beetech

Как мы делали суперапп «Дилер онлайн»
и что из этого получилось

Скирюк Олег

Frontend Team Lead

Санкт-Петербург, 19 октября 2024

Олег Скирюк

Frontend Team Lead в билайн

@skiryukoleg

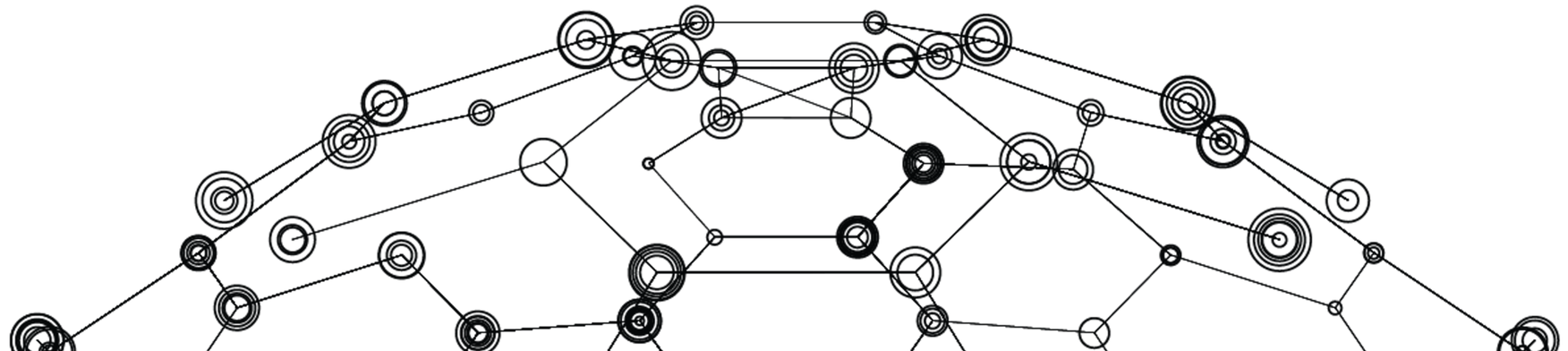
> Создаю, автоматизирую, оптимизирую процессы не только в работе, но и в жизни

О чем расскажу:

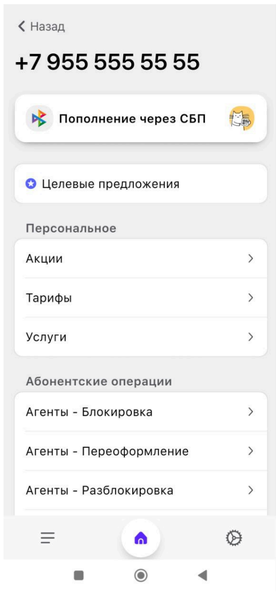
- о суперапп приложении
- о подходах, которые рассматривались к структуре приложения
- про нюансы выбранной структуры, что необходимо учесть

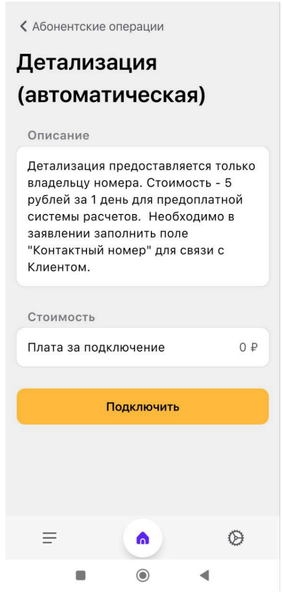
И...

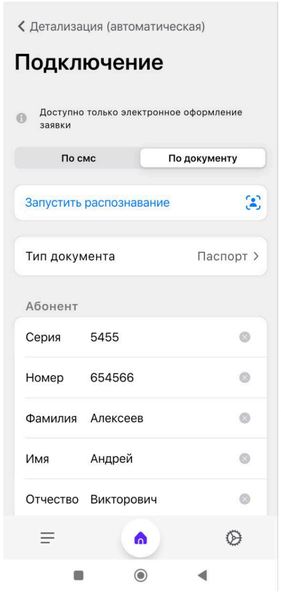
далее коснемся вопросов управления зависимостями и состоянием
и подведем итоги

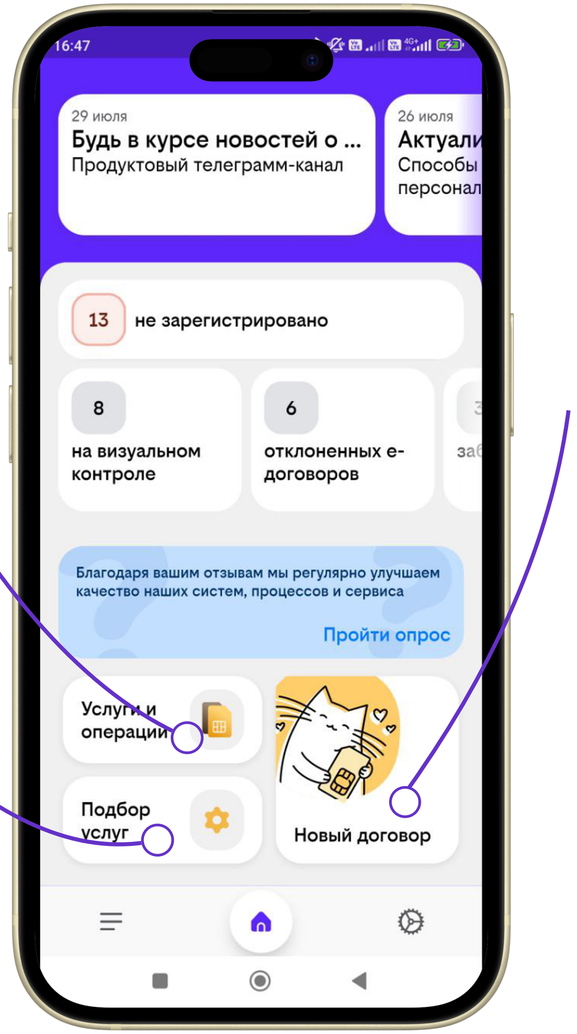


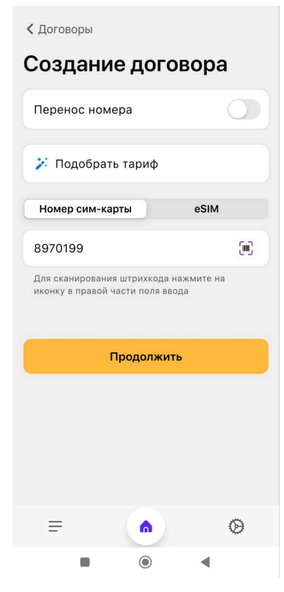
Приложение для партнеров (дилеров)

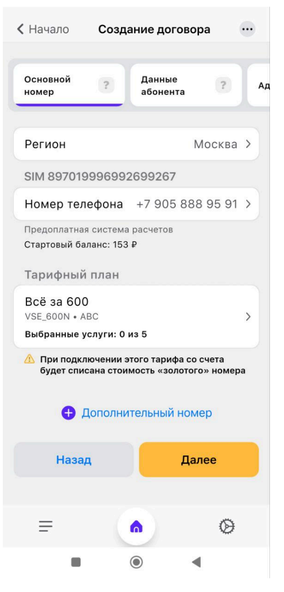


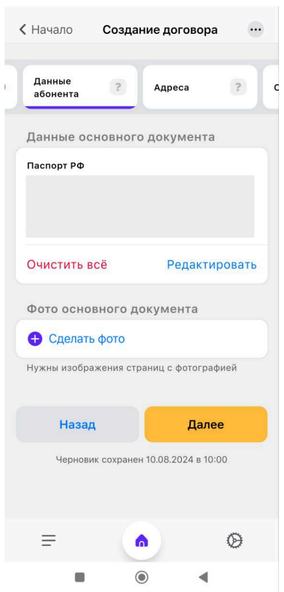


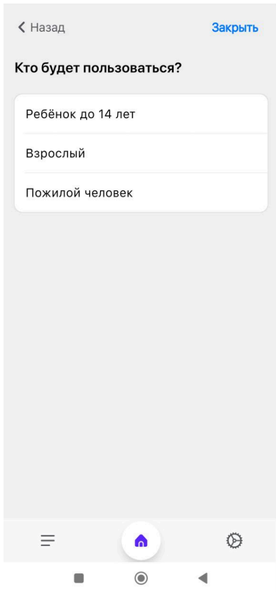


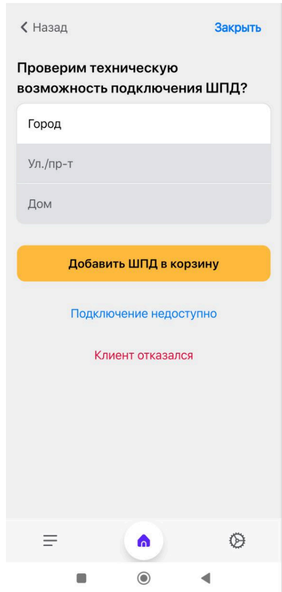


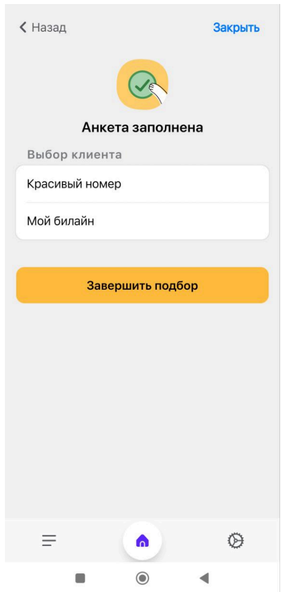


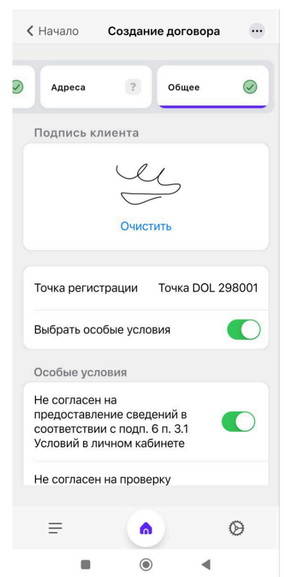








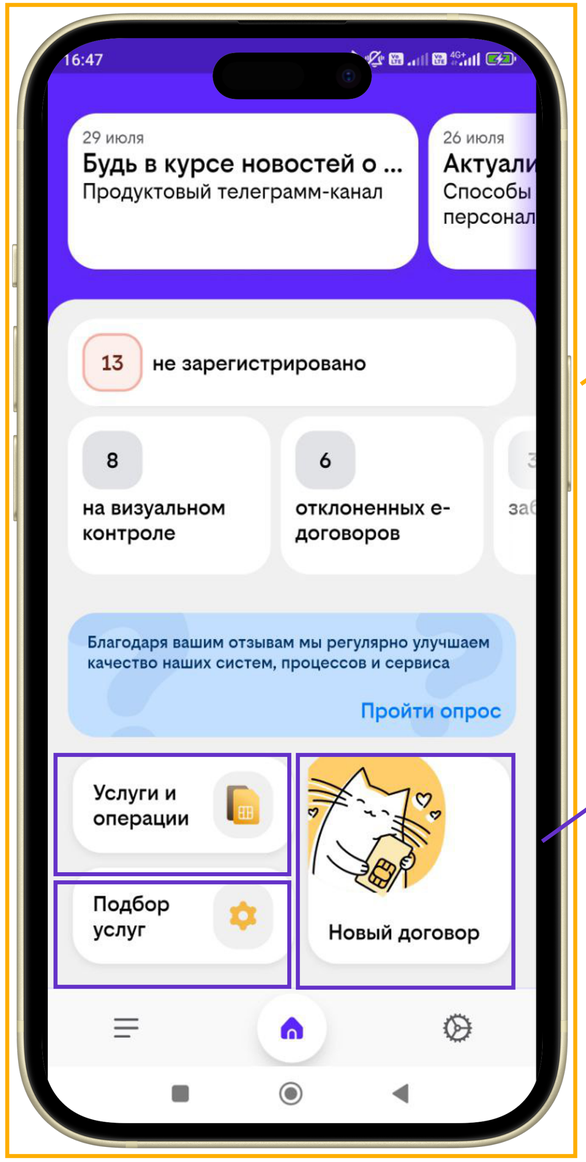




Дилер онлайн (ДОЛ)

