Autenticação em Aplicações React

Tópicos

- **Introdução: Autenticação e Autorização**
- Armazenando e utilizando dados de autenticação
- Controlando acesso

Motivação

- Em sistemas utilizados por múltiplos usuários, é bastante comum que certos dados sejam restritos a usuários específicos
- Algumas informações são privadas, e somente o usuário que as criou deve poder acessá-las
- Outras devem poder ser acessadas somente por um grupo de usuários
  - Posts disponíveis somente para seguidores
  - Funcionalidades disponíveis somente para assinantes

Autenticação e Autorização

Autenticação 👱

- **Autenticação** representa o conceito de verificar a **identidade** de um usuário
- “O usuário é realmente quem ele diz ser?”
- Depende de uma ou mais informações que somente o usuário tenha acesso
  - Senha
  - Email/celular
  - Impressão digital
  - etc...

Autorização 🔐

- Autorização representa o conceito de verificar o **direito** de um usuário a **acessar um dado** ou **realizar determinada ação**
- Possibilidade de separar funcionalidades específicas para usuários diferentes

Junção 👱 + 🔐

- Os dois conceitos são frequentemente confundidos, apesar de serem diferentes

- Ao **autenticar** um usuário (saber quem é), já sabemos quais **autorizações** (saber o que pode fazer) ele possui

Como identificar usuários?

- É preciso armazenar as informações de autenticação
  - Normalmente responsabilidade do backend
- Front coleta e envia essas informações
- Backend confirma se informações estão corretas, **autenticando** o usuário
- Assim, é possível saber que o usuário é quem ele diz ser

Como controlar a autorização?

- Dado que sabemos a identidade de um usuário, como liberar somente os recursos aos quais ele tem acesso?
  - Validar no front não é suficiente
- Todas as requisições precisam conter uma identificação
  - Armazenar login e senha poderia ser perigoso em caso de vazamento de dados
- Backend gera uma string “aleatória” que identifica o usuário, que deve ser enviada nas requisições seguintes

O que o front precisa fazer?

- Garantir que a identificação do usuário seja passada ao servidor **em cada requisição**

- **Bloquear** acesso a páginas e funcionalidades específicas, que necessitam de autorização

