Assignment #7

App Title: Gardening Journal App

Design and implement a Gardening Journal App using Android Kotlin that incorporates ViewModel, LiveData, Navigation component, NavHostFragment, Room database, and coroutine concepts.

Navigation Structure:

- Create a gardening app with at least three main screens: Home, Garden Log, and Plant Details.
- Use the Navigation component to implement navigation between these screens.
- Utilize a NavHostFragment for hosting the navigation graph.

ViewModel and LiveData:

- Implement a ViewModel for each screen to manage the UI-related data.
- Use LiveData to observe and update the UI based on the underlying data changes.
- Ensure that the ViewModel survives configuration changes.

Room Database:

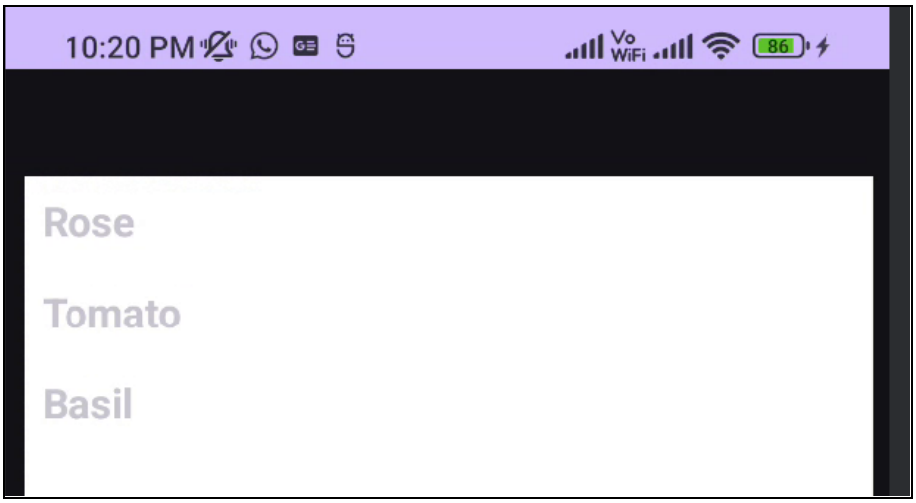
- Create an entity class representing a plant with attributes like name, type, watering frequency, and planting date.
- Implement a Room database to store and retrieve plant data.
- Use a DAO (Data Access Object) to perform database operations.

Coroutines:

- Use coroutines to perform asynchronous operations, such as database queries.
- Ensure that database operations run on a background thread to avoid blocking the main thread.

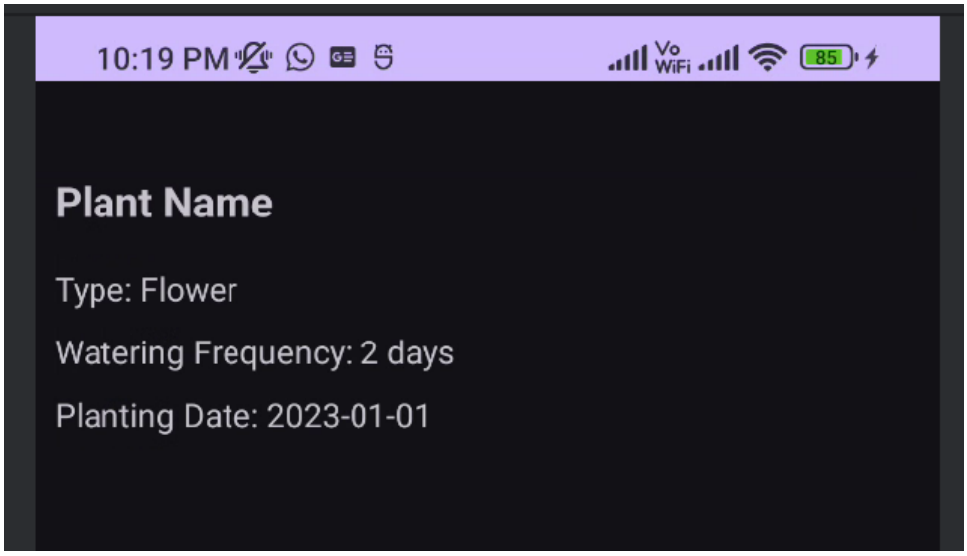
Garden Log Screen:

- Display a list of plants from the Room database in a RecyclerView on the Garden Log screen.
- Allow users to add new plants to the database by entering the plant details.
- Implement a coroutine to insert plant data into the database asynchronously.



