**Введение**

В последние годы рынок электронных услуг и продаж претерпевает значительные изменения, что обуславливает необходимость создания удобных и функциональных платформ для взаимодействия продавцов и покупателей. Одним из наиболее актуальных решений в данной области являются маркетплейсы, которые позволяют объединить различные товарные предложения в одном месте, обеспечивая пользователей широким выбором и удобством покупок. В этой связи возникает необходимость разработки эффективного маркетплейса, который сможет отвечать современным требованиям пользователей и обеспечивать высокую производительность .Целью данной работы является разработка маркетплейса на основе фреймворка Django, который будет включать в себя все основные функции для удобного управления товарами, пользователями и заказами. Для достижения этой цели необходимо решить следующие задачи:

1. **Изучение существующих решений**: Провести анализ существующих маркетплейсов, их функциональности и архитектуры, выявить их преимущества и недостатки.
2. **Проектирование архитектуры системы**: Разработать архитектуру маркетплейса, определив основные подсистемы и их взаимодействие.
3. **Разработка функциональных требований**: Сформулировать требования к системе, включая функциональные и нефункциональные характеристики.
4. **Реализация системы**: На основе полученных требований разработать маркетплейс, используя фреймворк Django.
5. **Тестирование и оптимизация**: Провести тестирование системы на предмет функциональности и производительности, а также оптимизировать её работу.

Работа состоит из нескольких разделов, каждый из которых раскрывает отдельные аспекты темы. В первом разделе представлен обзор существующих решений в области маркетплейсов, а также анализ их функциональности. Второй раздел посвящен проектированию архитектуры разрабатываемой системы, где описываются все подсистемы и их взаимодействие. В третьем разделе формулируются требования к системе, включая как функциональные, так и технические аспекты. Четвертый раздел описывает процесс реализации системы, а также ключевые технологии, использованные в разработке. В заключительном разделе проводятся тестирование и оценка полученных результатов, а также предлагаются рекомендации по дальнейшему развитию системы.Таким образом, данная работа направлена на создание современного и эффективного маркетплейса на фреймворке Django, который будет способен удовлетворять потребности пользователей и обеспечивать высокий уровень сервиса. Результаты исследования могут быть полезны как для разработчиков, так и для пользователей, заинтересованных в внедрении и использовании подобной системы.

**2. Проектная часть**

В данном разделе представлены проектные решения для создания маркетплейса на фреймворке Django. Мы рассмотрим выбор средств и методов проектирования, обоснуем их, а также опишем процесс проектирования системы с использованием иллюстрационного материала.

**1. Выбор средств и методов проектирования**

Для разработки маркетплейса был выбран фреймворк **Django**, который является мощным инструментом для создания веб-приложений на языке Python. Основные причины выбора Django включают:

* **Быстрая разработка**: Django предоставляет множество встроенных функций, что позволяет сократить время на разработку.
* **Безопасность**: Фреймворк включает средства для защиты от распространенных уязвимостей, таких как SQL-инъекции и межсайтовый скриптинг (XSS).
* **Масштабируемость**: Django позволяет легко масштабировать приложение по мере роста числа пользователей и объема данных.
* **Сообщество и документация**: Широкое сообщество разработчиков и обширная документация упрощают процесс обучения и решения возникающих проблем.

Метод проектирования, который будет использоваться в данной работе, основан на **агильной методологии**. Это позволит гибко реагировать на изменения требований и обеспечит возможность итеративной разработки, что особенно важно для динамичных проектов, таких как маркетплейсы.

**2. Процесс проектирования системы**

Процесс проектирования системы включает несколько ключевых этапов:

1. **Анализ требований**: На этом этапе были собраны и проанализированы требования пользователей и заказчика. Это позволило определить основные функции, которые должен выполнять маркетплейс, такие как управление товарами, обработка заказов и взаимодействие с пользователями.
2. **Проектирование архитектуры**: Архитектура системы была спроектирована с использованием шаблона **Model-View-Template (MVT)**, который является основой Django. Это позволяет разделить логику приложения, пользовательский интерфейс и данные, что упрощает поддержку и развитие системы.
3. **Проектирование базы данных**: Для хранения данных о пользователях, товарах и заказах была разработана схема базы данных. Основные сущности включают:
   * **Пользователь**: хранит информацию о зарегистрированных пользователях.
   * **Товар**: содержит данные о товарах, доступных для продажи.
   * **Заказ**: фиксирует информацию о заказах, сделанных пользователями.
4. **Проектирование пользовательского интерфейса**: Интерфейс был спроектирован с учетом удобства использования и доступности. Основные страницы включают:
   * Главная страница с каталогом товаров.
   * Страница товара с подробной информацией.
   * Личный кабинет пользователя для управления заказами и профилем.

