федеральное государственное бюджетное образовательное учреждение высшего образования

«**Вологодский государственный университет**»

университетский колледж

Информационные системы и программирование

(наименование специальности)

**ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

Код работы отчета 09.02.07.13.00.26.2025\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Код специальности, код УК, регистрационный номер по журналу, год

Руководитель Смирнова Валентина Александровна

(уч. степень, звание, должность. Ф.И.О)

Выполнил (а) обучающийся Дронтусов Виктор Евгеньевич

(Ф.И.О)

Группа, курс ИСП 42–21, 4 курс

Дата сдачи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка по защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

Вологда

2025 г

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc198201694)

[ВВЕДЕНИЕ 3](#_Toc198201695)

[1 ЗНАКОМСТВО С ПРЕДПРИЯТИЕМ. ЗАКРЕПЛЕНИЕ РАБОЧЕГО МЕСТА 4](#_Toc198201696)

[1.1 Инструктаж по охране труда 4](#_Toc198201697)

[1.2 Описание предприятия и рабочего места 5](#_Toc198201698)

[2 ИЗУЧЕНИЕ ТЕХНИКО-ЭКОНОМИЧЕСКОЙ ХАРАКТЕРИСТИКИ ПРЕДПРИЯТИЯ 6](#_Toc198201699)

[3 АНАЛИЗ ИСПОЛЬЗУЕМЫХ СРЕДСТВ ОБРАБОТКИ ИНФОРМАЦИИ 9](#_Toc198201700)

[3.1 Описание предметной области 9](#_Toc198201701)

[3.2 Требования к программному обеспечению 10](#_Toc198201702)

[3.3 Обоснование выбора средств разработки 11](#_Toc198201703)

[4 АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ, СУЩЕСТВУЮЩИХ НА ПРЕДПРИЯТИИ 13](#_Toc198201704)

[5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 14](#_Toc198201705)

[6 ВЫПОЛНЕНИЕ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 49](#_Toc198201706)

[7 СБОР И ОБОБЩЕНИЕ МАТЕРИАЛА ДЛЯ ДИПЛОМНОГО ПРОЕКТИРОВАНИЯ 51](#_Toc198201707)

[ЗАКЛЮЧЕНИЕ 54](#_Toc198201708)

ВВЕДЕНИЕ

Производственная практика – это важная часть учебного процесса, позволяющая студентам закрепить теоретические знания на практике, приобрести профессиональные умения и познакомиться с реальной работой в выбранной сфере.

С 21 апреля по 17 мая 2025 года я проходил практику в компании ООО «Макси инновации» в должности программиста в отделе разработки программного обеспечения.

Целью преддипломной производственной практики является закрепление навыков разработки и поддержания компьютерных систем.

Задачи, которые необходимо решить для достижения поставленной цели:

* ознакомиться со структурой предприятия и пройти инструктаж по охране труда;
* изучить технико-экономическую характеристику предприятия;
* проанализировать используемые средства обработки информации;
* провести администрирование базы данных предприятия;
* разработать программные модули программного обеспечения;
* выполнить индивидуальное задание;
* собрать и обобщить материалы для дипломного проектирования.

1 ЗНАКОМСТВО С ПРЕДПРИЯТИЕМ. ЗАКРЕПЛЕНИЕ РАБОЧЕГО МЕСТА

1.1 Инструктаж по охране труда

В первую очередь в ходе прохождения производственной практики основное внимание было уделено изучению инструкций по охране труда и технике безопасности. В данных инструкциях подробно излагались предписания, регламентирующие разрешённые и запрещённые действия в рабочей зоне. Ключевые и наиболее значимые положения включали:

* требовалось поддерживать чистоту в рабочей зоне, избегая размещения посторонних объектов на поверхности стола;
* обязательной процедурой являлась проверка исправности электрооборудования перед его подключением к сети;
* категорически воспрещалось начинать трудовую деятельность при наличии признаков алкогольного или наркотического опьянения;
* курение в пределах рабочего пространства было строго запрещено;
* дисплей следовало располагать на 5 градусов ниже линии взгляда, соблюдая прямую плоскость расположения;
* оптимальное расстояние между глазами пользователя и монитором определялось диапазоном 60-80 см;
* требовалось активировать блокировку учётной записи при каждом временном отсутствии на рабочем месте;
* использование оборудования коллег без официального разрешения находилось под запретом;
* самостоятельный разбор технических устройств строго воспрещался;
* недопустимым считалось воздействие жидкостей на электротехнические устройства;
* запрещено подключать любые USB носители.

1.2 Описание предприятия и рабочего места

Компания ООО «Макси инновации» занимается разработкой программного обеспечения для розничной сети «Макси», состоящей из более чем 70 магазинов в 18 городах России. Сеть продаёт товары, закупаемые у сторонних поставщиков, а также продукцию собственного производства. Кроме розничной торговли, компания «Макси» работает в сфере строительства коммерческой и жилой недвижимости, а также занимается оптовыми поставками.

ООО «Макси инновации» разрабатывает различные программные продукты для автоматизации бизнес-процессов. В их числе — система управления торговыми операциями, корпоративная соцсеть с материалами для сотрудников, мобильные приложения и сайт для клиентов с возможностью просмотра ассортимента, оформления заказов и применения скидок, а также мобильная торговая платформа для терминалов сбора данных, которые используют сотрудники магазинов.

Практика проходила в бизнес-центре города Вологды по адресу улица Молодёжная, дом 25, кабинет 225. Для выполнения задач был предоставлен персональный компьютер с операционной системой Windows 10, доступ к корпоративной Wi-Fi-сети и учётная запись в системе GitLab.

2 ИЗУЧЕНИЕ ТЕХНИКО-ЭКОНОМИЧЕСКОЙ ХАРАКТЕРИСТИКИ ПРЕДПРИЯТИЯ

Технико-экономическое обоснование разработки – совокупность методов исследования и производства расчетов по оценке условий инвестирования с целью выбора оптимального варианта проекта и принятия решения о целесообразности его реализации. В данном разделе рассмотрим вопросы, связанные с экономическим обоснованием разработки.

Исходные данные для дальнейших расчетов приведены в соответствии с таблицей.

Таблица – Исходные данные

|  |  |  |
| --- | --- | --- |
| №п/п | Показатель | Значение |
| 1 | Стоимость ПК, руб. | 70000 |
| 2 | Срок полезного использования, лет | 3 |
| 3 | Число рабочих дней в году, дн. | 247 |
| 4 | Количество часов работы ПК в день, час | 6 |
| 5 | Заработная плата, в час | 315 |
| 6 | Начисления на заработную плату, % | 30,2 |
| 7 | Коэффициент накладных расходов | 0,1 |
| 8 | Плановая рентабельность, % | 20 |
| 9 | Затраты машинного времени на отладку программы, % | 10 |
| 10 | Расход электроэнергии в час, кВт | 0,5 |
| 11 | Стоимость кВт часа электроэнергии, руб. | 6,37 |

Определение затрат времени по стадиям разработки проекта представлено в соответствии с таблицей.

Таблица – Определение затрат времени по стадиям разработки проекта

|  |  |  |  |
| --- | --- | --- | --- |
| Стадия разработки проекта | Затраты времени, чел./часы | Поправочный коэффициент | Затраты времени с учетом коэффициента |
| 1. Разработка технического задания | 41 |  |  |
| 1.1. Затраты времени разработчика постановки задачи | 15 | 0,65 | 9,75 |

Продолжение таблицы

|  |  |  |  |
| --- | --- | --- | --- |
| 1.2. Затраты времени разработчика программного обеспечения | 30 | 0,35 | 10,5 |
| 2. Разработка эскизного проекта | 35 |  |  |
| 2.1. Затраты времени разработчика постановки задачи | 8 | 0,7 | 5,6 |
| 2.2. Затраты времени разработчика программного обеспечения | 28 | 0,3 | 8,4 |
| 3. Разработка технорабочего проекта | 310 |  |  |
| 3.1.Подготовка контента программного обеспечения | 7 | 0,2 | 1,4 |
| 3.2. Создание | 208 | 0,4 | 83,2 |
| 3.3. Оптимизация | 93 | 0,4 | 36,2 |
| 4. Внедрение | 9 |  |  |
| 4.1. Затраты времени разработчика постановки задачи | 7 | 0,5 | 3,5 |
| 4.2. Затраты времени разработчика программного обеспечения | 3 | 0,5 | 1,5 |
| Итого: | 399 |  | 160,05 |

Произведем расчет затрат.

С – стоимость электроэнергии рассчитывается по формуле (1):

|  |  |
| --- | --- |
| , руб., | (1) |

где – расход электроэнергии в час (по паспортным данным ПК), кВт;

– стоимость кВт часа электроэнергии.

руб.

Стоимость одного машинного часа рассчитывается по формуле (2):

|  |  |
| --- | --- |
| , руб., | (2) |

где – балансовая стоимость ПК;

– срок полезного использования;

– число рабочих дней в году;

– число часов работы ПК в день;

– стоимость электроэнергии.

= 18,93 руб.

Затраты, связанные с созданием данного программного продукта, рассчитываются по формуле (3):

|  |  |
| --- | --- |
| , руб., | (3) |

где затраты на создание программного продукта;

– время на составление программного продукта (чел/час);

– средняя заработная плата программиста (часовая);

– начисления на заработную плату;

– коэффициент накладных расходов;

– стоимость одного часа машинного времени;

– затраты машинного времени на отладку программы.

руб.

Договорная цена определяется по формуле (4):

|  |  |
| --- | --- |
| , руб., | (4) |

где R *–* плановая рентабельность.

руб.

Расчеты, которые были проведены в ходе выполнения дипломного проекта, указывают на то, что проект является экономически обоснованным, так как договорная цена выше, чем затраты проекта.

3 АНАЛИЗ ИСПОЛЬЗУЕМЫХ СРЕДСТВ ОБРАБОТКИ ИНФОРМАЦИИ

3.1 Описание предметной области

Конечные пользователи разрабатываемого сервиса — путешественники, командировочные сотрудники и туристы, которым необходимо быстро и удобно подбирать и бронировать отели в разных городах и странах. Они сталкиваются с такими проблемами, как сложность сравнения вариантов, отсутствие актуальной информации о наличии мест, неясные условия бронирования и отмены, а также ограниченные фильтры для персонализации поиска. Отели в системе могут быть представлены с различными параметрами: тип размещения (отель, апартаменты, хостел), удобства (Wi-Fi, завтрак, парковка) и отзывы гостей.

Каждое предложение включает детализированное описание: фотографии, список услуг, политику отмены, варианты оплаты и расположение на карте. Пользователь может фильтровать отели по цене, рейтингу или конкретным удобствам, а также сравнивать условия бронирования на разные даты.

Существующие решения, такие как Booking.com или Airbnb, предоставляют широкий выбор жилья, но не всегда учитывают индивидуальные предпочтения пользователей в гибком поиске. Например, агрегаторы вроде Expedia или Agoda предлагают пакетные туры, но не всегда позволяют тонко настроить параметры бронирования. Другие платформы, такие как Trivago, сравнивают цены, но не дают удобных инструментов для анализа отзывов или быстрого изменения условий бронирования. Это создаёт потребность в сервисе, который объединяет удобный поиск, детальную аналитику вариантов и гибкие условия бронирования в одном интерфейсе.

Таким образом, ключевой задачей системы станет предоставление интуитивно понятного инструмента для подбора отелей с учётом персональных требований пользователя, мгновенным подтверждением брони и возможностью адаптации параметров поиска в реальном времени.

3.2 Требования к программному обеспечению

Приложение должно быть разработано как веб-сайт с использованием языка программирования JavaScript. Среда разработки — WebStorm. Клиентская часть реализуется с помощью фреймворка React, серверная часть — на Express.js. Для хранения данных необходимо использовать реляционную базу данных MySQL.

В системе определены три основные роли: гость, зарегистрированный пользователь. Гость может просматривать список отелей с возможностью фильтрации по цене, рейтингу, удобствам и местоположению; бронировать номера с указанием дат заезда и выезда; проверять доступность номеров на выбранные даты; регистрироваться или входить в существующий аккаунт. Зарегистрированный пользователь может бронировать номера с указанием дат заезда и выезда; отменять бронирования в соответствии с условиями отеля; оставлять отзывы и оценки; добавлять отели в избранное; настраивать профиль; просматривать историю бронирований; добавлять способы оплаты.

Интерфейс должен включать главную страницу с формой поиска и блоками специальных предложений, страницу отеля с фотогалереей, описанием, отзывами и формой бронирования, а также личный кабинет пользователя с историей бронирований, настройками профиля и способами оплаты. Система должна поддерживать адаптивный дизайн для устройств с различными разрешениями экрана — от 320×568 до 1920×1080 пикселей. Время отклика сервера не должно превышать 10 секунд при стандартной нагрузке. Все передаваемые данные должны шифроваться с использованием HTTPS.

Ограничения системы: максимальное количество комнат для бронирования одним пользователем на один отель — 3 штуки; максимальное количество гостей — 5 человек; бронирование не доступно на прошлые даты; система не поддерживает офлайн-работу. Интерфейс должен быть локализован на русский и английский языки.

3.3 Обоснование выбора средств разработки

Для хранения данных системы бронирования отелей необходима база данных. Среди популярных решений для веб-приложений можно выделить MySQL, PostgreSQL, MongoDB и Firebase Realtime Database.

MySQL – это реляционная СУБД с открытым исходным кодом, которая идеально подходит для веб-приложений. Она поддерживает стандартные SQL-запросы, обладает высокой производительностью и надежностью. MySQL хорошо масштабируется и имеет широкую поддержку в хостинг-средах. Однако при очень высоких нагрузках может потребоваться оптимизация запросов и индексов.

PostgreSQL – это более продвинутая реляционная СУБД с поддержкой сложных типов данных и пользовательских функций. Она предлагает лучшую масштабируемость и надежность по сравнению с MySQL, но требует больше ресурсов и имеет более сложную настройку.

MongoDB – это документоориентированная NoSQL база данных, которая хорошо подходит для хранения неструктурированных данных. Она обеспечивает высокую производительность при работе с большими объемами данных, но менее подходит для сложных реляционных запросов, которые могут потребоваться в системе бронирования.

Firebase Realtime Database – это облачное NoSQL-решение от Google, которое обеспечивает синхронизацию данных в реальном времени. Оно удобно для мобильных приложений, но имеет ограничения по структуре данных и может быть дорогим при масштабировании.

Для клиентской части веб-приложения рассматривались такие фреймворки, как React, Angular и Vue.js. React был выбран благодаря своей гибкости, высокой производительности и богатой экосистеме. Он использует компонентный подход, что упрощает разработку и поддержку интерфейса. Angular предлагает более полный фреймворк, но имеет более высокий порог входа. Vue.js проще в изучении, но имеет меньшую экосистему по сравнению с React.

Для серверной части рассматривались Node.js с Express, Django и Ruby on Rails. Node.js с Express был выбран благодаря его производительности, асинхронной модели и возможности использовать JavaScript как на клиенте, так и на сервере. Django и Ruby on Rails предлагают больше встроенных функций, но менее гибки и требуют знания Python/Ruby соответственно.

Таким образом, для системы бронирования отелей был выбран следующий стек технологий: MySQL в качестве СУБД благодаря ее надежности и производительности для реляционных данных, React для клиентской части из-за его гибкости и производительности, и Node.js с Express для серверной части благодаря его эффективности и унификации языка программирования. WebStorm был выбран как IDE, поскольку предоставляет отличную поддержку JavaScript и React-разработки.

4 АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ, СУЩЕСТВУЮЩИХ НА ПРЕДПРИЯТИИ

Была создана база данных со следующими сущностями:

* users (пользователи).

Сущность users содержит информацию о пользователях: уникальный идентификатор (id), имя (firstname), фамилия (lastname), электронная почта (email), пароль (password), номер телефона (phone\_number), страна (country). Каждый пользователь представлен отдельной записью, что позволяет однозначно идентифицировать его в системе, в соответствии с рисунком 1.

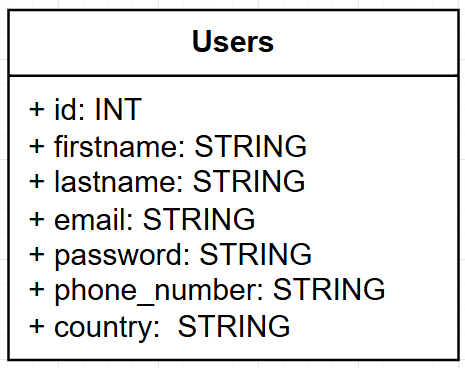


Рисунок 1 – ER-диаграмма сущностей в базе данных

Таким образом, база данных готова к интеграции с приложением, обеспечивая надёжное хранение и обработку данных о пользователях.

5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Во время прохождения преддипломной производственной практики было необходимо разработать прототип веб-приложения для бронирования отелей.

Идея приложения заключается в создании удобной платформы, где пользователи могут быстро находить и бронировать отели по всему миру, сравнивать цены и условия, а также оставлять отзывы. Это позволит упростить процесс поиска и бронирования жилья, объединив в одном сервисе каталог отелей, систему бронирования и отзывов.

Приложение разрабатывалось с использованием языка программирования JavaScript. Для клиентской части был использован фреймворк React, для серверной - Express.js. В качестве базы данных применялась MySQL.

В первую очередь был создан главный экран, содержащий основную информацию. Главный экран в соответствии с рисунком 2.

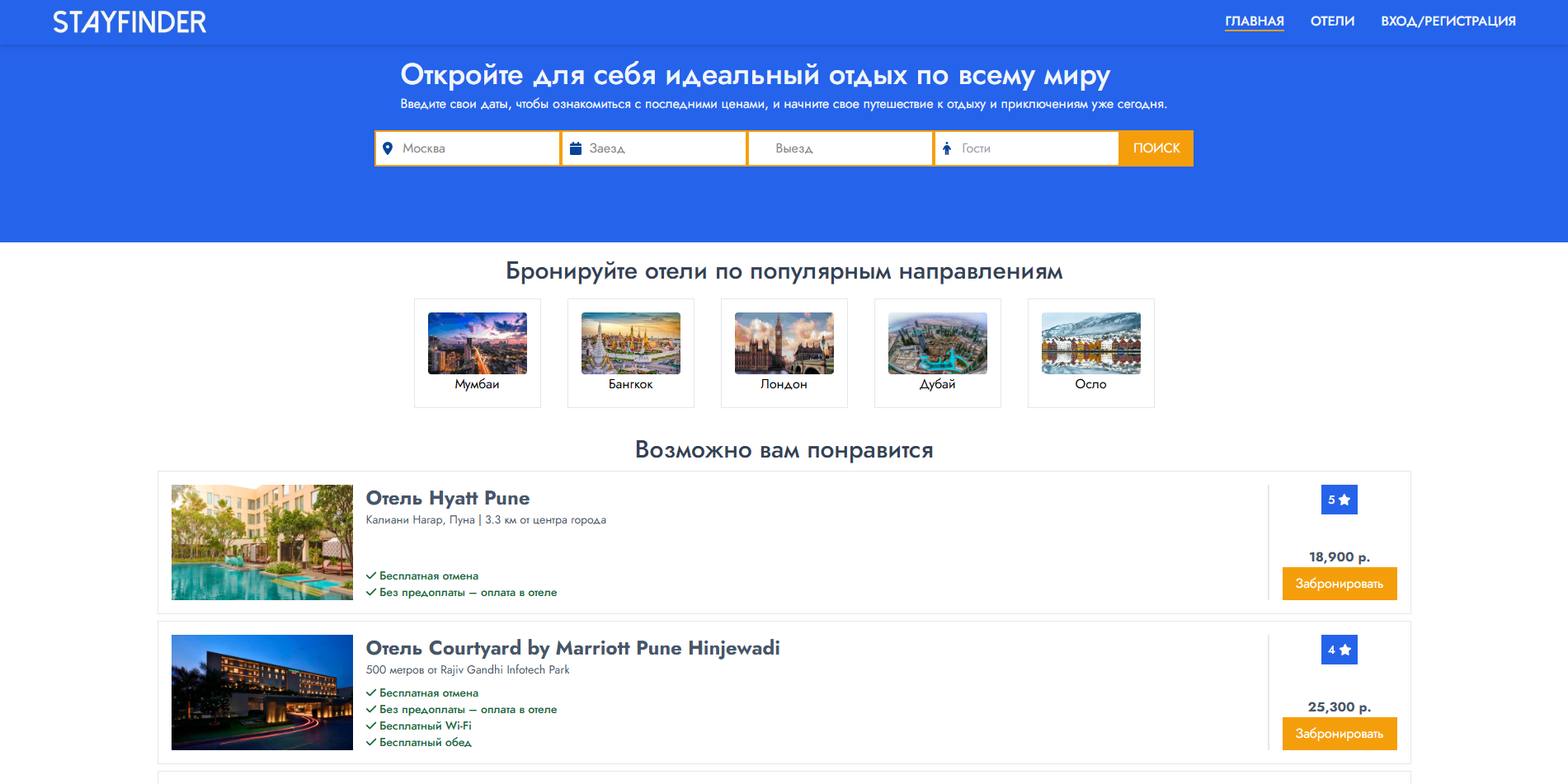


Рисунок 2 – Главный экран

Код главного экрана:

import HeroCover from './components/hero-cover/HeroCover';

import PopularLocations from './components/popular-locations/popular-locations';

import { networkAdapter } from 'services/NetworkAdapter';

import { useState, useEffect, useCallback } from 'react';

import { MAX\_GUESTS\_INPUT\_VALUE } from 'utils/constants';

import ResultsContainer from 'components/results-container/ResultsContainer';

import { formatDate } from 'utils/date-helpers';

import { useNavigate } from 'react-router-dom';

import \_debounce from 'lodash/debounce';

const Home = () => {

const navigate = useNavigate();

const [isDatePickerVisible, setisDatePickerVisible] = useState(false);

const [locationInputValue, setLocationInputValue] = useState('москва');

const [numGuestsInputValue, setNumGuestsInputValue] = useState('');

const [popularDestinationsData, setPopularDestinationsData] = useState({

isLoading: true,

data: [],

errors: [],

});

const [hotelsResults, setHotelsResults] = useState({

isLoading: true,

data: [],

errors: [],

});

const [availableCities, setAvailableCities] = useState([]);

const [filteredTypeheadResults, setFilteredTypeheadResults] = useState([]);

const debounceFn = useCallback(\_debounce(queryResults, 1000), []);

const [dateRange, setDateRange] = useState([

{

startDate: null,

endDate: null,

key: 'selection',

},

]);

const onDatePickerIconClick = () => {

setisDatePickerVisible(!isDatePickerVisible);

};

const onLocationChangeInput = async (newValue) => {

setLocationInputValue(newValue);

debounceFn(newValue, availableCities);

};

function queryResults(query, availableCities) {

const filteredResults = availableCities.filter((city) =>

city.toLowerCase().includes(query.toLowerCase())

);

setFilteredTypeheadResults(filteredResults);

}

const onNumGuestsInputChange = (numGuests) => {

if (

(numGuests < MAX\_GUESTS\_INPUT\_VALUE && numGuests > 0) ||

numGuests === ''

) {

setNumGuestsInputValue(numGuests);

}

};

const onDateChangeHandler = (ranges) => {

setDateRange([ranges.selection]);

};

const onSearchButtonAction = () => {

const numGuest = Number(numGuestsInputValue);

const checkInDate = formatDate(dateRange[0].startDate) ?? '';

const checkOutDate = formatDate(dateRange[0].endDate) ?? '';

const city = locationInputValue;

navigate('/hotels', {

state: {

numGuest,

checkInDate,

checkOutDate,

city,

},

});

};

useEffect(() => {

const getInitialData = async () => {

const popularDestinationsResponse = await networkAdapter.get(

'/api/popularDestinations'

);

const hotelsResultsResponse =

await networkAdapter.get('/api/nearbyHotels');

const availableCitiesResponse = await networkAdapter.get(

'/api/availableCities'

);

if (availableCitiesResponse) {

setAvailableCities(availableCitiesResponse.data.elements);

}

if (popularDestinationsResponse) {

setPopularDestinationsData({

isLoading: false,

data: popularDestinationsResponse.data.elements,

errors: popularDestinationsResponse.errors,

});

}

if (hotelsResultsResponse) {

setHotelsResults({

isLoading: false,

data: hotelsResultsResponse.data.elements,

errors: hotelsResultsResponse.errors,

});

}

};

getInitialData();

}, []);

return (

<>

<HeroCover

locationInputValue={locationInputValue}

numGuestsInputValue={numGuestsInputValue}

locationTypeheadResults={filteredTypeheadResults}

isDatePickerVisible={isDatePickerVisible}

setisDatePickerVisible={setisDatePickerVisible}

onLocationChangeInput={onLocationChangeInput}

onNumGuestsInputChange={onNumGuestsInputChange}

dateRange={dateRange}

onDateChangeHandler={onDateChangeHandler}

onDatePickerIconClick={onDatePickerIconClick}

onSearchButtonAction={onSearchButtonAction}

/>

<div className="container mx-auto">

<PopularLocations popularDestinationsData={popularDestinationsData} />

<div className="my-8">

<h2 className="text-3xl font-medium text-slate-700 text-center my-2">

Возможно вам понравится

</h2>

<ResultsContainer

hotelsResults={hotelsResults}

enableFilters={false}

/>

</div>

</div>

</>

);

};

export default Home;

Далее была разработана страница поиска отелей, где пользователь ищет необходимый отель с фильтрами и сортировкой в соответствии с рисунком 3.

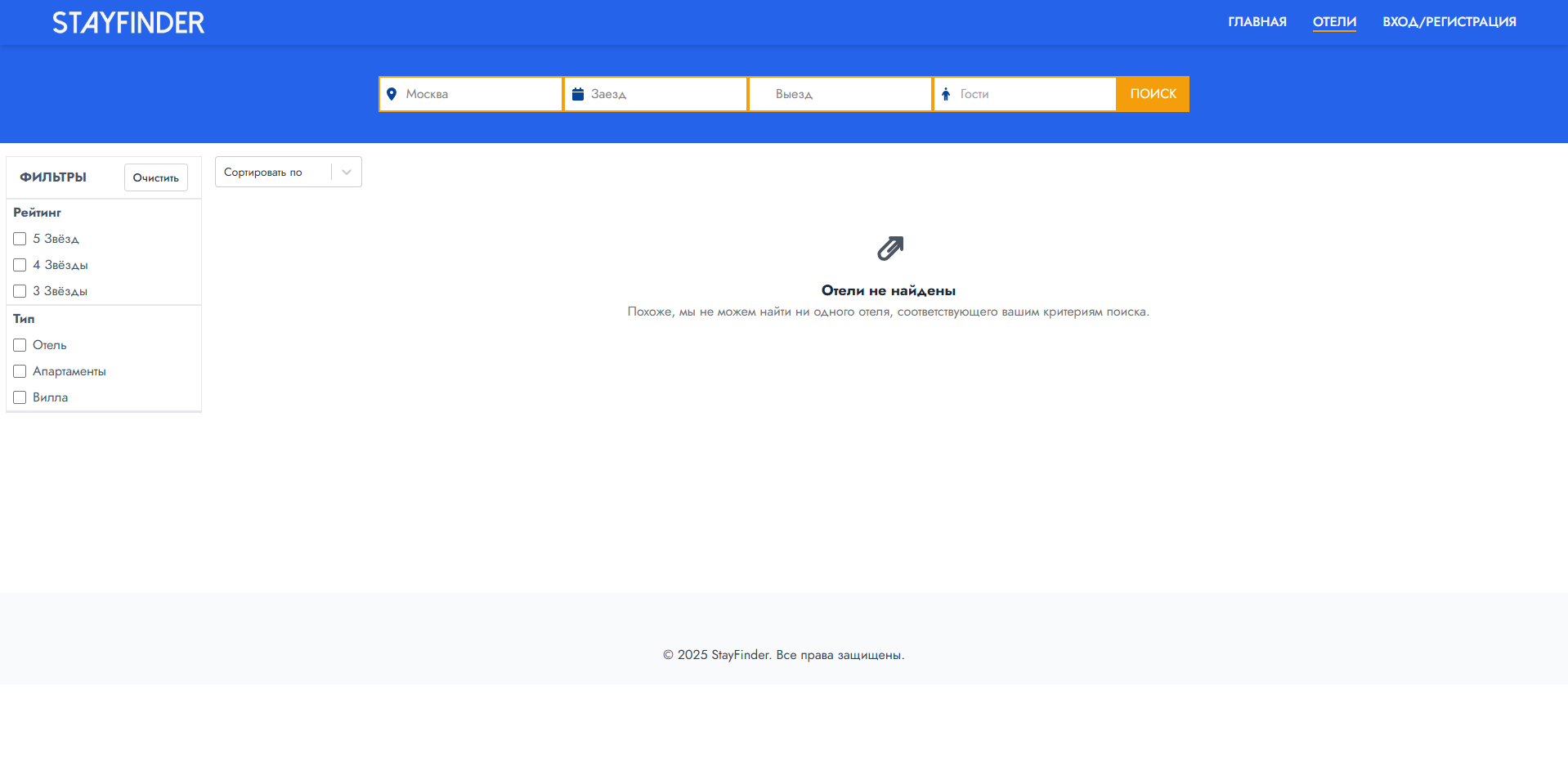


Рисунок 3 – Страница поиска

Код экрана со списком книг:

import React, { useState, useEffect, useCallback } from 'react';

import GlobalSearchBox from 'components/global-search-box/GlobalSearchbox';

import ResultsContainer from 'components/results-container/ResultsContainer';

import { networkAdapter } from 'services/NetworkAdapter';

import isEmpty from 'utils/helpers';

import { MAX\_GUESTS\_INPUT\_VALUE } from 'utils/constants';

import { formatDate } from 'utils/date-helpers';

import { useLocation, useSearchParams } from 'react-router-dom';

import { parse } from 'date-fns';

import PaginationController from 'components/ux/pagination-controller/PaginationController';

import { SORTING\_FILTER\_LABELS } from 'utils/constants';

import \_debounce from 'lodash/debounce';

const HotelsSearch = () => {

const [isDatePickerVisible, setisDatePickerVisible] = useState(false);

const [locationInputValue, setLocationInputValue] = useState('москва');

const [numGuestsInputValue, setNumGuestsInputValue] = useState('');

const [availableCities, setAvailableCities] = useState([]);

const [currentResultsPage, setCurrentResultsPage] = useState(1);

const [filtersData, setFiltersData] = useState({

isLoading: true,

data: [],

errors: [],

});

const [hotelsResults, setHotelsResults] = useState({

isLoading: true,

data: [],

errors: [],

});

const [dateRange, setDateRange] = useState([

{

startDate: null,

endDate: null,

key: 'selection',

},

]);

const [sortByFilterValue, setSortByFilterValue] = useState({

value: 'default',

label: 'Сортировать по',

});

const [selectedFiltersState, setSelectedFiltersState] = useState({});

const [filteredTypeheadResults, setFilteredTypeheadResults] = useState([]);

const debounceFn = useCallback(\_debounce(queryResults, 1000), []);

const [searchParams, setSearchParams] = useSearchParams();

const location = useLocation();

const sortingFilterOptions = [

{ value: 'default', label: 'Сортировать по' },

{ value: 'priceLowToHigh', label: SORTING\_FILTER\_LABELS.PRICE\_LOW\_TO\_HIGH },

{ value: 'priceHighToLow', label: SORTING\_FILTER\_LABELS.PRICE\_HIGH\_TO\_LOW },

];

const onSortingFilterChange = (selectedOption) => {

setSortByFilterValue(selectedOption);

};

const onFiltersUpdate = (updatedFilter) => {

setSelectedFiltersState(

selectedFiltersState.map((filterGroup) => {

if (filterGroup.filterId === updatedFilter.filterId) {

return {

...filterGroup,

filters: filterGroup.filters.map((filter) => {

if (filter.id === updatedFilter.id) {

return {

...filter,

isSelected: !filter.isSelected,

};

}

return filter;

}),

};

}

return filterGroup;

})

);

};

const onDateChangeHandler = (ranges) => {

setDateRange([ranges.selection]);

};

const onSearchButtonAction = () => {

const activeFilters = getActiveFilters();

const numGuest = Number(numGuestsInputValue);

const checkInDate = formatDate(dateRange.startDate) ?? '';

const checkOutDate = formatDate(dateRange.endDate) ?? '';

setSearchParams({

city: locationInputValue,

numGuests: numGuestsInputValue,

});

fetchHotels({

city: locationInputValue,

...activeFilters,

guests: numGuest,

checkInDate,

checkOutDate,

});

};

const getActiveFilters = () => {

const filters = {};

selectedFiltersState.forEach((category) => {

const selectedValues = category.filters

.filter((filter) => filter.isSelected)

.map((filter) => filter.value);

if (selectedValues.length > 0) {

filters[category.filterId] = selectedValues;

}

});

if (!isEmpty(filters)) {

return filters;

}

return null;

};

const onDatePickerIconClick = () => {

setisDatePickerVisible(!isDatePickerVisible);

};

const onLocationChangeInput = async (newValue) => {

setLocationInputValue(newValue);

debounceFn(newValue, availableCities);

};

function queryResults(query, availableCities) {

const filteredResults = availableCities

.filter((city) => city.toLowerCase().includes(query.toLowerCase()))

.slice(0, 5);

setFilteredTypeheadResults(filteredResults);

}

const onNumGuestsInputChange = (numGuests) => {

if (numGuests < MAX\_GUESTS\_INPUT\_VALUE && numGuests > 0) {

setNumGuestsInputValue(numGuests);

}

};

const onClearFiltersAction = () => {

const hasActiveFilters = selectedFiltersState.some((filterGroup) =>

filterGroup.filters.some((filter) => filter.isSelected)

);

if (hasActiveFilters) {

setSelectedFiltersState(

selectedFiltersState.map((filterGroup) => ({

...filterGroup,

filters: filterGroup.filters.map((filter) => ({

...filter,

isSelected: false,

})),

}))

);

}

};

const fetchHotels = async (filters) => {

setHotelsResults({

isLoading: true,

data: [],

errors: [],

});

const hotelsResultsResponse = await networkAdapter.get('/api/hotels', {

filters: JSON.stringify(filters),

currentPage: currentResultsPage,

advancedFilters: JSON.stringify([

{

sortBy: sortByFilterValue.value,

},

]),

});

if (hotelsResultsResponse) {

setHotelsResults({

isLoading: false,

data: hotelsResultsResponse.data.elements,

errors: hotelsResultsResponse.errors,

metadata: hotelsResultsResponse.metadata,

pagination: hotelsResultsResponse.paging,

});

}

};

const getVerticalFiltersData = async () => {

const filtersDataResponse = await networkAdapter.get(

'api/hotels/verticalFilters'

);

if (filtersDataResponse) {

setFiltersData({

isLoading: false,

data: filtersDataResponse.data.elements,

errors: filtersDataResponse.errors,

});

}

};

const handlePageChange = (page) => {

setCurrentResultsPage(page);

};

const handlePreviousPageChange = () => {

setCurrentResultsPage((prev) => {

if (prev <= 1) return prev;

return prev - 1;

});

};

const handleNextPageChange = () => {

setCurrentResultsPage((prev) => {

if (prev >= hotelsResults.pagination.totalPages) return prev;

return prev + 1;

});

};

const fetchAvailableCities = async () => {

const availableCitiesResponse = await networkAdapter.get(

'/api/availableCities'

);

if (availableCitiesResponse) {

setAvailableCities(availableCitiesResponse.data.elements);

}

};

useEffect(() => {

fetchAvailableCities();

getVerticalFiltersData();

}, []);

useEffect(() => {

if (searchParams.get('city')) {

setLocationInputValue(searchParams.get('city'));

}

if (searchParams.get('numGuests')) {

setNumGuestsInputValue(searchParams.get('numGuests'));

}

}, [searchParams]);

useEffect(() => {

setSelectedFiltersState(

filtersData.data.map((filterGroup) => ({

...filterGroup,

filters: filterGroup.filters.map((filter) => ({

...filter,

isSelected: false,

})),

}))

);

}, [filtersData]);

useEffect(() => {

if (selectedFiltersState.length > 0) {

const activeFilters = getActiveFilters();

if (activeFilters) {

activeFilters.city = locationInputValue.toLowerCase();

fetchHotels(activeFilters);

} else {

fetchHotels({

city: locationInputValue,

});

}

}

}, [selectedFiltersState, currentResultsPage, sortByFilterValue]);

useEffect(() => {

if (location.state) {

const { city, numGuest, checkInDate, checkOutDate } = location.state;

if (numGuest) {

setNumGuestsInputValue(numGuest.toString());

}

setLocationInputValue(city);

if (checkInDate && checkOutDate) {

setDateRange([

{

startDate: parse(checkInDate, 'dd/MM/yyyy', new Date()),

endDate: parse(checkOutDate, 'dd/MM/yyyy', new Date()),

key: 'selection',

},

]);

}

}

}, [location]);

return (

<div className="hotels">

<div className="bg-blue-600 px-2 lg:h-[120px] h-[220px] flex items-center justify-center">

<GlobalSearchBox

locationInputValue={locationInputValue}

locationTypeheadResults={filteredTypeheadResults}

numGuestsInputValue={numGuestsInputValue}

isDatePickerVisible={isDatePickerVisible}

setisDatePickerVisible={setisDatePickerVisible}

onLocationChangeInput={onLocationChangeInput}

onNumGuestsInputChange={onNumGuestsInputChange}

dateRange={dateRange}

onDateChangeHandler={onDateChangeHandler}

onDatePickerIconClick={onDatePickerIconClick}

onSearchButtonAction={onSearchButtonAction}

/>

</div>

<div className="my-4"></div>

<div className="w-[180px]"></div>

<ResultsContainer

hotelsResults={hotelsResults}

enableFilters={true}

filtersData={filtersData}

onFiltersUpdate={onFiltersUpdate}

onClearFiltersAction={onClearFiltersAction}

selectedFiltersState={selectedFiltersState}

sortByFilterValue={sortByFilterValue}

onSortingFilterChange={onSortingFilterChange}

sortingFilterOptions={sortingFilterOptions}

/>

{hotelsResults.pagination?.totalPages > 1 && (

<div className="my-4">

<PaginationController

currentPage={currentResultsPage}

totalPages={hotelsResults.pagination?.totalPages}

handlePageChange={handlePageChange}

handlePreviousPageChange={handlePreviousPageChange}

handleNextPageChange={handleNextPageChange}

/>

</div>

)}

</div>

);

};

export default HotelsSearch;

Следующей на разработку стала страница самого отеля в соответствии с рисунком 4.

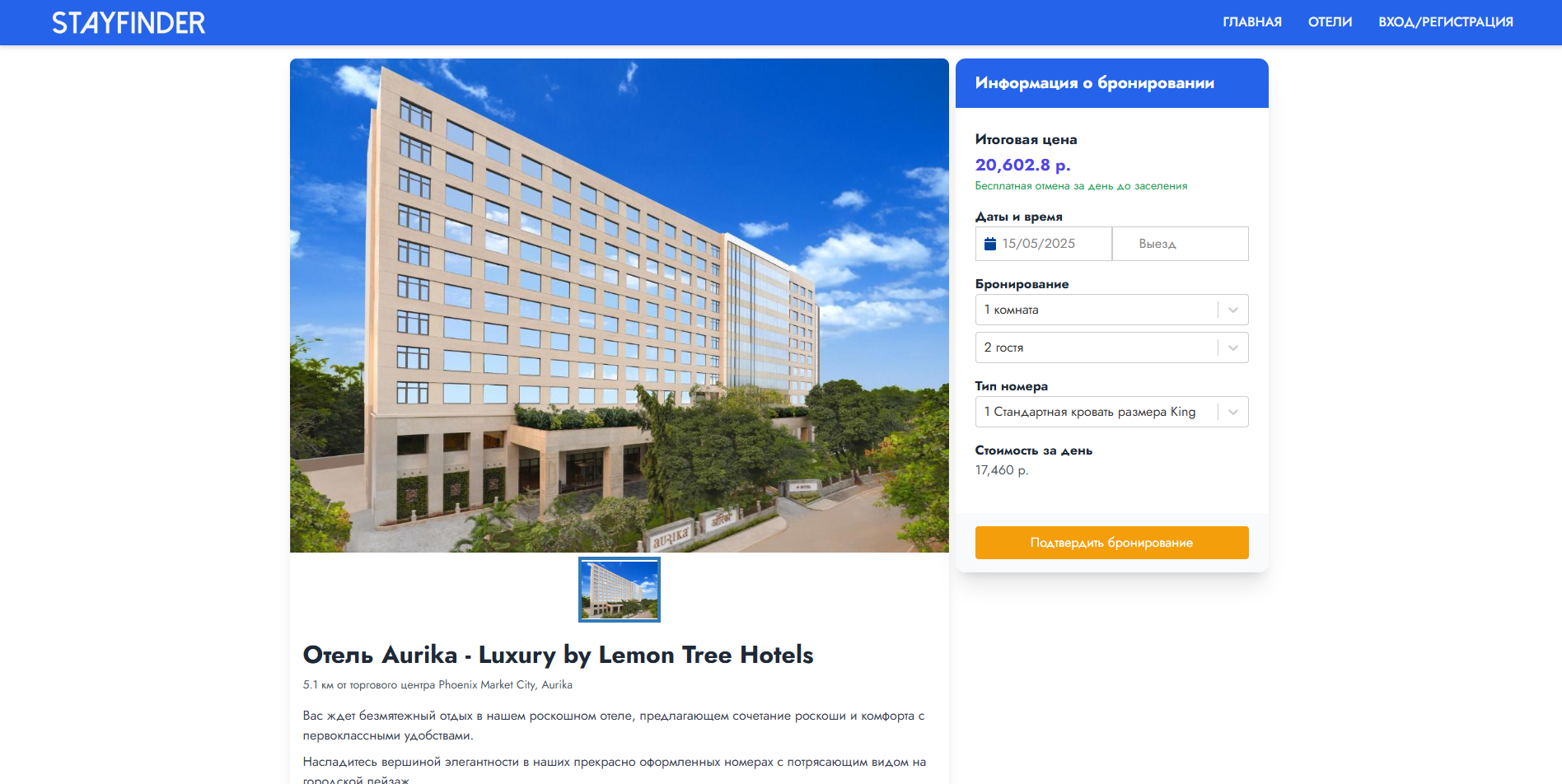


Рисунок 4 – Страница информации об отеле

Код страницы отеля:

import React, { useEffect, useState } from 'react';

import { useParams } from 'react-router-dom';

import { networkAdapter } from 'services/NetworkAdapter';

import HotelDetailsViewCard from './components/hotel-details-view-card/HotelDetailsViewCard';

import HotelDetailsViewCardSkeleton from './components/hotel-details-view-card-skeleton/HotelDetailsViewCardSkeleton';

const HotelDetails = () => {

const { hotelId } = useParams();

const [hotelDetails, setHotelDetails] = useState({

isLoading: true,

data: {},

});

useEffect(() => {

const fetchHotelDetails = async () => {

const response = await networkAdapter.get(`/api/hotel/${hotelId}`);

setHotelDetails({

isLoading: false,

data: response.data,

});

};

fetchHotelDetails();

}, [hotelId]);

return (

<>

{hotelDetails.isLoading ? (

<HotelDetailsViewCardSkeleton />

) : (

<HotelDetailsViewCard hotelDetails={hotelDetails.data} />

)}

</>

);

};

export default HotelDetails;

Далее была создана страница бронирования в соответствии с рисунком 5.

