**Поволжский Государственный Университет Телекоммуникаций и Информатики**Кафедра «Программная инженерия»

|  |  |
| --- | --- |
| Сдана на проверку  «\_\_» \_\_\_\_2024 г. | Допустить к защите  «\_\_» \_\_\_\_\_ 2024 г.  Защищена с оценкой  «\_\_\_» \_\_\_\_\_\_\_\_ 2024 г. |

**КУРСОВАЯ РАБОТА**По дисциплине: «Прикладное программирование»  
На тему: «Разработка клиент-серверного web-приложения интернет-магазин фастфуда»   
  
Пояснительная записка

|  |  |
| --- | --- |
| Студент группы ИВТ-13\_\_\_\_\_\_\_ Руженцев А. Ю.  (роспись) (ФИО)  Руководитель \_\_\_\_\_\_\_\_ к.т.н., доц. Ахметшина Э. Г.  (роспись) (ФИО) | 0 210198 0  (№ зачетной книжки) |

Самара 2023

# Рецензия

# **Содержание**

[Описание предметной области. Актуальность 4](#_Toc3" \o "#_Toc3)

[Описание программы. Общие сведения 5](#_Toc4" \o "#_Toc4)

[Функциональное назначение 5](#_Toc5" \o "#_Toc5)

[Логическая модель базы данных 7](#_Toc6" \o "#_Toc6)

[Физическая модель базы данных 9](#_Toc7" \o "#_Toc7)

[Диаграмма классов 10](#_Toc8" \o "#_Toc8)

[Диаграмма компонентов 11](#_Toc9" \o "#_Toc9)

[Диаграмма вариантов использования 12](#_Toc10" \o "#_Toc10)

[Диаграмма последовательности 13](#_Toc11" \o "#_Toc11)

[Демонстрация работы приложения 14](#_Toc12" \o "#_Toc12)

[Список используемых источников 21](#_Toc13" \o "#_Toc13)

[Приложение А – Листинг программного кода 22](#_Toc14" \o "#_Toc14)

# Описание предметной области. Актуальность

Развитие интернет-магазина фаст-фуда представляет собой стратегически важное направление в ответ на постоянно растущий запрос общества. В современном мире, где темп жизни стремительно увеличивается, люди сталкиваются с ограниченным временем на приготовление пищи или посещение ресторанов. В этом контексте, интернет-магазин фаст-фуда становится необходимым инструментом для удовлетворения потребностей в быстром и удобном питании.

Одним из ключевых аспектов актуальности данной разработки является обеспечение клиентам удобства в процессе заказа. С использованием онлайн-платформы, клиенты имеют возможность выбирать блюда из разнообразного меню, делать заказы в удобное для них время и место. Этот подход минимизирует временные затраты на выбор и заказ пищи, что особенно ценно в современном ритме жизни, где каждая минута имеет значение.

Кроме того, оперативная доставка становится ключевым преимуществом интернет-магазина фаст-фуда. Заказы могут быть доставлены прямо к двери клиента в кратчайшие сроки, что обеспечивает удобство и экономит время. Это особенно важно для занятых людей, не имеющих возможности или желания посещать фаст-фудовые точки.

Наконец, разнообразие предлагаемого меню на интернет-платформе удовлетворяет различные вкусовые предпочтения клиентов. Широкий выбор блюд, включая различные кухни и диетические опции, делает интернет-магазин фаст-фуда привлекательным для разнообразной аудитории.

Таким образом, разработка интернет-магазина фаст-фуда не только соответствует современным требованиям общества к удобству и быстроте, но и предоставляет эффективное решение для тех, кто ценит свое время и желает наслаждаться разнообразным питанием без лишних усилий.

# Описание программы. Общие сведения

Приложение – веб-ресурс. Установка не требуется.

Язык программирования: TypeScript, Java.

Среда разработки: Intellij IDEA

Объем проекта: 173,6 МБ (вместе с исходным кодом).

Исходный код (Java и TypeScript): 943 строк.

# Функциональное назначение

Разрабатываемое приложение предназначено для создания эффективного и удобного механизма, обеспечивающего процесс заказа и доставки фаст-фуда. Основной целью приложения является предоставление пользователям мгновенного доступа к разнообразному меню, позволяя быстро и легко оформлять заказы и получать продукцию в указанное место доставки. В свете современных требований общества к оперативности и удобству в быстром питании, особенно в условиях ограниченного времени, приложение представляет собой ответ на растущий спрос.

Инновационный подход приложения заключается в предоставлении клиентам широкого выбора блюд, доступных для моментального заказа. Пользователи имеют возможность быстро ознакомиться с меню, выбрать оптимальные варианты и оформить заказ в несколько простых шагов. Это создает удобство для занятых людей, которые ценят оперативность и мобильность в своем образе жизни.

В целом, приложение не просто упрощает процесс заказа и доставки фаст-фуда, но и подстраивается под динамичные потребности современного общества. Его функциональность и удобство использования делают его значимым ответом на растущий запрос общества к удобству и оперативности в области быстрого питания, предоставляя пользователям мгновенный доступ к разнообразному ассортименту и быстрой доставке.

**Backend-часть** приложения разрабатывается с использованием Spring Boot - фреймворка для создания Java-приложений. Spring Boot обеспечивает высокую производительность, простоту в разработке и масштабируемость. Он также предоставляет инструменты для построения эффективных RESTful API, обеспечивая эффективное взаимодействие между клиентской и серверной частями приложения.

**Frontend-часть** построена на React.js и Vite. React.js обеспечивает создание динамичного и отзывчивого пользовательского интерфейса, а Vite — возможность быстрой загрузки страниц и разработки эффективных веб-приложений. Эти технологии обеспечивают превосходное взаимодействие пользователей с интерфейсом приложения, что является ключевым фактором для удовлетворения требований современных пользователей.

**Хранение данных** осуществляется при помощи Postgres - реляционной базы данных. Postgres обеспечивает стабильное и надежное хранение информации о меню, заказах и других важных сущностях приложения. Реляционная структура базы данных позволяет эффективно управлять данными и обеспечивает их целостность.

Использование технологий Spring Boot, React.js, Vite и Postgres в совокупности создает мощное, современное приложение, которое эффективно отвечает на потребности пользователей. Этот технологический стек обеспечивает не только высокую производительность и отзывчивость интерфейса, но и обеспечивает стабильность и легкость в поддержке, что является важным фактором для долгосрочной успешной эксплуатации приложения.

# Логическая модель базы данных

1. Таблица "clients":

* id: Уникальный идентификатор клиента, является первичным ключом таблицы.
* firstname: Имя клиента.
* lastname: Фамилия клиента.
* phone: Номер телефона клиента.

1. Таблица "dishes":

* id: Уникальный идентификатор блюда, является первичным ключом таблицы.
* composition: Состав блюда.
* description: Описание блюда.
* image: Объектный идентификатор (oid) для хранения изображения блюда.
* name: Название блюда.
* price: Цена блюда.

1. Таблица "orders":

* id: Уникальный идентификатор заказа, является первичным ключом таблицы.
* address: Адрес доставки заказа.
* comment: Комментарий к заказу.
* count: Количество единиц заказанного блюда.
* client\_id: Внешний ключ, связанный с "id" в таблице "clients". Указывает на клиента, сделавшего заказ. Обеспечивает связь сущности заказа с клиентом.
* dish\_id: Внешний ключ, связанный с "id" в таблице "dishes". Указывает на блюдо, включенное в заказ. Обеспечивает связь сущности заказа с блюдом.

Уникальное ограничение на поле client\_id гарантирует, что каждый клиент может сделать только один заказ, предотвращая дублирование информации. Связи сущности заказа с клиентом и блюдом обеспечивают целостность данных и облегчают выполнение запросов, связанных с анализом заказов по клиентам и блюдам.

