COM222 DESENVOLVIMENTO DE SISTEMAS WEB

Aula 14: Programação Server-side

Node + MySQL

- Conteúdo
  - Criação e população de banco relacional com Node
  - Criação de uma API server-side
  - Utilização dos verbos HTTP para acesso à API
    - GET
    - POST
    - DELETE
    - PATCH

Criação do banco de dados

- Abra phpMyAdmin e crie o banco nodetest
- Crie o projeto node
  - mkdir nodemysql
  - cd nodemysql
  - npm init
- Arquivo package.json será criado

```
{
  "name": "nodemysql",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Node + MySQL

- Na pasta do projeto, digite
  - npm install –S mysql
- Comando instala a extensão que permite usar Node com MySql
- Abra o VS Code e crie o arquivo create-table.js
- Este arquivo será usado para conectar com o banco, criar e popular uma tabela. Insira o código a seguir no arquivo.

```
const mysql  = require('mysql');
const connection = mysql.createConnection({
  host     : 'localhost',
  port     : 3306,
  user     : 'root',
  password : '',
  database : 'nodetest'
});
```

Node + MySQL

- Vamos agora escrever código para criar e popular uma tabela

```
function createTable(conn){

    const sql = "CREATE TABLE IF NOT EXISTS Clientes (\n"+
                "ID int NOT NULL AUTO_INCREMENT,\n"+
                "Nome varchar(150) NOT NULL,\n"+
                "CPF char(11) NOT NULL,\n"+
                "PRIMARY KEY (ID)\n"+
                ");";

    conn.query(sql, function (error, results, fields){
        if(error) return console.log(error);
        console.log('criou a tabela!');
    });
}
```

Node + MySQL

- Vamos agora escrever código para criar e popular uma tabela

```
function addRows(conn){
  const sql = "INSERT INTO Clientes(Nome,CPF) VALUES ?";
  const values = [
        ['Antonio Cintra', '12345678901'],
        ['Paulo Silveira', '09876543210'],
        ['Ricardo Almeida', '12312312399']
      ];
  conn.query(sql, [values], function (error, results, fields){
          if(error) return console.log(error);
          console.log('adicionou registros!');
          conn.end();//fecha a conexão
      });
}
```

Node + MySQL

- Agora basta inserir o código responsável por conectar e executar as funções

```
connection.connect(function(err){
    if(err) return console.log(err);
    console.log('conectou!');
    createTable(connection);
    addRows(connection);
})
```

Criando uma API

- Agora que já temos o banco funcionando, podemos criar uma API para acessá-lo
  - Usaremos Express
    - “O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel.”
    - Facilita a criação de aplicações server-side e APIs
- Vamos começar adicionando a dependência do Express
  - npm install –S express body-parser

Criando uma API

- Agora vamos criar o arquivo index.js
  - Responsável por tratar as requisições
- Vamos começar definindo constantes e configurando o Express para usar o body-parser
  - Vai permitir posts no formato URLEncoded e JSON

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const port = 3000; //porta padrão
const mysql = require('mysql');

//configurando o body parser para pegar POSTS mais tarde
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

Criando uma API

- Em seguida, vamos criar um roteador e dentro dele definir uma regra inicial que apenas exibe uma mensagem de sucesso quando o usuário requisitar um GET na raiz da API (/)

```
//definindo as rotas
const router = express.Router();
router.get('/', (req, res) => res.json({ message: 'Funcionando!' }));
app.use('/', router);  // requisições que chegam na raiz devem ser enviadas para o router
```

- Finalmente, adicionamos o código para dar o start no servidor da API

```
//inicia o servidor
app.listen(port);
console.log('API funcionando!');
```

Criando uma API

- Para testar a API, execute no console
  - node index.js
- Acesse
  - Localhost:3000

Criando a listagem de clientes

- Agora que a API está funcionando, podemos adicionar rotas
- Vamos adicionar a rota /clientes para listar os clientes cadastrados no banco (select)
  - Para isso, vamos criar uma função que executa queries SQL fazendo conexão a cada uso
    - Existem técnicas mais eficientes, que serão vistas mais à frente
    - Essa função será usada para fazer todas as operações no banco

Criando a listagem de clientes