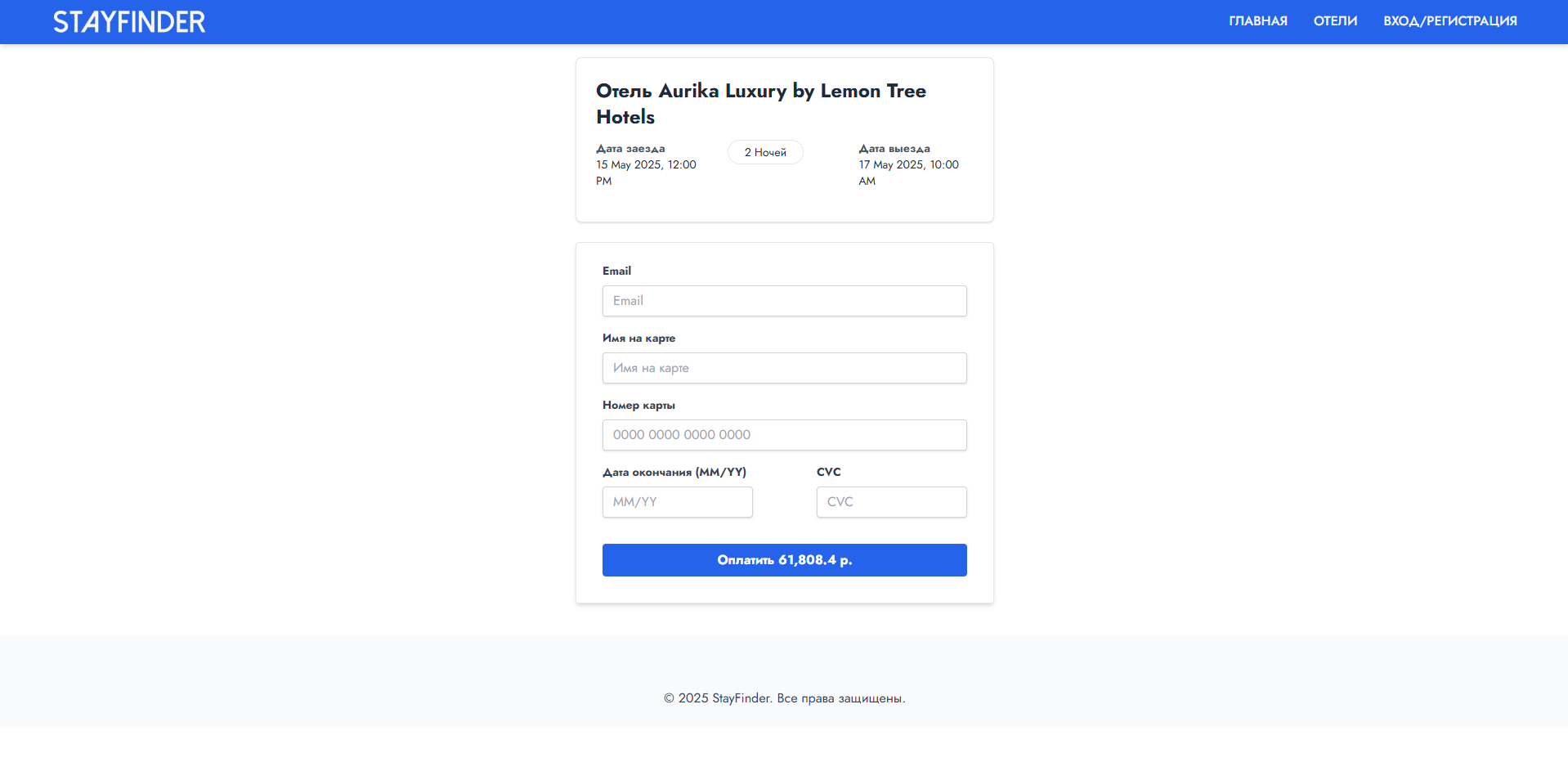


Рисунок 5 – Страница бронирования

Код страницы бронирования:

import React, { useEffect, useState } from 'react';

import FinalBookingSummary from './components/final-booking-summary/FinalBookingSummary';

import { useLocation } from 'react-router-dom';

import { useNavigate } from 'react-router-dom';

import { getReadableMonthFormat } from 'utils/date-helpers';

import { useSearchParams } from 'react-router-dom';

import { AuthContext } from 'contexts/AuthContext';

import { useContext } from 'react';

import { networkAdapter } from 'services/NetworkAdapter';

import Loader from 'components/ux/loader/loader';

import Toast from 'components/ux/toast/Toast';

const Checkout = () => {

const [errors, setErrors] = useState({});

const location = useLocation();

const navigate = useNavigate();

const [searchParams] = useSearchParams();

const [toastMessage, setToastMessage] = useState('');

const { isAuthenticated, userDetails } = useContext(AuthContext);

const [isSubmitDisabled, setIsSubmitDisabled] = useState(false);

const [paymentConfirmationDetails, setPaymentConfirmationDetails] = useState({

isLoading: false,

data: {},

});

const dismissToast = () => {

setToastMessage('');

};

const [formData, setFormData] = useState({

email: userDetails?.email ? userDetails?.email : '',

nameOnCard: '',

cardNumber: '',

expiry: '',

cvc: '',

});

const checkInDateTime = `${getReadableMonthFormat(

searchParams.get('checkIn')

)}, ${location.state?.checkInTime}`;

const checkOutDateTime = `${getReadableMonthFormat(

searchParams.get('checkOut')

)}, ${location.state?.checkOutTime}`;

useEffect(() => {

const locationState = location.state;

const checkIn = searchParams.get('checkIn');

const checkOut = searchParams.get('checkOut');

if (!locationState || !checkIn || !checkOut) {

const hotelCode = searchParams.get('hotelCode');

navigate(`/hotel/${hotelCode}`);

}

}, [location, navigate, searchParams]);

const handleChange = (e) => {

const { name, value } = e.target;

const isValid = validationSchema[name](value);

setFormData({ ...formData, [name]: value });

setErrors({ ...errors, [name]: !isValid });

};

const handleSubmit = async (e) => {

e.preventDefault();

let isValid = true;

const newErrors = {};

Object.keys(formData).forEach((field) => {

const isFieldValid = validationSchema[field](formData[field]);

newErrors[field] = !isFieldValid;

isValid = isValid && isFieldValid;

});

setErrors(newErrors);

if (!isValid) {

return;

}

setIsSubmitDisabled(true);

setPaymentConfirmationDetails({

isLoading: true,

data: {},

});

const response = await networkAdapter.post(

'/api/payments/confirmation',

formData

);

if (response && response.data && response.errors.length === 0) {

setPaymentConfirmationDetails({

isLoading: false,

data: response.data,

});

const hotelName = searchParams.get('hotelName').replaceAll('-', '\_');

navigate(`/booking-confirmation?payment=sucess&hotel=${hotelName}`, {

state: {

confirmationData: response.data,

},

});

} else {

setToastMessage('Оплата не удалась. Повторите попытку.');

setIsSubmitDisabled(false);

setPaymentConfirmationDetails({

isLoading: false,

data: {},

});

}

};

return (

<div className="flex flex-col justify-center items-center">

<FinalBookingSummary

hotelName={searchParams.get('hotelName').replaceAll('-', ' ')}

checkIn={checkInDateTime}

checkOut={checkOutDateTime}

isAuthenticated={isAuthenticated}

phone={userDetails?.phone}

email={userDetails?.email}

fullName={userDetails?.fullName}

/>

<div className="relative bg-white border shadow-md rounded px-8 pt-6 pb-8 mb-4 w-full max-w-lg mx-auto">

{paymentConfirmationDetails.isLoading && (

<Loader

isFullScreen={true}

loaderText={'Оплата в процессе, подождите!'}

/>

)}

<form

onSubmit={handleSubmit}

className={` ${

paymentConfirmationDetails.isLoading ? 'opacity-40' : ''

}`}

>

<InputField

label="Email"

type="email"

name="email"

value={formData.email}

onChange={handleChange}

placeholder="Email"

required={true}

error={errors.email}

/>

<InputField

label="Имя на карте"

type="text"

name="nameOnCard"

value={formData.nameOnCard}

onChange={handleChange}

placeholder="Имя на карте"

required={true}

error={errors.nameOnCard}

/>

<InputField

label="Номер карты"

type="text"

name="cardNumber"

value={formData.cardNumber}

onChange={handleChange}

placeholder="0000 0000 0000 0000"

required={true}

error={errors.cardNumber}

/>

<div className="flex mb-4 justify-between">

<InputField

label="Дата окончания (MM/YY)"

type="text"

name="expiry"

value={formData.expiry}

onChange={handleChange}

placeholder="MM/YY"

required={true}

error={errors.expiry}

/>

<InputField

label="CVC"

type="text"

name="cvc"

value={formData.cvc}

onChange={handleChange}

placeholder="CVC"

required={true}

error={errors.cvc}

/>

</div>

<div className="flex items-center justify-between">

<button

className={`bg-blue-600 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline w-full transition duration-300 ${

isSubmitDisabled

? 'opacity-50 cursor-not-allowed'

: 'hover:bg-blue-700'

}`}

type="submit"

disabled={isSubmitDisabled}

>

Оплатить {location.state?.total}

</button>

</div>

</form>

{toastMessage && (

<div className="my-4">

<Toast

message={toastMessage}

type={'error'}

dismissError={dismissToast}

/>

</div>

)}

</div>

</div>

);

};

const InputField = ({

label,

type,

name,

value,

onChange,

placeholder,

required,

error,

}) => (

<div className="mb-4">

<label

className="block text-gray-700 text-sm font-bold mb-2"

htmlFor={name}

>

{label}

</label>

<input

className={`shadow appearance-none border ${

error ? 'border-red-500' : 'border-gray-300'

} rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline`}

id={name}

type={type}

name={name}

value={value}

onChange={onChange}

placeholder={placeholder}

required={required}

aria-invalid={error ? 'true' : 'false'}

/>

{error && (

<p className="text-red-500 text-xs my-1">Please check this field.</p>

)}

</div>

);

const validationSchema = {

email: (value) => /\S+@\S+\.\S+/.test(value),

nameOnCard: (value) => value.trim() !== '',

cardNumber: (value) => /^\d{16}$/.test(value),

expiry: (value) => /^(0[1-9]|1[0-2])\/\d{2}$/.test(value),

cvc: (value) => /^\d{3,4}$/.test(value),

};

export default Checkout;

Следующей была разработана страница входа в соответствии с рисунком 6.

