FROM MODULES TO PACKAGES

AN ARCHITECTURAL APPROACH

A BIT OF HISTORY

- As an Android developer I was used to gradle modules
- A project consists of at least one gradle module
- Each module is completely independent from each other
- You implement a module in the gradle file of another one by:

```
dependencies {
  implementation(project(":library:core"))
}
```

Different approaches for modularization

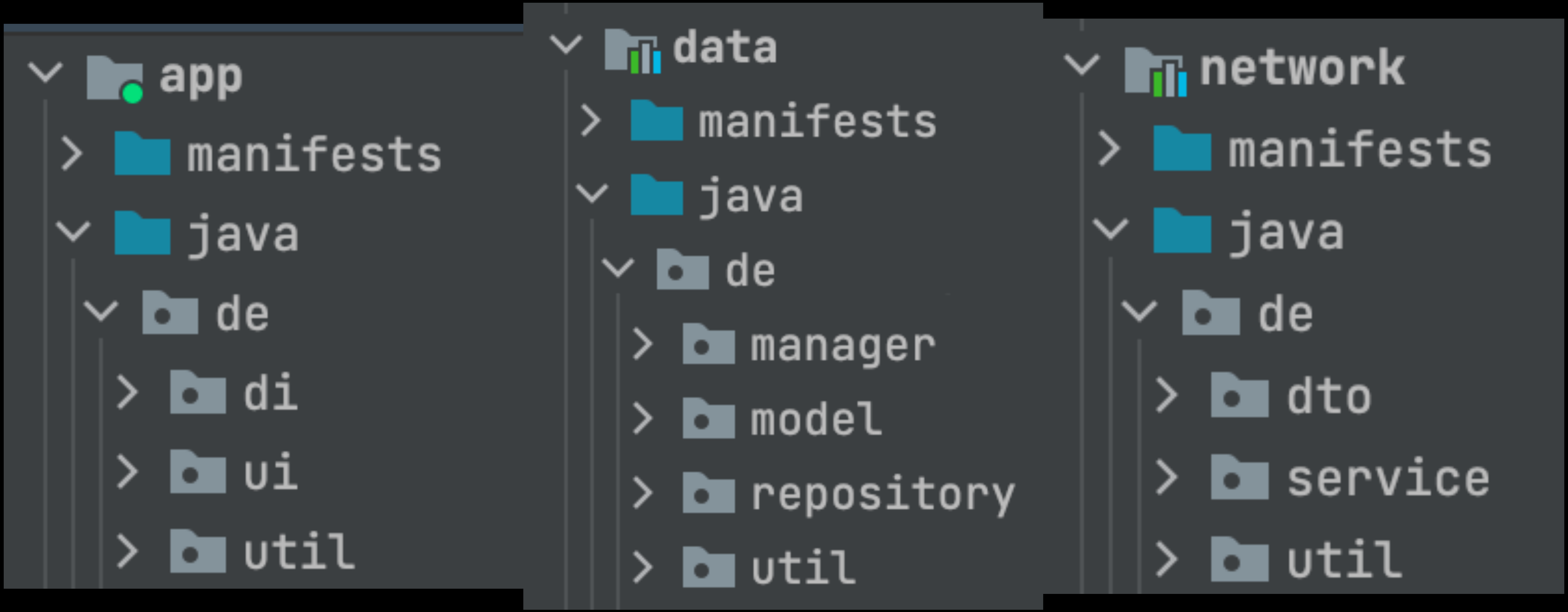
- Layered / multi-tier architecture
- Feature-based architecture

MODULES IN MULTI-TIER ARCHITECTURE

Back in 2018, I stated in a project's README

“[these tiers] are extracted to different gradle modules to ensure that you do always stick to the following "chain of knowledge": presentation --> data --> network”

