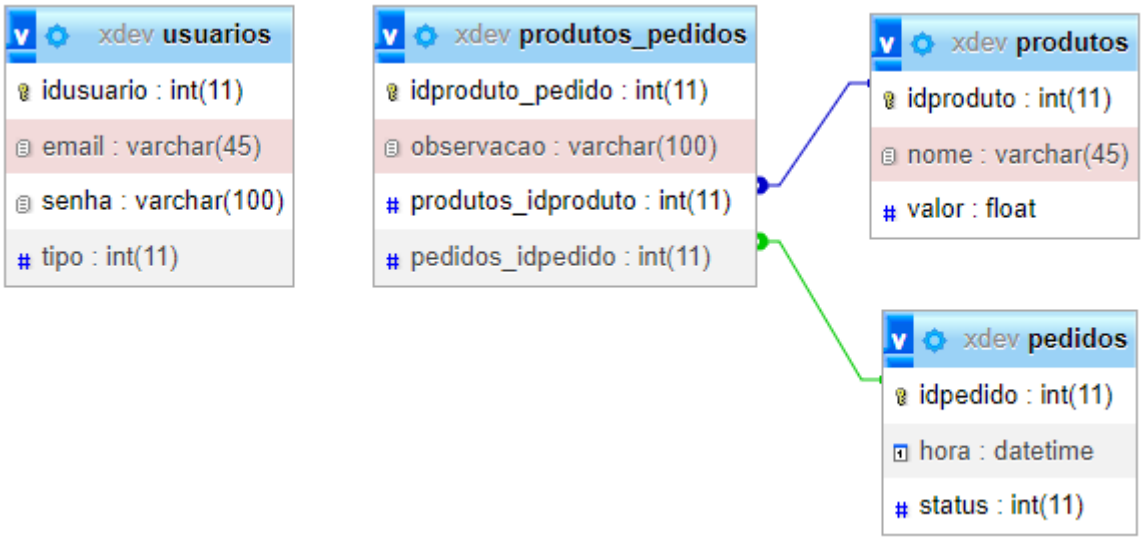
Projeto Lanchonete

Objetivo: Desenvolver uma aplicação web chamada Lanchonete que consome um serviço backend A aplicação consiste em um CRUD de produtos, pedidos e relação entre produto e pedido, exibindo-as em uma interface de usuário atraente. A aplicação terá autenticação do usuário, assim como um CRUD de usuários

Modelo Entidade e Relacionamento (MER)

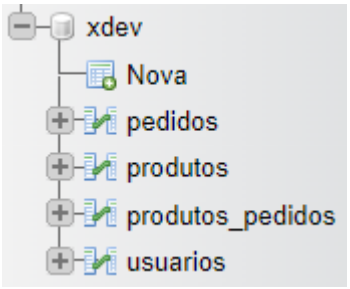


Script Banco de Dados xdev

Iniciar o XAMPP e criar a database xdev

```
create database xdev
```

Importar para o MySQL o arquivo **xdev.sql**



Desenvolvendo o BackEnd

O que é uma Application Programming Interface (API)?

- 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.

O que é REST (Representational State Transfer)?

- É um conjunto de restrições utilizadas para que as requisições HTTP atendam às diretrizes definidas na arquitetura.
- 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.
- Tipos de requisições HTTP:
    - **GET**: Usado para trazer informações
    - **POST**: Para criar/adicionar informações
    - **PUT**: Para atualizar informações
    - **DELETE**: Para excluir informações

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)

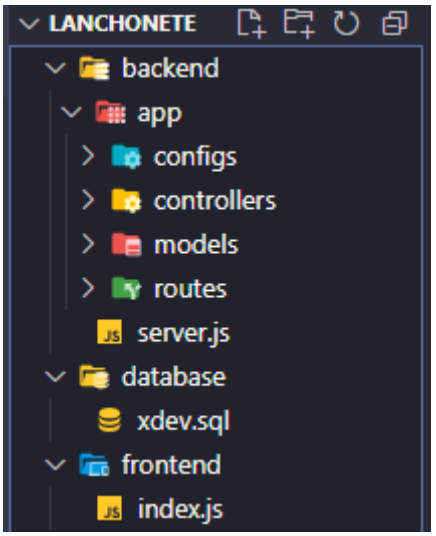
Express JS

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

Ferramentas auxiliares

- Advanced REST client
- Postman
- insomnia.rest/download

Configuração do Projeto Lanchonete (BackEnd)



Bibliotecas

Dentro da pasta backend instale os pacotes a seguir, após o comando será criado automaticamente a pasta node_modules e os arquivos package-lock.json e package.json

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

Codificação do Projeto

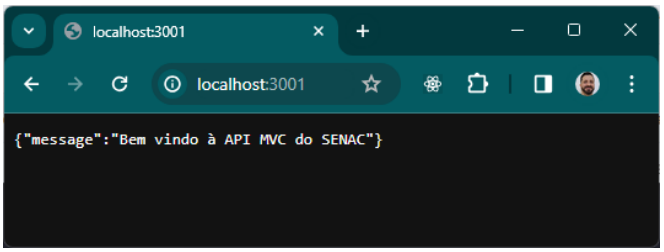
Vamos primeiramente criar o servidor que rodará nossa api backend, no arquivo server.js adicione o seguinte código:

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(3001, () => {
    console.log("Servidor rodando na porta 3001");
});
```

Com o comando **`npm start`** podemos testar se o servidor está rodando corretamente no endereço http://localhost:3001.

No navegador deverá aparecer a seguinte mensagem:



Para a conexão com o banco de dados MySql iremos criar dois arquivos de configuração que deverá ficar na pasta **configs** do projeto. O primeiro com o nome db.config.js possui os parâmetros de acesso ao Banco de Dados. O segundo chamado db.js contém os módulos de acesso ao MySql.

db.config.js

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

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

CRUD Tabela Produtos

Utilizaremos o Modelo MVC (Model, View, Controller) para as operações com a tabela Produtos.

A primeira parte a ser desenvolvida é o Model da tabela produtos, onde será especificado as ações que poderemos realizar com o banco de dados. Na pasta models cria o arquivo **produto.model.js.**

produto.model.js

```
const sql = require("../configs/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 = (id, result) => {
};
//Seleciona todos os produtos
ProdutoModel.getAll = result => {
}
//Atualizar produto por id
ProdutoModel.updateById = (id, produto, result) => {
}
//Remover produto por id
ProdutoModel.remove = (id, result) => {
}
//Remover todos os produtos
ProdutoModel.removeAll = (result) => {
}

module.exports = ProdutoModel;
```

A segunda etapa a ser desenvolvida é o Controller da tabela produtos, onde será especificado as regras de negócio e permissões de acesso. Na pasta controller criar o arquivo **produto.controller.js.**

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) => {}
```

Por fim criaremos as rotas de acesso da aplicação. Na pasta routes criar o arquivo **produto.routes.js**

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/:id", produtoController.findById);
    app.put("/produtos/:id", produtoController.update);
    app.delete("/produtos/:id", produtoController.delete);
    app.delete("/produtos", produtoController.deleteAll);
}
```

Definindo as configurações de acesso à tabela de produtos, precisamos informar ao servidor as novas rotas criadas. Adicionar a linha abaixo logo após a a declaração do express.urlencoded e reinicie o servidor node:

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}));

//linhas das rotas
require("./app/routes/produto.routes.js")(app);

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

app.listen(3001, () => {
    console.log("Servidor rodando na porta 3001");
});
```

Implementação do Model do Produto

Vamos alterar o arquivo produto.model.js, adicionando os métodos a seguir na estrutura já criada.

produto.model.js

```
//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);
    })
}
```

```
//Seleciona o produto por ID
ProdutoModel.findById = (id, result) => {
    sql.query("SELECT * FROM produtos WHERE idproduto = "+id, (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({ type : "not_found"}, null);
            console.log("Produto não encontrado");
        }
    })
};

//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});
    })
};

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

//Remover produto por id
ProdutoModel.remove = (id, result) => {
    sql.query("DELETE FROM produtos WHERE idproduto = ?", id, (err,
res) => {
        if (err) {
```

```
            console.log("erro: ", err);
            result(err, null);
        } else if (res.affectedRows == 0) {
            result({ type: "not_found" }, null);
        } else {
            result(null, res);
        }
    });
}

//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);
        }
    });
}
```

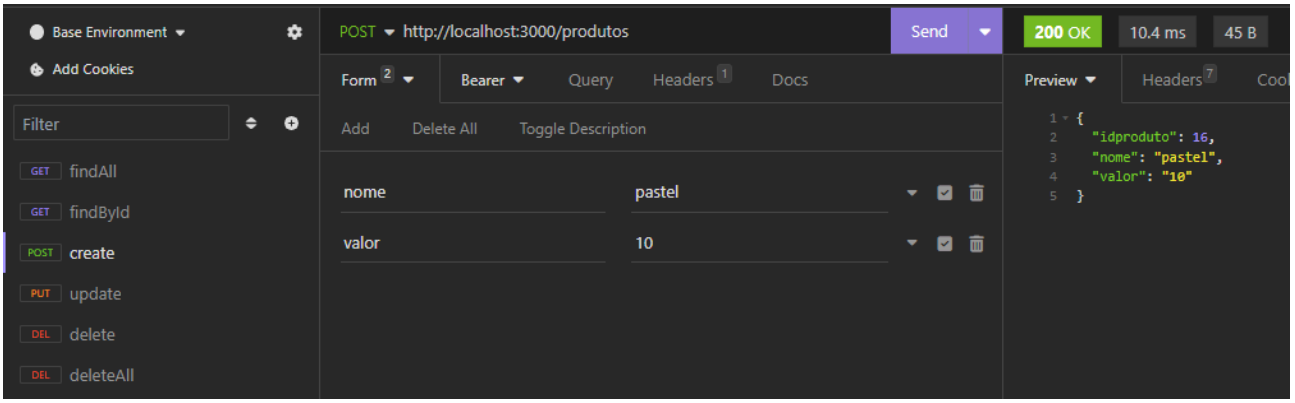
produto.controller.js

```
//Seleciona todos os produtos
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);
        }
    });
}

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

```
}

//Cria novo produto no banco
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);
            }
        });
    }
}
//Atualizar produto por id
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.id, produto, (err, data)
=> {
            if (err) {
                if (err.type == "not_found") {
                    res.status(404).send({
                        message: "Produto não encontrado."
                    })
                } else {
                    res.status(500).send({
                        message: "Erro ao atualizar produto."
                    })
                }
            } else {
                res.send(data);
            }
        });
    }
}

