

## materail de apoio

Site: [Geração Tech](#)  
Curso: Formação em Desenvolvedor Web - Online  
Livro: materail de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS  
Data: sexta-feira, 2 ago. 2024, 08:41

# Índice

## **1. CRUD Avançado**

1.1. Vídeo Aula

## **2. Configurando o Servidor**

2.1. Vídeo Aula

## **3. Criação e Gerenciamento de Posts**

3.1. Vídeo Aula

# 1. CRUD Avançado

## Configuração Atual

### Estrutura do Projeto

No projeto, temos a seguinte estrutura:

```
go
└─ Copiar código

my-backend-project/

├─
├─ node_modules/
├─ src/
│ └─ controllers/
│   └─ usuarioController.js
│   └─ models/
│     └─ usuarioModel.js
│     └─ routes/
│       └─ usuarioRoutes.js
│       └─ app.js
│       └─ server.js
└─ package.json
└─ package-lock.json
```

## 1.1. Vídeo Aula

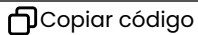
dia 7 express 1 converted



## 2. Configurando o Servidor

No arquivo `server.js`, configuramos o servidor básico do Express:

javascript

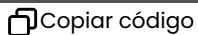


```
const express = require('express'); const app = express(); const usuarioRoutes = require('./routes/usuarioRoutes');
app.use(express.json()); app.use('/usuarios', usuarioRoutes); const PORT = 3000; app.listen(PORT, () => {
console.log(`Servidor rodando na porta ${PORT}`); });
```

### Definindo Rotas

No arquivo `routes/usuarioRoutes.js`, configuramos as rotas para a API de usuários:

javascript



```
const express = require('express'); const router = express.Router(); const usuarioController =
require('../controllers/usuarioController'); router.get('/', usuarioController.listar); router.get('/:id',
usuarioController.consultarPorId); router.post('/', usuarioController.criar); router.put('/:id',
usuarioController.atualizar); router.delete('/:id', usuarioController.excluir); module.exports = router;
```

### Criando Controladores

No arquivo `controllers/usuarioController.js`, implementamos a lógica para manipulação dos usuários:

javascript



```
const Usuario = require('../models/usuarioModel'); class UsuarioController { static listar(req, res) {
res.status(200).json(Usuario.listar()); } static consultarPorId(req, res) { const id = req.params.id; const usuario =
Usuario.consultarPorId(id); if (usuario) { res.status(200).json(usuario); } else { res.status(404).json({ message: 'Usuário
não encontrado' }); } } static criar(req, res) { const novoUsuario = req.body; Usuario.criar(novoUsuario);
res.status(201).json({ message: 'Usuário criado com sucesso' }); } static atualizar(req, res) { const id = req.params.id;
const dadosAtualizados = req.body; Usuario.atualizar(id, dadosAtualizados); res.status(200).json({ message: 'Usuário
atualizado com sucesso' }); } static excluir(req, res) { const id = req.params.id; Usuario.excluir(id);
res.status(200).json({ message: 'Usuário excluído com sucesso' }); } } module.exports = UsuarioController;
```

### Modelando a Classe Usuario

No arquivo `models/usuarioModel.js`, definimos a classe `Usuario` com seus métodos estáticos:

javascript



```
class Usuario { static usuarios = [ { id: 1, nome: 'Admin', login: 'admin' }, { id: 2, nome: 'Teste', login: 'teste' } ];
static listar() { return this.usuarios; } static consultarPorId(id) { return this.usuarios.find(usuario => usuario.id ==
id); } static criar(usuario) { usuario.id = this.usuarios.length + 1; this.usuarios.push(usuario); } static atualizar(id,
dadosAtualizados) { const index = this.usuarios.findIndex(usuario => usuario.id == id); if (index !== -1) {
this.usuarios[index] = { ...this.usuarios[index], ...dadosAtualizados }; } } static excluir(id) { this.usuarios =
this.usuarios.filter(usuario => usuario.id != id); } } module.exports = Usuario;
```

### Testando a API

#### Iniciando o Servidor

Para iniciar o servidor, execute o comando:

bash

 Copiar código

```
node src/server.js
```

## Testando as Rotas

Utilize uma ferramenta como Postman ou Insomnia para testar as seguintes rotas:


- **GET /usuarios:** Lista todos os usuários.
- **GET /usuarios/**
  - : Consulta um usuário por ID.
- **POST /usuarios:** Cria um novo usuário.
- **PUT /usuarios/**
  - : Atualiza o usuário com o ID especificado.
- **DELETE /usuarios/**
  - : Remove o usuário com o ID especificado.