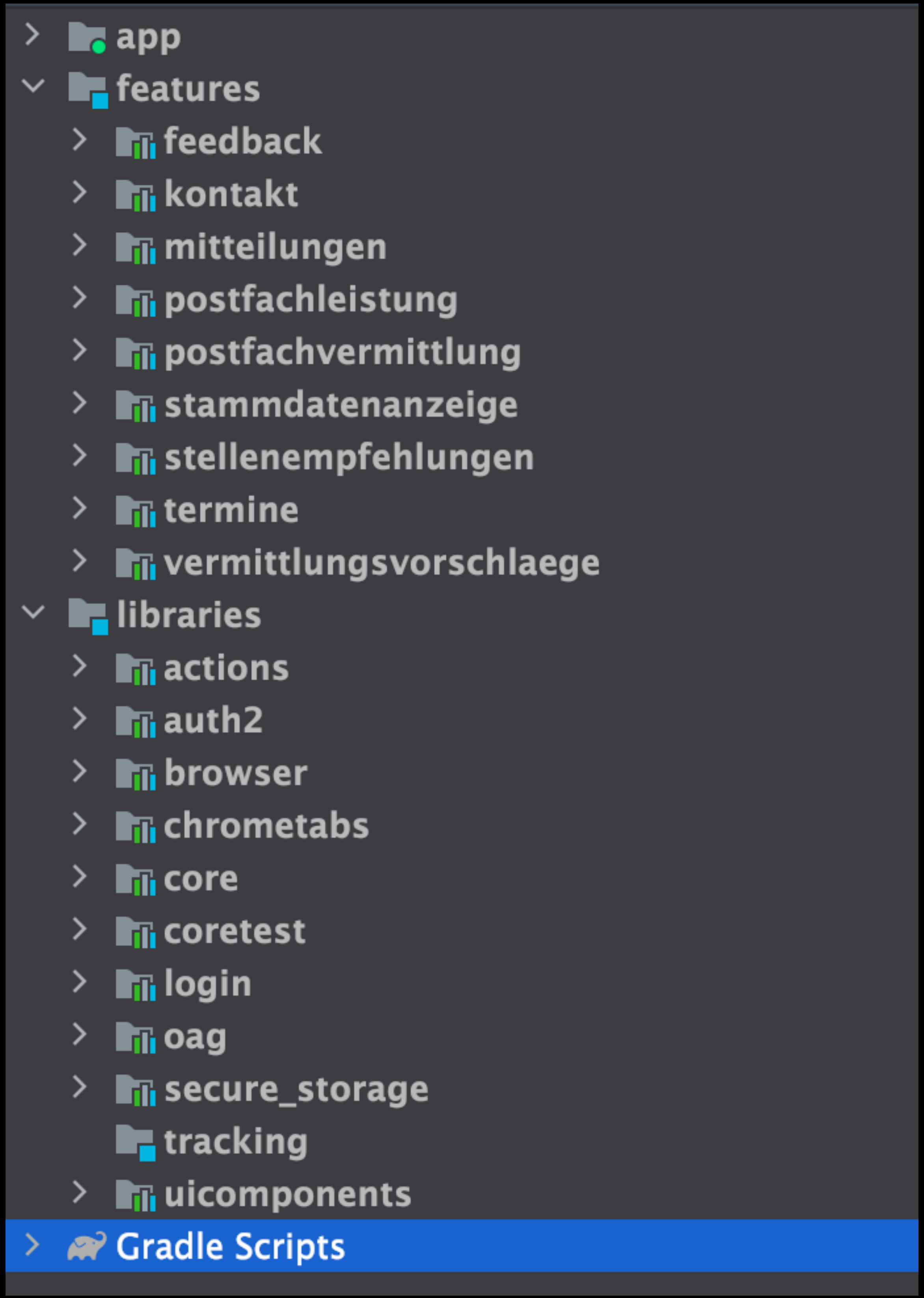
MODULES IN MULTI-TIER ARCHITECTURE



```
app
  manifests
  java
    de
      di
      ui
      util
```

```
data
  manifests
  java
    de
      manager
      model
      repository
      util
```

```
network
  manifests
  java
    de
      dto
      service
      util
```



```
> app
v features
    > feedback
    > kontakt
    > mitteilungen
    > postfachleistung
    > postfachvermittlung
    > stammdatenanzeige
    > stellenempfehlungen
    > termine
    > vermittlungsvorschlaege
v libraries
    > actions
    > auth2
    > browser
    > chrometabs
    > core
    > coretest
    > login
    > oag
    > secure_storage
      tracking
    > uicomponents
> Gradle Scripts
```

MODULES IN FEATURE ARCHITECTURE

In 2020, I built a new app from scratch meant for millions of users. It used a feature-based approach and was inspired by a blog post by Jeroen Mols [1].