- Guardar informações do usuário para que **o login não precise ser refeito**

Fluxo Comum 

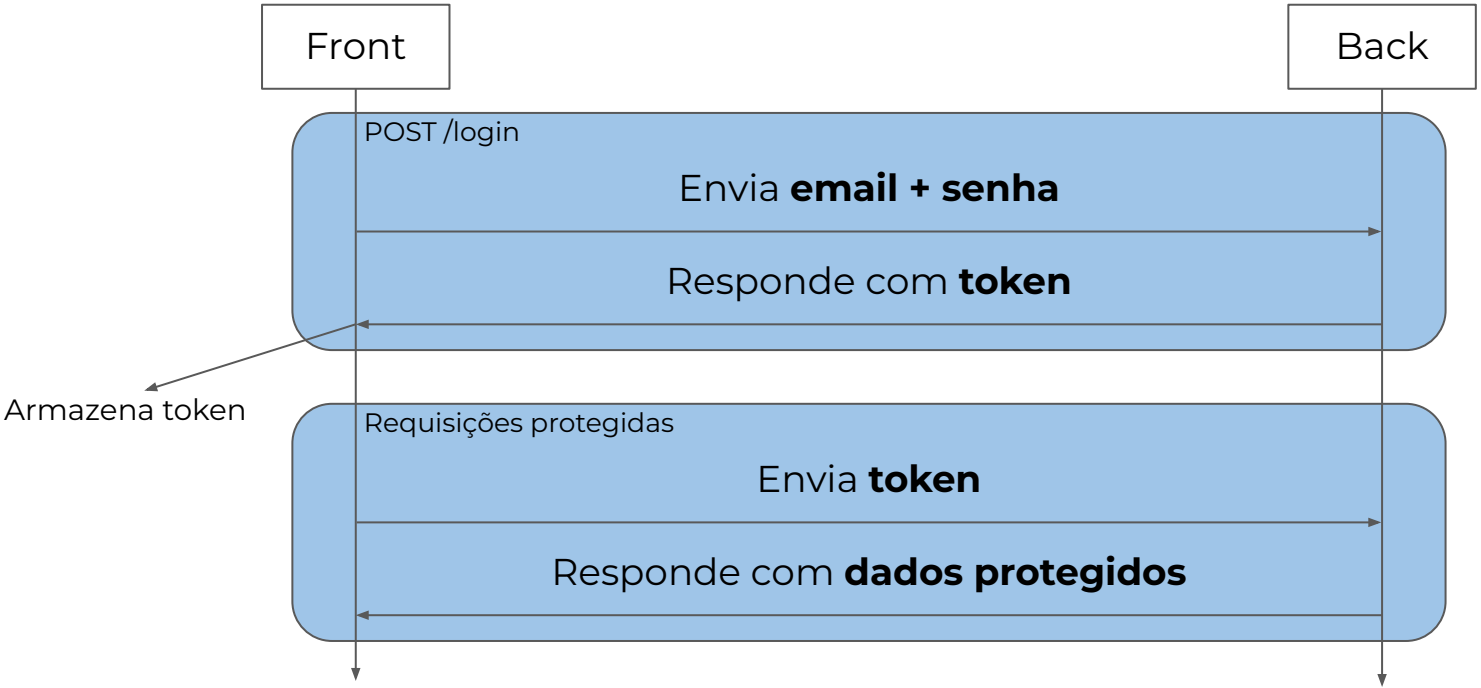
1. Usuário insere login e senha

2. Front envia informações para o servidor

3. Servidor confirma informações e responde um **token** em caso de sucesso

4. Front guarda informações de autenticação e as envia junto com todas as requisições

Fluxo Comum


Front
Back
POST /login
Envia email + senha
Responde com token
Armazena token
Requisições protegidas
Envia token
Responde com dados protegidos

- Onde e como guardar o token?
- Como incluir o token nas requisições?
- Como controlar o acesso a páginas específicas?

Armazenando dados e utilizando dados de autenticação

Contexto 🔐

- Após ter autenticado o usuário pela primeira vez, é importante que ele continue autenticado

- Todas as requisições que necessitarem de autenticação devem ser "**assinadas"**

- Para realizar essa assinatura, é preciso guardar suas informações

Armazenar o quê? 🔐

- Backend responderá requisição de autenticação (normalmente, o login) com um **token**

- Um token é um identificador qualquer, normalmente uma sequência de caracteres

- Existem vários tipos de token. O mais comum hoje em dia é o **JWT**

JWT - O que é? 🔐

- JWT significa **JSON Web Token**
- É um padrão de tokens muito usado para autenticação
- Permite guardar dados arbitrários de usuário e uma **data de expiração**
- É **encriptado**, garantindo que um JWT gerado com a chave errada não será válido

Onde armazenar 🗃️

- Agora que temos nosso JWT, precisamos **armazená-lo** para que ele seja incluído nas próximas requisições

- Duas opções para armazenamento:
  - **Cookies**
  - **Local Storage**

Cookies 🍪

- Quando o backend responde a requisição de autenticação com o **token**, ele responde com um **header que define o cookie**

- A partir daí, o navegador **já inclui o cookie automaticamente** nas requisições feitas pelo site para o mesmo domínio

- Ou seja, quando utilizamos os cookies, as **configurações partem do backend**

Local Storage 🗄️

- Quando o backend responde a requisição de autenticação com o **token**, é o **front que fica responsável por guardar ele no Local Storage**

- Depois de guardado, em todas as requisições, buscamos o valor no Local Storage e **mandamos no header da requisição**

Qual usar 🍪🗄️

- Na maioria das vezes, vamos depender de como o backend responde/espera o token

- Decisão de projeto que deve envolver devs de front e back

- Por enquanto, vamos usar a estratégia do LocalStorage, pois para Cookies funcionarem são necessários mais passos de configuração no back

Exemplo - token no Local Storage

```
1  import axios from "axios"
2
3  const LoginPage = () => {
4    // variáveis do estado que guardam o input controlado
5    const [email, setEmail] = useState("")
6    const [password, setPassword] = useState("")
7
8    const login = () => {
9        const body = {
10         email: email,
11         password: password
12       };
13
14       // faz requisição de login pro back, passando email e senha
15       // cadastrados com o endpoint signup
16       axios.post("https://minha-api.com/login", body).then(response => {
17
18           // salva o token enviado pelo back no localstorage
19           window.localStorage.setItem("token", response.data.token);
20       })
21   };
22
23   ...
```

