

Создание веб-приложения для получения данных о погоде с помощью языка программирования JavaScript и HTML/CSS

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассмотрен процесс создания веб-приложения для получения информации о погоде в населенных пунктах. Приложение разработано на языке программирования JavaScript, HTML/CSS и API портала «weatherstack.com». Результатом исследования является веб-приложение и описание его разработки.

Ключевые слова: JavaScript, API, HTML, CSS

Creating a web application for obtaining weather data using the JavaScript and HTML/CSS programming language

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article describes the process of creating a web application for obtaining weather information in populated areas. The application is developed in the JavaScript programming language, HTML/CSS and the portal API «weatherstack.com». The result of the research is a web application and a description of its development.

Keywords: JavaScript, API, HTML, CSS

1 Введение

1.1 Актуальность

Актуальность исследуемой проблемы обусловлена несколькими ключевыми факторами. В последние годы веб-технологии, такие как HTML, CSS и JavaScript, стали основными инструментами для разработки интерактивных и динамичных веб-приложений. Эти технологии позволяют создавать кроссплатформенные приложения, которые работают на любых устройствах с доступом к интернету. Информация о погоде является важной для различных сфер деятельности, включая сельское хозяйство, транспорт, туризм и повседневную жизнь. Создание веб-приложения для получения данных о погоде позволяет пользователям быстро и удобно получать актуальную информацию, что способствует принятию обоснованных решений. Существует множество API (Application Programming Interface),

предоставляющих данные о погоде. Использование этих API в сочетании с JavaScript позволяет разработчикам создавать мощные и функциональные приложения, которые могут интегрироваться с другими сервисами и системами. Современные пользователи ожидают высокого уровня удобства и интерактивности от веб-приложений. Использование HTML/CSS для создания привлекательного и интуитивно понятного интерфейса, а также JavaScript для обеспечения динамического взаимодействия с пользователем, позволяет создавать приложения, которые соответствуют этим ожиданиям. Также разработка веб-приложений с использованием открытых технологий, с помощью которых будет реализоваться веб-приложение, является экономически эффективной. Эти технологии не требуют значительных затрат на лицензирование и поддержку, что делает их привлекательными, например, для стартапов и малых предприятий. Таким образом, исследование выбранной темы имеет высокую актуальность и практическую значимость, как с точки зрения технических возможностей, так и с точки зрения пользовательских потребностей и экономической эффективности.

1.2 Обзор исследований

А.М. Шор в своём научном труде произвёл обзор и сравнение популярных подходов взаимодействия с веб-сервисами [1]. Ш.М. Рахимов и Д.А. Федоров изучили вопрос создания веб-приложений для работы с API [2]. С.А.А. Сугаипов, А.Р. Юнусов и Х.Л. Себаева исследовали интеграцию принципов адаптивного веб-дизайна с ведущими фреймворками JavaScript React, Angular и Vue [3]. А.Г. Дьяков и А.Н. Шмелёва описали создание веб-приложений с наглядным примером, рассказывается о видах веб-приложений и об одном из самых популярных языков программирования – JavaScript [4]. А.Р. Горлов, А.В. Сухарский и Д.А. Смирнов рассмотрели использование языка сценария в JavaScript для создания веб-сайта [5].

1.3 Цель исследования

Целью исследования является создание веб-приложения для получения данных о погоде в населенных пунктах с помощью языка программирования JavaScript, HTML/CSS и технологии API.

2 Материалы и методы

Для создания приложения используется язык программирования JavaScript, язык гипертекстовой разметки HTML, таблицы стилей CSS и API сервиса «weatherstack.com». В качестве среды разработки выступает Visual Studio Code.

3 Результаты и обсуждения

Перед непосредственной разработкой веб-приложения необходимо получить API. Для этого используется бесплатный сервис «weatherstack». Он представляет из себя веб сайта, где нужно зарегистрироваться для получения

ключа API. После данного этапа получается API, с помощью которого будет браться информация о погоде с сервиса.

Перед началом программирования создаётся папка проекта, где создаются три файла – index.html, script.js и style.css. Первый файл отмечает за разметку веб страницы, к которому подключается второй файл с инструкциями JavaScript. Данный файл является основным файлом, так как содержит весь функционал разрабатываемого веб-приложения. Последний файл представляет из себя таблицы стилей для оформления внешнего вида приложения. Также создаётся папка для ресурсов проекта, а именно папка «img», в которой будут находиться графические данные – картинки и .gif файлы.

