Титульный лист

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 Node.JS В РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЯ «Служба доставки еды «ДодоПтица»

Разработчик: студент группы 3ИСПР1

Булаткин Д.А.

(подпись)

Оценка защиты курсового проекта

« »

Дата защиты

« » 2024

Руководитель проекта:

преподаватель Степанов С.О.

(подпись)

Ногинск, 2024 г

Содержание

[ВВЕДЕНИЕ 3](#_Toc169786706)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc169786707)

[1.общие сведения о разработке веб-приложения на фреймворке 4](#_Toc169786708)

[1.1Основные функции и требования 4](#_Toc169786709)

[1.2Архитектура приложения 5](#_Toc169786710)

[2.Инсталляция программного обеспечения 7](#_Toc169786711)

[2.1. Установка Node.js 7](#_Toc169786712)

[2.2. Установка Nodemon 7](#_Toc169786713)

[2.3. Создание репозитория на GitHub 7](#_Toc169786714)

[3.Руководство программиста. 9](#_Toc169786715)

[3.1. Написание клиентской части с помощью React. 9](#_Toc169786716)

[3.2.Написание серверной части с помощью Node.JS. 19](#_Toc169786717)

[3.3Структура админ панели 25](#_Toc169786718)

[Заключение 28](#_Toc169786719)

[Список источников и документации 31](#_Toc169786720)

[ПРИЛОЖЕНИЯ 32](#_Toc169786721)

[Приложение 1: Итоговый вид веб-приложения 33](#_Toc169786722)

# ВВЕДЕНИЕ

Рынок онлайн-заказа еды стремительно развивается, обуславливая рост спроса на веб-приложения для удобного заказа еды из ресторанов. Одним из популярных фреймворков для разработки таких приложений является Node.js, обладающий высокой производительностью, масштабируемостью и удобством использования.

Цель курсовой работы: разработка веб-приложения «Служба доставки еды «ДодоПтица»» с использованием фреймворка Node.js.

Задачи курсовой работы:

Изучение теоретических основ фреймворка Node.js.

Разработка архитектуры веб-приложения.

Реализация функционала: Регистрация и авторизация пользователей. Просмотр меню ресторана. Оформление заказа. Отслеживание статуса заказа. Личный кабинет пользователя. Тестирование и устранение ошибок.

Область применения: Заказ еды из ресторанов. Повышение удобства обслуживания клиентов. Увеличение прибыли ресторанов.

Современное состояние систем на рынке:

Существует множество веб-приложений для заказа еды (Яндекс.Еда, Delivery Club, SberFood, VK Еда). Веб-приложение «ДодоПтица» должно обладать конкурентными преимуществами.

Ожидаемые результаты:

Разработка веб-приложения «ДодоПтица», соответствующего современным требованиям.

# ОСНОВНАЯ ЧАСТЬ

## 1.общие сведения о разработке веб-приложения на фреймворке

Для реализации веб-приложения «ДодоПтица» был выбран следующий стек технологий:

* **Node.js**: Серверная платформа, основанная на движке JavaScript V8, обеспечивающая высокую производительность и масштабируемость. Node.js был выбран благодаря его асинхронной природе и возможности использовать JavaScript как на стороне сервера, так и на стороне клиента.
* **Express.js**: Легкий и гибкий фреймворк для Node.js, используемый для построения серверной части приложения. Express.js обеспечивает простой и интуитивно понятный интерфейс для создания маршрутов и обработки HTTP-запросов.
* **React.js**: Библиотека JavaScript для построения пользовательских интерфейсов. React.js позволяет создавать динамичные и отзывчивые интерфейсы, которые обеспечивают удобство и простоту использования для конечных пользователей.
* **MongoDB**: Документо-ориентированная база данных, идеально подходящая для хранения данных веб-приложений благодаря своей гибкости и масштабируемости. MongoDB обеспечивает эффективное хранение и доступ к данным, что особенно важно для приложений с большим количеством запросов.
* **Stripe**: Платежный сервис, интегрированный в приложение для обеспечения безопасных и удобных платежей. Stripe предоставляет надежные API для обработки транзакций, что делает его отличным выбором для веб-приложений электронной коммерции.

### 1.1Основные функции и требования

Разработка веб-приложения «ДодоПтица» включала в себя реализацию ряда функциональных требований, обеспечивающих полный цикл обслуживания пользователей и управления контентом. Основные требования к сайту включают:

1. **Авторизация и регистрация**: Пользователи должны иметь возможность создавать учетные записи и входить в систему. Для этого была реализована система авторизации и регистрации с использованием JSON Web Tokens (JWT) для обеспечения безопасного хранения и передачи данных аутентификации.
2. **Ежедневные и еженедельные акции**: В системе предусмотрено управление акциями, которые могут обновляться ежедневно и еженедельно. Эти акции отображаются на главной странице и привлекают внимание пользователей, предлагая специальные предложения и скидки.
3. **Каталог товаров**: Каталог включает не менее 40 товаров, сгруппированных по категориям. Пользователи могут просматривать товары, фильтровать их по категориям.
4. **Корзина товаров**: Пользователи могут добавлять товары в корзину, изменять их количество или удалять ненужные позиции. Корзина сохраняется на протяжении сеанса пользователя и готова к оформлению заказа в любой момент.
5. **Промокоды на скидку**: Возможность применения промокодов для получения скидок на заказ. Промокоды могут быть введены на этапе оформления заказа, и система автоматически пересчитывает итоговую сумму с учетом скидки.
6. **Выбор способа доставки**: Пользователи могут выбрать между доставкой по указанному адресу или самовывозом из пункта выдачи. Эта опция позволяет гибко адаптировать услуги под предпочтения клиентов.
7. **Уведомления по электронной почте**: после оформления заказа пользователю и администратору отправляется электронное письмо с подробной информацией о заказе. Это обеспечивает удобство для клиента и помогает администратору своевременно обрабатывать заказы.
8. **Отслеживание статуса заказа**: Пользователи могут отслеживать текущий статус своего заказа через интерфейс приложения. Заказы проходят через несколько этапов: оформлен, принят, доставляется и завершен.
9. **Панель администратора**: Административная панель предоставляет доступ к функциям управления контентом и заказами. Администраторы могут добавлять и удалять акции, промокоды, товары, а также просматривать и изменять статус заказов. Также предусмотрена возможность создания учетных записей для курьеров.

### 1.2Архитектура приложения

Веб-приложение «ДодоПтица» разработано на основе архитектуры клиент-сервер.

* **Клиентская часть** реализована с использованием React.js и взаимодействует с сервером через RESTful API. Компоненты React обеспечивают динамическое обновление пользовательского интерфейса без необходимости перезагрузки страницы.
* **Серверная часть** построена на базе Node.js с использованием фреймворка Express.js. Сервер обрабатывает запросы от клиента, взаимодействует с базой данных MongoDB и управляет платежами через интеграцию с Stripe.

## 2.Инсталляция программного обеспечения

### 2.1. **Установка Node.js**

Скачайте установщик Node.js с официального сайта https://nodejs.org/en/download/package-manager для вашей операционной системы (Windows, macOS, Linux) и следуйте инструкциям мастера установки.

Проверьте установку, открыв командную строку (Windows) или терминал (macOS/Linux) и введя команды node -v и npm -v. Должны отобразиться версии Node.js и npm соответственно.

### 2.2. Установка Nodemon

Откройте командную строку/терминал, перейдите в каталог вашего проекта и выполните команду npm install nodemon --save-dev.

Проверьте установку командой nodemon -v. Должна отобразиться версия Nodemon.

### 2.3. Создание репозитория на GitHub

Зарегистрируйтесь на GitHub на https://github.com/ и создайте новый репозиторий, например, kurs. В командной строке/терминале перейдите в каталог проекта и выполните команду git init.Скопируйте URL-адрес вашего репозитория GitHub и выполните команду git remote add origin <URL-адрес-репозитория>.Введите команду git add . для добавления всех файлов проекта в отслеживание. Введите команду git commit -m "Первый коммит".Введите команду git push -u origin master. Скачайте и установите GitHub Desktop с сайта https://desktop.github.com/ для вашей операционной системы.Подключитесь к своей учетной записи GitHub в приложении. Клонируйте репозиторий: Нажмите кнопку "Клонировать репозиторий", введите URL-адрес вашего репозитория, выберите папку для клонирования и нажмите "Клонировать".Работайте с локальным репозиторием:

В GitHub Desktop вы можете отслеживать изменения, создавать коммиты и отправлять их в удаленный репозиторий, а также использовать дополнительные функции. В GitHub Desktop вы можете отслеживать изменения, создавать коммиты и отправлять их в удаленный репозиторий, а также использовать дополнительные функции.

