# Acessando MySQL via Middleware

## EC021 - Tópicos Avançados II

## Sistemas Distribuídos

Prof. Adauto Mendes | E-mail: adauto.mendes@inatel.br

# Introdução

- O NodeJS fornece uma biblioteca própria para se fazer conexão com banco de dados.
- Para esta prática, usaremos a biblioteca chamada ‘mysql’

# Instalação

- Para instalar esta biblioteca, abra o terminar na pasta do seu projeto e execute o comando:

```
npm install mysql
```

- Com a instalação bem sucedida, o seu arquivo package.json ficará com as dependências atualizadas:

```
"dependencies": {
   "mysql": "^2.15.0",
   "restify": "^6.3.4"
}
```

# Configuração

- No início seu arquivo index.js, importe a biblioteca do mysql:

```
var mysql = require('mysql');
```

- Agora criaremos um objeto que irá armazenar as credenciais de conexão com o BD:

```
var con = {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'ec021'
};
```

# Método – Inserir (1)

- Com a conexão configurada, inicialmente faremos o método “inserir”:

```
/**
 * Montando um objeto toddy com
 * os dados que vieram do body da request
*/
var toddy = {
    lote: req.body.lote,
    conteudo: req.body.conteudo,
    validade: req.body.validade
}
```

# Método – Inserir (2)

- Abriremos a conexão e escreveremos a query:

```
/** Abrindo a conexão com o BD */
var connection = mysql.createConnection(con);
connection.connect();

/** Escrevendo query que será executada */
var strQuery = "INSERT INTO toddy (lote, conteudo, validade)" +
               "VALUES ('"+ toddy.lote + "', " + toddy.conteudo +
               ", '" + toddy.validade + "');";

/** Exibindo query no console */
console.log(strQuery);
```

# Método – Inserir (3.1)

- Agora vamos executar a query, e processar o resultado

```
/** Executando query e processando resultados */
connection.query(strQuery, function(err, rows, fields) {
    if (!err) {
        res.json(rows);
    } else {
        res.json(err);
    }
});
```

# Método – Inserir (3.2)

- Vamos entender o que foi feito aqui:

```
/** Executando query e processando resultados */
connection.query(strQuery, function(err, rows, fields) {
```

- Estamos passando 2 parâmetros para função ‘query’ do objeto ‘connection’.
  - **O primeiro parâmetro** é a query que vamos executar;
  - **O segundo parâmetro** é o que chamamos de função de call-back: função que será executada para processar e retornar os resultados da query. Ela recebe 3 parâmetros:
    - **err** => eventuais mensagens de erro;
    - **rows** => linhas que foram criadas/alteradas/selecionadas na execução da query;
    - **fields** => campos da tabela que foram selecionados.

# Método – Inserir (3.3)

- Sendo assim, usamos o IF abaixo para garantir que nossa query não causou nenhum erro, e caso cause erro, retornaremos um jSon contendo dados do erro causado.

```
if (!err) { //Se não houver erros
        res.json(rows); //Retornamos as linhas
} else { //Caso contrário
        res.json(err); //Retornamos dados sobre o erro
}
```

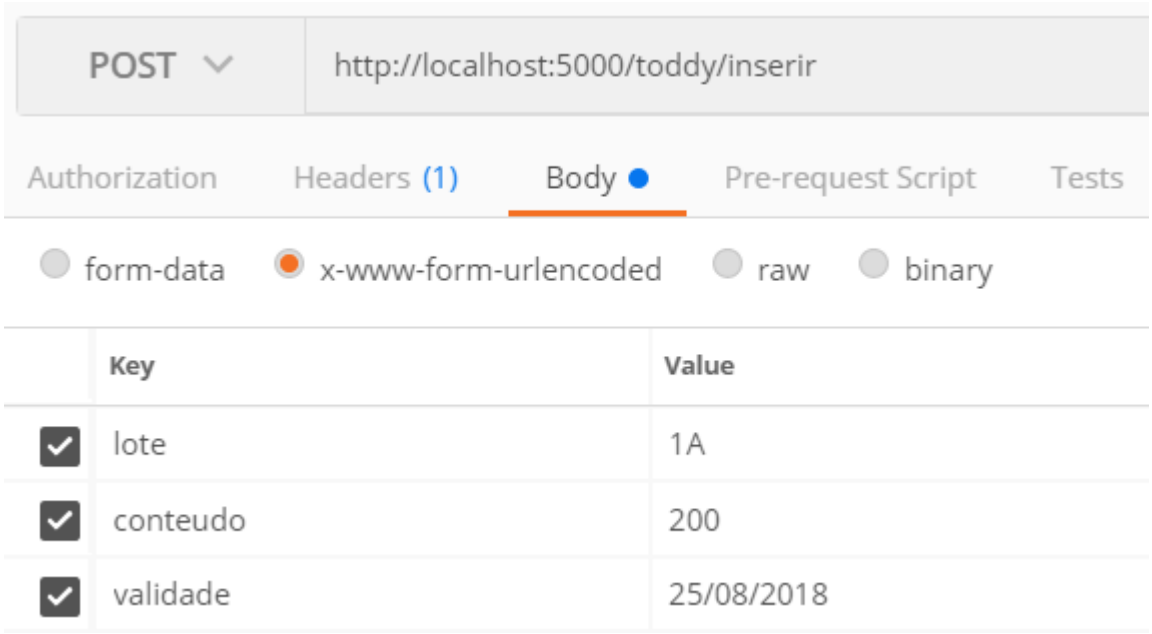
# Método – Inserir (4)

