App

LifestyleApplication : Application

+ val applicationScope : CoroutineScope
+ val userDatabase : UserRoomDatabase
+ val userRepository : UserRepository
+ val weatherRepository : WeatherRepository

UserViewModel extends AndroidViewModel

+ data : LiveData<UserData> { get() }
- repository : UserRepository

+ UserViewModel(LifestyleApplication)
+ getDailyCalorieIntake() : Float
+ calculateBMR() : Float
+ calculateSedentaryCalNeed() : Float
+ calculateLightlyActiveCalNeed() : Float
+ calculateActiveCalNeed() : Float
+ calculateVeryActiveCalNeed() : Float
+ toString() : String
+ refreshLocation(Activity, (Location)->Unit)

UserRepository

+ data : MutableLiveData<UserData>

+ UserRepository(LifestyleApplication)
+ update()

companion object

@Volatile
- instance: UserRepository?

+ synchronized getInstance(Application) : UserRepository

- inner class FetchLocationTask

FetchLocationTask

+ executorService : ExecutorService
+ mainThreadHandler : HandlerCompat

+ execute()
- postToMainThread(Location)

@Entity (tableName = "user_table") UserTable

+ @field:PrimaryKey uuid: UUID
+ userJson: String

UserData

+ name: String?
+ age : Int
+ location: Location?
+ locationName: String?
+ textLocation : TextLocation
+ height : Float
+ weight : Float
+ sex : Sex
+ activityLevel : ActivityLevel
+ lastUsedModule : LastUsedModule
+ @Transient profilePictureThumbnail : Bitmap?
- uuid : UUID

@Database(entities=UserData, WeatherData)
abstract LifestyleRoomDatabase : RoomDatabase

+ abstract lifestyleDao() : LifestyleDao

companion object

- volatile instance : LifestyleRoomDatabase
- val databaseExecutor : ExecutorService
- roomDatabaseCallback : RoomDatabase.Callback

+ synchronized getInstance(Context, CoroutineScope) : LifestyleRoomDatabase

- class DatabaseCallback : RoomDatabase.Callback
- inner class PopulateDatabaseTask

@Dao LifestyleDao

+ @Insert(onConflict=UPDATE) insertWeather(WeatherData)
+ @Insert(onConflict=IGNORE) insertUser(User)
+ @Query deleteAllWeather()
+ @Query deleteUser(User)
+ @Query getAllUsers() : Flow<List<User>>
+ @Query getAllWeather() : Flow<List<WeatherData>>

DatabaseCallback

+ onCreate(SupportSQLiteDatabase)

PopulateDatabaseTask

+ PopulateDatabaseTask(UserDatabase)
+ executorService : ExecutorService
+ mainThreadHandler : HandlerCompat

+ execute()

WeatherViewModel extends AndroidViewModel

+ data : LiveData<WeatherData> { get() }
- repository : WeatherRepository

+ WeatherViewModel(LifestyleApplication)
+ setLocation(Location)

WeatherRepository

+ data : MutableLiveData<WeatherData>

+ WeatherRepository(LifestlyeApplication)
+ setLocation(Location)
+ update()
- loadData()

companion object

@Volatile
- instance: WeatherRepository?

+ synchronized getInstance(Application) : WeatherRepository

- inner class FetchWeatherTask

FetchWeatherTask

+ executorService : ExecutorService
+ mainThreadHandler : HandlerCompat

+ execute(Location)
- postToMainThread(String)

WeatherData

+ coord: Coord?
+ weather : List<Weather>?
+ main : Main?
+ visibility : Double?
+ wind : Wind?
+ rain : Rain?
+ snow : Snow?
+ dt : Double?
+ sys : Sys?
+ timezone : Double?

+ inner class Coord
+ inner class Weather
+ inner class Main
+ inner class Wind
+ inner class Rain
+ inner class Snow
+ inner class Sys

@Entity(tableName = "weather_table") WeatherTable

+ @field:PrimaryKey coord: Coord?
+ weatherJson: String

UserDaoTests
---
insertWeather(WeatherData)
deleteAllWeather()
getAllWeather()

WeatherDaoTests
---
insertUser(User)
deleteUser(User)
getAllUsers()

WeatherViewModelTests
---
setLocation(Location)

UserViewModelTests
---
getDailyCalorieIntake()
calculateBMR()
calculateSedentaryCalNeed()
calculateLightlyActiveCalNeed()
calculateActiveCalNeed()
calculateVeryActiveCalNeed()
refreshLocation(Activity, (Location)->Unit)

WeatherRepositoryTests
---
setLocation(location)
update()
loadData()

fetchWeatherData()

getInstance()

UserRepositoryTests
---
update()
loadData()

fetchUserData()

getInstance()

Other testing

User Testing: Test the app to ensure that it is functional from a user perspective and that things gets updated properly. Test
- Profile module
- BMRCalculator module
- Weather module
- Maps/Hiking module

Lifecycle Testing: Ensure that screen rotation, exiting the app and returning to it, etc. doesn't change or affect the data in the app in any way. Ensure that data is being saved properly as well (especially with the database changes).

Unit Testing: Refactor, fix, and debug any previous unit tests from Project 1.
