

## NÍVEL BÁSICO



PROJETO 04

(CONTEÚDO DISPONÍVEL) {  
**SISTEMA DE;**  
**AUTENTICAÇÃO;**  
**DE USERS;**  
(end);  
})();

#PORTFÓLIOBOOSTPROGRAM

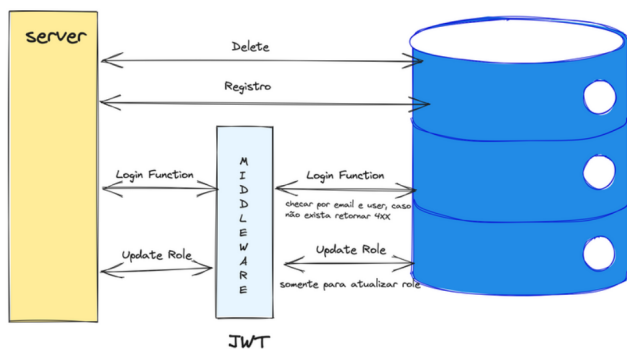
**CONHECIMENTOS REQUIRIDOS:**



**BACK-END**

# WIREFRAME

## System Design



### Componentes

**Servidor web:** O servidor web é responsável por lidar com solicitações e respostas HTTP. Também é responsável por autenticar usuários e proteger as rotas com o middleware jwt.

**Banco de dados Mongo:** O banco de dados Mongo armazena os dados do usuário.

**Front-end:** O front-end é responsável por exibir a interface do usuário e interagir com o servidor web.

### Arquitetura

**Camada de apresentação:** A camada de apresentação é responsável por exibir a interface do usuário. É implementado usando HTML, CSS e JavaScript.

**Camada de aplicação:** A camada de aplicação é responsável por lidar com solicitações e respostas HTTP. É implementado usando Node.js e Express.

**Camada de dados:** A camada de dados é responsável por armazenar os dados do usuário. É implementado usando Mongo.

### Segurança

O middleware jwt é usado para autenticar usuários e proteger as rotas. Os dados do usuário são armazenados no banco de dados Mongo, que é um banco de dados seguro.

O front-end é protegido usando HTTPS.

# SISTEMA DE AUTENTICAÇÃO DE USERS

Muitas das vezes nos pegamos utilizando diversos serviços de **autenticação**, aqui mesmo menciono sobre o **clerk** e o **nextAuth**. Só que dessa vez vamos criar nosso próprio sistema de autenticação

## TECH STACK

- ➔ NodeJS
- ➔ Express
- ➔ Vercel (conhecimento básico de serverless)



## LIBRARIES

- ➔ JWT
- ➔ Bcrypt
- ➔ bodyParser



## BRIEFING

A idéia é criar um sistema de autenticação simples utilizando **nodeJS**. Esse projeto terá somente algumas linhas de **frontend** no nível extra, projeto focado em na construção de **portfolio back-end**.

## NÍVEL ÚNICO

Vamos implementar as rotas necessárias para **autenticação**, **recuperar senha**, **reset de senha**, **registro** e **ativação**.

## NÍVEL EXTRA

Fazer uma **interface web simples** que não seja necessário interagir somente via **back-end/postman**.

## REQUISITOS DETALHADOS

### Configurar um banco de dados Mongo

- ➔ Instale os pacotes **express** e **nodemon** via **npm** após iniciar um repositório.
- ➔ Inicie o servidor com o comando `nodemon server.js`.
- ➔ O **server.js** irá conter todos os arquivos necessários para rodarmos a aplicação.
- ➔ Conecte-se ao banco de dados com o comando **mongo**.
- ➔ Crie o **schema do usuário** com o comando

```
db.createCollection("users").
```
- ➔ Execute as operações **CRUD** (**criar**, **ler**, **atualizar** e **excluir**) no banco de dados.





### Função de registro

- Crie uma função de **registro** com o comando  

```
app.post("/register", register).
```
- A função de registro deve aceitar um **objeto JSON** com os campos **username** e **password**.
- A função de registro deve **inserir o usuário** no banco de dados.

### Função de login

- Crie uma função de **login** com o comando  

```
app.post("/login", login).
```
- A função de login deve aceitar um **objeto JSON** com os campos **username** e **password**.
- A função de login deve retornar um **token JWT** se o usuário for autenticado.

### Função de atualização



- Crie uma função de **atualização** com o comando  

```
app.put("/users/id", update).
```
- A função de atualização deve aceitar um **objeto JSON** com os campos **username** e **password**.
- A função de atualização deve **atualizar o usuário** no banco de dados.

### Função de exclusão

- ➡ Crie uma função de **exclusão** com o comando  

```
app.delete("/users/:id", delete).
```
- ➡ A função de exclusão deve aceitar o **ID do usuário como parâmetro**.
- ➡ A função de exclusão deve **excluir o usuário** do banco de dados.

### Hashear as senhas dos usuários

- ➡ Use o pacote **bcrypt** para **hashear** as senhas dos usuários.
- ➡ A função de **registro** e a função de **login** devem usar as **senhas hash**.

### Refatorar a função de login

- ➡ **Remova o código** para **hashear** as senhas da função de login.
- ➡ Use o **middleware jwt** para **autenticar os usuários**.
- ➡ Autenticar usuários com **JSON Web Token (JWT)**
- ➡ Instale o pacote **jsonwebtoken**.
- ➡ Crie um **middleware jwt** para **autenticar os usuários**.
- ➡ A função de **login** deve usar o **middleware jwt**.

### Autenticar usuários com JSON Web Token (JWT)

- ➡ Instale o pacote `jsonwebtoken`.
- ➡ Crie um `middleware jwt` para `autenticar` os usuários.
- ➡ A função de `login` deve usar o `middleware jwt`.

### Refatorar a função de registro

- ➡ `Remova o código` para `hashear` as senhas da função de registro.
- ➡ Use o `middleware jwt` para `autenticar` os usuários.

### Autenticação de administrador

- ➡ Crie uma rota de `administrador` com o comando  

```
app.get("/admin", isAdmin, (req, res) => {...}).
```
- ➡ Proteja a rota de administrador com o `middleware jwt`.
- ➡ Popule o banco de dados com um `usuário administrador`.

### Autenticação de usuário básico

- ➡ Crie uma rota de `usuário` com o comando  

```
app.get("/users", isAuthenticated, (req, res) => {...}).
```
- ➡ Proteja a rota de usuário com o `middleware jwt`.



### Proteger as rotas



Proteja todas as rotas com o **middleware jwt**.

### Popular o banco de dados com um usuário administrador



Use o comando

```
db.insertOne({username: "admin", password: "password"})
```

para **popular** o banco de dados com um **usuário administrador**.

### Criar o formulário de login usando EJS



Crie um arquivo **login.ejs** e adicione o **código HTML** para o **formulário de login**.

### Adicionar usuários registrados à rota



**Atualize** a rota de usuário para exibir todos os usuários registrados.

### Funcionalidade de logout



Crie uma rota de **logout** com o comando

```
app.get("/logout", logout).
```



Implemente a **funcionalidade de logout** na rota de **logout**.