[1] More about the architecture approach can be found in this blog by Jeroen Mols

EXAMPLE

```
           app
            |
      ______|______
     |      |      |
 feature 1  |    feature 2
  ___|______|______|___
 |          |          |
lib 1     lib 2      lib 3
```

THEORY
FEATURES

“self-contained, full-screen UI level features [...]. Feature modules never directly depend on each other.”
Jeroen Mols

THEORY LIBRARIES

“functionality shared across multiple features. Different libraries can depend on each other”
Jeroen Mols

APPLICATION

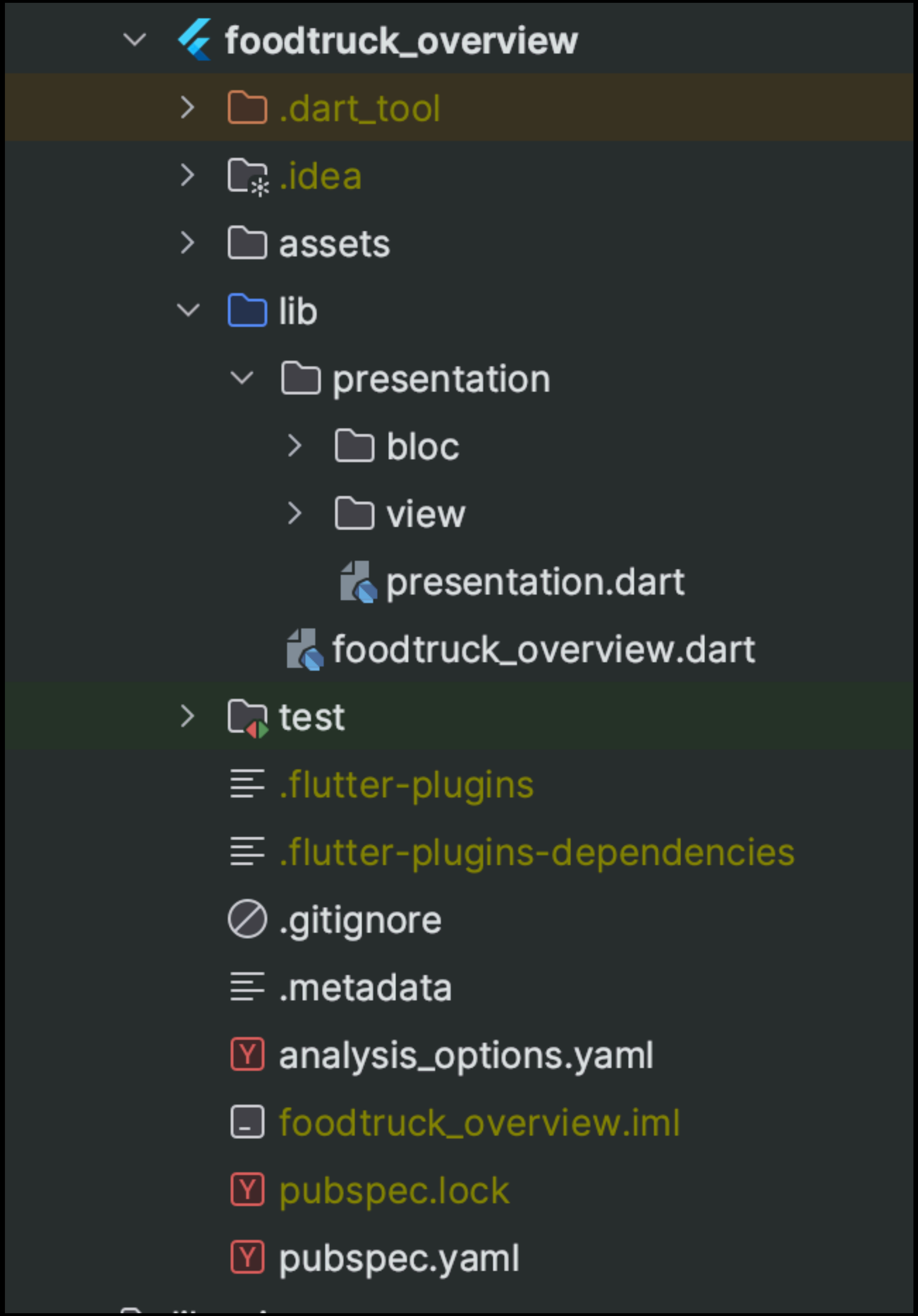
- The exact same approach can be used in Dart
- gradle modules become packages
- A Dart project doesn't contain at least one package but could be itself considered a package
- A Dart project can contain / reference multiple packages