//Remover produto por id
exports.delete = (req, res) => {
    produtoModel.remove(req.params.id, (err, data) => {
```

```
        if (err) {
            if (err.type == "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" });
        }
    })
}

//Remover todos os produtos
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" });
        }
    })
}
```

Finalizado as etapas anteriores, agora vamos testar nossa api de cadastro de produtos utilizando o insomnia para criar as requisições http.



Agora é com vocês, faça a adaptação para a tabela Pedido

CRUD Tabela Pedidos

Utilizaremos o Modelo MVC (Model, View, Controller) para as operações com a tabela Pedidos.

Vamos criar o Model da tabela pedidos, onde será especificado as ações que poderemos realizar com o banco de dados. Na pasta models cria o arquivo **pedido.model.js.**

pedido.model.js

```
const sql = require("../configs/db.js");
```

```
//Construtor
const PedidoModel = function(pedido){
    this.hora = pedido.hora;
    this.status = pedido.status;
}
//Seleciona todos os pedidos
PedidoModel.getAll = result => {
    sql.query("SELECT * FROM pedidos", (err, res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        console.log("pedidos: ", res);
        result(null, res);
    })
}

//Seleciona o pedido por ID
PedidoModel.findById = (id, result) => {
    sql.query("SELECT * FROM pedidos WHERE idpedido = "+id, (err,
res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        if (res.length){
            console.log("Pedido Encontrado", res[0]);
            result(null, res[0]);
        } else {
            result({ type : "not_found"}, null);
            console.log("Pedido não encontrado");
        }
    })
};

//Cria novo pedido no banco
PedidoModel.create = (pedido, result) => {
    sql.query("INSERT INTO pedidos SET ?", pedido, (err, res) =>{
        if (err){
            console.log("Erro: ", err);
            result(err, null);
            return;
        }
        console.log("Pedido criado: ", {idpedido: res.insertId,
...pedido});
        result(null, {idpedido: res.insertId, ...pedido});
    })
};

//Atualizar pedido por id
PedidoModel.updateById = (id, pedido, result) => {
    sql.query("UPDATE pedidos SET hora = ?, status = ? WHERE
idpedido= ?", [pedido.hora, pedido.status, id], (err, res) =>{
        if (err){
            console.log("erro: ", err);
```

```
            result(null, err);
        }
        else if (res.affectedRows == 0){
            result({ type: "not_found"}, null);
        }
        else{
            console.log("Pedido atualizado: ", {idpedido: id,
...pedido});
            result(null, {idpedido: id, ...pedido});
        }
    });
}

//Remover pedido por id
PedidoModel.remove = (id, result) => {
    sql.query("DELETE FROM pedidos WHERE idpedido = ?", id, (err,
res) => {
        if (err) {
            console.log("erro: ", err);
            result(err, null);
        } else if (res.affectedRows == 0) {
            result({ type: "not_found" }, null);
        } else {
            result(null, res);
        }
    });
}

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


module.exports = PedidoModel;
```

Desenvolver o Controller da tabela pedidos, onde será especificado as regras de negócio e permissões de acesso. Na pasta controller criar o arquivo pedido.controller.js.

pedido.controller.js

```
const PedidoModel = require("../models/pedido.model.js");

//Seleciona todos os pedidos
exports.findAll = (req, res) => {
    PedidoModel.getAll((err, data) => {
        if(err){
            res.status(500).send({
                message: err.message || "Ocorreu algum erro
```

```
desconhecido!"
            });
        } else {
            res.send(data);
        }
    });
}

//Seleciona o pedido por ID
exports.findById = (req, res) => {
    PedidoModel.findById(req.params.id, (err, data) => {
        if (err) {
            if (err.type == "not_found") {
                res.status(404).send({
                    message: "Pedido não encontrado. ID: " +
req.params.id
                });
            } else {
                res.status(500).send({
                    message: "Erro ao retornar o pedido com ID: " +
req.params.id
                });
            }
        } else {
            res.send(data);
        }
    });
}

//Cria novo pedido no banco
exports.create = (req, res) => {
    if (!req.hora.nome || !req.status.valor) {
        res.status(400).send({
            message: "Conteúdo do corpo da requisição vazia."
        });
    } else {
        const pedido = new PedidoModel({
            hora: req.body.hora,
            status: req.body.status
        });
        PedidoModel.create(pedido, (err, data) => {
            if (err) {
                res.status(500).send({
                    message: err.message || "Ocorreu um erro"
                });
            } else {
                res.send(data);
            }
        });
    }
}
//Atualizar pedido por id
exports.update = (req, res) => {
    if (!req.body.hora || !req.body.status) {
        res.status(400).send({
            message: "Conteúdo do corpo da requisição vazia."
        });
```

```
    } else {
        const pedido = new PedidoModel({
            hora: req.body.hora,
            status: req.body.status
        });
        PedidoModel.updateById(req.params.id, pedido, (err, data) =>
{
            if (err) {
                if (err.type == "not_found") {
                    res.status(404).send({
                        message: "Pedido não encontrado."
                    })
                } else {
                    res.status(500).send({
                        message: "Erro ao atualizar pedido."
                    })
                }
            } else {
                res.send(data);
            }
        });
    }
}

