Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Томский государственный университет систем управления и радиоэлектроники (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

**«Разработка мобильного приложения для интернет-магазина linza.tomsk.ru»**

Отчет о выполнении этапа II группового проектного обучения (ГПО)

Проект ГПО АОИ-2106

Ответственный исполнитель проекта студент гр. 429-3

Д.Д. Агафонов

« » 20\_\_

Проверил:

Руководители проекта

заведующий каф. АОИ, ТУСУР, к.э.н.

А.А. Сидоров



оценка

« » 20\_\_

Младший научный сотрудник

И.Д. Тикшаев

« » 20\_\_

Принял:

Ответственный за ГПО на кафедре

ст. преподаватель каф. АОИ

М.А. Шишанина

« » 20\_\_

Томск 2022

Исполнители проекта ГПО АОИ-2106:

Студент гр. 429-3 Д.Д. Агафонов

Студент гр. 429-1 Д.А. Казначеева

Студент гр. 429-2 И.В. Коротков

Студент гр. 429-2 А.Н. Кузнецов

Студент гр. 429-1 С.А. Лузин

Студент гр. 429-3 Д.В. Хазов

Студент гр. 429-2 Б.Ф. Шабаев

**Оглавление**

[Введение 4](#_gjdgxs)

[1 ЗАДАЧИ НА СЕМЕСТР 5](#_30j0zll)

[2 ХОД РАБОТЫ 6](#_1fob9te)

[2.1 Управление проектом 6](#_3znysh7)

[2.2 Выбор платежного сервиса 8](#_2et92p0)

[2.3 Доработка дизайна приложения 9](#_tyjcwt)

[2.4 Front-end 14](#_3dy6vkm)

[2.4.1 Внедрение архитектурного паттерна flutter\_bloc 14](#_1t3h5sf)

[2.4.2 Разработка 14](#_4d34og8)

[2.5 Back-end 28](#_2s8eyo1)

[2.5.1 Разработка административной части 28](#_3rdcrjn)

[2.5.2 Разработка API 30](#_1ksv4uv)

[2.5.3 Размещение сервера на хостинг 34](#_44sinio)

[2.6 Тестирование 36](#_2jxsxqh)

[Заключение 40](#_z337ya)

[Список литературы 41](#_3j2qqm3)

# Введение

Большое стремление каждого человека достигнуть максимального комфорта в каждой из сфер жизни, затронуло и международную паутину Интернет. Пользователь, желая всегда оставаться в сети, использует в качестве коммуникатора телефон.

Это обусловило появление мобильного Интернета. При пребывании вне дома, либо во время путешествий и командировок можно вместо ноутбука подключаться с помощью планшета, либо аналогичного вида техники.

Все основные приложения делятся на те, которые необходимы для приятного времяпровождения, и те, которые используются исключительно в рабочих целях. Создание мобильных приложений второго типа более распространено. Продукты деятельности плотно вошли в такие жизненные направления как медицина, государственные организации и даже производственные компании. Развлекательные утилиты могут отыграть роль инструментов маркетинга для большинства предприятия, но даже это не позволяет им составить конкуренцию по сфере применения деловому направлению.

Спрос на подобные приложения для мобильных устройств стабильно растет уже несколько последних лет. Напрашивается вывод, что на сегодняшний день актуальность разработки таких приложений вполне целесообразна и обязательно получит соответствующее признание пользователей.

Мобильный разработчик — это программист, который пишет приложения для мобильных устройств. Большинство носимых устройств работает под управлением Android или iOS. Поэтому мобильные разработчики пишут софт для этих операционных систем.

# 1 ЗАДАЧИ НА СЕМЕСТР

Агафонов Д.Д. – оценка возможностей перевода команды на новый метод работы. Изменение структуры доски задач и ее ведение: создание задач для команды; курирование и контроль исполнения в облачной программе для управления проектами «Trello». Организация коммуникаций в команде. Написание научной статьи на тему «Мобильное приложение как средство продажи товаров». Тестирование API с помощью Postman.

Казначеева Д.А. – теоретическая подготовка по вариантам способов оплаты онлайн-покупок и онлайн-сервисов в период кризис-этапа в 2022 году с целью выбора лучшего для внедрения в приложение и написание статьи по данной теме.

Коротков И.В. – написание статьи с требованиями к дизайну для продвижения мобильного приложения онлайн-магазина. Доработка дизайна интерфейса приложения по предложенным в статье требованиям.

Кузнецов А.Н. – переход к архитектурному паттерну BLoC (реализация Cubit), front-end разработка, реализация страниц согласно измененному дизайну интерфейса приложения.

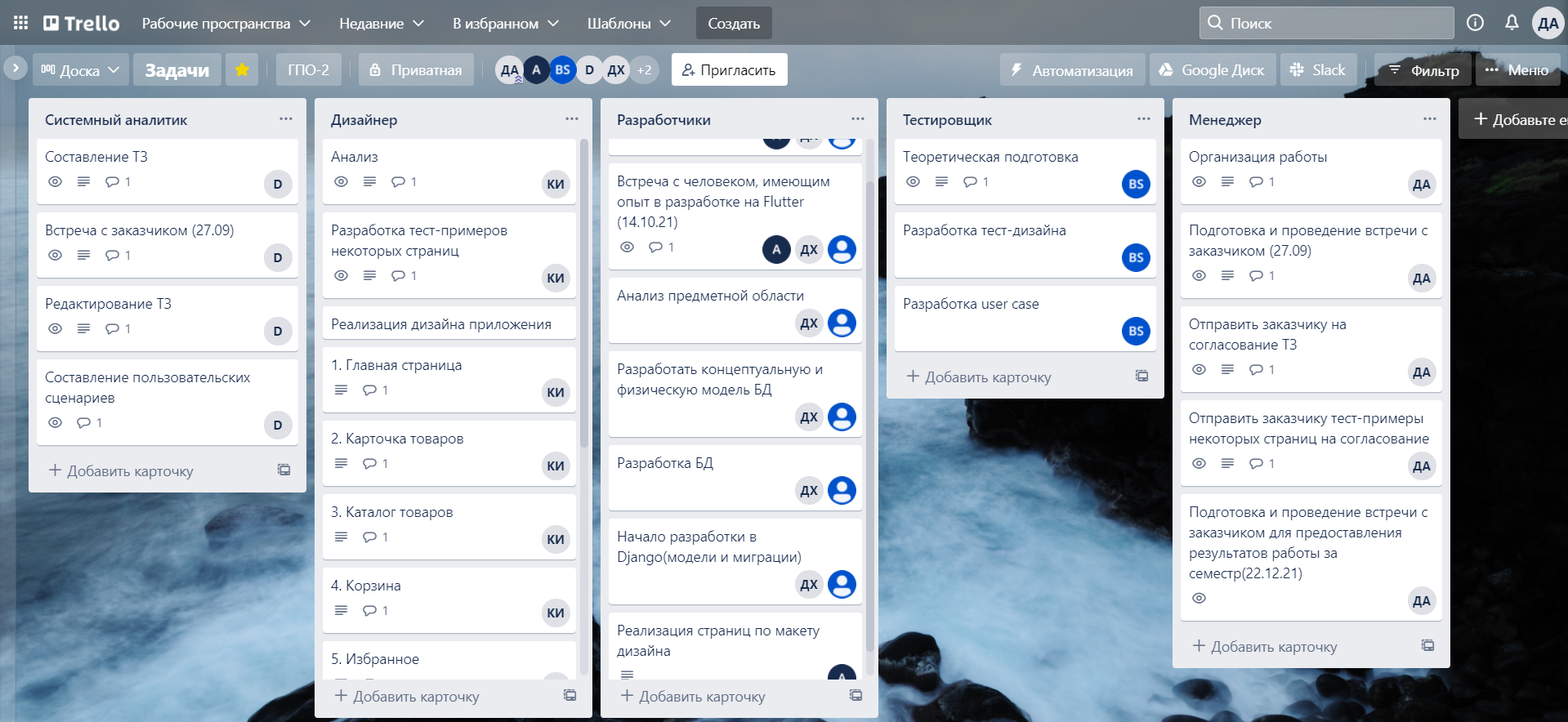
Лузин С.А. и Хазов Д.В. – разработка интерфейса программирования приложений (API), админ панели и размещение на удаленный хостинг.

Шабаев Б.Ф – написание статьи по теме «Мобильное приложение как средство продажи товаров». Составление плана описываемых проверок, разработка стратегии тестирования и тестирование документации. Тестирование прототипа проекта.