APPLICATION

- You implement a package in your pubspec.yaml like any other dependency

```
dependencies:
  flutter:
    sdk: flutter

  # modules
  core:
    path: modules/libraries/core
```



```
foodtruck_overview
  .dart_tool
  .idea
  assets
  lib
    presentation
      bloc
      view
      presentation.dart
    foodtruck_overview.dart
  test
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  analysis_options.yaml
  foodtruck_overview.iml
  pubspec.lock
  pubspec.yaml
```

PACKAGE

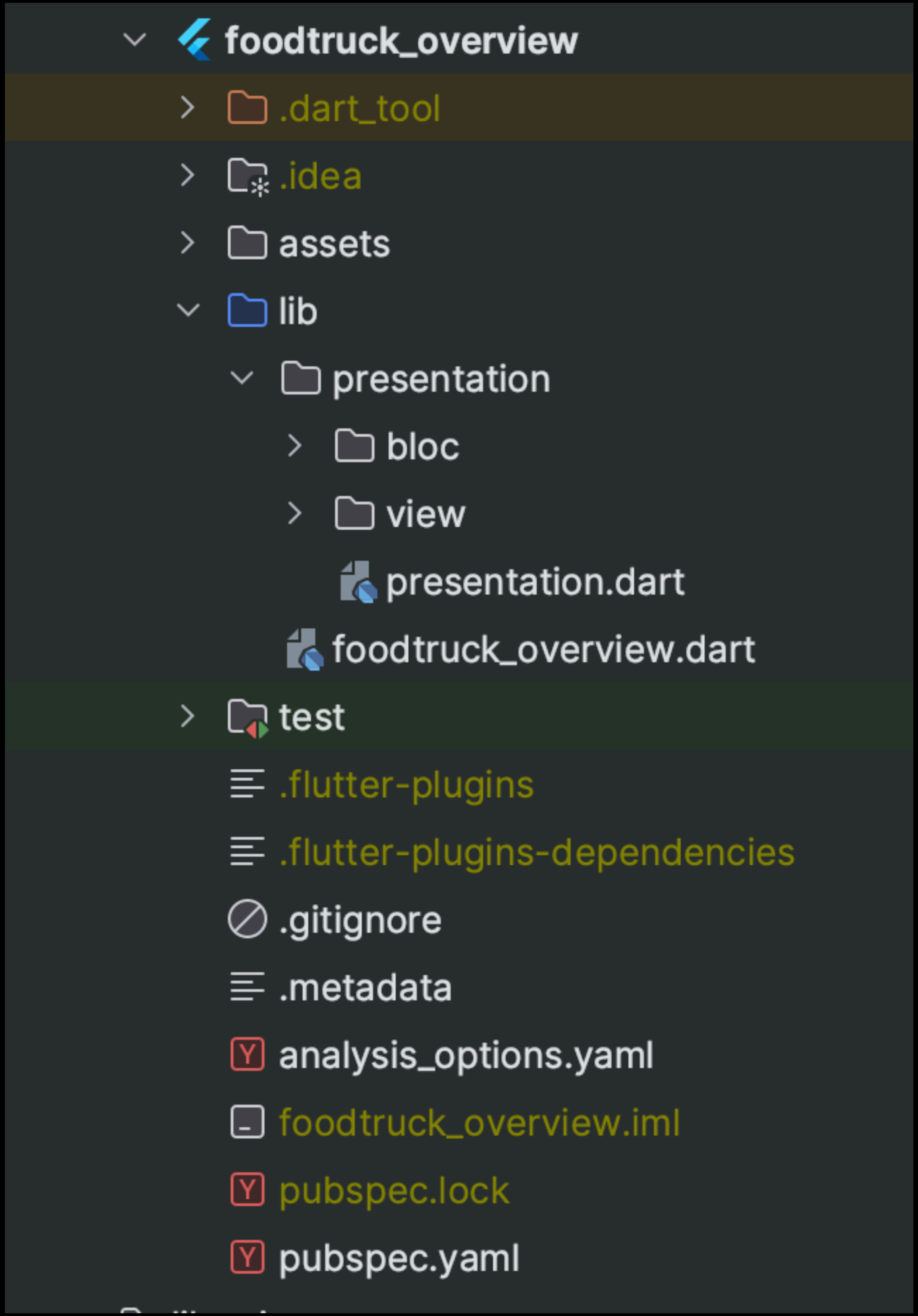
» A package is its own independent code base