## 3.Руководство программиста.

### 3.1. Написание клиентской части с помощью React.

После инициализации проекта:

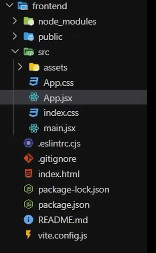


Рисунок 1

Далее создадим папки Pages и Components. Сначала я создал папку NavBar внутри папки Pages. Это помогло организовать компоненты, относящиеся к навигационной панели, в одном месте.

Я создал файл NavBar.jsx в папке NavBar и написал следующий код:

import React, { useContext, useState } from 'react'

import './Navbar.css'

import {assets} from '../../assets/assets'

import {Link, useNavigate} from 'react-router-dom'

import { StoreContext } from '../../Context/StoreContext'

const Navbar = ({setShowLogin}) => {

    const [menu,setMenu]=useState("home");

    const {getTotalCartAmount,token,setToken}=useContext(StoreContext);

    const navigate = useNavigate();

    const logout =()=>{

        localStorage.removeItem("token");

        setToken("");

        navigate("/");

    }

    return (

        <div className='navbar'>

        <Link to='/'><img src={assets.logo} alt='' className='logo'/></Link>

        <ul className='navbar-menu'>

            <Link to='/' onClick={()=>setMenu("home")} className={menu==="home"?"active":""}>Домой</Link>

            <a href='#explore-menu' onClick={()=>setMenu("menu")} className={menu==="menu"?"active":""}>Меню</a>

            <a href='#footer' onClick={()=>setMenu("contact-us")} className={menu==="contact-us"?"active":""}>Обратная связь</a>

        </ul>

        <div className="navbar-right">

            <img src={assets.search\_icon} alt="" />

            <div className="navbar-search-icon">

                <Link to='/cart'><img src={assets.basket\_icon} alt="" /></Link>

                <div className={getTotalCartAmount()===0?"":"dot"}></div>

            </div>

            {!token?<button onClick={()=>setShowLogin(true)} >Войти</button>

            : <div className="navbar-profile">

                <img src={assets.profile\_icon} alt=''></img>

                <ul className="nav-profile-dropdown">

                    <li onClick={()=>navigate('/myorders')}><img src={assets.bag\_icon} alt="" /><p>Мои заказы</p></li>

                    <hr />

                    <li onClick={logout}><img src={assets.logout\_icon} alt="" /><p>Выйти</p></li>

                </ul>

            </div>

            }

        </div>

    </div>

  )

}

export default Navbar

Этот код создаёт компонент NavBar с несколькими функциональными элементами:

Логотип с ссылкой на главную страницу.

Меню навигации, включающее ссылки "Домой", "Меню" и "Обратная связь".

Иконка поиска и корзина, которая отображает значок, если в корзине есть товары.

Кнопка входа или профиль пользователя с выпадающим меню, включающим "Мои заказы" и "Выйти", если пользователь авторизован.

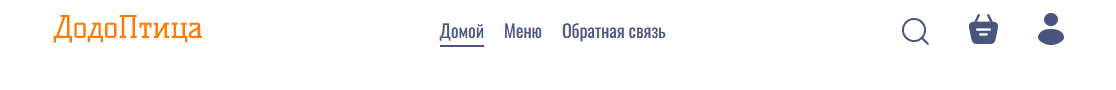


Рисунок 2

Для управления состоянием навигации и авторизации использовал React Hooks: useState для отслеживания текущего выбранного пункта меню и useContext для доступа к глобальному состоянию контекста.

Создал компонент Cart, который отображает содержимое корзины покупок, позволяет пользователю применять купоны и переходить к оформлению заказа. Использовал:

useContext для получения данных о корзине из глобального контекста.

useState для управления состояниями, такими как код купона и скидка.

axios для отправки запросов на сервер.

Основные функции:

Отображение списка товаров в корзине.

Расчет итоговой стоимости товаров с учётом применённой скидки.

Применение купонов на скидку и обработка ошибок.

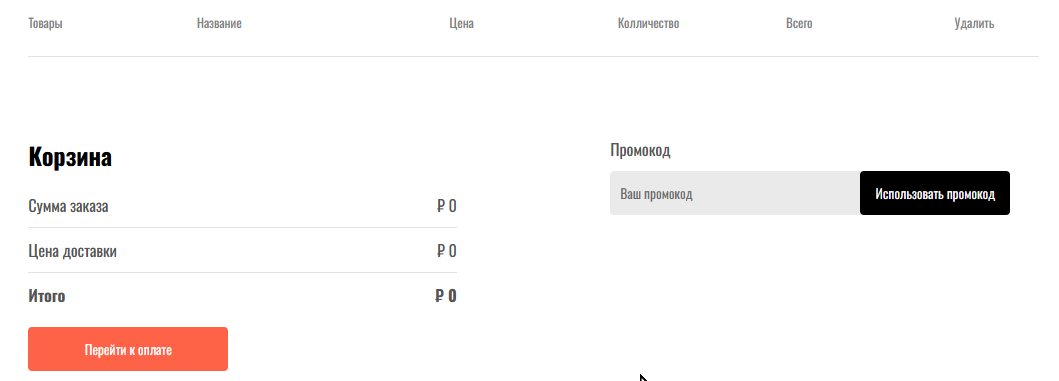


Рисунок 3

Создал компонент PlaceOrder, который позволяет пользователю ввести данные для доставки и завершить процесс покупки. Использовал:

useState и useEffect для управления состояниями формы и применения скидки.

useContext для получения данных о корзине и токене пользователя.

axios для отправки данных заказа на сервер.

Основные функции:

Обработка формы с данными о доставке.

Расчет итоговой суммы заказа с учётом скидки и стоимости доставки.

Отправка данных заказа на сервер и редирект пользователя на страницу подтверждения оплаты. Создал компонент MyOrders, который отображает список всех заказов пользователя. Использовал:

useContext для доступа к токену пользователя и URL API.

axios для получения данных о заказах с сервера.

useEffect для загрузки данных при монтировании компонента.

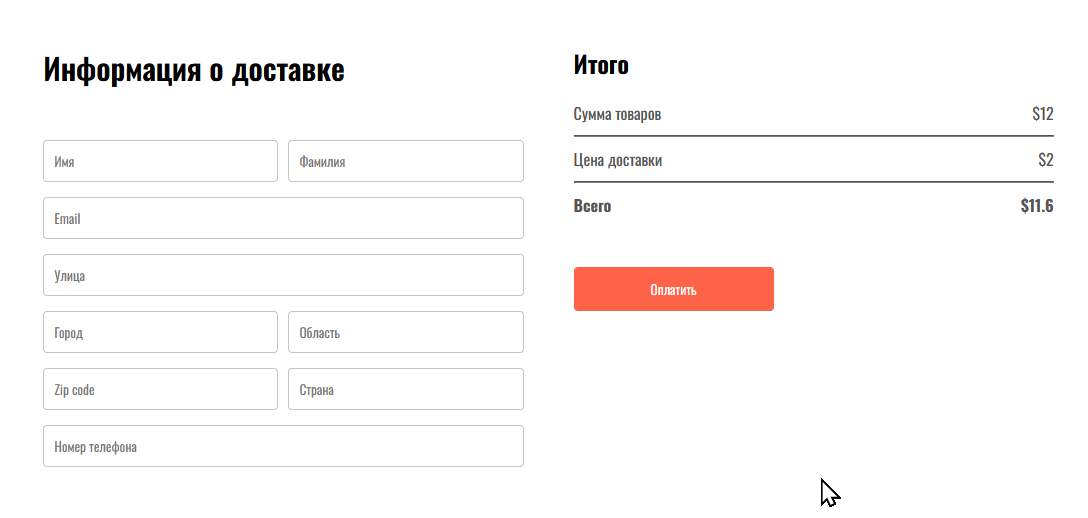


Рисунок 4

Создал компонент MyOrders, который отображает список всех заказов пользователя. Использовал:

Основные функции:

useContext для доступа к токену пользователя и URL API.

axios для получения данных о заказах с сервера.

useEffect для загрузки данных при монтировании компонента.

Запрос и отображение данных о заказах пользователя.

Обновление списка заказов по нажатию кнопки.

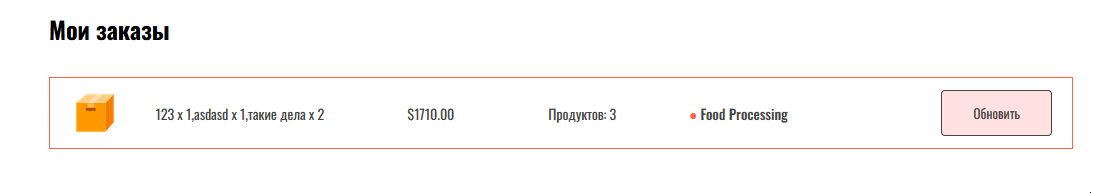


Рисунок 5

Создал компонент Home, который является основным входом на сайт и включает в себя несколько дочерних компонентов, таких как заголовок, меню и отображение пищи.

Вот так выглядит его код:

import React, { useState } from 'react'

import './Home.css'

import Header from '../../components/Header/Header';

import ExploreMenu from '../../components/ExploreMenu/ExploreMenu';

import FoodDisplay from '../../components/FoodDisplay/FoodDisplay';

import DailySales from '../../components/DailySales/DailySales';

const Home = () => {

  const [category,setCategory]=useState("All");

  return (

    <div>

      <Header/>

      <DailySales />

      <ExploreMenu category={category} setCategory={setCategory}/>

      <FoodDisplay category={category}/>

    </div>

  )

}

export default Home;

Основные функции:

Отображение заголовка и приветственного текста.

Показ доступных категорий меню и блюд.

Компоненты Header, ExploreMenu, FoodDisplay, и DailySales обеспечивают функциональность главной страницы.

Создал компонент Verify, который обрабатывает верификацию платежа и перенаправляет пользователя в зависимости от результата.

Он нужен для проверки статуса платежа с помощью параметров URL и перенаправления пользователя в зависимости от результата верификации.

алее идёт создание компонентов для отображения информации на экране пользователя и обработку его действий.

LoginPopup.jsx:

Этот компонент отвечает за всплывающее окно входа и регистрации. Он использует React-хуки useState для управления текущим состоянием ("Sign-Up" или "Login") и данными пользователя (имя, email, пароль). Пользователь может переключаться между регистрацией и входом. При отправке формы компонент отправляет запрос на сервер с помощью axios для регистрации или входа пользователя. В зависимости от ответа сервера компонент либо сохраняет токен авторизации и скрывает окно, либо отображает сообщение об ошибке.

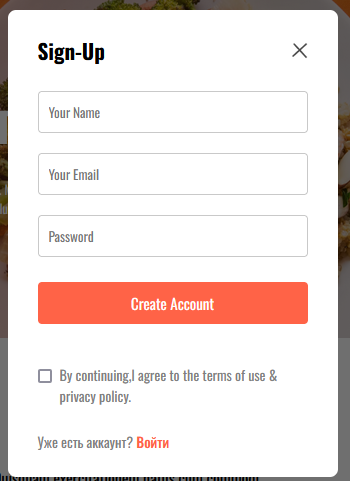


Рисунок 6

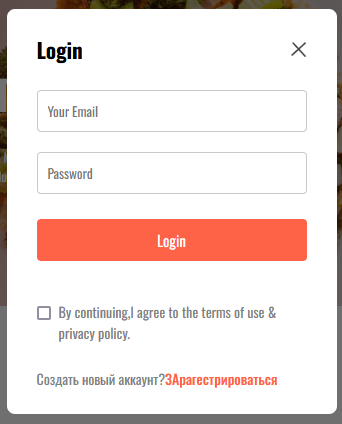


Рисунок 7

Header.jsx:

Этот компонент отображает заголовок сайта, короткое описание сервиса и кнопку "Меню". Он использует базовый HTML и стилизуется с помощью CSS-классов.

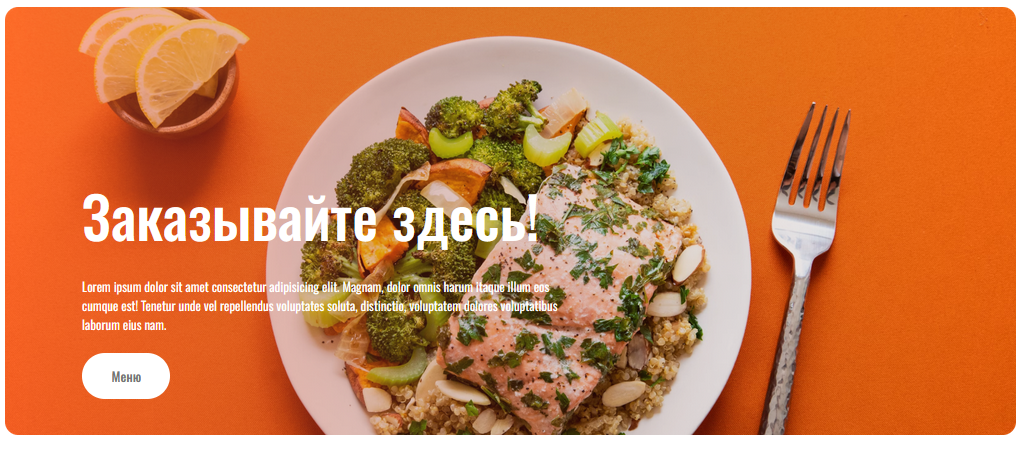


Рисунок 8

Footer.jsx:

Этот компонент отображает нижний колонтитул сайта, включающий логотип, информацию о компании, ссылки на социальные сети, контактные данные и уведомление об авторских правах. Он использует импортированные изображения и стилизуется с помощью CSS-классов.

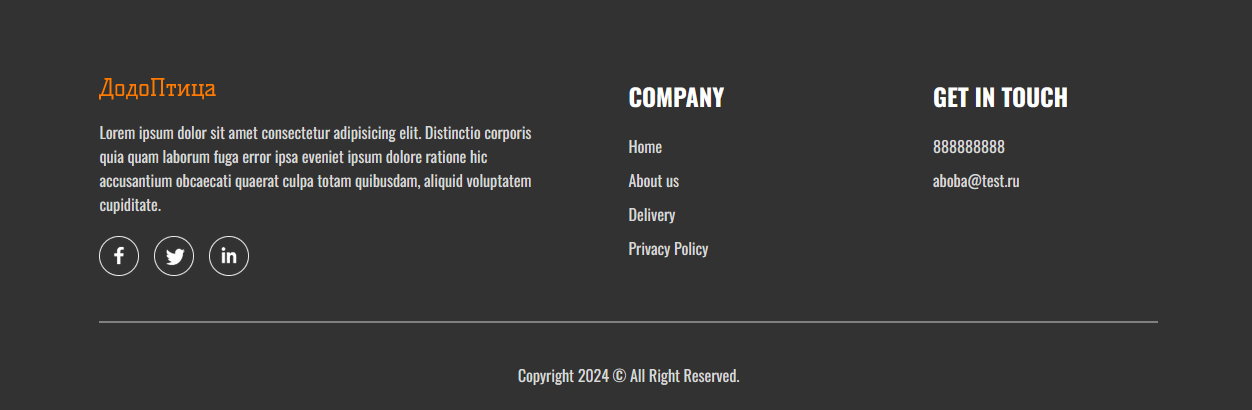


Рисунок 9

FoodItem.jsx:

Этот компонент отображает отдельный элемент меню. Он получает информацию об элементе (id, название, цена, описание, изображение) в качестве props. Компонент использует StoreContext для доступа к данным корзины (элементы и функции добавления/удаления) и отображает либо значок "добавить" либо счетчик количества и кнопки "добавить/удалить" в зависимости от наличия элемента в корзине



Рисунок 10

FoodDisplay.jsx:

Этот компонент отображает список элементов меню на основе выбранной категории. Он получает категорию в качестве props и использует StoreContext для доступа к полному списку меню. Затем компонент фильтрует список по категории и отображает каждый элемент с помощью компонента FoodItem.

ExploreMenu.jsx:

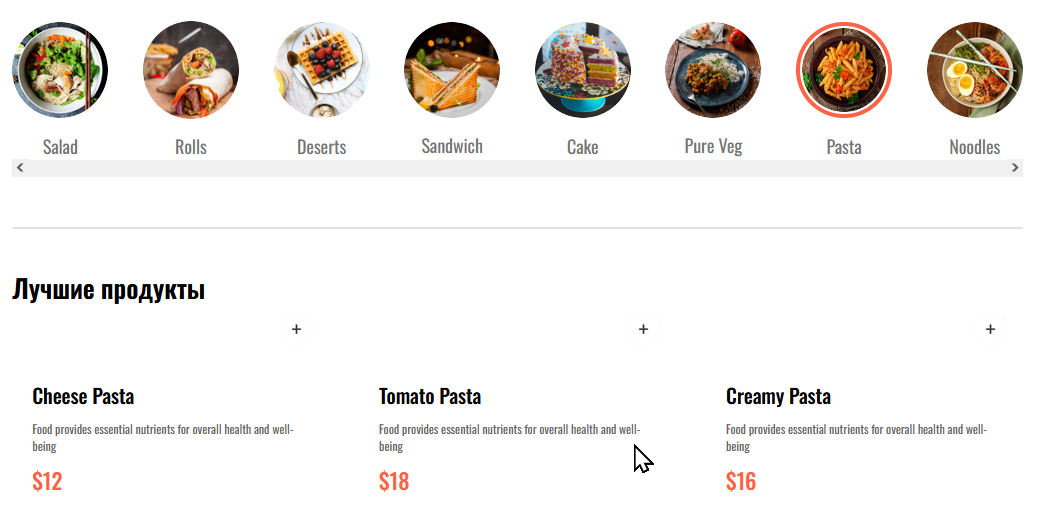
Этот компонент отображает раздел изучения меню с описанием и кнопками выбора категорий. Он получает текущую выбранную категорию и функцию для ее обновления. Компонент отображает список категорий, полученных из отдельного файла assets, и визуально выделяет выбранную категорию. 

