Создаем React App для генерации изображений ИИ DALL-E API OpenAI на NodeJs

Описание

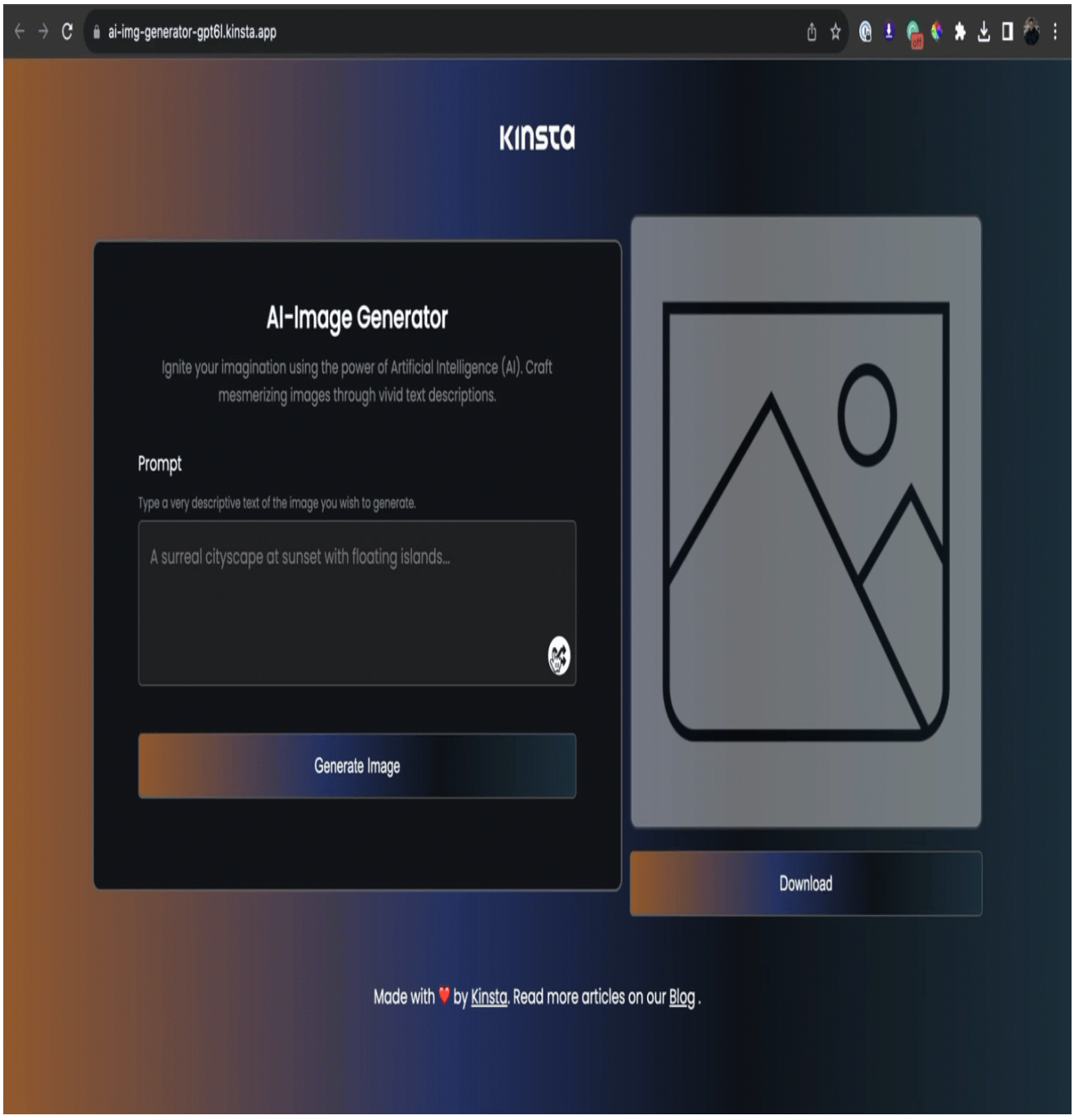
В современном мире, где инновации в области технологий изменяют наше восприятие реальности, искусственный интеллект (ИИ) играет ключевую роль, открывая новые горизонты в области творчества и программирования. Искусственный интеллект уже давно не ограничивается традиционными задачами: он учится, рассуждает, решает сложные проблемы и даже расширяет границы искусства, предлагая невероятные возможности для генерации изображений.

