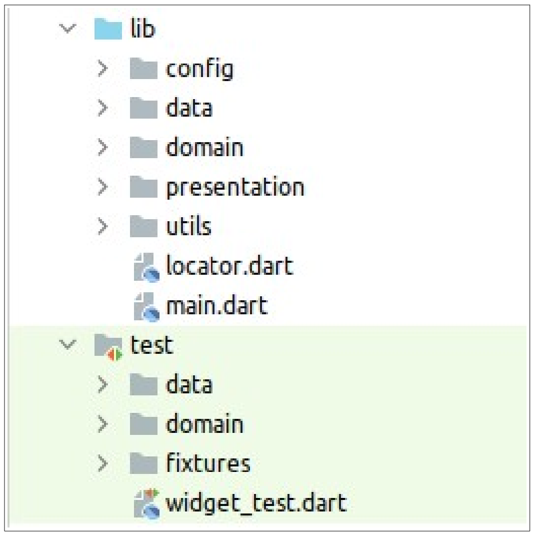
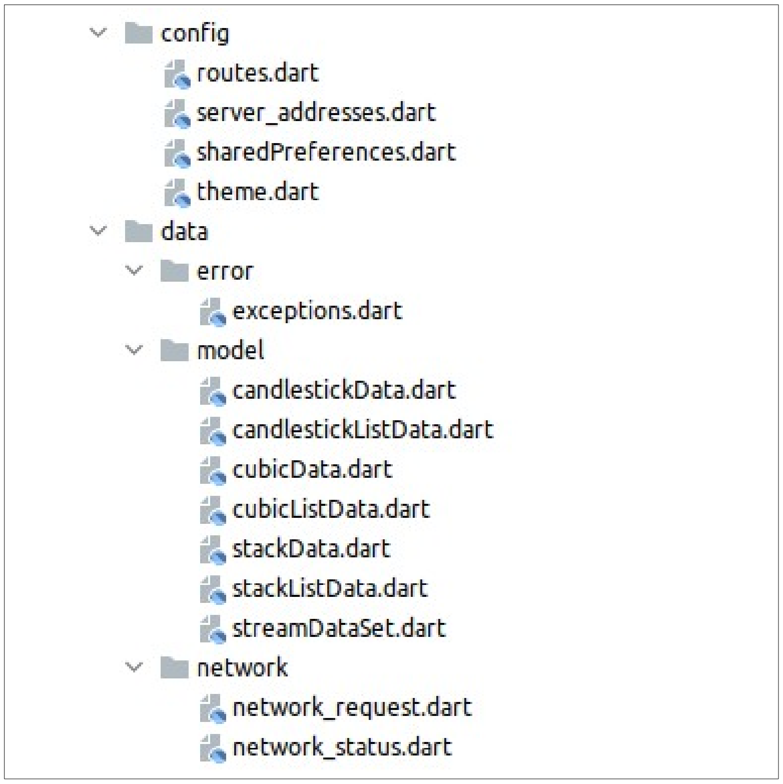
Document



```
lib
  config
  data
  domain
  presentation
  utils
  locator.dart
  main.dart
test
  data
  domain
  fixtures
  widget_test.dart
```

In this project I’ve separated the responsibilities and used technologies to increase de-coupling and also make the project testable and scalable.

I’ve tried to separate the presentation from business logic and the architecture of the project makes it easy to maintain the code and develop it and although it was a sample project, its structure really ready to be extended.

