Android Fundamentals Project Self-Evaluation

Instructions: Once you've completed your Final Project, please respond to the questions below. This is a chance for you to briefly explain to the grader your thought-process during development. Once you are done, include this with the source code and accompanying files you are submitting. Then, give yourself a pat on the back for making a great app!

Questions about Required Components

Permissions

Please elaborate on why you chose the permissions in your app.

Currently the following permissions are used:

- **android.permission.INTERNET**:
  Required in order to get information from the external REST services.
- **android.permission.READ_SYNC_SETTINGS**:
  Required in order to read sync adapter settings.
- **android.permission.WRITE_SYNC_SETTINGS**:
  Required in order to control sync adapter settings, like adding a periodic sync.
- **android.permission.AUTHENTICATE_ACCOUNTS**:
  Required in order to use an authenticator for the sync adapter.

Content Provider

What is the name of your Content Provider, and how is it backed? **(For example, Sunshine's Content Provider is named `WeatherProvider` backed by an SQLite database, with two tables: `weather` and `location`.)**

The Content Provider is called **net.oldervoll.flightschedule.data.FlightProvider**, and it is backed by an **SQLite database**. The database contains the following tables:

- **airline**: for storing information about all airlines used by flights.
- **airport**: for storing information about all airports used by flights.
- **status**: for storing information about status codes used by flights.
- **flight**: for storing information about flights. This is where the flight schedule is stored.

What backend does it talk to? **(For example, Sunshine talks to the OpenWeatherMap API.)**

The Content Provider talks to a REST service from the Norwegian company Avinor (description only available in Norwegian). This application uses the following REST endpoints for fetching data:

- Flight information: List of flights for a specified airport. This includes name of airport the flight arrives from or departs to, the name of the airline, the scheduled time, any status time and status message, the gate, the luggage belt etc. No need to update these data more often than every 3 minutes.
- Airline information: Mapping between airline codes (identifiers) and the name of the airline (like BA=British Airways). No need to update these data more than once every 24 hours.
- Airport information: Mapping between airport codes (identifiers) and the name of the airport (like BGO=Bergen). No need to update these data more than once every 24 hours.
- Status code information: Mapping between status codes (identifiers) and the name of the status (like D=Departed). No need to update these data more than once every 24 hours.

If your app uses a SyncAdapter, what is it called? What mechanism is used to actually talk over the network? (For example, Sunshine uses `HttpURLConnection` to talk to the network, but your app may use a third-party library to do the talking.)

The app uses a SyncAdapter called **net.oldervoll.flightschedue.sync.SyncAdapter** and uses the Retrofit library to talk with the backend over HTTP using a REST service provided by Avinor. The REST service is defined in **net.oldervoll.flightschedule.remote.AvinorService** and receives a payload in XML format. The XML data is parsed to Java classes using the Simple framework. The Retrofit REST adapter is built in the **net.oldervoll.flightschedule.remote.AvinorServiceImpl()** constructor.

What loaders/adapters are used?

Data is loaded from the Content Provider into a **ListView** using a **CursorLoader** defined in **net.oldervoll.flightschedule.FlightFragment**. The **CursorAdapter** **net.oldervoll.flightschedule.FlightAdapter** is used to populate the ListView with data from the CursorLoader.

For the detail view the **net.oldervoll.flightschedule.DetailFragment** contains a CursorLoader. Here the views are populated with data directly from the **onLoadFinished** callback, so no adapters are used here.

User/App State

Please elaborate on how/where your app correctly preserves and restores user or app state. (See rubric for examples on this question)

In **net.oldervoll.flightschedule.FlightFragment** the position in the **ListView** is saved in a **Bundle** in the **onSaveInstanceState** callback method. The position is then fetched in the **onCreateView** callback method from the **savedInstanceState** Bundle if that is not null. The ListView is then scrolled to the saved position in the **onLoadFinished** callback method. In this way it is possible to rotate the app and still preserve the position in the list.

The following user state is also saved and persisted from the **net.oldervoll.flightschedule.SettingsActivity** to **SharedPreferences**:

- The **airport** which we should receive data for.
- The **flight** that notifications should be generated for.
- A flag for enabling or disabling notifications.
- A flag for enabling or disabling rich content in the ShareActionProvider.
- The time for the last sync of data updated every 24 hours.
- The time for the last notification sent (used to prevent overwhelming the user with notifications).

Questions about Optional Components

Answer the questions that are applicable to your final project

Notifications

Please elaborate on how/where you implemented Notifications in your app:

The user can select a flight for which he wants notifications when the status changes. This can for instance be a flight the user is going to travel with. These notifications gives the user instant feedback when a flight is delayed, when the gate is changed etc. Notifications are sent for every state change, as well as 60 minutes before departure of a flight. Notifications are based on the effective schedule time for a flight, taken any delays into account.

Notifications are sent from the **net.oldervoll.flightschedule.sync.SyncAdapter** in the **onPerformSync** callback method, right after the data is loaded from the remote REST service. We make sure to minimise the number of notifications that are sent, and also to reuse existing notifications that are not removed. The **NotificationCompat.Builder** class is used to create notifications, and the **NotoficationManager** is used to send them.

ShareActionProvider

Please elaborate on how/where you implemented ShareActionProvider:

The ShareActionProvider is implemented in **net.oldervoll.flightschedule.DetailFragment**. A description of a flight is created in the **onLoadFinished** callback, and this description is made available via ShareActionProvider. The shared intent can be created in two different types:

- A plain text version containing status information about the flight.
- A rich content (HTML) version containing status information about the flight, the arrival/destination icon for the flight and a link to detailed flight information about this specific flight at the Avinor web site. It also contains an image visualising either arrival or departure.

Broadcast Events

Please elaborate on how/where you implemented Broadcast Events:

The detail fragment in the app includes a custom view **net.oldervoll.flightschedule.CompassView** that shows the direction between the departing airport and the arriving airport. In order to get the direction the **Google Geocoding API** is used. The call to the Google Geocoding API is done using the Retrofit REST client and is implemented in **net.oldervoll.flightschedule.remote.GoogleGeocodingService**. This service is called from the detail fragment using the **IntentService** called **net.oldervoll.flightschedule.GeoService**.

The detail fragment registers a **BroadcastReceiver net.oldervoll.flightschedule.DetailFragment$GeoReceiver** with **LocalBroadcastManager** in order to get notified when the calculation of direction has finished (the registration is done from the **onCreateView** method in DetailFragment because we need the **ViewHolder** object in the BroadcastReceiver. The GeoService then uses the LocalBroadcastManager to broadcast an intent with the direction when that has been calculated (sendBroadcast is done from the **onHandleIntent** method in **net.oldervoll.flightschedule.GeoService**.

The use of BroadcastReceiver and Custom View is not a crucial part of the app. It is primarily used in order to learn these mechanisms. A more useful usage of the same mechanisms could have been to show weather information including wind direction for the airports.

Custom Views

Please elaborate on how/where you implemented Custom Views:

A **Custom View** in form of a compass is used to visualise the flight direction (from the departing airport to the arriving airport). The custom view is implemented in the class **net.oldervoll.flightschedule.CompassView** and is used is the layouts **fragment_detail.xml** and **fragment_detail_wide.xml**. The direction of the compass needle is set from the method **onReceive** in the **BroadcastReceiver** called **GeoReceiver** (inside **net.oldervoll.flightschedule.DetailFragment**).

The use of BroadcastReceiver and Custom View is not a crucial part of the app. It is primarily used in order to learn these mechanisms. A more useful usage of the same mechanisms could have been to show weather information including wind direction for the airports.