Что мы построим

В этой статье мы сосредоточимся на одном из самых захватывающих применений ИИ — создании изображений. Мы шаг за шагом покажем, как можно создать приложение на React, которое будет взаимодействовать с API OpenAI DALL-E с помощью сервера на Node.js. Это руководство предназначено для тех, кто хочет погрузиться в мир ИИ и узнать, как использовать его возможности для создания уникальных, интригующих визуальных произведений.

Наши инструкции помогут вам разработать приложение, которое не только демонстрирует потенциал ИИ в генерации изображений, но и предоставляет инструменты для их создания на основе ваших текстовых подсказок.

Приготовьтесь окунуться в удивительный мир ИИ, где ваши творческие идеи превращаются в визуальные шедевры



Генератор изображений AI в действии, создающий яркие и креативные изображения с использованием DALL-E API.

Необходимые условия

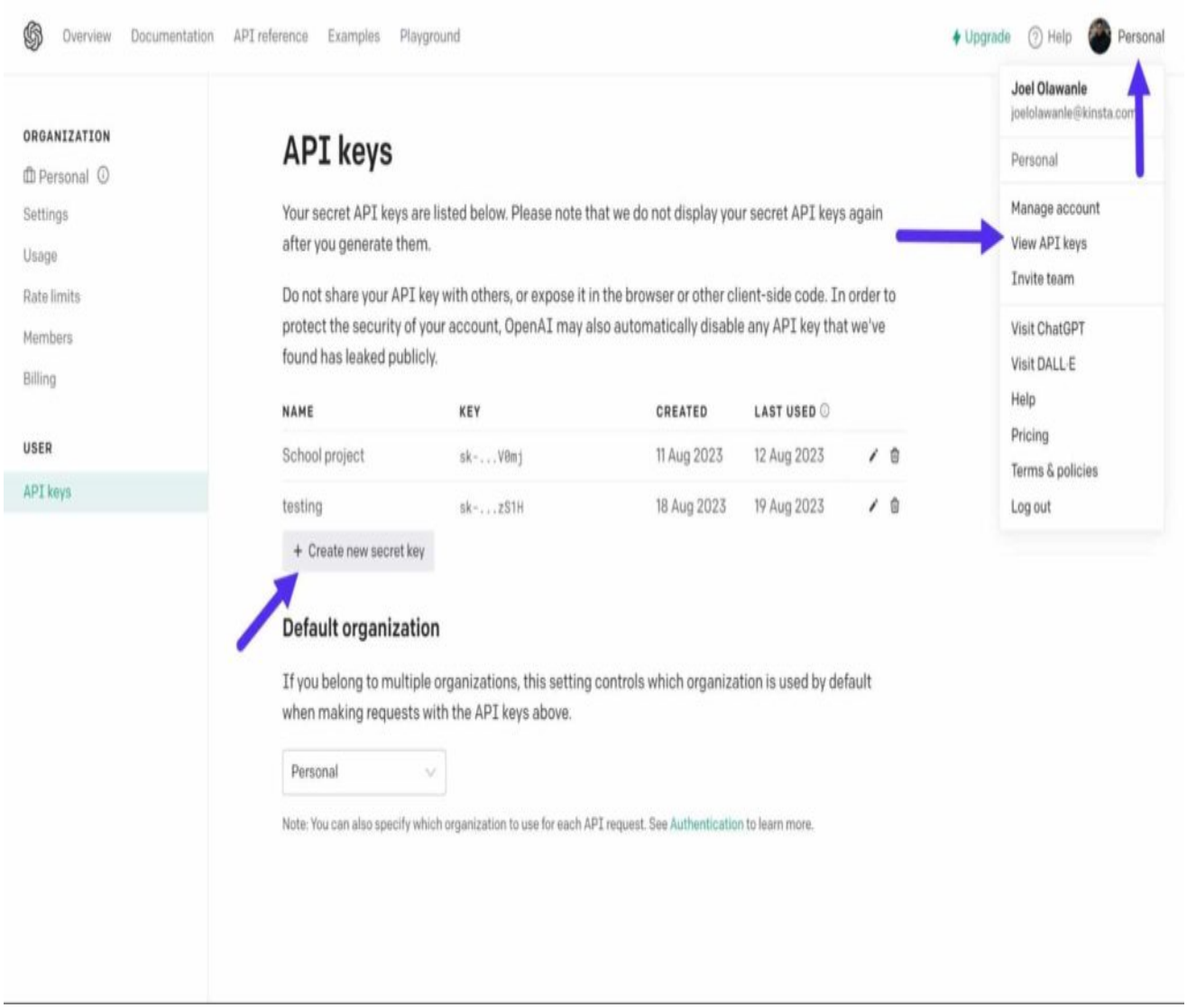
Для работы над этим проектом необходимо иметь:

- Фундаментальное понимание HTML, CSS и JavaScript
- Базовые знания React и Node.js
- На компьютере должны быть установлены Node.js и npm(Node Package Manager) или yarn

Что такое API OpenAI DALL-E?

OpenAI API — это облачная платформа, предоставляющая разработчикам доступ к предварительно обученным моделям искусственного интеллекта OpenAI, таким как DALL-E и GPT-3. Это позволяет разработчикам добавлять в свои программы такие функции ИИ, как обобщение, перевод, генерация изображений и модификация, без разработки и обучения своих моделей.

Чтобы воспользоваться OpenAI API, создайте учетную запись с помощью аккаунта Google или электронной почты на сайте OpenAI и получите API-ключ. Чтобы сгенерировать API-ключ, нажмите кнопку **Personal** в правом верхнем углу сайта и выберите пункт **View API keys**.



Процесс создания секретного ключа OpenAI API.

Нажмите **кнопку «Создать новый секретный ключ»** и сохраните ключ где-нибудь. Вы будете использовать его в этом приложении для взаимодействия с API DALL-E OpenAI.

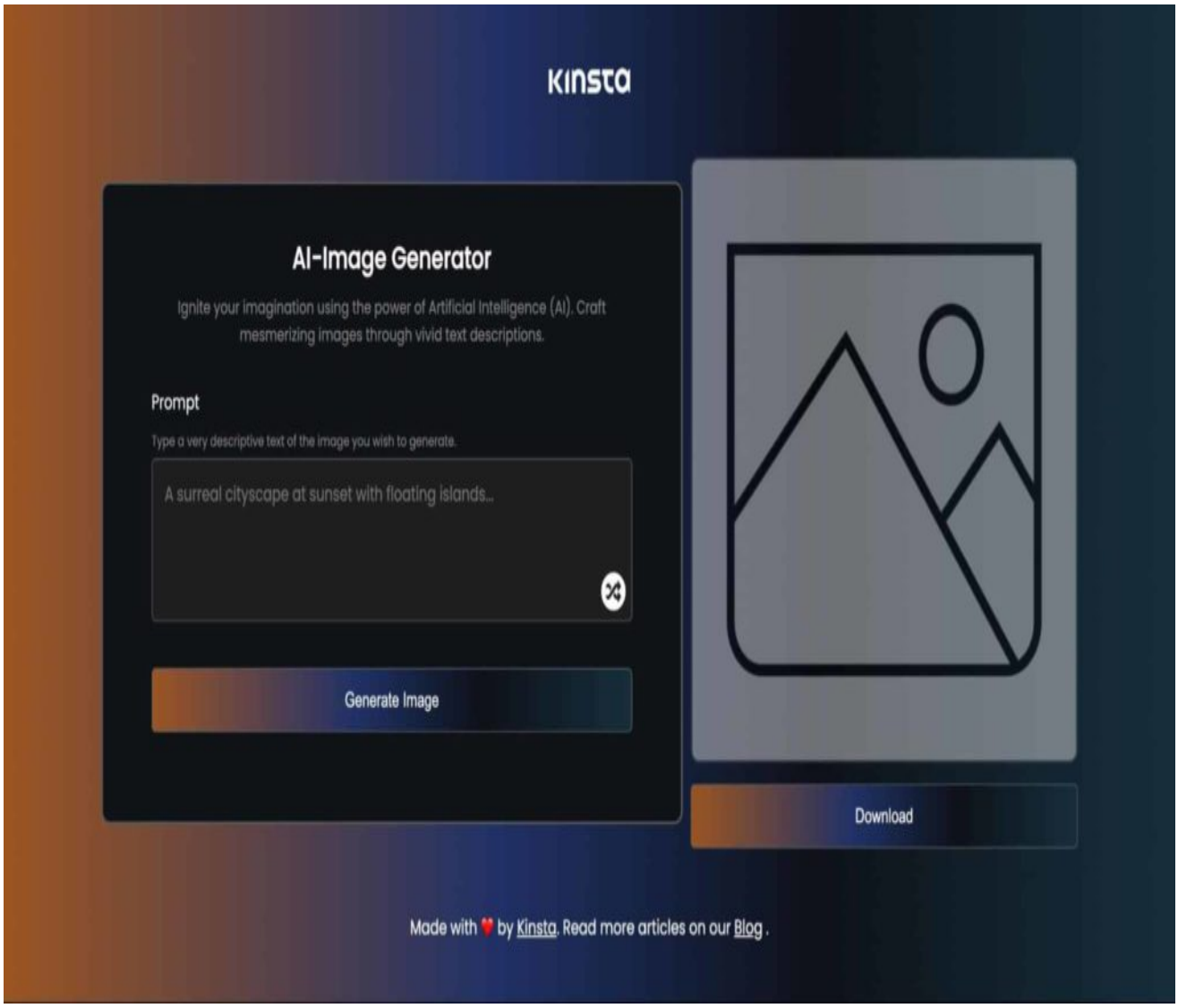
Настройка среды разработки

Готовый проект с GitHub

Вы можете создать React-приложение с нуля и разработать свой собственный интерфейс, а можете воспользоваться нашим стартовым шаблоном на Git, выполнив следующие действия:

1. Для этого проекта посетите **[репозиторий на GitHub](#)**.
2. Выберите **Use this template** > **Create a new repository**, чтобы скопировать стартовый код в репозиторий в своей учетной записи GitHub (установите флажок, чтобы **включить все ветки**).
3. Перетащите репозиторий на локальный компьютер и переключитесь на ветку starter-files с помощью команды: `git checkout starter-files`.
4. Установите необходимые зависимости, выполнив команду `npm install`.

После завершения установки можно запустить проект на локальном компьютере командой `npm run start`. В результате проект станет доступен по адресу http://localhost:3000/.



Пользовательский интерфейс приложения-генератора изображений с

искусственным интеллектом, демонстрирующий возможности искусственного интеллекта в создании изображений.

Разбираем файлы проекта

В этом проекте мы добавили все необходимые зависимости для вашего React-приложения. Вот обзор того, что было установлено:

Зависимости проекта

- **file-server:** Эта библиотека упрощает процесс загрузки сгенерированных изображений. Она связана с кнопкой загрузки, что обеспечивает плавное взаимодействие с пользователем.
- **uuid:** Эта библиотека присваивает уникальную идентификацию каждому изображению. Это предотвращает возможность использования изображений с одинаковыми именами файлов по умолчанию, сохраняя порядок и четкость.
- **react-icons:** Интегрированная в проект, эта библиотека позволяет легко внедрять иконки, повышая визуальную привлекательность вашего приложения.

Основные папки и файлы

В основе приложения React лежит папка **src**. Именно в ней находится основной JavaScript-код для Webpack. Давайте разберемся, какие файлы и папки находятся в папке **src**:

- **assets:** В этом каталоге находятся изображения и загрузчик gif, используемые в проекте.
- **data:** В этой папке находится файл **index.js**, экспортирующий массив из 30 подсказок. Эти подсказки могут быть использованы для генерации разнообразных и случайных изображений. Не стесняйтесь редактировать его.
- **index.css:** Здесь хранятся стили, используемые в данном проекте.

Разбор папки Utils

Внутри этой папки в файле **index.js** определены две функции многократного использования. Первая функция рандомизирует выбор подсказок, описывающих различные изображения, которые могут быть сгенерированы.

```
import { randomPrompts } ?? '../data';
```

```
export const getRandomPrompt = () => {
    const randomIndex = Math.floor(Math.random() * randomPrompts.length);
    const randomPrompt = randomPrompts[randomIndex];

    return randomPrompt;
}
```

Вторая функция обрабатывает загрузку сгенерированных изображений, используя зависимость **file-saver**. Обе функции созданы для обеспечения модульности и эффективности, и при необходимости их можно удобно импортировать в компоненты.

```
import FileSaver from 'file-saver';
import { v4 as uuidv4 } from 'uuid';

export async function downloadImage(photo) {
    const _id = uuidv4();
    FileSaver.saveAs(photo, `download-${_id}.jpg`);
}
```

В приведенном выше коде зависимость **uuid** присваивает каждому сгенерированному файлу изображения уникальный идентификатор, поэтому они не могут иметь одинаковое имя файла.