Рисунок 11

DailySales.jsx:

Этот компонент отображает список ежедневных скидок. Он использует хуки useState для управления состоянием данных, загрузки и ошибки. При монтировании компонента выполняется запрос на сервер с помощью axios для получения информации о скидках. В зависимости от полученных данных компонент отображает либо сообщение о загрузке, либо сообщение об ошибке, либо список доступных скидок (название товара, количество, цена).

Это код app.jsx который запускается при помощи прописывания npm run dev в консоли:

import React , { useState }from 'react'

import Navbar from './pages/Navbar/Navbar';

import { Route, Routes } from 'react-router-dom';

import Home from './pages/Home/Home'

import Cart from './pages/Cart/Cart'

import PlaceOrder from './pages/PlaceOrder/PlaceOrder';

import Footer from './components/Footer/Footer';

import LoginPopup from'./components/LoginPopup/LoginPopup';

import Verify from './pages/Verify/Verify';

import MyOrders from './pages/MyOrders/MyOrders'

const App = () => {

  const [showLogin,setShowLogin]=useState(false)

  return (

    <>

    {showLogin?<LoginPopup setShowLogin={setShowLogin}/>:<></>}

    <div className='app'>

      <Navbar setShowLogin={setShowLogin}/>

      <Routes>

        <Route path='/' element={<Home/>}/>

        <Route path='/cart' element={<Cart/>}/>

        <Route path='/order' element={<PlaceOrder/>}/>

        <Route path='/verify' element={<Verify />} />

        <Route path='/myorders' element={<MyOrders />} />

      </Routes>

    </div>

    <Footer/>

    </>

  )

}

export default App;

### 3.2.Написание серверной части с помощью Node.JS.

Структура проекта:

Для организации кода проекта я использовал следующую структуру:

server.js: Главный файл приложения. Здесь происходит инициализация приложения Express, подключение необходимых библиотек, настройка роутов и запуск сервера.

Config/db.js: Файл конфигурации, где хранятся параметры подключения к базе данных и секретный ключ для JWT-авторизации.

controllers/: Папка для хранения функций-контроллеров, обрабатывающих запросы пользователей к определенным ресурсам (например, пользователям, заказам).

middleware/: Папка для хранения функций-посредников, выполняющих действия перед обработкой запроса контроллером (например, проверка авторизации).

models/: Папка для хранения моделей данных, представляющих структуру документов в базе данных (например, модель пользователя).

routes/: Папка для хранения роутов, которые связывают HTTP-методы (GET, POST, PUT, DELETE) с определенными функциями-контроллерами.

package.json: Файл с описанием проекта, его зависимостями и скриптами.

uploads/:Папка для хранения изображений товаров загруженных через админ панель.

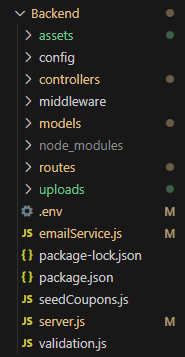


Рисунок 12

Зависимости проекта:

Для реализации функциональности серверной части сайта мы использовали следующие библиотеки:

Express: Фреймворк для создания веб-приложений на Node.js.

cors: Библиотека для обработки кросс-доменных запросов (CORS).

mongoose: Библиотека для работы с базой данных MongoDB.

dotenv: Библиотека для загрузки переменных окружения из файла .env.

jsonwebtoken (JWT): Библиотека для создания и проверки JSON Web Tokens (JWT) для авторизации пользователей.

bcrypt: Библиотека для хеширования паролей пользователей.

validator: Библиотека для валидации данных (например, email, пароль).

Stripe: Библиотека для интеграции с платежным шлюзом Stripe (используется для обработки платежей за заказы).

nodemailer: Библиотека для отправки электронных писем (используется для отправки подтверждений заказов пользователям и администратору).

Теперь я распишу за что отвечает каждый файл в папке backend.

Server.js

Этот файл является главным входным файлом для сервера. Здесь происходит настройка Express.js приложения, подключение к базе данных MongoDB с помощью Mongoose, импорт различных маршрутов и промежуточного программного обеспечения (middleware). Сервер прослушивает порт 4000 и обрабатывает входящие HTTP-запросы, направляя их соответствующим маршрутам.

config/db.js

Этот файл содержит функцию connectDB(), которая устанавливает соединение с базой данных MongoDB с использованием Mongoose. Строка подключения берется из переменной окружения.

controllers/userController.js

Этот файл содержит функции для обработки операций, связанных с пользователями, таких как регистрация, аутентификация и создание JSON Web Tokens (JWT) для аутентификации. Он использует библиотеки bcrypt для хеширования паролей, jsonwebtoken для создания JWT и validator для проверки электронной почты.

middleware/auth.js

Этот файл содержит промежуточное программное обеспечение authMiddleware, которое проверяет JWT в заголовках входящих запросов. Если токен действителен, он декодирует идентификатор пользователя и добавляет его в тело запроса для дальнейшей обработки.

models/userModel.js

Этот файл определяет схему Mongoose для модели пользователя, которая включает имя, электронную почту, пароль, роль и данные корзины. Он также создает модель Mongoose на основе этой схемы.

models/couponModel.js

Этот файл определяет схему Mongoose для модели купона. Схема содержит следующие поля:

code: строка, обязательное поле, уникальное значение для кода купона.

discount: число, обязательное поле, представляющее размер скидки в процентах.

expiryDate: дата, обязательное поле, указывающее срок действия купона.

active: логическое значение, по умолчанию true, указывающее, активен ли купон.

На основе этой схемы создается модель Mongoose Coupon, которая затем экспортируется для использования в других частях приложения.

models/foodModel.js

Этот файл определяет схему Mongoose для модели продукта питания. Схема содержит следующие поля:

name: строка, обязательное поле для названия продукта.

description: строка, обязательное поле для описания продукта.

price: число, обязательное поле для цены продукта.

image: строка, обязательное поле для пути к изображению продукта.

category: строка, обязательное поле для категории продукта.

На основе этой схемы создается модель Mongoose foodModel, которая затем экспортируется для использования в других частях приложения.

models/orderModel.js

Этот файл определяет схему Mongoose для модели заказа. Схема содержит следующие поля:

userId: строка, обязательное поле для идентификатора пользователя, сделавшего заказ.

items: массив, обязательное поле для списка заказанных продуктов.

amount: число, обязательное поле для общей суммы заказа.

address: объект, обязательное поле для адреса доставки.

status: строка, по умолчанию "Food Processing", указывающая текущий статус заказа.

date: дата, по умолчанию текущая дата, указывающая дату размещения заказа.

payment: логическое значение, по умолчанию false, указывающее, был ли заказ оплачен.

На основе этой схемы создается модель Mongoose orderModel, которая затем экспортируется для использования в других частях приложения.

models/saleModel.js

Этот файл определяет схему Mongoose для модели акции на продукты. Схема состоит из двух вложенных схем:

saleItemSchema: схема для отдельного продукта в акции, содержащая следующие поля:

itemId: идентификатор продукта из коллекции food.

quantity: количество продуктов в акции.

price: цена продукта в акции.

saleSchema: основная схема для акции, содержащая следующие поля:

date: дата, обязательное поле для даты проведения акции.

items: массив объектов saleItemSchema, представляющий список продуктов в акции.

