Configurando Rotas e Controladores

Objetivos da Aula:

- Compreender o que são rotas e controladores
- Configurar rotas para as funcionalidades da API
- Criar controladores para manipular as requisições

1. O que são Rotas?

Conceito:

Rotas são os caminhos definidos na sua aplicação que respondem às requisições HTTP. Cada rota é associada a uma função que é executada quando a rota é acessada. As rotas são usadas para definir como a aplicação responde às requisições para diferentes endpoints (URLs).

Os endpoints de API são a localização de um recurso que permite que dois sistemas interajam entre si. O software não pode se comunicar com outras ferramentas sem APIs, e as APIs não funcionam sem endpoints.

2. O que são Controladores?

Conceito:

Controladores são funções que contêm a lógica para manipular as requisições e respostas HTTP. Eles são responsáveis por processar os dados, interagir com o banco de dados, e retornar as respostas apropriadas para o cliente.

Status de Respostas

Códigos de Status Comuns

- 200 OK: Requisição bem-sucedida.
- 201 Created: Recurso criado com sucesso.
- 400 Bad Request: Requisição inválida.
- 401 Unauthorized: Autenticação necessária.
- 404 Not Found: Recurso não encontrado.
- 500 Internal Server Error: Erro no servidor.

3. Configurando Rotas e Controladores

3.1 Criar a Estrutura de Pastas

1. Crie uma pasta chamada **routes** na raiz do projeto.

A pasta **routes** armazenará os arquivos de definição de rotas da aplicação.

2. Crie uma pasta chamada **controllers** na raiz do projeto:

Esta pasta armazenará os arquivos de controladores da aplicação.

3.2 Configurar Rotas para Transações

1. Dentro da pasta **routes,** crie um arquivo **transactions.js:**

   Este arquivo conterá as definições de rotas relacionadas às transações financeiras.

2. Adicione o seguinte código ao arquivo **transactions.js:**

Este código define uma rota básica para obter todas as transações**.**

Abra o arquivo **transactions.js** em seu editor de texto e adicione:

Código JavaScript

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

const router = express.Router(); // Cria um novo roteador

const transactionsController = require('../controllers/transactionsController'); // Importa o controlador de transações

// Definindo uma rota para obter todas as transações

router.get('/', transactionsController.getAllTransactions);

// Exportando o roteador

module.exports = router;

Explicação do Código:

- **const express = require('express**');: Importa o framework Express.
- **const router = express.Router();:** Cria um novo roteador que será usado para definir as rotas.

- **const transactionsController = require('../controllers/transactionsController');:** Importa o controlador de transações.
- **router.get('/', transactionsController.getAllTransactions);:** Define uma rota GET para a URL raiz das transações (/). Quando um cliente faz uma requisição GET para essa URL, a função getAllTransactions do controlador será executada.
- **module.exports = router;:** Exporta o roteador para que possa ser utilizado em outros arquivos.

3.3 Configurar Controladores para Transações

1. Dentro da pasta **controllers,** crie um arquivo **transactionsController.js**:

Este arquivo conterá as funções que manipulam as requisições relacionadas às transações financeiras.

2. Adicione o seguinte código ao arquivo **transactionsController.js**:

Este código define a função para obter todas as transações.

Abra o arquivo **transactionsController.js** em seu editor de texto e adicione:

Código JavaScript

```
const db = require('../config/db'); // Importa a conexão com o banco de dados

// Função para obter todas as transações
const getAllTransactions = (req, res) => {
  db.query('SELECT * FROM transactions', (err, results) => {
    if (err) {
      console.error('Erro ao obter transações:', err);
      res.status(500).send('Erro ao obter transações');
      return;
    }
    res.json(results);
  });
};
```

module.exports = {

getAllTransactions

};

Explicação do Código:

- **const db = require('../config/db');: Importa a conexão com o banco de dados.**
- **const getAllTransactions = (req, res) => {...};: Define uma função arrow chamada getAllTransactions que recebe dois parâmetros, req (requisição) e res (resposta). Esta função será usada para obter todas as transações do banco de dados.**
- **db.query('SELECT * FROM transactions', (err, results) => {...});: Executa uma query SQL para selecionar todas as transações na tabela transactions. A função callback recebe dois parâmetros, err (erro) e results (resultados).**
- **if (err) {...};: Verifica se houve um erro ao executar a query. Se houve, exibe uma mensagem de erro no console e envia uma resposta de erro para o cliente.**
- **res.json(results);: Se a query foi executada com sucesso, envia os resultados como uma resposta JSON para o cliente.**
- **module.exports = { getAllTransactions };: Exporta a função getAllTransactions para que possa ser utilizada em outros arquivos.**

3.4 Atualizar o Arquivo server.js

1. Atualize o arquivo **server.js** para utilizar as rotas:

Abra o arquivo **server.js** em seu editor de texto e adicione as seguintes linhas:

// Importar as Bibliotecas

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

// Configurar as Variáveis de ambiente

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

// Importar as Bibliotecas

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

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

const db = require('./config/db'); // Importa a conexão com o banco de dados

- **const transactionsRoutes = require('./routes/transactions');** // Importa as rotas de transações

// Inicializar nova aplicação Express

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

// Configurar 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

// Usar as rotas de transações para todas as requisições que começam com /api/transactions

- **app.use('/api/transactions', transactionsRoutes);**

// Rota inicial para testar o servidor

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

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

});

// Configurar o servidor para 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}`); // Loga uma mensagem informando que o servidor está rodando

});

Explicação do Código:

- const transactionsRoutes = require('./routes/transactions');: Importa as rotas de transações.
- app.use('/api/transactions', transactionsRoutes);: Configura o servidor para usar as rotas de transações para todas as requisições que começam com /api/transactions.

