

Flutter Fundamentals

Complete application



Peter Kassenaar –
info@kassenaar.com



Complete application

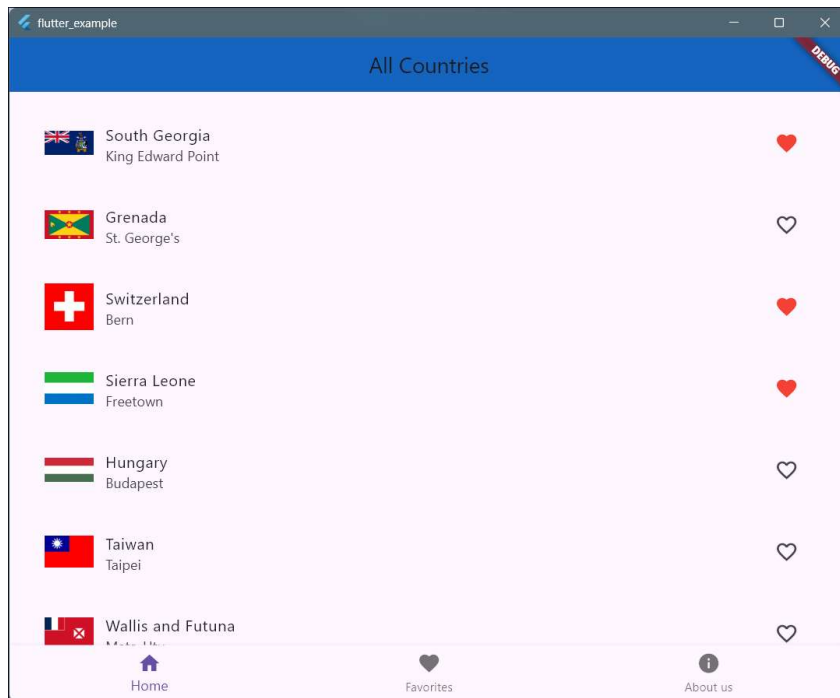
A complete application, using http for communication,
putting results in a bloc store and using bottom navigation

Requirements



1. A [complete application](#), using http for communication to fetch countries
2. Put fetched countries in a [bloc / state management](#) store
3. Show a [list](#) of fetched countries
4. Mark a country as [Favorite/Unfavorite](#)
5. Show a [list of favored countries](#)
6. Show an [About us](#) page
7. Use a [BottomNavigationBar](#) to switch between screens

Visual Result



1. update CountriesBloc



Starting point: `_430-payload`, fetching a list of countries and putting it in the store


- First: **extend the bloc** to listen to a `ToggleFavorite` event

```
class CountriesBloc extends Bloc<CountriesEvent, CountriesState> {  
  CountriesBloc() : super(CountriesInitial()) {  
    // 1. Listen to the FetchCountries event.  
    on<FetchCountries>((event, emit) async {}  
  
    // 2. Listen to the ToggleFavorite event  
    on<ToggleFavorite>((event, emit){}  
  }  
}
```

Event ToggleFavorite in CountriesBloc



- Approach: `check if a country exists` in the array `favorites` and add/remove it
- Then, `emit` the new state with
 - `Countries`
 - `Favorites`



```
on<ToggleFavorite>((ToggleFavorite event, Emitter<CountriesState> emit){
  if (state is CountriesLoaded) {
    final currentState = state as CountriesLoaded;
    final favorites = List<Map<String, dynamic>>.from(currentState.favorites);

    // 5. Check if the country exists
    final bool existingCountry = favorites.any(
      (country) =>
        country['name']['common'] == event.country['name']['common'],
    );

    // 6. Toggle favorite
    if (existingCountry != false) {
      favorites.remove(event.country); // Remove from favorites
    } else {
      favorites.add(event.country); // Add to favorites
    }

    // 7. Emit new state
    emit(
      CountriesLoaded(
        countries: currentState.countries,
        favorites: favorites,
      ),
    );
  }
});
```



2. Update CountriesEvent

- Extend `countries_event.dart`
- Add an event to toggle the favorite state:

```
// ...  
class FetchCountries extends CountriesEvent {}  
  
// 3. Event: marking a country as Favorite. The list consists  
// of a Map<String, dynamic>, containing a list of favorite countries.  
class ToggleFavorite extends CountriesEvent {  
  final Map<String, dynamic>country;  
  
  ToggleFavorite({required this.country});  
  
  @override  
  List<Object?> get props => [country];  
}
```




3. Update CountriesState

The class `CountriesState` represents all possible states of `CountriesBloc`

- Add the list of favorites to the `CountriesState` class
- Also return the list of favorites!

