

Создание музыкального плеера на языке программирования JavaScript

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассмотрен процесс создания веб-приложения для прослушивания музыки. Веб-приложение написано с использованием языка программирования JavaScript, HTML и CSS. Результатом исследования является рабочий вариант программы с описанием её работы.

Ключевые слова: JavaScript, HTML, веб-программирование

Creating a music player in the JavaScript programming language

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article describes the process of creating a web application for listening to music. The web application is written using the JavaScript, HTML, and CSS programming languages. The result of the research is a working version of the program with a description of its operation.

Keywords: JavaScript, HTML, Web programming

1 Введение

1.1 Актуальность

Создание музыкального плеера на языке программирования JavaScript является актуальной задачей в современном мире, учитывая растущую популярность веб-технологий и необходимость в многофункциональных и удобных инструментах для воспроизведения музыки. В условиях быстрого развития интернета и мобильных устройств, веб-приложения становятся все более востребованными, и JavaScript, как один из самых распространенных языков программирования для веб-разработки, предоставляет мощные инструменты для создания интерактивных и динамичных приложений. Исследование в данной области позволит не только углубить знания и навыки в области веб-разработки, но и применить их на практике, создавая современные и функциональные решения.

1.2 Обзор исследований

Д.К. Аксенова, Т.К. Душаева и А.М. Алиев в совместном исследовании изучили функционал языков C и C++ на примере создания музыкальной

программы [1]. Ф.А. Десятириков и В.И. Лютин описали принцип аудио воспроизведения в разрабатываемом приложении [2]. А.З. Энес, З.В. Нагоев и К.Ф. Край разработали программу перевода с иностранных языков аудиозаписи [3]. Архангельский Т.А., Созинова О.А. описали принцип работы программы, воспроизводящей аудиофайлы [4]. Д.Р. Вихляев в своей статье рассмотрел принцип обработки звука на языке программирования Python [5].

1.3 Цель исследования

Целью исследования является создание аудио плеера с использованием веб-технологий таких, как JavaScript, HTML и CSS.

2 Материалы и методы

Для создания веб-приложения используется язык программирования JavaScript, язык разметки HTML и таблицы стилей CSS. В качестве IDE используется программа Visual Studio Code.

3 Результаты и обсуждения

Подготовка к созданию приложения включает в себя создание трёх папок: «audio», «images» и «js». Данные папки хранят в себе в зависимости от названия аудиофайлы, изображения и скрипты проекта. Отдельным файлом в корнях проекта создаётся файл index.html, который будет отвечать за внешний вид приложения вместе с файлами CSS. Также для корректной и удобной работы используются плагин «Live Server» для Visual Studio Code. Данный плагин удобен тем, что позволяет запускать локальный сервер и отслеживать все изменения файлов разрабатываемого веб-приложения в реальном времени.

В начале проекта происходит написание кода в файле «index.html» (рис. 1).

Данный HTML-код представляет собой структуру веб-страницы для музыкального плеера. Внутри тега «<body>» находится основной контейнер с классом «root», который содержит заголовок «<h1>» с текстом "Музыкальный плеер". Внутри этого контейнера расположен основной блок «<main>» с классом «app», который включает в себя несколько секций для управления плеером и отображения списка треков. Внутри блока «<main>» находится контейнер «<div>» с классом «content», который содержит три основные секции: «handling», «current» и «list». Секция «handling» включает в себя элементы управления плеером, такие как регулировка громкости («handling-sound»), повторение («handling-repeat») и случайное воспроизведение («handling-shuffle»). Каждый из этих элементов содержит иконку, представленную SVG-элементом, и, в случае регулировки громкости, также ползунок («<input>» типа «range»), который позволяет пользователю изменять уровень громкости от 0 до 1 с шагом 0.1. Секция «current» предназначена для отображения текущего трека, а секция «list» содержит заголовок «All tracks» и контейнер «items», который будет использоваться

для отображения списка всех треков. В конце HTML-документа подключается JavaScript-файл «scripts.js» с помощью соответствующего тега, который будет содержать логику для работы плеера, включая обработку событий и управление воспроизведением музыки.



Рисунок 1. Разметка веб-приложения

Разметка приложения создана. Визуальная составляющая часть разметки находится в файлах CSS, на которых внимание заостряться не будет.

Следующим шагом создаются два вспомогательные скрипта – «data.js» и «utils.js».

Первый скрипт предназначен для хранения информации о аудиофайлах – идентификатор файла, путь на музыкальный файл, жанр, название песни, исполнитель и год создания песни (рис. 2).



