Escuela de Ingenería Informática

Karl Christian Deilmann

Programacíon de applicacíones moviles nativas

Report 6 - CodeLab Unit 6: Data persistence

5 of Oct - 29 of Oct

ULPGC

Contents

Chapter 1

1.1 CodeLab Task: Build a flight search app

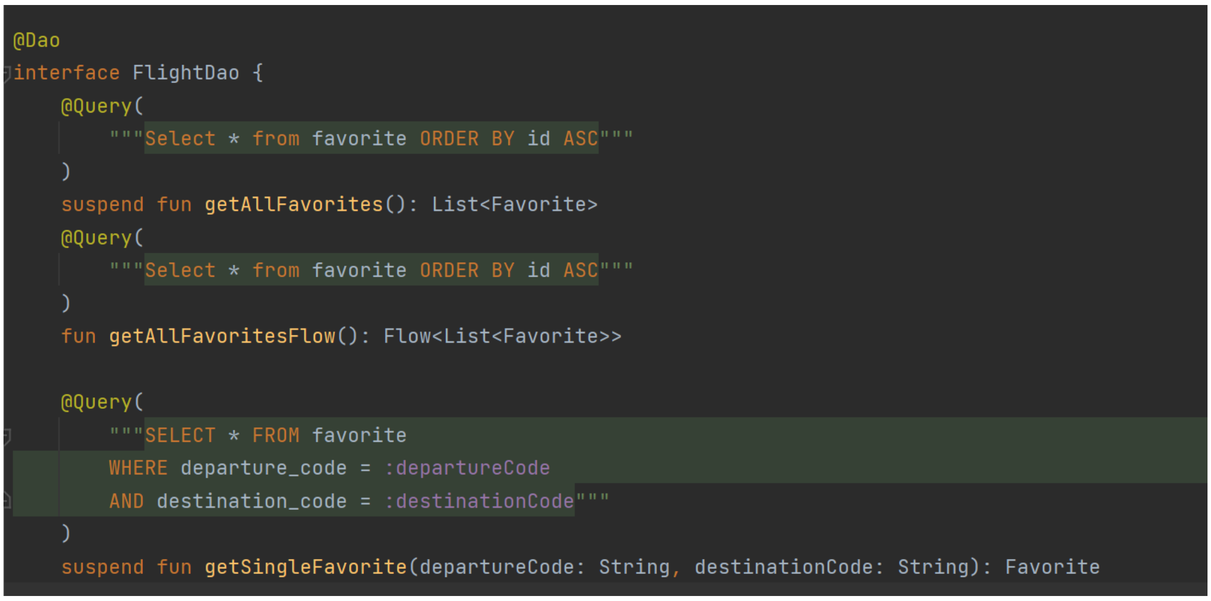
The final task of the CodeLab unit 6 was to build a *Flight search app*. The requirements were:

- Provide a text field for the user to enter an airport name or International Air Transport Association (IATA) airport identifier.
- Query the database to provide autocomplete suggestions as the user types.
- When the user chooses a suggestion, generate a list of available flights from that airport, including the IATA identifier and airport name to other airports in the database.
- Let the user save favorite individual routes.
- When no search query is entered, display all the user-selected favorite routes in a list.
- Save the search text with Preferences DataStore. When the user reopens the app, the search text, if any, needs to prepopulate the text field with appropriate results from the database.

For the design of the screens, we use again the *Jetpack Compose* in Android Studio. Screen items such as the search bar can be simply included and arranged.