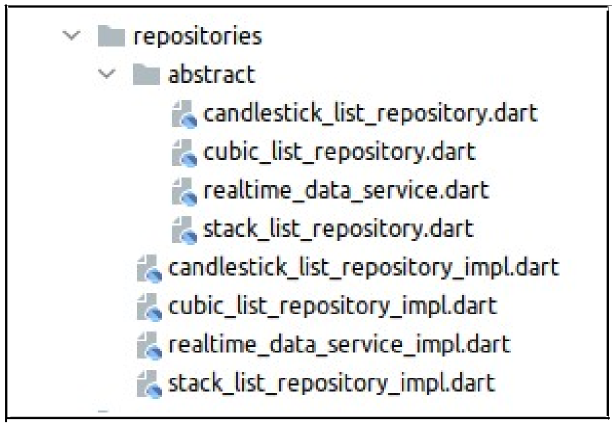


```
config
    routes.dart
    server_addresses.dart
    sharedPreferences.dart
    theme.dart
data
    error
        exceptions.dart
    model
        candlestickData.dart
        candlestickListData.dart
        cubicData.dart
        cubicListData.dart
        stackData.dart
        stackListData.dart
        streamDataSet.dart
    network
        network_request.dart
        network_status.dart
```

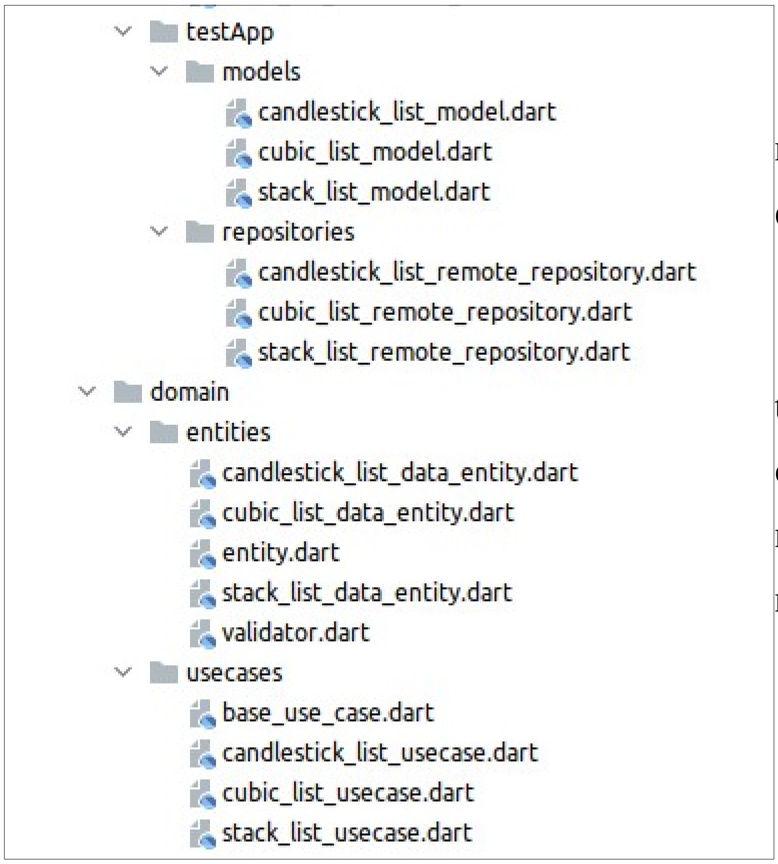
The configuration of the project includes the server addresses and colors and provides the routes of project parts and also implements the communication by sharedPreferences.

Data folder includes exception info and the model of data that is used in API calls. The data structure is defined in such a way that it can accept data in Json format. The responsibilities of network request including different types of requests and preparing the data to call api are placed in the network folder.

The other part of data folder is repositories that is used to abstract and implement the important part of preparing server requests and also checking the network connection.