На основе saleSchema создается модель Mongoose saleModel, которая затем экспортируется для использования в других частях приложения.

routes/userRoute.js

Этот файл определяет маршруты для операций, связанных с пользователями, таких как регистрация и аутентификация, и подключает соответствующие функции контроллера.

package.json

Этот файл содержит метаданные проекта, список зависимостей и скрипты для запуска сервера с помощью nodemon.

controllers/cartController.js

Этот файл содержит функции для обработки операций, связанных с корзиной пользователя, таких как добавление товаров в корзину, удаление товаров из корзины и получение содержимого корзины.

routes/cartRoute.js

Этот файл определяет маршруты для операций, связанных с корзиной, и подключает соответствующие функции контроллера. Он также использует authMiddleware для проверки аутентификации пользователя.

controllers/foodController.js

Этот файл содержит функции для обработки операций, связанных с продуктами питания, таких как добавление новых продуктов, получение списка продуктов и удаление продуктов. Он использует библиотеку multer для обработки загрузки изображений продуктов.

routes/foodRoute.js

Этот файл определяет маршруты для операций, связанных с продуктами питания, и подключает соответствующие функции контроллера.

controllers/orderController.js

Этот файл содержит функции для обработки операций, связанных с заказами, таких как размещение заказа, проверка оплаты заказа, получение списка заказов пользователя и обновление статуса заказа. Он использует библиотеку stripe для обработки платежей через Stripe.

routes/orderRoute.js

Этот файл определяет маршруты для операций, связанных с заказами, и подключает соответствующие функции контроллера. Он также использует authMiddleware для проверки аутентификации пользователя.

emailService.js

Этот файл содержит функцию sendOrderEmail, которая отправляет электронные письма с подтверждением заказа клиенту и администратору. Он использует библиотеку nodemailer для отправки электронных писем через SMTP-сервер Яндекса.

routes/saleRoute.js

Этот файл определяет маршруты для операций, связанных с акциями и скидками на продукты. Он позволяет добавлять новые акции и получать список акций на текущий день.

routes/couponRoutes.js

Этот файл определяет маршруты для операций, связанных с купонами на скидку. Он позволяет создавать, получать и применять купоны. Весь этот код работает совместно, обеспечивая функциональность серверной части веб-приложения для доставки еды, включая аутентификацию пользователей, управление корзиной, операции с продуктами питания, размещение и отслеживание заказов, обработку платежей, отправку электронных писем и управление акциями и купонами.

3.Написание админ панели с помощью React и Node.JS.

### 3.3Структура админ панели

админ панель состоит из следующих основных файлов и папок:

app.jsx: Главный компонент React, который содержит навигацию и маршруты для различных страниц.

main.jsx: Точка входа для React-приложения, где происходит рендеринг компонента App.

pages/add/add.jsx: Страница для добавления новых продуктов.

pages/list/list.jsx: Страница для отображения списка всех продуктов и управления ими.

pages/orders/orders.jsx: Страница для отображения и управления заказами.

components/navbar/navbar.jsx: Компонент навигационной панели.

components/sidebar/sidebar.jsx: Компонент боковой панели навигации.

Функциональность

app.jsx

Этот файл содержит главный компонент React App. Он импортирует необходимые компоненты (Navbar, Sidebar) и страницы (Add, List, Orders). Компонент App отвечает за отображение навигационной панели, боковой панели и маршрутов для различных страниц с помощью react-router-dom.

main.jsx

Этот файл является точкой входа для React-приложения. Он импортирует компонент App и рендерит его внутри BrowserRouter из react-router-dom. Это позволяет использовать маршрутизацию в приложении.

pages/add/add.jsx

Эта страница отвечает за добавление новых продуктов. Она содержит форму, где пользователь может загрузить изображение продукта, ввести его название, описание, цену и выбрать категорию. Также есть возможность отметить, является ли продукт акционным. При отправке формы данные отправляются на сервер с помощью axios. В зависимости от типа продукта (обычный или акционный) используется соответствующий конечный путь API. После успешного добавления продукта форма очищается, и отображается уведомление.

pages/list/list.jsx

Эта страница отображает список всех продуктов. Она получает данные с сервера с помощью axios и отображает их в табличном виде. Пользователь может выбрать один или несколько продуктов для дальнейших действий На этой странице также присутствует компонент SaleForm, который позволяет добавлять выбранные продукты в акцию.

pages/orders/orders.jsx

Эта страница отображает список всех заказов. Она получает данные с сервера с помощью axios и отображает их в виде списка. Для каждого заказа отображается информация о продуктах, адресе доставки, общей сумме и текущем статусе заказа. Пользователь может изменить статус заказа с помощью выпадающего списка. При изменении статуса данные отправляются на сервер с помощью axios.

components/navbar/navbar.jsx

Этот компонент отвечает за отображение навигационной панели. Он содержит логотип и изображение профиля пользователя.

components/sidebar/sidebar.jsx

Этот компонент отвечает за отображение боковой панели навигации. Он содержит ссылки на страницы "Добавить продукт", "Все продукты" и "Заказы". Ссылки реализованы с помощью NavLink из react-router-dom.

# Заключение

Проект веб-приложения «ДодоПтица» для службы доставки еды был выполнен с использованием современного технологического стека, включающего Node.js, Express.js, React.js, MongoDB и Stripe. В процессе работы над проектом мы достигли поставленных целей, что позволяет утверждать, что разработка успешна и отвечает заявленным требованиям.

Обобщение хода решения

В рамках разработки веб-приложения были выполнены следующие основные задачи:

Выбор стека технологий и их интеграция:

Node.js и Express.js использовались для создания производительного и масштабируемого сервера, способного обрабатывать большое количество запросов.

React.js был выбран для разработки динамичного и отзывчивого пользовательского интерфейса, что улучшило взаимодействие пользователей с приложением.

MongoDB обеспечил гибкое и эффективное хранение данных, необходимое для обработки заказов и управления контентом.

Stripe был интегрирован для обработки безопасных и надежных платежей, что является критически важным аспектом для любого приложения электронной коммерции.

Реализация функциональных требований:

Авторизация и регистрация пользователей: были реализованы возможности для создания учетных записей и входа в систему с использованием JWT для обеспечения безопасного доступа.

Управление акциями и каталогом товаров: Система поддерживает управление ежедневными и еженедельными акциями, а также отображение каталога с возможностью фильтрации по категориям.

Корзина товаров и промокоды: Пользователи могут добавлять товары в корзину, применять промокоды и изменять содержимое корзины перед оформлением заказа.

Выбор способа доставки и уведомления: Реализована возможность выбора между доставкой и самовывозом, а также отправка уведомлений по электронной почте о статусе заказа.

Отслеживание статуса заказа: Пользователи могут отслеживать этапы обработки своих заказов через интерфейс приложения.

Административная панель: Предоставлена панель для управления контентом, акциями, заказами и учетными записями курьеров.

Архитектура клиент-сервер:

Клиентская часть была реализована на базе React.js, обеспечивая интерактивное взаимодействие с пользователями и обновление данных в реальном времени без перезагрузки страниц.

Серверная часть, построенная на Node.js и Express.js, обрабатывает запросы, взаимодействует с базой данных и управляет платежами, обеспечивая надежную и быструю работу приложения.

Интеграция и тестирование:

Проведены тщательные тесты для проверки корректности всех аспектов функциональности, включая обработку заказов, управление пользователями, платежи и уведомления.

Веб-приложение было развернуто и протестировано на локальном сервере, что позволило убедиться в его работоспособности и стабильности.

Итог

В результате проведенной работы была достигнута основная цель проекта — разработка веб-приложения для службы доставки еды «ДодоПтица», соответствующего современным стандартам и требованиям. Все заявленные функции были успешно реализованы, что подтверждает выполнение поставленных задач.

Актуальность проекта в условиях современного рынка очевидна: потребность в удобных и надежных онлайн-сервисах для доставки еды продолжает расти. Веб-приложение «ДодоПтица» предлагает конкурентоспособное решение, которое способно удовлетворить запросы пользователей и обеспечить эффективное обслуживание.

В ходе выполнения проекта были получены важные навыки и знания в области веб-разработки с использованием Node.js и сопутствующих технологий. Разработанное приложение демонстрирует, как современные инструменты могут быть использованы для создания высокопроизводительных и удобных сервисов.

Дальнейшие шаги могут включать расширение функциональности приложения, оптимизацию производительности и улучшение пользовательского интерфейса на основе полученных отзывов и анализа использования.

Таким образом, проект завершен успешно, и все задачи, поставленные в начале работы, были достигнуты.

Эти выводы демонстрируют, что работа была выполнена на высоком уровне, и разработанное веб-приложение полностью соответствует заявленным требованиям и целям.

