USCS
UNIVERSIDADE DE SÃO CAETANO DO SUL

Linguagem de Programação

Prof. Renato Carioca Duarte

USCS
UNIVERSIDADE DE SÃO CAETANO DO SUL

Unidade 09

Backend API Rest

Backend

- As linguagens de programação mais comuns para o desenvolvimento backend incluem **JavaScript** (**Node.js**), Python, Ruby, Java, PHP, C#, Go, entre outras.
- Existem vários frameworks projetados para facilitar o desenvolvimento backend, como:
  - **Express** (para Node.js),
  - Django (para Python),
  - Ruby on Rails (para Ruby),
  - Spring Boot (para Java) e muitos outros.

Backend

- O código do backend geralmente é executado em **servidores**, que podem ser locais, em data centers ou na nuvem.
- Servidores populares incluem Apache, Nginx, e servidores de aplicativos como Tomcat.
- O backend frequentemente interage com **bancos de dados** para armazenar, recuperar e manipular informações.
- Pode ser um banco de dados relacional (como **MySQL**, PostgreSQL, SQL Server), um banco de dados NoSQL (como MongoDB, Cassandra), ou outros tipos de sistemas de armazenamento de dados.
- **APIs** permitem a comunicação entre o frontend e o backend, ou entre diferentes partes de um sistema. Elas podem adotar padrões como REST ou GraphQL.

API

- API (Application Programming Interface) é um conjunto de regras e protocolos que permitem a comunicação e interação entre diferentes softwares.
- Ela define as maneiras pelas quais os componentes de software devem interagir, facilitando a troca de informações e funcionalidades entre diferentes sistemas, aplicativos ou serviços.

API

- As APIs podem ser comparadas a um contrato entre dois programas: um programa define como suas funções, classes e métodos podem ser acessados e utilizados por outros programas.
- Isso permite que desenvolvedores utilizem os recursos oferecidos por um software sem precisar entender todos os detalhes internos do funcionamento desse software.

Vantagens de API

- As APIs permitem que desenvolvedores aproveitem as capacidades de outros softwares sem precisar reescrever todo o código do zero.
- Isso promove a reutilização de código, a integração de sistemas, a criação de aplicativos mais eficientes e a expansão das funcionalidades de um aplicativo através da conexão com outros serviços.

API REST

- Uma **API REST (Representational State Transfer)** é um estilo arquitetural para criar serviços web que segue princípios e convenções específicas.
- Ela é uma abordagem popular para construir APIs da web que permitem a comunicação e interação entre diferentes sistemas de uma maneira padronizada, eficiente e escalável.

API REST

- As APIs REST seguem um conjunto de princípios que incluem:

1. Estado Representacional

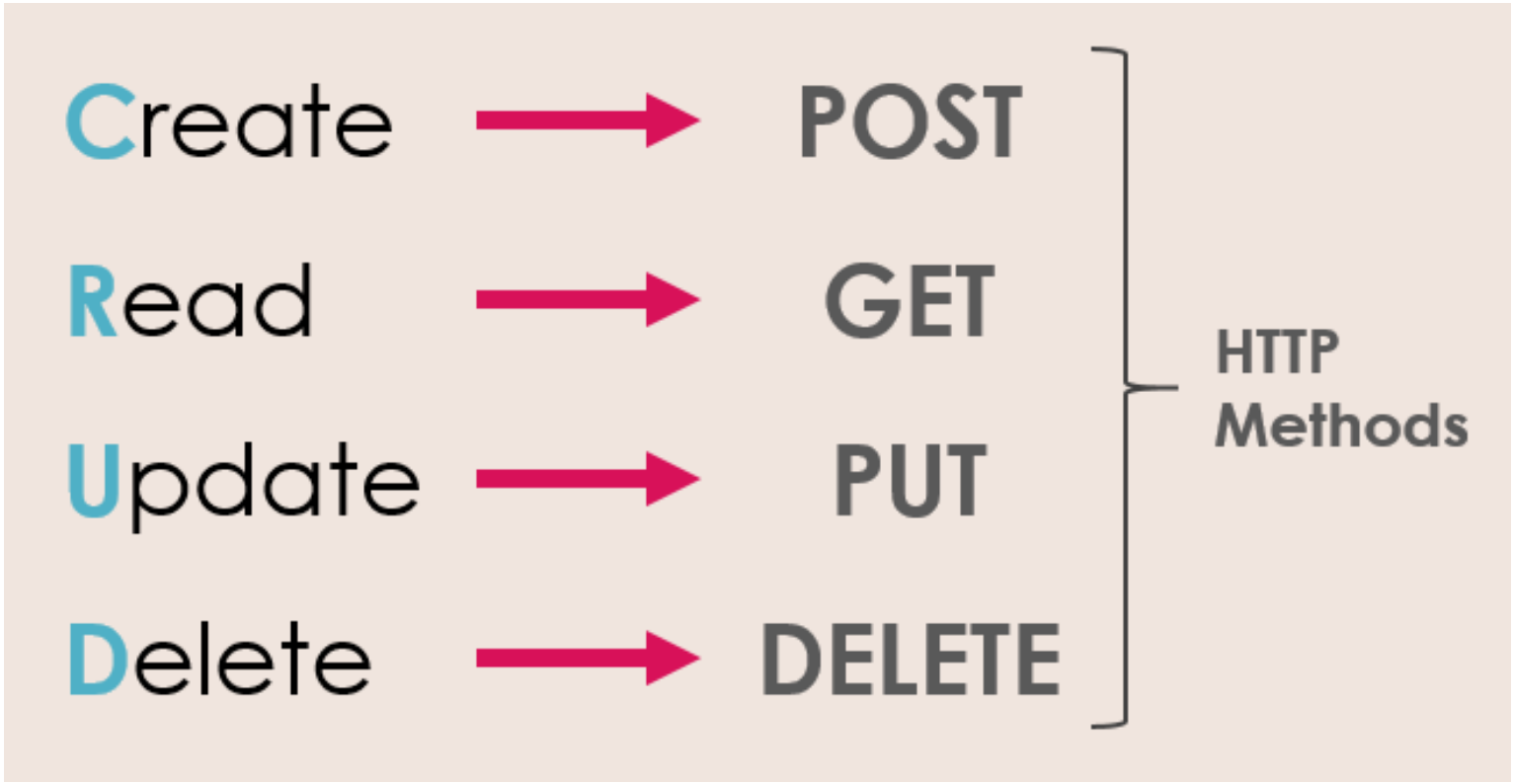
- Os recursos (dados) são representados em um formato específico, como JSON ou XML, e são identificados por URLs únicas.
- Cada URL representa um recurso específico.

API REST

2. Operações HTTP:

- As operações CRUD (Create, Read, Update, Delete) são mapeadas para os métodos HTTP:
  - GET: Obter informações.
  - POST: Criar novos recursos.
  - PUT: Atualizar recursos existentes.
  - DELETE: Remover recursos

API REST



API REST

3. Sem Estado (Stateless):

- Cada requisição para o servidor contém todas as informações necessárias para que o servidor entenda e processe a requisição.
- O servidor não mantém informações de estado sobre a sessão do cliente.

4. Cache:

- As respostas da API podem ser armazenadas em cache para melhorar o desempenho.

API REST

5. Interface Uniforme:

- A API deve seguir uma interface uniforme para acessar e manipular recursos, o que facilita a compreensão e o uso da API.

6. Sistema Cliente-Servidor:

- A API é projetada com uma separação clara entre o cliente (aplicativo que consome a API) e o servidor (aplicativo que fornece a API).

