

Mobile Development with Flutter: **Clean Architecture and Modular Monolith Design Considerations**

Sergiy Tytenko
Valentyna Polienova
Vladyslav Fedenko



01

**What is Flutter.
Architecture
basics.**



Flutter's approach

- High-performance rendering engine → **customizability**
- Widget-centric architecture → **scalability**
- Customizable widgets to mimic the native controls → **native look**
- Hot reload feature → **developer productivity boost**

Flutter vs React Native



Flutter




React Native

UI Components

Own rendering engine

Native components for UI

Performance

 No Bridge

JavaScript Bridge

Programming Language

Dart

JavaScript

Hot Reload



Community & Ecosystem

Quickly developing

More third-party libraries

Cisno

Examples of
Flutter Apps



ebay



Software Design

Architectural Patterns

Architectural Patterns, often called “Software Design Patterns,” address recurring architectural problems. For example, the Model-View-Controller (MVC) pattern helps to separate the user interface from the model. In other words, it resolves a specific issue.

Architectural Styles




Architectural Style is a broader design approach that provides a high-level organizational blueprint rather than direct solutions to problems. Styles deal with structure and design philosophy, including guidelines, principles, and standards.



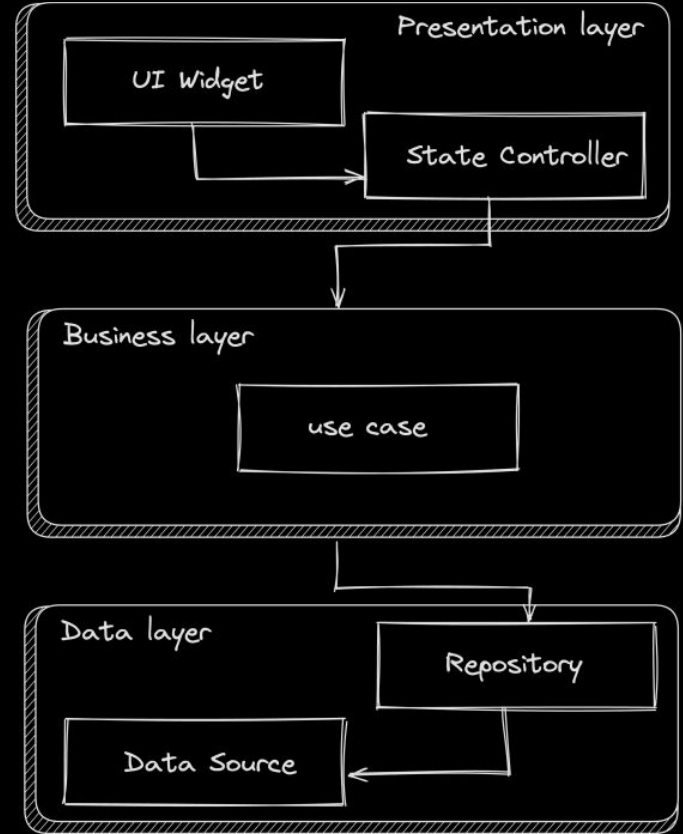
How to choose the right architecture?

In Flutter development, choosing the right UI architecture involves weighing the specific **needs and goals of the project**.

It's about balancing technical requirements, the application's complexity, and the development team's dynamics.



1. **Presentation (UI) Layer:** This is the topmost layer, primarily dealing with the user interface.
Responsibilities: displaying information, capture user events and pass them to the next layer.
2. **Business Logic Layer:** Often referred to as the domain layer, it contains the application's core business logic.
Responsibilities: process data, apply business rules and validations.
3. **Data Layer:** The bottom layer is where data management occurs.
Responsibilities: data persistence and retrieval. Abstracting the origin of a data from other layers.



