МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение Московской области «Ногинский колледж»

Курсовой проект

по МДК.09.01 Проектирование и разработка веб-приложений

ПМ.09 Проектирование, разработка и оптимизация веб-приложений

Тема:

ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА FLASK В РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЯ «СЛУЖБА ДОСТАВКИ ЕДЫ “КЕБАБХАУС”»

Разработчик: студент группы 3ИСПР

Егоров Д.Н.

(подпись)

Оценка защиты курсового проекта

« »

Дата защиты

« » 2024

Руководитель проекта:

преподаватель Степанов С.О.

(подпись)

Ногинск, 2024 г

Оглавление

[Введение 3](#_Toc169288047)

[Основная часть 5](#_Toc169288048)

[1. Установка всех необходимых для работы зависимостей, подготовка к работе 5](#_Toc169288049)

[Разработка веб-приложения 6](#_Toc169288050)

[1.1. Frontend часть (HTML, JavaScript и SCSS.) 6](#_Toc169288051)

[1.2. Backend часть (Flask) 11](#_Toc169288052)

[Заключение 27](#_Toc169288053)

# Введение

**Задача:** Главная цель в курсовой работе – изучение нового framework’a под названием Flask и разработка с помощью него веб-приложения с требуемым в техническом задании функционалом.

**Актуальность**: Веб-приложения находят широкое применение в самых разных областях, включая электронную коммерцию, социальные сети, образовательные платформы и системы управления контентом. В данном проекте рассматривается разработка приложения для управления задачами, которое будет полезно как для индивидуальных пользователей, так и для небольших команд.

На сегодняшний день рынок доставки еды представлен множеством платформ, предлагающих разнообразные блюда и кулинарные предложения. С ростом популярности онлайн-заказов еды конкуренция среди сервисов доставки становится все более острой, что побуждает их постоянно совершенствоваться и использовать новые стратегии для привлечения клиентов.

Сервисы доставки еды предлагают широкий ассортимент блюд: от фаст-фуда и пиццы до блюд высокой кухни и вегетарианских опций. Многие из них также предоставляют дополнительные услуги, такие как собственные бренды, программы лояльности и скидок, что помогает удерживать клиентов и стимулировать повторные заказы.

Одним из значимых трендов в сфере доставки еды является быстрое развитие мобильных приложений и мобильной торговли. Пользователи предпочитают заказывать еду с мобильных устройств благодаря удобству и доступности, поэтому многие сервисы активно развивают мобильные версии своих сайтов и создают собственные приложения.

Система доставки еды также сталкивается с проблемами в области защиты данных и конфиденциальности пользователей. В связи с ростом онлайн-мошенничества и утечек данных возрастает ответственность сервисов за защиту личной информации клиентов и обеспечение безопасных платежей.

В общем, рынок доставки еды демонстрирует стабильный рост и разнообразие, предлагая клиентам широкий выбор блюд и кулинарных услуг, удобные способы оплаты и доставки, а также возможность пользоваться скидками и акциями. Для успешной работы в этой сфере важно следить за актуальными трендами, улучшать сервис и адаптироваться к изменяющимся потребностям клиентов.

**Задачи:**

1. Выбрать тему курсовой работы.
2. Изучить фреймворк Flask.
3. Изучить дополнительные библиотеки Python, необходимые для дополнения функционала.
4. Написать веб-приложение при помощи вышеперечисленного Flask в качестве backend, а также SASS в качестве frontend.
5. Написать и защитить курсовую работу.

# Основная часть

## Установка всех необходимых для работы зависимостей, подготовка к работе

Для начала работы с фреймворком Flask, необходимо для начала скачать Язык Программирования, на котором основывается данный фреймворк. В нашем случае это Python.

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

После установки Python, для установки самого Flask следует зайти в командную строку (cmd) и прописать команду python3 -m pip install flask (либо же pip3 install flask). Дополнительные модули – flask-login (python3 -m pip install flask-login), smtplib, sqlite3, string, random, os (идут в комплекте при установке Python)

Для установки компилятора SASS, нужно зайти в программу Microsoft Visual Studio Code (далее - VSCode), зайти в раздел Extensions, далее в поиске написать “Live Sass Compiler” и нажать кнопку Install.

В том же VSCode нужно установить пакеты зависимостей python, дабы программа могла определять ошибки и правильно определять код.

На этом установка зависимостей закончена, можно приступить к написанию frontend части.

# Разработка веб-приложения

## Frontend часть (HTML, JavaScript и SCSS.)

Для начала работы нам нужно создать файлы html и scss. Обычно главная страница называется index.html. Открываем её в редакторе, таком как VSCode, который предоставляет удобные инструменты для форматирования HTML. Чтобы быстро создать шаблон страницы, достаточно ввести "!" и нажать Enter – VSCode автоматически сгенерирует базовую структуру сайта.

Сайт разрабатывается с учетом принципа "сверху-вниз". Сначала мы создали Header – верхнюю часть страницы. Он состоит из двух частей: top-nav с указанием номера телефона и надписью "круглосуточно", и обычного navbar с ссылками на главную страницу, меню и отдельную панель для курьеров. Учитывая, что это сайт службы доставки еды, кнопки авторизации и корзины необходимы для удобства пользователей.

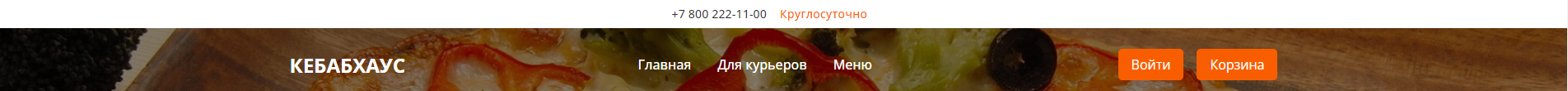


Рис.1. Header.

Затем я перешел к созданию основной секции, которую пользователь увидит сразу при заходе на сайт. Решил не разделять её и Header, а сделал так, чтобы Header "наезжал" на секцию, что, на мой взгляд, выглядит более эстетично, чем их разделять. В этом блоке сайт приветствует пользователя и предлагает заказать блюда на любой вкус. Ниже текста расположена кнопка, ведущая к основному меню с блюдами.

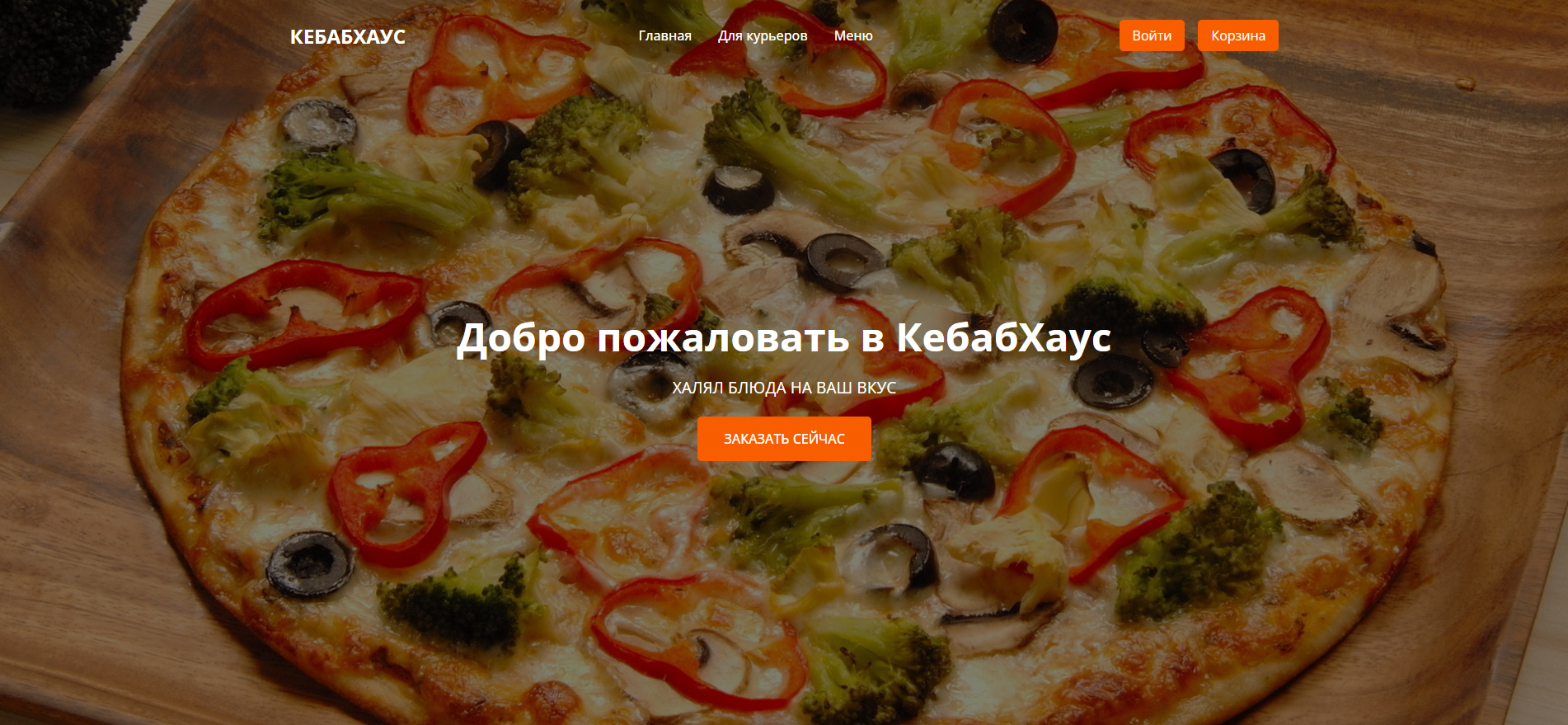


Рис. 2. Секция Home.

Ниже располагается секция с основным меню. Поскольку мы занимаемся только frontend-разработкой, здесь пока оставляем место пустым. Фактическое меню будет загружаться из базы данных в дальнейшем. Для удобства пользователей добавлены кнопки фильтрации товаров по категориям, реализованные с помощью JavaScript. Вот пример кода, который выполняет эту задачу:

function filterProducts(category) {

    const productList = document.getElementById('product-list');

    const products = productList.querySelectorAll('.product');

    products.forEach(product => {

        if (category === 'all' || product.dataset.category === category) {

            product.style.display = 'block';

        } else {

            product.style.display = 'none';

        }

    });

}

Далее идет секция с акциями, которая также остается пустой, поскольку сами акции будут загружаться из базы данных. После секции с акциями идет секция "О нас", где представлена информация о компании и другие детали.

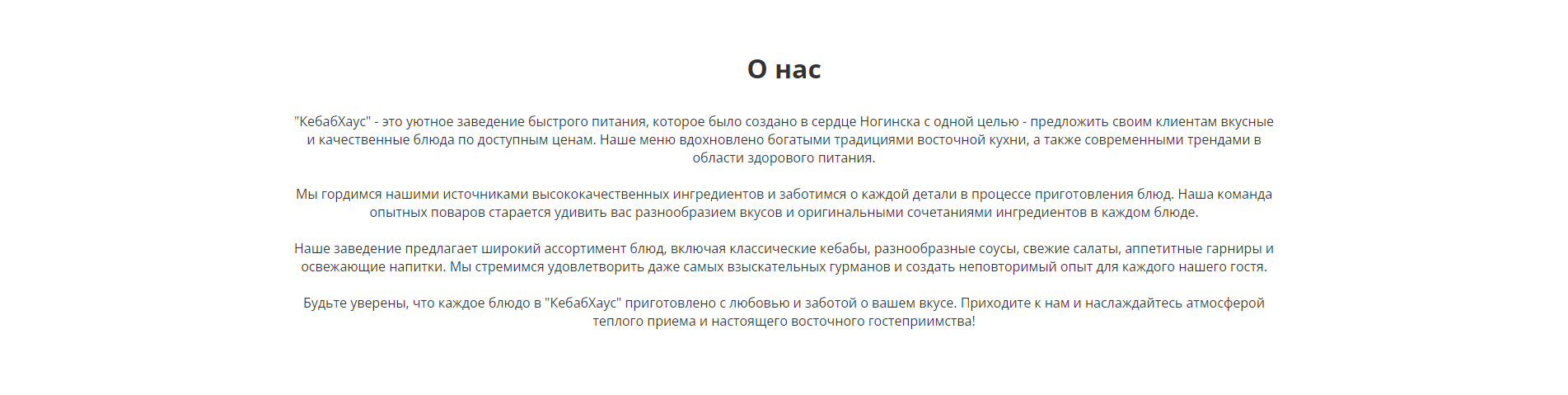


Рис. 3. Секция “About”.

В завершение главной страницы располагается Footer, который находится в самом низу страницы. Он включает в себя ссылки на социальные сети, адреса и/или юридическую информацию.

В моем веб-приложении я добавил ссылки на социальные сети, разделы "О нас" и "Помощь". Внизу страницы также разместил информацию о копирайте.

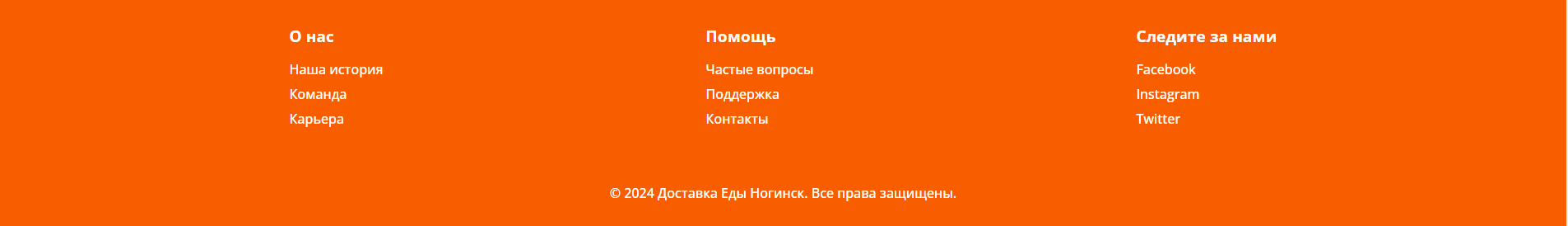


Рис. 4. Footer.

Отлично, разработка основной страницы завершена! Теперь на очереди страницы Корзины, Панели курьера, Админ Панели и Авторизации.

Разработка страницы Авторизации не была чем-то сложным. Она включает в себя форму с полями для ввода логина и пароля.

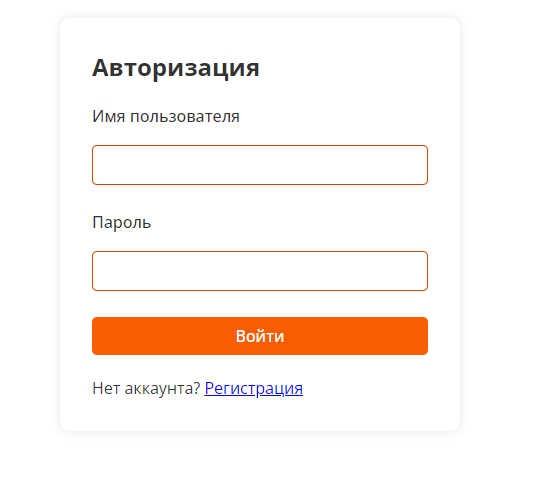


Рис. 5. Авторизация.

Страница корзины также не отличается сложностью. Она включает в себя товары, добавленные пользователем, поле для ввода промокода, который в будущем обеспечит процентную скидку, а также поля для ввода номера телефона и адреса электронной почты. Кроме того, на странице предусмотрен выбор между самовывозом и доставкой.

