Архитектура White Label на Flutter

ТОНЕМ В АБСТРАКЦИЯХ

Кто говорит?

Марк Абраменко @ Surf

1. **Engineering Manager** в команде Flutter в Surf
2. **Разработчик**: больше 4 лет во Flutter, бэкграунд на Android и React Native
3. С 2022 помогаю делать **Flutter Dev Podcast**

Для кого этот доклад?

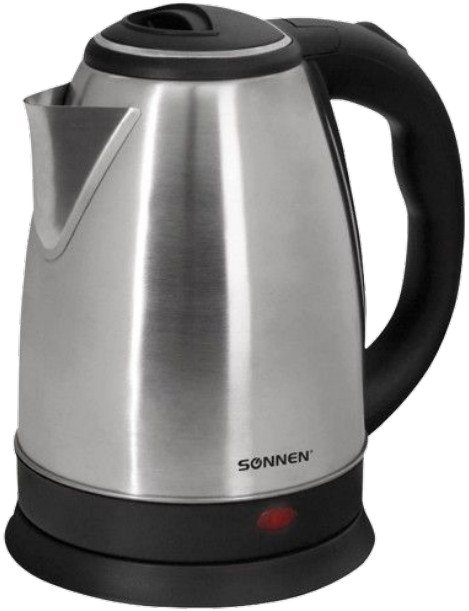
1. **Кто задумывается** о разработке white label, но не знает, с чего начать
2. **Кто уже начал писать** white label на Flutter и хочет узнать, “какие подводные”
3. Да и просто для тех, кто пишет **кастомизируемые приложения** на Flutter

White Label

Создание типового продукта, который позволяет сократить ресурсы на разработку ценой гибкости

White Label

Создание **типового** продукта, который позволяет сократить ресурсы на разработку ценой гибкости


SONNEN


SONNEN


HOMESTAR


SONNEN


Zapa
1971

White Label

Создание типового продукта, который позволяет **сократить ресурсы** на разработку ценой гибкости


HOMESTAR


SONNEN


Zapa
1971

Сокращение ресурсозатрат

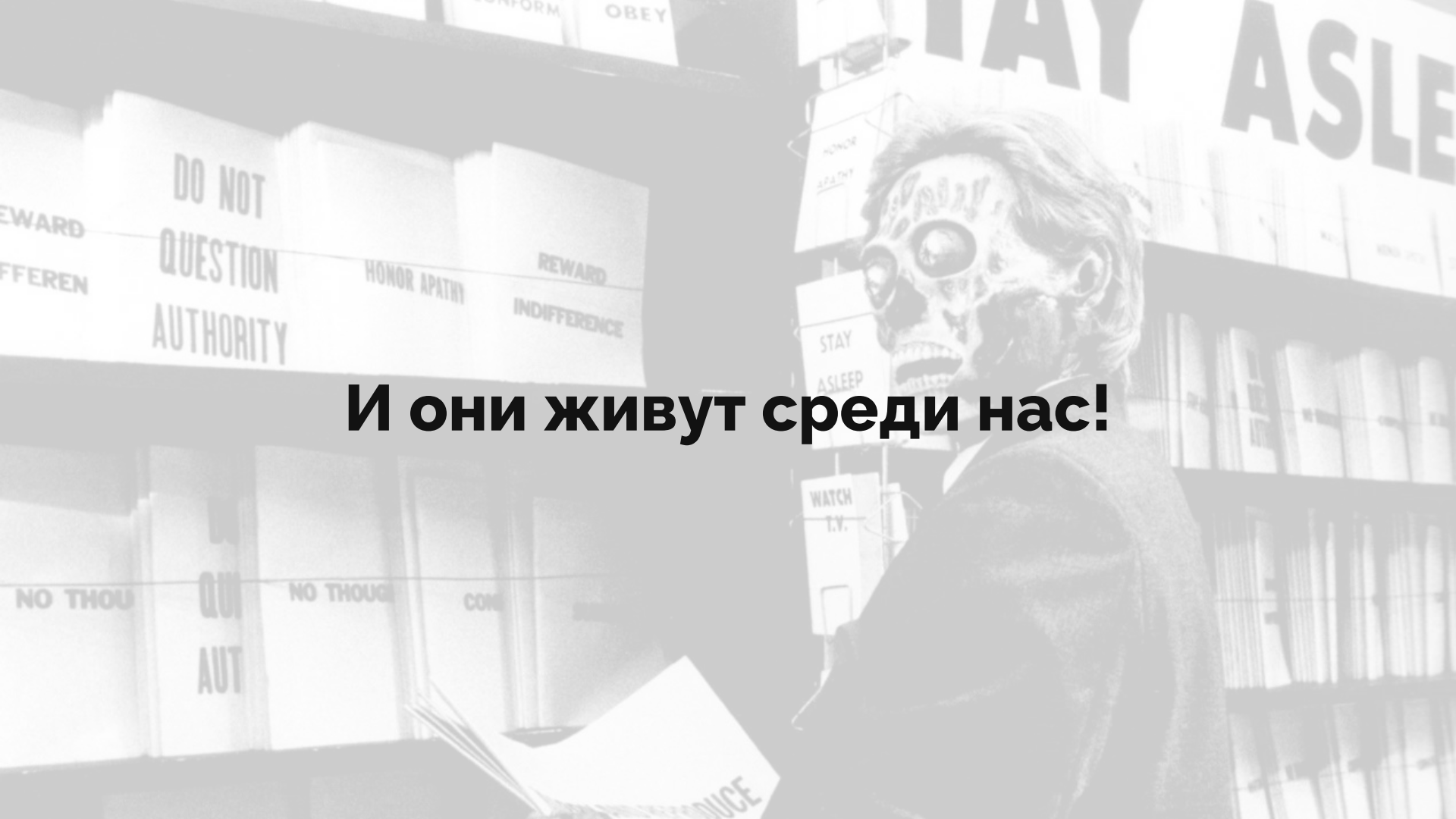
1. Они производятся на одном заводе
2. Из одних и тех же деталей
3. Они проектировались только один раз

White Label

Создание типового продукта, который позволяет сократить ресурсы на разработку ценой **гибкости**

Отсутствие гибкости

1. Компания может поместить на чайник свой логотип
2. Возможно, компания может поменять цвет пластика
3. Промдизайн, металл, электронику, новые фичи — вряд ли



И они живут среди нас!

