

Создание веб-словаря на базе бесплатного API с использованием JavaScript

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассмотрен процесс создания веб-словаря, предоставляющий объяснение слов, а также произношение слова на английском языке. Программа реализована на языке программирования JavaScript, HTML-разметки и таблицы стилей CSS. Результатом исследования является само веб-приложение и описание его работы.

Ключевые слова: JavaScript, HTML, веб-разработка

Creating a web dictionary based on a free API using JavaScript

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the process of creating a web dictionary that provides an explanation of words, as well as the pronunciation of a word in English. The program is implemented in the JavaScript programming language, HTML markup and CSS style sheets. The result of the research is the web application itself and a description of its operation.

Keywords: JavaScript, HTML, web development

1 Введение

1.1 Актуальность

В современном мире веб-технологии играют ключевую роль в повседневной жизни и профессиональной деятельности. Они обеспечивают доступ к информации, упрощают коммуникацию и позволяют автоматизировать множество процессов. Среди различных веб-приложений особое место занимают веб-словари, которые предоставляют пользователям возможность быстро и удобно находить значения слов, переводы и другие лингвистические данные. Создание веб-словаря на базе бесплатного API с использованием JavaScript является актуальной задачей, так как позволяет создать доступное и функциональное приложение, которое может быть полезно широкому кругу пользователей.

1.2 Обзор исследований

С.Э. Шмаков, Н.Н. Смекалин и Е.В. Щенникова в своей статье рассмотрели способ создания Web-приложения на языке JavaScript с использованием стороннего API-интерфейса [1]. К.А. Лехов, Д.Д. Сперанский, М.А. Митрохин и Н.С. Карамышева изучили вопрос использования API социальной сети «ВКонтакте» в качестве инструмента извлечения данных [2]. О.Н. Уткина в своём исследовании изучили API на примере использования API Яндекс.Карт для разработки интерактивной карты [3]. С.А.А. Сугаипов, А.Р. Юнусов и Себаева Х.Л. исследовали интеграцию принципов адаптивного веб-дизайна с ведущими фреймворками JavaScript React, Angular и Vue [4]. Н.Д. Чистоступов сравнил современные JavaScript фреймворки, а также описал их различные аспекты [5].

1.3 Цель исследования

Целью исследования является создание веб-словаря с возможностью озвучивания произношения слова на базе бесплатного API и языка программирования JavaScript.

2 Материалы и методы

Для создания приложения используется язык программирования JavaScript, HTML-разметка и таблица стилей CSS. В качестве сетевой технологии выступает API бесплатного интернет словаря. Используемой IDE выступает Microsoft Visual Code.

3 Результаты и обсуждения

Проект приложения состоит из трёх папок, которые группируют файлы по назначению: медиа файлы, разметка страницы, стили для страницы и скрипты.

Внешний вид приложения представлен просто в виде поисковой строки, кнопок и списка выдачи результата (рис. 1).

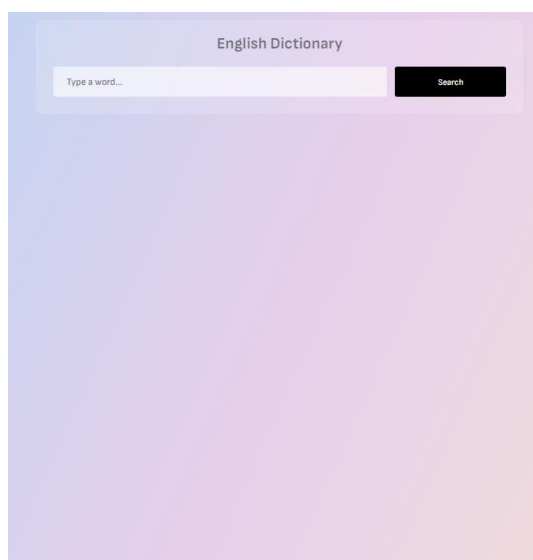


Рисунок 1. Внешний вид приложения

Данная визуальная составляющая приложения размечена в файле "index.html" (рис. 2).

```
<body>
  <div class="app">
    <main class="main">
      <h1>English Dictionary</h1>

      <form class="form">
        <div class="form-group">
          <input
            type="text"
            placeholder="Type a word..."
            name="word"
            id="word-input"
            autocomplete="off"
            value=""
          />
        </div>
        <button type="submit">Search</button>
      </form>

      <div class="error"></div>

      <div class="results">
        <div class="results-info">
          <div class="results-word"></div>

          <div class="results-sound">
            
          </div>
        </div>

        <div class="results-list"></div>
      </div>
    </main>
  </div>

  <script src="./assets/js/scripts.js"></script>
</body>
```

