State Management

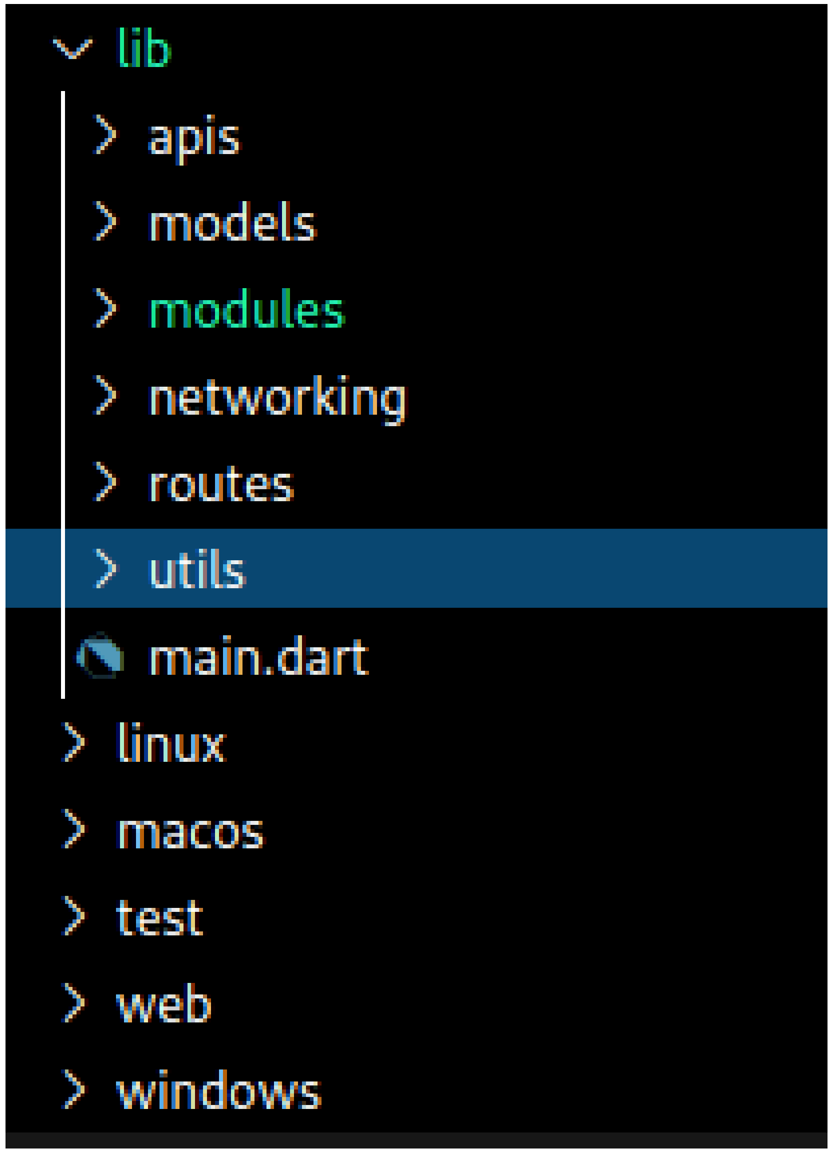
What is State Management?

- State management is a crucial aspect of Flutter development, as it involves handling and updating the state of your application.
- Flutter provides various ways to manage state, and the choice depends on the complexity of your app and your specific requirements.

GetX State Management

- GetX is a popular state management package in Flutter that provides a simple and powerful solution for managing the state of your application.
- It combines state management, dependency injection, and route management.
- To use GetX in your Flutter project, install using "flutter get" CMD.

File Structure Example



```
v lib
  > apis
  > models
  > modules
  > networking
  > routes
  > utils
    main.dart
> linux
> macos
> test
> web
> windows
```

Key points

- Controllers
- Routes
- Screens
- Bindings
- Feel free to adapt this structure to fit the specific needs and complexity of your Flutter project. GetX provides flexibility, and you can organize your code based on your preferences and project requirements.

Asynchronous Functions

Introduction

- In Flutter, asynchronous programming is essential for handling time-consuming tasks, such as network requests, file I/O, and other operations that may cause the application to freeze if executed synchronously.
- Dart, the programming language used in Flutter, supports asynchronous programming through the use of the async and await keywords.
- These keywords enable developers to write asynchronous code in a more readable and structured manner.

Asynchronous Functions

- Future and async/await:
  - Dart uses the Future class to represent asynchronous operations.
  - An asynchronous function is declared using the async keyword, and the await keyword is used inside the function to wait for the completion of a Future.
- Using async with Widgets:
  - Asynchronous functions are often used in Flutter widgets to perform tasks such as fetching data before rendering the UI.
  - Use the FutureBuilder widget to handle asynchronous operations within the widget tree.

Stream-based Asynchronous Programming

- Stream and StreamController:
  - Dart provides the Stream class for handling sequences of asynchronous events.
  - Combine it with StreamController to create and manipulate streams.

Conclusion

- Flutter's support for asynchronous programming through Dart's async and await syntax allows developers to write efficient and responsive applications.
- Whether you're fetching data from a server, handling user input, or performing any other asynchronous task, understanding and using asynchronous functions is crucial for building robust and responsive Flutter applications.

API Integration with Dio

- Dio is a powerful HTTP client for Dart that simplifies the process of making HTTP requests in Flutter.
- It supports various features such as interceptors, cancellation, FormData, file downloading/uploading, and more.

GetxController

Introduction

- In Flutter, GetX provides a convenient and powerful state management solution, and the GetxController is a fundamental part of it.
- A GetxController is essentially a class that manages the logic and state of a widget or a set of widgets.
- It helps in separating the business logic from the UI layer, making the code more organized and maintainable.

Extends GetxController

- A class becomes a GetxController by extending the GetxController class.
- This gives the class access to all the functionalities provided by GetX for state management.

Lifecycle Management

- GetxController has built-in lifecycle management, meaning it automatically disposes of resources when they are no longer needed.
- It helps prevent memory leaks by clearing resources when the associated widget is removed from the widget tree.

State Management

- The primary role of a GetxController is to manage the state of your application.
- You can define observable variables using Rx types provided by GetX. Changes to these variables automatically trigger updates in the UI that depends on them.

GetX bindings

- In GetX, bindings are used to inject dependencies or initialize resources before a page is created.
- Bindings provide a way to separate the creation of controllers or services from the UI, making the code more modular and maintainable.
- Bindings are associated with specific pages or routes and are executed before the widget tree is built.

GetX routing

- In GetX, routing is an integral part of managing navigation between different screens or pages in a Flutter application.
- GetX provides a simple and powerful routing system that makes it easy to navigate between pages and pass data between them.

Form Widget

- In Flutter, the Form widget is used to create and manage a form, which is a collection of form fields.
- Form fields are used to collect user input, such as text, numbers, and selections.
- The Form widget helps in managing the state of the form and validating user input.

Model class

- In Flutter, a model class is a Dart class that represents a data structure or entity in your application.
- Model classes are commonly used to structure and organize data retrieved from APIs, databases, or other sources.
- They encapsulate the properties and behavior related to a specific entity, making it easier to work with and maintain the data.