# 2 ХОД РАБОТЫ

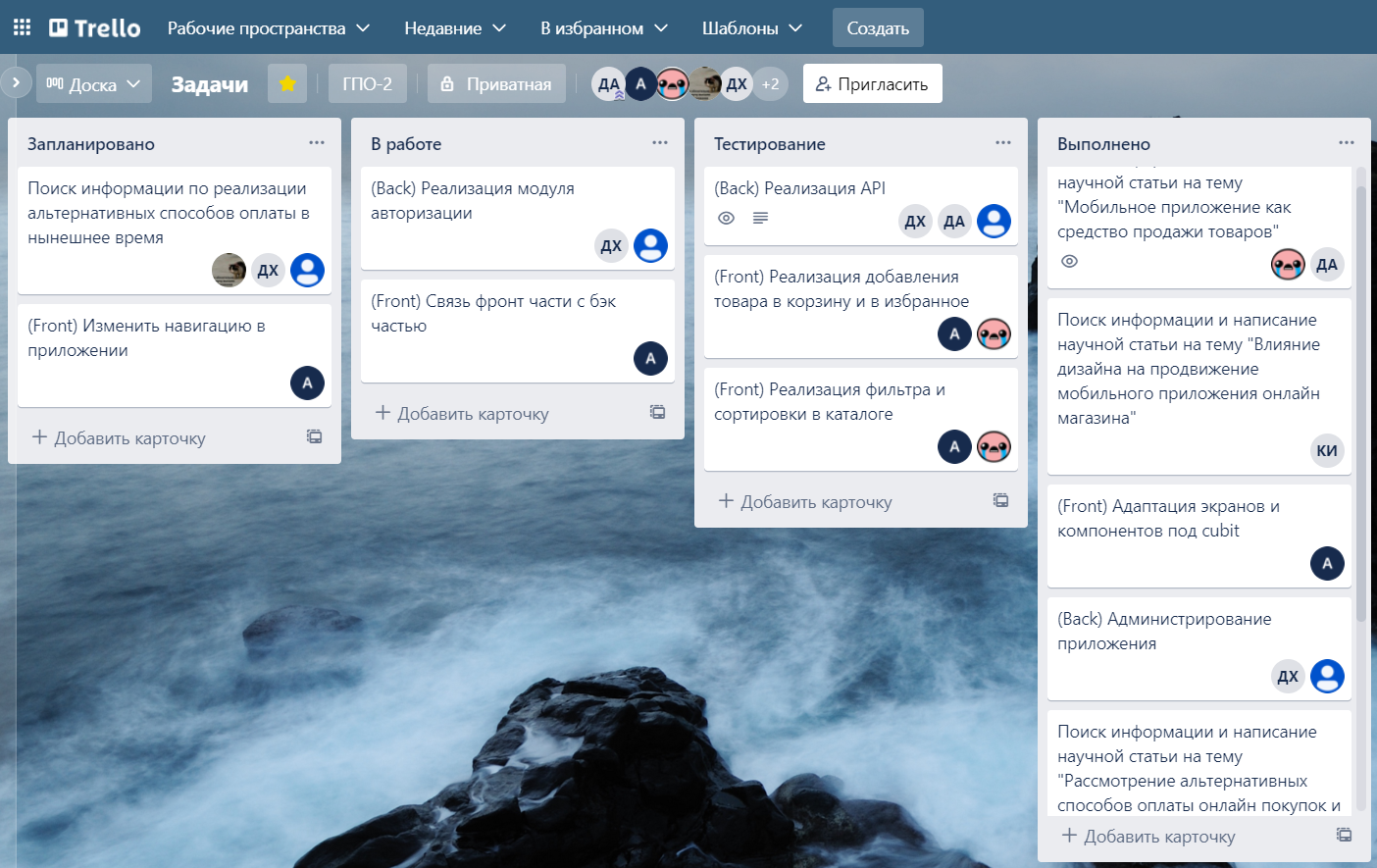
## 2.1 Управление проектом

Ведение задач для предстоящей работы команды было решено продолжить в Trello — облачная программа для управления проектами небольших групп (рисунок 2.1).

   
Рисунок 2.1 – Доска задач в «Trello» до изменений

Однако её реализация на прошлом этапе была неправильной и неудобной. Поэтому появилась задача изменить прошлую структуру доски на более правильную и удобную. В прошлой структуре столбцы имели наименования роли участника команды, а карточки отображали их задачи. Задачи добавлялись сверху вниз, где самая нижняя - текущая. По мере выполнения карточки человек, работающий по данной задаче, должен был пометить комментарием о том, что задача закрыта.

Trello использует парадигму для управления проектами, известную как Kanban. На стандартной Kanban-доске процесс состоит из трех шагов: «Запланировано», «В работе» и «Выполнено». Однако доску можно настроить в соответствии с процессом, принятым в той или иной команде, в зависимости от ее размеров, структуры и целей. В нашем случае добавлен дополнительный шаг «Тестирование», для проверки выполнения данной задачи. Имена столбцов меняются в зависимости от проекта, но важно сохранять их последовательность ― это ключевая ценность Kanban, которую называют потоком. Kanban-карточки ― это задачи, которые движутся по потоку и перетекают в другие столбцы в зависимости от их состояния. На доске отражаются все процессы. В Kanban это называется управлением потоком. Также к каждой карточке прикрепляется её исполнитель (рисунок 2.2)

  
Рисунок 2.2 – Доска задач в «Trello» после изменений

В дальнейшем планируется потренировать навыки работы с данной доской, объяснить принципы работы с помощью игр-симуляторов «FutureBan» и «getKanban». С ними проще усвоить основные принципы Kanban-досок и научиться с ними работать.

Для примера приведено описание игры «FutureBan». Команде необходимо подготовить праздничный стол. Сначала участники составляют список блюд, записывая их на стикерах (одно блюдо на стикер). Нужно придумать не менее 20-25 блюд. Затем они приступают к приготовлению блюд, передвигая стикеры по стадиям на Kanban-доске. Стикеры передвигаются по следующим правилам. Каждый член команды получает монетку. Для того, чтобы осуществить ход, ему нужно ее подбросить. Если выпадает «орел», то стикер блокируется. Если выпадает решка, то он может передвинуть стикер по доске или разблокировать ранее заблокированный стикер.

## 2.2 Выбор платежного сервиса

В связи с ограничением работы многих зарубежных платежных сервисов, перед аналитиком легла задача изучения российских аналогов. Для разрабатываемого приложения требовалось выбрать наиболее удобный для пользователей и простой для внедрения сервис оплаты. Дополнительным затруднением выступили ограничения работы таких платежных систем как Visa, Mastercard, а также чуть менее популярных среди российского населения - Western Union и PayPal.

Изучив заявления зарубежных компаний, стало ясно, что основные функции - снятие наличных средств, осуществление переводов по номеру карты, возможность оплачивать покупки как в простых магазинах, так и в онлайн-магазинах России - останутся неизменными. Стоит обратить внимание, что актуальной в нынешней ситуации становится система «Мир», которая была целенаправленно создана для устойчивой работы в период кризиса и санкций.

Дальше требовалось изучить российские платежные сервисы и последующем выбором лучшего. Хорошей Российской альтернативой зарубежным платежным сервисам является «ЮKassa» ― сервис по приёму и обработке транзакций через интернет. Данный сервис сотрудничает с организациями, которые зарегистрированы в России и за рубежом. Транзакции принимаются через ЮKassa более чем 20 вариантами: со счета банка, с помощью электронных кошельков, через мобильные приложения и приложения Сбербанка, Альфа-Банка и Тинькофф Банка, через мобильные номера.

Для сервиса Юkassa серьезным конкурентом является Robokassa, которая агрегирует в себе 26 платежных систем, включая банковские карты, электронные деньги, терминалы, карты рассрочки и другие методы оплаты. Робокассой пользуется более 30 000 компаний по всей России, и число клиентов растет ежедневно. Но, в отличие от аналогичного сервиса Юkassa, у Робокассы нет собственного мобильного приложения, что для некоторых пользователей может показаться большим минусом.

После подробного изучения сервисов Юkassa и Robokassa, был сделан выбор в пользу первого платежного сервиса. Одним из ключевых его преимуществ над конкурентом стало то, что для разработчиков Юkassa предлагает единый API для обработки транзакций, отправки чеков, проведения возвратов и других операций. Для интеграции предлагается использовать готовые библиотеки для серверного взаимодействия и для встраивания платежных форм на сайт и в мобильное приложение. Среди них: серверные SDK (официальные библиотеки PHP и Python, библиотеки Node.js и .NET), Мобильные SDK (iOS SDK, Android SDK), Веб SDK (Виджет ЮKassa, Checkout.js).

В период кризиса ЮKassa продолжает работу без каких-либо изменений. Платежи принимаются через сервис в российских интернет-магазинах без изменений.

## 2.3 Доработка дизайна приложения

Дизайн напрямую влияет на юзабилити приложения, поэтому к нему были определены требования, которые повысят удобство использования приложения, а также увеличат популярность проекта. В ходе работы над проектом было решено изучить информацию, которая может помочь в продвижении приложения с помощью его дизайна. На основе изученного материала, была написана научная статья, содержащая в себе описание некоторых характеристик дизайна пользовательского интерфейса приложения. Также в статье были предложены требования к дизайну приложения онлайн-магазина для роста популярности проекта.

Ниже кратко перечислены требования для повышения популярности приложения онлайн-магазина, описанные в написанной научной статье:

1. все элементы дизайна в приложении должны подчиняться единым стилистическим нормам;
2. приложение должно иметь минимальное количество рекламных блоков;
3. дизайн не должен быть перенасыщен необычными элементами, следует делать его простым и понятным;
4. все элементы, предназначенные для навигации, должны выделяться на фоне приложения;
5. цвета текста и фона должны быть подобраны так, чтобы это не затрудняло восприятие информации;
6. стиль приложения должен быть уникальным;
7. приложение не должно содержать лишних функций;
8. дизайн должен снижать время на рутинные операции до минимума;
9. приложение должно иметь простую и лаконичную иконку.

Изменения дизайна интерфейса приложения начались с карточки товара (рисунок 2.3). В них были изменены кнопки для добавления продукта в корзину или в избранное. Для них был выбран единый стиль и цвет, что поможет пользователю быстро разобраться в функционале этих кнопок.



Рисунок 2.3 – Измененный стиль кнопок в карточке товара

На странице «Каталог» некоторые элементы управления были переделаны под общий стиль приложения (рисунок 2.4). Эти изменения существенно повышают читаемость приложения для пользователя, что служит плюсом в продвижении проекта.



Рисунок 2.4 – Каталог товаров

При нажатии на товар, открывается карточка товара (рисунок 2.5). Здесь были изменены цвета кнопок для добавления товара в корзину или избранное, была добавлена кнопка, возвращающая предыдущий экран.