Code for countries_events.dart

```
...
// 2. The initial state
class CountriesInitial extends CountriesState {}

...
// State property to hold the successfully fetched list of countries.
// 4. It is now extended with a list of favorite countries.
class CountriesLoaded extends CountriesState {
  final List countries;
  final List favorites;

  // constructor, using named properties here
  CountriesLoaded({required this.countries, required this.favorites});

  // 5. When getting the state, return countries AND favorites.
  @override
  List<Object?> get props => [countries, favorites];
}
...
```



4. Update All Countries screen

Updating the Home Screen, containing all countries
(home_countries.dart)

- Major change: add a `trailing` icon to toggle the favorite state
- On **pressing the Favorite icon**, emit the `ToggleFavorite()` event for the current country
- We can (still) use a `StatelessWidget()` for this.

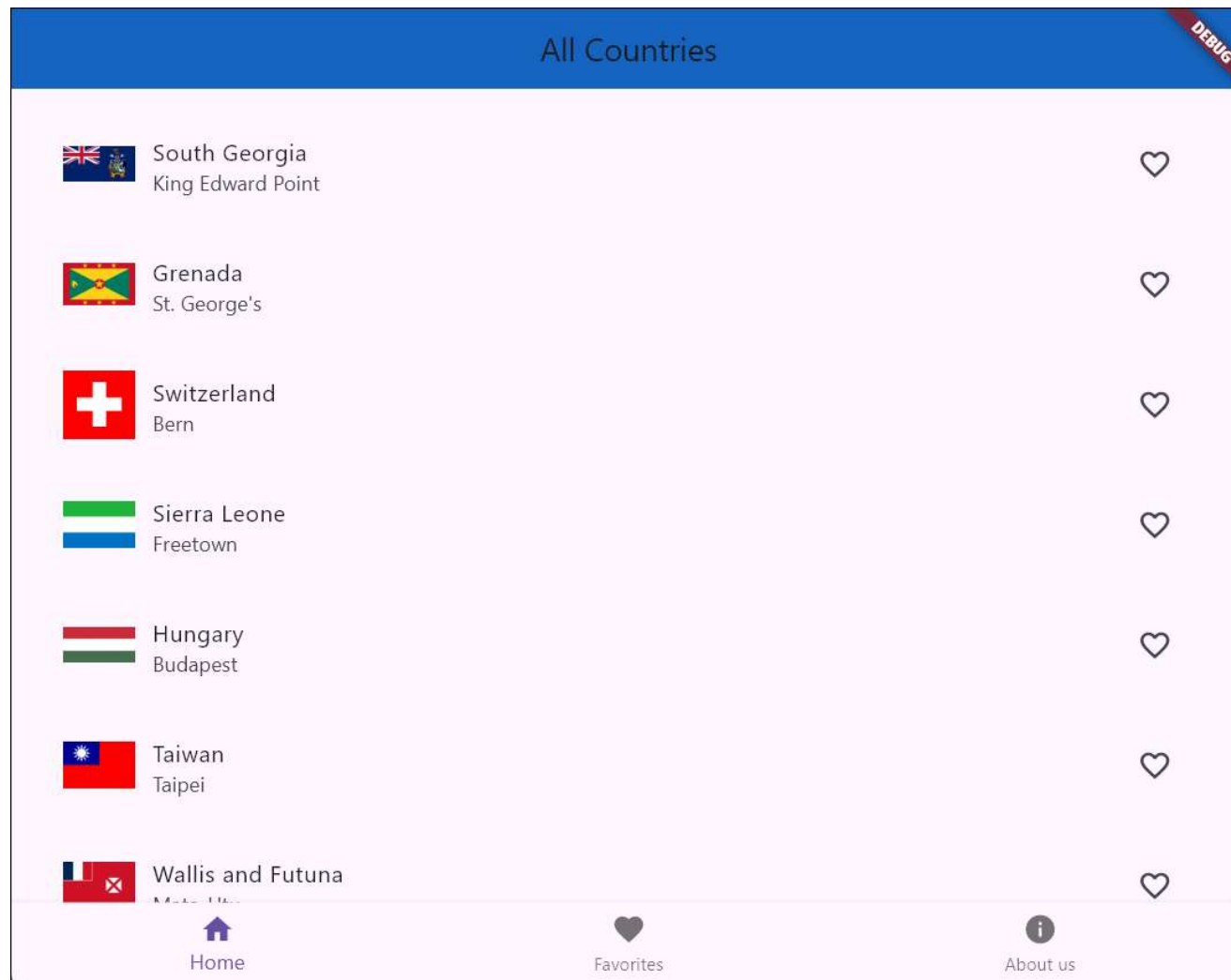
Code for countries_home.dart



```
// abstract...
itemBuilder: (BuildContext context, int index) {
  final country = state.countries[index];
  final isFavorite = state.favorites.contains(country);

  // 6. Every list item wrapped in a Padding() widget.
  return Padding(
    ...
    // 8. A trailing button to toggle the 'Favorite' state
    trailing: IconButton(
      icon: Icon(
        isFavorite ? Icons.favorite : Icons.favorite_border,
        color: isFavorite ? Colors.red : null,
      ),
      onPressed: () {
        // 9. Emit event if the Favorite state is toggled.
        context.read<CountriesBloc>().add(
          ToggleFavorite(country: country),
        );
      },
    ),
    ...
  );
}
```