# Физическая модель базы данных

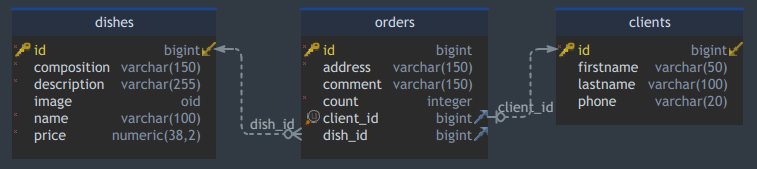


Рис. 1.1– Физическая модель базы данных

# Диаграмма классов

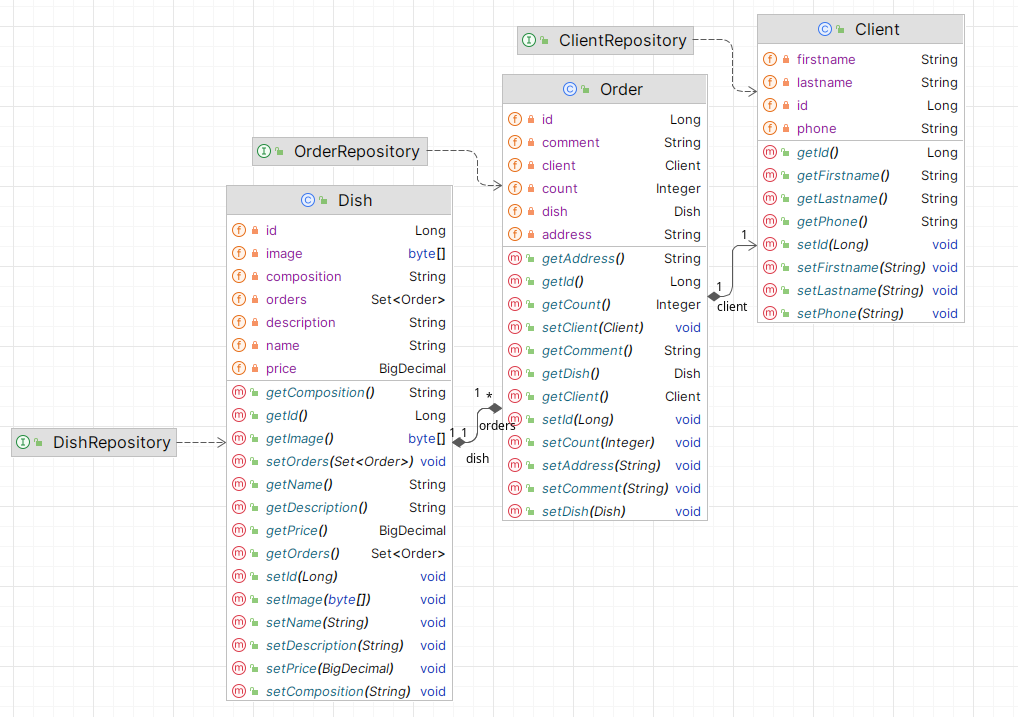


Рис. 2.1 – Диаграмма классов

# Диаграмма компонентов

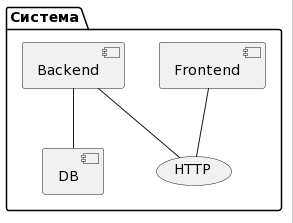


Рис. 3.1 – Диаграмма компонентов

# Диаграмма вариантов использования

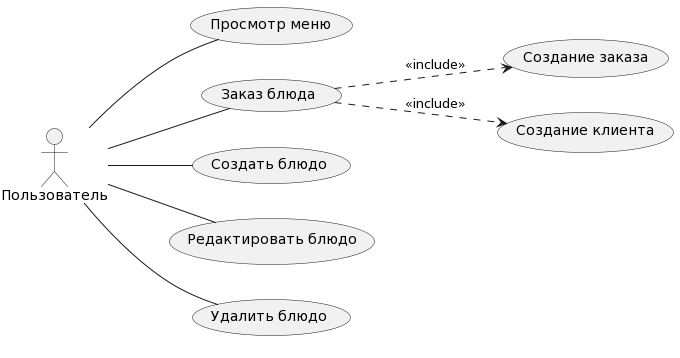


Рис. 4.1 – Диаграмма вариантов использования

# Диаграмма последовательности

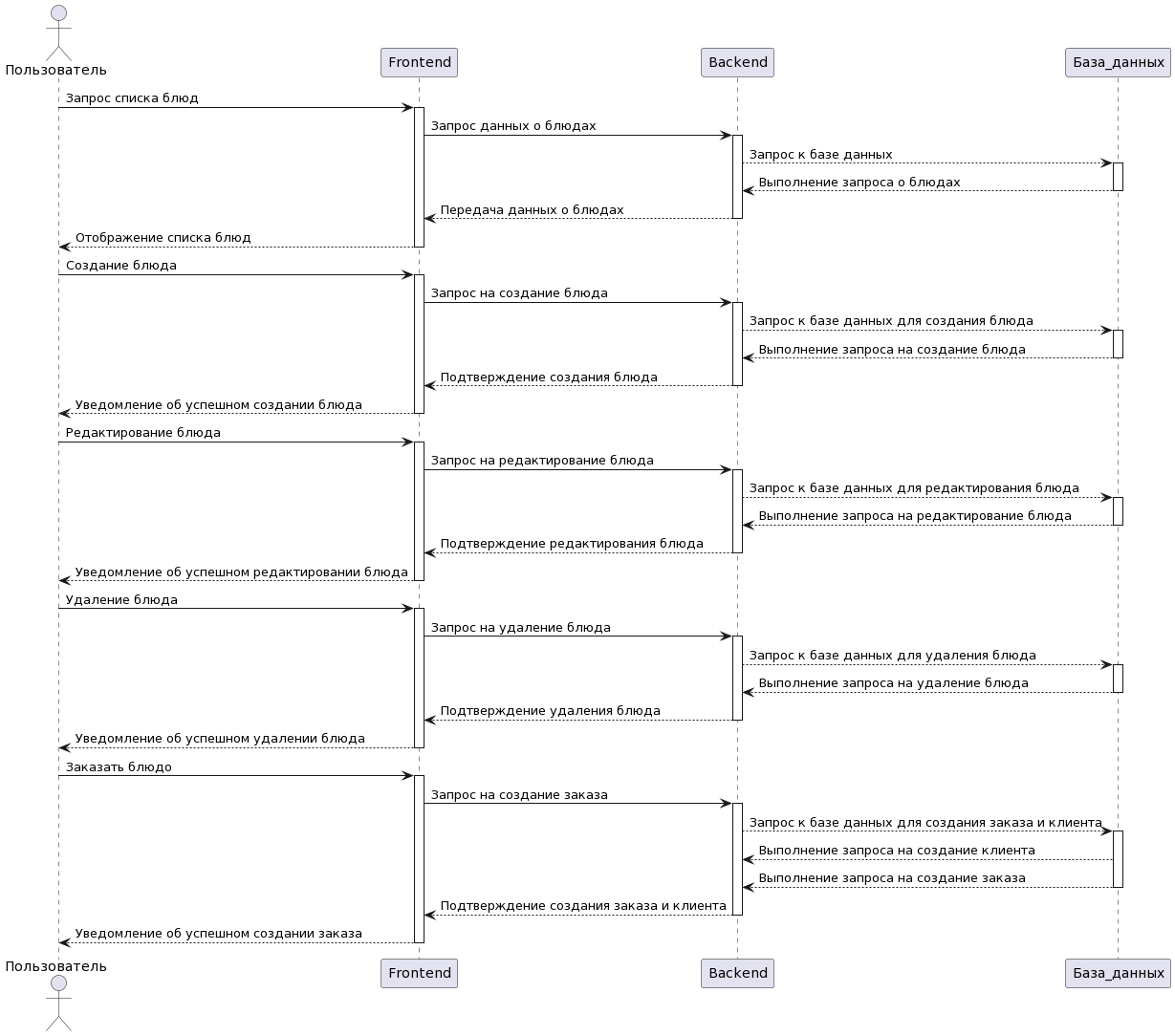


