

Москалева Ю.П.

# Стикеры «Солнышко»

Flutter

Симферополь  
Декабрь 2023 г.

## **СОДЕРЖАНИЕ**

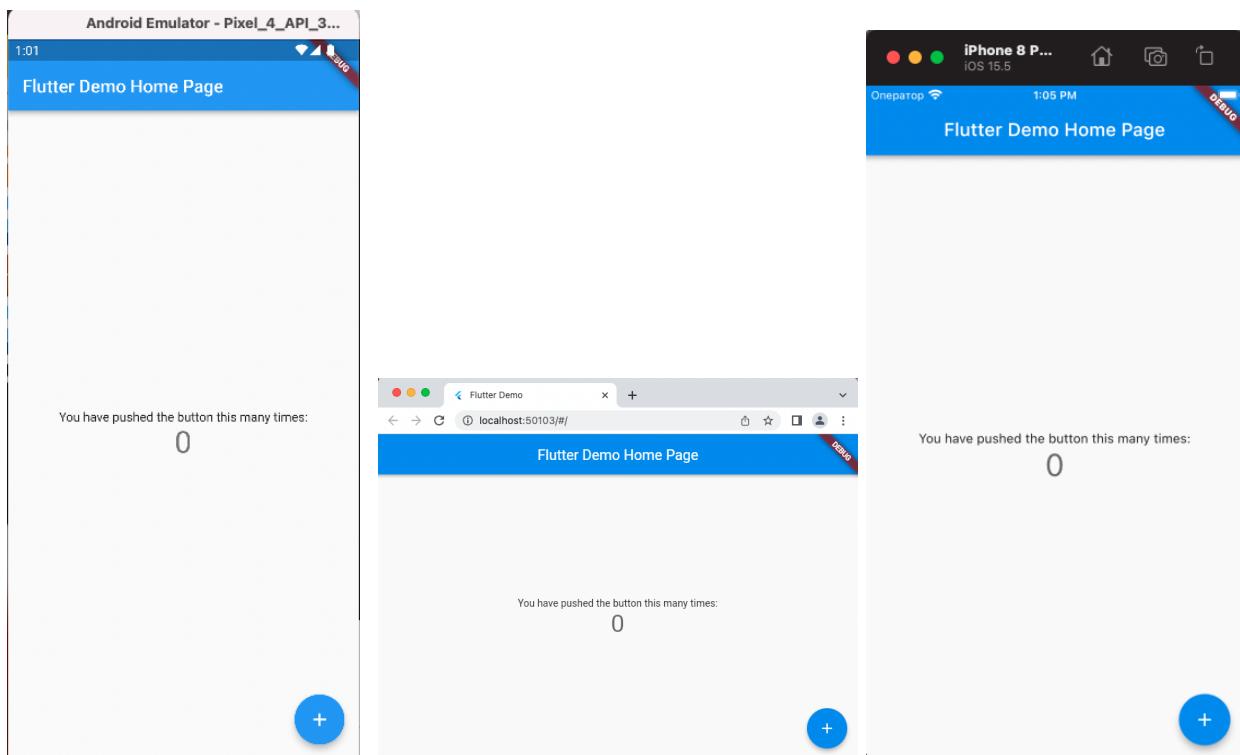
1.	Введение.....	4
1.1.	Создание проекта.....	4
2.	UI KIT .....	4
2.1.	Картинки и иконки .....	4
2.2.	Палитра цветов .....	10
2.3.	Тема.....	12
2.4.	Стили текстов.....	15
3.	Верстка экрана StickerList .....	18
3.1.	Классы данных .....	23
3.2.	Экран StickerList .....	33
3.3.	Верстка AppBar .....	35
3.4.	Верстка текстовых блоков .....	45
3.5.	Верстка поиска.....	49
3.6.	Верстка категорий .....	56
3.7.	Логика клика на категорию .....	61
3.8	Extension .....	65
3.9	Виджет StickerListView .....	67
3.10	Проверка текущей темы.....	71
3.11	Верстка карточки наклейки .....	72
4.	Верстка корзины.....	75
4.1	Экран корзины .....	75
4.2	AppBar экрана корзины.....	76
4.3	Обертка для пустой корзины .....	77
4.4	Данные для корзины.....	80
4.5	Верстка списка корзины.....	83
4.6	Верстка стоимости корзины .....	90
5.	Верстка любимых наклеек .....	94

5.1 Экран FavoriteScreen.....	94
5.2 AppBar и EmptyWrapper экрана любимых наклеек.....	96
5.3 Верстка карточки любимой наклейки .....	98
5.4 Экран профиля .....	102
6. Верстка деталей .....	103
6.1 Экран деталей наклейки.....	103
6.2 AppBar экрана деталей .....	105
6.3 Информация о наклейке.....	109
7. Навигация.....	115
7.1 TabBar .....	115
7.2 TabBar навигация.....	119
7.3 Навигация на детальный просмотр страницы .....	120
8. Бизнес логика без доп. Библиотек .....	123
8.1 Структура состояния и данные первого экрана.....	123
8.2 Подсветка категории на экране списка наклеек .....	126
8.3 Продукты по категориям на экране наклеек.....	128
8.4 Детальный просмотр наклейки .....	129
8.5 Управление количеством на экране деталей .....	130
8.6 Пустая корзина.....	132
8.7 Добавление наклейки в корзину .....	134
8.8 Стоимость наклейки в списке наклеек корзины.....	136
8.9 Стоимость корзины.....	138
8.10 Изменение количества в корзине.....	141
8.11 Удаление наклейки из корзины.....	143
8.12 Чистка корзины по клику на Checkout.....	145
8.13 Пустой экран любимых наклеек.....	147
8.14 Добавление наклейки в любимые наклейки .....	148
8.15 Переключение темы с ValueNotifier.....	152

# 1. ВВЕДЕНИЕ

## 1.1. СОЗДАНИЕ ПРОЕКТА

Создаем новый проект `sun_stikers`, подключаем iOS, Android, Web. Проверяем, что проект собирается для всех трех платформ.



Теперь можно начинать разработку.

## 2. UI KIT

### 2.1. КАРТИНКИ И ИКОНКИ

Предполагаем, что мы не будем тратить время на поиски картинок и иконок, а воспользуемся папкой с готовыми картинками и иконками (архивный файл `assets` прилагается). В папке `assets` две папки `fonts` (с иконками) и `images` (с картинками):

- 1) `fonts`
- 2) `images`

В корень проекта добавляем папку `assets`. Подключим сначала картинки.

В файле pubspec.yaml находим

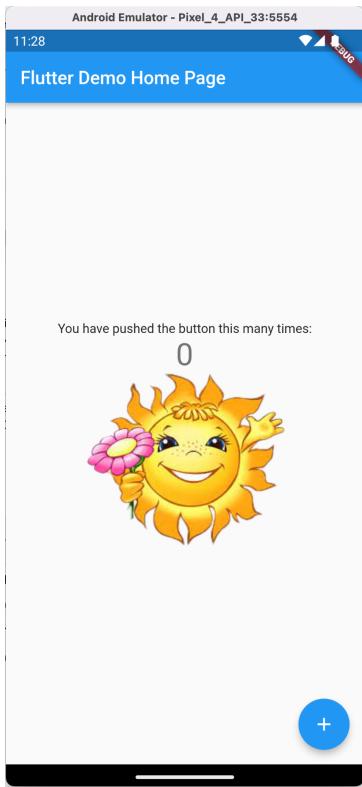
```
# assets:  
#   - images/a_dot_burr.jpeg  
#   - images/a_dot_ham.jpeg
```

и заменяем на

```
assets:  
  - assets/images/
```

Теперь следует сделать Hot Restart и можно пользоваться картинками. В файле main.dart добавим картинку в существующий стартовый проект.

```
child: Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  children: <Widget>[  
    const Text(  
      'You have pushed the button this many times:',  
    ),  
    Text(  
      '_counter',  
      style: Theme.of(context).textTheme.headlineMedium,  
    ),  
    Image.asset('assets/images/profile_pic.png')  
  ],  
) ,
```



### Из правил хорошего тона.

Пользоваться картинками таким образом (прямыми ссылки на assets) считается дурным тоном. Весь код проекта располагается в папке lib. В корне этой папки создадим папку ui\_kit, где будем располагать файлы поддержки ui. В папке ui\_kit создаем файл app\_assets.dart. Для того, чтобы подключение зависимостей в шапке каждого файла было компактным, в папке ui\_kit создадим вспомогательный файл \_ui\_kit.dart. В этом файле будем собирать зависимости для подключения всех зависимостей папки одной строкой.

### Листинг файла lib/ui\_kit/app\_assets.dart.

```
class AppAsset {  
  const AppAsset._();  
  
  static const apple = "assets/images/apple.png";  
  static const ball = "assets/images/ball.png";  
  static const balloon = "assets/images/balloon.png";  
  static const bear = "assets/images/bear.png";  
  static const berry = "assets/images/berry.png";  
  static const dandelion = "assets/images/dandelion.png";  
  static const dolphin = "assets/images/dolphin.png";  
  static const dinosaur = "assets/images/dinosaur.png";  
  static const elephant = "assets/images/elephant.png";  
  static const firtree = "assets/images/firtree.png";  
  static const fish = "assets/images/fish.png";  
  static const flower = "assets/images/flower.png";
```

```
static const home = "assets/images/home.png";
static const mushroom = "assets/images/mushroom.png";
static const pear = "assets/images/pear.png";
static const penguin = "assets/images/penguin.png";
static const raccoon = "assets/images/raccoon.png";
static const snail = "assets/images/snail.png";
static const star = "assets/images/star.png";
static const train = "assets/images/train.png";
static const umbrella = "assets/images/umbrella.png";
static const watermelon = "assets/images/watermelon.png";
static const profileImage = "assets/images/profile_pic.png";
static const emptyCart = "assets/images/empty_cart.png";
static const emptyFavorite = "assets/images/empty_favorite.png";
}
```

Листинг файла lib/ui\_kit/\_ui\_kit.dart.

```
export 'app_assets.dart';
```

**Правило хорошего тона заключается в том, чтобы в коде не использовать константы.**

В классе просто сосредоточены все пути к картинкам.

Теперь строку пути к картинке можно взять через статическую переменную класса AppAssets.

```
import 'package:flutter/material.dart';
import 'package:sun_stickers/ui_kit/_ui_kit.dart';

...
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    const Text(
      'You have pushed the button this many times:',
    ),
    Text(
      '$_counter',
      style: Theme.of(context).textTheme.headlineMedium,
    ),
    Image.asset(AppAsset.profileImage)
  ],
),
```

Подключим иконки. Преобразовать иконки svg в шрифт можно сделать с использованием нескольких онлайн ресурсов.

Например можно использовать <https://www.fluttericon.com/>. Как пользоваться генератором сейчас обсуждать не будем. В результате генерации получаем ttf файл и соответствующий класс. ttf файл мы считаем готовым (в папке fonts). Соответствующий класс поставляется генератором.

В файле pubspec.yaml находим пример подключения шрифтов

```
# To add custom fonts to your application, add a fonts
# section here,
# in this "flutter" section. Each entry in this list
# should have a
# "family" key with the font family name, and a "fonts"
# key with a
# list giving the asset and other descriptors for the
# font. For
# example:
# fonts:
#   - family: Schyler
#     fonts:
#       - asset: fonts/Schyler-Regular.ttf
#       - asset: fonts/Schyler-Italic.ttf
#         style: italic
#   - family: Trajan Pro
#     fonts:
#       - asset: fonts/TrajanPro.ttf
#       - asset: fonts/TrajanPro_Bold.ttf
#         weight: 700
# 
```

И по примеру подключаем файл шрифта из папки fonts.

```
fonts:
  - family: AppIcon
    fonts:
      - asset: assets/fonts/AppIcon.ttf
```

В папку ui\_kit добавляем файл app\_icon.dart и добавляем в него класс из генератора.

Листинг файла lib/ui\_kit/app\_icon.dart:

```
import 'package:flutter/material.dart';

class AppIcon {
  AppIcon._();

  static const _kFontFam = 'AppIcon';
  static const String? _kFontPkg = null;

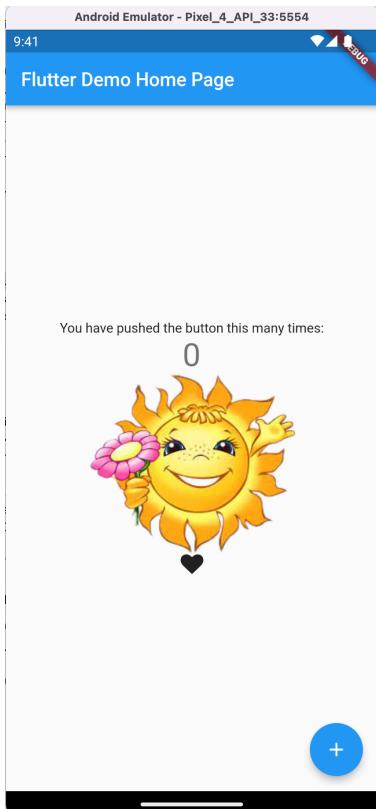
  static const IconData heart = IconData(
    0xe801,
    fontFamily: _kFontFam,
    fontPackage: _kFontPkg,
  );
  static const IconData outlinedHeart = IconData(
    0xe802,
    fontFamily: _kFontFam,
    fontPackage: _kFontPkg,
  );
}
```

Листинг файла lib/ui\_kit/\_ui\_kit.dart.

```
export 'app_assets.dart';
export 'app_icon.dart';
```

Теперь следует сделать Hot Restart и можно пользоваться иконками. В файле main.dart добавим иконку прямо под картинкой.

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    const Text(
      'You have pushed the button this many times:',
    ),
    Text(
      '_$_counter',
      style: Theme.of(context).textTheme.headlineMedium,
    ),
    Image.asset(AppAsset.profileImage),
    const Icon(AppIcon.heart)
  ],
),
```



Сделаем первый коммит.

```
git branch  
* master
```

## 2.2. ПАЛИТРА ЦВЕТОВ

Создаем новую ветку

```
git checkout -b sun_2_2
```

**Из правил хорошего тона. К правилам форошего тона относится создание файла со всеми цветами, которые используются в приложении.**

В папке ui\_kit создаем новый файл app\_color.dart.  
Листинг файла app\_color.dart.

```
import 'package:flutter/material.dart';  
  
class AppColor {  
  const AppColor._();  
  
  static const primaryDark = Color(0xFF18172B);
```

```
static const dark = Color(0xFF1F1F30);
static const primaryLight = Color(0xFFFFFFFF);
static const light = Color(0xFFFFf3f6fa);
static const accent = Color(0xFFFFD8629);
static const yellow = Color(0xFFFFBA49);
}
```

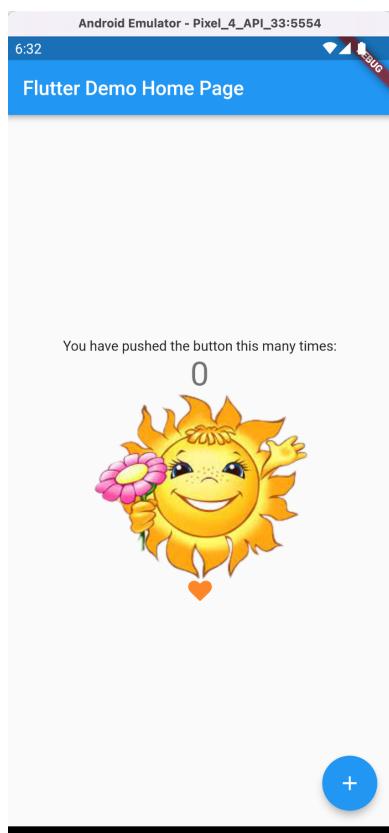
В файл `_ui_kit.dart` добавим `export` нового файла в папке `ui`.

Листинг файла `_ui_kit.dart`.

```
export 'app_assets.dart';
export 'app_icon.dart';
export 'app_color.dart';
```

Проверим что палитра цветов доступна в приложении.  
Расскрасим оранжевым иконку.

```
...
const Icon(
  AppIcon.heart,
  color: AppColor.accent,
)
...
```



```
% git add .
% git commit -m'Палитра'
% git branch
* sun_2_2
  master
```

## 2.3. ТЕМА

Создаем новую ветку

```
git checkout -b sun_2_3
```

В папке ui\_kit создадим новый файл app\_theme.dart.  
Листинг файла app\_theme.dart:

```
import 'package:flutter/material.dart';

class AppTheme {
  const AppTheme._();

  static ThemeData lightTheme = ThemeData();
  static ThemeData darkTheme = ThemeData();
}
```

В файл \_ui\_kit.dart добавим новый файл.

Листинг файла \_ui\_kit.dart:

```
export 'app_assets.dart';
export 'app_color.dart';
export 'app_icon.dart';
export 'app_theme.dart';
```

Структура файла app\_theme.dart ровно такая как у всех файлов папки ui\_kit. ThemeData() – содержит дефолтные настройки и подключение такой темы не поменяет цвета и стили заданные по умолчанию. Добавим к светлой теме одно свойство и подключим тему к приложению.

Свойства, которые можно переопределять в теме можно посмотреть нажав на ThemeData(). Выберем одно из свойств и определим в светлой теме оранжевым.

Листинг файла app\_theme.dart.

```
import '_ui_kit.dart';
import 'package:flutter/material.dart';

class AppTheme {
  const AppTheme._();

  static ThemeData lightTheme = ThemeData(indicatorColor:
AppColor.accent);

  static ThemeData darkTheme = ThemeData();
}
```

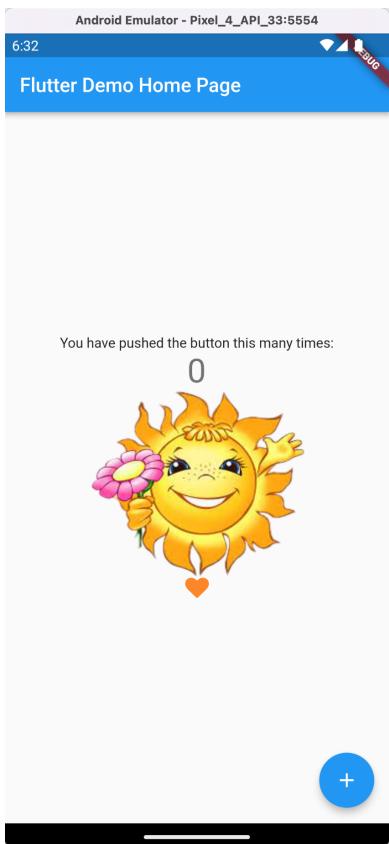
Подключим к приложению светлую тему. Для этого в MaterialApp добавляем тему.

В файле main.dart:

```
MaterialApp(
  title: 'Flutter Demo',
  theme: AppTheme.lightTheme,
  home: const MyHomePage(title: 'Flutter Demo Home Page'),
);
```

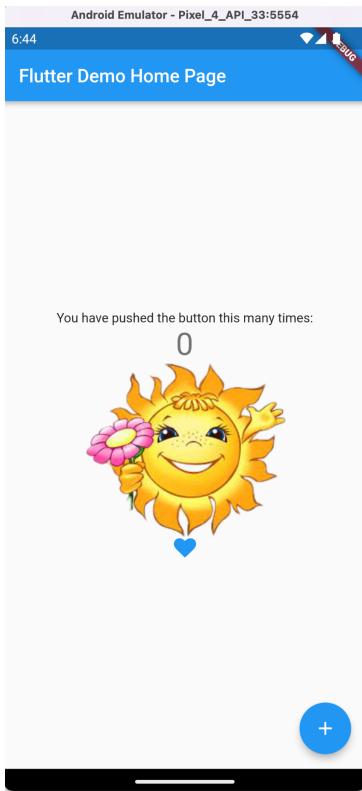
Цвет иконки возьмем из темы:

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    const Text(
      'You have pushed the button this many times:',
    ),
    Text(
      '$_counter',
      style: Theme.of(context).textTheme.headlineMedium,
    ),
    Image.asset(AppAsset.profileImage),
    Icon(
      AppIcon.heart,
      color: Theme.of(context).indicatorColor,
    ),
  ],
),
```



Подключим вторую тему

```
MaterialApp(  
    title: 'Flutter Demo',  
    theme: AppTheme.darkTheme,  
    home: const MyHomePage(title: 'Flutter Demo Home  
Page'),  
);
```



Цвет иконки синий, параметр `indicatorColor` для темной темы мы не определяли,

```
static ThemeData lightTheme = ThemeData(indicatorColor:  
AppColor.accent);
```

```
static ThemeData darkTheme = ThemeData();
```

это значит, что синий цвет, это дефолтный цвет параметра `indicatorColor` для `ThemeData()`.

```
% git add .  
% git commit -m'Тема'  
% git branch  
  sun_2_2  
* sun_2_3  
  master
```

## 2.4. СТИЛИ ТЕКСТОВ

Создаем новую ветку

```
git checkout -b sun_2_4
```

В папку ui\_kit добавим новый файл app\_text\_style.dart. Листинг файла app\_text\_style.dart.

```
import 'package:flutter/material.dart';

class AppTextStyle {
  const AppTextStyle._();

  static TextStyle h1Style = const TextStyle(
    fontSize: 25,
    fontWeight: FontWeight.bold,
    color: Colors.black,
  );

  static TextStyle h2Style = const TextStyle(
    fontSize: 19,
    fontWeight: FontWeight.bold,
    color: Colors.black,
  );

  static TextStyle h3Style = const TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
    color: Colors.black,
  );

  static TextStyle h4StyleLight = const TextStyle(
    fontSize: 15,
    fontWeight: FontWeight.w500,
    color: Colors.black,
  );

  static TextStyle h5StyleLight = const TextStyle(
    fontSize: 17,
    fontWeight: FontWeight.w400,
    color: Colors.black87,
  );

  static TextStyle bodyTextLight = const TextStyle(
    fontSize: 13,
    fontWeight: FontWeight.w600,
    color: Colors.black45,
  );

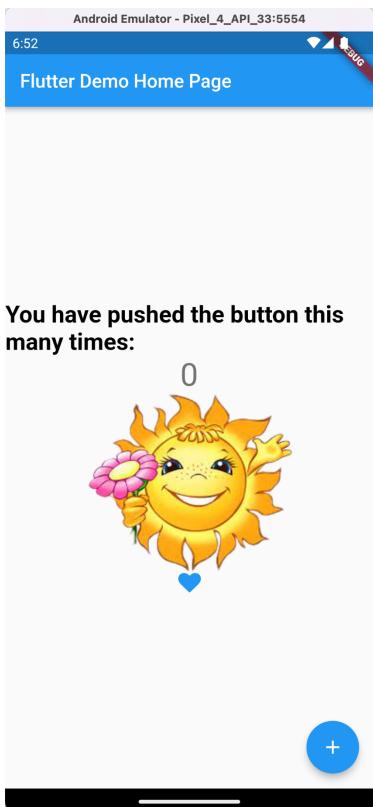
  static TextStyle subtitleLight = const TextStyle(
    fontSize: 12,
    fontWeight: FontWeight.bold,
    color: Colors.black45,
  );
}
```

В файл `_ui_kit.dart` добавим новый файл.  
Листинг файла `_ui_kit.dart`.

```
export 'app_assets.dart';
export 'app_color.dart';
export 'app_icon.dart';
export 'app_theme.dart';
export 'app_text_style.dart';
```

Применим один из стилей к тексту на экране.

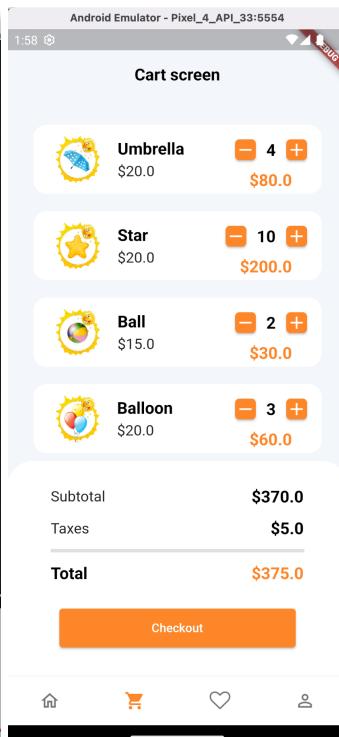
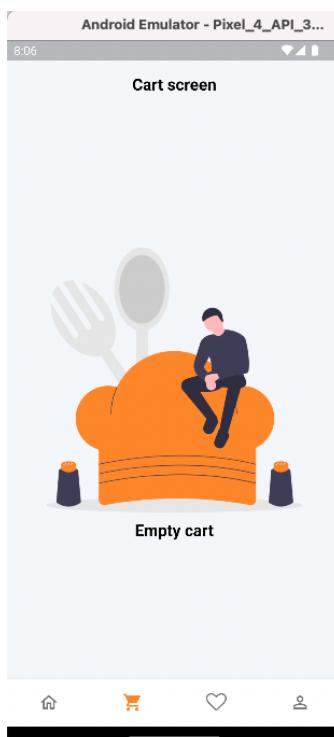
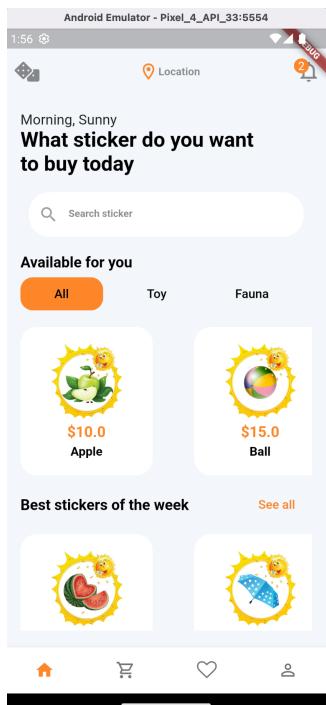
```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Text(
      'You have pushed the button this many times:',
    style: AppTextStyle.h1Style,
    ),
    Text(
      '$_counter',
      style: Theme.of(context).textTheme.headlineMedium,
    ),
    Image.asset(AppAsset.profileImage),
    Icon(
      AppIcon.heart,
      color: Theme.of(context).indicatorColor,
    )
  ],
),
```

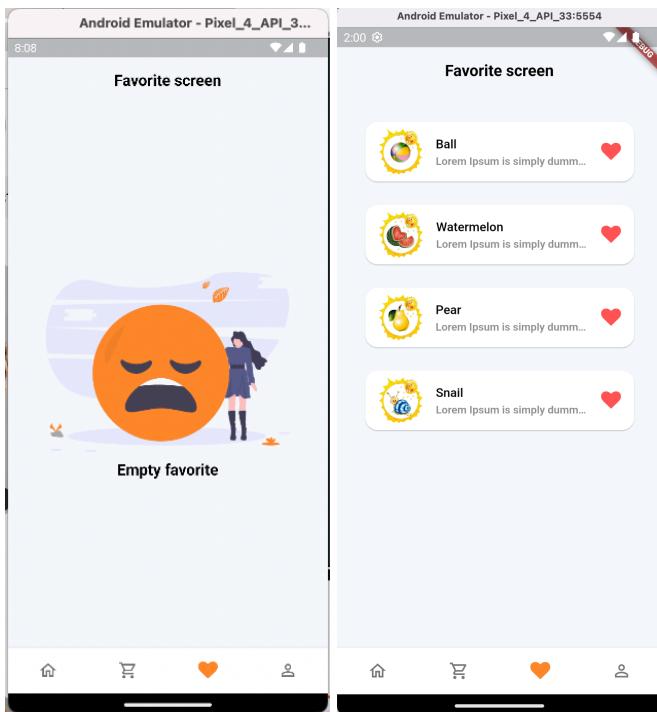


```
% git add .
% git commit -m'Стили текстов'
% git branch
  sun_2_2
  sun_2_3
* sun_2_4
  master
```