Visual for countries_home.dart





5. Create a 'Favorites' screen

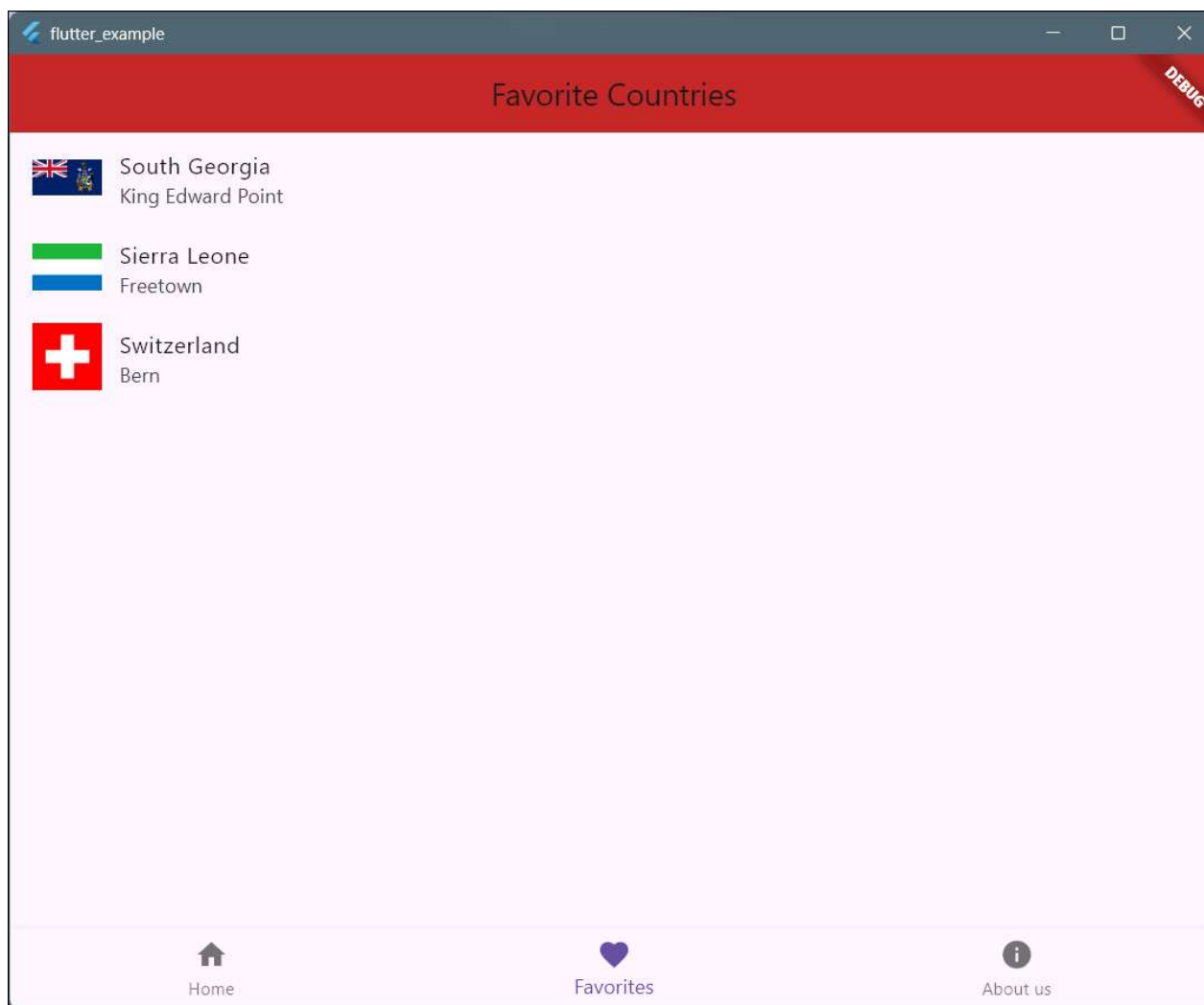
- Create `countries_favorites.dart`
- A simple screen that shows all favorite countries from the state
- Also use a `StatelessWidget()` here, because all state comes from the store/bloc!