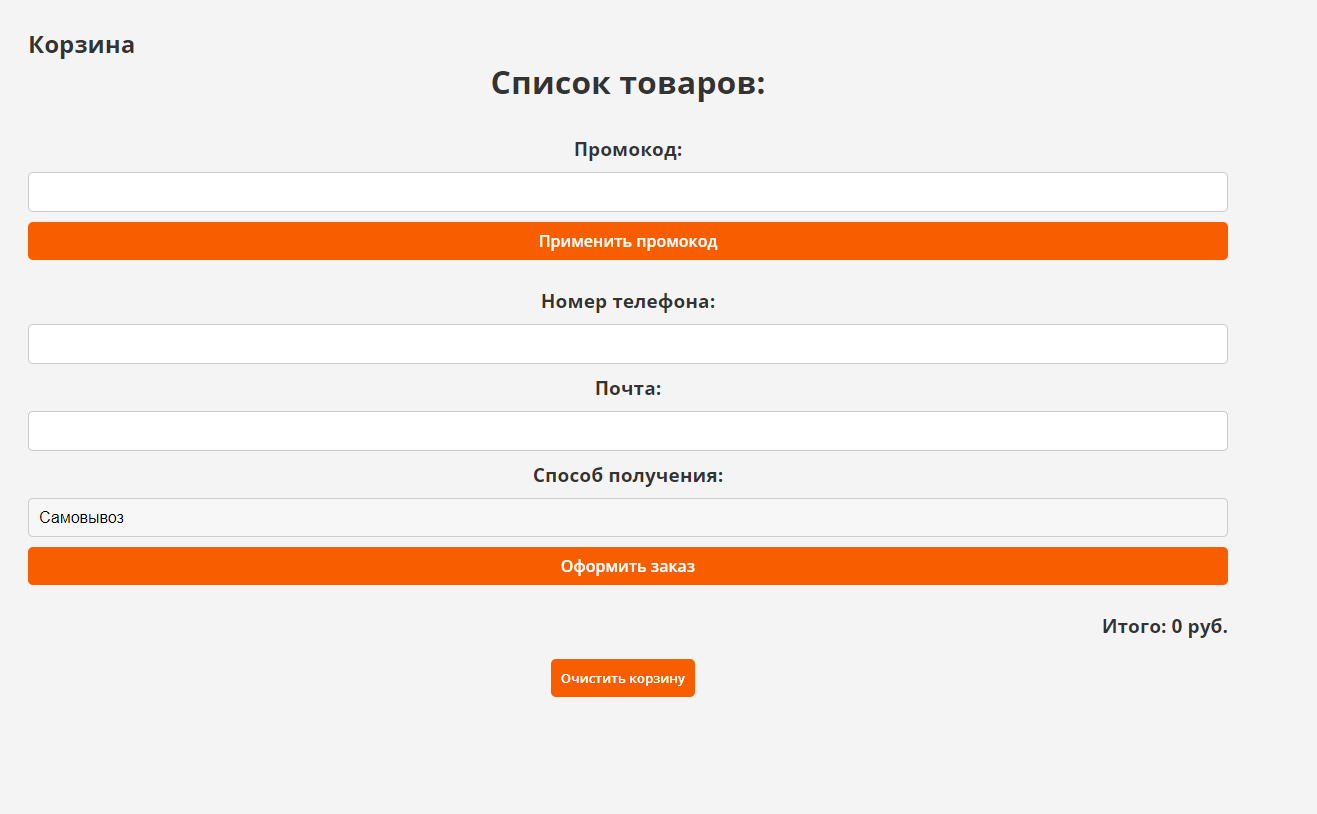


Рис. 6. Корзина.

В панели курьера представлены заказы с указанием их номеров, текущего статуса и возможностью изменить статус.

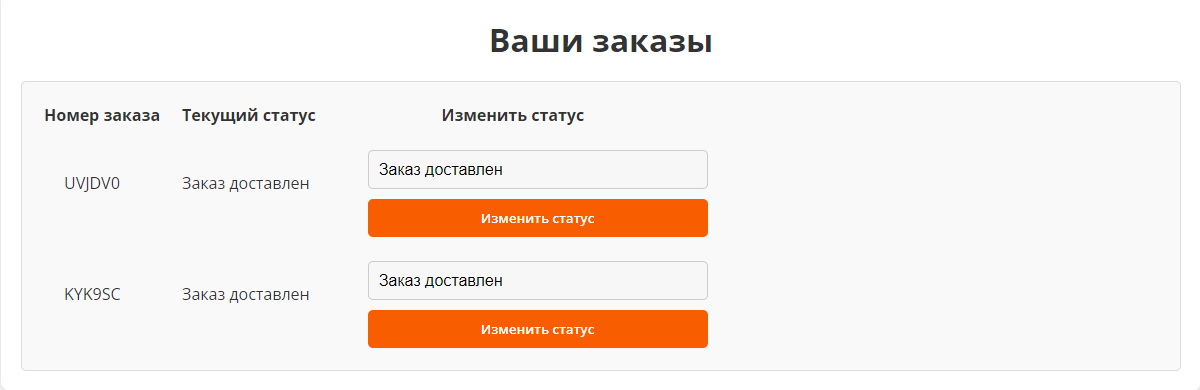


Рис 7. Панель курьера.

Самой сложной была админ-панель, как в frontend плане, так и в backend.

На панели администратора представлены различные разделы для управления акциями, промокодами, товарами, курьерами и административными правами.

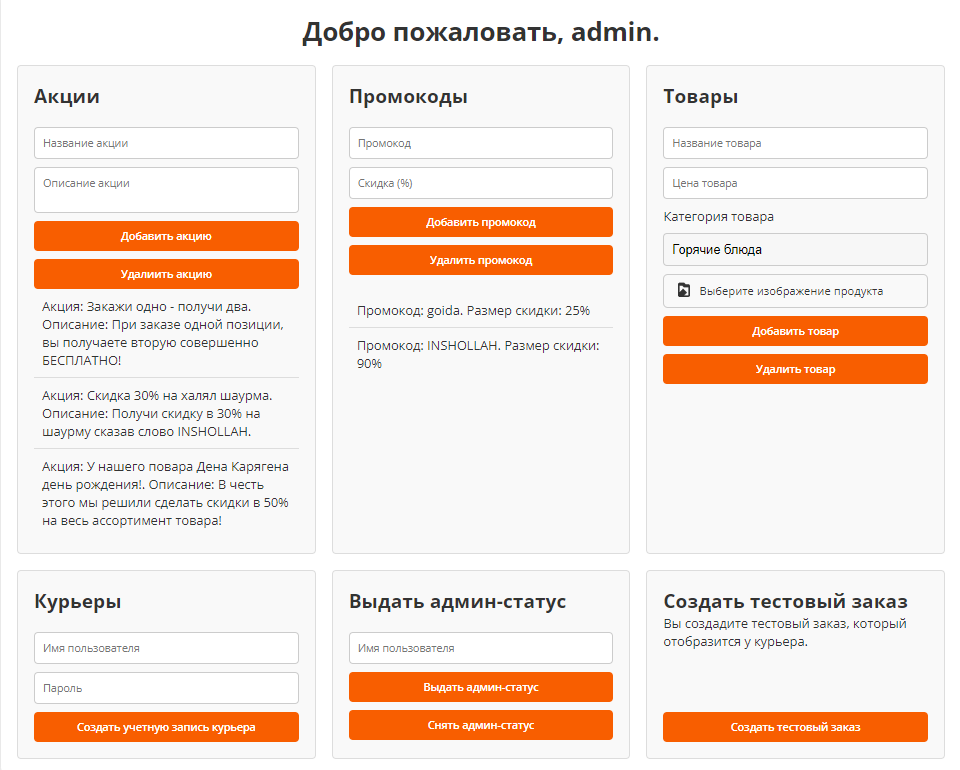


Рис. 8. Админ-панель.

Этот интерфейс позволяет администратору легко управлять всеми аспектами работы системы: создавать и удалять акции, промокоды и товары, а также управлять правами пользователей и курьеров.

## Backend часть (Flask)

Для начала работы нужно создать файлы, в котором будет писаться весь код. Основной файл запуска сервера называется app.py, в нем можно писать код единым целым, но я сделал иначе. Я разделил код на разные файлы. К примеру за страницу с админ-панелью отвечает файл admin\_routes.py, а за страницу с авторизацией отвечает auth\_routes.py.

По такому принципу был написан весь backend. Первым, чем я начал заниматься – авторизацией. Это было наверное самое простое из всего, хотя и не сказать что это просто в целом.

