

ROTEIRO DE AULA – Autenticação JWT em React JS

Tema: Aula 1.1 – Autenticação JWT em React JS

Data: 21/11/2019

Semana:

Tempo estimado: 4 horas.

Requisitos e ferramentas para aula:

VsCode, arquivos gufos_base (HTML + CSS), projeto gufos_ui (React JS).

Descrever roteiro de aula:

1. Adicionar uma Claim personalizada no Login Controller da API, para que a propriedade Tipo Usuário seja acessada no Front-End.

```
// Define as Claims (dados da sessão)
var claims = new[] {
    new Claim(JwtRegisteredClaimNames.NameId, userInfo.Nome),
    new Claim(JwtRegisteredClaimNames.Email, userInfo.Email),
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
    new Claim(ClaimTypes.Role, userInfo.TipoUsuario.Titulo.ToString()),
    new Claim("permissao", userInfo.TipoUsuario.Titulo.ToString())
};
```

2. Componentizar o cabeçalho da aplicação.

```
<header class="cabecalhoPrincipal">
  <div class="container">
    

    <nav class="cabecalhoPrincipal-nav">
      <a>Home</a>
      <a>Eventos</a>
      <a>Contato</a>
      <a class="cabecalhoPrincipal-nav-login" href="login.html">Login</a>
    </nav>
  </div>
</header>
```

3. Importar os arquivos CSS da aplicação.

```
<!-- Estilos -->
<link rel="stylesheet" href="./assets/css/flexbox.css" />
<link rel="stylesheet" href="./assets/css/reset.css" />
<link rel="stylesheet" href="./assets/css/style.css" />
<link rel="stylesheet" href="./assets/css/cabecalho.css" />
<link rel="stylesheet" href="./assets/css/rodape.css" />
```

4. Implementar a tela de Login. ****Se não já tiver sido implementada. Colocar os inputs e form.**

5. Pegar os dados do form.

```
this.state = {  
  email: "",  
  senha: "",  
  erroMensagem: ""  
};
```

6. Introduzir os alunos ao service de API, dizendo: Agora precisamos fazer a requisição com a API, para isso iremos criar um serviço que facilite a comunicação com ela, passando todas configurações necessárias de uma só vez.
7. ****Opcional: Instalar as dependências do 'axios' para utilizá-lo no projeto.**
 - a. 'yarn add --dev axios' para instalar via yarn.
 - b. 'npm install --save axios'.
8. ****Opcional: Criar um service de API para fazer as requisições.**

```
import axios from 'axios';  
  
const api = axios.create({  
  baseURL: "http://localhost:5000/api",  
  headers: {  
    "Content-Type" : "application/json",  
    "Authorization": "Bearer " + localStorage.getItem("usuario-gufos")  
  }  
});  
  
export default api;
```

9. Fazer a requisição utilizando o service API, caso tenha sido implementado. Somente colocar o token no LocalStorage e enviar para a página de cadastro de Eventos. Não fazer validações ainda.

```
api.post("/login", {
  email: this.state.email,
  senha: this.state.senha
})
.then(data => {
  console.log("status", data.status);
  if (data.status === 200) {
    console.log(data);
    localStorage.setItem("usuario-gufos", data.data.token);

    //Verifica o tipo de usuário e redireciona para a página default
    console.log("parse", parseJwt().permissao);

    if (parseJwt().permissao === "Administrador") {
      this.props.history.push("/eventos/cadastrar");
    } else {
      this.props.history.push("/eventos");
    }
  }
})
.catch(erro => {
  this.setState({ erroMensagem: "Email ou senha inválido" });
  alert(this.state.erroMensagem);
});
```

10. Testar o código, mostrando que o token está sendo colocado no LocalStorage do navegador.
11. Validar se o usuário está logado no cabeçalho. Perguntar para os alunos, como podemos saber qual usuário está logado com o token? “Utilizando a Claim ‘permissao’” e para isso vamos criar um service que decodifique o token.
12. Para fazermos a validação será necessário instalar o JWT Decode. Por isso todos devem executar: “npm install –save jwt-decode”

```
export const usuarioAutenticado = () => localStorage.getItem("usuario-gufos") !== null;

export const parseJwt = () => {
  let jwtDecode = require("jwt-decode"); // Importando o jwt-decode

  var token = localStorage.getItem("usuario-gufos");

  return jwtDecode(token);
};
```

13. Colocar um console.log no Login.js quando este trouxer o token, para verificar se o token foi decodificado.

14. Fazer as validações no Cabecalho.js.

```
render() {  
  const NavAdmin = (  
    <div>  
      <Link to="/tiposeventos">Tipos Eventos</Link>  
      <Link to="/eventos/cadastrar">Eventos</Link>  
      <span onClick={this.logout.bind(this)}>Sair</span>  
    </div>  
  );  
  
  const NavComum = (  
    <div>  
      <Link to="/eventos">Eventos</Link>  
      <span onClick={this.logout.bind(this)}>Sair</span>  
    </div>  
  );  
  
  const NavDeslogado = (  
    <div>  
      <Link to="/login">Login</Link>  
    </div>  
  );  
  
  // Header v1.0  
  return (  
    <header className="cabecalhoPrincipal">  
      <div className="container">  
        <img src={logo} alt="Gufos" />  
  
        <nav className="cabecalhoPrincipal-nav">  
          <div>  
            <Link to="/">Home</Link>  
          </div>  
  
          {(usuarioAutenticado() && parseJwt().permissao) === "Administrador" ? (  
            NavAdmin  
          ) : (usuarioAutenticado() && parseJwt().permissao) === "Aluno" ? (  
            NavComum  
          ) : (  
            NavDeslogado  
          )}  
        </nav>  
      </div>  
    </header>  
  );  
}
```

15. Fazer método para deslogar.

```
logout()  
{  
  localStorage.removeItem("usuario-gufos");  
  this.props.history.push("/");  
  
  // alert("Usuário deslogado com sucesso!");  
}
```

16. Demonstrar que o login funciona, mas ainda assim, um usuário comum pode acessar páginas que somente um usuário administrador poderia, por isso devemos adicionar a validação no Routes.js.

```
//Verifica se o usuário esta logado e se o role é do tipo Admin
const PermissaoAdmin = ({ component: Component }) => (
  <Route
    render={props =>
      usuarioAutenticado() && parseJwt().permissao == "Administrador" ? (
        <Component {...props} />
      ) : (
        <Redirect to={{ pathname: "/login" }} />
      )
    }
  />
);

//Verifica se o usuário esta logado e se o role é do tipo Comum
const PermissaoComum = ({ component: Component }) => (
  <Route
    render={props =>
      usuarioAutenticado() && parseJwt().permissao == "Aluno" ? (
        <Component {...props} />
      ) : (
        <Redirect to={{ pathname: "/login" }} />
      )
    }
  />
);

const routing = (
  <Router>
    <div>
      <Switch>
        <Route exact path="/" component={App} />
        <Route path="/login" component={Login} />

        <PermissaoAdmin path="/tiposeventos" component={TiposEventos} />

        <PermissaoComum exact path="/eventos" component={EventoIndex} />
        <PermissaoAdmin path="/eventos/cadastrar" component={EventoCadastro} />

        <Route component={NaoEncontrada} />
      </Switch>
    </div>
  </Router>
);
```

Descrever atividades:

Os alunos deverão acompanhar a exemplificação tirando dúvidas quando for necessário.

Tempo atividades: 4hrs

Descrever dinâmica:

Tempo de dinâmica:

Descrever atividade extra: