Автономная некоммерческая профессиональная образовательная организация
«Московский колледж социально-экономических и информационных технологий»

ОТЧЕТ

по производственной практике

ПП 01.01 «Учебная практика»
индекс по ПП и наименование практики

Проектирование и разработка информационных систем (ПМ.05)
индекс по ПП и наименование профессионального модуля

Специальность 09.02.06 «Сетевое и системное администрирование»
код и наименование специальности

Обучающийся Баранов Евгений Павлович
(фамилия, имя, отчество)

Группа
«ИС3/09»

Руководитель по практической подготовке от колледжа

Лоскутов Иван Андреевич
(фамилия, имя, отчество)

«23» Декабря 2025 года

Оглавление

Лабораторная работа №1 «Перенос сайта на React» 3

Лабораторная работа №2 «нововведения» 11

Лабораторная работа №3 «обновление contacts.jsx»..... 21

Лабораторная работа №4 «обновления в App.jsx»..... 26

Лабораторная работа №5 «About.jsx»..... 45

Лабораторная работа №6 «создание системы авторизации без Backend»..... 50

Лабораторная работа №1 «Перенос сайта на React»

Цель работы: освоить перенос готового статического сайта на React с использованием JSX, разбивая разметку на компоненты и настраивая клиентский роутинг.

1.Пункт первый.

Перед началом переноса были выбраны стек технологий: React, Vite, React Router и библиотека компонентов MUI, что позволило совместить существующую вёрстку с современным фронтенд-фреймворком. Был инициализирован новый React-проект на базе Vite: создана структура каталогов src, assets, файлы компонентов и общий файл стилей styles.css. Вёрстка исходного сайта была проанализирована: определены основные страницы («Главная», «О нас», «Контакты»), повторяющиеся элементы (шапка, подвал, секции контента), а также интерактивные элементы (модальные окна, форма).

Древо проекта выглядит вот таким образом:

Src/

About.jsx

App.jsx

Contacts.jsx

Main.jsx

Styles.css

ThemeProviderMUI.jsx

Assets/

Atlus-1.jpg

Atlus-2.jpg

Team-logo.jpg

Student.jpg

1.Пункт второй.

Каждая HTML-страница была вынесена в отдельный React-компонент: App.jsx (главная), About.jsx (о компании), Contacts.jsx (контакты). В процессе переноса выполнена адаптация разметки под JSX: атрибуты class заменены на className, for на htmlFor, а все встроенные <script> и обработчики кликов были удалены и переписаны на логику React. Общие элементы оформления (шапка, навигация, подвал) сохранены в одинаковом виде на всех страницах, что упростило последующую интеграцию роутинга и переключения темы.

На рисунке 1 показан код импорта.

```
import { useState } from 'react';
import { Link } from 'react-router-dom';
import atlus1 from './assets/atlus-1.jpg';
import atlus2 from './assets/atlus-2.jpg';
```

Рисунок 1 - Импорты useState, Link

imgModalOpen хранит, открыта ли сейчас модалка (true/false), imgSrc — какой именно src картинки показывать. openImageModal(src) ставит адрес картинки в imgSrc, открывает модалку (setImgModalOpen(true)) и блокирует прокрутку страницы. closeImageModal() закрывает модалку, очищает imgSrc и возвращает прокрутку.

На рисунке 2 изображен код для открытия и закрытия модального окна картинки:

```
function App({ toggleTheme, mode }) {
  const [imgModalOpen, setImgModalOpen] = useState(false);
  const [imgSrc, setImgSrc] = useState('');

  const openImageModal = (src) => {
    setImgSrc(src);
    setImgModalOpen(true);
    document.body.style.overflow = 'hidden';
  };

  const closeImageModal = () => {
    setImgModalOpen(false);
    setImgSrc('');
    document.body.style.overflow = 'auto';
  };
```

Рисунок 2 - Функция для открытия/закрытия модального окна

3.Пункт третий. Настройка клиентского роутинга с react-router-dom.

Для организации навигации без перезагрузки страницы был установлен и подключён пакет react-router-dom.

В файле main.jsx настроен BrowserRouter, внутри которого описаны маршруты /, /about, /contacts, каждому маршруту сопоставлен соответствующий компонент. Обычные ссылки <a href="..."> в шапке были заменены на компоненты <Link to="...">, благодаря чему переходы между страницами происходят мгновенно и без перерисовки всего документа.

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

На рисунке 3 показана конфигурация маршрутов.

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <AppThemeProvider>
      {({ toggleTheme, mode }) => (
        <BrowserRouter>
          <Routes>
            <Route path="/" element={<App toggleTheme={toggleTheme} mode={mode} />} />
            <Route path="/about" element={<About toggleTheme={toggleTheme} mode={mode} />} />
            <Route path="/contacts" element={<Contacts toggleTheme={toggleTheme} mode={mode} />} />
          </Routes>
        </BrowserRouter>
      )}
    </AppThemeProvider>
  </React.StrictMode>,
);
```

Рисунок 3 - Конфигурация BrowserRouter для навигации

4. Пункт четвертый. Реализация модальных окон на React.

Модальное окно просмотра изображений в разделе «Портфолио» было переписано с нативного JavaScript на React: теперь его состояние (открыто/закрыто и текущий src изображения) хранится в useState, а само окно рендерится условно через JSX. Аналогичным образом реализована модальная форма обратной связи на странице контактов: состояние открытия, закрытие по фону и по крестику, блокировка прокрутки body при открытом окне. Старые скрипты с addEventListener и манипуляциями DOM были полностью удалены, вся логика перенесена внутрь React-компонентов, что сделало код чище и проще для сопровождения.

На рисунке 4 показано модальное окно картинки.

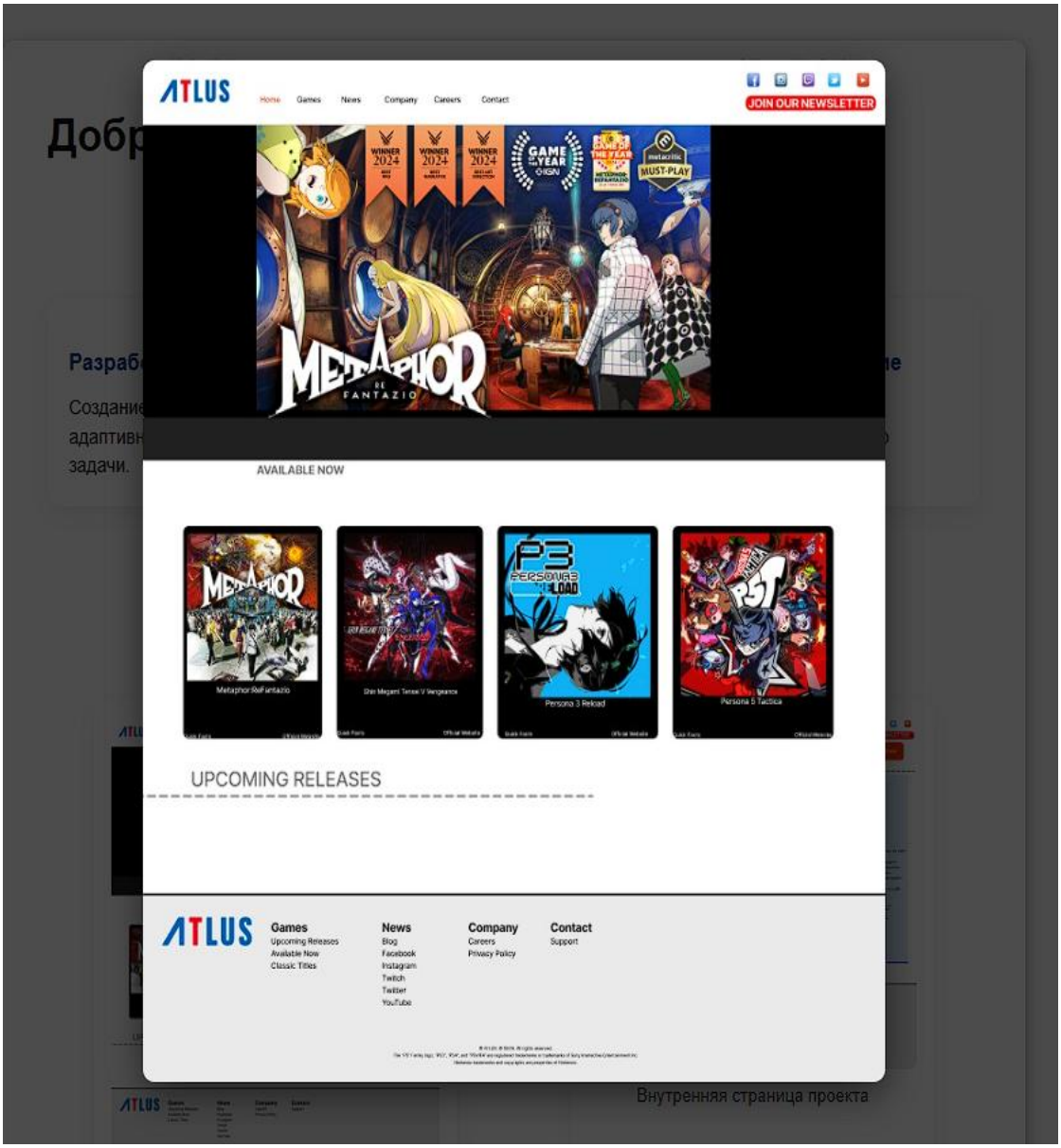


Рисунок 4 - модальное окно картинки

На рисунке 5 показано модальное окно для формы обратной связи.

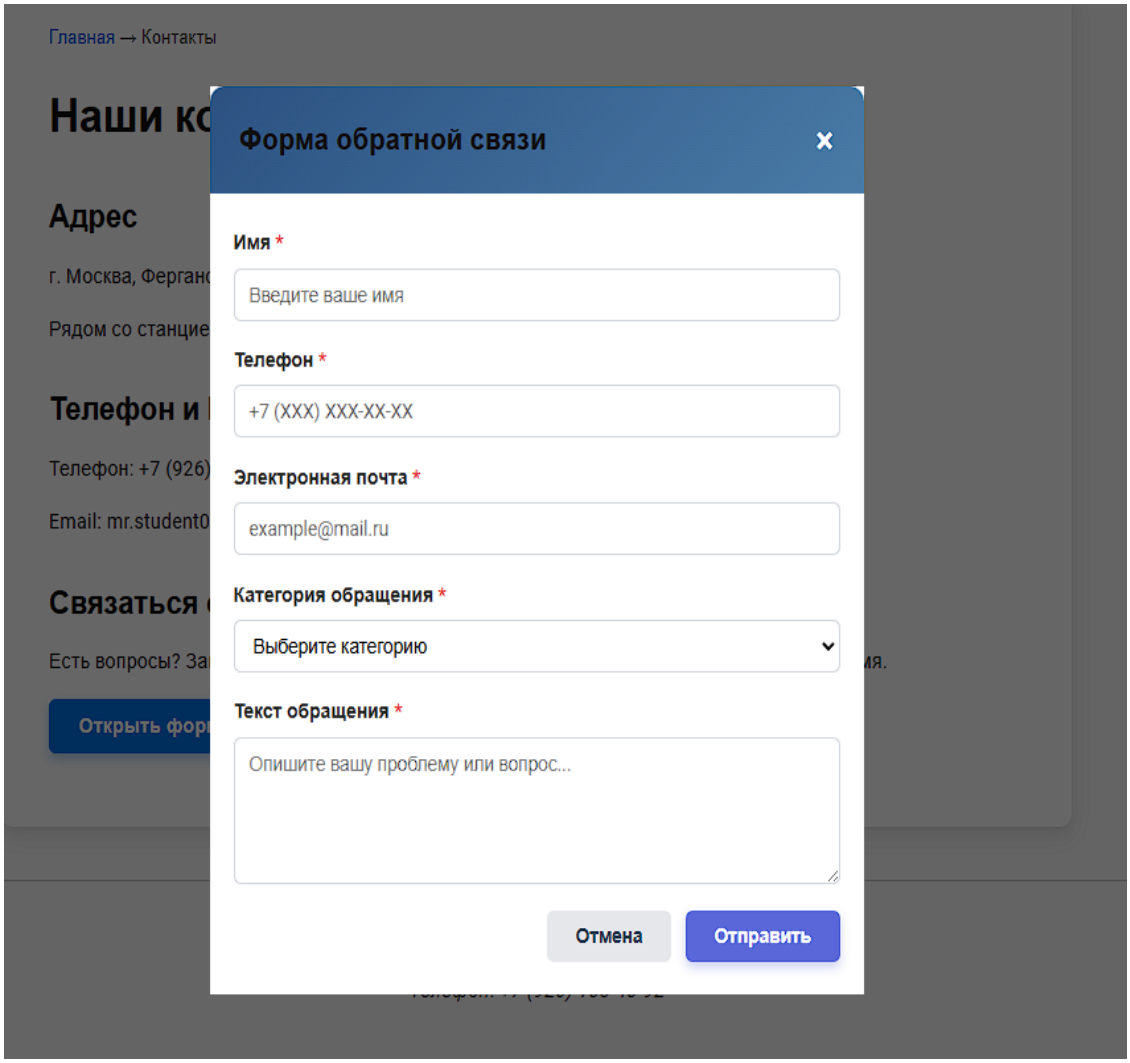


Рисунок 5 - модальное окно формы обратной связи

На картинке 6 показан код для формы обратной связи.