Parte Prática (1 hora):

Parte Prática (1 hora):

Ação:

1. Criar a pasta **routes** e definir as rotas de transações.
2. Criar a pasta **controllers** e definir o controlador de transações.
3. Atualizar o arquivo **transactions.js** para utilizar o controlador.
4. Atualizar o arquivo **server.js** para utilizar as rotas.
5. Testar a rota **GET** para **/api/transactions** para garantir que está retornando todas as transações.

Passo a Passo para Inserção dos Dados:

1. **Inserindo Dados no Banco de Dados**

Antes de testar a rota GET para /api/transactions, precisamos inserir alguns dados na tabela transactions. Você pode fazer isso usando o MySQL Workbench ou outra ferramenta similar.

Ação:

- Abra o **MySQL Workbench** e conecte-se ao seu banco de dados.

- Inserir Usuário na Tabela **users**

Vamos inserir um usuário na tabela **users** para garantir que haja um **user_id** válido que possa ser referenciado na tabela **transactions**.

- Execute a seguinte **query SQL** para inserir um usuário na tabela **users**:

Código SQL

```
USE finance_db;
```

INSERT INTO users (name, email, password, birth_date) VALUES

('Joana Dark', 'joanadark@apifinance.com', '123456', '1990-01-01');

2. **Inserir Transações na Tabela transactions**

Depois de inserir o usuário, podemos inserir as transações usando o **user_id** do usuário recém-criado.

- **Obtenha o id do usuário recém-criado**:

Execute a seguinte **query SQL** para obter o id do usuário:

Código SQL

SELECT id FROM users WHERE email = ' joanadark@apifinance.com ';

Anote o id retornado.

3. **Execute as seguintes queries SQL para inserir dados na tabela transactions:**

Código SQL

USE finance_db;

INSERT INTO transactions (date, amount, description, category, account, user_id) VALUES

('2023-07-01', 150.00, 'Compra no supermercado', 'Alimentação', 'Cartão de Crédito', 1),

('2023-07-02', 75.50, 'Combustível', 'Transporte', 'Cartão de Débito', 1),

('2023-07-03', 200.00, 'Jantar no restaurante', 'Lazer', 'Cartão de Crédito', 1),

('2023-07-04', 50.00, 'Compra de livros', 'Educação', 'Cartão de Débito', 1),

('2023-07-05', 120.00, 'Manutenção do carro', 'Transporte', 'Dinheiro', 1),

('2023-07-06', 300.00, 'Compra de roupas', 'Vestuário', 'Cartão de Crédito', 1),

('2023-07-07', 40.00, 'Assinatura de streaming', 'Entretenimento', 'Cartão de Débito', 1);

Teste Prático com Insomnia:

Após inserir os dados, você pode usar o **Insomnia** para testar a rota **GET** para **/api/transactions** e verificar se todas as transações foram inseridas corretamente.

Passos para Testar com Insomnia:

1. **Abra o Insomnia**:

Se ainda não tem o Insomnia instalado, você pode baixá-lo e instalá-lo a partir do site oficial.

2. **Faça o login**:

Clique no ícone de **New Collection** de o nome de **ApiFinance**.

3. **Crie uma Nova Requisição**:

Clique em New HTTP Request.

Dê um nome à sua requisição, por exemplo, "Get Transactions".

Selecione o método HTTP como "GET".

4. **Configure a URL da Requisição**:

Na barra de URL, insira http://localhost:3000/api/transactions.

5. **Envie a Requisição**:

Clique no botão "Send" para enviar a requisição.

6. **Verifique a Resposta**:

Verifique o painel de resposta no Insomnia.

A resposta deve conter uma lista de transações no formato JSON, retornando todas as transações armazenadas no banco de dados.

Exemplo de Resposta Esperada:

200 OK

Código json


```
[
  {
    "id": 1,
    "date": "2023-07-01",
    "amount": 150.00,
```

```
  "description": "Compra no supermercado",
  "category": "Alimentação",
  "account": "Cartão de Crédito",
  "user_id": 1
 },
 {
  "id": 2,
  "date": "2023-07-02",
  "amount": 75.50,
  "description": "Combustível",
  "category": "Transporte",
  "account": "Cartão de Débito",
  "user_id": 1
 },
 {
  "id": 3,
  "date": "2023-07-03",
  "amount": 200.00,
  "description": "Jantar no restaurante",
  "category": "Lazer",
  "account": "Cartão de Crédito",
  "user_id": 1
 },
 {
  "id": 4,
  "date": "2023-07-04",
  "amount": 50.00,
  "description": "Compra de livros",
  "category": "Educação",
  "account": "Cartão de Débito",
  "user_id": 1
 },
 {
  "id": 5,
  "date": "2023-07-05",
  "amount": 120.00,
  "description": "Manutenção do carro",
  "category": "Transporte",
  "account": "Dinheiro",
  "user_id": 1
 },
 {
  "id": 6,
  "date": "2023-07-06",
  "amount": 300.00,
  "description": "Compra de roupas",
  "category": "Vestuário",
  "account": "Cartão de Crédito",
  "user_id": 1
 },
 {
  "id": 7,
```

```
  "date": "2023-07-07",
  "amount": 40.00,
  "description": "Assinatura de streaming",
  "category": "Entretenimento",
  "account": "Cartão de Débito",
  "user_id": 1
 }
]
```


Resolução de Problemas Comuns

Erro 404 (Not Found)

- Verifique se o servidor está rodando corretamente.
- Certifique-se de que a URL está correta e que a rota /api/transactions está configurada no server.js.

Erro 500 (Internal Server Error)

- Verifique os logs do servidor para identificar a causa do erro.
- Certifique-se de que a conexão com o banco de dados está configurada corretamente.