Рисунок 2. Массив с данными об аудиофайлах

Следующий скрипт позволяет обрабатывать длительность аудиофайла и преобразовывать в указанный формат времени «минуты: секунды», а также перемешивать название аудиофайлов в списке плеера (рис. 3).

```
const formatTime = (time) => (time < 10 ? `0${time}` : time);

export const toMinAndSec = (duration) => {
  const minutes = formatTime(Math.floor(duration / 60));
  const seconds = formatTime(Math.floor(duration - minutes * 60));

  return `${minutes}:${seconds}`;
};

export const shuffle = (array) => array.sort(() => 0.5 - Math.random());
```

Рисунок 3. Скрипт преобразование длительности песни и перемешивание списка песен плеера

Данные скрипты были вынесены в отдельные файлы для лучшей читабельности кода проекта.

Последним этапом разработки веб-приложения является написание главной логики программы, которая будет содержаться в файле «script.js».

Первоначально происходит импорт дополнительных скриптов, объявление главного объекта скрипта, метода инициализации плеера (рис. 4).

```
import { data } from "../data.js";
import { shuffle, toMinAndSec } from "../utils.js";

const AudioController = {
  state: {
    audios: [],
    current: {},
    repeating: false,
    playing: false,
    volume: 0.5,
  },

  init() {
    this.initVariables();
    this.initEvents();
    this.renderAudios();
  },

  initVariables() {
    this.playButton = null;
    this.audioList = document.querySelector(".items");
    this.currentItem = document.querySelector(".current");
    this.repeatButton = document.querySelector(".handling-repeat");
    this.volumeButton = document.querySelector(".controls-volume");
    this.shuffleButton = document.querySelector(".handling-shuffle");
  },

  initEvents() {
    this.audioList.addEventListener("click", this.handleItem.bind(this));
    this.repeatButton.addEventListener("click", this.handleRepeat.bind(this));
    this.volumeButton.addEventListener("change", this.handleVolume.bind(this));
    this.shuffleButton.addEventListener("click", this.handleShuffle.bind(this));
  },
};
```

Рисунок 4. Содержимое «script.js» - инициализация модуля плеера

Объект «AudioController» имеет несколько методов и свойств. Свойство «state» хранит текущее состояние плеера, включая массив аудиофайлов («audios»), текущий воспроизводимый трек («current»), флаги для повторения («repeating») и воспроизведения («playing»), а также уровень громкости («volume»).

Метод «init» инициализирует плеер, вызывая методы «initVariables», «initEvents» и «renderAudios». Метод «initVariables» инициализирует переменные, связанные с элементами интерфейса, такими как кнопки

воспроизведения, повторения, регулировки громкости и перемешивания, а также контейнеры для списка треков и текущего трека.

Метод «initEvents» добавляет обработчики событий к элементам интерфейса. Например, он добавляет обработчик клика к списку треков, обработчик клика к кнопке повторения, обработчик изменения к ползунку громкости и обработчик клика к кнопке перемешивания.

Далее происходит объявление методов перемешивания, обработки громкости, повторения песен, воспроизведение и паузы, переключение на предыдущие и следующие песни (рис. 5).

```

handleShuffle() {
  const { children } = this.audiolist;
  const shuffled = shuffle([...children]);

  this.audiolist.innerHTML = "";
  shuffled.forEach((item) => this.audiolist.appendChild(item));
},

handleVolume({ target: { value } }) {
  const { current } = this.state;

  this.state.volume = value;

  if (!current?.audio) return;

  current.audio.volume = value;
},

handleRepeat({ currentTarget }) {
  const { repeating } = this.state;

  currentTarget.classList.toggle("active", !repeating);
  this.state.repeating = !repeating;
},

handleAudioPlay() {
  const { playing, current } = this.state;
  const { audio } = current;

  !playing ? audio.play() : audio.pause();

  this.state.playing = !playing;
  this.playButton.classList.toggle("playing", !playing);
},

handleNext() {
  const { current } = this.state;

  const currentItem = document.querySelector(`[data-id="${current.id}"]`);
  const next = currentItem.nextSibling?.dataset;
  const first = this.audiolist.firstChild?.dataset;

  const itemId = next?.id || first?.id;

  if (!itemId) return;

  this.setCurrentItem(itemId);
},

handlePrev() {
  const { current } = this.state;

  const currentItem = document.querySelector(`[data-id="${current.id}"]`);
  const prev = currentItem.previousSibling?.dataset;
  const last = this.audiolist.lastChild?.dataset;

  const itemId = prev?.id || last?.id;

  if (!itemId) return;

  this.setCurrentItem(itemId);
},

```

Рисунок 5. Функции управления плеером: переключение песен, повтор, пауза/воспроизведение, изменение громкости

Метод «handleShuffle» перемешивает элементы списка треков. Он извлекает все дочерние элементы из контейнера «audiolist», перемешивает их с помощью функции «shuffle», очищает контейнер и затем добавляет перемешанные элементы обратно в контейнер.

