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


Сільпо
Examples of
Flutter Apps
ebay
BMW

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

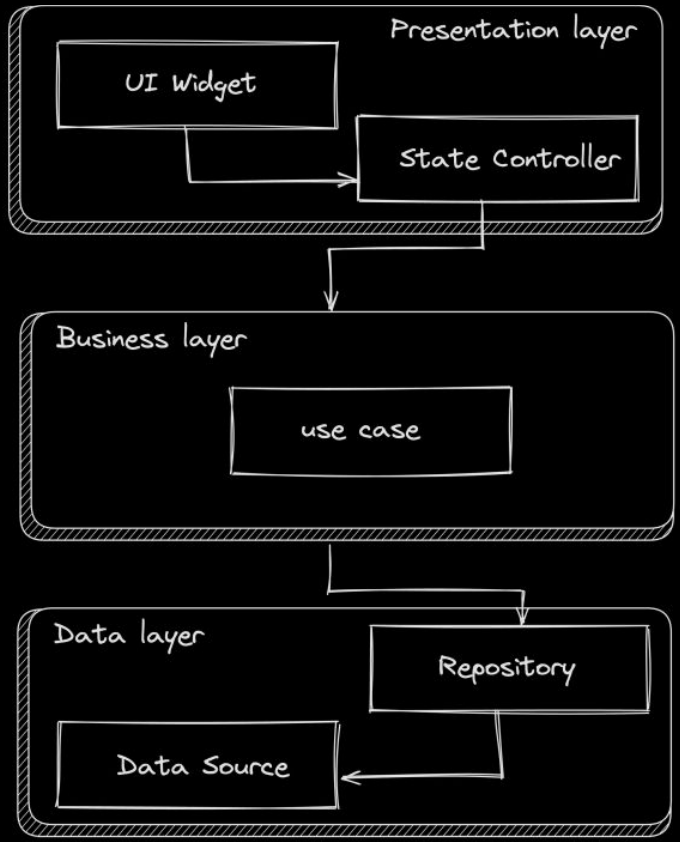
Architectural Style is a broader design approach that provides a high-level organizational blueprint rather than direct solutions to problems. Styles deal with structure and design philosophy, including guidelines, principles, and standards.

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.



Flutter Engineering (2024). Majid Hajian.