```
function Contacts({ toggleTheme, mode }) {
  const [isModalOpen, setModalOpen] = useState(false);

  const [formData, setFormData] = useState({
    name: '',
    phone: '',
    email: '',
    category: '',
    message: '',
  });

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

  const openModal = () => {
    setModalOpen(true);
    document.body.style.overflow = 'hidden';
  };

  const closeModal = () => {
    setModalOpen(false);
    document.body.style.overflow = 'auto';
  };

  const handleChange = (e) => {
    const { name, value } = e.target;

    setFormData((prev) => ({
      ...prev,
      [name]: value,
    }));

    if (name === 'email') {
      if (value && !value.includes('@')) {
        setErrors((prev) => ({ ...prev, email: 'Некорректный email' }));
      } else {
        setErrors((prev) => ({ ...prev, email: '' }));
      }
    }
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    const newErrors = {};
```

Рисунок 6 - код модального окна для формы

На рисунке 7 показано продолжение кода для формы обратной связи

```
  if (!formData.name.trim()) {
    newErrors.name = 'Введите имя';
  }

  if (!formData.phone.trim()) {
    newErrors.phone = 'Введите телефон';
  }

  if (!formData.email.trim()) {
    newErrors.email = 'Введите email';
  } else if (!formData.email.includes('@')) {
    newErrors.email = 'Некорректный email';
  }

  if (!formData.category) {
    newErrors.category = 'Выберите категорию обращения';
  }

  if (!formData.message.trim()) {
    newErrors.message = 'Опишите вашу проблему или вопрос';
  }

  setErrors(newErrors);

  if (Object.keys(newErrors).length === 0) {
    alert('Форма отправлена! Спасибо за обращение.');
    closeModal();
    setFormData({
      name: '',
      phone: '',
      email: '',
      category: '',
      message: '',
    });
    setErrors({});
  }
};

return (
```

Рисунок 7 - продолжение кода для модального окна для формы

Вывод: Мы успешно перенесли наш сайт на React.

Лабораторная работа №2 «нововведения»

1. Пункт первый. Валидация форм и обработка данных.

Для формы обратной связи и регистрационной формы создано единое состояние formData, которое хранит значения всех полей (имя, телефон, email, категория, сообщение). Дополнительно введено состояние errors для хранения сообщений об ошибках. Реализована «живая» валидация email при вводе (проверка наличия @) и итоговая комплексная проверка всех полей при отправке формы. Ошибки отображаются непосредственно под соответствующими полями, что повышает удобство пользователя и демонстрирует базовые принципы работы с формами в React.

На картинке 8 показано как именно это выглядит в коде:

```
const newErrors = {};

if (!formData.name.trim()) {
  newErrors.name = 'Введите имя';
}

if (!formData.phone.trim()) {
  newErrors.phone = 'Введите телефон';
}

if (!formData.email.trim()) {
  newErrors.email = 'Введите email';
} else if (!formData.email.includes('@')) {
  newErrors.email = 'Некорректный email';
}

if (!formData.category) {
  newErrors.category = 'Выберите категорию обращения';
}

if (!formData.message.trim()) {
  newErrors.message = 'Опишите вашу проблему или вопрос';
}

setErrors(newErrors);
```

Рисунок 8 - Валидация формы

На картинке 9 показано что будет если оставить все поля пустыми, а электронную почту ввести некорректно

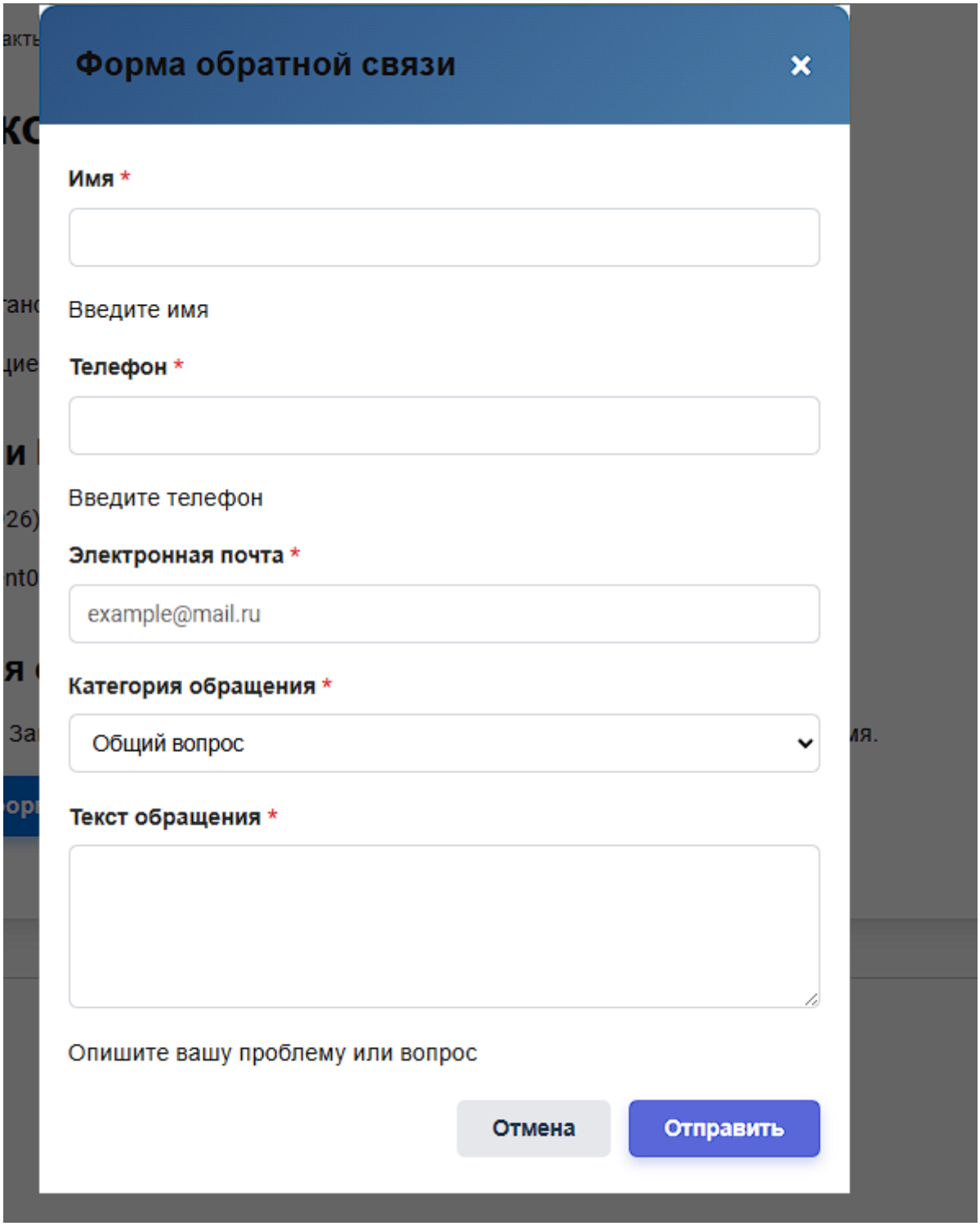


Рисунок 9 - Тестирование формы

2. Пункт второй. Подключение MUI и реализация светлой/тёмной темы. В проект была добавлена библиотека MUI, создан компонент AppThemeProvider, внутри которого формируется тема через createTheme с режимами light и dark, а также используется ThemeProvider и CssBaseline для применения темы ко всему приложению. Переключение темы реализовано с помощью состояния mode, которое сохраняется в localStorage, а также прокидывается в основные компоненты (App, About, Contacts) как пропсы mode и toggleTheme. Внешний вид страниц был адаптирован под обе темы: фон центрального блока, карточек и модальных окон меняется в зависимости от режима, цвета текста и полей форм подстраиваются через CSS-классы theme-light и theme-dark на элементе body.

Для реализации светлой/тёмной темы нужно создать файл ThemeProviderMUI.jsx.

В него нужно вписать код который показан на рисунке 10.

```jsx
import { useMemo, useState, useEffect } from 'react';
import { ThemeProvider, createTheme } from '@mui/material/styles';
import CssBaseline from '@mui/material/CssBaseline';

export function AppThemeProvider({ children }) {
  const [mode, setMode] = useState(
    () => localStorage.getItem('themeMode') || 'light',
  );

  useEffect(() => {
    localStorage.setItem('themeMode', mode);
  }, [mode]);
  useEffect(() => {
  document.body.classList.toggle('theme-dark', mode === 'dark');
  document.body.classList.toggle('theme-light', mode === 'light');
}, [mode]);

  const theme = useMemo(
    () =>
      createTheme({
        palette: {
          mode,
          primary: {
            main: '#0073e6',
          },
          background: {
            default: mode === 'light' ? '#f9f9f9' : '#121212',
            paper: mode === 'light' ? '#ffffff' : '#1f1f1f',
          },
        },
      }),
    [mode],
  );

