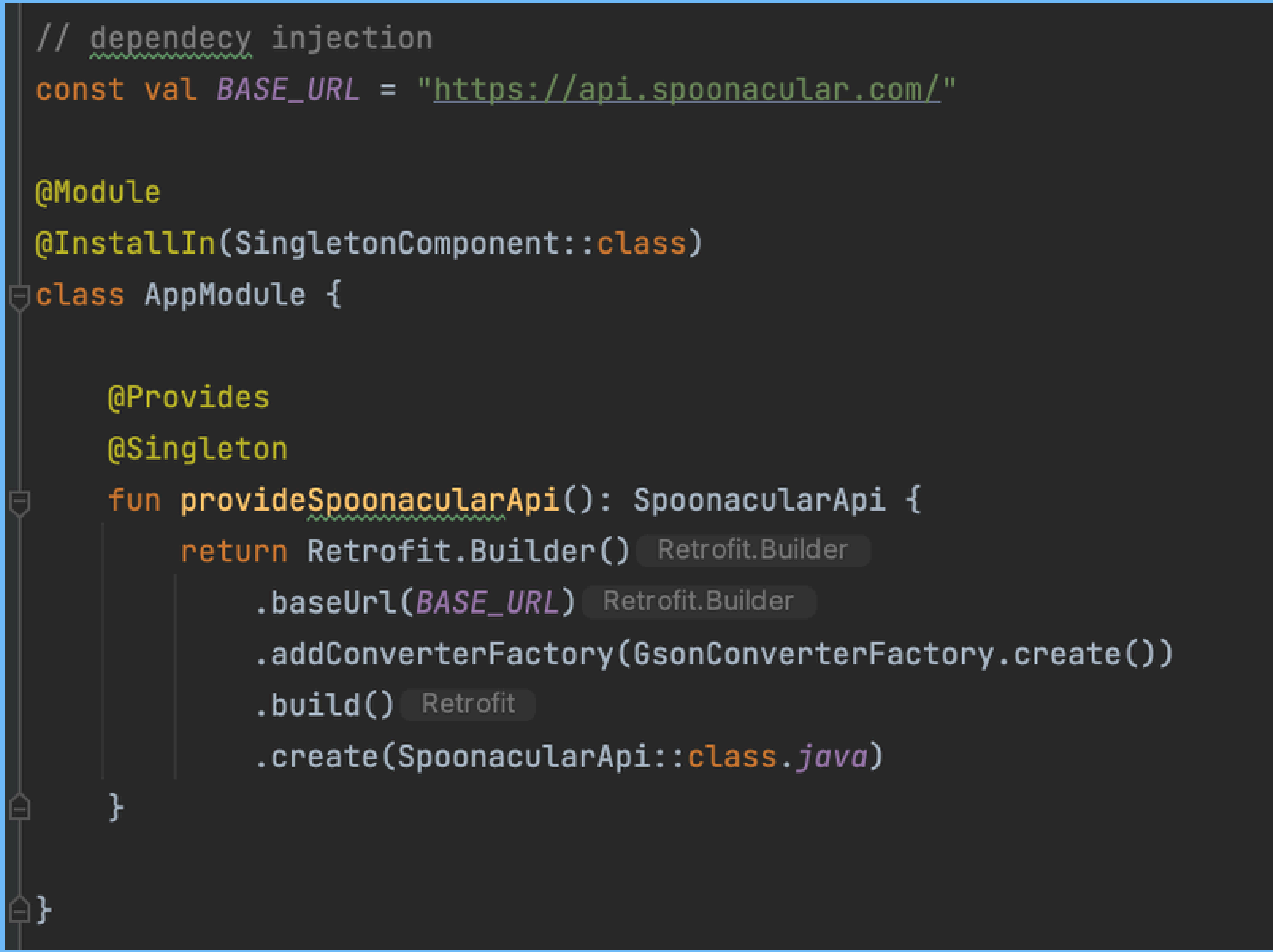


```kotlin
const val API_KEY = "a94a464ccbf14692aef3a6bdc20c1db9"

@Singleton
interface SpoonacularApi {
    @GET(value = "mealplanner/generate")
    suspend fun getMealPlan(
        @Query("timeFrame") timeFrame: String = "week",
        @Query("targetCalories") targetCalories: String = "2500",
        @Query("apiKey") apiKey: String = API_KEY
    ): MealPlan
}
```

We utilized Retrofit for network communication to efficiently handle data exchange



```
// 20240101154953
// https://api.spoonacular.com/mealplanner/generate?timeFrame=week&apiKey=a94a464ccb

{
  "week": {
    "monday": {
      "meals": [
        {
          "id": 622598,
          "imageType": "jpg",
          "title": "Pittata - Pizza Frittata",
          "readyInMinutes": 30,
          "servings": 2,
          "sourceUrl": "https://spoonacular.com/pittata-pizza-frittata-622598"
        },
        {
          "id": 1697535,
          "imageType": "jpg",
          "title": "Panera Spicy Thai Salad",
          "readyInMinutes": 20,
          "servings": 4,
          "sourceUrl": "https://spoonacular.com/panera-spicy-thai-salad-1697535"
        },
        {
          "id": 647687,
          "imageType": "jpg",
          "title": "Ratatouille",
          "readyInMinutes": 75,
          "servings": 8,
          "sourceUrl": "https://spoonacular.com/ratatouille-647687"
        }
      ],
      "nutrients": {
        "calories": 2500.18,
```



```kotlin
// dependecy injection
const val BASE_URL = "https://api.spoonacular.com/"

@Module
@InstallIn(SingletonComponent::class)
class AppModule {

    @Provides
    @Singleton
    fun provideSpoonacularApi(): SpoonacularApi {
        return Retrofit.Builder() Retrofit.Builder
            .baseUrl(BASE_URL) Retrofit.Builder
            .addConverterFactory(GsonConverterFactory.create())
            .build() Retrofit
            .create(SpoonacularApi::class.java)
    }

}
```

That's It!

Fitness Helper App

MuscleMaster