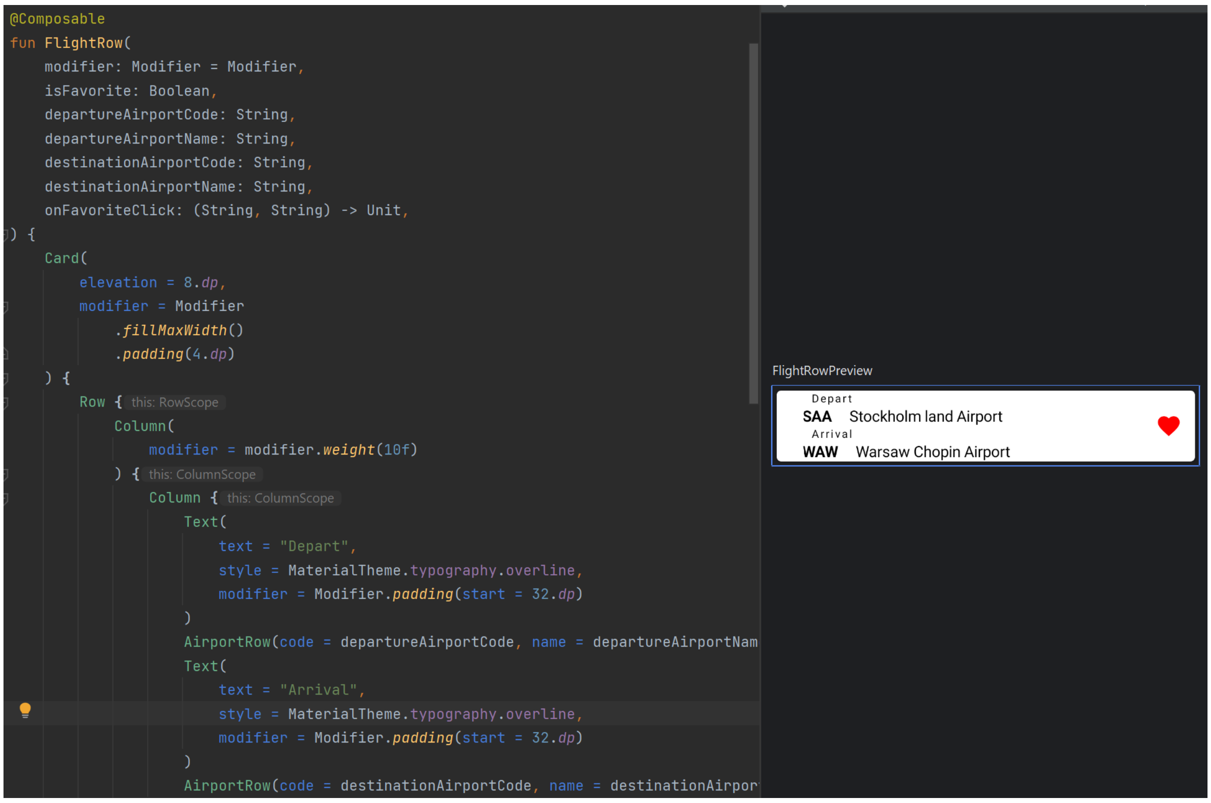
To access the repopulated database, we define ou DAOs to perform the SQL queries. A part of the code can be seen in Fig. 1.1.

To show flight results we build a *Composable* to display the data as seen in Fig. 1.2. We also include a heart-shaped icon to enable a user to save this item to the favourite list. These favourites will be shown on the main screen (the search screen) if the there is no ongoing search. A depiction can be seen in Fig. 1.3c 1.3c. A depiction of the state when searching for the