**3. Разработанные диаграммы**

Изображение выглядит как текст, диаграмма, рисунок, шаблон

Автоматически созданное описание

Рисунок 1

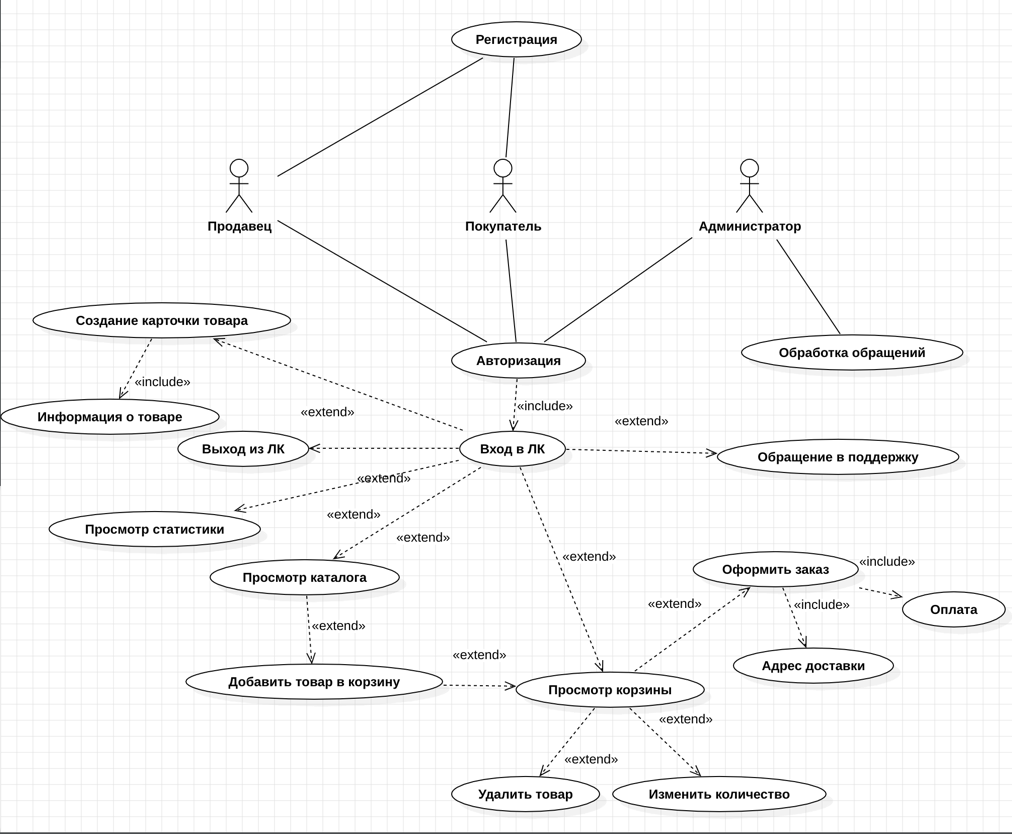


Рисунок 2

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описаниеИзображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание   
Рисунок 4

*Изображение выглядит как текст, диаграмма, линия, План

Автоматически созданное описание*  
Рисунок 5

Диаграмма покупки товара представлена на рисунке 1

*Изображение выглядит как текст, снимок экрана, диаграмма, программное обеспечение

Автоматически созданное описание*

Рисунок 1

Диаграмма продажи товара представлена на рисунке 2.

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

Рисунок 2

*Изображение выглядит как текст, снимок экрана, число, диаграмма

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, число, диаграмма

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание*

*Изображение выглядит как текст, снимок экрана, диаграмма, программное обеспечение

Автоматически созданное описание*

**3. Разработка интерфейса и программная реализация системы**

* 1. **Технические характеристики**

Разработка производилась на операционной системе Windows 11. Используемое IDE: PyCharm Professional. Браузер: Яндекс браузер 24.4.1.899 (64- bit). Python 3.12.

