

Documentation de Sécurisation de la Partie Frontend LabXpert

Introduction

Cette documentation décrit les mesures de sécurité mises en place dans la partie frontend de l'application de gestion immobilière. Ces mesures visent à protéger les données sensibles et à garantir l'accès sécurisé aux fonctionnalités de l'application.

Utilisation de JWT pour l'Authentification

JSON Web Tokens (JWT) sont utilisés pour gérer l'authentification des utilisateurs. Lorsqu'un utilisateur se connecte avec succès, un token JWT est généré et renvoyé par le serveur. Ce token est ensuite stocké dans le navigateur de l'utilisateur.

Intercepteur HTTP pour l'Autorisation

Un intercepteur HTTP est utilisé pour intercepter chaque requête HTTP sortante de l'application. Cet intercepteur vérifie si l'utilisateur est authentifié en récupérant le token JWT stocké dans le navigateur. Si l'utilisateur est authentifié, le token est ajouté à l'en-tête de la requête pour autoriser l'accès aux ressources protégées sur le serveur.

```
export class AuthentificationInterceptorService implements HttpInterceptor {  
  constructor(private authService : AuthenticationService) { }  
  
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>  
  {  
    return this.authService.user.pipe(take(1), exhaustMap(user => {  
      if(!user)  
      {  
        return next.handle(req);  
      }  
  
      const modifiedRequest = req.clone({headers: new HttpHeaders({'Authorization': 'Bearer ' + user.token})});  
      return next.handle(modifiedRequest);  
    }));  
  }  
}
```

Garde de Route pour le Contrôle d'Accès

Des gardes de route sont mis en place pour contrôler l'accès aux différentes pages de l'application en fonction du rôle de l'utilisateur. Par exemple, seuls les utilisateurs avec les rôles "Admin" ou "Responsable" peuvent accéder au tableau de bord administratif, tandis que les techniciens sont redirigés vers la page des patients.

```
const routes: Routes = [
  { path: 'admin', component: NavbarComponent, children: [
    { path: 'dashboard', component: DashboardComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'fournisseurs', component: FournisseursComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'fournisseurs/add', component: SaveFournisseurComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'fournisseurs/update/:id', component: UpdateFournisseurComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'reactifs', component: ReactifsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable', 'Technicien' } },
    { path: 'reactifs/add', component: SaveReactifComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'reactifs/update/:id', component: UpdateReactifComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'materials', component: MaterialsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'materials/add', component: SaveMaterialComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable' } },
    { path: 'patients', component: PatientsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Responsable', 'Technicien' } },
    { path: 'patients/save', component: SavePatientComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
    { path: 'patients/update/:id', component: UpdatePatientComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
    { path: 'echontillons/echontillon-analyses/:id', component: EchontillonAnalysesComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Technicien', 'Responsable' } },
    { path: 'echontillons/save', component: SaveEchontillonComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
    { path: 'echontillons', component: EchontillonsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Technicien', 'Responsable' } },
    { path: 'echontillons/update/:id', component: UpdateEchontillonComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
    { path: 'echontillon-materials', component: EchontillonMaterialsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Admin', 'Technicien', 'Responsable' } },
    { path: 'echontillon-materials/save', component: SaveEchontillonMaterialsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
    { path: 'echontillon-materials/update/:id', component: UpdateEchontillonMaterialsComponent, canActivate: [AuthenticationGuardService], data: { role: 'Technicien', 'Responsable' } },
  ] },
  { path: 'auth', component: AuthenticationComponent },
  { path: '', redirectTo: 'auth', pathMatch: 'full' },
];
```

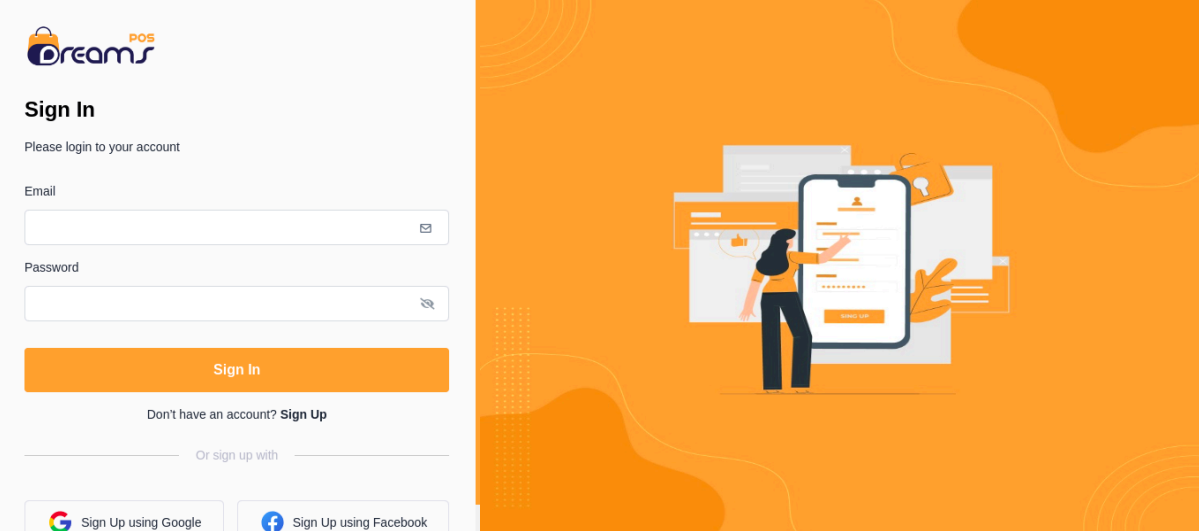
Stockage Sécurisé du Token


Le token JWT est stocké de manière sécurisée dans le navigateur de l'utilisateur à l'aide du stockage local (localStorage). Cela permet de maintenir la session de l'utilisateur même après la fermeture et la réouverture du navigateur.