Функция «handleVolume» обрабатывает изменение громкости. Он обновляет значение громкости в состоянии плеера и, если текущий трек существует, устанавливает его громкость на новое значение.

Реализация «handleRepeat» управляет функцией повторения трека. Он переключает класс active на кнопке повторения и обновляет состояние повторения в плеере.

Метод «handleAudioPlay» управляет воспроизведением и паузой текущего трека. Он проверяет текущее состояние воспроизведения и либо запускает, либо останавливает трек, а также обновляет состояние воспроизведения и класс кнопки воспроизведения.

Метод «handleNext» переключает на следующий трек. Он находит текущий элемент в списке треков, определяет следующий элемент и, если его нет, переходит к первому треку в списке. После устанавливает новый текущий трек.

Решение «handlePrev» переключает на предыдущий трек. Он находит текущий элемент в списке треков, определяет предыдущий элемент и, если его нет, переходит к последнему треку в списке. Затем он устанавливает новый текущий трек.

Описанные методы обеспечивают основную функциональность аудиоплеера, позволяя пользователю управлять воспроизведением, громкостью, повторением и перемешиванием треков, а также переключаться между треками. Они делают интерфейс плеера интерактивным и удобным для использования.

Затем осуществляется создание функций, обеспечивающих интерактивное управление воспроизведением треков и обновление интерфейса в реальном времени (рис. 6).

```
handlePlayer() {  
  const play = document.querySelector(".controls-play");  
  const next = document.querySelector(".controls-next");  
  const prev = document.querySelector(".controls-prev");  
  
  this.playButton = play;  
  
  play.addEventListener("click", this.handleAudioPlay.bind(this));  
  next.addEventListener("click", this.handleNext.bind(this));  
  prev.addEventListener("click", this.handlePrev.bind(this));  
},  
  
audioUpdateHandler({ audio, duration }) {  
  const progress = document.querySelector(".progress-current");  
  const timeline = document.querySelector(".timeline-start");  
  
  audio.addEventListener("timeupdate", ({ target }) => {  
    const { currentTime } = target;  
    const width = (currentTime * 100) / duration;  
  
    timeline.innerHTML = toMinAndSec(currentTime);  
    progress.style.width = `${width}%`;  
  });  
  
  audio.addEventListener("ended", ({ target }) => {  
    target.currentTime = 0;  
    progress.style.width = `0%`;  
  
    this.state.repeating ? target.play() : this.handleNext();  
  });  
},
```

Рисунок 6. Функции интерактивного управления и обновления интерфейса в реальном времени

Одна из функций инициализирует кнопки управления плеером и добавляет к ним обработчики событий. Она находит элементы кнопок воспроизведения, перехода к следующему треку и перехода к предыдущему треку, сохраняет ссылку на кнопку воспроизведения и добавляет обработчики событий клика к каждой из кнопок. Эти обработчики вызывают соответствующие методы для управления воспроизведением, переходом к следующему и предыдущему треку.

Другая функция обновляет интерфейс плеера в реальном времени на основе текущего состояния воспроизведения аудио. Она принимает объект с аудиоэлементом и продолжительностью трека, находит элементы прогресса воспроизведения и текущего времени, добавляет обработчик события «timeupdate» к аудиоэлементу для обновления прогресса и текущего времени, а также обработчик события «ended» для управления повторением трека или переходом к следующему треку.

Следующие методы дополняют предыдущие и увеличивают интерактивность (рис. 7).

```

pauseCurrentAudio() {
  const {
    current: { audio },
  } = this.state;

  if (!audio) return;

  audio.pause();
  audio.currentTime = 0;
},

togglePlaying() {
  const { playing, current } = this.state;
  const { audio } = current;

  playing ? audio.play() : audio.pause();

  this.playButton.classList.toggle("playing", playing);
},

setCurrentItem(itemId) {
  const current = this.state.audios.find(({ id }) => +id === +itemId);

  if (!current) return;

  this.pauseCurrentAudio();

  this.state.current = current;
  this.currentItem.innerHTML = this.renderCurrentItem(current);

  current.audio.volume = this.state.volume;

  this.handlePlayer();
  this.audioUpdateHandler(current);

  setTimeout(() => {
    this.togglePlaying();
  }, 5);

  handleItem(({ target }) {
    const { id } = target.dataset;

    if (!id) return;

    this.setCurrentItem(id);
  },

```

Рисунок 8. Дополнительные функции интерактивного упарвления и обновления интерфейса в реальном времени

Первая из функций останавливает воспроизведение текущего трека, если он существует, и сбрасывает его текущее время на ноль. Это позволяет приостановить воспроизведение и подготовить трек к повторному запуску.

