МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Р.Е. АЛЕКСЕЕВА»

(НГТУ)

|  |  |  |
| --- | --- | --- |
| Институт | Институт радиоэлектроники и информационных технологий | |
| Направление подготовки | 09.03.02 Информационные системы и технологии | |
| Направленность (профиль) образовательной программы | | Информационные технологии в дизайне |
| Кафедра | Графические информационные системы | |

**ОТЧЕТ**

**по прохождению Ознакомительной практики**

|  |  |
| --- | --- |
| Выполнил |  |
| Студент гр. \_\_Калякин Данила Сергеевич 23-ИСТ-4-1\_\_\_\_\_\_\_ | Ф.И.О. |
| Руководитель практики от кафедры Филинских А.Д. | |
| Степень, звание | к.т.н., доцент |
|  |  |
|  |  |
| Отчет защищен с оценкой | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Дата защиты | «19» июля 2024г. |

Нижний Новгород

2024 год

**Оглавление**

[ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ 3](#_Toc172041075)

[Введение. 6](#_Toc172041076)

[Основная часть отчета в соответствии с заданием на практику 7](#_Toc172041077)

[Заключение 16](#_Toc172041078)

[Список литературы 17](#_Toc172041079)

[Приложение 18](#_Toc172041080)

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Р.Е. АЛЕКСЕЕВА»

(НГТУ)

Кафедра Графические информационные системы

|  |  |
| --- | --- |
|  | Утверждаю: |
|  | Заведующий кафедрой |
|  | А.Д. Филинских |
|  | «06» июля 2024г. |

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на Учебную практику (Ознакомительная практика)

Студента гр. 23-ИСТ-4-1 Ф.И.О Калякина Данилы Сергеевича

Направление подготовки/специальность: 09.03.02 Информационные системы и технологии

Профиль подготовки: Информационные технологии в дизайне

Место прохождения практики НГТУ им. Р.Е. Алексеева, кафедра «Графические информационные системы»

Время прохождения практики

Дата начала практики «06» июля 2024 г.

Дата окончания практики «19» июля 2024г.

**Тема индивидуального задания:**

**Разработка контактной формы с валидацией полей и сохранением данных в файл .json**

Содержание практики

*Во время прохождения практики студент обязан:*

**Ознакомиться:** с разделом HTML о формах и элементы формы (input, textarea, button). С возможностями CSS для стилизации формы (цвета, шрифты, размеры, расположение элементов). С возможностями JS для добавления функционала (проверка полей, обработка нажатия кнопки отправки).

**Изучить:** следующие концепции:

1. Проверка обязательных полей (required).
2. Проверка формата email (pattern).
3. Проверка минимальной и максимальной длины текстовых полей (maxlength, minlength).
4. Проверка пароля на соответствие определенным требованиям (длина, использование букв, цифр, символов).
5. Обработка события нажатия кнопки отправки формы (addEventListener).
6. Поднятие сервера с помощью Node.js

**Выполнить следующие виды работ по приобретению практических навыков:**

Исследовать описанные выше концепции, произвести верстку страницы на html, добавить необходимые стили в css, научиться практически применять Node.js: устанавливать зависимости, поднимать сервер, а также изучить стандартные команды для выполнения описанных задач.

Должность на практике практикант

Планируемые результаты обучения при прохождении практики

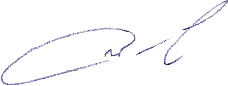
| **Планируемые результаты освоения образовательной программы** | **Планируемые результаты обучения при прохождении практики** | | | |
| --- | --- | --- | --- | --- |
| **Код компетенции** | **Знать** | **Уметь** | **Владеть** | |
| ОПК-1 Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности | * основы математики, физики, вычислительной техники и программирования; | * решать стандартные профессиональные задачи с применением естественнонаучных и общеинженерных знаний, методов математического анализа и моделирования; | * навыками теоретического и экспериментального исследования объектов профессиональной деятельности; |
| ОПК-3 Способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности; | * принципы, методы и средства решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности | - решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности | * навыками подготовки обзоров, аннотаций, составления рефератов, научных докладов, публикаций и библиографии по научно-исследовательской работе с учетом требований информационной безопасности | |
| ОПК-6 Способен разрабатывать алгоритмы и программы, пригодные для практического применения в области информационных систем и технологий; | |  | | --- | | * основные языки программирования и работы с базами данных, операционные системы и оболочки, современные программные среды разработки информационных систем и технологий | | * применять языки программирования и работы с базами данных, современные программные среды разработки информационных систем и технологий для автоматизации бизнес-процессов, решения прикладных задач различных классов, ведения баз данных и информационных хранилищ | * навыками программирования, отладки и тестирования прототипов программно-технических комплексов задач | |

Руководитель практики от кафедры

Зав. каф. ГИС к.т.н, доцент Филинских А.Д,

(ученое звание и степень) (подпись) Ф.И.О.

Задание на практику получил:

Студент \_\_\_\_\_\_\_Калякин Данила Сергеевич\_\_\_\_\_\_\_

(подпись) (ФИО) «06» июля 2024г.

**РАБОЧИЙ ГРАФИК (ПЛАН)**

**ПРОВЕДЕНИЯ Учебной практики**

Студента гр.23-ИСТ-4-1 Ф.И.О. \_\_\_\_\_Калякин Данила Сергеевич\_\_\_\_\_

| №№ п/п | Разделы (этапы) практики  (заполняется в соответствии с заданием) | Сроки выполнения  с «\_\_»\_\_\_\_\_2024 по «\_\_»\_\_\_\_\_2024 | Отметка о выполнении (подпись руководителя практики) |
| --- | --- | --- | --- |
| 1 | **Подготовительный (организационный) этап** | 06 июля 2024г  07 июля 2024г. |  |
| 1.1 | Определение базы прохождения практики | 06 июля 2024 |  |
| 1.2 | Организационное собрание для разъяснения целей, задач, содержания и порядка прохождения практики | 07 июля 2024 |  |
| 1.3 | Получение индивидуального задания |  |  |
| 2 | **Выполнение индивидуального задания:** | 08 июля 2024г  17 июля 2024г |  |
| 2.1 | Установка необходимого ПО, в частности Node.js | 09 июля 2024г |  |
| 2.2 | Настройка проекта и установка зависимостей Node.js | 10 июля 2024г |  |
| 2.3 | Разработка макета страницы (от руки) | 11 июля 2024г |  |
| 2.4 | Верстка страницы на html и css | 12 июля 2024г |  |
| 2.5 | Обработка пользовательских исключений для контактной формы, реализация файла index.js | 13 июля 2024г |  |
| 2.6 | **Поднятие сервера Node.js и обработка введенных пользователем данных с последующей отправкой в contact.json** | 14 июля 2024г |  |
| 2.6 | Тестирование проекта и отладка ошибок | 15 июля 2024г |  |
| 2.6 | Написание документационных комментариев | 16 июля 2024г |  |
| 3 | Заключительный этап | 18 июля 2024г  19 июля 2024г |  |
| 3.1 | *Написание и оформление отчета по практике* | 18 июля 2024г |  |
| 3.2 | *Защита отчета по практике* | 19 июля 2024г |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Руководитель практики от кафедры

доцент, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_ \_Филинских А.Д,\_\_\_\_\_\_\_\_\_

*(ученые звание и степень) (подпись)* Ф.И.О.

# Введение.

Целью практики в первую очередь является изучение языка JavaScript и способов его технического применения. В качестве примера был разработан двухстраничный сайт с возможностью оставить контактные данные. Информация о пользователях записывается в файл contact.json, обрабатывается программой и отображается на второй странице сайта. Доступ ко второй странице осуществляется нажатием кнопки, которая расположена в поле ввода данных пользователем. Для реализации данного проекта необходимы знания работы с HTML, CSS, Node.js и JavaScript.

# Основная часть отчета в соответствии с заданием на практику

Информация об использованных технологиях

**HTML (HyperText Markup Language)** — это основной язык разметки для создания веб-страниц. Он используется для структурирования и отображения контента в браузерах. HTML включает в себя элементы, которые обозначаются тегами, такими как: [3]

* Структурные элементы:

<html>: корневой элемент, содержащий весь документ.

<head>: содержит метаданные, такие как <title>, <meta>, <link>.

<body>: содержит видимый контент страницы.

* Элементы заголовков и абзацев:

<h1> - <h6>: заголовки разных уровней.

<p>: абзацы текста.

* Ссылки и изображения:

<a>: гиперссылки.

<img>: изображения.

* Списки:

<ul>: ненумерованные списки.

<ol>: нумерованные списки.

<li>: элементы списка.

* Таблицы:

<table>: таблицы.

<tr>: строки таблицы.

<td>: ячейки таблицы.

HTML также поддерживает семантические элементы, такие как <header>, <footer>, <article>, и <section>, которые улучшают структуру и доступность контента.

**CSS (Cascading Style Sheets)** — это язык описания внешнего вида и форматирования документов, написанных на HTML. Он позволяет отделить содержание веб-страницы от ее представления, что упрощает разработку и обслуживание сайтов.

Основные концепции CSS [4]

* Правила CSS

Селекторы: определяют, к каким HTML-элементам будут применены стили. Например, p для всех абзацев, .class для элементов с определенным классом, #id для элементов с определенным идентификатором.

Свойства и значения: определяют, какие стили будут применены. Например, color: red; изменяет цвет текста на красный.

* Каскадирование и наследование

Каскадирование: определяет, как применять несколько стилей к одному элементу, учитывая приоритет и специфичность селекторов.

Наследование: некоторые свойства могут наследоваться от родительских элементов к дочерним, например, font-family.

* Основные свойства CSS

Текст и шрифты: color, font-size, text-align.

Цвет и фон: background-color, background-image.

Блочная модель: margin, padding, border, width, height.

Расположение элементов: display, position, float, flex, grid.

* Подключение CSS к HTML

Внешние стили: <link rel="stylesheet" href="styles.css">.

Встроенные стили: <style> внутри <head>.

Инлайн-стили: атрибут style внутри HTML-элементов, например, <p style="color: red;">.

CSS делает веб-страницы более привлекательными и функциональными, улучшая пользовательский опыт и предоставляя возможности для адаптивного дизайна.

**JavaScript** — это динамический, объектно-ориентированный язык программирования, используемый для создания интерактивных и динамичных веб-страниц. Он является одной из основных технологий веб-разработки наряду с HTML и CSS. [2]

Основные характеристики JavaScript

* Интеграция с HTML и CSS

JavaScript может изменять HTML-контент и CSS-стили на лету, взаимодействуя с элементами через DOM (Document Object Model).

* Событийно-ориентированное программирование

JavaScript обрабатывает различные события, такие как клики мышью, нажатия клавиш и загрузка страницы, позволяя создавать интерактивные веб-приложения.

* Функции и объекты

JavaScript поддерживает функциональное программирование, что позволяет определять и использовать функции.

Объектно-ориентированный подход позволяет создавать объекты с методами и свойствами для моделирования реальных сущностей.

* Асинхронное программирование

JavaScript поддерживает асинхронные операции, такие как AJAX-запросы и работу с Promises, что позволяет выполнять длительные операции без блокировки интерфейса.

* Библиотеки и фреймворки

Существуют популярные библиотеки и фреймворки, такие как jQuery, React, Angular, и Vue.js, которые упрощают разработку и увеличивают производительность.

* Среда выполнения

JavaScript изначально работал только в браузерах, но благодаря платформе Node.js теперь используется и на серверной стороне для создания масштабируемых сетевых приложений.

JavaScript является мощным инструментом для создания современных веб-приложений, обеспечивая динамичность, интерактивность и улучшенное взаимодействие с пользователем.

**Node.js** — это открытая серверная платформа, основанная на движке V8 от Google, которая позволяет выполнять JavaScript-код вне браузера. Node.js используется для создания масштабируемых сетевых приложений и серверов.

Основные характеристики Node.js [1]

* Асинхронная и событийно-ориентированная модель

Node.js использует неблокирующий ввод-вывод и событийно-ориентированную модель, что позволяет обрабатывать большое количество одновременных соединений с высокой производительностью.

* Однопоточная архитектура

Несмотря на то, что Node.js работает в одном потоке, он способен эффективно обрабатывать множество задач благодаря асинхронному выполнению операций.

* Пакетный менеджер npm

Node.js включает npm (Node Package Manager), который предоставляет доступ к огромной экосистеме библиотек и модулей, упрощающих разработку и ускоряющих создание приложений.

* Широкое применение

Node.js используется для создания серверов, RESTful API, веб-приложений в реальном времени (например, чаты и онлайн-игры), микросервисов и инструментов для автоматизации разработки.

* Масштабируемость и производительность

Подходит для создания высоконагруженных систем, таких как серверы для потоковой передачи данных, поскольку позволяет эффективно обрабатывать параллельные запросы.

* Кроссплатформенность

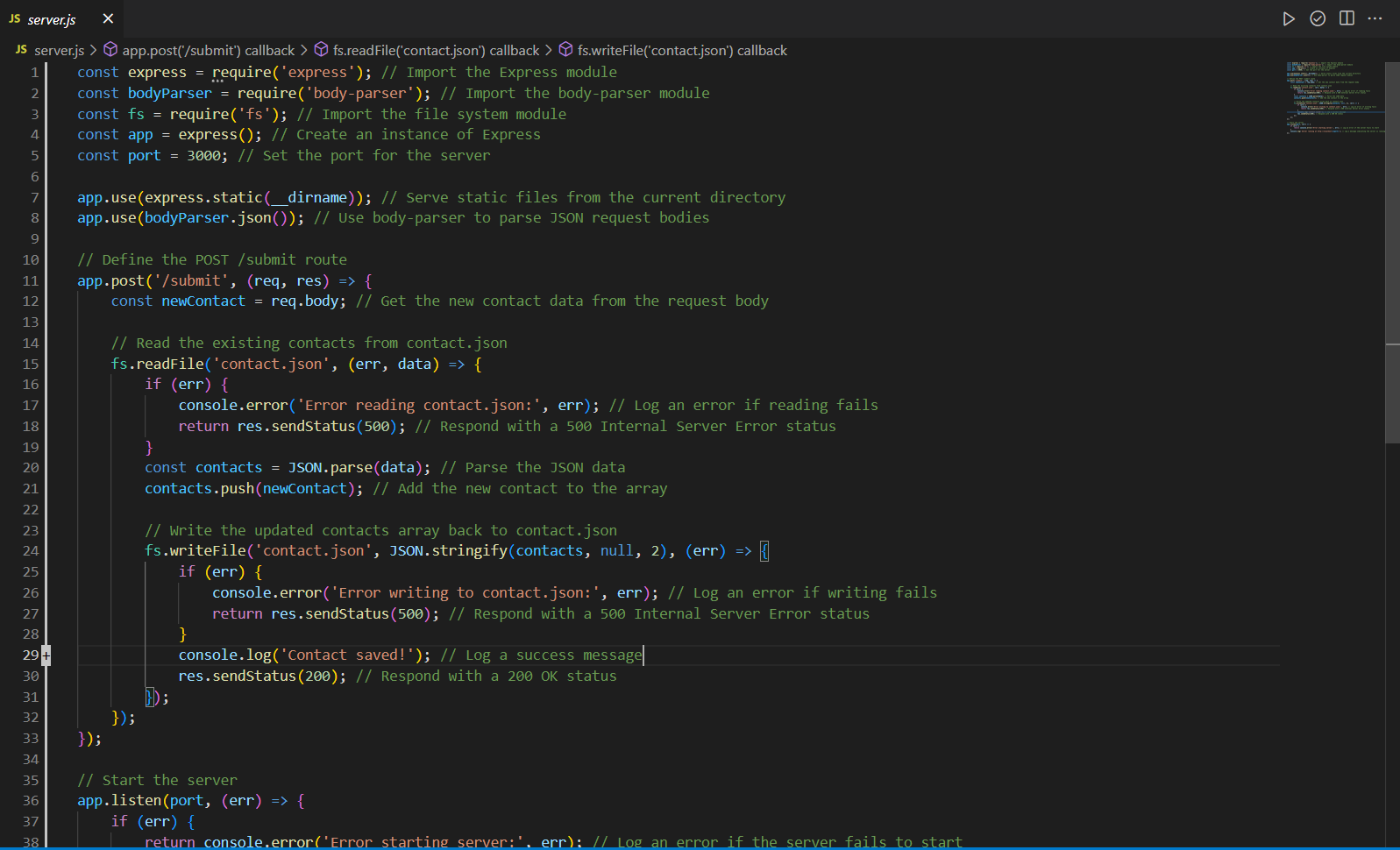
Node.js работает на различных операционных системах, включая Windows, macOS и Linux.