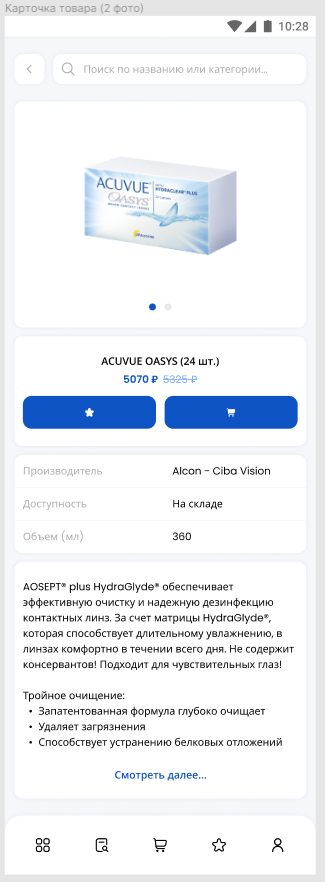


Рисунок 2.5 – Карточка товара

После перехода пользователя на страницу личного кабинета, ему необходимо авторизоваться или зарегистрироваться. Поэтому в приложении открывается форма авторизации (рисунок 2.6). Здесь были изменены кнопки для поддержания общего уникального стиля приложения.

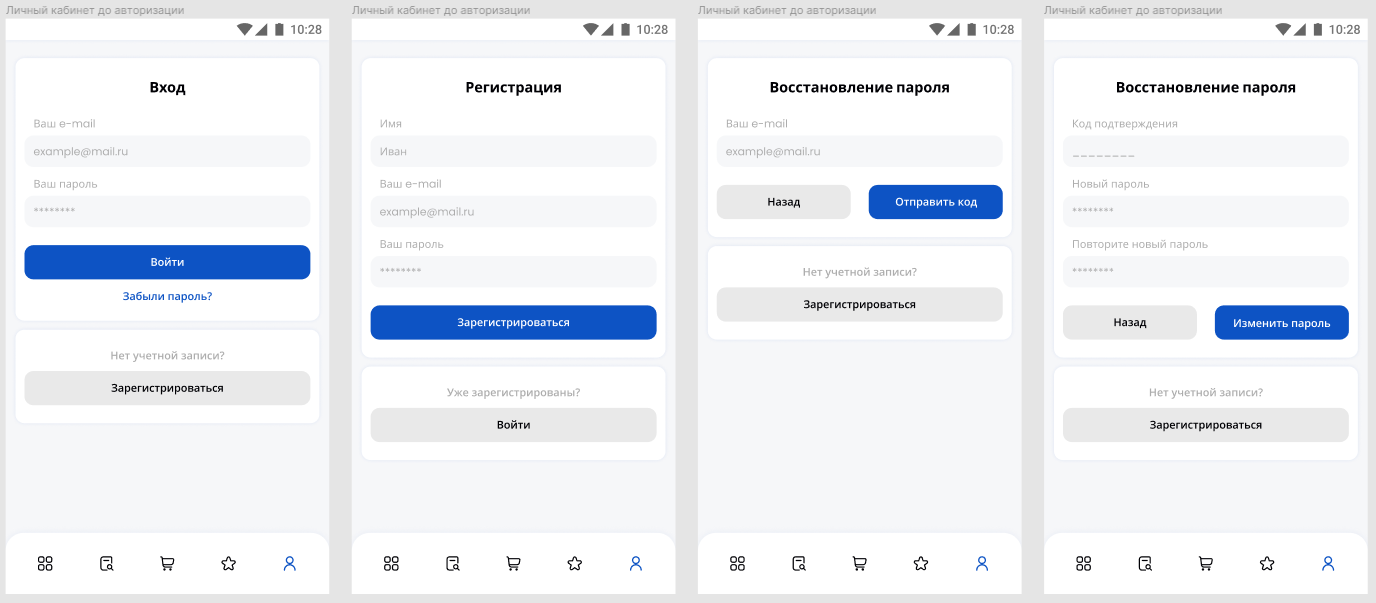


Рисунок 2.6 – Форма авторизации

Все требования из написанной научной статьи были соблюдены в дизайне проекта мобильного приложения онлайн-магазина iClinic. Этот подход должен повлечь за собой повышение потока пользователей, заинтересовавшихся развивающимся проектом.

## 2.4 Front-end

### 2.4.1 Внедрение архитектурного паттерна flutter\_bloc

Библиотека flutter\_bloc является менеджером состояний приложения. С помощью этой библиотеки можно:

* точно понимать, в каком состоянии находится приложение в любой момент времени;
* легко тестировать;
* повторно использовать компоненты;
* разрабатывать быстрые и реактивные приложения.

У библиотеки flutter\_bloc есть две реализации: Bloc и Cubit. В нашем проекте используется Cubit.

Отличие в работе cubit и bloc продемонстрировано на слайде.

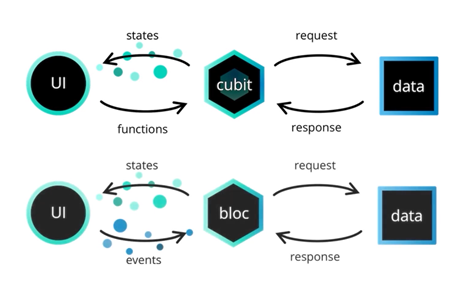


Рисунок 2.7 – Демонстрация работы cubit и bloc

### 2.4.2 Разработка

Внедрен Cubit на загрузочный экран, теперь экран имеет несколько состояний, а главные из них это состояние загрузки приложения и состояние, когда приложение готово к работе.

Код состояний загрузочного экрана:

abstract class SplashScreenState {}

class SplashScreenLoadingState extends SplashScreenState {}

class SplashScreenLoadedState extends SplashScreenState {

final List<Product> products;

SplashScreenLoadedState({required this.products});

}

class SplashScreenErrorState extends SplashScreenState {

final String errorMsg;

SplashScreenErrorState({required this.errorMsg});

}

Во время загрузки экрана выполняется функция cubit, которая подгружает данные с сервера, и когда данные будут получены, то откроется главная страница приложения.

Код функции cubit:

class SplashScreenCubit extends Cubit<SplashScreenCubit> {

SplashScreenCubit() : super(SplashScreenLoadingState());

Future<void> getProducts() async {

try {

Map<String, dynamic> response =

await Api.get(method: Constants.METHOD\_GET);

List<Products> convertData = (response['products'] as List)

.map((product) => Product.fromJson(product))

.toList();

emit(SplashScreenLoadedState(products: convertData));

} on APIException catch (error) {

if (error.code == 0) {

emit(SplashScreenErrorState(errorMsg: 'No internet connection'));

} else {

emit(SplashScreenErrorState(errorMsg: 'Sorry, unknown error'));

}

}

}

}

Чтобы пользоваться функциями cubit необходимо обернуть нужный виджет/экран в BlocProvider, который даст доступ к указанному cubit.

Код SplashScreenProvider:

class SplashScreenProvider extends StatelessWidget {

static String routeName = "/splash";

const SplashScreenProvider({Key? key}) : super(key: key);

@override

Widget build(BuildContext context) {

return BlocProvider<SplashScreenCubit>(

create: (context) => SplashScreenCubit(),

child: SplashScreen(),

);

}

}

Таким же образом cubit внедрялся в остальные виджеты и экраны, чтобы разделять код, отвечающий за бизнес-логику и графический интерфейс.

После внедрения cubit, архитектура виджета/экрана становится как на рисунке 2.8.

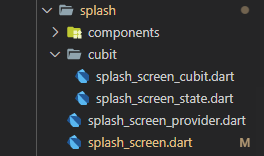


Рисунок 2.8 – Архитектура экрана загрузки

В карточку продукта добавлен параметр, в который необходимо передавать виджет кнопки, это сделано для того, чтобы можно было добавлять кнопку, которая необходима в данной ситуации.

Код карточки продукта:

class ProductCard extends StatelessWidget {

ProductCard({

Key? key,

required this.buttonUp,

required this.product,

}) : super(key: key);

final Product product;

final Widget buttonUp;

@override

Widget build(BuildContext context) {

return Padding(

padding: EdgeInsets.only(left: 10),

child: Container(

height: 217.h,

width: 164.w,

decoration: BoxDecoration(

color: kWhiteColor,

borderRadius: BorderRadius.circular(10),

boxShadow: [

BoxShadow(

color: kBlueColor.withOpacity(0.1),

blurRadius: 5,

),

],

),

child: Column(

children: [

Stack(

alignment: AlignmentDirectional.topEnd,

children: [

ProductImage(product: product),

Column(

children: [

Container(

padding: EdgeInsets.only(top: 10.h, right: 10.w),

child: Column(

children: [

buttonUp,

SizedBox(

height: 5.h,

),

ButtonAddCartProvider(product: product),

],

),

),

],

),

],

),

ProductTitle(product: product),

ProductPrice(product: product),

],

),

),

);

}

}

Например, как показано на рисунке 2.9 и 2.10, на главной странице у карточки товара в правом верхнем углу находится кнопка “Добавить в избранное”, а на странице избранных товаров передается совсем другая кнопка, которая удаляет продукт из списка избранных товаров.



Рисунок 2.9 – Карточка продукта с кнопкой “Добавить в избранное”



Рисунок 2.10 – Карточка продукта с кнопкой “Удалить продукт”

Реализовано отображение товаров в каталоге в виде сетки с прокруткой вниз. Экран каталога представлен на рисунке 2.11.

