Packaged Architecture

Or, How to use the power of dart packages (and Melos) to structure your project

Flutter Bytes, Nov 2022

Michael Soliman, Flutter Developer (@_michaelsoli_)

Agenda

- Architecture Overview
- Domain-Centric Architecture
- App Folder Structure
- Packaging app layers
- Introduction to Melos

Architecture Overview

Architecture Overview

Architectural Patterns

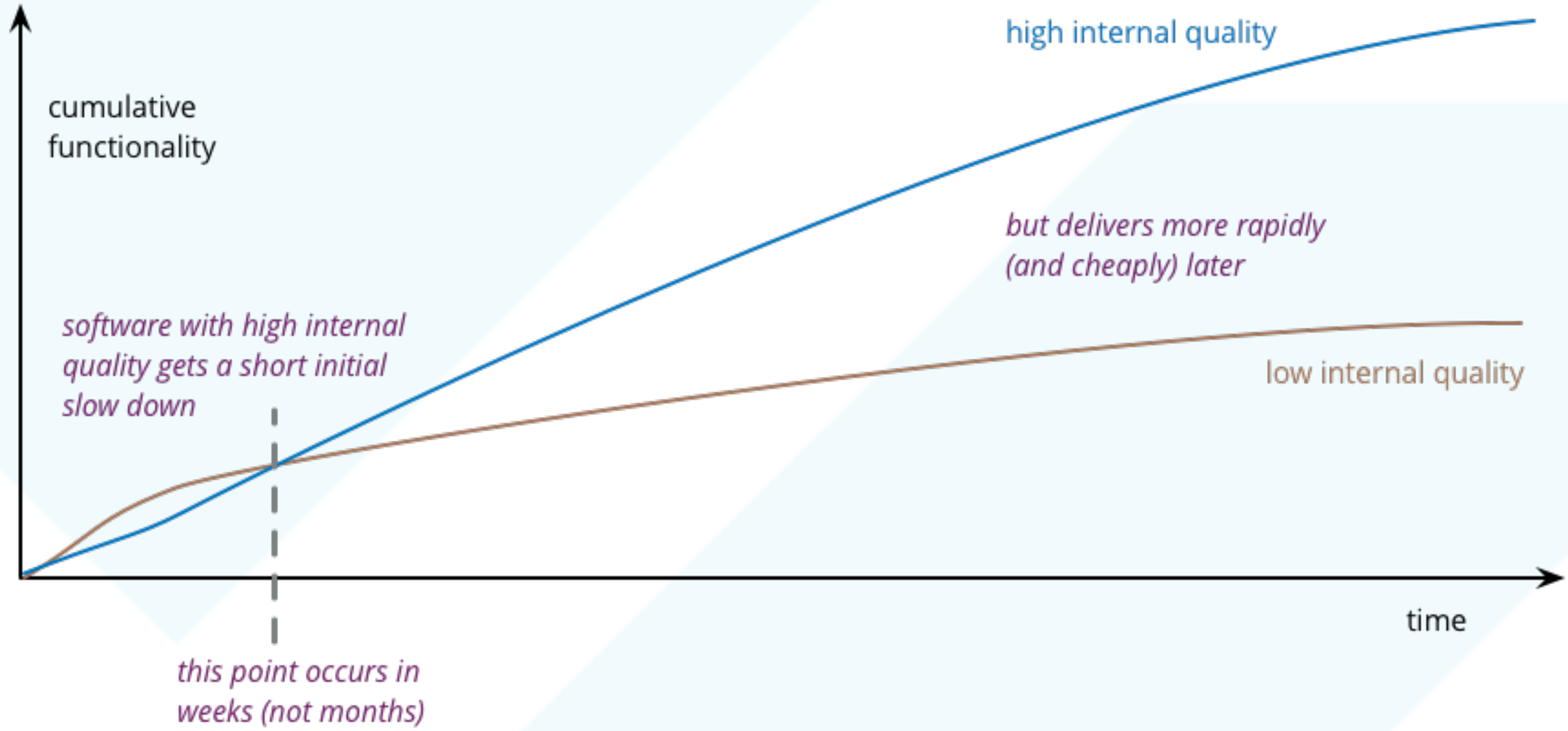
- Layered Architecture (n-tiers)
- Onion Architecture
- Clean Architecture
- DDD

