Mobile App Development - Cooking Recipe App

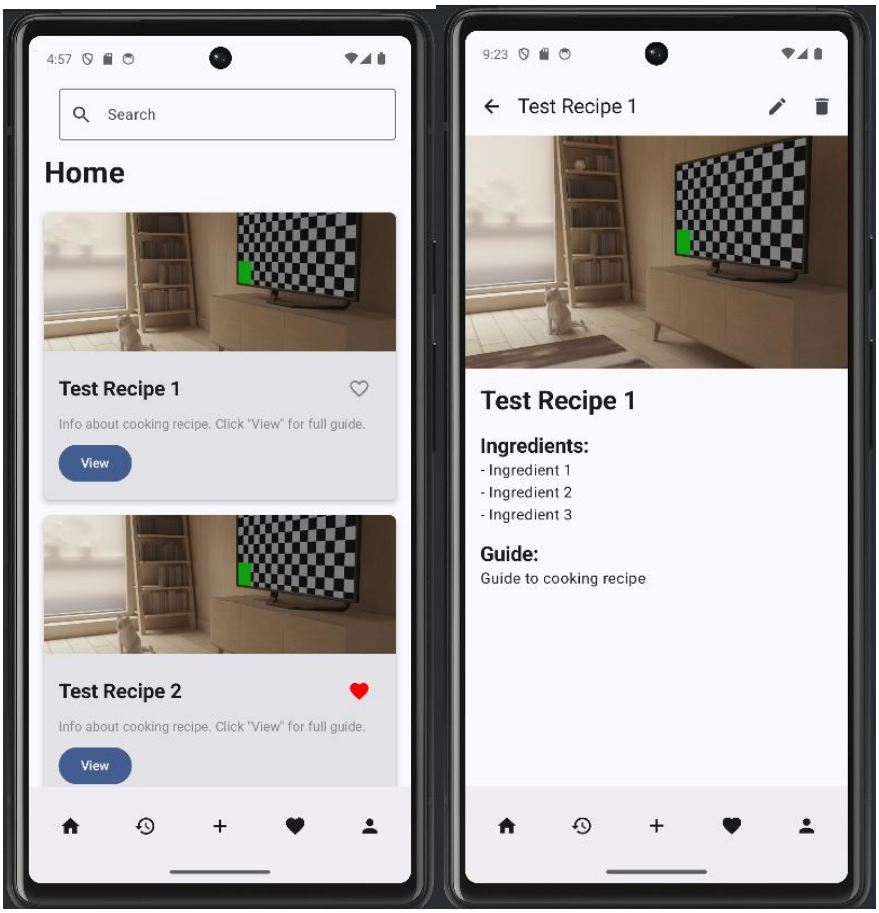
Christopher Lavin – S00251319

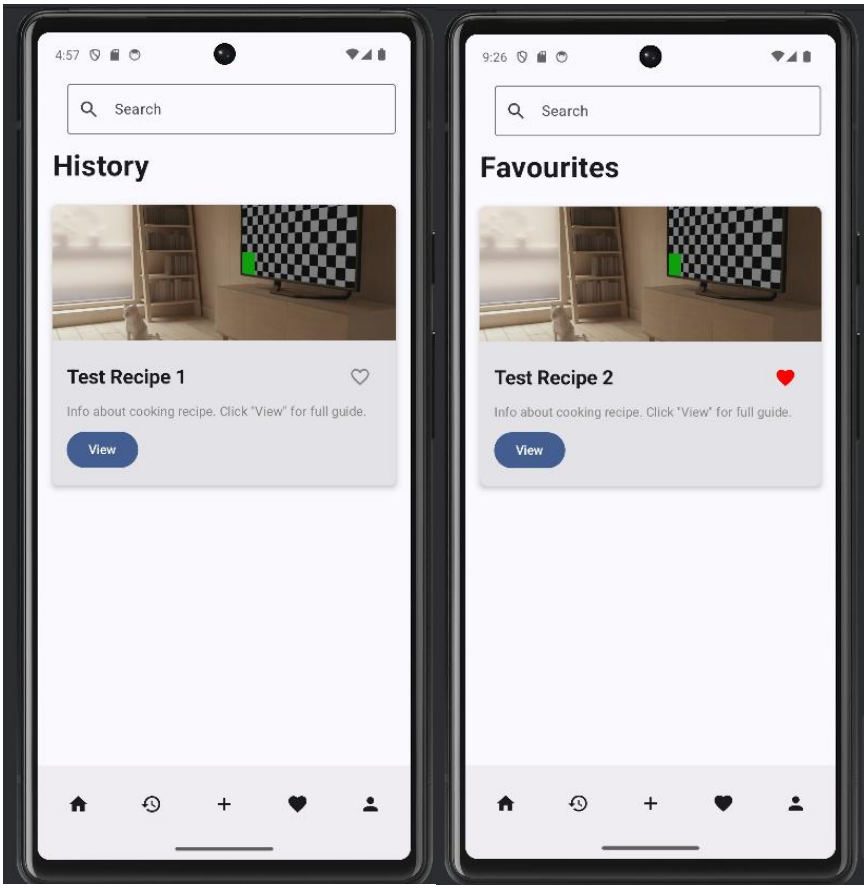
GitHub Repository: https://github.com/ChrisLavin04/CookingRecipeApp