* Использование JavaScript

Позволяет использовать один язык программирования как на стороне клиента, так и на стороне сервера, что упрощает разработку и поддержку приложений.

Разработка

В самом начале разработки проекта необходимо настроить среду выполнения и установить все необходимые пакеты данных. После установки Node.js и настройки зависимостей в файле package.json необходимо было настроить работу локального сервера





* Импорт необходимых модулей:

const express = require('express'); // Импорт модуля Express

const bodyParser = require('body-parser'); // Импорт модуля body-parser

const fs = require('fs'); // Импорт модуля для работы с файловой системой

* Express — это фреймворк для Node.js, облегчающий создание веб-серверов.
* body-parser — middleware для обработки тела HTTP-запросов (например, JSON-данных).
* fs — модуль для работы с файловой системой (чтение и запись файлов).
* Создание экземпляра Express и установка порта:

const app = express(); // Создание экземпляра Express

const port = 3000; // Установка порта для сервера

* Настройка middleware:

app.use(express.static(\_\_dirname)); // Обслуживание статических файлов из текущего каталога

app.use(bodyParser.json()); // Использование body-parser для обработки JSON-тел запросов

* express.static(\_\_dirname) позволяет серверу обслуживать статические файлы (HTML, CSS, JavaScript) из текущего каталога.
* bodyParser.json() автоматически парсит JSON-тела запросов и делает их доступными в req.body.
* Определение маршрута POST /submit:
* app.post('/submit', (req, res) => {

const newContact = req.body; // Получение новых данных контакта из тела запроса

* Чтение существующих контактов из contact.json:

fs.readFile('contact.json', (err, data) => {

if (err) {

console.error('Error reading contact.json:', err); // Логирование ошибки при чтении

return res.sendStatus(500); // Ответ с кодом 500 Internal Server Error

}

* const contacts = JSON.parse(data); // Парсинг JSON-данных
* contacts.push(newContact); // Добавление нового контакта в массив
* Запись обновленного массива контактов обратно в contact.json:

fs.writeFile('contact.json', JSON.stringify(contacts, null, 2), (err) => {

if (err) {

console.error('Error writing to contact.json:', err); // Логирование ошибки при записи

return res.sendStatus(500); // Ответ с кодом 500 Internal Server Error

}

console.log('Contact saved!'); // Логирование успешного сохранения

res.sendStatus(200); // Ответ с кодом 200 OK

});

});

});

* Запуск сервера:

app.listen(port, (err) => {

if (err) {

return console.error('Error starting server:', err); // Логирование ошибки при запуске сервера