  const toggleTheme = () => {
    setMode((prev) => (prev === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      {}
      {typeof children === 'function' ? children({ mode, toggleTheme }) : children}
    </ThemeProvider>
  );
}
```

Рисунок 10 - код для ThemeProviderMUI.jsx

На рисунке 11 показана главная страница на светлой теме.

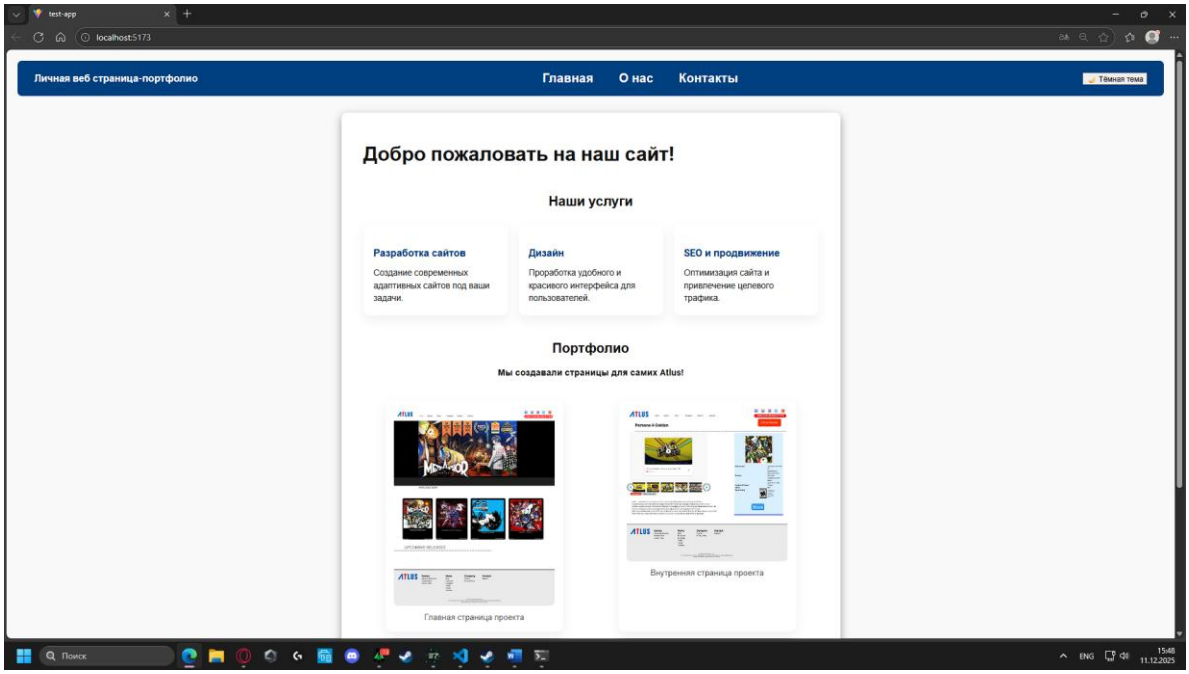


Рисунок 11 - светлая тема

На рисунке 12 показана главная страница на тёмной теме.

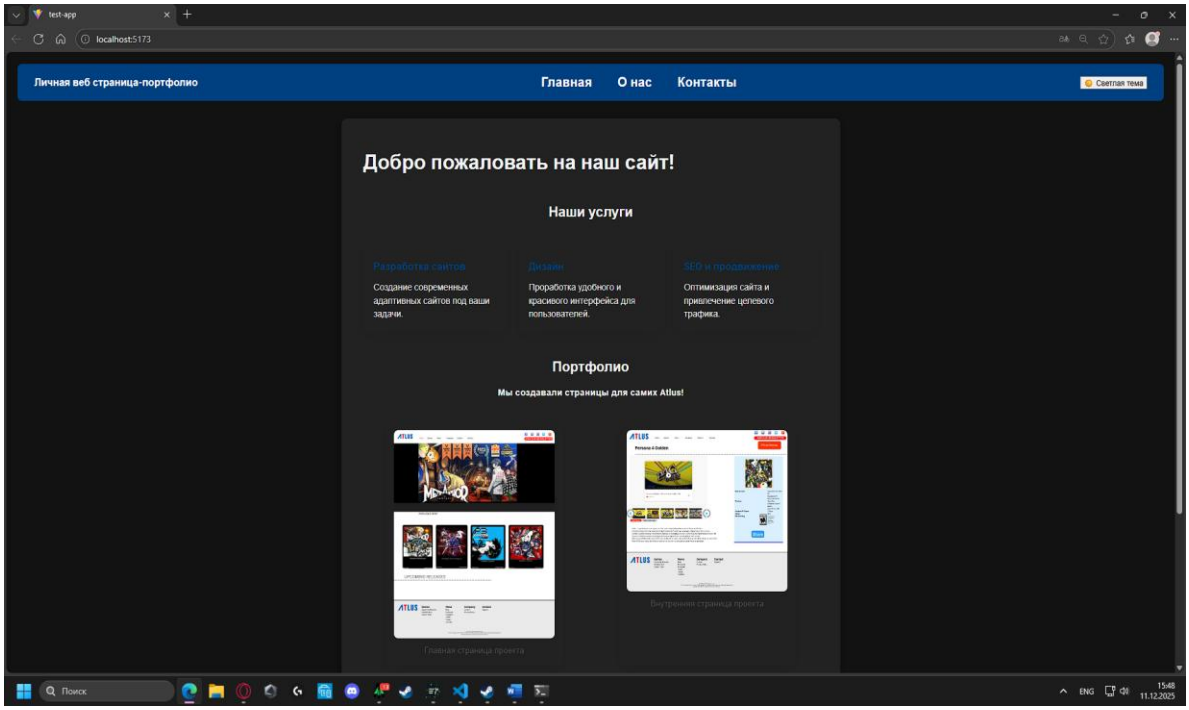


Рисунок 12 - темная тема

На рисунке 13 показана темная тема на странице о нас.

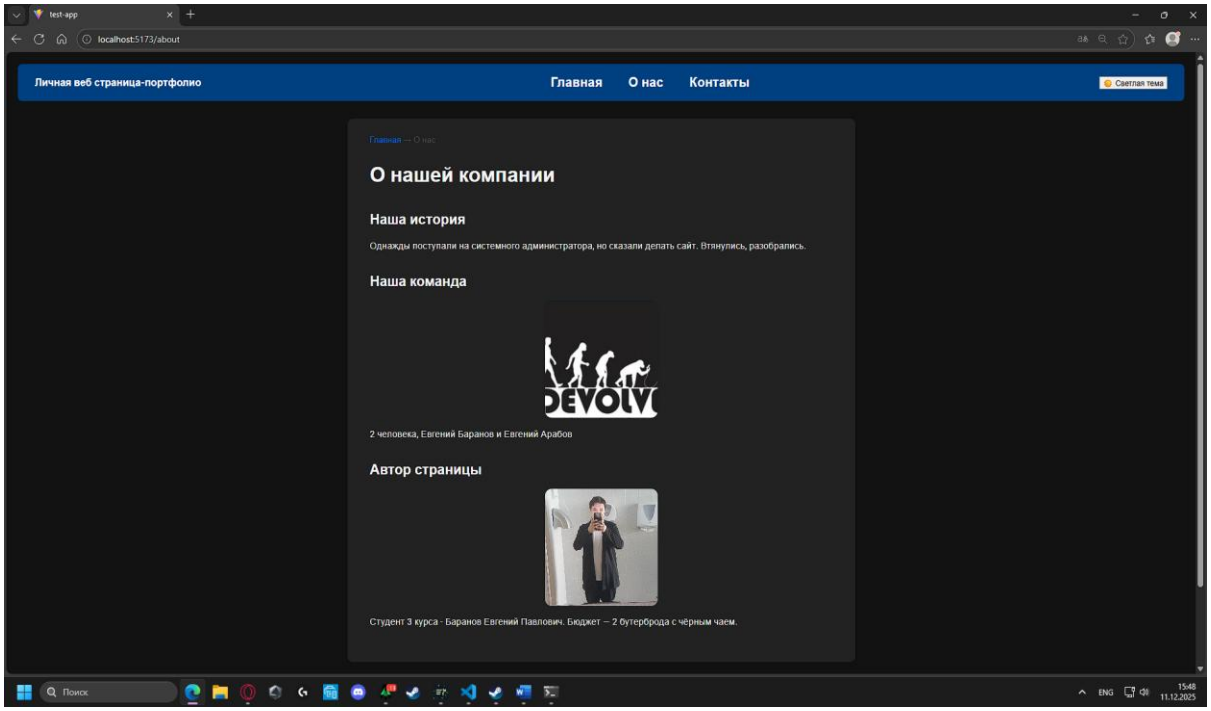


Рисунок 13 - темная тема о нас

На рисунке 14 показана темная тема на странице контакты.

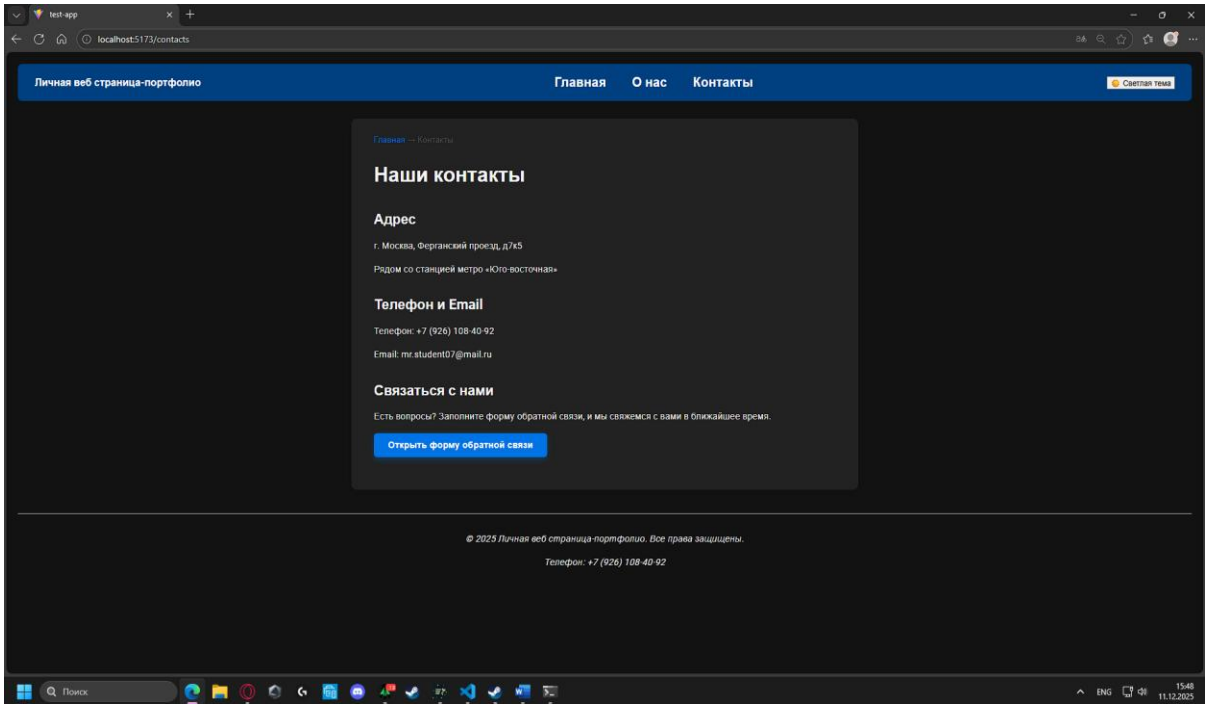


Рисунок 14 - темная тема контакты

На рисунке 15 показана темная тема формы обратной связи.

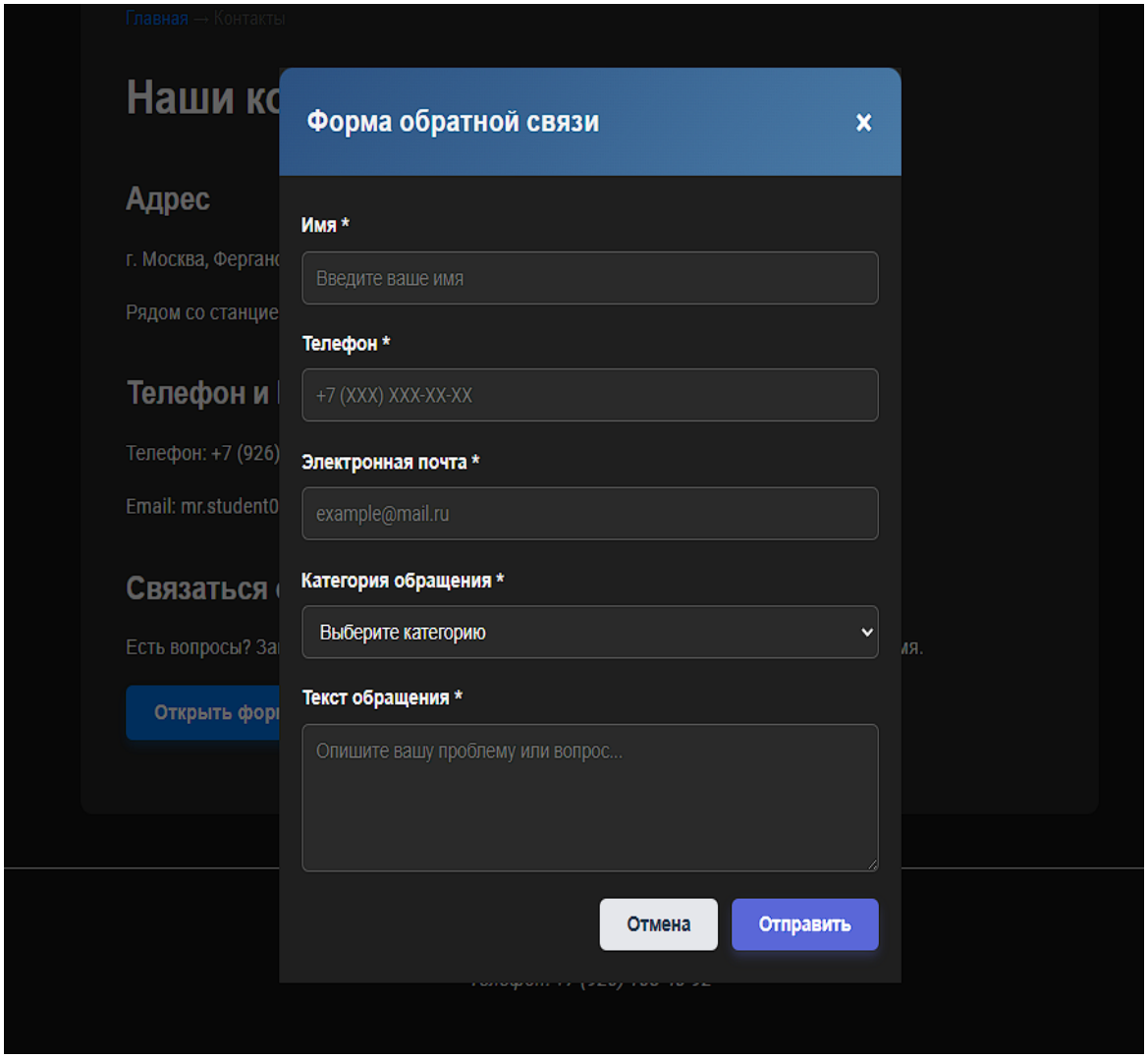


Рисунок 15 - темная тема формы обратной связи

На рисунке 16 показан import main.jsx

main.jsx

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import App from './App.jsx';
import About from './About.jsx';
import Contacts from './contacts.jsx';
import { AppThemeProvider } from './ThemeProviderMUI.jsx';
import './styles.css';
```

Рисунок 16 - main.jsx

На картинке 17 показана проверка LightHouse на доступность.