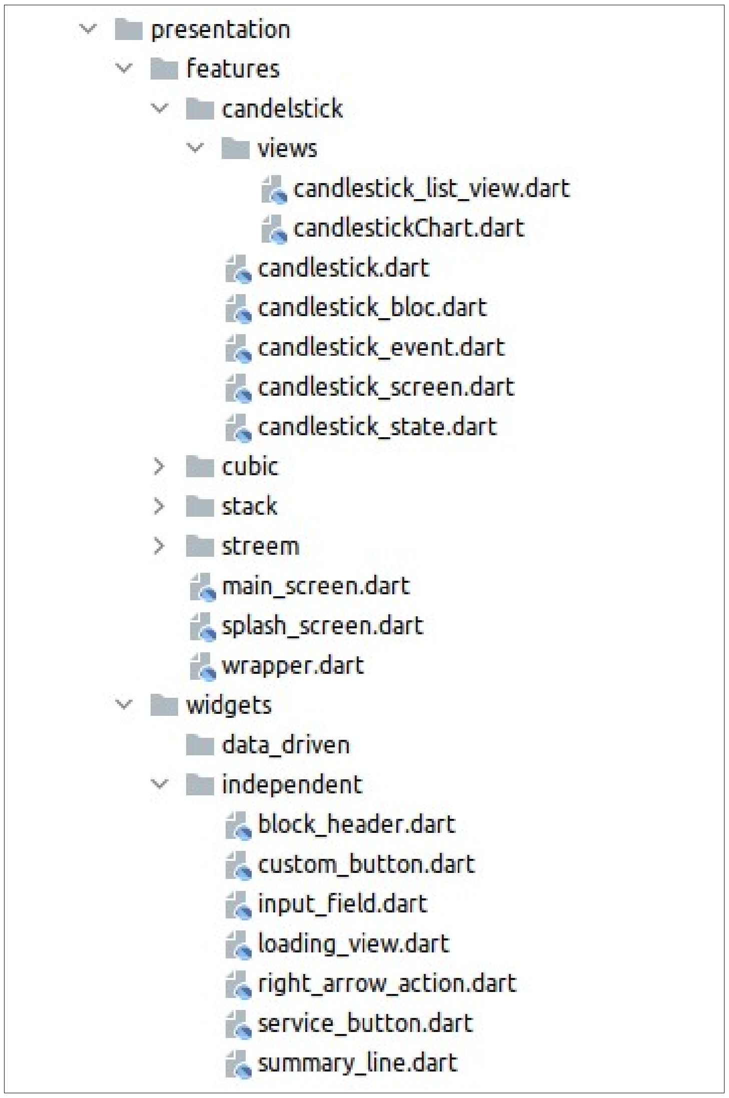
```
repositories
  abstract
    candlestick_list_repository.dart
    cubic_list_repository.dart
    realtime_data_service.dart
    stack_list_repository.dart
  candlestick_list_repository_impl.dart
  cubic_list_repository_impl.dart
  realtime_data_service_impl.dart
  stack_list_repository_impl.dart
```



```
  testApp
    models
      candlestick_list_model.dart
      cubic_list_model.dart
      stack_list_model.dart
    repositories
      candlestick_list_remote_repository.dart
      cubic_list_remote_repository.dart
      stack_list_remote_repository.dart
domain
  entities
    candlestick_list_data_entity.dart
    cubic_list_data_entity.dart
    entity.dart
    stack_list_data_entity.dart
    validator.dart
  usecases
    base_use_case.dart
    candlestick_list_usecase.dart
    cubic_list_usecase.dart
    stack_list_usecase.dart
```

Model files are responsible for mapping the received Json data put each data in its place.

Remote repositories that are placed in testApp folder, play the key role in API calls and each of them is the heart of network request for the related network process.

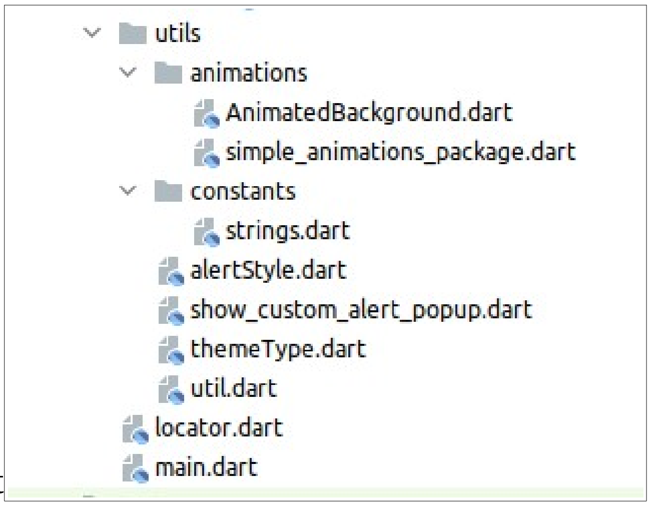


```
presentation
  features
    candelstick
      views
        candlestick_list_view.dart
        candlestickChart.dart
      candlestick.dart
      candlestick_bloc.dart
      candlestick_event.dart
      candlestick_screen.dart
      candlestick_state.dart
    cubic
    stack
    streem
    main_screen.dart
    splash_screen.dart
    wrapper.dart
  widgets
    data_driven
    independent
      block_header.dart
      custom_button.dart
      input_field.dart
      loading_view.dart
      right_arrow_action.dart
      service_button.dart
      summary_line.dart
```