Компоненты приложения

Компоненты — это небольшие блоки кода, разделенные для облегчения сопровождения и понимания кода. Для данного проекта было создано три компонента: **Header.jsx**, **Footer.jsx** и **Form.jsx**. Основным компонентом является компонент Form, в котором происходит прием и передача входных данных в файл **App.jsx** с функцией `generateImage`, добавленной в качестве события `onClick` для кнопки **Generate Image**.

В компоненте Form создается состояние для хранения и обновления подсказки. Кроме того, функция позволяет нажимать на случайный значок для генерации случайных подсказок. Это возможно благодаря функции `handleRandomPrompt`, которая использует уже настроенную функцию `getRandomPrompt`. При нажатии на пиктограмму она получает случайную подсказку и обновляет ею состояние:

```
const handleRandomPrompt = () => {
    const randomPrompt = getRandomPrompt();
    setPrompt(randomPrompt)
}
```

Разбор файла App.jsx

Здесь находится большая часть кода. Здесь собраны все компоненты. Здесь же находится область, предназначенная для отображения сгенерированного изображения. Если изображение еще не сгенерировано, то отображается картинка-заместитель (Preview image).

Внутри этого файла осуществляется управление двумя состояниями:

- `isGenerating`: Оно позволяет определить, генерируется ли в данный момент изображение. По умолчанию оно имеет значение false.
- `generatedImage`: В этом состоянии хранится информация о сгенерированном изображении.

Кроме того, импортируется служебная функция `downloadImage`, позволяющая запускать загрузку сгенерированного изображения при нажатии кнопки **Download**: `downloadImage(generatedImage.photo)} >`

Теперь, когда вы разобрались со стартовыми файлами и настроили свой проект. Давайте начнем работать с логикой этого приложения.

Генерация изображений с помощью OpenAI’s DALL-E API

Чтобы использовать возможности API DALL-E от OpenAI, необходимо с помощью Node.js создать сервер. Внутри этого сервера будет создан POST-маршрут. Этот маршрут будет отвечать за получение текста запроса, отправленного из приложения React, и его использование для генерации изображения.

Чтобы начать работу, установите необходимые зависимости в каталог проекта, выполнив следующую команду:

```
npm i express cors openai
```

Кроме того, установите следующие зависимости в качестве dev-зависимостей. Эти инструменты помогут в настройке сервера Node.js:

```
npm i -D dotenv nodemon
```

Установленные зависимости поясняются следующим образом:

- **express:** Эта библиотека помогает создать сервер на Node.js.
- **cors:** CORS обеспечивает безопасное взаимодействие между различными доменами.
- **openai:** Эта зависимость предоставляет доступ к API DALL-E от OpenAI.
- **dotenv:** dotenv помогает управлять переменными окружения.
- **nodemon:** nodemon — это инструмент разработки, который отслеживает изменения в ваших файлах и автоматически перезапускает сервер.

После успешной установки создайте файл **server.js** в корне вашего проекта. Именно в нем будет храниться весь код вашего сервера.

В файле **server.js** импортируйте только что установленные библиотеки и инстанцируйте их:

```
// ?????? ??????????? ?????????
const express = require('express');
const cors = require('cors');
require('dotenv').config();
const OpenAI = require('openai');

// ??????? ????????? ?????????? Express
const app = express();

// ???????? ?????-??????????? ????? ????????? (CORS)
app.use(cors());

// ??????????? Express ??? ??????? JSON-?????? ? ???????????? ??????????? ??
app.use(express.json({ limit: '50mb' }));

// ??????? ????????? ?????? OpenAI ? ????????? ???? API-????
const openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
});

// ?????????? ??????? ??? ??????? ???????
const startServer = async () => {
    app.listen(8080, () => console.log('?????? ??????? ?? ????? 8080'))
};

// ????? ??????? startServer ??? ?????? ???????????? ?????????? ?????
startServer();
```

В приведенном выше коде выполняется импорт необходимых библиотек. Затем создаем экземпляр приложения Express с помощью `const app = express();`. После этого включите CORS. Далее Express настраивается на обработку входящих JSON-

данных с указанием ограничения на размер данных в `50 ??`.