Advantages of the three-layered architecture

Separation of Concerns

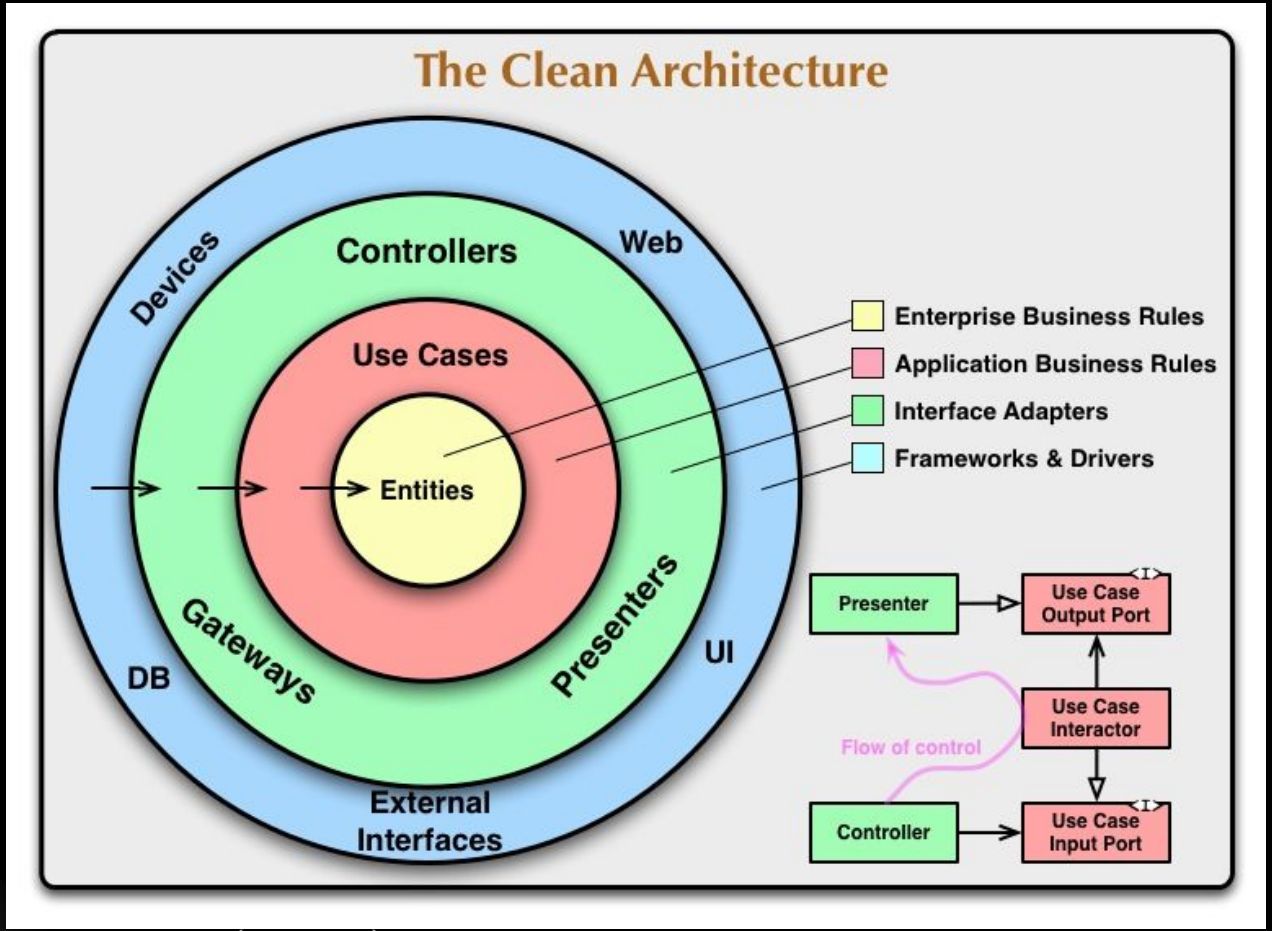
Easier Testing and Debugging

Loose coupling

Flexibility

02

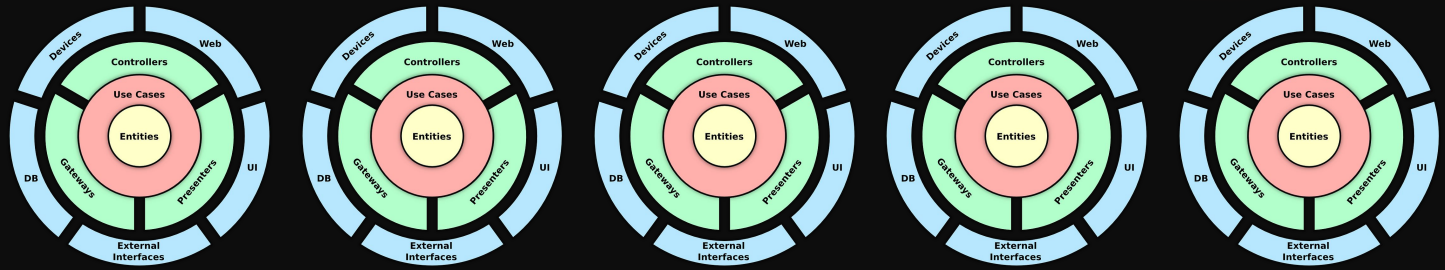
Clean Architecture and its benefits for mobile development



Robert C. Martin (Uncle Bob)
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

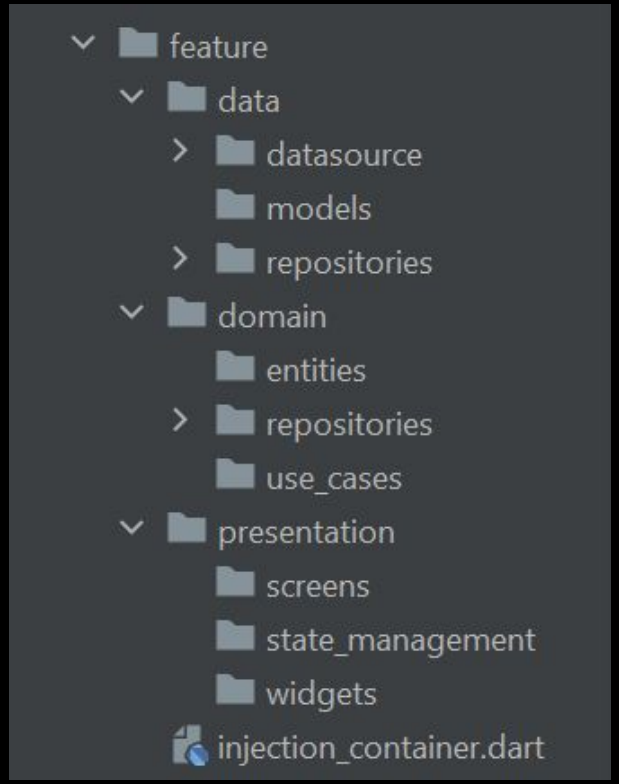
Feature-First Modular Clean Architecture