//Remover produpedido por id
exports.delete = (req, res) => {
    PedidoModel.remove(req.params.id, (err, data) => {
        if (err) {
            if (err.type == "not_found") {
                res.status(404).send({ message: "Pedido não
encontrado." })
            } else {
                res.status(500).send({ message: "Erro ao deletar
pedido." })
            }
        } else {
            res.send({ message: "Pedido deletado com sucesso" });
        }
    })
}

//Remover todos os pedidos
exports.deleteAll = (req, res) => {
    PedidoModel.removeAll((err, data) => {
        if (err) {
            res.status(500).send({ message: "Erro ao deletar pedido."
})
        } else {
            res.send({ message: "TODOS os Pedidos deletado com
sucesso" });
        }
    })
}
```

Definir as rotas para Pedido

pedido.routes.js

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

    app.post("/pedidos", pedidoController.create);
    app.get("/pedidos", pedidoController.findAll);
    app.get("/pedidos/:id", pedidoController.findById);
    app.put("/pedidos/:id", pedidoController.update);
    app.delete("/pedidos/:id", pedidoController.delete);
    app.delete("/pedidos", pedidoController.deleteAll);
}
```

Adicionar as novas rotas do pedido no servidor

server.js

```
… codigo

//linhas das rotas
require("./app/routes/produto.routes.js")(app);
require("./app/routes/pedido.routes.js")(app);

… codigo
```

CRUD Tabela Produtos_Pedidos

Utilizaremos o Modelo MVC (Model, View, Controller) para as operações com a tabela de relacionamento Produtos_Pedidos.

Vamos criar o Model da tabela produtos_pedidos, onde será especificado as ações que poderemos realizar com o banco de dados. Na pasta models cria o arquivo **produto_pedido.model.js.**

produto_pedido.model.js.

```
const sql = require("../configs/db.js");
//Construtor
const ProdutoPedidoModel = function(produtos_pedidos){
    this.observacao = produtos_pedidos.observacao;
    this.produtos_idproduto = produtos_pedidos.produtos_idproduto;
    this.pedidos_idpedido = produtos_pedidos.pedidos_idpedido;
}
//Seleciona todos os produtos
ProdutoPedidoModel.getAll = result => {
    sql.query("SELECT * FROM produtos_pedidos", (err, res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        console.log("produtos_pedidos: ", res);
        result(null, res);
    })
```

```
}

//Seleciona o produto por ID
ProdutoPedidoModel.findById = (id, result) => {
    sql.query("SELECT * FROM produtos_pedidos WHERE idproduto_pedido
= "+id, (err, res) =>{
        if (err) {
            console.log("erro: ", err);
            result(null, err);
            return;
        }
        if (res.length){
            console.log("Produto_Pedido Encontrado", res[0]);
            result(null, res[0]);
        } else {
            result({ type : "not_found"}, null);
            console.log("Produto_Pedido não encontrado");
        }
    })
};

//Cria novo produto no banco
ProdutoPedidoModel.create = (produtos_pedidos, result) => {
    sql.query("INSERT INTO produtos_pedidos SET ?", produtos_pedidos,
(err, res) =>{
        if (err){
            console.log("Erro: ", err);
            result(err, null);
            return;
        }
        console.log("Produto_Pedido criado: ", {idproduto_pedido:
res.insertId, ...produtos_pedidos});
        result(null, {idproduto_pedido: res.insertId,
...produtos_pedidos});
    })
};

//Atualizar produto por id
ProdutoPedidoModel.updateById = (id, produtos_pedidos, result) => {
    sql.query("UPDATE produtos_pedidos SET observacao = ?,
produtos_idproduto = ?, pedidos_idpedido = ? WHERE idproduto_pedido=
?", [produtos_pedidos.observacao,
produtos_pedidos.produtos_idproduto,
produtos_pedidos.pedidos_idpedido, id], (err, res) =>{
        if (err){
            console.log("erro: ", err);
            result(null, err);
        }
        else if (res.affectedRows == 0){
            result({ type: "not_found"}, null);
        }
        else{
            console.log("Produto_Pedido atualizado: ",
{idproduto_pedido: id, ...produtos_pedidos});
            result(null, {idproduto_pedido: id,
...produtos_pedidos});
        }
```

```
    });
}

//Remover produto por id
ProdutoPedidoModel.remove = (id, result) => {
    sql.query("DELETE FROM produtos_pedidos WHERE idproduto_pedido =
?", id, (err, res) => {
        if (err) {
            console.log("erro: ", err);
            result(err, null);
        } else if (res.affectedRows == 0) {
            result({ type: "not_found" }, null);
        } else {
            result(null, res);
        }
    });
}

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

module.exports = ProdutoPedidoModel;
```

Desenvolver o Controller da tabela produtos_pedidos, onde será especificado as regras de negócio e permissões de acesso. Na pasta controller criar o arquivo produtos_pedidos.controller.js.

produtos_pedidos.controller.js

```
const ProdutoPedidoModel =
require("../models/produto_pedido.model.js");

//Seleciona todos os produtos_pedidos
exports.findAll = (req, res) => {
    ProdutoPedidoModel.getAll((err, data) => {
        if(err){
            res.status(500).send({
                message: err.message || "Ocorreu algum erro
desconhecido!"
            });
        } else {
            res.send(data);
        }
    });
}

//Seleciona o produto_pedido por ID
```