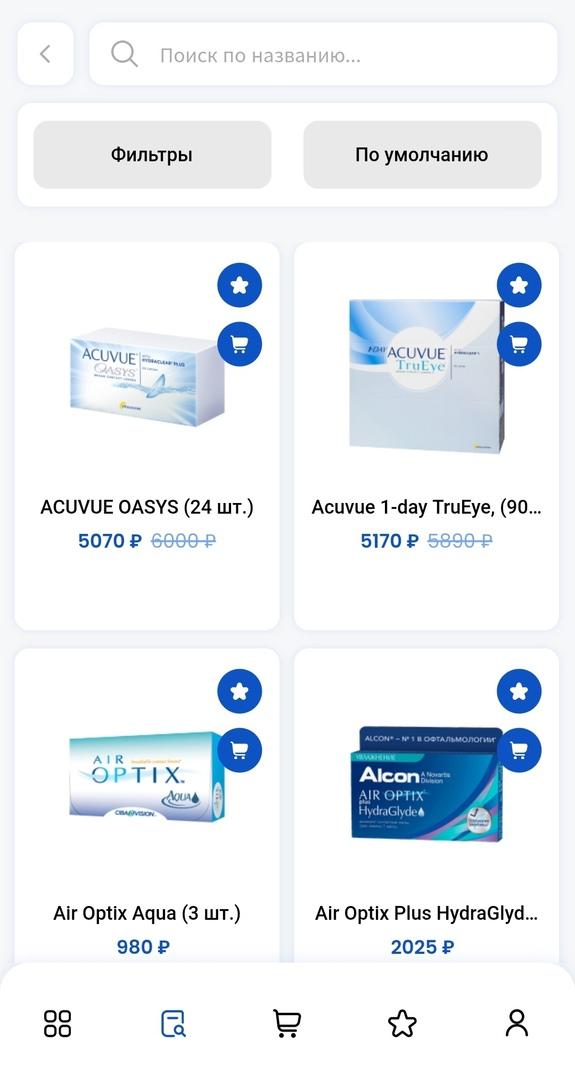


Рисунок 2.11 – Экран каталога

После нажатия на кнопку “Фильтры” или на рядом находящуюся кнопку сортировки, появляется соответствующее диалоговое окно, продемонстрированное на рисунках 2.12 и 2.13.

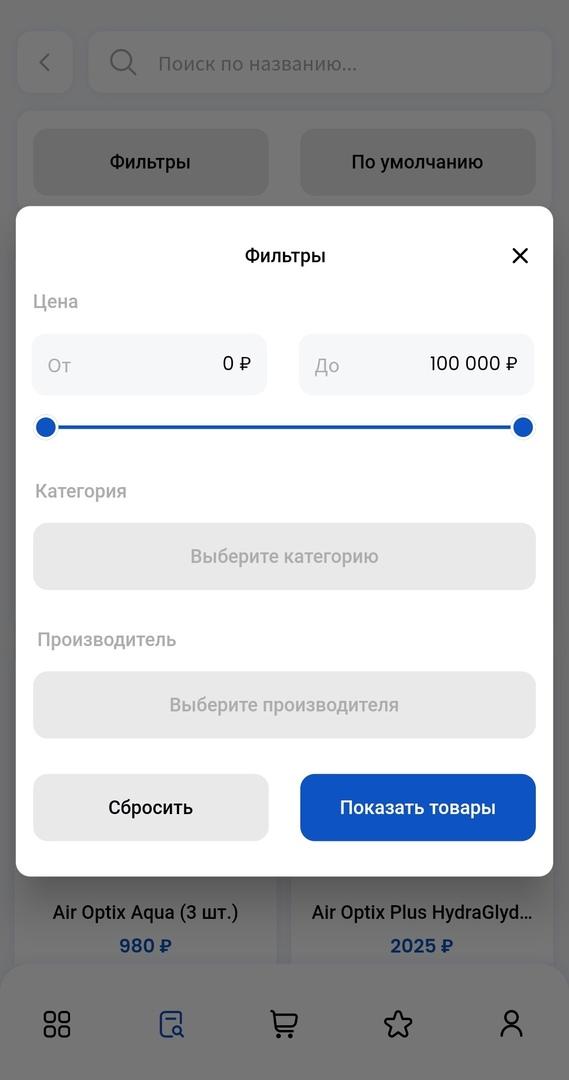


Рисунок 2.12 – Диалоговое окно фильтров

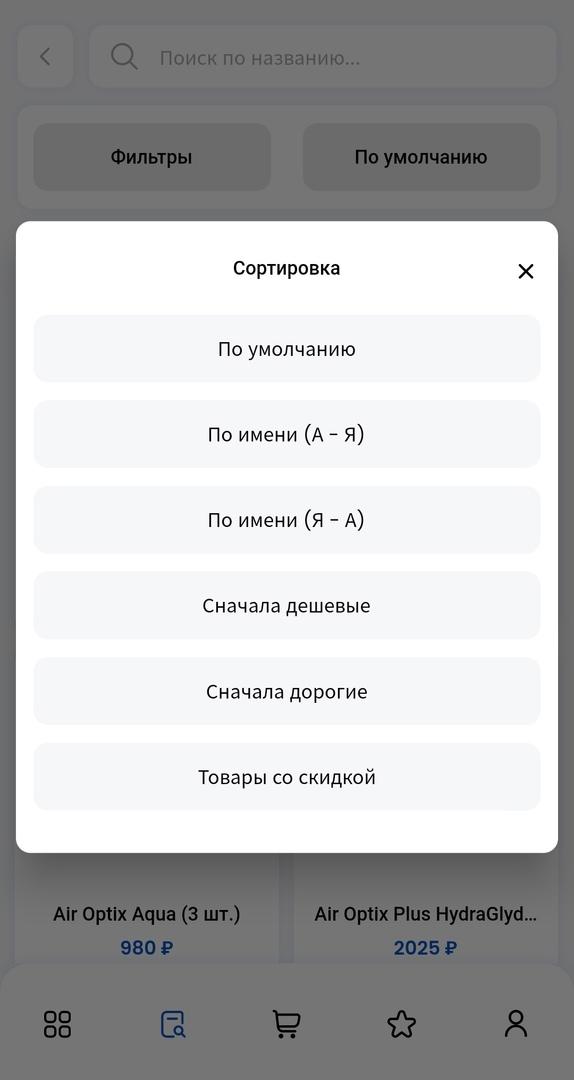


Рисунок 2.13 – Диалоговое окно сортировки

Код экрана каталога:

class BodyDetailsCatalog extends StatelessWidget {

const BodyDetailsCatalog({Key? key, required this.category\_name})

: super(key: key);

final String category\_name;

@override

Widget build(BuildContext context) {

return SafeArea(

child: SingleChildScrollView(

child: Column(

children: [

Row(

children: [

Padding(

padding: EdgeInsets.only(left: 11.w, top: 20.h, right: 10.w),

child: ButtonBack(),

),

Padding(

padding: EdgeInsets.only(top: 20.h, right: 11.w),

child: Header(

width: 293,

),

),

],

),

Padding(

padding: EdgeInsets.only(left: 11.w, right: 11.w, top: 10.h),

child: ButtonsFilterSort(),

),

Padding(

padding: EdgeInsets.only(top: 20.h),

child: GridView.builder(

padding: EdgeInsets.only(right: 10.w, bottom: 20.h),

shrinkWrap: true,

physics: ScrollPhysics(),

itemCount: categoryProducts.length,

gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

mainAxisSpacing: 10.h,

mainAxisExtent: 217.h,

crossAxisCount: 2,

),

itemBuilder: (context, index) {

return GestureDetector(

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => DetailsProductScreen(

product: categoryProducts[index])));

},

child: ProductCard(product: categoryProducts[index], buttonUp: ButtonFavouriteProvider(product: categoryProducts[index])),

);

}),

),

],

)),

);

}

}

Реализована функция добавления товаров в список избранных. После нажатия на кнопку “Добавить в избранное” на карточке продукта, добавленный элемент будет отображаться на экране избранных товаров, как показано на рисунке 2.14.

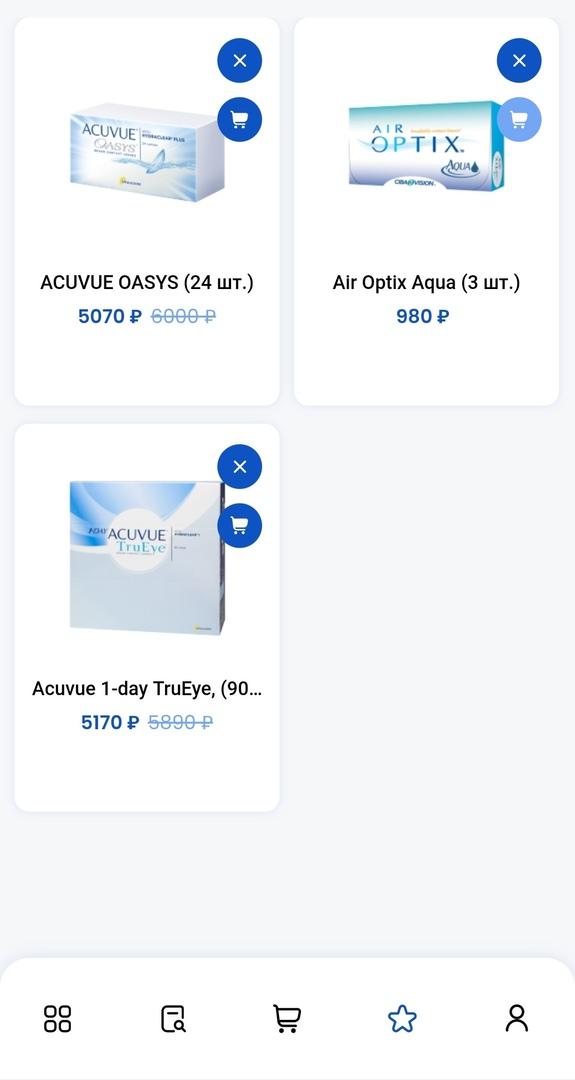


Рисунок 2.14 – Экран избранных товаров

Также была реализована функция добавления товаров в корзину. После нажатия на кнопку “Добавить в корзину” на карточке продукта, появляется диалоговое окно для выбора опций, продемонстрированное на рисунке 2.15. После нажатия, на этом диалоговом окне, кнопки “Добавить товар в корзину”, появляется оповещение в виде снэк-бара с сообщением – “Товар был успешно добавлен в корзину”, как показано на рисунке 2.16. Добавленный продукт будет отображаться на экране корзины, как показано на рисунке 2.17.