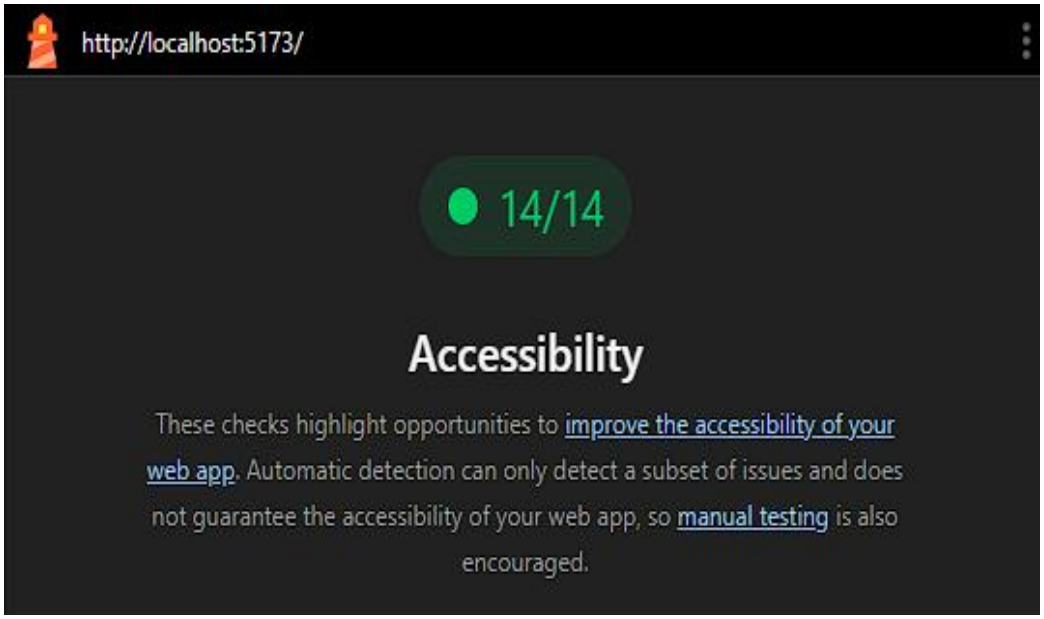


Рисунок 17 – Lighthouse

На рисунке 18 показана адаптивность под мобильные устройства

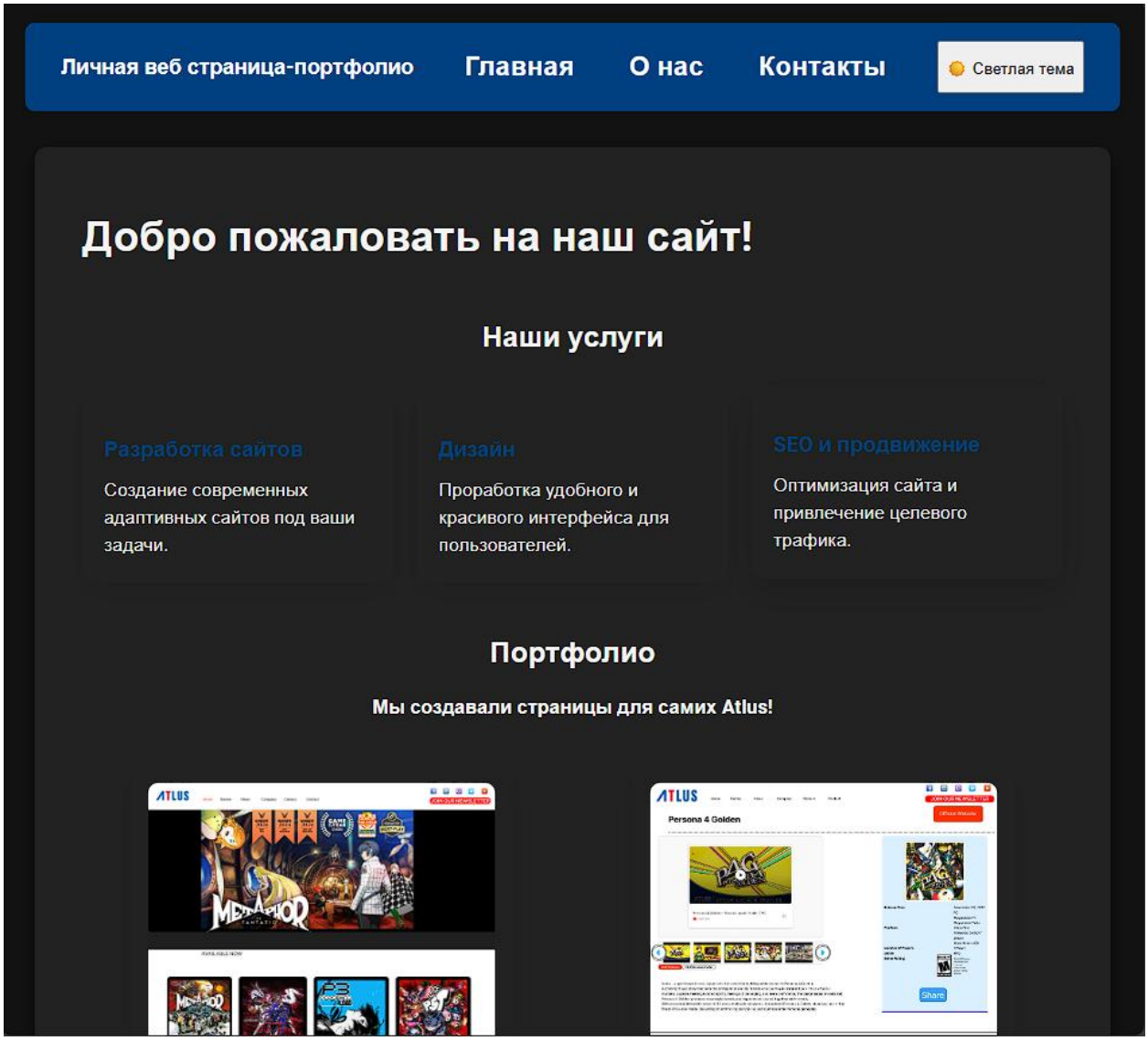


Рисунок 18 - Адаптивность

Вывод: В ходе работы была реализована клиентская валидация форм и подключена библиотека MUI, что существенно повысило качество и удобство веб-приложения. Настройка состояний formData и errors позволила проверять поля в реальном времени и при отправке формы, отображая понятные сообщения об ошибках прямо под соответствующими полями.

Лабораторная работа №3 «обновление contacts.jsx»

1. Пункт первый.

Toast-уведомление после отправки формы - это маленькое всплывающее сообщение в углу экрана, которое на несколько секунд показывает результат действия пользователя (например, «Сообщение отправлено успешно»), не перезагружая страницу и не блокируя интерфейс. При отправке твоей формы «Контакты» будет появляться компактное уведомление в правом нижнем углу: зелёно
е при успехе, красное при ошибке.Через пару секунд оно автоматически скрывается, не требуя от пользователя дополнительных действий, при этом остаётся доступным для скринридеров через aria-live. После успешной отправки формы в модальном окне показывается всплывающее уведомление (Snackbar) в правом нижнем углу. Внутри Snackbar используется компонент Alert с зелёным («success») стилем и текстом «Сообщение отправлено! Спасибо за обращение.».

```
import Snackbar from '@mui/material/Snackbar';
import MuiAlert from '@mui/material/Alert';
const Alert = (props) => <MuiAlert elevation={6} variant="filled" {...props} />;
```

На рисунке 19 показана разметка Snackbar.

```jsx
<Snackbar
  open={snackbarOpen}
  autoHideDuration={4000}
  onClose={handleSnackbarClose}
  anchorOrigin={{ vertical: 'bottom', horizontal: 'right' }}
>
  <Alert onClose={handleSnackbarClose} severity="success" sx={{ width: '100%' }}>
    Сообщение отправлено! Спасибо за обращение.
  </Alert>
</Snackbar>
```

Рисунок 19 - разметка Snackbar JSX

На рисунке 20 - логика показа toast-уведомления после успешной валидации формы обратной связи.

```jsx
if (Object.keys(newErrors).length === 0) {
  setSnackbarOpen(true);
  closeModal();
  setFormData({
    name: '',
    phone: '',
    email: '',
    category: '',
    message: '',
  });
  setErrors({});
}
};
```

Рисунок 20 - логика показа toast-уведомления

На рисунке 21 показан пример отображения toast-уведомления

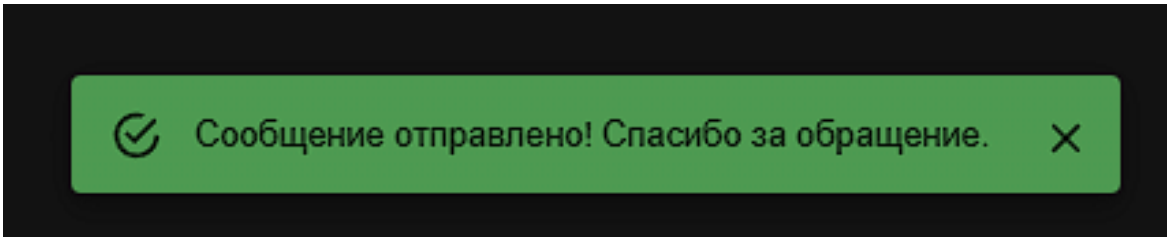


Рисунок 21 - отображение toast-уведомления

2. Пункт второй.

Карта с местоположением на странице контактов - это встроенный интерактивный блок Google Maps, который помогает пользователю визуально понять, где находится офис и как к нему добраться. Карта оформлена в обёртку map-wrapper, чтобы её можно было сделать адаптивной по ширине и задать высоту (350 пикселей и ширина 100%). В Google Maps был найден нужный адрес, установлен маркер, затем через меню «Поделиться» - «Встроить карту» был скопирован HTML-код с тегом iframe.

На рисунке 22 показан код для карты.

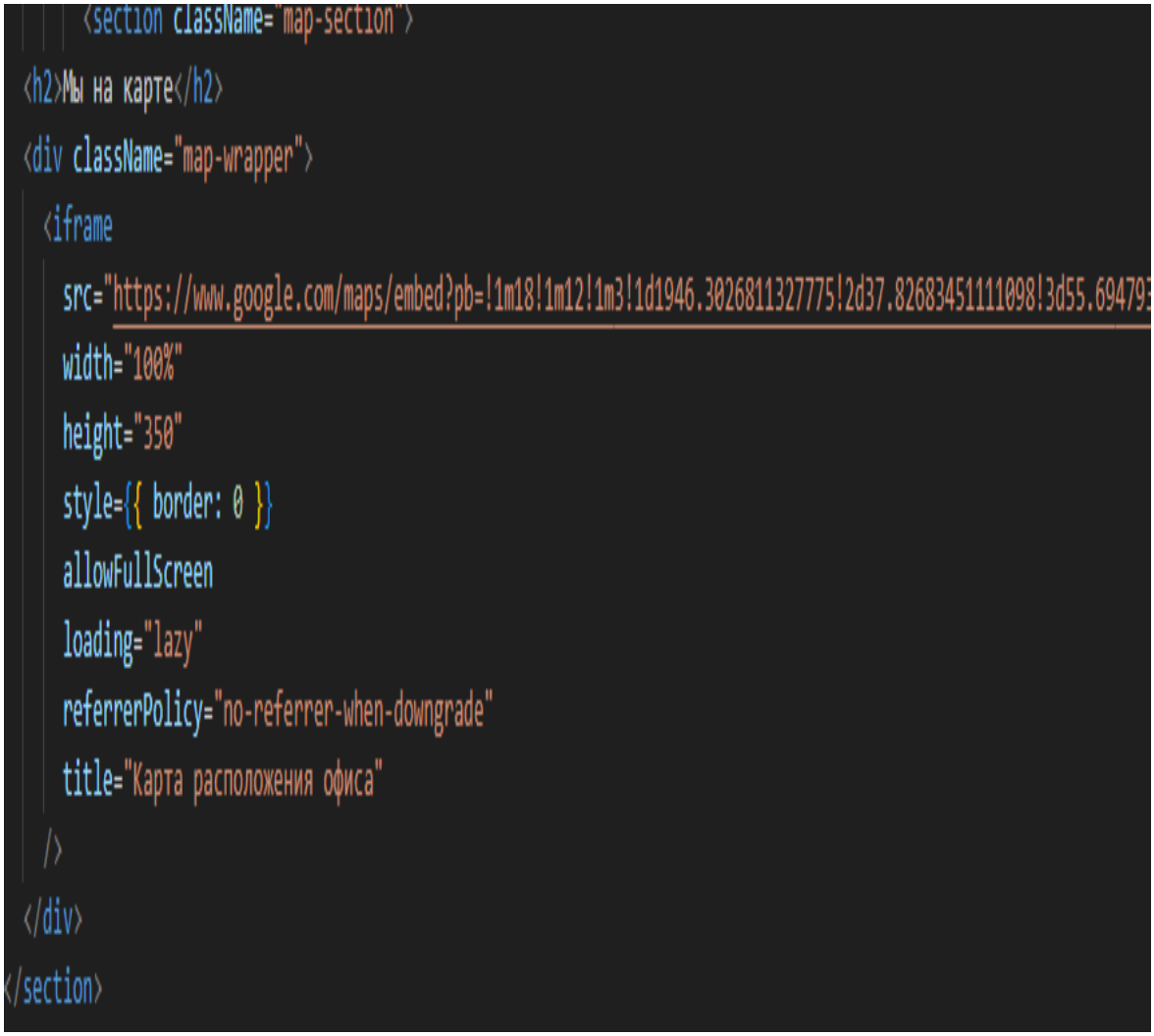


Рисунок 22 - код для карты

На рисунке 23 показан css для блока карты

```css
.map-section {
  margin-top: 25px;
}

.map-wrapper {
  margin-top: 10px;
  border-radius: 10px;
  overflow: hidden;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.15);
}

.map-wrapper iframe {
  display: block;
}
```

Рисунок 23 - css для карты

Вывод: В результате проведённых доработок страница контактов превратилась из простого списка данных в полноценный интерактивный и современный интерфейс. Пользователь теперь может не только увидеть текстовый адрес и контактную информацию, но и визуально оценить местоположение организации с помощью встроенной карты Google Maps, а также открыть модальное окно с формой обратной связи, пройти валидацию полей и получить понятное toast-уведомление (Snackbar) об успешной отправке сообщения.

