FruitApp Report

Petko Mikov

FruitApp is a simple Android app, that uses Compose for the UI, and additionally ViewModel, Room and RetroFit among others. In summary the app fetches data about fruits from a REST API FruitVice (https://www.fruityvice.com/). After receiving this data it is stored in a List as a Fruit object, nutrition is saved in a saved in a separate Nutrition object, which is referenced in the Fruit object. This was done due to the format of the JSON Api response, where nutrition is presented as an object. Subsequently, the retrieved data is displayed in a scrollable list on the home screen of the app. Initially, only basic information such as the name, family, order, and genus of the fruit are visible. Clicking on a fruit reveals more detailed information, and users can add or remove fruits from their favourites list. Navigation between the home and favourites screens is facilitated by the navigation bar (or rail/drawer for different screen sizes). To explain the structure of the application and choices made along the way, ill separate the rest of this document in a couple of main points.

Web Service

The Api service is defined by an interface and include one method which defines a get request to the "/all" api endpoint. RetroFit deals with the implementation. The retrofit builder is defined in the default app container. It also transforms the incoming JSON into Fruit object by using a converter.

Repository

One repository interface is defined, which is than extended the *FruitRepositoryImpl* class. In the beginning this was separated in two implementing classes – one for the API and one for the database, because the learning material only including examples for one or the other and it was unclear how to combine them. However, during development, it became evident that combining them was more practical. FruitRepositoryImpl serves as the implementation, calling either the API service or the fruitDao based on the method. FruitDao is a simple interface annotated with Room's @Dao, containing three methods for interacting with the database. Fruits are stored based on the FruitDB entity, which flattens the structure of the Fruit object, as it was deemed unnecessary to store nutrition in a separate table and reference it.

UI State & ViewModel

The *FruitUiState* include values for the current page type, a map which maps page type to a list of fruits, current selected Fruit, and a Boolean for whether the main or details page is displayed. The UiState is controlled by the ViewModel. There is a private mutable value and an immutable state flow that is exposed to external classes. The view model includes functions to add and remove some fruit from favourites, to fetch fruits from the database and the database, and to update the current shown fruit and page. The view model implements the singleton pattern and has a companion object with an initializer.

Compose – Views Structure

The initial Composable function that is called is *FruitHomeScreen* from Kotlin file with the same name. Here a list of *NavigationItemContent* is initiated, after which a *PermanentNavigationDrawer* is created and *FruitAppContent* is called, which include the list with the fruits, in the case of bigger devices. For smaller devices *FruitAppContent* or *FruitDetailsScreen* is called based on an *isShowingHomepage* variable in the *UiState*. The app does not use *NavControlle/NavHost/NavGraph* since I deemed that to be unnecessary for the required purposes. A navigation bar or rail is created in *FruitAppContent* for small or midsize devices, respectively.

FruitAppContent is called for all screen sizes. It creates the *NavigationRail* for mid-sized screen and the *BottomNavigationBar* for small ones. For bigger screen it calls *FruitListAndDetailContent*, and for small and midsized screen *FruitListOnlyContent*. Both composables create a *LazyColumn* for the Fruits. This was chosen because of the dynamic loading and scrolling functionalities, which it implements. Each fruit is in a *FruitListItem* composable. The list elements are costumized with cut-off diagonal edges through a Shape value that is used when instantiating the *FruitAppTheme* class. *FruitListOnlyContent* also creates the header in the main screen. *FruitListAndDetailContent* also call the *FruitDetailScreen*. Therefore the app List-Detail UI layout pattern.

When clicking on some fruit from the list *isShowingHomepage* and *currentSelectedFruit* are updated in the UiState. Based on those details page is shown for non-big screens with the information for the appropriate fruit. Apart from the detailed fruit information, this view also includes a button to add / remove some fruit from favourite. The appropriate button is shown based on Boolean variable that checks whether the *currentSelectedFruit* is contained in the list of favourite fruits.

Utility Classes

There are three enum classes for storing state variable – FruitNavigationType (what kind of navigation is used, based on the screen size – drawer, rail, or bar), FruitContentType (whether details view is displayed alongside the list), and PageType (home for all fruits or favourites only)

Material Theme

The Material Theme, a standard library for customizing Android apps, was utilized via the Material Theme Builder to select colors that match the raspberry logo. The main color is pink, with different shades, complemented by blue in some areas. Additionally, view shapes were customized, visible in list elements, the details view, and the add/remove button.