Для начала была создана БД, в которой была создана таблица users

        cursor.execute('''CREATE TABLE IF NOT EXISTS users (

                            id INTEGER PRIMARY KEY AUTOINCREMENT,

                            username TEXT UNIQUE NOT NULL,

                            password TEXT NOT NULL,

                            is\_admin TEXT NOT NULL,

                            is\_courier TEXT NOT NULL)''')

В таблице указывается id пользователя, его имя пользователя, пароль, статус администратора и статус курьера.

После создания таблицы, я приступил к написанию регистрации, а затем и авторизации. Код регистрации выглядит следующим образом:

@auth\_routes.route('/register', methods=['GET', 'POST'])

def register():

    if request.method == 'POST':

        username = request.form['username']

        password = request.form['password']

        hashed\_password = generate\_password\_hash(password, method='pbkdf2:sha256')

        try:

            with sqlite3.connect(Config.DATABASE) as conn:

                cursor = conn.cursor()

                cursor.execute('''INSERT INTO users (username, password, is\_admin, is\_courier) VALUES (?, ?, ?, ?)''',

                               (username, hashed\_password, "no", "no"))

                conn.commit()

                cursor.execute('SELECT id FROM users WHERE username = ?', (username,))

                user\_id = cursor.fetchone()[0]

                new\_user = User(id=user\_id, username=username, password=hashed\_password, is\_admin=False, is\_courier=False)

                login\_user(new\_user)

                return redirect(url\_for('main\_routes.index'))

        except sqlite3.IntegrityError:

            flash('Пользователь с таким именем уже существует.', 'error')

            return redirect(url\_for('auth\_routes.register'))

    return render\_template('register.html')

Для начала мы находим поля username и password в HTML. Затем генерируем хешированный пароль для обеспечения безопасности, чтобы в случае утечки данных злоумышленники не могли использовать их. После этого обращаемся к базе данных и заполняем введенные данные. Происходит автоматическая авторизация в аккаунт пользователя. Если имя пользователя уже занято, сайт выдаст ошибку о том, что такое имя пользователя уже существует.

Авторизация выглядит еще проще:

@auth\_routes.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        username = request.form['username']

        password = request.form['password']

        user = User.find\_by\_username(username)

        if user and check\_password\_hash(user.password, password):

            login\_user(user)

            return redirect(url\_for('main\_routes.index'))

        else:

            flash('Неверное имя пользователя или пароль.', 'error')

    return render\_template('login.html')

Здесь мы просто сверяем данные с тем, что записано в бд. В качестве поиска по username используется следующая функция:

    @staticmethod

    def find\_by\_username(username):

        with sqlite3.connect(Config.DATABASE) as conn:

            cursor = conn.cursor()

            cursor.execute('SELECT \* FROM users WHERE username = ?', (username,))

            user = cursor.fetchone()

            if user:

                return User(\*user)

        return None

В итоге мы получаем полностью функциональную авторизацию.

Следующим этапом моей работы стала разработка панели администратора. Сначала я создал таблицы в базе данных: для акций — таблицу sales, для тестовых заказов — таблицу test\_orders, а для промокодов — таблицу orders. Первым делом я настроил механизм присвоения админ-статуса другому пользователю. Поскольку у нас уже есть поле is\_admin, эта задача не представляет сложности: мы просто находим заданное имя пользователя и меняем его статус с no на yes.

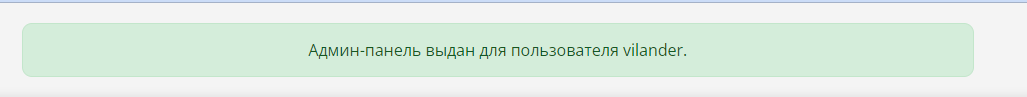


Рис. 9. Выдача админ-статуса пользователю.

Присвоение статуса курьера другому пользователю — процесс несложный. Я решил внедрить возможность регистрации нового пользователя сразу со статусом курьера. Это выполняется аналогично стандартной регистрации, но вместо "no" в статусе "is\_courier" указывается "yes". Такой пользователь может не только делать заказы, но и получить доступ к панели курьера.

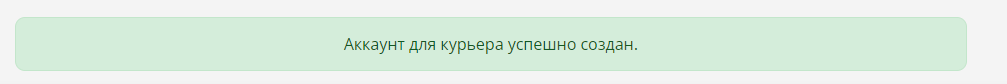
****

Рис. 10. Создание аккаунта курьера.

Система промокодов в реализации была уже труднее. Само создание промокода в админ-панели – несложное, для этого нужно просто как и с формой регистрации найти нужные поля и заполнить таблицу ими. Функция выглядит следующим образом:

def create\_delete\_promocode():

    promocode = request.form['promocode']

    discount = int(request.form['discount'])

    action = request.form['action2']

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        if action == 'create':

            cursor.execute('''INSERT INTO orders (promocode, discount) VALUES (?, ?)''', (promocode, discount))

            conn.commit()

            flash(f"Промокод {promocode} успешно создан.", 'success')

        else:

            cursor.execute('''DELETE FROM orders WHERE promocode = ? AND discount = ?''', (promocode, discount))

            conn.commit()

            if cursor.rowcount == 0:

                flash(f"Промокод {promocode} не найден.", 'error')

            else:

                flash(f"Промокод {promocode} успешно удален.", 'success')

Добавление акций работает совершенно аналогично, только вместо суммы промокода мы используем описание.

Создание тестового заказа работает через генерацию случайного набора 6 символов, для этого используется функция:

def generate\_order\_number(length=6):

    characters = string.ascii\_uppercase + string.digits

    return ''.join(random.choice(characters) for \_ in range(length))

Далее мы заполняем таблицу значением, которое у нас получилось, добавляя к этому статус заказа. Изначально заказ создается со статусом “Готовится”, но каждые 30 секунд статус автоматически изменяется на последующий:

    STATUS\_LIST = [

        "Готовится",

        "Готов, курьер спешит за ним",

        "Курьер забрал заказ и направляется к вам",

        "Заказ доставлен"

    ]

Тестовый заказ отображается в панели курьера, он может изменять ему статус вручную.

