Teste PW

1. Esta imagem representa a arquitetura de um sistema web que contém várias tecnologias na seu ambiente, sendo essas:

O FrontEnd que é a interface gráfica do utilizador, normalmente utilizando o Html,Css e o JavaScript.

O BackEnd é a estrutura que possibilita a operação do sistema, é intermediário entre o frontEnd e a base de Dados

1. Protocolo Cantina:

Objetivo: Estabelecer regras e procedimentos para garantir a ordem e eficiência da cantina do IPVC.

1-Entrada

Os alunos deveram dirigir-se a cantina IPVC com o seu cartão de estudante.

2-Horários

A cantina abre as suas portas as 11:45h e tem o seu fecho ás 14:00h.

3-Marcação de refeição

A refeição deverá ser marcada no sistema até as 11h do mesmo dia.

4-Refeição

Cada aluno terá direito a 1 refeição completa, onde devera realizá-la dentro do estabelecimento

Os protocolos são essenciais para a organização e bom funcionamento de qualquer

1. Neste caso o getElement refere-se ao id de um objeto, sendo o id único, ou seja, não pode existir objetos com os mesmos id’s, já os getElements está a referir-se as tagName’s e as Classes, em que estas podem ser utilizadas por vários objetos

Por exemplo:

document.getElementById(1) -> Este irá procurar o objeto que o seu id seja 1.

document.getElementsByClassName(‘imagem’) -> este ira procurar todos os objetos que pertencem á class ‘imagem’

4-

JSON:

{

    "atores": [

      {

        "id": 1,

        "nome": "Leonardo DiCaprio",

        "idade": 48,

        "filmes": [

          {

            "id": 1,

            "titulo": "Inception",

            "ano": 2010

          },

          {

            "id": 2,

            "titulo": "Shutter Island",

            "ano": 2010

          }

        ]

      },

      {

        "id": 2,

        "nome": "Tom Cruise",

        "idade": 55,

        "filmes": [

          {

            "id": 3,

            "titulo": "Top Gun Maverick",

            "ano": 2022

          },

          {

            "id": 4,

            "titulo": "Missão Impossível",

            "ano": 2027

          }

        ]

      }

    ]

  }

XML:

<registo>

  <atores>

    <ator>

      <id>1</id>

      <nome>Leonardo DiCaprio</nome>

      <idade>48</idade>

      <filmes>

        <filme>

          <id>1</id>

          <titulo>Inception</titulo>

          <ano>2010</ano>

        </filme>

        <filme>

          <id>2</id>

          <titulo>Shutter Island</titulo>

          <ano>2010</ano>

        </filme>

      </filmes>

    </ator>

    <ator>

      <id>2</id>

      <nome>Tom Cruise</nome>

      <idade>55</idade>

      <filmes>

        <filme>

          <id>3</id>

          <titulo>Top Gun Maverick</titulo>

          <ano>2022</ano>

        </filme>

        <filme>

          <id>4</id>

          <titulo>Missão Impossível</titulo>

          <ano>2027</ano>

        </filme>

      </filmes>

    </ator>

  </atores>

</registo>

Comparando os resultados JSON e XML conseguimos observar que ambos os formatos representam a mesma estrutura de dados com sintaxe diferente, tendo vantagens o JSOn por ser mais fácil para leitura humana, enquanto o xml é mais bem estruturado.

Parte |||-

O <p> é utilizado para criar um novo parágrafo de texto em que adiciona espaçamento antes e depois do texto para criar uma separação entre o texto e o resto, já a tag <pre> é utilizada para exibir texto pré-formatado sendo este renderizado exatamente como está em html, ou seja deixando todos os espaços brancos, quebra linhas e qualquer tipo de formatação.

GITHUB-

Parte V|-

1-

  // products.js

const express = require('express');

const productsRouter = express.Router();

// Importar o controlador de produtos

const productsController = require('../controllers/products');

// Rota para obter todos os produtos

productsRouter.get('/', productsController.getAll);

// Rota para obter um produto por ID

productsRouter.get('/:id', productsController.getById);

// Rota para criar um novo produto

productsRouter.post('/', productsController.create);

// Rota para atualizar um produto existente

productsRouter.put('/:id', productsController.update);

// Rota para excluir um produto

productsRouter.delete('/:id', productsController.delete);

module.exports = productsRouter;

2.

const express = require('express');

Importa o módulo express,

const productsRouter = express.Router();

Cria um objeto Router para definir as rotas relacionadas aos produtos

const productsController = require('../controllers/products');

Importa o controlador de produtos (productsController)

productsRouter.get('/', productsController.getAll);

Define uma rota para obter todos os produtos

productsRouter.get('/:id', productsController.getById);

Define uma rota para obter um produto por ID.

productsRouter.post('/', productsController.create);

Define uma rota para criar um novo produto.

productsRouter.put('/:id', productsController.update);

Define uma rota para atualizar um produto existente

productsRouter.delete('/:id', productsController.delete);

Define uma rota para excluir um produto.

3.

{

"produtos": [

{

"id": 1,

"name": "Product A",

"price": 29.99,

"description": "."

},

{

"id": 2,

"name": "Product B",

"price": 19.99,

"description": "."

}

]

}

// productsController.js

const products = require('./products.json'); // Carrega os dados do ficheiro JSON

function getAll(req, res) {

  res.json(products);

}

module.exports = {

  getAll

};

// productsController.js

function getById(req, res) {

  const productId = parseInt(req.params.id);

  const product = products.find(p => p.id === productId);

  if (!product) {

    return res.status(404).json({ error: 'Product not found' });

  }

  res.json(product);

}

module.exports = {

  getAll,

  getById

};

// productsController.js

function create(req, res) {

  const newProduct = req.body; // Assume que o corpo da requisição contém os dados do novo produto

  newProduct.id = products.length + 1; // Gera um novo ID

  products.push(newProduct);

  res.status(201).json(newProduct);

}

module.exports = {

  getAll,

  getById,

  create

};

// productsController.js

function update(req, res) {

  const productId = parseInt(req.params.id);

  const updatedProduct = req.body; // Assume que o corpo da requisição contém os dados atualizados

  const index = products.findIndex(p => p.id === productId);

  if (index === -1) {

    return res.status(404).json({ error: 'Product not found' });

  }

  products[index] = { ...products[index], ...updatedProduct };

  res.json(products[index]);

}

module.exports = {

  getAll,

  getById,

  create,

  update

};

// productsController.js

function remove(req, res) {

  const productId = parseInt(req.params.id);

  const index = products.findIndex(p => p.id === productId);

  if (index === -1) {

    return res.status(404).json({ error: 'Product not found' });

  }

  const deletedProduct = products.splice(index, 1)[0];

  res.json(deletedProduct);

}

module.exports = {

  getAll,

  getById,

  create,

  update,

  remove

};