Desenv. BackEnd

Prof. M. Marcelo Petri

Olá! Sejam bem vindos

DISCIPLINA/UNIDADE CURRICULAR	Desenvolvimento BackEnd
TEMA DA AULA	“API (Node.js + MySQL)”
INDICADOR	4. Desenvolve tratamento de requisições POST e GET do front-end, de acordo com a linguagem de programação back-end. 8. Desenvolve recursos de chamadas assíncronas, de acordo com requisitos do projeto.
CRONOGRAMA	- Integração com front-end: tratamento de requisição GET e POST, processamento de dados de formulários e resposta ao front-end. - Chamadas assíncronas: integração com requisições assíncronas de front-end, padrões XML e Json.
ATIVIDADE	Desenvolvimento um servidor API Rest com acesso ao banco de dados
HABILIDADES	Codificar páginas utilizando as ferramentas de programação web.
CRITÉRIOS	Codificação

API

- **“Application Programming Interface”.**
- APIs são “tradutores” com a função de conectar sistemas, softwares e aplicativos.
- Uma API permite que sua solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados.
- Isso simplifica o desenvolvimento de aplicações, gerando economia de tempo e dinheiro.

REST

- **REST (Representational State Transfer):** é um conjunto de restrições utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura.

- Tipos de requisições HTTP:
  - **GET:** Usado para trazer informações
  - **POST:** Para criar/adicionar informações
  - **PUT:** Para atualizar informações
  - **DELETE:** Para excluir informações

REST

- Usa o protocolo HTTP (verbos, accept header, códigos de estado HTTP, Content-Type) de forma explícita e representativa para se comunicar. URIs são usadas para expor a estrutura do serviço. Utiliza uma notação comum para transferência de dados como XML e JSON.
- Não possui estado entre essas comunicações, ou seja, cada comunicação é independente e uniforme (padronizada) precisando passar toda informação necessária.
- Ele deve facilitar o cache de conteúdo no cliente.
- Deve ter clara definição do que faz parte do cliente e do servidor. O cliente não precisa saber como o servidor armazena dados, por exemplo. Assim cada implementação não depende de outra e se torna mais escalável.
- Permite o uso em camadas também facilitando a escalabilidade, confiabilidade e segurança.

REST API

- Há uma grande variação sobre as formas de utilização de APIS.

- As redes sociais, por exemplo, fornecem APIs que podem ser utilizadas em outros sites para recuperar as informações de uma página.

- Exemplo:
  - Realizar um comentário no facebook sem a necessidade de estar dentro da aplicação (comentários de blogs externos)

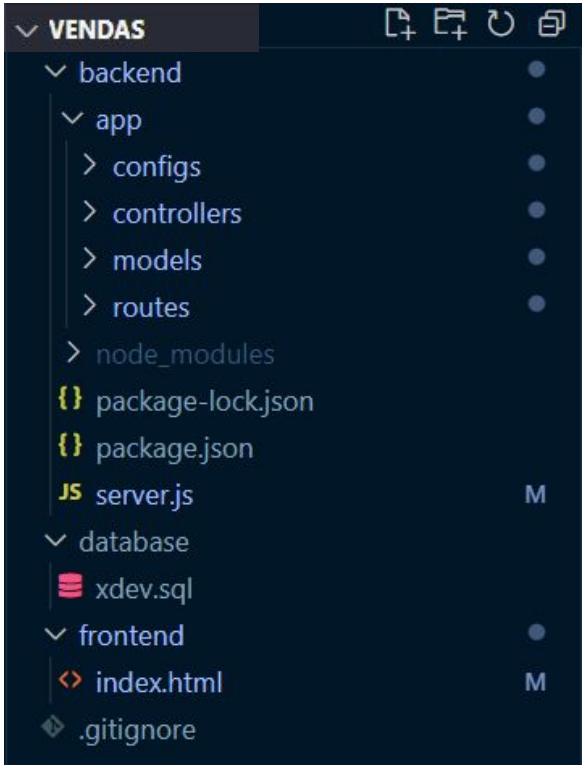
Express JS

- O Express.js é um framework Node que simplifica a criação de APIs para o Node.
- Ver exemplo projeto Calculadora

Ferramentas para Teste

- Instalação plugin do Chrome:
  - Advanced REST client
  - Postman
  - insomnia.rest/download

Projeto Vendas



Instalações Iniciais

- Bibliotecas
  - `npm install express mysql body-parser --save`

Servidor

- server.js

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

//parser para requisições content-type:
application/x-www-form-urlencoded-json
app.use(express.urlencoded({extended:true}));

app.get("/", (req, res) =>{
  res.json({
      message: "Bem vindo à API MVC do SENAC"
    })
});

