SUI на Flutter (The Room)

00 | Представлюсь

01 | Что такое SUI?

02 | Наша идея реализации

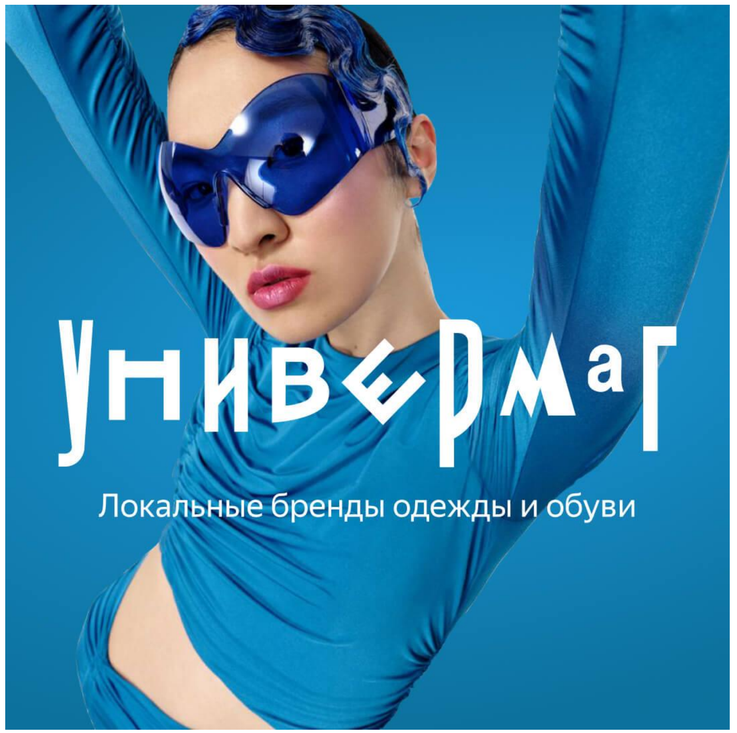
03 | Проблемы и решения

04 | Направление движения

05 | Конец!

Яндекс

The ~~Room~~



Гена Евстратов, руководитель службы

С чего всё началось

Кто я такой?

- Меня зовут Гена, я занимаюсь мобильной разработкой в Яндексе
- Возможно, вы где-то уже меня видели
- Потому что я много говорю про Flutter и продвигаю его внутри компании и не только

Жизненный путь

- Какое-то время я занимался Яндекс.ПРО, мы его решили написать для iOS
- Написали, заканчиваем переписывать на Flutter android приложение
- Пишем ещё части для доставки, курьеров еды и лавки, услуг и курьеров маркета

Flutter в Яндексе

Flutter в Яндексе есть не только в ПРО

- Драйв
- Q
- Практикум
- Всякие другие штуки

Flutter в Яндексе

> **Пока на Flutter из больших приложений у нас есть только приложения для исполнителей**

Но недавно меня позвали в маркет немного помочь с клиентским приложением

Зачем маркету понадобился Flutter?

- Хотелось делать быстрые эксперименты с UI
- И не только быстрые, но и красивые
- И не только красивые, но и управляемые с сервера, и одинаковые на всех платформах

То есть мы захотели сделать Server-Driven UI.

Управляемый с сервера UI

Что это вообще такое

> **Из-за того, что время релиза мобильных приложений ненулевое и, зачастую, ещё и непредсказуемое, хочется уметь управлять их интерфейсом и поведением вне этого цикла**

К этому есть несколько подходов, давайте посмотрим, какие

Подход браузера

> **Сервер отдаёт верстку и данные вместе, клиент не понимает семантику того, что показывает. В пределе это всем знакомый веб.**

Есть приложения, работающие по такому принципу, например, показывающие веб страницы в качестве своих экранов и обеспечивающие прослойку между вебом и железом/нужными библиотеками

Плюсы «подхода браузера»

- Мы виртуально не ограничены ничем при создании UI, ну кроме возможностей самого браузера
- Если «браузер» достаточно мощен, мы можем реализовать очень много на его базе (см. веб-браузер)