Используемая база данных: MySQL. Характеристики MySQL отображены на рисунке 2.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1

# Виртуальное окружение и используемые библиотеки

Создание и активация виртуального окружения осуществляется следующими командами:

python -m venv .venv

.\.venv\Scripts\activate.bat

Устанавливаем нужные библиотеки с помощью pip:

pip3 install Django mysqlclient telegram-django-bot python-dotenv

Записываем все библиотеки в файл requirements.txt с помощью команды: pip3 freeze > requirements.txt (Приложение А).

# Старт проекта Django и создание приложений.

Создаем проект с помощью команды и не создаем лишние папки с помощью точки в конце:

(.venv) PS C:\Users\admin\Documents\pythonProject2> django-admin startproject main .

После создания создается основная папка проекта (рисунок 2.2).

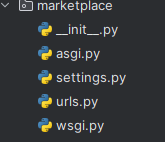


Рисунок 2.2

В файле settings.py настраиваем проект, загружаем виртуальное окружение и убираем SECRET\_KEY в виртуальное окружение .env.

Далее, в командной строке MySQL создаем базу данных под названием

«marketplace» и вписываем все данные о пользователе и о базе данных в файл

.env.

В settings.py настраиваем БД, вписываем все данные о пользователе и базе данных (рисунок 2.3)

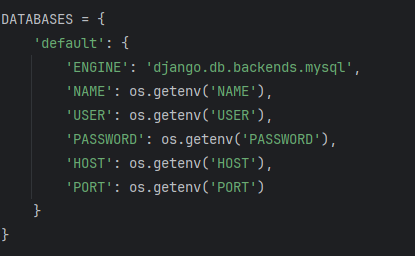


Рисунок 2.3

После этого, разбиваем сайт на приложения (для модульности), а именно:

* + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp buyer .\apps\buyer\ - приложение покупателя;
  + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp seller .\apps\seller\ - приложение продавца;
  + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp product .\apps\product\ - приложение продуктов;
  + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp main .\apps\main\ - приложение главной страницы;
  + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp telegram\_bot – приложение для Telegram бота;
  + (.venv) PS C:\Users\admin\Documents\pythonProject2> python manage.py startapp CustomUser – приложение для пользователя.

Получается следующая структура проекта (рисунок 4)

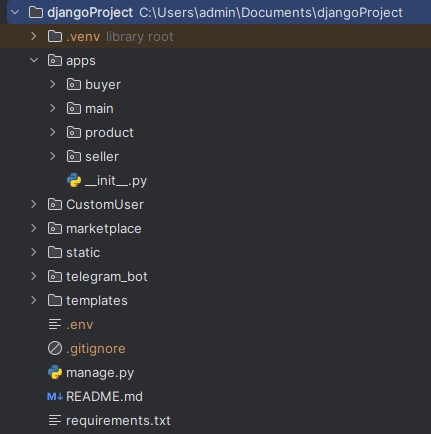


Рисунок 2.4

# Создание моделей.

Модель для телеграмм бота создана для администраторов(рисунок 2.5).

Важные элементы модели Owner:

* + Role – обработка роли. Нужна для того, чтобы обрабатывать пользователей, кто пишет в бота.
  + Telegram\_id – аутентификация пользователя по его Telegram id.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.5

Стандартная модель пользователя в Django ORM недостаточна для реализации. Чтобы добавить поля, нужно было создать модель CustomUser, который наследуется от AbstractUser. С помощью этого получилось добавить в модель пользователя необходимые поля, а именно: inn, seller, telephone\_number, bonus, last\_name, age, seller, permission\_to\_sell.

Модель продавца и покупателя одинакова (рис. 2.6). Чтобы отличить покупателя от продавца было добавлено два поля:

1. Seller – является ли пользователь продавцом (имеет форму булева значения);
2. Permission\_to\_sell – разрешение на продажу (имеет форму булева значения).



Рисунок 2.6

При регистрации пользователя поле seller ставится по умолчанию на False. Данное поле может изменить только администратор, предварительно одобрив заявку на продавца в Telegram боте.