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

Banco de Dados

```
CREATE DATABASE IF NOT EXISTS `xdev` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `xdev`;
```

Conexão com o Banco de Dados

- db.config.js

```
//Arquivo de configurações BD
module.exports = {
    HOST: "localhost",
    USER: "root",
    PASSWORD: "",
    DB: "xdev",
    PORT: "3306"
}
```

Configuração Módulo do MySQL

- db.js

```
//Modulo do MySQL
const mysql = require("mysql");
const dbConfig = require("../configs/db.config.js");

//Cria uma conexão com o BD
const connection = mysql.createConnection({
    host: dbConfig.HOST,
    user: dbConfig.USER,
    password: dbConfig.PASSWORD,
    database: dbConfig.DB,
     port: dbConfig.PORT
});

//Nova conexao com o MySQL
connection.connect(error=>{
    if (error) throw error;
    console.log("Banco de Dados Conectado!");
});
module.exports = connection;
```

Tabela Produtos

```
CREATE TABLE `produtos` (
  `idproduto` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(45) DEFAULT NULL,
  `valor` decimal(10,0) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
INSERT INTO `produtos` (`idproduto`, `nome`, `valor`) VALUES
(0, 'Camisa Polo', '50.00'),
(1, 'Sapatenis', '300.00'),
(2, 'Saia', '50.00'),
(3, 'Chinelo Havainas', '70.00'),
(4, 'Pijama', '90.00'),
(5, 'Boné', '30.00');
```

Projeto Vendas - Produto

- produto.model.js

```
const sql = require("./db.js");
//Construtor
const ProdutoModel = function(produto){
    this.nome = produto.nome;
    this.valor = produto.valor;
}
//Cria novo produto no banco
ProdutoModel.create = (produto, result) => {
};
//Seleciona produto por ID
ProdutoModel.findById = (produtoId, result) => {
};
//Seleciona todos os produtos
ProdutoModel.getAll = result => {
}
```

```
//Atualizar produto por id
ProdutoModel.updateById = (produtoId, produto,
result) => {
}
//Remover produto por id
ProdutoModel.remove = (produtoId, result) => {
}
//Remover todos os produtos
ProdutoModel.removeAll = (result) => {
}

module.exports = ProdutoModel;
```

Projeto Vendas - Produto

- produto.controller.js

```
const produtoModel =
require("../models/produto.model.js");

exports.create = (req, res) => {}

exports.findAll = (req, res) => {}

exports.findById = (req, res) => {}

exports.update = (req, res) => {}

exports.delete = (req, res) => {}

exports.deleteAll = (req, res) => {}
```

Projeto Vendas - Produto

- produto.routes.js

```
module.exports = app => {
    const produtoController = require("../controllers/produto.controller.js");

    app.post("/produtos", produtoController.create);
    app.get("/produtos", produtoController.findAll);
    app.get("/produtos/:produtoId", produtoController.findById);
    app.put("/produtos/:produtoId", produtoController.update);
    app.delete("/produtos/:produtoId", produtoController.delete);
    app.delete("/produtos", produtoController.deleteAll);
}
```

Projeto Vendas - Produto

- server.js
  - Adicionar a linha abaixo logo após a instância do app e reiniciar o servidor node:

```
require("./app/routes/produto.routes.js")(app);
```

Projeto Vendas - Produto

- produto.model.js
  - Implementar getAll

```
//Seleciona todos os produtos
ProdutoModel.getAll = result => {
    sql.query("SELECT * FROM produtos", (err, res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        console.log("produtos: ", res);
        result(null, res);
    })
}
```

Projeto Vendas - Produto

- produto.controller.js
  - Implementar findAll

```
exports.findAll = (req, res) => {
    produtoModel.getAll((err, data) => {
        if(err){
            res.status(500).send({
                message: err.message || "Ocorreu algum erro desconhecido!"
            });
        } else {
            res.send(data);
        }
    });
}
```

Projeto Vendas - Produto

- produto.model.js
  - Implementar findById

```
ProdutoModel.findById = (produtoId, result) => {
    sql.query("SELECT * FROM produtos WHERE idproduto =
"+produtoId, (err, res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        if (res.length){
            console.log("Produto Encontrado", res[0]);
            result(null, res[0]);
        } else {
            result({ kind: "not_found"}, null);
            console.log("Produto não encontrado");
        }
    })
};
```

