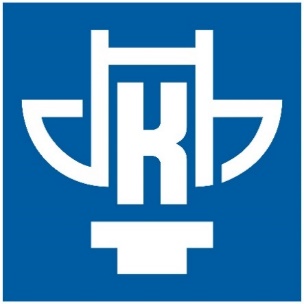
BỘ XÂY DỰNG VIỆT NAM

TRƯỜNG ĐẠI HỌC KIẾN TRÚC HÀ NỘI

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----



**BÀI NGHIÊN CỨU**

**NHÓM 6**

**ĐỀ TÀI:** **XÂY DỰNG WEBSITE**

**GHI CHÚ THÔNG TIN NGHỆ SĨ**

**Giảng viên hướng dẫn : cô NGUYỄN THỊ HẠNH**

**Thành viên nhóm : PHẠM TUẤN MINH**

**NGUYỄN THÀNH TRUNG**

**HOÀNG NGỌC THÀNH**

**LỚP : 20CN5**

**MỤC LỤC**

[1. Giới thiệu về HTML 5](#_Toc146542103)

[1.1. Hướng dẫn tạo một tệp HTML cơ bản. 5](#_Toc146542104)

[1.2. Tạo trang web với tiêu đề, đoạn văn bản, hình ảnh và siêu liên kết. 5](#_Toc146542105)

[1.3. Hướng dẫn về cú pháp và các phần tử HTML cơ bản. 6](#_Toc146542106)

[2. Cơ bản về CSS 6](#_Toc146542107)

[2.1. Hướng dẫn cách thêm kiểu cho trang web bằng CSS. 6](#_Toc146542108)

[2.2. Tạo các lớp và áp dụng kiểu bằng cách sử dụng lựa chọn. 7](#_Toc146542109)

[2.3. Thử nghiệm với các thuộc tính CSS cơ bản như font chữ, màu sắc, độ rộng và chiều cao. 8](#_Toc146542110)

[3. JavaScript cơ bản : 8](#_Toc146542111)

[3.1. Giới thiệu cú pháp JavaScript cơ bản. 8](#_Toc146542112)

[3.1.1. App.js 8](#_Toc146542113)

[3.1.1.1. Import React và các thư viện cần thiết: 8](#_Toc146542114)

[3.1.1.2. Thiết lập thư viện FontAwesome 9](#_Toc146542115)

[3.1.1.3. Component `App 9](#_Toc146542116)

[3.1.2. serviceWorker.js 10](#_Toc146542117)

[3.1.2.1. Kiểm tra xem trang web đang chạy trên localhost hay không 10](#_Toc146542118)

[3.1.2.2.. Hàm `register(config)` 11](#_Toc146542119)

[3.1.2.3. Hàm `registerValidSW(swUrl, config)` 11](#_Toc146542120)

[3.1.2.4. Hàm `checkValidServiceWorker(swUrl, config)`: 11](#_Toc146542121)

[3.1.2.5. Hàm `unregister()`: 12](#_Toc146542122)

[3.2. Hướng dẫn tạo biến, hàm, và sử dụng sự kiện. 12](#_Toc146542123)

[3.2.1. Import thư viện và các thành phần 13](#_Toc146542124)

[3.2.2. Thiết lập thư viện FontAwesome 14](#_Toc146542125)

[3.2.3. Component `App`: 14](#_Toc146542126)

[3.3. Tạo các ví dụ đơn giản như hiển thị thông báo khi nút được nhấn. 14](#_Toc146542127)

[3.3.1. `navigator.serviceWorker.register(swUrl)`: 16](#_Toc146542128)

[3.3.2. `.then(registration => { ... })`: 16](#_Toc146542129)

[3.3.3. `registration.onupdatefound = () => { ... }`: 16](#_Toc146542130)

[3.3.4. `const installingWorker = registration.installing;`: 16](#_Toc146542131)

[3.3.5. `if (installingWorker == null) { return; }`: 16](#_Toc146542132)

[3.3.6. `installingWorker.onstatechange = () => { ... }`: 16](#_Toc146542133)

[3.3.7. `if (installingWorker.state === 'installed') { ... }`: 17](#_Toc146542134)

[3.3.8. `if (navigator.serviceWorker.controller) { ... }`: 17](#_Toc146542135)

[3.3.9. `console.log('New content is available and will be used when all tabs for this page are closed...');`: 17](#_Toc146542136)

[3.3.10. `config.onUpdate(registration);`: 17](#_Toc146542137)

[3.3.11. Nếu không có trình duyệt Service Worker chạy: 17](#_Toc146542138)

[3.3.12. `config.onSuccess(registration);`: 17](#_Toc146542139)

[3.3.13. `.catch(error => { ... })`: 17](#_Toc146542140)

[4. HTML Forms và Input Elements 18](#_Toc146542141)

[4.1. Cách tạo biểu mẫu HTML đơn giản với các phần tử nhập liệu như ô văn bản, ô chọn, và nút gửi 18](#_Toc146542142)

[4.1.1. `<!DOCTYPE html>` 18](#_Toc146542143)

[4.1.2. `<html lang="en">` 18](#_Toc146542144)

[4.1.3. `<head>` 18](#_Toc146542145)

[4.2. Tạo kiểu cho biểu mẫu bằng CSS 19](#_Toc146542146)

[4.2.1. `.App` 20](#_Toc146542147)

[4.2.2. `.App-logo` 20](#_Toc146542148)

[4.2.3. `.App-header` 20](#_Toc146542149)

[4.2.4. `.App-link` 21](#_Toc146542150)

[4.2.5. `.navbar-item img` 21](#_Toc146542151)

[4.2.6. `.musician-card` 21](#_Toc146542152)

[4.2.7. `@keyframes App-logo-spin` 21](#_Toc146542153)

[4.3. Sử dụng JavaScript để xử lý dữ liệu từ biểu mẫu 21](#_Toc146542154)

[4.3.1. Import các thư viện và thành phần 22](#_Toc146542155)

[4.3.2. `library.add(faEdit);` 23](#_Toc146542156)

[4.3.3. `class App extends Component { ... }` 23](#_Toc146542157)

[4.3.4. `render() { ... }` 23](#_Toc146542158)

[4.3.5. Trong `render()`,có một giao diện HTML và React được sử dụng để định tuyến các trang web 24](#_Toc146542159)

[4.3.6. `export default App;` 24](#_Toc146542160)

[5. Quy trình phát triển web 25](#_Toc146542161)

[5.1. Giới thiệu về quy trình phát triển web từ lập kế hoạch đến triển khai trang web. 25](#_Toc146542162)

[5.2. Xác định các bước quan trọng như phân tích yêu cầu, thiết kế giao diện, phát triển, kiểm thử, và triển khai. 25](#_Toc146542163)

[5.3. Hướng dẫn về vai trò và trách nhiệm của từng thành viên trong nhóm, bao gồm quản lý dự án và phát triển. 26](#_Toc146542164)

# 1. Giới thiệu về HTML

# 1.1. Hướng dẫn tạo một tệp HTML cơ bản.

Trong tệp `Index.html`, em đã tạo một trang web HTML cơ bản. Đây là cấu trúc HTML cơ bản của trang web. Tệp này bao gồm các phần như `<!DOCTYPE html>`, `<html>`, `<head>`, và `<body>`.

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico" />

    <meta

      name="viewport"

      content="width=device-width, initial-scale=1, shrink-to-fit=no"

    />

    <meta name="theme-color" content="#000000" />

    <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

    <title>Musician App</title>

  </head>

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>

</html>

# 1.2. Tạo trang web với tiêu đề, đoạn văn bản, hình ảnh và siêu liên kết.

Trong tệp `Index.html`, đã thêm tiêu đề trang web ("Musician App") và một đoạn văn bản trong thẻ `<body>`. Điều này minh họa cách HTML được sử dụng để hiển thị nội dung trên trang web.

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>

</html>

# 1.3. Hướng dẫn về cú pháp và các phần tử HTML cơ bản.

Trong tệp `Index.html`, đã sử dụng cú pháp HTML cơ bản như `<meta>`, `<link>`, và `<div>`. Cú pháp này được sử dụng để thêm thông tin meta, liên kết tới các tệp khác, và chứa nội dung trên trang web.

# 2. Cơ bản về CSS

# 2.1. Hướng dẫn cách thêm kiểu cho trang web bằng CSS.

Trong tệp `App.css`, đã định nghĩa các quy tắc CSS để tùy chỉnh giao diện trang web. Ví dụ, đã định dạng các phần tử có class `.App`, `.App-logo`, `.App-header`, và `.App-link`.

.App {

  text-align: center;

}

.App-logo {

  animation: App-logo-spin infinite 20s linear;

  height: 40vmin;

  pointer-events: none;

}

.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-link {

  color: #61dafb;

}

.navbar-item img {

  max-height: 3.3em;

}

.musician-card {

  width: 30%;

  float: left;

  margin-right: 1em;

}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

# 2.2. Tạo các lớp và áp dụng kiểu bằng cách sử dụng lựa chọn.

**`.App` :** Đây đang định nghĩa kiểu dáng cho một phần tử có tên lớp là "App". Nó đặt căn chỉnh văn bản ở giữa trang.

**`.App-logo` :** Đây đang định nghĩa kiểu dáng cho một phần tử có tên lớp là "App-logo". Nó bao gồm một hiệu ứng gọi là "App-logo-spin" làm cho phần tử xoay vô tận trong khoảng 20 giây với tốc độ đều (linear). Nó cũng đặt chiều cao của phần tử là 40vmin và không cho phép tương tác bằng pointer-events: none.