Создание товаров было одним из самых сложных задач, так как непонятно было как сделать так, чтобы фото добавлялись в бд и в последующем отображались на сайте, но я нашел выход. По итогу код добавления товаров выглядит следующим образом.

def create\_product():

    name = request.form['product\_name']

    price = request.form['product\_price']

    category = request.form['category']

    action = request.form['action4']

    file = request.files.get('product\_image')

    filename = None

    if file and allowed\_file(file.filename):

        filename = secure\_filename(file.filename)

        filepath = os.path.join(current\_app.config['UPLOAD\_FOLDER'], filename)

        file.save(filepath)

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        if action == 'create':

            cursor.execute('''INSERT INTO products (name, price, category, image) VALUES (?, ?, ?, ?)''', (name, price, category, filename))

            conn.commit()

            flash(f"Товар успешно добавлен.", 'success')

        else:

            cursor.execute('''DELETE FROM products WHERE name = ? AND price = ? AND category = ?''', (name, price, category))

            conn.commit()

            if cursor.rowcount == 0:

                flash(f"Данный товар не найден.", 'error')

            else:

                flash(f"Товар успешно удален", 'success')

        return redirect(url\_for('admin\_routes.admin\_panel'))

Вкратце, как это работает: создается отдельная папка с изображениями для товаров, название изображения сохраняется в базе данных. При отображении товара на сайте используется это название, чтобы подставить соответствующее изображение, благодаря чему фото корректно отображается на сайте.

Перейдем к панели курьера. Здесь все в разы проще, нежели с админ панелью, так как нам нужно всего лишь отобразить данные из бд и изменять статус заказа.

Для отображения заказов и изменения статуса0020мы изменяем HTML добавляя:

            <tbody>

                {% for order in orders %}

                <tr>

                    <td class="status">{{ order[1] }}</td>

                    <td class="status">{{ order[2] }}</td>

                    <td>

                        <br>

                        <form method="POST" action="{{ url\_for('courier\_routes.courier\_panel') }}">

                            <input type="hidden" name="order\_id" value="{{ order[0] }}">

                            <select name="status">

                                <option value="Готовится" {% if order[2] == 'Готовится' %}selected{% endif %}>Готовится</option>

                                <option value="Готов, курьер спешит за ним" {% if order[2] == 'Готов, курьер спешит за ним' %}selected{% endif %}>Готов, курьер спешит за ним</option>

                                <option value="Курьер забрал заказ и направляется к вам" {% if order[2] == 'Курьер забрал заказ и направляется к вам' %}selected{% endif %}>Курьер забрал заказ и направляется к вам</option>

                                <option value="Заказ доставлен" {% if order[2] == 'Заказ доставлен' %}selected{% endif %}>Заказ доставлен</option>

                            </select>

                            <button type="submit">Изменить статус</button>

                        </form>

                    </td>

                </tr>

                {% endfor %}

            </tbody>

Для изменения статуса заказа мы обращаемся к бд, заменяя статус на тот, который мы выбрали:

@courier\_routes.route('/courier', methods=['GET', 'POST'])

@courier\_required

def courier\_panel():

    if request.method == 'POST':

        order\_id = request.form['order\_id']

        new\_status = request.form['status']

        with sqlite3.connect(Config.DATABASE) as conn:

            cursor = conn.cursor()

            cursor.execute('''UPDATE test\_orders SET status = ? WHERE id = ?''', (new\_status, order\_id))

            conn.commit()

            flash('Статус заказа успешно обновлен.', 'success')

        return redirect(url\_for('courier\_routes.courier\_panel'))

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        cursor.execute('SELECT id, order\_number, status FROM test\_orders')

        orders = cursor.fetchall()

    return render\_template('couriers.html', current\_user=current\_user, orders=orders)

Курьер панель готова, осталось отобразить товары на странице, отобразить акции, доделать корзину и страницу после оплаты товара.

Для отображения товара на странице, мы редактируем HTML

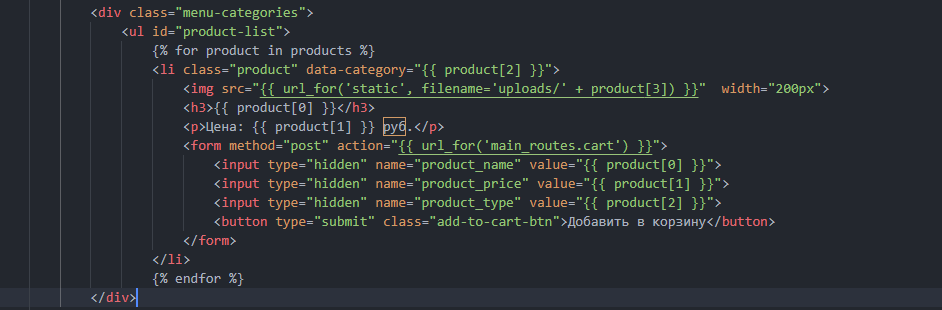


Рис.11. Новый код для отображения товаров.

Так же мы изменяем route главной странице в main\_routes.py, добавляя туда отображение товаров из БД.

@main\_routes.route('/')

def index():

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        cursor.execute('''SELECT name, price, category, image FROM products''')

        products = cursor.fetchall()

Теперь товары отображаются на странице, можно посмотреть фото, цену, название, в будущем можно будет добавить в корзину.

Для добавления товара в корзину, мы обновляем route под названием cart, добавляя туда отображение того что мы добавили в корзину.

@main\_routes.route('/cart', methods=['GET', 'POST'])

@login\_required

def cart():

    if request.method == 'POST':

        product\_name = request.form.get('product\_name')

        product\_price = request.form.get('product\_price')

        product\_type = request.form.get('product\_type')

        if 'cart' not in session:

            session['cart'] = []

        session['cart'].append({

            'name': product\_name,

            'price': float(product\_price),

            'product\_type': product\_type,

            'quantity': 1

        })

        session.modified = True

        return redirect(url\_for('main\_routes.index'))

    cart\_items = session.get('cart', [])

    total = calculate\_total(cart\_items)

    return render\_template('cart.html', cart\_items=cart\_items, total=total)

Отображение акций работает аналогично товарам, нам просто нужно изменить HTML и route главной страницы, добавив просмотр из бд таблицы с акциями.

        cursor.execute('''SELECT name, desc FROM sales''')

        sales = cursor.fetchall()

    return render\_template('index.html', products=products, sales=sales)

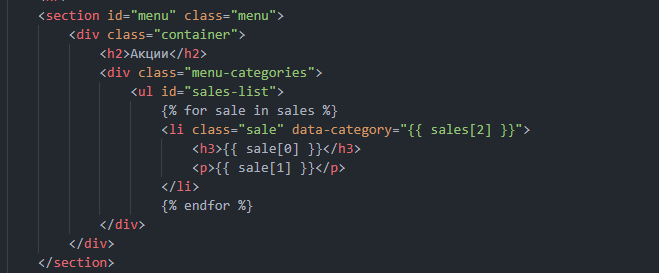


Рис.12. Отображение акций.