Модели товаров, категорий, подкатегорий и корзины реализованы в приложении product (Приложение С). В моделях реализованы связи к другим моделям при помощи связей ForeignKey и ManyToMany.

# Определение маршрутов страниц и обработка запросов

Для обработки запросов, в каждом приложении есть views.py, в котором реализуется обработка запросов. В приложениях реализованы следующие обработчики:

* 1. Приложение Buyer
     1. BuyerRegisterView(CreateView) – обрабатывает регистрацию пользователя (наследуется от класса CreateView);
     2. BuyerRegisterView(CreateView) – обрабатывает регистрацию пользователя (наследуется от класса CreateView);
     3. BuyerLoginView(LoginView) – обрабатывает логирование пользователя (наследуется от класса LoginView)
     4. logout\_view(request) – обрабатывает запрос выхода пользователя из системы (реализован обычной функцией def)

Этим обработчикам соответствуют следующие маршруты (рис. 2.7).

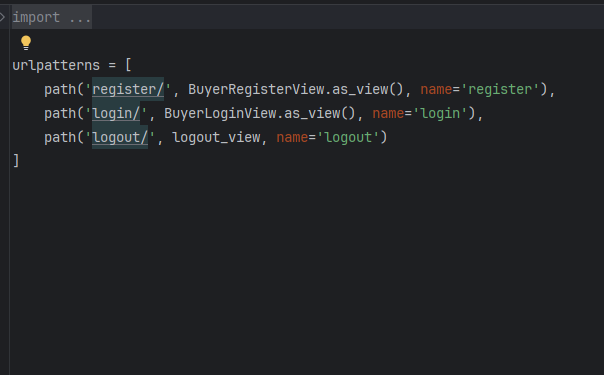


Рисунок 2.7

* 1. Приложение Seller
     1. seller\_index – обрабатывает начальную страницу продавца.
     2. seller\_register – регистрация заявки на разрешение продажи на маркетплейсе.
     3. seller\_profile – отображение профиля продавца.

Этим обработчикам соответствуют следующие маршруты (рис. 2.8).



Рисунок 2.8

* 1. Приложение main
     1. Index – обрабатывает стартовую страницу для всех пользователь.
  2. Приложение seller
     1. categories\_list – отображает категории.
     2. card\_subcategory – отображает подкатегории.
     3. product\_list – выводит список продуктов, выбранной категории.
     4. view\_cart – корзина пользователя.
     5. add\_to\_cart – добавить товар в корзину.
     6. remove\_from\_cart – удаление товара с корзины.

Этим обработчикам соответствуют следующие маршруты (рис. 2.9).

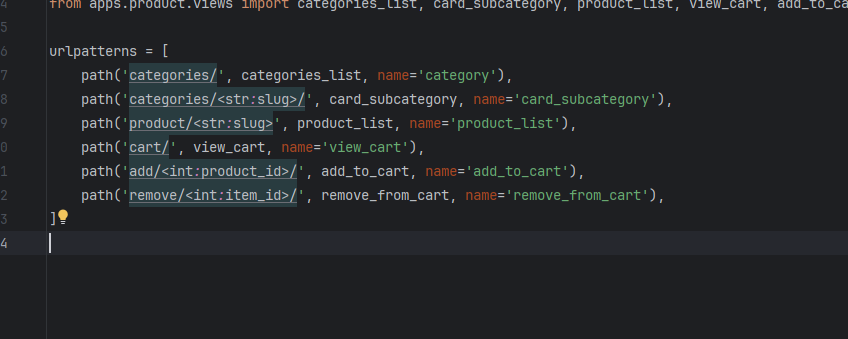


Рисунок 2.9

# Шаблонизация Django

В фреймворке Django поддерживается шаблонизация HTML страниц. Это позволяет не повторять код, тем самым и поддерживается принцип разработки DRY[2](#_bookmark23). Поэтому в корневой папке templates был создан базовый шаблон base.html (Приложение В), который будет основой для остальных страниц.

Строка {% load static %} загружает все статические файлы (стили, js, картинки). Конструкция {% block content %} сделана для того, чтобы в этот промежуток вставлялась другая html разметка.