Вторая функция переключает состояние воспроизведения текущего трека. Если трек в данный момент воспроизводится, он будет приостановлен, и наоборот. Эта функция также обновляет класс кнопки воспроизведения, чтобы визуально отразить текущее состояние воспроизведения.

Метод установки текущего трека находит трек по его идентификатору и обновляет состояние плеера, чтобы отразить выбранный трек. Она останавливает воспроизведение текущего трека, обновляет интерфейс для отображения нового трека, устанавливает громкость и инициализирует обработчики событий для кнопок управления плеером. После этого она запускает воспроизведение нового трека с небольшой задержкой.

Функция обработки выбора трека из списка извлекает идентификатор выбранного трека и вызывает функцию установки текущего трека, чтобы обновить состояние плеера и интерфейс.

Отображении плеера осуществляется путём двух функций рендеринга HTML-разметки (рис. 9 и 10).

```

renderCurrentItem({ link, track, year, group, duration }) {
  const [image] = link.split(".");

  return `<div
    class="current-image"
    style="background-image: url(./assets/images/${image}.jpg)"
  ></div>

  <div class="current-info">
    <div class="current-info__top">
      <div class="current-info__titles">
        <h2 class="current-info__group">${group}</h2>
        <h3 class="current-info__track">${track}</h3>
      </div>

      <div class="current-info__year">${year}</div>
    </div>

    <div class="controls">
      <div class="controls-buttons">
        <button class="controls-button controls-prev">
          <svg class="icon-arrow">
            <use xlink:href="./assets/images/sprite.svg#arrow"></use>
          </svg>
        </button>

        <button class="controls-button controls-play">
          <svg class="icon-pause">
            <use xlink:href="./assets/images/sprite.svg#pause"></use>
          </svg>
          <svg class="icon-play">
            <use xlink:href="./assets/images/sprite.svg#play"></use>
          </svg>
        </button>

        <button class="controls-button controls-next">
          <svg class="icon-arrow">
            <use xlink:href="./assets/images/sprite.svg#arrow"></use>
          </svg>
        </button>
      </div>

      <div class="controls-progress">
        <div class="progress">
          <div class="progress-current"></div>
        </div>

        <div class="timeline">
          <span class="timeline-start">00:00</span>
          <span class="timeline-end">${toMinAndSec(duration)}</span>
        </div>
      </div>
    </div>
  </div>`;
}

```

Рисноук 9. Функция для рендеринга информации о текущем треке в интерфейсе аудиоплеера

```

renderItem({ id, link, track, genre, group, duration }) {
  const [image] = link.split(".");

  return `<div class="item" data-id="${id}">
    <div
      class="item-image"
      style="background-image: url(./assets/images/${image}.jpg)"
    ></div>

    <div class="item-titles">
      <h2 class="item-group">${group}</h2>
      <h3 class="item-track">${track}</h3>
    </div>

    <p class="item-duration">${toMinAndSec(duration)}</p>
    <p class="item-genre">${genre}</p>

    <button class="item-play">
      <svg class="icon-play">
        <use xlink:href="./assets/images/sprite.svg#play"></use>
      </svg>
    </button>
  </div>`;
}

```

Рисноук 10. Функция для рендеринга информации о треке в списке треков аудиоплеера

Эти функции обеспечивают визуальное отображение информации о треках в интерфейсе плеера, делая его более информативным и интерактивным для пользователя. Они позволяют легко выбирать и воспроизводить треки из списка, а также управлять их воспроизведением и отображением в реальном времени.

В конце скрипта прописывается код, который загружает и отображает список треков в интерфейсе аудиоплеера (рис. 11).

```

loadAudioData(audio) {
  this.audioList.innerHTML += this.renderItem(audio);
},

renderAudios() {
  data.forEach((item) => {
    const audio = new Audio(`./assets/audio/${item.link}`);

    audio.addEventListener("loadeddata", () => {
      const newItem = { ...item, duration: audio.duration, audio };

      this.state.audios.push(newItem);
      this.loadAudioData(newItem);
    });
  });
},
};

```

Рисунок 11. Загрузка и отображение списка песен в интерфейсе аудиоплеера

Функция «loadAudioData» добавляет HTML-разметку трека в контейнер. Функция «renderAudios» инициализирует и загружает данные о всех треках, создает объекты «Audio», добавляет их в состояние плеера и отображает в интерфейсе.