Давайте подумаем, как реализовать

Модульная архитектура



ХОСТ ПРИЛОЖЕНИЕ

МОДУЛИ

Услуги и операции

Подключение/отключение услуг

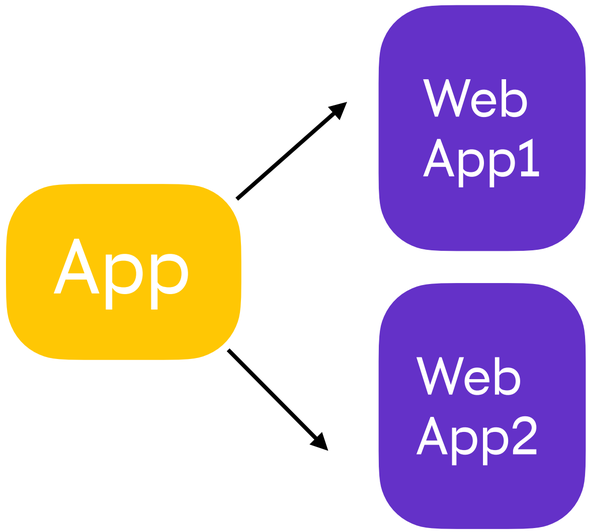
Договоры

Создание и ведение договоров

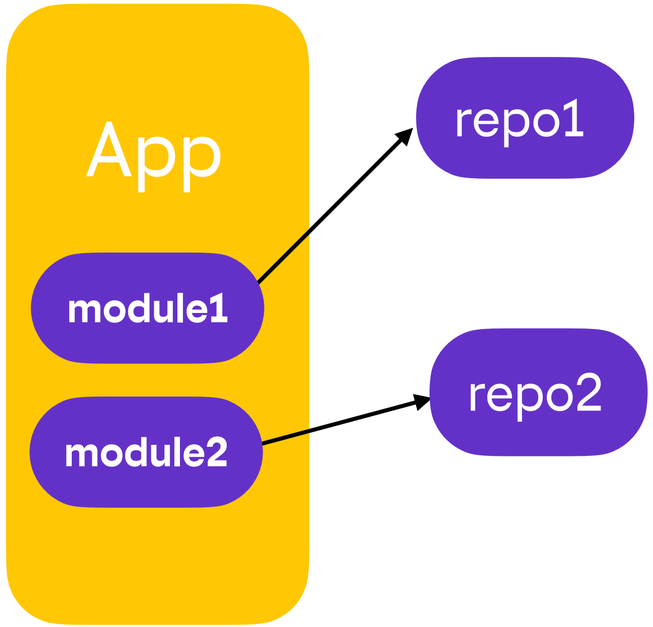
Подбор товаров и услуг

Подбор, анализ продаж

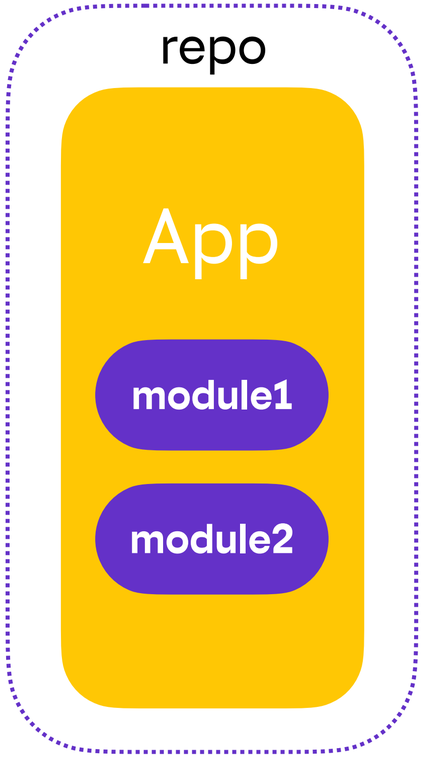
Модульная архитектура



Платформа
с Mini Apps



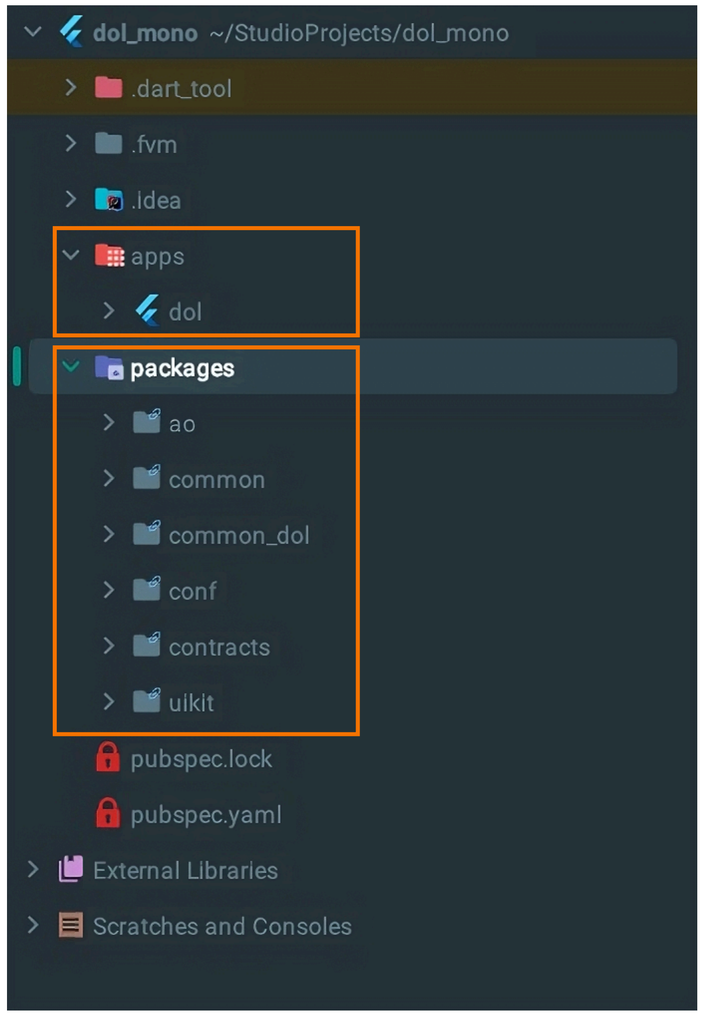
Мультирепозиторий



Монорепозиторий

Как управлять монорепозиторием?

Пробуем простой подход: создаем хост-приложение и модули (packages)



- Ручное получение зависимости
- Ручная кодогенерация
- Нет возможности автоматизации

Melos: альтернатива управления монорепозиторием

Melos - это инструмент, оптимизирующий и автоматизирующий рабочий процесс управления монорепозиториями с помощью git и Pub

melos.yaml - основной конфигурационный файл

```
name: beeline_dol

sdkPath: .fvm/flutter_sdk

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

command:
  bootstrap:
    usePubspecOverrides: true

scripts: ...
```

Ссылка на расположение sdk, используем fvm

Расположение пакетов и приложений

Настройки

Скрипты

Автоматизация Melos

Melos предоставляет возможность создавать и запускать скрипты

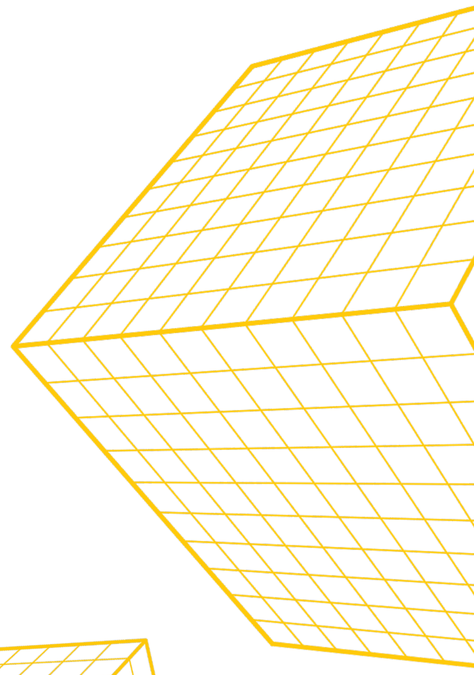
Запуск скриптов происходит с помощью команды **melos run <script>**

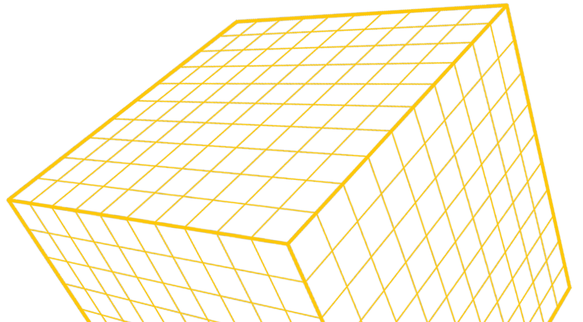
Скрипты описываются в файле melos.yaml

```yaml
name: ...
version: ...

scripts:
  prepare: melos bs && melos run build
  generate:common:
    run: melos exec
    --fail-fast --scope="*common*"
    --flutter
    -- "flutter pub run build_runner build
    --delete-conflicting-outputs"
```

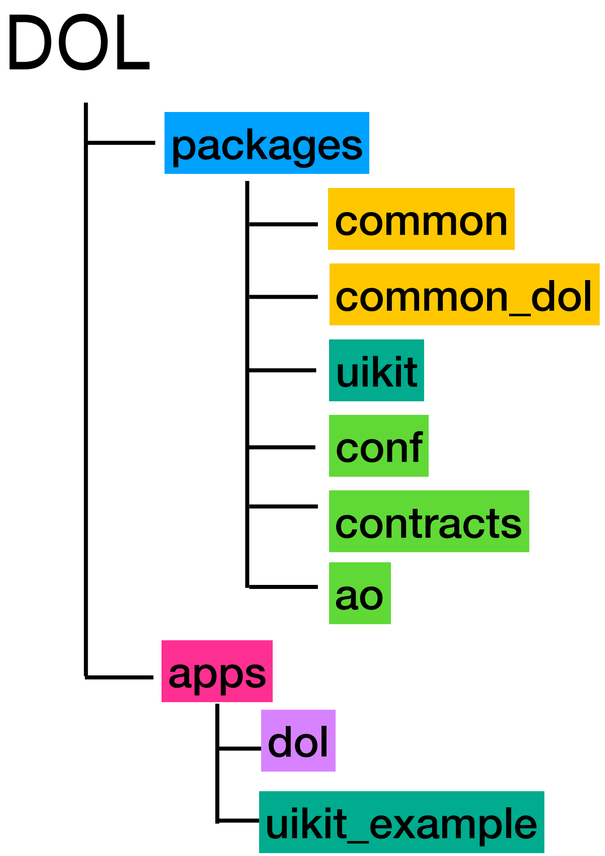
melos.yaml

- Кодогенерация
- Проверка кода
- Запуск тестов
- Сборка и загрузка в стор





Модульная архитектура:



Общие пакеты (модули)

UIKit (компоненты дизайн-системы)

Feature пакеты (модули)

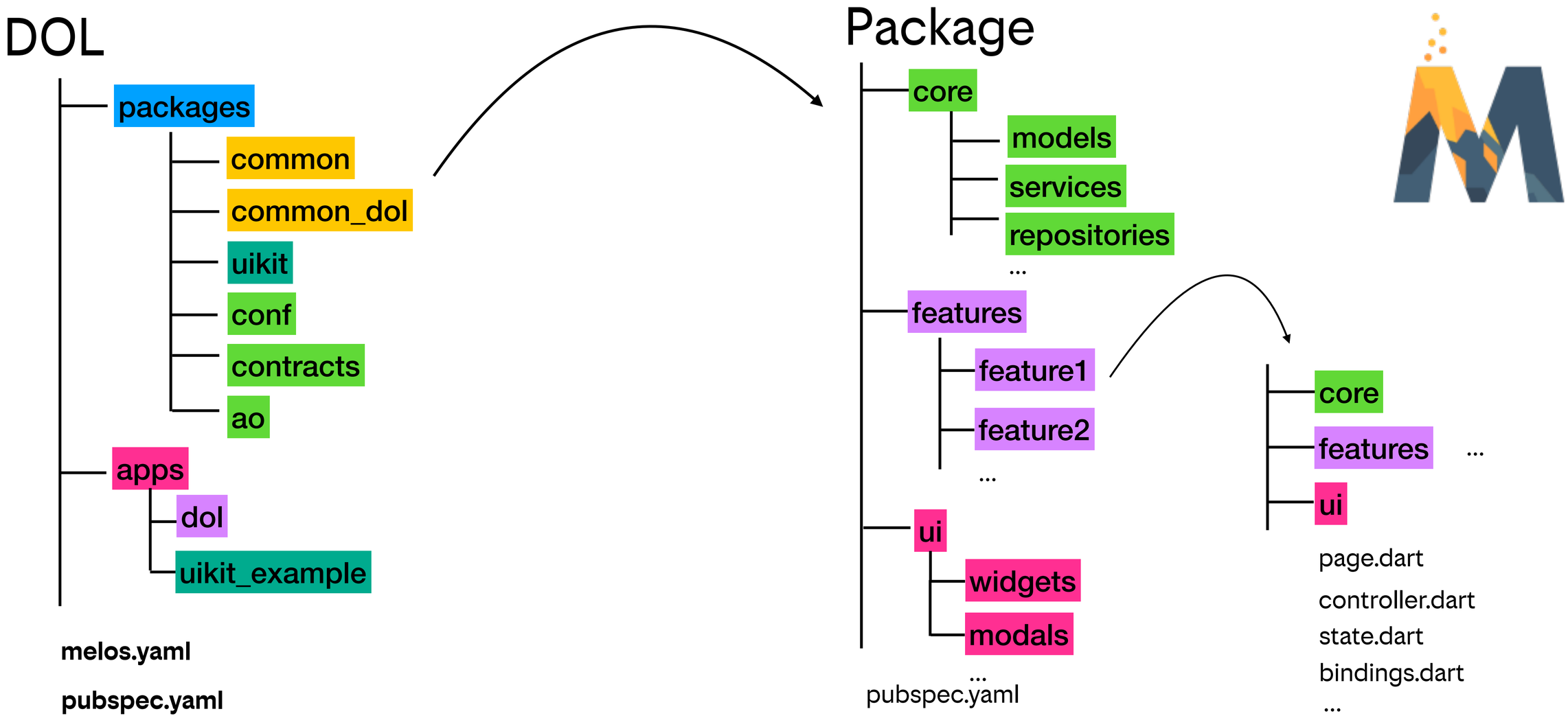
Хост приложение

Example для UIKit

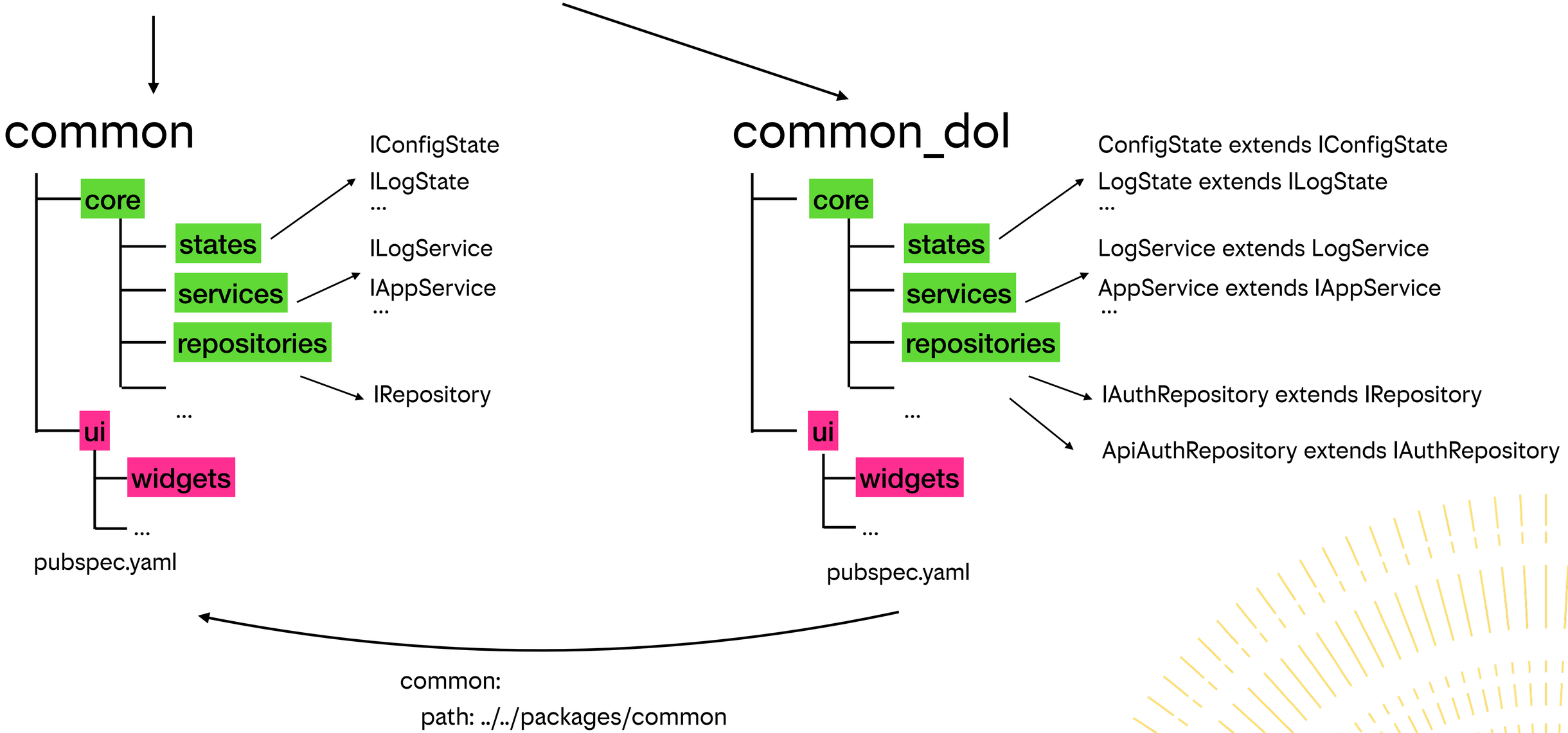
melos.yaml

pubspec.yaml

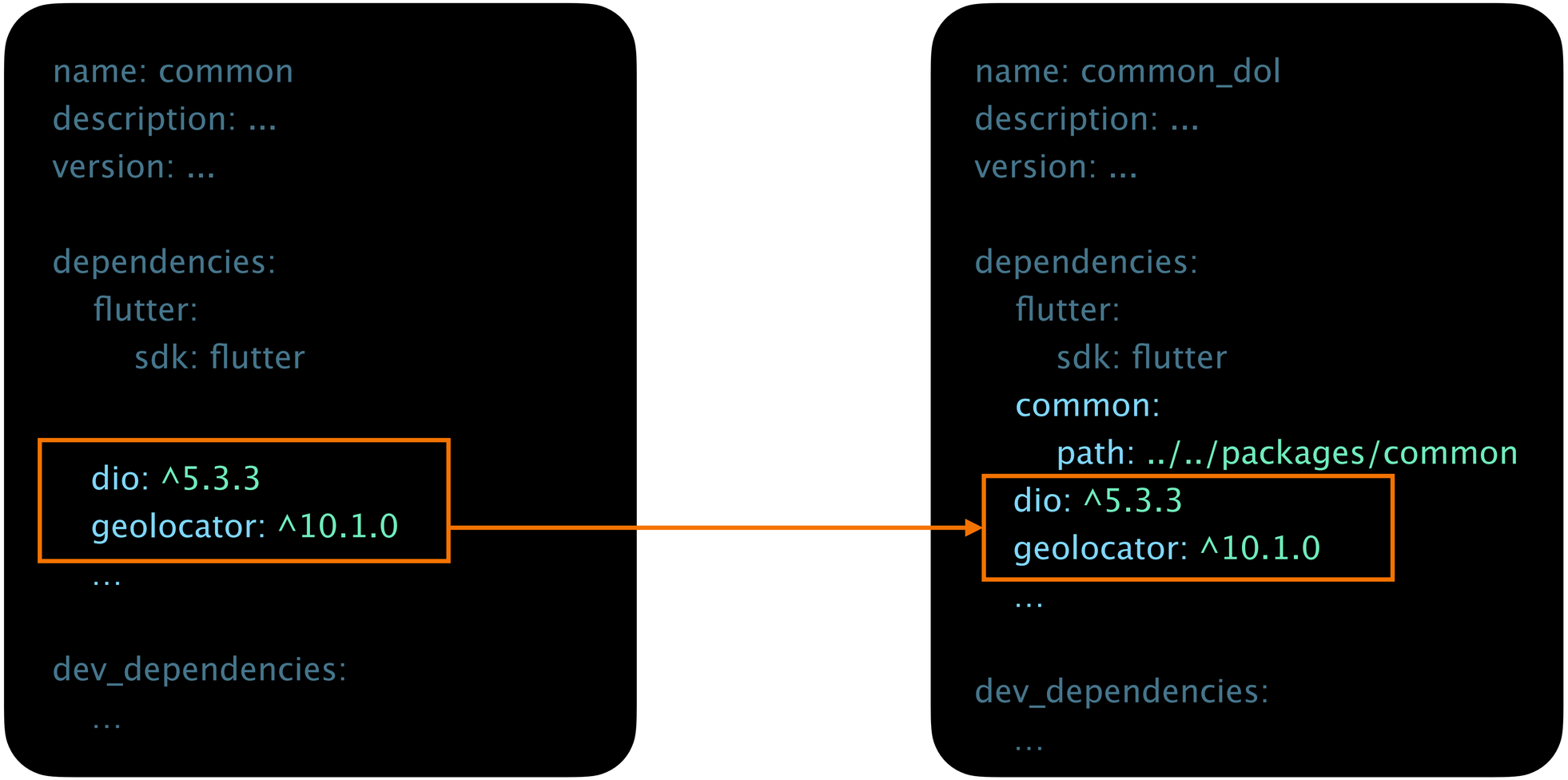
Модульная архитектура



Общие модули



Импорты пакетов

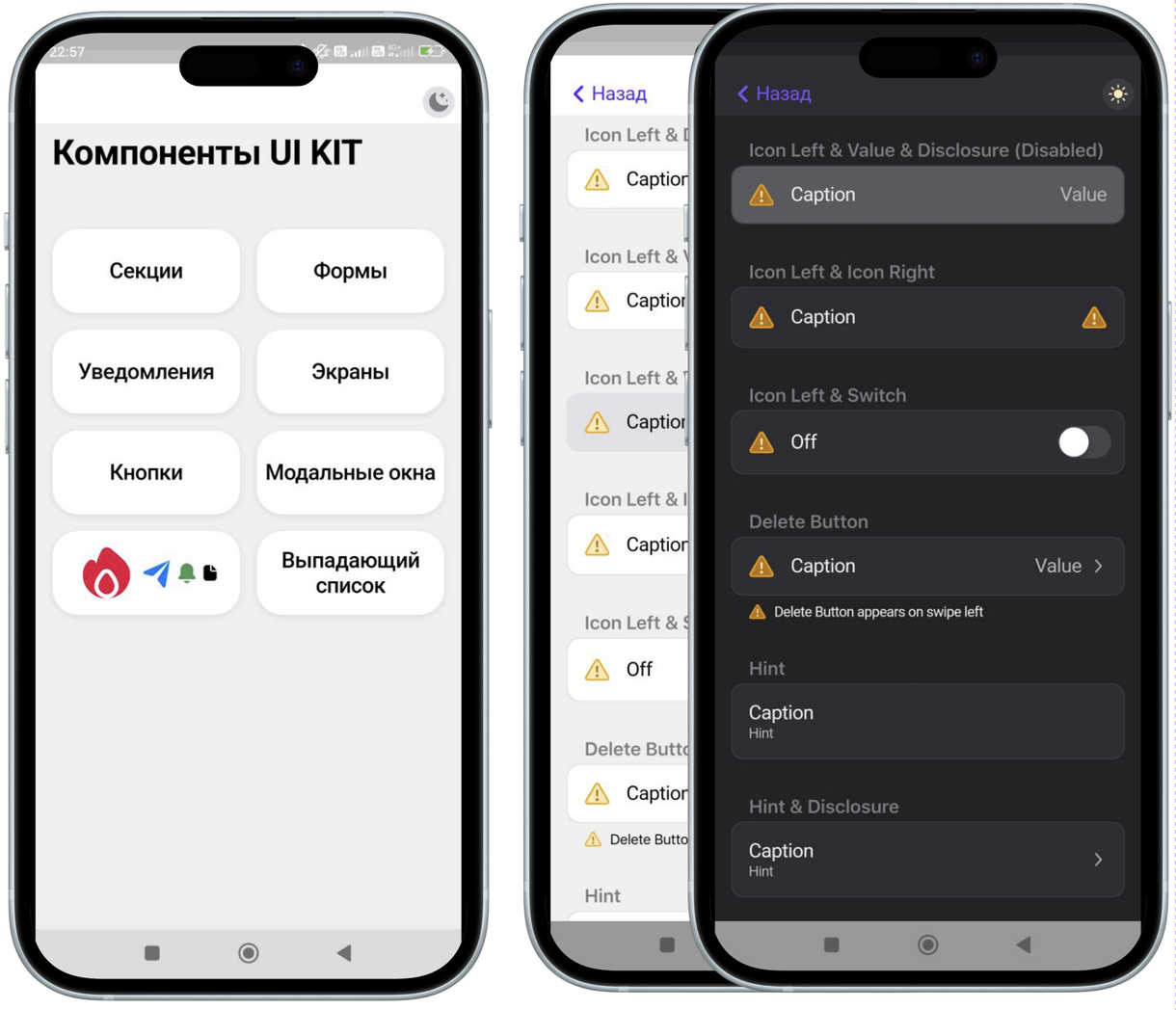


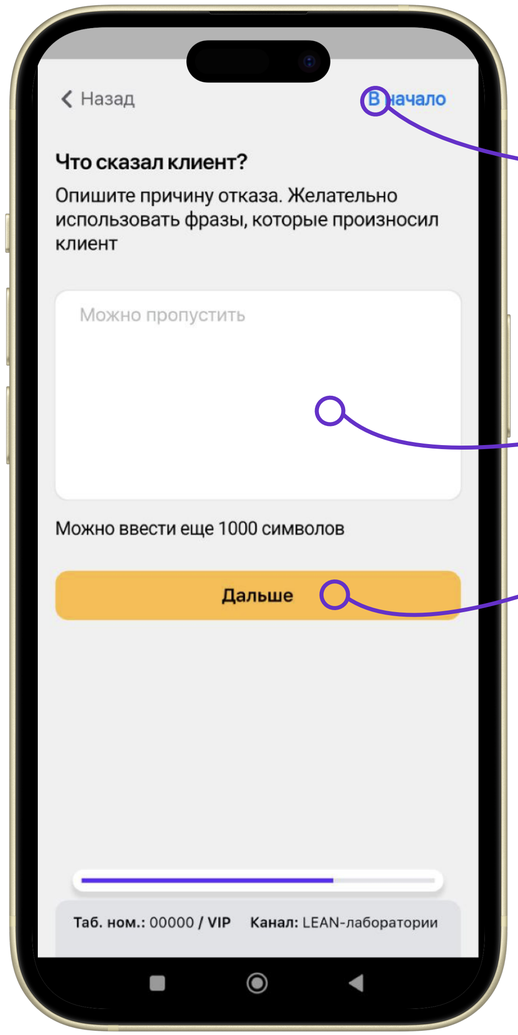
common/pubspec.yaml

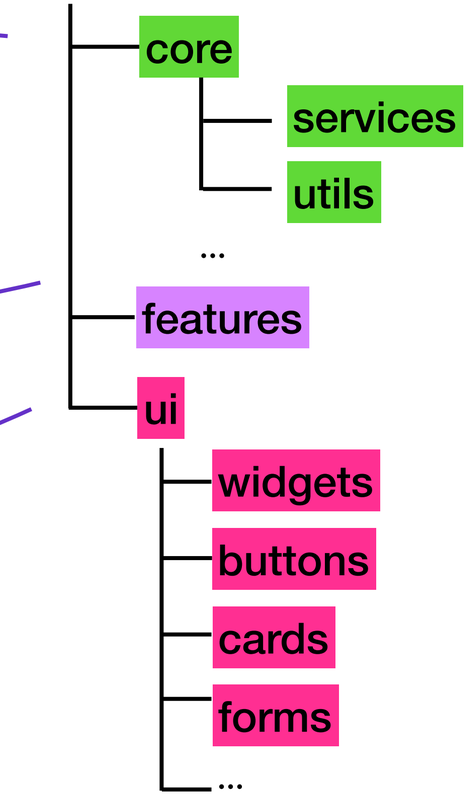
common_dol/pubspec.yaml

UIKit

Компоненты дизайн-системы

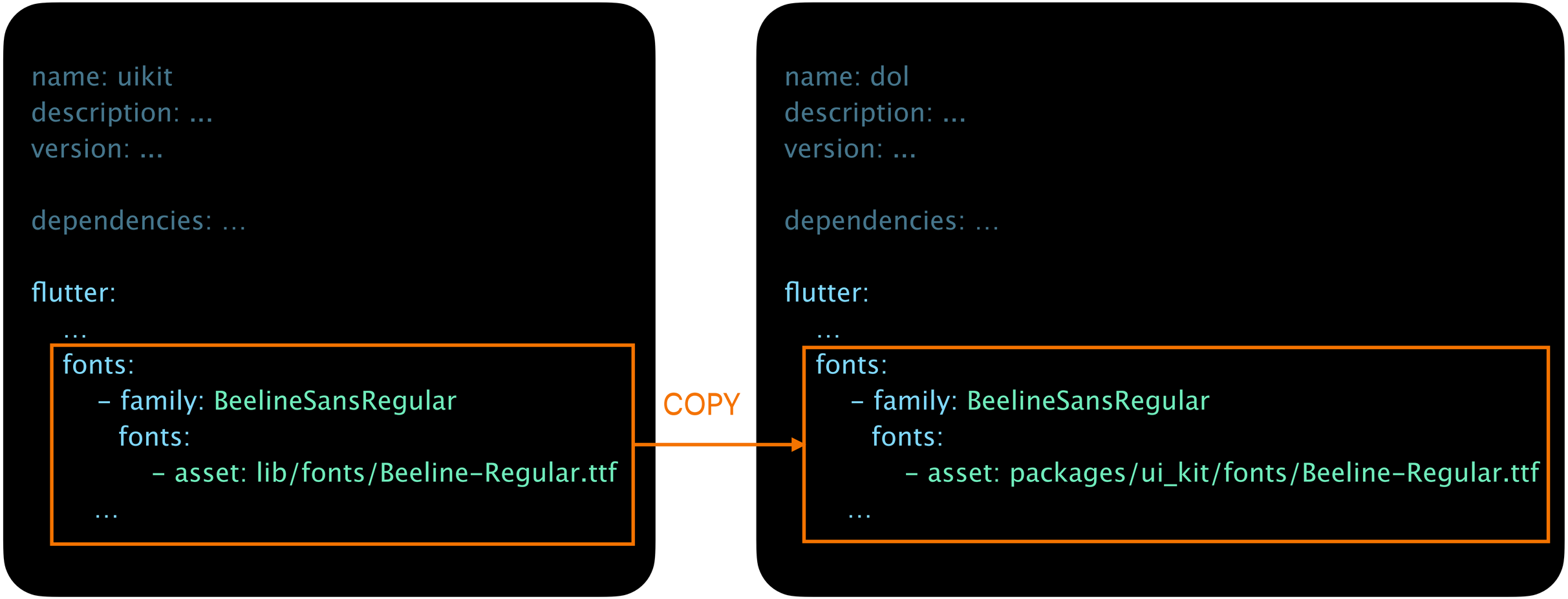






pubspec.yaml

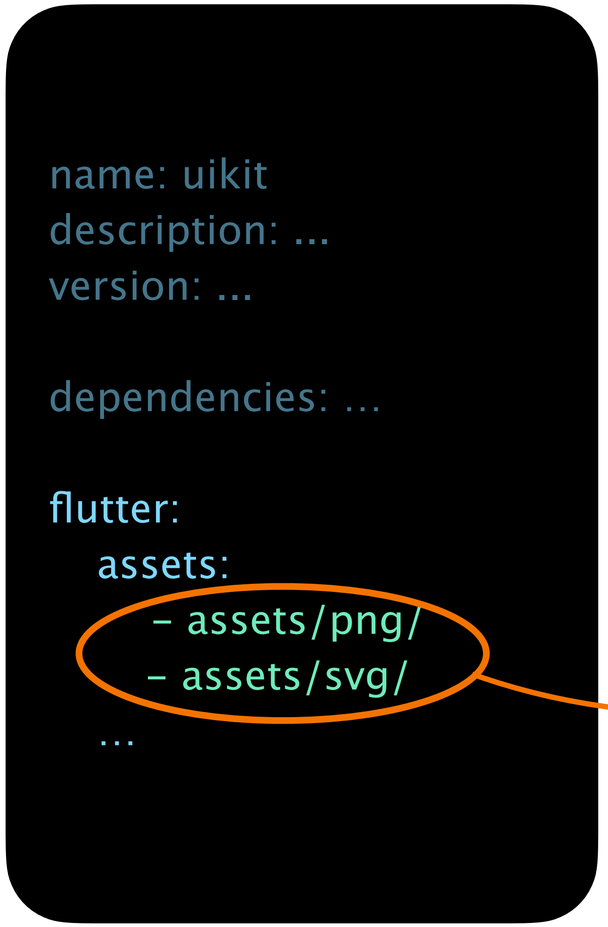
Шрифты



uikit/pubspec.yaml

dol/pubspec.yaml

Изображения, иконки



uikit/pubspec.yaml

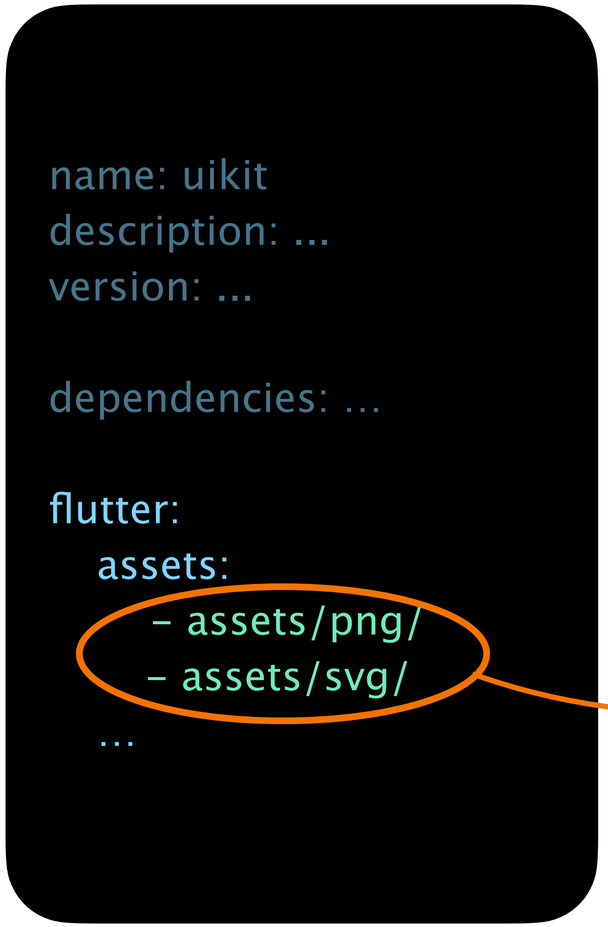
```dart
class UkSvgAssets {
    static String get dangerIconLight
            => _getIcon('icn-danger-light');

    static String get dangerIconDark
            => _getIcon('icn-danger-dark');
    ...