# Создание Telegram бота для обработки заявок

Telegram бот разработан в отдельном приложении и помещен в папку management. Бот запускается отдельно от проекта, нужно это для того, чтобы не помещать бота в процесс демон и не нагружать систему.

Запросы к базе данных осуществляются через ORM Django. Сам бот помещен в класс Command, который наследуется от BaseCommand. Это нужно для того, чтобы мы могли запустить бота через manage.py. В Command реализована функция handle, в которой запускается бот. API\_KEY бота загружается с основного файла settings.py. При старте бот обрабатывает команду /start. В ответ пользователю он отправляет приветственное письмо, телеграмм id пользователя и тег главного администратора, который сможет добавить в систему. При отправке команд не авторизированного пользователя он не обрабатывает команды, так как его нету в

БД.

2 Принцип программирования DRY (англ. Don’t repeat yourself, в переводе — «Не повторяй себя») предполагает разделение крупной системы на более мелкие, неповторяющиеся компоненты.

Для проверки пользователя, была реализована функция check\_role, которая отправляет запрос к БД на существования пользователя (рис. 2.10) и возвращает кортеж.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.10

Также реализованы функции, которые формируют сообщения. А именно:

1. format\_message – формирует сообщение заявки
2. format\_administrator\_list – отправляет список администраторов.

Если роль пользователя была проверена успешно, то ему открывается доступ к двум командам /enter, /work, /orders. Если роль владельца, то открывается доступ к /add\_administrator и /delete\_administrator

Бот запускается с помощью команды: python manage.py bot

# Создание БД в MySQL

Перед миграцией БД, создаем БД под названием marketplace с помощью команды:

Create database marketplace;

Далее создаем и выполняем миграции (python manage.py makemigrations, python manage.py migrate), чтобы Django ORM создал базу данных.

После миграций появились следующие таблицы (рис. 2.11).

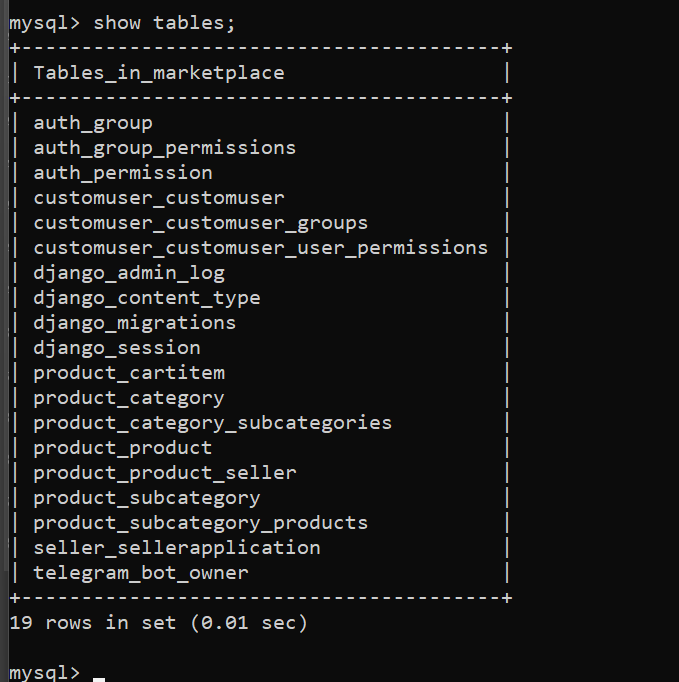


Рисунок 2.11

# Тестирование

Создаем суперпользователя, чтобы добавить Telegram id владельца.

Запускаем проект и бота. Стартовая страница (рис. 2.12)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2.12

Чтобы зайти в админ панель, нужно перейти по ссылке 127.0.0.1:8000/admin и добавляем в Telegram бота владельца (рис. 2.13).