» It has a `pubspec.yaml`

» It has its own tests

» It has its own `lib` folder

» The package exposes via `export` statement what is visible outside of the package



```
foodtruck_overview
  .dart_tool
  .idea
  assets
  lib
    presentation
      bloc
      view
      presentation.dart
    foodtruck_overview.dart
  test
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  analysis_options.yaml
  foodtruck_overview.iml
  pubspec.lock
  pubspec.yaml
```

PACKAGE

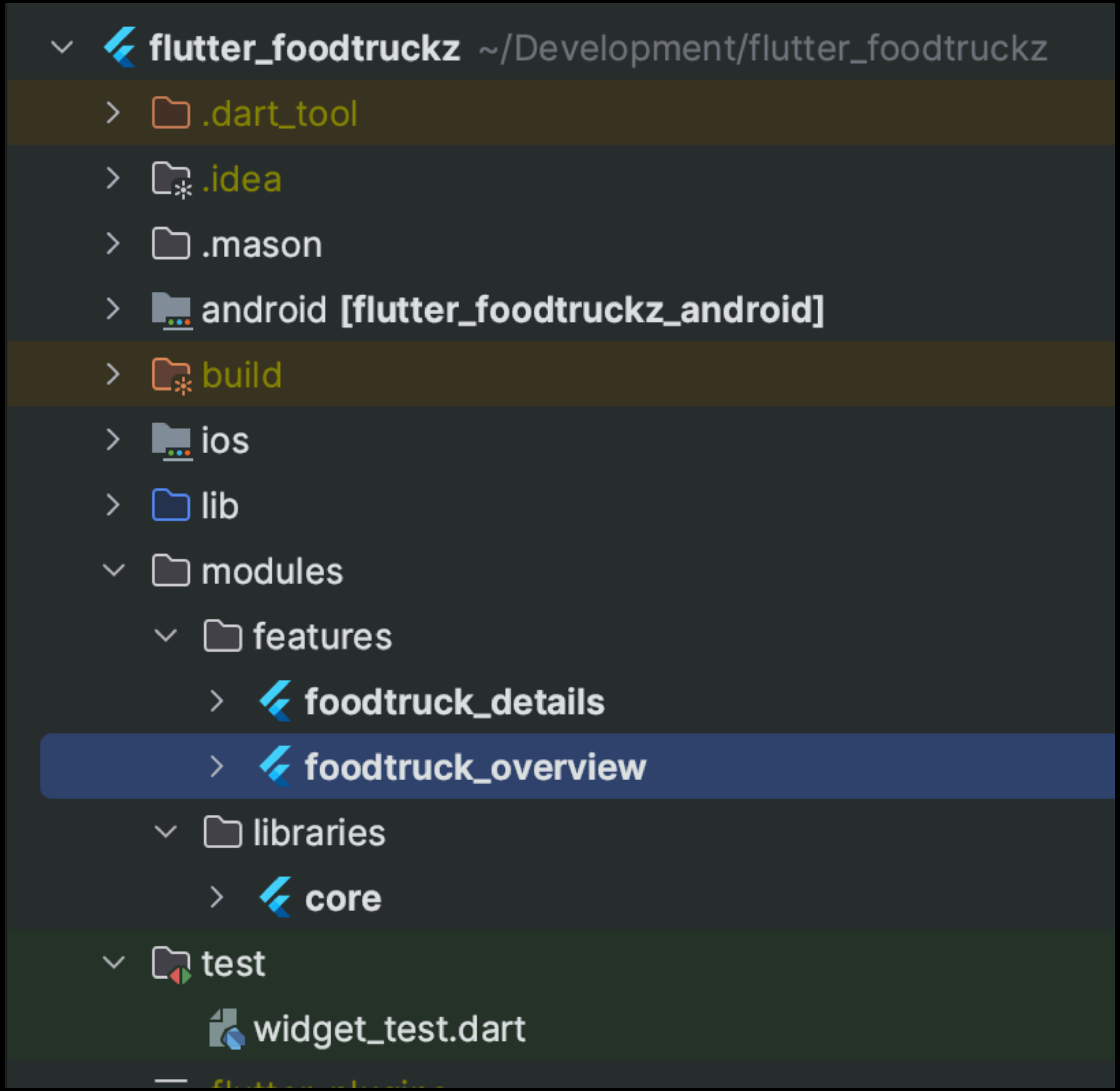
foodtruck_overview.dart:

```
library foodtruck_overview;

export 'presentation/presentation.dart';
```