После этого создается экземпляр класса OpenAI с использованием ключа OpenAI API. Создайте в корне проекта файл **.env** и добавьте в него свой API-ключ с помощью переменной `OPENAI_API_KEY`. Наконец, вы определяете асинхронную функцию `startServer` и вызываете ее, чтобы привести сервер в движение.

Теперь вы настроили свой файл **server.js**. Давайте создадим POST-маршрут, который можно использовать в React-приложении для взаимодействия с этим сервером:

```
app.post('/api', async (req, res) => {
    try {
        const { prompt } = req.body;
        const response = await openai.images.generate({
            prompt,
            n: 1,
            size: '1024x1024',
            response_format: 'b64_json',
        });
        const image = response.data[0].b64_json;
        res.status(200).json({ photo: image });
    } catch (error) {
        console.error(error);
    }
});
```

В этом коде маршрут задан как `/api`, и он предназначен для обработки входящих POST-запросов. Внутри функции обратного вызова маршрута с помощью `req.body` передаются данные, отправленные из приложения React, а именно значение `prompt`.

Затем вызывается метод `images.generate` библиотеки OpenAI. Этот метод принимает предоставленный запрос и генерирует в ответ изображение. Такие параметры, как `n`, определяют количество генерируемых изображений (в данном случае только одно), `size` задает размеры изображения, а `response_format` указывает формат, в котором должен быть предоставлен ответ(в данном случае `b64_json`).

После генерации изображения вы извлекаете данные о нем из ответа и сохраняете их в переменной `image`. Затем с помощью `res.status(200).json({ photo: image })` вы отправляете обратно в приложение React ответ в формате JSON с данными сгенерированного изображения, устанавливая статус HTTP равным `200` (чтоозначает успех).

В случае возникновения ошибок во время этого процесса код в блоке `catch` будет выполнен с записью ошибки в консоль для отладки.

Теперь сервер готов! Укажем команду, которая будет использоваться для запуска нашего сервера, в объекте `scripts` файла package.json:

```
"scripts": {
  "dev:frontend": "react-scripts start",
  "dev:backend": "nodemon server.js",
  "build": "react-scripts build",
},
```

Теперь при запуске `npm run dev:backend` ваш сервер будет запускаться на http://localhost:8080/, а при запуске `npm run dev:frontend` ваше React-приложение будет запускаться на http://localhost:3000/. Убедитесь, что оба приложения запущены на разных терминалах.

Выполнение HTTP-запросов от React к серверу Node.js

В файле **App.jsx** создадим функцию `generateImage`, которая срабатывает при нажатии кнопки **Generate Image** в компоненте **Form.jsx**. Эта функция принимает два параметра: `prompt` и `setPrompt` из компонента **Form.jsx**.

В функции `generateImage` выполните HTTP POST-запрос к серверу Node.js:

```
const generateImage = async (prompt, setPrompt) => {
    if (prompt) {
        try {
            setIsGenerating(true);
            const response = await fetch(
                'http://localhost:8080/api',
                {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({
```

```
                        prompt,
                    }),
                }
            );
            const data = await response.json();
            setGeneratedImage({
                photo: `${data.photo}`,
                altText: prompt,
            });
        } catch (err) {
            alert(err);
        } finally {
            setPrompt('');
            setIsGenerating(false);
        }
    } else {
        alert('Please provide proper prompt');
    }
};
```

В приведенном выше коде проверяется, имеет ли параметр `prompt` значение, затем состояние `isGenerating` устанавливается в `true`, поскольку операция запускается. Это приведет к появлению загрузчика на экране, поскольку в файле **App.jsx** мы имеем такой код, управляющий отображением загрузчика:

```
{isGenerating && (
     className="loader-comp">

)}
```

Далее с помощью метода `fetch()` выполняем POST-запрос к серверу, используя http://localhost:8080/api — именно поэтому мы установили CORS, так как взаимодействуем с API на другом URL. В качестве тела сообщения мы используем запрос. Затем извлекаем ответ, полученный от сервера Node.js, и устанавливаем его в состояние `generatedImage`.