Рис. 5.1 – Диаграмма последовательности

# Демонстрация работы приложения

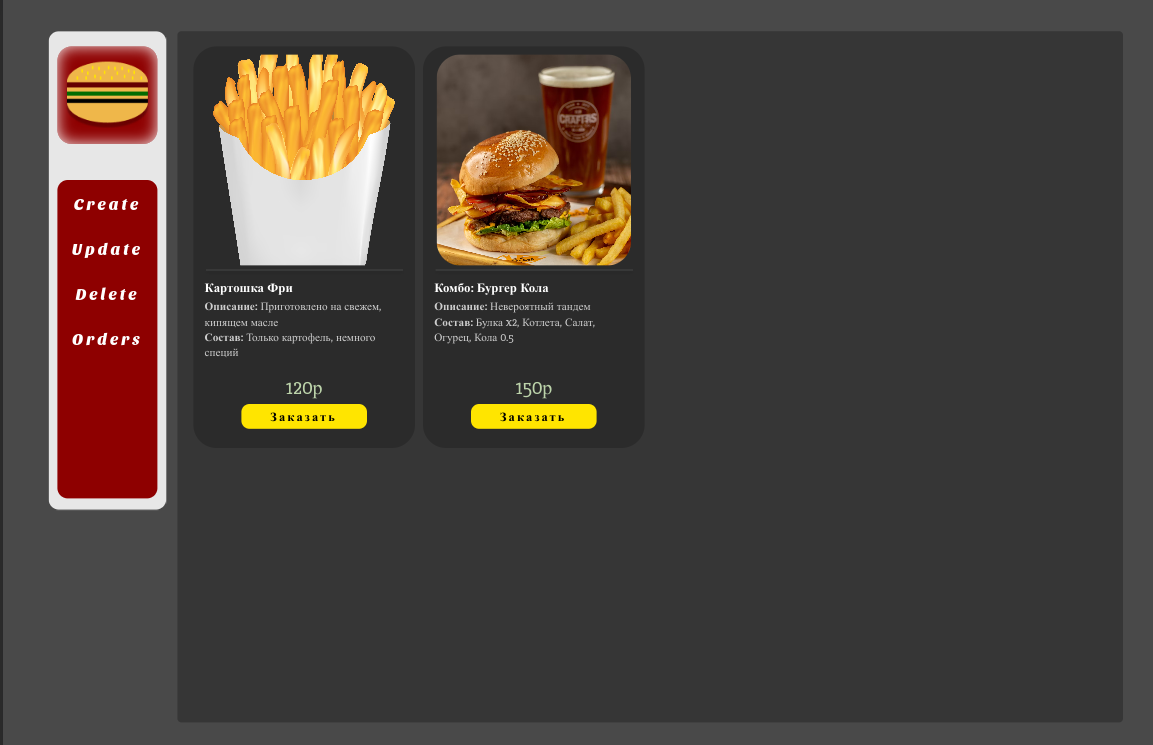


Рис. 6.1 – Главная страница приложения

При открытии web-приложения в браузере, пользователь попадает на единственную страницу с открытым разделом меню. (Рис. 6.1)

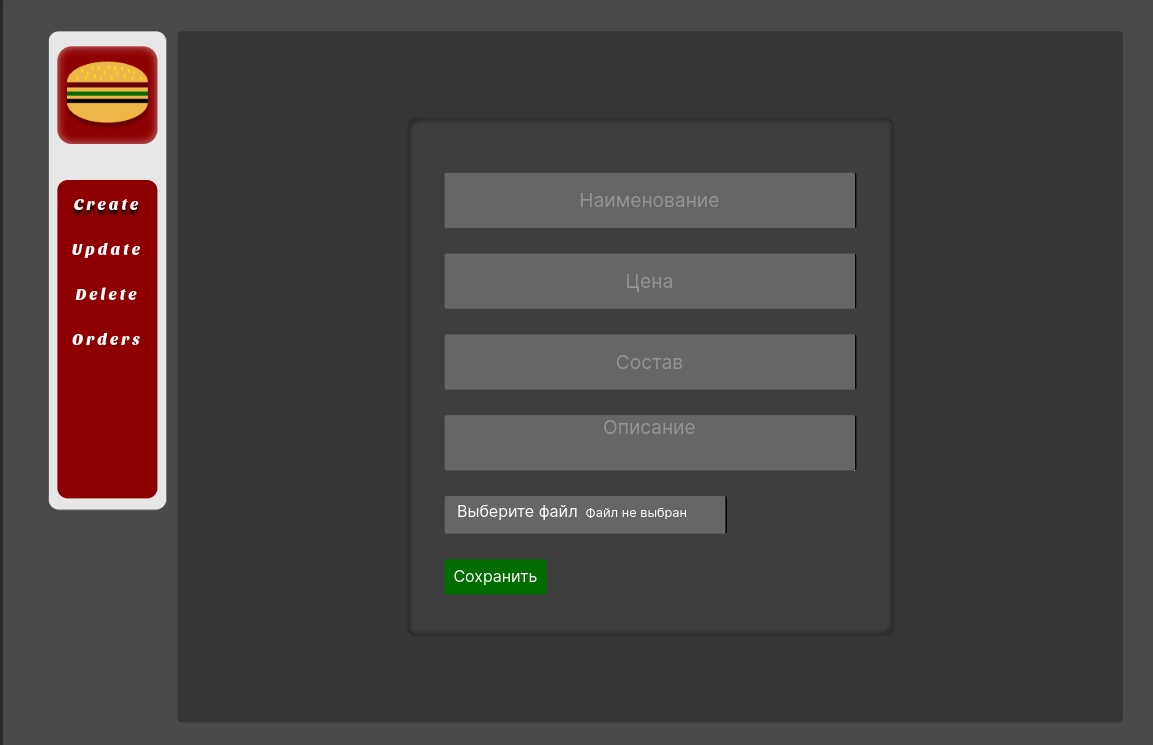


Рис. 6.2 – Создание блюда

Если пользователь нажимает на «Create» в меню навигации, он получает форму для создания нового блюда. (Рис. 6.2)