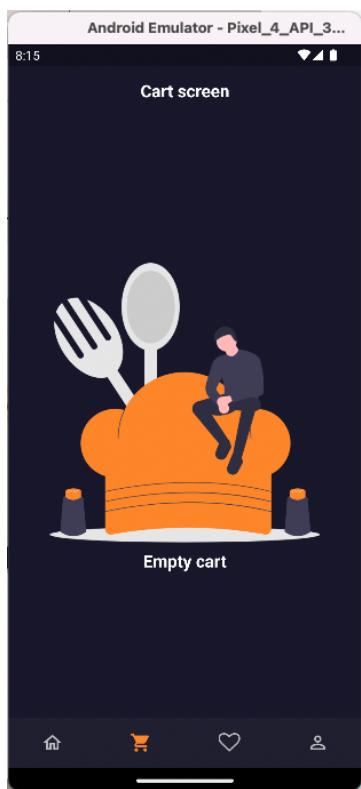
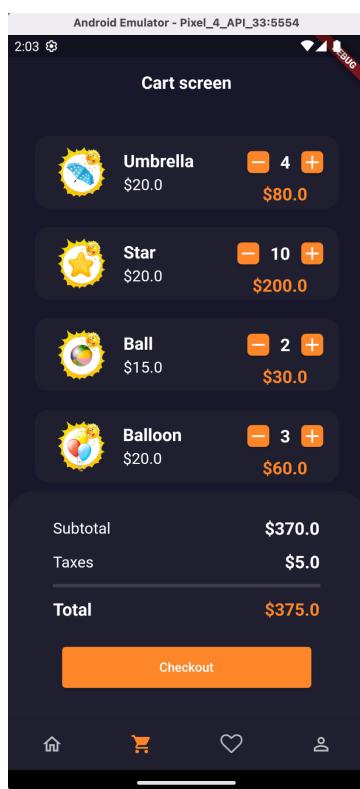
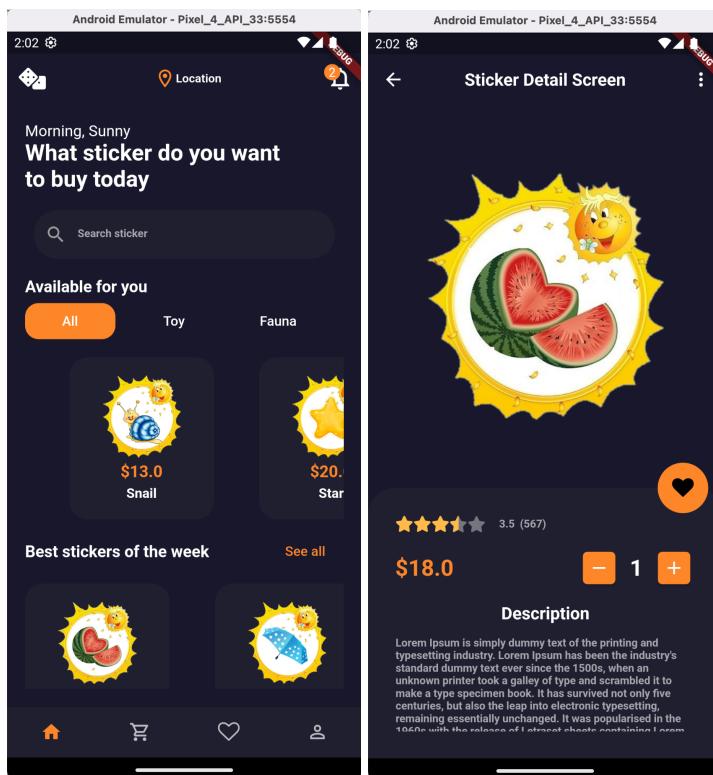
### 3. ВЕРСТКА ЭКРАНА STICKERLIST

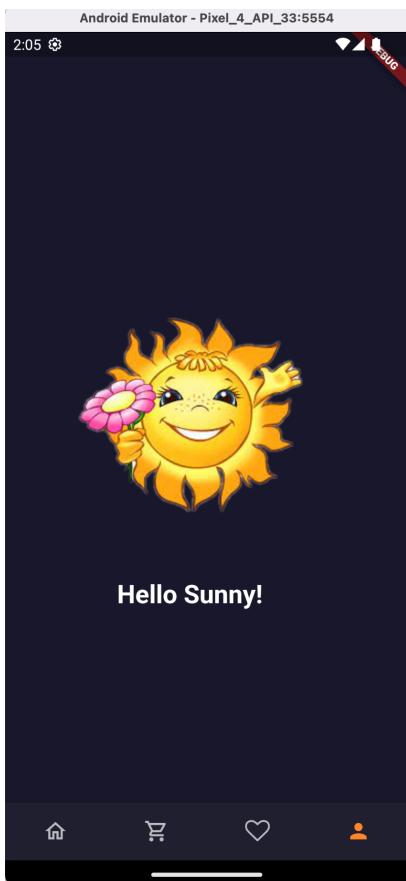
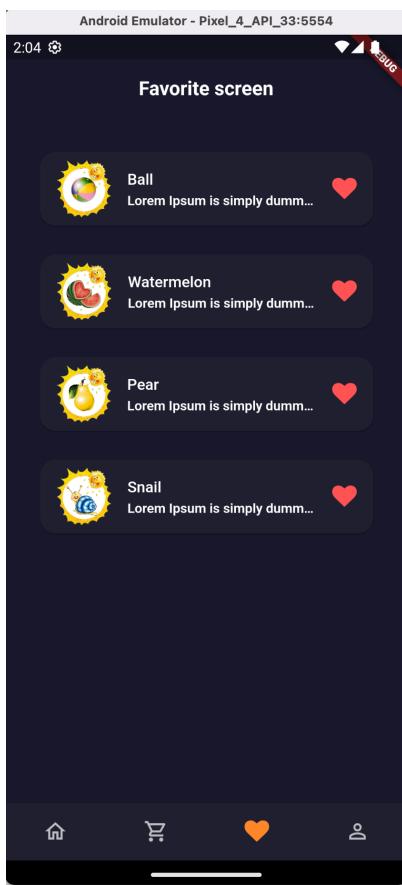
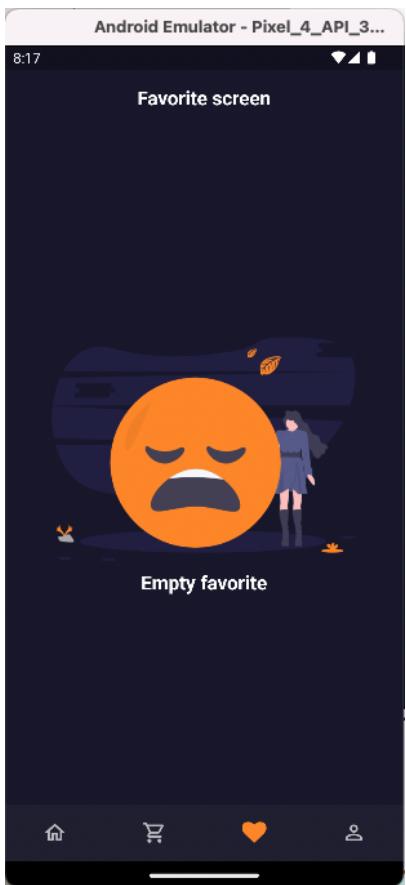
Экраны приложения в светлой теме:





Экраны приложения в темной теме:



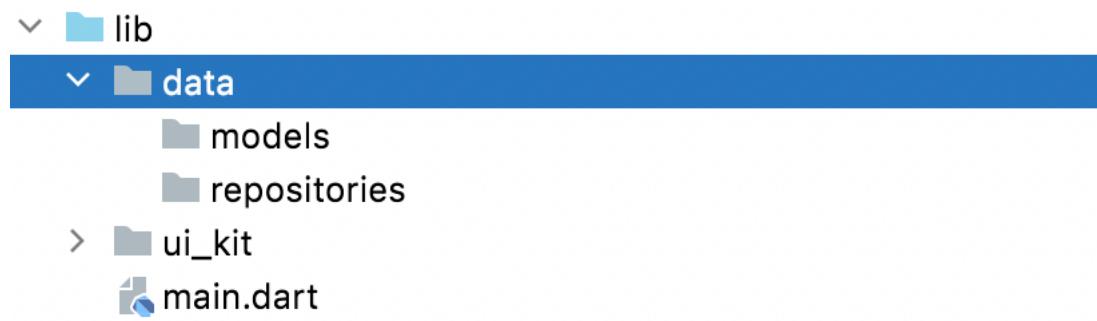


### 3.1. КЛАССЫ ДАННЫХ

Создаем новую ветку

```
git checkout -b sun_3_1
```

Организуем слой данных. В папке lib создадим папку data.



В папке models расположим классы данных, которые используются в приложении. В папке repositories должны быть созданы сервисы, которые запрашивают данные (источником данных могут быть сетевые запросы или локальные хранилища) и преобразуют те данные которые пришли в данные, которые используются в приложении. Данные которые прилетают по сетевым запросам обычно называют DTO (data transfer object), в репозиториях DTO преобразовывают в модели (данные, которые используются в приложении).

В папку data добавим файл \_data.dart. Этот файл будет содержать export файлов из этой папки.

Создадим три модели (Sticker, StickerCategory, BottomNavigationItem). Это типы данных которые будут использоваться в приложении.

Листинг файла data/models/sticker.dart.

```
enum StickerType { all, toy, fauna, plant, berry, fruit, other }

class Sticker {
    int id;
    String image;
    String name;
    double price;
    int quantity;
    bool isFavorite;
    String description;
    double score;
    StickerType type;
```

```
    int voter;
    bool cart;

    Sticker(
        this.id,
        this.image,
        this.name,
        this.price,
        this.quantity,
        this.isFavorite,
        this.description,
        this.score,
        this.type,
        this.voter,
        this.cart,
    );
}
```

Листинг файла `data/models/_models.dart`.

```
export 'sticker.dart';
```

Листинг файла `data/models/sticker_category.dart`.

```
import '_models.dart';

class StickerCategory {
    final StickerType type;
    bool isSelected;

    StickerCategory(this.type, this.isSelected);
}
```

Листинг файла `data/models/_models.dart`.

```
export 'sticker.dart';
export 'sticker_category.dart';
```

Листинг файла `data/models/bottom_navigation_item.dart`.

```
import 'package:flutter/material.dart';

class BottomNavigationItem {
    final Widget disableIcon;
    final Widget enableIcon;
    String label;
    bool isSelected;
```

```
        BottomNavigationItem(  
            this.disableIcon,  
            this.enableIcon,  
            this.label,  
            {  
                this.isSelected = false,  
            }  
        );  
    }  
}
```

Листинг файла `data/models/_models.dart`.

```
export 'bottom_navigation_item.dart';  
export 'sticker.dart';  
export 'sticker_category.dart';
```

Реализацией сервисов запроса данных в папке `repositories` займемся позже, а сейчас для доступа к данным в корне папки `data` создадим файл `app_data.dart`.

Листинг файла `data/app_data.dart`:

```
import 'package:flutter/material.dart';  
  
import '../ui_kit/_ui_kit.dart';  
import 'models/_models.dart';  
  
class AppData {  
    const AppData._();  
  
    static const dummyText = "Lorem Ipsum is simply dummy text of  
the printing and typesetting "  
        "industry. Lorem Ipsum has been the industry's standard  
dummy text ever "  
        "since the 1500s, when an unknown printer took a galley of  
type and "  
        "scrambled it to make a type specimen book. It has survived  
not only five "  
        "centuries, but also the leap into electronic typesetting,  
remaining "  
        "essentially unchanged. It was popularised in the 1960s with  
the release "  
        "of Letraset sheets containing Lorem Ipsum passages, and  
more recently "  
        "with desktop publishing software like Aldus PageMaker  
including versions "  
        "of Lorem Ipsum.";  
  
    static List<Sticker> stickers = [  
        Sticker(
```

```
1,
AppAsset.apple,
"Apple",
10.0,
1,
false,
dummyText,
5.0,
StickerType.fruit,
150,
false,
),
Sticker(
2,
AppAsset.ball,
"Ball",
15.0,
1,
false,
dummyText,
3.5,
StickerType.toy,
652,
false,
),
Sticker(
3,
AppAsset.balloon,
"Balloon",
20.0,
1,
false,
dummyText,
4.0,
StickerType.toy,
723,
false,
),
Sticker(
4,
AppAsset.bear,
"Bear",
40.0,
1,
false,
dummyText,
2.5,
StickerType.toy,
456,
false,
),
Sticker(
```

```
5,
AppAsset.berry,
"Strawberry",
10.0,
1,
false,
dummyText,
4.5,
StickerType.berry,
650,
false,
),
Sticker(
6,
AppAsset.dandelion,
"Dandelion",
20.0,
1,
false,
dummyText,
1.5,
StickerType.plant,
350,
false,
),
Sticker(
7,
AppAsset.dinosaur,
"Dinosaur",
12.0,
1,
false,
dummyText,
3.5,
StickerType.fauna,
265,
false,
),
Sticker(
8,
AppAsset.dolphin,
"Dolphin",
30.0,
1,
false,
dummyText,
4.0,
StickerType.fauna,
890,
false,
),
Sticker(
```

```
9,
AppAsset.elephant,
"Elephant",
10.0,
1,
false,
dummyText,
5.0,
StickerType.fauna,
900,
false,
),
Sticker(
10,
AppAsset.firtree,
"Fir-tree",
15.0,
1,
false,
dummyText,
3.5,
StickerType.plant,
420,
false,
),
Sticker(
11,
AppAsset.fish,
"Fish",
25.0,
1,
false,
dummyText,
3.0,
StickerType.fauna,
263,
false,
),
Sticker(
12,
AppAsset.flower,
"Flower",
20.0,
1,
false,
dummyText,
5.0,
StickerType.plant,
560,
false,
),
Sticker(
```

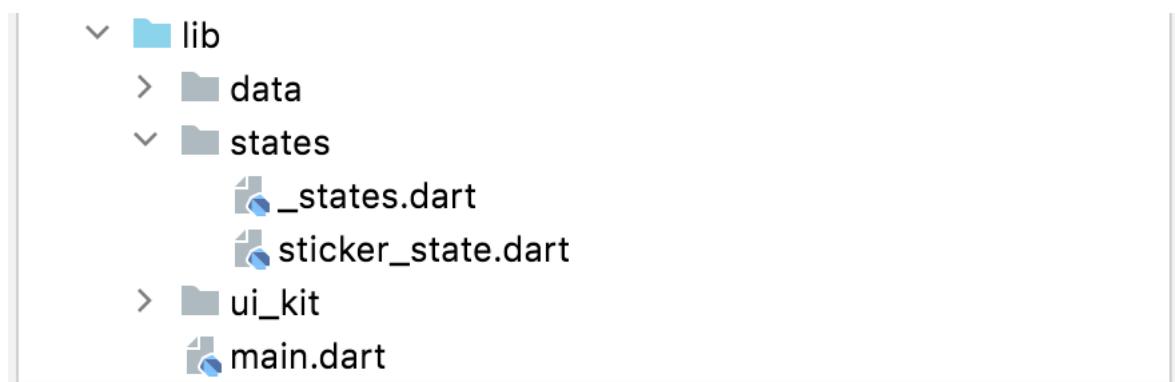
```
13,
AppAsset.home,
"Home",
15.0,
1,
false,
dummyText,
2.5,
StickerType.other,
361,
false,
),
Sticker(
14,
AppAsset.mushroom,
"Mushroom",
12.0,
1,
false,
dummyText,
4.5,
StickerType.other,
915,
false,
),
Sticker(
15,
AppAsset.pear,
"Pear",
10.0,
1,
false,
dummyText,
3.5,
StickerType.fruit,
210,
false,
),
Sticker(
16,
AppAsset.penguin,
"Penguin",
5.0,
1,
false,
dummyText,
4.0,
StickerType.fauna,
304,
false,
),
Sticker(
```

```
17,
AppAsset.raccoon,
"Raccoon",
15.0,
1,
false,
dummyText,
4.5,
StickerType.fauna,
356,
false,
),
Sticker(
18,
AppAsset.snail,
"Snail",
13.0,
1,
false,
dummyText,
3.0,
StickerType.fauna,
203,
false,
),
Sticker(
19,
AppAsset.star,
"Star",
20.0,
1,
false,
dummyText,
1.0,
StickerType.toy,
278,
false,
),
Sticker(
20,
AppAsset.train,
"Train",
10.0,
1,
false,
dummyText,
1.5,
StickerType.toy,
734,
false,
),
Sticker(
```

```
21,
AppAsset.umbrella,
"Umbrella",
20.0,
1,
false,
dummyText,
2.5,
StickerType.other,
671,
false,
),
Sticker(
22,
AppAsset.watermelon,
"Watermelon",
18.0,
1,
false,
dummyText,
3.5,
StickerType.berry,
567,
false,
),
];
static List<BottomNavigationItem> bottomNavigationItems = [
BottomNavigationItem(
const Icon(Icons.home_outlined),
const Icon(Icons.home),
'Home',
isSelected: true,
),
BottomNavigationItem(
const Icon(Icons.shopping_cart_outlined),
const Icon(Icons.shopping_cart),
'Shopping cart',
),
BottomNavigationItem(
const Icon(AppIcon.outlinedHeart),
const Icon(AppIcon.heart),
'Favorite',
),
BottomNavigationItem(
const Icon(Icons.person_outline),
const Icon(Icons.person),
'Profile',
)
];
static List<StickerCategory> categories = [
```

```
        StickerCategory(StickerType.all, true),
        StickerCategory(StickerType.toy, false),
        StickerCategory(StickerType.fauna, false),
        StickerCategory(StickerType.plant, false),
        StickerCategory(StickerType.berry, false),
        StickerCategory(StickerType.fruit, false),
        StickerCategory(StickerType.other, false),
    ];
}
```

В корне lib создадим папку states. В этой папке будут находиться файлы управления состоянием приложения. Создадим в этой папке файл sticker\_state.dart.



Листинг файла states/sticker\_state.dart.

```
class StickerState {
    StickerState._();
    static final _instance = StickerState._();
    factory StickerState() => _instance;
}
```

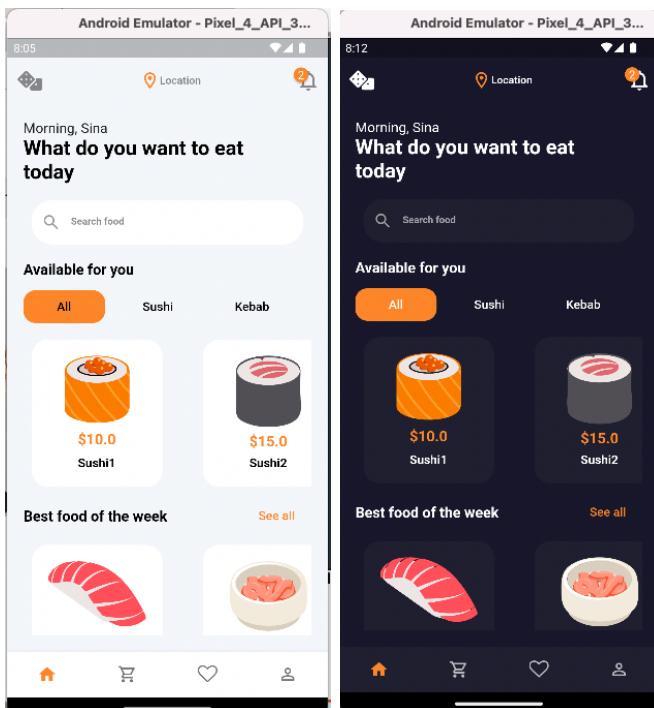
Класс StickerState реализует паттерн программирования – синглтон (одиночка). Вызов конструктора возвращает один и тот же экземпляр класса StickerState.

```
% git add .
% git commit -m'Данные'
% git branch
  sun_2_2
  sun_2_3
  sun_2_4
* sun_3_1
  master
```

### 3.2. ЭКРАН STICKERLIST

```
git checkout -b sun_3_2
```

В корне lib создадим папку ui. В этой папке еще две папки screens и widgets. Папка screens предназначена для экранов, папка widgets для общих виджетов приложения.



В папке screens создаем файл sticker\_list\_screen.dart. Листинг файла sticker\_list\_screen.dart

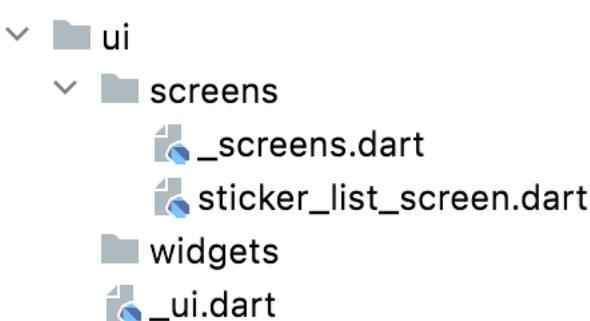
```
import 'package:flutter/material.dart';

class StickerList extends StatefulWidget {
  const StickerList({super.key});

  @override
  State<StatefulWidget> createState() =>
  StickerListState();
}

class StickerListState extends State<StickerList> {
  @override
```

```
    Widget build(BuildContext context) => Scaffold();  
}
```



В папке ui создаем вспомогательный файл \_ui.dart. В папке screens – вспомогательный файл \_screens.dart. В папке widgets – вспомогательный файл \_widgets.dart.

Листинг файла lib/ui/screens/\_screens.dart

```
export 'sticker_list_screen.dart';
```

Листинг файла lib/ui/\_ui.dart

```
export 'screens/_screens.dart';
```

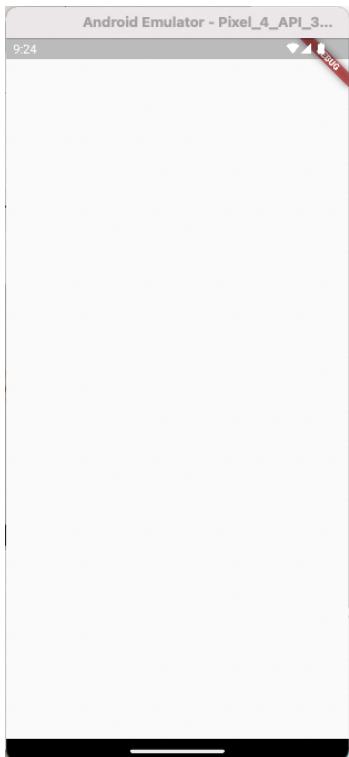
Листинг файла lib/main.dart

```
import 'package:flutter/material.dart';  
  
import 'ui/_ui.dart';  
  
void main() {  
  runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      title: 'Sunny Stickers',  
      //theme: AppTheme.lightTheme,  
      home: StickerList(),  
    );  
  }  
}
```

Только что созданный экран StickerList подключаем как первый экран:

```
home: StickerList(),
```

Запускаем проект.



```
% git add .
% git commit -m'Экран StickerList'
% git branch
  sun_2_2
  sun_2_3
  sun_2_4
  sun_3_1
* sun_3_2
  master
```

### 3.3. ВЕРСТКА APPBAR

```
git checkout -b sun_3_3
```

Для верстки кубиков в левом верхнем углу и бейджа на колокольчике в правом верхнем углу добавим две библиотеки. В файл pubspec.yaml

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  font_awesome_flutter: ^10.4.0
```

```
badges: ^3.0.3
```

И запустим команду pub get.

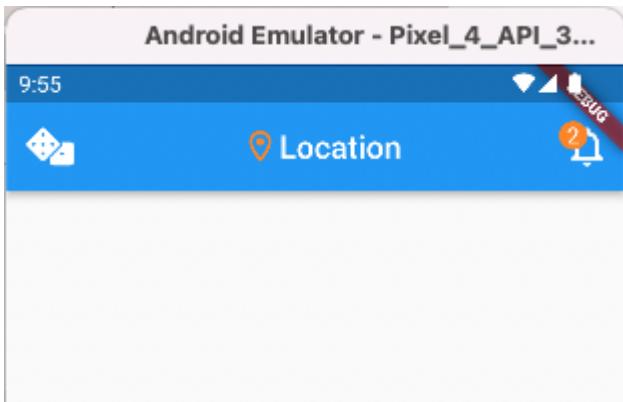
В библиотеке badges используется класс Badge, и в пакете material ( из пакета SDK) тоже есть класс Badge. И для того, чтобы определиться, каким из классов мы собираемся пользоваться, подключение импортов делается следующим образом:

```
import 'package:badges/badges.dart';
import 'package:flutter/material.dart' hide Badge;
```

hide Badge и означает – игнорировать тот класс Badge, который в SDK.

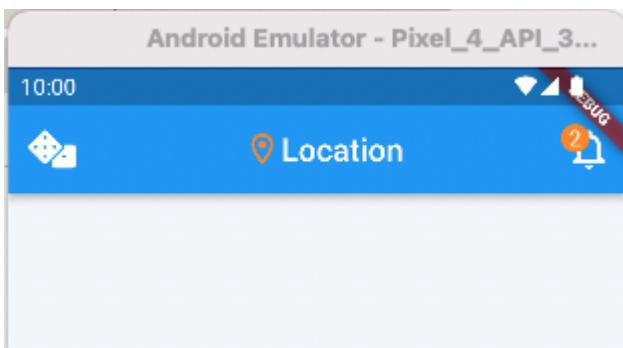
Добавим в Scaffold верстку AppBar

```
@override
Widget build(BuildContext context) => Scaffold(
    appBar: AppBar(
        leading: IconButton(
            icon: const FaIcon(FontAwesomeIcons.dice),
            onPressed: () {},
        ),
        title: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: const [Icon(Icons.location_on_outlined),
                color: AppColor.accent), Text("Location")],
        ),
        actions: [
            IconButton(
                onPressed: () {},
                icon: Badge(
                    badgeStyle: const BadgeStyle(
                        badgeColor: AppColor.accent
                    ),
                    badgeContent: const Text(
                        "2",
                        style: TextStyle(color: Colors.white),
                    ),
                    position: BadgePosition.topStart(start: -3),
                    child: const Icon(Icons.notifications_none,
                        size: 30),
                ),
            ],
        ],
    );
);
```

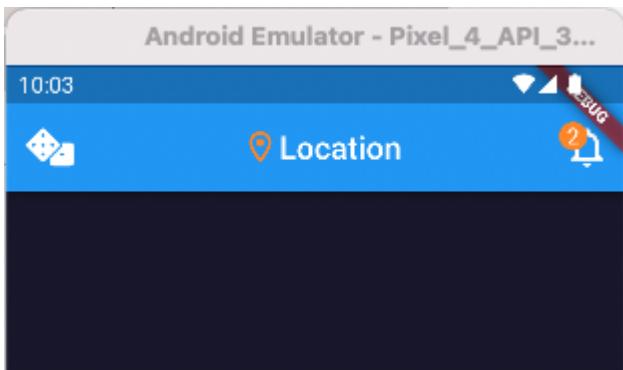


Выставим цвет бекграунда Scaffold

```
Scaffold(  
    backgroundColor: AppColor.primaryLight,  
    appBar: AppBar(  
        ...  
    ),  
);
```



```
Scaffold(  
    backgroundColor: AppColor.primaryDark,  
    appBar: AppBar(  
        ...  
    ),  
);
```



Теперь убираем цвет фона в тему и убираем этот цвет с экрана. Теперь в зависимости от темы фон Scaffold будет либо серый либо черный.

Добавляем в тему.

Листинг lib/ui\_kit/app\_theme.dart

```
import 'package:flutter/material.dart';

import '_ui_kit.dart';

class AppTheme {
  const AppTheme._();

  static ThemeData lightTheme = ThemeData(
    scaffoldBackgroundColor: AppColor.primaryLight
  );

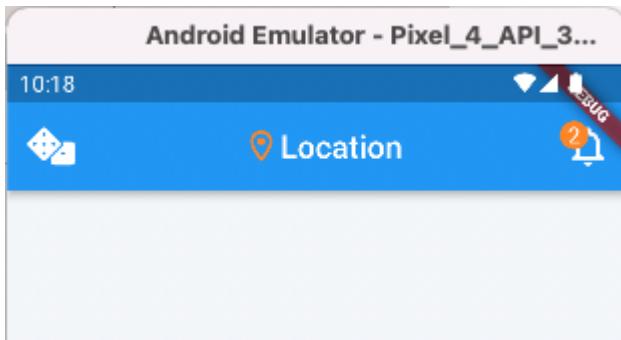
  static ThemeData darkTheme = ThemeData(
    scaffoldBackgroundColor: AppColor.primaryDark
  );
}
```

Удаляем цвет фона с экрана.

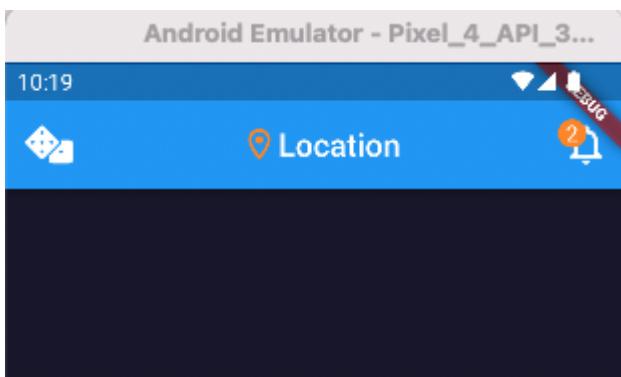
```
Scaffold(
  appBar: AppBar(
    ...
  ),
);
```

Добавляем тему в MaterialApp и проверяем

```
MaterialApp(
  title: 'Sunny Stickers',
  theme: AppTheme.lightTheme,
  home: const StickerList(),
);
```



```
MaterialApp(  
    title: 'Sunny Stickers',  
    theme: AppTheme.darkTheme,  
    home: const StickerList(),  
) ;
```

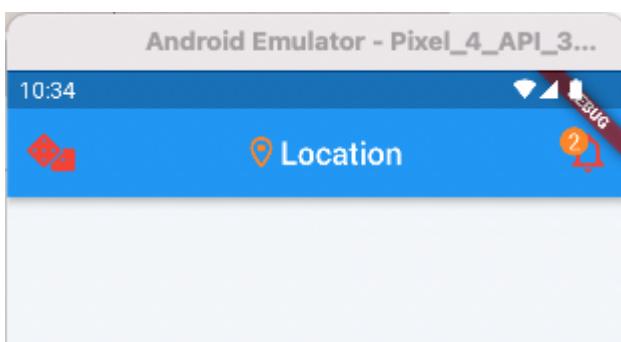


Перейдем к стилизации AppBar.

Изучим некоторые свойства AppBar (все свойства можно получить в списке параметров виджета AppBar)

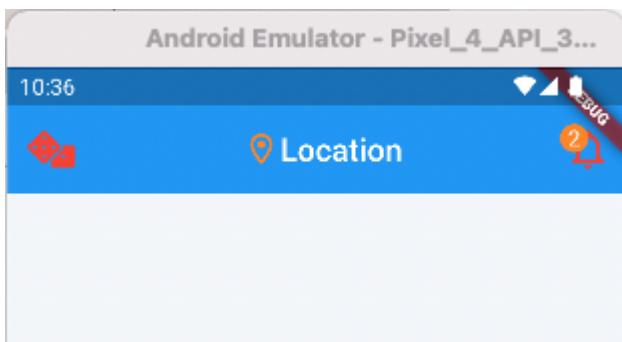
Рассмотрим за что отвечает свойство iconTheme (управление цветом иконки – кубики и колокольчик)

```
appBar: AppBar(  
    iconTheme: IconThemeData(color: Colors.red),  
    ...
```



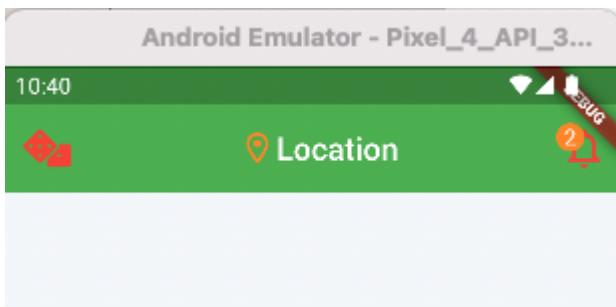
Рассмотрим за что отвечает свойство `elevation` (управление тенью AppBar)

```
appBar: AppBar(  
    iconTheme: IconThemeData(color: Colors.red),  
    elevation: 0,  
    ...
```



Рассмотрим за что отвечает свойство `backgroundColor` (цвет AppBar).