Как только состояние `generatedImage` получит значение, на экран будет выведено изображение:

```
{generatedImage.photo ? (
    <img
        src={generatedImage.photo}
        alt={generatedImage.altText}
        className="imgg ai-img"
    />
) : (
    <img
        src={preview}
        alt="preview"
        className="imgg preview-img"
    />
```

```
)}
```

Вот как будет выглядеть ваш полный файл **App.jsx**:

```
import { Form, Footer, Header } from './components';
import preview from '.';
import Loader from '.'
import { downloadImage } from './utils';
import { useState } from 'react';

const App = () => {
    const [isGenerating, setIsGenerating] = useState(false);
    const [generatedImage, setGeneratedImage] = useState({
        photo: null,
        altText: null,
    });

    const generateImage = async (prompt, setPrompt) => {
        if (prompt) {
            try {
                setIsGenerating(true);
                const response = await fetch(
                    'http://localhost:8080/api',
                    {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json',
                        },
                        body: JSON.stringify({
                            prompt,
                        }),
                    }
                );
                const data = await response.json();
                setGeneratedImage({
                    photo: `${data.photo}`,
                    altText: prompt,
                });
            } catch (err) {
                alert(err);
            } finally {
                setPrompt('');
                setIsGenerating(false);
            }
        } else {
            alert('Please provide proper prompt');
        }
    };

    return (
        <div className='container'>
            <Header />
            <main className="flex-container">
                <Form generateImage={generateImage} prompt={prompt} />
                <div className="image-container">
```

```
                    {generatedImage.photo ? (
                        <img
                            src={generatedImage.photo}
                            alt={generatedImage.altText}
                            className="imgg ai-img"
                        />
                    ) : (
                        <img
                            src={preview}
                            alt="preview"
                            className="imgg preview-img"
                        />
                    )}
                    {isGenerating && (
                        <div className="loader-comp">
                            <img src={Loader} alt="" className='loader-img' />
                        </div>
                    )}
                    <button
                        className="btn"
                        onClick={() => downloadImage(generatedImage.photo)}
                    >
                        Download
                    </button>
                </div>
            </main>
            <Footer />
        </div>
    );
};

export default App;
```

Развертывание приложения

Итак, вы успешно создали React-приложение, взаимодействующее с Node.js, что делает его приложением полного стека. Теперь давайте развернем это приложение на боевом сервере.

Сначала настройте сервер на обслуживание статических файлов, созданных в процессе сборки React-приложения. Для этого импортируем модуль `path` и используем его для обслуживания статических файлов:

```
const path = require('path');

app.use(express.static(path.resolve(__dirname, './build')));
```

После выполнения команды `npm run build && npm run dev:backend` ваше React-приложение с полным стеком будет загружено по адресу http://localhost:8080/. Это

связано с тем, что React-приложение компилируется в статические файлы в папке **build**. Затем эти файлы включаются в ваш сервер Node.js в виде статического каталога. Следовательно, при запуске сервера Node приложение будет доступно.

Перед развертыванием кода на выбранном вами Git-провайдере (Bitbucket, GitHub или GitLab) не забудьте изменить URL HTTP-запроса в файле **App.jsx**. Замените `http://localhost:8080/api` на `/api`, поскольку URL будет дополнен.

Наконец, в файле **package.json** добавьте команду script для сервера Node.js, который будет использоваться для развертывания:

```
"scripts": {
  // ..
  "start": "node server.js",
},
```

Далее разместите код на выбранном вами Git-провайдере или разверните репозиторий на сервере!

Резюме

Подходя к концу нашего путешествия по созданию полноценного приложения с использованием API DALL-E от OpenAI, React и Node.js, мы не только освоили основы, но и открыли для себя мощь искусственного интеллекта в области генерации изображений. Это руководство предоставило вам инструменты и знания для создания приложения, которое сочетает в себе креативность и технический прогресс.

Теперь, когда вы освоили этот процесс, перед вами открываются широкие возможности для дальнейшего исследования и творчества в мире ИИ. Новые модели искусственного интеллекта появляются каждый день, расширяя границы возможного и предлагая еще более удивительные способы визуализации ваших

идей. Экспериментируйте, исследуйте и не бойтесь реализовывать амбициозные проекты, которые можно развернуть как на локальном сервере, так и в продакшен-среде.

Этот опыт не только показывает, как технологии могут служить инструментом для творчества, но и подчеркивает, что в современном мире программирования нет предела для того, что вы можете создать, используя силу искусственного интеллекта. Пусть это руководство станет вашей отправной точкой в путешествии по миру ИИ, полном творческих возможностей и технических открытий.

Дата Создания
25.10.2023