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

//Cria novo produto_pedido no banco
exports.create = (req, res) => {
    if (!req.body.observacao || !req.body.produtos_idproduto ||
!req.body.pedidos_idpedido) {
        res.status(400).send({
            message: "Conteúdo do corpo da requisição vazia."
        });
    } else {
        const produto_pedido = new ProdutoPedidoModel({
            observacao: req.body.observacao,
            produtos_idproduto: req.body.produtos_idproduto,
            pedidos_idpedido: req.body.pedidos_idpedido
        });
        ProdutoPedidoModel.create(produto_pedido, (err, data) => {
            if (err) {
                res.status(500).send({
                    message: err.message || "Ocorreu um erro"
                });
            } else {
                res.send(data);
            }
        });
    }
}
//Atualizar produto_pedido por id
exports.update = (req, res) => {
    if (!req.body.observacao || !req.body.produtos_idproduto ||
!req.body.pedidos_idpedido) {
        res.status(400).send({
            message: "Conteúdo do corpo da requisição vazia."
        });
    } else {
        const produto_pedido = new ProdutoPedidoModel({
            observacao: req.body.observacao,
            produtos_idproduto: req.body.produtos_idproduto,
            pedidos_idpedido: req.body.pedidos_idpedido
        });
```

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

//Remover produto por id
exports.delete = (req, res) => {
    ProdutoPedidoModel.remove(req.params.id, (err, data) => {
        if (err) {
            if (err.type == "not_found") {
                res.status(404).send({ message: "Produto_Pedido não
encontrado." })
            } else {
                res.status(500).send({ message: "Erro ao deletar
produto_pedido." })
            }
        } else {
            res.send({ message: "Produto_Pedido deletado com sucesso"
});
        }
    })
}

//Remover todos os produtos
exports.deleteAll = (req, res) => {
    ProdutoPedidoModel.removeAll((err, data) => {
        if (err) {
            res.status(500).send({ message: "Erro ao deletar
Produto_Pedido." })
        } else {
            res.send({ message: "TODOS os Produto_Pedido deletado com
sucesso" });
        }
    })
}
```

Definir as rotas para produtos_pedidos

produtos_pedidos.routes.js

```
module.exports = app => {
    const produto_pedidoController =
```

```
require("../controllers/produto_pedido.controller.js");

    app.post("/produtos_pedidos", produto_pedidoController.create);
    app.get("/produtos_pedidos", produto_pedidoController.findAll);
    app.get("/produtos_pedidos/:id",
produto_pedidoController.findById);
    app.put("/produtos_pedidos/:id",
produto_pedidoController.update);
    app.delete("/produtos_pedidos/:id",
produto_pedidoController.delete);
    app.delete("/produtos_pedidos",
produto_pedidoController.deleteAll);
}
```

Adicionar as novas rotas do pedido no servidor

server.js

```
… codigo

//linhas das rotas
require("./app/routes/produto.routes.js")(app);
require("./app/routes/pedido.routes.js")(app);
require("./app/routes/produto_pedido.routes.js")(app);

… codigo
```

Autenticação e Autorização

Em qualquer sistema seguro, precisamos garantir que SABEMOS QUEM SÃO as pessoas que estão utilizando e que APENAS AS PESSOAS CERTAS tenham autorização para acessar determinados arquivos.

- Autenticar um usuário é sabem QUEM ele é. (login no sistemas)
- Autorizar um usuário é saber O QUE ele pode acessar. (permissões)

No exemplo da lanchonete, podemos pensar em ao menos 3 tipos de usuários:

- **Administrador**: poderá registrar os produtos que existem na lanchonete.
- **Balcão**: poderá gerar um novo pedido
- **Cozinha**: terá apenas uma lista de pedidos a serem preparados e poderá modificar o status do pedido para ”pronto”, assim que estiver preparado

Todos esses tipos de usuários deverão estar devidamente autenticados no sistema (logados) para que acessem as determinadas telas. (autenticação)

Exemplo de Autorização:

Quando pedimos um lanche em um Drive, o primeiro atendente abre nosso pedido, indicando para a cozinha o que precisa ser preparado. Uma vez terminado o preparo do nosso pedido, o segundo atendente embala, enche o copo de refri caso precise e nos entrega o pedido.

Exemplo de Autenticação:

Todos esses usuários precisam estar logados no mesmo sistema para que tenham acesso às telas.

JSON

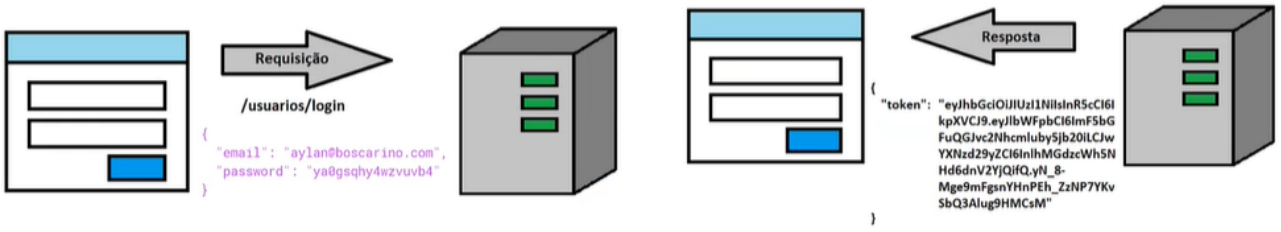
JavaScript Object Notation é um padrão de dados utilizados principalmente para troca de mensagens entre serviços/sistemas. Possui a característica “chave: valor” dos objetos javascript e padroniza a forma como trabalhamos com os dados.


```
[
    {
      "description": "quarter",
      "mode": "REQUIRED",
      "name": "qtr",
      "type": "STRING"
    },
    {
      "description": "sales representative",
      "mode": "NULLABLE",
      "name": "rep",
      "type": "STRING"
    },
    {
      "description": "total sales",
      "mode": "NULLABLE",
      "name": "sales",
      "type": "INTEGER"
    }
]
```