Начало работы начинается с создания и наполнение файла index.html (рис. 1).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>

    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="app">
      <div id="root">
        
      </div>

      <div class="popup" id="popup">
        <form action="GET" class="form" id="form">
          <div class="form-group">
            <input
              type="text"
              value=""
              name="city"
              class="form-input"
              id="text-input"
              placeholder="London"
              required
            />
            <input
              type="submit"
              class="form-submit"
              id="submit-button"
              value="OK"
            />
          </div>
        </form>

        <div class="popup-close" id="close">
          <span></span>
        </div>
      </div>
    </div>

    <script src="./script.js"></script>
  </body>
</html>
```

Рисунок 1. Содержимое index.html

С помощью средств Visual Code создаётся шаблонный код HTML. В данном шаблонном коде объявляются такие параметры, как язык, кодировка UTF-8, название страницы, ссылка на подключаемый файл со скриптом и мета-тэг для адаптации страницы под мобильные устройства.

Следующим шагом является программирование формы веб-приложения. Для этого создаётся главный контейнер `div` с `id="app"`, который будет использоваться для обращения к данному элементу скрипта и таблицы стилей. Далее создаётся второй `div` элементы, который помещается во внутрь основного. Данный контейнер содержит в себе тег `img` с `gif` файлом. В совокупности данный участок кода будет создавать экран загрузки веб-приложения при включении. Затем программируется третий контейнер, который также является вложенным в первый. Этот элемент будет представлять из себя `popup` – всплывающее окно, в котором нужно будет вводить название населённого пункта.

Внутри описанного элемента находится тэг `form` с атрибутами: «`action="GET"`» — метод отправки данных; «`class="form"`» — класс для стилизации с помощью таблицы стилей; «`id="form"`» — идентификатор для манипуляции с помощью CSS и JavaScript. Для удобства использования формы создаются также два тега `input` (поля ввода). Первый представляет из себя текстовое поле с параметрами типа поля, именем, идентификатором, текстом-подсказкой и атрибутом для обязательного заполнения поля. Второе поле представляет из себя кнопку (`type="submit"`) для отправки данных, введённых в вышеописанном элементе. Для закрытия всплывающего окна используется отдельный контейнер с классом «`popup-close`». При нажатии на него будет происходить закрытие формы.

Внешний вид приложения с первичными методами разработан. Далее следует написать непосредственно функционал приложения в файле `script.js`. Данный файл является самым объёмным, так как в нём находится вся логика приложения. Так как работать программа будет с API, то для этого создаётся константа, куда помещается ключ API. Далее создаются константы, которые будут хранить пути до элементов, объявленных в `index.html`, для непосредственной обработки действий. Следующим действием является создание объекта для хранения данных о погоде. Объект содержит свойства информационного характера такие, как название города, температуру, время, показатель дня и ночи, описание. Помимо классических свойств описываемого элемента используется свойства типа объекта, которое хранит в себе специфическую информацию о погоде такую, как облачность, влажность, скорость ветра, давление, ультрафиолетовый индекс и видимость.

Описанный участок кода подготавливает основу для работы с API погоды и взаимодействия с пользователем через форму и всплывающее окно. Он создает константы для хранения URL API и элементов DOM (рис. 2)

```
const link =
  "http://api.weatherstack.com/current?access_key=335e3544859fae40cea97d3043df91ef";

const root = document.getElementById("root");
const popup = document.getElementById("popup");
const textInput = document.getElementById("text-input");
const form = document.getElementById("form");

let store = {
  city: "Minsk",
  temperature: 0,
  observationTime: "00:00 AM",
  isDay: "yes",
  description: "",
  properties: {
    cloudcover: {},
    humidity: {},
    windSpeed: {},
    pressure: {},
    uvIndex: {},
    visibility: {},
  },
};
```

Рисунок 2. Создание констант для хранения DOM элементов, ключа API и объекта с полями информации о погоде

Хранилища созданы. После этого реализуется асинхронная функция «fetchData».

```
const fetchData = async () => {
  try {
    const query = localStorage.getItem("query") || store.city;
    const result = await fetch(`${link}&query=${query}`);
    const data = await result.json();

    const {
      current: {
        cloudcover,
        temperature,
        humidity,
        observation_time: observationTime,
        pressure,
        uv_index: uvIndex,
        visibility,
        is_day: isDay,
        weather_descriptions: description,
        wind_speed: windSpeed,
      },
      location: { name },
    } = data;

    ...store,
    isDay,
    city: name,
    temperature,
    observationTime,
    description: description[0],
    properties: {
      cloudcover: {
        title: "cloudcover",
        value: `${cloudcover}%`,
        icon: "cloud.png",
      },
      humidity: {
        title: "humidity",
        value: `${humidity}%`,
        icon: "humidity.png",
      },
      windSpeed: {
        title: "wind speed",
        value: `${windSpeed} km/h`,
        icon: "wind.png",
      },
      pressure: {
        title: "pressure",
        value: `${pressure} %`,
        icon: "gauge.png",
      },
      uvIndex: {
        title: "uv Index",
        value: `${uvIndex} / 100`,
        icon: "uv-index.png",
      },
      visibility: {
        title: "visibility",
        value: `${visibility}%`,
        icon: "visibility.png",
      },
    },
  };

  renderComponent();
} catch (err) {
  console.log(err);
}
```

Рисунок 3. Асинхронная функция «fetchData»

Данный тип функции позволяет использовать ключевое слово «await» для ожидания завершения асинхронных операций, а именно запросы к API. В начале функции создаются три константы. Первая константа используется для получения запроса из «localStorage» или использования значение «city» из объекта «store». Это позволяет сохранять последний запрос пользователя между сессиями. Оставшиеся две константы отправляют запрос к API погоды с использованием URL, хранящегося в переменной «link», и параметра «query». Результат запроса сохраняется в переменной «result», а затем преобразуется в JSON и сохраняется в переменной «data».

После данных констант следует деструктуризация данных. Данный процесс позволяет извлечь нужные свойства из полученных данных API. Деструктуризация позволяет легко извлекать нужные данные из объекта «data». Следующим шагом является обновление объекта «store». На данном этапе вышеупомянутое поле обновляется новыми данными. Здесь же используется оператор распространения «(...store)», чтобы сохранить существующие свойства объекта «store», и затем обновить их новыми значениями. В конце функции прописывается вызов функции «renderComponent», которая будет обновлять интерфейс пользователя на основе данных из объекта «store».

Для генерации HTML разметки, то есть внешнего вида отображения данных, применяются следующие функции.

Функция «getImage» отвечает за получение той или иной иконки из ресурса проекта для отображения (рис. 4).

```
const getImage = (description) => {  
  const value = description.toLowerCase();  
  
  switch (value) {  
    case "partly cloudy":  
      return "partly.png";  
    case "cloud":  
      return "cloud.png";  
    case "fog":  
      return "fog.png";  
    case "sunny":  
      return "sunny.png";  
    case "cloud":  
      return "cloud.png";  
    default:  
      return "the.png";  
  }  
};
```

Рисунок 4. Функция «getImage»

Эта функция принимает строку «description» и возвращает имя изображения на основе значения этой строки. Первый шаг метода представляет из себя преобразование строки в нижний регистр, а на втором шаге проверяются значения имён и данных.

Следующая функция «renderProperty» принимает объект properties и возвращает HTML-разметку для каждого свойства (рис. 5).

```
const renderProperty = (properties) => {
  return Object.values(properties)
    .map(({ title, value, icon }) => {
    return `<div class="property">
      <div class="property-icon">
        
      </div>
      <div class="property-info">
        <div class="property-info__value">${value}</div>
        <div class="property-info__description">${title}</div>
      </div>
    </div>`;
  })
  .join("");
};
```

Рисунок 5. Функция «renderProperty»

Поэтапно функция возвращает массив значений объекта «properties». Далее метод «map» преобразует каждое значение в строку HTML, используя деструктуризацию для извлечения свойств «title», «value» и «icon». В заключении метод «join("")» объединяет все строки HTML в одну строку.

Последняя функция «markup» используется для генерации HTML-разметки для отображения данных о погоде (рис. 6).

```
const markup = () => {
  const { city, description, observationTime, temperature, isDay, properties } =
    store;
  const containerClass = isDay === "yes" ? "is-day" : "";

  return `<div class="container ${containerClass}">
    <div class="top">
      <div class="city">
        <div class="city-subtitle">Weather Today in</div>
        <div class="city-title" id="city">
          <span>${city}</span>
        </div>
      </div>
      <div class="city-info">
        <div class="top-left">
          
          <div class="description">${description}</div>
        </div>
        <div class="top-right">
          <div class="city-info__subtitle">as of ${observationTime}</div>
          <div class="city-info__title">${temperature}°</div>
        </div>
      </div>
      <div id="properties">${renderProperty(properties)}</div>
    </div>`;
};
```

Рисунок 6. Функция «markup»

Данный метод извлекает свойства «city», «description», «observationTime», «temperature», «isDay» и «properties» из объекта «store». Затем определяет класс контейнера «containerClass» в зависимости от значения «isDay». Если «isDay» равно «yes», то добавляется класс «is-day». В результате возвращает строку HTML, которая включает: контейнер с классом «container» и дополнительным классом «containerClass»; блок с информацией о городе «city»; блок с информацией о погоде «city-info», включающий иконку погоды, описание, время наблюдения и температуру; блок с дополнительными свойствами погоды «properties», который генерируется с помощью функции «renderProperty».

В конце создаются вспомогательные функции (рис. 7).