```
appBar: AppBar(  
    iconTheme: IconThemeData(color: Colors.red),  
    elevation: 0,  
    backgroundColor: Colors.green,  
    ...
```



Стиль AppBar перенесем в тему.

Стиль текста "Location" тоже перенесем в тему и в верстке вытащим стиль из текущей темы.

Листинг lib/ui\_kit/app\_theme.dart

```
import 'package:flutter/material.dart';  
  
import '_ui_kit.dart';
```

```
class AppTheme {  
  const AppTheme._();  
  
  static ThemeData lightTheme = ThemeData(  
    scaffoldBackgroundColor: AppColor.primaryLight,  
    appBarTheme: AppBarTheme(  
      backgroundColor: Colors.transparent,  
      elevation: 0,  
      iconTheme: const IconThemeData(color: Colors.black45),  
      centerTitle: true,  
      titleTextStyle: AppTextStyle.h2Style,  
    ),  
    textTheme: TextTheme(  
      displayLarge: AppTextStyle.h1Style,  
      displayMedium: AppTextStyle.h2Style,  
      displaySmall: AppTextStyle.h3Style,  
      headlineMedium: AppTextStyle.h4StyleLight,  
      headlineSmall: AppTextStyle.h5StyleLight,  
      bodyLarge: AppTextStyle.bodyTextLight,  
      titleMedium: AppTextStyle.subtitleLight,  
    ),  
  );  
  
  static ThemeData darkTheme = ThemeData(  
    scaffoldBackgroundColor: AppColor.primaryDark,  
    appBarTheme: AppBarTheme(  
      backgroundColor: Colors.transparent,  
      elevation: 0,  
      toolbarTextStyle: const TextStyle(color: Colors.white),  
      centerTitle: true,  
      iconTheme: const IconThemeData(color: Colors.white),  
      titleTextStyle: AppTextStyle.h2Style,  
    ),  
    textTheme: TextTheme(  
      displayLarge: AppTextStyle.h1Style.copyWith(  
          color: Colors.white),  
      displayMedium: AppTextStyle.h2Style.copyWith(  
          color: Colors.white),  
      displaySmall: AppTextStyle.h3Style.copyWith(  
          color: Colors.white),  
      headlineMedium: AppTextStyle.h4StyleLight.copyWith(  
          color: Colors.white),  
      headlineSmall: AppTextStyle.h5StyleLight.copyWith(  
          color: Colors.white),  
      bodyLarge: AppTextStyle.bodyTextLight.copyWith(  
          color: Colors.white),  
      titleMedium: AppTextStyle.subtitleLight.copyWith(  
          color: Colors.white60),  
    ),  
  );  
}
```

## Листинг ui/screens/sticker\_list\_screen.dart

```
import 'package:badges/badges.dart';
import 'package:flutter/material.dart' hide Badge;
import 'package:font_awesome_flutter/font_awesome_flutter.dart';

import '../../ui_kit/_ui_kit.dart';

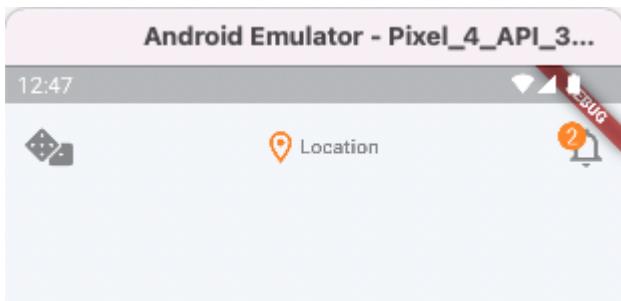
class StickerList extends StatefulWidget {
  const StickerList({super.key});

  @override
  State<StatefulWidget> createState() => StickerListState();
}

class StickerListState extends State<StickerList> {
  @override
  Widget build(BuildContext context) => Scaffold(
    appBar: AppBar(
      leading: IconButton(
        icon: const FaIcon(FontAwesomeIcons.dice),
        onPressed: () {},
      ),
      title: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const Icon(Icons.location_on_outlined,
            color: AppColor.accent),
          Text(
            "Location",
            style: Theme.of(context).textTheme.bodyLarge,
          )
        ],
      ),
      actions: [
        IconButton(
          onPressed: () {},
          icon: Badge(
            badgeStyle: const BadgeStyle(
              badgeColor: AppColor.accent),
            badgeContent: const Text(
              "2",
              style: TextStyle(color: Colors.white),
            ),
            position: BadgePosition.topStart(start: -3),
            child: const Icon(Icons.notifications_none, size: 30),
          ),
        ),
      ],
    );
}
```

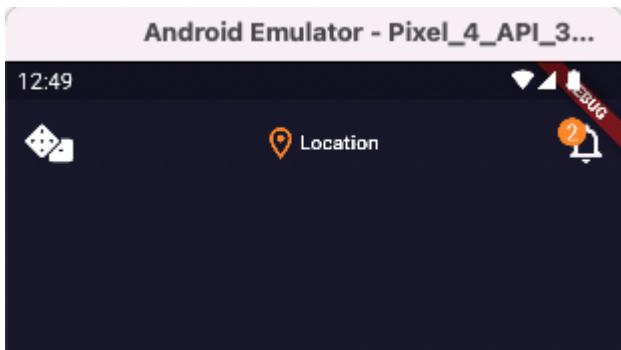
Проверим AppBar для светлой и темной темы. Для этого выставляем светлую тему в main.dart.

```
MaterialApp(  
    title: 'Sunny Stickers',  
    theme: AppTheme.lightTheme,  
    home: const StickerList(),  
) ;
```



Выставляем темную тему в main.dart.

```
MaterialApp(  
    title: 'Sunny Stickers',  
    theme: AppTheme.darkTheme,  
    home: const StickerList(),  
) ;
```



Синтаксис для получения стиля текста из текущей темы следующий:

```
Text(  
    "Location",  
    style: Theme.of(context).textTheme.bodyLarge,  
)
```

Чтобы верстка легче читалась, вынесем верстку AppBar в отдельный метод.

```
PreferredSizeWidget _appBar(BuildContext context) {
  return AppBar(
    leading: IconButton(
      icon: const FaIcon(FontAwesomeIcons.dice),
      onPressed: controller.changeTheme,
    ),
    title: Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Icon(Icons.location_on_outlined,
          color: AppColor.accent),
        Text("Location",
          style: Theme.of(context).textTheme.bodyLarge)
      ],
    ),
    actions: [
      IconButton(
        onPressed: () {},
        icon: Badge(
          badgeStyle: const BadgeStyle(
            badgeColor: AppColor.accent),
          badgeContent: const Text(
            "2",
            style: TextStyle(color: Colors.white),
          ),
          position: BadgePosition.topStart(start: -3),
          child: const Icon(Icons.notifications_none, size: 30),
        ),
      ),
    ],
  );
}
```

После чего листинг экрана станет более читабельным.  
Листинг ui/screens/sticker\_list\_screen.dart.

```
class StickerListState extends State<StickerList> {
  @override
  Widget build(BuildContext context) => Scaffold(
    appBar: _appBar(context),
  );

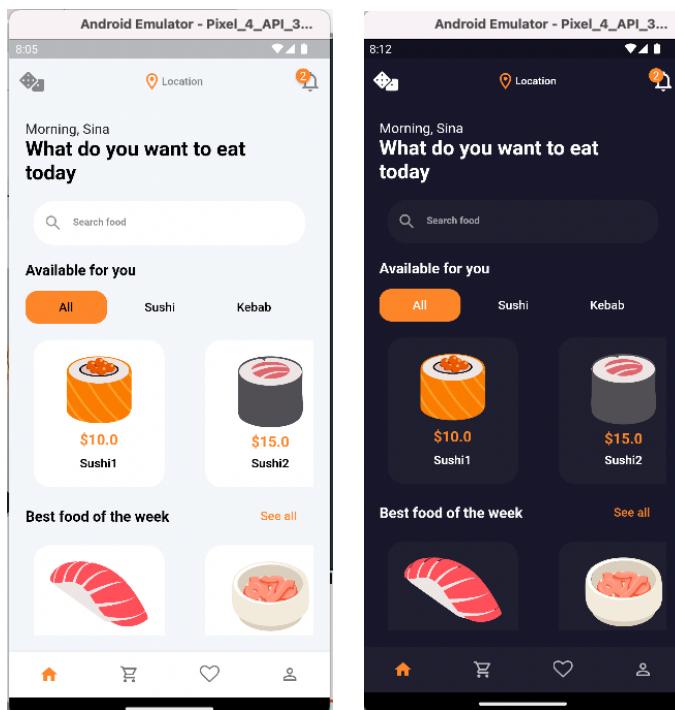
  PreferredSizeWidget _appBar(BuildContext context){
    return AppBar(
      ...
    );
  }
}
```

```
% git add .
% git commit -m'AppBar'
% git branch
  sun_2_2
  sun_2_3
  sun_2_4
  sun_3_1
  sun_3_2
* sun_3_3
  master
```

### 3.4. ВЕРСТКА ТЕКСТОВЫХ БЛОКОВ

```
git checkout -b sun_3_4
```

Выделим теперь текстовые блоки первого экрана



Первый блок: Приветствие: Morning, Sunny (Доброе утро Солнышко)

Второй блок: Заголовок: What sticker do you want to buy today (Какую наклейку ты хочешь купить сегодня)

Третий блок: Поиск (Поле ввода для поиска наклейки)

Четвертый блок: Подзаголовок: Available for you (Наклейки которые можно купить)

Пятый блок: Категории

Шестой блок: Список наклеек

Седьмой блок: Подзаголовок: Best stickers of the week  
(Лучшее предложение недели)

Восьмой блок: Список наклеек.

Все эти блоки расположим в одну колонку (Column). В настоящем параграфе сделаем верстку текстовых блоков.  
Приветствие, Заголовок и два подзаголовка – это текстовые блоки.

В Scaffold добавляем свойство body.

```
Scaffold(  
    appBar: _appBar(context),  
    body: Padding(  
        padding: const EdgeInsets.all(20),  
        child: SingleChildScrollView(  
            child: Column(  
                mainAxisAlignment: MainAxisAlignment.start,  
                children: [  
                    ...  
                ],  
            ),  
        ),  
    ),  
);
```

Первый блок: Приветствие.

```
Text(  
    "Morning, Sunny",  
    style: Theme.of(context).textTheme.headlineSmall,  
,
```

Второй блок: Заголовок.

```
Text(  
    "What sticker do you want\n to buy today",  
    style: Theme.of(context).textTheme.displayLarge,  
,
```

Четвертый блок: Подзаголовок.

```
Text(  
    "Available for you",  
    style: Theme.of(context).textTheme.displaySmall,  
) ,
```

Седьмой блок: Подзаголовок.

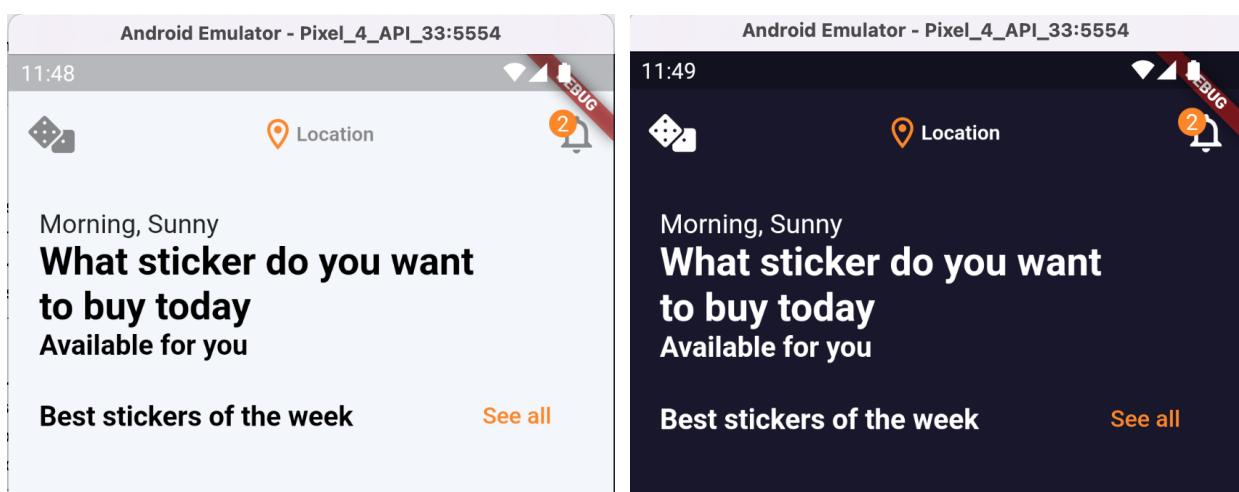
```
Padding(  
    padding: const EdgeInsets.only(top: 25, bottom: 5),  
    child: Row(  
        mainAxisAlignment: MainAxisAlignment.spaceBetween,  
        children: [  
            Text(  
                "Best stickers of the week",  
                style: Theme.of(context).textTheme.displaySmall,  
            ),  
            Padding(  
                padding: const EdgeInsets.only(right: 20),  
                child: Text(  
                    "See all",  
                    style:  
                        Theme.of(context).textTheme.headlineMedium?.copyWith(  
                            color: AppColor.accent),  
                ),  
            ),  
        ],  
    ),  
) ,
```

Добавим в колонку заготовленные 4 блока.

Листинг ui/screens/sticker\_list\_screen.dart

```
Scaffold(  
    appBar: _appBar(context),  
    body: Padding(  
        padding: const EdgeInsets.all(20),  
        child: SingleChildScrollView(  
            child: Column(  
                crossAxisAlignment: CrossAxisAlignment.start,  
                children: [  
                    Text(  
                        "Morning, Sunny",  
                        style: Theme.of(context).textTheme.headlineSmall,  
                    ),  
                    Text(  
                        "What sticker do you want\n\tto buy today",  
                        style: Theme.of(context).textTheme.displayLarge,  
                    ),  
                    Text(  
                        "Available for you",  
                    ),  
                ],  
            ),  
        ),  
    ),  
)
```

```
        style: Theme.of(context).textTheme.displaySmall,
    ),
    Padding(
        padding: const EdgeInsets.only(top: 25,
            bottom: 5),
        child: Row(
            mainAxisAlignment:
                MainAxisAlignment.spaceBetween,
            children: [
                Text(
                    "Best stickers of the week",
                    style:
                        Theme.of(context).textTheme.displaySmall,
                ),
                Padding(
                    padding: const EdgeInsets.only(right: 20),
                    child: Text(
                        "See all",
                        style:
                            Theme.of(context).textTheme.headlineMedium?.copyWith(
                                color: AppColor.accent),
                    ),
                ),
            ],
        ),
    ),
),
)
```



```
% git add .
% git commit -m'Текстовые блоки'
% git branch
```

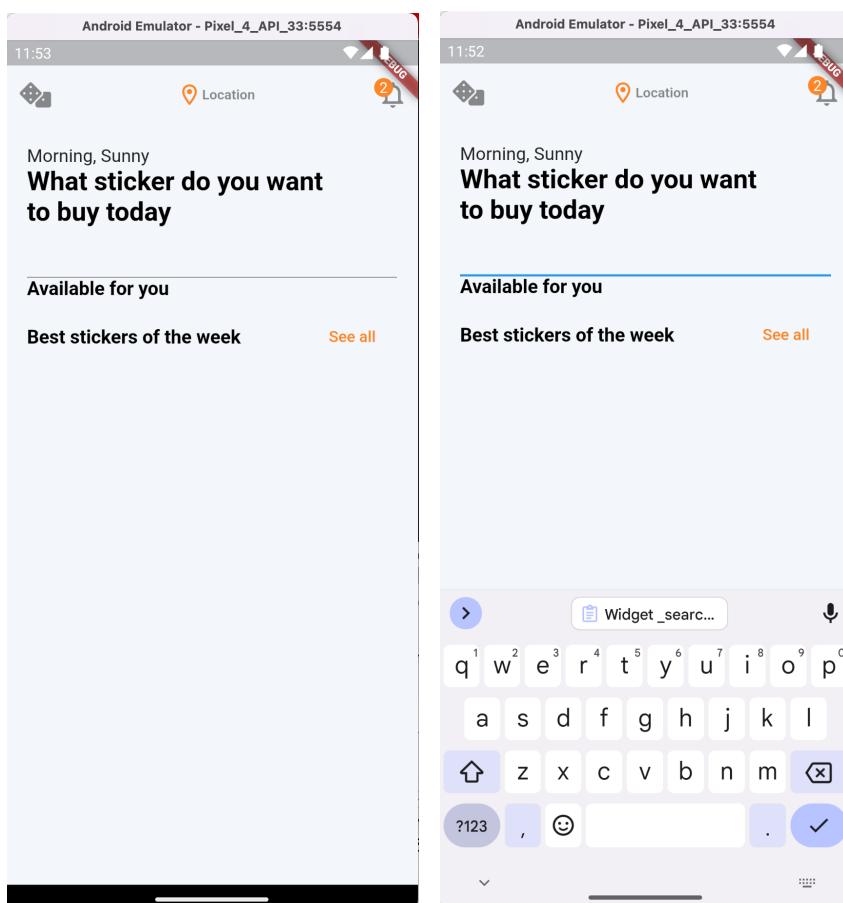
```
sun_2_2
sun_2_3
sun_2_4
sun_3_1
sun_3_2
sun_3_3
* sun_3_4
master
```

### 3.5. ВЕРСТКА ПОИСКА

```
git checkout -b sun_3_5
```

Вынесем поле ввода в отдельный метод и добавим между вторым и четвертым блоком.

```
Widget _searchBar() {
    return const TextField();
}
```



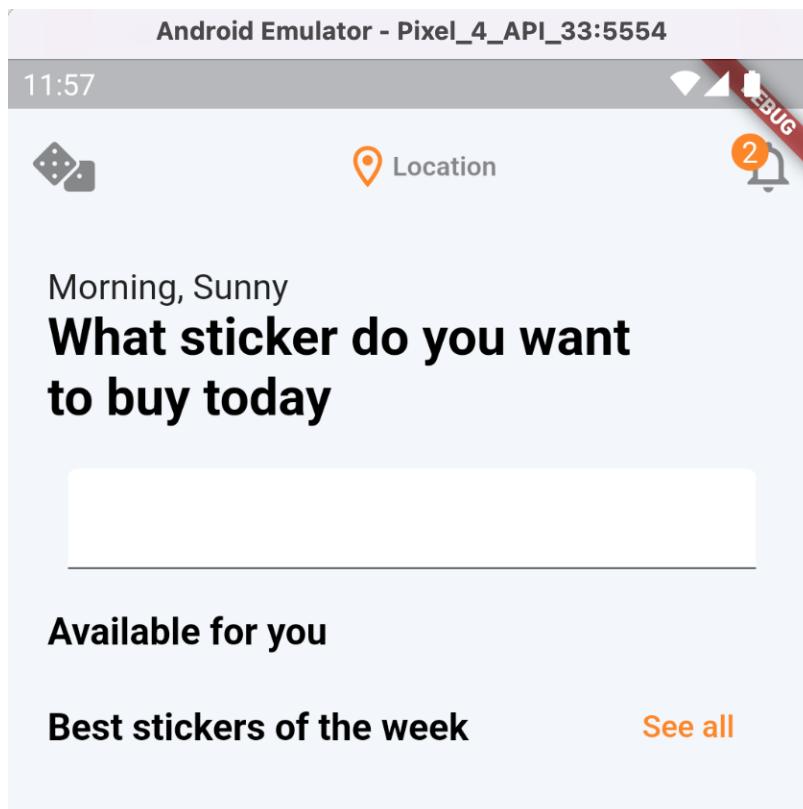
Добавим padding по горизонтали и вертикали для поля ввода и займемся стилизацией.

```
Widget _searchBar() {  
    return const Padding(  
        padding: EdgeInsets.symmetric(vertical: 20,  
horizontal: 10),  
        child: TextField(),  
    );  
}
```

Рассмотрим возможности стилизации:

1) свойства filled и fillColor.

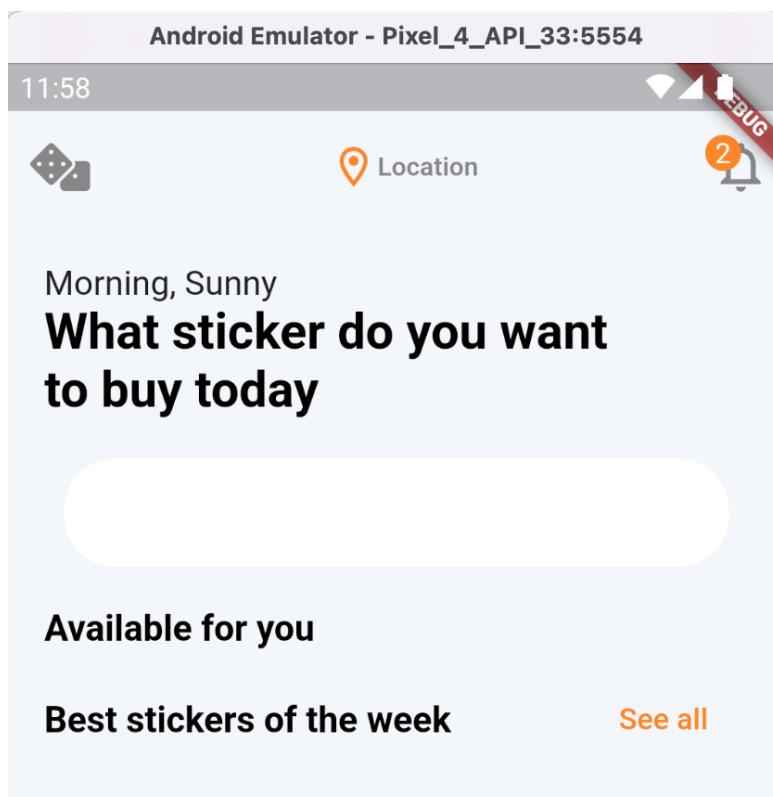
```
TextField(  
    decoration: InputDecoration(filled: true, fillColor:  
Colors.white),  
)
```



2) свойства enabledBorder и focusedBorder.

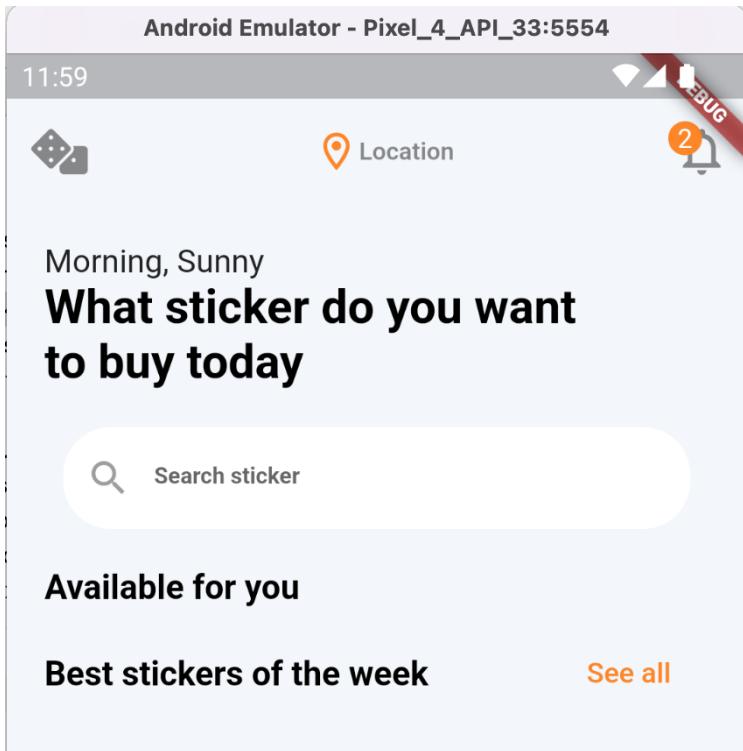
```
TextField(  
    decoration: InputDecoration(  
        filled: true,
```

```
        fillColor: Colors.white,  
        enabledBorder: OutlineInputBorder(  
            borderRadius: BorderRadius.all(Radius.circular(25)),  
            borderSide: BorderSide(color: Colors.transparent),  
        ),  
        focusedBorder: OutlineInputBorder(  
            borderRadius: BorderRadius.all(Radius.circular(25)),  
            borderSide: BorderSide(color: Colors.transparent),  
        ),  
)
```



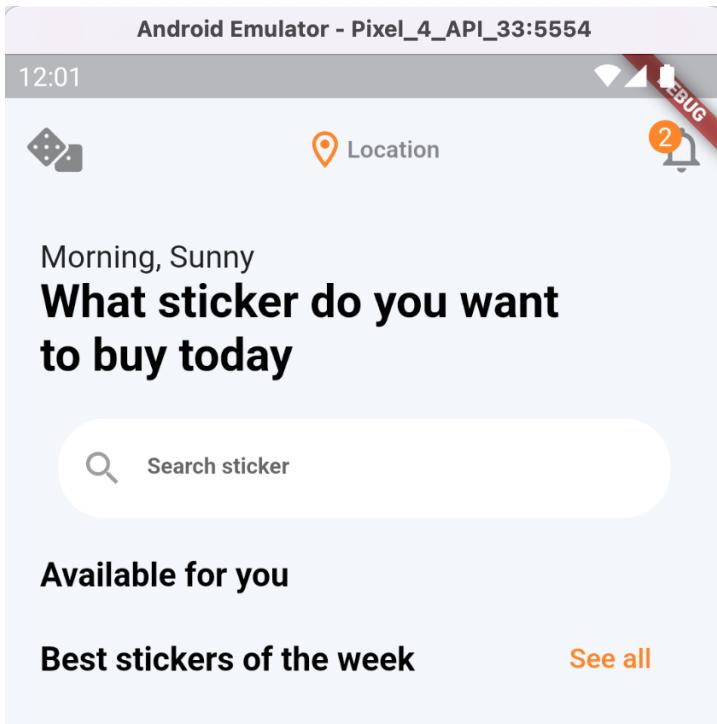
3) свойство prefixIcon и hintText.