- Por fim, encerramos as conexões:

```
/** Encerrando conexão com o BD */
connection.end();

/** Encerrando método da REST API */
next();
```

# Método – Inserir (5)

- Vamos ao PostMan! Configuraremos a nossa **request** assim:



# Método – Inserir (6.1)

- Receberemos este resultado:


```
{
    "fieldCount": 0,
    "affectedRows": 1,
    "insertId": 1,
    "serverStatus": 2,
    "warningCount": 0,
    "message": "",
    "protocol41": true,
    "changedRows": 0
}
```

# Método – Inserir (6.2)

- Informações importantes:

```
{
    "fieldCount": 0,
    "affectedRows": 1,
    "insertId": 1,
    "serverStatus": 2,
    "warningCount": 0,
    "message": "",
    "protocol41": true,
    "changedRows": 0
}
```

Quantidade de campos selecionados (em um INSERT não selecionamos nada)

# Método – Inserir (6.3)

- Informações importantes:

```
{
    "fieldCount": 0,
    "affectedRows": 1,
    "insertId": 1,
    "serverStatus": 2,
    "warningCount": 0,
    "message": "",
    "protocol41": true,
    "changedRows": 0
}
```

Quantidade de linhas alteradas (inserimos apenas uma linha).

# Método – Inserir (6.4)

- Informações importantes:

```
{
    "fieldCount": 0,
    "affectedRows": 1,
    "insertId": 1,
    "serverStatus": 2,
    "warningCount": 0,
    "message": "",
    "protocol41": true,
    "changedRows": 0
}
```

ID da linha inserida (baseada na informação de auto incremento que o MySQL fornece).

# Método – Listar (1)

- O método listar seguirá a mesma ideia do INSERT (exceto porque nossa **request** não terá **body**, por ser **GET**):

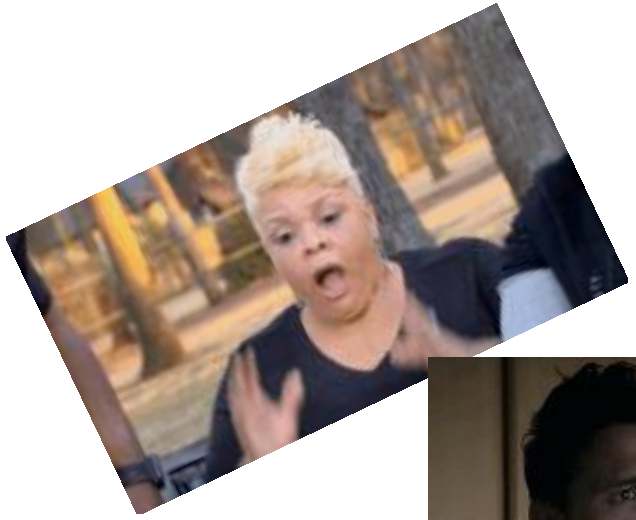
```
/** Abrindo a conexão com o BD */
var connection = mysql.createConnection(con);
connection.connect();

/** Escrevendo query que será executada */
var strQuery = "SELECT id, lote, conteudo, validade FROM toddy;";
```

# Método – Listar (2.1)

```
/** Exibindo query no console */
console.log(strQuery);

/** Executando query e processando resultados */
connection.query(strQuery, function(err, rows, fields) {
    if (!err) { //Se não houver erros
        res.json(rows); //Retornamos as linhas
    } else { //Caso contrário
        res.json(err); //Retornamos dados sobre o erro
    }
});

/** Encerrando conexão com o BD */
connection.end();

/** Encerrando método da REST API */
next();
```

# Método – Listar (2.2)

- Espera ai!
- Nós vamos simplesmente enviar as ‘rows’ do jeito que vem do banco de dados?




# Método – Listar (2.3)

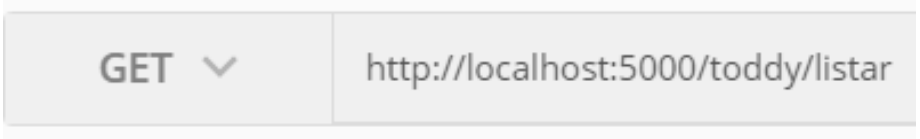
- A resposta é **SIM**!
- E por que?

O papel do MIDDLEWARE é funcionar como uma interface responsável por interceptar diferenças operacionais mascarando uma integração de várias linguagens e padrões de comunicação através de processos bem definidos. Porém não ficará a cargo dele processar este tipo de dado.

Seu papel é somente servir como um ponto de acesso aos dados, o processamento da **response** fica a cargo do cliente.

# Método – Listar (3)

- Executando nossa request no Postman:




- Temos este retorno (note que o resultado está entre [ ] o que denota que isto é um array):


```
[
    {
        "id": 2,
        "lote": "1A",
        "conteudo": 200,
        "validade": "25/08/2018"
    },
    {
        "id": 3,
        "lote": "2A",
        "conteudo": 2000,
        "validade": "25/08/2020"
    }
]
```

# Sua vez!

- Sua missão hoje é alterar os métodos:
  - Atualizar: passe o id, lote, conteúdo e data de validade do toddy, e atualize todas os campos baseados no ID;
  - Excluir: passe o ID e delete a linha referente àquele ID;
- Desafio: você deverá criar um 5º método GET para que seja possível buscar um toddy por ID.
  - Dica: você deverá incluir a linha abaixo após configurar o servidor, para ser possível receber parâmetros na URL:

```
server.use(restify.plugins.queryParser());
```