USCS
UNIVERSIDADE DE SÃO CAETANO DO SUL

Linguagem de Programação

Prof. Renato Carioca Duarte

USCS
UNIVERSIDADE DE SÃO CAETANO DO SUL

Unidade 10

API Rest
FRONTEND

Frontend

- Frontend refere-se à parte da aplicação que os usuários veem e interagem diretamente.
- Em desenvolvimento web e móvel, o frontend é tudo o que é renderizado nos navegadores ou nas telas dos dispositivos dos usuários.
- Basicamente, é a interface gráfica da aplicação.

Tecnologias Frontend

As tecnologias frontend são usadas para criar a interface do usuário em aplicações web. Isso inclui:

- **HTML** (Linguagem de Marcação de Hipertexto) para estruturar o conteúdo.
- **CSS** (Folhas de Estilo em Cascata) para estilizar e formatar a aparência.
- **JavaScript** para adicionar interatividade e dinamismo.

- Frameworks e bibliotecas como React, Angular, Vue.js, Bootstrap, entre outros, são frequentemente utilizados para facilitar o desenvolvimento frontend e tornar as interfaces mais interativas e responsivas.

Frontend

- Um design responsivo garante que a interface do usuário se ajuste adequadamente a diferentes tamanhos de tela e dispositivos, como desktops, tablets e smartphones.

- O frontend muitas vezes se comunica com o backend (ou servidor) para solicitar ou enviar dados.
- Esta comunicação é geralmente feita usando APIs (Application Programming Interfaces), frequentemente no formato REST ou GraphQL.

Frontend

- A otimização do frontend é essencial para garantir tempos de carregamento rápidos e uma boa experiência do usuário.
- Isso pode incluir a minimização de arquivos CSS e JavaScript, otimização de imagens, entre outras técnicas.

- UX (Experiência do Usuário) refere-se à experiência global do usuário ao interagir com a aplicação, enquanto UI (Interface do Usuário) foca especificamente no design e layout visual da aplicação.
- Ambos são aspectos cruciais do desenvolvimento frontend.

Tipos de Frontend

Aplicações Web Tradicionais:

- Baseiam-se em renderização no lado do servidor, onde cada interação ou pedido geralmente resulta em uma nova página enviada pelo servidor.
- Exemplos incluem aplicações desenvolvidas com PHP, ASP.NET, Ruby on Rails, entre outros.

Single Page Applications (SPAs):

- SPAs são aplicações web que carregam uma única página e atualizam dinamicamente conforme o usuário interage.
- SPAs são comuns com frameworks como React, Angular e Vue.js.

Tipos de Frontend

Aplicações Móveis Nativas:

- São desenvolvidas para rodar diretamente em dispositivos móveis específicos, como Android ou iOS.
- Elas são escritas em linguagens específicas para cada plataforma (Java/Kotlin para Android, Objective-C/Swift para iOS).

Aplicações Móveis Híbridas:

- Essas aplicações são desenvolvidas usando tecnologias da web (HTML, CSS e JavaScript) e depois empacotadas dentro de um "invólucro" nativo que permite que elas sejam instaladas e executadas como uma aplicação nativa.
- Exemplos incluem aplicações construídas com Cordova, PhoneGap e Ionic.

Tipos de Frontend

Aplicações de Desktop:

- Embora tradicionalmente as aplicações de desktop não sejam consideradas "frontend" no sentido web, há ferramentas e frameworks, como Electron, que permitem que os desenvolvedores criem aplicações de desktop usando tecnologias da web.

Outros tipos:

- Aplicações Móveis Baseadas em Web
- Progressive Web Apps
- WebGL e WebVR
- etc

Single Page Applications (SPAs)

- Uma Single Page Application (SPA) é um tipo de aplicação web ou site que interage com o usuário dinamicamente, reescrevendo a página atual, em vez de carregar páginas inteiras a partir do servidor.
- Isso significa que, após a página web inicial ser carregada, a SPA só atualiza as partes que são necessárias em resposta às ações do usuário.

- SPAs oferecem uma experiência mais fluida para os usuários, semelhante a aplicativos desktop ou móveis, porque a maioria das interações ocorre no lado do cliente, sem a necessidade de contínuos pedidos de páginas completas ao servidor.

Single Page Applications (SPAs)

- Embora a primeira carga possa ser um pouco mais lenta, porque a lógica necessária (HTML, CSS e JavaScript) é carregada de uma vez, as interações subsequentes são geralmente muito mais rápidas, pois somente os dados (geralmente em formato JSON) são transmitidos entre o cliente e o servidor.