# Список источников и документации

# ПРИЛОЖЕНИЯ

## Приложение 1: Итоговый вид веб-приложения



Рисунок 13

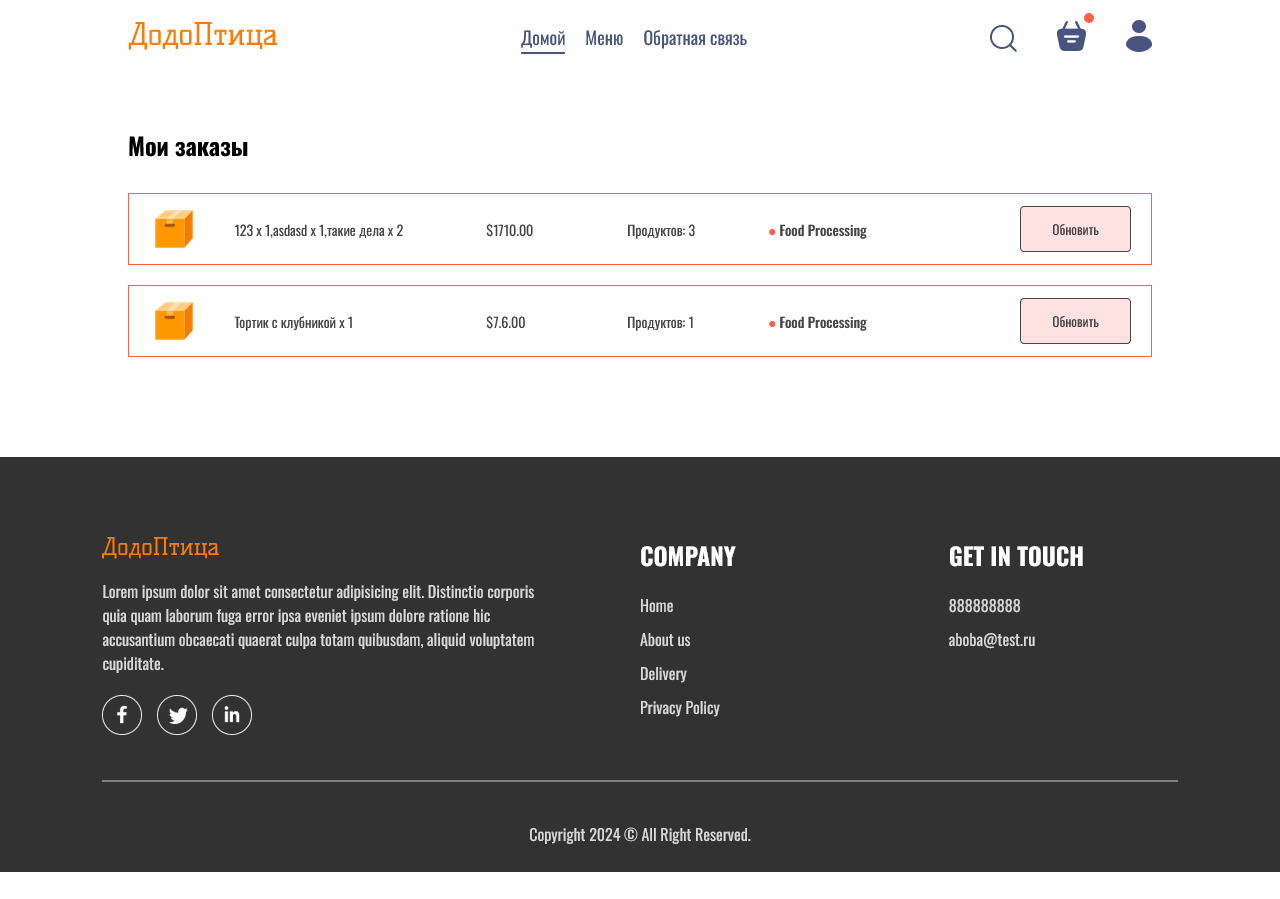


Рисунок 14