```
// função deve ficar no final do arquivo
function execSQLQuery(sqlQry, res){
  const connection = mysql.createConnection({
    host     : 'localhost',
    port     : 3306,
    user     : 'root',
    password : '',
    database : 'nodetest'
  });

  connection.query(sqlQry, function(error, results, fields){
      if(error)
        res.json(error);
      else
        res.json(results);
      connection.end();
      console.log('executou!');
  });
}
```

Criando a listagem de clientes

- Agora vamos criar a rota /clientes logo abaixo da rota raiz

```
router.get('/clientes', (req, res) =>{
    execSQLQuery('SELECT * FROM Clientes', res);
})
```

- Após salvar e executar node index.js, podemos consultar os clientes no browser
  - localhost:3000/clientes

Pesquisa de clientes

- Vamos agora pesquisar clientes pelo código
  - Para isso, vamos modificar a rota criada no passo anterior para aceitar um parâmetro opcional ID

```
router.get('/clientes/:id?', (req, res) =>{
    let filter = '';
    if(req.params.id) filter = ' WHERE ID=' + parseInt(req.params.id);
    execSQLQuery('SELECT * FROM Clientes' + filter, res);
})
```

Exclusão de clientes

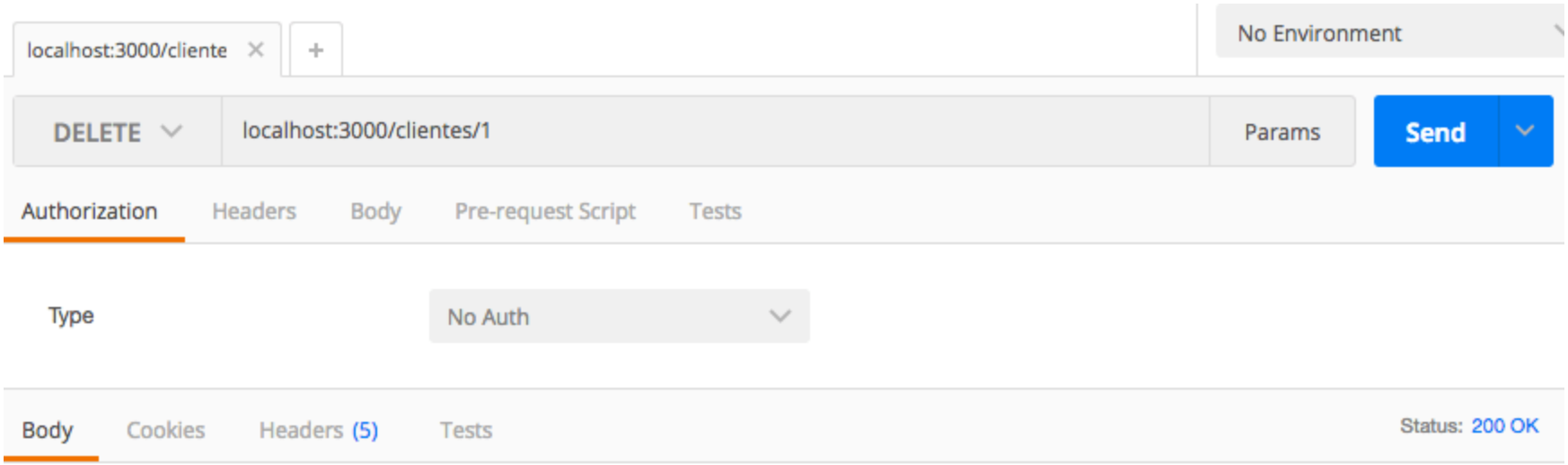
- Na exclusão, vamos fazer um procedimento parecido com a pesquisa, porém mudando o verbo HTTP de GET para DELETE
  - Vamos usar a rota /delete/clientes