Code for countries_favorites.dart



```
...
class CountriesFavorites extends StatelessWidget {
  Widget build(BuildContext context) {
    ...
    body: BlocBuilder<CountriesBloc, CountriesState>(
      builder: (context, state) {
        if (state is CountriesLoaded) {
          if (state.favorites.isEmpty) {
            return const Center(child: Text('No favorite countries yet.));
          }
          // because this is the same ListView.builder() as in countries_home.dart, we
          // could also move this to its own widget/file.
          return ListView.builder(
            itemCount: state.favorites.length,
            itemBuilder: (context, index) {
              ...
            );
          } else {
            return const Center(child: CircularProgressIndicator());
          }
        },
      ),
    ...
  }
}
```

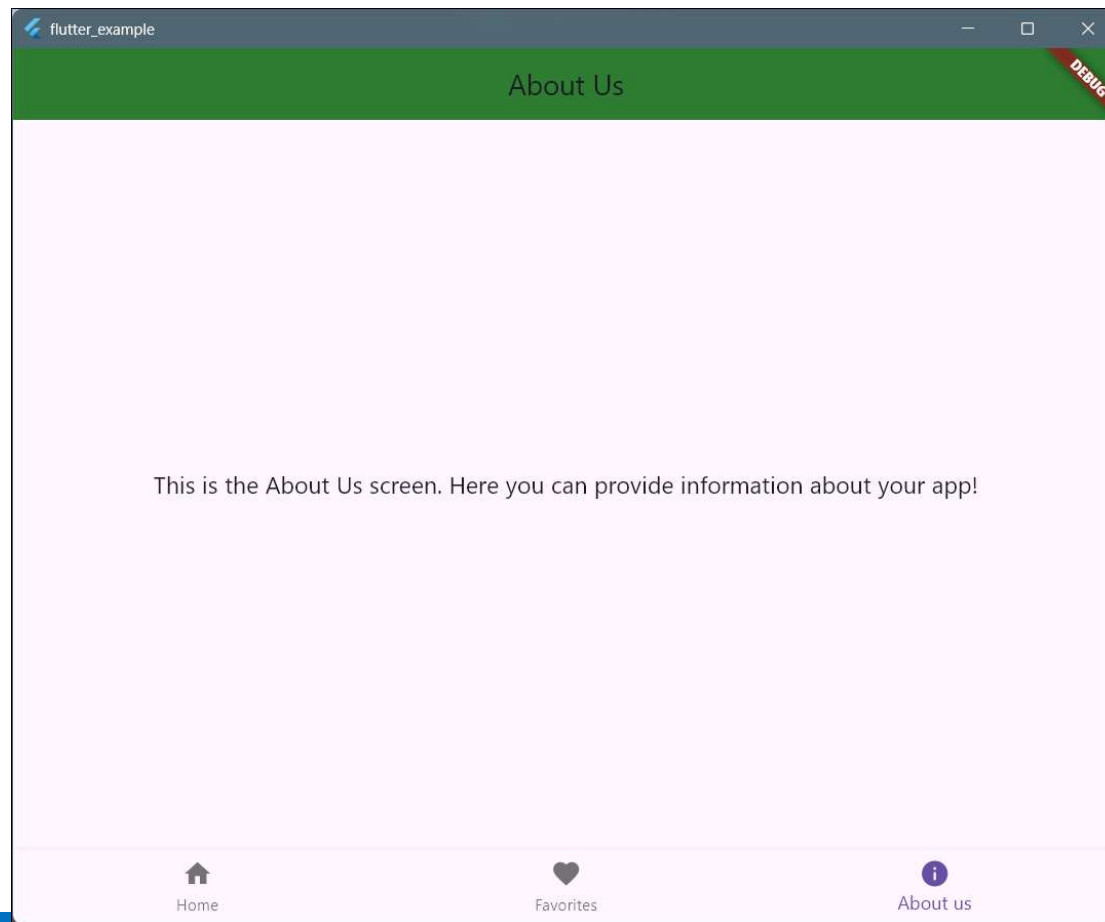
Visual for `countries_favorites.dart`



6. Create the 'About us' screen



- Create `about.dart` and fill with relevant data





7. Update Main application

- Update `main.dart` to integrate the `BottomNavigationBar` with the three screens
- Also provide the `BlocProvider()`, since state is used in multiple screens
 - Otherwise we would have to create separate `BlocProvider()`'s on each page.

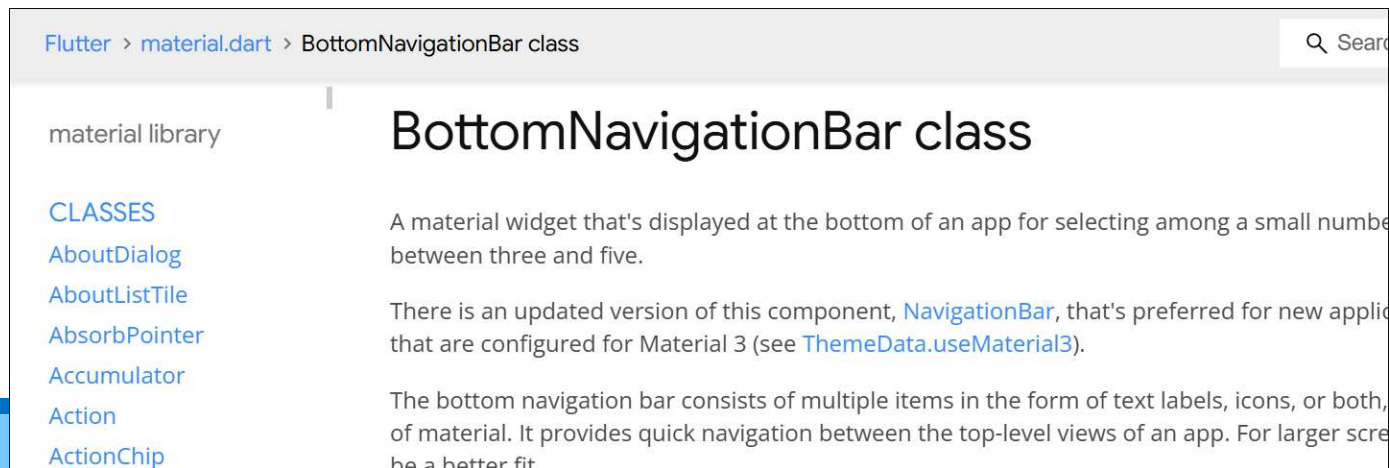


```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return BlocProvider(  
      create: (context) => CountriesBloc()..add(FetchCountries()),  
      child: MaterialApp(  
        // The Bottom navigation bar now holds the  
        // Scaffold() that wraps all pages/screens  
        home: const BottomNavBar(),  
      ),  
    );  
  }  
}
```