    static String _getIcon(String icon) =>
        'packages/ui_kit/assets/svg/$icon.svg';
    ...
}
```

uikit/uk_svg_assets.dart

```dart
class Widget extends StatelessWidget {
    const Widget({super.key});

    @override
    Widget build(BuildContext context) {
      return SvgPicture.asset(
          UkThemed.of(context).useLight
            ? UkSvgAssets.dangerIconLight
            : UkSvgAssets.dangerIconDark,
          ),
    }
}
```

dol/widget.dart

Изображения, иконки



```yaml
name: uikit
description: ...
version: ...

dependencies: ...

flutter:
  assets:
    - assets/png/
    - assets/svg/
  ...
```

uikit/pubspec.yaml

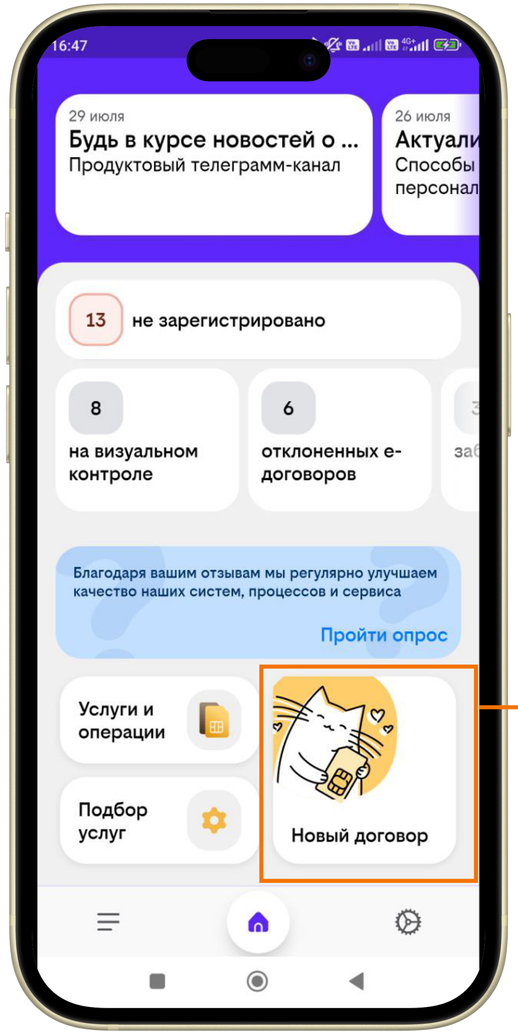
```dart
class Widget extends StatelessWidget {
  const Widget({super.key});

  @override
  Widget build(BuildContext context) {
    return SvgPicture.asset(
      UkThemed.of(context).useLight
        ? 'assets/svg/icn-danger-light.svg'
        : 'assets/svg/icn-danger-dark.svg',