JSON Web Token (JWT)

É um padrão utilizado para autenticar usuários por meio de um token assinado. O usuário loga pela primeira vez e a API ou o sistema responde com um token (que foi criado com um segredo interno e possui uma data de expiração).

Depois disso, cada vez que for fazer uma nova requisição, o usuário envia esse token para que a API identifique que ele é quem ele diz ser.



Próximas atividades:

- Modificar o banco de dados para incluir a tabela de usuários e perfis;
- Criar rota, model e controller para usuários;
- Desenvolver método de login (verificar se usuário e senha estão corretos);
- configurar JWT no nodejs;
- Modificar o retorno do método de login para que ele responda com un JSON Web Token;
- Criar middleware que verifique se usuário está autenticado e se possui perfil com permissão suficiente para acessar determinado método cada vez que receber uma nova requisição.
- Regras:
    - Administração: tipo 1;

- Balcão: tipo 2;
- Cozinha: tipo 3.

Bibliotecas

Precisamos instalar 3 novos módulos para permitir a utilização do JWT:

- `npm install cors jsonwebtoken bcryptjs -save`

Script para a tabela de usuários

```
CREATE TABLE IF NOT EXISTS `xDev`.`usuarios` (
      `idusuarios` INT NOT NULL AUTO_INCREMENT,
      `email` VARCHAR(45) NOT NULL,
      `senha` VARCHAR(100) NOT NULL,
      `tipo` INT NOT NULL,
      PRIMARY KEY (`idusuarios`)
)
ENGINE = InnoDB;
```

Na pasta config devemos criar um novo arquivo chamado **auth.cofig.js**, onde será definida a chave secreta para acesso ao sistema web.

auth.cofig.js

```
//Palavra segredo
module.exports = {
    secret: "senac-chave-secreta"
}
```

CRUD da tabela Usuarios

usuario.model.js

```
const sql = require("../configs/db");

const Usuario = function (usuario){
    this.email = usuario.email;
    this.senha = usuario.senha;
    this.tipo = usuario.tipo;
}

Usuario.create = (usuario, result) => {
    sql.query("INSERT INTO usuarios SET ?", usuario, (err, res) => {
        if (err)
            result (err, null);
        else
            result(null, "Usuário criado com suscesso.");
    });
}

Usuario.findByEmail = (email, result) => {
    sql.query("SELECT * FROM usuarios WHERE email = ?", email, (err,
res) => {
```

```
        if (err)
            result(err, null);
        else if (res.length)
            result(null, res[0]);
        else
            result({type: "not_found"}, null);
    });
}

Usuario.findById = (id, result) => {
    sql.query("SELECT * FROM usuarios WHERE idusuario = ?", id, (err,
res) => {
        if (err)
            result(err, null);
        else if (res.length)
            result(null, res[0]);
        else
            result({type: "not_found"}, null);
    });
}

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

        console.log("usuarios: ", res);
        result(null, res);
    })
};

//Atualizar usuario por id
Usuario.updateById = (id, usuario, result) => {
    sql.query("UPDATE usuarios SET email = ?, senha = ?, tipo = ?
WHERE idusuario = ?",
              [usuario.email, usuario.senha, usuario.tipo, id], (err,
res) => {
                    if (err){
                        console.log("erro: ", err);
                        result(null, err);
                    } else if (res.affectedRows == 0){
                        result({ type: "not_found"}, null);
                    } else {
                        console.log("Usuário atualizado: ",
{idusuario: id, ...usuario});
                        result(null, {idusuario: id, ...usuario});
                    }
              });
};
//Remover usuario por id
Usuario.remove = (id, result) => {
    sql.query("DELETE FROM usuarios WHERE idusuario = ?", id, (err,
res) =>{
```

```
        if (err) {
            console.log("erro: ", err);
            result(err, null);
        } else if (res.affectedRows == 0){
            result({ type: "not_found"}, null);
        } else {
            result(null, res);
        }
    });
};

module.exports = Usuario;
```