Кто же это?

1. **Букинги** чего угодно — отелей, билетов
2. **Рестораны** и **кафе**
3. Приложения для трамваев в разных городах уж слишком сильно похожи
4. Корпоративные приложения: фитнес, программа лояльности

Как к этому относятся сторы?

Не существует явных запретов white label, но...

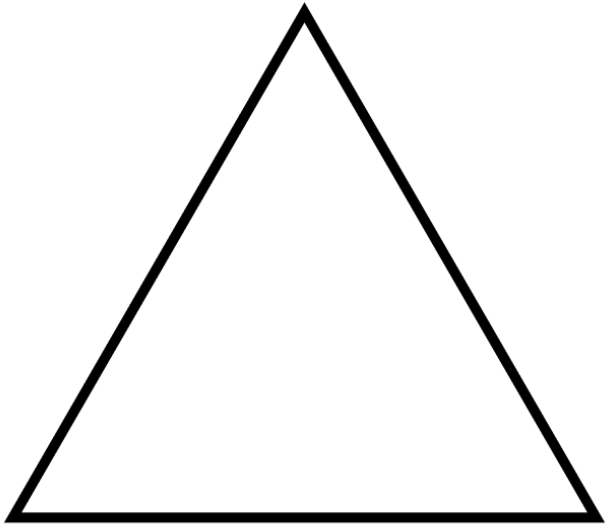
1. Apple явно запрещает клоны и рескины приложений (4.2 и 4.3)
2. Слышал много историй, когда релиз заканчивался неудачей даже через разные аккаунты
3. Решается повышением уровня кастомизации: цвета, тексты

1. В соглашении есть политика спама и повторяющегося контента
2. Можно иметь даже одинаковый брендинг при условии различий в контенте

С чего начинается архитектура

Хорошая архитектура не может быть спроектирована без опоры на предметную область

КАЧЕСТВЕННО

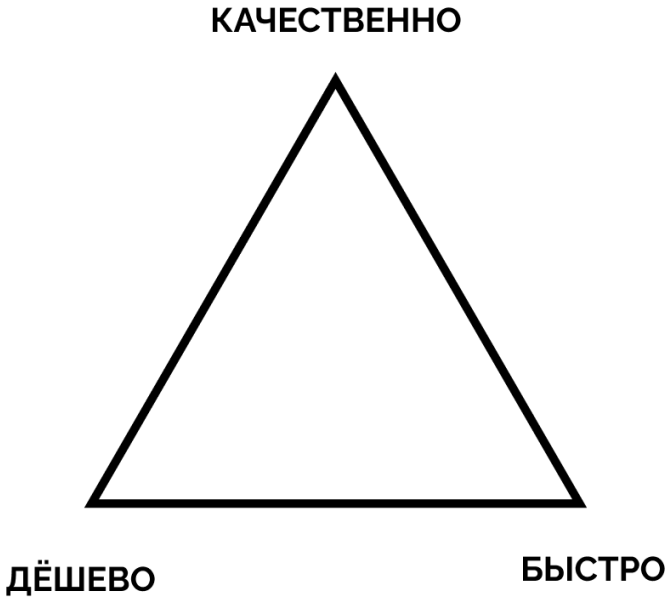


ДЁШЕВО

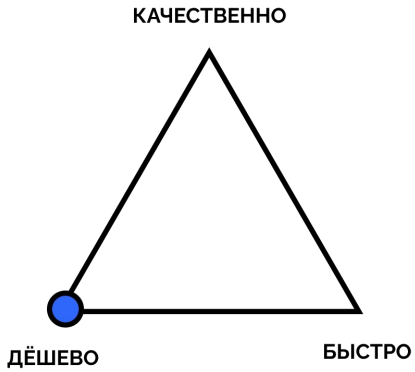
БЫСТРО

Предположим, к нам обращается ряд компаний владельцев сетей отелей, которые хотят приложение для бронирования номеров в своих отелях, но только для своей сети

Небольшая задача



Обратимся к крайностям треугольника

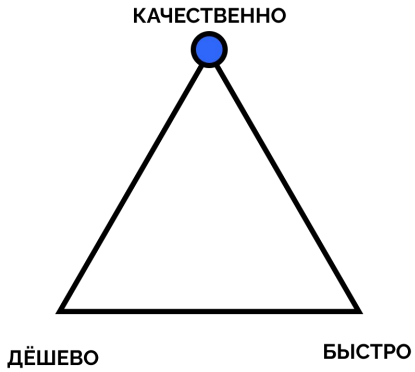


Предположим у компании неограниченное количество средств. Нужен ли им White Label?

HESOYAM 🤑

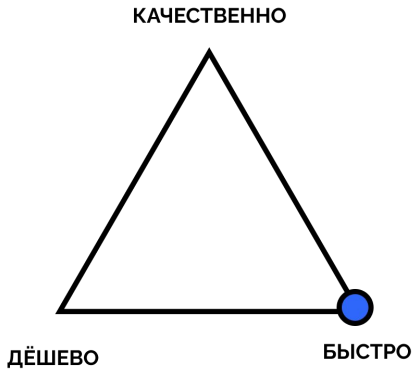
Качество?

То, насколько продукт попадает в стандарты и желание клиента



Компания хочет сделать что-то необычное. Нужен ли White Label?