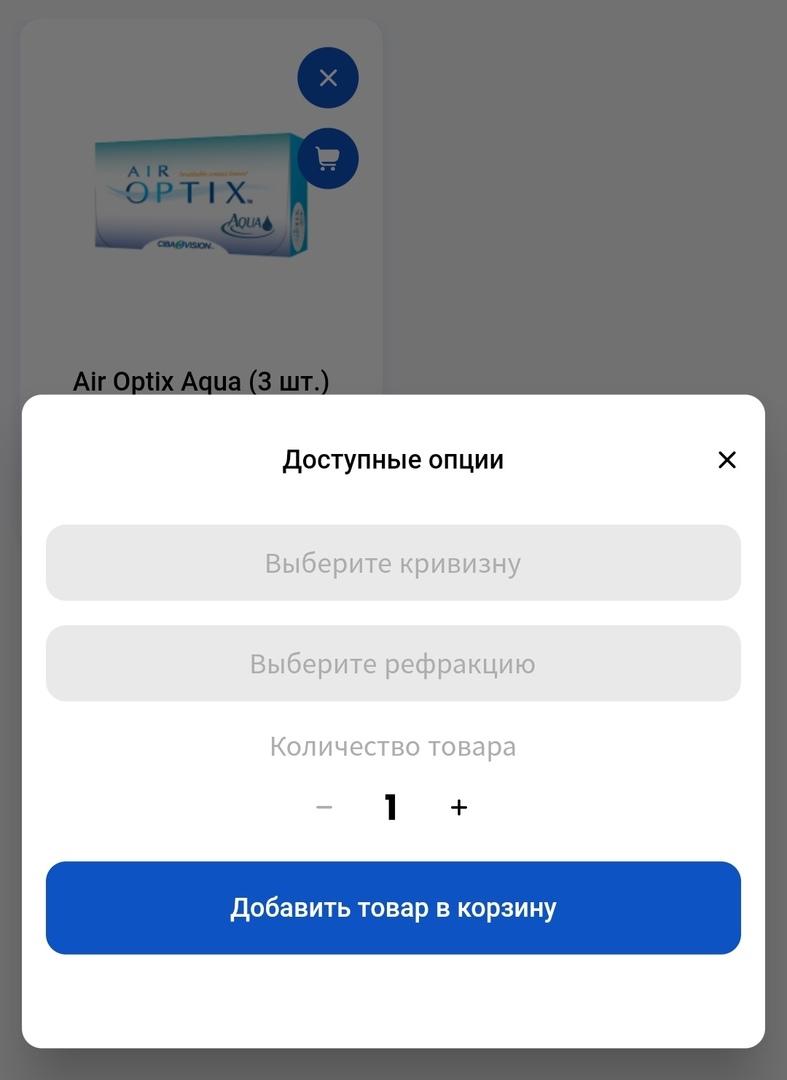


Рисунок 2.15 – Диалоговое окно выбора опций

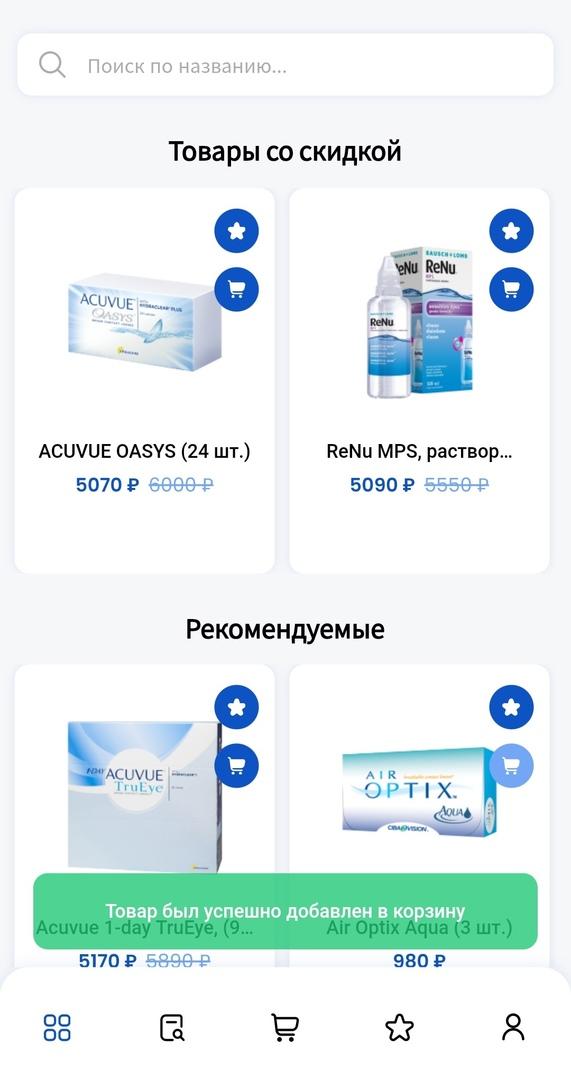


Рисунок 2.16 – Оповещение в виде снэк-бара

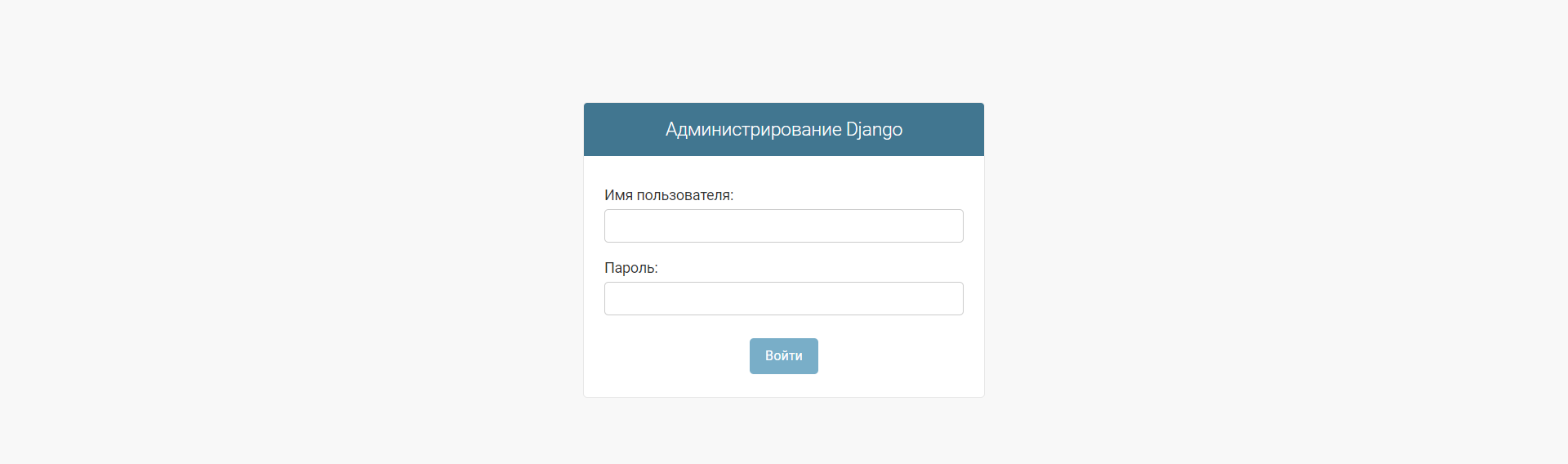


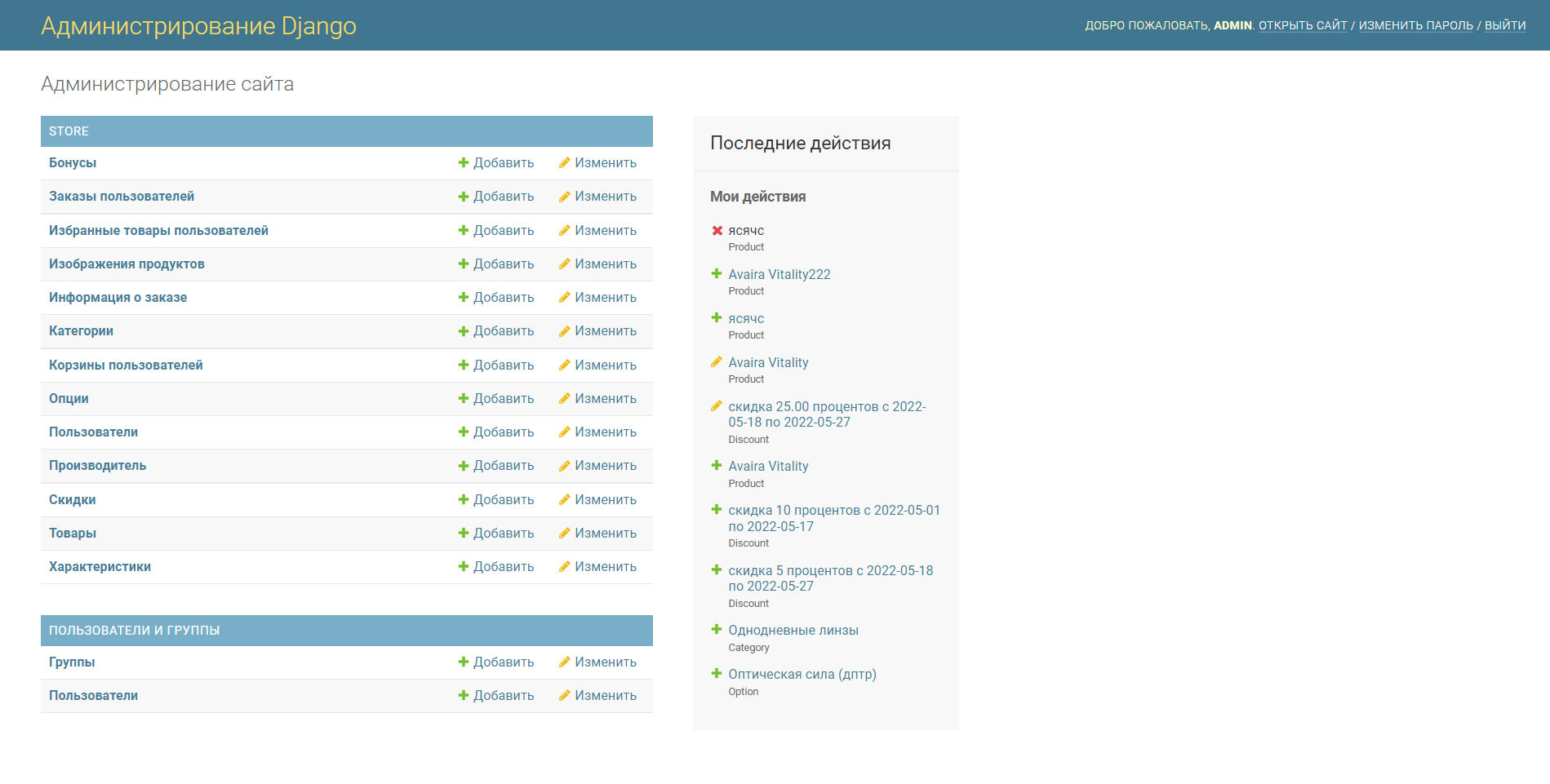
Рисунок 2.17 – Экран корзины с добавленными товарами