Главная страница полностью готова, мы видим все товары, можем их сортировать, добавлять в корзину, смотреть новые акции.

Перейдем к корзине, отображение товаров у нас уже есть, осталось до конца реализовать систему промокодов, отстук на почту пользователю и администратору, сделать изменение кол-ва товаров в корзине.

Система промокодов далась мне очень сложно, так как нужно было сделать высчет из суммы в процентах.

Расчет скидкии происходит следующим образом:

def calculate\_total(cart\_items):

    total = sum(item['price'] \* item['quantity'] for item in cart\_items)

    discount = session.get('discount', 0)

    if discount:

        total = total \* (1 - discount / 100)

    return total

Само применение промокода сделано следующим образом:

@main\_routes.route('/apply\_promo', methods=['POST'])

def apply\_promo():

    promo\_code = request.form['promo\_code']

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        cursor.execute('''SELECT discount FROM orders WHERE promocode = ?''', (promo\_code,))

        promo = cursor.fetchone()

        if promo:

            session['discount'] = promo[0]

            flash(f'Промокод {promo\_code} успешно применен', 'success')

        else:

            flash("Неверный промокод", "error")

    return redirect(url\_for('main\_routes.cart'))

Отстук на почту я делал с помощью библиотеки smtplib. Нам нужно создать новую функцию, в которой будут входить такие параметры как: smtp server, smtp port, данные для входа в аккаунт, а так же логика отправки сообщения:

def send\_email(to\_email, subject, message):

    smtp\_server = 'smtp.mail.ru'

    smtp\_port = 465

    smtp\_user = 'mrsyrkus4@mail.ru'

    smtp\_password = 'XW5JueT0bHFS0tfUa1B8'

    msg = MIMEMultipart()

    msg['From'] = smtp\_user

    msg['To'] = to\_email

    msg['Subject'] = subject

    msg.attach(MIMEText(message, 'plain'))

    try:

        server = smtplib.SMTP\_SSL(smtp\_server, smtp\_port)

        server.login(smtp\_user, smtp\_password)

        server.send\_message(msg)

        server.quit()

        print("Сообщение отправлено успешно")

    except Exception as e:

        print(f"Сообщение не отправелено: {e}")

Чтобы добавить изменение кол-ва товаров в корзине, нужно изменить HTML код, а так же добавить новый route.

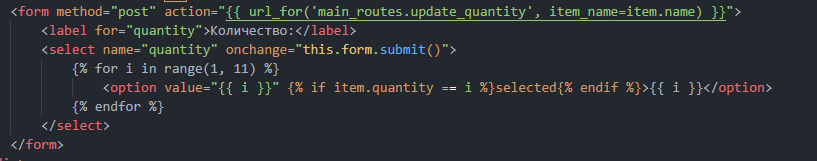


Рис. 13. Измененный HTML код.

В таком случае, route update\_quantity будет выглядить следующим образом:

@main\_routes.route('/update\_quantity/<item\_name>', methods=['POST'])

@login\_required

def update\_quantity(item\_name):

    new\_quantity = int(request.form['quantity'])

    for item in session['cart']:

        if item['name'] == item\_name:

            item['quantity'] = new\_quantity

            break

    session.modified = True

    return redirect(url\_for('main\_routes.cart'))

Здесь мы добавляем радиус от 1 до 10, а далее с помощью функции записываем количество товара в сессию.

Последнее, что нам осталось сделать – выбор доставки, а так же редирект на другую страницу, где пишется статус заказа и то, что мы заказали. Чтобы добавить выбор доставки, мы добавляем в HTML селектор, через тег <option>. Так же добавим поле с вводом адреса, и скроем его через display: none. В дальнейшем оно нам еще понадобится. Чтобы поле появлялось при выборе Доставки на дом, мы воспользуемся JavaScript. Пропишем следующий код:

function toggleAddressField() {

    var deliveryOption = document.getElementById("delivery\_option").value;

    var addressField = document.getElementById("address\_field");

    if (deliveryOption === "Delivery") {

        addressField.style.display = "block";

    } else {

        addressField.style.display = "none";

    }

}

Теперь при выборе Доставки, мы увидим поле для ввода адреса.

Теперь нужно дать применение этим полям, в нашем случае они нужны для отстука на почту пользователю и администратору. Администратору приходит информация о номере телефона клиента, его товары, самовывоз или доставка, а так же адрес, если он выбрал доставку.

Зададим переменную ADMIN\_EMAIL и укажем туда почту администратора, благодаря smtplib, мы будем отправлять на почту ADMIN\_EMAIL всю информацию о заказе.

Для отправки самых сообщений используем этот отрывок кода:

message\_for\_admin = f"Новый заказ!\n\nНомер заказа: {order\_number}\n\nНомер телефона: {phone}\n\nСпособ получения: {delivery}\n\nАдрес заказчика: {address}\n\nСписок товаров:\n"

    for item in cart\_items:

        message\_for\_admin += f"- {item['name']} ({item['price']} руб.) x {item['quantity']}\n"

    message\_for\_admin += f"\nИтого: {sum(float(item['price']) \* item['quantity'] for item in cart\_items)} руб."

    message = f"Спасибо за ваш заказ!\n\nНомер заказа: {order\_number}\n\nСписок товаров:\n"

    for item in cart\_items:

        message += f"- {item['name']} ({item['price']} руб.) x {item['quantity']}\n"

    message += f"\nИтого: {sum(float(item['price']) \* item['quantity'] for item in cart\_items)} руб."

    send\_email(email, "Ваш заказ принят", message)

    send\_email(Config.ADMIN\_EMAIL, "Новый заказ", message\_for\_admin)

Чтобы переходить на страницу с итоговым заказом, мы должны создать новый route с названием checkout. Так же в HTML пропишем путь до него, указав {{url\_for('main\_routes.checkout') }}.