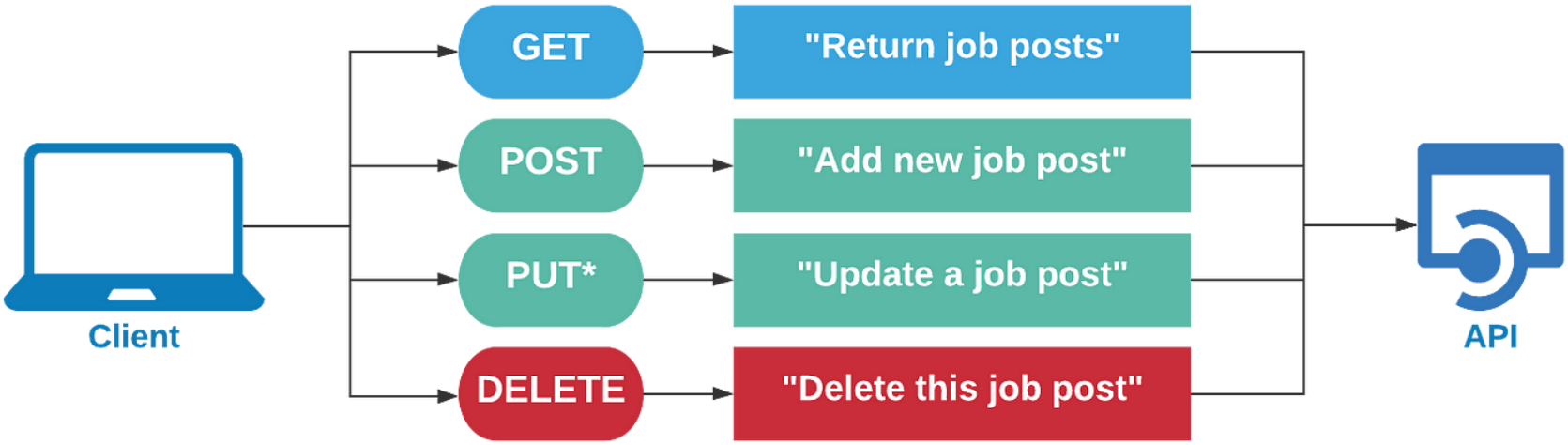
API REST

- As APIs REST são amplamente usadas para criar serviços web que oferecem acesso a recursos e funcionalidades por meio da Internet.
- Elas são preferidas por sua simplicidade, escalabilidade e facilidade de integração com diferentes plataformas e linguagens de programação.

Exemplo de API REST

- Um exemplo de API REST é uma API que permite que um aplicativo cliente acesse informações de produtos em um banco de dados de comércio eletrônico.
- O aplicativo cliente pode usar requisições HTTP:
  - GET para obter detalhes do produto,
  - POST para adicionar novos produtos,
  - PUT para atualizar informações de produtos existentes e
  - DELETE para remover produtos.
- Os dados são geralmente transmitidos em formato JSON ou XML.

Exemplo de API REST



Implementando API Rest

- Implementar uma API REST em JavaScript envolve a criação de **endpoints** que correspondem a diferentes recursos e operações (GET, POST, PUT, DELETE), seguindo os princípios e convenções de uma arquitetura RESTful.
- Abaixo um exemplo básico de como implementar uma API REST simples usando o Node.js e o framework Express.

Implementando API Rest server1.js

```
// Importação das bibliotecas necessárias
const express = require("express");
const cors = require("cors");

// Criação do servidor Express
const app = express();

// Definição da porta do servidor
const port = 3000;

// Configurações do Express
app.use(express.json());

// Configuração do CORS
app.use(cors());
```

Implementando API Rest server1.js

```
// Criando um array de produtos
let produtos = [
  { id: 1, nome: "Produto A" },
  { id: 2, nome: "Produto B" },
];

// Rota para listar todos os produtos
app.get("/produtos", (req, res) => {
  // Retorna uma lista de produtos
  res.json(produtos);
});
```

Implementando API Rest server1.js

```
// Rota para obter um produto por ID
app.get("/produtos/:id", (req, res) => {
  // Obtém o ID do produto
  const id = parseInt(req.params.id);
  // Procura o produto no array
  const produto = produtos.find((produto) => produto.id === id);
  // Retorna o produto encontrado ou um erro
  if (produto) {
    res.json(produto);
  } else {
    res.status(404).json({ message: "Produto não encontrado" });
  }
});
```

Implementando API Rest server1.js


```
// Rota para atualizar um produto
app.put("/produtos/:id", (req, res) => {
  // Obtém o ID do produto
  const id = parseInt(req.params.id);
  // Obtém os dados do produto
  const updatedproduto = req.body;
  // Procura o produto no array
  const index = produtos.findIndex((produto) => produto.id === id);
  // Atualiza o produto encontrado ou retorna um erro
  if (index !== -1) {
    produtos[index] = { ...produtos[index], ...updatedproduto };
    res.json(produtos[index]);
  } else {
    res.status(404).json({ message: "Produto não encontrado" });
  }
});
```

Implementando API Rest server1.js

```
// Rota para adicionar um novo produto
app.post("/produtos", (req, res) => {
  // Obtém os dados do produto
  const newproduto = req.body;
  // Adiciona o produto ao array
  produtos.push(newproduto);
  // Retorna o novo produto
  res.status(201).json(newproduto);
});
```

Implementando API Rest server1.js

```
// Rota para remover um produto
app.delete("/produtos/:id", (req, res) => {
  // Obtém o ID do produto
  const id = parseInt(req.params.id);
  // Procura o produto no array
  const index = produtos.findIndex((produto) => produto.id === id);
  // Remove o produto encontrado ou retorna um erro
  if (index !== -1) {
    const removedproduto = produtos.splice(index, 1);
    res.json(removedproduto[0]);
  } else {
    res.status(404).json({ message: "Produto não encontrado" });
  }
});
```

Implementando API Rest server1.js


```
// Inicia o servidor
app.listen(port, () => {
  console.log(`Servidor iniciado na porta ${port}`);
});
```

Dúvidas

??????