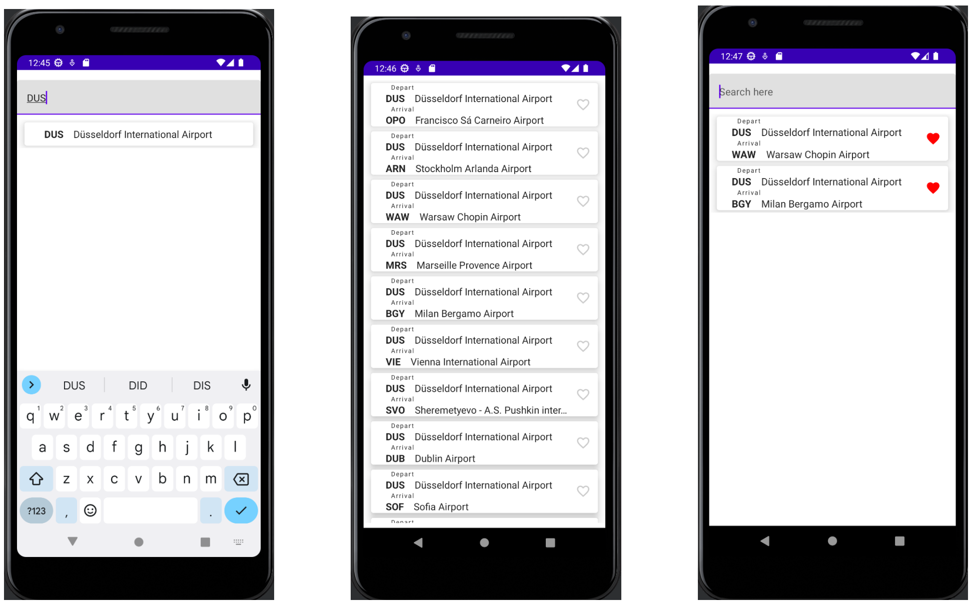
```
@Dao
interface FlightDao {
    @Query(
        """Select * from favorite ORDER BY id ASC"""
    )
    suspend fun getAllFavorites(): List<Favorite>
    @Query(
        """Select * from favorite ORDER BY id ASC"""
    )
    fun getAllFavoritesFlow(): Flow<List<Favorite>>

    @Query(
        """SELECT * FROM favorite
        WHERE departure_code = :departureCode
        AND destination_code = :destinationCode"""
    )
    suspend fun getSingleFavorite(departureCode: String, destinationCode: String): Favorite
```

Figure 1.1: A selection of the DAOs defined for the flight search app



```
@Composable
fun FlightRow(
    modifier: Modifier = Modifier,
    isFavorite: Boolean,
    departureAirportCode: String,
    departureAirportName: String,
    destinationAirportCode: String,
    destinationAirportName: String,
    onFavoriteClick: (String, String) -> Unit,
) {
    Card(
        elevation = 8.dp,
        modifier = Modifier
            .fillMaxWidth()
            .padding(4.dp)
    ) {
        Row { this: RowScope
            Column(
                modifier = modifier.weight(10f)
            ) { this: ColumnScope
                Column { this: ColumnScope
                    Text(
                        text = "Depart",
                        style = MaterialTheme.typography.overline,
                        modifier = Modifier.padding(start = 32.dp)
                    )
                    AirportRow(code = departureAirportCode, name = departureAirportNam
                    Text(
                        text = "Arrival",
                        style = MaterialTheme.typography.overline,
                        modifier = Modifier.padding(start = 32.dp)
                    )
                    AirportRow(code = destinationAirportCode, name = destinationAirpor
```

Figure 1.2: Preview and parts of code for a flight info item as a composable


12:45
DUS
DUS Düsseldorf International Airport

12:46
Depart DUS Düsseldorf International Airport
Arrival OPO Francisco Sá Carneiro Airport
Depart DUS Düsseldorf International Airport
Arrival ARN Stockholm Arlanda Airport
Depart DUS Düsseldorf International Airport
Arrival WAW Warsaw Chopin Airport
Depart DUS Düsseldorf International Airport
Arrival MRS Marseille Provence Airport
Depart DUS Düsseldorf International Airport
Arrival BGY Milan Bergamo Airport
Depart DUS Düsseldorf International Airport
Arrival VIE Vienna International Airport
Depart DUS Düsseldorf International Airport
Arrival SVO Sheremetyevo - A.S. Pushkin inter...
Depart DUS Düsseldorf International Airport
Arrival DUB Dublin Airport
Depart DUS Düsseldorf International Airport
Arrival SOF Sofia Airport

12:47
Search here
Depart DUS Düsseldorf International Airport
Arrival WAW Warsaw Chopin Airport
Depart DUS Düsseldorf International Airport
Arrival BGY Milan Bergamo Airport

(a) (b) (c)

Figure 1.3: (a) Search screen with a search request entered, showing results for "*DUS*" Airport
(b) Results of the search query that includes "*DUS*"
(c) Search screen with favourites displayed below

"*DUS*" is shown in 1.3a alongside with the results of all the flight containing said airport in Fig. 1.3b. The results containing the airport are accessed by tapping on the search result itself.