      package: 'ui_kit',
    );
  }
}
```

dol/widget.dart

```dart
class Widget extends StatelessWidget {
  const Widget({super.key});

  @override
  Widget build(BuildContext context) {
    return Image.asset(
      UkThemed.of(context).useLight
        ? 'assets/png/icn-danger-light.png'
        : 'assets/png/icn-danger-dark.png',
      package: 'ui_kit',
    );
  }
}
```

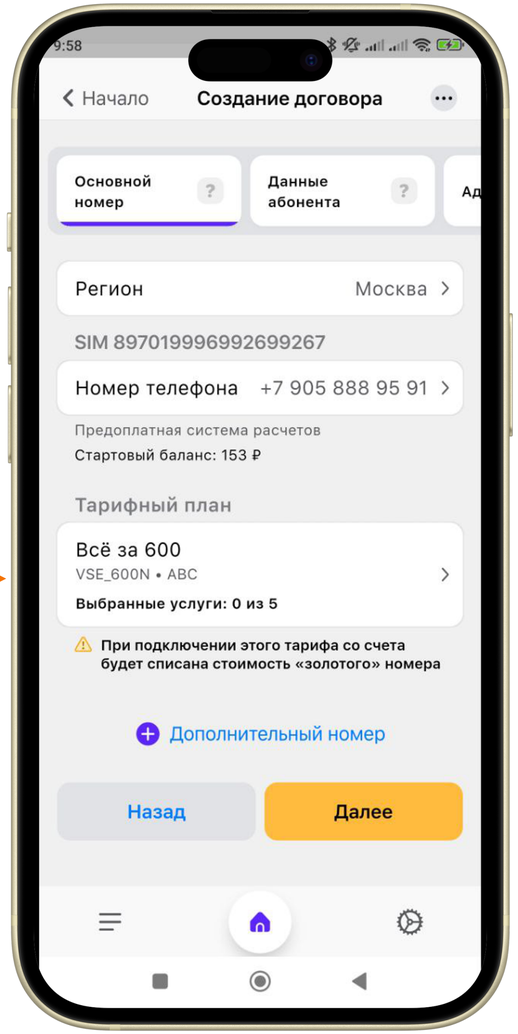
dol/widget.dart

Feature модули



contracts

- core
  - states
    - ContractState
    - ...
  - repositories
    - IContractsRepository extends IRepository
    - ApiContractsRepository extends IContractsRepository
    - ...
  - ...
- features
  - edit-contract
    - core
    - ui
      - edit-contract-page.dart
      - edit-contract-controller.dart
      - ...
  - ...
- pubspec.yaml



Dol App

DOL

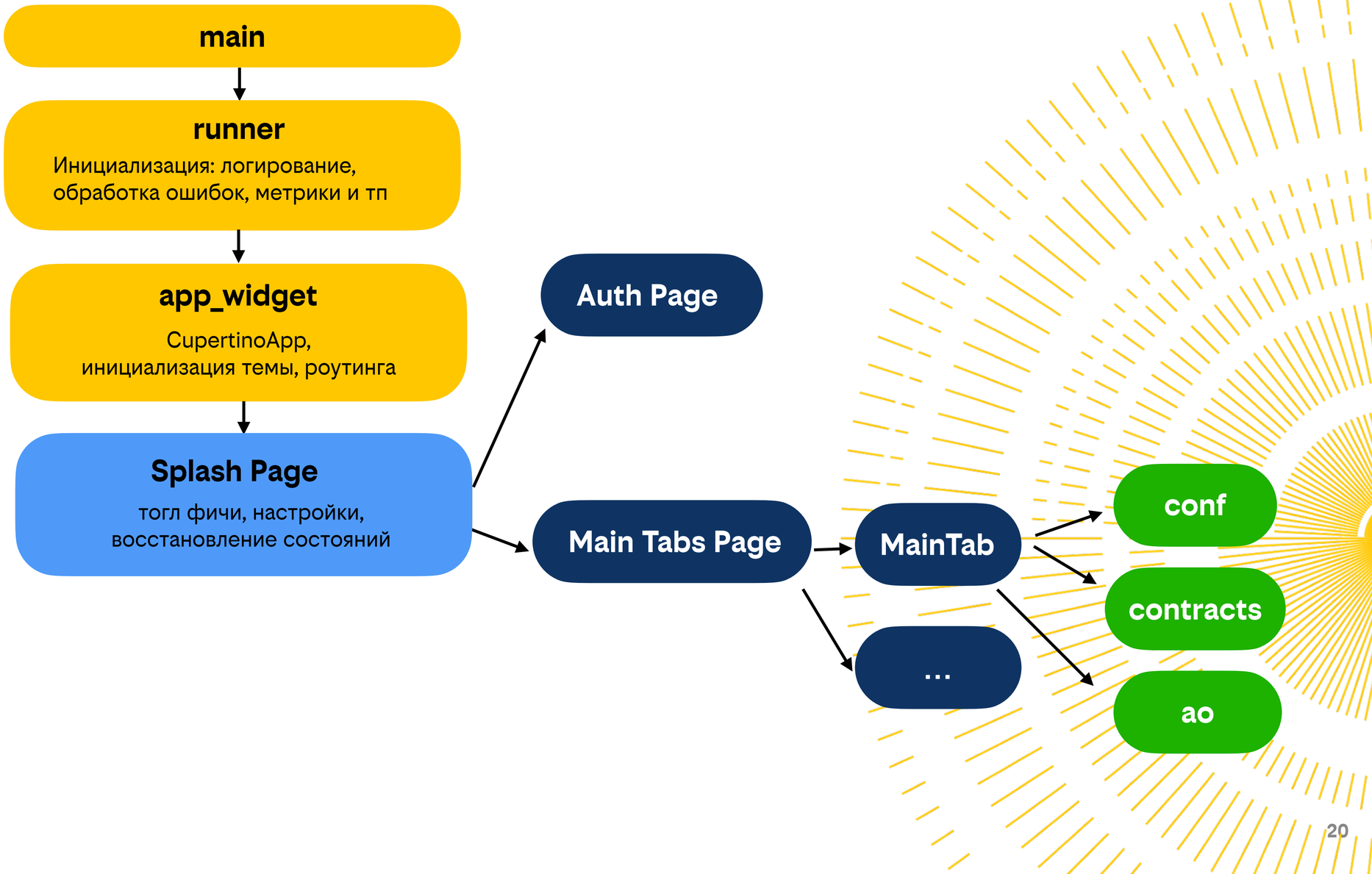
- core
  - states
  - models
  - repositories
  - ...
- features
- ui
  - widgets
  - ...