```
TextField(  
    decoration: InputDecoration(  
        ...  
        hintText: 'Search sticker',  
        prefixIcon: Icon(Icons.search, color: Colors.grey),  
    ),  
)
```



4) свойство contentPadding.

```
TextField(  
  decoration: InputDecoration(  
    ...  
    contentPadding: EdgeInsets.all(20),  
    hintText: 'Search sticker',  
    prefixIcon: Icon(Icons.search, color: Colors.grey),  
  ),  
)
```



hintText и prefixIcon оставим на экране, все остальные свойства уберем в тему. Для получения стиля темной темы – изменяем свойство fillColor и добавляем в тему hintColor. Листинг lib/ui\_kit/app\_theme.dart.

```
import 'package:flutter/material.dart';

import '_ui_kit.dart';

class AppTheme {
  const AppTheme._();

  static ThemeData lightTheme = ThemeData(
    ...
    hintColor: Colors.black45,
    inputDecorationTheme: const InputDecorationTheme(
      border: OutlineInputBorder(borderSide: BorderSide.none),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.all(Radius.circular(25)),
        borderSide: BorderSide(color: Colors.transparent),
      ),
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.all(Radius.circular(25)),
        borderSide: BorderSide(color: Colors.transparent),
      ),
      filled: true,
```

```
        contentPadding: EdgeInsets.all(20),
        fillColor: Colors.white,
    ),
);

static ThemeData darkTheme = ThemeData(
    ...
    hintColor: Colors.white60,
    inputDecorationTheme: const InputDecorationTheme(
        enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.all(Radius.circular(25)),
            borderSide: BorderSide(color: Colors.transparent),
        ),
        focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.all(Radius.circular(25)),
            borderSide: BorderSide(color: Colors.transparent),
        ),
        filled: true,
        contentPadding: EdgeInsets.all(20),
        fillColor: AppColor.dark,
    ),
);
}
```

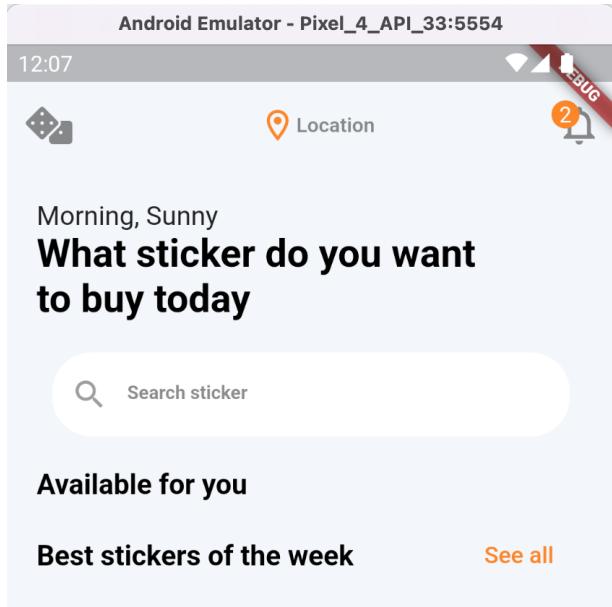
и функция `_searchBar` примет вид:

```
Widget _searchBar() {
    return const Padding(
        padding: EdgeInsets.symmetric(vertical: 20,
horizontal: 10),
        child: TextField(
            decoration: InputDecoration(
                hintText: 'Search sticker',
                prefixIcon: Icon(Icons.search, color:
Colors.grey),
            ),
        ),
    );
}
```

Проверяем для светлой темы.

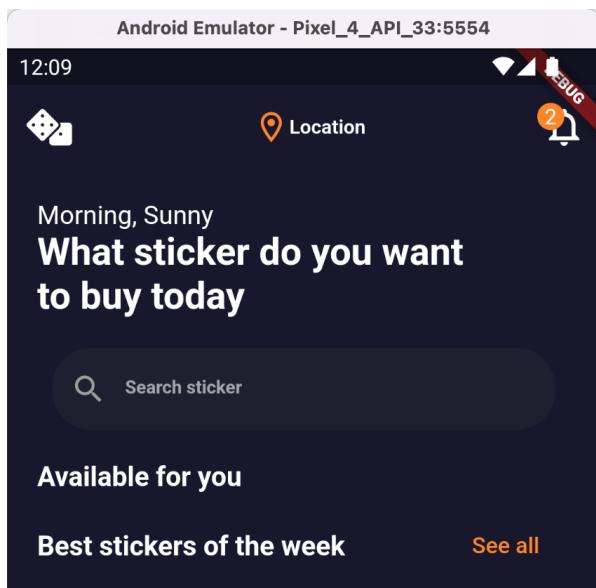
`MaterialApp(`

```
        title: 'Sunny Stickers',
        theme: AppTheme.lightTheme,
        home: const StickerList(),
)
```



Проверяем темную тему.

```
MaterialApp(
    title: 'Sunny Stickers',
    theme: AppTheme.darkTheme,
    home: const StickerList(),
)
```



```
% git add .
% git commit -m'Поиск'
% git branch
...
* sun_3_5
  master
```

### 3.6. ВЕРСТКА КАТЕГОРИЙ

```
git checkout -b sun_3_6
```

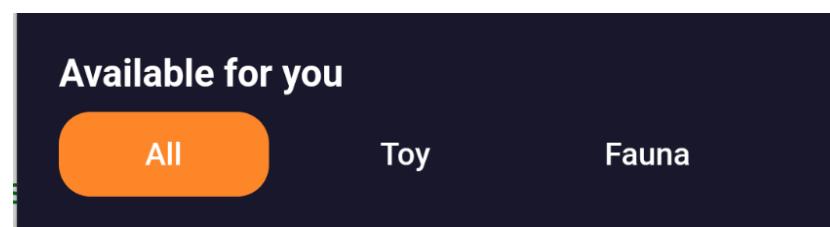
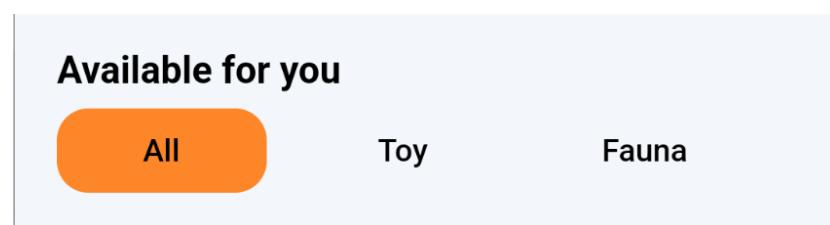
Блок категорий, как и блок поиска вынесем в приватную функцию. Выделение блока в отдельную функцию улучшает читабельность верстки.

```
@override
Widget build(BuildContext context) => Scaffold(
    appBar: _appBar(context),
    body: Padding(
        padding: const EdgeInsets.all(20),
        child: SingleChildScrollView(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.start,
                children: [
                    Text(
                        ...
                    ),
                    Text(
                        ...
                    ),
                    _searchBar(),
                    Text(
                        ...
                    ),
                    _categories(),
                    Padding(
                        ...
                    ),
                    ],
                ),
            ),
        ),
    ),
```

```
        ],
        ),
        ),
        );
    );
```

```
Widget _categories(){
    return Container();
}
```

Метод `_categories()` возвращает заглушку. И можно начинать верстку.

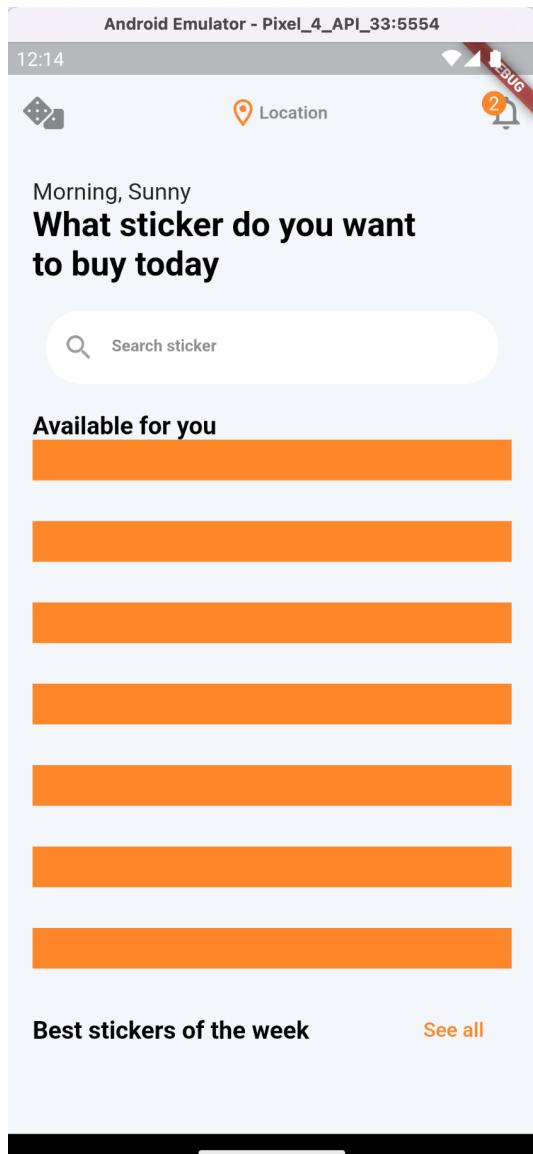


Отличия в верстке светлой и темной темы это цвет текста. Выберем в качестве базового виджета `ListView.separated` – список с разделителем.

Верстку начнем с кода:

```
Widget _categories() {
    return SizedBox(
        height: 400,
        child: ListView.separated(
            itemBuilder: (_, index) {
                return Container(
                    width: 100,
                    height: 30,
                    color: AppColor.accent,
                );
            },
            separatorBuilder: (_, __) => Container(
                width: 15,
```

```
        height: 30,  
    ),  
    itemCount: 20),  
);  
}
```



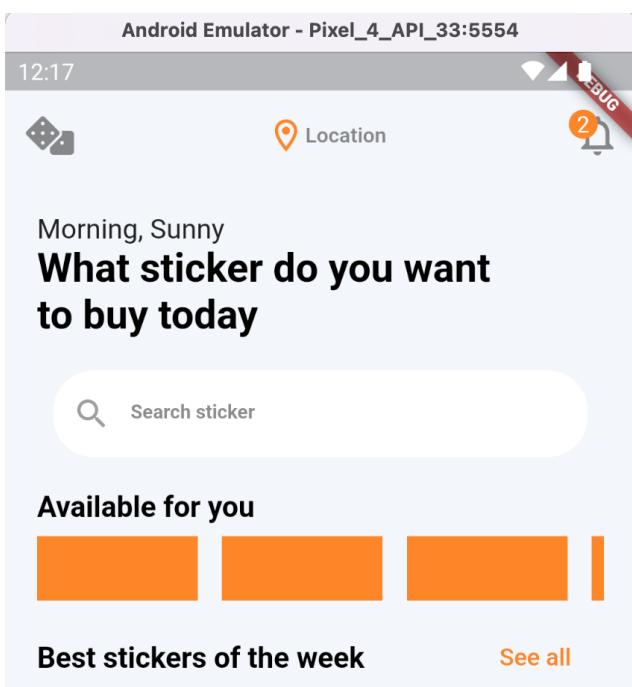
1) Сделаем отступ между подзаголовком и категориями.

```
Widget _categories() {  
    return Padding(  
        padding: const EdgeInsets.only(top: 8.0),  
        child: SizedBox(  
            height: 400,  
            child: ListView.separated(  
                ...  
            ),  
        ),
```

```
    );  
}
```

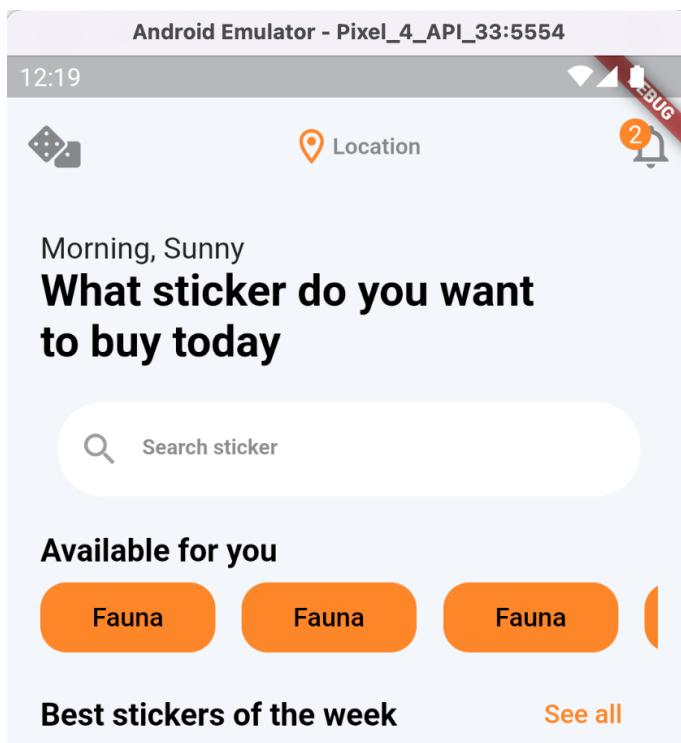
2) Изменим дефолтный вертикальный скрол на горизонтальный, поправим высоту SizedBox и уберем высоту в контейнерах.

```
Widget _categories() {  
    return Padding(  
        padding: const EdgeInsets.only(top: 8.0),  
        child: SizedBox(  
            height: 40,  
            child: ListView.separated(  
                scrollDirection: Axis.horizontal,  
                itemBuilder: (_, index) {  
                    return Container(  
                        width: 100,  
                        color: AppColor.accent,  
                    );  
                },  
                separatorBuilder: (_, __) => Container(  
                    width: 15,  
                ),  
                itemCount: 20),  
        ),  
    );  
}
```



### 3) Стилизуем контейнер категории

```
Container(  
    width: 100,  
    alignment: Alignment.center,  
    decoration: const BoxDecoration(  
        color: AppColor.accent,  
        borderRadius: BorderRadius.all(  
            Radius.circular(15),  
        ),  
    ),  
    child: Text(  
        'Kebab',  
        style: Theme.of(context).textTheme.headlineMedium,  
    ),  
)
```



### 4) Добавим клик на категорию

```
itemBuilder: (_, index) {  
    return GestureDetector(  
        onTap: () {  
            print('Кликнули на категорию');  
        },  
        child: Container(  
            width: 100,
```

```
    ), ...  
};  
}
```

Верстка списка категорий закончена.

```
% git add .  
% git commit -m'Верстка категорий'  
% git branch  
...  
* sun_3_6  
master
```

### 3.7. ЛОГИКА КЛИКА НА КАТЕГОРИЮ

```
git checkout -b sun_3_7
```

Подключим данные к готовой верстке.  
Данные по категориям заготовлены в файле  
`data/app_data.dart`.

```
static List<StickerCategory> categories = [  
  StickerCategory(StickerType.all, true),  
  StickerCategory(StickerType.toy, false),  
  StickerCategory(StickerType.fauna, false),  
  StickerCategory(StickerType.plant, false),  
  StickerCategory(StickerType.berry, false),  
  StickerCategory(StickerType.fruit, false),  
  StickerCategory(StickerType.other, false),  
];
```

Напомним структуру `StickerCategory` и `StickerType`:

```
class StickerCategory {  
  final StickerType type;  
  bool isSelected;  
  
  StickerCategory(this.type, this.isSelected);  
}
```

```
enum StickerType { all, toy, fauna, plant, berry, fruit, other }
```

Добавляем логику на экране.

```
class StickerListState extends State<StickerList> {
  var categories = AppData.categories;

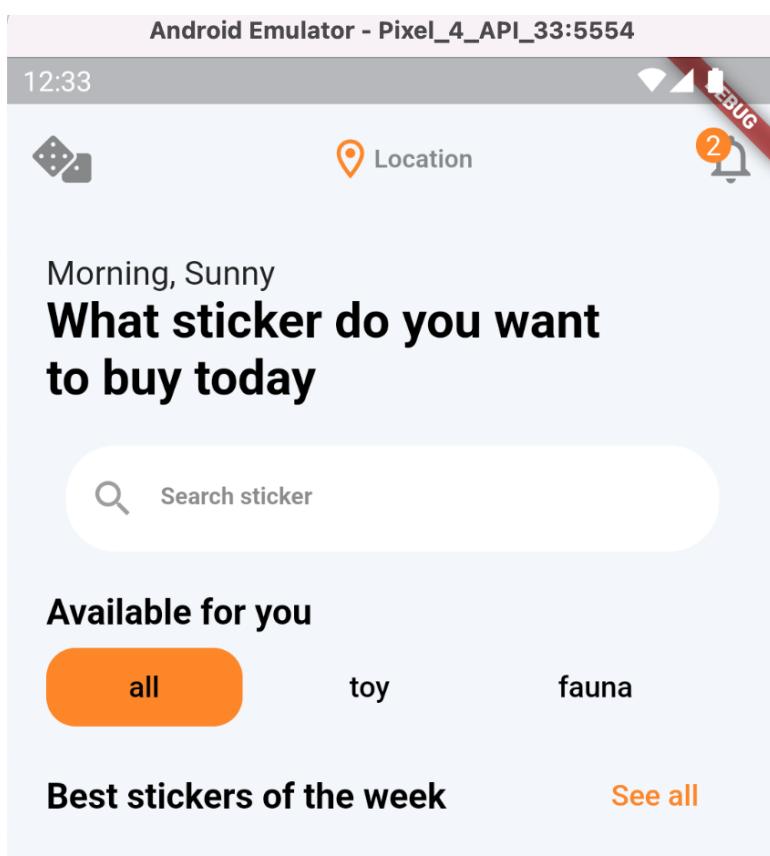
  @override
  Widget build(BuildContext context) => Scaffold(
);

Widget _categories() {
  return Padding(
    padding: const EdgeInsets.only(top: 8.0),
    child: SizedBox(
      height: 40,
      child: ListView.separated(
        scrollDirection: Axis.horizontal,
        itemBuilder: (_, index) {
          final category = categories[index];
          return GestureDetector(
            onTap: () {
              print('Кликнули на категорию');
            },
            child: Container(
              width: 100,
              alignment: Alignment.center,
              decoration: BoxDecoration(
                color: category.isSelected ?
                  LightThemeColor.accent :
                  Colors.transparent,
                borderRadius: const BorderRadius.all(
                  Radius.circular(15),
                ),
              ),
              child: Text(
                category.type.name,
                style: Theme.of(context).textTheme.headlineMedium,
              ),
            ),
          );
        },
      );
}
```

```

        },
        separatorBuilder: (_, _) => Container(
            width: 15,
        ),
        itemCount: categories.length),
    ),
);
}
}

```



Пять мест в коде экрана обеспечивает отображение категорий.

- 1) Объявление переменной categories и инициализация данными из класса AppData.
- 2) Получение категории (StickerCategory) по индексу в ListView (перед отображением категории).
- 3) Цвет бекграунда категории.
- 4) Название категории.
- 5) Количество элементов в ListView.

Реализуем переключение выделенной категории по клику.

Рассмотрим метод, который меняет выбранную категорию. В реализованной логике, если isSelected равно true, то категория подсвечивается оранжевым. При нажатии на категорию будем передавать в метод индекс выбранной категории (в массиве категорий categories).

```
void onCategoryTap(int selectedIndex) {  
    //Меняем выбранную категорию  
    categories.asMap().forEach((index, category) {  
        category.isSelected = index == selectedIndex;  
    });  
    //Проверяем, что категория поменялась  
    for (final category in categories) {  
        print('${category.type.name}  
${category.isSelected}');  
    }  
}
```

Подключаем метод к клику:

```
onTap: () {  
    print('Кликнули на категорию');  
    onCategoryTap(index);  
},
```

Проверим работу метода.

Кликнули на категорию  
all false  
toy true  
fauna false  
plant false  
berry false  
fruit false  
other false

Для того чтобы перерисовался экран для новой выбранной категории необходимо вызвать метод setState((){}). Удалим проверочный код:

```
void onCategoryTap(int selectedIndex) {  
    //Меняем выбранную категорию  
    categories.asMap().forEach((index, category) {  
        category.isSelected = index == selectedIndex;
```

```

    });
    setState(() {});
}

onTap: () {
    onCategoryTap(index);
},

```

```

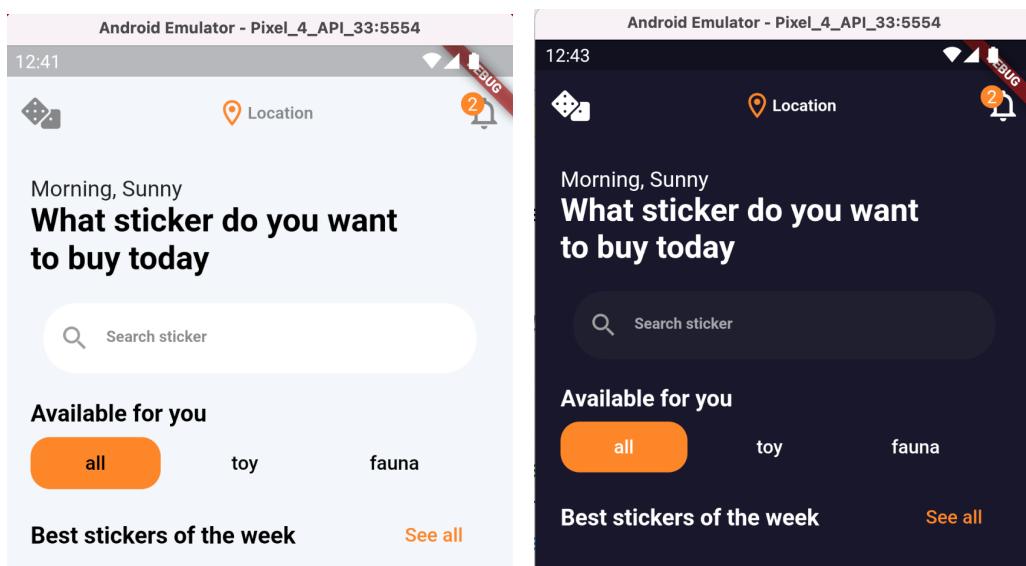
% git add .
% git commit -m'Логика клика'
% git branch
...
* sun_3_7
  master

```

### 3.8 EXTENSION

`git checkout -b sun_3_8`

Рассмотрим текущее состояние верстки экрана списка еды.



Замечание по верстке, которое сразу бросается в глаза, это то, что название категорий начинается с маленькой буквы.

Исправим это.

```

Text(
  category.type.name,

```

```
    style: Theme.of(context).textTheme.headlineMedium,  
)
```

Напомним структуру StickerCategory и StickerType:

```
class StickerCategory {  
  final StickerType type;  
  bool isSelected;  
  
  StickerCategory(this.type, this.isSelected);  
}  
  
enum StickerType { all, toy, fauna, plant, berry, fruit,  
other }
```

Значит в виджете Text мы получаем название списка enum. Поэтому то что надо сделать, это сделать первую букву большой.

Сделаем это используя технику extension.

В папке ui создадим папку extensions. В папке extensions создадим файл app\_extensions.dart.

Листинг файла lib/ui/extensions/app\_extensions.dart

```
extension StringExtension on String {  
  String get firstCapital => this[0].toUpperCase() +  
substring(1, length);  
}
```

Листинг файла lib/ui/extensions/\_extensions.dart

```
export 'app_extensions.dart';
```

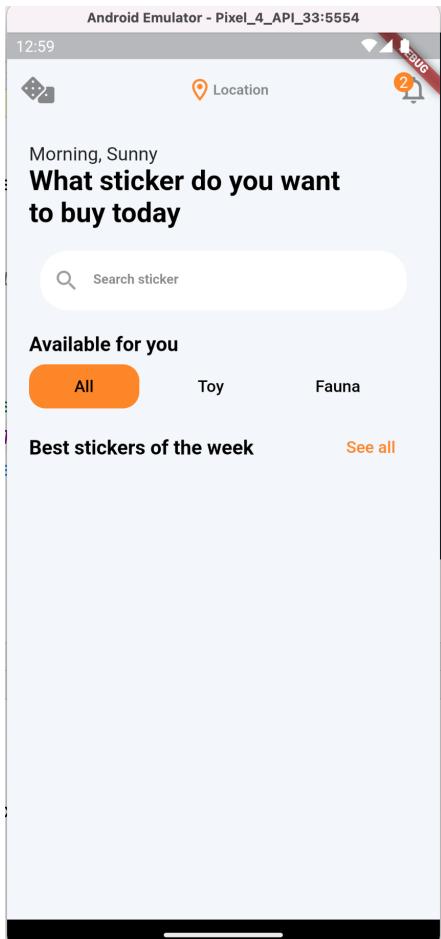
Листинг файла lib/ui/\_ui.dart

```
export 'extensions/_extensions.dart';  
export 'screens/_screens.dart';
```

Для строк задаем переменную (getter) firstCapital – getter берет первую букву и делает конкатенацию со строкой без первой буквы.

Используем расширение в коде.

```
Text(  
    category.type.name.firstCapital,  
    style: Theme.of(context).textTheme.headlineMedium,  
)
```



```
% git add .  
% git commit -m'Extensions'  
% git branch  
...  
* sun_3_8  
  master
```

### 3.9 ВИДЖЕТ STICKERLISTVIEW

```
git checkout -b sun_3_9
```

Для завершения верстки первого экрана осталось добавить два блока – списка наклеек. Первый список – это список

наклеек доступных для покупки. Второй список – это список наклеек – лучшее предложение недели.

### **Правило хорошего тона.**

Одно из правил хорошего тона – DRY (Don't repeat yourself), что означает – не повторяйтесь. Два списка, которые предстоит сверстать, по дизайну не отличаются. Для того, чтобы не повторять код можно выделить общий код в функцию. Для AppBar верстку выделяли в отдельную функцию, для блока поиска – верстку тоже выделяли в отдельную функцию и для списка категорий, тоже создали отдельную функцию. Это было сделано для улучшения читабельности кода. Для списка еды тоже можно выделить верстку списка еды в отдельную функцию. Кроме выделения в отдельную функцию, сделать декомпозицию и реализовать DRY можно через отдельный виджет. Виджет может быть использован на любом экране приложения. Для общих виджетов уже подготовлена папка – ui/widgets. В этой папке создадим файл для виджета sticker\_list\_view.dart и файл для экспортов \_widgets.dart.

Листинг файла ui/widgets/sticker\_list\_view.dart

```
import 'package:flutter/material.dart';

import '../../../../../data/_data.dart';

class StickerListView extends StatelessWidget {
  const StickerListView({super.key, required
this.stickers, this.isReversed = false});

  final List<Sticker> stickers;
  final bool isReversed;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 200,
      child: ListView.separated(
        scrollDirection: Axis.horizontal,
        padding: const EdgeInsets.only(top: 20),
        itemBuilder: (_, index) {
          return Container(
            width: 160,
```

```
        decoration: const BoxDecoration(
            color: Colors.orange,
            borderRadius:
BorderRadius.all(Radius.circular(20)),
        ),
    },
},
separatorBuilder: (_, _) {
    return Container(
        width: 50,
    );
},
itemCount: stickers.length),
);
}
}
```

Листинг файла ui/widgets/\_widgets.dart

```
export 'sticker_list_view.dart';
```

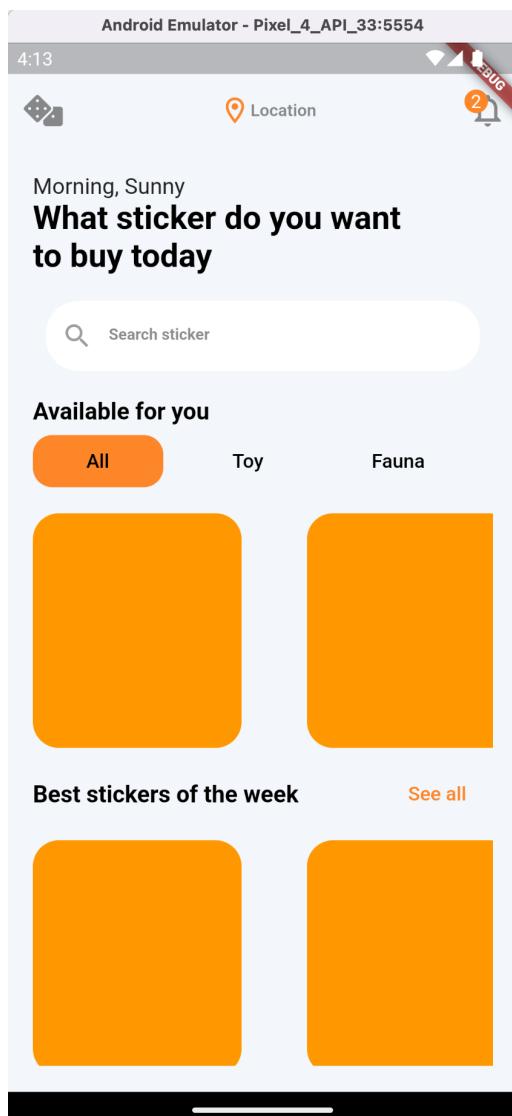
Листинг файла ui/\_ui.dart

```
export 'extensions/_extensions.dart';
export 'screens/_screens.dart';
export 'widgets/_widgets.dart';
```

Добавим виджет списка еды в верстку первого экрана.

