boost by erre.

DOCUMENTAÇÃO DE BACKEND PARA REGISTO E AUTENTICAÇÃO DE UTILIZADORES

Filipe Silva

BACKEND

Tecnologias:

- Node.js
- Express.js
- MongoDB
- JSON Web Token
- Multer
- bcrypt

Instruções:

- A máquina deverá ter instalado o **Node**.

- Clonar o repositório (ou efetuar o download do mesmo):

git clone https://github.com/fgalego/boost-challenge.git

- Aceder ao diretório do projeto:

cd backend

- Preencher as variáveis no ficheiro .env:

DB_USER= (Username da base de dados)
DB_PASSWORD= (Password da base de dados)
DB_LINK= (Link para a base de dados)
JWT_SECRET= (Chave secreta)

- Completar/verificar o link mongoose em “conn.js” na pasta “db”:

`mongodb+srv://${dbUser}:${dbPassword}@${dbLink}`

- Instalar o NPM:

npm install

- Iniciar a aplicação:

npm start

API DE REGISTO E AUTENTICAÇÃO DE UTILIZADORES

A API permite que os utilizadores se registem, façam login, atualizem os seus perfis e carreguem avatares. Para garantir a segurança e a privacidade dos dados, a autenticação é realizada usando JSON Web Tokens (JWT).

Base URL

A URL base para todos os endpoints da API é:

http://localhost:5000/api

Autenticação

A autenticação é necessária para acessar alguns endpoints da API. Deverá ser incluído o token JWT no cabeçalho de autorização da solicitação. O token JWT é gerado durante o processo de login.

Exemplo de cabeçalho de autorização:

Authorization: Bearer SEU_TOKEN_JWT_AQUI

Endpoints Disponíveis

1. Registo de Utilizador

Regista um novo utilizador na aplicação.

Método HTTP: POST

URL: /register

Corpo da Solicitação (JSON):

```
{
  "name": "seunome",
  "email": "seuemail@example.com",
  "password": "suapalavrapasse",
  “confirmpassword”: “suapalavrapasse”
}
```

Resposta de Sucesso (200 OK):

```
{
  "message": "authenticated with success!"
}
```

2. Login de Utilizador

Permite que um utilizador existente faça login e receba um token JWT.

Método HTTP: POST

URL: /login

Corpo da Solicitação (JSON):

```
{
  "email": "seuemail@example.com",
  "password": "suapalavrapasse"
}
```

Resposta de Sucesso (200 OK):

```
{
  "message": "authenticated with success!",
  "token": "seu_token_jwt_aqui",
  "userid": "id_do_usuário"
}
```

3. Logout de Utilizador

Permite que um utilizador faça logout, invalidando o token JWT.

Método HTTP: POST

URL: /logout

Resposta de Sucesso (200 OK):

```
{
  "message": "logged out!”
}
```

4. Perfil de Utilizador

Obtém informações do perfil do utilizador autenticado.

Método HTTP: GET

URL: /profile

Resposta de Sucesso (200 OK):

```
{
“_id”: “id_do_usuário”,
“name”: "seunome”,
“email”: “seuemail@example.com”,
“createdat”: “data_e_hora_da_criação”,
“updatedat”: “data_e_hora_da_última_atualização”

(caso o usuário já possua avatar no perfil):
 “image”: “nome_da_imagem.extensão”
}
```

5. Alterar Dados do Utilizador

Permite que o utilizador autenticado atualize os seus dados, incluindo nome de utilizador, email e palavra-passe.

Método HTTP: PUT

URL: /profile

Corpo da Solicitação (JSON):

```
{
  “name”: “novonome”,
  “email”: “novoemail@example.com”",
  “password”: “novapalavrapasse”,
  “confirmpassword”: “novapalavrapasse”
}
```

Resposta de Sucesso (200 OK):

```
{
  "message": "User updated!"
}
```

6. Upload de Avatar

Permite que o utilizador autenticado faça upload de um avatar (imagem).

Método HTTP: POST

URL: /edit/id_do_usuário

Corpo da Solicitação (Formulário com multipart/form-data):

Campo de arquivo: image

Resposta de Sucesso (200 OK):

```
{
  "message": "User avatar updated!
}
```

Erros da API

A API pode retornar erros com mensagens explicativas em caso de solicitações inválidas.

Exemplo de possíveis respostas de erros (400 Bad Request e 422 Unprocessable Entity) :

```
{
  “message”: “A valid name is required.”,
};
```

```
{
  “message”: “Passwords do not match”,
};
```

```
{
  “message”: “Invalid Token!”
}
```