The main parts of UI are placed in Presentation folder that includes views and BloC related processes and states management.
Each screen of this project has a folder in presentation and affected regularly by the lifecycle that is generated by BloC features.

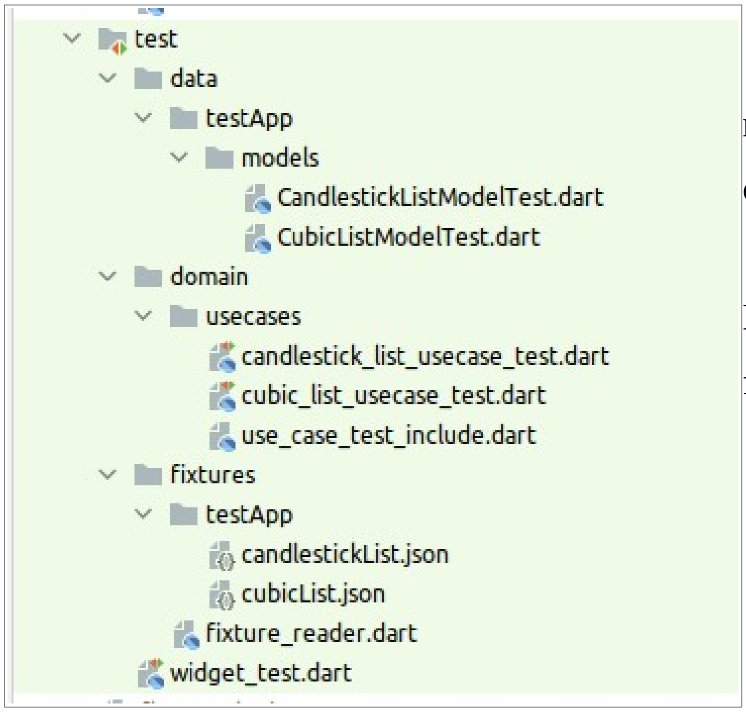
MainScreen contains bottomNavigator and the user can access to all screens by using of it.
Widgets folder contains the data driven and independent widgets that are used in different parts of the project.

Utils includes animations, Strings and generally constants which are using during scaffolds, widgets and functions through the project.



```
utils
  animations
    AnimatedBackground.dart
    simple_animations_package.dart
  constants
    strings.dart
  alertStyle.dart
  show_custom_alert_popup.dart
  themeType.dart
  util.dart
locator.dart
main.dart
```

I’ve used GetIt as a service locator.
I’m using it because as our App grows, we will need to put the app's logic in classes that are separated from Widgets. Keeping our widgets from having direct dependencies makes the code better organized and easier to test and maintain.



```
test
  data
    testApp
      models
        CandlestickListModelTest.dart
        CubicListModelTest.dart
  domain
    usecases
      candlestick_list_usecase_test.dart
      cubic_list_usecase_test.dart
      use_case_test_include.dart
  fixtures
    testApp
      candlestickList.json
      cubicList.json
    fixture_reader.dart
  widget_test.dart
```

And finally I wrote Unit Tests on models and usecases which are used in candlestick and cubic charts.
To do this I employed Flutter_test, Mockito and some mocked data as the fixture.