Сам route выглядит следующим образом:

@main\_routes.route('/checkout', methods=['POST'])

@login\_required

def checkout():

    cart\_items = session.get('cart', [])

    email = request.form['email']

    phone = request.form['phone']

    address = request.form['address']

    delivery = request.form['delivery\_option']

    order\_number = generate\_order\_number()

    status = "Готовится"

    with sqlite3.connect(Config.DATABASE) as conn:

        cursor = conn.cursor()

        cursor.execute('''INSERT INTO test\_orders (order\_number, status) VALUES (?, ?)''',

                       (order\_number, status))

        order\_id = cursor.lastrowid

        for item in cart\_items:

            cursor.execute('''INSERT INTO order\_items (order\_id, name, price, quantity) VALUES (?, ?, ?, ?)''',

                           (order\_id, item['name'], item['price'], item['quantity']))

        conn.commit()

    message\_for\_admin = f"Новый заказ!\n\nНомер заказа: {order\_number}\n\nНомер телефона: {phone}\n\nСпособ получения: {delivery}\n\nАдрес заказчика: {address}\n\nСписок товаров:\n"

    for item in cart\_items:

        message\_for\_admin += f"- {item['name']} ({item['price']} руб.) x {item['quantity']}\n"

    message\_for\_admin += f"\nИтого: {sum(float(item['price']) \* item['quantity'] for item in cart\_items)} руб."

    message = f"Спасибо за ваш заказ!\n\nНомер заказа: {order\_number}\n\nСписок товаров:\n"

    for item in cart\_items:

        message += f"- {item['name']} ({item['price']} руб.) x {item['quantity']}\n"

    message += f"\nИтого: {sum(float(item['price']) \* item['quantity'] for item in cart\_items)} руб."

    send\_email(email, "Ваш заказ принят", message)

    send\_email(Config.ADMIN\_EMAIL, "Новый заказ", message\_for\_admin)

    session.pop('cart', None)

    session.pop('discount', None)

    return redirect(url\_for('main\_routes.view\_order', order\_id=order\_id))

Осталось только подредактировать страницу, дабы видеть то, что мы заказали, а так же видеть номер заказа.

В итоге мы получаем данный код:

    <div class="container">

        <h1>Статус заказа</h1>

        <div class="order-info">

            <p><strong>Номер заказа:</strong> {{ order\_number }}</p>

            <p><strong>Статус заказа:</strong> {{ status }}</p>

        </div>

        <h2>Вы заказали:</h2>

        <div class="cart-items">

            {% for item in cart\_items %}

            <div class="cart-item">

                <p>{{ item.name }}</p>

                <p>{{ item.price }} руб.</p>

            </div>

            {% endfor %}

        </div>

    </div>

Backend составляющая полностью готова. Осталось загрузить файлы сайта на сервер, купить для него домен, прикрутить его к серверу и запустить сам сайт на хостинге.

Домен был куплен на сервисе REG.RU, соответственно настройка происходит там. Купленный домен – http://kvmrnn.xyz.

С помощью бесплатного хостинга Beget я не смог запустить сайт, скорее всего этот фреймворк не поддерживается на данном хостинге. Поэтому я был вынужден оплатить сервер на хостинге <https://madhost.pw/>.

И так, по SFTP я перекинул файлы на сервер, далее начал заниматься установкой всех зависимостей на сервер. Установил NGINX (sudo apt install python3-pip python3-venv nginx), gunicorn (sudo python3 -m pip install gunicorn), создал и запустил виртуальное пространство venv, так же создал systemd daemon, отвечающий за работу python в фоне с авторестратом. После чего я сконфигурировал nginx.confg, куда прописал реверс прокси порта gunicorn на порт 80 и наш домен.

Теперь мы можем зайти на наш сайт и увидеть, что все полностью работает и функционирует как нужно.

## Заключение

В ходе выполнения курсового проекта была разработана система для управления заказами и корзиной в рамках веб-приложения. Основной целью данного проекта было создание удобного и функционального интерфейса для пользователей, позволяющего им просматривать и изменять свои заказы, а также управлять содержимым корзины.

Решение поставленных задач

1. **Разработка структуры базы данных**:
   * Были созданы необходимые таблицы базы данных: users, orders, order\_items, products, test\_orders, sales. Эти таблицы обеспечивают хранение информации о пользователях, заказах, товарах и их состояниях.
2. **Реализация функциональности корзины**:
   * Была реализована возможность добавления товаров в корзину, отображение списка товаров в корзине с указанием их количества и цены.
   * Добавлена функциональность изменения количества товаров в корзине, что обеспечивает гибкость и удобство для пользователей при оформлении заказов.
3. **Управление статусами заказов**:
   * Созданы механизмы для отображения и изменения статусов заказов как для пользователей, так и для курьеров. Курьеры могут изменять статус заказа, что позволяет пользователям отслеживать текущий статус своего заказа в реальном времени.
4. **Интерфейс и взаимодействие с пользователем**:
   * Разработан удобный интерфейс, обеспечивающий легкий доступ к информации о заказах и товарах. Внедрение шаблонов HTML и CSS позволило создать современный и интуитивно понятный интерфейс.
   * Реализована система применения промокодов и расчета итоговой суммы заказа с учетом скидок, что добавляет дополнительные возможности для пользователей.

В результате выполнения данного курсового проекта была достигнута основная цель – разработка системы управления заказами и корзиной в веб-приложении. Все поставленные задачи были успешно выполнены.

Разработанная система успешно обеспечивает удобство и функциональность для пользователей, позволяя им легко управлять своими заказами и корзиной. В ходе выполнения проекта были применены современные подходы и технологии, что гарантирует надежность и масштабируемость системы.

Таким образом, все цели, поставленные в рамках данной курсовой работы, были достигнуты. Полученные результаты демонстрируют успешное выполнение проекта и подтверждают возможность дальнейшего расширения и улучшения системы в будущем.