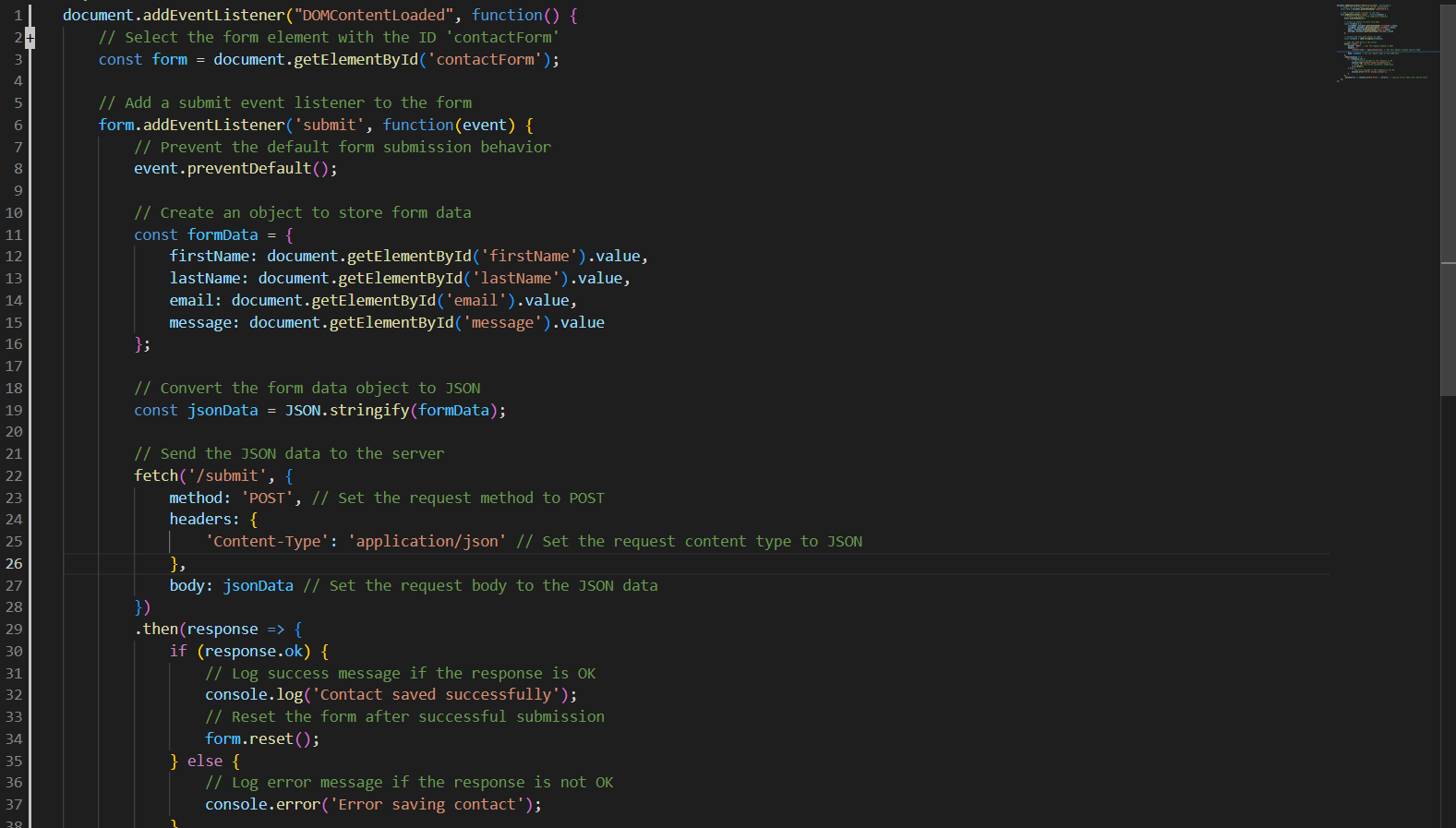
}

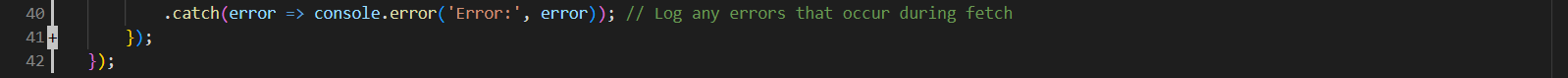
console.log(`Server running at http://localhost:${port}`); // Логирование успешного запуска сервера

});

После настройки и запуска сервера осуществляется верстка двух страниц сайта в первую очередь это файлы index.html и style.css, а также index2.html и style2.css.

После тестирования верстки на адаптивность, которая добавляется посредством использования классов с ключевым словом row, а также используя @media необходимо было разработать файлы обработки и валидации данных введенных в контактную форму.





* document.addEventListener("DOMContentLoaded", function() { ... });

Этот код добавляет обработчик события DOMContentLoaded, который выполняется, когда весь HTML-документ загружен и разобран, но внешние ресурсы (картинки, стили и т.д.) могут быть еще не загружены.

* const form = document.getElementById('contactForm');

Здесь выбирается элемент формы с ID contactForm и сохраняется в переменную form.

* form.addEventListener('submit', function(event) { ... });

Добавляется обработчик события submit для формы, который будет вызван при отправке формы.

* event.preventDefault();

Предотвращается стандартное поведение отправки формы, чтобы можно было обработать данные формы с помощью JavaScript.

* Создание объекта для данных формы:

const formData = {

firstName: document.getElementById('firstName').value,

lastName: document.getElementById('lastName').value,

email: document.getElementById('email').value,

message: document.getElementById('message').value

};

Получаются значения из полей формы (имя, фамилия, email и сообщение) и сохраняются в объекте formData.

* const jsonData = JSON.stringify(formData);

Преобразует объект formData в JSON-строку для отправки на сервер.

* fetch('/submit', { ... })

Используется функция fetch для отправки HTTP-запроса на сервер по адресу /submit.

* method: 'POST'

Устанавливается метод запроса как POST.

* headers: { 'Content-Type': 'application/json' }

Устанавливаются заголовки запроса, чтобы указать, что данные в запросе находятся в формате JSON.

* body: jsonData

Устанавливается тело запроса, содержащее JSON-строку с данными формы.

* response.then(response => { ... })

Обрабатывается ответ от сервера. Если ответ успешен (статус OK), выполняется соответствующий блок кода.

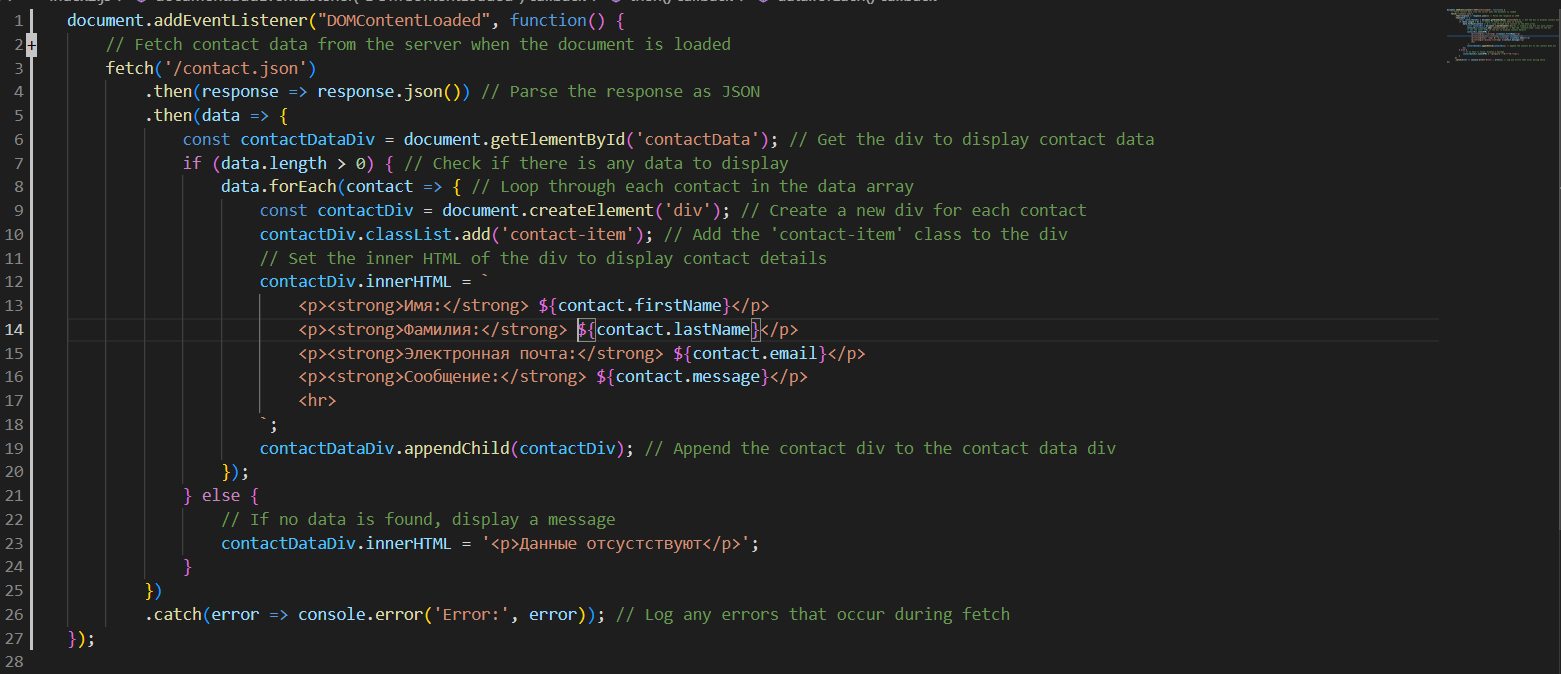
* if (response.ok) { ... } else { ... }

Проверяется статус ответа. Если ответ успешен (response.ok), логируется сообщение об успешном сохранении и форма сбрасывается (form.reset()). В противном случае логируется ошибка.

* catch(error => console.error('Error:', error));

Логируется любая ошибка, которая может произойти во время выполнения fetch.

После разработки и тестирования файла index.js, который предназначен для обработки данных на 1 странице, необходимо было перейти к разработке файла index2.js, который необходим для отображения данных на странице и чтения файла contact.json.



* Добавление обработчика события для загрузки DOM:

document.addEventListener("DOMContentLoaded", function() { ... });

Этот код выполняет функцию, когда весь HTML-документ загружен и разобран (но не обязательно все внешние ресурсы, такие как изображения и стили).