**`.App-header**`: Đây đang định nghĩa kiểu dáng cho một phần tử có tên lớp là "App-header". Nó đặt màu nền là #282c34 (một màu tối), đảm bảo chiều cao tối thiểu là 100vh (chiều cao của khung nhìn trình duyệt), và biến nó thành một container flex với bố cục cột. Nó căn chỉnh các phần tử con của nó (các phần tử bên trong) vào giữa cả theo chiều ngang và chiều dọc. Kích thước phông chữ được đặt sao cho tương đối với kích thước của khung nhìn (2vmin lớn hơn 10px), và màu văn bản là màu trắng.

**`.App-link`** : Đây đang định nghĩa kiểu dáng cho một phần tử có tên lớp là "App-link". Nó đặt màu văn bản là #61dafb, một màu xanh dương.

**`.navbar-item img`** : Đây có vẻ nhắm vào các hình ảnh bên trong các phần tử có tên lớp là "navbar-item". Nó đặt chiều cao tối đa của các hình ảnh này là 3.3em.

**`.musician-card`** : Đây đang định nghĩa kiểu dáng cho các phần tử có tên lớp là "musician-card". Nó đặt chiều rộng là 30%, cho phép phần tử nổi theo hướng trái (float: left), và thêm một lề 1em bên phải.

**`@keyframes App-logo-spin`**: Đây đang định nghĩa một keyframe CSS animation có tên là "App-logo-spin". Nó chỉ định cách một phần tử sẽ biến đổi từ một trạng thái (from) thành trạng thái (to) khác bằng cách quay từ 0 độ đến 360 độ.

# 2.3. Thử nghiệm với các thuộc tính CSS cơ bản như font chữ, màu sắc, độ rộng và chiều cao.

Trong tệp `App.css`, chúng em đã sử dụng các thuộc tính CSS như `font-size`, `color`, `background-color`, `width`, và `height` để tùy chỉnh giao diện trang web. Điều này cho phép thay đổi kích thước, màu sắc và định dạng của các phần tử.

# 3. JavaScript cơ bản :

# 3.1. Giới thiệu cú pháp JavaScript cơ bản.

JavaScript được sử dụng trong tệp `App.js` và `serviceWorker.js`.

# 3.1.1. App.js

# 3.1.1.1. Import React và các thư viện cần thiết:

- `import React, { Component } from 'react';`: Đây là việc import React và Component từ thư viện 'react'. React là thư viện chính cho việc xây dựng giao diện người dùng, và Component là một lớp cơ sở để xây dựng các thành phần (components) trong React.

- `import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';`: Đoạn mã này import các thành phần từ thư viện 'react-router-dom', mà thường được sử dụng để quản lý định tuyến (routing) trong ứng dụng React.

- `import './App.css';`: Đoạn mã này import một tệp CSS có tên 'App.css' để áp dụng kiểu dáng cho các thành phần trong ứng dụng.

- `import Navbar from './components/Navbar';`, `import Home from './components/Home';`, `import Footer from './components/Footer';`: Đoạn mã này import các thành phần (components) được xác định trong các tệp 'Navbar.js', 'Home.js', và 'Footer.js'. Các thành phần này là các phần của ứng dụng và sẽ được sử dụng trong phần render của `App` component.

# 3.1.1.2. Thiết lập thư viện FontAwesome

- `import { library } from '@fortawesome/fontawesome-svg-core';`: Đoạn mã này import thư viện chính của FontAwesome.

- `import { faEdit } from '@fortawesome/free-solid-svg-icons'`: Import một biểu tượng cụ thể từ FontAwesome, trong trường hợp này là biểu tượng "faEdit" (biểu tượng chỉnh sửa). FontAwesome là một thư viện biểu tượng vector phổ biến mà có thể sử dụng để thêm biểu tượng vào trang web của mình.

- `library.add(faEdit);`: Thêm biểu tượng "faEdit" vào thư viện FontAwesome để sau đó có thể sử dụng nó trong các thành phần khác của ứng dụng.

# 3.1.1.3. Component `App

- `class App extends Component { ... }`: Đây là một component React chính của ứng dụng. Nó được xây dựng trên cơ sở của lớp `Component` từ React.

- Trong phần `render()`, `App` component trả về một cấu trúc giao diện bao gồm `Router` (cho định tuyến), `Navbar` (thanh điều hướng), `Switch` (để chuyển đổi giữa các routes), `Route` (định nghĩa route cho trang Home), và `Footer` (chân trang).

- `<Router>`: Đây là thành phần gốc cho định tuyến của ứng dụng. Nó bao quanh tất cả các thành phần liên quan đến định tuyến.

- `<Switch>`: Thành phần này sử dụng để đảm bảo chỉ một route duy nhất được hiển thị tại một thời điểm.