presentation.dart:

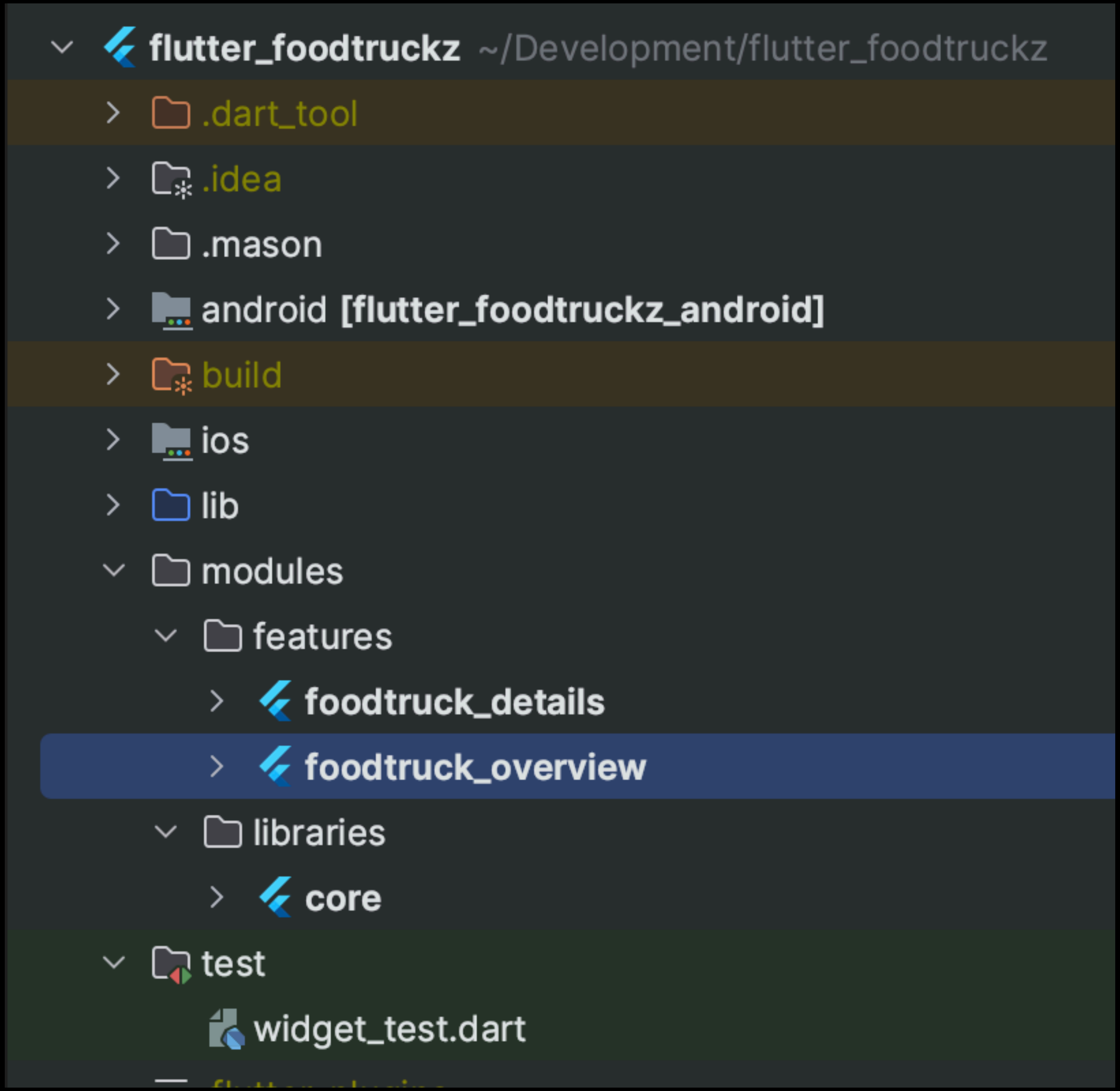
```
export 'bloc/overview_bloc.dart';
export 'view/overview_page.dart';
```



```
flutter_foodtruckz ~/Development/flutter_foodtruckz
  .dart_tool
  .idea
  .mason
  android [flutter_foodtruckz_android]
  build
  ios
  lib
  modules
    features
      foodtruck_details
      foodtruck_overview
    libraries
      core
  test
    widget_test.dart
```

