Session-storage.service.ts

import { Injectable } from '@angular/core';

import { SessionStorageModel } from '../\_models/SessionStorageModel';

import { TokenModel } from '../\_models/TokenModel';

import {

EIMS\_SESSION\_ID\_TOKEN, EIMS\_SESSION\_REFERESH\_TOKEN, EIMS\_SESSION\_ExpirationTime, SERVICE\_SESSION\_ID\_TOKEN,

SERVICE\_SESSION\_REFERESH\_TOKEN, SERVICE\_SESSION\_ExpirationTime, TENANTNAME\_ID, AUTH\_TYPE\_ID, APP\_TITLE\_ID,

DEFAULT\_TITLE, APP\_FAV\_ICON, DEFAULT\_APP\_FAV\_ICON, TENANT\_ID

} from '../\_constants/authConst';

@Injectable()

export class SessionStorageService {

sessionStorgaeModel: SessionStorageModel = new SessionStorageModel();

constructor() { }

code: string;

public set(key: string, value: string) {

this.sessionStorgaeModel[key] = value;

//sessionStorage.setItem(key,value);

}

get(key: string): string {

//return sessionStorage.getItem(key);

return this.sessionStorgaeModel[key]

}

remove(key: string) {

// sessionStorage.removeItem(key);

this.sessionStorgaeModel[key] = null;

}

clearSession() {

this.sessionStorgaeModel = new SessionStorageModel();

}

public setEIMSToken(token: TokenModel) {

sessionStorage.setItem(EIMS\_SESSION\_ID\_TOKEN, token.access\_token);

sessionStorage.setItem(EIMS\_SESSION\_REFERESH\_TOKEN, token.refresh\_token);

sessionStorage.setItem(EIMS\_SESSION\_ExpirationTime, token.expires\_in.toString());

}

getEIMSToken() {

const token: TokenModel = new TokenModel();

token.access\_token = sessionStorage.getItem(EIMS\_SESSION\_ID\_TOKEN);

token.refresh\_token = sessionStorage.getItem(EIMS\_SESSION\_REFERESH\_TOKEN);

token.expires\_in = Number.parseInt(sessionStorage.getItem(EIMS\_SESSION\_ExpirationTime));

return token;

}

setServiceToken(token: TokenModel) {

localStorage.setItem(SERVICE\_SESSION\_ID\_TOKEN, token.access\_token);

// sessionStorage.setItem(SERVICE\_SESSION\_REFERESH\_TOKEN, token.refresh\_token);

localStorage.setItem(SERVICE\_SESSION\_ExpirationTime, token.expires\_in.toString());

localStorage.setItem(SERVICE\_SESSION\_REFERESH\_TOKEN, token.refresh\_token);

}

getServiceToken() {

const token: TokenModel = new TokenModel();

token.access\_token = localStorage.getItem(SERVICE\_SESSION\_ID\_TOKEN);

token.refresh\_token = localStorage.getItem(SERVICE\_SESSION\_REFERESH\_TOKEN);

const expireTime = localStorage.getItem(SERVICE\_SESSION\_ExpirationTime);

if (expireTime) {

token.expires\_in = Number.parseInt(expireTime);

} else {

token.expires\_in = 300000;

}

return token;

}

geTenantName() {

return localStorage.getItem(TENANTNAME\_ID);

}

getAuthTypeId() {

return localStorage.getItem(AUTH\_TYPE\_ID);

}

setAuthTypeId(id) {

localStorage.setItem(AUTH\_TYPE\_ID, id);

}

getAppTitle() {

const title = localStorage.getItem(APP\_TITLE\_ID);

return title ? title : DEFAULT\_TITLE;

}

setAppTitle(titie) {

localStorage.setItem(APP\_TITLE\_ID, titie);

}

seTenantName(tenant) {

localStorage.setItem(TENANTNAME\_ID, tenant);

}

removeTenantName() {

localStorage.removeItem(TENANTNAME\_ID);

}

getTenantId() {

return localStorage.getItem(TENANT\_ID);

}

clear() {

const tenantName = this.geTenantName();

const appTitle = this.getAppTitle();

const authId = this.getAuthTypeId();

sessionStorage.clear();

localStorage.clear();

this.setAppTitle(appTitle);

this.seTenantName(tenantName);

this.setAuthTypeId(authId);