- `<Route>`: Định nghĩa một route cho trang chính (path="/") và liên kết nó với component `Home`. Điều này có nghĩa là khi route là "/" (trang chính), component `Home` sẽ được hiển thị.

- `<Navbar>` và `<Footer>`: Đây là các thành phần giao diện, như thanh điều hướng và chân trang, mà đã import ở trước đó và đưa vào trong `Router`.

# 3.1.2. serviceWorker.js

serviceWorker.js được sử dụng để quản lý việc đăng ký và cài đặt Service Worker trong ứng dụng web, giúp tạo ra các tính năng như cache nội dung để có thể hoạt động offline và tăng hiệu suất trải nghiệm người dùng.

# 3.1.2.1. Kiểm tra xem trang web đang chạy trên localhost hay không

const isLocalhost = Boolean(

  window.location.hostname === 'localhost' ||

    // [::1] is the IPv6 localhost address.

    window.location.hostname === '[::1]' ||

    // 127.0.0.1/8 is considered localhost for IPv4.

    window.location.hostname.match(

      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/

    )

);

- Đoạn mã này xác định xem trang web hiện đang chạy trên máy chủ localhost hay không bằng cách so sánh địa chỉ hostname với các giá trị "localhost", "[::1]" (địa chỉ IPv6 localhost), và một loạt các địa chỉ IPv4 liên quan đến localhost.

# 3.1.2.2.. Hàm `register(config)`

- Hàm này được sử dụng để đăng ký Service Worker cho ứng dụng web.

- Đầu tiên, nó kiểm tra xem ứng dụng đang chạy ở môi trường sản xuất và trình duyệt có hỗ trợ Service Worker hay không.

- Nếu ứng dụng không chạy trên localhost và trình duyệt hỗ trợ Service Worker, nó sẽ tiến hành đăng ký Service Worker bằng cách gọi `navigator.serviceWorker.register(swUrl)`.

- Nếu ứng dụng đang chạy trên localhost, nó kiểm tra xem Service Worker đã tồn tại hay chưa, sau đó thực hiện một số xử lý liên quan đến Service Worker và hiển thị thông báo lên console.

# 3.1.2.3. Hàm `registerValidSW(swUrl, config)`

- Hàm này được gọi sau khi Service Worker đã được đăng ký thành công.

- Nó theo dõi trạng thái của Service Worker và hiển thị thông báo trên console khi Service Worker đã được cài đặt (state === 'installed') và khi Service Worker đã sẵn sàng phục vụ trang web.

# 3.1.2.4. Hàm `checkValidServiceWorker(swUrl, config)`:

- Hàm này kiểm tra xem Service Worker có tồn tại hay không bằng cách thử tải tệp service-worker.js từ máy chủ.

- Nếu không tìm thấy Service Worker hoặc tệp không phải là tệp JavaScript, nó sẽ hủy đăng ký Service Worker và làm tải lại trang web để loại bỏ Service Worker.

- Nếu tìm thấy Service Worker, nó sẽ gọi `registerValidSW` để đảm bảo rằng Service Worker đã đăng ký đúng cách.

# 3.1.2.5. Hàm `unregister()`:

- Hàm này sử dụng để hủy đăng ký Service Worker nếu nó đã được đăng ký.

- Nó kiểm tra xem trình duyệt có hỗ trợ Service Worker không và sau đó thực hiện hủy đăng ký nếu có.

# 3.2. Hướng dẫn tạo biến, hàm, và sử dụng sự kiện.

Trong `App.js`, đã sử dụng JavaScript để xử lý các thành phần như biến, hàm và sự kiện (routing).

import React, { Component } from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import './App.css';

import Navbar from './components/Navbar';

import Home from './components/Home';

import Footer from './components/Footer';

import { library } from '@fortawesome/fontawesome-svg-core';

import { faEdit } from '@fortawesome/free-solid-svg-icons'

library.add(faEdit);

class App extends Component {

  render() {

    return (

      <div className="App">

        <Router>

          <div>

            <Navbar />

            <Switch>

              <Route exact path="/" component={Home} />

            </Switch>

            <Footer />

          </div>

        </Router>

      </div>

    );

  }

}

export default App;

# 3.2.1. Import thư viện và các thành phần

- `import React, { Component } from 'react';`: Dòng này import React và Component từ thư viện 'react'. React là thư viện chính cho việc xây dựng giao diện người dùng, và Component là một lớp cơ sở để xây dựng các thành phần (components) trong React.

- `import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';`: Đoạn mã này import các thành phần từ thư viện 'react-router-dom', mà thường được sử dụng để quản lý định tuyến (routing) trong ứng dụng React.

- `import './App.css';`: Đoạn mã này import một tệp CSS có tên 'App.css' để áp dụng kiểu dáng cho các thành phần trong ứng dụng.