Лабораторная работа №4 «обновления в App.jsx»

1. Пункт первый.

Блок часто задаваемых вопросов позволяет пользователю быстро получить ответы на типичные вопросы, не связываясь с поддержкой. FAQ оформлен в виде аккордеона: у каждого вопроса есть заголовок-кнопка, при нажатии которой разворачивается ответ, а повторное нажатие его скрывает. Такой паттерн экономит место на странице и делает чтение более удобным. В начале компонента был создан массив объектов с вопросами и ответами, например:

id — уникальный идентификатор вопроса.

question — текст вопроса.

answer — текст ответа.

Далее через useState хранится идентификатор текущего «открытого» вопроса (или null, если все свернуты). Обработчик клика по вопросу сравнивает id с текущим и либо раскрывает новый пункт, либо закрывает уже открытый, если пользователь нажал на него повторно. Это позволяет управлять аккордеоном как «контролируемым» компонентом: состояние хранится в React, а разметка просто отражает его. Чтобы сделать раздел менее шаблонным и показать работу с массивами, был добавлен шаг с перемешиванием FAQ. Внутри компонента создавалась копия исходного массива, после чего к нему применялся алгоритм перетасовки: в цикле элементы меняются местами с случайно выбранными индексами. В результате при каждом обновлении страницы вопросы отображаются в новом порядке, но логика аккордеона при этом не ломается, так как для key и для сравнения используется стабильный id. Такой подход

демонстрирует умение работать с данными в React и не нарушать принцип уникальных ключей в списках. При верстке FAQ были учтены базовые требования доступности: заголовок вопроса оформлен как кнопка, которая получает фокус с клавиатуры и по нажатию клавиши Enter/Space также раскрывает ответ. Состояние «открыто/закрыто» можно дополнительно описать через aria-expanded и связывание заголовка с панелью ответа по aria-controls и id, что соответствует рекомендациям по доступным аккордеонам.

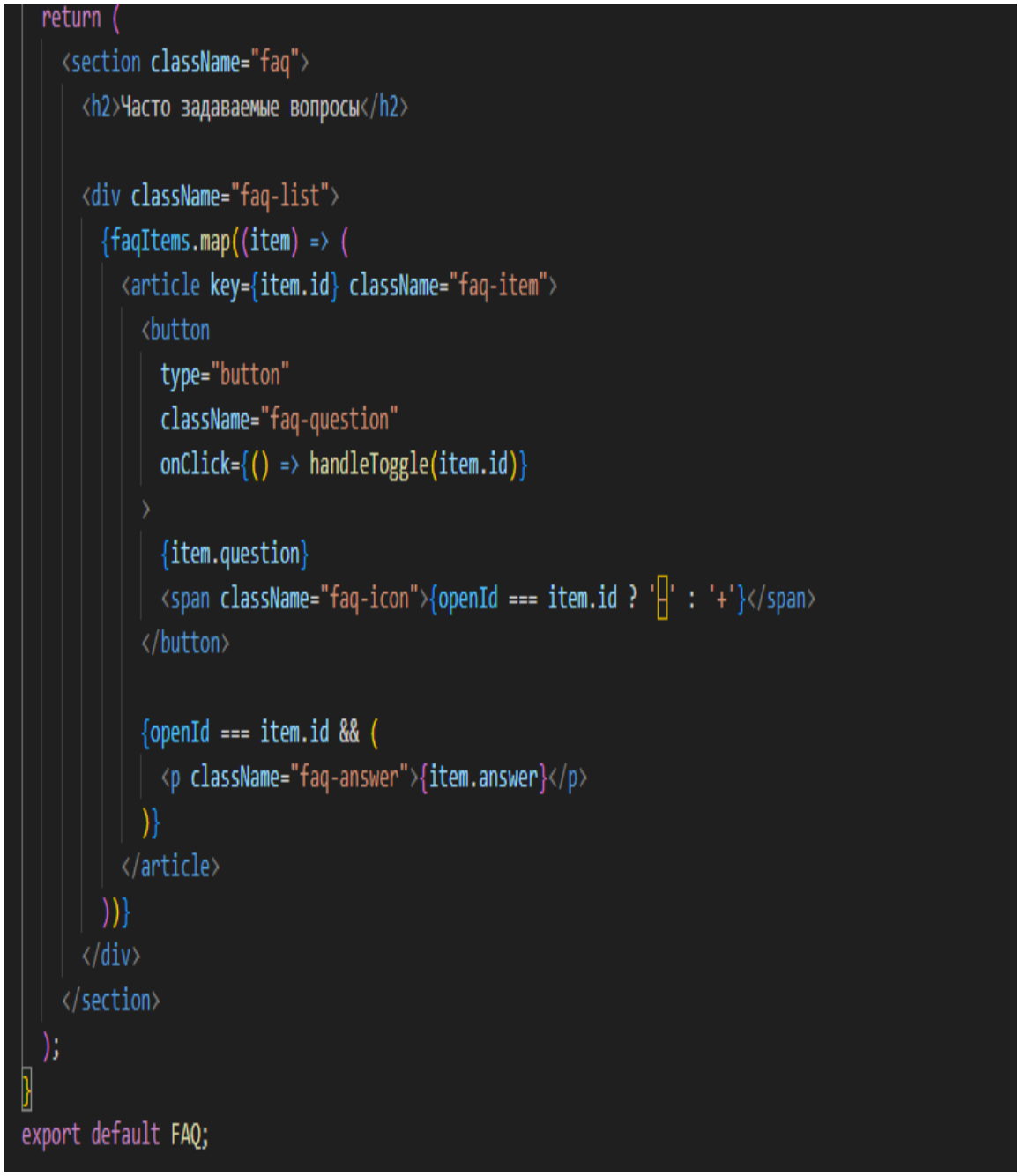
```
import FAQ from './FAQ.jsx';
```

На рисунке 24 показана 1 часть кода FAQ.

```
import React, { useState } from 'react';
const faqItems = [
  {
    id: 1,
    question: 'Сколько времени занимает разработка сайта?',
    answer: 'В среднем от 1 до 4 недель, в зависимости от объёма страниц и сложности дизайна.',
  },
  {
    id: 2,
    question: 'Можно ли доработать уже существующий сайт?',
    answer: 'Да, мы можем доработать текущий проект: обновить дизайн, добавить страницы или исправи
  },
  {
    id: 3,
    question: 'Помогаете ли вы с доменом и хостингом?',
    answer: 'Да, подскажем подходящий тариф, поможем зарегистрировать домен и разместить сайт.',
  },
  {
    id: 4,
    question: 'Работаете ли вы по договору?',
    answer: 'Да, при необходимости оформляем договор и спецификацию работ.',
  },
];
function FAQ() {
  const [openId, setOpenId] = useState(null);
  const handleToggle = (id) => {
    setOpenId((prev) => (prev === id ? null : id));
  };
  return (
```

Рисунок 24 - 1 часть кода

На рисунке 25 показана 2 часть кода FAQ.



```
  return (
    <section className="faq">
      <h2>Часто задаваемые вопросы</h2>

      <div className="faq-list">
        {faqItems.map((item) => (
          <article key={item.id} className="faq-item">
            <button
              type="button"
              className="faq-question"
              onClick={() => handleToggle(item.id)}
            >
              {item.question}
              <span className="faq-icon">{openId === item.id ? '-' : '+'}</span>
            </button>

            {openId === item.id && (
              <p className="faq-answer">{item.answer}</p>
            )}
          </article>
        ))}
      </div>
    </section>
  );
}
export default FAQ;
```

Рисунок 25 - вторая часть кода

На рисунке 26 показано как это выглядит.

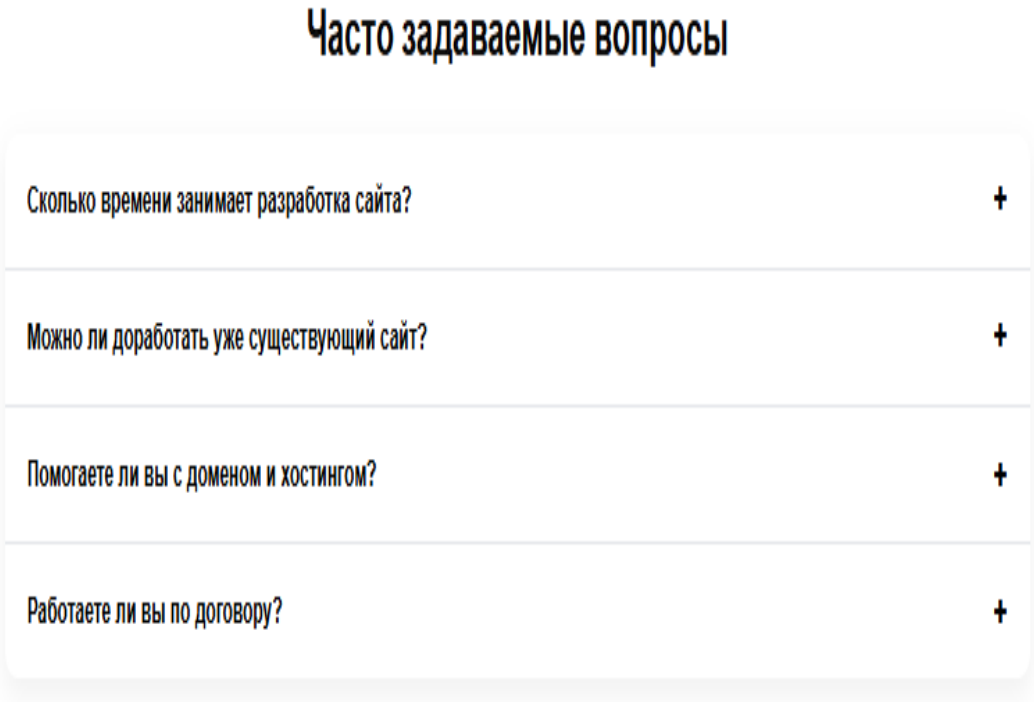


Рисунок 26 - блок FAQ

При раскрытии каждого из вопросов внизу показывается ответ на тот или иной вопрос.

На рисунке 27 изображено что будет если развернуть вопрос.

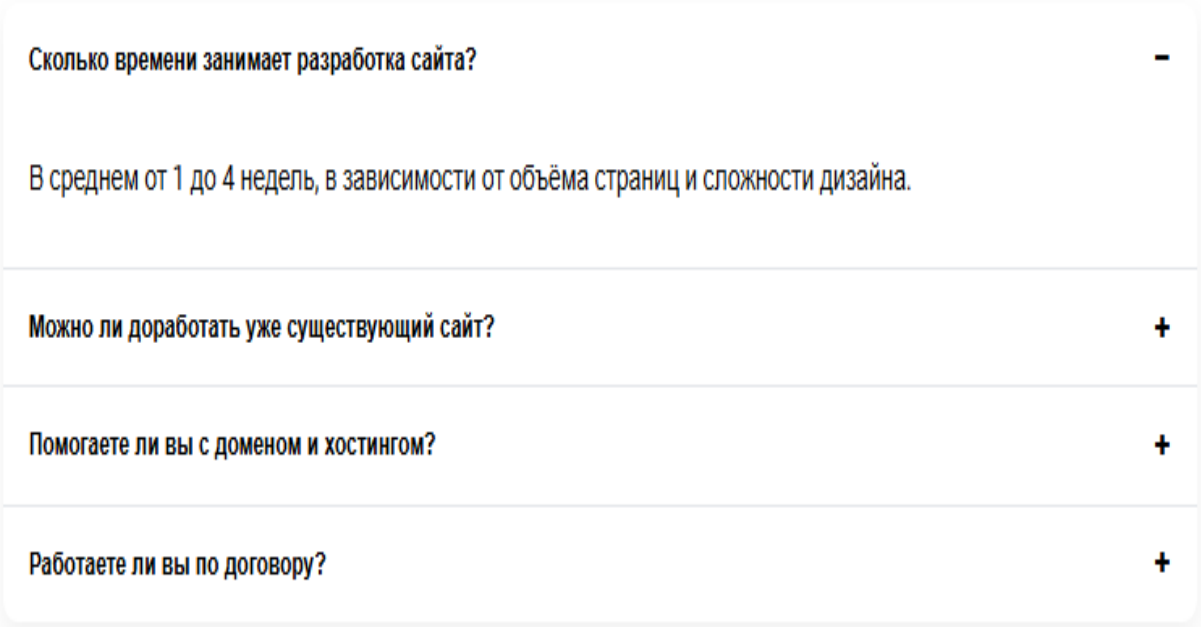


Рисунок 27 - развернут вопрос

При разворачивании следующего вопроса ответ другого сразу же скрывается, что не позволяет открыть сразу все ответы.