Минусы «подхода браузера»

- Возможности веб-браузера сильно больше, но и сам браузер тяжелее и неповоротливее
- На разных устройствах веб-браузер ведет себя немного по-разному. И приложение, соответственно, тоже
- Неизбежно в браузере или зарождается свой, или приносится чужой язык (см. JavaScript)

Подход конструктора / «ЛЕГО»

> **Сервер отдаёт верстку (шаблоны) и данные раздельно, клиент понимает семантику верстки (например, списки), и при рендеринге верстки наполняет шаблоны данными**

Это напоминает шаблонизаторы вроде jinja и moustache. Мы получаем шаблон и данные, и по этим параметрам генерируем финальный UI.

Плюсы «подхода лего»

- Работает быстрее, чем браузер
- Данные отделены от представления, можно легко реализовывать всякие темы/скины и прочее
- Можно легко переделывать верстку, потому что она состоит не из примитивов, а из блоков

Минусы «подхода лего»

- Добавить новый семантический блок можно только с релизом приложения, и поэтому есть проблема «хвоста версий»
- Ограниченное количество доступных семантических блоков
- Сложность сильной кастомизации самих блоков без релиза приложений

Что мы сделали?

Решили взять лучшее из двух миров

> **Оба подхода не лишены плюсов и минусов. Мы решили их объединить, чтобы соединить гибкость одного и простоту второго.**

- Использовать семантические блоки и разделение данных и представления
- При этом сами блоки верстать на атомах дизайн-библиотеки
- Flutter даст здесь гарантированно одинаковое поведение на разных платформах

KISS

> **Keep It Simple, Stupid. Увидели цель и не увидели препятствий.**

Цели:

- Хотелось (и продолжает хотеться) сделать всё максимально просто
- Человеко-читаемый формат
- Отсутствие привязки к конкретным реализациям бекенда и инфоаструктуры

KISS

Вспомнили, что в Flutter всё это виджет и написали что-то вроде такого.

Пока это чисто подход номер один, но посмотрим, куда нас это заведёт.

```
{
  "type": "Container",
  "color": "ffffffff",
  "child": {
    "type": "Center",
    "child": {
      "type": "Text",
      "text": "Hello, world!"
    }
  }
}
```

Парсер мой парсер

Для парсинга этого описания надо:

- Посмотреть на поле «type»
- Сделать виджет таким названием и заполнить параметры конструктора полями из json

Задача не очень сложная, но очень муторная. Поэтому мы решили использовать библиотеку **dynamic_widget**

Добавляем свои виджеты

Реализовав вот такой класс мы можем парсить и экспортировать произвольный самодельный виджет.

Экспорт я не применял, поэтому на месте этих методов оставлял заглушки.

```dart
/// extends this class to make a Flutter widget parser.
abstract class WidgetParser {
  /// parse the json map into a flutter widget.
  Widget parse(Map<String, dynamic> map, BuildContext buildContext,
      ClickListener? listener);

  /// the widget type name for example:
  /// {"type" : "Text", "data" : "Denny"}
  /// if you want to make a flutter Text widget, you should implement this
  /// method return "Text", for more details, please see
  /// @TextWidgetParser
  String get widgetName;

  /// export the runtime widget to json
  Map<String, dynamic>? export(Widget? widget, BuildContext? buildContext);

  /// match current widget
  Type get widgetType;

  bool matchWidgetForExport(Widget? widget) => widget.runtimeType == widgetType;
}
```