Advantages of the three-layered architecture

Separation of Concerns

Easier Testing and Debugging

Loose coupling

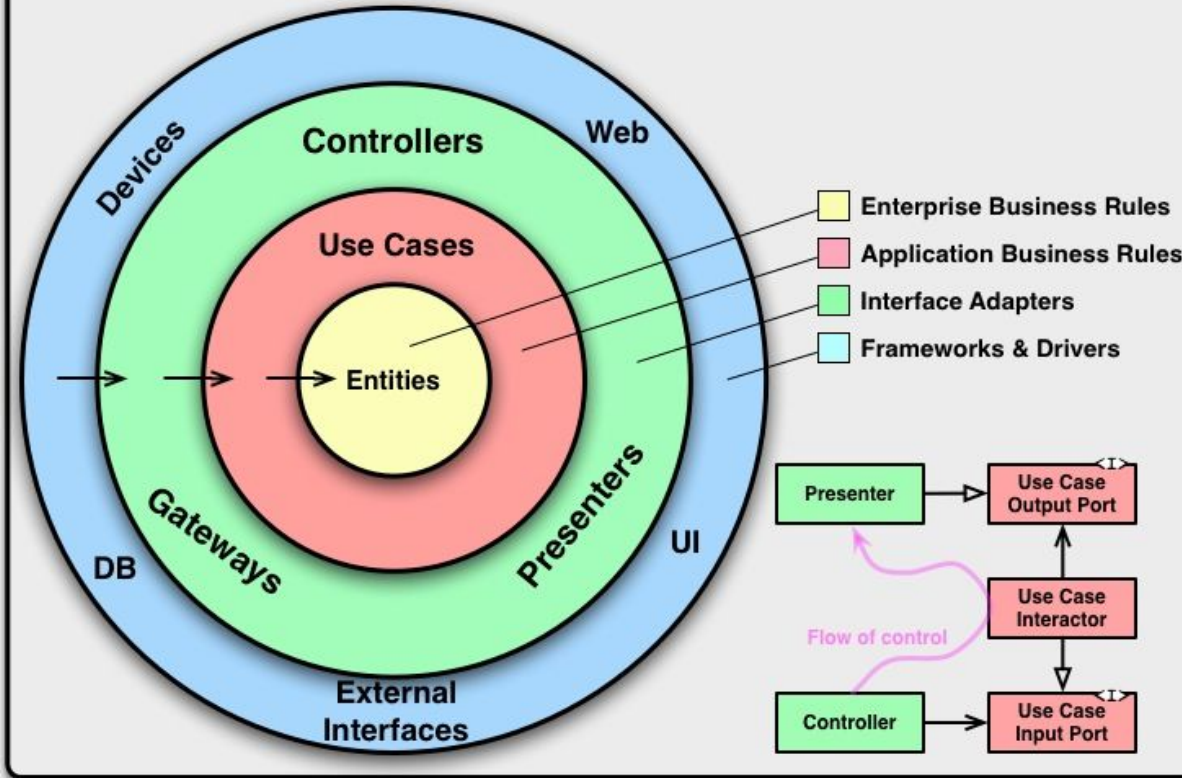
Flexibility



02

**Clean Architecture
and its benefits for
mobile development**

The Clean Architecture

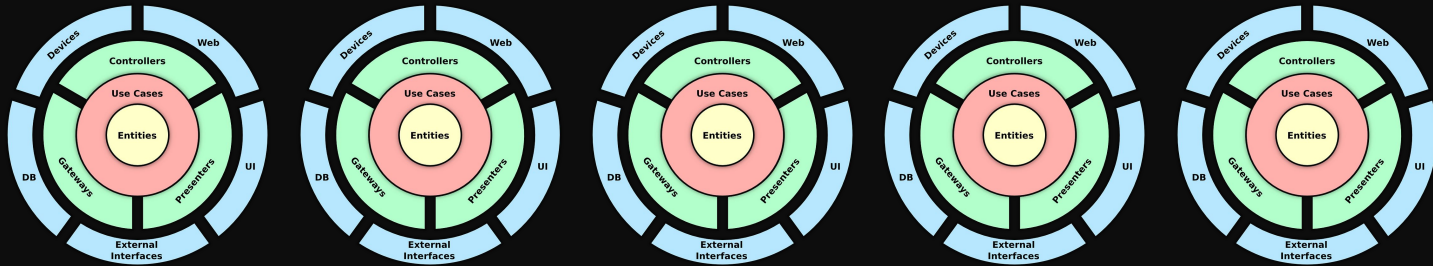


Robert C. Martin (Uncle Bob)

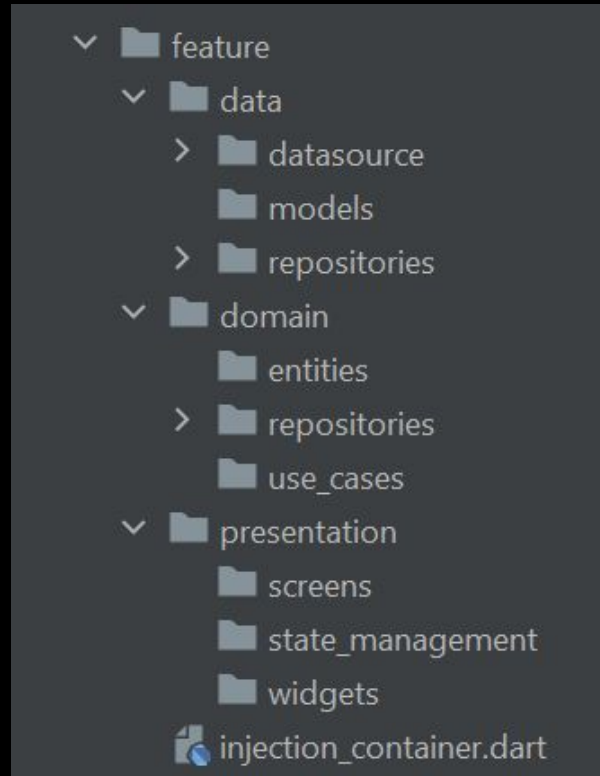
<https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