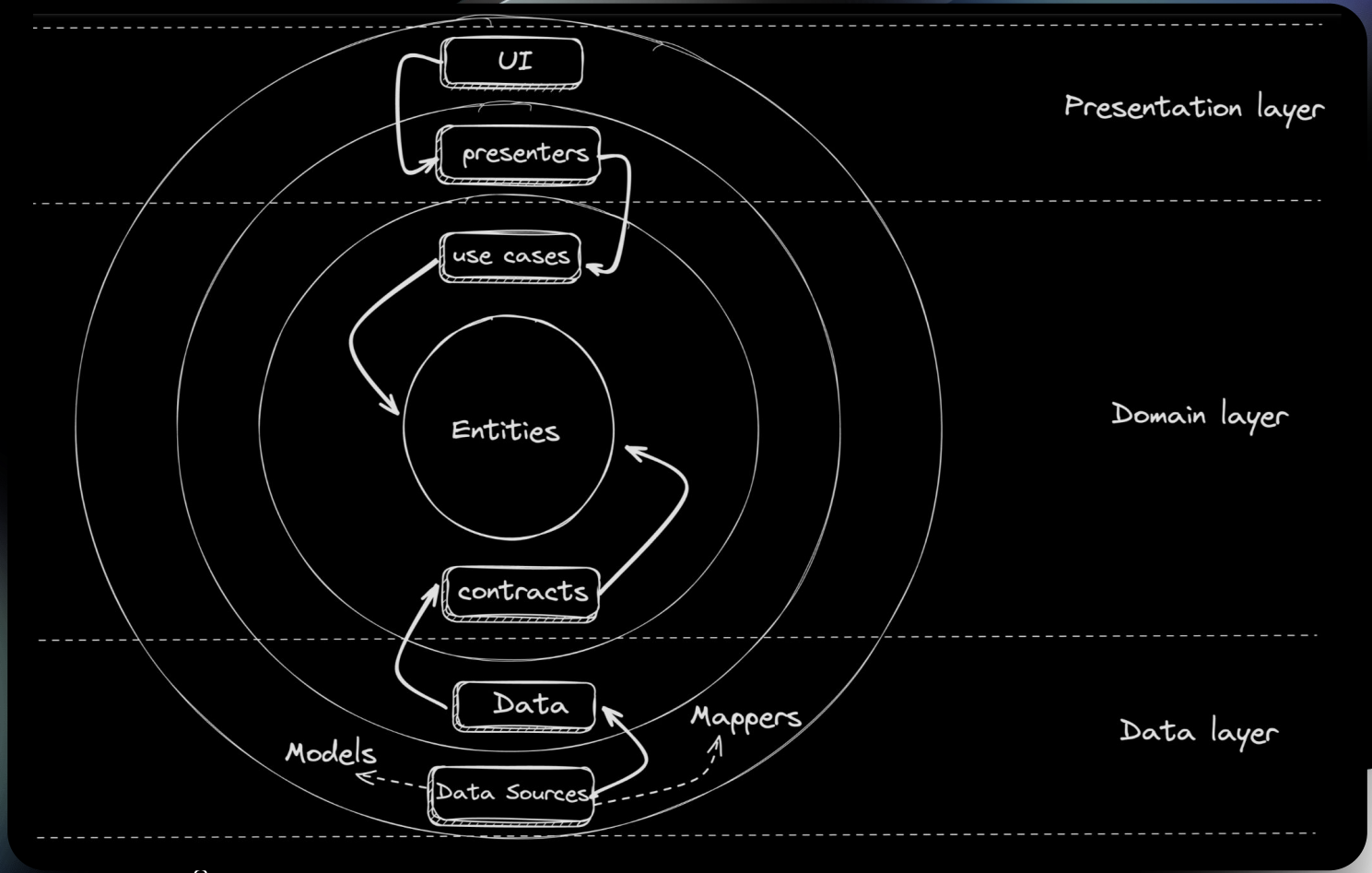
Features



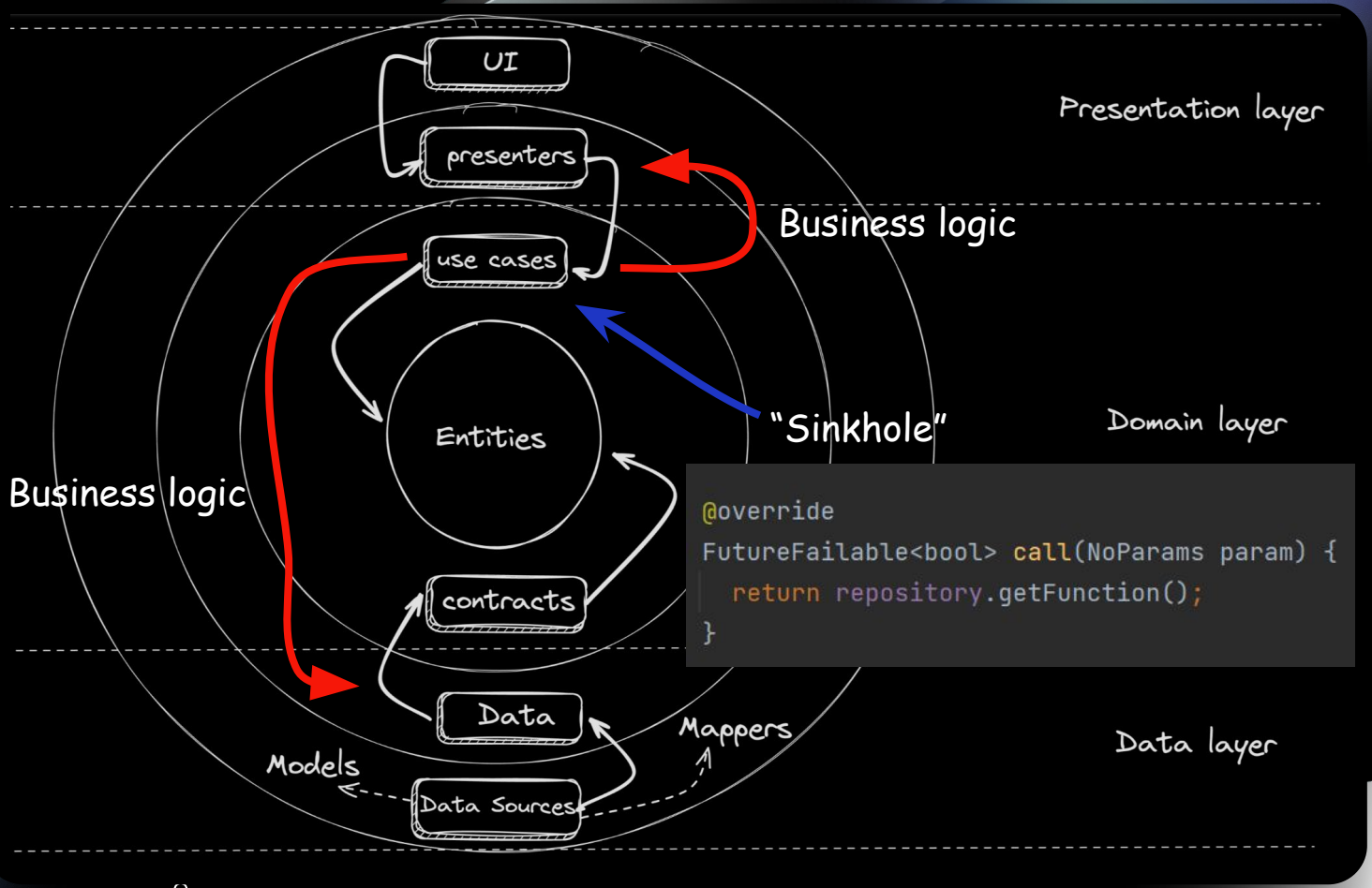
Matej Rešetár Feature-First Modular Clean Architecture