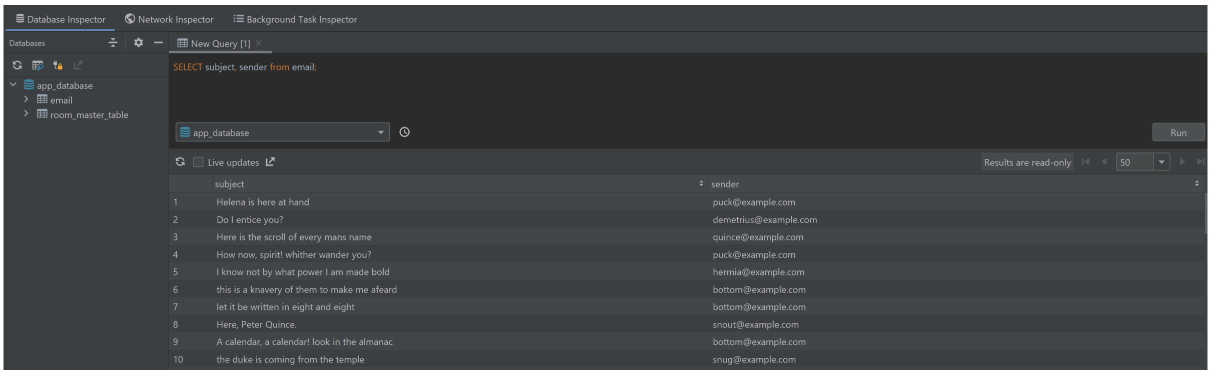
Finally to save the state of the search bar, we use Android *DataStore* to memorize and retrieve the content of the search bar on start of the app. We store the search string and reclaim it on start-up again. The results are then fetched from the SQLite database as per usual.

1.2 CodeLab: A report on usability

The CodeLab tutorials walk us through the main aspects of using a SQLite database to store and retrieve data efficiently. In Fig. 1.4 we see an example on how to use SQL queries to retrieve elements from the data base. The auto-generation features of Kotlin help facilitate the process of adding new query calls to a project. The Android*Room* is used to tie everything together and use the data in the view of an app. For non-relational data such as preferences, user settings and app states, the tutorial provides basic knowledge about Android *DataStore*.

Problems

Android Studio asks to update *Gradle*, but after the update process, the project breaks. This leads us to believe that the tutorials are not up to date. Furthermore, some of the demo projects



```
SELECT subject, sender from email;
```

	subject	sender
1	Helena is here at hand	puck@example.com
2	Do I entice you?	demetrius@example.com
3	Here is the scroll of every mans name	quince@example.com
4	How now, spirit! whither wander you?	puck@example.com
5	I know not by what power I am made bold	hermia@example.com
6	this is a knavery of them to make me afeard	bottom@example.com
7	let it be written in eight and eight	bottom@example.com
8	Here, Peter Quince.	snout@example.com
9	A calendar, a calendar! look in the almanac	bottom@example.com
10	the duke is coming from the temple	snug@example.com

Figure 1.4: A CodeLab example on how to use SQL queries

(both starter and solution code) straight up refuse to compile and work. Especially the *Bus Schedule* demo was not usable at all and needed some time to find the problem/conflict in the code.

Personal notes

Android *Room* is convoluted and complex. A lot has to be accepted as *given* without further explanation. It is hard for newcomers to understand the concepts because already in the tutorials, they use exclusively optimised and high-level code structures which, while they might help learn the best practices, make it really hard to understand the concepts of the tutorial. It feels like the tutorial is not helping much without learning the entire *Room* documentation by heart on the side.