* Загрузка данных контактов с сервера:

fetch('/contact.json')

.then(response => response.json()) // Парсинг ответа как JSON

.then(data => { ... })

.catch(error => console.error('Error:', error)); // Логирование ошибок

* + fetch('/contact.json') отправляет HTTP-запрос на сервер для получения файла contact.json.
  + response.json() преобразует ответ от сервера в объект JavaScript.
  + Если запрос завершился успешно, данные передаются в следующую цепочку .then(data => { ... }).
  + Если произошла ошибка во время выполнения запроса, она логируется с помощью .catch(error => console.error('Error:', error)).
* Обработка данных контактов:

.then(data => {

const contactDataDiv = document.getElementById('contactData'); // Получение элемента div для отображения данных контактов

Получение элемента с ID contactData, в котором будут отображаться данные контактов.

* Проверка наличия данных:

if (data.length > 0) { ... } else { ... }

Проверка, есть ли данные в массиве data. Если данные присутствуют (data.length > 0), выполняется блок кода внутри if.

* Цикл для отображения каждого контакта:

data.forEach(contact => {

const contactDiv = document.createElement('div'); // Создание нового div для каждого контакта

contactDiv.classList.add('contact-item'); // Добавление класса 'contact-item' к div

Для каждого контакта в массиве data создается новый элемент div.

Добавление класса contact-item к этому div для стилизации.

* Установка внутреннего HTML для отображения деталей контакта:

contactDiv.innerHTML = `

<p><strong>Имя:</strong> ${contact.firstName}</p>

<p><strong>Фамилия:</strong> ${contact.lastName}</p>

<p><strong>Электронная почта:</strong> ${contact.email}</p>

<p><strong>Сообщение:</strong> ${contact.message}</p>

<hr>

`;

contactDataDiv.appendChild(contactDiv); // Добавление div контакта в основной div

Внутренний HTML нового div устанавливается так, чтобы отображать имя, фамилию, электронную почту и сообщение контакта.

После этого новый div добавляется в contactDataDiv, чтобы он отображался на странице.

* Обработка случая, когда данных нет:

} else {

contactDataDiv.innerHTML = '<p>Данные отсутствуют</p>';

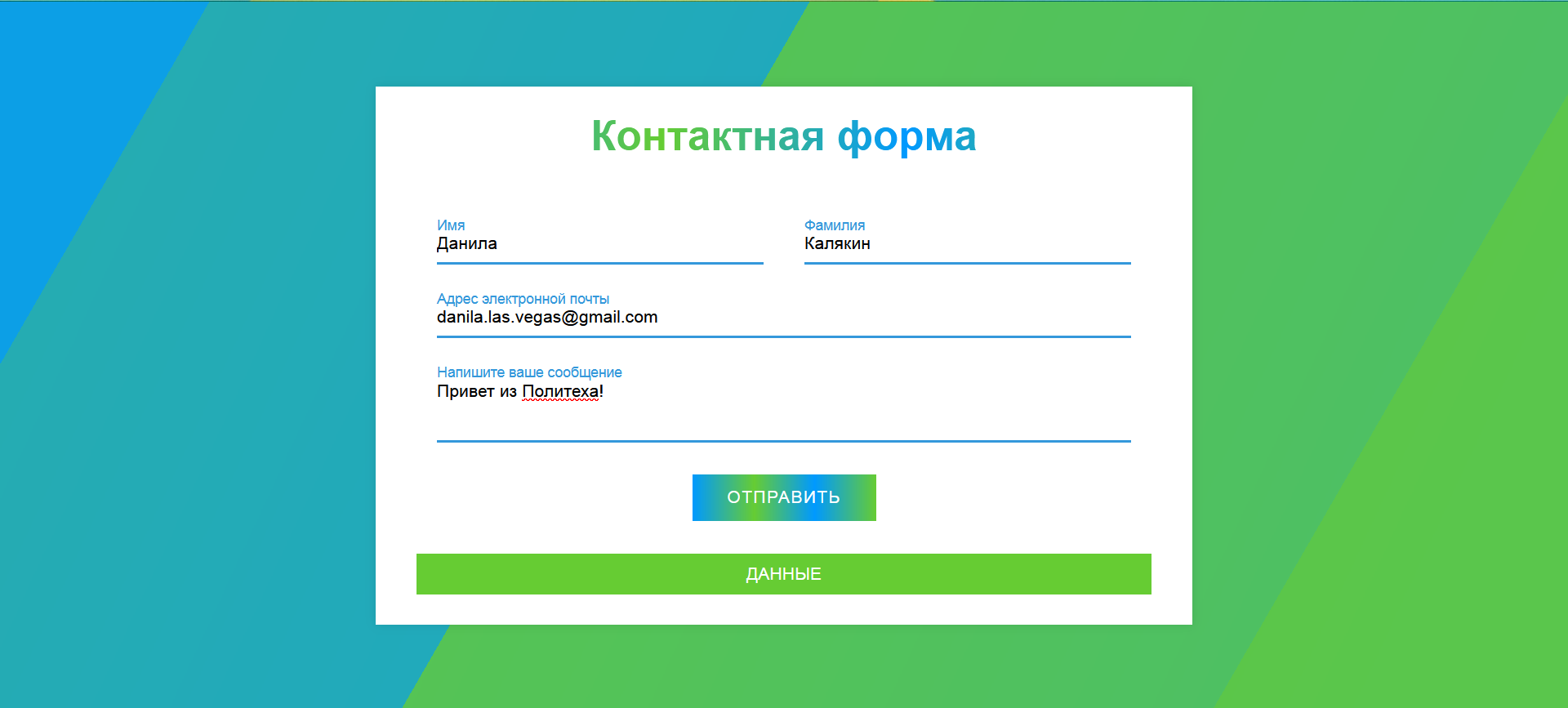
}

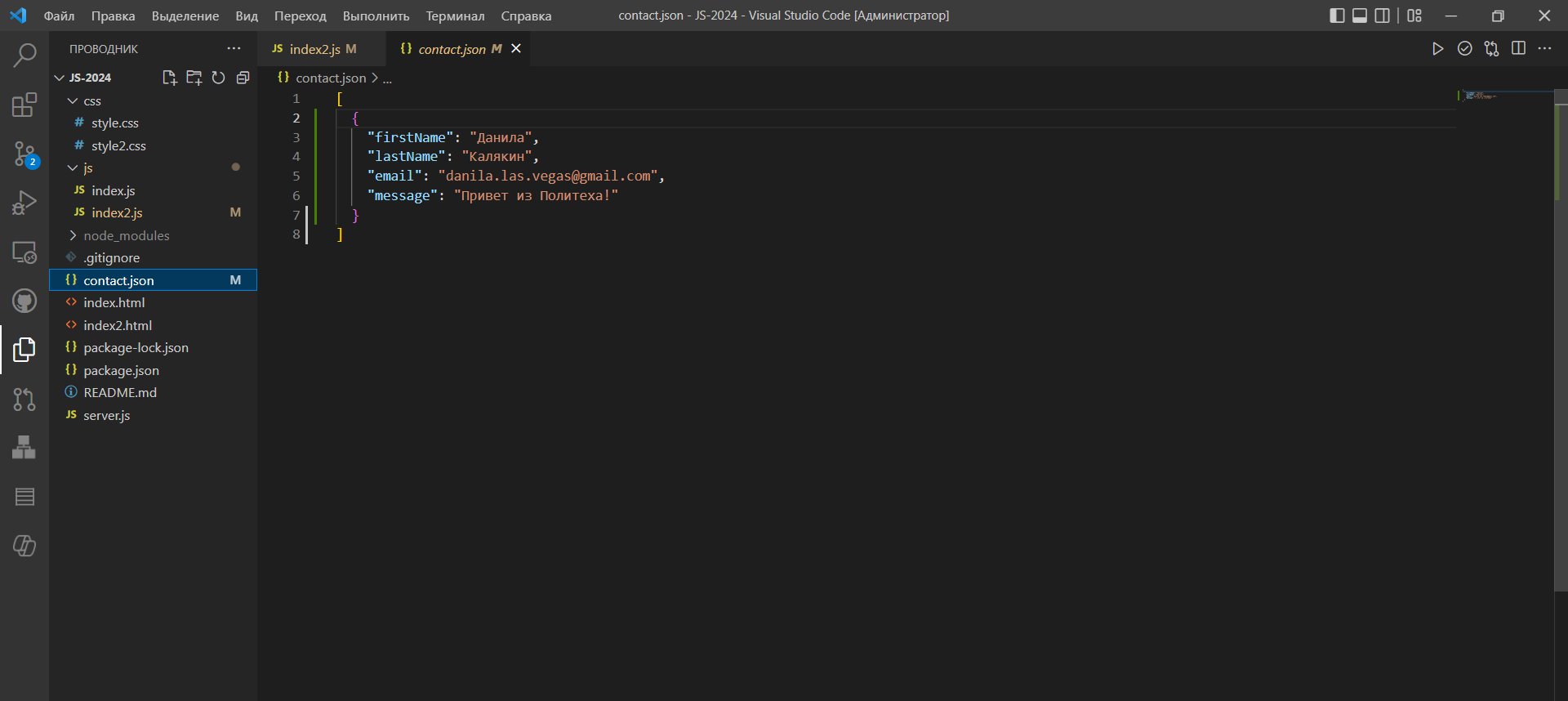
Если массив данных пустой, в contactDataDiv добавляется сообщение "No data to display".