```
Scaffold(
    appBar: _appBar(context),
    body: Padding(
        padding: const EdgeInsets.all(20),
        child: SingleChildScrollView(
            child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                    Text(...),
                    Text(...),
                    _searchBar(),
                    Text(...),
                    _categories(),
                ],
            ),
        ),
    ),
);
```

```
        StickerListView(stickers:  
AppData.stickers),  
        Padding(...),  
        StickerListView(stickers:  
AppData.stickers),  
        ],  
        ),  
        ),  
        ),  
    );
```



```
% git add .  
% git commit -m 'Виджет еды'  
% git branch  
...  
* sun_3_9  
master
```

### 3.10 ПРОВЕРКА ТЕКУЩЕЙ ТЕМЫ

```
git checkout -b sun_3_10
```

Один из самых простых способов определения текущей темы это параметр темы `brightness`. Добавим этот параметр в каждую из тем.

```
class AppTheme {  
  const AppTheme._();  
  
  static ThemeData lightTheme = ThemeData(  
    brightness: Brightness.light,  
    ...  
  );  
  
  static ThemeData darkTheme = ThemeData(  
    brightness: Brightness.dark,  
    ...  
  );
```

Используя этот параметр установим цвет карточки наклейки в зависимости от темы.

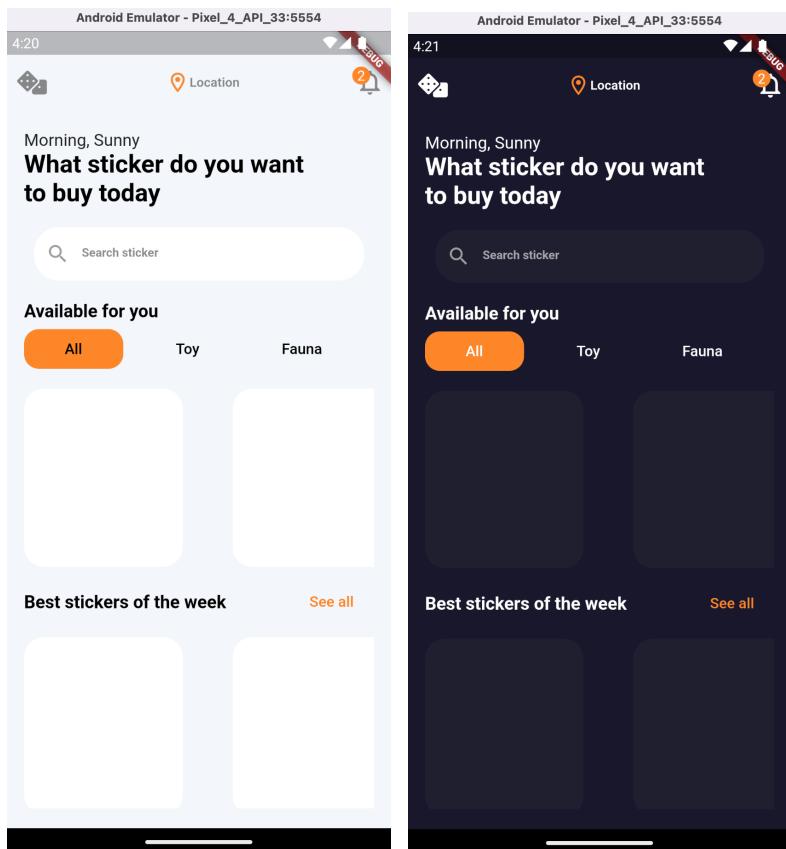
```
@override  
Widget build(BuildContext context) {  
  final isDark = Theme.of(context).brightness ==  
Brightness.dark;  
  return SizedBox(  
    height: 200,  
  ...
```

Теперь цвет карточки наклейки можно задать с учетом темы.

```
Container(  
  width: 160,  
  decoration: BoxDecoration(  
    color: isDark ?  
      AppColor.dark :  
      Colors.white,  
    borderRadius: const BorderRadius.all(Radius.circular(20)),  
  ),
```

);

Проверяем.



```
% git add .
% git commit -m'Проверка темы'
% git branch
...
* sun_3_10
  master
```

### 3.11 ВЕРСТКА КАРТОЧКИ НАКЛЕЙКИ

```
git checkout -b sun_3_11
```

Из массива наклеек `stickers` по индексу берем каждый элемент массива и отображаем в колонке.

```
itemBuilder: (_, index) {
  Sticker sticker = isReversed ?
    stickers.reversed.toList()[index] : stickers[index];
  return Container(
    ...
  );
}
```

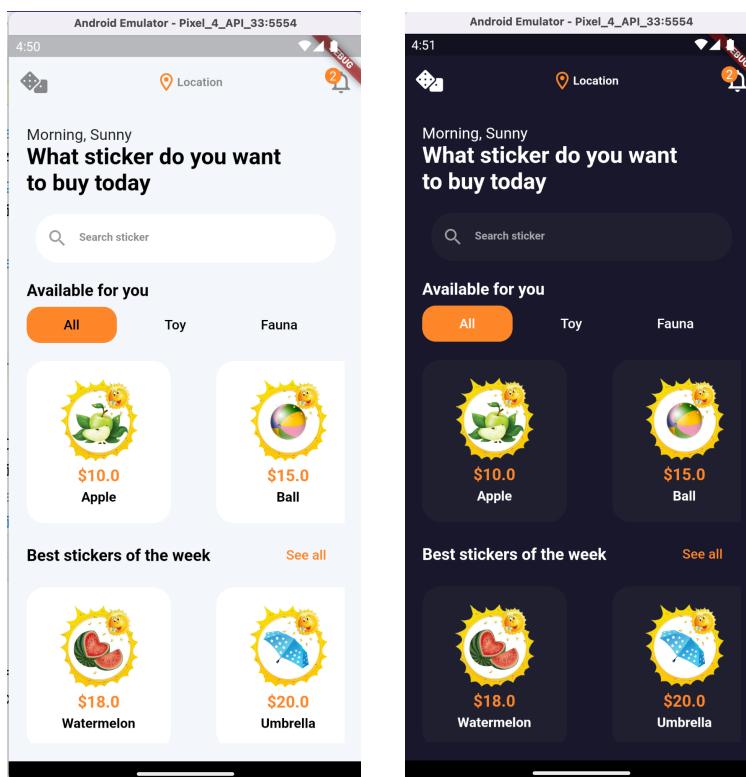
```
        child: Padding(
            padding: const EdgeInsets.all(20),
            child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                    Image.asset(sticker.image, scale: 6),
                    Text(
                        "\$${sticker.price}",
                        style: AppTextStyle.h3Style.copyWith(color: AppColor.accent),
                    ),
                    Text(
                        sticker.name,
                        style: Theme.of(context).textTheme.headlineMedium?.copyWith(fontWeight: FontWeight.bold),
                    ),
                    ],
                )));
},
```

```
Scaffold(
    appBar: _appBar(context),
    body: Padding(
        padding: const EdgeInsets.all(20),
        child: SingleChildScrollView(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.start,
                children: [
                    Text(...),
                    Text(...),
                    _searchBar(),
                    Text(...),
                    _categories(),
                    StickerView(stickers: AppData.stickers),
                    Padding(...),
                    StickerView(stickers: AppData.stickers,
                        isReversed: true),
                ],
            ),
        ),
    ),
)
```

```
        ),  
        ),  
        );
```

Обернем карточку наклейки кликом GestureDetector.

```
itemBuilder: (_, index) {  
  Sticker sticker = isReversed ?  
    stickers.reversed.toList()[index] : stickers[index];  
  return GestureDetector(  
    onTap: (){  
      print('Клик на карточку');  
    },  
    child: Container(  
      ...  
    );  
,
```



```
% git add .  
% git commit -m'Верстка карточки наклейки'  
% git branch  
...  
* sun_3_11  
master
```

## 4. ВЕРСТКА КОРЗИНЫ

### 4.1 ЭКРАН КОРЗИНЫ

```
git checkout -b sun_4_1
```

В папке экранов ui/screens создаем новый файл cart\_screen.dart.

Листинг файла ui/screens/cart\_screen.dart.

```
import 'package:flutter/material.dart';

class CartScreen extends StatefulWidget {
  const CartScreen({super.key});

  @override
  State<CartScreen> createState() => CartScreenState();
}

class CartScreenState extends State<CartScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold();
  }
}
```

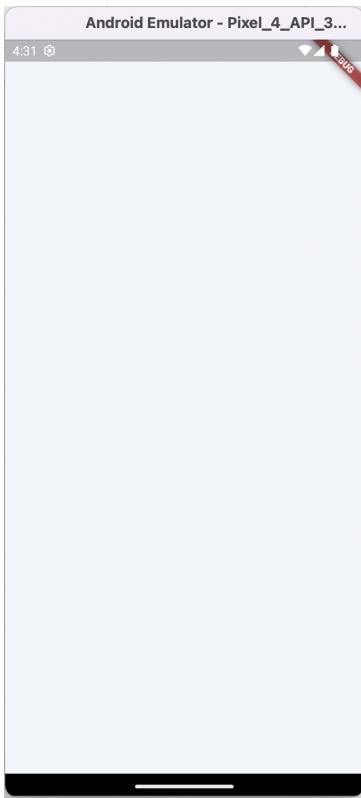
Листинг файла ui/screens/\_screens.dart.

```
export 'cart_screen.dart';
export 'sticker_list_screen.dart';
```

Подключим экран корзины в MaterialApp.

```
MaterialApp(
  title: 'Sunny Stickers',
  theme: AppTheme.lightTheme,
  home: const CartScreen(),
);
```

Запустим проект.



Можно начинать верстку.

```
% git add .
% git commit -m'Экран корзины'
% git branch
  ...
* sun_4_1
  master
```

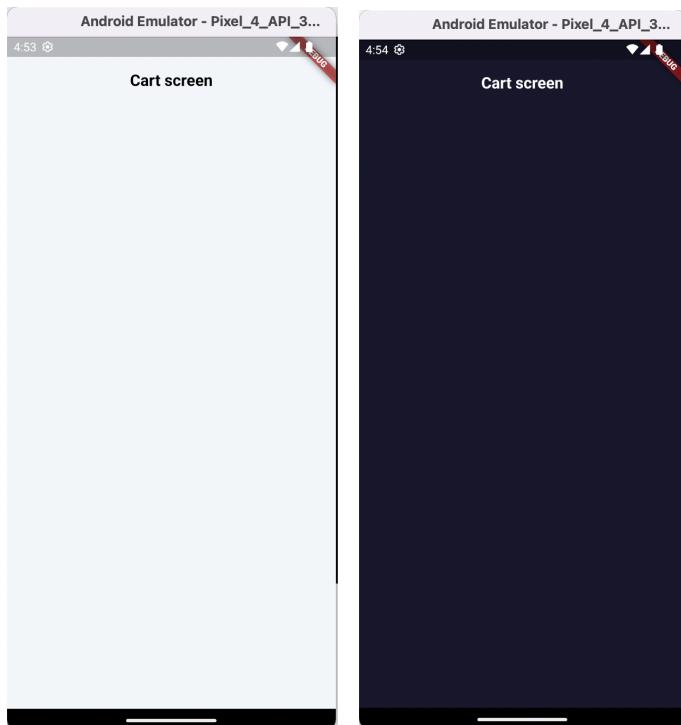
## 4.2 APPBAR ЭКРАНА КОРЗИНЫ

```
git checkout -b sun_4_2
```

Аналогично верстке экрана StickerList вынесем создание AppBar в отдельную функцию.

```
class CartScreenState extends State<CartScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: _appBar(context),
    );
}
```

```
PreferredSizeWidget _appBar(BuildContext context) {  
    return AppBar(  
        title: Text(  
            "Cart screen",  
            style: Theme.of(context).textTheme.displayMedium,  
        ),  
    );  
}  
}
```



```
% git add .  
% git commit -m 'AppBar экрана корзины'  
% git branch  
...  
* sun_4_2  
master
```

### 4.3 ОБЕРТКА ДЛЯ ПУСТОЙ КОРЗИНЫ

```
git checkout -b sun_4_3
```

Для экранов корзины и любимой еды имеет смысл создать виджет пустой корзины и пустого списка любимых наклеек.

То есть один виджет для двух экранов. Для общих виджетов в приложении создана папка ui/widgets. В папке ui/widgets создадим файл empty\_wrapper.dart.

Листинг файла empty\_wrapper.dart.

```
import 'package:flutter/material.dart';

enum EmptyWrapperType { cart, favorite }

class EmptyWrapper extends StatelessWidget {
    const EmptyWrapper({super.key,
        required this.type,
        required this.title,
        required this.isEmpty,
        required this.child
    });

    final EmptyWrapperType type;
    final String title;
    final bool isEmpty;
    final Widget child;

    @override
    Widget build(BuildContext context) {
        return isEmpty?
            const Center(child: Text('Здесь будет Empty
View')),):
            child;
    }
}
```

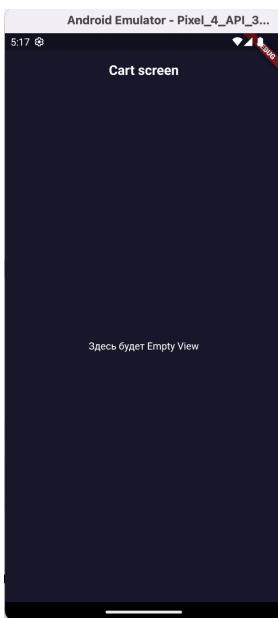
Листинг файла ui/widgets/\_widgets.dart.

```
export 'empty_wrapper.dart';
export 'sticker_list_view.dart';
```

Подключим созданный виджет к экрану корзины.

```
Scaffold(
    appBar: _appBar(context),
```

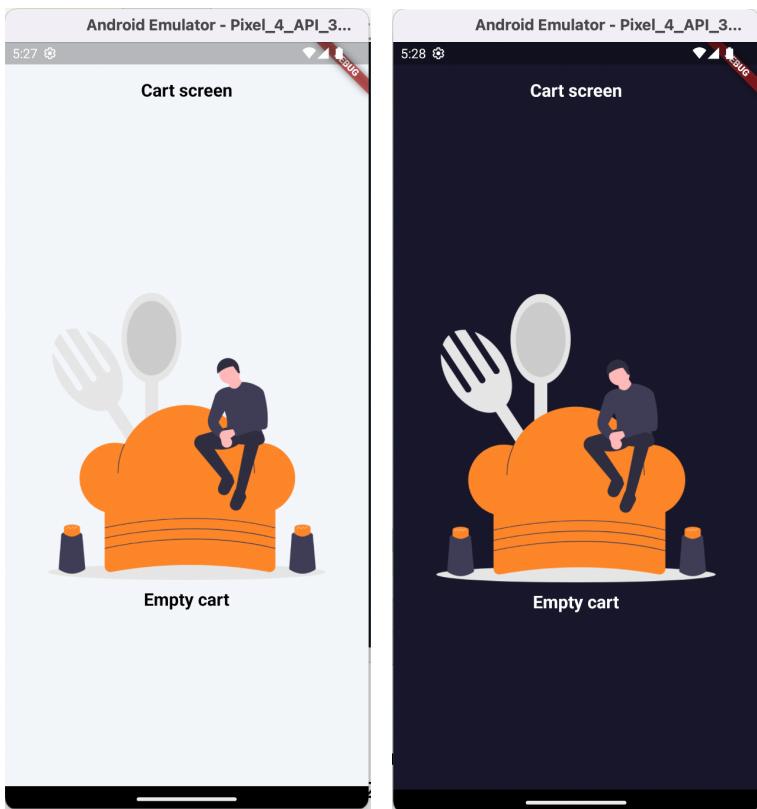
```
body: EmptyWrapper(  
    title: "Empty cart",  
    isEmpty: true,  
    child: Container(),  
) ,  
) ;
```



В виджете EmptyWrapper сделаем верстку пустого экрана.

```
Center(  
    child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: [  
            type == EmptyWrapperType.cart  
                ? Image.asset(AppAsset.emptyCart, width: 300)  
                : Image.asset(AppAsset.emptyFavorite, width:  
            300),  
            const SizedBox(height: 10),  
            Text(title, style:  
Theme.of(context).textTheme.displayMedium)  
        ],  
    ),  
)
```

Проверим.



```
% git add .
% git commit -m 'Обертка для пустой корзины'
% git branch
...
* sun_4_3
  master
```

#### 4.4 ДАННЫЕ ДЛЯ КОРЗИНЫ

```
git checkout -b sun_4_4
```

В файл `data/app_data.dart` добавим данные для корзины.

```
static List<Sticker> cartItems = [stickers[0],
stickers[1], stickers[2]];
```

Теперь корзина не пустая.

```
class CartScreenState extends State<CartScreen> {
  var cartItems = AppData.cartItems;
```

```
@override
```

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: _appBar(context),
    body: EmptyWrapper(
      title: "Empty cart",
      isEmpty: cartItems.isEmpty,
      child: Container(),
    ),
  );
}
```

Создадим две функции. Одна функция будет отвечать за список продуктов в корзине, вторая за подсчет стоимости корзины.

```
Widget _cartListView() {
  return ListView.separated(
    padding: const EdgeInsets.all(30),
    itemCount: 20,
    itemBuilder: (_, index) {
      return Container(
        height: 60,
        width: double.infinity,
        padding: const EdgeInsets.all(5),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
          color: Colors.white,
        ),
        child: Text(
          '${index + 1}',
          style: const TextStyle(fontSize: 24,
fontWeight: FontWeight.bold),
        ),
      );
    },
    separatorBuilder: (_, __) => Container(
      height: 20,
    ),
  );
}

Widget _bottomAppBar() {
  return BottomAppBar(
    child: SizedBox(
```

```
        height: 300,
        child: Container(
            color: Colors.white,
        )));
}
```

Подключим эти две функции к верстке.

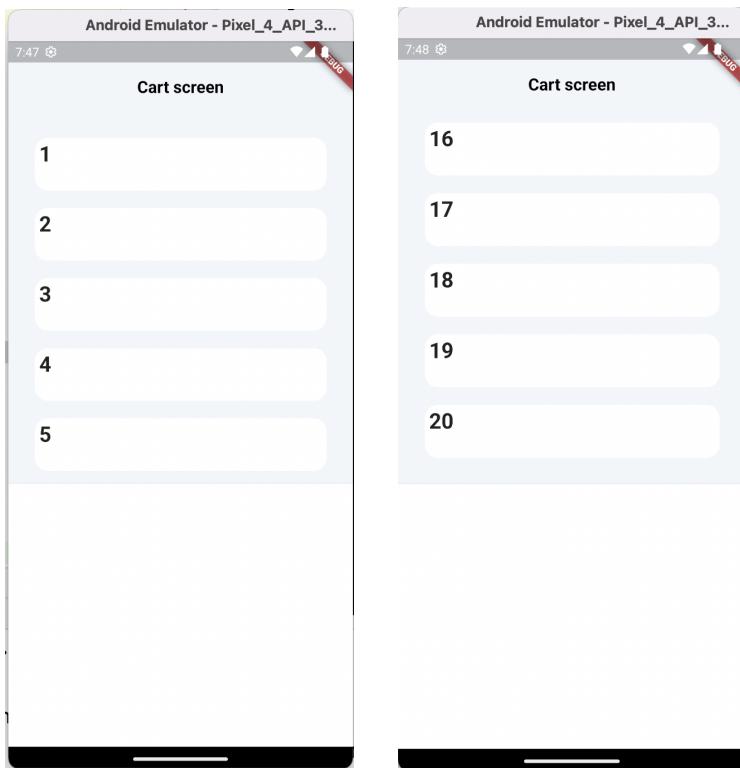
```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: _appBar(context),
        body: EmptyWrapper(
            title: "Empty cart",
            isEmpty: cartItems.isEmpty,
            child: _cartListView(),
        ),
        bottomNavigationBar: _bottomAppBar(),
    );
}

Widget _cartListView() {...}
```

```
Widget _bottomAppBar() {...}
```

Подсчет стоимости корзины для пустой корзины спрячем.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: _appBar(context),
        body: EmptyWrapper(
            title: "Empty cart",
            isEmpty: cartItems.isEmpty,
            child: _cartListView(),
        ),
        bottomNavigationBar: cartItems.isEmpty ? const
SizedBox() : _bottomAppBar(),
    );
}
```



Перейдем к верстке списка и подсчета корзины.

```
% git add .
% git commit -m 'Данные для корзины'
% git branch
...
* sun_4_4
 master
```

## 4.5 ВЕРСТКА СПИСКА КОРЗИНЫ

```
git checkout -b sun_4_5
```

Дизайн наклейки из списка следующий.



Выделим в отдельный виджет:



Для создания виджетов в папке ui созданы папка widgets.  
Создадим новый файл counter\_button.dart  
Листинг файла ui/widgets/counter\_button.dart.

```
import 'package:flutter/material.dart';

import ' ../../ui_kit/_ui_kit.dart';

class CounterButton extends StatelessWidget {
  const CounterButton(
    {Key? key,
    required this.onIncrementTap,
    required this.onDecrementTap,
    required this.label,
    this.padding = 10.0,
    this.size = const Size(36, 36),
    this.orientation = Axis.horizontal})
    : super(key: key);

  final VoidCallback onIncrementTap;
  final VoidCallback onDecrementTap;
  final Widget label;
  final Axis orientation;
  final Size size;
  final double padding;

  @override
  Widget build(BuildContext context) {
    return orientation == Axis.horizontal
```

```

        ? Row(mainAxisAlignment: MainAxisAlignment.end,
children: body())
        : Column(children: body().reversed.toList());
    }

Widget button(Icon icon, Function() onTap) {
    return RawMaterialButton(
        constraints: BoxConstraints.tight(size),
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(5)),
        fillColor: AppColor.accent,
        onPressed: () => onTap(),
        child: icon,
    );
}

List<Widget> body() {
    return [
        button(
            const Icon(Icons.remove, color: Colors.white),
            onDecrementTap,
        ),
        Padding(
            padding: EdgeInsets.symmetric(horizontal:
padding),
            child: label,
        ),
        button(const Icon(Icons.add, color: Colors.white),
onIncrementTap),
    ];
}
}

```

Добавим новый виджет к элементу списка.

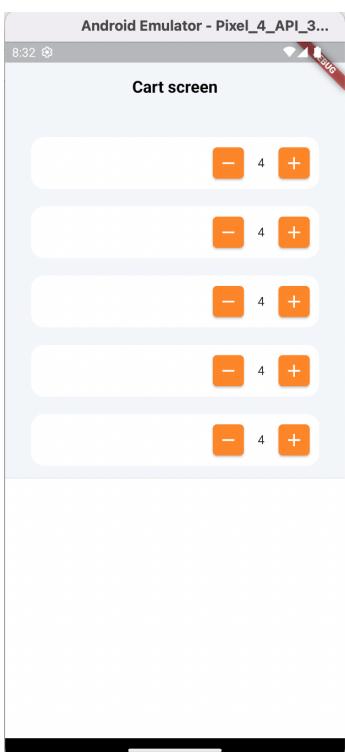
```

Widget _cartListView() {
    return ListView.separated(
        padding: const EdgeInsets.all(30),
        itemCount: 20,
        itemBuilder: (_, index) {
            return Container(
                ...
                child: CounterButton(
                    onDecrementTap: () {

```

```
        print('Нажали на уменьшение количества');
    },
    onIncrementTap: () {
        print('Нажали на увеличение количества');
    },
    label: Text('4'),
),
);
},
),
separatorBuilder: (_, __) => Container(
    height: 20,
),
);
}
}
```

Получим:



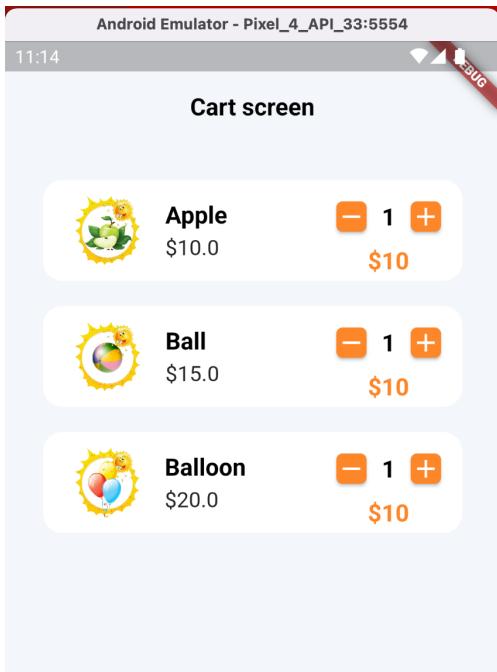
Закончим верстку элемента списка.

```
Widget _cartListView() {
    return ListView.separated(
        padding: const EdgeInsets.all(30),
        itemCount: cartItems.length,
        itemBuilder: (_, index) {
            final sticker = cartItems[index];
```

```
return Container(
    width: double.infinity,
    padding: const EdgeInsets.all(5),
    decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(15),
        color: Colors.white,
    ),
    child: Row(
        mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
        children: [
            const SizedBox(width: 20),
            Image.asset(sticker.image, scale: 10),
            const SizedBox(width: 20),
            Column(
                crossAxisAlignment:
CrossAxisAlignment.start,
                children: [
                    Text(
                        sticker.name,
                        style:
Theme.of(context).textTheme.displayMedium,
                    ),
                    const SizedBox(height: 5),
                    Text(
                        "\${sticker.price}",
                        style:
Theme.of(context).textTheme.headlineSmall,
                    ),
                ],
            ),
            const Spacer(),
            Column(
                children: [
                    CounterButton(
                        onIncrementTap: () {
                            print('Увеличить количество');
                        },
                        onDecrementTap: () {
                            print('Уменьшить количество');
                        },
                        size: const Size(24, 24),
                        padding: 0,
                        label: Text(

```

```
        sticker.quantity.toString(),
        style:
Theme.of(context).textTheme.displayMedium,
),
),
Text(
"\$10",
style:
AppTextStyle.h2Style.copyWith(
color: AppColor.accent
),
),
],
),
),
);
},
separatorBuilder: (_, __) => Container(
height: 20,
),
);
}
}
```



К каждой карточке корзины добавим удаление смахиванием с экрана. Для этого надо изменить верстку.

!!!! РАССКАЗАТЬ ПРО БИБЛИОТЕКУ

import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart';

Container из верстки списка обернем виджетом Dismissible.