## 2.5 Back-end

### 2.5.1 Разработка административной части

Back-end является важной частью приложения, ведь он отвечает за осуществление функционирования внутренней части приложения. Также с помощью back-end’a осуществляется реализация админ-панели, через которую идет управление сайтом. Для нашего приложения была реализована админ-панель, а также графический интерфейс для удобного администрирования. Благодаря данному решению у администратора приложения будет возможность в любой точке мира с доступом к интернету добавить новые товары, отредактировать уже существующие, удалить их, и даже начислить скидку на какие-либо продукты. На рисунках 2.8 - 2.10 представлен интерфейс администратора.

Рисунок 2.18 – Вход в админ-панель

Рисунок 2.19 – Админ-панель

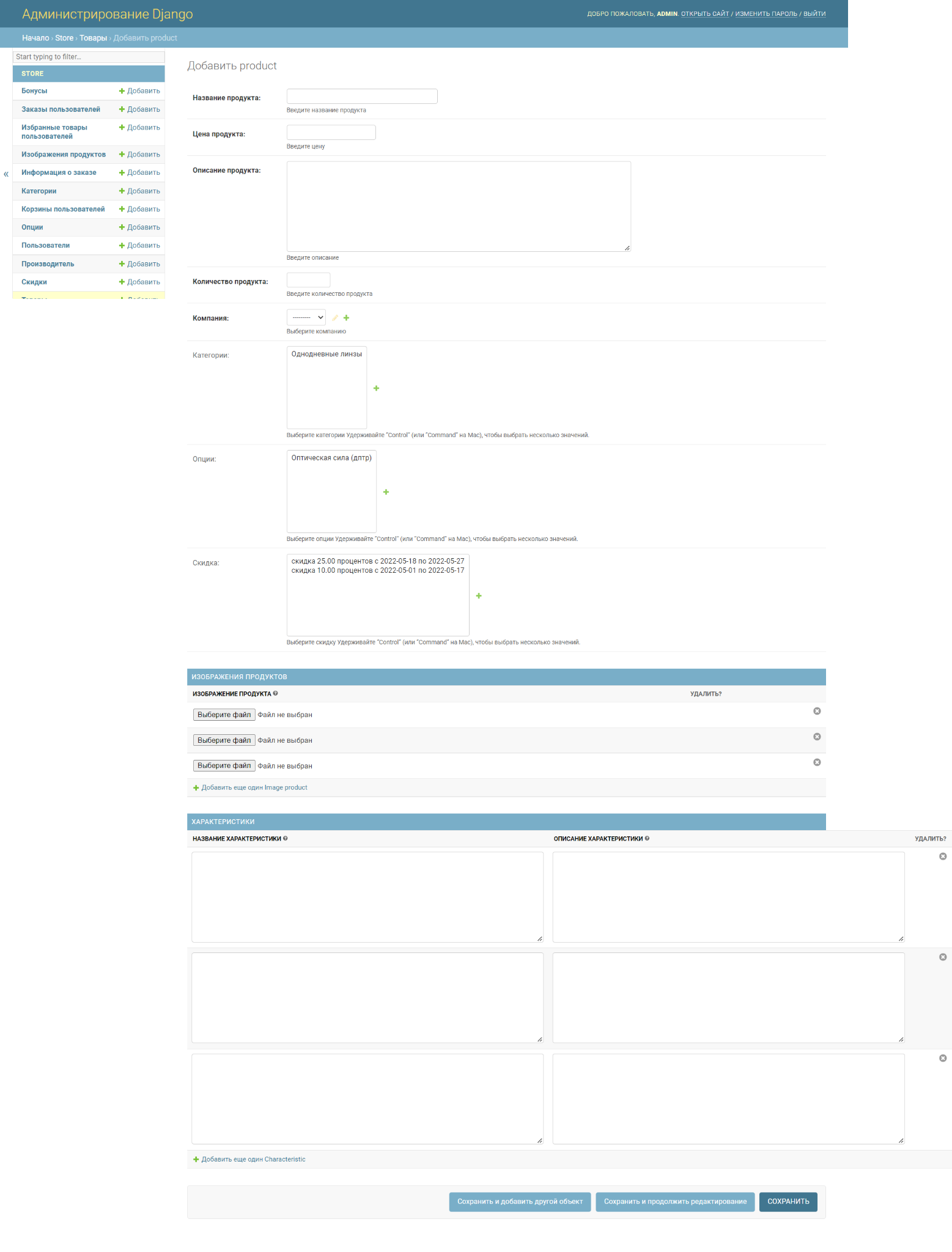


Рисунок 2.20 – Форма добавления продукта в админ-панель

Таким образом можно в удобном виде администрировать приложение. Для разработки данной части изучался и использовался Django admin.

# 

### 2.5.2 Разработка API

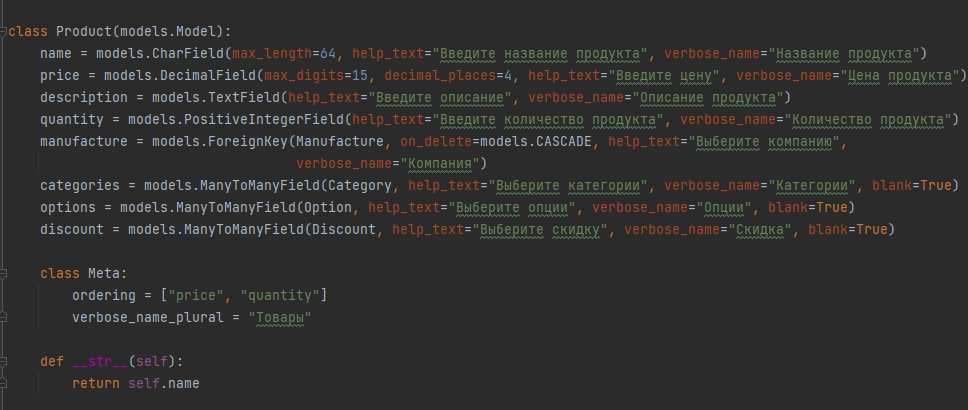
REST API — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful.

Термин состоит из двух аббревиатур, которые расшифровываются следующим образом. **API (Application Programming Interface)** — это код, который позволяет двум приложениям обмениваться данными с сервера. На русском языке его принято называть программным интерфейсом приложения. **REST (Representational State Transfer)** — это способ создания API с помощью протокола HTTP.

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространенный способ организации API.

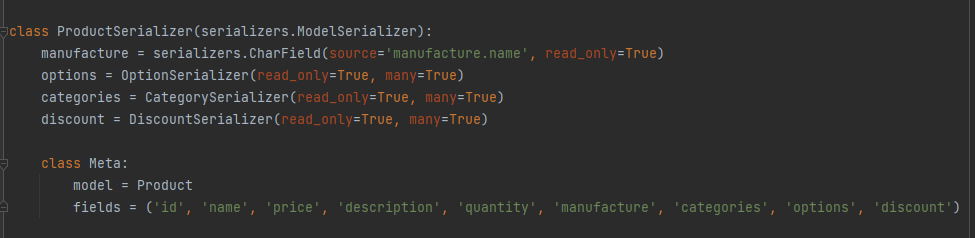
В данном проекте было решено реализовывать REST API на базе Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта.

Первым делом были модернизированы ранее созданные модели и добавлены новый. На рисунке 2.11 пример основной модели товара в базе данных.

Рисунок 2.21 – модель товара в базе данных

Следующим шагом была импортирована библиотека Serializers. **Сериализаторы** — это объекты, которые позволяют преобразовывать сложные данные, такие как наборы запросов и экземпляры моделей, в собственные типы данных Python, которые затем могут быть легко преобразованы в JSON, XML или другие типы содержимого. Сериализаторы также обеспечивают десериализацию, позволяя преобразовывать разобранные данные обратно в сложные типы после предварительной проверки входящих данных.

Для выгрузки данных из каждой модели базы данных были созданы сериализаторы. Пример одного из реализованных сериализаторов можно рассмотреть на рисунке 2.12, который выгружает информацию из модели товаров.

Рисунок 2.22 – Сериализатор товаров

Теперь было необходимо импортировать библиотеку Viewsets и создать типы представления информации для каждого и сериализаторов.

Класс **Viewset —** этопросто тип представления на основе класса, который не предоставляет никаких обработчиков методов, таких как .get() или .post(), а вместо этого предоставляет такие действия, как .list() и .create(). Обработчики методов для набора ViewSet привязываются к соответствующим действиям только в момент финализации представления с помощью метода .as\_view().

В данном проекте были разработаны viewset’ы для каждого из ранее реализованных сериализаторов. На рисунке 2.13 изображен класс viewset для сериализатора товаров.