Особая идея и большие амбиции 🪄

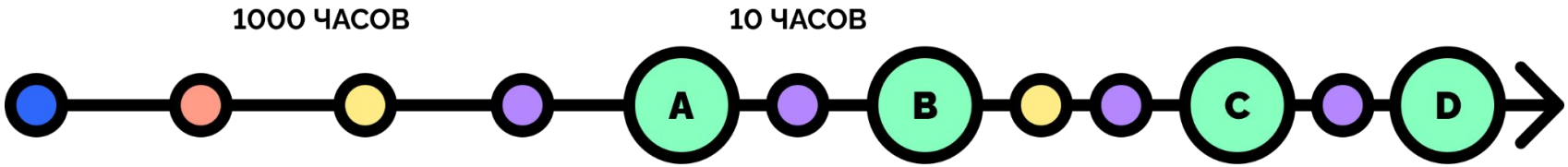


Компании срочно нужно выйти на рынок. Нужен ли White Label?

Нужно сделать быстро!

Time-to-Market

Главная метрика скорости в контексте выпуска продукта

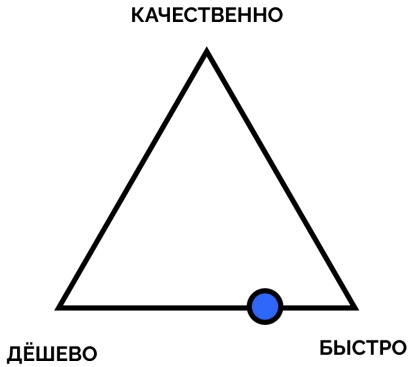


Time-to-Market

Для последующих клиентов

Промежуточные выводы

1. Качество — это не о White Label, мы не сможем удовлетворять всем потребностям
2. Мы можем лавировать между бюджетом и скоростью при разработке
3. Любой уход в сторону качества будет нести издержки по бюджету и скорости



И причём тут архитектура?

Используйте понимание места White Label на рынке, чтобы принимать архитектурные решения

Архитектор — это менеджер

Такой же, как тимлид, техлид или продакт

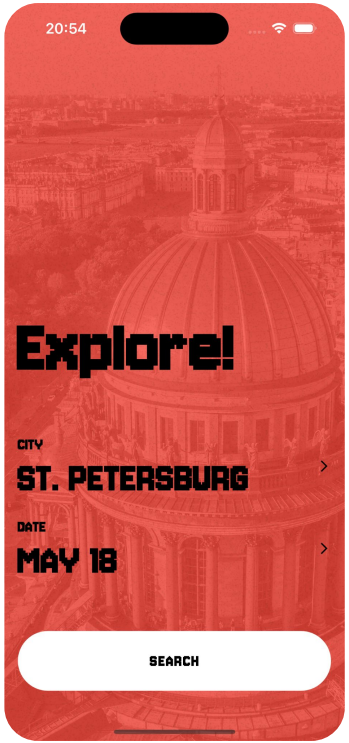
Задачи архитектора

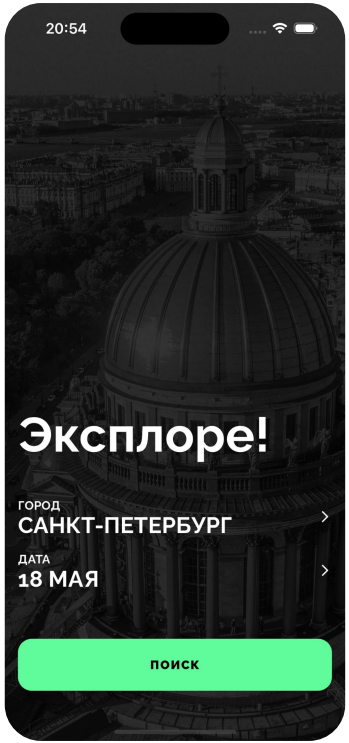
1. ✅Изучение бизнес-требований
2. ➡️SOONОбеспечение командной работы
3. Реализация грамотного **технического решения** 👉

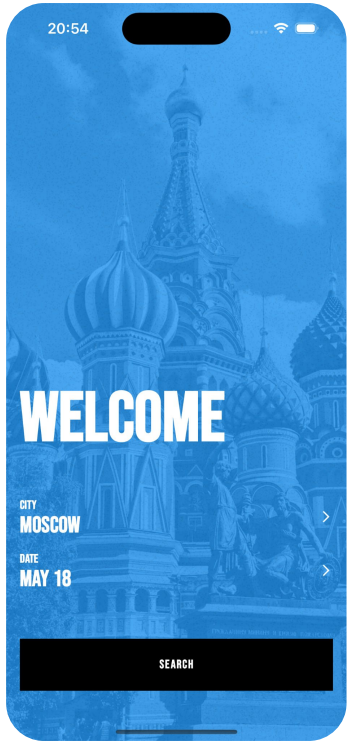
Архитектор — это технарь

Почему Flutter? 

1. Сокращение **стоимости разработки** и **time-to-market** за счёт платформ.
2. На самом деле, это может быть любая современная кроссплатформа: RN или KMP.
3. Дальнейшие рекомендации актуальны не только для Flutter, но код точно только для него.