Architecture Overview

Benefits

- Separation of Concerns
- Maintainability and Scalability
- Quality Software

Architecture Overview



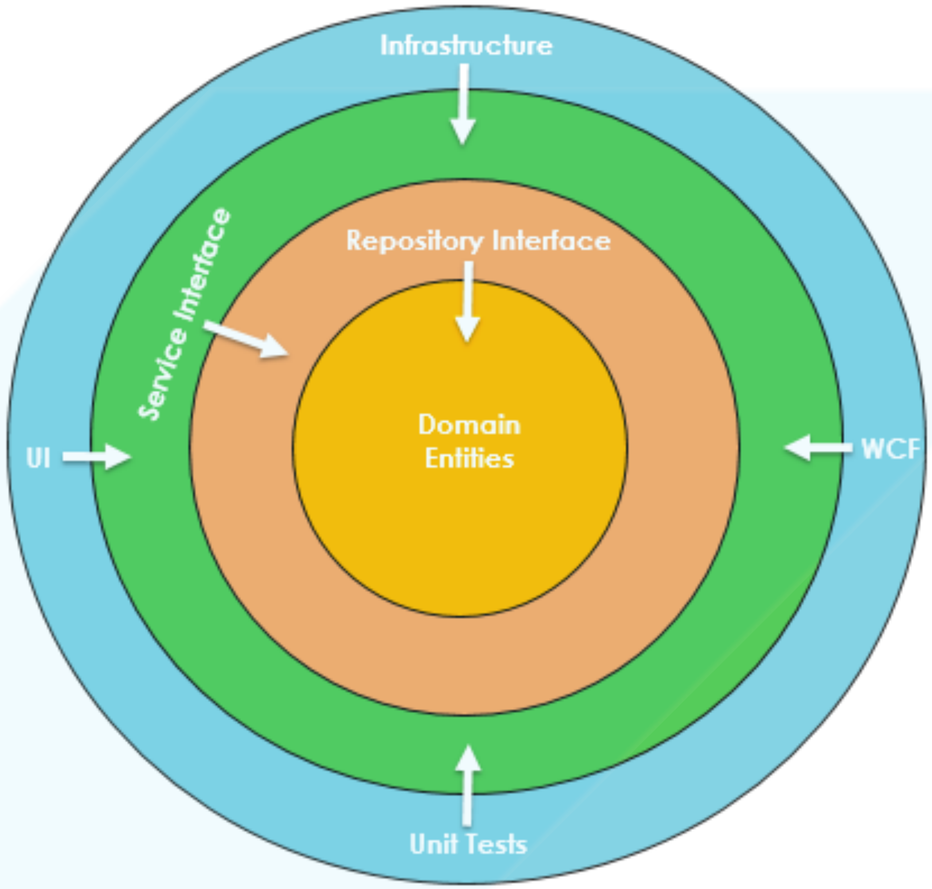
Fowler, M. (2019, May 29). Is High Quality Software Worth the Cost? *martinFowler*

Domain-Centric Architecture

Domain-Centric Architecture

Domain-Centric Architecture is any architecture that puts the domain layer centered without any dependencies on other layers. That makes our domain model

- Independent of Frameworks
- Independent of UI
- Independent of Data Sources
- Testable



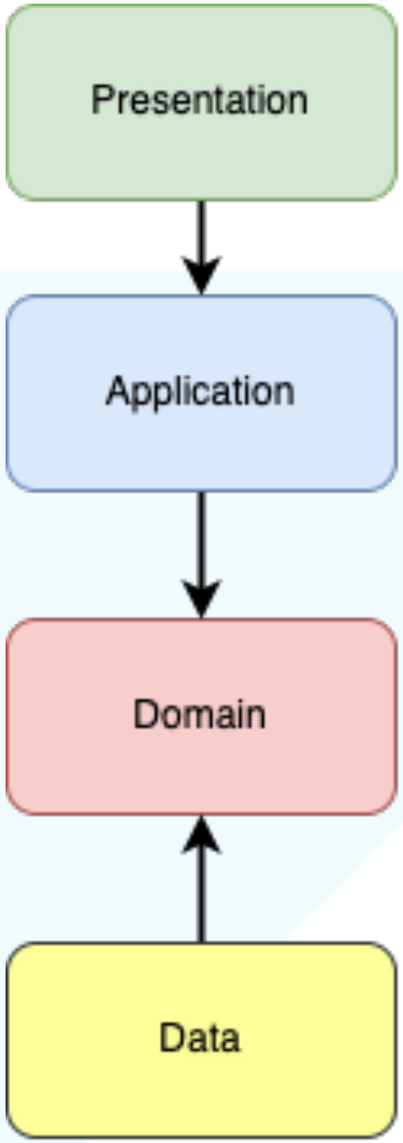
Onion Architecture

Domain-Centric Architecture

In a Flutter app

Layers (Simplified)