- `import Navbar from './components/Navbar';`, `import Home from './components/Home';`, `import Footer from './components/Footer';`: Đoạn mã này import các thành phần (components) được định nghĩa trong các tệp 'Navbar.js', 'Home.js', và 'Footer.js'. Các thành phần này là các phần của ứng dụng và sẽ được sử dụng trong phần render của `App` component.

# 3.2.2. Thiết lập thư viện FontAwesome

- `import { library } from '@fortawesome/fontawesome-svg-core';`: Dòng này import thư viện chính của FontAwesome.

- `import { faEdit } from '@fortawesome/free-solid-svg-icons'`: Import một biểu tượng cụ thể từ FontAwesome, trong trường hợp này là biểu tượng "faEdit" (biểu tượng chỉnh sửa). FontAwesome là một thư viện biểu tượng vector phổ biến mà có thể sử dụng để thêm biểu tượng vào trang web của mình.

- `library.add(faEdit);`: Thêm biểu tượng "faEdit" vào thư viện FontAwesome để sau đó có thể sử dụng nó trong các thành phần khác của ứng dụng.

# 3.2.3. Component `App`:

- `class App extends Component { ... }`: Đây là một component React chính của ứng dụng. Nó được xây dựng trên cơ sở của lớp `Component` từ React.

- Trong phần `render()`, `App` component trả về một cấu trúc giao diện bao gồm `Router` (cho định tuyến), `Navbar` (thanh điều hướng), `Switch` (để chuyển đổi giữa các routes), `Route` (định nghĩa route cho trang Home), và `Footer` (chân trang).

- `<Router>`: Đây là thành phần gốc cho định tuyến của ứng dụng. Nó bao quanh tất cả các thành phần liên quan đến định tuyến.

- `<Switch>`: Thành phần này sử dụng để đảm bảo chỉ một route duy nhất được hiển thị tại một thời điểm.

- `<Route>`: Định nghĩa một route cho trang chính (path="/") và liên kết nó với component `Home`. Điều này có nghĩa là khi route là "/" (trang chính), component `Home` sẽ được hiển thị.

- `<Navbar>` và `<Footer>`: Đây là các thành phần giao diện, như thanh điều hướng và chân trang, mà đã import ở trước đó và đưa vào trong `Router`.

# 3.3. Tạo các ví dụ đơn giản như hiển thị thông báo khi nút được nhấn.

Trong `serviceWorker.js`, bọn em đã sử dụng JavaScript để đăng ký một service worker và xử lý các sự kiện liên quan đến nó.

function registerValidSW(swUrl, config) {

  navigator.serviceWorker

    .register(swUrl)

    .then(registration => {

      registration.onupdatefound = () => {

        const installingWorker = registration.installing;

        if (installingWorker == null) {

          return;

        }

        installingWorker.onstatechange = () => {

          if (installingWorker.state === 'installed') {

            if (navigator.serviceWorker.controller) {

              // At this point, the updated precached content has been fetched,

              // but the previous service worker will still serve the older

              // content until all client tabs are closed.

              console.log(

                'New content is available and will be used when all ' +

                  'tabs for this page are closed. See https://bit.ly/CRA-PWA.'

              );

              // Execute callback

              if (config && config.onUpdate) {

                config.onUpdate(registration);

              }

            } else {

              // At this point, everything has been precached.

              // It's the perfect time to display a

              // "Content is cached for offline use." message.

              console.log('Content is cached for offline use.');

              // Execute callback

              if (config && config.onSuccess) {

                config.onSuccess(registration);

              }

            }

          }

        };

      };

    })

    .catch(error => {

      console.error('Error during service worker registration:', error);

    });

}

# 3.3.1. `navigator.serviceWorker.register(swUrl)`:

- Dòng này đăng ký một Service Worker bằng cách sử dụng `swUrl` (URL của tệp Service Worker) và trả về một Promise (hứa hẹn).

# 3.3.2. `.then(registration => { ... })`:

- Khi Service Worker đã được đăng ký thành công, Promise sẽ được giải quyết (resolve) và truyền vào hàm callback với biến `registration` đại diện cho đối tượng đăng ký (registration object) của Service Worker.

# 3.3.3. `registration.onupdatefound = () => { ... }`:

- Khi sự kiện "updatefound" xảy ra trong quá trình cài đặt Service Worker (khi có một phiên bản mới của Service Worker được tải về), một hàm callback sẽ được thực thi.

# 3.3.4. `const installingWorker = registration.installing;`:

- Biến `installingWorker` chứa đối tượng Service Worker đang được cài đặt.

# 3.3.5. `if (installingWorker == null) { return; }`:

- Kiểm tra nếu `installingWorker` không tồn tại hoặc bằng `null`, thì thoát khỏi hàm.

# 3.3.6. `installingWorker.onstatechange = () => { ... }`:

- Khi trạng thái của Service Worker đang được cài đặt thay đổi, hàm callback này sẽ được thực thi.