```
itemBuilder: (_, index) {
  final sticker = cartItems[index];
  return Dismissible(
    direction: DismissDirection.endToStart,
    onDismissed: (direction) {
      if (direction == DismissDirection.endToStart) {
        print('Удаляем');
      }
    },
  ),
```

```
        key: UniqueKey(),
        background: Row(...),
        child: Container(...)
      );
},
}
```

Верстка background:

```
background: Row(
  children: [
    Container(
      padding: const EdgeInsets.symmetric(
        horizontal: 15,
        vertical: 25,
      ),
      decoration: BoxDecoration(
        color: Colors.redAccent,
        borderRadius: BorderRadius.circular(15),
      ),
      child: const FaIcon(FontAwesomeIcons.trash),
    ),
  ],
)
```

Проверяем работу верстки. Из списка еды смахиванием с экрана справа налево и наклейка удаляется из корзины.

```
% git add .
% git commit -m'Верстка списка корзины'
% git branch
...
* sun_4_5
  master
```

## 4.6 ВЕРСТКА СТОИМОСТИ КОРЗИНЫ

```
git checkout -b sun_4_6
```

Верстку стоимости корзины делаем в методе `_bottomAppBar()`.

```
Widget _bottomAppBar() {
```

```
return ClipRRect(
  borderRadius: const BorderRadius.only(
    topLeft: Radius.circular(30),
    topRight: Radius.circular(30),
  ),
  child: BottomAppBar(
    child: SizedBox(
      height: 250,
      child: Container(
        color: Theme.of(context).brightness ==
          Brightness.dark
            ? AppColor.dark
            : Colors.white,
        child: Padding(
          padding: const EdgeInsets.all(30),
          child: SingleChildScrollView(
            child: Column(
              children: [
                Padding(
                  padding:
                    const EdgeInsets.symmetric(
                      horizontal: 20
                    ),
                ),
                child: Row(
                  mainAxisAlignment:
                    MainAxisAlignment.spaceBetween,
                  children: [
                    Text(
                      "Subtotal",
                      style:
                        Theme.of(context).textTheme.headlineSmall,
                    ),
                    Text(
                      "\$111",
                      style:
                        Theme.of(context).textTheme.displayMedium,
                    ),
                  ],
                ),
                const SizedBox(height: 15),
                Padding(
                  padding:
                    const EdgeInsets.symmetric(horizontal: 20),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  ),
);
```

```
        child: Row(
            mainAxisAlignment:
                MainAxisAlignment.spaceBetween,
            children: [
                Text(
                    "Taxes",
                    style:
                        Theme.of(context).textTheme.headlineSmall,
                ),
                Text(
                    "\$\{5.00}",
                    style:
                        Theme.of(context).textTheme.displayMedium,
                ),
            ],
        ),
        const Padding(
            padding:
                EdgeInsets.symmetric(horizontal: 20),
            child: Divider(thickness: 4.0,
                height: 30.0),
        ),
        Padding(
            padding:
                const EdgeInsets.symmetric(horizontal: 20),
            child: Row(
                mainAxisAlignment:
                    MainAxisAlignment.spaceBetween,
                children: [
                    Text(
                        "Total",
                        style:
                            Theme.of(context).textTheme.displayMedium,
                    ),
                    Text(
                        "\$120.0",
                        style:
                            AppTextStyle.h2Style.copyWith(
                                color: AppColor.accent,
                            ),
                    ),
                ],
            ),
        ),
    ],
),
```

```
        ),
        const SizedBox(height: 30),
        SizedBox(
            width: double.infinity,
            height: 45,
            child: Padding(
                padding:
                    const EdgeInsets.symmetric(horizontal: 30),
                child: ElevatedButton(
                    onPressed: () {},
                    child:
                        const Text("Checkout"),
                    ),
                ),
            ),
        ],
    ),
),
));
);
};

}
```

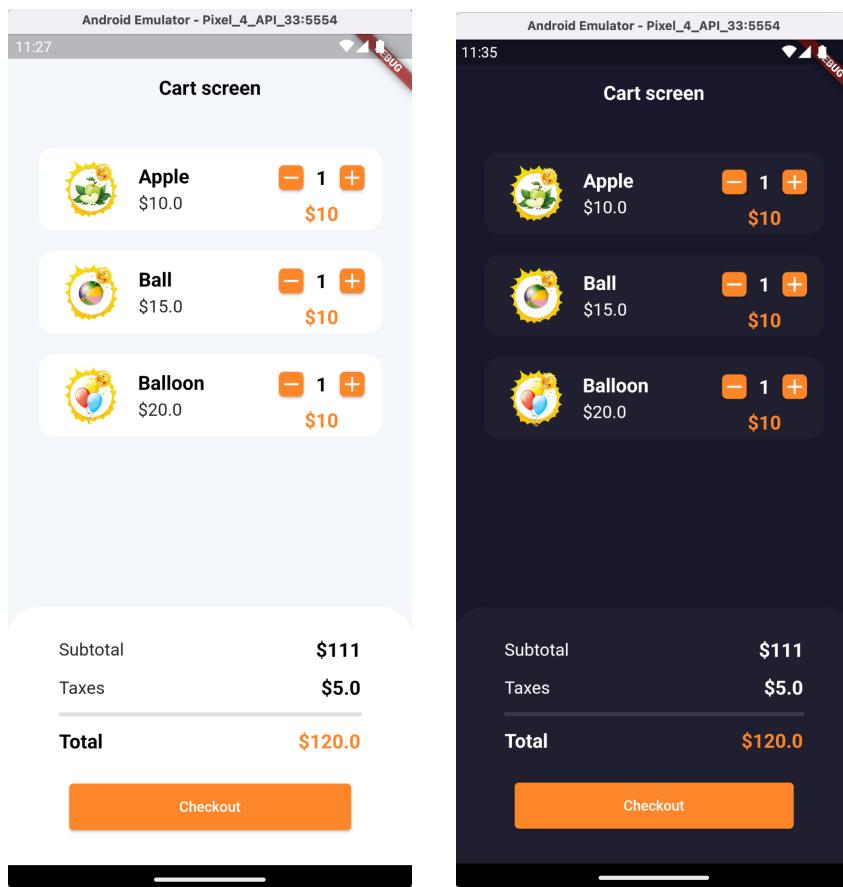
Для стилизации кнопки ElevatedButton воспользуемся темой.

```
class AppTheme {  
    const AppTheme._();  
  
    static ThemeData lightTheme = ThemeData(  
        ...  
        elevatedButtonTheme: ElevatedButtonThemeData(  
            style: ButtonStyle(  
                backgroundColor:  
                    MaterialStateProperty.all<Color>(  
                        LightThemeColor.accent,  
                    ),  
                ),  
            ),  
        ),  
        ...  
    );  
  
    static ThemeData darkTheme = ThemeData(  
        ...  
        elevatedButtonTheme: ElevatedButtonThemeData(  
            ...  
        ),  
    );  
}
```

```

        style: ButtonStyle(
      backgroundColor:
        MaterialStateProperty.all<Color>(
          LightThemeColor.accent,
        ),
        ),
      ),
    ),
    ...
);

```



```

% git add .
% git commit -m'Верстка стоимости корзины'
% git branch
...
* sun_4_6
  master

```

## 5. ВЕРСТКА ЛЮБИМЫХ НАКЛЕЕК

### 5.1 ЭКРАН FAVORITESCREEN

```
git checkout -b sun_5_1
```

В папке экранов ui/screens создаем новый файл favorite\_screen.dart.

Листинг файла ui/screens/favorite\_screen.dart.

```
import 'package:flutter/material.dart';

class FavoriteScreen extends StatefulWidget {
  const FavoriteScreen({super.key});

  @override
  State<FavoriteScreen> createState() =>
  FavoriteScreenState();
}

class FavoriteScreenState extends State<FavoriteScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold();
  }
}
```

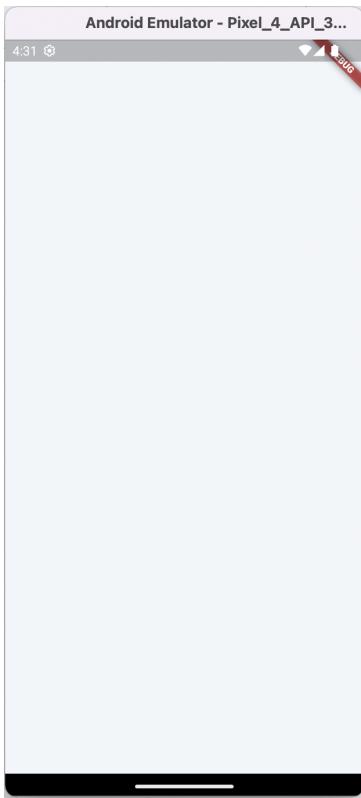
Листинг файла ui/screens/\_screens.dart.

```
export 'cart_screen.dart';
export 'favorite_screen.dart';
export 'sticker_list_screen.dart';
```

Подключим экран корзины в MaterialApp.

```
MaterialApp(
  title: 'Sunny Stickers',
  theme: AppTheme.lightTheme,
  home: const FavoriteScreen(),
);
```

Запустим проект.



Можно начинать верстку.

```
% git add .
% git commit -m'Экран любимых наклеек'
% git branch
  ...
* sun_5_1
  master
```

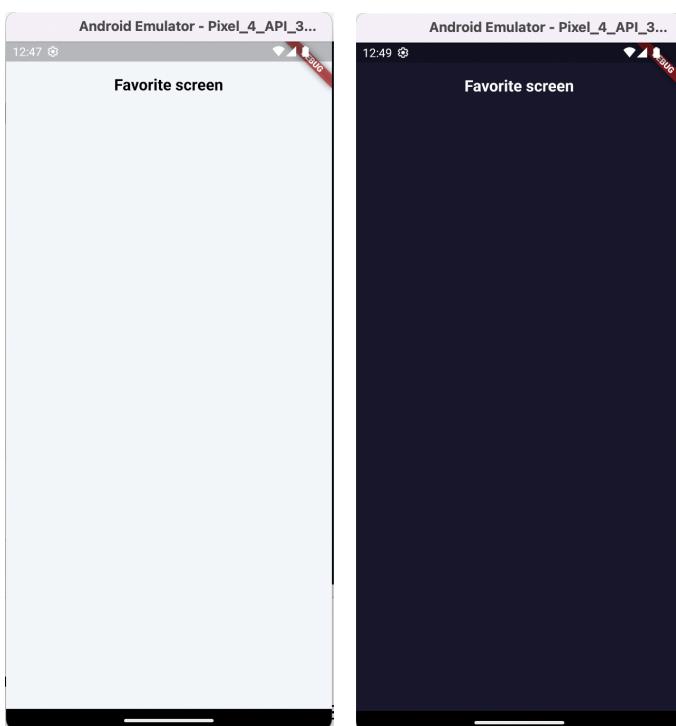
## 5.2 APPBAR И EMPTYWRAPPER ЭКРАНА ЛЮБИМЫХ НАКЛЕЕК

```
git checkout -b sun_5_2
```

Аналогично верстке экрана StickerList и CartScreen вынесем создание AppBar в отдельную функцию.

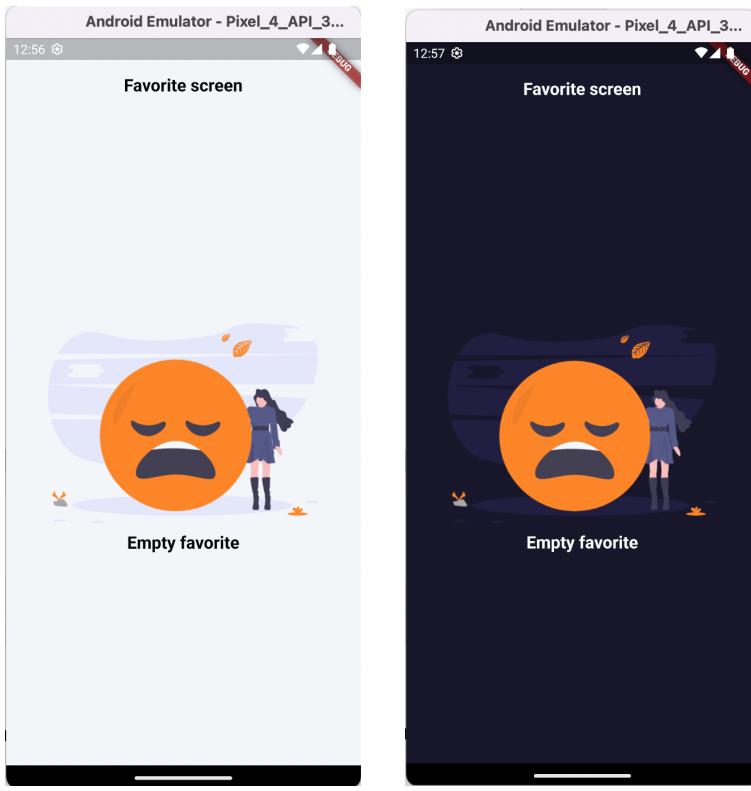
```
class FavoriteScreenState extends State<FavoriteScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: _appBar(context),
    );
}
```

```
PreferredSizeWidget _appBar(BuildContext context) {  
    return AppBar(  
        title: Text(  
            "Favorite screen",  
            style: Theme.of(context).textTheme.displayMedium,  
        ),  
    );  
}  
}
```



Когда разрабатывалась обертка пустой корзины была заложена обертка пустого экрана любимой еды, поэтому обертку пустого экрана надо просто подключить.

```
Scaffold(  
    appBar: _appBar(context),  
    body: EmptyWrapper(  
        type: EmptyWrapperType.favorite,  
        title: 'Empty favorite',  
        isEmpty: true,  
        child: Container(),  
    ),  
);
```



```
% git add .
% git commit -m'AppBar и EmptyWrapper экрана любимых
наклеек'
% git branch
...
* sun_5_2
master
```

### 5.3 ВЕРСТКА КАРТОЧКИ ЛЮБИМОЙ НАКЛЕЙКИ

```
git checkout -b sun_5_3
```

В файл `data/app_data.dart` добавим данные для экрана любимых наклеек.

```
static List<Sticker> favoriteItems = [
  stickers[0]..isFavorite = true,
  stickers[1]..isFavorite = true,
  stickers[2]..isFavorite = true
];
```

Теперь экран любимых наклеек не пуст.

```
class FavoriteScreenState extends State<FavoriteScreen> {
  var favoriteItems = AppData.favoriteItems;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: _appBar(context),
      body: EmptyWrapper(
        type: EmptyWrapperType.favorite,
        title: "Empty favorite",
        isEmpty: favoriteItems.isEmpty,
        child: Container(),
      ),
    );
  }
}
```

Создадим функцию, которая будет отвечать за список любимых наклеек.

```
Widget _favoriteListView() {
  return ListView.separated(
    padding: const EdgeInsets.all(30),
    itemCount: 20,
    itemBuilder: (_, index) {
      return Container(
        height: 60,
        width: double.infinity,
        padding: const EdgeInsets.all(5),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
          color: Colors.white,
        ),
        child: Text(
          '${index + 1}',
          style: const TextStyle(fontSize: 24,
          fontWeight: FontWeight.bold),
        ),
      );
    },
    separatorBuilder: (_, __) => Container(
      height: 20,
    ),
  );
}
```

```
    );  
}
```

И подключим.

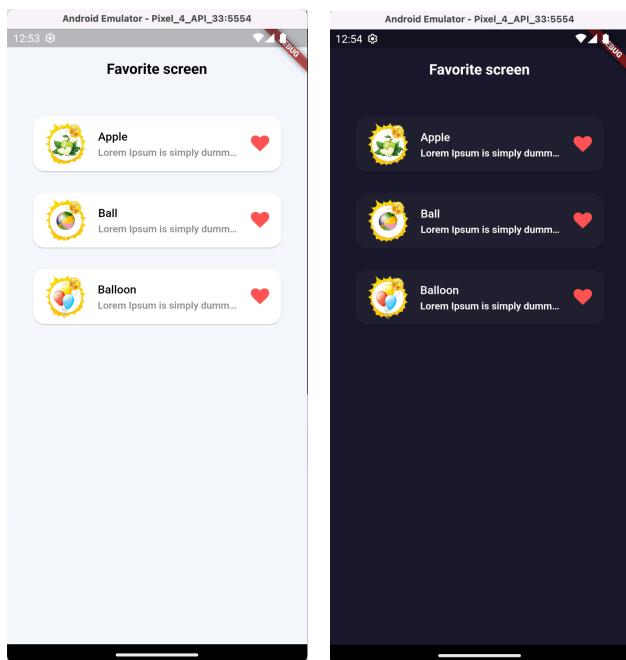
```
Scaffold(  
    appBar: _appBar(context),  
    body: EmptyWrapper(  
        type: EmptyWrapperType.favorite,  
        title: 'Empty favorite',  
        isEmpty: favoriteItems.isEmpty,  
        child: _favoriteListView(),  
    ),  
) ;
```



Используем готовые виджеты `Card` и `ListTile` для верстки элементов списка любимых наклеек.

```
itemCount: favoriteItems.length,  
itemBuilder: (_, index) {  
    Sticker sticker = favoriteItems[index];  
    return Card(  
        color: Theme.of(context).brightness ==  
            Brightness.light  
            ? Colors.white  
            : AppColor.dark,  
        shape: RoundedRectangleBorder(  
            borderRadius: BorderRadius.circular(15.0),
```

```
),
child: ListTile(
  title: Text(
    sticker.name,
    style:
      Theme.of(context).textTheme.headlineMedium,
  ),
  leading: Image.asset(sticker.image),
  subtitle: Text(
    sticker.description,
    overflow: TextOverflow.ellipsis,
    style: Theme.of(context).textTheme.bodyLarge,
  ),
  trailing: const Icon(AppIcon.heart, color:
Colors.redAccent),
),
);
},
```



```
% git add .
% git commit -m'Верстка карточки любимой наклейки'
% git branch
...
* sun_5_3
master
```

## 5.4 ЭКРАН ПРОФИЛЯ

```
git checkout -b sun_5_4
```

В папке экранов ui/screens создаем новый файл profile\_screen.dart.

Листинг файла ui/screens/profile\_screen.dart.

```
import 'package:flutter/material.dart';

import ' ../../ui_kit/_ui_kit.dart';

class ProfileScreen extends StatelessWidget {
  const ProfileScreen({Key? key}) : super(key: key);

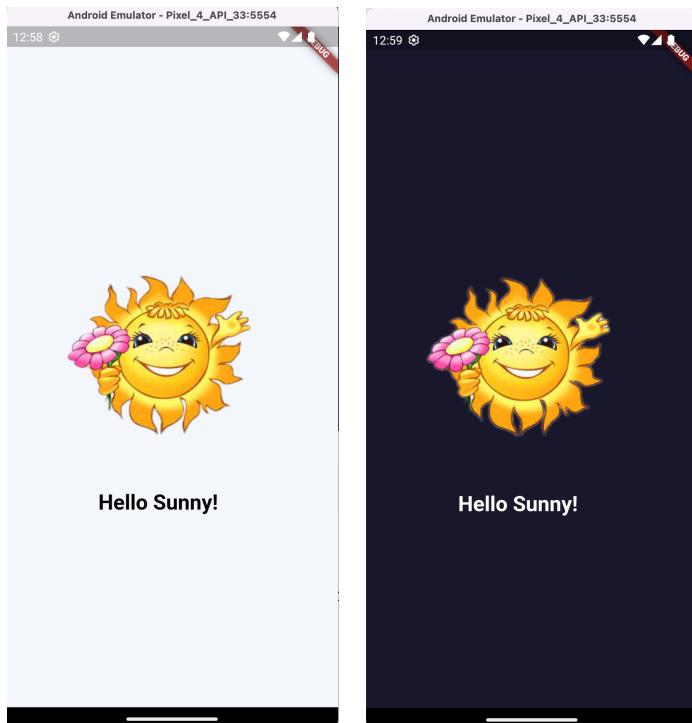
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.all(30),
            child: Image.asset(AppAsset.profileImage,
              width: 300),
          ),
          Text(
            "Hello Sunny!",
            style:
                Theme.of(context).textTheme.displayLarge,
          ),
        ],
      );
  }
}
```

Подключим экран корзины в MaterialApp.

```
MaterialApp(
  title: 'Sunny Stickers',
  theme: AppTheme.lightTheme,
```

```
    home: const ProfileScreen(),  
);
```

Запустим проект.



```
% git add .  
% git commit -m 'Экран профиля'  
% git branch  
...  
* sun_5_4  
master
```

## 6. ВЕРСТКА ДЕТАЛЕЙ

### 6.1 ЭКРАН ДЕТАЛЕЙ НАКЛЕЙКИ

```
git checkout -b sun_6_1
```

В папке экранов ui/screens создаем новый файл `sticker_detail_screen.dart`.

Листинг файла `ui/screens/sticker_detail_screen.dart`.

```
import 'package:flutter/material.dart';  
  
class StickerDetail extends StatefulWidget {  
  const StickerDetail({super.key});
```

```
  @override
  State<StickerDetail> createState() =>
  StickerDetailState();
}

class StickerDetailState extends State<StickerDetail> {
  @override
  Widget build(BuildContext context) {
    return Scaffold();
  }
}
```

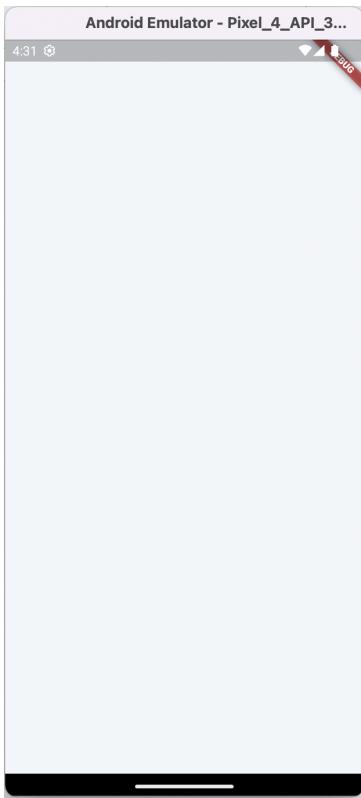
Листинг файла ui/screens/\_screens.dart.

```
export 'cart_screen.dart';
export 'favorite_screen.dart';
export 'profile_screen.dart';
export 'sticker_detail_screen.dart';
export 'sticker_list_screen.dart';
```

Подключим экран корзины в MaterialApp.

```
MaterialApp(
  title: 'Sunny Stickers',
  theme: AppTheme.lightTheme,
  home: const StickerDetail(),
);
```

Запустим проект.



Можно начинать верстку.

```
% git add .
% git commit -m'Экран деталей'
% git branch
  ...
* sun_6_1
  master
```

## 6.2 APPBAR ЭКРАНА ДЕТАЛЕЙ

```
git checkout -b sun_6_2
```

Аналогично верстке экрана StickerList, CartScreen, FavoriteScreen вынесем создание AppBar в отдельную функцию.

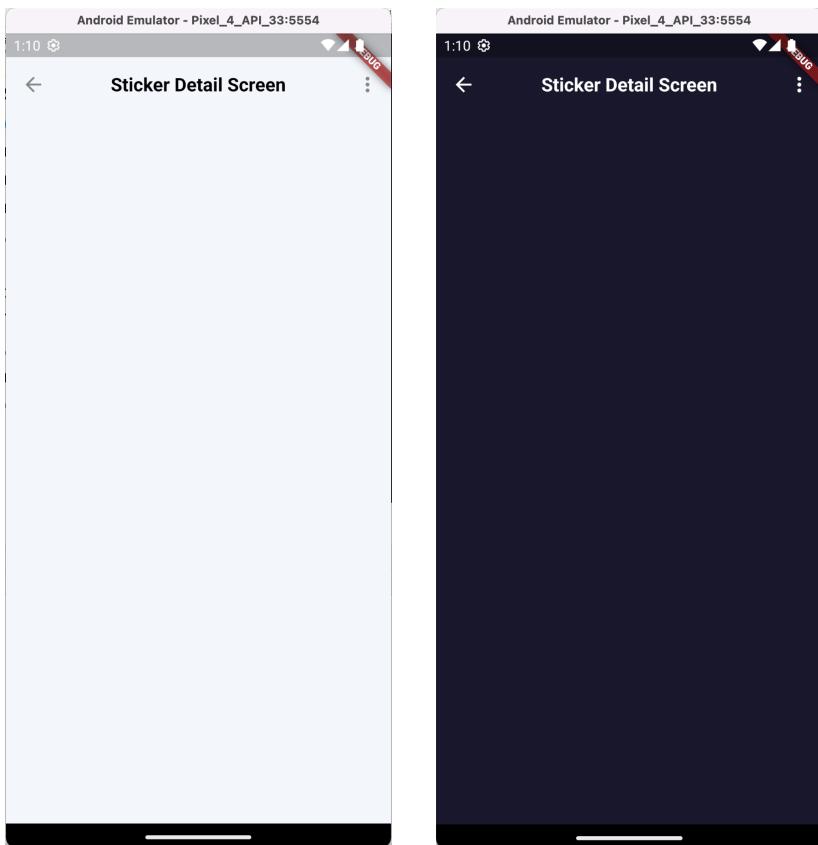
```
class StickerDetailState extends State<StickerDetail> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
        appBar: _appBar(context),
    );
}

PreferredSizeWidget _appBar(BuildContext context) {
    return AppBar(
        ...
    );
}
}
```

В AppBar определим leading, title, actions.

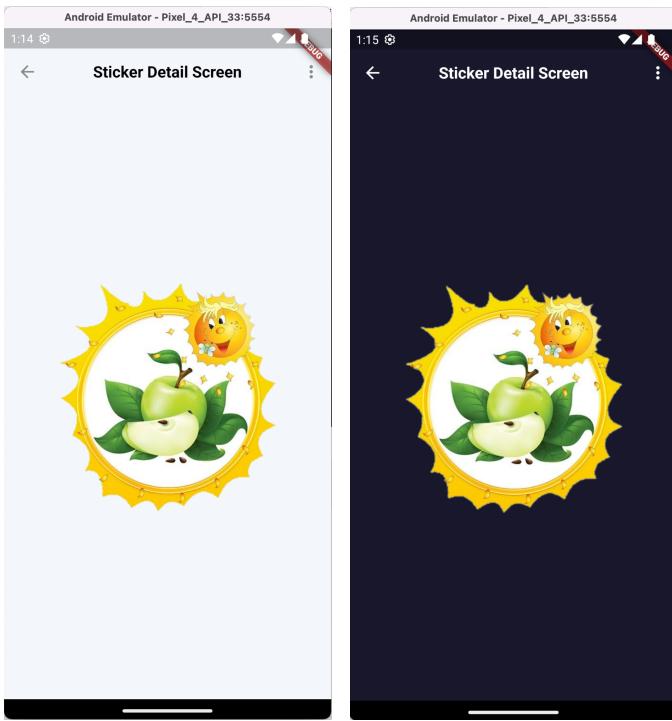
```
PreferredSizeWidget _appBar(BuildContext context) {
    return AppBar(
        leading: IconButton(
            onPressed: () {},
            icon: Icon(Icons.arrow_back),
        ),
        title: Text(
            'Sticker Detail Screen',
            style: TextStyle(color:
Theme.of(context).brightness == Brightness.light ?
Colors.black : Colors.white),
        ),
        actions: [IconButton(onPressed: () {}, icon:
Icon(Icons.more_vert))],
    );
}
```



Следующая, очень простая часть верстки это body.

```
class StickerDetailState extends State<StickerDetail> {
    final sticker = AppData.stickers[0];

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: _appBar(context),
            body: Center(child: Image.asset(sticker.image,
scale: 2)),
        );
    }
}
```



Следующим шагом добавим FloatingActionButton.