Projeto Vendas - Produto

- produto.controller.js
  - Implementar findById

```
exports.findById = (req, res) => {
    produtoModel.findById(req.params.produtoId, (err, data) => {
        if (err) {
            if (err.kind == "not_found") {
                res.status(404).send({
                    message: "Produto não encontrado. ID: " +
req.params.produtoId
                });
            } else {
                res.status(500).send({
                    message: "Erro ao retornar o produto com ID: " +
req.params.produtoId
                });
            }
        } else {
            res.send(data);
        }
    });
}
```

Projeto Vendas - Produto

- INSERT - produto.model.js

```
//Cria novo produto no banco
ProdutoModel.create = (produto, result) => {
    sql.query("INSERT INTO produtos SET ?", produto, (err, res) =>{
        if (err){
            console.log("Erro: ", err);
            result(err, null);
            return;
        }
        console.log("Produto criado: ", {idproduto: res.insertId, ...produto});
        result(null, {idproduto: res.insertId, ...produto});
    })
};
```

Projeto Vendas - Produto

- INSERT - produto.controller.js

```
exports.create = (req, res) => {
    if (!req.body.nome || !req.body.valor) {
        res.status(400).send({
            message: "Conteúdo do corpo da
requisição vazia."
        });
    } else {
        const produto = new produtoModel({
            nome: req.body.nome,
            valor: req.body.valor
        });
```

```
        produtoModel.create(produto, (err, data) => {
            if (err) {
                res.status(500).send({
                    message: err.message || "Ocorreu
um erro"
                });
            } else {
                res.send(data);
            }
        });
    }
}
```

Projeto Vendas - Produto

- UPDATE- produto.model.js

```
//Atualizar produto por id
ProdutoModel.updateById = (produtoId, produto, result) => {
    sql.query("UPDATE produtos SET nome = ?, valor = ? WHERE idproduto = ?", [produto.nome, produto.valor, produtoId], (err, res) =>{
        if (err){
            console.log("erro: ", err);
            result(null, err);
        }
        else if (res.affectedRows == 0){
            result({ kind: "not_found"}, null);
        }
        else{
            console.log("Produto atualizado: ", {idproduto: produtoId, ...produto});
            result(null, {idproduto: produtoId, ...produto});
        }
    });
}
```

Projeto Vendas - Produto

- UPDATE- produto.controller.js

```
exports.update = (req, res) => {
    if (!req.body.nome || !req.body.valor) {
        res.status(400).send({
            message: "Conteúdo do corpo da
requisição vazia."
        });
    } else {
        const produto = new produtoModel({
            nome: req.body.nome,
            valor: req.body.valor
        });

produtoModel.updateById(req.params.produtoId,
produto, (err, data) => {
            if (err) {
                if (err.kind == "not_found") {
                    res.status(404).send({
                        message: "Produto não encontrado."
                    })
                } else {
                    res.status(500).send({
                        message: "Erro ao atualizar
produto."
                    })
                }
            } else {
                res.send(data);
            }
        });
    }
}
```

Projeto Vendas - Produto

- DELETE - produto.model.js

```
//Remover produto por id
ProdutoModel.remove = (produtoId, result) => {
    sql.query("DELETE FROM produtos WHERE idproduto = ?", produtoId, (err, res) =>
{
        if (err) {
            console.log("erro: ", err);
            result(err, null);
        } else if (res.affectedRows == 0) {
            result({ kind: "not_found" }, null);
        } else {
            result(null, res);
        }
    });
}
```

Projeto Vendas - Produto

- DELETE- produto.controller.js

```
exports.delete = (req, res) => {
    produtoModel.remove(req.params.produtoId, (err, data) => {
        if (err) {
            if (err.kind == "not_found") {
                res.status(404).send({ message: "Produto não encontrado." })
            } else {
                res.status(500).send({ message: "Erro ao deletar produto." })
            }
        } else {
            res.send({ message: "Produto deletado com sucesso" });
        }
    })
}
```

Projeto Vendas - Produto

- DELETE ALL - produto.model.js

```
//Remover todos os produtos
ProdutoModel.removeAll = (result) => {
    sql.query("DELETE FROM produtos ", (err, res) => {
        if (err) {
            console.log("erro: ", err);
            result(err);
        } else {
            result(null);
        }
    });
}
```