```
router.delete('/delete/clientes/:id', (req, res) =>{
    execSQLQuery('DELETE FROM Clientes WHERE ID=' + parseInt(req.params.id), res);
})
```

- Para testar DELETE e POST temos que usar o POSTMAN
  - https://www.getpostman.com/downloads/

Exclusão de clientes

- Para testar DELETE e POST temos que usar o POSTMAN
  - https://www.getpostman.com/downloads/



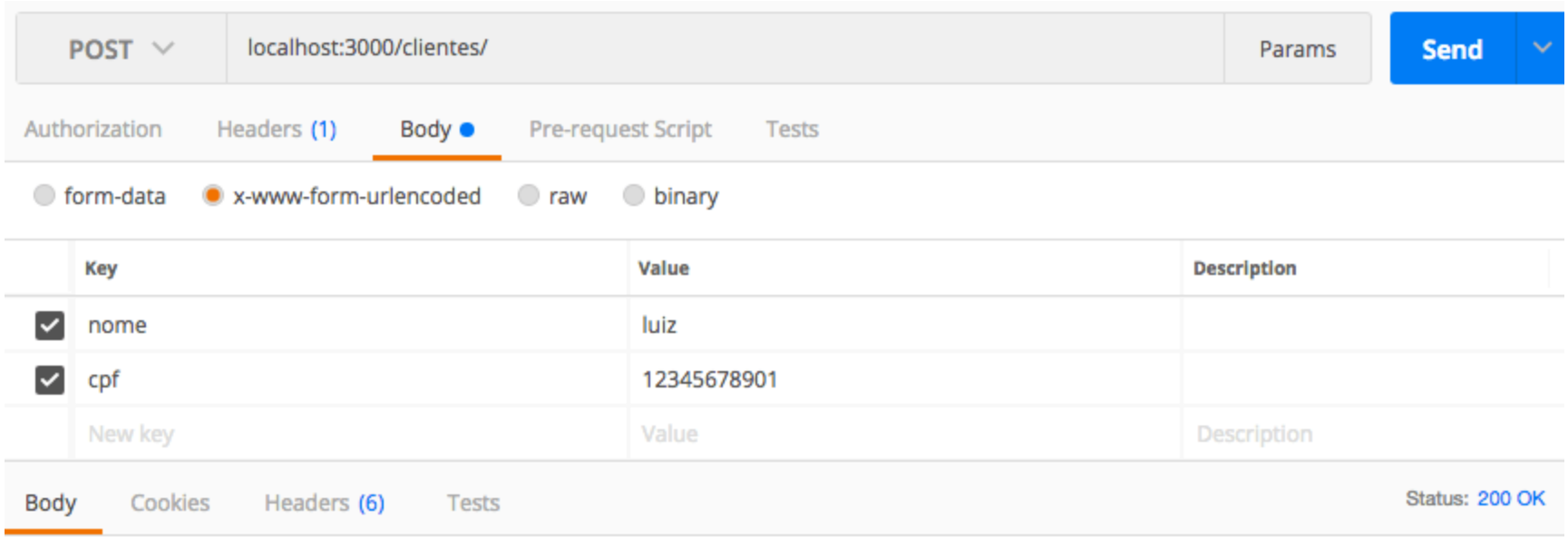
Adição de clientes

- Vamos adicionar clientes usando um POST na rota /add/clientes