20:54
Explore!
CITY
ST. PETERSBURG
DATE
MAY 18
SEARCH

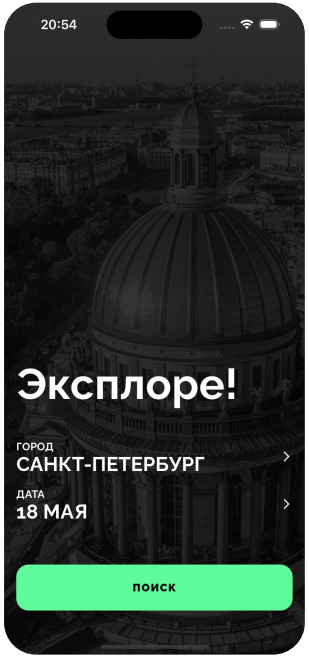

20:54
Эксплоре!
ГОРОД
САНКТ-ПЕТЕРБУРГ
ДАТА
18 МАЯ
ПОИСК

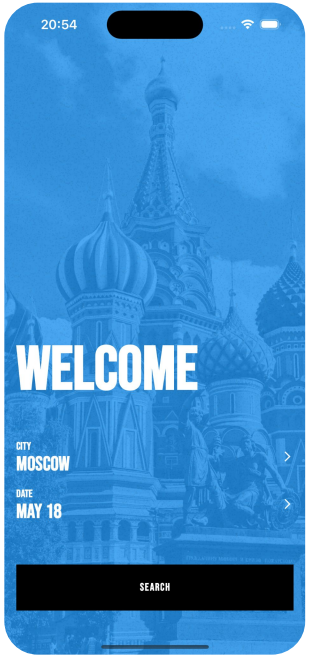

20:54
WELCOME
CITY
MOSCOW
DATE
MAY 18
SEARCH

```dart
@override
Widget build(BuildContext context) {
  return WhiteLabelApp(
    title: 'Black App',
    theme: const WLTheme(
      colorScheme: WLColorScheme.light(),
      typography: WLTypography(),
      overrides: WLThemeOverrides(),
      dimensions: WLDimensions(),
    ), // WLTheme
    appAssembly: AppAssembly(),
  ); // WhiteLabelApp
}
```

Пример на Github

Аккаунт @mpkander





Каждый цвет — вендор

UI

Пользовательский интерфейс

1. UI — самая изменчивая часть white label.
2. Большая часть работы фронтендера состоит из UI, как ни странно.
3. Цена ошибки при проектировании UI очень велика в White Label.

10/10

Так бы я оценил важность проектирования UI

Вам просто необходима дизайн-система

Начните с брендинга

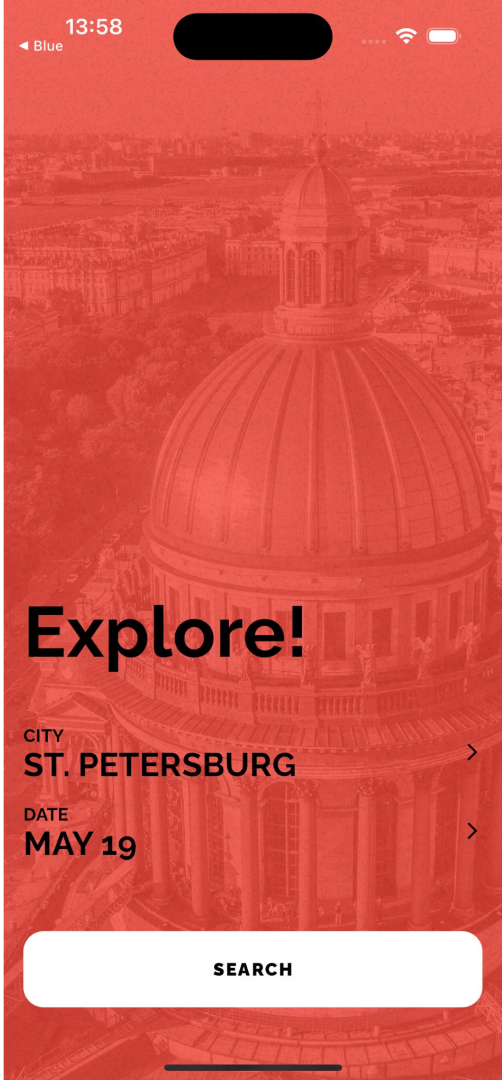
```
const WLColorScheme.light({
  this.brand = const Color(0xFF000000),
  this.onBrand = const Color(0xFFFFFFFF),
  this.background = const Color(0xFFFFFFFF),
  this.onBackground = const Color(0xFF000000),
  this.surface = const Color(0xFFFFFFFF),
  this.onSurface = const Color(0xFF000000),
  this.action = const Color(0xFF60fc9c),
  this.onAction = const Color(0xFF000000),
});
```

Цветовая схема

1. При правильно спроектированной дизайн-системе количество цветов даже в большом приложении может быть до 20-30
2. В своём примере я использовал дизайн-систему Material — так проще соотнести с Flutter

```
colorScheme: WLColorScheme.light(
  brand: Colors.white,
  onBrand: Colors.black,
  action: Colors.black,
  onAction: Colors.white,
), // WLColorScheme.light
```

```
colorScheme: WLColorScheme.light(
  brand: Colors.red,
  onBrand: Colors.black,
  action: Colors.white,
  onAction: Colors.black,
), // WLColorScheme.light
```



```
final Color brand;
final Color onBrand;
final Color action;
final Color onAction;
```

```
final Color blueButton;
final Color backgroundOnMainScreen;
final Color cardTextLight;
```

Избегайте конкретики

Не отсылайтесь к цветам или конкретным UI-компонентам

```
const WLColorScheme.dark({
  this.brand = const Color(0xFF000000),
  this.onBrand = const Color(0xFFFFFFFF),
  this.background = const Color(0xFFFFFFFF),
  this.onBackground = const Color(0xFF000000),
  this.surface = const Color(0xFFFFFFFF),
  this.onSurface = const Color(0xFF000000),
  this.action = const Color(0xFF000000),
  this.onAction = const Color(0xFFFFFFFF),
});
```

Не бойтесь одинаковых цветов

Не относитесь толерантно к небольшим различиям в теме

```dart
class WLThemeOverrides extends ThemeExtension<WLThemeOverrides> {
  final Color myExtraSpecificButtonColor;
```

