GetX Documentation

Introduction

Overview of GetX

GetX is a lightweight and powerful Flutter package that simplifies state management, dependency injection, and navigation in Flutter applications. It provides a simple yet robust architecture for building scalable and maintainable Flutter apps with minimal boilerplate code.

GetX has 3 basic principles.

- ✓ PRODUCTIVITY
  GetX এর এই নীতি মূলত অর্থ হল কাজের প্রক্রিয়াগুলি সহজ ও দ্রুত করা। এটি প্রোগ্রামারদের কাজকে সহায়ক করে এবং তাদের সমস্যা সমাধানে সাহায্য করে।
- ✓ PERFORMANCE
  এই নীতিতে মূলত অর্থ হল অ্যাপ্লিকেশনের দ্রুততা বাড়ানো। GetX পারফর্মেন্স বৃদ্ধি করে এবং অ্যাপ্লিকেশনের স্পীড বাড়ায়।
- ✓ ORGANIZATION
  এই নীতিতে গুরুত্ব হল প্রোগ্রামারদের কোড সংরক্ষণ এবং ব্যবস্থাপনা। GetX এপ্লিকেশন সংগঠিত রয়েছে এবং উন্নত ডকুমেন্টেশন এবং টেস্টিং সাহায্য করে।

Purpose of the Documentation

The purpose of this documentation is to provide a comprehensive guide to using GetX in your Flutter projects. It aims to help developers understand the core concepts of GetX, how to integrate it into their applications, and leverage its features to build efficient and feature-rich Flutter apps.

Installation Guide

1. Add Dependency:
   ```
   dependencies:
     get: ^4.3.8
   ```
2. Install Package:

```
flutter pub get
```

3. Import GetX:

```
import 'package:get/get.dart';
```

4. Initialize GetX:

```
void main() {
  runApp(MyApp());
}
```

Core Concepts

State Management

1. Reactive State Management

GetX offers a reactive state management approach, allowing you to easily react to changes in your application's state. This is achieved through observables, which are streams that emit values whenever the state they represent changes.

- ✓ Observables: Streams that emit values when state changes.
- ✓ Reactive Widgets: Widgets that automatically update in response to state changes.
- ✓ Listening to Observables: Updating UI based on observable changes.

2. Controller Management

In GetX, controllers are used to manage the state of specific parts of your application. Controllers encapsulate the business logic and state of a particular feature or screen, making it easier to organize and maintain your codebase.

- ✓ Creating Controllers: Manage state and logic for specific parts of the app.
- ✓ Dependency Injection: Inject controllers into widgets for efficient state management.
- ✓ Controller Lifecycle: Initialize, dispose, and reuse controllers effectively.

3. Global State Management

GetX provides mechanisms for managing global state across your entire application. This allows you to share state between different parts of your UI and ensure consistency across the entire user experience.

- ✓ Global Controllers: Manage shared state across the app.
- ✓ Global Bindings: Initialize and manage global controllers.
- ✓ State Persistence: Persist global state across app launches

4. State Management Best Practices

To ensure that your application remains maintainable and performant, it's essential to follow best practices when managing state with GetX. These best practices help you write clean, efficient, and scalable code that is easier to debug and maintain.

- ✓ Separation of Concerns: Keep UI and business logic separate.
- ✓ Immutable State: Use immutable state for reliability.
- ✓ Minimizing Rebuilds: Reduce unnecessary UI updates.
- ✓ Testing State Management: Ensure correctness and reliability with tests.

Dependency Injection

Dependency Injection (DI) is a design pattern commonly used in software development to facilitate the management of dependencies between various components of an application. GetX provides a powerful DI mechanism that allows for easy management and injection of dependencies within your application.

1. Registering Dependencies

To use dependency injection with GetX, you first need to register your dependencies. This can be done using the **Get.put()** method, where you specify the type of dependency and its corresponding instance or factory.

```
import 'package:get/get.dart';

class MyDependency {}

void main() {

  Get.put(MyDependency()); // Registering a singleton instance

}
```

2. Retrieving Dependencies

Once your dependencies are registered, you can retrieve them anywhere within your application using the **Get.find()** method.

```
import 'package:get/get.dart';

class MyController extends GetxController {

  final MyDependency dependency = Get.find();

}
```

3. Dependency Injection with Lazy Loading

GetX also supports lazy loading of dependencies, where the dependency is instantiated only when it is first accessed.

```
import 'package:get/get.dart';

class MyController extends GetxController {

  MyDependency? _dependency;


  MyDependency get dependency {

    _dependency ??= Get.find();

    return _dependency!;

  }

}
```

Routing and Navigation

Routing and navigation are essential aspects of any application, allowing users to navigate between different screens or pages seamlessly. In GetX, routing and navigation are handled efficiently through the **GetMaterialApp** widget and the **Get.to()** method.

Basic Navigation

Using Get.to ()

The simplest way to navigate to a new screen is by using the Get.to() method. Here's how you can use it:

```
// Navigate to a new screen

Get.to(NextScreen());
```

উপরের কোডে, যখন ব্যবহারকারী বাটনে ক্লিক করবেন, তখন আমরা **Get.to()** ফাংশন ব্যবহার করে দ্বিতীয় পেজে নেভিগেট করব।

Using Get. toNamed ()

```
Get.toNamed('/details');
```

উপরের কোডে, যখন ব্যবহারকারী বাটনে ক্লিক করবেন, তখন আমরা `Get.toNamed()` ব্যবহার করে নাম `/` details'দিয়ে সেকেন্ড পেজে নেভিগেট করব। এটি নামের সাথে সম্পর্কিত নেভিগেশন নেম সেট করা যেতে পারে পেজের জন্য যেখানে আপনি নেভিগেট করতে চান।