Рисунок 2. Файл разметки веб-страницы приложения

В данном коде представлен HTML-документ, который служит основой для веб-словаря. Структура документа включает основной контейнер с классом "app", внутри которого находится главный блок "main" с классом "main". Внутри этого блока размещен заголовок первого уровня (h1) с текстом "English Dictionary".

Далее следует форма с классом "form", содержащая поле ввода типа "text" с атрибутом "placeholder", который предлагает пользователю ввести слово. Поле ввода имеет идентификатор "word-input" и атрибут "autocomplete" со значением "off", что предотвращает автозаполнение. Кнопка отправки формы имеет тип "submit" и текст "Search". Ниже формы расположен блок для отображения ошибок с классом "error".

Последний блок с классом "results" предназначен для отображения результатов поиска. Внутри него находится контейнер "results-info", который включает элемент для отображения искомого слова с классом "results-word" и элемент для отображения иконки звука с классом "results-sound". Иконка

звука представлена изображением, источник которого указан в атрибуте "src". Завершает структуру блок "results-list", предназначенный для отображения списка результатов.

В конце документа подключается JavaScript-файл "scripts.js", который содержит логику работы веб-словаря (рис. 3). Этот файл отвечает за обработку введенных данных, взаимодействие с API и отображение результатов поиска.

```
let state = {
  word: "",
  meanings: [],
  phonetics: [],
};

const url = "https://api.dictionaryapi.dev/api/v2/entries/en/";
const input = document.getElementById("word-input");
const form = document.querySelector(".form");
const containerWord = document.querySelector(".results-word");
const soundButton = document.querySelector(".results-sound");
const resultsWrapper = document.querySelector(".results");
const resultsList = document.querySelector(".results-list");
const errorContainer = document.querySelector(".error");

const showError = (error) => {
  errorContainer.style.display = "block";
  resultsWrapper.style.display = "none";

  errorContainer.innerText = error.message;
};

const renderDefinition = (itemDefinition) => {
  const example = itemDefinition.example
  ? `<div class="results-item__example">
    <p>Example: <span>${itemDefinition.example}</span></p>
  </div>`
  : "";

  return `<div class="results-item__definition">
    <p>${itemDefinition.definition}</p>
    ${example}
  </div>`;
};

const getDefinitions = (definitions) => {
  return definitions.map(renderDefinition).join("");
};
```

Рисунок 3. Объявление состояния (state) и переменных, необходимых для работы с API и DOM-элементами

В данном фрагменте кода происходит объявление состояния (state) и переменных, необходимых для работы с API и DOM-элементами.

Состояние (state) представлено объектом, содержащим три свойства: "word" для хранения введенного слова, "meanings" для хранения значений

слова и "phonetics" для хранения фонетической информации. Переменная "url" содержит базовый URL для запросов к API словаря. Далее идут переменные, которые ссылаются на различные элементы DOM: поле ввода (input), форма (form), контейнер для отображения искомого слова (containerWord), кнопка воспроизведения звука (soundButton), обертка для результатов (resultsWrapper), список результатов (resultsList) и контейнер для отображения ошибок (errorContainer).

Функция "showError" принимает объект ошибки и отображает сообщение об ошибке в контейнере "errorContainer". Она также скрывает обертку для результатов (resultsWrapper) и делает контейнер ошибок видимым.

Метод "renderDefinition" принимает объект определения слова (itemDefinition) и возвращает HTML-строку, представляющую определение и, при наличии, пример использования слова. Если пример существует, он добавляется в HTML-строку в виде блока с классом "results-item__example". В противном случае, пример не отображается. Возвращаемая HTML-строка содержит определение слова внутри блока с классом "results-item__definition".