- Presentation Layer (Screens, Widgets)
- Application Layer (Providers/BLoCs, States)
- Domain Layer (Entities, Repositories)
- Data Layer (Models, Repositories, Data Sources)



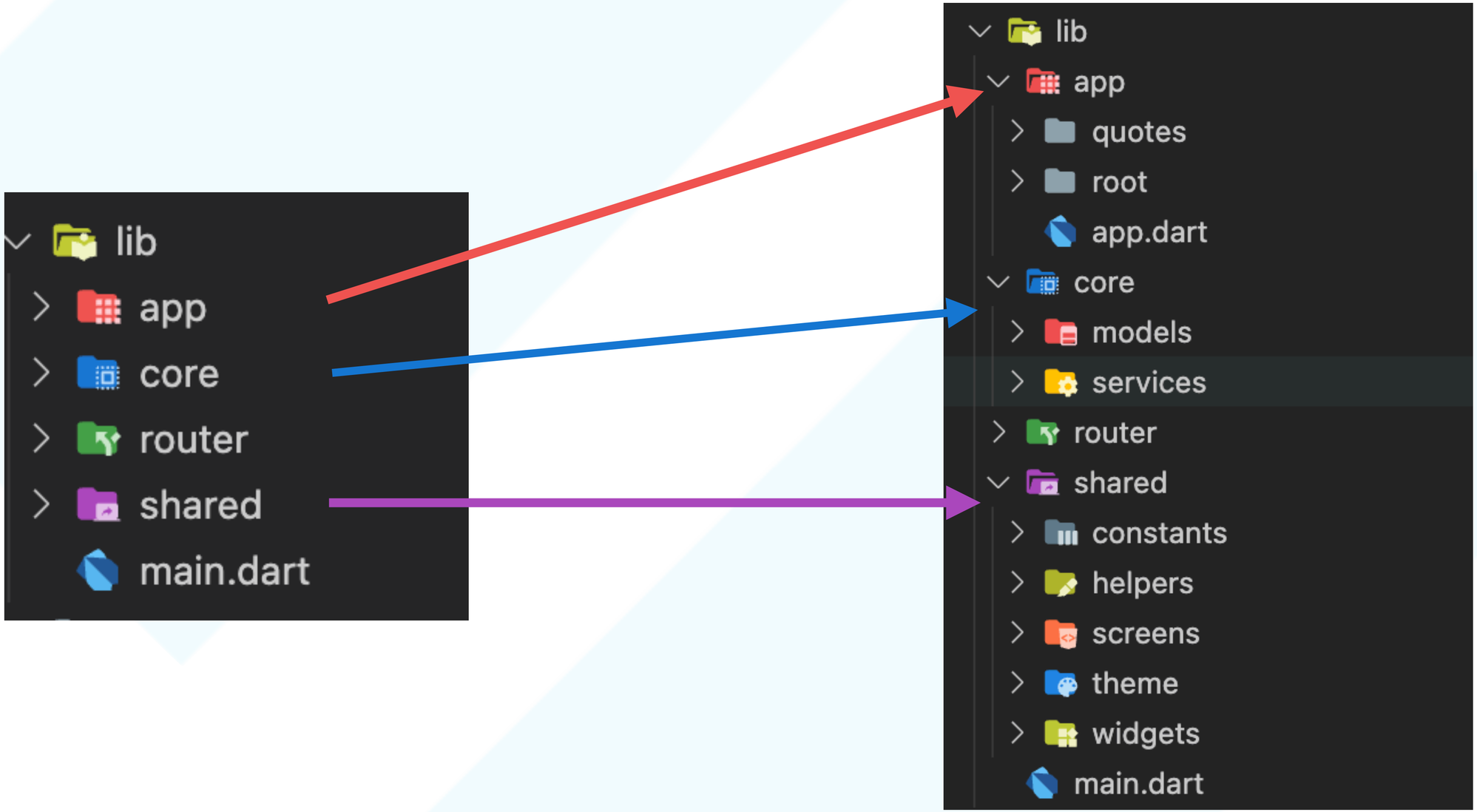
Domain-Centric Architecture

The Dependency Role:

- Source Code Dependencies can only point inwards

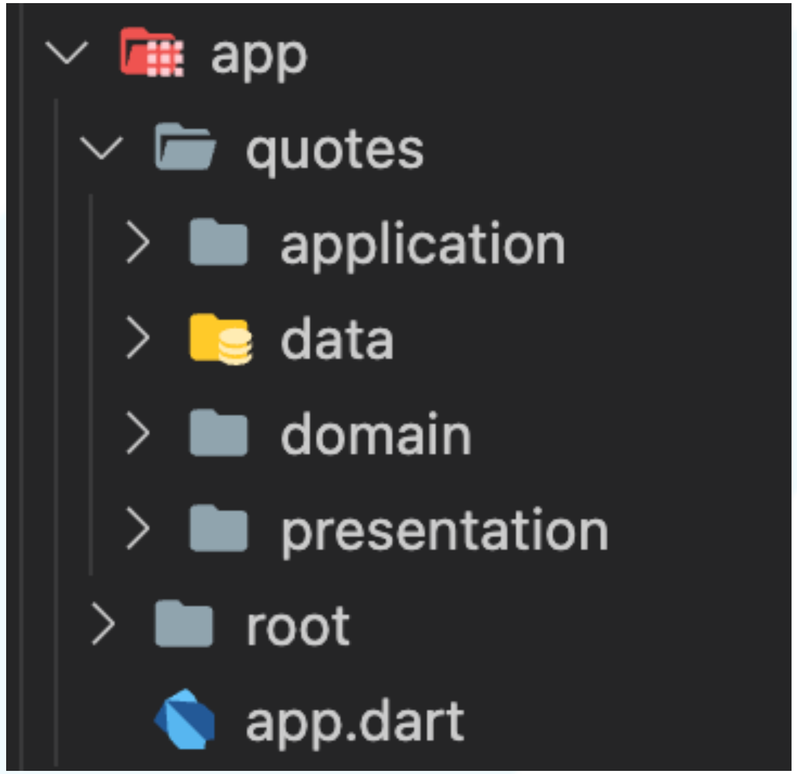
App Folder Structure

App Folder Structure (original)



App Folder Structure (original)

- app dir (Feature-First Approach)



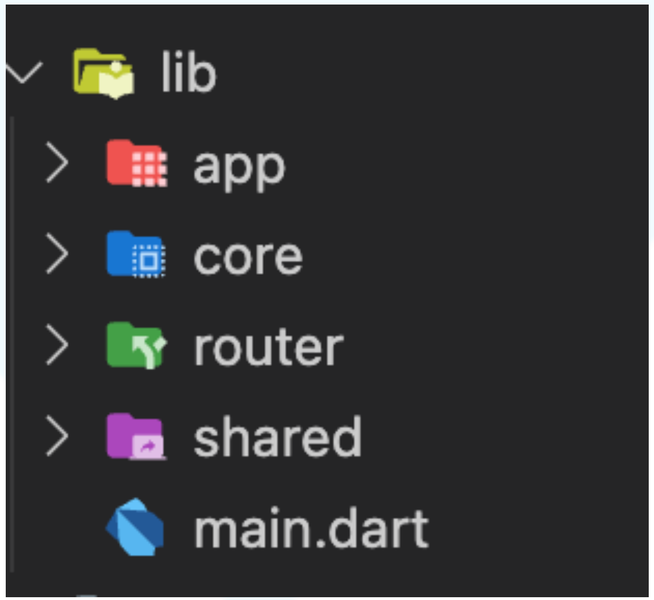
App Folder Structure (original)

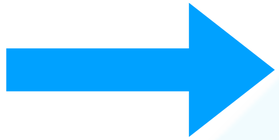
- pubspec.yaml (dependencies)
- will be a lot more In a normal-size flutter app