На рисунке 28 показан css для FAQ.

```css
.faq {
  max-width: 900px;
  margin: 40px auto;
}

.faq h2 {
  text-align: center;
  margin-bottom: 20px;
}

.faq-list {
  border-radius: 10px;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.06);
  background: #ffffff;
  overflow: hidden;
}

.faq-item + .faq-item {
  border-top: 1px solid #e5e7eb;
}

.faq-question {
  width: 100%;
  padding: 14px 18px;
  text-align: left;
  background: none;
  border: none;
  font-weight: 600;
  cursor: pointer;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.faq-icon {
  font-size: 1.2rem;
}

.faq-answer {
  padding: 0 18px 14px 18px;
  color: #555;
  font-size: 0.95rem;
}
```

Рисунок 28 - css FAQ

2. Пункт второй

Блок «Этапы работы» был реализован как отдельная секция на главной странице, которая наглядно показывает пользователю, как строится процесс разработки сайта: Анализ - Дизайн - Разработка - Тестирование - Поддержка. Для этого внутри компонента был создан массив шагов с номерами, названиями и краткими описаниями, а затем этот массив выводится в JSX с помощью map в виде набора карточек с единым стилем. Каждая карточка содержит крупный номер шага, заголовок и поясняющий текст, а сама секция оформлена как адаптивная сетка, которая корректно перестраивается на разных размерах экрана. Такое решение одновременно улучшает понятность для пользователя (он видит последовательность работ и понимает, что будет происходить с его проектом) и демонстрирует в отчёте работу с массивами данных, компонентным подходом и современным UI-паттерном «steps / workflow» в React-приложении.

На рисунке 29 показан код для блока этапов работ.

```
const steps = [
  {
    id: 1,
    title: 'Анализ',
    text: 'Изучаем цели и задачи, собираем требования и особенности бизнеса.',
  },
  {
    id: 2,
    title: 'Дизайн',
    text: 'Проектируем структуру страниц и создаём прототипы интерфейсов.',
  },
  {
    id: 3,
    title: 'Разработка',
    text: 'Верстаем интерфейсы, настраиваем логику и подключаем необходимые сервисы.',
  },
  {
    id: 4,
    title: 'Тестирование',
    text: 'Проверяем работу сайта на разных устройствах и исправляем ошибки.',
  },
  {
    id: 5,
    title: 'Поддержка',
    text: 'Следим за стабильной работой проекта и помогаем с обновлениями.',
  },
];

return (
```

Рисунок 29 - Блок этапы работ

На рисунке 30 показано как это выглядит на сайте.

Рисунок 30 - этапы работы

3. Пункт третий.

Блок с отзывами на главной странице представляет собой отдельный смысловой модуль, задача которого - показать потенциальному пользователю, что с сайтом и его автором уже имели дело другие люди и остались довольны результатом. Такой элемент часто называют «социальным доказательством», и он является важной частью современных лендингов и портфолио: посетитель видит не только абстрактные услуги и примеры работ, но и живые отклики, написанные от лица «реальных» заказчиков или пользователей. В рамках проекта это позволяет продемонстрировать, что страница не просто формально перечисляет возможности, а выстраивает доверительную коммуникацию с теми, кто её просматривает.

Технически блок отзывов был реализован в виде отдельного React-компонента Testimonials, который подключается в App.jsx как обычный дочерний компонент и рендерится в общей структуре страницы после секции портфолио. Такой подход подчёркивает компонентное мышление: логика и разметка отзывов изолированы от остального кода, поэтому при необходимости компонент можно переиспользовать на других страницах или дорабатывать независимо от основного контента. Размещение именно после портфолио выбрано осознанно: пользователь сначала знакомится с реальными примерами работ, а затем сразу видит, как эти работы или услуги оценивают другие люди, что усиливает эффект восприятия.

Внутри компонента Testimonials используется массив объектов с данными об отзывах: у каждого элемента есть как минимум имя автора и текст отзыва, при желании могут быть добавлены дополнительные поля, такие как должность, компания или оценка. Такой формат хранения информации удобен тем, что добавление или изменение отзывов сводится к работе с данными, а не с верстkой: чтобы показать новый отзыв, достаточно дописать ещё один объект в массив, не трогая структуру JSX. Вывод отзывов на страницу организован с помощью метода map, который перебирает массив и для каждого элемента создаёт JSX-карточку с единым шаблоном оформления. Это демонстрирует умение работать с коллекциями данных в React и поддерживать принцип «один шаблон — много элементов».

С точки зрения верстки и визуального оформления каждый отзыв представлен в виде отдельной карточки с небольшим отступом, фоном и скруглёнными углами, что визуально отделяет блок отзывов от остального текста и делает его легче читаемым. Карточки располагаются в сетке: на широких экранах они могут стоять в один или два столбца, а на мобильных - выстраиваться друг под другом, что обеспечивает адаптивность и удобство чтения с разных устройств. Внутри карточки, как правило, есть заголовок с именем автора (иногда выделенный более крупным шрифтом) и основной текст отзыва, оформленный обычным абзацем. Это создаёт привычную для пользователя структуру: сначала он видит, кто говорит, а затем - что именно он говорит.

Отдельное внимание можно уделить вопросу доступности такого блока. Хотя отзывы по своей сути являются статическим контентом, важно соблюдать базовые правила: использовать корректные уровни заголовков (например, общий заголовок секции «Отзывы» как <h2> и имена авторов как заголовки карточек уровнем ниже) и избегать излишних декоративных элементов, которые могут запутать пользователей скринридеров. В отчёте можно отметить, что структура блока отзывов выстроена логично и семантически понятно: есть обёртка-секция, заголовок раздела и повторяющийся список карточек, что облегчает навигацию по странице и соответствует рекомендациям по доступной верстке.

В целом реализация блока с отзывами решает сразу несколько задач. Во-первых, повышает доверие к содержимому страницы за счёт демонстрации мнений «третьих лиц». Во-вторых, показывает в отчёте владение ключевыми приёмами работы в React: создание отдельного компонента, хранение данных в массиве объектов, вывод этих данных в виде списка карточек через map, а также базовую адаптивную верстку. В-третьих, подчёркивает ориентацию на пользовательский опыт: блок отзывов логично вписан в сценарий просмотра страницы и делает её более живой и завершённой по восприятию.

На рисунке 31 показана 1 часть кода блока отзывов .

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

const testimonials = [
  {
    id: 1,
    name: 'Анна',
    role: 'владелец интернет-магазина',
    text: 'Команда быстро сделала адаптивный сайт и помогла с запуском рекламы.',
  },
  {
    id: 2,
    name: 'Игорь',
    role: 'индивидуальный предприниматель',
    text: 'Понравилось, что всё объясняли простым языком и учли мои пожелания по дизайну.',
  },
  {
    id: 3,
    name: 'Мария',
    role: 'маркетолог',
    text: 'Сайт стал загружаться быстрее, а заявки с формы теперь приходят стабильно.',
  },
  {
    id: 4,
    name: 'Дмитрий',
    role: 'руководитель проекта',
    text: 'Сделали понятную админку, теперь команда сама обновляет контент на сайте.',
  },
  {
    id: 5,
    name: 'Елена',
    role: 'фриланс-дизайнер',
    text: 'Быстро сверстали мой макет и помогли с хостингом и доменом.',
  },
];

function getRandomIndex(exceptIndex) {
  const max = testimonials.length;
  if (max <= 1) return 0;

  let index = exceptIndex;
  while (index === exceptIndex) {
    index = Math.floor(Math.random() * max);
  }
  return index;
}
```

Рисунок 31 - 1 часть кода

На рисунке 32 показана 2 часть кода.

```jsx
function Testimonials() {
  const [currentIndex, setCurrentIndex] = useState(0);
  const current = testimonials[currentIndex];

  const handleRandomClick = () => {
    const nextIndex = getRandomIndex(currentIndex);
    setCurrentIndex(nextIndex);
  };

  return (
    <section className="testimonials">
      <h2>Отзывы клиентов</h2>

      <article className="testimonial-card testimonial-card--single">
        <p className="testimonial-text">“{current.text}”</p>
        <p className="testimonial-author">
          {current.name}
          {' — '}
          <span>{current.role}</span>
        </p>

        <button
          type="button"
          className="testimonial-random-btn"
          onClick={handleRandomClick}
        >
          Случайный отзыв
        </button>
      </article>
    </section>
  );
}

export default Testimonials;
```

Рисунок 32 - 2 часть кода

Внедряется это в App.jsx очень просто, надо лишь вписать в main <Testimonials />.

На рисунке 33 показан css для testimonials.

```css
.testimonials {
  max-width: 900px;
  margin: 40px auto;
}

.testimonials h2 {
  text-align: center;
  margin-bottom: 20px;
}

.testimonials-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
  gap: 20px;
}

.testimonial-card {
  padding: 18px 20px;
  border-radius: 10px;
  background: #ffffff;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.08);
}