После программирования логики, осуществляется тестирование веб-приложения (рис. 12)

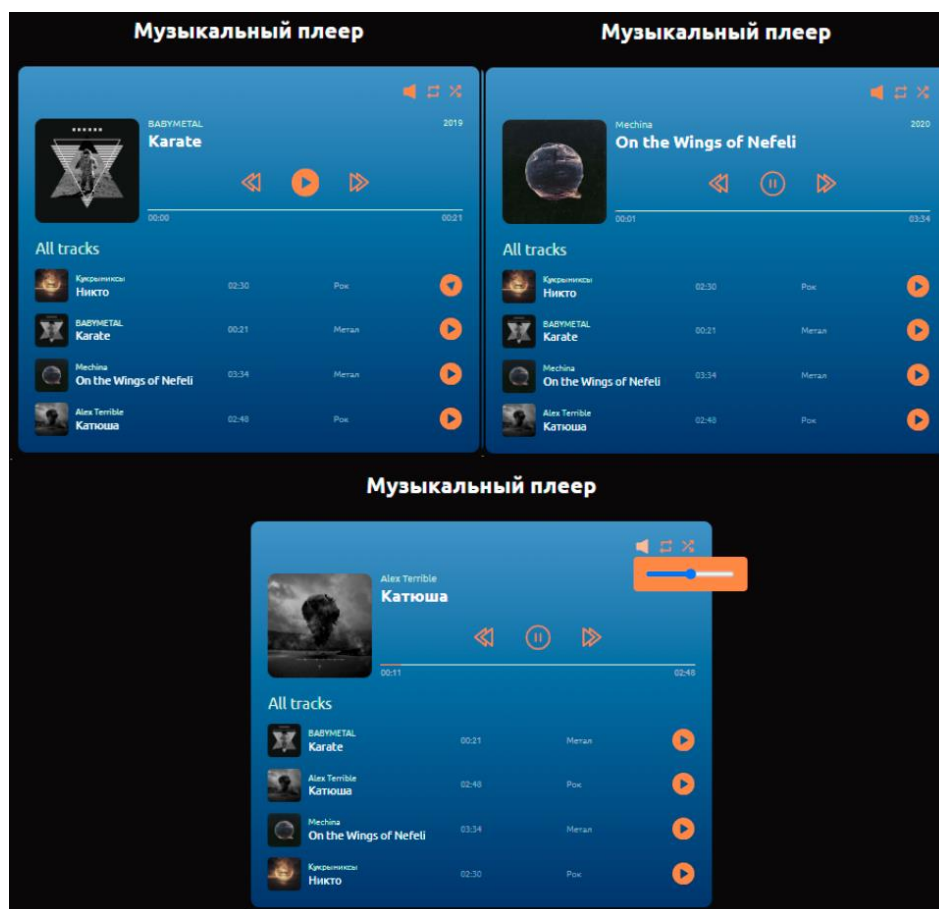


Рисунок 12. Демонстрация работы приложения

Выводы

Таким образом в процессе исследования было создано веб-приложение аудиоплеер, который выполняет все свойственные функции аналогичных программ. Также было приведено подробное описание хода разработки программы.

Библиографический список

1. Аксенова Д.К., Душаева Т.К., Алиев А.М. Функции языка программирования C, C++ // В сборнике: Междисциплинарность науки как фактор инновационного развития. сборник статей по итогам Международной научно-практической конференции. 2018. С. 65-68.3
2. Десятириков Ф.А., Лютин В.И. Android-программа для тестирования в аудио и 2D среде взаимодействия с пользователем // Свидетельство о регистрации программы для ЭВМ RU 2022669630, 21.10.2022. Заявка № 2022666630 от 11.09.2022.
3. Энес А.З., Нагоев З.В., Край К.Ф. Программа для распределенной платформы аудио переводов // Свидетельство о регистрации программы для ЭВМ RU 2024610267, 09.01.2024. Заявка от 19.12.2023.
4. Архангельский Т.А., Созинова О.А. Мультимедийный корпус языка идиш // Научно-техническая информация. Серия 2: Информационные процессы и системы. 2015. № 3. С. 18-24.
5. Вихляев Д.Р. Регулирование скорости звука с помощью языка программирования Python // Постулат. 2021. № 10 (72).