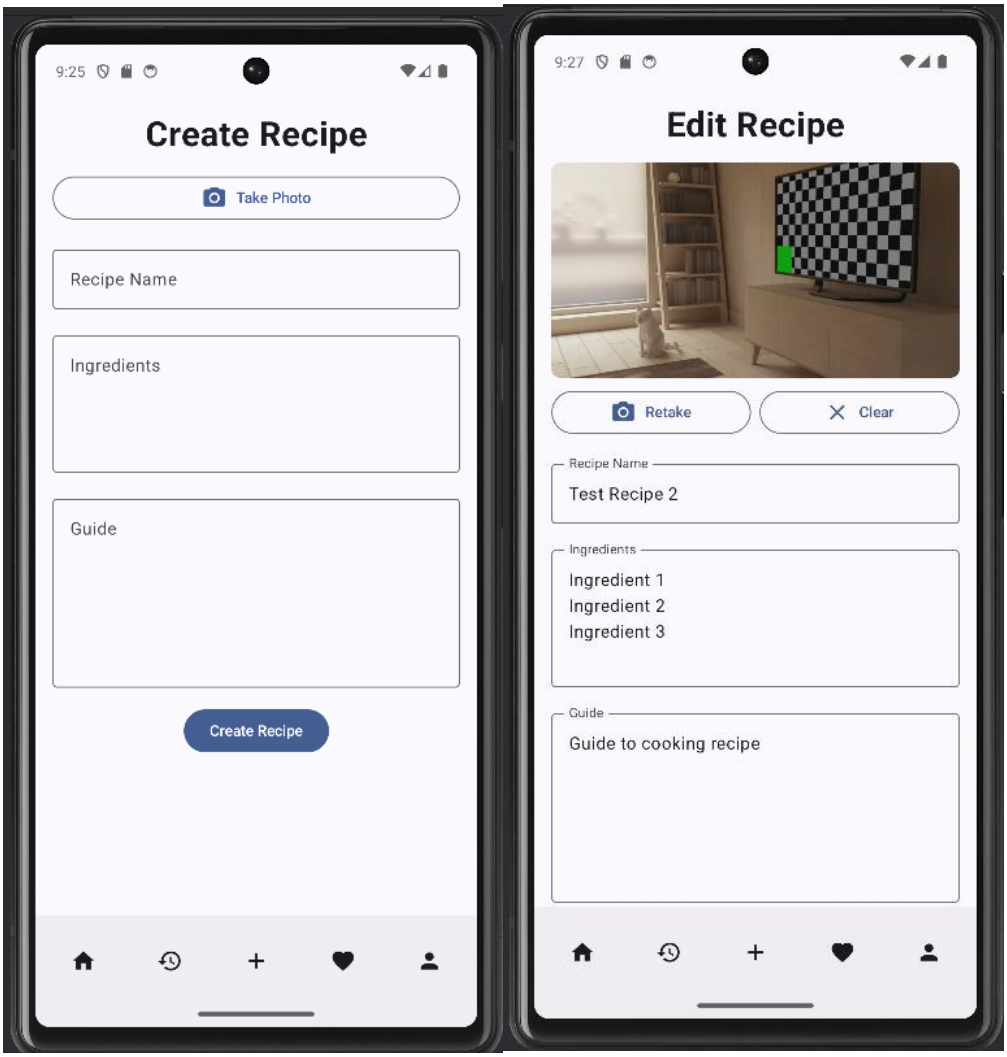
App Description

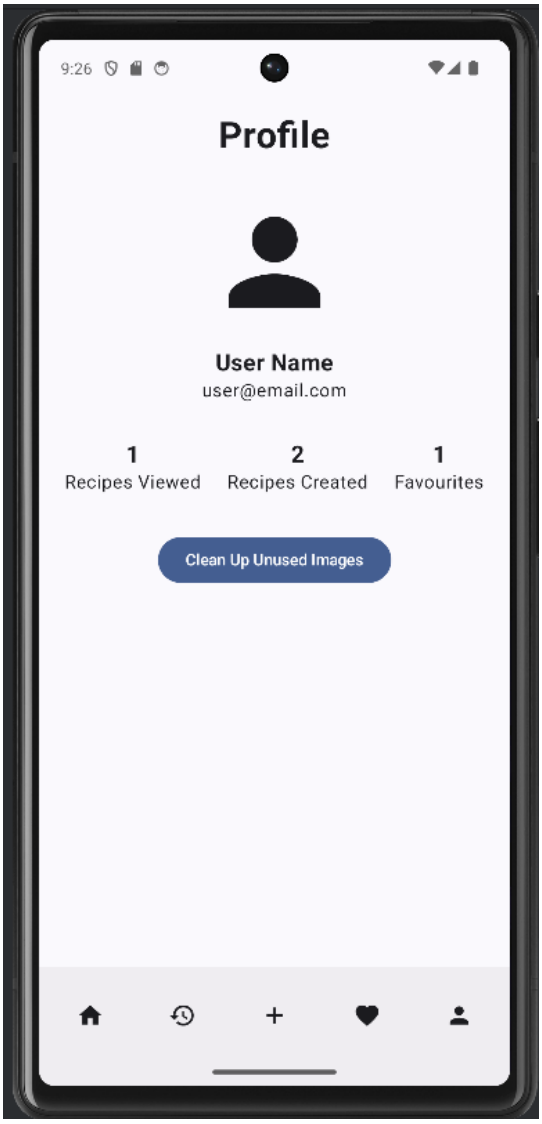
A cooking recipe management application built with Jetpack Compose that allows users to create, update, and organize cooking recipes. Users can capture photos using the device camera, mark favourites, track recently viewed recipes, and search for recipes in their collection.

Preview




4:57
Search
History
Test Recipe 1
Info about cooking recipe. Click "View" for full guide.
View
9:26
Search
Favourites
Test Recipe 2
Info about cooking recipe. Click "View" for full guide.
View


9:25
Create Recipe
Take Photo
Recipe Name
Ingredients
Guide
Create Recipe
9:27
Edit Recipe
Retake
Clear
Recipe Name
Test Recipe 2
Ingredients
Ingredient 1
Ingredient 2
Ingredient 3
Guide
Guide to cooking recipe


9:26
Profile
User Name
user@email.com
1
Recipes Viewed
2
Recipes Created
1
Favourites
Clean Up Unused Images

Core Functionality

1. **Complete CRUD Operations**

- Create new recipes with custom photos.

- Read/view recipe details with ingredients and a guide.

- Update existing recipe information.

- Delete recipes permanently.

2. **Data Persistence**

- Created recipes are listed in the app.

- Recipes stay saved even after relaunching app.

3. **Data Filters**

- History Screen only shows viewed recipes.

- Favourites Screen only shows favourited recipes.

- Search bar that filters only recipes that fit the search query.

- All filters made through SQL.

4. Validation

- All fields except for photo require an input.

Screens and Navigation

1. Home Screen (Home Icon)
- Displays all recipes in the database.

2. Recipe Details Screen (Clicking "View" on a recipe)

- Displays the picture, title, ingredients and guide for a selected recipe.

3. Create Recipe Screen (Plus icon)

- Displays an image capture button and empty fields to enter recipe details.

4. Edit Recipe Screen (Pencil Icon in Recipe Details Screen)

- Displays an image retake button and filled in fields to edit details of an existing recipe.

5. History Screen (Clock and round arrow Icon)

- Displays exclusively recipes that have been viewed by the user before.

6. Favourites Screen (Heart Icon)

- Displays exclusively recipes that have been marked as favourite.