```
const togglePopupClass = () => {
  popup.classList.toggle("active");
};

const renderComponent = () => {
  root.innerHTML = markup();

  const city = document.getElementById("city");
  city.addEventListener("click", togglePopupClass);
};

const handleInput = (e) => {
  store = {
    ...store,
    city: e.target.value,
  };
};

const handleSubmit = (e) => {
  e.preventDefault();
  const value = store.city;

  if (!value) return null;

  localStorage.setItem("query", value);
  fetchData();
  togglePopupClass();
};

form.addEventListener("submit", handleSubmit);
textInput.addEventListener("input", handleInput);
```

Рисунок 7. Функции обновления и обработки

Функция «togglePopupClass» добавляет или удаляет класс «active» у элемента «popup»; «renderComponent» обновляет содержимое элемента «root» с помощью HTML-разметки, сгенерированной функцией «markup»; «handleInput» обновляет свойство «city» в объекте «store» на основе значения, введенного пользователем в текстовое поле; «handleSubmit» обрабатывает отправку формы. После описанных методов следуют два обработчика

событий к форме и текстовому полю. В конце прописывается вызов функции «fetchData» для первоначального запроса данных о погоде при загрузке страницы.

После разработки приложения следует его тестирование (рис. 9 и 10).

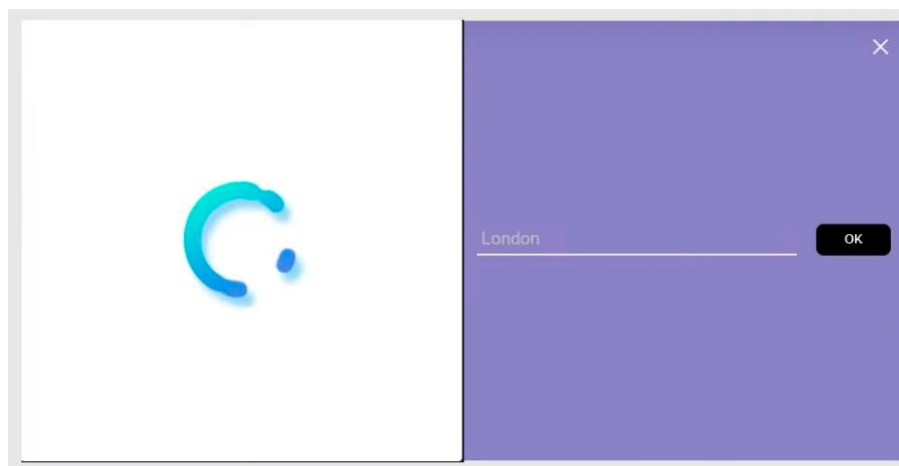


Рисунок 9. Экран загрузки и всплывающее окно для ввода города



Рисунок 10. Результат работы веб-приложения

В результате исследования было разработано веб-приложения для получения данных о погоде. Приложение написано на языке программирования JavaScript, HTML/CSS и интерфейсом API сервиса «weatherstack».

Библиографический список

1. Шор А.М. Сравнительный анализ подходов в разработке API веб-приложений // StudNet. 2020. Т. 3. № 9. С. 533-540.
2. Рахимов Ш.М., Федоров Д.А. Разработка простейших веб-приложений создание API // В сборнике: МОЛОДОЙ ИССЛЕДОВАТЕЛЬ: ВЫЗОВЫ И ПЕРСПЕКТИВЫ. сборник статей по материалам СХХХIV международной научно-практической конференции. 2019. С. 63-67.
3. Сугаипов С.А.А., Юнусов А.Р., Себаева Х.Л. Создание адаптивных веб-приложений с использованием фреймворков JavaScript // Экономика и предпринимательство. 2023. № 12 (161). С. 942-946.
4. Дьяков А.Г., Шмелёва А.Н. JavaScript для создания веб приложений // Уральский научный вестник. 2022. Т. 10. № 2. С. 70-74.
5. Горлов А.Р., Сухарский А.В., Смирнов Д.А. Использование языка сценария в JavaScript для создания веб-сайта // В сборнике: Перспективные аспекты моделирования систем и процессов. Материалы Всероссийской научно-практической конференции. Воронеж, 2023. С. 43-48.