# 3.3.7. `if (installingWorker.state === 'installed') { ... }`:

- Kiểm tra nếu trạng thái của Service Worker đã được cài đặt (installed), thì tiếp tục kiểm tra các điều kiện sau đây:

# 3.3.8. `if (navigator.serviceWorker.controller) { ... }`:

- Kiểm tra xem trình duyệt đã có một Service Worker chạy (controller) hay chưa. Nếu đã có, có nghĩa là có một phiên bản trước đó của Service Worker đang chạy.

# 3.3.9. `console.log('New content is available and will be used when all tabs for this page are closed...');`:

- In ra console một thông báo cho người dùng, thông báo rằng có nội dung mới đã sẵn sàng và sẽ được sử dụng khi tất cả các tab trang web cho trang này đều được đóng lại. Đây là một thông báo cho người dùng biết rằng nội dung đã được cập nhật và sẽ có sẵn khi họ tải lại trang.

# 3.3.10. `config.onUpdate(registration);`:

- Nếu có một hàm callback `onUpdate` được định nghĩa trong đối tượng `config`, thì nó sẽ được gọi và truyền vào đối tượng đăng ký (`registration`) để thực hiện các hành động bổ sung khi có phiên bản mới của Service Worker.

# 3.3.11. Nếu không có trình duyệt Service Worker chạy:

- Nếu không có Service Worker nào đang chạy (trình duyệt chưa có Service Worker), thì sẽ thấy thông báo "Content is cached for offline use" (Nội dung đã được lưu trữ để sử dụng offline) trong console.

# 3.3.12. `config.onSuccess(registration);`:

- Nếu có một hàm callback `onSuccess` được định nghĩa trong đối tượng `config`, thì nó sẽ được gọi và truyền vào đối tượng đăng ký (`registration`) để thực hiện các hành động bổ sung khi Service Worker đã được cài đặt thành công.

# 3.3.13. `.catch(error => { ... })`:

- Nếu có lỗi trong quá trình đăng ký Service Worker, lỗi sẽ được bắt và xử lý trong phần này bằng cách in thông báo lỗi ra console.

# 4. HTML Forms và Input Elements

# 4.1. Cách tạo biểu mẫu HTML đơn giản với các phần tử nhập liệu như ô văn bản, ô chọn, và nút gửi

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico" />

    <meta

      name="viewport"

      content="width=device-width, initial-scale=1, shrink-to-fit=no"

    />

    <meta name="theme-color" content="#000000" />

    <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

    <title>Musician App</title>

  </head>

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>

</html>

# 4.1.1. `<!DOCTYPE html>`

Đây là khai báo xác định kiểu và phiên bản của tài liệu là HTML5. Nó cần phải nằm ở đầu của tài liệu HTML để chỉ ra kiểu tài liệu.

# 4.1.2. `<html lang="en">`

Đây là thẻ mở cho tài liệu HTML. Thuộc tính `lang` chỉ định ngôn ngữ của tài liệu là tiếng Anh.

# 4.1.3. `<head>`

Phần này chứa thông tin về tài liệu web, chẳng hạn như mã hóa ký tự, tiêu đề và liên kết đến các tài nguyên bên ngoài.

- `<meta charset="utf-8"/>`: Thẻ meta này chỉ định mã hóa ký tự cho tài liệu, trong trường hợp này là UTF-8, một mã hóa phổ biến được sử dụng để xử lý các ký tự và biểu tượng khác nhau.

- `<link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico"/>`: Dòng này chỉ định biểu tượng ngắn gọn (favicon) cho trang web, thường xuất hiện trong tab của trình duyệt hoặc thanh đánh dấu. `%PUBLIC\_URL%` là một giá trị địa chỉ URL tạm thời có thể được thay thế bằng URL thực tế trong quá trình xây dựng trang web.

- `<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"/>`: Thẻ meta này đặt các thuộc tính viewport cho thiết kế web đáp ứng. Nó đảm bảo rằng trang web sẽ thích nghi với các kích thước màn hình và thiết bị khác nhau.

- `<meta name="theme-color" content="#000000"/>`: Thẻ meta này định nghĩa màu chủ đề cho trang web.

# 4.2. Tạo kiểu cho biểu mẫu bằng CSS

.App {

  text-align: center;

}

.App-logo {

  animation: App-logo-spin infinite 20s linear;

  height: 40vmin;

  pointer-events: none;

}

.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-link {

  color: #61dafb;

}

.navbar-item img {

  max-height: 3.3em;

}

.musician-card {

  width: 30%;

  float: left;

  margin-right: 1em;

}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

# 4.2.1. `.App`

Đây là một lớp CSS được áp dụng cho các phần tử có lớp `App` trong HTML. Trong trường hợp này, nó đặt căn giữa nội dung của các phần tử có lớp này theo chiều ngang.