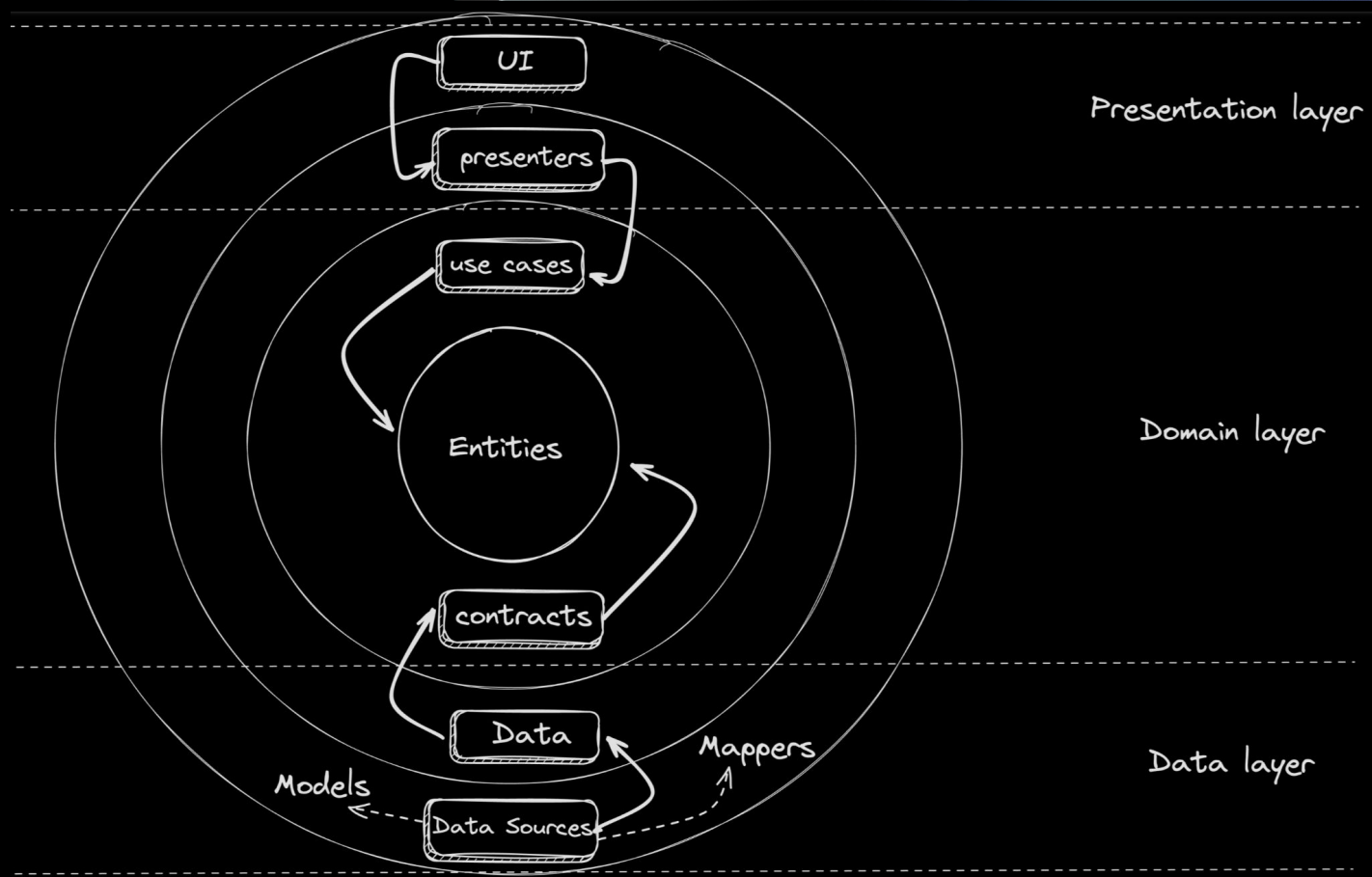
Feature-First Modular Clean Architecture