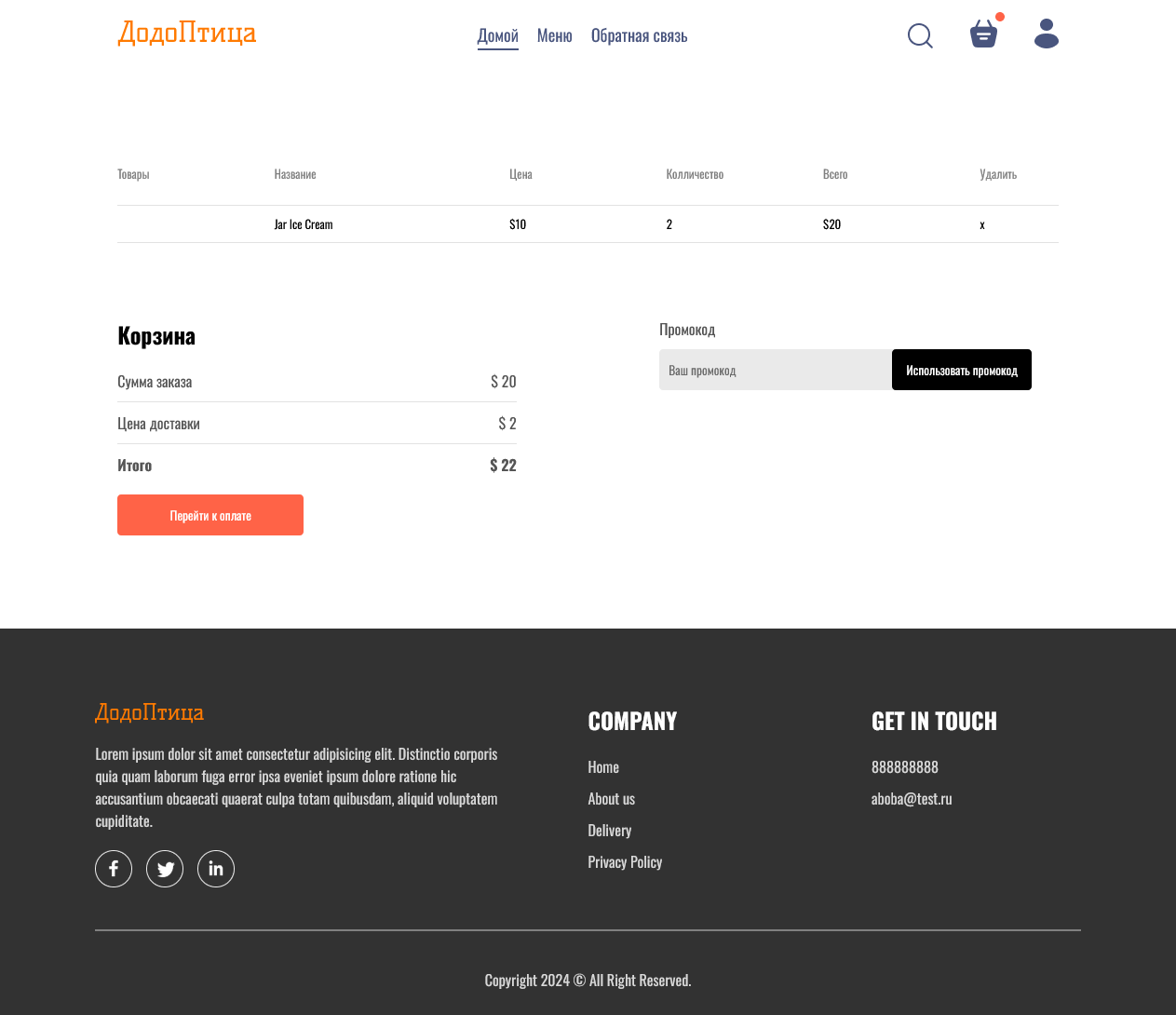


Рисунок 15

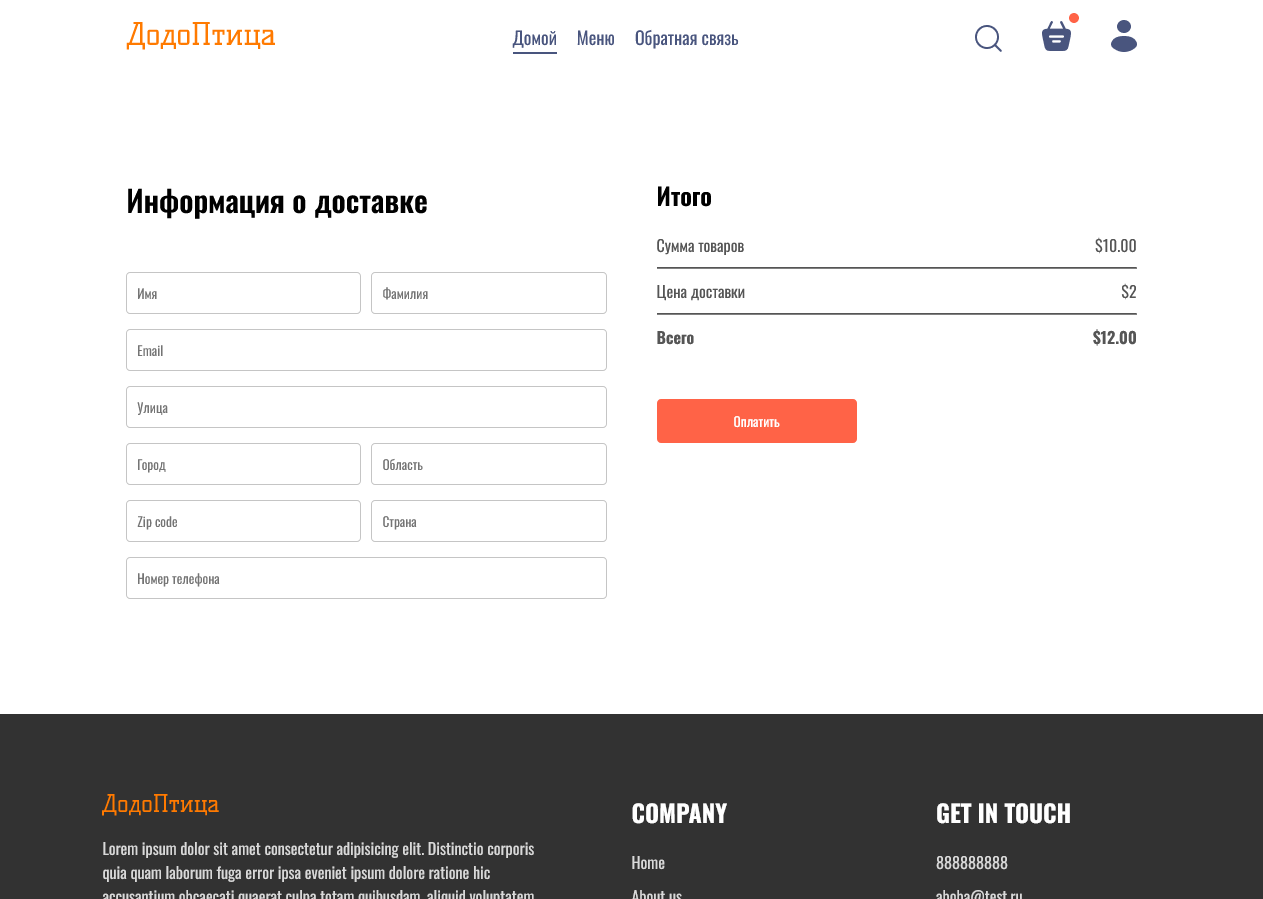


Рисунок 16

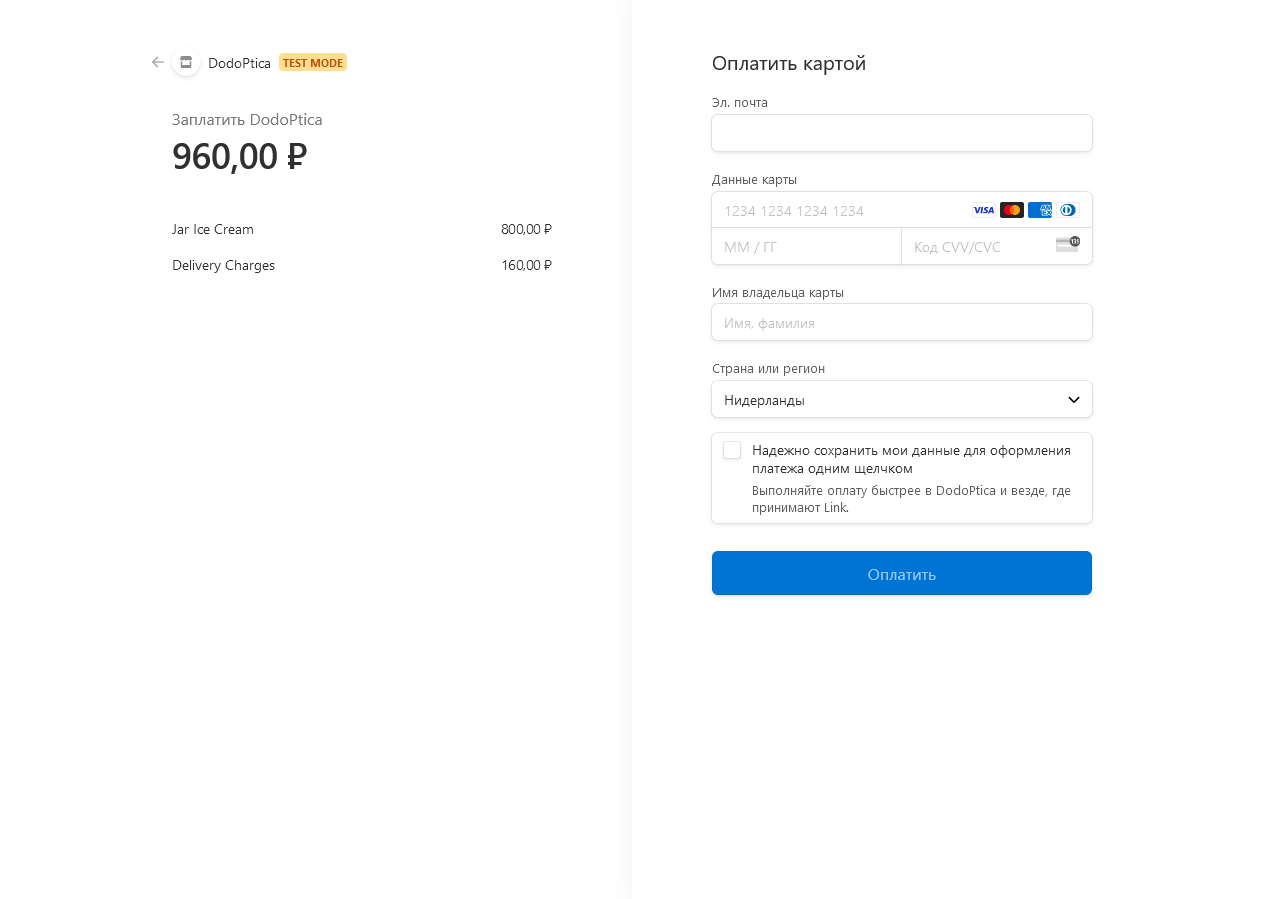


Рисунок 17