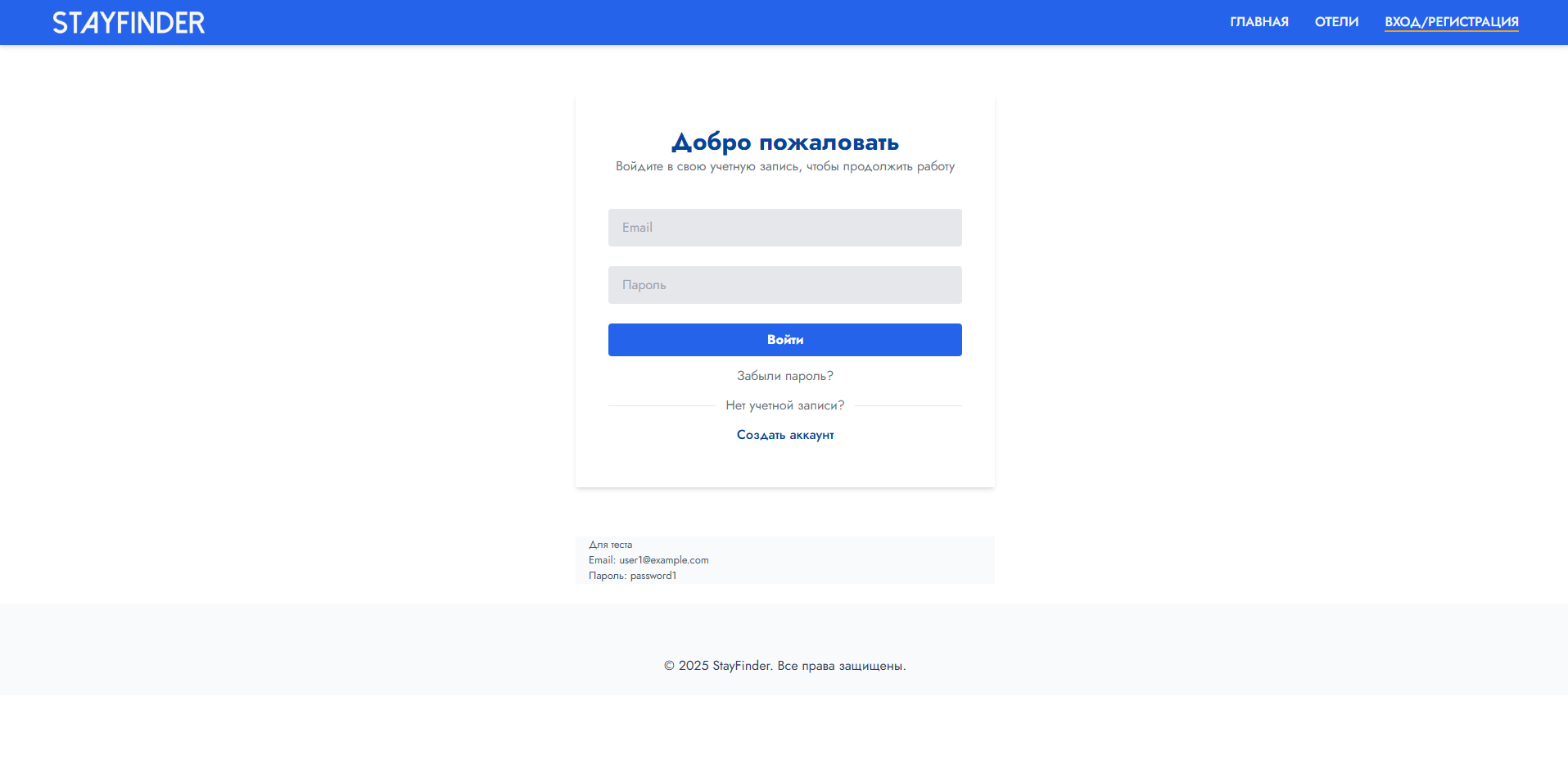


Рисунок 6 – Страница входа в систему

Для регистрации необходимо нажать на кнопку «Создать аккаунт». По нажатию откроется страница регистрации в соответствии с рисунком 7.

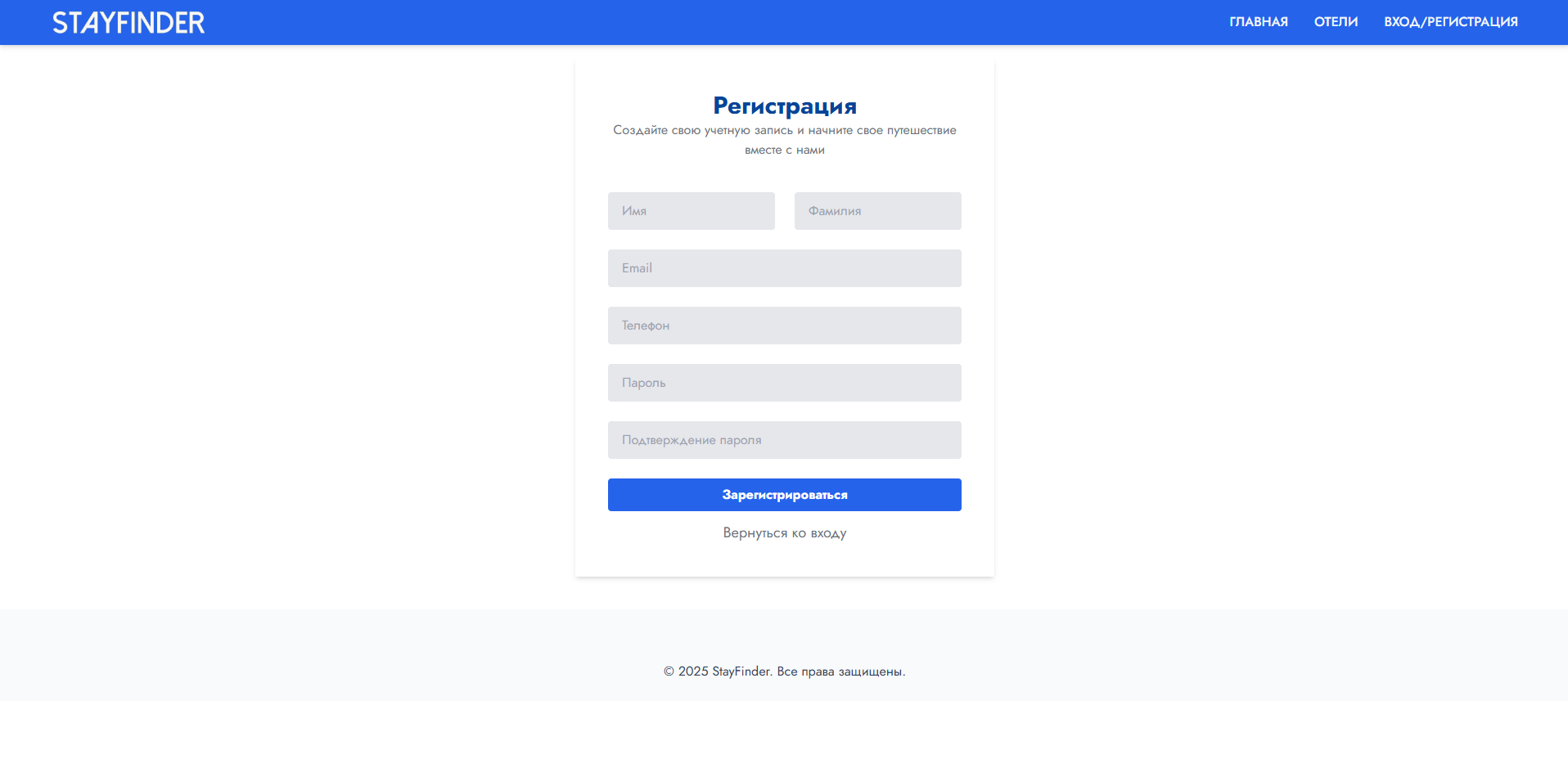


Рисунок 7 – Страница регистрации

Код страницы входа:

import { useState } from 'react';

import { Link } from 'react-router-dom';

import { networkAdapter } from 'services/NetworkAdapter';

import React, { useContext } from 'react';

import { AuthContext } from 'contexts/AuthContext';

import { useNavigate } from 'react-router-dom';

import validations from 'utils/validations';

import Toast from 'components/ux/toast/Toast';

import { LOGIN\_MESSAGES } from 'utils/constants';

const Login = () => {

const navigate = useNavigate();

const context = useContext(AuthContext);

const [loginData, setLoginData] = useState({

email: '',

password: '',

});

const [errorMessage, setErrorMessage] = useState(false);

const handleInputChange = (e) => {

setLoginData({ ...loginData, [e.target.name]: e.target.value });

};

const handleLoginSubmit = async (e) => {

e.preventDefault();

if (validations.validate('email', loginData.email)) {

const response = await networkAdapter.post('api/users/login', loginData);

if (response && response.data.token) {

context.triggerAuthCheck();

navigate('/user-profile');

} else if (response && response.errors.length > 0) {

setErrorMessage(response.errors[0]);

}

} else {

setErrorMessage(LOGIN\_MESSAGES.FAILED);

}

};

const dismissError = () => {

setErrorMessage('');

};

return (

<>

<div className="login\_\_form">

<div className="container mx-auto p-4 flex justify-center min-h-[600px] items-center">

<form

onSubmit={handleLoginSubmit}

className="w-full max-w-lg p-4 md:p-10 shadow-md"

>

<div className="text-center mb-10">

<h2 className="text-3xl font-extrabold text-brand">

Добро пожаловать

</h2>

<p className="text-gray-500">

Войдите в свою учетную запись, чтобы продолжить работу

</p>

</div>

<div className="mb-6">

<input

type="email"

name="email"

placeholder="Email"

value={loginData.email}

onChange={handleInputChange}

autoComplete="username"

className="appearance-none block w-full bg-gray-200 text-gray-700 border rounded py-3 px-4 mb-3 leading-tight focus:outline-none focus:bg-white"

/>

</div>

<div className="mb-6">

<input

type="password"

name="password"

placeholder="Пароль"

value={loginData.password}

onChange={handleInputChange}

autoComplete="current-password"

className="appearance-none block w-full bg-gray-200 text-gray-700 border rounded py-3 px-4 leading-tight focus:outline-none focus:bg-white"

/>

</div>

{errorMessage && (

<Toast

type="error"

message={errorMessage}

dismissError={dismissError}

/>

)}

<div className="items-center">

<div>

<button

type="submit"

className="bg-blue-600 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline w-full"

>

Войти

</button>

</div>

<div className="flex flex-wrap justify-center my-3 w-full">

<Link

to="/forgot-password"

className="inline-block align-baseline text-md text-gray-500 hover:text-blue-800 text-right"

>

Забыли пароль?