Свои виджеты

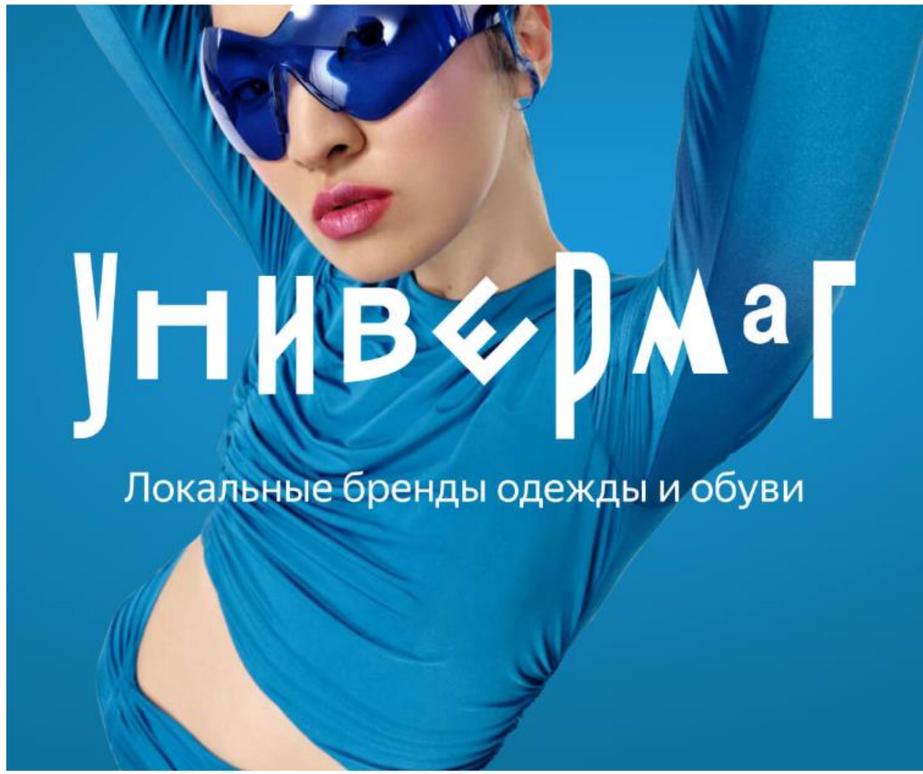
> **Их получилось немного, и большинство понадобились для того, чтобы решить вопрос отделения данных и представления (про это позже)**

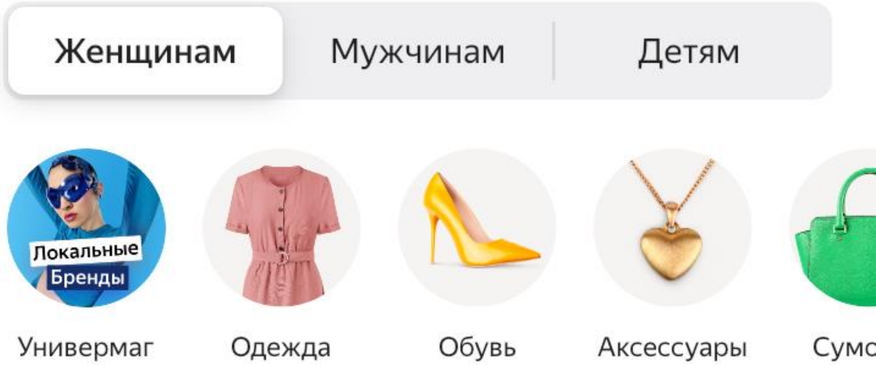
- Текст
- Картинка
- Список
- Сетка
- Ссылка (самое главное)
- И ещё пара, про которые позже