Небольшие различия имеют место

1. Не думайте, что даже идеальная дизайн-система спасёт вас от незначительных изменений
2. Заведите класс для очень конкретных цветов — `WLThemeOverrides`

```dart
final colorScheme = WLColorScheme.of(context);
```

```dart
static Color brand = const Color(0xFF000000);
```

Забудьте о статике

Только обращение через контекст

```dart
@immutable
final class WLColorScheme extends ThemeExtension<WLColorScheme> {
```

Класс темы

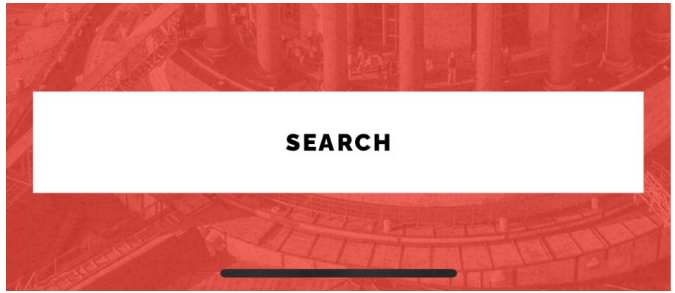
1. `ThemeExtension` позволяет привязать кастомную тему к основной теме аппа
2. Требует реализации `copyWith` и `lerp` — достаточно трудно поддерживать руками, велик шанс ошибки

```
theme: const WLTheme(
  colorScheme: WLColorScheme.light(),
  typography: WLTypography(),
  overrides: WLThemeOverrides(),
  dimensions: WLDimensions(),
), // WLTheme
```

Что ещё входит в конфигурацию?

1. ✅ Цветовая палитра
2. Размеры — отступы, паддинги и пространства
3. Радиусы скруглений
4. Текстовые стили и шрифты

```
dimensions: WLDimensions(
  corner100: BorderRadius.circular(0),
), // WLDimensions
```



```
final BorderRadius corner100;
final BorderRadius corner200;
final BorderRadius corner300;
```

Это не абсолютные значения!

1. **100** — самый большой
2. **200** — средний
3. **300** — самый маленький

```dart
final BorderRadius corner100; // большой
final BorderRadius corner200; // средний
final BorderRadius corner250; // по менбше
final BorderRadius corner300; // ваще маленький жесть...
```

Это не абсолютные значения!

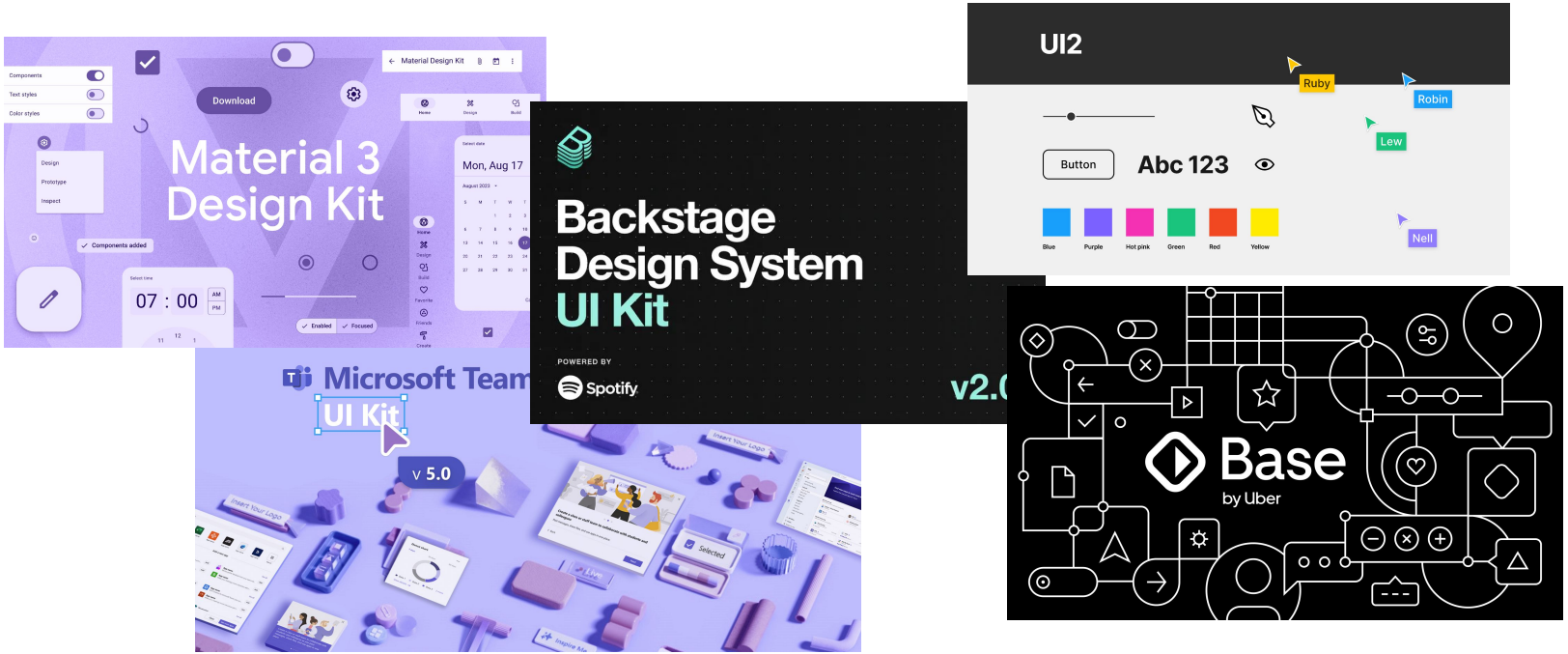
1. **100** — самый большой
2. **200** — средний
3. **250** — между средним и маленьким
4. **300** — самый маленький

```dart
final BorderRadius corner100 = BorderRadius.circular(32);
final BorderRadius corner200 = BorderRadius.circular(20);
final BorderRadius corner300 = BorderRadius.circular(12);
```

```dart
final BorderRadius corner32 = BorderRadius.circular(32);
final BorderRadius corner20 = BorderRadius.circular(20);
final BorderRadius corner12 = BorderRadius.circular(12);
```