```
Widget _floatingActionButton() {
    return FloatingActionButton(
        elevation: 0.0,
        backgroundColor: AppColor.accent,
        onPressed: () {},
        child: sticker.isFavorite ? const Icon(AppIcon.heart)
: const Icon(AppIcon.outlinedHeart),
    );
}
```

Подключим кнопку.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: _appBar(context),
        body: Center(
            child: Image.asset(sticker.image, scale: 2)
        ),
        floatingActionButton: _floatingActionButton(),
        floatingActionButtonLocation:
            FloatingActionButtonLocation.endDocked,
    );
}
```

```
% git add .
% git commit -m'AppBar экрана деталей'
% git branch
...
* sun_6_2
  master
```

## 6.3 ИНФОРМАЦИЯ О НАКЛЕЙКЕ

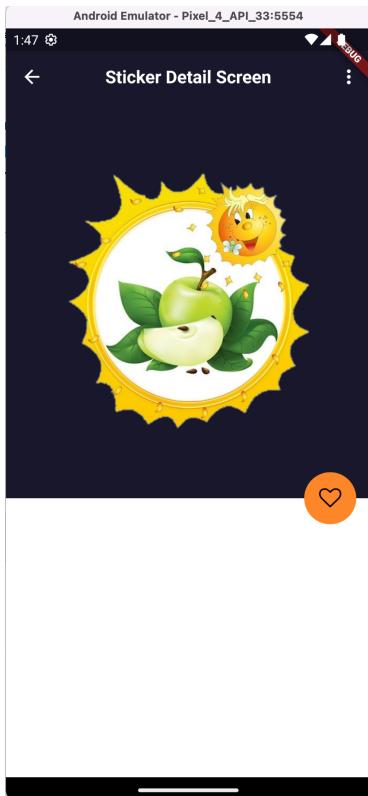
```
git checkout -b sun_6_3
```

Информацию о выбранной наклейке будем отображать в блоке bottomNavigationBar. И снова создадим метод в котором сосредоточим верстку детальной информации.

```
Widget _bottomAppBar() {
    return BottomAppBar(
        child: SizedBox(
            height: 300,
            child: Container(
                color: Colors.white,
            )));
}
```

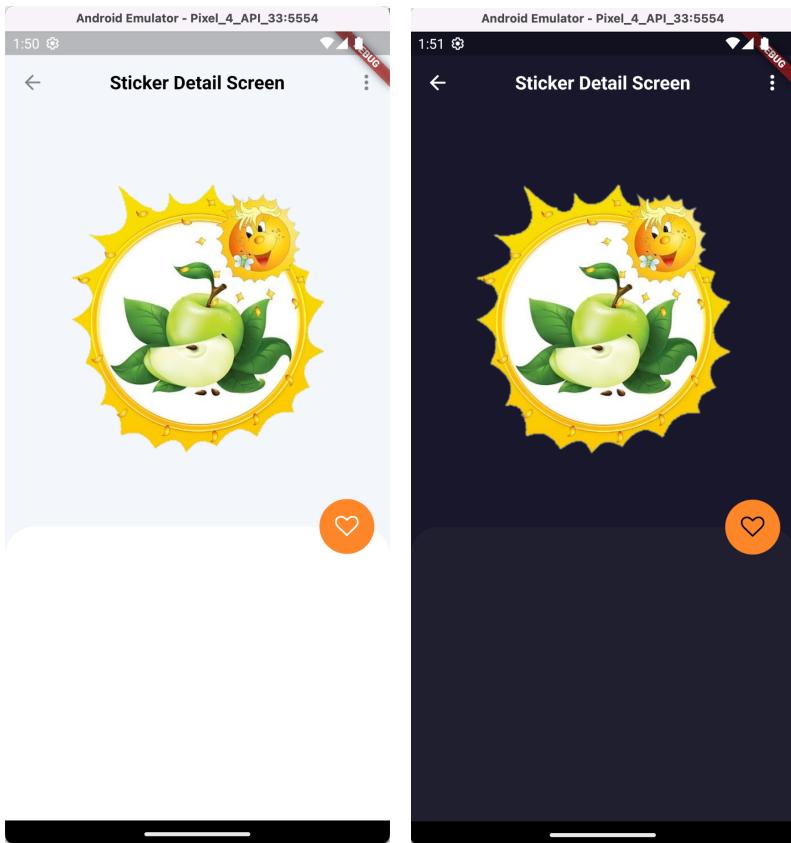
Подключим bottomNavigationBar.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: _appBar(context),
        body: Center(child: Image.asset(sticker.image, scale: 2)),
        floatingActionButton: _floatingActionButton(),
        floatingActionButtonLocation:
            FloatingActionButtonLocation.endDocked,
        bottomNavigationBar: _bottomAppBar(),
    );
}
```



Срежем углы и добавим темы.

```
Widget _bottomAppBar() {
    return ClipRRect(
        borderRadius: const BorderRadius.only(
            topLeft: Radius.circular(30),
            topRight: Radius.circular(30),
        ),
        child: BottomAppBar(
            child: SizedBox(
                height: 300,
                child: Container(
                    color: Theme.of(context).brightness ==
Brightness.dark ? AppColor.dark : Colors.white,
                ))));
}
```



Подготовим отступы, скрол и колонку.

```
Widget _bottomAppBar() {
  return ClipRRect(
    borderRadius: const BorderRadius.only(
      topLeft: Radius.circular(30),
      topRight: Radius.circular(30),
    ),
    child: BottomAppBar(
      child: SizedBox(
        height: 300,
        child: Container(
          color: Theme.of(context).brightness == Brightness.dark
            ? AppColor.dark : Colors.white,
          child: Padding(
            padding: const EdgeInsets.all(30),
            child: SingleChildScrollView(
              child: Column(
                children: [],
              ),
            ),
          ),
        ),
      ),
    ),
  );
}
```

```
        ))));
}
```

В колонке расположим 9 блоков.

Первый блок: рейтинг



Воспользуемся готовой библиотекой.

```
flutter_rating_bar: ^4.0.1
```

Код первого блока:

```
Row(
    children: [
        RatingBar.builder(
            itemPadding: EdgeInsets.zero,
            itemSize: 20,
            initialRating: sticker.score,
            minRating: 1,
            direction: Axis.horizontal,
            allowHalfRating: true,
            itemCount: 5,
            glow: false,
            ignoreGestures: true,
            itemBuilder: (_, __) => const FaIcon(
                FontAwesomeIcons.solidStar,
                color: AppColor.yellow,
            ),
            onRatingUpdate: (rating) {
                print('$rating');
            },
        ),
        const SizedBox(width: 15),
        Text(
            sticker.score.toString(),
            style: Theme.of(context).textTheme.titleMedium,
        ),
        const SizedBox(width: 5),
        Text(

```

```
        "(${sticker.voter})",
        style: Theme.of(context).textTheme.titleMedium,
    ),
),
)
```

Второй блок: Отступ.

```
const SizedBox(height: 15),
```

Третий блок: Цена и управление количеством.

```
Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
        Text(
            "\$${sticker.price}",
            style:
Theme.of(context).textTheme.displayLarge?.copyWith(color:
AppColor.accent),
        ),
        CounterButton(
            onIncrementTap: () {},
            onDecrementTap: () {},
            label: Text(
                sticker.quantity.toString(),
                style: Theme.of(context).textTheme.displayLarge,
            ),
        ),
    ],
)
```

Четвертый блок: Отступ.

```
const SizedBox(height: 15),
```

Пятый блок: Заголовок.

```
Text(
    "Description",
    style: Theme.of(context).textTheme.displayMedium,
)
```

Шестой блок: Отступ.

```
const SizedBox(height: 15),
```

Седьмой блок: Описание.

```
Text(  
  sticker.description,  
  style: Theme.of(context).textTheme.titleMedium,  
)
```

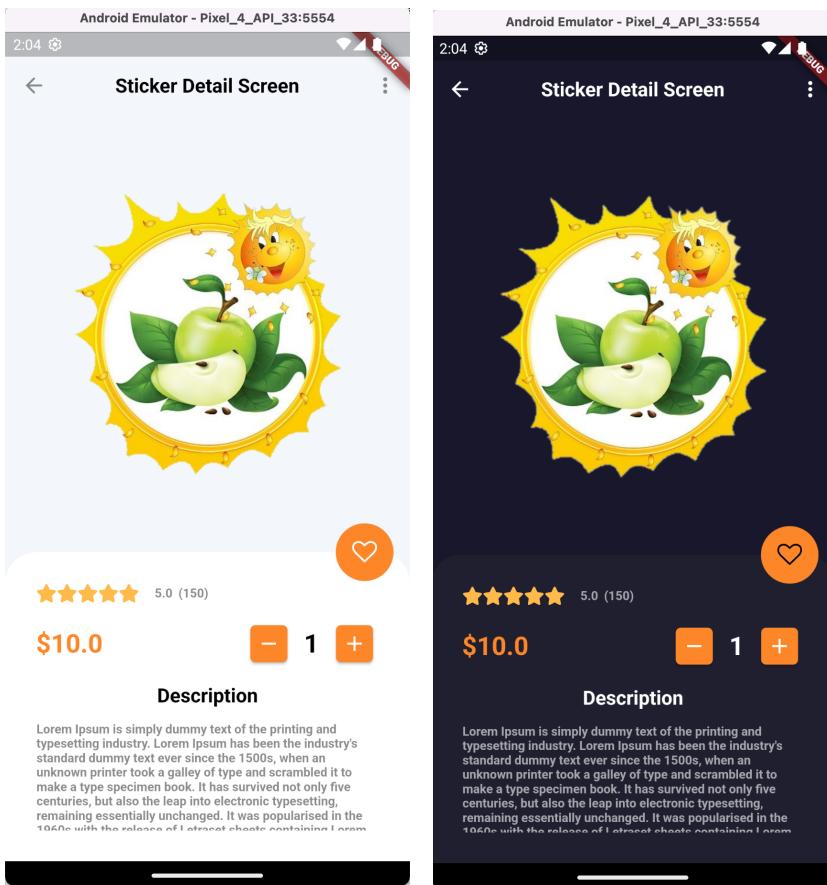
Восьмой блок: Отступ.

```
const SizedBox(height: 30),
```

Девятый блок: Кнопка.

```
SizedBox(  
  width: double.infinity,  
  height: 45,  
  child: Padding(  
    padding: const EdgeInsets.symmetric(horizontal: 30),  
    child: ElevatedButton(  
      onPressed: (){},  
      child: const Text("Add to cart"),  
    ),  
  ),  
)
```

Собираем все блоки в колонку и получаем.



```
% git add .
% git commit -m 'AppBar экрана деталей'
% git branch
...
* sun_6_3
master
```

## 7. НАВИГАЦИЯ

### 7.1 TABBAR

```
git checkout -b sun_7_1
```

В папке экранов ui/screens создаем новый файл `home_screen.dart`.

Листинг файла `ui/screens/home_screen.dart`.

```
import 'package:flutter/material.dart';
```

```
class HomeScreen extends StatefulWidget {
```

```
const HomeScreen({Key? key}) : super(key: key);

@Override
State<HomeScreen> createState() => HomeScreenState();
}

class HomeScreenState extends State<HomeScreen> {
@Override
Widget build(BuildContext context) {
    return Container();
}
}
```

Листинг файла ui/screens/\_screens.dart.

```
export 'cart_screen.dart';
export 'favorite_screen.dart';
export 'home_screen.dart';
export 'profile_screen.dart';
export 'sticker_detail_screen.dart';
export 'sticker_list_screen.dart';
```

Для организации TabBar понадобятся две переменные.

```
final List<Widget> screens = [const StickerList(), const
CartScreen(), const FavoriteScreen(), const
ProfileScreen()];
int currentIndex = 0;
```

В Scaffold определяем два параметра:

- 1) body – туда помещаем IndexStack (с текущим индексом и массивом экранов)
- 2) bottomNavigationBar – с массивом BottomNavigationBarItem

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        body: SafeArea(
            child: IndexedStack(
                index: currentIndex,
```

```
        children: screens,
    ),
),
bottomNavigationBar: BottomNavigationBar(
    currentIndex: currentIndex,
    onTap: (index) {},
    selectedFontSize: 0,
    items: AppData.bottomNavigationItems.map(
        (element) {
            return BottomNavigationBarItem(
                icon: element.disableIcon,
                label: element.label,
                activeIcon: element.enableIcon,
            );
        },
    ).toList(),
),
);
}
```

Стиль для BottomNavigationBarItem добавим в тему.

```
class AppTheme {
const AppTheme._();

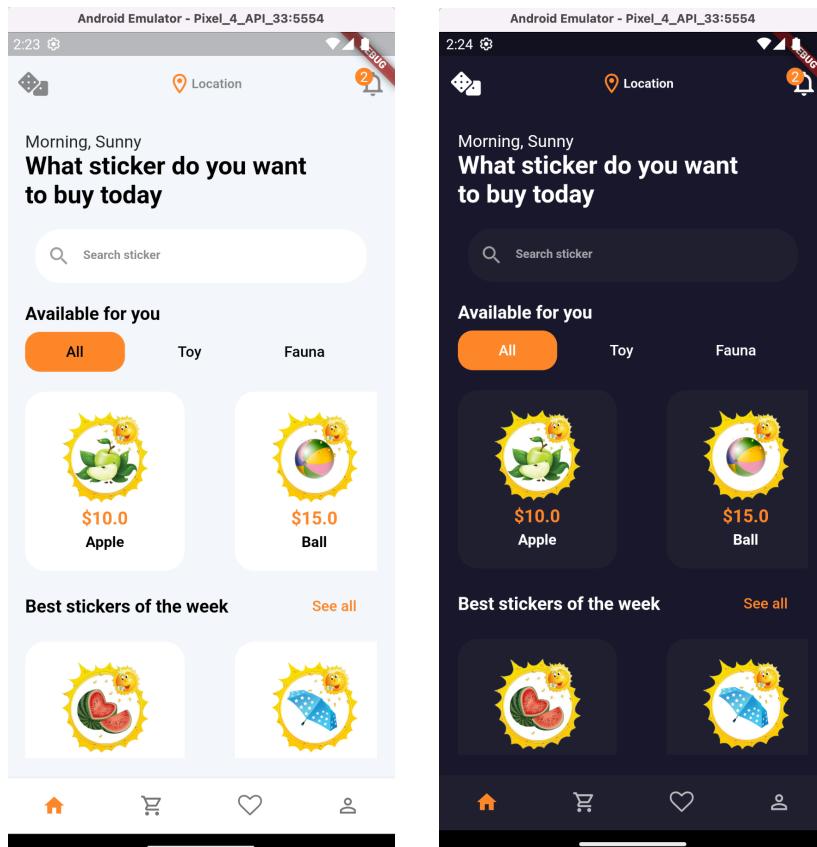
static ThemeData lightTheme = ThemeData(
...
bottomNavigationBarTheme: const
BottomNavigationBarThemeData(
    type: BottomNavigationBarType.fixed,
    backgroundColor: Colors.white,
    selectedItemColor: LightThemeColor.accent,
),
iconTheme: const IconThemeData(color:
    Colors.black45),
bottomAppBarTheme: const BottomAppBarTheme(color:
    Colors.white),
);

static ThemeData darkTheme = ThemeData(
...
```

```
bottomNavigationBarTheme: const  
BottomNavigationBarThemeData(  
    type: BottomNavigationBarType.fixed,  
    backgroundColor: DarkThemeColor.primaryLight,  
    selectedItemColor: LightThemeColor.accent,  
    unselectedItemColor: Colors.white70,  
,  
iconTheme: const IconThemeData(color: Colors.white),  
bottomAppBarTheme: const BottomAppBarTheme(  
color: DarkThemeColor.primaryLight,  
,  
);
```

Подключаем TabBar в MaterialApp.

```
MaterialApp(  
    title: 'Sunny Stickers',  
    theme: AppTheme.lightTheme,  
    home: const HomeScreen(),  
) ;
```



```
% git add .  
% git commit -m'TabBar'
```

```
% git branch
...
* sun_7_1
  master
```

## 7.2 TABBAR НАВИГАЦИЯ

```
git checkout -b sun_7_2
```

На экране ui/screens/home\_screen.dart подключим переключение табов. Переменная обеспечивающая переключение табов currentIndex:

```
class HomeScreenState extends State<HomeScreen> {
  ...
  int currentIndex = 0;
  ...
}
```

Шаги подключения:

1) Создаем метод, который по клику переопределяет currentIndex и запускает перерисовку экрана  
setState((){})

```
class HomeScreenState extends State<HomeScreen> {
  ...
  int currentIndex = 0;

  void onTabTap(int index) {
    if (currentIndex == index) return;
    currentIndex = index;
    setState(() {});
  }
  ...
}
```

2) Подключаем обработчик клика

```
BottomNavigationBar(
  currentIndex: currentIndex,
  onTap: onTabTap,
```

...

Переключение табов работает.

```
% git add .
% git commit -m'TabBar навигация'
% git branch
...
* sun_7_2
  master
```

## 7.3 НАВИГАЦИЯ НА ДЕТАЛЬНЫЙ ПРОСМОТР СТРАНИЦЫ

```
git checkout -b sun_7_3
```

При нажатии на каждую карточку наклейки откроем экран детального просмотра наклейки.

В файле ui/widgets/sticker\_list\_view.dart находим место клика на карточку

```
ListView.separated(
  ...
  itemBuilder: (_, index) {
    Sticker sticker = isReversed ?
      stickers.reversed.toList()[index] :
      stickers[index];
    return GestureDetector(
      onTap: () {
        print('Клик на карточку');
      },
      child: Container(
        width: 160,
      ...
    
```

Реализуем самый простой переход на экран детального просмотра.

```
onTap: () {
  print('Клик на карточку');
```

```
        Navigator.of(context).push(
            MaterialPageRoute(
                builder: (_) => const StickerDetail()
            )
        );
    },
},
```

Переход на экран детального просмотра.

Рассмотрим два варианта возвращения на список наклеек.

Первый вариант:

В файле ui/screens/sticker\_detail\_screen.dart, в методе \_appBar закомментируем поле leading. В этом случае метод push из пункта 7.2 обеспечит отображение стрелочки в навигационной панели и возвращение на экран списка наклеек.

```
PreferredSizeWidget _appBar(BuildContext context) {
    return AppBar(
        // leading: IconButton(
        //     onPressed: () {},
        //     icon: const Icon(Icons.arrow_back),
        // ),
        title: Text(
            'Sticker Detail Screen',
            style: TextStyle(color:
Theme.of(context).brightness == Brightness.light ?
Colors.black : Colors.white),
        ),
        actions: [IconButton(
            onPressed: () {}, icon:
Icon(Icons.more_vert))],
    );
}
```

Второй вариант: Возвращение на предыдущий экран вручную.

```
PreferredSizeWidget _appBar(BuildContext context) {
    return AppBar(
        leading: IconButton(
            onPressed: () {
                Navigator.of(context).pop();
            }
        ),
    );
}
```

```
        },
        icon: const Icon(Icons.arrow_back),
    ),
    title: Text(
        'Sticker Detail Screen',
        style: TextStyle(color:
Theme.of(context).brightness == Brightness.light ?
Colors.black : Colors.white),
    ),
    actions: [IconButton(
        onPressed: () {},
        icon: Icon(Icons.more_vert))],
)
);
}
```

onPressed можно записать короче с использованием стрелочной функции.

```
leading: IconButton(
    onPressed: () => Navigator.of(context).pop(),
    icon: const Icon(Icons.arrow_back),
),
```

Вся навигация подключена.

```
% git add .
% git commit -m'Навигация на детальный просмотр'
% git branch
...
* sun_7_3
  master
```

Верстку приложения закончили. Переключимся на ветку master и вмержим в мастер всю верстку.

```
% git checkout master
Switched to branch 'master'
% git merge sun_7_3
```

## 8. БИЗНЕС ЛОГИКА БЕЗ ДОП. БИБЛИОТЕК

### 8.1 СТРУКТУРА СОСТОЯНИЯ И ДАННЫЕ ПЕРВОГО ЭКРАНА

```
git checkout -b sun_8_1
```

В классе состояния StickerState реализуем бизнес логику приложения.

14 шагов логики

1. Подсветка выбранной категории
2. Продукты по категории
3. Детали: отображение продукта
4. Детали: количество
5. Корзина: управление пустой корзиной
6. Детали: добавление в корзину
7. Корзина: список в корзине
8. Корзина: стоимость корзины
9. Корзина: количество
10. Корзина: удаление
11. Корзина: чистка корзина на checkout
12. Любимые: управление пустым экраном
13. Детали: Добавление/удаление любимые
14. Смена темы

Состояние разобьем на четыре логические части.

```
class StickerState {  
    StickerState._();  
    static final _instance = StickerState._();  
    factory StickerState() => _instance;  
  
    //Ключи  
  
    //Переменные
```

```
//Действия  
//Вспомогательные методы  
}
```

Логику приложения выстроим на 6-ти переменных. Из 6-ти переменных 2 переменные основные.

```
List<StickerCategory> categories = AppData.categories;  
List<Sticker> stickers = AppData.stickers;
```

Кроме двух основных переменных нам понадобятся еще 4 вспомогательные переменные.

```
List<Sticker> stickersByCategory = AppData.stickers;  
List<Sticker> get cart => stickers.where((element) =>  
element.cart).toList();  
List<Sticker> get favorite => stickers.where((element) =>  
element.isFavorite).toList();  
ValueNotifier<bool> isLigth = ValueNotifier(true);
```

Итого, всю логику приложения выстроим на 6 переменных. Кроме 6 переменных в классе StickerState будут два ключа, 8 действий и два вспомогательных метода.

```
import 'package:flutter/material.dart';  
  
import '../data/_data.dart';  
import '../ui/_ui.dart';  
  
class StickerState {  
  StickerState._();  
  static final _instance = StickerState._();  
  factory StickerState() => _instance;  
  
  //Ключи  
  GlobalKey<CartScreenState> cartKey = GlobalKey();  
  GlobalKey<FavoriteScreenState> favoriteKey =  
    GlobalKey();  
  
  //Переменные
```

```

List<StickerCategory> categories = AppData.categories;
List<Sticker> stickers = AppData.stickers;
List<Sticker> stickersByCategory = AppData.stickers;
List<Sticker> get cart => stickers.where((element) =>
element.cart).toList();
List<Sticker> get favorite => stickers.where((element) => element.isFavorite).toList();
ValueNotifier<bool> isLigth = ValueNotifier(true);

//Действия
Future<void> onCategoryTap(StickerCategory category)
async {}

Future<void> onIncreaseQuantityTap(Sticker sticker) async {}

Future<void> onDecreaseQuantityTap(Sticker sticker) async {}

Future<void> onAddToCartTap(Sticker sticker) async {}

Future<void> onRemoveFromCartTap(Sticker sticker) async {}

Future<void> onCheckOutTap() async {}

Future<void> onAddRemoveFavoriteTap(Sticker sticker)
async {}

void toggleTheme() {}

//Вспомогательные методы
String stickerPrice(Sticker sticker) {}

double get subtotal {}

```

Переключим данные первого экрана с AppData, на данные из StickerState.

Изменения на экране ui/screens/sticker\_list\_screen.dart.

Было:

```
Widget _categories() {
  final categories = AppData.categories;
```

Стало:

```
Widget _categories() {
    final categories = StickerState().categories;

Было (Продукты по категориям):
StickerListView(stickers: AppData.stickers),
Стало:
StickerListView(stickers:
                StickerState().stickersByCategory),
```

Было (Лучшие предложения недели):

```
StickerListView(
    stickers: AppData.stickers,
    isReversed: true,
),
Стало
StickerListView(
    stickers: StickerState().stickers,
    isReversed: true,
),
```

```
% git add .
% git commit -m'Структура данных'
% git branch
...
* sun_8_1
  master
```

## 8.2 ПОДСВЕТКА КАТЕГОРИИ НА ЭКРАНЕ СПИСКА НАКЛЕЕК

```
git checkout -b sun_8_2
```

Реализуем подсветку категорий в три шага.  
В классе состояния StickerState уже добавлен шаблон для клика на категорию.

```
Future<void> onCategoryTap(StickerCategory category)
async {
}
```

Первый шаг: Реализуем переключение выбранной категории.

```
Future<void> onCategoryTap(StickerCategory category)
async {
    categories.map((e) {
        if (e.type == category.type) {
            e.isSelected = true;
        } else {
            e.isSelected = false;
        }
    }).toList();
}
```

Второй шаг: Добавляем на экране ui/screens/sticker\_list\_screen.dart метод для вызова этого действия на экране.

```
void onCategoryTap(StickerCategory category) async {
    await StickerState().onCategoryTap(category);
    setState(() {});
}
```

Третий шаг: Поключаем метод к клику на категорию.

```
ListView.separated(
    scrollDirection: Axis.horizontal,
    itemBuilder: (_, index) {
        final category = categories[index];
        return GestureDetector(
            onTap: () {
                onCategoryTap(categories[index]);
            },
        );
    },
)
```

Проверяем, что подсветка выбора категории на экране списка работает.

```
% git add .
% git commit -m'Подсветка выбранной категории'
% git branch
...
* sun_8_2
  master
```

## 8.3 ПРОДУКТЫ ПО КАТЕГОРИЯМ НА ЭКРАНЕ НАКЛЕЕК

```
git checkout -b sun_8_3
```

Напомним, что для наклеек по категориям в классе состояния StickerState была заготовлена переменная.

```
//Переменные  
...  
List<Sticker> stickersByCategory = AppData.stickers;  
...
```

Эта переменная подключена к списку продуктов по категориям.

```
StickerListView(stickers:  
                StickerState().stickersByCategory)
```

Для реализации функциональности продуктов по категориям остается только дописать логику фильтрации продуктов в метод

```
Future<void> onCategoryTap(StickerCategory category)  
async {  
    ...  
}
```

А именно:

```
Future<void> onCategoryTap(StickerCategory category)  
async {  
    categories.map((e) {  
        if (e.type == category.type) {  
            e.isSelected = true;  
        } else {  
            e.isSelected = false;  
        }  
    }).toList();  
    if (category.type == StickerType.all) {  
        stickersByCategory = stickers;  
    } else {  
        stickersByCategory = stickers.where((e) =>  
            e.type == category.type).toList();  
    }  
}
```

```
    }  
}
```

Проверяем, что отображение наклеек по категориям работает.

```
% git add .  
% git commit -m'Наклейки по категориям'  
% git branch  
...  
* sun_8_3  
master
```

## 8.4 ДЕТАЛЬНЫЙ ПРОСМОТР НАКЛЕЙКИ

```
git checkout -b sun_8_4
```

Для детального просмотра наклейки передадим в конструктор класса `StickerDetail` (`ui/screens/sticker_detail_screen.dart`), выбранную наклейку.

Было:

```
class StickerDetailState extends State<StickerDetail> {  
  Sticker get sticker => AppData.stickers[0];  
  ...
```

Преобразуем.

Первый шаг. Добавляем новый параметр в конструктор.

```
class StickerDetail extends StatefulWidget {  
  const StickerDetail({super.key, required  
this.sticker});  
  final Sticker sticker;  
  
  @override  
  State<StickerDetail> createState() =>  
  StickerDetailState();  
}
```

Второй шаг. Передача наклейки на экран детального просмотра после клика (`ui/widgets/sticker_list_view.dart`).

```
...
    child: ListView.separated(
        scrollDirection: Axis.horizontal,
        padding: const EdgeInsets.only(top: 20),
        itemBuilder: (_, index) {
            Sticker sticker = isReversed
                ? stickers.reversed.toList()[index]
                : stickers[index];
            return GestureDetector(
                onTap: () {
                    print('Клик на карточку');
                    Navigator.of(context).push(MaterialPageRoute(
                        builder: (_) => StickerDetail(
                            sticker: sticker,
                        )));
                },
            );
        ...
    );
```

В файле состояния (`states/sticker_state.dart`) добавим переменную и метод установки этой переменной.

Третий шаг. Получение наклейки в классе `StickerDetailState` из `StickerDetail`. Класс `StickerDetail` доступен в `StickerDetailState` через переменную `widget`.

```
class StickerDetailState extends State<StickerDetail> {
    Sticker get sticker => widget.sticker;
    ...
}
```

Теперь можно проверить, что функциональность работает. При клике на наклейку детальный просмотр отображает выбранную наклейку.

```
% git add .
% git commit -m'Детальный просмотр наклейки'
% git branch
...
* sun_8_4
master
```

## 8.5 УПРАВЛЕНИЕ КОЛИЧЕСТВОМ НА ЭКРАНЕ ДЕТАЛЕЙ

```
git checkout -b sun_8_5
```

В классе состояния StickerState уже добавлены шаблоны увеличения и уменьшения количества наклейки (это 2 и 3 действие из восьми запланированных).

```
Future<void> onIncreaseQuantityTap(Sticker sticker) async
{
}
```

```
Future<void> onDecreaseQuantityTap(Sticker sticker) async
{
}
```

Реализуем управление количеством на экране деталий в три шага.

Первый шаг. Добавим в шаблоны логику.

```
Future<void> onIncreaseQuantityTap(Sticker sticker) async
{
    sticker.quantity++;
}
```

```
Future<void> onDecreaseQuantityTap(Sticker sticker) async
{
    if (sticker.quantity == 1) return;
    sticker.quantity--;
}
```

Второй шаг: Добавляем на экране ui/screens/sticker\_detail\_screen.dart методы для вызова этих действий на экране.