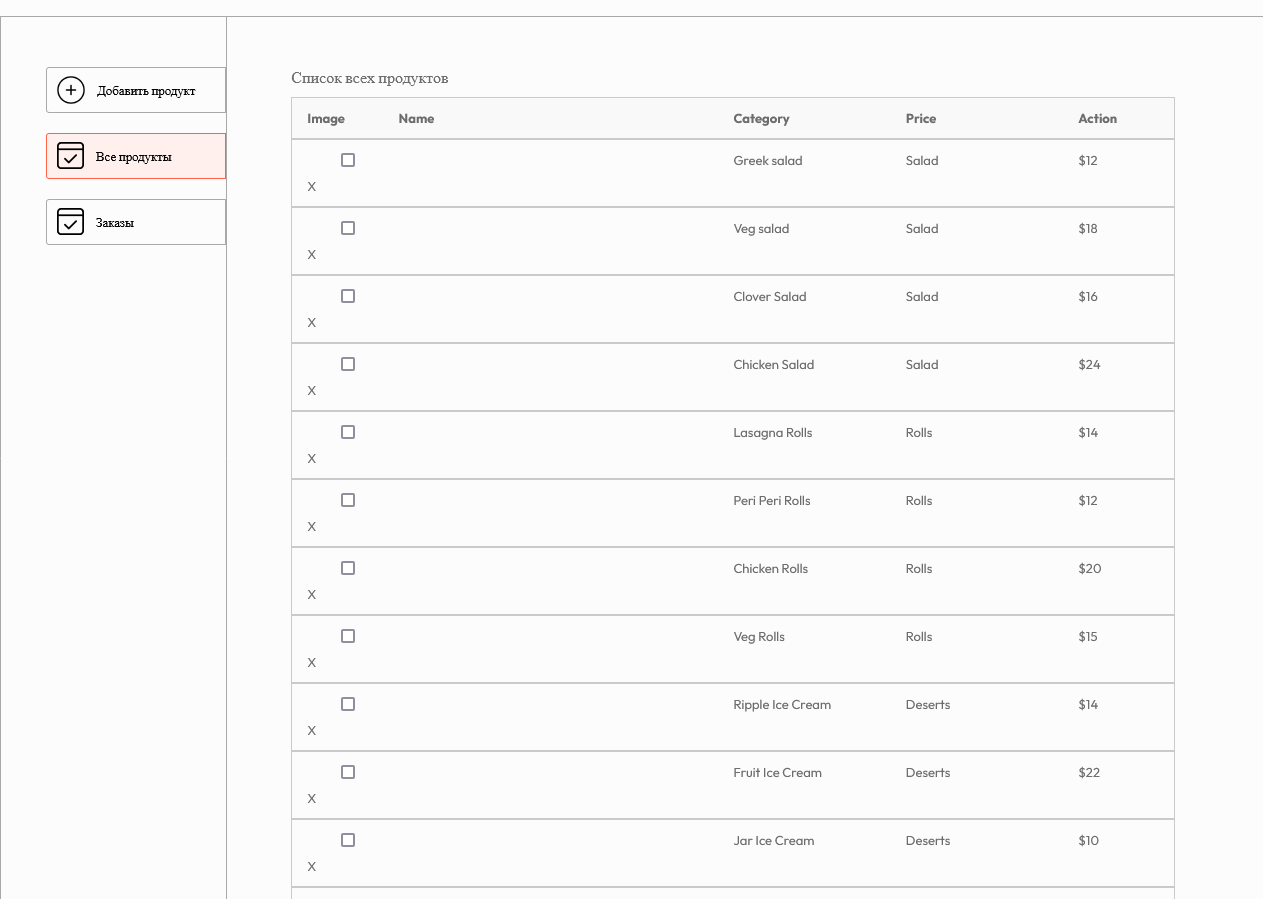


Рисунок 18

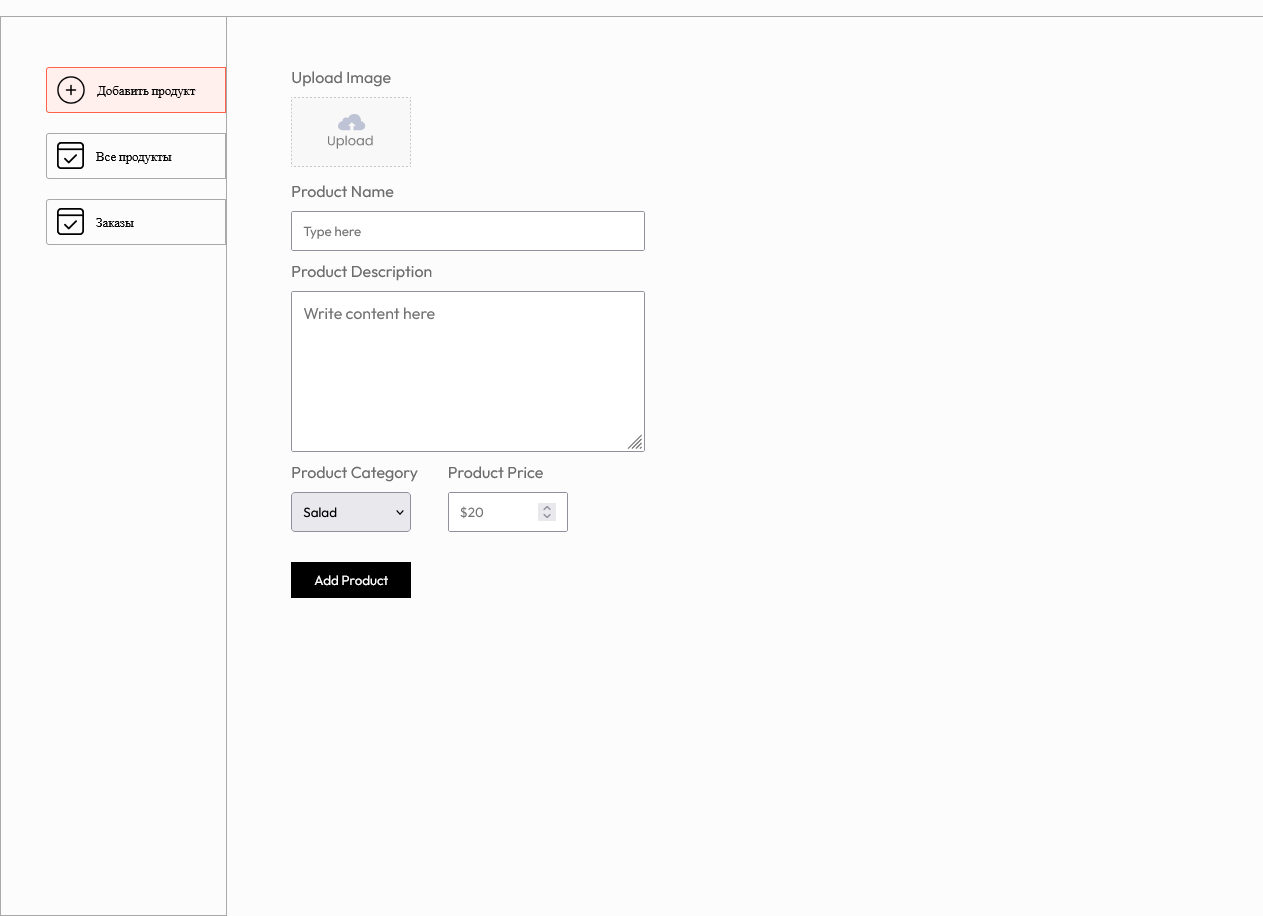


Рисунок 19

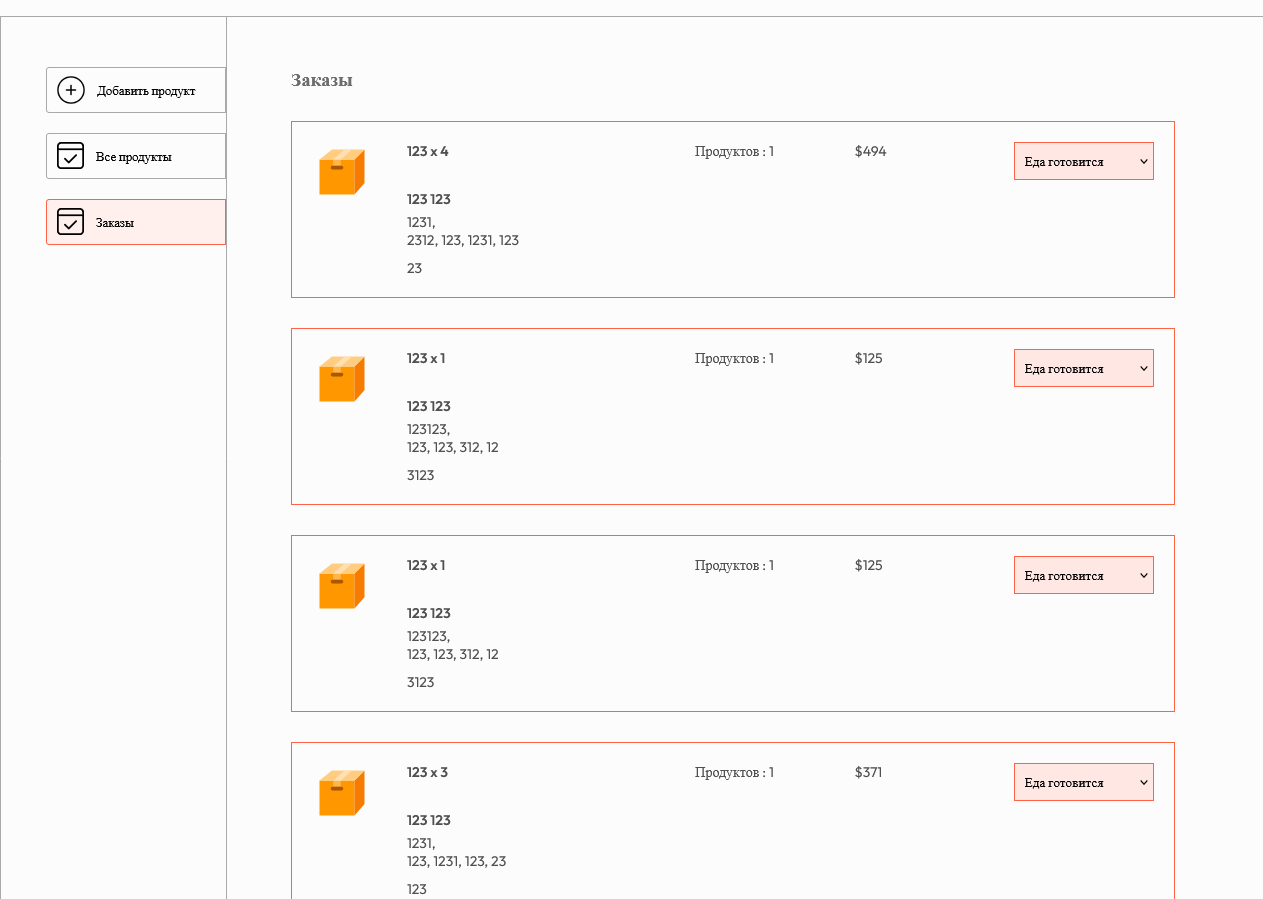


Рисунок 20