.testimonial-text {
  margin-bottom: 10px;
  font-style: italic;
}

.testimonial-author span {
  color: #555;
  font-size: 0.9rem;
}


.theme-dark .testimonial-card {
  background: #2a2a2a;
}

.testimonial-card--single {
  max-width: 600px;
  margin: 0 auto;
  text-align: center;
}
```

Рисунок 33 - css для блока отзывов

На рисунке 34 показано как это выглядит на сайте.

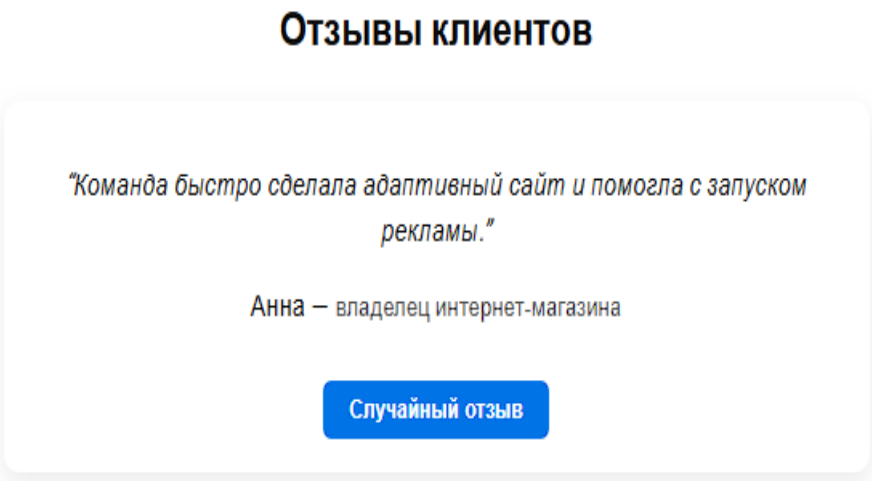


Рисунок 34 - отзывы ч.1

Нажав на кнопку «Случайный отзыв» выбирается и выводится 1 из 5 отзывов, написанных заранее. Численность отзывов можно как уменьшать, так и увеличивать. На рисунке 35 показана 2 часть отзывов.

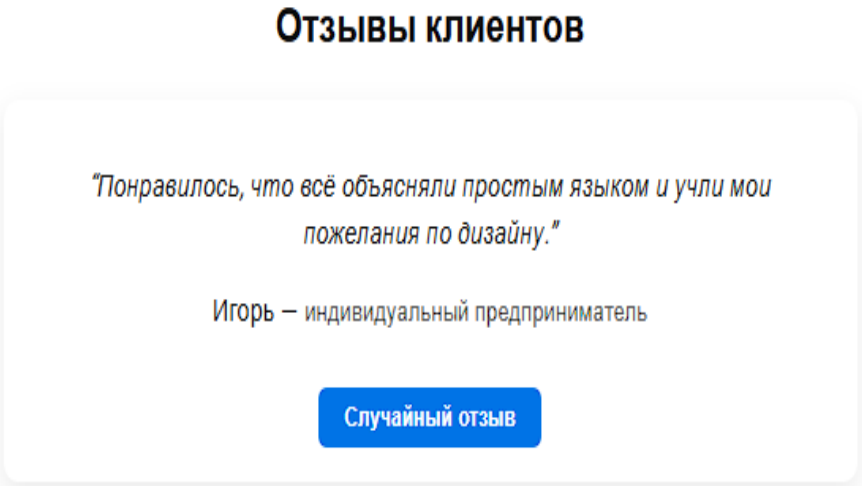


Рисунок 35 - отзывы ч.2

На рисунке 36 показана 3 часть отзывов.

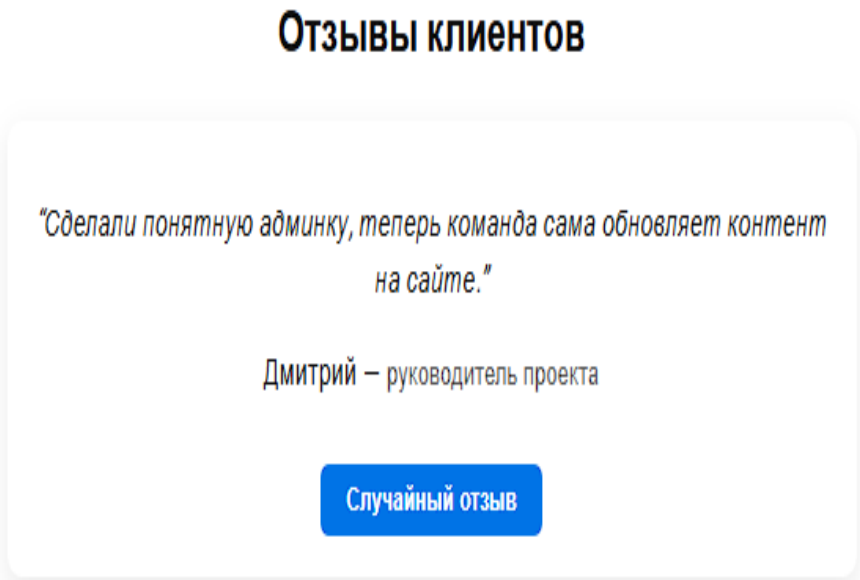


Рисунок 36 - отзывы ч.3

На рисунке 37 показана 4 часть отзывов.

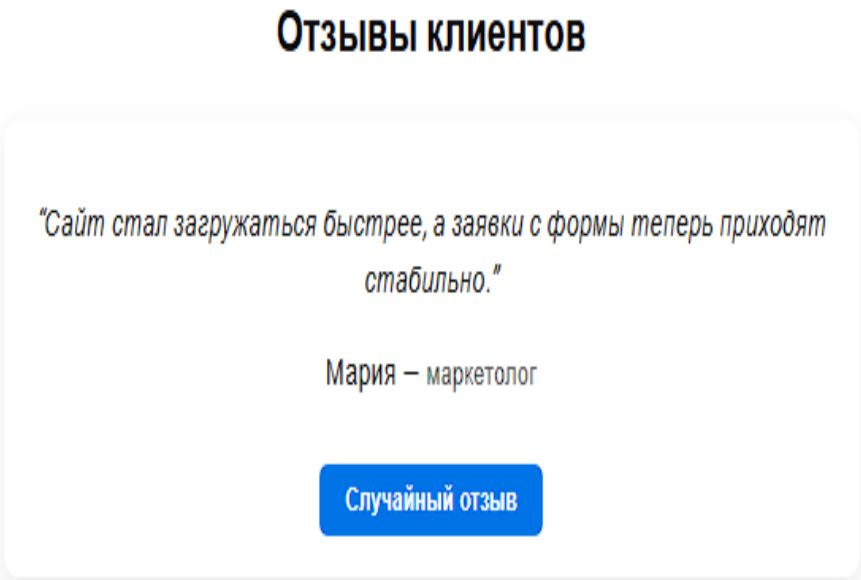


Рисунок 37 - отзывы ч.4

На рисунке 38 показана 5 часть отзывов.

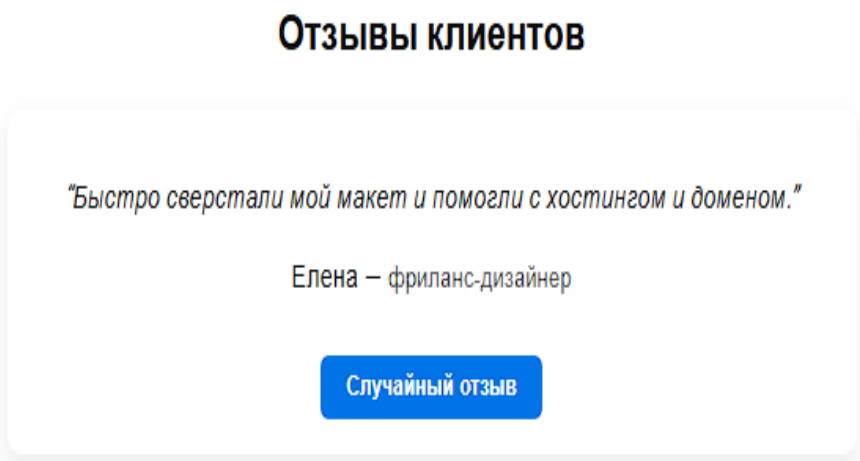


Рисунок 38 - отзывы ч.5

Вывод: Главный компонент App.jsx выполняет роль «центральной витрины» проекта: в нём сосредоточены все ключевые блоки главной страницы - шапка с навигацией и переключателем темы, секция с услугами, блок «Этапы работы», портфолио с модальным просмотром изображений, а также подключённые компоненты отзывов и FAQ. Через App.jsx пользователь получает целостное представление о сайте: кто его создал, какие услуги предлагаются, как выстроен процесс работы, какие есть примеры выполненных проектов и что о них говорят условные «клиенты». С технической точки зрения компонент демонстрирует использование состояния (управление модальным окном галереи), работу с массивом данных для этапов работы, взаимодействие с дочерними компонентами и базовую поддержку светлой/тёмной темы, что делает его хорошей иллюстрацией применения React для построения одностраничного портфолио.

Лабораторная работа №5 «About.jsx»

1. Пункт первый.

Страница about.jsx была существенно доработана и превратилась из простой текстовой «заглушки» в полноценный информационный блок о проекте и его авторах. Во-первых, помимо базового текста «Наша история» были добавлены наглядные визуальные секции: блок «Наша команда» с логотипом и перечислением участников, а также отдельный блок «Автор страницы» с фотографией студента и кратким описанием. Это делает раздел более «живым» - пользователь видит не абстрактный сайт, а конкретных людей за ним, что соответствует современным рекомендациям по построению About-страниц. Во-вторых, в компонент были добавлены новые смысловые блоки «Наши ценности» и «Навыки и стек», реализованные в виде карточек и списка «чипов». В секции ценностей каждая карточка описывает ключевые принципы работы (честность, качество кода, уважение к пользователю), что помогает структурировано рассказать о подходе к разработке, а не просто перечислить факты. Список навыков оформлен как массив строк, который выводится через map в виде визуально отделённых элементов — это демонстрирует владение базовыми навыками React (работа с массивами и JSX) и параллельно аккуратно показывает технический стек (HTML, CSS, JavaScript, React, адаптивная вёрстка и доступность). Кроме того, в about.jsx сохранена общая каркасная структура сайта: единая шапка с навигацией, переключатель светлой и тёмной темы, хлебные крошки для ориентации пользователя и общий футер, что подчёркивает целостность макета и единый стиль всех страниц. Для иллюстрации в отчёте целесообразно привести несколько скриншотов: общий вид страницы «О нас» целиком; фрагмент с блоками «Наша команда» и «Автор страницы»; отдельный кадр с карточками «Наши ценности»; и скрин кода, где показано формирование массива навыков и его вывод через map. Такие изображения наглядно подтвердят как визуальные изменения интерфейса, так и использование компонентного подхода и работы с данными в React.

На рисунке 39 показана первая часть кода для карточек стека.

```
const skills = [
  'HTML5 и семантическая вёрстка',
  'CSS3, Flexbox, Grid',
  'JavaScript (ES6+)',
  'React и компоненты',
  'Адаптивный дизайн',
  'Доступность (A11y)',
];
```

Рисунок 39 – стек ч.1

На рисунке 40 показана вторая часть кода для карточек стека

```
  <section className="about-skills">
    <h2>Навыки и стек</h2>
    <ul className="about-skills__list">
      {skills.map((skill) => (
        <li key={skill} className="about-skills__item">
          {skill}
        </li>
      ))}
    </ul>
  </section>
</main>
```

Рисунок 40 - стек ч.2

На рисунке 41 показано как это выглядит на сайте.

Рисунок 41 - блок стек на сайте

2. Пункт второй.

Также был добавлен блок «Наши ценности». На рисунке 42 показан код для этого блока.

```
<section className="about-values">
  <h2>Наши ценности</h2>
  <div className="about-values__grid">
    <article className="about-card">
      <h3>Честность и открытость</h3>
      <p>
        Говорим о сроках и возможностях честно, без лишних обещаний и
        скрытых условий.
      </p>
    </article>
    <article className="about-card">
      <h3>Качество кода</h3>
      <p>
        Стараемся писать понятный и поддерживаемый код, чтобы проект
        можно было развивать дальше.
      </p>
    </article>
    <article className="about-card">
      <h3>Уважение к пользователю</h3>
      <p>
        Думаем о том, как человек будет пользоваться страницей, а не
        только о том, «как бы сверстать».
      </p>
    </article>
  </div>
</section>
```

Рисунок 42 - блок стек и навыки

На рисунке 43 показано как это выглядит на сайте.

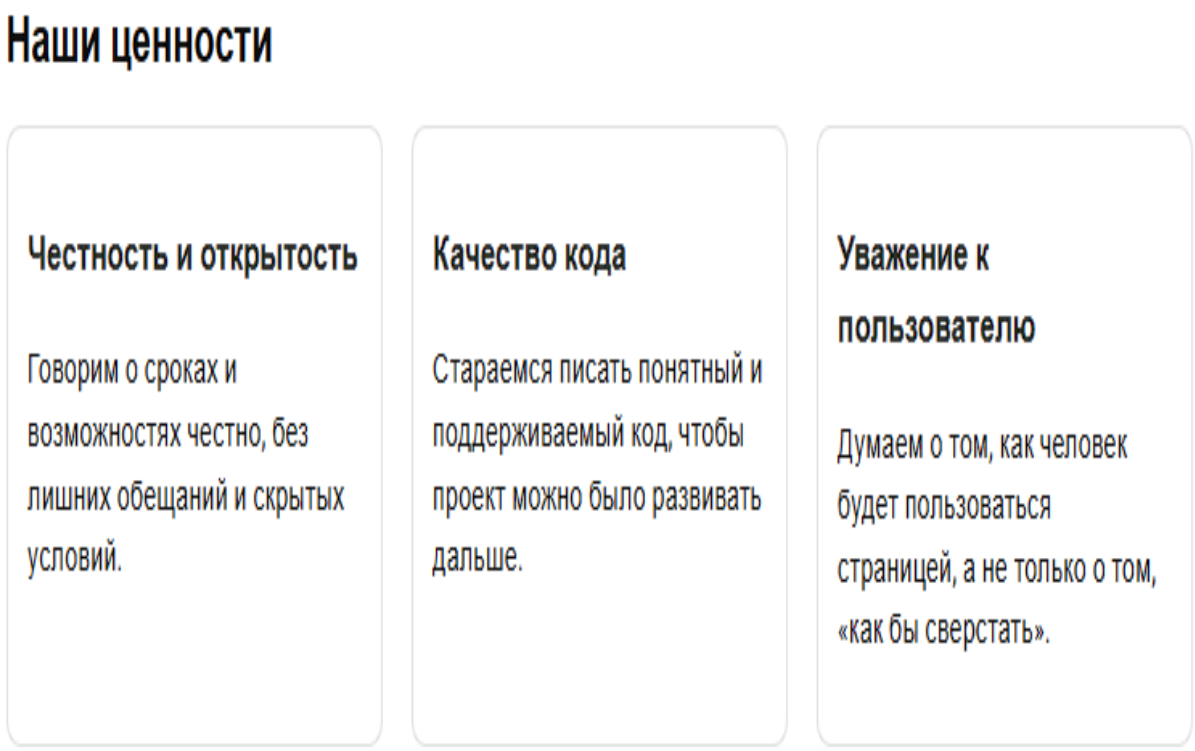


Рисунок 43 - наши ценности на сайте

Вывод: Страница about.jsx была заметно расширена и из простой текстовой секции превратилась в полноценный информативный раздел о проекте и его авторах. Во-первых, помимо базового описания истории добавлены наглядные блоки «Наша команда» и «Автор страницы» с изображениями и подписями, что делает представление разработчиков более личным и понятным для пользователя. Во-вторых, внедрены новые смысловые секции «Наши ценности» и «Навыки и стек»: первая в виде карточек чётко формулирует подход к работе (честность, качество кода, уважение к пользователю), а вторая с помощью списка навыков демонстрирует используемый технологический стек и аккуратно показывает уровень владения современными фронтенд-технологиями. В совокупности эти изменения усиливают доверие к сайту, делают About-страницу более структурированной и наглядно показывают умение работать с компонентной разметкой, массивами данных и адаптивным оформлением в React.

Лабораторная работа №6 «создание системы авторизации без Backend»

1. Пункт первый.

В начале файла подключаются необходимые хуки (useState) и функции роутера (Link, useNavigate), а также компонент получает через пропсы текущее состояние темы (mode), переключатель темы (toggleTheme), флаг авторизации (isAuth) и функцию login из AppThemeProvider. Это позволяет странице логина выглядеть и вести себя так же, как остальные разделы сайта, не дублируя логику темы и авторизации.Внутри компонента объявлены состояния email password и error, которые управляют значениями полей формы и выводом сообщений об ошибке. При отправке формы в обработчике handleSubmit выполняется простая проверка: оба поля не должны быть пустыми; при нарушении показывается текстовая ошибка под формой. Если данные введены, вызывается функция login(email), которая сохраняет email в localStorage и устанавливает глобальный флаг isAuth=true, после чего выполняется программная навигация navigate('/') на главную страницу. Благодаря этому после входа в шапке сразу появляется подпись «Вошли как …» и кнопка «Выйти», а состояние авторизации сохраняется между перезагрузками за счёт записи флага isAuth в локальное хранилище браузера. Отдельно можно отметить, что сама страница логина визуально вписана в общий дизайн сайта: в ней используется та же шапка с навигацией, кнопка переключения светлой/тёмной темы, а основной контент оформлен как простая форма с полями и кнопкой. Такой подход подчёркивает компонентный стиль разработки: логика

авторизации вынесена в AppThemeProvider, а login.jsx отвечает только за пользовательский интерфейс ввода и базовую валидацию. В отчёте важно явно подчеркнуть, что данная реализация авторизации носит учебный характер: пароли нигде не шифруются и не проверяются на сервере, а все данные хранятся только в localStorage и используются для демонстрации работы с формами, глобальным состоянием и условным рендерингом элементов интерфейса в React.

На рисунке 44 показана первая часть кода для Login.jsx.