PROJECT

- A project can contain multiple packages
- It has a `pubspec.yaml`
- It has its own tests
- It has its own `lib` folder



PROJECT

```
name: foodtruckz
description: Displays foodtrucks in Nuremberg for the current user location
publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=2.19.5 <3.0.0'

dependencies:
  flutter:
    sdk: flutter

  # modules
  core:
    path: modules/libraries/core
  foodtruck_overview:
    path: modules/features/foodtruck_overview
  foodtruck_details:
    path: modules/features/foodtruck_details
```

PROBLEMS

- How to sync overlapping dependencies over all packages?
- How to `flutter pub get` for all packages in one command
- How to `flutter analyze` for all packages in one command
- How to `flutter test` for all packages in one command

SOLUTION
MELOS

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

» Melos is maintained by invertase

» Also uses a `yaml` file for configuration

MELOS

A melos.yaml looks like

```yaml
name: uryde

packages:
  - modules/libries/**
  - modules/features/**
  - '*'

scripts:
  test:selective_unit_test:
    run: melos exec --dir-exists="test" --fail-fast -- flutter test --no-pub --coverage
    description: Run Flutter tests for a specific package in this project.
    select-package:
      flutter: true
      dir-exists: test

  test:
    run: melos run test:selective_unit_test --no-select
    description: Run all Flutter tests in this project.

  analyze:
    run: melos exec -- flutter analyze .
    description: Run `flutter analyze` in all packages.

  pub_get:
    run: melos exec -- flutter pub get .
    description: Run `flutter pub get` in all packages.
```

ALTERNATIVE APPROACH

Extract packages to their own git repository

» Could be a valid approach for a few truly independent modules, e.g. Firestore connector

» Can then be easily referenced in your `pubspec.yaml` via git url

```yaml
package1:
  git:
    url: https://<deploy_token_name>:<deploy_token_password>@gitlab.com/connectmobility/package1.git
    ref: 4.6.2

package2:
  git:
    url: https://<deploy_token_name>:<deploy_token_password>@gitlab.com/connectmobility/package2.git
    ref: develop
```

THANKS

SOURCES

- Modularization - A successful architecture
- Melos
- Managing multi-package Flutter projects with Melos
- Flutter Pubspec and private Gitlab Repositories