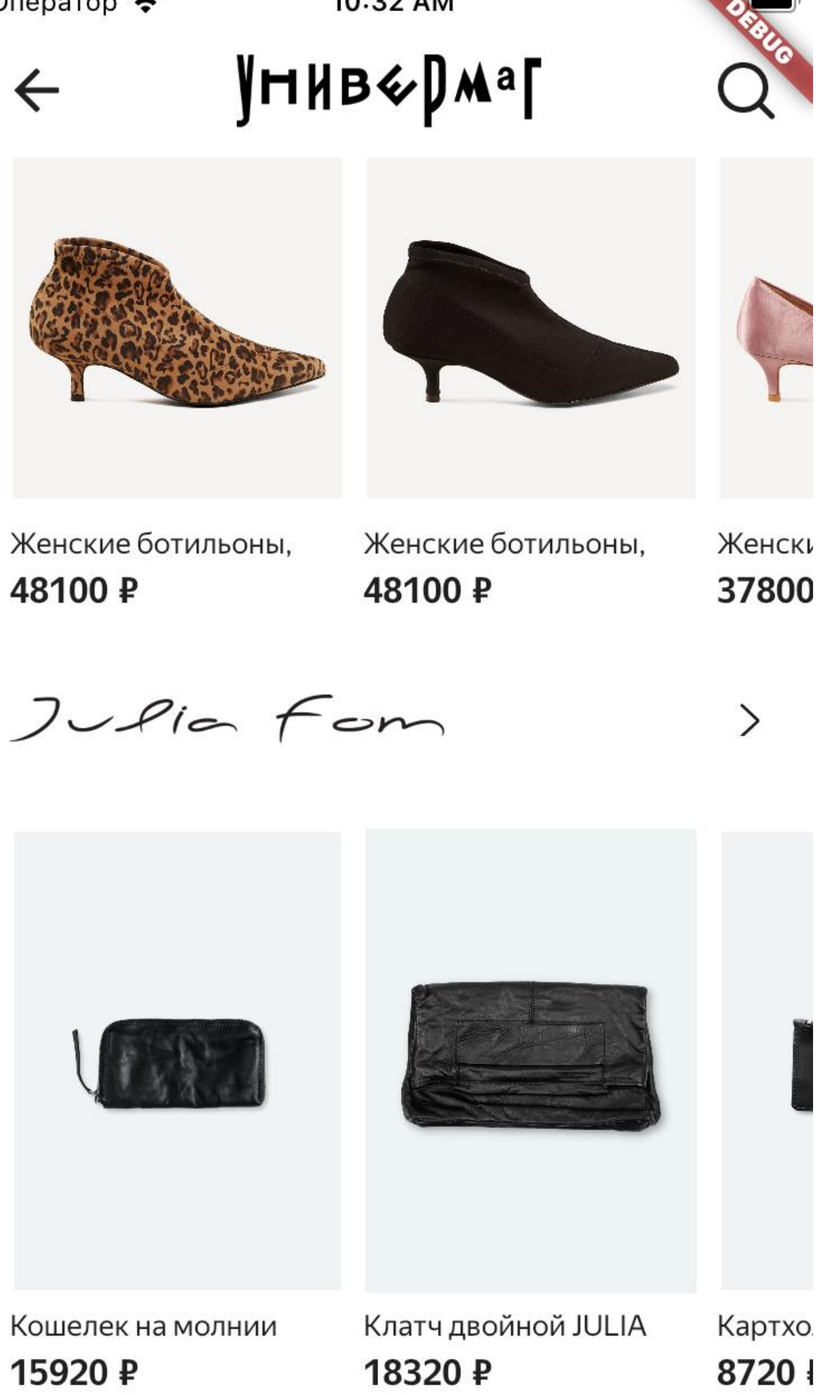
Первый результат

> **На моках получилось миленько, но в реальном мире существовал ряд Проблем**

- Многословность описания
- Данные соединены с представлением
- Нет взаимодействия с пользователем







Многословность описания

Напомню, что к этому моменту я всё ещё полностью писал в парадигме первого подхода (браузер) и семантических элементов у меня не было.

Кроме того, чтобы их ввести мне нужно было отделить представление от данных.

А пока каждый повторяющийся элемент приходилось описывать отдельно, кучей повторяющегося текста.

Ref to the rescue

> **Первым делом я научился переиспользовать куски описания. Это очень сократило объем описания верстки.**

```
{
    "type": "Root",
    "children": [
      {
        "type": "Text",
        "style": {},
        "text": "this is header"
      },
      {},
      {
        "type": "Text",
        "style": {},
        "text": "this is header"
      },
      {},
      {
        "type": "Text",
        "style": {},
        "text": "this is header"
      },
    ]
  }
}
```

превратилось в

```
{
  "header": {
    "type": "Text",
    "style": {},
    "text": "this is header"
  },
  "main": {
    "type": "Root",
    "children": [
      {"type": "Ref", "name": "header"},
      {},
      {"type": "Ref", "name": "header"},
      {},
      {"type": "Ref", "name": "header"},
    ]
  }
}
```

