UniSENAI

Módulo 8: APIs RESTful com Node.js e MySQL

Módulo 8: APIs RESTful com Node.js e MySQL

Objetivo:

Ensinar como estruturar, criar e documentar APIs RESTful robustas, integrando o Express.js ao banco de dados MySQL e implementando boas práticas para validação e organização de dados.

Conteúdo Detalhado:

Aula 8.1: Estruturando uma API RESTful com Express

O que é uma API RESTful?

- API (Application Programming Interface): Interface para comunicação entre sistemas.
- RESTful: Segue os princípios REST, como o uso de verbos HTTP, URLs organizadas e respostas padronizadas.

Estrutura do Projeto

- Crie as seguintes pastas e arquivos:

```
projeto-api/
|-- controllers/
|-- models/
|-- routes/
|-- app.js
|-- package.json
```

Exemplo de Configuração do Servidor

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

app.use(express.json());

app.listen(3000, () => {
  console.log('API rodando na porta 3000');
});
```

Aula 8.2: Criando endpoints para CRUD no MySQL

Modelo de Dados

- Exemplo de estrutura de tabela no MySQL:

```
CREATE TABLE usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255),
  email VARCHAR(255),
  senha VARCHAR(255)
);
```

Endpoints para CRUD

1. **Create (POST)**

```
app.post('/usuarios', (req, res) => {
  const { nome, email, senha } = req.body;
  connection.query('INSERT INTO usuarios (nome, email, senha) VALUES (?, ?, ?)',
[nome, email, senha], (err) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.status(201).send('Usuário criado com sucesso!');
  });
});
```

2. **Read (GET)**

```
app.get('/usuarios', (req, res) => {
  connection.query('SELECT * FROM usuarios', (err, results) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.json(results);
  });
});
```

3. **Update (PUT)**

```
app.put('/usuarios/:id', (req, res) => {
  const { nome, email, senha } = req.body;
  connection.query('UPDATE usuarios SET nome = ?, email = ?, senha = ? WHERE id = ?',
[nome, email, senha, req.params.id], (err) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.send('Usuário atualizado com sucesso!');
  });
});
```

4. **Delete (DELETE)**

```
app.delete('/usuarios/:id', (req, res) => {
  connection.query('DELETE FROM usuarios WHERE id = ?', [req.params.id], (err) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.send('Usuário excluído com sucesso!');
  });
});
```

Aula 8.3: Validação de dados com bibliotecas como Joi

O que é Joi?

- Uma biblioteca para validação de esquemas e dados recebidos via requisições.

Instalação

```
npm install joi
```

Exemplo de Uso

```
const Joi = require('joi');

const schema = Joi.object({
  nome: Joi.string().min(3).required(),
  email: Joi.string().email().required(),
  senha: Joi.string().min(6).required()
});

app.post('/usuarios', (req, res) => {
  const { error } = schema.validate(req.body);
  if (error) {
    res.status(400).send(error.details[0].message);
    return;
  }

  const { nome, email, senha } = req.body;
  connection.query('INSERT INTO usuarios (nome, email, senha) VALUES (?, ?, ?)',
[nome, email, senha], (err) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.status(201).send('Usuário criado com sucesso!');
  });
});
```

Aula 8.4: Implementando paginação e filtros nas consultas

Paginação

```
app.get('/usuarios', (req, res) => {
  const { page = 1, limit = 10 } = req.query;
  const offset = (page - 1) * limit;

  connection.query('SELECT * FROM usuarios LIMIT ? OFFSET ?', [parseInt(limit),
parseInt(offset)], (err, results) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.json(results);
  });
});
```

Filtros

```
app.get('/usuarios', (req, res) => {
  const { nome } = req.query;
  const query = nome ? 'SELECT * FROM usuarios WHERE nome LIKE ?' : 'SELECT * FROM
usuarios';
  const params = nome ? [`%${nome}%`] : [];

  connection.query(query, params, (err, results) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.json(results);
  });
});
```

Aula 8.5: Documentação da API com Swagger

O que é Swagger?

- Ferramenta que facilita a documentação de APIs RESTful.

Instalação do Swagger-UI

```
npm install swagger-ui-express
```

Exemplo de Configuração

```
const swaggerUi = require('swagger-ui-express');
const swaggerDocument = require('./swagger.json');

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

Lista de Exercícios

Questões Teóricas

1. O que caracteriza uma API RESTful?
2. Quais são os verbos HTTP mais usados em APIs e suas funções?
3. Explique a vantagem de estruturar projetos em pastas como `controllers`, `models` e `routes`.
4. O que é Joi e para que serve?
5. Como a paginação melhora a performance de uma API?
6. Qual é a finalidade do Swagger em projetos de API?
7. O que é o conceito de filtros em consultas?
8. Explique como implementar um endpoint seguro para criar registros no MySQL.
9. Quais são os principais benefícios de validar dados na camada da API?
10. Como o Express simplifica o desenvolvimento de APIs RESTful?

Questões Práticas

1. Estruture um projeto para uma API RESTful com as pastas adequadas.
2. Implemente um endpoint para listar todos os registros de uma tabela.
3. Crie um endpoint para inserir um novo registro no banco de dados, validando os dados com Joi.
4. Adicione paginação a um endpoint que lista registros.
5. Crie um endpoint que permita atualizar registros pelo ID.
6. Implemente um endpoint para excluir registros com base no ID.
7. Configure um filtro para buscar registros pelo nome em uma tabela.
8. Documente sua API com Swagger, incluindo os endpoints criados.
9. Teste a validação de dados no backend enviando requisições com dados inválidos.
10. Configure um middleware global que registre os logs de cada requisição recebida.

Instruções para a Entrega das Atividades

1. **Elaboração e Envio do Arquivo**
   - Responda todas as questões de forma clara e objetiva.
   - Gere um arquivo no formato **.PDF** contendo as respostas de cada questão.
   - Envie o arquivo para os e-mails dos professores responsáveis.
2. **Validação da Atividade**
   - Após o envio do arquivo, procure o(s) professor(es) para realizar a validação da atividade.
   - **Não inicie a próxima atividade sem antes validar a anterior com o professor.**
3. **Forma de Validação**
   - **Explicação Verbal**: Explique cada resposta verbalmente ao(s) professor(es).
   - **Perguntas e Respostas**: Esteja preparado para responder aos questionamentos do(s) professor(es) sobre o conteúdo das respostas.
   - **Orientação**: Receba orientações sobre a apresentação do(s) tema(s).