app.dart

main.dart

runner.dart

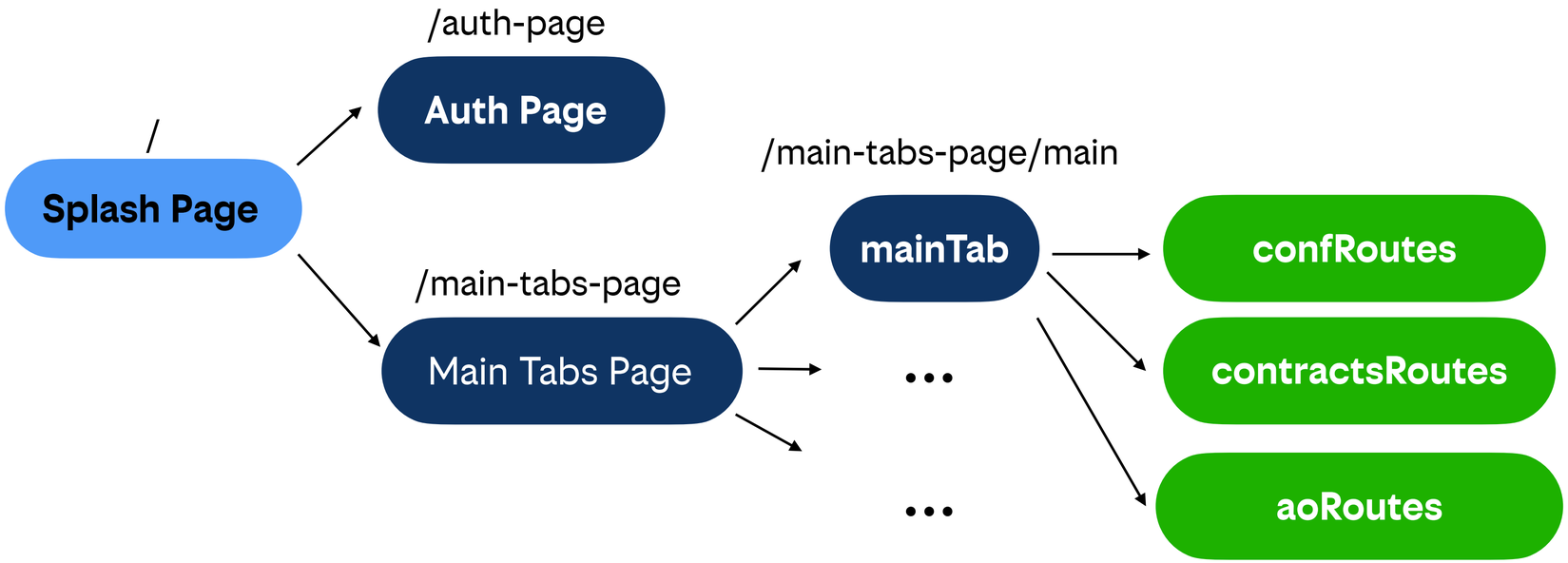
pubspec.yaml



Навигация

AUTOROUTE

```
@AutoRouterConfig(
  replaceInRouteName: 'Page,PageRoute',
  modules: [
    ConfRouter,
    ContractsRouter,
    AoRouter,
  ],
)
```

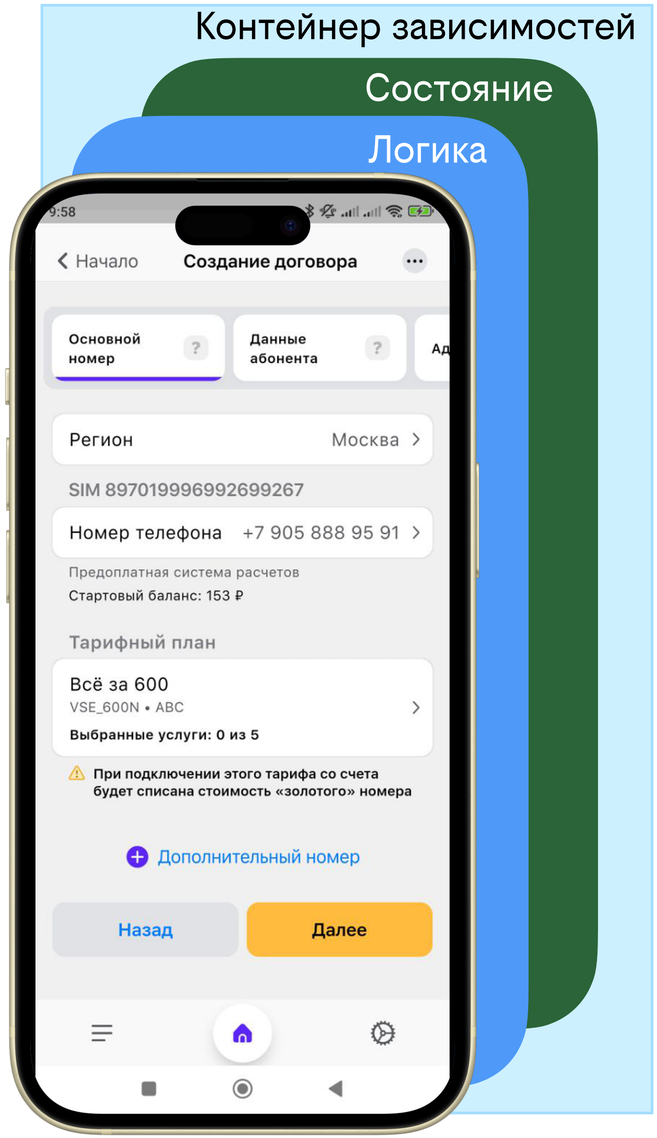


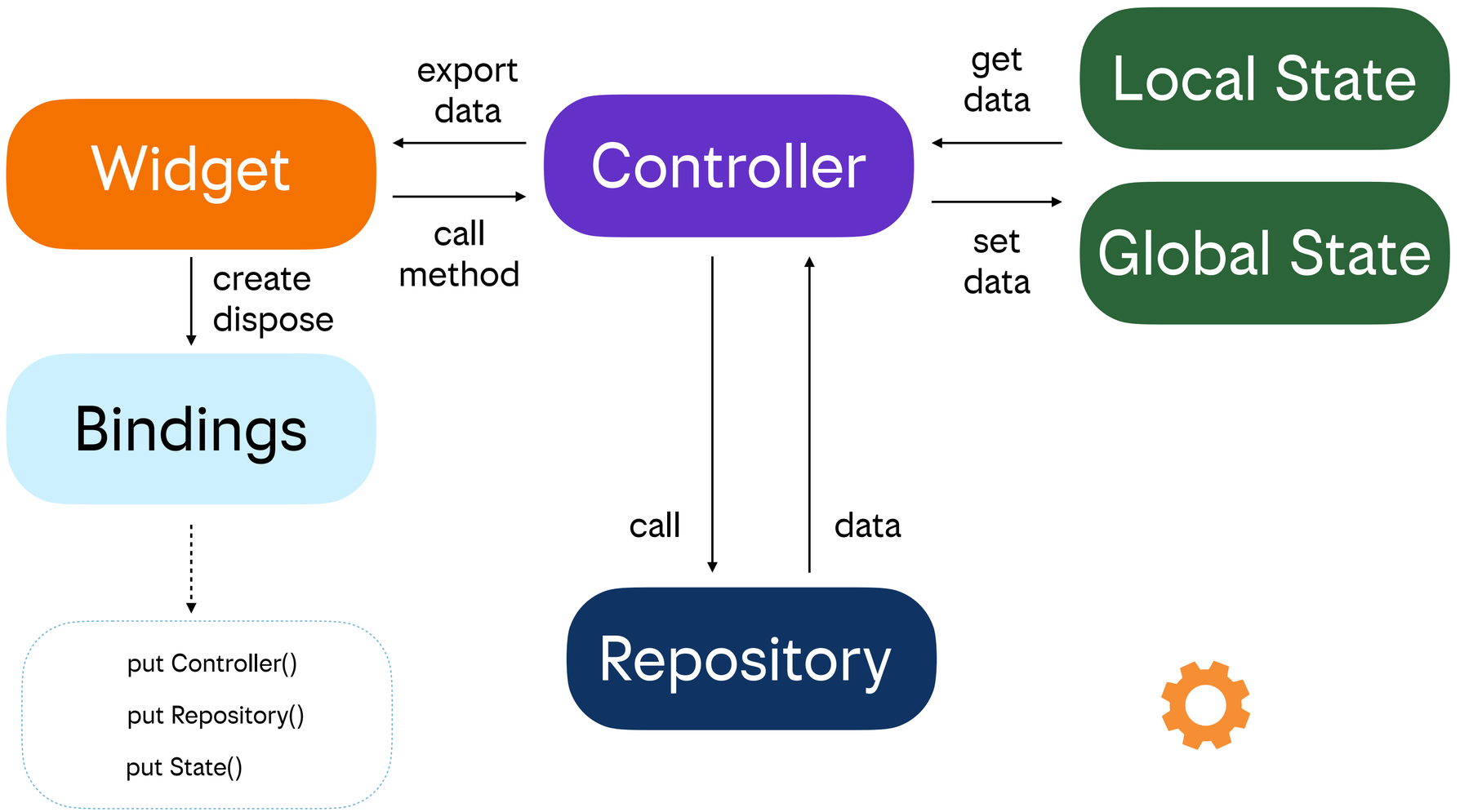
Router config

```
class AppRouter extends _$AppRouter {
  @override
  List<AutoRoute> get routes => [
        CupertinoRoute(
          path: Navigator.defaultRouteName,
          page: SplashPageRoute.page,
          initial: true,
        ),
        CupertinoRoute(
          path: AuthPage.routeName,
          page: AuthPageRoute.page,
        ),
        CupertinoRoute(
          path: MainTabsPage.routeName,
          page: MainTabsPageRoute.page,
          children: [
            ...,
            ...confRoutes,
            ...contractsRoutes,
            ...aoRoutes,
          ]
        ),
    ];
}
```

Углубимся в структуру экрана

Структура экрана





Контейнер зависимостей

Bindings

put Controller()

put Repository()

put State()