usuario.controller.js

```
const usuarioModel = require("../models/usuario.model.js");
const bcrypt = require("bcryptjs");
const config = require("../configs/auth.config.js");
const jwt = require("jsonwebtoken");

exports.signUp = (req, res) => {
    if (!req.body.email || !req.body.senha || !req.body.tipo){
        res.status(400).send({
            message: "E-mail, senha ou tipo não enviados."
        })
    } else {
        const usuario = new usuarioModel({
            email: req.body.email,
            senha: bcrypt.hashSync(req.body.senha, 8),
            tipo: req.body.tipo
        })

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

exports.signIn = (req, res) => {
    usuarioModel.findByEmail(req.body.email, (err, data) => {
        if (err) {
            if (err == "not_found"){
                res.status(404).send({
                    message: "Não foi encontrado usuario com o email
digitado."
                })
            } else {
                res.status(500).send({
                    message: "Ocorreu um erro ao buscar email do
usuário no sistema."
                })
```

```
            }
        } else {
            let validPassword = bcrypt.compareSync(req.body.senha,
data.senha);
            if (!validPassword){
                res.status(401).send({
                    accessToken: null,
                    message: "Senha inválida!"
                })
            } else {
                let token = jwt.sign({id: data.idusuario},
config.secret, {expiresIn: 86400}); //24h
                res.status(200).send({
                    accessToken: token,
                    id: data.idusuario,
                    email: data.email,
                    tipo: data.tipo
                })
            }
        }

    })
};

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

exports.update = (req, res) => {
    if (!req.body.email || !req.body.senha || !req.body.tipo){
        res.status(400).send({
            message: "E-mail, senha ou tipo não enviados."
        })
    } else {
        const usuario = new usuarioModel({
            email: req.body.email,
            senha: bcrypt.hashSync(req.body.senha, 8),
            tipo: req.body.tipo
        });
        usuarioModel.updateById(req.params.id, usuario, (err, data)
=> {
            if (err) {
                if (err.type == "not_found") {
                    res.status(404).send({
                        message: "Usuário não encontrado."
                    })
                } else {
                    res.status(500).send({
                        message: "Erro ao atualizar usuário."
```

```
                    })
                }
            } else {
                res.send(data);
            }
        });
    }
}

exports.delete = (req, res) => {
    usuarioModel.remove(req.params.id, (err, data) => {
        if (err) {
            if (err.type == "not_found"){
                res.status(404).send({message:"Usuário não
encontrado."})
            } else {
                res.status(500).send({message: "Erro ao deletar
usuário."})
            }
        } else {
            res.send({message: "Usuário deletado com sucesso"});
        }
    })
}
```

usuario.routes.js

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

    app.post("/signup", usuarioController.signUp);
    app.post("/signin", usuarioController.signIn);
    app.get("/usuarios", usuarioController.findAll);
    app.put("/usuarios/:idUsuario", usuarioController.update);
    app.delete("/usuarios/:idUsuario", usuarioController.delete);
}
```

MIDDLEWARES

Um middleware no Express é a maneira de fazer alguma coisa antes da requisição ser processada. Coisas como verificar se o usuário está autenticado, logar algum dado para análise ou qualquer coisa que precise ser feita antes de devolver uma resposta para a requisição.

Dessa forma vamos criar uma nova pasta chamada middlewares e dentro dessa pasta um arquivo chamado **auth_jwt_middleware.js.**

auth_jwt_middleware.js

```
const jwt = require("jsonwebtoken");
const config = require("../configs/auth.config.js");
const usuarioModel = require("../models/usuario.model.js");

verifyToken = (req, res, next) => {
```

```
    const { authorization } = req.headers;
    if (!authorization){
        return res.status(403).send({
            message: "Não possui token para autenticação."
        });
    } else {
        const [, token] = authorization.split(' ');
        jwt.verify(token, config.secret, (err, decoded) => {
            if (err){
                res.status(401).send({
                    message: "Acesso não autorizado. Credenciais
inválidas."
                });
            } else {
                req.id = decoded.id;
                next();
            }
        });
    }
}

isAdmin = (req, res, next) => {
    usuarioModel.findById(req.id, (err, data) => {
        if (data.tipo == 1){
            next();
        } else {
            res.status(403).send({
                message: "Você precisa ser administrador para
executar a ação!"
            })
        }
    });
}

isBalcao = (req, res, next) => {
    usuarioModel.findById(req.id, (err, data) => {
        if (data.tipo == 1 || data.tipo == 2){
            next();
        } else {
            res.status(403).send({
                message: "Você precisa ser do balcao de atendimento
para executar a ação!"
            })
        }
    });
}

isCozinha = (req, res, next) => {
    usuarioModel.findById(req.id, (err, data) => {
        if (data.tipo == 1 || data.tipo == 3){
            next();
        } else {
            res.status(403).send({
                message: "Você precisa ser da cozinha para executar a
ação!"
            })
        }
```

```
    });
}

module.exports = {
    verifyToken: verifyToken,
    isAdmin: isAdmin,
    isBalcao: isBalcao,
    isCozinha: isCozinha
}
```

Adicionar as novas rotas do usuario no servidor

server.js

```
… codigo

//linhas das rotas
require("./app/routes/produto.routes.js")(app);
require("./app/routes/pedido.routes.js")(app);
require("./app/routes/produto_pedido.routes.js")(app);
require("./app/routes/usuario.routes.js")(app);