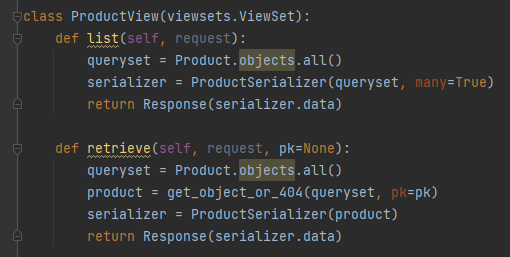
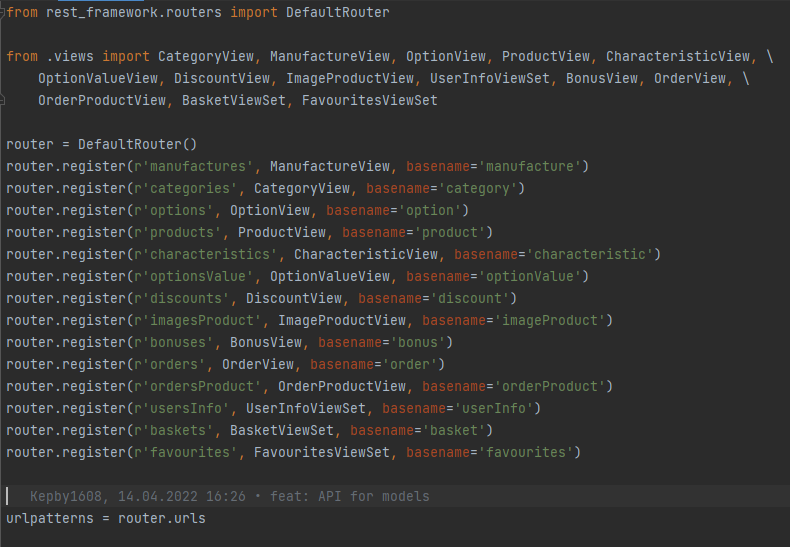
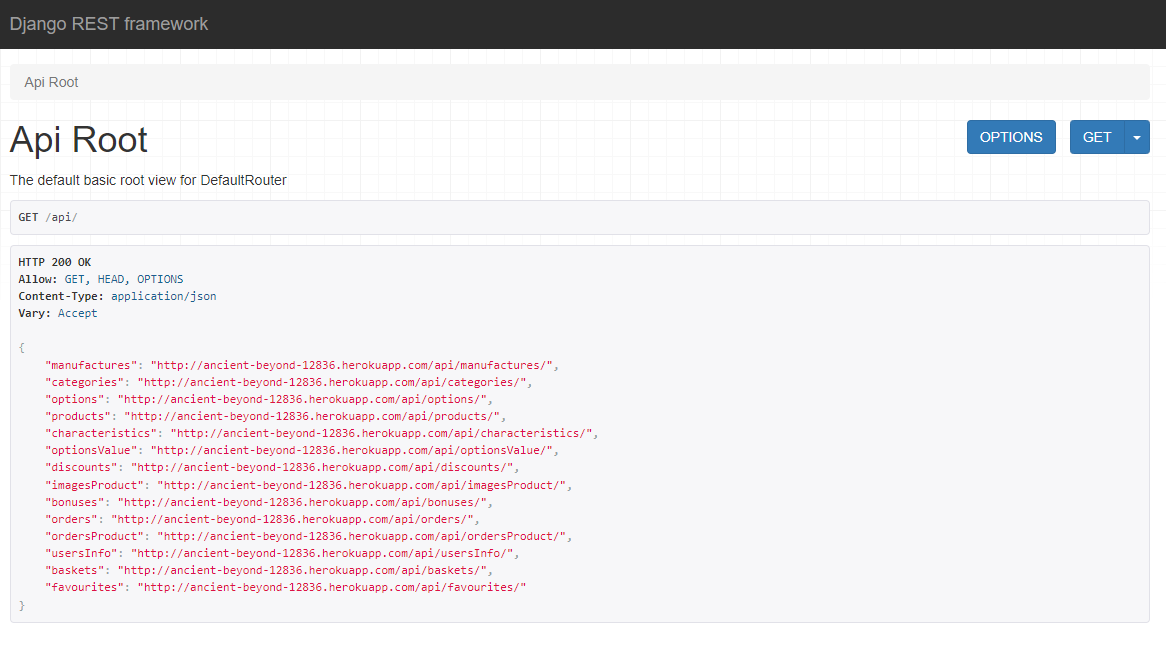


Рисунок 2.23 – Viewset для сериализатора товаров

Окончательным шагом реализации REST API было создание роутеров для каждого viewset’а.

Роутеры необходимы для возможности создания ссылок, с помощью которых осуществляется доступ к определенным данным. В нашем случае роутеры дают доступы к viewset’ам. Конечно для этого необходимо снова импортировать библиотеку DefaultRouter, после чего создать экземпляр класса, и с помощью встроенной функции .register() создать роутер для каждого viewset’а. На рисунке 2.14 изображен список всех созданных роутеров, а на рисунке 2.15 представлен API приложения.

Рисунок 2.24 – Список роутеров

Рисунок 2.25 – API приложения

Данный API позволяет соединить front-end с back-end частью приложения.

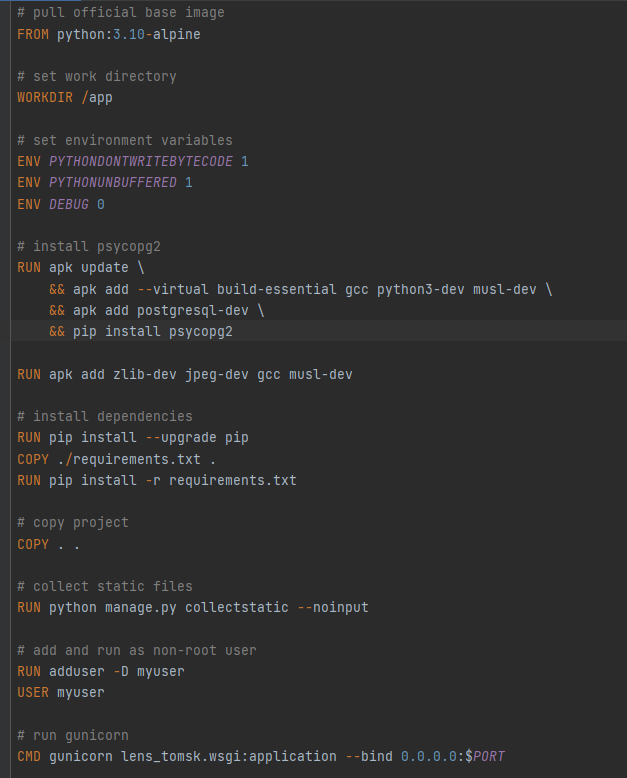
### 2.5.3 Размещение сервера на хостинг

Для доступа к API приложения с различных устройств необходимо разместить back-end приложение на хостинге. Были рассмотрены такие бесплатные облачные платформы, как Heroku и PythonAnywhere.

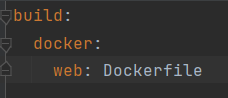
PythonAnywhere включает в себе запуск и редактирование web-приложений на основе распространенных веб-фреймворков Python, базу данных MySQL и PostgreSQL, онлайн консоль Bash, веб-редактор кода. Недостатком данной платформы является отсутствие поддержки новых версий Django, а также доступ к бесплатной версии, который ограничен по времени и позволяет использовать мощности сервера не более 100 секунд ежедневно, что стало причинами отказа от PythonAnywhere.

Heroku — Platform as a Service. Это означает, что платформа работает как сервис: предоставляет пользователю определенные функции и возможности, доступ к системам и ПО. При этом ее инфраструктура полностью скрыта. Для его использования был зарегистрирован аккаунт и установлен The Heroku Command Line Interface.

Перед размещением back-end части на хостинг необходимо было создать dockerfile и docker образ из этого файла, настроить проект. Для этого был создан dockerfile (рисунок 2.26).

  
Рисунок 2.26 – Настройки в dockerfile

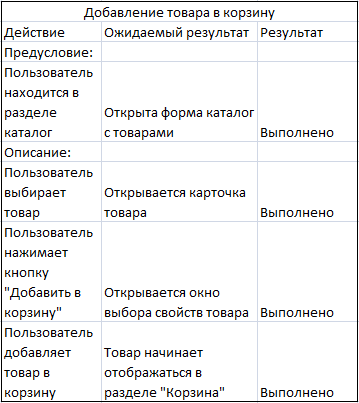
И для размещения на heroku был создан файл heroku.yml (рисунок 2.27)

  
Рисунок 2.27 – Часть кода из heroku.yml

С помощью Heroku CLI данный контейнер был опубликован на Heroku. Благодаря этому решению есть доступ из любой точки мира к API приложения.

## 2.6 Тестирование

На различных этапах разработки создавались тестовые сборки приложения для тестирования работоспособности функционала. Если во время тестирования находились ошибки или несоответствия ожидаемому результату, это сообщалось front-end разработчику. Тестирования выполнялись в рамках тест-кейсов (рисунок 2.28).

  
Рисунок 2.28 – Пример тест-кейса

На рисунке 2.29 показан эмулятор с запущенной тестовой сборкой приложения.

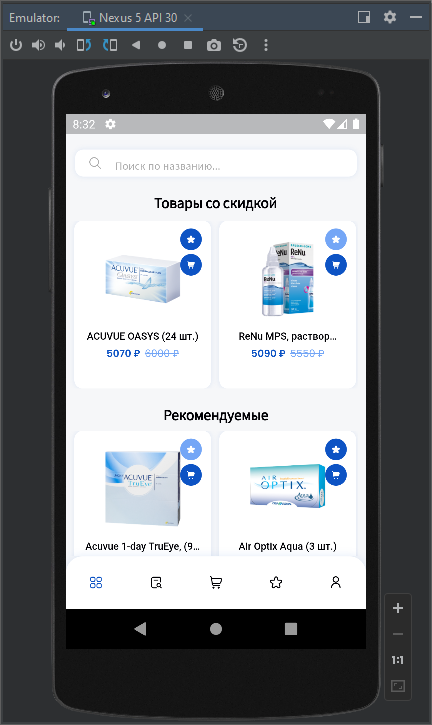


Рисунок 2.29 – Тестовая сборка

Также было проведено тестирование реализованного API, проводимое в соответствии с таблицей 2.1.