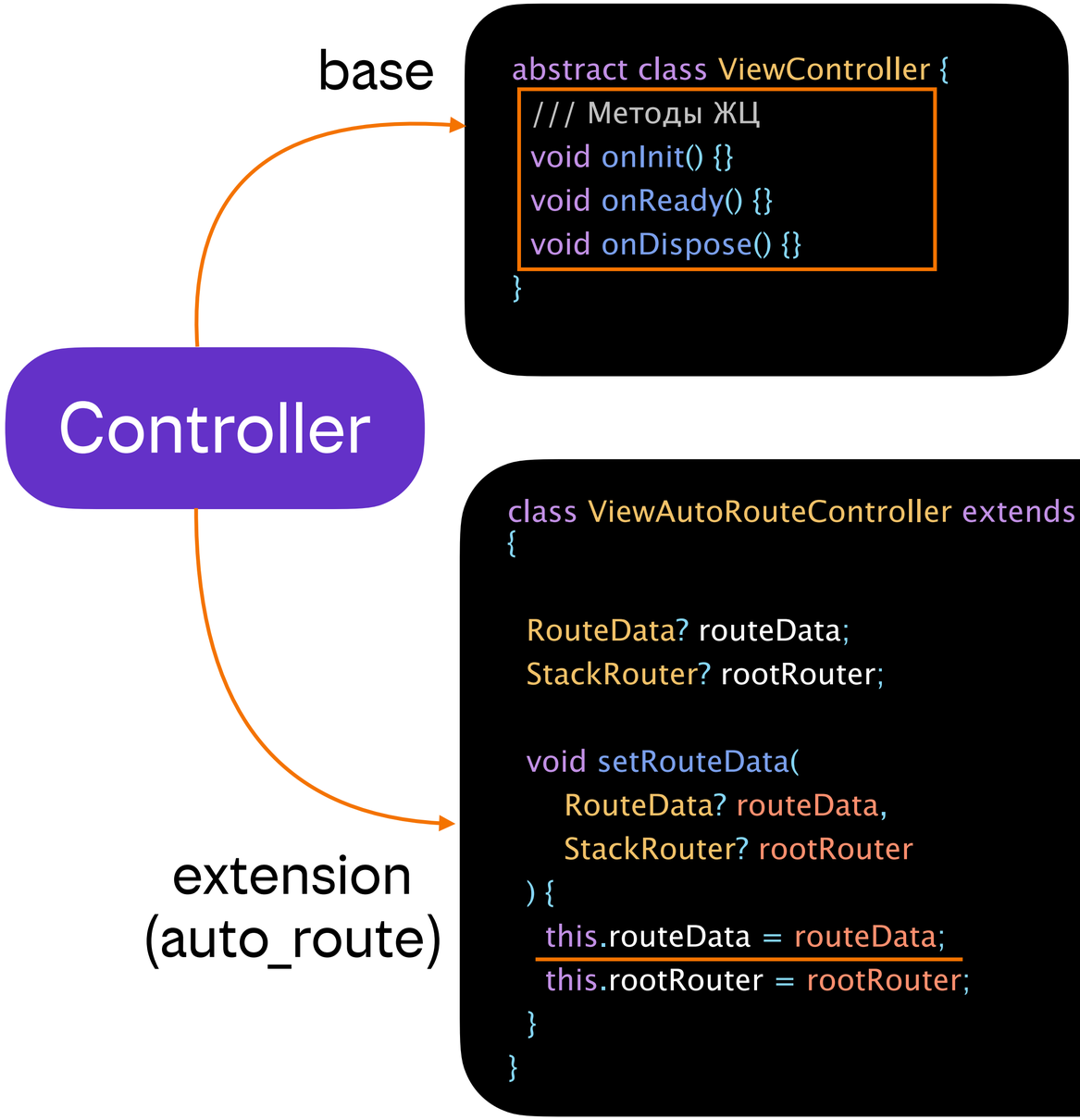
```
abstract class ViewBindings {
  final List<_Binding> _bindings = [];

  /// Создаем новый контейнер
  final getIt = GetIt.asNewInstance();

  /// Ссылка на родительский контейнер
  ViewBindings? parentBindings;

  setParentBindings(ViewBindings? viewBindings) {
    parentBindings = viewBindings;
  }

  /// Методы регистрации и удаления
  void register();
  void unregister() {
    ...
  }

  /// Метод поиска зависимости
  T get<T extends Object>({
    ...
  }) { }

  ...
}
```

из пакета GetIt

```
class AuthPageBinding extends ViewBindings {
  @override
  void register() {
    registerSingleton(AuthPageState());
    registerSingleton(
      AuthPageController(
        get<ConfigState>(),
        get<AppState>(),
        get<AuthPageState>(),
        ....
      ),
    );
  }
}
```

```
class AuthPageState {
  final login = RxNotifier<String?>(null);
}
```

ViewController: контроллер (логика)

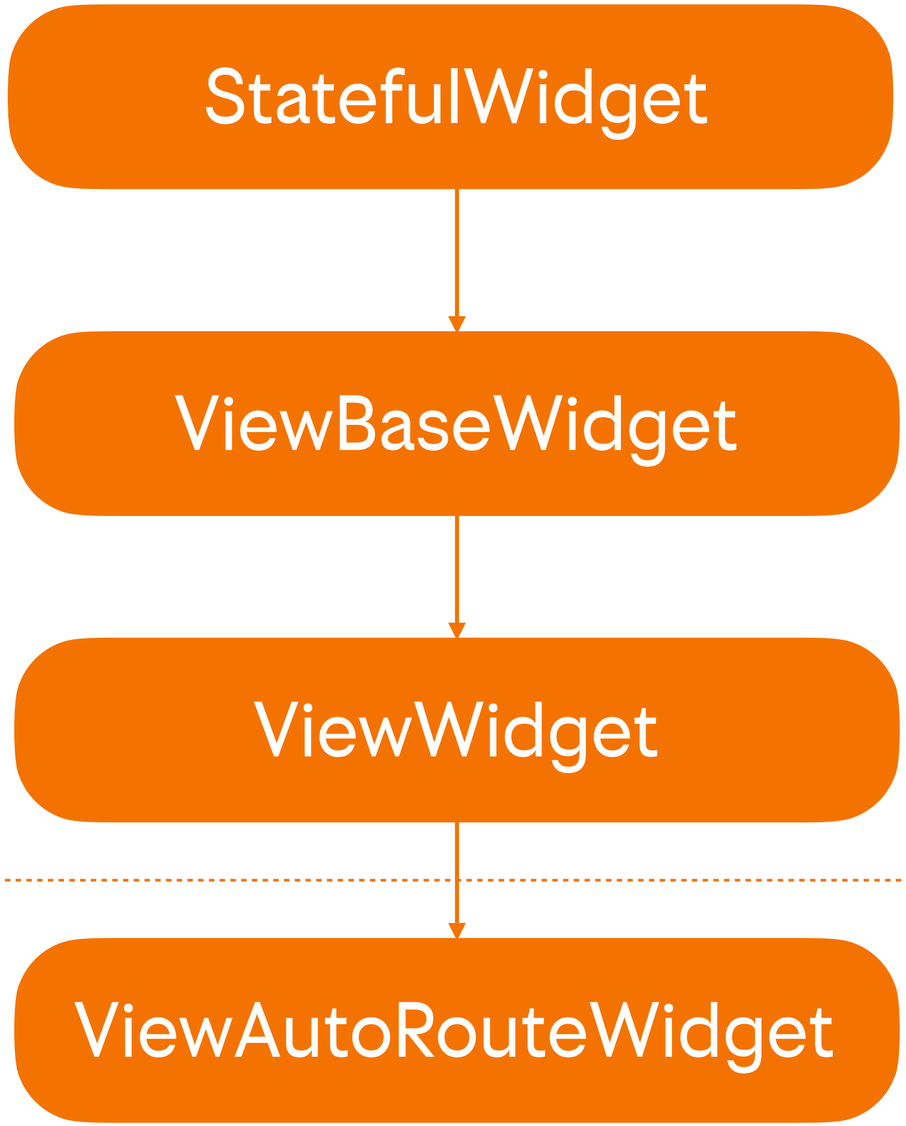



```dart
abstract class ViewController {
  /// Методы ЖЦ
  void onInit() {}
  void onReady() {}
  void onDispose() {}
}
```

```dart
class ViewAutoRouteController extends ViewController
{

  RouteData? routeData;
  StackRouter? rootRouter;

  void setRouteData(
      RouteData? routeData,
      StackRouter? rootRouter
  ) {
    this.routeData = routeData;
    this.rootRouter = rootRouter;
  }
}
```

```dart
class AuthPageController extends ViewController {

  /// Внедренные зависимости
  final ConfigState _configState;
  final AppState _appState;
  final SettingsState _settingsState;

  AuthPageController(
    this._configState,
    this._appState,
    this._settingsState,
  ) {}

  /// Экспортируем необходимые View данные
  String? get login => _state.login.value;

  /// Методы
  Future<void> logon({
    ...
  });
}
```

Widget: классификация



Стандартный виджет с состоянием

+ DI контейнер (bindings)

+ контроллер

extensions

+ интеграция с auto_route (данные роута)

ViewBaseWidget: представление

```dart
abstract class ViewBaseWidget extends StatefulWidget {
  /// Возможность задать контейнеры вручную
  final ViewBindings? parentBindings;
  final ViewBindings? bindings;

  const ViewBaseWidget({
      Key? key,
      this.parentBindings,
      this.bindings,
  }) : super(key: key);

  @override
  State<ViewBaseWidget> createState() => ViewBaseWidgetState();

  /// Переопределяемый метод создания контейнера
  ViewBindings createBindings();
}

class ViewBaseWidgetState<TWidget extends ViewBaseWidget>
    extends State<ViewBaseWidget> {

     ....

 }
```