Ref to the rescue

> **При этом внутри он устроен супер-несложно (пока что) и просто берет из словаря с верстkой данные и вставляет на своё место**

Тем не менее именно этот шаг переводит нас из подхода номер 1 в то, что мы хотим в итоге получить

```
1 Widget parse(Map<String, dynamic> map, BuildContext buildContext,
2         ClickListener? listener) =>
3   DynamicWidgetBuilder.buildFromMap(
4     WidgetsProvider.of(buildContext).widgets[map["name"]],
5     buildContext,
6     listener)!;
```

Как передавать данные внутрь?

> **Мы получили семантические шаблоны, но в них одни и те же данные. Сейчас мы это исправим**

Вспомним про контекст и **InheritedWidget.** Что если в контекст класть специфичные для нашего виджета данные, всегда в одно и то же место, а виджет будет их там искать?

```dart
class DataProvider extends InheritedWidget {
  const DataProvider({
    Key? key,
    required this.data,
    required Widget child,
  }) : super(key: key, child: child);

  final Map<String, dynamic> data;

  static DataProvider of(BuildContext context) =>
      context.dependOnInheritedWidgetOfExactType<DataProvider>()!;

  @override
  bool updateShouldNotify(covariant DataProvider oldWidget) =>
      data != oldWidget.data;
}
```

```
"header": {
  "type": "KatKitText",
  "style": {},
  "text": "${header}"
}
```

Использование контекста развязало мне руки

> **Стали возможны виджеты процессинга данных и виджеты условного рендеринга**

```
{
  "type": "DataConverter",
  "converter": "getSKUs",
  "dataKey": "items",
  "child": {
    "type": "DataConverter",
    "converter": "subset",
    "length": 20,
    "dataKey": "items",
    "child": {
```

```
{
  "type": "Condition",
  "condition": "nonEmpty",
  "dataKey": "items",
  "false": {
    "type": "Container"
  },
  "true": {
    "type": "Ref",
    "name": "Carousel"
  }
}
```

У них очевидная реализация, но при этом они полностью заканчивают переход к нашей целевой гибридной модели

Промежуточный итог

Всё вместе это собралось вот в такой виджет. Но у меня остались вот какие проблемы:

- Обработка ввода пользователя
- Аналитика

```
class RyvalWidget extends StatelessWidget {
  final Map<String, dynamic> widgets;
  final Map<String, dynamic> data;

  RyvalWidget({
    Key? key, required this.data, required this.widgets})
       : assert(widgets.isNotEmpty, "ryval data can not be empty"),
         super(key: key);

  @override
  Widget build(BuildContext context) => WidgetsProvider(
        widgets: widgets,
        child: Builder(
          builder: (context) => DataProvider(
            data: data,
            child: Builder(
              builder: (context) => DynamicWidgetBuilder.buildFromMap(
                WidgetsProvider.of(context).widgets["main"],
                context,
                NonResponseWidgetClickListener(),
              )!)))));
}
```

Обработка взаимодействия с пользователем

Что может сделать пользователь?

> **На самом деле пользователь может только куда-то нажать, и через это изменить состояние чего-то или вызвать какое-то действие**

Действием же может быть

- Переход куда-то по диплинку (или внутреннему роуту)
- Вызов метода на сервере
- Вызов метода хостового приложения

Как всегда, всё это виджет

Поэтому для обработки нажатий я определил ещё один кастомный элемент, который как раз и делает логику с предыдущего слайда

Своими параметрами он принимает:

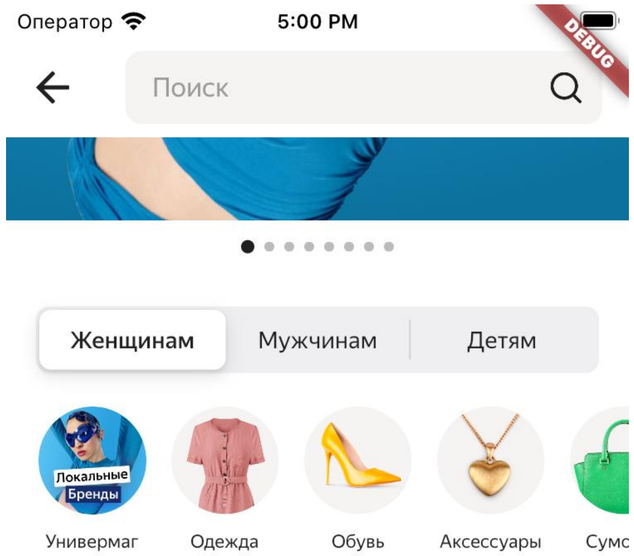
- Диплинк или действие при нажатии
- Наличие и тип анимации при нажатии

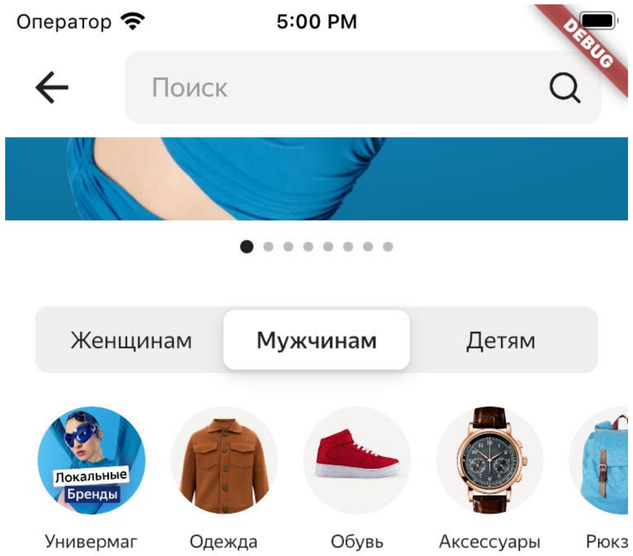
Если всё это не передать, то просто ничего не произойдёт

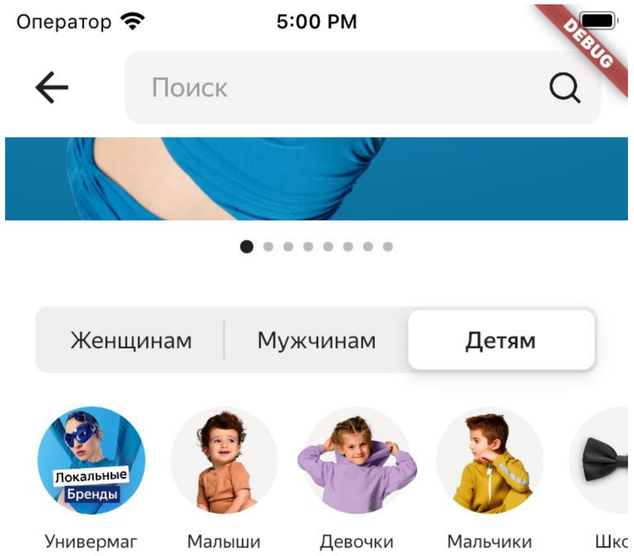
```
{
    "type": "KatKitKlicker",
    "deeplink": "${deeplink}",
    "child": {
        "type": "SizedBox",
        "height": 92.0,
        "width": 72.0,
        "child": {
            "type": "Column",
            "crossAxisAlignment": "center
            "mainAxisAlignment": "start",
            "mainAxisSize": "max",
            "children": [
                {
                    "type": "Expanded",
                    "flex": 1,
                    "child": {
                        "type": "KatKitIma
                        "aspectRatio": 1.0
                        "url": "${url}"
                    }
                },
                {
                    "type": "Padding",
                    "padding": "0,6,0,0",
                    "child": {
                        "type": "KatKitTex
                        "text": "${title}
                        "maxLines": 2,
```