Избегайте абсолютных значений

Не отсылайтесь к конкретном размеру в названии

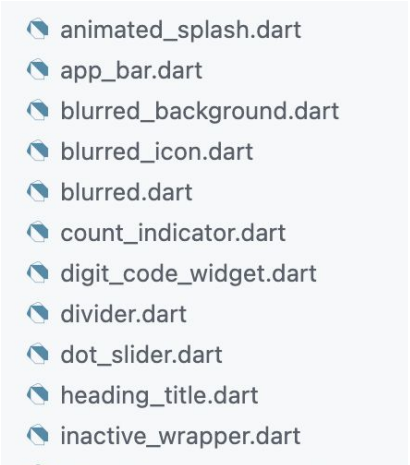


Изучать опыт — легко

Компании активно делятся своими дизайн-системами

UI-kit — лучшие практики

1. Вам нужен отдельный модуль (директория) для UI-kit
2. Оградите модуль UI-kit от внешних факторов — не импортируйте туда сущности из бизнес-логики
3. Введите ID для связи компонентов в Figma и виджетов в UI-kit
4. В UI-kit могут лежать `AppBar`, `Scaffold`, `NavigationBar`, но не сложные элементы интерфейса, которые завязаны на логике приложения.

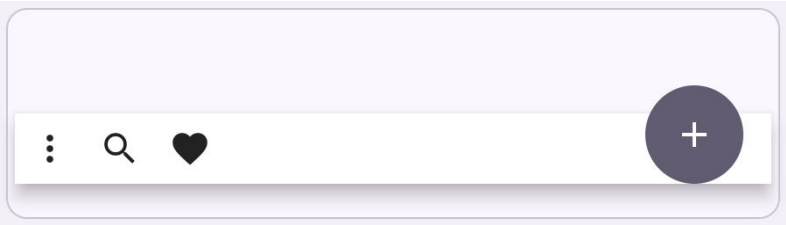


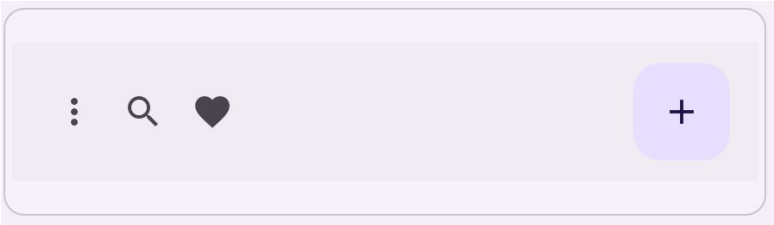
```
Container();
Text('');
Column();
```

```
Material();
AppBar();
MaterialButton();
```

Не злоупотребляйте виджетами Material

Material-виджеты только для Material-дизайна





Material 2

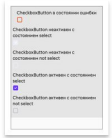
Material 3

Не связывайте себя изменчивостью внешней библиотеки

Golden-тесты

Они вам точно нужны

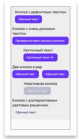

checkbox_button_skeleton.png


checkbox_button.png


chip.png

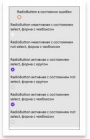

primary_button.png


secondary_button.png


small_button.png


primary_button_skeleton.png


radio_button_skeleton.png

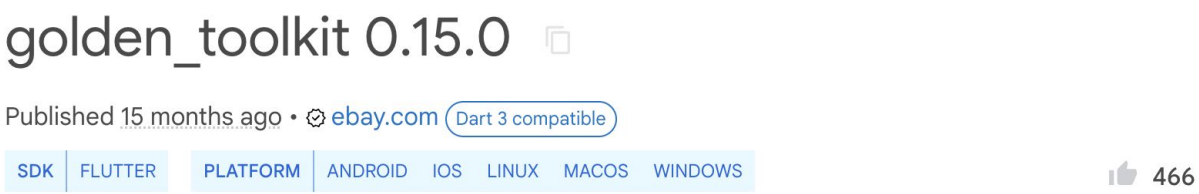

radio_button.png


secondary_button_skeleton.png


small_button_skeleton.png

Магия? 💫

1. Меняете пару строчек в теминге
2. Запускаете одну консольную команду
3. Ждёте несколько секунд
4. Получаете набор PNG со скриншотами всех экранов и UI-kit в приложении



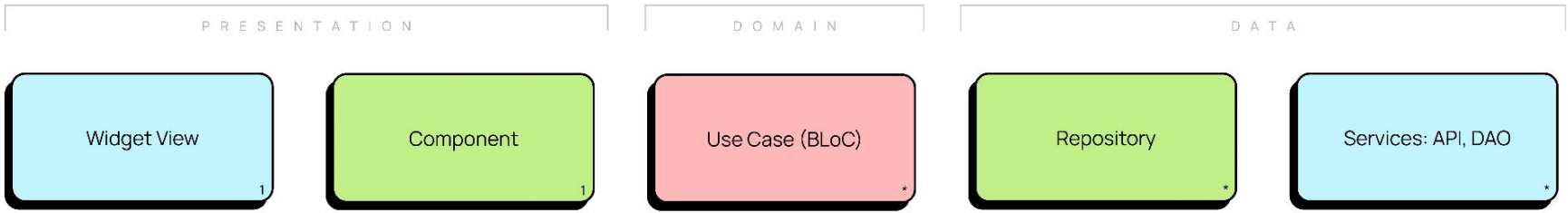
Недостатки

1. Есть ограничения на загрузку шрифтов — только один тип
2. Шрифты генерируются на всех платформах (macOS, Windows, Linux) по-разному
3. UI местами тоже: блюр, градиенты

Итог: UI

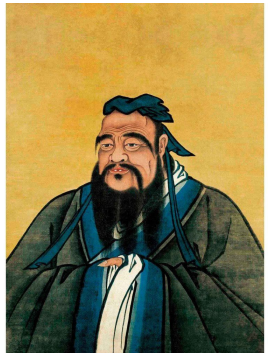
1. **Теминг** (цвета, шрифты, размеры) для снижения риска при изменениях.
2. **UI-kit** — чтобы собирать приложение из компонентов.
3. **Golden-тесты** — чтобы отлавливать изменения и молниеносно настраивать тему.

