Introdução e Configuração do Servidor

Objetivos da Aula:

- Compreender o que são Node.js e npm
- Configurar o ambiente de desenvolvimento
- Criar e configurar o servidor usando Express

1. O que é Node.js?

Conceito:

Node.js é um ambiente de execução JavaScript no servidor, permitindo que você execute código JavaScript fora do navegador.

É construído sobre o motor V8 do Google Chrome, que compila JavaScript para código de máquina nativo.

Node.js é assíncrono (não acontece em tempo real) e orientado a eventos, tornando-o eficiente e escalável para criar aplicações web de alto desempenho.

2. O que é npm?

Conceito:

npm (Node Package Manager) é o gerenciador de pacotes do Node.js.

Ele permite que você instale, atualize e remova bibliotecas e ferramentas adicionais, facilitando a gestão de dependências no seu projeto.

3. Configurando o Ambiente de Desenvolvimento

Ação:

1. Baixe e instale o Node.js a partir de nodejs.org.

O npm será instalado automaticamente com o Node.js.

Ação:

2. Verifique a instalação abrindo o terminal e executando:

node -v

npm -v

4. **Criando a Estrutura do Projeto**

Ação:

1. Crie uma nova pasta para o projeto, por exemplo, **apiFinance**.
2. Navegue até essa pasta usando o terminal ou prompt de comando:

cd caminho/para/apiFinance

Conceito:

Estamos criando a estrutura básica do projeto onde todo o código será armazenado.

5. **Inicializando um Projeto Node.js**

Ação:

1. Inicialize um novo projeto Node.js:

npm init -y

Conceito:

Isso criará um arquivo **package.json** com as configurações padrão. O arquivo **package.json** é usado para gerenciar as dependências e scripts do projeto.

6. **Instalando Dependências**

Ação:

1. Instale as bibliotecas necessárias para o projeto:

npm install express cors body-parser dotenv

npm install nodemon --save-dev

Conceito:

Explicação das Bibliotecas e Ferramentas:

- **express:** Um framework para criar servidores web e APIs. Ele facilita a construção de aplicativos web e APIs robustas.

- **cors:** Middleware para permitir requisições de diferentes origens.
- **body-parser:** Middleware para analisar o corpo das requisições HTTP.
- **dotenv:** Biblioteca para gerenciar variáveis de ambiente.
- **nodemon:** Ferramenta de desenvolvimento que reinicia automaticamente o servidor quando arquivos são modificados.

Introdução a Ferramentas e Conceitos no Node.js

Frameworks

Conceito:

Um framework é uma coleção de bibliotecas e ferramentas que fornecem uma estrutura predefinida para o desenvolvimento de aplicações. Frameworks facilitam o desenvolvimento ao oferecer soluções prontas para problemas comuns, permitindo que os desenvolvedores se concentrem na lógica específica do negócio em vez de reescrever código repetitivo.

Exemplos:

Express: Um framework minimalista para Node.js que facilita a criação de servidores web e APIs.

NestJS: Um framework para construir aplicações Node.js escaláveis e eficientes, usando TypeScript.

Middleware

Conceito:

Middleware é um código que fica entre o servidor e as requisições dos clientes. Ele processa requisições antes que elas cheguem às rotas finais, e pode ser usado para adicionar funcionalidades como autenticação, registro de logs, tratamento de erros, etc.

Exemplos:

body-parser: Middleware que analisa o corpo das requisições HTTP e disponibiliza os dados para o servidor.

cors: Middleware que habilita o CORS (Cross-Origin Resource Sharing) para permitir requisições de diferentes origens.

Bibliotecas

Conceito:

Bibliotecas são coleções de funções e módulos reutilizáveis que podem ser incorporadas em projetos para realizar tarefas específicas, sem impor uma estrutura rígida como um framework.

Exemplos:

dotenv: Biblioteca para carregar variáveis de ambiente de um arquivo **.env** para o **process.env**.

mysql2: Biblioteca para conectar e interagir com bancos de dados MySQL.

Outros Conceitos e Ferramentas no Node.js

Ferramentas de Desenvolvimento

nodemon: Ferramenta de desenvolvimento que reinicia automaticamente o servidor quando arquivos são modificados.

Gerenciamento de Pacotes

npm: Gerenciador de pacotes padrão para Node.js, usado para instalar, atualizar e remover bibliotecas e ferramentas.