Plant Details Screen:

- Create a Plant Details screen that displays detailed information about a selected plant.
- Use navigation arguments to pass the selected plant's ID to the Plant Details screen.
- Retrieve the plant details from the Room database using the ID.



Documentation:

- Provide a brief README file explaining the purpose of the app, how to build and run it, and any additional information about the implementation.

Share the codebase using a version control system (e.g., GitHub).
Include necessary instructions for setting up and running the project.

Refer to the next page for reference.

Populate the GardenLogFragment with sample data

```
val samplePlants = mutableListOf<Plant>()

// Add sample plants
samplePlants.add(Plant(name = "Rose", type = "Flower", wateringFrequency = 2, plantingDate = "2023-01-01"))
samplePlants.add(Plant(name = "Tomato", type = "Vegetable", wateringFrequency = 3, plantingDate = "2023-02-15"))
samplePlants.add(Plant(name = "Basil", type = "Herb", wateringFrequency = 1, plantingDate = "2023-03-10"))

    // Insert sample plants into the database
    for (plant in samplePlants) {
        viewModel.insert(plant)
    }
```

Sample code for GardenLogViewModel

```
class GardenLogViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: PlantRepository

    val allPlants: LiveData<List<Plant>>

    init {
        val plantDao = PlantDatabase.getDatabase(application).plantDao()
        repository = PlantRepository(application)
        allPlants = repository.allPlants
    }

    fun insert(plant: Plant) = viewModelScope.launch { this: CoroutineScope
        repository.insert(plant)
    }

    fun update(plant: Plant) = viewModelScope.launch { this: CoroutineScope
```

PlantDao.kt

```
@Dao
interface PlantDao {
    @Query("SELECT * FROM plants")
    fun getAllPlants(): LiveData<List<Plant>>

    @Insert
```

```
    suspend fun insert(plant: Plant)

    @Update
    suspend fun update(plant: Plant)

    @Query("DELETE FROM plants")
    suspend fun deleteAll()

    @Query("DELETE FROM plants WHERE id = :plantId")
    suspend fun delete(plantId: Int)

    @Query("SELECT * FROM plants WHERE id = :plantId")
    fun getPlantById(plantId: Int): LiveData<Plant>

   //…

}
```

```
@Entity(tableName = "plants")
data class Plant(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val name: String,
    val type: String,
    val wateringFrequency: Int,
    val plantingDate: String
)
```

```
@Database(entities = [Plant::class], version = 1, exportSchema = false)
abstract class PlantDatabase : RoomDatabase() {
    abstract fun plantDao(): PlantDao

    companion object {
        @Volatile
        private var INSTANCE: PlantDatabase? = null

        fun getDatabase(context: Context): PlantDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    PlantDatabase::class.java,
                    "plant_database"
```

```
            ).build()
            INSTANCE = instance
            instance
        }
    }
  }
}
```

```
class PlantRepository(application: Application) {
    private val plantDao: PlantDao
    val allPlants: LiveData<List<Plant>>

    init {
        val database = PlantDatabase.getDatabase(application)
        plantDao = database.plantDao()
        allPlants = plantDao.getAllPlants()
    }

    suspend fun insert(plant: Plant) {
        plantDao.insert(plant)
    }

    suspend fun update(plant: Plant) {
        plantDao.update(plant)
    }

    suspend fun delete(plant: Plant) {
        plantDao.delete(plant.id)
    }

    fun getPlantById(plantId: Int): LiveData<Plant> {
        return plantDao.getPlantById(plantId)
    }
}
```

```
class PlantDetailsFragment : Fragment() {

    private lateinit var viewModel: GardenLogViewModel
    private var plantId: Int = 0

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
```

```
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_plant_details, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Get the plantId from the arguments
        plantId = arguments?.getInt("plantId") ?: 0

        viewModel = ViewModelProvider(this).get(GardenLogViewModel::class.java)

        // Observe the plant details and update UI
        viewModel.getPlantById(plantId).observe(viewLifecycleOwner, Observer { plant ->
            plant?.let { displayPlantDetails(it) }
        })
    }

    private fun displayPlantDetails(plant: Plant) {
        // Update UI with plant details
        view?.findViewById<TextView>(R.id.plantNameTextView)?.text = plant.name
        view?.findViewById<TextView>(R.id.plantTypeTextView)?.text = "Type: ${plant.type}"
        view?.findViewById<TextView>(R.id.wateringFrequencyTextView)?.text = "Watering
Frequency: ${plant.wateringFrequency} days"
        view?.findViewById<TextView>(R.id.plantingDateTextView)?.text = "Planting Date:
${plant.plantingDate}"
    }
}
```