Следующий фрагмент скрипта позволяет обеспечивать отображение результатов поиска и вставку искомого слова в интерфейс веб-словаря (рис. 4).

```
const renderItem = (item) => {
  return `<div class="results-item">
    <div class="results-item__part">${item.partOfSpeech}</div>
    <div class="results-item__definitions">
      ${getDefinitions(item.definitions)}
    </div>
  </div>`;
};

const showResults = () => {
  resultsWrapper.style.display = "block";
  resultsList.innerHTML = "";

  state.meanings.forEach((item) => (resultsList.innerHTML += renderItem(item)));
};

const insertWord = () => {
  containerWord.innerText = state.word;
};
```

Рисунок 4. Функционал отображения результатов поиска и вставки

Функция "renderItem" принимает объект "item", представляющий собой часть речи и соответствующие определения слова. Она возвращает HTML-строку, которая включает в себя блок с классом "results-item". Внутри этого блока находится элемент с классом "results-item__part", содержащий часть речи, и элемент с классом "results-item__definitions", в который вставляются определения слова, полученные с помощью функции "getDefinitions". Метод "showResults" отвечает за отображение результатов поиска. Она делает обертку для результатов (resultsWrapper) видимой и очищает содержимое списка результатов (resultsList). Затем функция проходит по массиву

"meanings" в состоянии (state) и для каждого элемента вызывает функцию "renderItem", добавляя возвращаемую HTML-строку в список результатов.

Третья функция "insertWord" вставляет искомое слово в контейнер для отображения слова (containerWord). Она устанавливает текстовое содержимое этого контейнера равным значению свойства "word" в состоянии (state).

Для обработки отправки формы и выполнения запросов к API применяется следующий функционал (рис. 5).

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

  errorContainer.style.display = "none";

  if (!state.word.trim()) return;

  try {
    const response = await fetch(`${url}${state.word}`);
    const data = await response.json();

    if (response.ok && data.length) {
      const item = data[0];

      state = {
        ...state,
        meanings: item.meanings,
        phonetics: item.phonetics,
      };

      insertWord();
      showResults();
    } else {
      showError(data);
    }
  } catch (err) {
    console.log(err);
  }
};
```

Рисунок 5. Обработка формы и выполнение запросов к API

Здесь функция "handleSubmit" обрабатывает отправку формы и выполняет запрос к API для получения данных о введенном слове. Она предотвращает стандартное поведение формы и скрывает контейнер для отображения ошибок. Если поле ввода не пустое, выполняется запрос к API с использованием метода "fetch". URL запроса формируется путем конкатенации базового URL и значения введенного слова.

После получения ответа от API данные преобразуются в формат JSON. Если ответ успешен и данные содержат хотя бы один элемент, извлекается первый элемент данных. Состояние обновляется, добавляя в него массивы "meanings" и "phonetics" из полученного элемента. Затем вызываются

функции "insertWord" и "showResults" для отображения искомого слова и результатов поиска соответственно. Если ответ не успешен или данные пустые, вызывается "showError" для отображения сообщения об ошибке. В случае возникновения ошибки при выполнении запроса, она регистрируется в консоли. Эта функция обеспечивает взаимодействие с API и динамическое обновление интерфейса веб-словаря на основе введенных пользователем данных.

В конце разработки программируется кнопки для интерактивного взаимодействия с приложением (рис. 6).

```
const handleKeyup = (e) => {  
  const value = e.target.value;  
  
  state.word = value;  
};  
  
const handleSound = () => {  
  if (state.phonetics.length) {  
    const sound = state.phonetics[0];  
  
    if (sound.audio) {  
      new Audio(sound.audio).play();  
    }  
  }  
};  
  