- Muitas SPAs são construídas usando frameworks e bibliotecas JavaScript populares, como React, Angular e Vue.js, que fornecem ferramentas e estruturas para construir aplicações robustas e eficientes.

Elementos HTML de uma SPA

Estrutura Básica:

<html>: Elemento raiz que contém todo o HTML.

<head>: Contém metadados e links para CSS e JavaScript.

<body>: Contém o conteúdo visível da página.

Elementos HTML de uma SPA

Interface do Usuário:

<form>: Utilizado para criar, editar, e enviar dados para a API.

<input>: Campo para entrada de dados do usuário.

<button>: Botões para realizar ações (submeter /cancelar).

<table>: Usado para exibir dados em formato tabular.

<thead> e <tbody>: Seções de cabeçalho e corpo da tabela

 ou <section>: Containers para agrupar elementos e aplicar estilos CSS ou manipulação via JavaScript.

<h1>, <h2>, etc.: Títulos e subtítulos para seções da página.

Elementos HTML de uma SPA

Navegação (Opcional para SPAs):

<nav>: Container para elementos de navegação.

<a>: Links para navegar entre diferentes seções/views da SPA.

Elementos HTML de uma SPA

Multimídia (Se Necessário):

<img>: Para exibir imagens.

<video>: Para reproduzir vídeos.

Elementos HTML de uma SPA

Scripts:

<script>: Contém ou linka código JavaScript, que é essencial para a funcionalidade de SPA e interação com a API REST.


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF8">
    <meta name="viewport" content="width=devicewidth, initialscale=1.0">
    <title>Exemplo Estrutura Single Page Application</title>
    <!--CSS e JavaScript -->
</head>
<body>
    <nav>
        <!-- Links para navegação -->
    </nav>
        <form id="dataForm">
            <input type="text" id="dataInput">
            <button type="submit">Submit</button>
        </form>
        <table>
            <thead>
                <!-- Cabeçalho da Tabela -->
            </thead>
            <tbody>
                <!-- Dados da Tabela -->
            </tbody>
        </table>
    <script>
        // Código JavaScript para CRUD e SPA
    </script>
</body>
</html>
```

Pontos Importantes

- JavaScript e AJAX/Fetch API: Usado para realizar requisições HTTP sem recarregar a página (essencial para SPAs).
- Manipulação DOM: JavaScript também é usado para manipular o DOM, atualizando dinamicamente os elementos da página de acordo com as interações do usuário e respostas da API.
- CSS: Para estilizar os elementos HTML e melhorar a experiência do usuário.

Ferramentas/Framework

- Vue.js, React, ou Angular: Frameworks/libraries JavaScript populares para desenvolver SPAs eficientes e manuteníveis.
- Axios: Biblioteca JavaScript para realizar requisições HTTP, frequentemente usada com Vue.js ou React.
- Bootstrap ou Materialize: Frameworks CSS para estilizar e estruturar a UI de forma responsiva e com componentes préprontos.

API Fetch

- A função `fetch()` é uma API moderna do JavaScript para fazer requisições HTTP de forma assíncrona, ou seja, sem bloquear o fluxo principal de execução do código.
- Ela é uma alternativa mais recente e flexível ao `XMLHttpRequest`, que era a abordagem tradicional para fazer chamadas AJAX.

API Fetch

- Sintaxe básica

```
fetch(url, options)
```

- Entrada:
  - url é a URL do recurso que você deseja acessar.
  - options é um objeto opcional que permite configurar a requisição (como definir métodos, cabeçalhos, corpo, etc.).

- Retorno:
  - retorna uma Promise que resolve a resposta (objeto `Response`) da requisição.

Manipulação de Resposta - Exemplo

- Como o `fetch()` retorna uma Promise, você pode usar `.then()` para manipular a resposta ou `.catch()` para lidar com erros:

```
fetch(url)
  .then((response) => response.json()) // Transforma a resposta num objeto J
  .then((data) => console.log(data)) // Manipula o dado retornado.
  .catch((error) => console.error("Error:", error)); // Captura qualquer err
```

API Fetch - Exemplo