}

getAppFavIcon() {

const appFavIcon = localStorage.getItem(APP\_FAV\_ICON);

return appFavIcon ? appFavIcon : DEFAULT\_APP\_FAV\_ICON;

}

}

==================================================================

Session-storage.service.spec..ts

import { TestBed, inject } from '@angular/core/testing';

import { SessionStorageService } from '../../app/\_services/session-storage.service';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { of } from 'rxjs';

import { TokenModel } from '../\_models/TokenModel';

import {

EIMS\_SESSION\_ID\_TOKEN, EIMS\_SESSION\_REFERESH\_TOKEN, EIMS\_SESSION\_ExpirationTime, SERVICE\_SESSION\_ID\_TOKEN,

SERVICE\_SESSION\_REFERESH\_TOKEN, SERVICE\_SESSION\_ExpirationTime, TENANTNAME\_ID, AUTH\_TYPE\_ID, APP\_TITLE\_ID,

DEFAULT\_TITLE, APP\_FAV\_ICON, DEFAULT\_APP\_FAV\_ICON, TENANT\_ID } from

'../../app/\_constants/authConst';

describe('SessionStorageService', () => {

let sessionStorageService = SessionStorageService;

beforeEach(() => {

TestBed.configureTestingModule({

imports: [],

providers: [SessionStorageService],

declarations: []

});

sessionStorageService = TestBed.get(SessionStorageService);

});

it('should be created', () => {

expect(sessionStorageService).toBeTruthy();

});

it('should set key in session storage', inject([SessionStorageService], (sessionStorageService) => {

sessionStorageService.set('key', 'value');

expect(sessionStorageService.sessionStorgaeModel['key']).toEqual('value');

}));

it('should get key in session storage', inject([SessionStorageService], (sessionStorageService) => {

sessionStorageService.set('key', 'value');

let sessionValue = sessionStorageService.get('key');

expect(sessionValue).toEqual('value');

}));

it('should remove key from session storage', inject([SessionStorageService], (sessionStorageService) => {

sessionStorageService.set('key', 'value');

let sessionValue = sessionStorageService.get('key');

expect(sessionValue).toEqual('value');

sessionStorageService.remove('key');

sessionValue = sessionStorageService.get('key');

expect(sessionValue).toBeNull();

}));

it('should clear session storage model', inject([SessionStorageService], (sessionStorageService) => {

sessionStorageService.sessionStorgaeModel = { appId: 101 };

sessionStorageService.clear();

expect(sessionStorageService.sessionStorgaeModel.appId).toEqual(101);

}));

it('should set token in session storage', inject([SessionStorageService], (sessionStorageService) => {

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

sessionStorageService.setEIMSToken(token);

expect(sessionStorage.getItem(EIMS\_SESSION\_ID\_TOKEN)).toEqual(token.access\_token);

expect(sessionStorage.getItem(EIMS\_SESSION\_REFERESH\_TOKEN)).toEqual(token.refresh\_token);

expect(sessionStorage.getItem(EIMS\_SESSION\_ExpirationTime)).toEqual(token.expires\_in.toString());

}));

it('should get token in session storage', inject([SessionStorageService], (sessionStorageService) => {

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

sessionStorageService.setEIMSToken(token);

const result = sessionStorageService.getEIMSToken();

expect(result.access\_token).toEqual(token.access\_token);

expect(result.refresh\_token).toEqual(token.refresh\_token);

expect(result.expires\_in).toEqual(token.expires\_in);

}));

it('should set token in local storage', inject([SessionStorageService], (sessionStorageService) => {

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

sessionStorageService.setServiceToken(token);

expect(localStorage.getItem(SERVICE\_SESSION\_ID\_TOKEN)).toEqual(token.access\_token);

expect(localStorage.getItem(SERVICE\_SESSION\_ExpirationTime)).toEqual(token.expires\_in.toString());

expect(localStorage.getItem(SERVICE\_SESSION\_REFERESH\_TOKEN)).toEqual(token.refresh\_token);

}));

it('should get token in local storage', inject([SessionStorageService], (sessionStorageService) => {