// EVENTS  
input.addEventListener("keyup", handleKeyup);  
form.addEventListener("submit", handleSubmit);  
soundButton.addEventListener("click", handleSound);
```

Рисунок 6. Программирование кнопок управления и обработчиков событий

Первая функция "handleKeyup" обрабатывает событие "keyup" (отпускание клавиши) в поле ввода. Она обновляет состояние (state) значением введенного слова. Вторая "handleSound" воспроизводит звук, если в состоянии (state) присутствует фонетическая информация. Она извлекает первый элемент массива "phonetics" и, если он содержит аудио, воспроизводит его.

Далее добавляются обработчики событий: событие "keyup" на поле ввода (input) обрабатывается функцией "handleKeyup"; событие "submit" на форме (form) обрабатывается с помощью "handleSubmit"; "click" на кнопке воспроизведения звука (soundButton) обрабатывается методом "handleSound".

После программирования логики и внешнего вида производится тестирование и проверка его работоспособности (рис. 7).

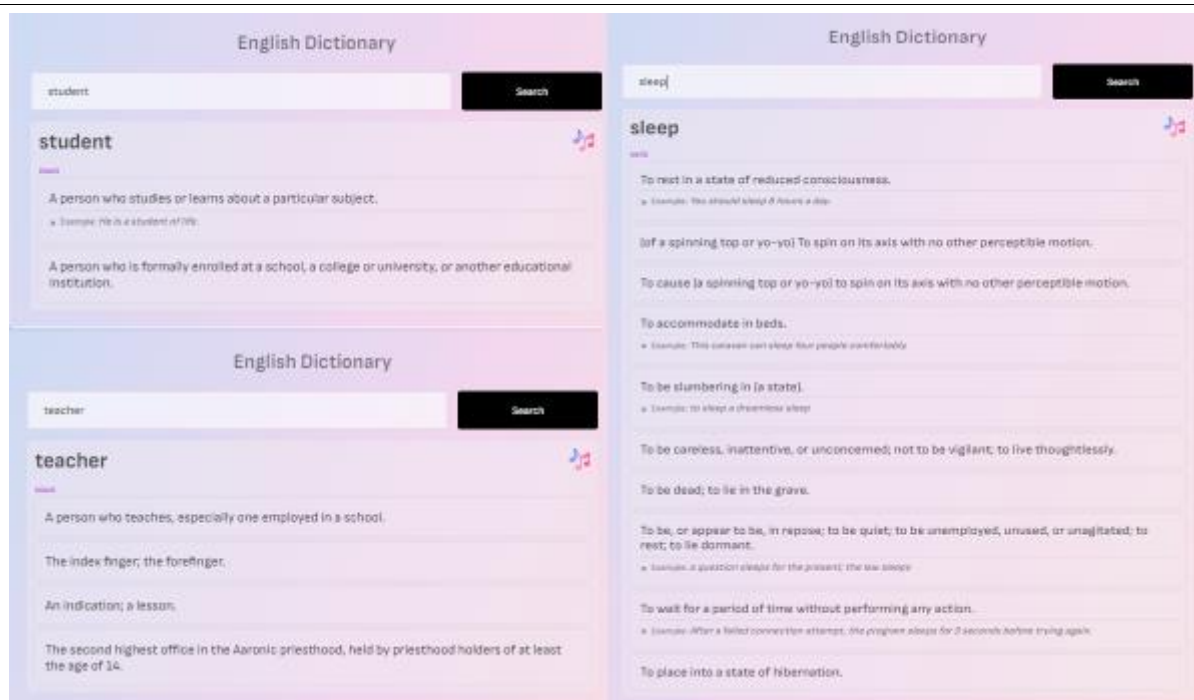


Рисунок 7. Результаты тестирования приложения

Таким образом было разработано веб-приложения «Словарь» с помощью открытого и бесплатного API и веб-технологий, а именно JavaScript, HTML-разметка и таблицы стилей.

Библиографический список

1. Шмаков С.Э., Смекалин Н.Н., Щенникова Е.В. Создание Web-приложения с использованием сторонних API // В сборнике: XLIX Огарёвские чтения. Материалы научной конференции: в 3 частях. Саранск, 2021. С. 668-671.
2. Лехов К.А., Сперанский Д.Д., Митрохин М.А., Карамышева Н.С. Система извлечения и анализа текстовых данных из социальных сетей для образовательного учреждения // Модели, системы, сети в экономике, технике, природе и обществе. 2021. № 1 (37). С. 128-136.
3. Уткина О.Н. Современные информационные технологии в экскурсионной деятельности // Сервис в России и за рубежом. 2023. Т. 17. № 6 (108). С. 26-35.
4. 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.
5. Чистоступов Н.Д. Сравнение современных JavaScript фреймворков для Web-приложений // В сборнике: Подготовка профессиональных кадров в магистратуре для цифровой экономики (ПКМ-2023). Всероссийская научно-техническая и научно-методическая конференция магистрантов и их руководителей. Сборник лучших докладов: в 2-х томах. Санкт-Петербург, 2024. С. 337-342.