7. **Configurando o Servidor Express**

Ação:

1. Crie um arquivo **server.js** na raiz do seu projeto.

Ação:

2. Adicione o seguinte código para importar as bibliotecas:

Código – javaScript

//importar as bibliotecas

const express = require('express'); // Importa o framework Express

const dotenv = require('dotenv'); // Importa o pacote dotenv para gerenciar variáveis de ambiente

const cors = require('cors'); // Importa o pacote cors para permitir requisições de diferentes origens

const bodyParser = require('body-parser'); // Importa o pacote body-parser para analisar o corpo das requisições HTTP

Conceito:

- **const express = require('express');:** Aqui estamos importando o framework Express e armazenando-o em uma constante chamada express. O Express é utilizado para criar e gerenciar servidores web e APIs.
- **const dotenv = require('dotenv');:** Estamos importando o pacote dotenv e armazenando-o em uma constante chamada dotenv. O dotenv é utilizado para gerenciar variáveis de ambiente, como credenciais de banco de dados.
- **const cors = require('cors');:** Estamos importando o pacote cors e armazenando-o em uma constante chamada cors. O CORS permite que o servidor aceite requisições de diferentes origens.
- **const bodyParser = require('body-parser');:** Estamos importando o pacote body-parser e armazenando-o em uma constante chamada bodyParser. O body-parser é utilizado para analisar o corpo das requisições HTTP, permitindo que o servidor manipule dados enviados pelo cliente.

Ação:

3. Adicione o seguinte código para configurar as variáveis de ambiente:

Código – JavaScript

//configurar as variáveis de ambiente

dotenv.config(); // Carrega as variáveis definidas no arquivo .env para process.env

Conceito:

- **dotenv.config();:** Aqui estamos configurando o **dotenv** para carregar as variáveis de ambiente definidas no arquivo **.env** para o **process.env.** Isso nos permite acessar essas variáveis em qualquer parte do nosso código usando **process.env.**

Ação:

4. Inicialize uma nova aplicação Express:

Código – Javascript

//Inicializa uma nova aplicação Express

const app = express(); // Inicializa uma nova aplicação Express

Conceito:

const app = express();: Aqui estamos inicializando uma nova aplicação Express e armazenando-a em uma constante chamada **app**. Essa aplicação será utilizada para configurar e gerenciar nosso servidor.

Ação:

5. Configure o CORS e o body-parser:

Código – Javascript

//Configura o CORS e o body-parser

app.use(cors()); // Habilita o CORS para todas as rotas

app.use(bodyParser.json()); // Configura o body-parser para analisar requisições JSON

Conceito:

app.use(cors());: Aqui estamos habilitando o CORS para todas as rotas da nossa aplicação. Isso permite que nosso servidor aceite requisições de diferentes origens.

app.use(bodyParser.json());: Aqui estamos configurando o body-parser para analisar requisições JSON. Isso permite que nosso servidor manipule dados enviados no formato JSON pelo cliente.

Ação:

6. Defina uma rota inicial para testar o servidor:

Código - javascript

//Rota inicial para testar o servidor

app.get('/', (req, res) => {

res.send('Servidor está rodando'); // Define uma rota inicial para testar o servidor

});

Conceito:

app.get('/', (req, res) => {...});: Aqui estamos definindo uma rota GET para a URL raiz **(/)**. Quando um cliente faz uma requisição GET para a URL raiz, o servidor responde com a mensagem **"Servidor está rodando"**.

res.send('Servidor está rodando');: Aqui estamos enviando a mensagem **"Servidor está rodando"** como resposta para a requisição.

(req, res) => {...}: Esta é uma **função arrow,** são uma forma concisa de escrever funções em JavaScript. Elas são especialmente úteis para funções de callback. A função recebe dois parâmetros, **req** (requisição) e **res** (resposta), e envia uma resposta ao cliente.

Ação:

7. Configure o servidor para escutar em uma porta específica:

Código - javascript

// Configura o servidor para escutar em uma porta específica

const PORT = process.env.PORT || 3000; // Define a porta a partir da variável de ambiente ou usa a porta 3000 como padrão

app.listen(PORT, () => {

console.log(`Servidor rodando na porta ${PORT}`);

}); // Escreve uma mensagem informando que o servidor está rodando

Conceito:

const PORT = process.env.PORT || 3000;: Aqui estamos definindo a porta que o servidor vai escutar. Primeiro, tentamos obter o valor da variável de ambiente **PORT**. Se essa variável não estiver definida, usamos a porta 3000 como padrão.

app.listen(PORT, ...): Aqui estamos configurando o servidor para escutar na porta especificada. Quando o servidor começa a escutar na porta, exibimos uma mensagem no console indicando que o servidor está rodando.

