

Arquitetura de Backend

Prof: Me. Guilherme Villaca

O que é Arquitetura de Back-end?

 Definição:

Conjunto de tecnologias, padrões e estratégias usadas para construir a camada de lógica e dados de uma aplicação.

Responsável pelo processamento de requisições, regras de negócio e persistência de dados.

 Componentes principais:

✓ Servidor de Aplicação (Node.js, Spring Boot, Django, etc.)

✓ Banco de Dados (MySQL, PostgreSQL, MongoDB)

✓ API para comunicação com o front-end (REST, GraphQL)

✓ Autenticação e Segurança (JWT, OAuth)

O que é Node.js?

 Node.js é um runtime de JavaScript baseado no motor V8 do Chrome.

- ✓ Executa código JavaScript no servidor.
- ✓ Baseado em eventos e assíncrono (não bloqueante).
- ✓ Possui um gerenciador de pacotes poderoso (NPM).

 Por que usar Node.js?

- ✓ Alta performance para I/O (entrada e saída de dados) intensivo.
- ✓ Suporte a WebSockets e microservices.
- ✓ Grande comunidade e ecossistema.



Exemplo “Hello, World”

Crie um arquivo server.js

Adicione ao arquivo:

```
console.log("Hello, World!");
```

Execute no terminal:

```
node server.js
```

Saída esperada

```
Hello, World!
```



Criando um servidor HTTP com Node.js

```
const http = require("http");

const server = http.createServer((req, res) => {

  res.writeHead(200, { "Content-Type": "text/plain" });

  res.end("Hello, World!\n");

});

server.listen(3000, () => {

  console.log("Servidor rodando em http://localhost:3000");

});
```

NPM – Node Package Manager

 Gerenciador de pacotes do Node.js.

 Instala bibliotecas e frameworks.

 Gerencia dependências de um projeto.

 Comandos básicos:

`npm init -y` # Inicializa um projeto Node.js

`npm install express` # Instala o Express.js

`npm install -g nodemon` # Instala uma ferramenta globalmente

`npm run start` # Executa o script de inicialização

Express.js – Criando APIs

 Framework minimalista para aplicações web em Node.js.

 Permite criar servidores HTTP rapidamente.

 Suporta middlewares para manipular requisições e respostas.

 Exemplo de servidor básico:

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
  res.send('Olá, mundo!');
});
app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

O que são Middlewares?

- ♦ Middleware é uma função intermediária que é executada entre a requisição (request) e a resposta (response).
- ♦ Ele pode modificar a requisição (req) ou a resposta (res) antes de enviá-la ao cliente.

1 Como funciona um middleware?

O middleware recebe três parâmetros: (req, res, next)

req → Objeto da requisição.

res → Objeto da resposta.

next → Função que chama o próximo middleware na pilha.


O que são Middlewares?

2 Exemplo Simples de Middleware

```
const express = require("express");
const app = express();
// Middleware que exibe informações da requisição
app.use((req, res, next) => {
  console.log(`Método: ${req.method}, Rota: ${req.url}`);
  next(); // Passa para o próximo middleware ou rota
});
app.get("/", (req, res) => {
  res.send("Hello, World!");
});
app.listen(3000, () => {
  console.log("Servidor rodando em http://localhost:3000");
});
```

O que é ORM e TypeORM?

 ORM (Object-Relational Mapping):

- ✓ Mapeia tabelas do banco de dados como objetos JavaScript.
 - ✓ Facilita interações com o banco sem escrever SQL manualmente.
-  TypeORM – ORM para Node.js com suporte a TypeScript.

O que é ORM e TypeORM?

📌 Exemplo de Model no TypeORM:

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";
```

```
@Entity()
```

```
export class User {
```

```
    @PrimaryGeneratedColumn()
```

```
    id: number;
```

```
    @Column()
```

```
    name: string;
```

```
    @Column()
```

```
    email: string;
```

```
}
```

Configuração com Banco de Dados MySQL no Node.js

1. Instalando Dependências

Para conectar o Node.js ao MySQL, instale o pacote necessário:

```
npm install mysql2
```

2. Conectando ao Banco de Dados:

```
const mysql = require('mysql2');
const connection =
mysql.createConnection({
  host: 'localhost',
  user: 'seu_usuario',
  password: 'sua_senha',
  database: 'seu_banco'
});

connection.connect(err => {
  if (err) {
    console.error('Erro de conexão:', err);
  } else {
    console.log('Conectado ao MySQL');
  }
});
```

Criando uma API REST com Node.js, Express e TypeORM

3. Criando e Manipulando Tabelas

```
const query = `CREATE TABLE usuarios (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nome VARCHAR(100) NOT NULL,  
  email VARCHAR(100) UNIQUE NOT NULL  
);`;  
  
connection.query(query, (err, result) => {  
  if (err) {  
    console.error("Erro ao criar tabela", err);  
  } else {  
    console.log("Tabela criada com sucesso");  
  }  
});
```

Criando uma API REST com Node.js, Express e TypeORM

4. Inserindo e Buscando Dados

Inserindo Dados

```
const insertQuery = "INSERT INTO usuarios (nome, email) VALUES (?, ?)";
```

```
const values = ['João', 'joao@email.com'];
```

```
connection.query(insertQuery, values, (err, result) => {
```

```
  if (err) {
```

```
    console.error("Erro ao inserir usuário", err);
```

```
  } else {
```

```
    console.log("Usuário inserido com sucesso");
```

```
  }
```

```
});
```

Criando uma API REST com Node.js, Express e TypeORM

4. Inserindo e Buscando Dados

Inserindo Dados

```
const insertQuery = "INSERT INTO usuarios (nome, email) VALUES (?, ?)";
```

```
const values = ['João', 'joao@email.com'];
```

```
connection.query(insertQuery, values, (err, result) => {
```

```
  if (err) {
```

```
    console.error("Erro ao inserir usuário", err);
```

```
  } else {
```

```
    console.log("Usuário inserido com sucesso");
```

```
  }
```

```
});
```

Criando uma API REST com Node.js, Express e TypeORM

Buscando Dados

```
const selectQuery = "SELECT * FROM usuarios";

connection.query(selectQuery, (err, results) => {

  if (err) {

    console.error("Erro ao buscar usuários", err);

  } else {

    console.log("Usuários encontrados:", results);

  }

});
```


Criando uma API REST com Node.js, Express e TypeORM

 Passos básicos:

① Criar um projeto Node.js:

```
mkdir minha-api && cd minha-api
```

```
npm init -y
```

② Instalar dependências:

```
npm install express typeorm reflect-metadata mysql2
```

③ Criar um servidor Express.js e conectar ao banco de dados.

Ferramentas de Desenvolvimento

 Além do NPM, outras ferramentas úteis para desenvolvimento back-end:

✓ Nodemon: Monitora mudanças no código e reinicia automaticamente.

✓ Postman / Insomnia: Testa APIs REST.

✓ Docker: Facilita deploy e gerenciamento de dependências.

✓ Jest / Mocha: Frameworks para testes automatizados.

Como usar o Postman para testar requisições? <https://www.postman.com/>