Научили виджеты в локальные состояния

Теперь виджеты могут меняться без запросов на сервер







Аналитика это клики и просмотры

С кликами мы научились работать на предыдущем слайде и теперь просто добавим в виджет обработки нажатия параметры, которые надо отправить в аналитику (если они есть)

```
{
    "deeplink": "some_link_here",
    "url": "images/32bca4ee-1d56-4589-9865-243687f43be2.png",
    "header": "Все товары",
    "eventName": "THIS_IS_EVENT_NAME",
    "eventParams": {
        "skuId": "${skuId}",
        "vendorName": "alrawash",
        "zoneName": "${header}"
    }
}

```

События видимости

Для этого я взял библиотеку **visibility_detector**, сделал отдельный виджет, который вызывает аналитику с заданными параметрами, когда его ребенок становится виден больше, чем на определённый процент

```dart
class KatKitPeeperWidgetParser extends WidgetParser {
  @override
  Map<String, dynamic>? export(Widget? widget, BuildContext? buildContext) {
    throw UnimplementedError();
  }

  @override
  Widget parse(
    Map<String, dynamic> map,
    BuildContext buildContext,
    ClickListener? listener,
  ) =>
      KatKitPeeper(
          threshold: map["threshold"] ?? 0.5,
          visibleEventName: map["visibleEventName"],
          invisibleEventName: map["invisibleEventName"],
          visibleEventParams: map["visibleEventParams"],
          invisibleEventParams: map["invisibleEventParams"],
          child: DynamicWidgetBuilder.buildFromMap(
              map["child"], buildContext, listener)!);

  @override
  String get widgetName => "KatKitPeeper";

  @override
  Type get widgetType => throw UnimplementedError();
}
```

Вот теперь точно в релиз!

> **У нас получилось что-то среднее между подходом браузера и подходом «лего», и оно хорошо работает!**

- Представление отделено от данных
- Есть большие семантические элементы, которые можно переиспользовать
- Приложение реагирует на действия пользователя, анимирует нажатия
- Умеет отправить аналитику, если это требуется
- И работает на всех поддерживаемых Flutter платформах

Немного статистики

Статистика, совсем чуть-чуть

Строк кода (без библиотек): ~4 тысячи

Чистое время разработки: ~30 часов одного человека

Экранов собрано: ~40 штук руками, пока не был написан генератор

Время на сборку 1 экрана руками: 30 минут

Время на сборку экрана в админке: до 5 минут

Что дальше?

Очевидно, что нам есть куда расти

> **Описанное решение это эксперимент в супер ранней стадии. На его базе уже запустили следующий, но это не значит, что в этом месте я остановился**

Дальше я расскажу про то, что **точно** планируется сделать в рамках данного проекта

Избавиться от dynamic_widget

> **Эта библиотека предназначена для подхода браузера, поэтому мы из неё используем только несколько кусочков парсера**

- Лишняя зависимость, которая практически не используется
- Код в ней очень несложный
- Так что мы уже запланировали время на то, чтобы от него избавиться

Но вообще это очень хорошее место для быстрого старта в этом направлении. Как только вы начнете вносить изменения в **dynamic_widget**, вы поймёте, нужна она вам в итоге или нет

Померять всё и вся

> **При внедрении текущего решения мы ориентировались только на метрики приложения и не смотрели на остальное**

Хочется понять

- Насколько дольше или быстрее в среднем стала сборка
- Насколько проще или сложнее стало работать разработчикам
- Провести серию более детальных замеров показателей самих приложений

Внедрить это решение для мобильного веба

> **В последнее время приложения проходят ревью достаочно непредсказуемо, и, так как разработка модуля идёт полным ходом, мы не всегда можем вовремя раскатить нужные изменения**

Поэтому мы сделали бридж для того, чтобы использовать в приложении этот модуль через webview и потом, при релизе нативной версии, переключаться на неё.