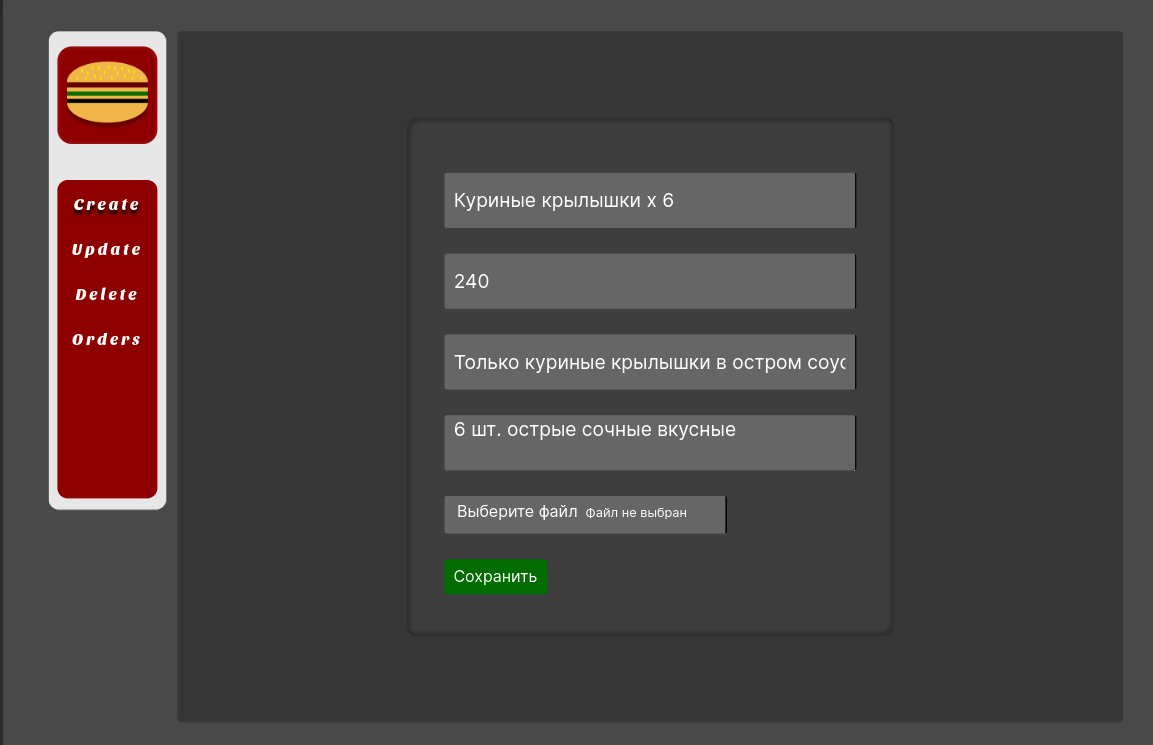


Рис. 6.3 – Ввод новых данных в форму

Для создания заполним форму данными блюда «куриные крылышки». (Рис. 6.3)

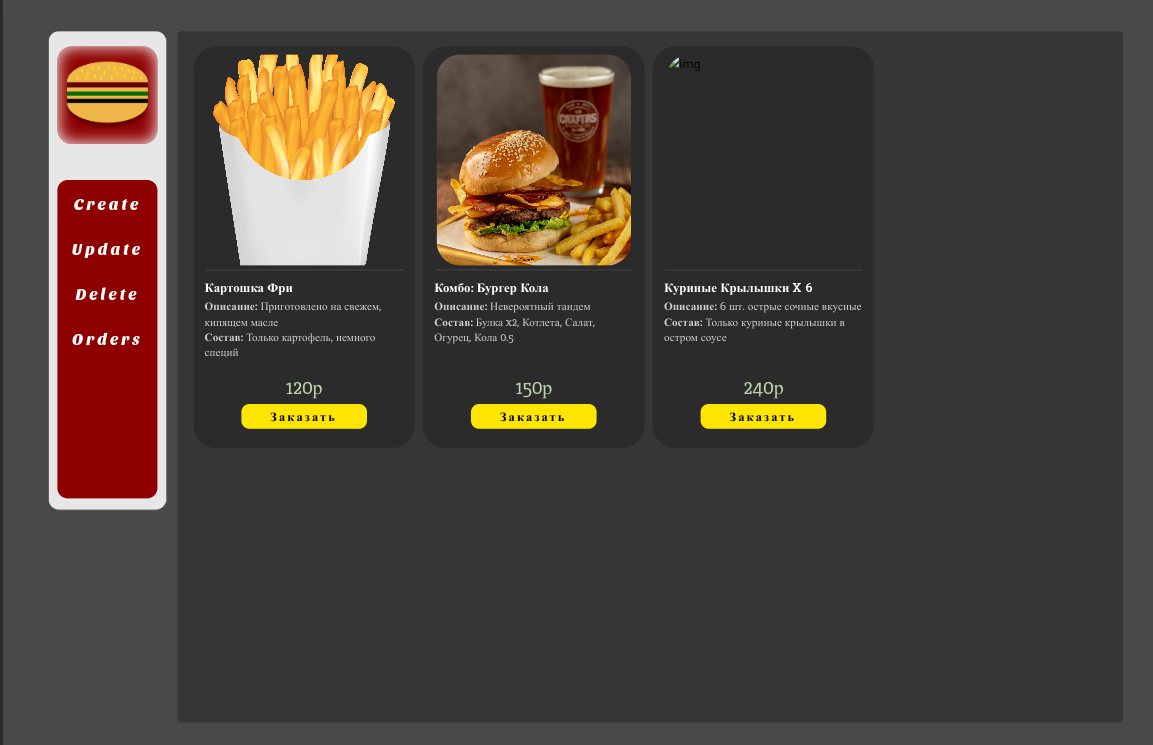


Рис. 6.4 – Результат создания блюда

После нажатия кнопки «Сохранить», пользователь попадает на раздел с меню и видит только что созданное блюдо. (Рис. 6.4)

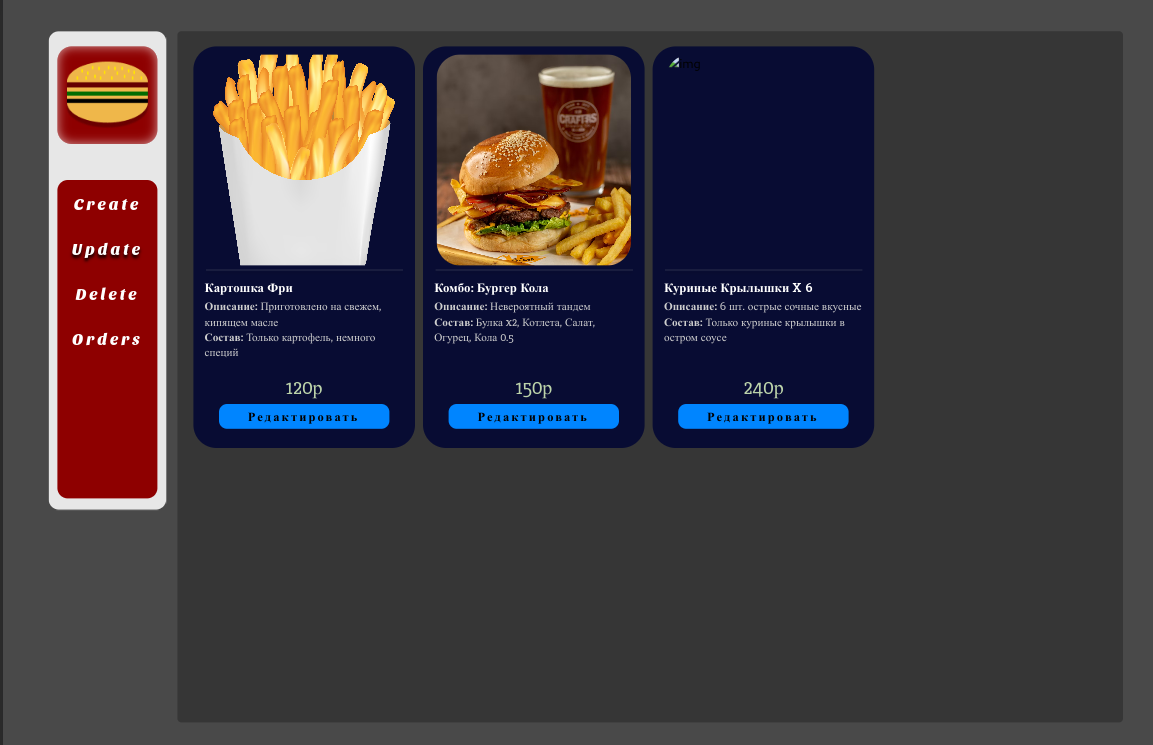


Рис. 6.5 – Раздел редактирования

В любой момент может понадобится отредактировать какие-то данные о блюде. (Рис. 6.5)