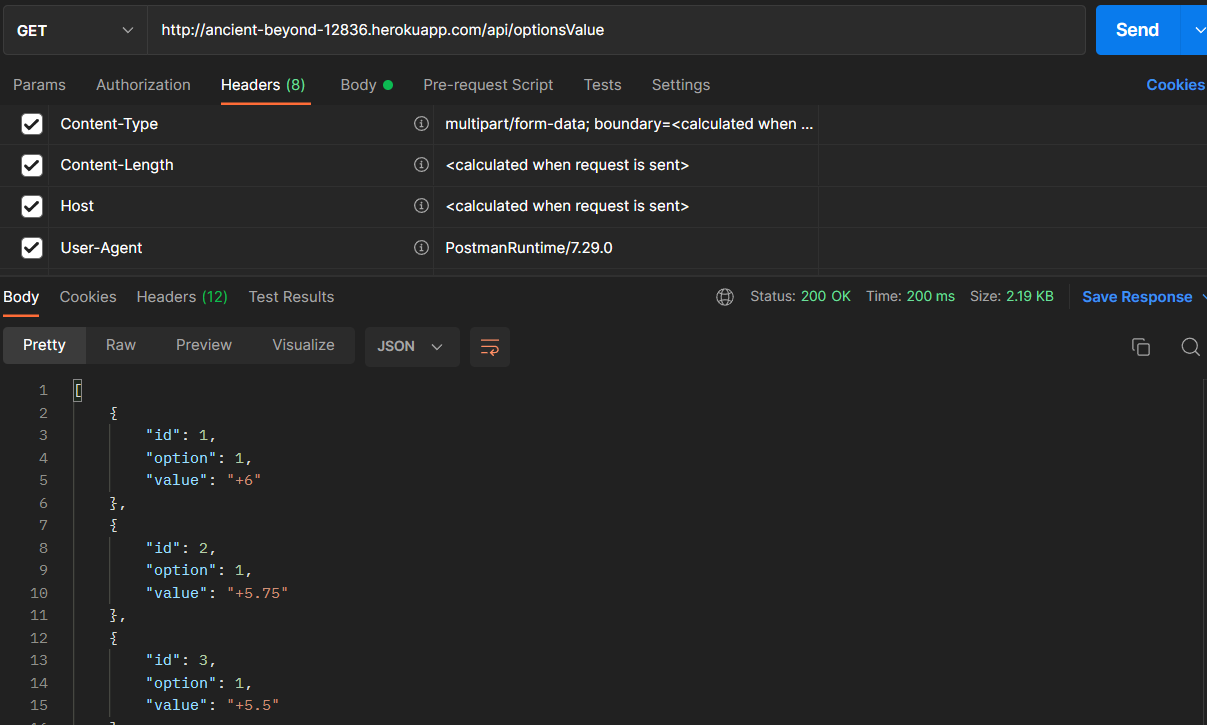
Таблица 2.1 – Описание тестирования API (GET запросы)

| Категория сценария тестирования | Категория тестового действия | Описание действия теста |
| --- | --- | --- |
| Основные позитивные тесты | | |
| Выполнить вызов API с валидными обязательными параметрами | Проверить код на состояние | 1. Все запросы должны возвращать код состояния HTTP 2XX  2. Код статуса возвращен согласно спецификации:  – 200 ОК для запросов GET  – и т.д |

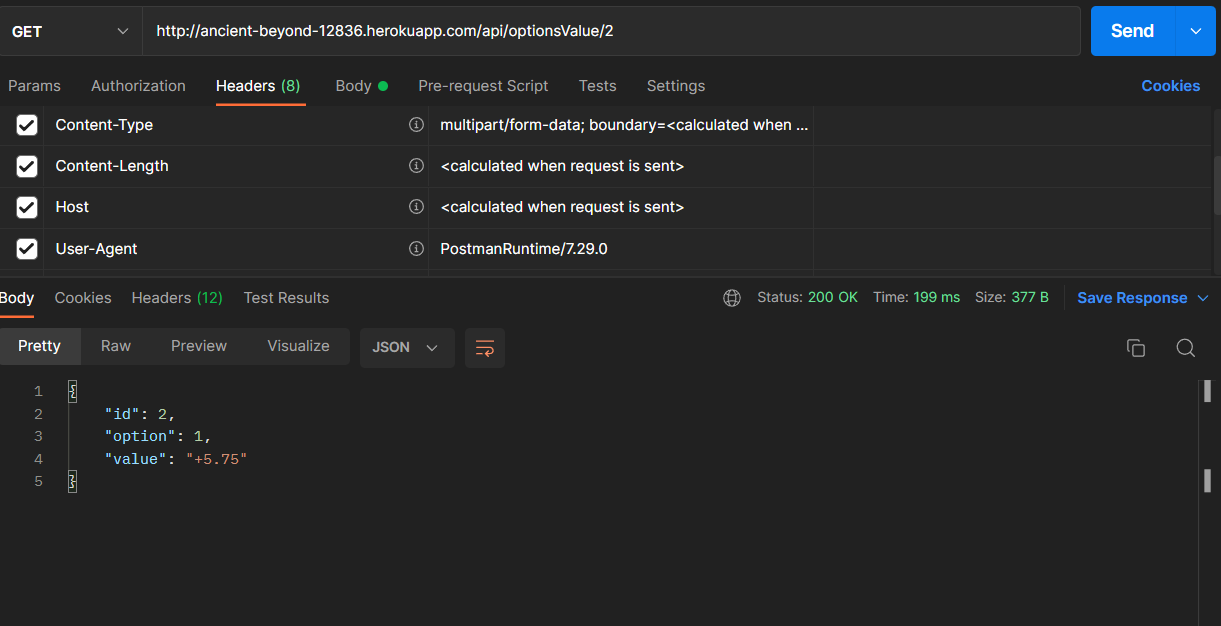
Окончание таблицы 2.1

| Категория сценария тестирования | Категория тестового действия | Описание действия теста |
| --- | --- | --- |
|  | Подтвердить загрузку ответа | 1. Ответ – правильно сформированный объект JSON  2. Структура ответа соответствует модели данных |
| Проверить производительность | Ответ получен своевременно ( в разумные ожидаемые сроки) |

На рисунке 2.30 представлен скриншот рабочего окна Postman — это платформа API, позволяющая разработчикам проектировать, создавать, тестировать и повторять свои API.

  
Рисунок 2.30 – GET запрос «optionsValue»

Результат тестирования запроса, присылающий в ответе конкретный объект, представлен на рисунке 2.31

  
Рисунок 2.31 – GET запрос «optionsValue» с параметром

Остальные запросы тестировались аналогичным способом и успешно прошли тестирование.

# Заключение

В течение данного семестра участниками проектной группы было написано три научных статьи, на основе которых был изменён дизайн интерфейса приложения и, в связи нынешними ограничениями, проанализированы альтернативные способы онлайн оплаты. Разработана серверная часть приложения, а именно интерфейс программирования приложений (API), админ панель и размещение на удаленный хостинг. По части front-end приложение было адаптировано под архитектурный паттерн BLoC (реализация Cubit). Также реализовано добавление товаров в корзину с появлением диалогового окна с возможными опциями и в список избранных. Полностью доработана детальная страница товара, отображение каталога в виде сетки, где добавлены возможности фильтрации и сортировки товаров.

В следующем семестре планируется завершение разработки мобильного приложения, а также добавление функции платежного модуля. Запуск приложения в PlayMarket.

# Список литературы

Что выбрать: ЮKassa (Яндекс.Касса) или Робокасса? [Электронный ресурс] — URL: https://otzyvmarketing.ru/articles/sravnenie-yandekskassa-ili-robokassa/ (Дата обращения: 08.05.22).

Что такое kanban-доска? [Электронный ресурс] — URL: https://www.atlassian.com/ru/agile/kanban/boards (Дата обращения: 24.03.22).

Как управлять задачами через карточки и не допустить революцию в офисе [Электронный ресурс] — URL: https://rb.ru/opinion/kartochki-v-ofise/ (Дата обращения: 31.03.22).

Документация по Flutter [Электронный ресурс] – URL: <https://flutterdocs.ru/> (Дата обращения: 01.04.2022).

Визуальная ASO оптимизация приложения и ее влияние на конверсию [Электронный ресурс] — URL: https://asomobile.net/blog/vizualnaya-aso-optimizacziya-prilozheniya-i-ee-vliyanie-na-konversiyu/ (Дата обращения: 13.04.22).

Руководство по фреймворку Flutter [Электронный ресурс] – URL: <https://metanit.com/dart/flutter/> (Дата обращения 15.04.2022).

Мировой рынок мобильных приложений: как привлекать и удерживать аудиторию [Электронный ресурс] — URL: https://netology.ru/blog/06-2021-mobilnye-prilozheniya (Дата обращения: 20.04.22).

Документация по Django [Электронный ресурс] – URL: https://docs.djangoproject.com/en/4.0 (Дата обращения: 20.03.2022).

Статьи по Django admin [Электронный ресурс] – URL: https://tutorial.djangogirls.org/ru/django\_admin/ (Дата обращения: 21.03.2021).

Документация по Django REST framework [Электронный ресурс] – URL: https://www.django-rest-framework.org/ (Дата обращения: 16.04.2021).

Статьи по Django REST framework [Электронный ресурс] – URL: https://webdevblog.ru/sozdanie-django-api-ispolzuya-djangorestframework-chast- (Дата обращения: 17.04.2021).

Документация по docker [Электронный ресурс] – URL: https://docs.docker.com/ (Дата обращения: 22.04.2021).

Статья по размещению сервера на heroku [Электронный ресурс] – URL: https://testdriven.io/blog/deploying-django-to-heroku-with-docker/ (Дата обращения: 24.04.2021).

Документация по heroku [Электронный ресурс] – URL: https://devcenter.heroku.com (Дата обращения: 24.04.2021).