```
fetch("https://api.example.com/data", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    key1: "value1",
    key2: "value2",
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Error:", error));
```

Implementando Frontend API Rest

Implementando Frontend API Rest

- O arquivo index.html abaixo é um frontend api rest que funciona em conjunto com o backend api rest server1.js descrito ao final da unidade 9.

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

Implementando Frontend API Rest

```
<body>
  <h2>Cadastramento de Produtos</h2>

  <!-- Formulário de cadastro de produtos -->
  <form id="productForm">
    <div>
      <!-- Campo para inserção do ID do produto -->
      <label for="id">ID:</label>
      <input type="number" id="id" placeholder="ID" required />
    </div>
    <div>
      <!-- Campo para inserção do nome do produto -->
      <label for="nome">Nome:</label>
      <input type="text" id="nome" placeholder="Nome" required />
    </div>
    <!-- Botões de submit e cancelamento -->
    <button type="submit">Cadastrar</button>
    <button type="button" id="cancelBtn">Cancelar</button>
  </form>
```


```html
<!-- Tabela para exibição dos produtos cadastrados -->
<table>
  <thead>
    <tr>
      <!-- Cabeçalhos da tabela -->
      <th>ID</th>
      <th>Nome</th>
      <th>Ações</th>
    </tr>
  </thead>
  <tbody id="productsTable"></tbody>
  <!-- Linhas da tabela serão inseridas dinamicamente com JavaScript -->
</table>
```

```
<script>
  /* Declaração da lista de produtos e variável de produto atual para edição *
  let produtos = [];
  let produtoAtual = null;

  /* Função para exibir os produtos na tabela */
  function displayProducts() {
    const tbody = document.getElementById("productsTable");
    tbody.innerHTML = "";
    produtos.forEach((produto) => {
      const row = document.createElement("tr");
      row.innerHTML = `
        <td>${produto.id}</td>
        <td>${produto.nome}</td>
        <td>
            <button onclick="editProduct(${produto.id})">Alterar</button>
            <button onclick="deleteProduct(${produto.id})">Remover</button>
        </td>
        `;
      tbody.appendChild(row);
    });
  }
```


```
/* Função para permitir a edição de um produto específico */
function editProduct(id) {
  produtoAtual = id;
  const produto = produtos.find((p) => p.id === id);
  document.getElementById("id").value = produto.id;
  document.getElementById("nome").value = produto.nome;

  document.getElementById("cancelBtn").style.display = "inline-block";
}
```


```
/* Função para deletar um produto específico via API */
function deleteProduct(id) {
  fetch("http://localhost:3000/produtos/" + id, {
    method: "DELETE",
  })
    .then(() => {
      /* retira do array produtos o produto com p.id == id */
      produtos = produtos.filter((p) => p.id !== id);
      displayProducts();
    })
    .catch((error) => console.error("Error:", error));
}
```

USCS
UNIVERSIDADE DE SÃO CAETANO DO SUL

```
/* Fetch inicial para obter os produtos da API e exibi-los na tabela */
fetch("http://localhost:3000/produtos")
  .then((response) => response.json())
  .then((data) => {
    produtos = data;
    displayProducts();
  })
  .catch((error) => console.error("Error:", error));
```

```
/* Event listener para gerenciar o submit do formulário */
document
  .getElementById("productForm")
  .addEventListener("submit", function (event) {
    event.preventDefault();
    const produto = {
      id: parseInt(document.getElementById("id").value),
      nome: document.getElementById("nome").value,
    };
    if (produtoAtual) {
      fetch("http://localhost:3000/produtos/" + produtoAtual, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(produto),
      })
        .then((response) => response.json())
        .then((data) => {
          const index = produtos.findIndex((p) => p.id === produtoAtual);
          produtos[index] = data;
          displayProducts();
          document.getElementById("productForm").reset();
          produtoAtual = null;
          document.getElementById("cancelBtn").style.display = "none";
        })
        .catch((error) => console.error("Error:", error));
    } else {
```


```
  } else {
    fetch("http://localhost:3000/produtos", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(produto),
    })
      .then((response) => response.json())
      .then((data) => {
        produtos.push(data);
        displayProducts();
        document.getElementById("productForm").reset();
      })
      .catch((error) => console.error("Error:", error));
  }
});
```

```
    /* Event listener para gerenciar o botão de cancelamento de edição */
    document
      .getElementById("cancelBtn")
      .addEventListener("click", function () {
        produtoAtual = null;
        document.getElementById("productForm").reset();
        this.style.display = "none";
      });

  </script>
</body>
</html>
```

Dúvidas

??????