# 4.2.2. `.App-logo`

Lớp này được áp dụng cho các phần tử có lớp `App-logo` trong HTML. Nó tạo một hiệu ứng quay vòng vô hạn cho các phần tử này trong suốt 20 giây và đặt chiều cao là 40% của kích thước viewport (40vmin). Ngoài ra, `pointer-events: none;` đảm bảo rằng không có sự tương tác nào với các phần tử này.

# 4.2.3. `.App-header`

Lớp này được áp dụng cho các phần tử có lớp `App-header` trong HTML. Nó định dạng phần header của trang web bằng cách đặt màu nền (#282c34), chiều cao tối thiểu (min-height) là 100% chiều cao của viewport (100vh), và căn giữa nội dung bên trong header. Các thuộc tính font-size và color đặt kiểu chữ và màu sắc cho nội dung trong header.

# 4.2.4. `.App-link`

Lớp này được áp dụng cho các phần tử có lớp `App-link` trong HTML. Nó đặt màu chữ cho các liên kết có lớp này thành màu xanh dương (#61dafb).

# 4.2.5. `.navbar-item img`

Lớp này được áp dụng cho hình ảnh bên trong các phần tử có lớp `navbar-item`. Nó giới hạn chiều cao tối đa của hình ảnh là 3.3em.

# 4.2.6. `.musician-card`

Lớp này định dạng các thẻ có lớp `musician-card`. Nó đặt kích thước chiều rộng (width) là 30%, căn lề sang trái (float: left), và một khoảng cách là 1em phía bên phải của các thẻ này.

# 4.2.7. `@keyframes App-logo-spin`

Đây là một khai báo keyframes để tạo hiệu ứng quay vòng cho phần tử có lớp `App-logo`. Hiệu ứng này xoay từ góc 0 độ đến 360 độ, tạo ra một vòng tròn đầy đủ khi được kích hoạt.

# 4.3. Sử dụng JavaScript để xử lý dữ liệu từ biểu mẫu

Trong biểu mẫu HTML, JavaScript có thể được sử dụng để xử lý dữ liệu khi người dùng điền vào biểu mẫu và kiểm tra tính hợp lệ của dữ liệu nhập.

import React, { Component } from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import './App.css';

import Navbar from './components/Navbar';

import Home from './components/Home';

import Footer from './components/Footer';

import { library } from '@fortawesome/fontawesome-svg-core';

import { faEdit } from '@fortawesome/free-solid-svg-icons'

library.add(faEdit);

class App extends Component {

  render() {

    return (

      <div className="App">

        <Router>

          <div>

            <Navbar />

            <Switch>

              <Route exact path="/" component={Home} />

            </Switch>

            <Footer />

          </div>

        </Router>

      </div>

    );

  }

}

export default App;

# 4.3.1. Import các thư viện và thành phần

- `import React, { Component } from 'react';`: Dòng này nhập React và Component từ thư viện React. React được sử dụng để tạo các thành phần và giao diện người dùng, trong khi Component là một lớp cơ sở để tạo các thành phần tùy chỉnh.

- `import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';`: Dòng này nhập các thành phần liên quan đến định tuyến (routing) trong React. `BrowserRouter` là một thành phần để quản lý định tuyến trong ứng dụng. `Route` được sử dụng để xác định các tuyến đường và thành phần liên quan. `Switch` giúp chọn ra tuyến đường duy nhất được khớp với URL hiện tại.

- `import './App.css';`: Dòng này nhập một tệp CSS có tên 'App.css' để tùy chỉnh giao diện ứng dụng.

- `import Navbar from './components/Navbar';`: Dòng này nhập một thành phần có tên là `Navbar` từ một tệp khác ('./components/Navbar.js') để sử dụng trong ứng dụng.

- `import Home from './components/Home';`: Tương tự như trên, dòng này nhập một thành phần có tên là `Home` từ một tệp khác ('./components/Home.js').

- `import Footer from './components/Footer';`: Dòng này nhập một thành phần có tên là `Footer` từ một tệp khác ('./components/Footer.js').

- `import { library } from '@fortawesome/fontawesome-svg-core';` và `import { faEdit } from '@fortawesome/free-solid-svg-icons'`: Đây là việc nhập thư viện Font Awesome để sử dụng biểu tượng faEdit trong ứng dụng. Thư viện này giúp quản lý và sử dụng các biểu tượng vector trong ứng dụng React.

# 4.3.2. `library.add(faEdit);`

Dòng này thêm biểu tượng `faEdit` vào thư viện Font Awesome để có thể sử dụng trong ứng dụng.

# 4.3.3. `class App extends Component { ... }`

Đây là một thành phần React tùy chỉnh có tên là `App`. Thành phần này được mở rộng từ `Component`, nghĩa là nó kế thừa các tính năng cơ bản của một thành phần React.

# 4.3.4. `render() { ... }`

Đây là phương thức `render()` của thành phần `App`. Phương thức này trả về giao diện của thành phần.