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

sessionStorageService.setServiceToken(token);

const result = sessionStorageService.getServiceToken(token);

expect(result.access\_token).toEqual(token.access\_token);

expect(result.refresh\_token).toEqual(token.refresh\_token);

expect(result.expires\_in).toEqual(token.expires\_in);

}));

it('should set tenantName in local storage', inject([SessionStorageService], (sessionStorageService) => {

const tenantName = "tenantNameOIPA";

sessionStorageService.seTenantName(tenantName);

expect(localStorage.getItem(TENANTNAME\_ID)).toEqual(tenantName);

}));

it('should get tenantName in local storage', inject([SessionStorageService], (sessionStorageService) => {

const tenantName = "tenantNameOIPA";

sessionStorageService.seTenantName(tenantName);

var name = sessionStorageService.geTenantName();

expect(tenantName).toEqual(name);

}));

it('should set AuthtypeId in local storage', inject([SessionStorageService], (sessionStorageService) => {

const authTypeId = 1;

sessionStorageService.setAuthTypeId(authTypeId);

expect(localStorage.getItem(AUTH\_TYPE\_ID)).toEqual(authTypeId.toString());

}));

it('should get AuthtypeId in local storage', inject([SessionStorageService], (sessionStorageService) => {

const authTypeId = 1;

sessionStorageService.setAuthTypeId(authTypeId);

var typeId = sessionStorageService.getAuthTypeId();

expect(typeId).toEqual(authTypeId.toString());

}));

it('should set AppTitle in local storage', inject([SessionStorageService], (sessionStorageService) => {

const appTitle = "Unification";

sessionStorageService.setAppTitle(appTitle);

expect(localStorage.getItem(APP\_TITLE\_ID)).toEqual(appTitle);

}));

it('should get AppTitle in local storage', inject([SessionStorageService], (sessionStorageService) => {

const appTitle = "Unification";

sessionStorageService.setAppTitle(appTitle);

var titleName = sessionStorageService.getAppTitle();

expect(titleName).toEqual(appTitle);

}));

it('should remove tenantName from local storage', inject([SessionStorageService], (sessionStorageService) => {

const tenantName = "tenantNameOIPA";

sessionStorageService.seTenantName(tenantName);

let sessionValue = sessionStorageService.geTenantName();

expect(sessionValue).toEqual(tenantName);

sessionStorageService.removeTenantName();

sessionValue = sessionStorageService.geTenantName();

expect(sessionValue).toBeNull();

}));

it('should get TenantId from local storage', inject([SessionStorageService], (sessionStorageService) => {

const tenantId = "12aa";

localStorage.setItem(TENANT\_ID, tenantId);

const result = sessionStorageService.getTenantId();

expect(result).toEqual(tenantId);

}));

it('should get AppFavIcon from local storage', inject([SessionStorageService], (sessionStorageService) => {

const appFavIcon = "testFavIcon";

localStorage.setItem(APP\_FAV\_ICON, appFavIcon);

const result = sessionStorageService.getAppFavIcon();

expect(result).toEqual(appFavIcon);

}));

it('should clear the ', inject([SessionStorageService], (sessionStorageService) => {

var tenantName = "tenantName";

sessionStorageService.seTenantName(tenantName);

var appTitle = "Unification";

sessionStorageService.setAppTitle(appTitle);

var authTypeId = 1;

sessionStorageService.setAuthTypeId(authTypeId);

sessionStorageService.clear();

expect(sessionStorageService.geTenantName()).toEqual(tenantName);

expect(sessionStorageService.getAppTitle()).toEqual(appTitle);

expect(sessionStorageService.getAuthTypeId()).toEqual(authTypeId.toString());

expect(sessionStorage.length).toEqual(0);

expect(localStorage.length).toEqual(3);

expect(localStorage.getItem(APP\_TITLE\_ID)).toEqual(appTitle);

expect(localStorage.getItem(TENANTNAME\_ID)).toEqual(tenantName);

expect(localStorage.getItem(AUTH\_TYPE\_ID)).toEqual(authTypeId.toString());

}));

});