</Link>

</div>

<div className="relative">

<div className="absolute left-0 right-0 flex justify-center items-center">

<div className="border-t w-full absolute"></div>

<span className="bg-white px-3 text-gray-500 z-10">

Нет учетной записи?

</span>

</div>

</div>

<div className="flex flex-wrap justify-center my-3 w-full mt-12">

<Link

to="/register"

className="inline-block align-baseline font-medium text-md text-brand hover:text-blue-800 text-right"

>

Создать аккаунт

</Link>

</div>

</div>

</form>

</div>

</div>

<div className="bg-slate-50 flex flex-col mx-auto w-full max-w-lg px-4">

<small className="text-slate-600">Для теста</small>

<small className="text-slate-600">Email: user1@example.com</small>

<small className="text-slate-600">Пароль: password1</small>

</div>

</>

);

};

export default Login;

Код страницы регистрации:

import { useState } from 'react';

import { Link } from 'react-router-dom';

import { networkAdapter } from 'services/NetworkAdapter';

import { useNavigate } from 'react-router-dom';

import Toast from 'components/ux/toast/Toast';

import { REGISTRATION\_MESSAGES } from 'utils/constants';

import { Formik, Form, Field } from 'formik';

import Schemas from 'utils/validation-schemas';

const Register = () => {

const navigate = useNavigate();

const [toastMessage, setToastMessage] = useState('');

const [toastType, setToastType] = useState('success');

const [showToast, setShowToast] = useState(false);

const handleSubmit = async (values) => {

const response = await networkAdapter.put('/api/users/register', values);

console.log('response', response);

if (response && response.errors && response.errors.length < 1) {

setToastMessage(REGISTRATION\_MESSAGES.SUCCESS);

setShowToast(true);

setTimeout(() => navigate('/login'), 2000);

} else {

setToastType('error');

setToastMessage(response.errors[0]);

setShowToast(true);

}

};

return (

<>

<div className="register\_\_form">

<div className="container mx-auto p-4 flex justify-center min-h-[600px] items-center">

<Formik

initialValues={{

firstName: '',

lastName: '',

email: '',

phoneNumber: '',

password: '',

confirmPassword: '',

}}

validationSchema={Schemas.signupSchema}

onSubmit={(values) => handleSubmit(values)}

>

{({ errors, touched }) => (

<Form>

<div className="w-full max-w-lg p-4 shadow-md md:p-10">

<div className="mb-10 text-center">

<h2 className="text-3xl font-extrabold text-brand">

Регистрация

</h2>

<p className="text-gray-500">

Создайте свою учетную запись и начните свое путешествие вместе с нами

</p>

</div>

<div className="flex flex-wrap mb-6 -mx-3">

<div className="w-full px-3 mb-6 md:w-1/2 md:mb-0 relative">

<Field

name="firstName"

placeholder="Имя"

autoComplete="given-name"

className={`${errors.firstName && touched.firstName ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

<div className="w-full px-3 md:w-1/2">

<Field

name="lastName"

placeholder="Фамилия"

autoComplete="family-name"

className={`${errors.lastName && touched.lastName ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

</div>

<div className="mb-6">

<Field

name="email"

placeholder="Email"

autoComplete="email"

className={`${errors.email && touched.email ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

<div className="mb-6">

<Field

name="phoneNumber"

placeholder="Телефон"

autoComplete="tel"

className={`${errors.phoneNumber && touched.phoneNumber ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

<div className="mb-6">

<Field

name="password"

placeholder="Пароль"

autoComplete="new-password"

className={`${errors.password && touched.password ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

<div className="mb-6">

<Field

name="confirmPassword"

placeholder="Подтверждение пароля"

autoComplete="new-password"

className={`${errors.confirmPassword && touched.confirmPassword ? 'border-red-500' : ''} border block w-full px-4 py-3 mb leading-tight text-gray-700 bg-gray-200 rounded appearance-none focus:outline-none focus:bg-white`}

/>

</div>

<div className="flex items-center w-full my-3">

<button

type="submit"

className="w-full px-4 py-2 font-bold text-white rounded bg-blue-600 hover:bg-blue-700 focus:outline-none focus:shadow-outline"

>

Зарегистрироваться

</button>

</div>

<Link

to="/login"

className="inline-block w-full text-lg text-center text-gray-500 align-baseline hover:text-blue-800"

>

Вернуться ко входу

</Link>

{showToast && (

<Toast

type={toastType}

message={toastMessage}

dismissError

/>

)}

</div>

</Form>

)}

</Formik>

</div>

</div>

</>

);

};

export default Register;

Далее была разработана страница профиля пользователя в соответствии с рисунком 8.

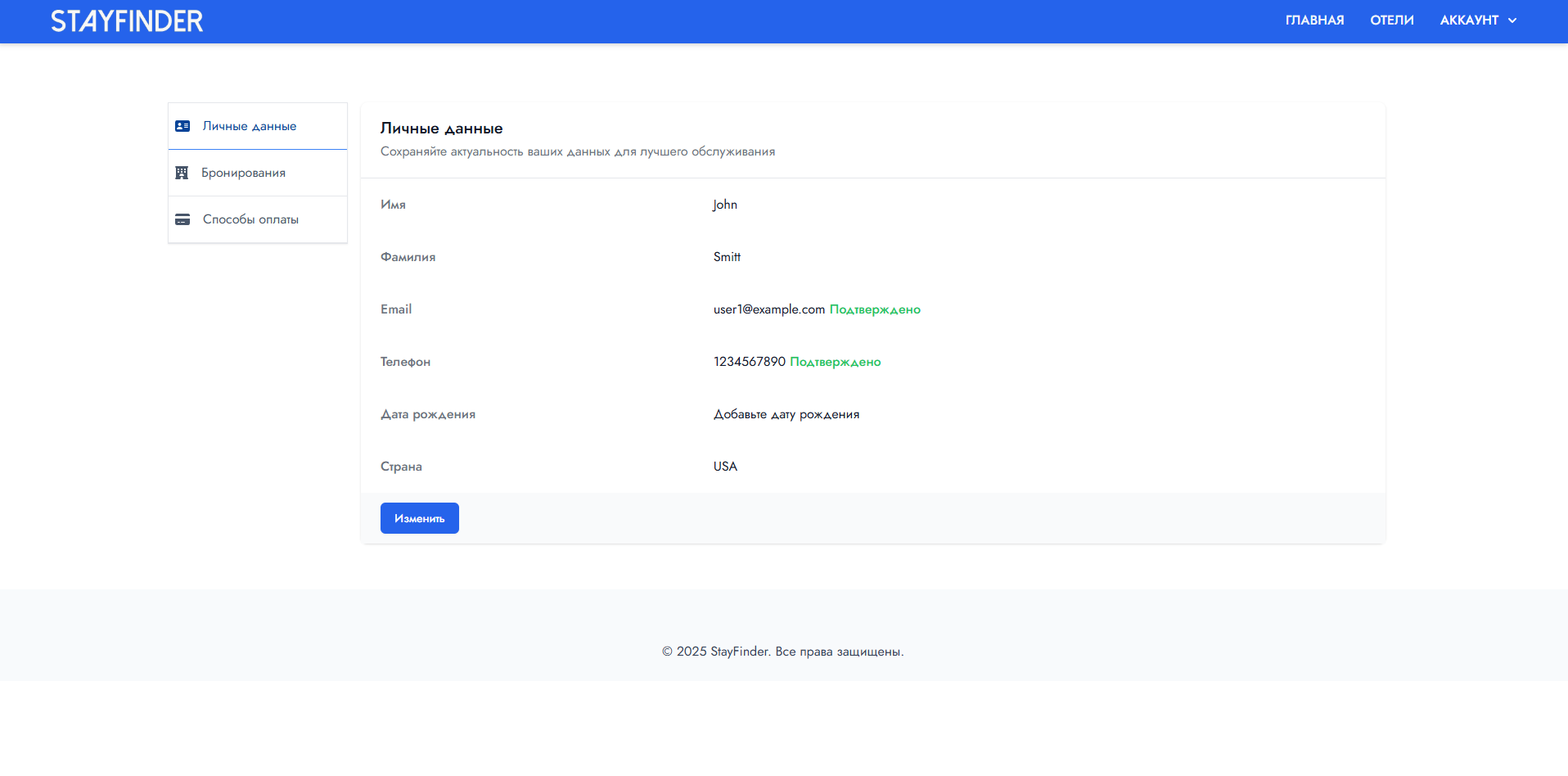


Рисунок 8 – Страница профиля пользователя

Код страницы профиля пользователя:

import React, { useState, useEffect, useRef } from 'react';

import Tabs from 'components/ux/tabs/Tabs';

import TabPanel from 'components/ux/tab-panel/TabPanel';

import {

faAddressCard,

faHotel,

faCreditCard,

} from '@fortawesome/free-solid-svg-icons';

import { AuthContext } from 'contexts/AuthContext';

import { networkAdapter } from 'services/NetworkAdapter';

import { useContext } from 'react';

import PaymentMethodsPanel from './components/PaymentsMethodsPanel';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

import { faBars } from '@fortawesome/free-solid-svg-icons';

import { faXmark } from '@fortawesome/free-solid-svg-icons';

import useOutsideClickHandler from 'hooks/useOutsideClickHandler';

import { useNavigate } from 'react-router-dom';

import BookingPanel from './components/BookingPanel';

import ProfileDetailsPanel from './components/ProfileDetailsPanel';

const UserProfile = () => {

const { userDetails } = useContext(AuthContext);

const navigate = useNavigate();

const wrapperRef = useRef();

const buttonRef = useRef();

const [isTabsVisible, setIsTabsVisible] = useState(false);

const [userBookingsData, setUserBookingsData] = useState({

isLoading: true,

data: [],

errors: [],

});

const [userPaymentMethodsData, setUserPaymentMethodsData] = useState({

isLoading: true,

data: [],

errors: [],

});

useOutsideClickHandler(wrapperRef, (event) => {

if (!buttonRef.current.contains(event.target)) {

setIsTabsVisible(false);

}

});

const onTabsMenuButtonAction = () => {

setIsTabsVisible(!isTabsVisible);

};

useEffect(() => {

if (!userDetails) {

navigate('/login');

}

}, [navigate, userDetails]);

useEffect(() => {

const getInitialData = async () => {

const userBookingsDataResponse = await networkAdapter.get(

'/api/users/bookings'

);

const userPaymentMethodsResponse = await networkAdapter.get(

'api/users/payment-methods'

);

if (userBookingsDataResponse && userBookingsDataResponse.data) {

setUserBookingsData({

isLoading: false,

data: userBookingsDataResponse.data.elements,

errors: userBookingsDataResponse.errors,

});

}

if (userPaymentMethodsResponse && userPaymentMethodsResponse.data) {

setUserPaymentMethodsData({

isLoading: false,

data: userPaymentMethodsResponse.data.elements,

errors: userPaymentMethodsResponse.errors,

});

}

};

getInitialData();

}, []);

return (

<>

<div className="container mx-auto p-4 my-10 min-h-[530px]">

<div className="mx-4">

<button

ref={buttonRef}

onClick={onTabsMenuButtonAction}

className="block md:hidden items-center px-4 py-1.5 border border-gray-300 font-medium rounded text-gray-700 bg-white hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500"

>

<FontAwesomeIcon

icon={isTabsVisible ? faXmark : faBars}

size="lg"

/>

</button>

</div>

<Tabs isTabsVisible={isTabsVisible} wrapperRef={wrapperRef}>

<TabPanel label="Личные данные" icon={faAddressCard}>

<ProfileDetailsPanel userDetails={userDetails} />

</TabPanel>

<TabPanel label="Бронирования" icon={faHotel}>

<BookingPanel bookings={userBookingsData.data} />

</TabPanel>

<TabPanel label="Способы оплаты" icon={faCreditCard}>

<PaymentMethodsPanel

userPaymentMethodsData={userPaymentMethodsData}

setUserPaymentMethodsData={setUserPaymentMethodsData}

/>

</TabPanel>

</Tabs>

</div>

</>

);

};

export default UserProfile;

В итоге прототип приложения был успешно разработан. Было проведено тестирование функционала, в ходе которого ошибок не было обнаружено.