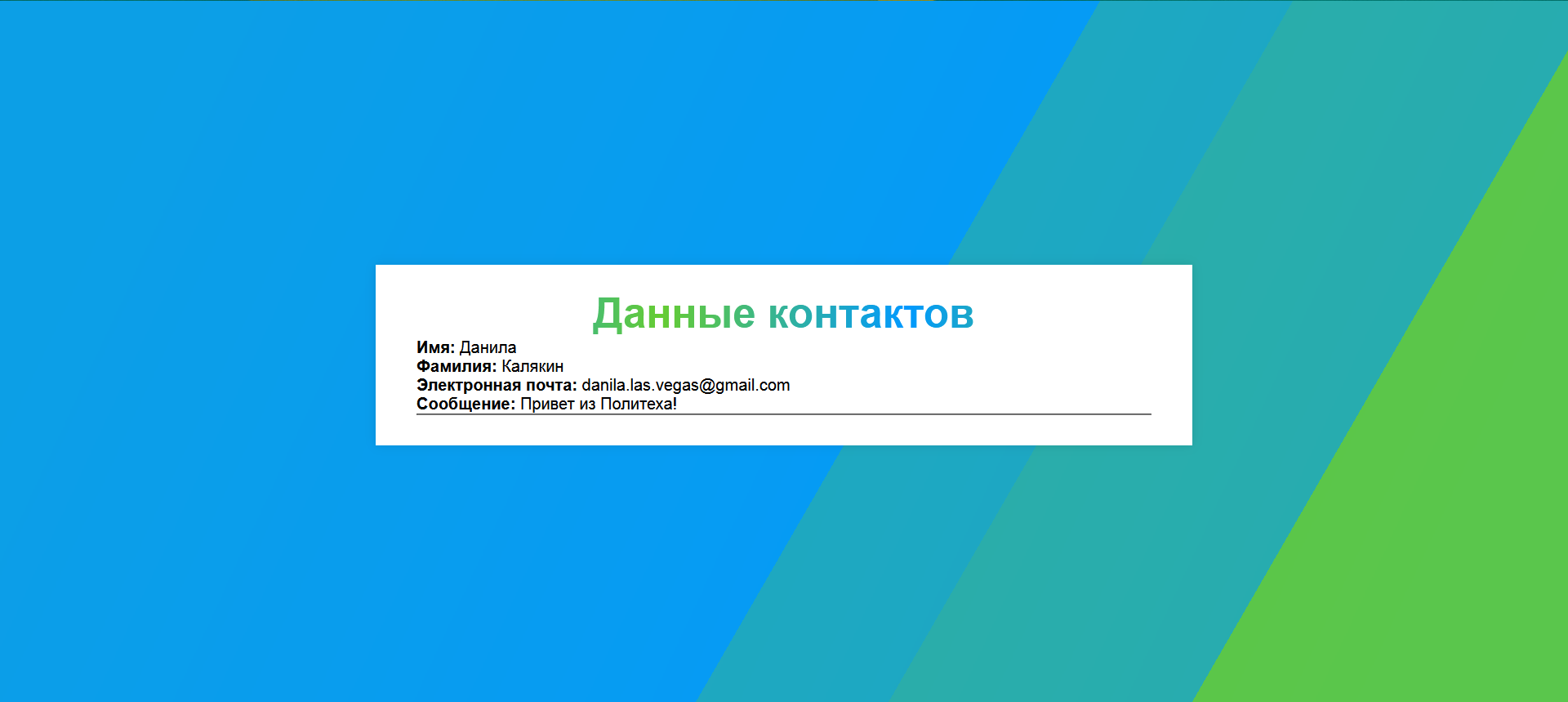
После разработки данного проекта вводятся документационные комментарии, после чего проект отправляется с помощью системы контроля версий на удаленный репозиторий.

# Заключение

Таким образом, было проведена работа с Node.js, HTML, CSS, JS. Была изучена необходимая документация для разработки данного проекта. Также проведена работа с настройкой и отправкой проекта на удаленный репозиторий с помощью системы контроля версий. Проведены успешные тесты работы сайта, проанализирован пользовательский опыт.







# Список литературы

|  |  |
| --- | --- |
| [1] | "Документация Node.js" https://nodejs.org/api/all.html |
| [2] | "Документация JavaScript" https://devdocs.io/javascript/ |
| [3] | "Документация HTML" https://developer.mozilla.org/ru/docs/Web/HTML |
| [4] | "Документация CSS" https://devdocs.io/css/ |

# 

# Приложение

index.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="css/style.css">

<title>Калякин Д.С</title>

</head>

<body>

<script src="js/index.js"></script>

<div class="bg"></div>

<div class="bg bg2"></div>

<div class="bg bg3"></div>

<div class="container">

<div class="text">

Контактная форма

</div>

<form id="contactForm" action="#">

<div class="form-row">

<div class="input-data">

<input type="text" id="firstName" name="firstName" required>

<div class="underline"></div>

<label for="firstName">Имя</label>

</div>

<div class="input-data">

<input type="text" id="lastName" name="lastName" required>

<div class="underline"></div>

<label for="lastName">Фамилия</label>

</div>

</div>

<div class="form-row">

<div class="input-data">

<input type="email" id="email" name="email" required>

<div class="underline"></div>

<label for="email">Адрес электронной почты</label>

</div>

</div>

<div class="form-row">

<div class="input-data textarea">

<textarea id="message" name="message" rows="8" cols="80" required></textarea>

<br />

<div class="underline"></div>

<label for="message">Напишите ваше сообщение</label>

<br />

</div>

</div>

<div class="form-row submit-btn">

<div class="input-data">

<div class="inner"></div>

<input type="submit" value="Отправить">

</div>

</div>

</form>

<div class="form-row">

<div class="input-data">

<button id="dataButton" class="data-btn">Данные</button>

</div>

</div>

</div>

<script>

document.getElementById('dataButton').addEventListener('click', function() {

window.location.href = 'index2.html';