… codigo
```

Bora testar no insominia, precisamos criar um usuário para cada tipo, ou seja, admin, cozinha e balcao.

Para testar as permissões, precisamos colocar as regras nas rotas com as quais definimos quem terá acesso às rotas.

produto.routes.js

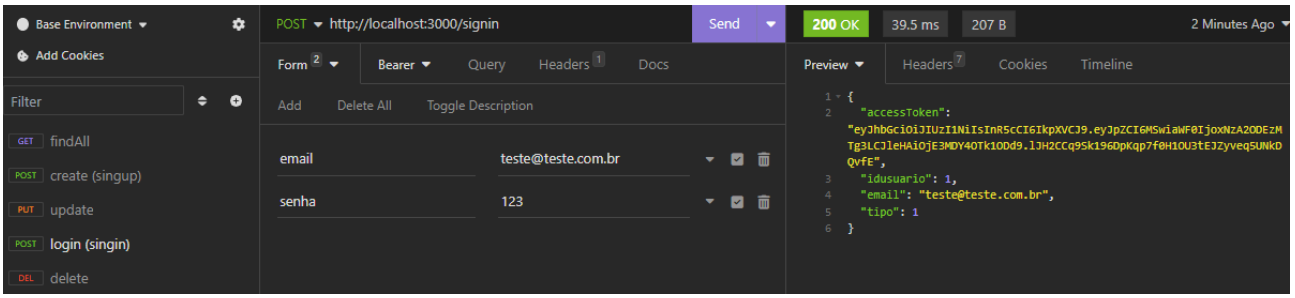
```
module.exports = app => {
    const produtoController =
require("../controllers/produto.controller.js");
    const auth = require("../middlewares/auth_jwt_middleware.js");

    app.post("/produtos", [auth.verifyToken, auth.isAdmin],
produtoController.create);
    app.get("/produtos", [auth.verifyToken],
produtoController.findAll);
    app.get("/produtos/:id", [auth.verifyToken],
produtoController.findById);
    app.put("/produtos/:id", [auth.verifyToken, auth.isAdmin],
produtoController.update);
    app.delete("/produtos/:id", [auth.verifyToken, auth.isAdmin],
produtoController.delete);
    app.delete("/produtos", [auth.verifyToken, auth.isAdmin],
produtoController.deleteAll);
}
```

pedido.routes.js

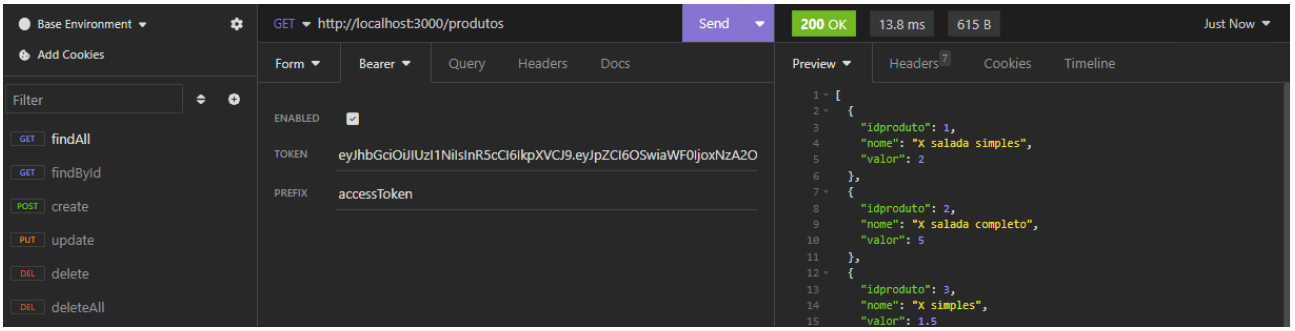
```
module.exports = app => {
    const pedidoController =
```

```
require("../controllers/pedido.controller.js");

    app.post("/pedidos", [auth.verifyToken, auth.isBalcao],
pedidoController.create);
    app.get("/pedidos", [auth.verifyToken],
pedidoController.findAll);
    app.get("/pedidos/:id", [auth.verifyToken],
pedidoController.findById);
    app.put("/pedidos/:id", [auth.verifyToken, auth.isBalcao],
pedidoController.update);
    app.delete("/pedidos/:id", [auth.verifyToken, auth.isBalcao],
pedidoController.delete);
    app.delete("/pedidos", [auth.verifyToken, auth.isBalcao],
pedidoController.deleteAll);
}
```

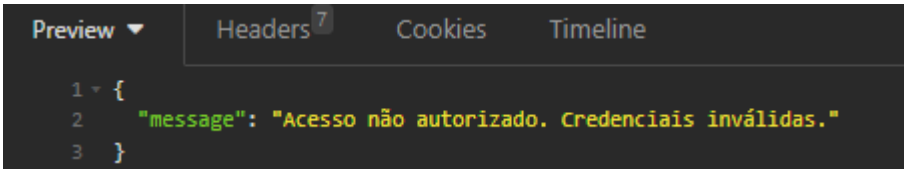
Após dada as permissões precisamos logar um usuário válido e pegar o token:



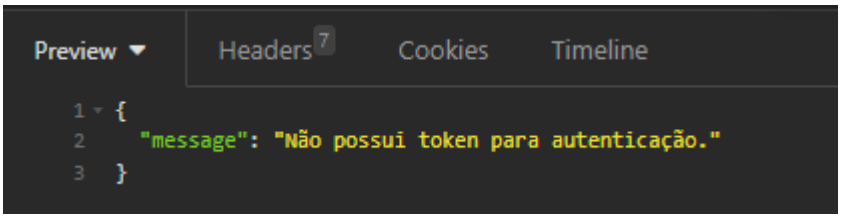
Copiar toda a “string” do token e configurar o Bare Token no insomnia



Caso esteja errado ou expirado o token ocorrerá a seguinte mensagem:



Caso não tenha token (usuário não se logou) ocorrerá a mensagem:


Preview
Headers 7
Cookies
Timeline
{
"message": "Não possui token para autenticação."
}