8. Create Bottom Navigation Bar

- Bottom Navigation Bar has a list of `_pages`
- Pages are bound to `BottomNavigationBarItems()`
- There is a generic function that sets the `index`
- The page belonging to the `index` is assigned as the `body:` of the `Scaffold()`
- api.flutter.dev/flutter/material/BottomNavigationBar-class.html



Main.dart logic



```
// start with the first icon selected.
int _selectedIndex = 0;

// A list with all possible pages from the app
final List<Widget> _pages = [
  const CountriesHome(),
  const CountriesFavorites(),
  const AboutScreen(),
];

// The user tapped a specific item in the bottom navigation bar.
// Set its index in the state
void _onNavigationItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}
```



Main.dart UI



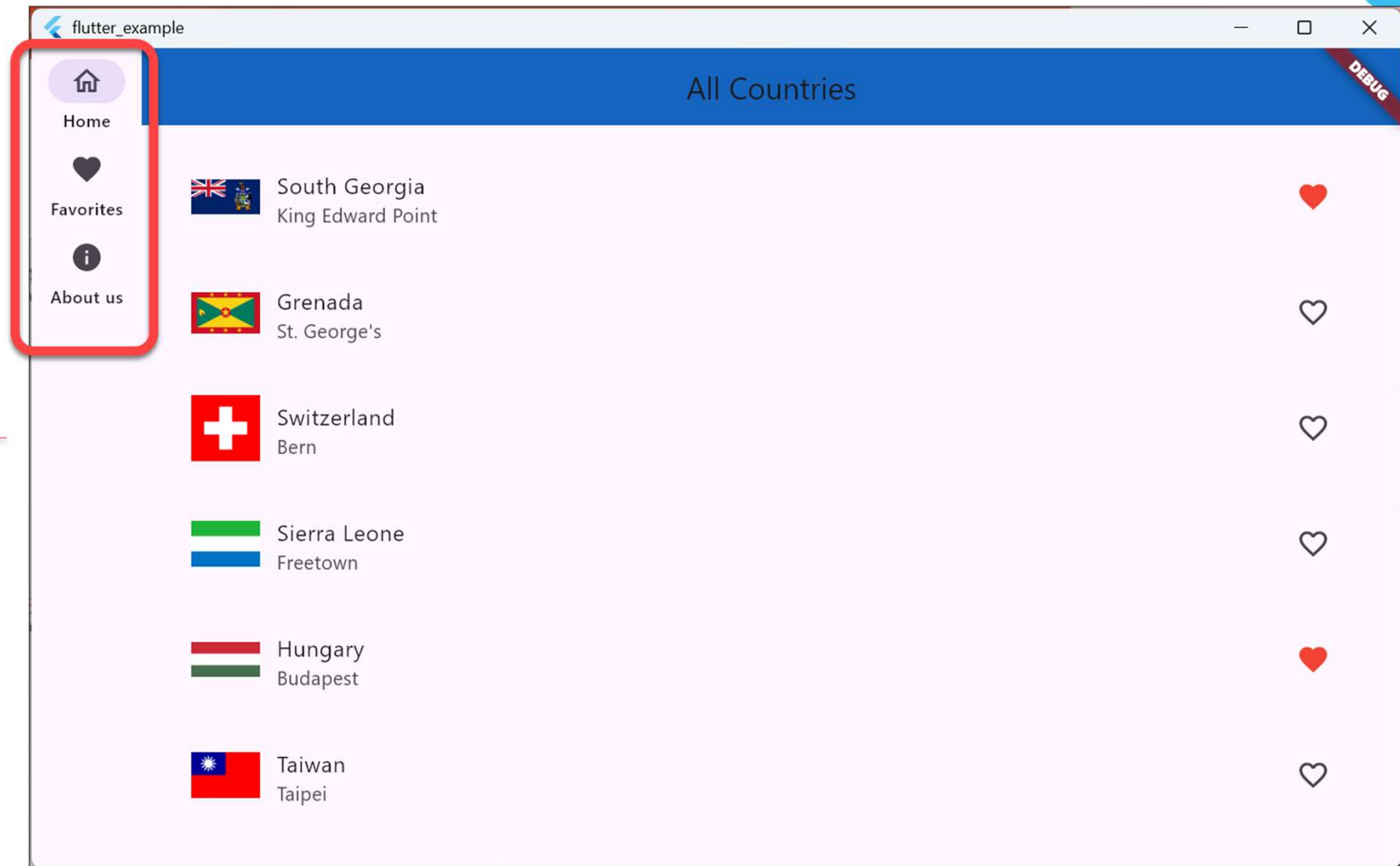
```
Widget build(BuildContext context) {  
  return Scaffold(  
    body: _pages[_selectedIndex],  
    bottomNavigationBar: BottomNavigationBar(  
      currentIndex: _selectedIndex,  
      onTap: _onNavigationItemTapped,  
      items: const [  
        BottomNavigationBarItem(  
          icon: Icon(Icons.home),  
          label: 'Home'),  
        BottomNavigationBarItem(  
          icon: Icon(Icons.favorite),  
          label: 'Favorites'),  
        BottomNavigationBarItem(  
          icon: Icon(Icons.info),  
          label: 'About us'),  
      ]),  
  );  
}
```