Exemplo de requisição de login que retorna um token para autenticação do usuário

Exemplo - token no Local Storage

```
1  import axios from "axios"
2
3  const ProtectedPage = () => {
4
5    const getProtectedData = () => {
6      // Pega o token salvo no local storage na requisção de login
7      const token = window.localStorage.getItem("token")
8
9      // faz a requisição protegida, passando o token no header Authorizathion
10     axios.get("https://minha-api.com/login", {
11         headers: {
12             Authorizathion: token
13         }
14     }).then(response => ...
```

Exemplo de requisição que pega dados protegidos de um banco de dados

Controlando Acesso

Autenticação Simples 👱

- Páginas que requerem autenticação simples exigem somente que o usuário esteja logado no sistema

- Para simplificar: **estar logado**, para o front, significa **possuir o token guardado no LocalStorage**

- Caso o token não esteja no LocalStorage, **o usuário não tem permissão para ver aquela página** e deve ser redirecionado para a página de login

Redirecionando 🔐

- **Após a página carregar**, verificamos se o token existe
- Para isso, usamos um `useEffect()`
- Caso o token não esteja no Local Storage, **redirecionamos** o usuário para a página de login
- Lembrando que devemos usar a função `push()` vinda do `history` para redirecionar

Redirecionando 🔐

- A existência do token no LocalStorage **não garante** que ele seja válido.
  - Tokens, especialmente JWT, possuem uma data de expiração (exp) que deve ser verificada antes de permitir o acesso a páginas protegidas.
- Ao carregar a página, decodificamos o token para acessar o campo exp e verificamos se ainda está dentro do prazo.
  - Se o token estiver expirado, removemos o token do armazenamento e redirecionamos o usuário para a página de login.

Exemplo - Redirecionamento

```jsx
import React, {useEffect} from 'react'
import {useHistory} from 'react-router-dom'

export const ProtectedPage = () => {
  // Acessa o history
  const history = useHistory()

  // useEffect para verificar o token
  useEffect(() => {
    // Pega o token salvo no localStorage
    const token = window.localStorage.getItem('token')

    if(token === null) {
      // Se token não existe, redireciona
      // para página de login
      history.push('/login')
    }
  }, [history])

  return <div>
    Essa página só deve ser acessada
    por usuários logados
  </div>
}
```

Exemplo - Redirecionamento

```jsx
import React, {useEffect} from 'react'
import {useHistory} from 'react-router-dom'

export const ProtectedPage = () => {
  // Acessa o history
  const history = useHistory()

  // useEffect para verificar o token
  useEffect(() => {
    // Pega o token salvo no localStorage
    const token = window.localStorage.getItem('token')

    if(token === null) {
      // Se token não existe, redireciona
      // para página de login
      history.push('/login')
    }
  }, [history])

  return <div>
    Essa página só deve ser acessada
    por usuários logados
  </div>
}
```

Essa lógica deve se repetir em qualquer página que for protegida. Portanto, é possível extraí-la para um custom hook

Exemplo - Custom Hook - Sem exp

```jsx
import {useEffect} from 'react'
import {useHistory} from 'react-router-dom'

export const useProtectedPage = () => {
  // Acessa o history
  const history = useHistory()

  // useEffect para verificar o token
  useEffect(() => {
    // Pega o token salvo no localStorage
    const token = window.localStorage.getItem('token')

    if(token === null) {
      // Se token não existe, redireciona
      // para página de login
      history.push('/login')
    }
  }, [history])
}
```

```jsx
import React, {useEffect} from 'react'
import {useHistory} from 'react-router-dom'
import {useProtectedPage} from '../hooks/useProtectedPage'

export const ProtectedPage = () => {
  useProtectedPage()

  return <div>
    Essa página só deve ser acessada
    por usuários logados
  </div>
}
```

Exemplo - Custom Hook - Com exp

```
export const useProtectedPage = () => {
  const history = useHistory();

  useEffect(() => {
    const token = localStorage.getItem("token");

    if (!token) {
      history.push("/login");
    }

    const decodedToken = JSON.parse(atob(token.split(".")[1]));
    const currentTime = Date.now() / 1000; // in seconds

    if (currentTime > decodedToken.exp) {
        localStorage.removeItem("token");
        history.push("/login");
    }

  }, [history]);
};
```

```
1  import React, {useEffect} from 'react'
2  import {useHistory} from 'react-router-dom'
3  import {useProtectedPage} from '../hooks/useProtectedPage'
4
5  export const ProtectedPage = () => {
6    useProtectedPage()
7
8    return <div>
9      Essa página só deve ser acessada
10     por usuários logados
11   </div>
12 }
```