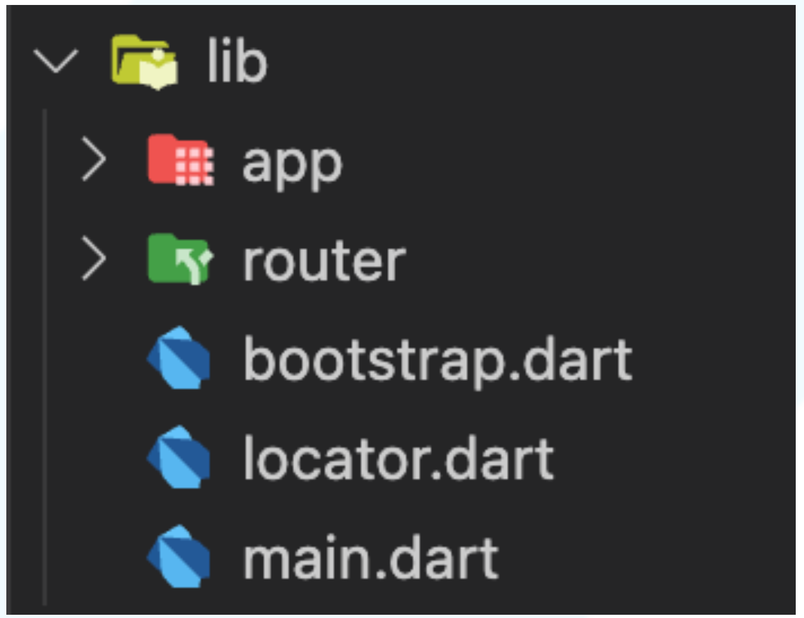
```yaml
dependencies:
  cupertino_icons: ^1.0.2
  flutter:
    sdk: flutter
  flutter_riverpod: ^2.0.2
  freezed_annotation: ^2.2.0
  go_router: ^5.1.1
  http: ^0.13.3
  json_annotation: ^4.7.0
  shared_preferences: ^2.0.7

dev_dependencies:
  build_runner: ^2.3.2
  flutter_test:
    sdk: flutter
  freezed: ^2.2.1
  json_serializable: ^6.5.4
  very_good_analysis: ^3.1.0

flutter:
  uses-material-design: true

  assets:
    - assets/images/

  fonts:
    - family: Poppins
      fonts:
```

Packaging App Layers

App Folder Structure (refactored)





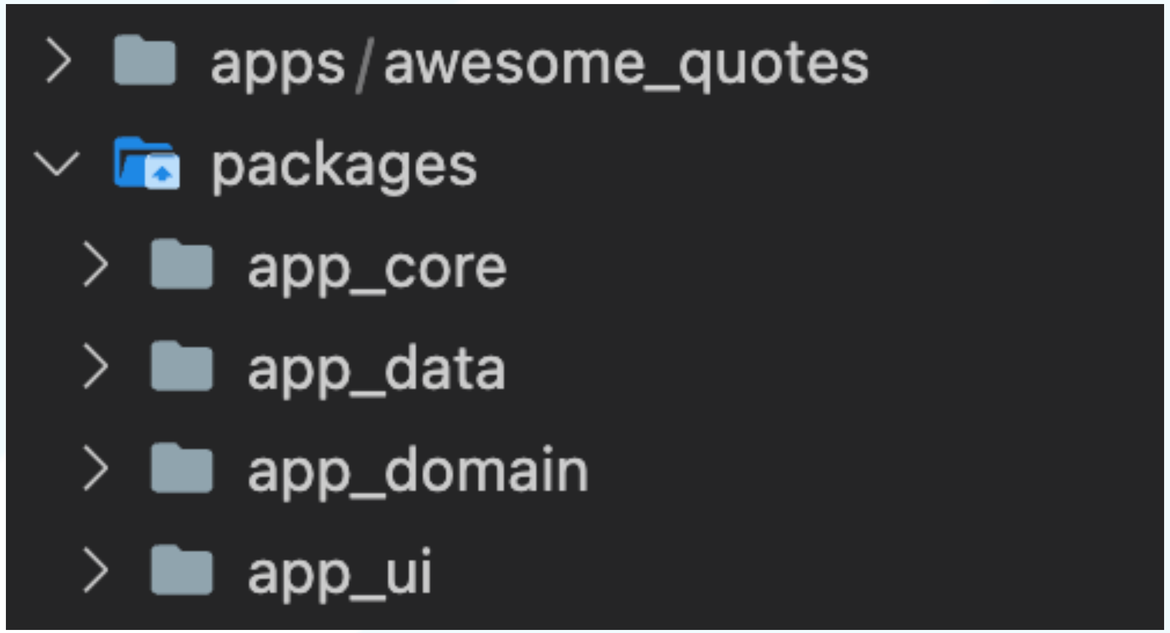


core and shared are moved to separate packages

App Folder Structure (refactored)