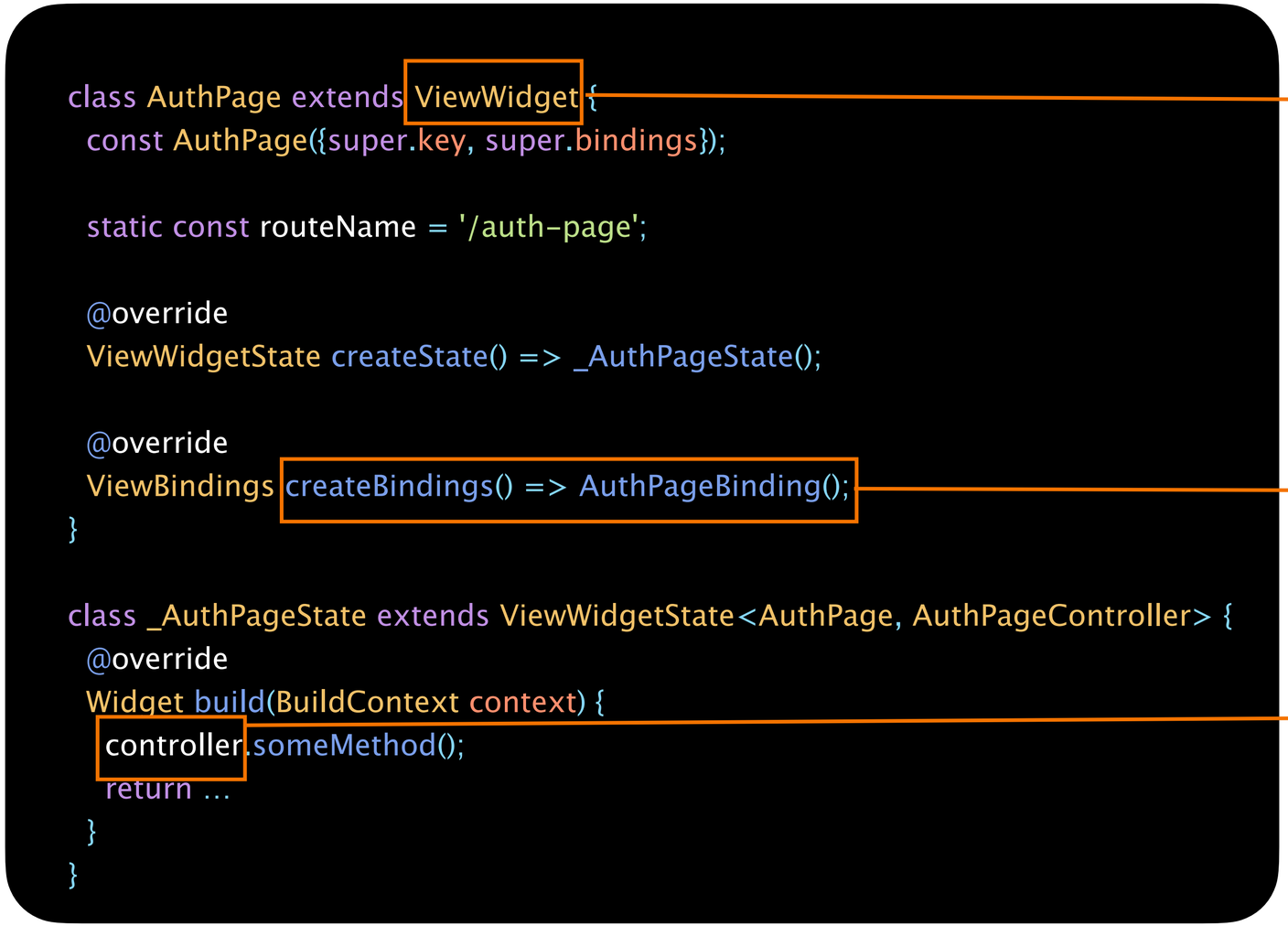
```dart
@override
 void initState() {
   super.initState();

  /// Определяем родительский контейнер
  final parentBindings = widget.parentBindings ??
     context
       .findAncestorStateOfType<ViewBaseWidgetState>()
      ?.bindings;

  /// Определяем текущий контейнер
  bindings = widget.bindings ?? widget.createBindings();

   /// Устанавливаем ссылку на родит. контейнер
  bindings.setParentBindings(parentBindings);

  /// Регистрируем зависимости
  bindings.register();
 }

 @override
 Widget build(BuildContext context) {
    return const Placeholder();
 }

 @override
 void dispose() {
   /// Уничтожаем зависимости
   bindings.unregister();
   super.dispose();
 }
```

ViewWidget: представление

```dart
abstract class ViewWidget extends ViewBaseWidget {
     const ViewWidget({
      super.key,
      super.parentBindings,
      super.bindings,
     });

    @override
    ViewWidgetState createState() => ViewWidgetState();
}

class ViewWidgetState<TWidget extends ViewWidget,
   TController extends ViewController>
extends ViewBaseWidgetState<TWidget> {

  late TController controller;

   ...
}
```

```dart
@override
void initState() {
  super.initState();

  /// Формируем ссылку на контроллер
  controller = bindings.get<TController>();

 /// Реализуем метод ЖЦ onInit
  controller.onInit();

 /// Реализуем метод ЖЦ onReady
  SchedulerBinding.instance.addPostFrameCallback(
    (_) => controller.onReady(),
  );
}

@override
Widget build(BuildContext context) {
  return const Placeholder();
}

@override
void dispose() {
  /// Реализуем метод ЖЦ onDispose
  controller.onDispose();
  super.dispose();
}
```

Пример View

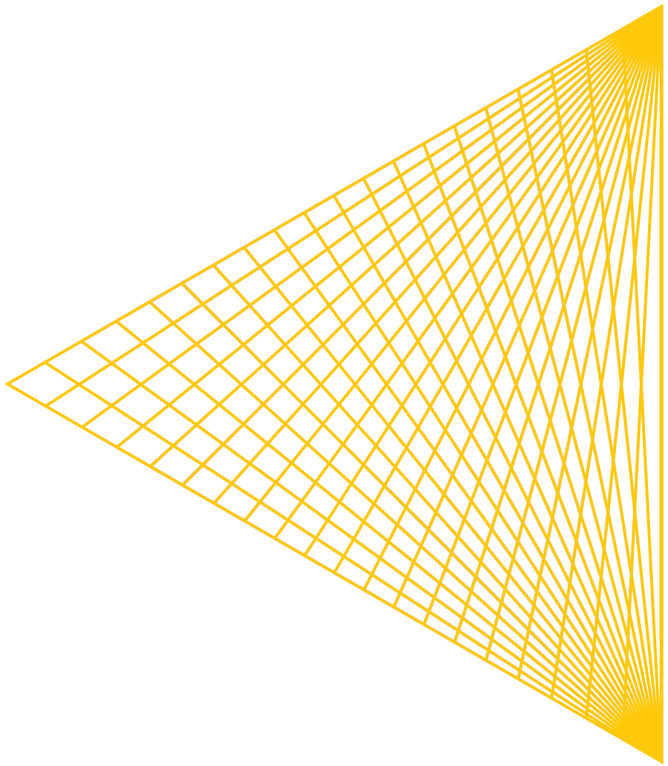


```
class AuthPage extends ViewWidget {
  const AuthPage({super.key, super.bindings});

  static const routeName = '/auth-page';

  @override
  ViewWidgetState createState() => _AuthPageState();

  @override
  ViewBindings createBindings() => AuthPageBinding();
}

class _AuthPageState extends ViewWidgetState<AuthPage, AuthPageController> {
  @override
  Widget build(BuildContext context) {
    controller.someMethod();
    return ...
  }
}
```

Обертка над **StatefulWidget**, инкапсулирует **создание bindings**, **получение controller** из bindings

Реализуем метод создания bindings

Ссылка на controller, можем выполнять различные методы



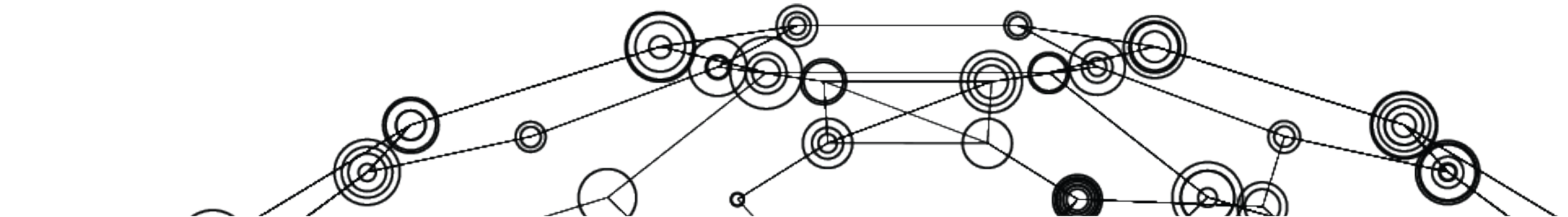
Что в итоге?


Обработка ошибок
Логирование
i18n
Управление состоянием
Управление зависимостями
Темизация
Структура проекта
Документация
Модульность
Навигация
CI/CD
Тестирование
Автоматизация разработки с ИИ
Отладка
Мониторинг

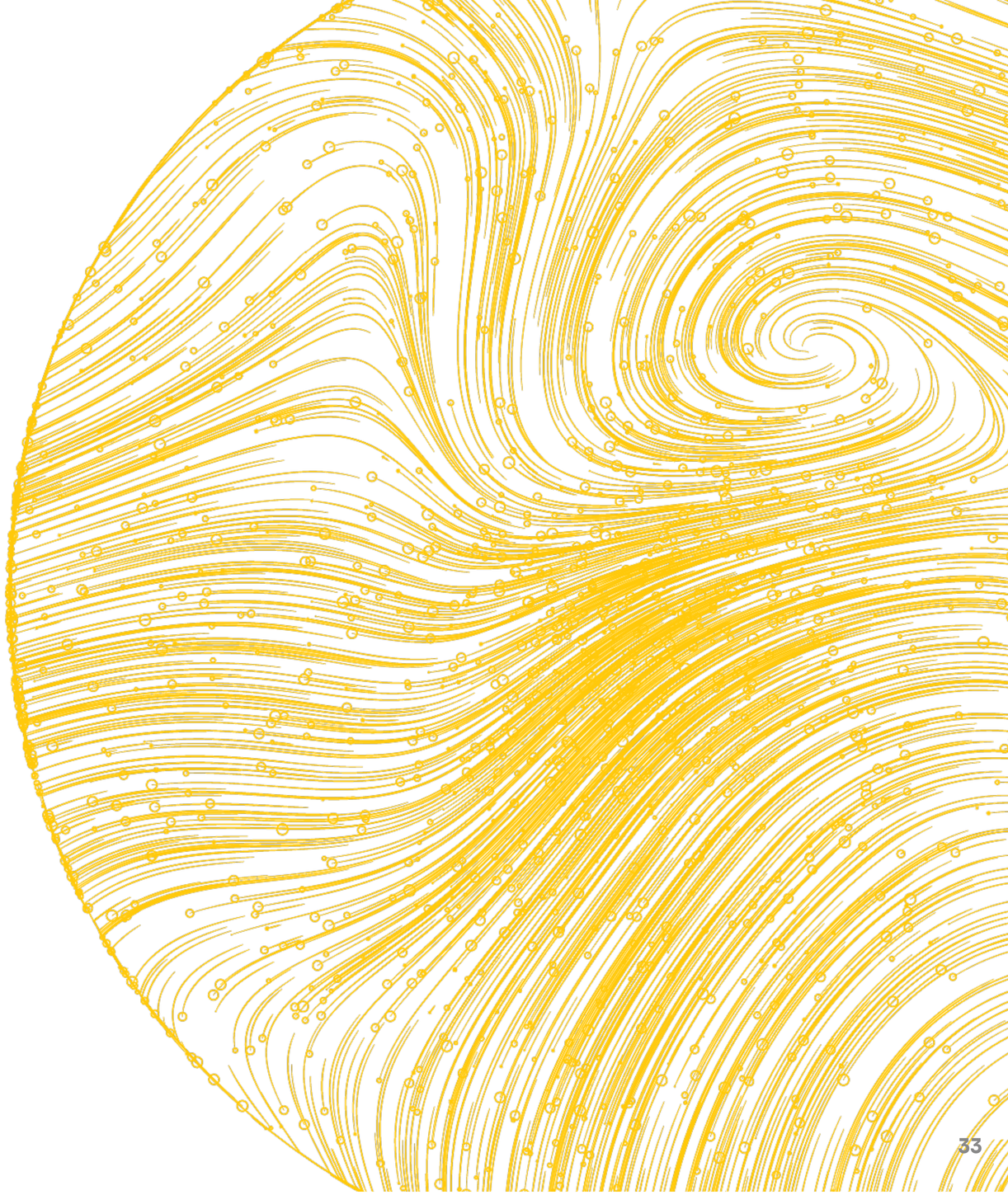
Итоги:

Если перед вами стоит задача продумать архитектуру, реализовать суперапп приложение, то:

- необходимо подойти к реализации модульно, выделить общие модули
- обеспечить понятную, гибкую и расширяемую структуру модуля
- продумать навигацию
- отделять представление, логику, данные
- позаботиться о связывании компонентов в единое целое
- автоматизировать рутинные операции



Вопросы?



Спасибо!

Олег Скирюк
Frontend Team Lead
@skiryukoleg