6 ВЫПОЛНЕНИЕ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

В рамках индивидуального задания предстояло разработать модуль для терминала сбора данных, предназначенного для фиксации нанесённых маркировочных кодов.

Компания «Макси» столкнулась с задачей маркировки собственной продукции. Уникальные коды маркировки формируются через платформу «Честный знак», после чего они распечатываются и наносятся на упаковку товаров. По завершении этого процесса необходимо зафиксировать каждый код и зарегистрировать продукцию в системе «Честный знак» для её последующей легализации и реализации.

Так как терминалы сбора данных работают под управлением Android, для них требуется специализированное программное обеспечение. Разработка осуществляется в Android Studio с использованием языков Java и Kotlin. В результате создано приложение с пользовательским интерфейсом, включающим список ранее отсканированных кодов, кнопку для добавления новых, а также элемент управления отправкой данных в систему. В отображаемом списке указаны наименование товара, его уникальный идентификатор в базе и количество считанных экземпляров, как представлено на рисунке 9.



Рисунок 9 – Макет экрана маркировок

При нажатии на кнопку добавления кода маркировки открывается экран сканирования кода в соответствии с рисунком 10, если в базе данных найден такой код, то он добавляется, иначе появляется ошибка «Неизвестный код маркировки».

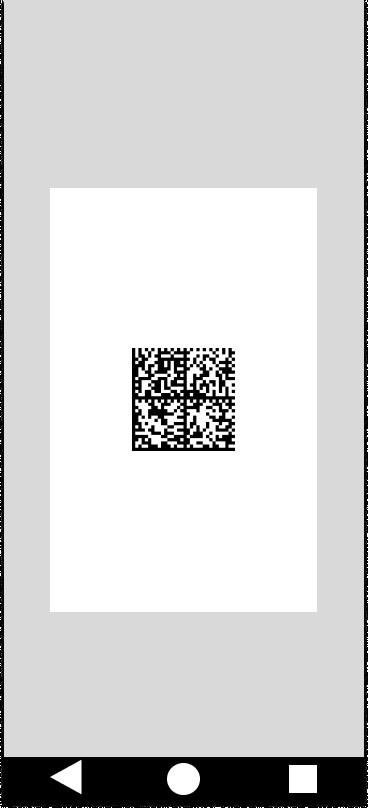


Рисунок 10 – Экран сканирования кода маркировки

7 СБОР И ОБОБЩЕНИЕ МАТЕРИАЛА ДЛЯ ДИПЛОМНОГО ПРОЕКТИРОВАНИЯ

Целью дипломного проекта является разработка веб-сервиса для бронирования отелей, предоставляющего пользователям возможность поиска, сравнения и резервирования жилья в режиме онлайн. Сервис создается с использованием современных технологий: клиентская часть реализуется на React, серверная — на Express.js, а для хранения данных применяется реляционная СУБД MySQL. В процессе разработки важно обеспечить безопасные условия труда программиста, минимизируя воздействие вредных факторов, таких как электростатические поля и зрительное утомление.

В настоящем разделе рассмотрены ключевые аспекты охраны труда, направленные на сохранение здоровья разработчика и повышение эффективности рабочего процесса.

Современная IT-индустрия предполагает длительную работу за компьютером, что сопряжено с рисками для физического и психического здоровья. Основными вредными факторами являются:

* Воздействие электростатических полей от электронной техники.
* Зрительное утомление из-за постоянной фокусировки на экране.

Для нейтрализации этих рисков разработан комплекс организационных и технических мер, соответствующих требованиям ГОСТ и СанПиН.

Электростатические поля, генерируемые компьютерной техникой, могут вызывать головные боли, повышенную утомляемость и раздражение слизистых оболочек. Для минимизации их воздействия применяются следующие меры:

Использование антистатических материалов. Рабочий стол программиста покрывается токопроводящими материалами, например, специальными ковриками с углеродным напылением, которые отводят статическое электричество. Напольные покрытия в помещении должны быть выполнены из материалов с низким удельным сопротивлением (линолеум с антистатическими свойствами, токорассеивающие ковролины) [2].

Ионизация воздуха. Установка ионизаторов в рабочей зоне снижает концентрацию заряженных частиц. Рекомендуется использовать устройства с регулируемой интенсивностью ионизации, размещая их на расстоянии 1–2 метров от рабочего места.

Контроль влажности воздуха. Поддержание относительной влажности в диапазоне 40–60% предотвращает накопление статического заряда. Для этого применяются увлажнители с функцией автоматического поддержания уровня влажности, а также системы климат-контроля.

Заземление оборудования. Все электронные устройства (системные блоки, мониторы, периферия) подключаются к заземленной электрической сети через розетки с контуром заземления. Дополнительно используется экранирование кабелей и корпусов техники.

Регулярная влажная уборка. Поверхности столов, полок и техники протираются антистатическими салфетками, что снижает накопление пыли — основного проводника статического электричества.

По данным исследований, более 70% программистов испытывают зрительное утомление к концу рабочего дня. Для профилактики нарушений зрения реализуются следующие меры:

Оптимизация параметров монитора.

Экран располагается на расстоянии 50–70 см от глаз, верхний край — на уровне линии взгляда.

Яркость настраивается в соответствии с освещением: при искусственном свете — 100–150 кд/м², при естественном — 200–300 кд/м².[4]

Контрастность устанавливается в соотношении 10:1 для текста и фона.

Частота обновления экрана — не менее 60 Гц для исключения мерцания [1].

Использование программных средств защиты зрения.

Применение приложений, регулирующих цветовую температуру экрана (например, f.lux или Night Shift). В вечернее время температура снижается до 2700–3000 K для уменьшения воздействия синего спектра.[3]

Активация режима «темной темы» в IDE и браузерах для снижения контрастности.

Организация режима труда и отдыха.

Каждый час непрерывной работы сопровождается 10-минутным перерывом. В это время выполняются упражнения для глаз в соответствии с рисунком 1.

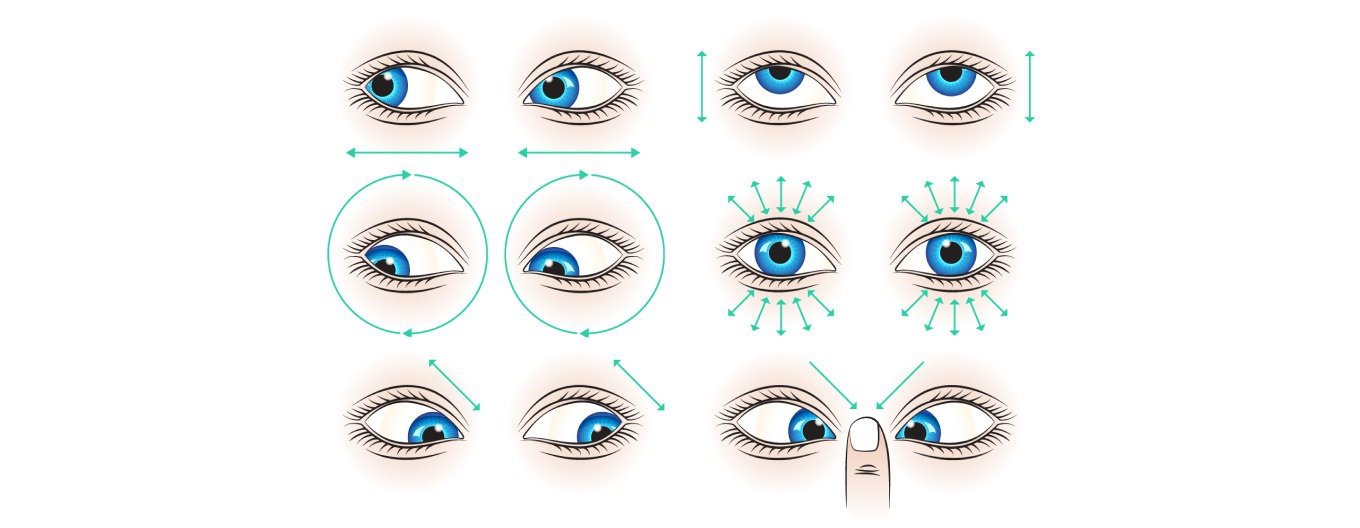


Рисунок 1 – Гимнастика для глаз

Фокусировка на удаленном объекте (5–6 метров) в течение 20 секунд.

Круговые движения глазами по часовой и против часовой стрелки.

Легкий массаж век круговыми движениями.

Для снижения нагрузки на аккомодацию глаз рекомендуется использование мониторов с диагональю не менее 24 дюймов и разрешением Full HD.

Освещение рабочего места.

Естественный свет должен падать сбоку от рабочего места, исключая блики на экране.

Искусственное освещение обеспечивается рассеянными источниками света (светодиодные панели, потолочные светильники) с интенсивностью 500–750 люкс. Настольные лампы с регулируемой яркостью дополняют общее освещение, но не создают контрастных теней [5].

Соблюдение перечисленных мер обеспечивает безопасные условия труда программиста, снижает риск профессиональных заболеваний и повышает продуктивность разработки. Регулярный контроль параметров рабочей среды, соблюдение режима труда и использование сертифицированного оборудования — ключевые элементы системы охраны труда в IT-сфере.

ЗАКЛЮЧЕНИЕ

В ходе прохождения производственной практики были закреплены полученные в процессе обучения знания. Была изучена компания, её бизнес-процессы, правила по охране труда, распорядок дня. Были настроены необходимые для работы программы. Была выполнена работа с тестовой базой данных Oracle. Провелась разработка программного модуля дипломного проекта на тему «Разработка веб-сервиса для бронирования отелей». Также было выполнено индивидуальное задание по разработке модуля для сканирования кодов маркировки с помощью терминалов сбора данных на операционной системе Android.

Поставленная цель производственной практики достигнута в полном объёме.