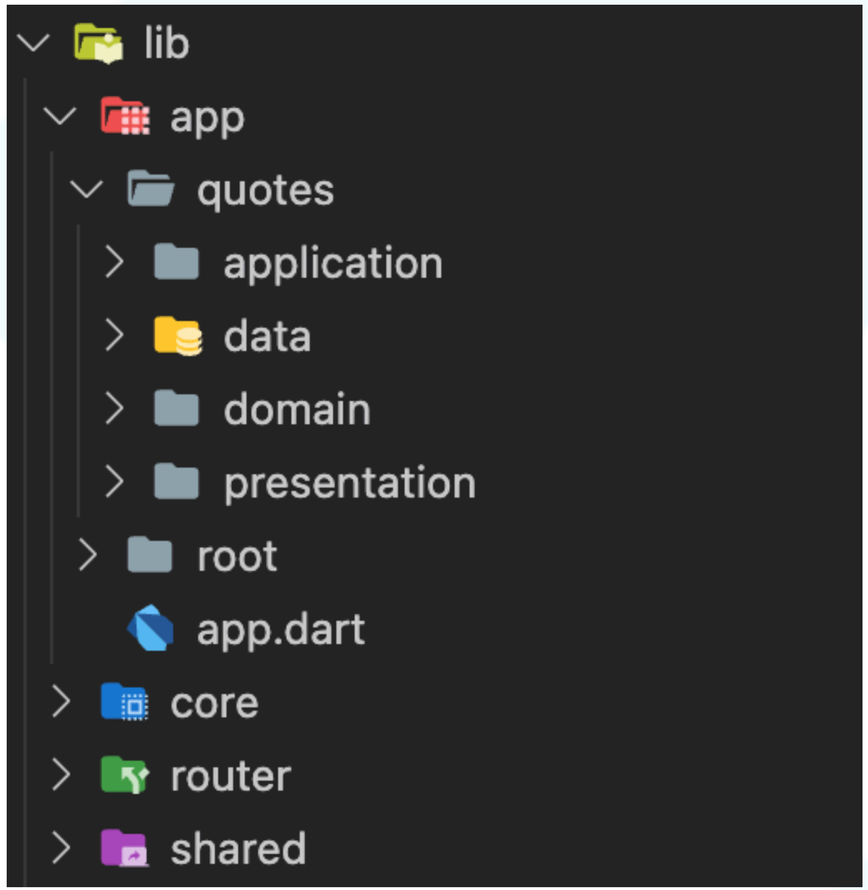
domain and data layers are also moved to separate packages

App Folder Structure (refactored)