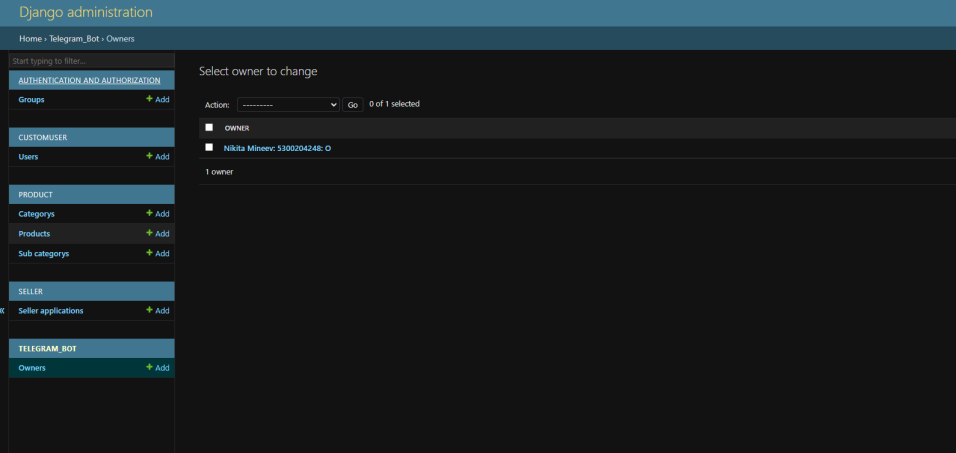


Рисунок 2.13

Теперь проверим бота (рис. 2.14 и рис.2.15)

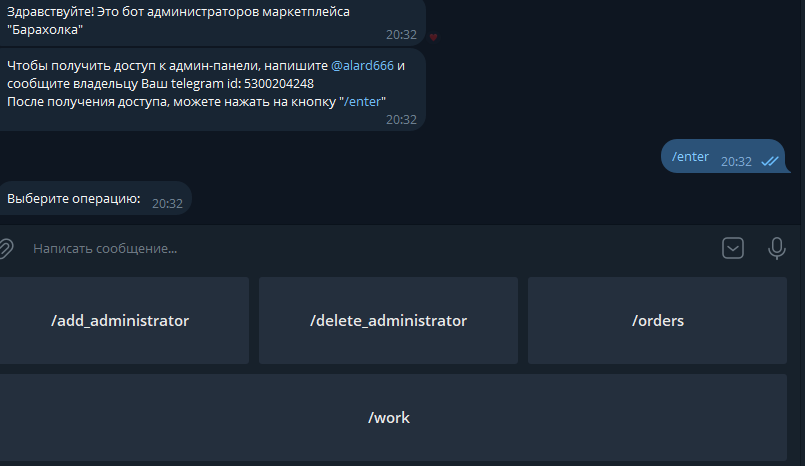


Рисунок 2.14

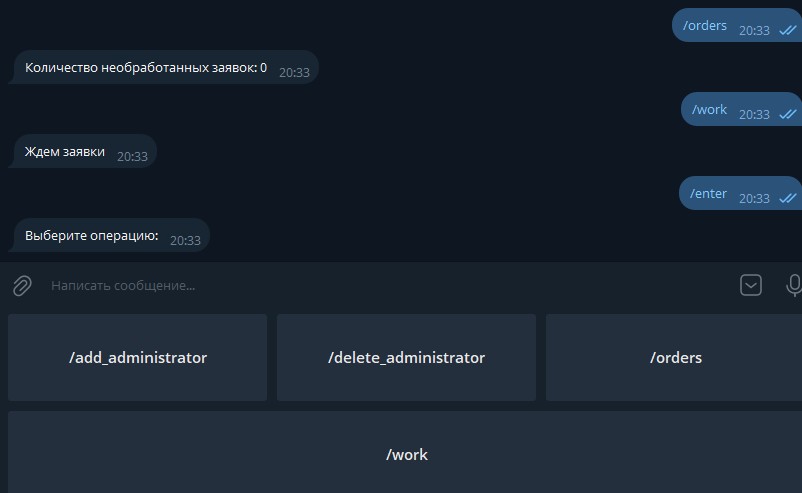


Рисунок 2.15

Создадим заявку на продавца (рис. 2.16).