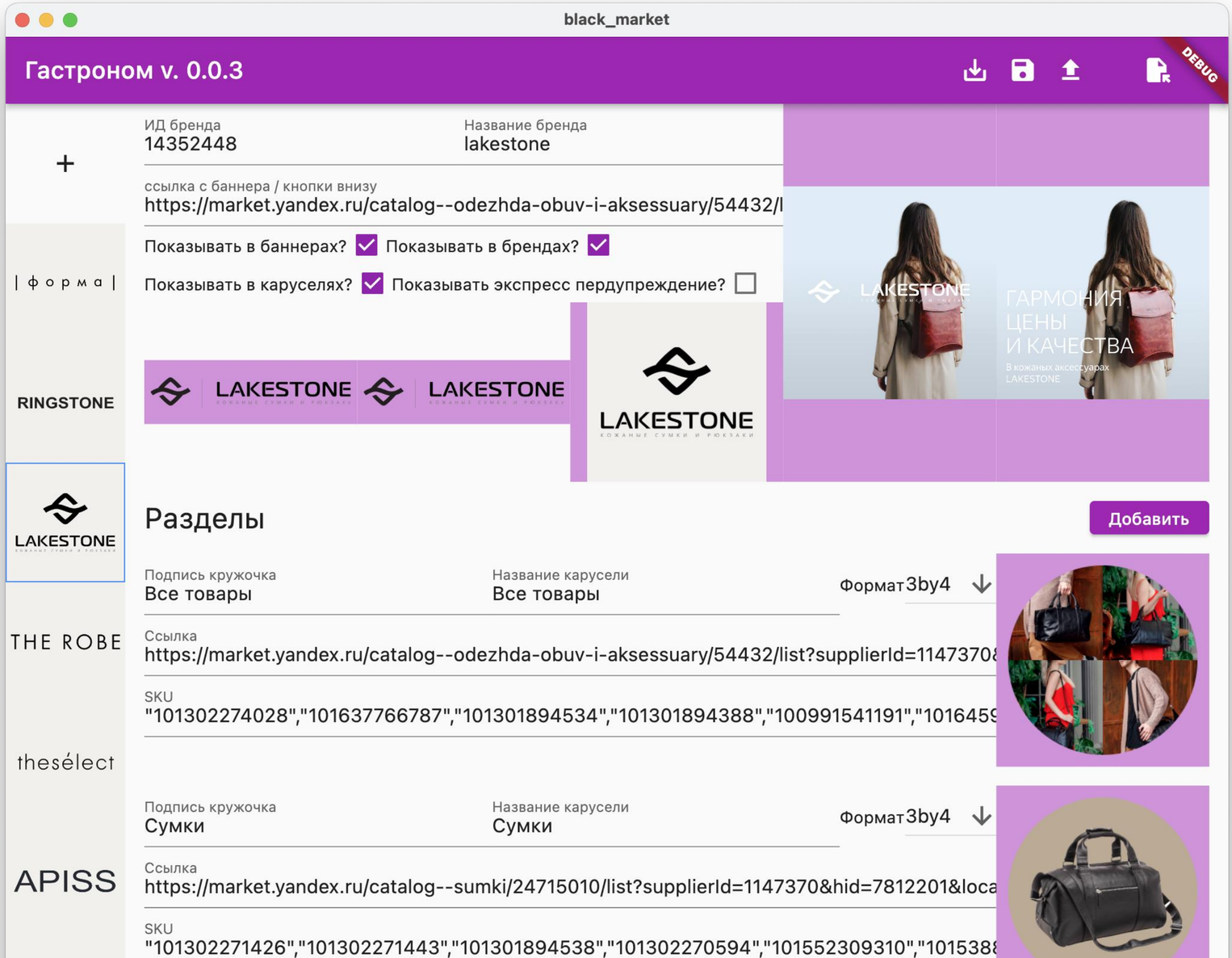
Кроме того, это позволит бесплатно получать всё то же самое в мобильном вебе в браузере.

Сделать полноценную админку для страниц

> **Около 40 страниц я собрал руками, устал и на базе того же кода сделал админку**

Она пока страшненькая, но дизайнер уже работает над этим.

Красивую версию отдадим партнерам.



Яндекс

Гена Евстратов

руководитель службы

genaevstratov@yandex-team.ru

@jewpacabra