# 4.3.5. Trong `render()`,có một giao diện HTML và React được sử dụng để định tuyến các trang web

- `<div className="App">...</div>`: Đây là một phần tử `div` có lớp CSS là "App". Nó là phần gốc của ứng dụng.

- `<Router>...</Router>`: Thành phần `Router` được sử dụng để quản lý định tuyến. Tất cả các tuyến đường và giao diện của ứng dụng nằm trong `<Router>`.

- `<Navbar />`: Thành phần `Navbar` được sử dụng để hiển thị thanh điều hướng trên trang web.

- `<Switch>...</Switch>`: Thành phần `Switch` quản lý các tuyến đường và chọn ra tuyến đường duy nhất khớp với URL.

- `<Route exact path="/" component={Home} />`: Đây là một tuyến đường. Nó xác định rằng khi URL trùng khớp với "/" (đường dẫn gốc), thành phần `Home` sẽ được hiển thị.

- `<Footer />`: Thành phần `Footer` được sử dụng để hiển thị phần chân trang của trang web.

# 4.3.6. `export default App;`

Dòng này xuất thành phần `App` để có thể sử dụng nó trong các tệp JavaScript khác.

# 5. Quy trình phát triển web

# 5.1. Giới thiệu về quy trình phát triển web từ lập kế hoạch đến triển khai trang web.

Trong dự án của bọn em, quy trình phát triển web bắt đầu từ việc xây dựng các thành phần front-end và back-end của ứng dụng web. Dưới đây là một số yếu tố quan trọng trong quy trình phát triển:

- Bọn em đã bắt đầu bằng việc tạo một trang HTML cơ bản (index.html) để tạo giao diện người dùng.

- Bọn em đã sử dụng CSS để thiết kế giao diện và định dạng các phần tử trên trang web.

- React được sử dụng để xây dựng ứng dụng web, và đã tạo các component như Navbar, Home, và Footer.

- Một số các thư viện bên ngoài như FontAwesome cũng đã được tích hợp vào ứng dụng .

- Bọn em đã sử dụng Express.js để xây dựng back-end của ứng dụng và cung cấp các API RESTful để quản lý dữ liệu.

# 5.2. Xác định các bước quan trọng như phân tích yêu cầu, thiết kế giao diện, phát triển, kiểm thử, và triển khai.

Quy trình phát triển web đã tuân theo các bước quan trọng như sau:

- Phân tích yêu cầu: Trước khi bắt đầu phát triển, bọn em đã phân tích yêu cầu của dự án để xác định chức năng và giao diện người dùng cần thiết.

- Thiết kế giao diện: Sau khi hiểu rõ yêu cầu, thiết kế giao diện người dùng bằng cách sử dụng HTML và CSS để tạo trang web với các thành phần như biểu mẫu, navbar, và footer.

- Phát triển: đã sử dụng React và Express.js để phát triển cả phía máy khách và phía máy chủ của ứng dụng. Các component React đã được xây dựng để tạo trải nghiệm người dùng tốt.

- Kiểm thử: Sau khi hoàn thành phát triển, đã tiến hành kiểm thử để đảm bảo rằng ứng dụng hoạt động đúng và không có lỗi. Có các bài kiểm tra (test cases) sử dụng thư viện như `supertest` để kiểm tra tính năng của các API RESTful.

- Triển khai: Trong trường hợp này, có thể thấy một số cấu hình liên quan đến triển khai và sử dụng Service Worker để làm cho ứng dụng có khả năng hoạt động offline.

# 5.3. Hướng dẫn về vai trò và trách nhiệm của từng thành viên trong nhóm, bao gồm quản lý dự án và phát triển.

- Quản lý dự án: Người quản lý dự án (Project Manager) có nhiệm vụ lập kế hoạch, theo dõi tiến độ, và đảm bảo rằng dự án được hoàn thành đúng hạn. Họ cũng quản lý tài nguyên và xác định ưu tiên.

- Phát triển front-end: Nhóm phát triển front-end chịu trách nhiệm xây dựng giao diện người dùng, tạo các component, và xử lý trải nghiệm người dùng trên trình duyệt.

- Phát triển back-end: Nhóm phát triển back-end xây dựng server, xử lý dữ liệu, và triển khai các API cho phía máy khách sử dụng.

- Kiểm thử: Nhóm kiểm thử kiểm tra ứng dụng để đảm bảo tính đúng đắn và hiệu suất. Họ phát hiện và báo cáo lỗi, sau đó đảm bảo rằng các lỗi này được sửa.

- Triển khai và quản lí hệ thống : Nhóm triển khai và quản lý hệ thống đảm bảo rằng ứng dụng được triển khai một cách đáng tin cậy trên môi trường sản xuất. Họ cũng quản lý cơ sở hạ tầng và bảo trì hệ thống.