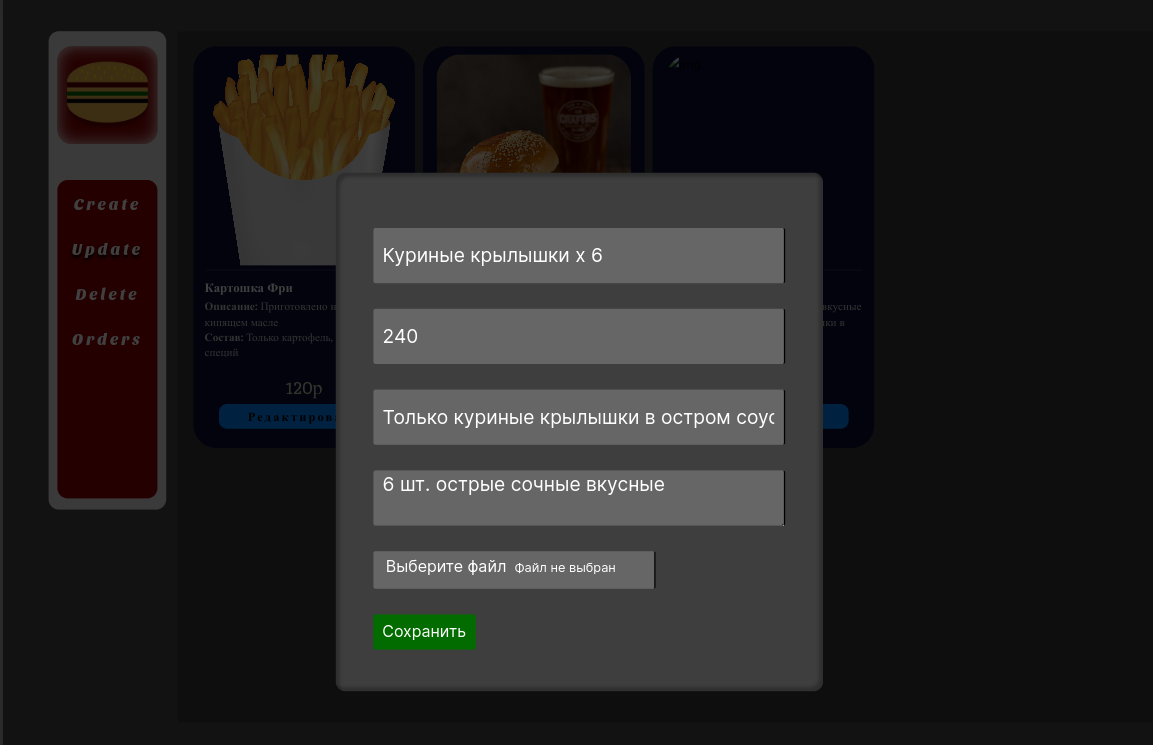


Рис. 6.6 – Форма редактирования

Пользователь создал новое блюдо, но забыл добавить изображение. С помощью редактирования можно исправить это упущение. (Рис. 6.6)

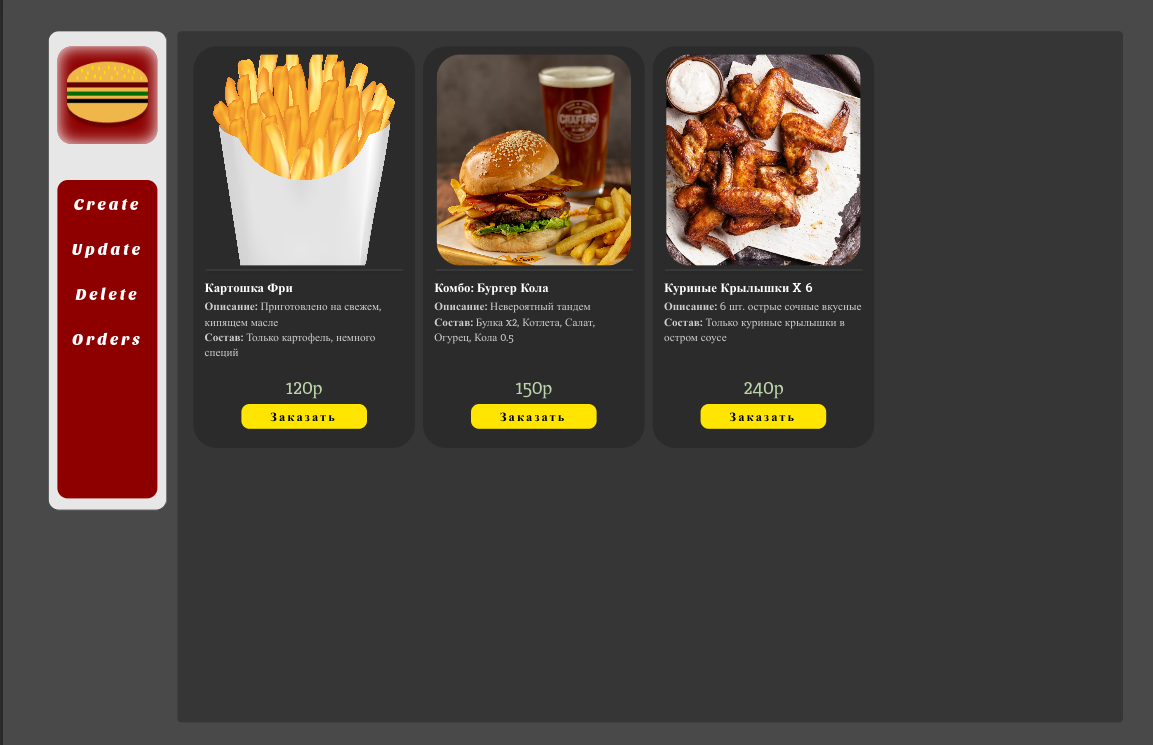


Рис. 6.7 – Результат редактирования

После редактирования видно, что у блюда с куриными крыльями появилось изображение. (Рис. 6.7)

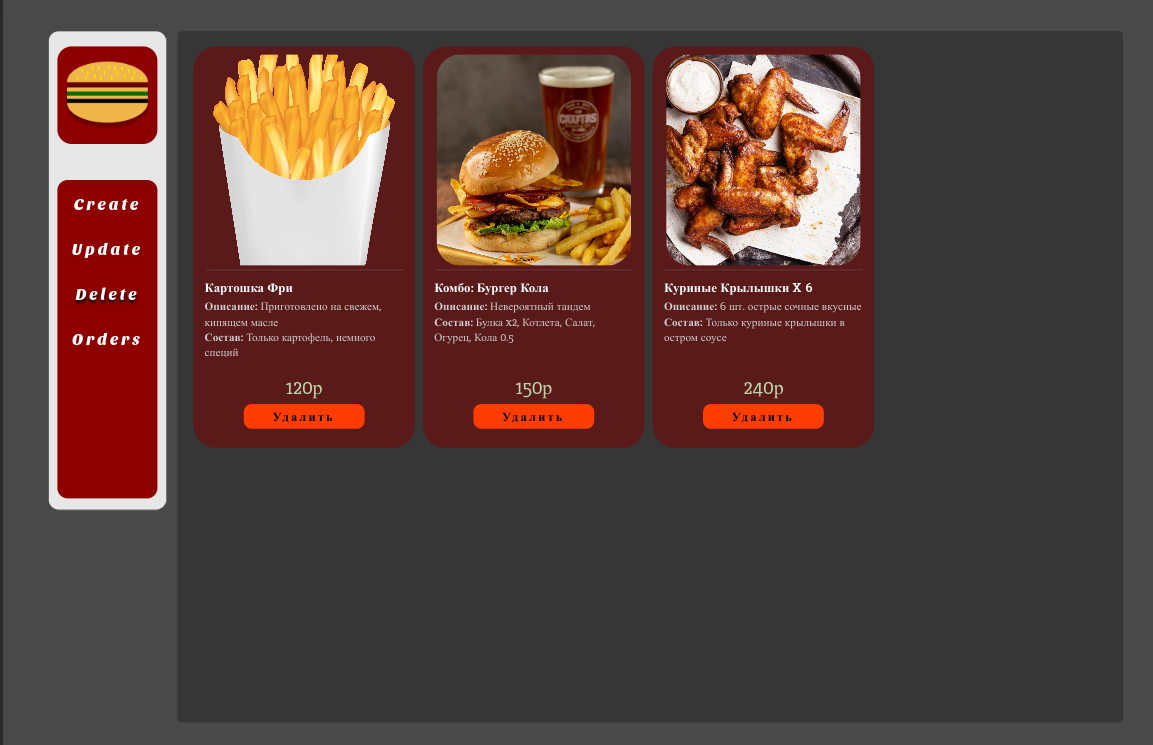


Рис. 6.8 – Удаление блюд

В системе предусмотрено удаление блюд из меню. Для этого нужно перейти в раздел для удаления. (Рис. 6.8)