==================================================================

Authentication.service.ts

//import { CustomHttpService } from './custom-http.service';

import { HttpParams, HttpHeaders, HttpClient } from '@angular/common/http';

import { inject } from '@angular/core/testing';

import { Injectable } from '@angular/core';

import { Observable, interval } from 'rxjs';

import { environment } from '../../environments/environment';

import { SessionStorageService } from './session-storage.service';

import {

GET\_USER\_CONFIGURATIONS, TOKEN\_By\_LOGIN, SERVICE\_REFRESH\_TOKEN,

FORGOT\_PASSWORD\_API, APP\_NAME, REMOVE\_REFRESH\_TOKEN\_API, RESET\_PASSWORD, USER\_AUTHORIZATION\_BY\_USERID, TENANT\_ID

} from '../\_constants/authConst';

import { TokenModel } from '../\_models/TokenModel';

import { AuthType } from '../\_constants/authType';

@Injectable()

export class AuthenticationService {

public EIMSTimer: Observable<number>;

public ServiceTimer: Observable<number>;

constructor(private httpService: HttpClient, private sessionStorage: SessionStorageService) {

this.EIMSTimer = null;

this.ServiceTimer = null;

}

validateAuthentication() {

const session = this.sessionStorage.getServiceToken();

const token = session.access\_token;

const refreshToken = session.refresh\_token;

const tenantName = this.sessionStorage.geTenantName();

const authTypeId = this.sessionStorage.getAuthTypeId();

const tenantId = this.sessionStorage.getTenantId();

let res = false;

if (tenantName && authTypeId && authTypeId === AuthType.SAML.toString()) {

res = true;

} else if (tenantName && authTypeId && authTypeId === AuthType.Oauth2.toString()) {

res = true;

} else if (tenantName && authTypeId && authTypeId === AuthType.UserIdPassword.toString() && refreshToken) {

res = true;

} else {

res = false;

}

return res;

}

Login(userName, password): Observable<any> {

let params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = {

userName: userName,

password: password

};

const headers = new HttpHeaders({

// Authorization: 'Bearer ' + this.sessionStorage.getEIMSToken().access\_token,

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + TOKEN\_By\_LOGIN;

return this.httpService.post(url, body, { headers: headers, params: params });

}

getServicerefreshAccessToken(): Observable<any> {

const refreshToken = this.sessionStorage.getServiceToken().refresh\_token;

let params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = JSON.stringify(refreshToken);

const headers = new HttpHeaders({

// Authorization: 'Bearer ' + this.sessionStorage.getEIMSToken().access\_token.toString(),

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + SERVICE\_REFRESH\_TOKEN;

return this.httpService.post(url, body, { headers: headers, params: params });

}

setTokenFromResponse(data: any) {

const expireDate = new Date(data.accessTokenExpiryTime);

const utcDate = new Date();

const startTime = Date.UTC(utcDate.getUTCFullYear(), utcDate.getUTCMonth(), utcDate.getUTCDate(),

utcDate.getUTCHours(), utcDate.getUTCMinutes(), utcDate.getUTCSeconds(), utcDate.getUTCMilliseconds());

const expireTime = Date.UTC(expireDate.getUTCFullYear(), expireDate.getUTCMonth(), expireDate.getUTCDate(),

expireDate.getUTCHours(), expireDate.getUTCMinutes(), expireDate.getUTCSeconds(), expireDate.getUTCMilliseconds());

const model = new TokenModel();

model.expires\_in = (expireTime - startTime);

model.access\_token = data.accessToken;

model.refresh\_token = data.refreshToken;

this.sessionStorage.setServiceToken(model);

}

forgotPassword(email): Observable<any> {

// const accessToken = this.sessionStorage.getServiceToken().access\_token;

let params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = {

'Email': email

};

const headers = new HttpHeaders({

// Authorization: 'Bearer ' + this.sessionStorage.getEIMSToken().access\_token.toString(),

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + FORGOT\_PASSWORD\_API;

return this.httpService.post(url, body, { headers: headers, params: params });

}

RemoveRefreshToken(): any {

const params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = JSON.stringify(this.sessionStorage.getServiceToken().refresh\_token);

const headers = new HttpHeaders({

Authorization: 'Bearer ' + this.sessionStorage.getServiceToken().access\_token.toString(),

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + REMOVE\_REFRESH\_TOKEN\_API;

return this.httpService.post(url, body, { headers: headers, params: params });