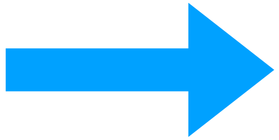


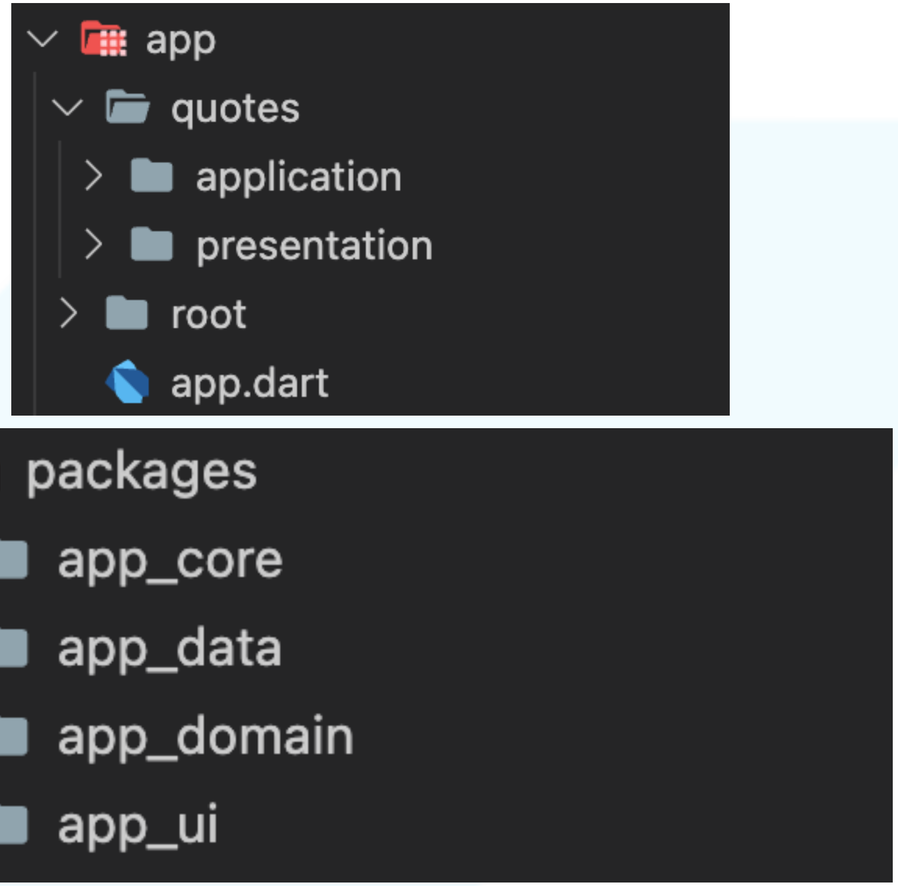
Overall Structure of our app

App Folder Structure (refactored)



Original





Refactored

App Folder Structure (refactored)

- pubspec.yaml (dependencies)
- Ideally, we won't need to add any other packages (except for required ones of course)
- Note that we also removed assets, as that's moved to our ui package

```yaml
dependencies:
  app_core:
    path: ../../packages/app_core
  app_data:
    path: ../../packages/app_data
  app_domain:
    path: ../../packages/app_domain
  app_ui:
    path: ../../packages/app_ui
  cupertino_icons: ^1.0.2
  flutter:
    sdk: flutter
  flutter_riverpod: ^2.0.2
  freezed_annotation: ^2.2.0
  go_router: ^5.1.1

dev_dependencies:
  build_runner: ^2.3.2
  flutter_test:
    sdk: flutter
  freezed: ^2.2.1
  very_good_analysis: ^3.1.0
```

Introduction to Melos

Introduction To Melos

- About
- Features
- Installation
- Setup
- Packages Graph

Introduction To Melos

About

- Melos is a CLI tool used to help manage Dart projects with multiple packages (also known as mono-repos).

Why do we need it?

- Simplest use case, Instead of running `flutter pub get` in every package, with Melos we just do it with a single command
- There are a lot more advanced use cases melos can do to help us manage our multi-package project!

Introduction To Melos

Features

- Automatic versioning & changelog generation.
- Automated publishing of packages to pub.dev.
- Local package linking and installation.
- Executing simultaneous commands across packages.
- Listing of local packages & their dependencies.

Introduction To Melos

Installation

- In any terminal window we run the following:

```
dart pub global activate melos
```

- There is also an IDE support for IntelliJ and VS Code, you can get more details here

Introduction To Melos

Setup

- First we create a melos.yaml file inside our root directory
- Within the melos.yaml, we add the following:
  - name: our project name
  - packages: the packages we want to add to our project
- And that's it! Now we need to bootstrap our project, we do that by running
melos run bootstrap

```yaml
name: quotes_app

packages:
  - apps/**
  - packages/**
```

Introduction To Melos

Packages Graph

- With the List command, we can generate a dependency graph for all of our packages
- And together with the VS Code Extension, you can preview your packages dependency graph with this command in vscode: Melos: show package graph
- remember The Dependency Role? We can use this graph to check if we're following it, as you can see, our domain package(layer) doesn't depend on any other package.

Thank YOU :)

References

- https://martinfowler.com/articles/is-quality-worth-cost.html
- https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- https://levelup.gitconnected.com/3-domain-centric-architectures-every-software-developer-should-know-a15727ada79f
- https://resocoder.com/flutter-clean-architecture-tdd/
- https://codewithandrea.com/articles/flutter-app-architecture-domain-model/
- https://github.com/mhadaily/flutter-architecture-ddd
- https://melos.invertase.dev/