Feature folder structure

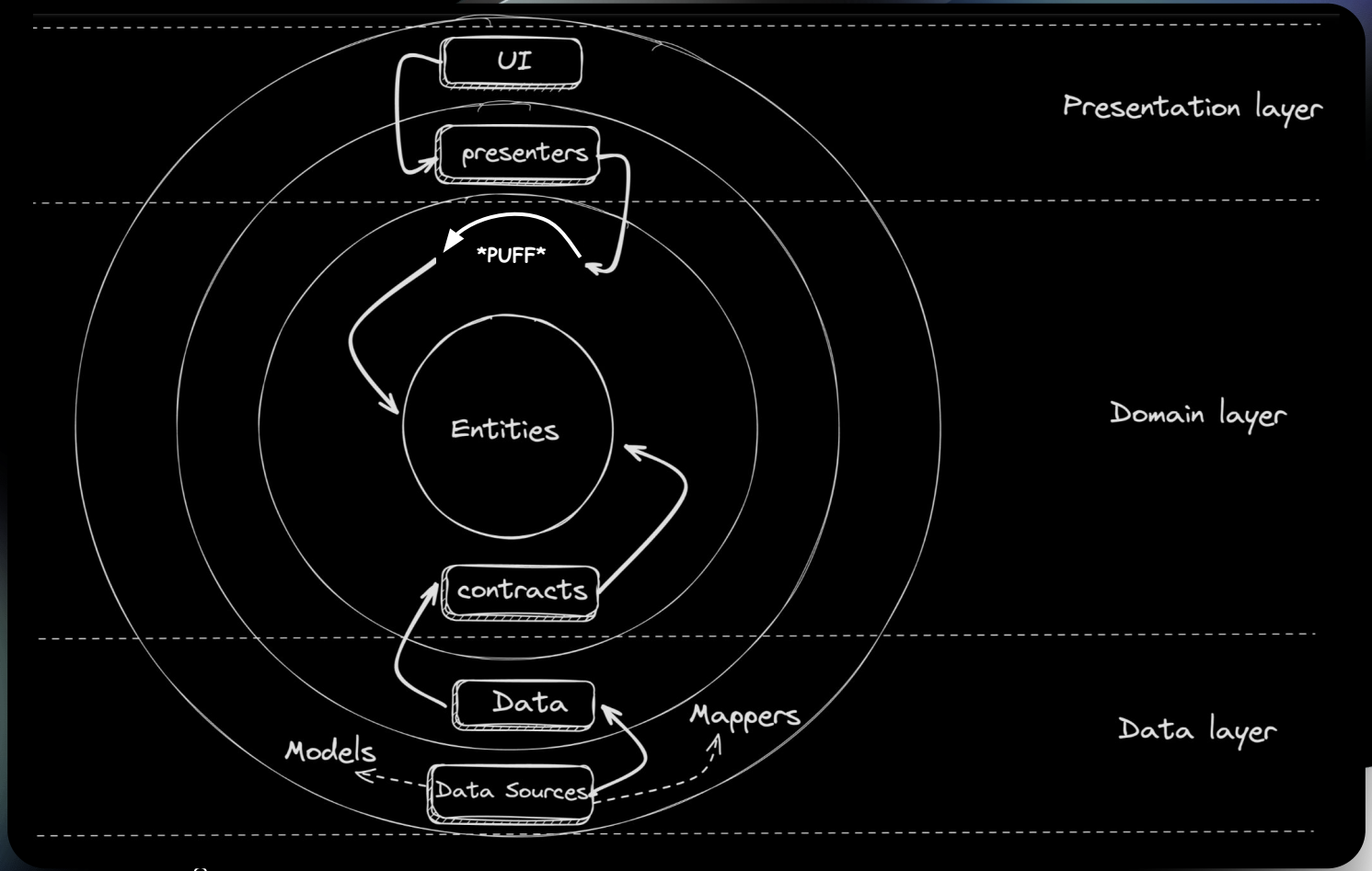




Flutter Engineering (2024). Majid Hajian.

