Atividade 3

Node.js, Express.js, Mongoose, BCrypt e JWT

Universidade Federal do Ceará

Projeto de Interfaces Web

Prof. Victor Farias

* **Data de entrega**: 05/07/21
* **Modo de entrega**
  + Código + vídeo de apresentação zipado pelo moodle
  + O vídeo também pode ser subido no YouTube e envia o link no arquivo zip
* **Instruções para o código**
  + Excluir node\_modules e adicionar os outros arquivos ao zip final
* **Instruções para o vídeo de apresentação:**
* Somente as funcionalidades que foram mostradas no vídeo receberam nota
* Não precisa falar, basta gravar a tela
  1. Lembrar de usar uma qualidade um pouco mais baixa, caso envie o vídeo pelo moodle, pois o moodle só aceita arquivos de até 20mb
  2. Pode usar algum software como o obs ou gravar a tela do computador ou celular
* O objetivo é demonstrar as funcionalidades pedidas usando o postman ou software similar.

**Roteiro do vídeo**

**Montando o banco**

* 1. Inserir um primeiro usuário pelo endpoint **POST** /api/usuários
  2. Gerar o token a partir do endpoint **POST** /api/usuarios/signin para usuário inserido no ponto 1
  3. Inserir um post para esse com **POST** /api/posts para o usuário passando o token de autenticação dele
  4. Inserir um comentário para o post do ponto 3 com **POST /**api/comentarios passando o token de autenticação do usuário

**Fazendo buscas**

* 1. Buscar todos os usuários com **GET /**api/usuarios passando o token
  2. Buscar todos os posts com **GET /**api/posts passando o token
  3. Buscar todos os comentários com **GET** /api/comentarios
  4. Mostrar as 3 coleções (usuário, posts e comentários) no MongoDB usando o MongoShell ou alguma aplicação parecida

**Removendo dados**

* 1. Insira um segundo usuário e obtenha o token dele
  2. Tente remover o comentário feito no ponto 4 usando **DELETE** /api/comentarios/<id\_comentario> com o token do segundo usuário (deve dar erro)
  3. Tente remover o comentário feito no ponto 4 usando **DELETE** /api/comentarios/<id\_comentario> com o token do primeiro usuário (deve ser ok)
  4. Tente remover post feito no ponto 3 usando **DELETE**  /api/usuarios/<id\_post> com o token do segundo usuário (deve dar erro)
  5. Tente remover post feito no ponto 3 usando **DELETE**  /api/usuarios/<id\_post> com o token do primeiro usuário (deve ser ok)
  6. Tente remover o primeiro usuário usando **DELETE**  /api/usuarios/<id\_usuario> usando o token do segundo usuário (deve dar erro)
  7. Tente remover o primeiro usuário usando **DELETE**  /api/usuarios/<id\_usuario> usando o token do segundo usuário (deve ser ok)
  8. Mostrar as coleções com os dados removidos
* Dica: Deixa todas as requisições prontas no postman e, depois, é só gravar executando-as.
* Tempo máximo do vídeo: 5 minutos

**Questões**

1. Todos os endpoints agora recebem e validam o token JWT menos em POST /usuarios/signin e POST /usuarios. Caso o token seja válido, executa o endpoint normalmente. Caso seja invalido, retornar mensagem de erro invalido para o usuário.
2. **Endpoints.** Construa os endpoints que se recebe, respeitando **exatamente** as estruturas do JSON de exemplo e **exatamente** nessas rotas. Agora os dados serão armazenados no MongoDB.  
    **Usuários**
   1. **POST** /api/usuarios (2 pontos)
      1. Recebe usuário e armazena em banco
      2. Deve ser armazenada somente a assinatura hash da senha em banco
      3. Exemplo do json de requisição:

{

“nome”: “Victor”,

“email”: “victor.aefarias@gmail.com”,

“senha”: “123”

}

* + 1. Retorna o mesmo usuário sem senha
  1. **POST** /api/usuarios/signin (3 pontos)
     1. Recebe email e senha do usuário
     2. Verifica se a senha recebida corresponde a assinatura hash do usuário no banco de dados
     3. Exemplo do json de requisição:

{

“email”: “victor.aefarias@gmail.com”,

“senha”: “123”

}

* + 1. Retorna o token com o id do usuário como payload
  1. **DELETE**  /api/usuarios/:id (1 ponto)
     1. Remove usuário com id dado
     2. Só permite a remoção se o usuário a ser deletado for o usuário com o id fornecido no token
  2. Manter o resto dos enpoints adicionando o recebimento e validação do token

**Posts**

* 1. **POST** /api/posts (1 ponto)

1. Recebe post, armazena em banco com id do usuário como o id do usuário do token
2. Exemplo json da requisição:

{

“texto”: “Oi, tudo bem?”,

“likes”: “6”,

}

1. Retorna o mesmo post
   1. **DELETE**  /api/posts/:id (1 ponto)
      1. Remove post com id dado
      2. Só permite a remoção se o post pertencer ao usuário logado
   2. Manter o resto dos enpoints adicionando o recebimento e validação do token

**Comentarios**

* 1. **POST /**api/comentarios (1 ponto)
     1. Recebe comentário, armazena em banco com id do usuário como o id do usuário do token
     2. Exemplo json de requisição:

{

“texto”:”Tudo certo e contigo?”,

“id\_post”: 1,

}

* 1. **DELETE** /api/comentarios/:id (1 ponto)
     1. Exclui comentário com id dado.
     2. Só permite excluir comentário de o comentário pertence ao usuário logado
  2. Manter o resto dos enpoints adicionando o recebimento e validação do token