}

RemoveRefreshTokenByParam(token, refreshToken) {

const params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = JSON.stringify(refreshToken);

const headers = new HttpHeaders({

Authorization: 'Bearer ' + token,

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + REMOVE\_REFRESH\_TOKEN\_API;

return this.httpService.post(url, body, { headers: headers, params: params });

}

resetLogin(userName, oldPassword, newPassword): any {

const params = new HttpParams({ /\*fromString: 'forKiosk=true'\*/ });

const body = {

'UserName': userName,

'OldPassword': oldPassword,

'NewPassword': newPassword

};

const headers = new HttpHeaders({

// Authorization: 'Bearer ' + this.sessionStorage.getEIMSToken().access\_token.toString(),

'TenantId': localStorage.getItem(TENANT\_ID),

'AppName': APP\_NAME,

'Content-Type': 'application/json'

});

const url = environment.baseUrl + RESET\_PASSWORD;

return this.httpService.post(url, body, { headers: headers, params: params });

}

clearSession() {

this.sessionStorage.clear();

}

}

==================================================================

Authentication.service.spec.ts

import { TestBed, inject } from '@angular/core/testing';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { of } from 'rxjs';

import { AuthenticationService } from './authentication.service';

import { SessionStorageService } from './session-storage.service';

import { TokenModel } from '../\_models/TokenModel';

fdescribe('Authentication Service', () => {

let authenticationService: AuthenticationService;

beforeEach(() => {

TestBed.configureTestingModule({

imports: [HttpClientModule],

providers: [AuthenticationService, SessionStorageService, HttpClient],

declarations: []

});

authenticationService = TestBed.get(AuthenticationService);

});

fit('should be created', () => {

expect(authenticationService).toBeTruthy();

});

fit('Should validate authentication token', inject([AuthenticationService, SessionStorageService], (authenticationService, sessionStorage) => {

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

spyOn(sessionStorage, 'getServiceToken').and.returnValue(token);

var result = authenticationService.validateAuthentication();

expect(result).toEqual(true);

}));

fit('Should return response from Login', inject([AuthenticationService, SessionStorageService, HttpClient], (authenticationService, sessionStorage, httpClient) => {

var userName = 'UserName';

var password = 'password';

var response = {

"status": {

"message": "",

"code": "200"

},

"data": {

"accessToken": "access\_token",

"refreshToken": "refresh\_token",

"accessTokenExpiryTime": "expiry\_time",

"isAutoGeneratedPassword": "auto\_generated",

}

};

spyOn(httpClient, 'post').and.returnValue(of(response));

authenticationService.Login(userName, password).subscribe((response) => {

expect(response.data.accessToken).toEqual('access\_token');

expect(response.data.refreshToken).toEqual('refresh\_token');

expect(response.data.accessTokenExpiryTime).toEqual('expiry\_time');

expect(response.data.isAutoGeneratedPassword).toEqual('auto\_generated');

expect(response.status.code).toEqual("200");

});

}));

fit('Should return response from getServicerefreshAccessToken', inject([AuthenticationService, SessionStorageService, HttpClient], (authenticationService, sessionStorage, httpClient) => {

var response = {

"status": {

"message": "",

"code": "200"

},

"data": {

"accessToken": "access\_token",

"refreshToken": "refresh\_token",

"accessTokenExpiryTime": "expiry\_time",

"isAutoGeneratedPassword": "auto\_generated",

}

};

let token: TokenModel = { access\_token: 'access\_token', expires\_in: 20, refresh\_token: 'refresh\_token', token\_type: 'token\_type', scope: 'scope' };

spyOn(sessionStorage, 'getServiceToken').and.returnValue(token);

spyOn(httpClient, 'post').and.returnValue(of(response));

authenticationService.getServicerefreshAccessToken(token.refresh\_token).subscribe((response) => {

expect(response.data.accessToken).toEqual("access\_token");

expect(response.data.refreshToken).toEqual("refresh\_token");

expect(response.data.accessTokenExpiryTime).toEqual("expiry\_time");

expect(response.data.isAutoGeneratedPassword).toEqual("auto\_generated");

expect(response.status.code).toEqual("200");

});

}));

});