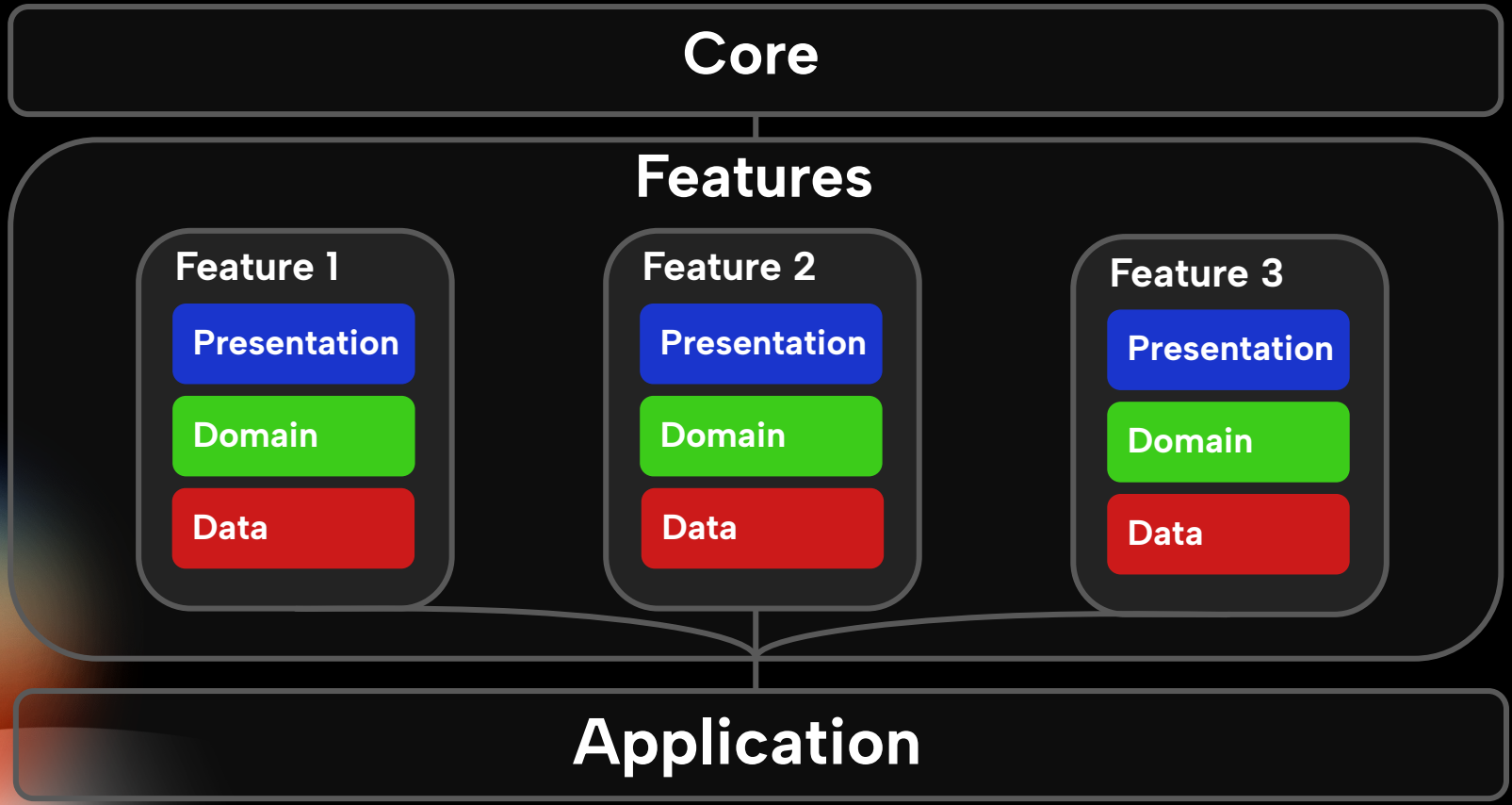


Flutter Engineering (2024). Majid Hajian.



Flutter Engineering (2024). Majid Hajian.