() => {...}: Esta é uma **função arrow,** são uma forma concisa de escrever funções em JavaScript. Elas são especialmente úteis para funções de callback. A função aqui é executada quando o servidor começa a escutar na porta especificada e exibe uma mensagem no console.

Passo 8: Configurando o Script de Início

Ação:

1. No arquivo **package.json**, adicione o script de início para o Nodemon:

json

```
"scripts": {
  "start": "nodemon server.js"
}
```

Conceito:

scripts: { "start": "nodemon server.js" }: Aqui estamos definindo um script no arquivo **package.json** para iniciar o servidor usando o Nodemon. Quando executamos **npm start**, o Nodemon iniciará o servidor e reiniciará automaticamente sempre que detectarmos mudanças nos arquivos.

Ação:

2. Inicie o servidor:

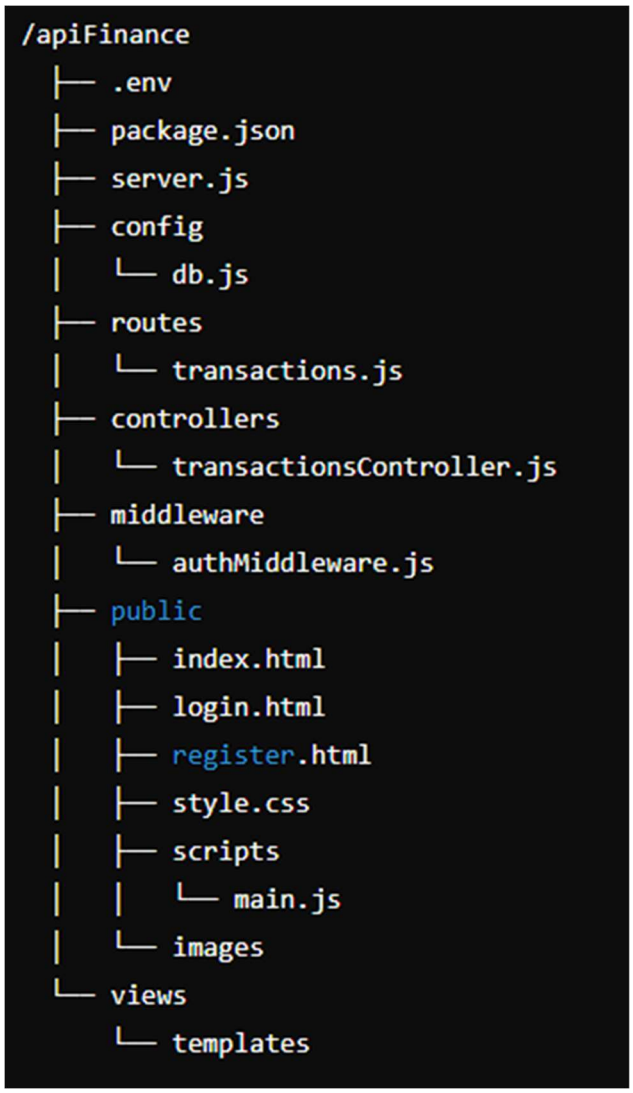
```
npm start
```

Conceito:

Ao executar **npm start**, o Nodemon iniciará o servidor e você deverá ver a mensagem **"Servidor rodando na porta 3000"** no terminal.

Estrutura do Projeto (Referência)

Para organizar seu projeto de maneira eficiente e clara, é uma boa prática separar o **frontend** do **backend**. Aqui está uma sugestão de estrutura de diretórios para incluir o **frontend** no seu projeto **apiFinance**:



Essa estrutura ajuda a manter a organização e a separação de responsabilidades entre o **backend** e o **frontend**.

Parte Prática (1 hora):

Ação:

1. Criar a estrutura de pastas do projeto.
2. Instalar Node.js e npm.
3. Inicializar um projeto Node.js.
4. Instalar as dependências necessárias.
5. Configurar e testar o servidor Express.