## Implementação de Métodos CRUD

### Método Consultar por ID

Vamos configurar a rota de consultar por ID. No arquivo `usuarioRoutes.js`, adicionamos:


javascript

 Copiar código

```
router.get('/:id', usuarioController.consultarPorId);
```

No controller, recebemos o parâmetro ID e passamos para o método do model:

javascript

 Copiar código

```
static consultarPorId(req, res) { const id = req.params.id; const usuario = Usuario.consultarPorId(id); if (usuario) {  
  res.status(200).json(usuario); } else { res.status(404).json({ message: 'Usuário não encontrado' }); } }
```

No model, implementamos a lógica para buscar o usuário por ID:

javascript


 Copiar código

```
static consultarPorId(id) { return this.usuarios.find(usuario => usuario.id == id); }
```

### Método Deletar

Configuramos a rota de deletar no arquivo `usuarioRoutes.js`:

javascript

 Copiar código

```
router.delete('/:id', usuarioController.excluir);
```

No controller, passamos o ID para o método de exclusão:

javascript

 Copiar código

```
static excluir(req, res) { const id = req.params.id; Usuario.excluir(id); res.status(200).json({ message: 'Usuário excluído com sucesso' }); }
```

No model, implementamos a exclusão do usuário:

javascript

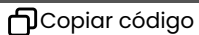


```
static excluir(id) { this.usuarios = this.usuarios.filter(usuario => usuario.id != id); }
```

## Método Atualizar

Configuramos a rota de atualização no arquivo `usuarioRoutes.js`:

javascript



```
router.put('/:id', usuarioController.atualizar);
```

No controller, recebemos os dados atualizados e passamos para o model:

javascript



```
static atualizar(req, res) { const id = req.params.id; const dadosAtualizados = req.body; Usuario.atualizar(id, dadosAtualizados); res.status(200).json({ message: 'Usuário atualizado com sucesso' }); }
```

No model, implementamos a lógica de atualização:

javascript



```
static atualizar(id, dadosAtualizados) { const index = this.usuarios.findIndex(usuario => usuario.id == id); if (index !== -1) { this.usuarios[index] = { ...this.usuarios[index], ...dadosAtualizados }; } }
```

## Conclusão

Nesta aula, configuramos métodos adicionais para o CRUD, incluindo consultar por ID, deletar e atualizar usuários.

Utilizamos o Insomnia para testar nossas rotas e validar a funcionalidade do back-end. Implementamos a arquitetura MVC para organizar melhor nosso código.

Pratiquem esses conceitos e explorem novas funcionalidades para aprimorar suas habilidades em desenvolvimento back-end.

Até a próxima aula!

## 2.1. Vídeo Aula

dia 7 express 2 converted





### 3. Criação e Gerenciamento de Posts

#### Estrutura do Projeto

#### Estrutura Atual

```
go
[icon] Copiar código
```

my-backend-project/

| └─ node\_modules/

└─ src/ |

└─ controllers/

| | └─ usuarioController.js

| | └─ postController.js

| └─ models/

| | └─ usuarioModel.js

| | └─ postModel.js

| └─ routes/

| | └─ usuarioRoutes.js

| | └─ postRoutes.js

| └─ app.js

| └─ server.js

└─ package.json


└─ package-lock.json

## Criando o Model para Posts

### postModel.js

Vamos criar um novo arquivo `postModel.js` dentro da pasta `models`.