```jsx
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
function Login({ mode, toggleTheme, isAuth, login }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();
  const isLight = mode === 'light';
  const handleSubmit = (event) => {
    event.preventDefault();
    if (!email.trim() || !password.trim()) {
      setError('Заполните email и пароль.');
      return;
    }
    login(email);
    navigate('/');
  };
  return (
    <>
      <header>
        <div className="header-left">
          <Link to="#">Личная веб страница-портфолио</Link>
        </div>
        <nav>
          <ul>
            <li><Link to="/">Главная</Link></li>
            <li><Link to="/about">О нас</Link></li>
            <li><Link to="/contacts">Контакты</Link></li>
          </ul>
        </nav>
        <button
          type="button"
          className="header__theme-toggle"
          onClick={toggleTheme}
        >
          {isLight ? '🌙 Тёмная тема' : '🌞 Светлая тема'}
        </button>
      </header>
      <main className={isLight ? 'page page--light' : 'page page--dark'}>
        <h1>Вход на сайт</h1>
        <form className="login-form" onSubmit={handleSubmit}>
          <label htmlFor="email">Email</label>
```

Рисунок 44 - Login.jsx ч.1

На рисунке 45 показана вторая часть кода для Login.jsx

```
          <input
            id="email"
            type="email"
            placeholder="example@mail.ru"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
          <label htmlFor="password">Пароль</label>
          <input
            id="password"
            type="password"
            placeholder="Ваш пароль"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          {error && <p className="form__error">{error}</p>}
          <button type="submit" className="btn-submit">
            Войти
          </button>
        </form>
      </main>
      <footer>
        <p>&copy; 2025 Личная веб страница-портфолио. Все права защищены.</p>
        <p>Телефон: +7 (926) 108-40-92</p>
      </footer>
    </>
  );
}
export default Login;
```

Рисунок 45 - Login.jsx ч.2

Также был затронут и main.jsx. На рисунке 46 показан код Route в main.jsx.

```
<Route
  path="/login"
  element={
    <Login
      toggleTheme={toggleTheme}
      mode={mode}
      isAuth={isAuth}
      login={login}
    />
  }
```

Рисунок 46 - Route в main.jsx

function App({ toggleTheme, mode, isAuth, logout }) {

const userEmail = localStorage.getItem('userEmail') || 'пользователь';

На рисунке 47 показан код для .css.

```css
.header__login-link,
.header__logout {
  padding: 6px 12px;
  border-radius: 6px;
  border: 1px solid #ffffff55;
  background: transparent;
  color: inherit;
  font-size: 0.9rem;
  cursor: pointer;
  text-decoration: none;
}

.header__user {
  font-size: 0.85rem;
  opacity: 0.9;
}
```

Рисунок 47 - .css для логина

Для App.jsx, About.jsx и Contacts.jsx нужно также вставить кнопку регистрации. Он везде идентичный поэтому покажу на примере App.jsx. На рисунке 48 показан код самого логина.

```
{isAuth ? (
  <>
    <span className="header__user">
      Вошли как {userEmail}
    </span>
    <button
      type="button"
      className="header__logout"
      onClick={logout}
    >
      Выйти
    </button>
  </>
) : (
  <Link to="/login" className="header__login-link">
    Войти
  </Link>
)}
```

Рисунок 48 - код логина на страницах

На рисунке 49 показано как это выглядит на сайте.

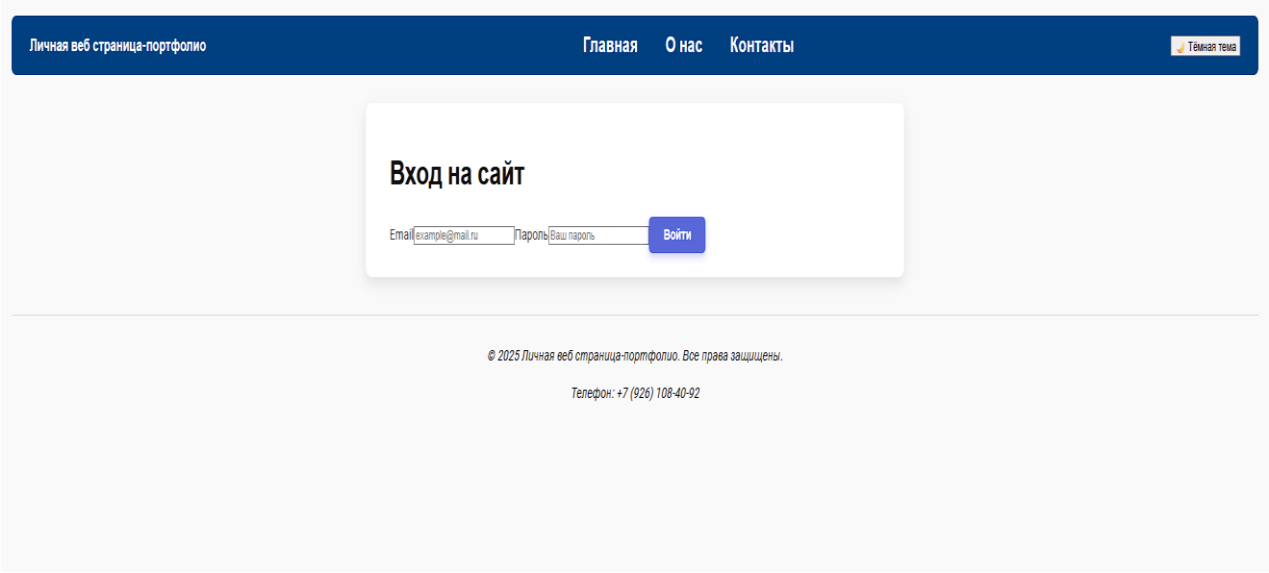


Рисунок 49 - окно регистрации

После введения данных и нажатия кнопки около переключения темы будет показываться ваша почта и кнопка выйти.

На рисунке 50 показано как это выглядит.

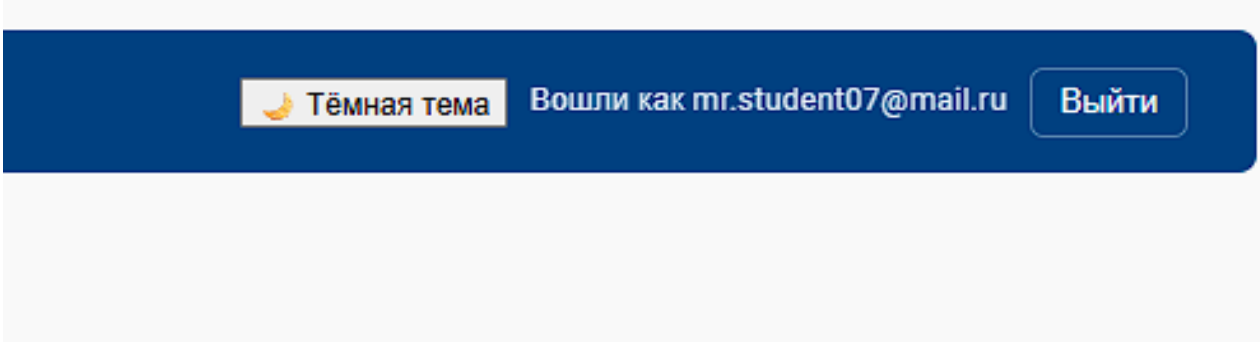


Рисунок 50 - после регистрации

На рисунке 51 показано как выглядит окно регистрации с тёмной темой.

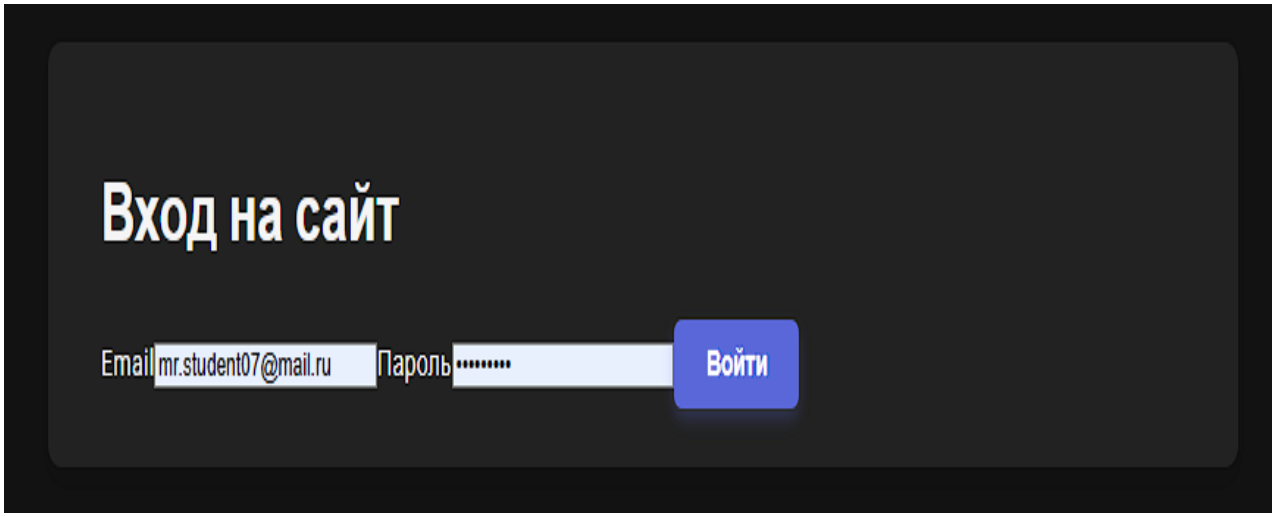


Рисунок 51 - регистрация темная тема

После заключительного обновления кода древо выглядит таким образом:

Src/

About.jsx

App.jsx

Contacts.jsx

Main.jsx

Styles.css

ThemeProviderMUI.jsx

Testimonials.jsx

FAQ.jsx

Login.jsx

Assets/

Atlus-1.jpg

Atlus-2.jpg

Team-logo.jpg

Student.jpg

Вывод: Компонент Login.jsx отвечает за ввод email и пароля, базовую валидацию и запуск процесса входа: при успешной отправке формы вызывается функция login, которая сохраняет адрес пользователя в localStorage и устанавливает глобальный флаг авторизации isAuth, после чего выполняется переход на главную страницу. Логика авторизации и хранения состояния была добавлена в AppThemeProvider: туда введены состояния isAuth, login и logout, а также синхронизация флага с localStorage, благодаря чему информация о входе не теряется при перезагрузке страницы.Во все основные страницы (App.jsx, About.jsx, Contacts.jsx) были добавлены новые пропсы isAuth и logout, а шапка сайта получила динамический блок: если пользователь не авторизован, показывается ссылка «Войти», ведущая на /login, а после входа — подпись «Вошли как …» с email из localStorage и кнопка «Выйти», сбрасывающая состояние авторизации. Таким образом, интерфейс наглядно отражает текущее состояние пользователя, а все изменения реализованы только на стороне клиента без полноценного бэкенда, что подходит для учебного портфолио и одновременно демонстрирует умение работать с формами, глобальным состоянием, localStorage и условным рендерингом в React.

Итоговый вывод: Итогом выполнения лабораторных работ по React и JSX стало создание полноценного одностраничного портфолио-сайта с несколькими связанными страницами, построенного на компонентном подходе и современном стеке фронтенд-технологий. Главный компонент App.jsx объединяет ключевые разделы: шапку с навигацией и переключателем светлой/тёмной темы, блок «Наши услуги», секцию «Этапы работы», портфолио с модальным просмотром изображений, а также отдельные компоненты FAQ и отзывов, что демонстрирует умение разбивать интерфейс на переиспользуемые JSX-компоненты и работать со списками данных через map. Отдельные страницы «О нас» и «Контакты» показывают расширенное владение React: в about.jsx реализованы смысловые блоки «Наша команда», «Автор страницы», «Наши ценности» и «Навыки и стек», основанные на массивах данных и адаптивных сетках, а в contacts.jsx — интерактивная страница с картой Google Maps, модальным окном формы обратной связи, валидацией полей и toast-уведомлением через Snackbar из MUI. Дополнительно в проект была внедрена учебная система авторизации: компонент Login.jsx, глобальные функции login/logout и флаг isAuth в провайдере темы, синхронизируемый с localStorage, что позволяет условно менять элементы шапки («Войти» / «Вошли как… / Выйти») без реального бэкенда, но с демонстрацией работы с формами, состоянием и хранением данных в браузере. В совокупности эти нововведения показали на практике ключевые темы курса по React и JSX: создание и композицию компонентов, работу с пропсами и состоянием,

условный рендеринг, обработку событий, формы и валидацию, маршрутизацию с React Router, подключение сторонних библиотек (Material UI) и использование JSX как основного способа описания пользовательского интерфейса. Такой результат соответствует типичному учебному портфолио-проекту на React: получился не просто статичный HTML-макет, а интерактивное, тематически оформленное приложение, демонстрирующее готовность применять React в реальных веб-проектах.