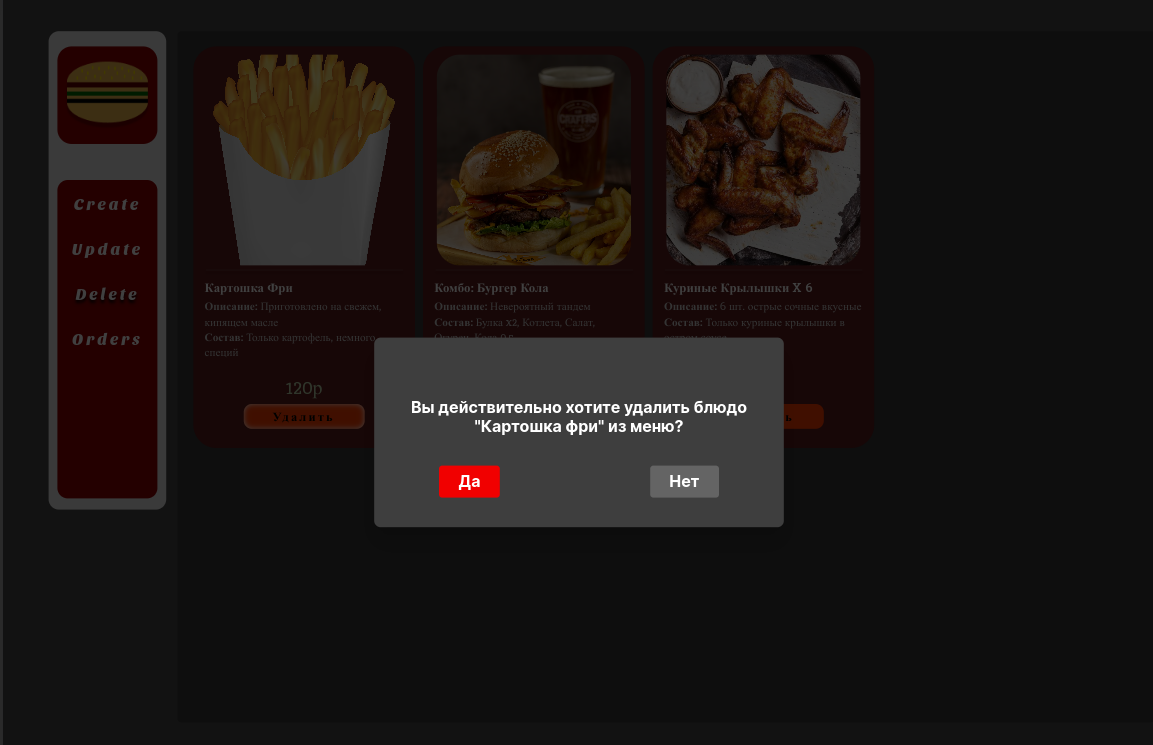


Рис. 6.9 – Удаление «Картошка Фри»

Пользователь полагает, что это блюдо стоит убрать из меню и поэтому удаляет его. (Рис. 6.9)

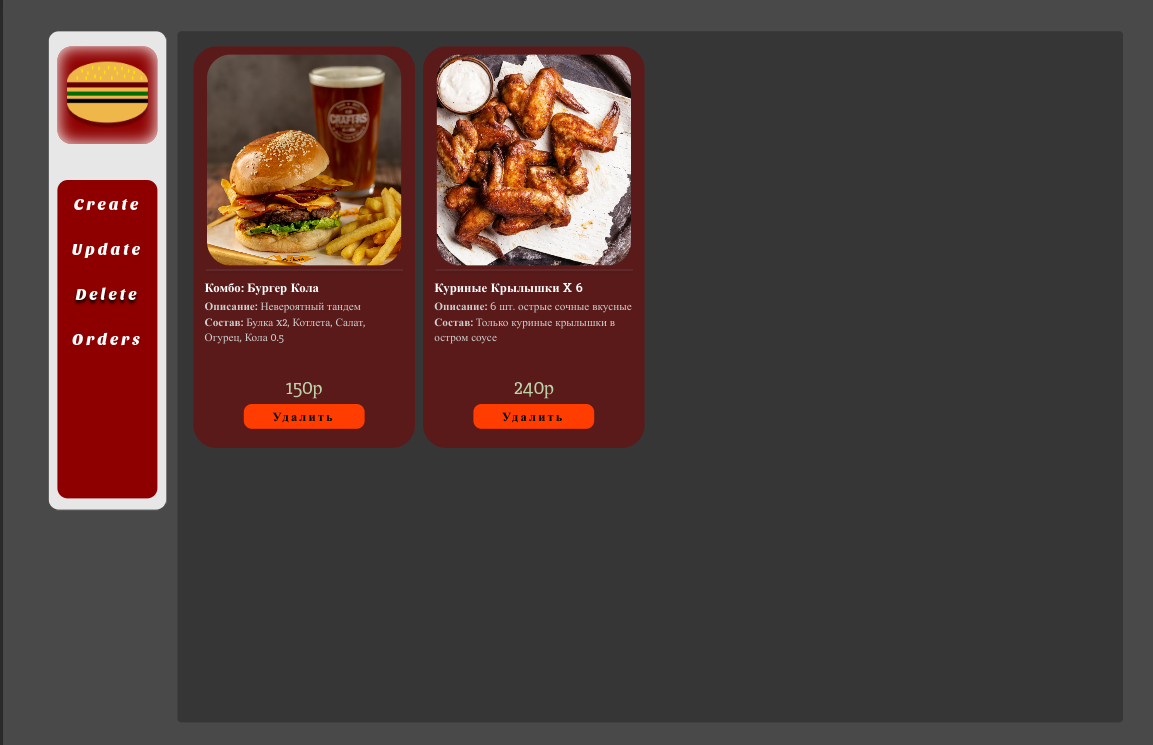


Рис. 6.10 – Результат удаления

После нажатия на кнопку «Да», блюдо успешно удаляется из меню. (Рис. 6.10)