Modular Monolith

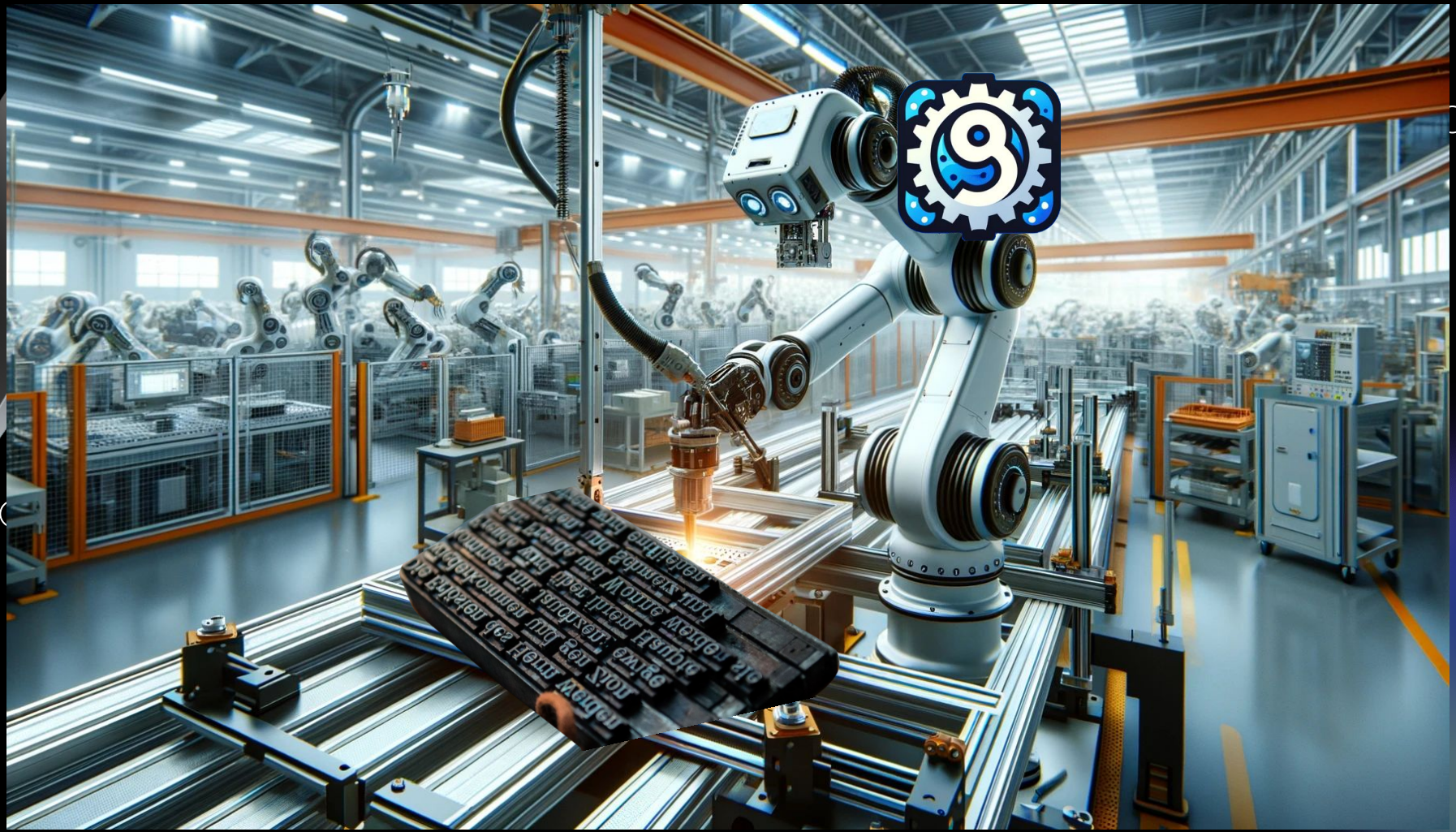


Benefits of Modular Monolith

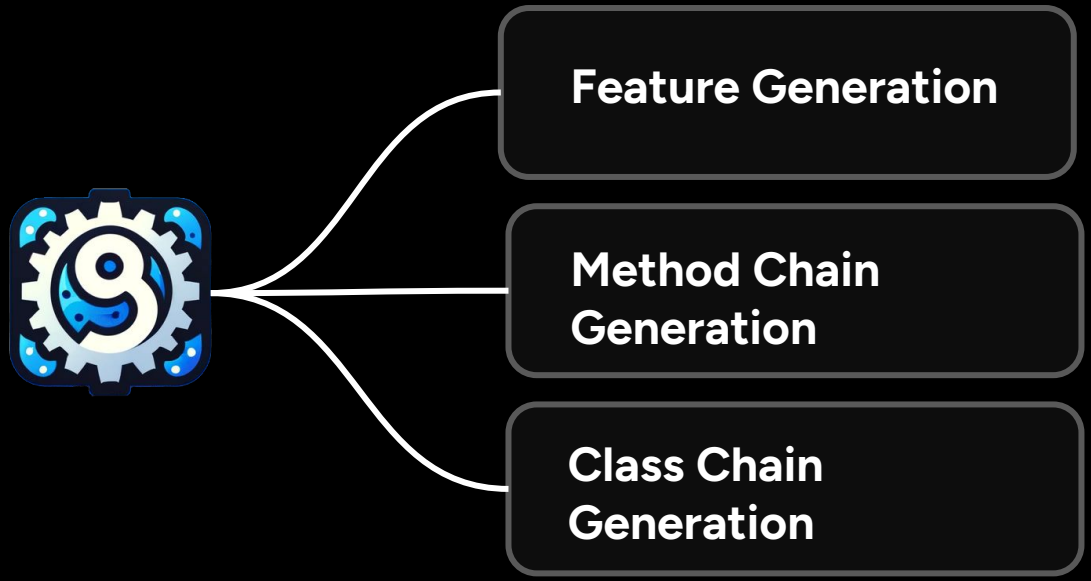
- Separation of Concerns
- Testability
- Maintainability
- Flexibility
- Adaptability
- Modifiability

