DESENVOLVIMENTO WEB III

Introdução ao Back-end: O que é, sua importância e tecnologias populares, node.js

Prof. Esp. Érick Henrique Pereira Nicolau

O que é Back-end?

- A parte "invisível" de uma aplicação web.
- Responsável pela lógica, processamento de dados e interação com o banco de dados.
- "Cérebro" da aplicação, trabalhando nos bastidores para entregar a experiência do usuário.

Analogia da Cozinha

- Front-end: A parte visível do restaurante, como o salão, mesas e decoração.
- Back-end: A cozinha, onde os pratos são preparados, os pedidos são gerenciados e os ingredientes são armazenados.

Importância do Back-end

- Lógica de Negócio: Define as regras e o funcionamento da aplicação.
- Gerenciamento de Dados: Armazena, recupera e manipula dados de forma segura.
- Segurança: Protege a aplicação contra ataques e vulnerabilidades.
- Escalabilidade: Permite que a aplicação cresça e suporte um número maior de usuários.

Tecnologias Populares

- Node.js: Ambiente de execução JavaScript para desenvolvimento back-end.
- Python: Linguagem de programação versátil e popular para back-end.
- Django: Framework Python para desenvolvimento web rápido e escalável.
- Outras opções: Ruby on Rails, PHP, Java, Go, etc.

Exemplo de Código - Node.js

```
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');
});
```

Explicação do Código - Node.js

- Importa o módulo Express.js para criar o servidor.
- Cria uma rota que responde com "Olá, mundo!" quando acessada.
- Inicia o servidor na porta 3000.

Comparativo Node.js vs. Python/Django

Característica	Node.js	Python/Django
Linguagem	JavaScript	Python
Performance	Alta	Alta
Curva de aprendizado	Moderada	Moderada
Comunidade	Grande e ativa	Grande e ativa
Frameworks	Express.js, NestJS, Koa, etc.	Django, Flask, Pyramid, etc.
Uso comum	Aplicações em tempo real, APIs, etc.	Aplicações web complexas, ciência de dados

Escolhendo a Tecnologia Certa

- Não existe uma resposta única.
- Depende do projeto, da equipe e das preferências pessoais.
- Experimente ambas as tecnologias e veja qual se adapta melhor ao seu estilo.

Pratica

- Certifique-se de ter o Node.js e o npm (gerenciador de pacotes do Node) instalados.
- Crie um diretório para o projeto e navegue até ele no terminal.
- Execute npm init -y para inicializar o projeto.
- Instale o Express: npm install express

O que é o node.js

- ambiente de execução JavaScript multiplataforma
- código aberto e gratuito
- permite criar servidores, aplicações da Web, ferramentas de linha de comando e programas de automação de tarefas.

O que é o express.js??

- O Express.js é um framework web minimalista e flexível para Node.js, utilizado para construir aplicações web e APIs de forma rápida e eficiente. Ele oferece recursos como:

Código básico

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('Olá, DSM FRANCA!');
});

app.listen(port, () => {
    console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Explicando

const express = require('express’);

- Importa o módulo Express:

const app = express();

- Cria uma instância do Express:

const port = 3000;

- Define a porta do servidor

Explicando

- app.get: Indica que o servidor deve responder a requisições HTTP do tipo GET
- '/': Especifica a rota raiz (a página principal) da sua aplicação
- (req, res): São os objetos de requisição e resposta, respectivamente. O req contém informações sobre a solicitação do cliente, e o res é usado para enviar a resposta de volta.
- res.send('Olá, DSM FRANCA!')

Continuando

- Podemos salvar nosso arquivo na pasta criada anteriormente
- Salvar com o nome de index.js
- Para iniciar o “servidor” basta digitar no terminal o comando node index.js
- Para testar o servidor, basta acessar através do navegador o endereço localhost:3000

Diferentes Métodos HTTP

GET

- Propósito: Solicitar um recurso do servidor.
- Funcionamento: O cliente envia uma solicitação GET para uma URL específica, e o servidor responde com os dados do recurso solicitado (por exemplo, uma página HTML, uma imagem, um arquivo JSON).
- Exemplo: Quando você digita um endereço web no seu navegador, ele envia uma solicitação GET para o servidor para obter a página web correspondente.

POST

- Propósito: Enviar dados para o servidor para criar ou atualizar um recurso.
- Funcionamento: O cliente envia uma solicitação POST com os dados no corpo da mensagem.
- O servidor processa os dados e cria ou atualiza o recurso correspondente.
- Exemplo: Quando você preenche um formulário online e clica em "Enviar", o navegador geralmente envia uma solicitação POST com os dados do formulário para o servidor.

PUT

- Propósito: Atualizar um recurso existente no servidor.
- Funcionamento: O cliente envia uma solicitação PUT com os dados completos do recurso atualizado no corpo da mensagem. O servidor substitui o recurso existente pelos novos dados.
- Exemplo: Você pode usar PUT para atualizar completamente as informações de um usuário, como nome, e-mail e endereço.