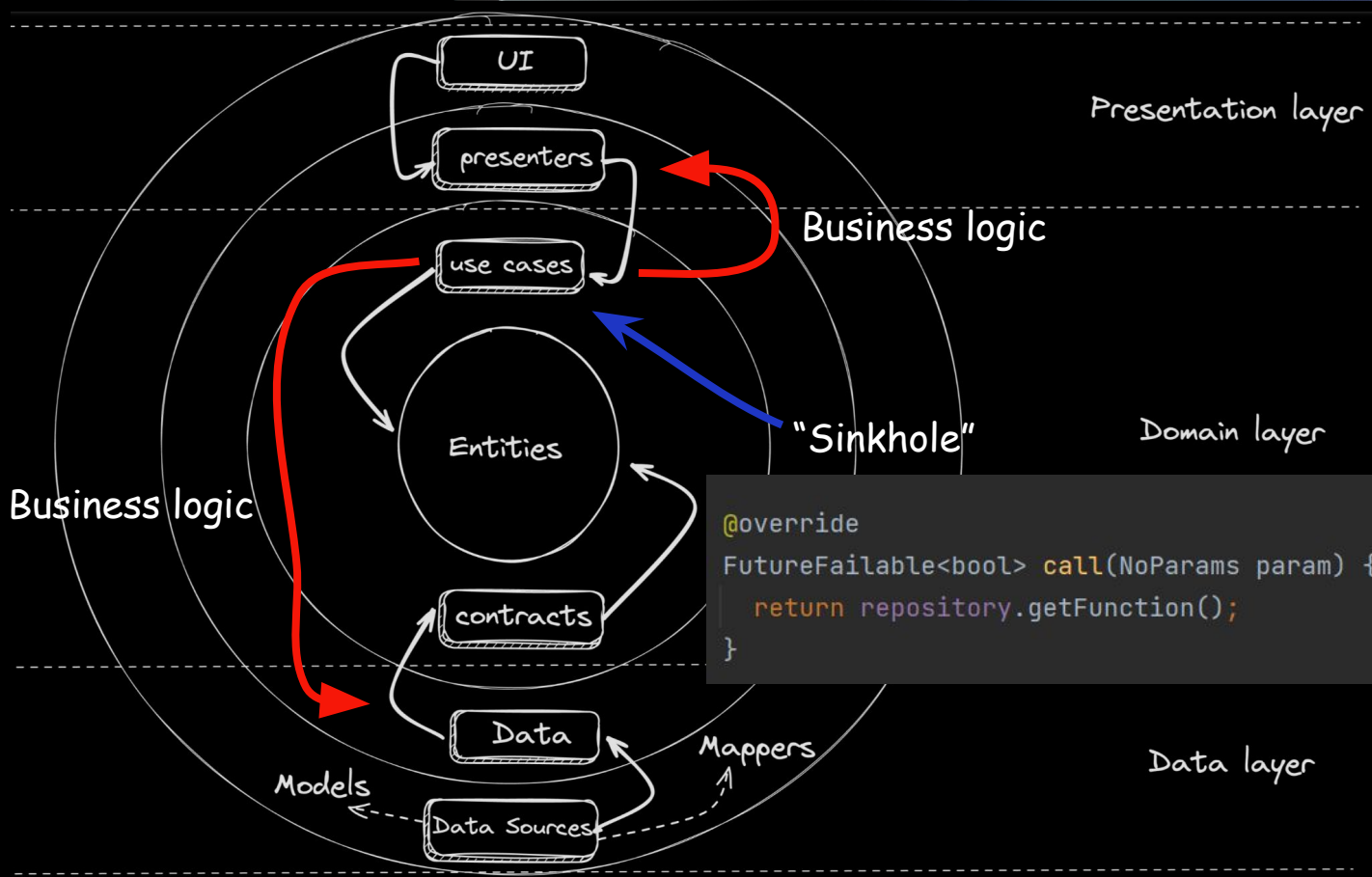
Features

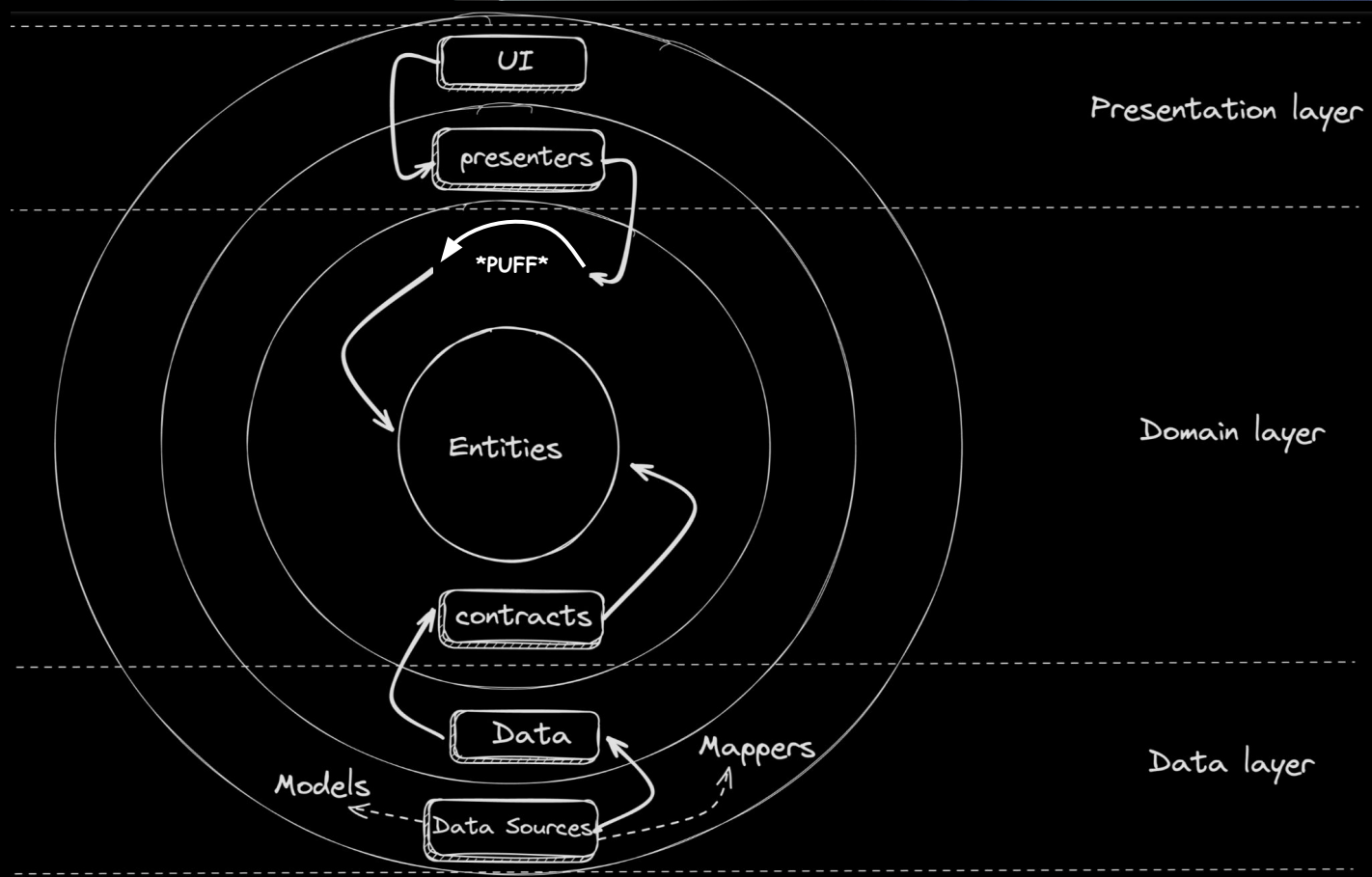


Feature folder structure

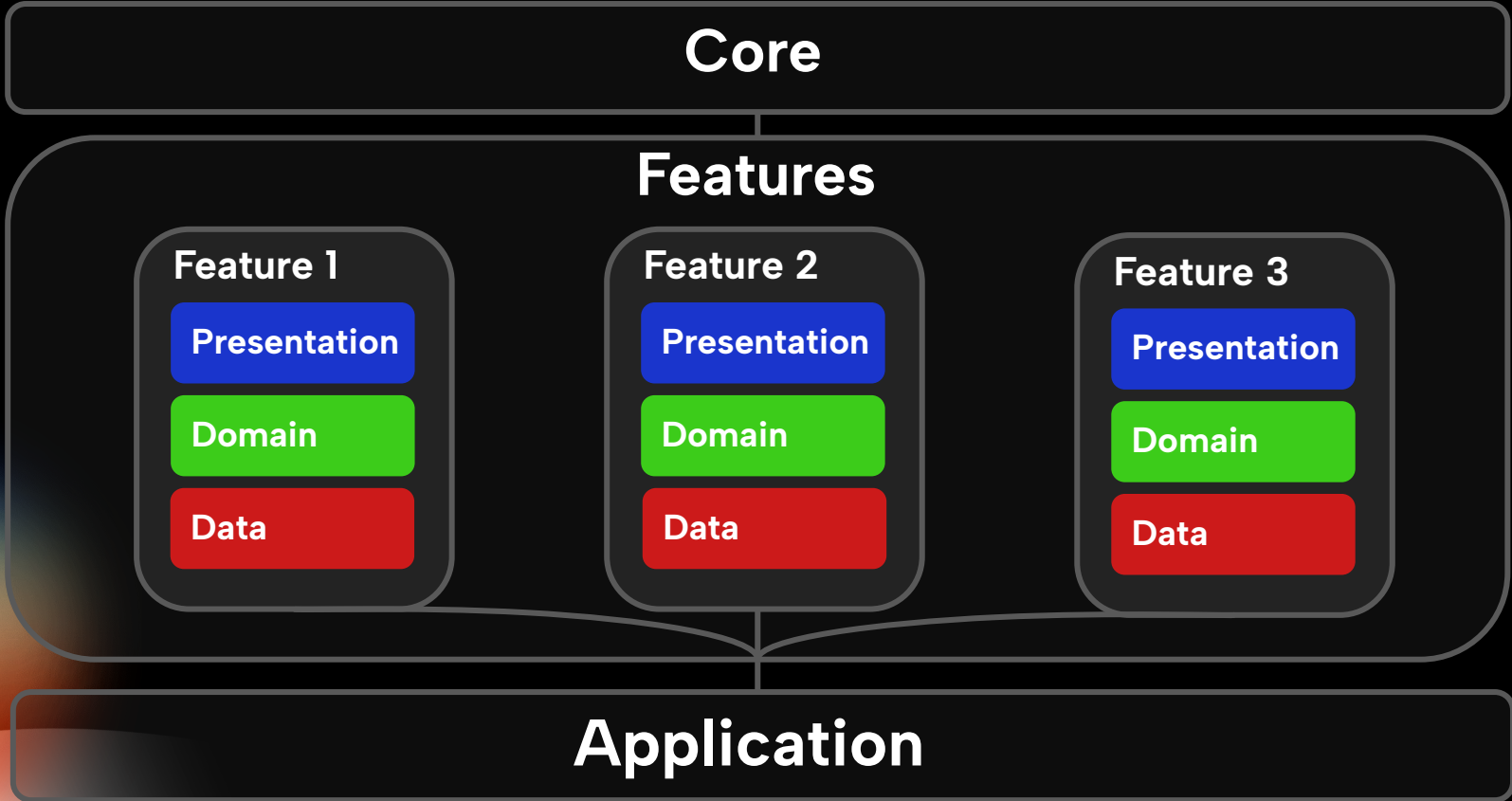








Modular Monolith



Benefits of Modular Monolith

Separation of Concerns

Testability

Maintainability

Flexibility

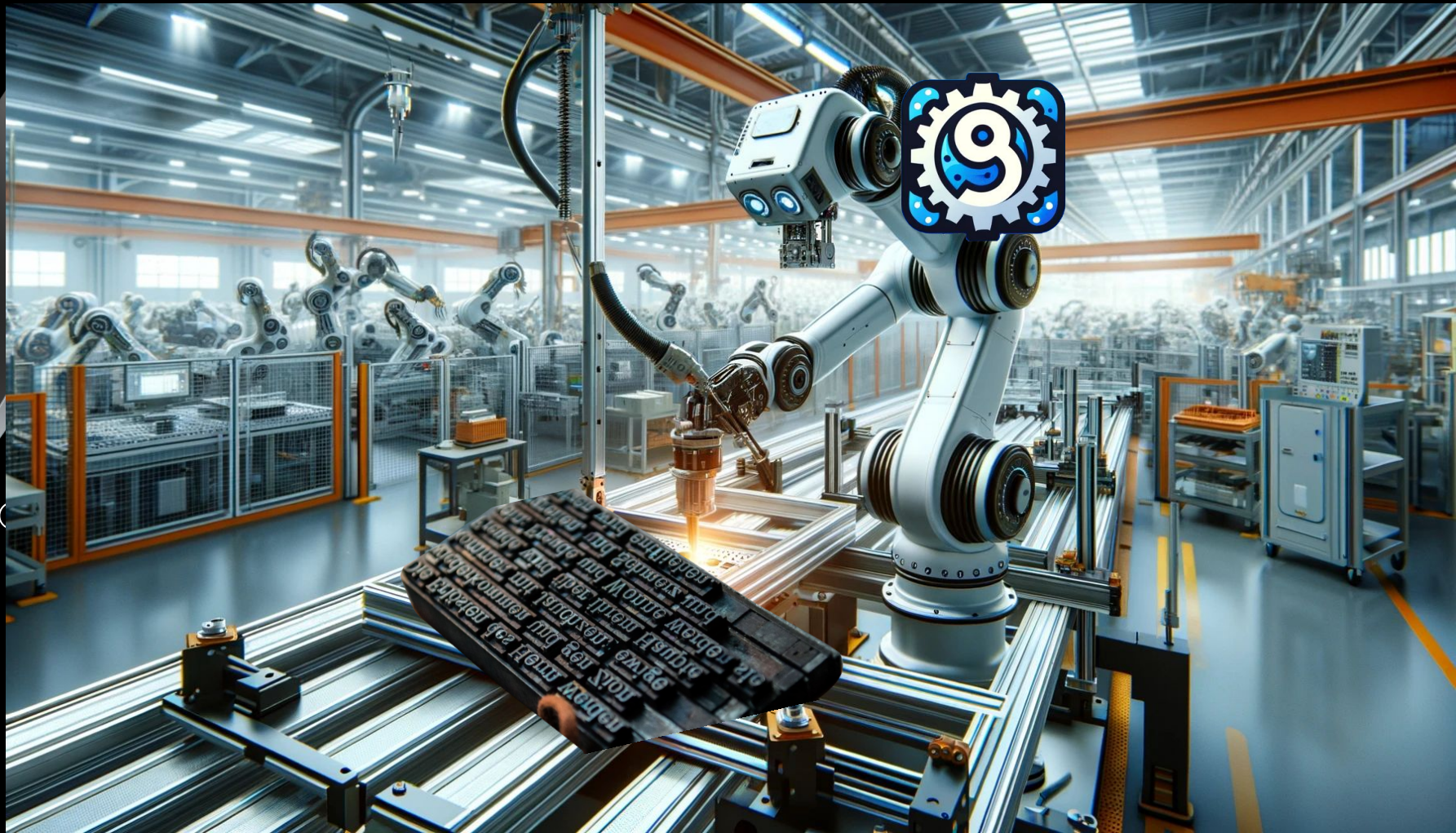
Adaptability

Modifiability



Developer

Boilerplate