Déconnexion Automatique

Une fonctionnalité de déconnexion automatique est mise en place pour déconnecter l'utilisateur après une période d'inactivité. Un minuteur est utilisé pour déclencher la déconnexion automatique une fois que le token JWT a expiré.

Mon interface de login





Sign In

Please login to your account


Email


Password

Sign In


Don't have an account? [Sign Up](#)

Or sign up with

 Sign Up using Google

 Sign Up using Facebook

Validation de cette form



Sign In

Please login to your account

Email

Email is required.

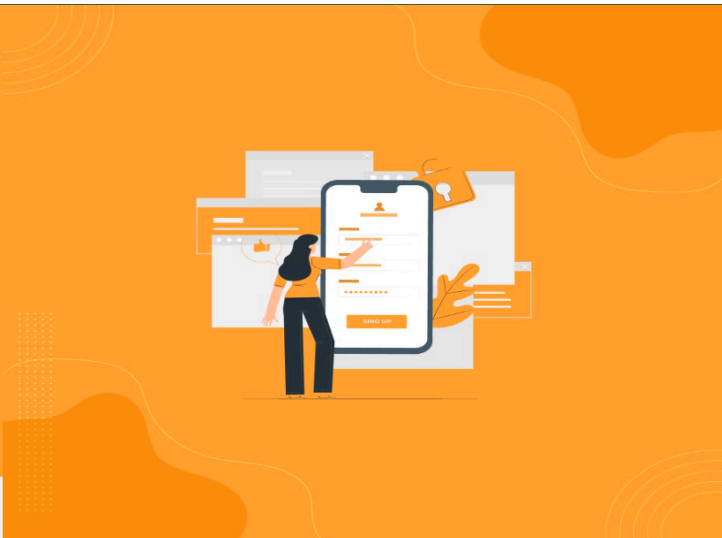
Password

Password is required.

Sign In

Don't have an account? [Sign Up](#)

Or sign up with



Authentification service avec ces méthodes

```
export class AuthentificationService {  
  
    private baseUrl = 'http://localhost:8081';  
    jwtHelperService = new JwtHelperService();  
    user = new BehaviorSubject<LoggerUser | null>(null);  
    tokenExpirationTimer: any;  
  
    constructor(private httpClient: HttpClient, private router: Router) { }  
  
    public login(user: LoginRequest) : Observable<LoginResponse>  
    {  
        const formData = new FormData();  
        formData.append("username", user.username);  
        formData.append("password", user.password);  
  
        return this.httpClient.post<LoginResponse>(this.baseUrl + "/login", formData);  
    }  
}
```

```

saveToken(jwtToken : LoginResponse)
{
  const decodedAccessToken = this.jwtHelperService.decodeToken(jwtToken.accessToken);
  const loggedUser = new LoggerUser(decodedAccessToken.sub, decodedAccessToken.roles, jwtToken.accessToken);
  this.user.next(loggedUser);
  this.autoLogout(this.getExpirationDate(decodedAccessToken.exp).valueOf() - new Date().valueOf());
  localStorage.setItem('userData', JSON.stringify(loggedUser));
  this.redirectLoggedInUser(decodedAccessToken, jwtToken.accessToken);
}

```

```

getExpirationDate(exp : number)
{
  const date = new Date(0);
  date.setUTCSeconds(exp);

  return date;
}

redirectLoggedInUser(decodedToken: any, accessToken:string)
{
  if(decodedToken.roles.includes("Admin") || decodedToken.roles.includes("Responsable")) this.router.navigate(['/admin/patients']);
  else if(decodedToken.roles.includes("Technicien"))
  {
    this.router.navigateByUrl("admin/patients")
  }
}

```

```

logout()
{
  localStorage.clear();
  this.user.next(null);
  this.router.navigate(['/']);

  if(this.tokenExpirationTimer)
  {
    clearTimeout(this.tokenExpirationTimer);
  }

  this.tokenExpirationTimer = null;
}

```

```

autoLogin()
{
  const userData: {
    username: string,
    roles: string[],
    _token: string,
    _expiration: Date
  } = JSON.parse(localStorage.getItem('userData'));

  if(!userData) return;

  const loadedUser = new LoggerUser(userData.username, userData.roles, userData._token, new Date(userData._expiration));

  if(loadedUser.token)
  {
    this.user.next(loadedUser);
    this.autoLogout(loadedUser._expiration.valueOf() - new Date().valueOf())
  }
}

```

```

autoLogout(expirationDuration : number)
{
  this.tokenExpirationTimer = setTimeout(() => {
    this.logout();
  }, expirationDuration);
}

```