DELETE

- Propósito: Excluir um recurso do servidor.
- Funcionamento: O cliente envia uma solicitação DELETE para a URL do recurso que deseja excluir. O servidor remove o recurso se ele existir.Exemplo:
- Você pode usar DELETE para remover um usuário específico do sistema.

Código v2

```
const express = require('express');
const app = express();
const port = 3000;

//necessário para analisar o corpo das requisições POST e PUT, que geralmente enviam dados em formato JSON.
app.use(express.json());

// Dados de exemplo (simulando, ideal seria um banco de dados)
let usuarios = [
  { id: 1, nome: ‘Ana' },
  { id: 2, nome: ‘Matheus' },
];
```

Método GET

```
// GET /usuarios - Lista todos os usuários (este é fácil)
app.get('/usuarios', (req, res) => {
  res.json(usuarios);
});
```

Método Get

```
// GET /usuarios/:id - Busca um usuário pelo ID
app.get('/usuarios/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const usuario = usuarios.find(u => u.id === id);
  if (usuario) {
    res.json(usuario);
  } else {
    res.status(404).json({ mensagem: 'Usuário não encontrado' });
  }
});
```

Método POST

```
// POST /usuarios - Cria um novo usuário
app.post('/usuarios', (req, res) => {
  const novoUsuario = req.body;
  novoUsuario.id = usuarios.length + 1;
  usuarios.push(novoUsuario);
  res.status(201).json(novoUsuario);
});
```

Método PUT

```
// PUT /usuarios/:id - Atualiza um usuário existente a partir do ID
app.put('/usuarios/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const usuarioIndex = usuarios.findIndex(u => u.id === id);
  if (usuarioIndex !== -1) {
    usuarios[usuarioIndex] = { ...usuarios[usuarioIndex], ...req.body };
    res.json(usuarios[usuarioIndex]);
  } else {
    res.status(404).json({ mensagem: 'Usuário não encontrado' });
  }
});
```

Método delete

```
// DELETE /usuarios/:id - Exclui um usuário de acordo com o id fornecido

app.delete('/usuarios/:id', (req, res) => {
  const id = parseInt(req.params.id);
  usuarios = usuarios.filter(u => u.id !== id);
  res.json({ mensagem: 'Usuário excluído com sucesso' });
});

//”LIGA” nosso servidor
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Ferramenta POSTMAN

- Postman é uma ferramenta muito útil para desenvolvedores e testadores de APIs (Application Programming Interfaces). Ele facilita o processo de criação, teste, documentação e monitoramento de APIs, tornando-o mais eficiente e produtivo.

Ferramenta POSTMAN

- Envio de requisições HTTP: O Postman permite enviar requisições HTTP (GET, POST, PUT, DELETE, etc.)
- Visualização de respostas: Ele exibe as respostas da API de forma organizada, incluindo o código de status, cabeçalhos e corpo da resposta (em formato JSON, XML, HTML, etc.).
- Testes automatizados: É possível criar testes automatizados para verificar se a API está retornando as respostas esperadas.
- Coleções e ambientes: O Postman permite organizar as requisições em coleções e definir variáveis de ambiente para facilitar o reuso e a colaboração entre equipes.
- Documentação da API: Ele pode gerar automaticamente a documentação da API com base nas requisições e respostas.
- Monitoramento da API: O Postman oferece recursos de monitoramento para verificar a disponibilidade e o desempenho da API, alertando sobre possíveis problemas.

Testando método GET

- Selecionar método GET
- Indicar a seguinte URL: localhost:3000/usuarios/
- Clicar em send

Encontrando um usuário pelo ID

- Mude a URL para http://localhost:3000/usuarios/1 (ou outro ID válido).
- Clicar em "Send".
- Aparecera o usuário correspondente

- OBS: "Usuário não encontrado“ aparecerá em caso de id invalido

Criando usuário

- Troque para o método POST
- url localhost:3000/usuarios.
- Na aba "Body", selecione "raw" e JSON como formato.
- No corpo da requisição, insira um objeto JSON com os dados do novo usuário { "nome": “Érick" }.
- Clique em "Send".
- Você deverá receber os dados do novo usuário, incluindo o ID atribuído.

Atualizando usuário

- Mude o método para PUT
- URL http://localhost:3000/usuarios/1 (ou outro ID válido).
- No corpo da requisição, insira um objeto JSON com os dados atualizados do usuário.
- Clique em "Send". Você deverá receber uma resposta com os dados do usuário atualizado.

Deletar usuário

- Mude o método para DELETE
- URL para http://localhost:3000/usuarios/1 (ou outro ID válido).
- Clique em "Send".
- Você deverá receber uma resposta confirmando a exclusão do usuário.