Boilerplate Generator Plugin

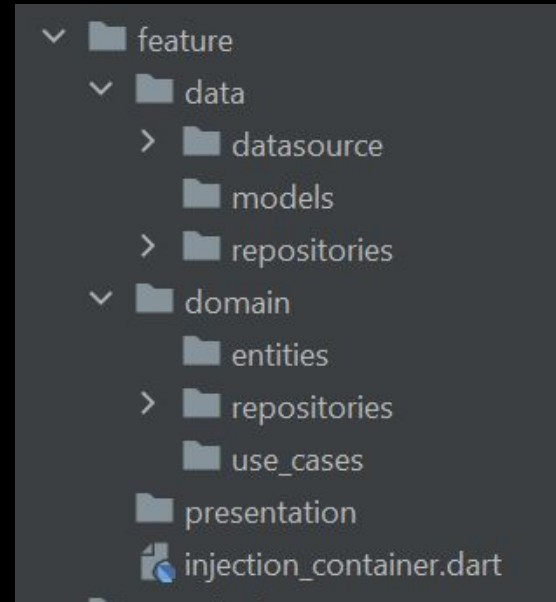
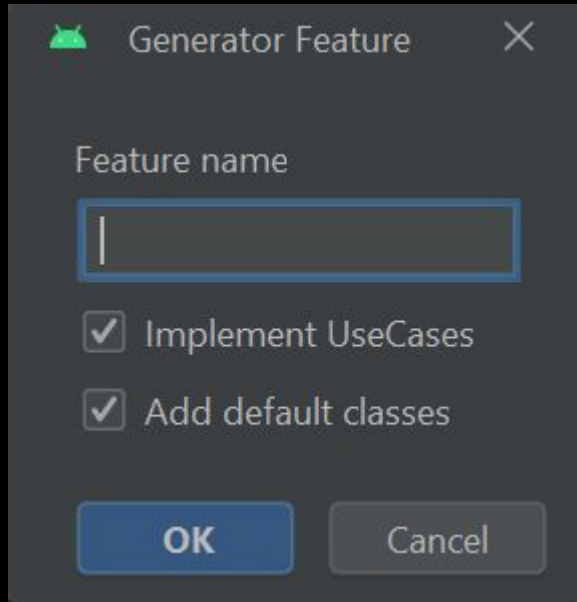


Feature Generation


**Method Chain
Generation**

**Class Chain
Generation**

Feature Generation



Class Chain Generation

 Generate Classes ✕

Class name

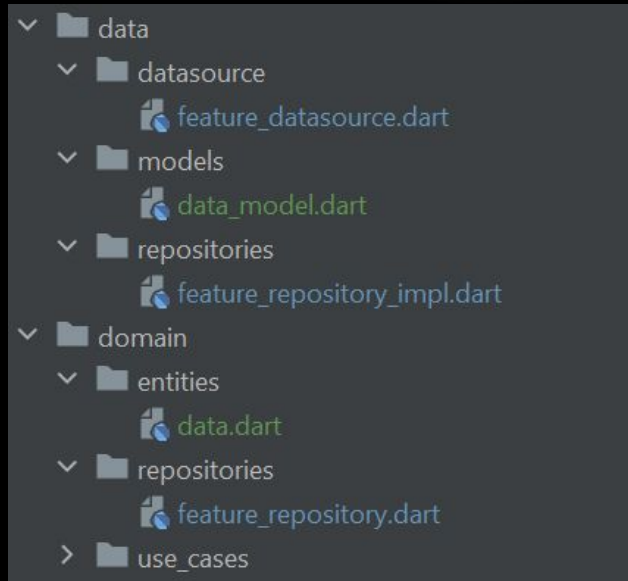
Create repository class

Choose existing datasource

Create datasource class

- data
 - datasource
 - feature_datasource.dart
 - models
 - repositories
 - feature_repository_impl.dart
- domain
 - entities
 - repositories
 - feature_repository.dart
 - use_cases

Method Chain Generation



A screenshot of the "Method Chain Generator" dialog box. The dialog is dark-themed and contains the following fields and options:

- Add UseCase
- Method name:
- Return type (bool or custom entity):
- The return type is a list of objects
- Create class for return type
- Method parameter type (common data types or custom entity):
- Create class for method parameter type
- Method parameter name:
- Add repository template content
- Add datasource template content
- Usecase repository:
- Repository datasource:
- Datasource rest method:
- Datasource API path:

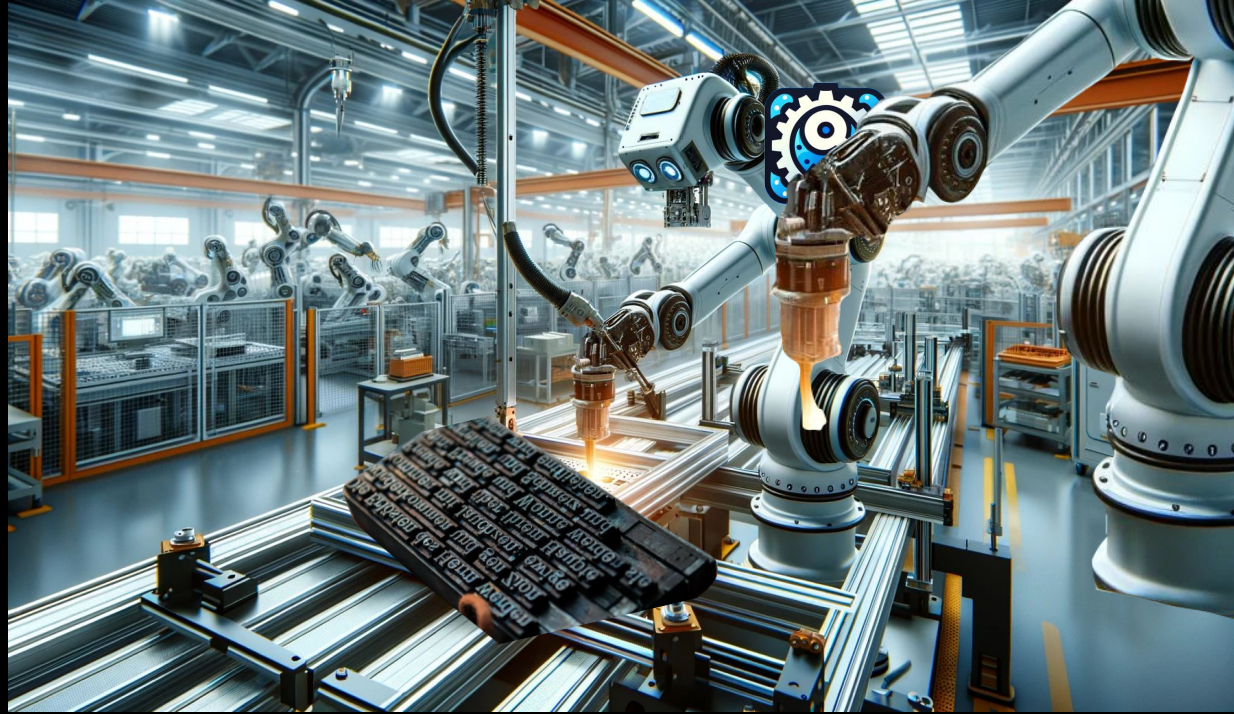
At the bottom right, there are two buttons: "OK" and "Cancel".

Method Chain Generation Code

```
abstract class FeatureDatasource {  
    Future<Data> getData();  
}  
  
class FeatureDatasourceImpl extends FeatureDatasource {  
    FeatureDatasourceImpl({required this.dio});  
  
    final Dio dio;  
  
    @override  
    Future<Data> getData() async {  
        final response = await dio.get('');  
  
        Data data = DataModel.fromJson(response.data);  
        return data;  
    }  
}
```

```
class FeatureRepositoryImpl extends FeatureRepository {  
    final FeatureDatasource featureDatasource;  
  
    FeatureRepositoryImpl({required this.featureDatasource,});  
  
    @override  
    FutureFailable<Data> getData() {  
        return RepositoryRequestHandler<Data>(  
            request: () => featureDatasource.getData(),  
            defaultFailure: Failure(),  
        );  
    }  
}
```

```
abstract class FeatureRepository {  
  
    FutureFailable<Data> getData();  
}
```



Conclusions

1. Flutter stands out as a powerful tool for rapidly developing cross-platform mobile applications.
2. Adopting a multilayer modular monolith approach proves to be a convenient strategy for designing mobile apps that are maintainable and easily modifiable.
3. It is imperative to steer clear of canonical solutions that result in excessive boilerplate code and to actively mitigate the appearance of the sinkhole anti-pattern.
4. Leveraging custom IDE plugins can significantly automate the coding process.
5. It is essential to be mindful of the terminology used to accurately characterize the architecture employed in our projects.

The background is a dark blue gradient, transitioning from a lighter blue at the top to a darker blue at the bottom. There are several white circles of varying sizes scattered across the scene. Two large, grey, semi-transparent, curved shapes are positioned at the bottom left and bottom right corners, resembling stylized planet surfaces or abstract forms.

Thank you!

Resources

1. Flutter Engineering (2024). Majid Hajian.