Projeto Vendas - Produto

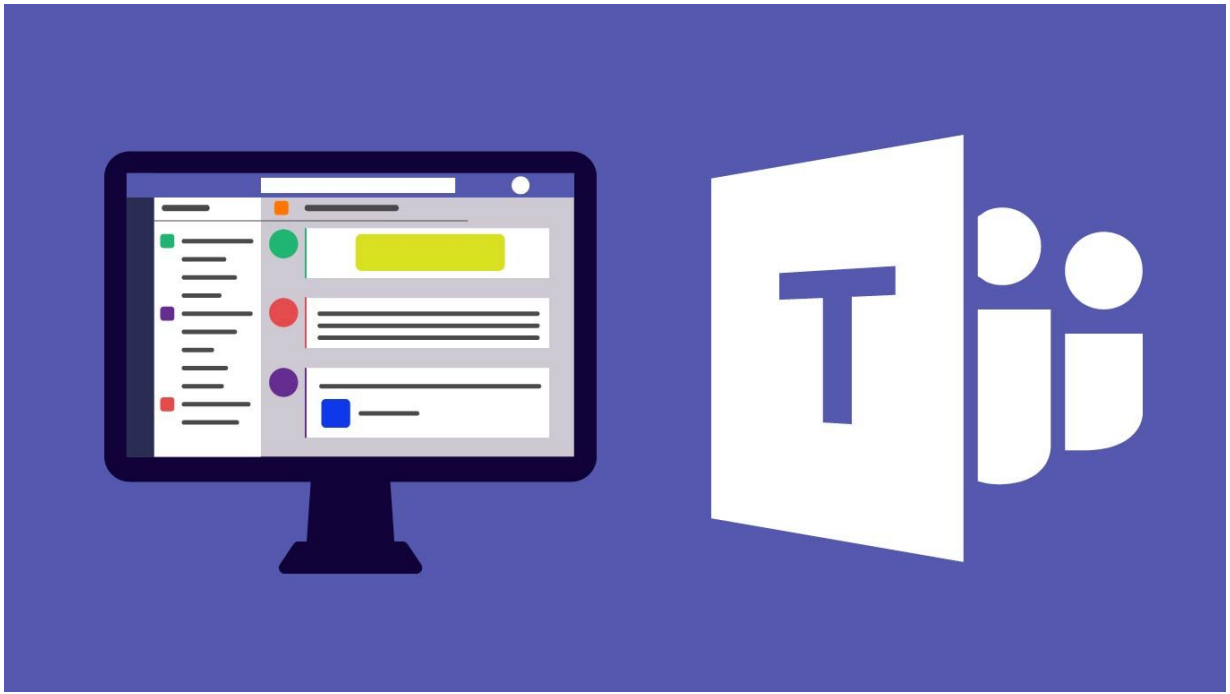
- DELETE ALL - produto.controller.js

```
exports.deleteAll = (req, res) => {
    produtoModel.removeAll((err, data) => {
        if (err) {
            res.status(500).send({ message: "Erro ao deletar produto." })
        } else {
            res.send({ message: "TODOS os Produtos deletados com sucesso" });
        }
    })
}
```

Projeto Vendas - Produto

- Testar

Atividades



Links

nodemon:

```
npm install -g nodemon
nodemon server.js
```

Por motivos de segurança, o PowerShell restringe a execução de scripts. Execute o comando (admin):

```
Set-ExecutionPolicy -ExecutionPolicy
RemoteSigned -Scope CurrentUser
```

Referências

- **Básicas: [OBRIGATORIAMENTE 3 BIBLIOGRAFIAS]**
- ARAÚJO, Everton Coimbra de. Desenvolvimento para web com Java. Florianópolis: Visual Books, 2010.
- QIAN et al. Desenvolvimento Web Java. LTC, 2010.
- GOMES, Yuri Marx Pereira. Java na web com JSF, Spring, Hibernate e Netbeans 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.

- **Complementar: [OBRIGATORIAMENTE 5 BIBLIOGRAFIAS]**
- GONÇALVES, Edson. Desenvolvendo aplicações web com netbeans ide 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.
- LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. Programação Java para Web. 2ª Ed. São Paulo: Novatec, 2015.
- SOUZA, Thiago Hernandes de. Java mais primefaces mais ireport: desenvolvendo um CRUD para web. Rio de Janeiro, RJ: Ciência Moderna, 2013.
- KALIN, Martin. Java Web Services: implementando. Rio de Janeiro, RJ: Alta Books, 2010.
- EIS, Diego. Guia Front-End: o caminho das pedras para ser um Dev Front-End. São Paulo: Casa do Código, 2015.
- SOUZA, Alberto. Spring MVC: domine o principal framework web Java. São Paulo: Casa do Código, 2015.