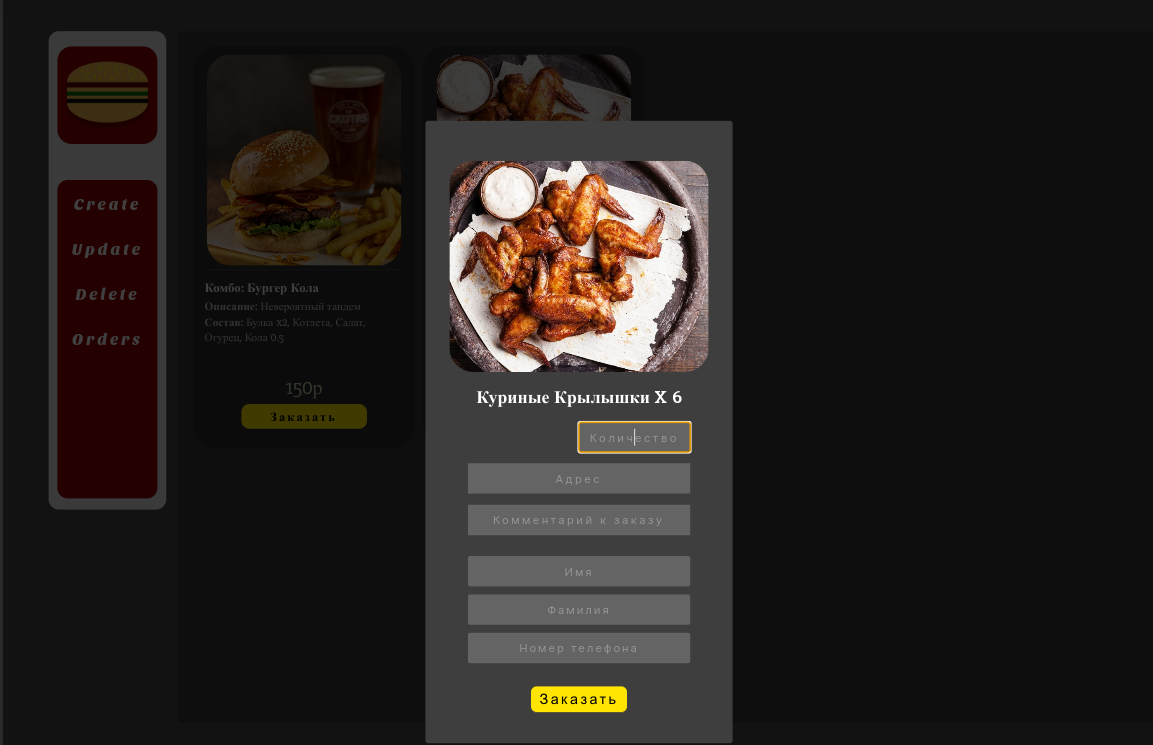


Рис. 6.11 – Заказ блюд

Если пользователь присмотрелся к какому-то блюду, он может его заказать, нажав на кнопку «Заказать» в карточке блюда. (Рис. 6.11)

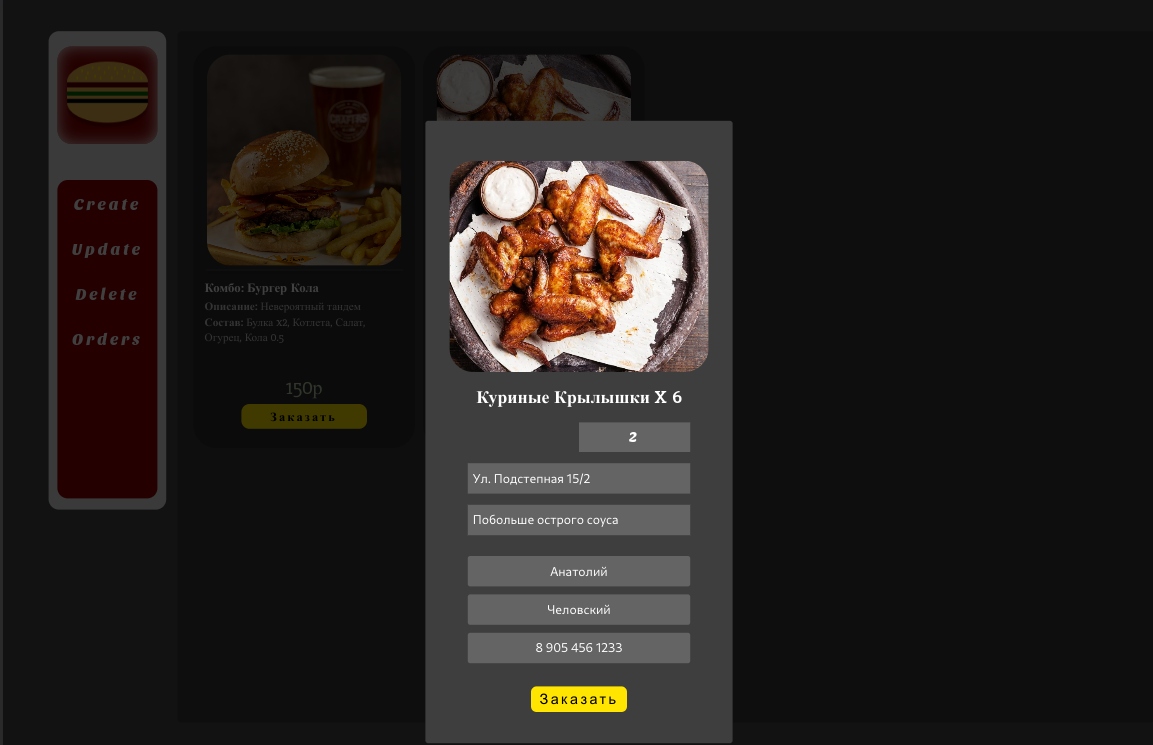


Рис. 6.12 – Заполненная форма заказа

Пользователь заполнил форму для заказа блюда. (Рис. 6.12)

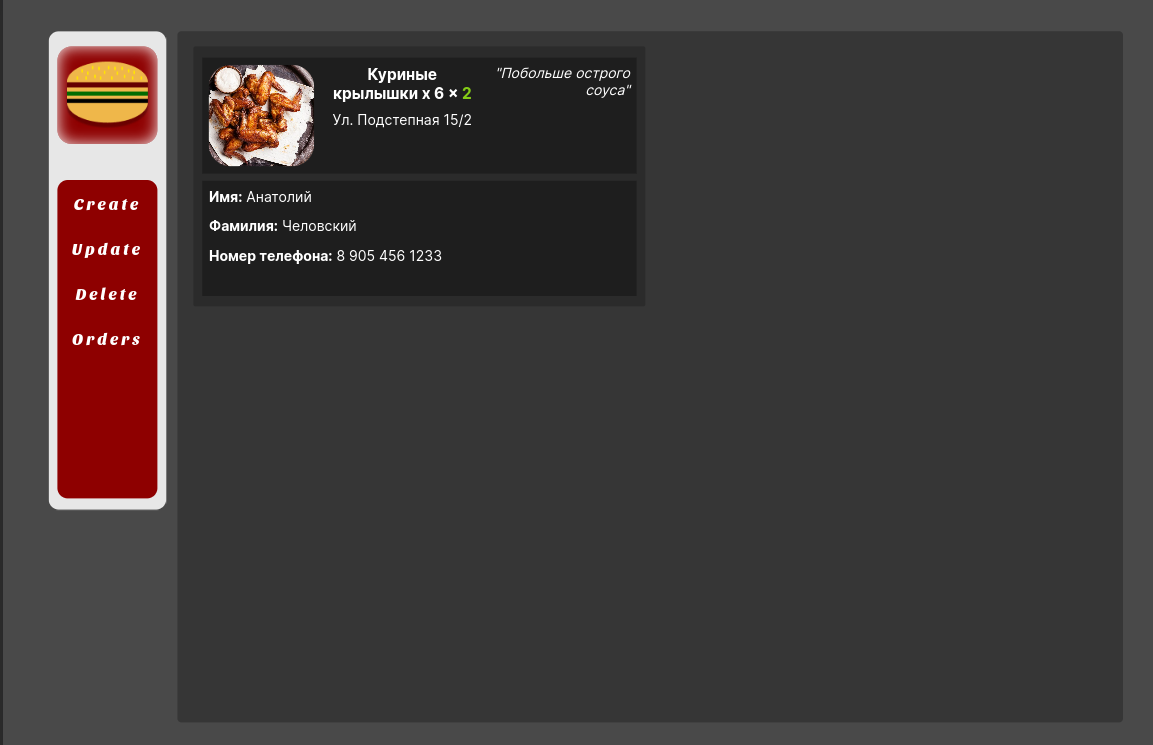


Рис. 6.13 – Раздел заказов

После создания заказа пользователь попадает в раздел «заказы», где видит данные о своем заказе. (Рис. 6.13)