```
router.post('/add/clientes', (req, res) =>{
    const nome = req.body.nome.substring(0,150);
    const cpf = req.body.cpf.substring(0,11);
    execSQLQuery(`INSERT INTO Clientes(Nome, CPF) VALUES('${nome}','${cpf}')`, res);
});
```

Adição de clientes

- Novamente temos que usar o POSTMAN
  - Dessa vez, temos que especificar os dados do cliente no Body do POST



Atualização de clientes

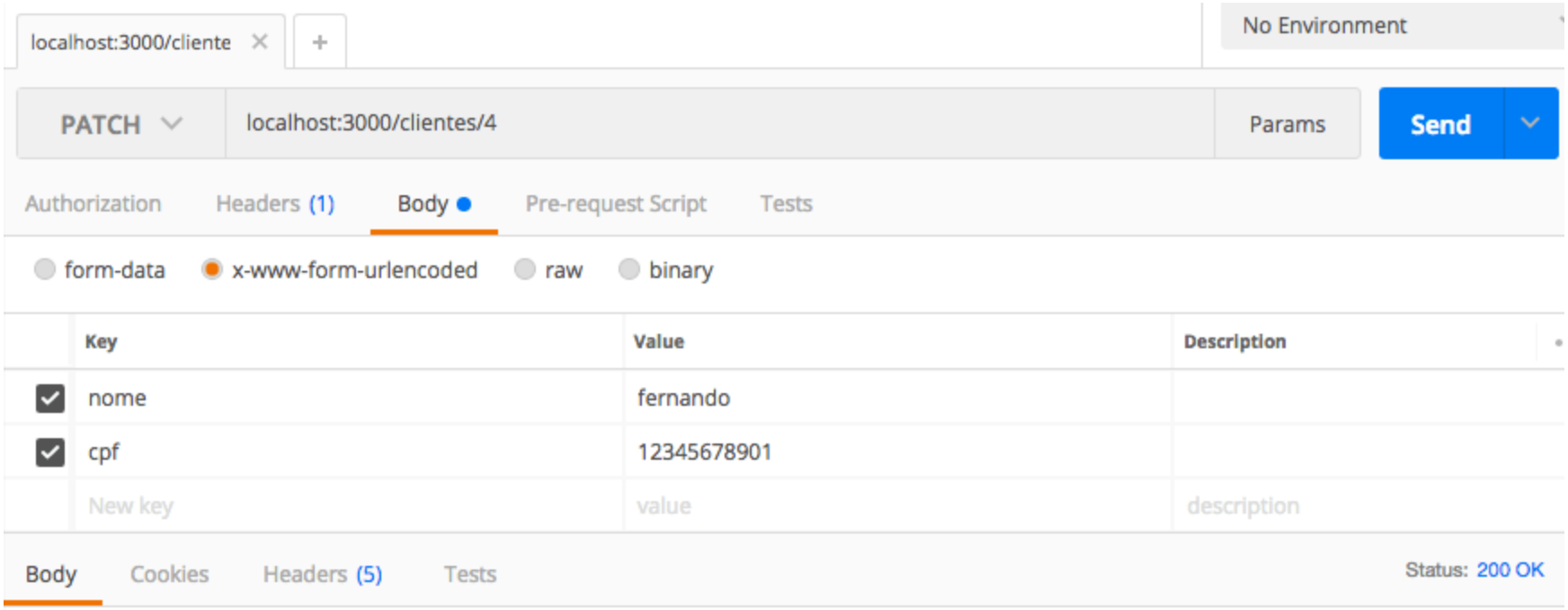
- Para atualizar clientes, temos que usar PUT ou PATCH
  - PUT atualiza todos os campos, então vamos usar PATCH
  - Vamos criar a seguinte rota PATCH: /update/clientes
    - Essa rota utiliza o ID do cliente para localizá-lo

```
router.patch('/update/clientes/:id', (req, res) =>{
    const id = parseInt(req.params.id);
    const nome = req.body.nome.substring(0,150);
    const cpf = req.body.cpf.substring(0,11);
    execSQLQuery(`UPDATE Clientes SET Nome='${nome}', CPF='${cpf}' WHERE ID=${id}`, res);
})
```

- No código acima, pegamos o ID que veio na URL e as demais informações que vieram no corpo da requisição
  - Em seguida, o UPDATE é montado com as variáveis locais executado

Atualização de clientes

- Usando o POSTMAN



Exercício para entregar

- Considere o seguinte códiog sql
  - CREATE TABLE aluno (codigo INTEGER NOT NULL, nome VARCHAR(25), dt_nasc DATE, PRIMARY KEY (codigo)) ENGINE = INNODB;
  - CREATE TABLE discip (codigo INTEGER NOT NULL, nome VARCHAR(25), creditos INTEGER, PRIMARY KEY (codigo)) ENGINE = INNODB;
  - CREATE TABLE matricula (codAluno INTEGER NOT NULL, codDiscip INTEGER NOT NULL, PRIMARY KEY (codAluno, codDiscip), INDEX (codAluno), FOREIGN KEY (codAluno) REFERENCES aluno(codigo) ON DELETE RESTRICT, INDEX (codDiscip), FOREIGN KEY (codDiscip) REFERENCES discip(codigo) ON DELETE RESTRICT) ENGINE = INNODB;

Exercício para entregar

- Implemente uma API que realize matrícula de alunos em disciplinas e permita listar os alunos matriculados em uma dada disciplina:
  1. Inserção de registros na tabela matrícula
  2. Listagem dos **nomes** dos alunos matriculados em uma dada disciplina (usar código da disciplina para permitir seleção)