7. Profile Screen (User icon)
- Displays the user's stats; number of created, viewed and favourited recipes

Hardware Integration - Camera

I added camera hardware integration using CameraX to allow users to use a real device camera to take a photo to display on their recipes. This is done simply by clicking the "Take photo" button when creating or editing recipes, then using their device camera to take a photo. The App also asks for permission to use the device camera before initiating the capture sequence.

Testing

The app was tested on an emulated Pixel 6 and a real Samsung galaxy A35 5G

The app looked and performed identically.

I also included Unit Tests in my project to test CRUD operations, queries, filters and data persistence.

Code Highlights

Recipe model used for objects in the database

```
@Entity(tableName = "recipes")
data class Recipe(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    @ColumnInfo(name = "ingredients")
    val ingredients: String, // Comma-separated for Room
    val guide: String,
    @DrawableRes val image: Int, // Drawable resource ID
    val imagePath: String? = null, // File path for camera-captured images
    val isFavorite: Boolean = false,
    val isViewed: Boolean = false
)
```

Code used for filtering and tracking stats using SQL, and CRUD operations

```kotlin
@Dao
interface RecipeDao {
    4 Usages  2 Implementations
    @Query( value = "SELECT * FROM recipes")
    fun getAllRecipes(): Flow<List<Recipe>>

    1 Usage  2 Implementations
    @Query( value = "SELECT * FROM recipes")
    suspend fun getAllRecipesOnce(): List<Recipe>

    2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE id = :id")
    fun getRecipe(id: Int): Flow<Recipe?>

    3 Usages  2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE isFavorite = 1")
    fun getFavoriteRecipes(): Flow<List<Recipe>>

    2 Usages  2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE isViewed = 1")
    fun getRecentlyViewedRecipes(): Flow<List<Recipe>>

    3 Usages  2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE name LIKE '%' || :query || '%' OR ingredients LIKE '%' || :query || '%'")
    fun searchRecipes(query: String): Flow<List<Recipe>>

    2 Usages  2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE isFavorite = 1 AND (name LIKE '%' || :query || '%' OR ingredients LIKE '%' || :query || '%')")
    fun searchFavoriteRecipes(query: String): Flow<List<Recipe>>

    1 Usage  2 Implementations
    @Query( value = "SELECT * FROM recipes WHERE isViewed = 1 AND (name LIKE '%' || :query || '%' OR ingredients LIKE '%' || :query || '%')")
    fun searchRecentlyViewedRecipes(query: String): Flow<List<Recipe>>
```

```kotlin
    3 Usages  2 Implementations
    @Query( value = "SELECT COUNT(*) FROM recipes")
    fun getRecipeCount(): Flow<Int>

    2 Usages  2 Implementations
    @Query( value = "SELECT COUNT(*) FROM recipes WHERE isFavorite = 1")
    fun getFavoriteCount(): Flow<Int>

    2 Usages  2 Implementations
    @Query( value = "SELECT COUNT(*) FROM recipes WHERE isViewed = 1")
    fun getViewedCount(): Flow<Int>

    12 Usages  2 Implementations
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertRecipe(recipe: Recipe): Long

    3 Usages  2 Implementations
    @Update
    suspend fun updateRecipe(recipe: Recipe)

    3 Usages  2 Implementations
    @Delete
    suspend fun deleteRecipe(recipe: Recipe)
}
```

App Database

```
8 Usages  1 Implementation
@Database(entities = [Recipe::class], version = 2, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    1 Implementation
    abstract fun recipeDao(): RecipeDao

    6 Usages
    companion object {
        2 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null

        1 Usage
        private val MIGRATION_1_2 = object : Migration( startVersion = 1,  endVersion = 2) {
            override fun migrate(database: SupportSQLiteDatabase) {
                database.execSQL( sql = "ALTER TABLE recipes ADD COLUMN imagePath TEXT")
            }
        }

        2 Usages
        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    klass = AppDatabase::class.java,
                    name = "cooking_recipe_db"
                )
                    .addMigrations( ...migrations = MIGRATION_1_2)
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Conclusion

I am happy to have a fully functional mobile app.

There are still some things I would have liked to do with this project, such as implement a proper user profile, but I was unsatisfied with the ideas I came up with and implementation I tried, so it is a very simple stats page in this app.

Overall, this project has been a very educational experience. I have applied everything I learned from the labs as well as my prior knowledge in areas like SQL into this project as much as I could.