Архитектура фичи



Архитектура фичи

Тут не будет сюрпризов, она ничем не отличается от обычного приложения



Народные мудрости

1. Не забывайте, что архитектура White Label сама по себе достаточно тяжеловесная и не усложняйте фичу
2. Не спорят только о вкусах и архитектурных решениях — выбирайте то, к котором лежит душа ❤️

Главная загадка человечества

Single App

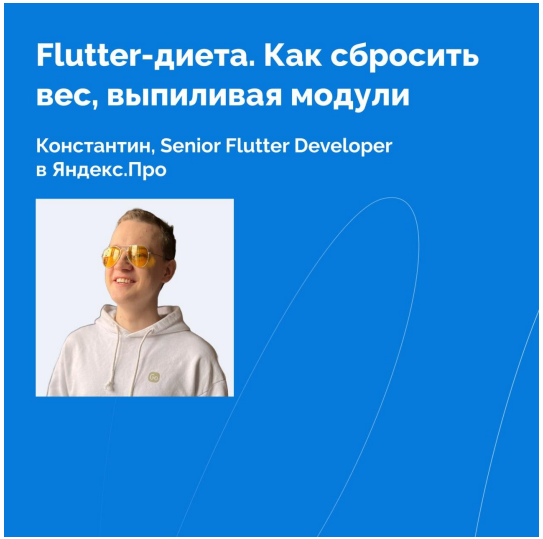
Shared Package

Single App

1. У нас есть всего **один Flutter-проект** — White Label App
2. Информация о всех вендорах хранится в этом одном проекте
3. Конфигурация происходит через flavors, feature-toggle

Проблема подключаемых фичей

1. Разным вендорам не нужны все фичи, существующие в вашем приложении
2. Эти фичи нужно как-то отключать
3. **В скомпилированном приложении не должна остаться информация об отключенных фичах**



TG: surf_flutter

Dart Define

Существует способ с dart define, но о нём лучше рассказал **Костя** из **Яндекс.Про** на нашем митапе в Москве

```
flutter run --flavor red
```

Flutter Flavors

Они стали чуть более прокачанными и я это упустил

```yaml
assets:
  - assets/common/
  - path: assets/red-one/
    flavors:
      - red
  - path: assets/blue-one/
    flavors:
      - blue
```

Отключение ресурсов

В зависимости от сборки

Support different dependencies when using flavors #46979

Open ghost opened this issue on Dec 13, 2019 · 24 comments

Отключать библиотеки нельзя

Но есть кастомные и костыльные решения

```dart
const String? appFlavor = String.fromEnvironment('FLUTTER_APP_FLAVOR') != '' ?
  String.fromEnvironment('FLUTTER_APP_FLAVOR') : null;
```

Зато есть доступ из Dart-кода

Он и поможет отключать ненужные фичи

Но нужно потрясти дерево

Tree Shaking

```
AuthFeature? createFeature() {
  const hasFeature = appFlavor != 'red' || appFlavor != 'blue';

  if (!hasFeature) return null;

  return AuthFeature();
}
```

Лёгким движением руки убираем часть кода

const — обязателен

```dart
if (featureToggle.hasAuthFeature) {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => AuthScreen(
        feature: AuthFeature(),
      ), // AuthScreen
    )); // MaterialPageRoute
}
```

Простого if — недостаточно

const — обязателен, не используйте динамические feature toggle

Single App — это здорово

С нюансами

Всё портит один факт

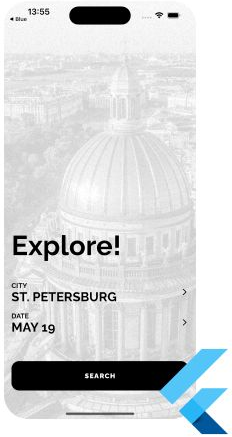
Вы уже начали разработку, но один из вендоров хочет забрать приложение в ин-хаус и разрабатывать его сам

В Single App все клиенты знают друг о друге

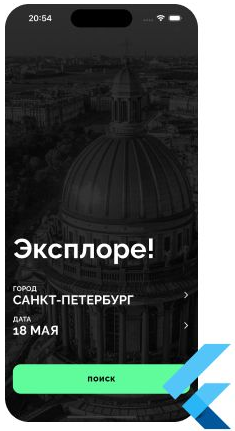
(и это проблема)

Четыре слова, которые мечтает услышать каждый разработчик

ДАВАЙТЕ НАПИШЕМ ВСЁ ЗАНОВО



white_label_core



Shared Package

1. Вместо одного Flutter-приложения — один (или несколько) Flutter-package
2. Приложения для разных вендоров существуют отдельно и имеют package в зависимостях

```
return MaterialApp(
  title: 'Flutter Demo',
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(
      seedColor: Colors.deepPurple,
    ), // ColorScheme.fromSeed
    useMaterial3: true,
  ), // ThemeData
  home: const MyHomePage(title: 'Flutter Demo Home Page'),
); // MaterialApp
```

С чего начинается обычный Flutter App

```dart
@override
Widget build(BuildContext context) {
  return WhiteLabelApp(
    title: 'Black App',
    theme: const WLTheme(
      colorScheme: WLColorScheme.light(),
      typography: WLTypography(),
      overrides: WLThemeOverrides(),
      dimensions: WLDimensions(),
    ), // WLTheme
    appAssembly: AppAssembly(),
  ); // WhiteLabelApp
}
```

С чего начинается White Label App

Что такое core-модуль?

