Diogo Sousa nº26118

Prova B

Questão 1:

Este diagrama representa uma arquitetura de software para aplicações web e móveis que interagem com uma base de dados através de serviços web.

HTML5 e JavaScript: Representa a interface de utilizador do lado do cliente que utiliza HTML5 e JavaScript. Este front-end interage com o servidor web através de pedidos HTTP, enviando e recebendo dados em formato JSON.

Web Service: É o intermediário que recebe os pedidos da aplicação web e da aplicação móvel, processa estes pedidos e comunica-se com a base de dados para obter ou armazenar informação. Este serviço web pode estar desenvolvido em diferentes tecnologias, como PHP ou Node.js, e normalmente envia respostas em formato JSON.

Base de dados: No caso, MySQL é o sistema de gestão de bases de dados utilizado para armazenar os dados. O serviço web interage com esta base de dados para realizar operações CRUD (Create, Read, Update, Delete).

As interfaces de utilizador (tanto web como móvel) enviam pedidos HTTP ao serviço web. Estes pedidos contêm dados em formato JSON.

O serviço web recebe estes pedidos, processa a informação e realiza as operações necessárias na base de dados (por exemplo, consulta, inserção, atualização, eliminação).

Depois de interagir com a base de dados, o serviço web envia de volta a resposta em formato JSON para as interfaces de utilizador, atualizando a informação mostrada ao utilizador.

Questão 2:

Os protocolos são conjuntos de regras que permitem a comunicação entre dispositivos em uma rede. Definem como os dados devem ser formatados, transmitidos e recebidos para garantir uma melhor comunicação entre os sistemas. Na programação web, os protocolos são fundamentais para o funcionamento da internet e da comunicação entre servidores e clientes.

Exemplo:

HTTP (HyperText Transfer Protocol):

Serviço: Utilizado para a comunicação entre navegadores e servidores web.

Exemplo: Quando se introduz um URL no navegador, este faz uma solicitação HTTP ao servidor onde o site está hospedado. O servidor responde com o conteúdo da página web, que é então renderizada no navegador.

Protocolo cantina ESTG:

```
class CantinaIPVC:
    def __init__(self):
        self.reservations = {}
        self.balance = {}

    def authenticate_student(self, student_id):
        # Aqui haveria um processo de autenticação, como leitura do
cartão de estudante
        print(f"Student {student_id} authenticated.")

    def make_reservation(self, student_id, menu_choice):
        if student_id not in self.reservations:
            self.reservations[student_id] = menu_choice
            print(f"Reservation made for student {student_id}:
{menu_choice}")
        else:
            print(f"Student {student_id} already has a reservation.")

    def process_payment(self, student_id, amount):
        if student_id in self.balance:
            if self.balance[student_id] >= amount:
                self.balance[student_id] -= amount
                print(f"Payment of {amount} processed for student
{student_id}. Remaining balance: {self.balance[student_id]}")
            else:
                print(f"Insufficient balance for student {student_id}.")
        else:
            print(f"No balance available for student {student_id}. Please
load funds.")

    def add_funds(self, student_id, amount):
        if student_id in self.balance:
            self.balance[student_id] += amount
        else:
            self.balance[student_id] = amount
        print(f"Funds added for student {student_id}. New balance:
{self.balance[student_id]}")

# Exemplo de uso
cantina = CantinaIPVC()
cantina.authenticate_student("12345")
cantina.add_funds("12345", 20)
cantina.make_reservation("12345", "Menu A")
cantina.process_payment("12345", 5)
```

Parte II

Questão 1:

A diferença encontra-se na quantidade de elementos que cada um retorna

getElementById retorna um único elemento com o ID especificado.

getElementsByTagName retorna uma coleção de elementos com o nome da tag especificado.

getElementsByClassName retorna uma coleção de elementos com a classe especificada.

getElementById**:** Este método retorna o elemento que tem o atributo `id` correspondente ao valor especificado. IDs devem ser únicos dentro de um documento HTML, portanto, este método sempre retorna um único elemento.

```
var headerElement = document.getElementById("header");
console.log(headerElement.textContent);
```

getElementsByTagName**:** Este método retorna uma coleção (HTMLCollection) de todos os elementos com o nome da tag especificado. Como pode haver vários elementos com o mesmo nome de tag em um documento, este método retorna uma lista de elementos

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  console.log(paragraphs[i].textContent);
}
```

getElementsByClassName**:** Este método retorna uma coleção (HTMLCollection) de todos os elementos que têm a classe especificada. Como muitos elementos podem compartilhar a mesma classe, este método retorna uma lista de elementos.

```
var boxes = document.getElementsByClassName("box");
for (var i = 0; i < boxes.length; i++) {
  console.log(boxes[i].textContent);
}
```

Questão 2:

Ficheiro JSON:

```
[{"Ator":"Vin Diesel","Filme":"Fast and Furious"},
 {"Ator":"Tom Holland","Filme":"Spider Man"},
{"Ator":"Brad Pitt","Filme":"Fight Club"},
 {"Ator":"Robert Downey Jr","Filme":"Iron Man"},
 {"Ator":"Leonardo DiCaprio","Filme":"Titanic"},
 {"Ator": "Chris Evans","Filme":"Captain America"}]
```

Ficheiro XML:

```xml
<atores>
    <ator>
        <nome>Vin Diesel</nome>
        <filme>Fast and Furious</filme>
    </ator>
    <ator>
        <nome>Tom Holland</nome>
        <filme>Spider Man</filme>
    </ator>
    <ator>
        <nome>Brad Pitt</nome>
        <filme>Fight Club</filme>
    </ator>
    <ator>
        <nome>Robert Downey Jr</nome>
        <filme>Iron Man</filme>
    </ator>
    <ator>
        <nome>Leonardo DiCaprio</nome>
        <filme>Titanic</filme>
    </ator>
    <ator>
        <nome>Chris Evans</nome>
        <filme>Captain America</filme>
    </ator>
</atores>
```

Parte III

Questão 1:

<p> : É usado para parágrafos de texto normal. Ele adiciona espaço entre as linhas e geralmente é usado para texto que precisa ser formatado em parágrafos separados.

<pre>: É usado para texto pré-formatado, onde espaços em branco e quebras de linha são preservados exatamente como estão no código HTML. É útil para exibir texto que requer formatação específica, como código de programação ou mensagens de e-mail.

Questão 2:

<meta charset="utf-8"> diz ao navegador qual o conjunto de carateres que deve ser usado para interpretar e exibir o texto da página.

Basicamente, esta tag garante que o navegador interprete corretamente os carateres especiais, acentos e símbolos utilizados na página da web. Sem ela alguns dos carateres, podem não ser interpretados corretamente e consequentemente, causar erros na leitura ao utilizador ou desformatação da página.

Parte IV

Questão 1:

```
<!DOCTYPE html>
<html lang="pt">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Produtos de Rede</title>
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f2f2f2;
  }
  .container {
    width: 80%;
    margin: 50px auto;
  }
  table {
    width: 100%;
    border-collapse: collapse;
    border: 2px solid #333;
  }
  th, td {
    border: 2px solid #333;
    padding: 20px;
    text-align: center;
  }
  th {
    background-color: #333;
    color: #fff;
  }
  .brand-logo {
    max-width: 100px;
    max-height: 100px;
  }
  tbody tr:nth-child(odd) {
    background-color: #ff0000;
  }
  tbody tr:nth-child(even) {
    background-color: #33ff00;
```

```
  }
</style>
</head>
<body>
  <div class="container">
    <table>
      <thead>
        <tr>
          <th>Marca</th>
          <th>Logo</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Cisco</td>
          <td><img class="brand-logo" src="https://logodownload.org/wp-content/uploads/2014/04/cisco-logo-0.png" alt="Cisco"></td>
        </tr>
        <tr>
          <td>Juniper Networks</td>
          <td><img class="brand-logo" src="https://www.logo.wine/a/logo/Juniper_Networks/Juniper_Networks-Logo.wine.svg" alt="Juniper Networks"></td>
        </tr>
        <tr>
          <td>Palo Alto</td>
          <td><img class="brand-logo" src="https://getlogovector.com/wp-content/uploads/2020/09/palo-alto-networks-inc-logo-vector.png" alt="Palo Alto"></td>
        </tr>
        <tr>
          <td>Arista Networks</td>
          <td><img class="brand-logo" src="https://www.tvarsolutions.com/wp-content/uploads/2019/04/arista-1024x262-1.png" alt="Arista Networks"></td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

Parte IV

Questão 2:

Ficheiro routes/products.js

1ª linha cria um router Express para lidar com rotas de produtos.

2ª linha importa o controlador para lidar com a lógica dos produtos.

3ª linha importa o middleware de autenticação.

4ª linha exporta o router de produtos para uso em outros lugares da aplicação.

Ficheiro controllers/products.js:

As duas primeiras linhas importam módulos de outros arquivos.

As próximas cinco linhas exportam funções assíncronas com diferentes propósitos (listar todos os produtos, obter um produto por ID, criar um novo produto, atualizar um produto e excluir um produto). Essas funções são disponibilizadas para uso em outros arquivos.

Ficheiro JSON


```
[
    {
      "id": 1,
      "name": "Product A",
      "price": 10.99,
      "description": "Description for Product A"
    },
    {
      "id": 2,
      "name": "Product B",
      "price": 19.99,
      "description": "Description for Product B"
    },
    {
      "id": 3,
      "name": "Product C",
      "price": 15.49,
      "description": "Description for Product C"
    }
  ]
```