Chapter 5: Dialogs

Standard Dialogs

Use the `DialogService` for standard dialogs.

```dart
// locator.dart
import 'package:stacked_services/stacked_services.dart';

final locator = GetIt.instance;

void setupLocator() {
  locator.registerLazySingleton(() => DialogService());
}
```

```dart
// home_viewmodel.dart
import 'package:stacked_services/stacked_services.dart';
import 'locator.dart';

class HomeViewModel extends BaseViewModel {
  final DialogService _dialogService = locator<DialogService>();

  void showDialog() {
```

```
    _dialogService.showDialog(
      title: 'Dialog Title',
      description: 'This is the dialog description',
    );
  }
}
```

Custom Dialogs

You can create custom dialogs by extending `CustomDialogBuilder`.

```dart
// custom_dialog.dart
import 'package:flutter/material.dart';
import 'package:stacked_services/stacked_services.dart';

class CustomDialog extends StatelessWidget {
  final DialogRequest request;
  final Function(DialogResponse) completer;

  CustomDialog({required this.request, required this.completer});

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
```

```
      title: Text(request.title!),
      content: Text(request.description!),
      actions: <Widget>[
        TextButton(
          onPressed: () => completer(DialogResponse(confirmed: true)),
          child: Text('Confirm'),
        ),
        TextButton(
          onPressed: () => completer(DialogResponse(confirmed: false)),
          child: Text('Cancel'),
        ),
      ],
    );
  }
}
```

```dart
// locator.dart
import 'package:stacked_services/stacked_services.dart';
import 'custom_dialog.dart';

final locator = GetIt.instance;

void setupLocator() {
```

```
  locator.registerLazySingleton(() => DialogService());

  locator<DialogService>().registerCustomDialogBuilders({

    'custom': (context, dialogRequest, completer) =>

        CustomDialog(request: dialogRequest, completer: completer),

  });

}
```

Chapter 6: Stacked CLI

Installation and Setup

Install the Stacked CLI globally using pub:

```sh
dart pub global activate stacked_cli
```

Ensure the installation path is added to your system?s PATH environment variable.

Generating Files

Generate a new view with the CLI:

```sh
stacked create view home
```

CLI Commands Overview

- `create view [name]`: Creates a new view and its corresponding ViewModel.
- `create service [name]`: Generates a new service.
- `create dialog [name]`: Creates a new custom dialog.

Chapter 7: Building Real-world Applications

Notes App

Requirements

- Create, read, update, and delete notes
- Use local storage for persistence
- Implement state management with Stacked

Implementation

- Set up the project structure with `stacked create view home`.
- Create services for handling data operations.
- Use `ViewModelBuilder` for UI and ViewModel interaction.

Example Code

```dart
// note_service.dart
import 'package:stacked/stacked.dart';

class NoteService with ReactiveServiceMixin {
  // Implementation of CRUD operations
}
```

To-Do App

Requirements

- Create, read, update, and delete tasks
- Implement task completion functionality
- Use an API for backend operations

Implementation

- Set up views and viewmodels with the Stacked CLI.
- Implement API services for data fetching.
- Use reactive services for state management.

Example Code

```dart
// task_service.dart
import 'package:stacked/stacked.dart';

class TaskService with ReactiveServiceMixin {
  // Implementation of CRUD operations
}
```

Chapter 8: Advanced Topics

API Services

Implementing API services involves creating a service class and using HTTP packages like `http` or `dio`.

Example Code

```dart
// api_service.dart
import 'package:dio/dio.dart';

class ApiService {
  final Dio _dio = Dio();

  Future<Response> getData(String endpoint) {
    return _dio.get(endpoint);
  }
}
```

Dependency Injection

Use `get_it` for dependency injection to manage service instances.

Example Code

```dart
```

```
// locator.dart

import 'package:get_it/get_it.dart';


final locator = GetIt.instance;


void setupLocator() {
  locator.registerLazySingleton(() => ApiService());
}
```

State Management

Use reactive services and `ViewModelBuilder` to manage and react to state changes.

Example Code

```dart
// home_viewmodel.dart

import 'package:stacked/stacked.dart';


class HomeViewModel extends BaseViewModel {
  // State management logic
}
```

Testing

Write unit and widget tests to ensure the reliability of your application.

Example Code

```dart
// home_viewmodel_test.dart
import 'package:flutter_test/flutter_test.dart';
import 'package:myapp/viewmodels/home_viewmodel.dart';

void main() {
  test('HomeViewModel test', () {
    var viewModel = HomeViewModel();
    expect(viewModel.someProperty, someValue);
  });
}
```

Chapter 9: Best Practices

Code Organization

- Follow a consistent directory structure.
- Separate UI, business logic, and services.

State Management Strategies

- Use reactive services for global state.
- Use ViewModels for local state.

Performance Optimization

- Avoid rebuilding widgets unnecessarily.
- Use efficient data structures and algorithms.

Mastering Stacked in Flutter: From Beginner to Expert

Chapter 10: Conclusion and Further Reading

Recap

- Reviewed the core concepts of Stacked.
- Built real-world applications.
- Covered advanced topics and best practices.

Resources for Continued Learning

- [Official Stacked Documentation](https://pub.dev/packages/stacked)
- [Flutter Documentation](https://flutter.dev/docs)
- [Stacked GitHub Repository](https://github.com/FilledStacks/stacked)