# Список используемых источников

1. Бен Фрейн "CSS3 в примерах", Питер, 2020, 288 стр. (ориг. название: "CSS in Depth", Manning Publications)
2. Эрик Фримен, Элизабет Робсон "TypeScript: Подробное руководство", Вильямс, 2022, 992 стр. (ориг. "TypeScript: The Definitive Guide" от O'Reilly Media)
3. Крейг Уоллс "Spring Boot в действии", Питер, 2020, 616 стр. (ориг. название: "Spring Boot in Action", Manning Publications)
4. Крейг Уоллс "Spring в действии", Питер, 2020, 768 стр. (ориг. название: "Spring in Action", Manning Publications)
5. Герберт Шилдт "Java: Руководство для начинающих", ДМК Пресс, 2021, 704 стр. (ориг. название: "Java: A Beginner's Guide", McGraw-Hill)
6. Эрик Фримен, Элизабет Робсон "HTML и CSS. Дизайн и верстка сайтов", Вильямс, 2021, 720 стр. (ориг. название: "Head First HTML and CSS", O'Reilly Media)
7. Джон Дакетт "HTML5: Руководство разработчика", Питер, 2021, 944 стр. (ориг. название: "HTML5 Developer's Guide", McGraw-Hill)
8. Дэвид Флэнаган "JavaScript: Подробное руководство", Вильямс, 2022, 1224 стр. (ориг. название: "JavaScript: The Definitive Guide", O'Reilly Media)
9. Крис Шейфер, Джон Уайсс "Spring Framework. Практика разработки Enterprise-приложений", Питер, 2022, 784 стр. (ориг. название: "Pro Spring 5", Apress)
10. Дэвид Геккер, Джон Оспорн "React: Быстрый старт", Питер, 2021, 320 стр. (ориг. название: "Learning React: Modern Patterns for Developing React Apps", O'Reilly Media)

# Приложение А – Листинг программного кода

package ru.michael.backend.entity;  
import { useEffect, useState } from 'react';

import Container from './components/Container'

import { DishData } from './components/card/dish/DishCard'

import { NavigationPanel } from './components/nav/NavigationPanel';

import { States } from './Constans';

import DishForm from './components/form/dish/DishForm';

import { Dishes } from './components/dishes/Dishes';

import { ClientData, Order, OrderData } from './components/form/order/OrderForm';

import { Orders } from './components/orders/Orders';

function App() {

const [state, setSelectedView] = useState<States>(States.READ);

type StatesMap = {

[key: string]: JSX.Element

}

const statesMap: StatesMap = {

[States.READ]: <Dishes appState={States.READ} />,

[States.CREATE]: <div className="w-full h-full flex items-center justify-center">

<DishForm onSubmit={(data) => {

const postData = (formData: DishData) => {

fetch("/api/dishes", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(formData)

})

.then(resp => resp.json())

.then(json => console.log(json))

.then(() => {

document.dispatchEvent(

new CustomEvent<DishData>("dishCreated", {

detail: data

})

);

})

}

postData(data);

}} />

</div>,

[States.UPDATE]: <Dishes appState={States.UPDATE} />,

[States.DELETE]: <Dishes appState={States.DELETE} />,

[States.ORDERS]: <Orders />

};

useEffect(() => {

const dishCreatedEventListener = (e: CustomEvent<DishData>) => {

setSelectedView(States.READ);

document.dispatchEvent(new CustomEvent<States>("stateChanged", { detail: States.READ }));

}

document.addEventListener("dishCreated", dishCreatedEventListener as EventListener);

return () => {

document.removeEventListener("dishCreated",

dishCreatedEventListener as EventListener);

}

}, []);

useEffect(() => {

const handleOrderCreateEventListener = (e: CustomEvent<Order>) => {

const postData = (url: string, data: ClientData | OrderData) => {

return fetch(url, {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(data)

})

};

const order = e.detail;

postData("/api/clients", e.detail.clientData)

.then(resp => resp.json())

.then(json => {

const clientResponse = json as ClientData;

order.orderData.client = clientResponse.\_links.self.href;

postData("/api/orders", e.detail.orderData)

.then(resp => resp.json())

.then(() => {

setSelectedView(States.ORDERS);

document.dispatchEvent(new CustomEvent<States>("stateChanged", { detail: States.ORDERS }));

});

});

};

document.addEventListener("orderCreated", handleOrderCreateEventListener as EventListener);

return () => document.removeEventListener("orderCreated", handleOrderCreateEventListener as EventListener);

}, [])

return (

<>

<div className="main-container">

<NavigationPanel onChangeState={(selectedLink: States) => {

setSelectedView(selectedLink);

document.dispatchEvent(new CustomEvent<States>("stateChanged", { detail: selectedLink }));

}} />

<Container content={

<>

{

statesMap[state]

}

</>

} />

</div>

</>

);

}

export default App

import "./Navigation.css";

import icon from "../../assets/BurgerIcon.png";

import { States } from "../../Constans";

import { useEffect, useState } from "react";

export const NavigationPanel = ({ onChangeState }: { onChangeState: (state: States) => void }): JSX.Element => {

const [state, setSelectedState] = useState<States>(States.READ);

const setState = (e: React.MouseEvent<HTMLAnchorElement, MouseEvent>, state: States): void => {

e.preventDefault();

setSelectedState(state);

onChangeState(state);

}

useEffect(() => {

const handleChangeState = (e: CustomEvent<States>) => {

setSelectedState(e.detail);

}

document.addEventListener("stateChanged", handleChangeState as EventListener);

return () => document.removeEventListener("stateChanged", handleChangeState as EventListener);

}, []);

return (

<div className="navbar">

<div className="icon-container">

<a className="icon" href="" onClick={(e) => setState(e, States.READ)}>

<img src={icon} alt="Icon" />

</a>

</div>

import "./Navigation.css";

import icon from "../../assets/BurgerIcon.png";

import { States } from "../../Constans";

import { useEffect, useState } from "react";

export const NavigationPanel = ({ onChangeState }: { onChangeState: (state: States) => void }): JSX.Element => {

const [state, setSelectedState] = useState<States>(States.READ);

const setState = (e: React.MouseEvent<HTMLAnchorElement, MouseEvent>, state: States): void => {

e.preventDefault();

setSelectedState(state);

onChangeState(state);

}

useEffect(() => {

const handleChangeState = (e: CustomEvent<States>) => {

setSelectedState(e.detail);

}

document.addEventListener("stateChanged", handleChangeState as EventListener);

return () => document.removeEventListener("stateChanged", handleChangeState as EventListener);

}, []);

return (

<div className="navbar">

<div className="icon-container">

<a className="icon" href="" onClick={(e) => setState(e, States.READ)}>

<img src={icon} alt="Icon" />

</a>

</div>

<div className="links-container">

<a className={"link w-full" + (state === States.CREATE ? " selected" : "")} href="" onClick={(e) => setState(e, States.CREATE)}>Create</a>

<a className={"link w-full " + (state === States.UPDATE ? " selected" : "")} href="" onClick={(e) => setState(e, States.UPDATE)}>Update</a>

<a className={"link w-full " + (state === States.DELETE ? " selected" : "")} href="" onClick={(e) => setState(e, States.DELETE)}>Delete</a>

<a className={"link w-full " + (state === States.ORDERS ? " selected" : "")} href="" onClick={(e) => setState(e, States.ORDERS)}>Orders</a>

</div>

</div >

);

} <div className="links-container">

<a className={"link w-full" + (state === States.CREATE ? " selected" : "")} href="" onClick={(e) => setState(e, States.CREATE)}>Create</a>

<a className={"link w-full " + (state === States.UPDATE ? " selected" : "")} href="" onClick={(e) => setState(e, States.UPDATE)}>Update</a>

<a className={"link w-full " + (state === States.DELETE ? " selected" : "")} href="" onClick={(e) => setState(e, States.DELETE)}>Delete</a>

<a className={"link w-full " + (state === States.ORDERS ? " selected" : "")} href="" onClick={(e) => setState(e, States.ORDERS)}>Orders</a>

</div>

</div >

);

}