On Desktop: `NavigationRail()` widget



- On Desktop applications, a `NavigationRail()` might be more suitable
- Same principle, some changes:
 - Now use a `Row()` for the main layout
 - Wrap the content in an `Expanded()` widget
- Lots of properties for customizing available:
 - `selectedIndex` - Tracks the currently selected index.
 - `onDestinationSelected` - Handles user interactions and updates the `_selectedIndex`
 - `Destinations` - Lists the items shown in the navigation rail with icons and text labels.

App with NavigationRail()



More info

[Flutter](#) > [material.dart](#) > [NavigationRail class](#)

Search API Docs

material library

CLASSES

[AboutDialog](#)

[AboutListTile](#)

[AbsorbPointer](#)

[Accumulator](#)

[Action](#)

[ActionChip](#)

[ActionDispatcher](#)

[ActionIconTheme](#)

[ActionIconThemeData](#)

[ActionListener](#)

[Actions](#)

[ActivateAction](#)

[ActivateIntent](#)

[Adaptation](#)

[AdaptiveTextSelectionT...](#)

[AlertDialog](#)

[Align](#)


[Alignment](#)

[AlignmentDirectional](#)

[AlignmentGeometry](#)

NavigationRail class

A Material Design widget that is meant to be displayed at the left or right of an app to navigate between a small number of views, typically between three and five.



The navigation rail is meant for layouts with wide viewports, such as a desktop web or tablet landscape layout. For smaller layouts, like mobile portrait, a [BottomNavigationBar](#) should be used instead.

A navigation rail is usually used as the first or last element of a [Row](#) which defines the app's [Scaffold](#) body.

The appearance of all of the [NavigationRails](#) within an app can be specified with [NavigationRailTheme](#). The default values for null theme properties are based on the [Theme](#)'s [ThemeData.textTheme](#), [ThemeData.iconTheme](#), and [ThemeData.colorScheme](#).

Adaptive layouts can build different instances of the [Scaffold](#) in order to have a navigation rail for more horizontal layouts and a bottom navigation bar for more vertical layouts. See [the adaptive_scaffold.dart sample](#) for an example.

<https://api.flutter.dev/flutter/material/NavigationRail-class.html>

Workshop



- Use `../examples/_440-complete-application` as inspiration
- Create your own application using the JSONPlaceholder API with users
- Fetch users, add the option to toggle users as favorite/unfavorite
- **Optional:** implement *swipe-to-delete* from Favorites