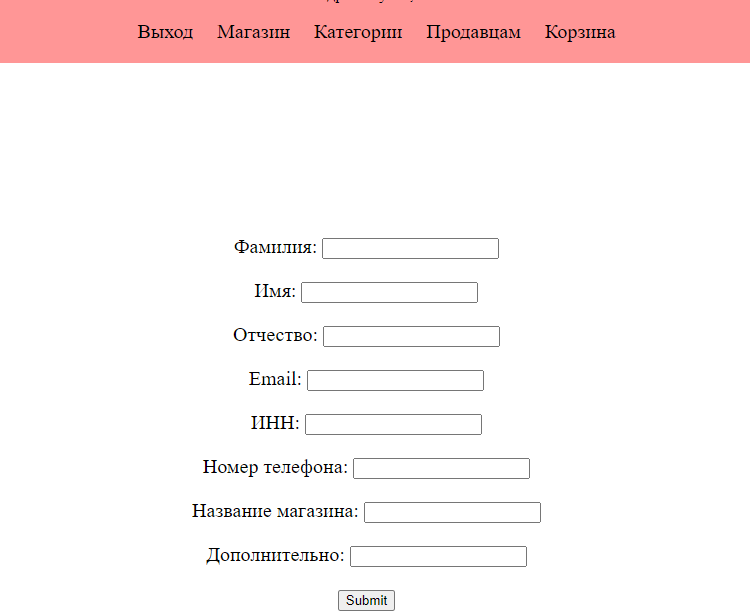


Рисунок 2.16

Обрабатываем заявку в Telegram боте. После одобрения генерируется случайный пароль (рис. 2.17).

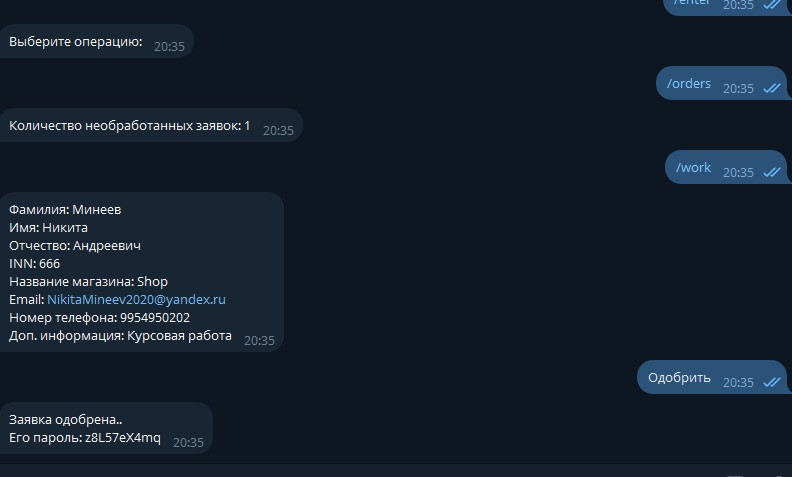


Рисунок 2.17

Зарегистрируем покупателя и кинем продукты в корзину (рис. 2.18).

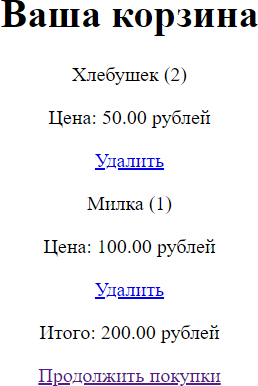


Рисунок 2.18

1. Григорьев, А. В. (2021). **Разработка веб-приложений на Django**. Москва: Издательство "Наука".
2. Сидоров, И. П. (2022). **Создание интернет-магазина на Django**. Санкт-Петербург: Издательство "Питер".
3. Кузнецов, В. А. (2023). **Django для профессионалов**. Екатеринбург: Издательство "Урал".
4. Петрова, Е. С. (2020). **Основы проектирования веб-приложений**. Новосибирск: Издательство "Сибирское образование".
5. Иванов, Д. Н. (2022). **Электронная коммерция: теория и практика**. Казань: Издательство "Казанский университет".
6. Федоров, А. В. (2023). **Python и Django: создание веб-приложений**. Ростов-на-Дону: Издательство "Феникс".
7. Смирнова, Т. И. (2021). **Маркетплейсы: как создать успешный бизнес**. Москва: Издательство "Альпина Паблишер".
8. Лебедев, С. А. (2022). **Веб-разработка на Python**. Минск: Издательство "Технопринт".
9. Ковалев, Р. П. (2023). **Современные подходы к разработке веб-приложений**. Владивосток: Издательство "Дальний Восток".
10. Соловьев, М. В. (2020). **Интернет-магазины: от идеи до реализации**. Челябинск: Издательство "Южный Урал".