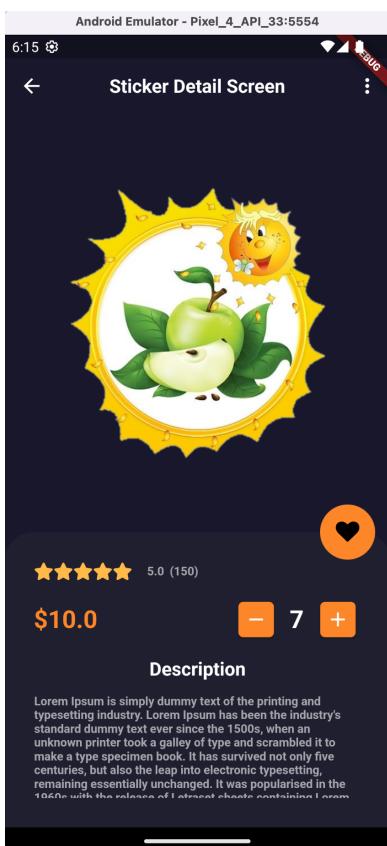
```
void onIncreaseQuantityTap() async {
    await StickerState().onIncreaseQuantityTap(sticker);
    setState(() {});
}
```

```
void onDecreaseQuantityTap() async {
    await StickerState().onDecreaseQuantityTap(sticker);
    setState(() {});
}
```

Третий шаг: Поключаем методы к кликам компонента CounterButton.

```
CounterButton(  
    onIncrementTap: onIncreaseQuantityTap,  
    onDecrementTap: onDecreaseQuantityTap,  
    label: Text(  
        sticker.quantity.toString(),  
        style: Theme.of(context).textTheme.displayLarge,  
    ),  
)
```

Проверяем, что управление количеством наклеек на детальном просмотре наклейки, работает.



```
% git add .  
% git commit -m 'Управление количеством наклеек'  
% git branch  
...  
* sun_8_5  
master
```

## 8.6 ПУСТАЯ КОРЗИНА

```
git checkout -b sun_8_6
```

В файле состояния (`states/sticker_state.dart`) уже создана переменная `cart` (4-ая из 6-ти).

```
List<Sticker> get cart => stickers.where((element) =>  
    element.cart).toList();
```

Напомним, что для реализации логики приложения создано 6 переменных, два ключа, 8 действий, 2 дополнительных метода. В классе `CartScreen` (`ui/screens/cart_screen.dart`) переключим список наклеек корзины на переменную из состояния `cart`.

В файле `ui/screens/cart_screen.dart` было.

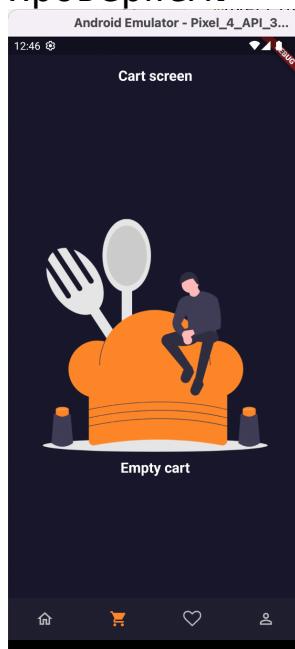
```
class CartScreenState extends State<CartScreen> {  
    final cartItems = AppData.cartItems;  
    ...
```

Подключаем.

```
class CartScreenState extends State<CartScreen> {  
    List<Sticker> get cartItems => StickerState().cart;  
    ...
```

Пока список наклеек в корзине пустой, на экране должен отображаться дизайн пустой корзины.

Проверяем.



```
% git add .
% git commit -m'Логика пустой корзины'
% git branch
...
* sun_8_6
  master
```

## 8.7 ДОБАВЛЕНИЕ НАКЛЕЙКИ В КОРЗИНУ

```
git checkout -b sun_8_7
```

Обеспечим в нашем приложении возможность обновления экрана корзины. Необходимость обновления экрана корзины возникает когда мы с экрана детального просмотра будем добавлять наклейку в корзину. Для этого, на экране корзины CartScreen (ui/screens/cart\_screen.dart) добавим метод.

```
class CartScreenState extends State<CartScreen> {
  List<Sticker> get cartItems => StickerState().cart;

  void update() {
    setState(() {});
  }
}
```

Этот метод будем вызывать через ключ.

```
 GlobalKey<CartScreenState> cartKey = GlobalKey();
```

Это первый из двух запланированных ключей. Напомним, что всего для реализации логики создано 2 ключа, 6 переменных, 8 действий, 2 вспомогательных метода.

Синтаксис вызова метода update.

```
cartKey.currentState?.update();
```

Для того, чтобы этот метод сработал нужно передать ключ cartKey в экземпляр экрана CartScreen. Экземпляр экрана CartScreen создается в классе HomeScreen (ui/screens/home\_screen.dart).

```
class HomeScreenState extends State<HomeScreen> {
    final List<Widget> screens = [
        const StickerList(),
        CartScreen(
            key: StickerState().cartKey,
        ),
    ];
}
```

После создания возможности обновления корзины можно переходить непосредственно к реализации функциональности добавления наклейки в корзину.

В классе состояния StickerState уже добавлен шаблон действия добавления наклейки в корзину (это 4-ое действие из восьми запланированных).

```
Future<void> onAddToCartTap(Sticker sticker) async {
}
```

Реализуем добавление наклейки в корзину в три шага.  
Первый шаг. Добавим в шаблон логику.

```
Future<void> onAddToCartTap(Sticker sticker) async {
    sticker.cart = true;
    cartKey.currentState?.update();
}
```

Второй шаг: Добавляем на экране ui/screens/sticker\_detail\_screen.dart метод для вызова действия onAddToCartTap на экране.

```
void onAddToCartTap() {
    StickerState().onAddToCartTap(sticker);
}
```

Третий шаг: Поключаем метод к клику на кнопку «Add to cart».

```
ElevatedButton(
    onPressed: onAddToCartTap,
    child: const Text("Add to cart"),
```

) ,

Проверяем, что добавление наклейки в корзину, работает. Для этого, открываем детальный просмотр любой наклейки, нажимаем на кнопку «Добавить в корзину», закрываем детальный просмотр и открываем корзину. Добавленная в корзину наклейка должна появиться в списке наклеек корзины.

```
% git add .
% git commit -m'Добавление наклейку в корзину'
% git branch
...
* sun_8_7
  master
```

## 8.8 СТОИМОСТЬ НАКЛЕЙКИ В СПИСКЕ НАКЛЕЕК КОРЗИНЫ.

```
git checkout -b sun_8_8
```

Реализуем стоимость наклейки в зависимости от количества. Стоимость наклейки равна количеству наклеек умноженному на цену одной наклейки.

В верстке списка наклеек корзины стоимость не реализована.

```
Text(
  "\$10",
  style: AppTextStyle.h2Style.copyWith(color:
    AppColor.accent),
)
```

В StickerState классе, для расчета стоимости наклейки создан шаблон вспомогательного метода:

```
String stickerPrice(Sticker sticker) {}
```

Реализуем логику в шаблоне.

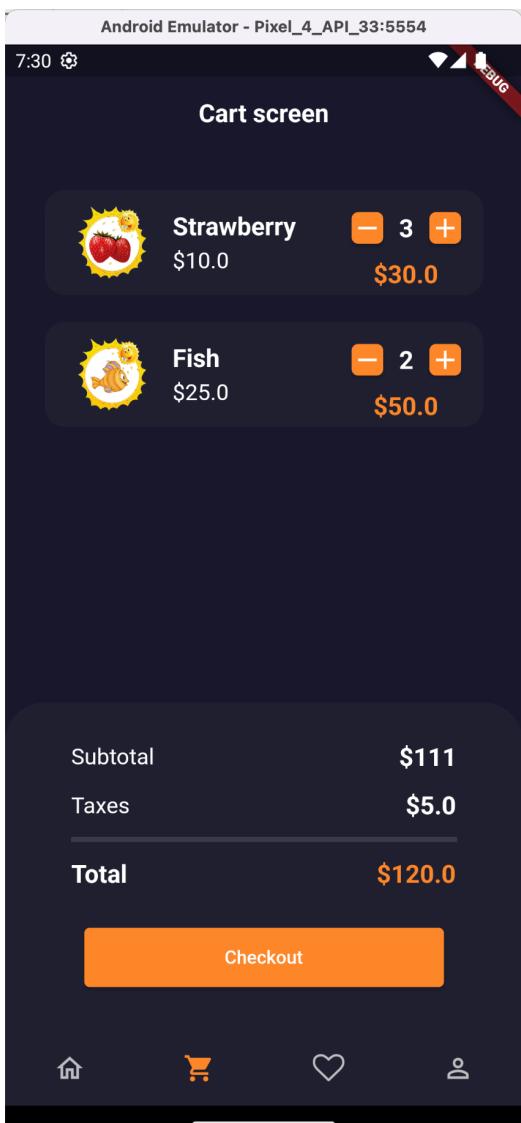
```
String stickerPrice(Sticker sticker) {
  return (sticker.quantity * sticker.price).toString();
```

}

Воспользуемся этим методом для вычисления стоимости.

```
Text( "\$\${StickerState().stickerPrice(sticker)}",
    style: AppTextStyle.h2Style.copyWith(color:
AppColor.accent),
)
```

Получаем правильную стоимость наклейки в списке корзины.



```
% git add .
% git commit -m'Стоимость наклейки в списке корзины'
% git branch
...
* sun_8_8
master
```

## 8.9 СТОИМОСТЬ КОРЗИНЫ.

```
git checkout -b sun_8_9
```

В стоимости корзины необходимо расчитать два числа:

1) Subtotal

```
Text(  
    "Subtotal",  
    style: Theme.of(context).textTheme.headlineSmall,  
)  
,  
Text(  
    "\$111",  
    style: Theme.of(context).textTheme.displayMedium,  
)  
,
```

2) Total

```
Text(  
    "Total",  
    style: Theme.of(context).textTheme.displayMedium,  
)  
,  
Text(  
    "\$120.0",  
    style: AppTextStyle.h2Style.copyWith(  
        color: AppColor.accent,  
)  
,
```

В StickerState, в блоке вспомогательных методов создан шаблон геттера subtotal.

```
double get subtotal {}
```

Реализуем вычисление стоимости наклеек из списка в корзине.

```
double get subtotal {  
    double amount = 0.0;  
    for (var element in cart) {  
        amount = amount + element.price * element.quantity;  
    }
```

```
    return amount;  
}
```

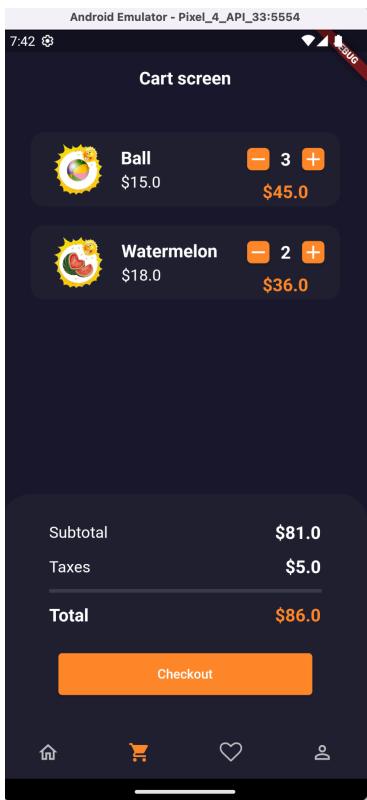
На экране корзины добавим переменную для Taxes.

```
class CartScreenState extends State<CartScreen> {  
  List<Sticker> get cartItems => StickerState().cart;  
  double taxes = 5.0;  
  ...
```

После этого вывод стоимости корзины примет вид.

```
child: Column(  
  children: [  
    Padding(  
      padding: const EdgeInsets.symmetric(horizontal:  
        20),  
      child: Row(  
        mainAxisAlignment:  
          MainAxisAlignment.spaceBetween,  
        children: [  
          Text(  
            "Subtotal",  
            style:  
              Theme.of(context).textTheme.headlineSmall,  
            ),  
          Text(  
            "\$${StickerState().subtotal.toString()",  
            style:  
              Theme.of(context).textTheme.displayMedium,  
            ),  
          ],  
        ),  
      ),  
      const SizedBox(height: 15),  
      Padding(  
        padding: const EdgeInsets.symmetric(horizontal:  
          20),  
        child: Row(  
          mainAxisAlignment:  
            MainAxisAlignment.spaceBetween,  
          children: [
```

```
        Text(
            "Taxes",
            style:
                Theme.of(context).textTheme.headlineSmall,
        ),
        Text(
            "\${taxes.toString()}",
            style:
                Theme.of(context).textTheme.displayMedium,
        ),
    ],
),
const Padding(
    padding: EdgeInsets.symmetric(horizontal: 20),
    child: Divider(thickness: 4.0, height: 30.0),
),
Padding(
    padding: const EdgeInsets.symmetric(horizontal:
        20),
    child: Row(
        mainAxisAlignment:
            MainAxisAlignment.spaceBetween,
        children: [
            Text(
                "Total",
                style:
                    Theme.of(context).textTheme.displayMedium,
            ),
            Text(
                "\${(StickerState().subtotal +
                    taxes).toString()}",
                style: AppTextStyle.h2Style.copyWith(
                    color: AppColor.accent,
                ),
            ),
        ],
),
),
```



```
% git add .
% git commit -m'Стоимость корзины'
% git branch
...
* sun_8_9
  master
```

## 8.10 ИЗМЕНЕНИЕ КОЛИЧЕСТВА В КОРЗИНЕ.

```
git checkout -b sun_8_10
```

В классе StickerState методы управления количеством уже созданы.

```
Future<void> onIncreaseQuantityTap(Sticker sticker) async
{
    sticker.quantity++;
}

Future<void> onDecreaseQuantityTap(Sticker sticker) async
{
    if (sticker.quantity == 1) return;
    sticker.quantity--;
}
```

Создадим методы на экране корзины, для вызова методов состояния и перерисовывания экрана.

```
class CartScreenState extends State<CartScreen> {
    List<Sticker> get cartItems => StickerState().cart;
    double taxes = 5.0;

    void update() {
        setState(() {});
    }

    void onIncreaseQuantityTap(Sticker sticker) async {
        await StickerState().onIncreaseQuantityTap(sticker);
        setState(() {});
    }

    void onDecreaseQuantityTap(Sticker sticker) async {
        await StickerState().onDecreaseQuantityTap(sticker);
        setState(() {});
    }

    ...
}
```

Как и на экране детального просмотра, изменение количества наклеек в корзине происходит через компонент CounterButton.

```
CounterButton(
    onIncrementTap: () {
        print('Увеличить количество');
    },
    onDecrementTap: () {
        print('Уменьшить количество');
    },
    size: const Size(24, 24),
    padding: 0,
    label: Text(
        sticker.quantity.toString(),
        style: Theme.of(context).textTheme.displayMedium,
    ),
),
```

Прикрутим эти функции к компоненту CounterButton.

```
CounterButton(
    onIncrementTap: () => onIncrementTap(sticker),
    onDecrementTap: () => onDecrementTap(sticker),
    size: const Size(24, 24),
    padding: 0,
    label: Text(
        sticker.quantity.toString(),
        style: Theme.of(context).textTheme.displayMedium,
    ),
),
```

Проверим, что управление количеством наклеек в корзине работает корректно. При увеличении или уменьшении количества перерисовывается список и пересчитывается стоимость корзины.

```
% git add .
% git commit -m'Изменение количества в корзине'
% git branch
...
* sun_8_10
master
```

## 8.11 УДАЛЕНИЕ НАКЛЕЙКИ ИЗ КОРЗИНЫ.

```
git checkout -b sun_8_11
```

К каждой карточке корзины добавлено удаление смахиванием с экрана. Для этого Container из верстки списка обернут виджетом Dismissible.

```
itemBuilder: (_, index) {
    final sticker = cartItems[index];

    return Dismissible(
        direction: DismissDirection.endToStart,
        onDismissed: (direction) {
            if (direction == DismissDirection.endToStart) {
                print('Удаляем');
            }
        },
        key: UniqueKey(),
        background: Row(...),
```

```
        child: Container(...)  
    );  
,
```

Проверим работу верстки. Из списка еды смахиванием с экрана справа налево наклейка удаляется из корзины. Стоимость корзины не пересчитывается и при повторном открытии корзины весь список на месте.

Реализуем удаление.

В классе состояния StickerState уже добавлен шаблон удаления наклейки из корзины (это 5-ое действие из восьми запланированных).

```
Future<void> onRemoveFromCartTap(Sticker sticker) async {  
}
```

Реализуем управление количеством на экране деталий в три шага.

Первый шаг. Добавим в шаблон логику.

```
Future<void> onRemoveFromCartTap(Sticker sticker) async {  
    sticker.cart = false;  
    sticker.quantity = 1;  
}
```

Второй шаг: Добавляем на экране ui/screens/cart\_screen.dart метод для вызова onRemoveFromCartTap действия на экране.

```
void onRemoveFromCartTap(Sticker sticker) async {  
    await StickerState().onRemoveFromCartTap(sticker);  
    setState(() {});  
}
```

Третий шаг: Поключаем метод в виджете Dismissible.

```
Dismissible(  
    direction: DismissDirection.endToStart,  
    onDismissed: (direction) {  
        if (direction == DismissDirection.endToStart) {
```

```
        onRemoveFromCartTap(sticker);
    }
},
key: UniqueKey(),
...

```

Проверяем, что функционал удаления наклейки из корзины, работает. При удалении всех элементов списка, корректно отображается пустая корзина.

```
% git add .
% git commit -m'Удаление наклеек из корзины'
% git branch
...
* sun_8_11
master
```

## 8.12 ЧИСТКА КОРЗИНЫ ПО КЛИКУ НА CHECKOUT.

```
git checkout -b sun_8_12
```

Реализуем очистку корзины на нажатие кнопки CheckOut.

В классе состояния StickerState уже добавлен шаблон действия на нажатие кнопки CheckOut (это 6-ое действие из восьми запланированных).

```
Future<void> onCheckOutTap() async {}
```

Реализуем в логику очистки корзины в три шага.  
Первый шаг. Добавим в шаблон логику.

```
Future<void> onCheckOutTap() async {
    for (var element in cart) {
        element.cart = false;
        element.quantity = 1;
    }
}
```

Второй шаг: Добавляем на экране ui/screens/cart\_screen.dart метод для вызова onCheckOutTap действия на экране.

```
void onCheckOutTap() async {
    await StickerState().onCheckOutTap();
    setState(() {});
}
```

Третий шаг: Прикуручиваем метод к кнопке CheckOut.

```
ElevatedButton(
    onPressed: onCheckOutTap,
    child: const Text("Checkout"),
),
...
...
```

Проверяем, что функционал отчистки корзины, работает. При удалении всех элементов списка, корректно отображается пустая корзина.

Заметим, что вся логика экрана корзины(ui/screens/cart\_screen.dart) закончена. Полный список переменных и методов, обеспечивающих логику экрана следующий.

```
class CartScreenState extends State<CartScreen> {
    List<Sticker> get cartItems => StickerState().cart;
    double taxes = 5.0;

    void update() {
        setState(() {});
    }

    void onIncreaseQuantityTap(Sticker sticker) async {
        await StickerState().onIncreaseQuantityTap(sticker);
        setState(() {});
    }

    void onDecreaseQuantityTap(Sticker sticker) async {
        await StickerState().onDecreaseQuantityTap(sticker);
        setState(() {});
    }
}
```

```

void onRemoveFromCartTap(Sticker sticker) async {
  await StickerState().onRemoveFromCartTap(sticker);
  setState(() {});
}

void onCheckOutTap() async {
  await StickerState().onCheckOutTap();
  setState(() {});
}

@Override
Widget build(BuildContext context) {
  ...
}

% git add .
% git commit -m'Очистка корзины'
% git branch
  ...
* sun_8_12
  master

```

## 8.13 ПУСТОЙ ЭКРАН ЛЮБИМЫХ НАКЛЕЕК.

`git checkout -b sun_8_13`

В файле состояния (`states/sticker_state.dart`) уже создана переменная `favorite` (5-ая из 6-ти).

```
List<Sticker> get favorite => stickers.where((element) =>
element.isFavorite).toList();
```

Напомним, что для реализации логики приложения создано 6 переменных, два ключа, 8 действий, 2 дополнительных метода. В классе `FavoriteScreen` (`ui/screens/favorite_screen.dart`) переключим список наклеек корзины на переменную из состояния `favorite`.

В файле `ui/screens/favorite_screen.dart` было.

```
class FavoriteScreenState extends State<FavoriteScreen> {  
    final favoriteItems = AppData.favoriteItems;  
    ...
```

Подключаем.

```
class FavoriteScreenState extends State<FavoriteScreen> {  
    List<Sticker> get favoriteItems =>  
        StickerState().favorite;  
    ...
```

Пока список любимых наклеек, на экране должен отображаться дизайн пустого экрана любимых наклеек.

Проверяем.

```
% git add .  
% git commit -m'Пустой экран любимых наклеек'  
% git branch  
...  
* sun_8_13  
  master
```

## 8.14 ДОБАВЛЕНИЕ НАКЛЕЙКИ В ЛЮБИМЫЕ НАКЛЕЙКИ

```
git checkout -b sun_8_14
```

В пункте 8.7 для экрана корзины была реализована возможность обновления экрана корзины. Необходимость обновления экрана корзины возникла, когда мы с экрана детального просмотра добавляем наклейку в корзину. Аналогично возможности обновления экрана корзины нам нужна возможность обновления экрана любимых настроек. Необходимость обновления экрана любимых настроек возникает когда мы с экрана детального просмотра будем добавлять наклейку в любимые или удалять наклейку из любимых кликая на сердечко.

Обеспечим в нашем приложении возможность обновления экрана корзины.

Для этого, на экране любимых наклеек FavoriteScreen (ui/screens/favorite\_screen.dart) добавим метод.

```
class FavoriteScreenState extends State<FavoriteScreen> {  
    List<Sticker> get favoriteItems =>  
    StickerState().favorite;  
  
    void update() {  
        setState(() {});  
    }  
    ...
```

Этот метод будем вызывать через ключ.

```
 GlobalKey<FavoriteScreenState> favoriteKey = GlobalKey();
```

Это второй из двух запланированных ключей. Напомним, что всего для реализации логики создано 2 ключа, 6 переменных, 8 действий, 2 вспомогательных метода.

Синтаксис вызова метода update.

```
favoriteKey.currentState?.update();
```

Для того, чтобы этот метод сработал нужно передать ключ favoriteKey в экземпляр экрана FavoriteScreen. Экземпляр экрана FavoriteScreen создается в классе HomeScreen (ui/screens/home\_screen.dart).

```
class HomeScreenState extends State<HomeScreen> {  
    final List<Widget> screens = [  
        const StickerList(),  
        CartScreen(  
            key: StickerState().cartKey,  
        ),  
        FavoriteScreen(  
            key: StickerState().favoriteKey,  
        ),  
        const ProfileScreen()  
    ];  
    ...
```

После создания возможности обновления экрана любимых наклеек можно переходить непосредственно к реализации функциональности добавления наклейки в избранные.

В классе состояния StickerState уже добавлен шаблон действия на нажатие иконки сердечка (это 7-ое действие из восьми запланированных).

```
Future<void> onAddRemoveFavoriteTap(Sticker sticker) async  
{}
```

Реализуем логику упорядочения любимыми наклейками в три шага.

Первый шаг. Добавим в шаблон логику.

```
Future<void> onAddRemoveFavoriteTap(Sticker sticker)  
async {  
    sticker.isFavorite = !sticker.isFavorite;  
    favoriteKey.currentState?.update();  
}
```

Второй шаг: Добавляем на экране ui/screens/sticker\_detail\_screen.dart метод для вызова действия onAddRemoveFavoriteTap на экране.

```
void onAddRemoveFavoriteTap() async {  
    await StickerState().onAddRemoveFavoriteTap(sticker);  
    setState(() {});  
}
```

Третий шаг: Подключим этот метод к кнопке.

...

```
@override  
Widget build(BuildContext context) {  
    return Scaffold(  
        appBar: _appBar(context),  
        body: Center(child: Image.asset(sticker.image,  
                                         scale: 2)),  
        floatingActionButton: _floatingActionButton(),
```

```

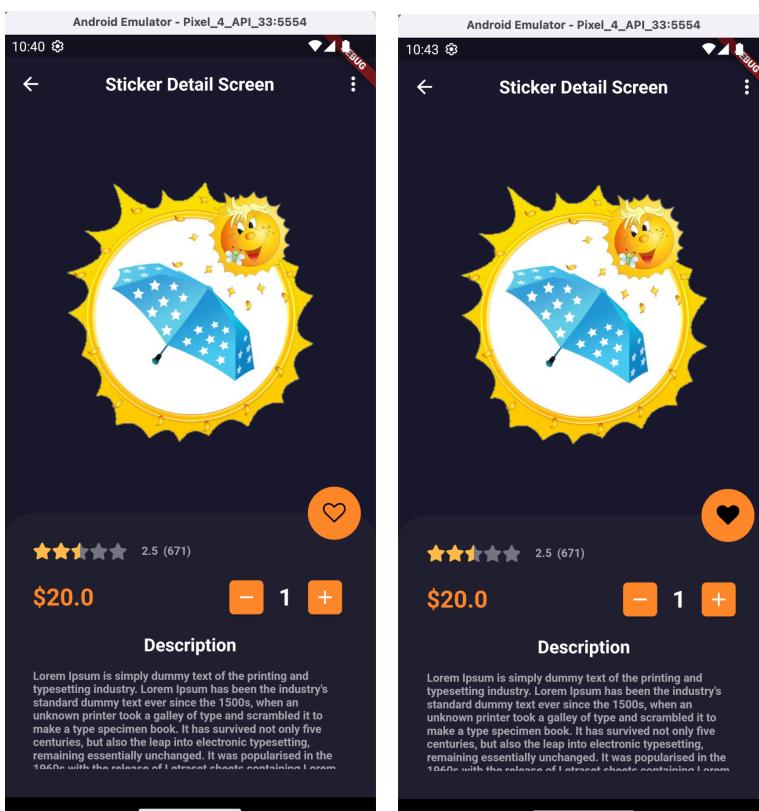
floatingActionButtonLocation:
    FloatingActionButtonLocation.endDocked,
bottomNavigationBar: _bottomAppBar(),
);
}

Widget _floatingActionButton() {
    return FloatingActionButton(
        elevation: 0.0,
        backgroundColor: AppColor.accent,
        onPressed: onAddRemoveFavoriteTap,
        child: sticker.isFavorite
            ? const Icon(AppIcon.heart)
            : const Icon(AppIcon.outlinedHeart),
    );
}

...

```

Проверим, что переключение иконки работает.



Проверим, что экран любимых наклеек обновляется после добавления продукта в любимые наклейки. Для этого добавляем продукт в любимые наклейки, переходим на вкладку любимых наклеек и проверяем, что экран любимых наклеек уже не пустой.

```
% git add .
% git commit -m'Добавление наклеек в любимые наклейки'
% git branch
...
* sun_8_14
  master
```

## 8.15 ПЕРЕКЛЮЧЕНИЕ ТЕМЫ С VALUENOTIFIER.

```
git checkout -b sun_8_15
```

Для переключения темы используем последнюю из шести переменных в StickerState (states/sticker\_state.dart).

```
ValueNotifier<bool> isLight = ValueNotifier(true);
```

И реализуем шаблон последнего действия

```
void toggleTheme() {
    isLigth.value = !isLigth.value;
}
```

Для перестройки виджета, по изменению значения isLight, виджет оборачивается в ValueListenableBuilder. Виджет, который будем оборачивать это MaterialApp.

В файле main.dart обернем MaterialApp.

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

    // This widget is the root of your application.
    @override
    Widget build(BuildContext context) {
        return ValueListenableBuilder(
```

```
        valueListenable: StickerState().isLight,
        builder: (_, isLight, __) {
            return MaterialApp(
                title: 'Sunny Stickers',
                theme: isLight
                    ? AppTheme.lightTheme
                    : AppTheme.darkTheme,
                home: const HomeScreen(),
            );
        },
    );
}
```

Теперь осталось только вызвать метод изменения значение переменной `isLight` в клике на кубики (левый верхний угол первого экрана `ui/screens/sticker_list_screen.dart`).

```
PreferredSizeWidget _appBar(BuildContext context) {
    return AppBar(
        leading: IconButton(
            icon: const FaIcon(FontAwesomeIcons.dice),
            onPressed: StickerState().toggleTheme,
        ),
        ...
    );
}

% git add .
% git commit -m'Переключение темы'
% git branch
...
* sun_8_17
  master
```