});

</script>

</body>

</html>

Index2.html

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="css/style2.css">

<title>Данные контактов</title>

</head>

<body>

<script src="js/index2.js"></script>

<div class="bg"></div>

<div class="bg bg2"></div>

<div class="bg bg3"></div>

<div class="container">

<div class="text">

Данные контактов

</div>

<div id="contactData"></div>

</div>

</body>

</html>

Index.js

document.addEventListener("DOMContentLoaded", function() {

// Select the form element with the ID 'contactForm'

const form = document.getElementById('contactForm');

// Add a submit event listener to the form

form.addEventListener('submit', function(event) {

// Prevent the default form submission behavior

event.preventDefault();

// Create an object to store form data

const formData = {

firstName: document.getElementById('firstName').value,

lastName: document.getElementById('lastName').value,

email: document.getElementById('email').value,

message: document.getElementById('message').value

};

// Convert the form data object to JSON

const jsonData = JSON.stringify(formData);

// Send the JSON data to the server

fetch('/submit', {

method: 'POST', // Set the request method to POST

headers: {

'Content-Type': 'application/json' // Set the request content type to JSON

},

body: jsonData // Set the request body to the JSON data

})

.then(response => {

if (response.ok) {

// Log success message if the response is OK

console.log('Contact saved successfully');

// Reset the form after successful submission

form.reset();

} else {

// Log error message if the response is not OK

console.error('Error saving contact');

}

})

.catch(error => console.error('Error:', error)); // Log any errors that occur during fetch

});

});

Index2.js

document.addEventListener("DOMContentLoaded", function() {

// Fetch contact data from the server when the document is loaded

fetch('/contact.json')

.then(response => response.json()) // Parse the response as JSON

.then(data => {

const contactDataDiv = document.getElementById('contactData'); // Get the div to display contact data

if (data.length > 0) { // Check if there is any data to display

data.forEach(contact => { // Loop through each contact in the data array

const contactDiv = document.createElement('div'); // Create a new div for each contact

contactDiv.classList.add('contact-item'); // Add the 'contact-item' class to the div

// Set the inner HTML of the div to display contact details

contactDiv.innerHTML = `

<p><strong>Имя:</strong> ${contact.firstName}</p>

<p><strong>Фамилия:</strong> ${contact.lastName}</p>

<p><strong>Электронная почта:</strong> ${contact.email}</p>

<p><strong>Сообщение:</strong> ${contact.message}</p>

<hr>

`;

contactDataDiv.appendChild(contactDiv); // Append the contact div to the contact data div

});

} else {

// If no data is found, display a message

contactDataDiv.innerHTML = '<p>Данные отсустствуют</p>';

}

})

.catch(error => console.error('Error:', error)); // Log any errors that occur during fetch

});

Style.css

\* {

margin: 0;

padding: 0;

outline: none;

box-sizing: border-box;

font-family: 'Poppins', sans-serif;

}

/\* Background animation \*/

.bg {

animation: slide 10s ease-in-out infinite alternate;

background-image: linear-gradient(-60deg, #6c3 50%, #09f 50%);

bottom: 0;

left: -50%;

opacity: .5;

position: fixed;

right: -50%;

top: 0;

z-index: -1;

}

/\* Alternate animation direction and duration for background \*/

.bg2 {

animation-direction: alternate-reverse;

animation-duration: 4s;

}

.bg3 {

animation-duration: 5s;

}

h1 {

font-family: monospace;

}

/\* Keyframes for background animation \*/

@keyframes slide {

0% {

transform: translateX(-25%);

}

100% {

transform: translateX(25%);

}

}

/\* Import Google Fonts \*/

@import url('https://fonts.googleapis.com/css?family=Poppins:400,500,600,700&display=swap');

/\* Body styling \*/

body {

display: flex;

align-items: center;

justify-content: center;

min-height: 100vh;

padding: 10px;

font-family: 'Poppins', sans-serif;

background: linear-gradient(115deg, #6c3 10%, #09f 90%);

}

/\* Container styling \*/

.container {

max-width: 800px;

background: #fff;

width: 800px;

padding: 25px 40px 10px 40px;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

}

/\* Text styling inside the container \*/

.container .text {

text-align: center;

font-size: 41px;

font-weight: 600;

font-family: 'Poppins', sans-serif;

background: -webkit-linear-gradient(right, #6c3, #09f, #6c3, #09f);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

}

/\* Form styling \*/

.container form {

padding: 30px 0 0 0;

}

.container form .form-row {

display: flex;

margin: 32px 0;

}

/\* Input fields styling \*/

form .form-row .input-data {

width: 100%;

height: 40px;

margin: 0 20px;

position: relative;

}

form .form-row .textarea {

height: 70px;

}

/\* Text input and textarea styling \*/

.input-data input,

.textarea textarea {

display: block;

width: 100%;

height: 100%;

border: none;

font-size: 17px;

border-bottom: 2px solid rgba(0, 0, 0, 0.12);

}

/\* Styling for the "Data" button \*/

.data-btn {

background-color: #6c3;

color: #fff;

border: none;

padding: 10px 20px;

font-size: 17px;

font-weight: 500;

text-transform: uppercase;

cursor: pointer;

margin: 20px 0;

display: block;

width: 100%;

text-align: center;

}

.data-btn:hover {

background-color: #56a320;

}

/\* Label transformations for focus and valid states \*/

.input-data input:focus ~ label, .textarea textarea:focus ~ label,

.input-data input:valid ~ label, .textarea textarea:valid ~ label {

transform: translateY(-20px);

font-size: 14px;

color: #3498db;

}

/\* Textarea styling \*/

.textarea textarea {

resize: none;

padding-top: 10px;

}

/\* Label styling \*/

.input-data label {

position: absolute;

pointer-events: none;

bottom: 10px;

font-size: 16px;

transition: all 0.3s ease;

}

.textarea label {

width: 100%;

bottom: 40px;

background: #fff;

}

/\* Underline styling \*/

.input-data .underline {

position: absolute;

bottom: 0;

height: 2px;

width: 100%;

}

.input-data .underline:before {

position: absolute;

content: "";

height: 2px;

width: 100%;

background: #3498db;

transform: scaleX(0);

transform-origin: center;

transition: transform 0.3s ease;

}

.input-data input:focus ~ .underline:before,

.input-data input:valid ~ .underline:before,

.textarea textarea:focus ~ .underline:before,

.textarea textarea:valid ~ .underline:before {

transform: scale(1);

}

/\* Submit button styling \*/

.submit-btn {

display: flex;

justify-content: center; /\* Center alignment \*/

}

.submit-btn .input-data {

overflow: hidden;

height: 45px !important;

width: 25% !important;

position: relative;

}

/\* Inner gradient animation for submit button \*/

.submit-btn .input-data .inner {

height: 100%;

width: 100%;

background: -webkit-linear-gradient(right, #6c3, #09f, #6c3, #09f);

position: absolute;

top: 0;

left: 0;

transition: all 0.4s;

}

.submit-btn .input-data:hover .inner {

left: 0;

}

/\* Submit button text styling \*/

.submit-btn .input-data input {

background: none;

border: none;

color: #fff;

font-size: 17px;

font-weight: 500;

text-transform: uppercase;

letter-spacing: 1px;

cursor: pointer;

position: relative;

z-index: 2;

}

/\* Media query for responsiveness \*/

@media (max-width: 700px) {

.container .text {

font-size: 30px;