1. `WhiteLabelApp` — точка входа для настройки пакета
2. Принимает различные конфиги: теминг, локализацию, нетворк, ассеты и многие другие
3. Экспортирует контракты для взаимодействия с core-частью
4. Содержит дефолтные настройки

```yaml
dependencies:
  flutter:
    sdk: flutter

  white_label_core:
    path: ../../white_label_core
```

Удобство разработки

1. Вам нужно иметь эталон — обычно он называется **example** (у меня в проекте — **white**)
2. Эталон вы и будете препарировать, дебажить и собирать сборки для тестирования
3. Нет никаких неудобств с точки зрения IDE — вы можете настроить запуск эталона прямо из модуля

```
assets:
  - assets/images/

fonts:
  - family: Raleway
    fonts:
```

```
const WLAssetImagesData({
  this.mainBackground = const AssetImage(
    'assets/images/main_bg.jpg',
    package: packageName,
  ),
  this.noise = const AssetImage(
    'assets/images/noise.png',
    package: packageName,
  ),
});
```

Добавление ресурсов

1. Добавление ресурсов ничем не отличается от обычного приложения
2. (!) Но обратите внимание на параметр `package` у ассетов, шрифтов и иконок

```
name: white_label_core
description: "White Label Core Flutter Package"
version: 0.4.1
```

Версионирование

Ещё одно побочное преимущество данного подхода

```dart
class TranslationEn extends WLTranslation {
  TranslationEn([super.locale = 'en']);

  @override
  String get appName => 'White Label';

  @override
  String get mainTitle => 'Explore!';

  @override
  String get mainCity => 'City';
```

Локализация

1. Локализация — точка изменения тестовых ресурсов для каждого вендора
2. Самое удобное — использование встроенной `flutter_localizations` и `intl`
3. Если вам необходимы arb-файлы, то вы столкнетесь с некоторой болью при генерации кода
4. Но делать локализацию без кодогенерации также больно

```dart
class CompositeAnalyticsService implements AnalyticsService {
  final List<AnalyticsService> _services = [];

  void addService(AnalyticsService service) {
    _services.add(service);
  }

  @override
  void track(AnalyticsEvent event) {
    for (var service in _services) {
      service.track(event);
    }
  }
}
```

Логи и аналитика

1. Наконец-то нужно будет вспомнить паттерны проектирования — я вспомнил компоновщик
2. Обязательно сделайте базовую реализацию для debug-режима
3. Экспортируйте контракты, но **не храните реализацию внутри**
4. Вынесите реализацию для Sentry, Firebase и других типичных сервисов в отдельные модули

Итог. Что должно быть в конфиге?

1. **Теминг:** цвета, шрифты, размеры.
2. **Ассеты:** картинки, иконки.
3. **Логгирование и аналитика:** экспорт контрактов, подключение “извне”.
4. **Локализация:** управление текстовыми данными приложений.
5. **Feature-toggle:** отключение фичей и изменение флоу.
6. **Конфигурация сети**.

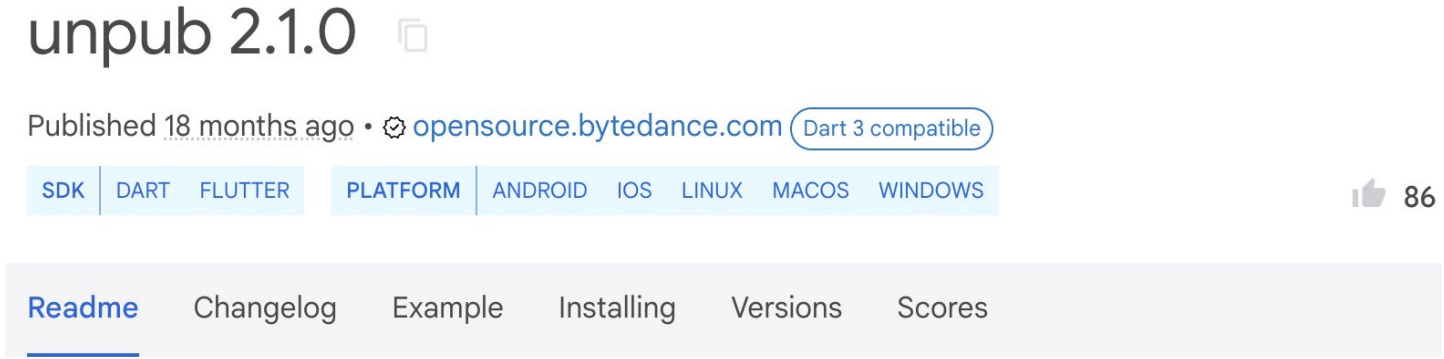
ТОП-3 ОШИБОК

1. Не пренебрегайте настройкой CI для core-модуля ни в коем случае
2. Проверяйте golden-тесты на CI
3. Сделайте автоматическую поставку для каждого вендора
4. Хот-фикс для 7 вендоров — это 7 релизов параллельно

CI/CD

1. Сделайте шаблон для развёртывания с подробной инструкцией
2. Флейворы, сертификаты, Firebase — всё это настраивается для каждого вендора отдельно
3. Хорошие шаблон и инструкция сэкономит вам **огромное количество времени**

Шаблонизация



Если ваше приложение превратится не в единый модуль, а набор модулей — это отличный вариант

Хостинг

ИТОГИ

Что в итоге?

1. Не вложиться в проектирование white label **будет ошибкой**.
2. Я рекомендую использовать **вариант с package**.
3. Не бойтесь вкладываться в white label, **его можно сделать крутым**.

Полезные ссылки и источники