javascript

 Copiar código


```
class PostModel { static lista = [ { id: 1, titulo: 'Primeiro Post', conteudo: 'Conteúdo do primeiro post', userId: 1 }, {
id: 2, titulo: 'Segundo Post', conteudo: 'Conteúdo do segundo post', userId: 2 } ]; static listar() { return this.lista; }
static consultarPorId(id) { return this.lista.find(post => post.id === id); } static criar(data) { this.lista.push(data);
return data; } static atualizar(id, data) { const index = this.lista.findIndex(post => post.id === id); if (index !== -1) {
this.lista[index] = { ...this.lista[index], ...data }; return this.lista[index]; } return null; } static deletar(id) { const
index = this.lista.findIndex(post => post.id === id); if (index !== -1) { const deleted = this.lista.splice(index, 1);
return deleted[0]; } return null; } } module.exports = PostModel;
```

## Criando o Controller para Posts

### postController.js

Vamos criar um novo arquivo `postController.js` dentro da pasta `controllers`.

javascript

 Copiar código

```
const PostModel = require('../models/postModel'); class PostController { static listar(req, res) { const posts =
PostModel.listar(); res.status(200).json(posts); } static consultarPorId(req, res) { const id = parseInt(req.params.id, 10);
const post = PostModel.consultarPorId(id); if (post) { res.status(200).json(post); } else { res.status(404).json({ mensagem:
'Post não encontrado' }); } } static criar(req, res) { const data = req.body; const novoPost = PostModel.criar(data);
res.status(201).json(novoPost); } static atualizar(req, res) { const id = parseInt(req.params.id, 10); const data =
req.body; const postAtualizado = PostModel.atualizar(id, data); if (postAtualizado) { res.status(200).json(postAtualizado);
} else { res.status(404).json({ mensagem: 'Post não encontrado' }); } } static deletar(req, res) { const id =
parseInt(req.params.id, 10); const postDeletado = PostModel.deletar(id); if (postDeletado) { res.status(200).json({
mensagem: 'Post deletado com sucesso' }); } else { res.status(404).json({ mensagem: 'Post não encontrado' }); } } }
module.exports = PostController;
```

## Criando as Rotas para Posts

### postRoutes.js

Vamos criar um novo arquivo `postRoutes.js` dentro da pasta `routes`.

javascript

 Copiar código


```
const express = require('express'); const router = express.Router(); const PostController =
require('../controllers/postController'); router.get('/', PostController.listar); router.get('/:id',
PostController.consultarPorId); router.post('/', PostController.criar); router.put('/:id', PostController.atualizar);
router.delete('/:id', PostController.deletar); module.exports = router;
```

## Integrando as Rotas no Servidor

### server.js

Vamos atualizar o arquivo `server.js` para incluir as novas rotas de posts.

javascript

 Copiar código

```
const express = require('express'); const app = express(); const usuarioRoutes = require('./routes/usuarioRoutes'); const
postRoutes = require('./routes/postRoutes'); app.use(express.json()); app.use('/usuarios', usuarioRoutes); app.use('/posts',
```

```
postRoutes); const PORT = 3000; app.listen(PORT, () => { console.log(`Servidor rodando na porta ${PORT}`); });
```

## Testando as Rotas com Insomnia

### Configurando Insomnia

#### 1. Listar Posts:

- Método: GET
- URL: `http://localhost:3000/posts`

#### 2. Consultar Post por ID:

- Método: GET
- URL: `http://localhost:3000/posts/1`

#### 3. Criar Post:

- Método: POST
- URL: `http://localhost:3000/posts`
- Body (JSON):

json

 Copiar código

```
{ "id": 3, "titulo": "Novo Post", "conteudo": "Conteúdo do novo post", "userId": 1 }
```

#### 4. Atualizar Post:

- Método: PUT
- URL: `http://localhost:3000/posts/1`
- Body (JSON):

json

 Copiar código

```
{ "titulo": "Post Atualizado", "conteudo": "Conteúdo atualizado do post" }
```

#### 5. Deletar Post:

- Método: DELETE
- URL: `http://localhost:3000/posts/1`

## Considerações Finais

Hoje, criamos e configuramos o CRUD de posts para nosso blog.

Aprendemos a criar Models, Controllers e rotas no Express.

Também testamos nossas rotas usando o Insomnia.

Continuem praticando esses conceitos e até a próxima aula!

---

Estudem e pratiquem bastante. Até a próxima aula!

### 3.1. Vídeo Aula

dia 7 express 3 converted