Developer

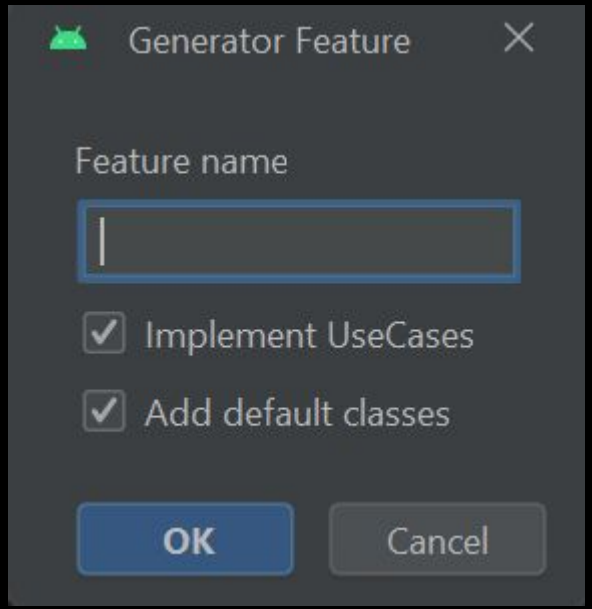
Boilerplate

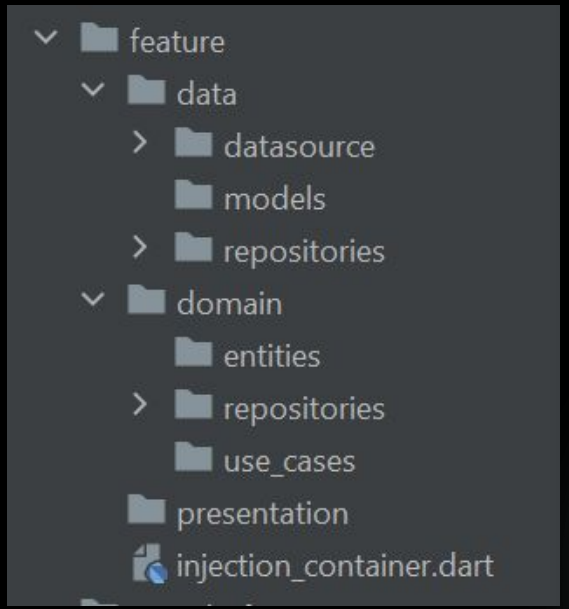


Boilerplate Generator Plugin

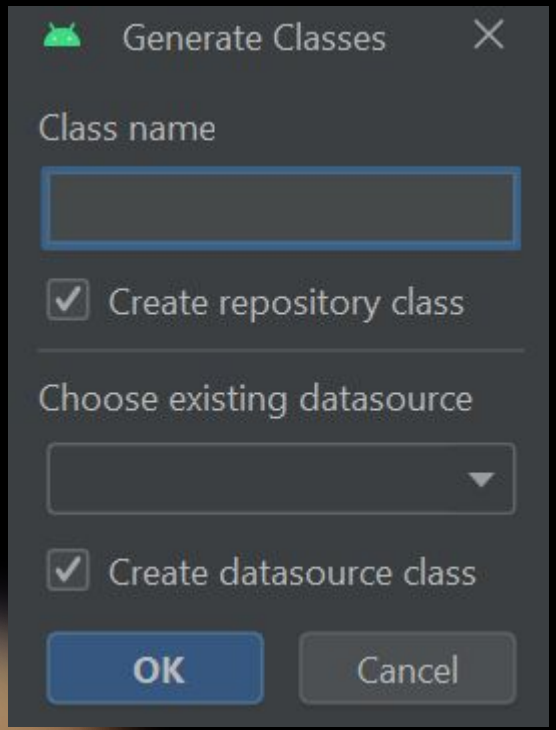


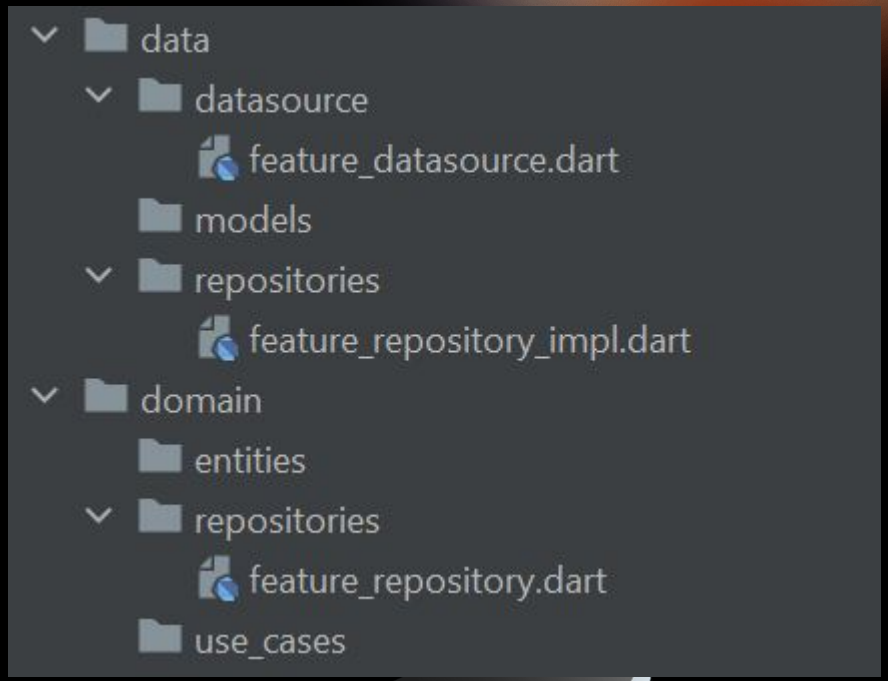
Feature Generation



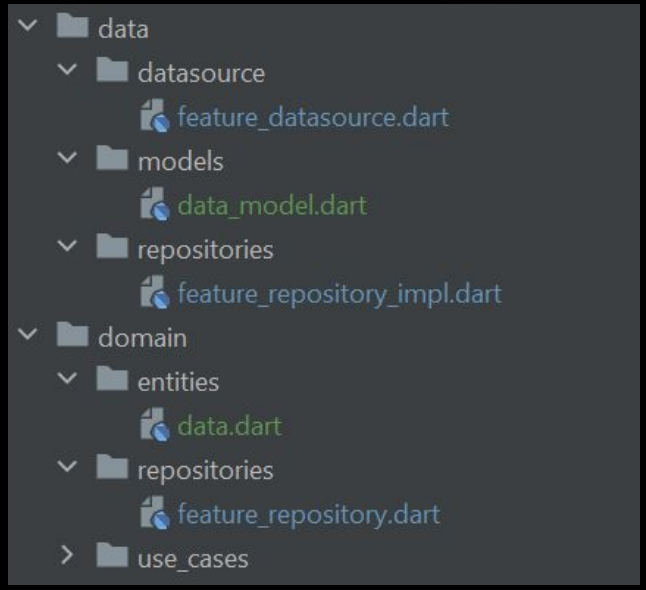


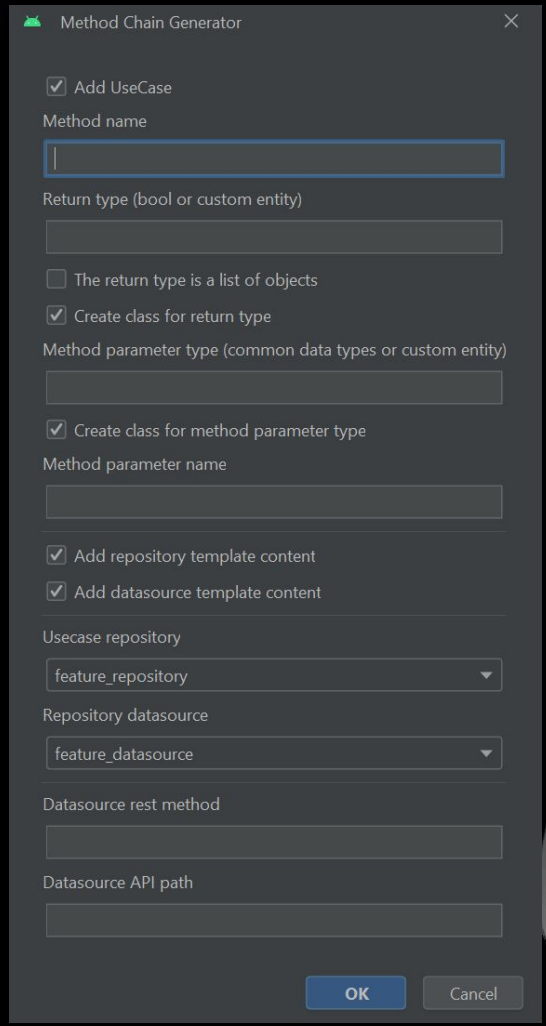
Class Chain Generation





Method Chain Generation



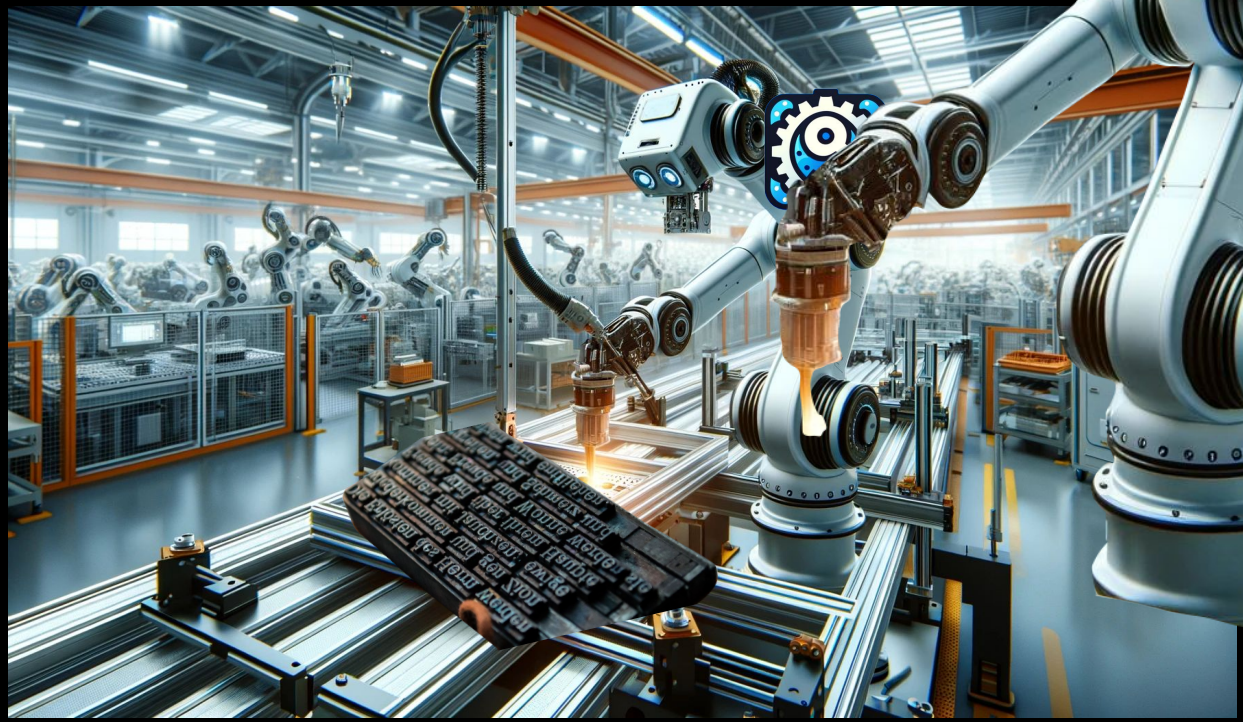


Method Chain Generation Code

```dart
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;
  }
}
```

```dart
class FeatureRepositoryImpl extends FeatureRepository {
  final FeatureDatasource featureDatasource;

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

}
```

```dart
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.

Thank you!

Resources

1. Flutter Engineering (2024). Majid Hajian.