}

.container form {

padding: 10px 0 0 0;

}

.container form .form-row {

display: block;

}

form .form-row .input-data {

margin: 35px 0 !important;

}

.submit-btn .input-data {

width: 40% !important;

}

}

Style2.css

\* {

margin: 0;

padding: 0;

outline: none;

box-sizing: border-box;

font-family: 'Poppins', sans-serif;

}

/\* Background animation \*/

.bg {

animation: slide 10s ease-in-out infinite alternate;

background-image: linear-gradient(-60deg, #6c3 50%, #09f 50%);

bottom: 0;

left: -50%;

opacity: .5;

position: fixed;

right: -50%;

top: 0;

z-index: -1;

}

/\* Alternate animation direction and duration for background \*/

.bg2 {

animation-direction: alternate-reverse;

animation-duration: 4s;

}

.bg3 {

animation-duration: 5s;

}

/\* Keyframes for background animation \*/

@keyframes slide {

0% {

transform: translateX(-25%);

}

100% {

transform: translateX(25%);

}

}

/\* Body styling \*/

body {

display: flex;

align-items: center;

justify-content: center;

min-height: 100vh;

padding: 10px;

font-family: 'Poppins', sans-serif;

background: linear-gradient(115deg, #6c3 10%, #09f 90%);

}

/\* Container styling \*/

.container {

max-width: 800px;

background: #fff;

width: 800px;

padding: 25px 40px 10px 40px;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

}

/\* Text styling inside the container \*/

.container .text {

text-align: center;

font-size: 41px;

font-weight: 600;

font-family: 'Poppins', sans-serif;

background: -webkit-linear-gradient(right, #6c3, #09f, #6c3, #09f);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

}

/\* Form styling \*/

.container form {

padding: 30px 0 0 0;

}

/\* Form row styling \*/

.container form .form-row {

display: flex;

margin: 32px 0;

}

/\* Input fields styling \*/

form .form-row .input-data {

width: 100%;

height: 40px;

margin: 0 20px;

position: relative;

}

/\* Textarea styling \*/

form .form-row .textarea {

height: 70px;

}

/\* Text input and textarea styling \*/

.input-data input,

.textarea textarea {

display: block;

width: 100%;

height: 100%;

border: none;

font-size: 17px;

border-bottom: 2px solid rgba(0, 0, 0, 0.12);

}

/\* Label transformations for focus and valid states \*/

.input-data input:focus ~ label, .textarea textarea:focus ~ label,

.input-data input:valid ~ label, .textarea textarea:valid ~ label {

transform: translateY(-20px);

font-size: 14px;

color: #3498db;

}

/\* Textarea styling \*/

.textarea textarea {

resize: none;

padding-top: 10px;

}

/\* Label styling \*/

.input-data label {

position: absolute;

pointer-events: none;

bottom: 10px;

font-size: 16px;

transition: all 0.3s ease;

}

/\* Textarea label styling \*/

.textarea label {

width: 100%;

bottom: 40px;

background: #fff;

}

/\* Underline styling \*/

.input-data .underline {

position: absolute;

bottom: 0;

height: 2px;

width: 100%;

}

/\* Underline before element styling \*/

.input-data .underline:before {

position: absolute;

content: "";

height: 2px;

width: 100%;

background: #3498db;

transform: scaleX(0);

transform-origin: center;

transition: transform 0.3s ease;

}

/\* Underline transform on focus and valid states \*/

.input-data input:focus ~ .underline:before,

.input-data input:valid ~ .underline:before,

.textarea textarea:focus ~ .underline:before,

.textarea textarea:valid ~ .underline:before {

transform: scale(1);

}

/\* Submit button container styling \*/

.submit-btn {

display: flex;

justify-content: center; /\* Center alignment \*/

}

/\* Submit button inner container styling \*/

.submit-btn .input-data {

overflow: hidden;

height: 45px !important;

width: 25% !important;

position: relative;

}

/\* Inner gradient animation for submit button \*/

.submit-btn .input-data .inner {

height: 100%;

width: 100%;

background: -webkit-linear-gradient(right, #6c3, #09f, #6c3, #09f);

position: absolute;

top: 0;

left: 0;

transition: all 0.4s;

}

/\* Inner element transition on hover \*/

.submit-btn .input-data:hover .inner {

left: 0;

}

/\* Submit button text styling \*/

.submit-btn .input-data input {

background: none;

border: none;

color: #fff;

font-size: 17px;

font-weight: 500;

text-transform: uppercase;

letter-spacing: 1px;

cursor: pointer;

position: relative;

z-index: 2;

}

/\* Styling for the "Data" button \*/

.data-btn {

background-color: #6c3;

color: #fff;

border: none;

padding: 10px 20px;

font-size: 17px;

font-weight: 500;

text-transform: uppercase;

cursor: pointer;

margin: 20px 0;

display: block;

width: 100%;

text-align: center;

}

/\* Hover effect for "Data" button \*/

.data-btn:hover {

background-color: #56a320;

}

/\* Contact item margin \*/

.contact-item {

margin-bottom: 20px;

}

/\* Media query for responsiveness \*/

@media (max-width: 700px) {

.container .text {

font-size: 30px;

}

.container form {

padding: 10px 0 0 0;

}

.container form .form-row {

display: block;

}

form .form-row .input-data {

margin: 35px 0 !important;

}

.submit-btn .input-data {

width: 40% !important;

}

.data-btn {

width: auto;

padding: 10px;

}

}

Package.json

{

"name": "my\_project",

"version": "1.0.0",

"description": "",

"main": "server.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node server.js"

},

"repository": {

"type": "git",

"url": "git+https://github.com/ARISTOKRAT-TP/JS-2024.git"

},

"author": "DanilKalyakin",

"license": "ISC",

"bugs": {

"url": "https://github.com/ARISTOKRAT-TP/JS-2024/issues"

},

"homepage": "https://github.com/ARISTOKRAT-TP/JS-2024#readme",

"dependencies": {

"express": "^4.17.1",

"body-parser": "^1.19.0"

}

}

Server.js

const express = require('express'); // Import the Express module

const bodyParser = require('body-parser'); // Import the body-parser module

const fs = require('fs'); // Import the file system module

const app = express(); // Create an instance of Express

const port = 3000; // Set the port for the server

app.use(express.static(\_\_dirname)); // Serve static files from the current directory

app.use(bodyParser.json()); // Use body-parser to parse JSON request bodies

// Define the POST /submit route

app.post('/submit', (req, res) => {

const newContact = req.body; // Get the new contact data from the request body

// Read the existing contacts from contact.json

fs.readFile('contact.json', (err, data) => {

if (err) {

console.error('Error reading contact.json:', err); // Log an error if reading fails

return res.sendStatus(500); // Respond with a 500 Internal Server Error status

}

const contacts = JSON.parse(data); // Parse the JSON data

contacts.push(newContact); // Add the new contact to the array

// Write the updated contacts array back to contact.json

fs.writeFile('contact.json', JSON.stringify(contacts, null, 2), (err) => {

if (err) {

console.error('Error writing to contact.json:', err); // Log an error if writing fails

return res.sendStatus(500); // Respond with a 500 Internal Server Error status

}

console.log('Contact saved!'); // Log a success message

res.sendStatus(200); // Respond with a 200 OK status

});

});

});

// Start the server

app.listen(port, (err) => {

if (err) {

return console.error('Error starting server:', err); // Log an error if the server fails to start

}

console.log(`Server running at http://localhost:${port}`); // Log a message indicating the server is running

});