Using Get. back ()

```
Get.back();
```

Get.back() ফাংশনটি GetX ফ্রেমওয়ার্কে ব্যবহার করা হয় যেন ব্যবহারকারী একটি পেজ থেকে পূর্ববর্তী পেজে ফিরতে পারেন।

Using Get. off ()

```
Get.off(NextScreen());
```

এটি Get.off() ফাংশন ব্যবহার করে আমরা সহজেই বর্তমান পেজ থেকে প্রিভিউস পেজে ফিরে যেতে পারি। এটি খুব সহজ এবং দ্রুত প্রিভিউস পেজে ফিরে যেতে সাহায্য করে।

Using Get. offAll ()

```
Get.offAll(NextScreen());
```

সমস্ত পূর্বে ওপেন করা স্ক্রীন বন্ধ করে নতুন স্ক্রীন প্রদর্শন করে। এটি অনেকটা অ্যাপ্লিকেশনে লগআউট অথবা হোম পেজে ফিরে যাওয়ার জন্য ব্যবহৃত হয়।

Passing Arguments

To send data to the next screen:

```
Get.to(NextScreen(), arguments: {'key': 'value'});
```

স্থানান্তর করা যেখানে আমাদের একটি পেজ থেকে অন্য পেজে ডেটা পাঠানোর প্রয়োজন হয়, GetX এই সমস্যা সহজ করে তুলে ধরে। এটি সহজেই ডেটা প্রেরণ করতে এবং পেজের নির্দিষ্ট কার্যকলাপগুলিতে ব্যবহার করতে পারে।

Returning Data

To receive data from the next screen:

```
final result = await Get.to(NextScreen());
```

Named Routes

GetX supports named routes for better organization and readability of your code. Here's how you can define named routes:

```
GetMaterialApp(
  initialRoute: '/',
  getPages: [
    GetPage(name: '/', page: () => FirstScreen()),
    GetPage(name: '/second', page: () => SecondScreen()),
  ],
);
```

নেমড রুটগুলি ব্যবহার করে রুট ম্যানেজমেন্ট করা সহজ এবং সুবিধাজনক হতে পারে। এটি অ্যাপ্লিকেশনের পৃষ্ঠাগুলির নাম অনুসারে নেভিগেশনের সুবিধা সরবরাহ করে। নেমড

রুটগুলি ব্যবহার করলে, আপনার পৃষ্ঠাগুলির নামগুলির পরিবর্তে রুট এবং এই প্রকারে কোনও পরিবর্তন করতে হবে না।

Bottom Navigation

```
bottomNavigationBar: Obx(() => BottomNavigationBar(
  currentIndex: controller.selectedIndex.value,
  onTap: (index) => controller.changePage(index),
  items: [
    BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
    BottomNavigationBarItem(icon: Icon(Icons.settings), label: 'Settings'),
  ],
)),
```

Dialogs and Snack bars

Dialogs and snack bars are crucial components in user interfaces for displaying important information, notifications, alerts, and collecting user input. GetX provides easy-to-use methods for showing dialogs and snack bars in your application.

Dialogs

Dialogs are modal UI components that overlay the main application content to display critical information or prompt the user for input. GetX offers a simple way to create and show dialogs with minimal code.

Creating Dialogs

To create a dialog in GetX, you can use the Get.dialog() method and pass a Dialog widget as its argument. Here's a basic example:

```
Get.dialog(
  AlertDialog(
    title: Text('Dialog Title'),
    content: Text('Dialog content goes here.'),
    actions: [
      TextButton(
        onPressed: () => Get.back(),
        child: Text('Close'),
      ),
    ],
  ),
);
```

Snack bars

Snackbars are lightweight notifications that appear at the bottom of the screen to provide feedback to the user. They are commonly used to display brief messages or alerts.

Showing Snackbars

```
Get.snackbar(
  'Snackbar Title',
  'Snackbar message goes here.',
  snackPosition: SnackPosition.BOTTOM,
);
```

Bottom sheets

GetX provides easy-to-use Bottom sheets for presenting supplementary content or actions in your Flutter app.

Types

- ✓ Persistent: Remains visible during app interaction.
- ✓ Modal: Requires interaction before returning to app content.

Usage

1. Controllers

- ✓ Creating Controllers: Define controllers to manage the state and logic of your application.
- ✓ Accessing Controllers: Use Get.put() to instantiate controllers and access them throughout your app.
- ✓ State Management: Use controllers to store and update the state of your app. Access state variables using controller.variableName.
- ✓ Injecting Dependencies: Inject dependencies into controllers using Get.put() or Get.lazyPut().

2. Observables

- ✓ Defining Observables: Create observables using Rx classes like RxInt, RxString, etc.
- ✓ Using Reactive Programming: Subscribe to observable changes using .obs or .value.
- ✓ Listening to Changes: Use .obs or .value to update UI components automatically when data changes.
- ✓ Disposing Observables: Call .dispose() to dispose of observables when they are no longer needed.

3. Models

- ✓ Defining Models: Create plain Dart classes to represent data structures.
- ✓ Mapping Data to Models: Map data received from APIs or other sources to model objects.
- ✓ Validation and Serialization: Implement validation and serialization methods within model classes as needed.

4. Services

- ✓ Creating Services: Define services to encapsulate business logic and interact with external resources.
- ✓ Dependency Injection: Inject services into controllers using Get.put() or Get.lazyPut().
- ✓ API Integration: Integrate services with external APIs using HTTP packages like http or dio.

5. Bindings

- ✓ Binding Data to UI: Use Obx() or .obs to bind data from controllers or observables to UI elements.
- ✓ Event Handling: Bind UI events to controller methods using onPressed: () => controller.methodName().
- ✓ Two-Way Binding: Implement two-way binding using controllers and input fields.