**Présentation Architecture Angular**

Projet réalisé par Valentin CIRCOSTA





EFREI

ANGULAR

Table des matières

[Point d’entrée 2](#_Toc209128761)

[src/main.ts : Bootstrap de l’application 2](#_Toc209128762)

[src/app/app.config.ts : Configuration globale 3](#_Toc209128763)

[src/app/app.component.ts : Composant racine 4](#_Toc209128764)

[Routing et Navigation 5](#_Toc209128765)

# Point d’entrée

## src/main.ts : Bootstrap de l’application

Ce fichier est le point d’entrée de l’application Angular.  
Il initialise le projet, configure les services principaux et démarre le composant racine.

Rôles principaux :

* Démarrage : bootstrapApplication(AppComponent) lance l’application avec l’approche standalone (plus moderne qu’AppModule).
* Routing : provideRouter(routes) enregistre les routes définies dans app.routes.ts.
* HTTP : provideHttpClient active HttpClient et applique deux intercepteurs :
  + authInterceptor → ajoute automatiquement le JWT dans les requêtes.
  + errorInterceptor → gère les erreurs globalement (ex. mauvais mot de passe, accès interdit).
* Composant racine : AppComponent contient le <router-outlet> pour afficher les pages.
* Gestion des erreurs : .catch(err => console.error(err)) log les erreurs critiques au lancement.

En résumé : main.ts est le centre de configuration qui lance l’app, active le routing, sécurise les appels HTTP et gère les erreurs globales.

## src/app/app.config.ts : Configuration globale

Rôles principaux :

* Zone.js : provideZoneChangeDetection({ eventCoalescing: true }) → optimise la détection des changements en regroupant les événements pour de meilleures performances.
* Routing : provideRouter(routes) → enregistre toutes les routes définies dans app.routes.ts.
* HTTP Client : provideHttpClient(...) → configure les appels HTTP avec deux intercepteurs globaux :
  + authInterceptor : ajoute automatiquement le JWT aux requêtes.
  + errorInterceptor : centralise la gestion des erreurs (ex. mauvais mot de passe, accès interdit).

En résumé : ce fichier définit la configuration principale de l’app Angular (perf, routes, HTTP + sécurité/erreurs).

## src/app/app.component.ts : Composant racine

Rôle principal : c’est le point d’entrée visuel de l’app, celui qui contient le router-outlet pour afficher les différentes pages selon les routes.

Imports :

* RouterOutlet & RouterModule → gestion de la navigation.
* CommonModule & HttpClientModule → fonctionnalités Angular de base et appels HTTP.

Services injectés :

* AuthService → gestion de l’authentification.
* Router → permet de rediriger l’utilisateur.

Fonction logout() :

* Déconnecte l’utilisateur via auth.logout().
* Redirige automatiquement vers la page de connexion /auth/login.

En résumé : AppComponent est le composant racine qui gère l’affichage des pages et permet la déconnexion avec redirection vers le login.

# Routing et Navigation

## src/app/app.routes.ts : Routes principales

* auth → charge les routes liées à l’authentification (/auth/...).
* reservations → accessible uniquement si l’utilisateur est connecté (authGuard).
* admin → réservé aux administrateurs (adminGuard).
* '' → redirige automatiquement vers la page de login (/auth/login).

En résumé : ce fichier définit la navigation principale de l’application, avec une sécurité basée sur des guards pour protéger certaines pages (auth et admin).

## src/app/features/\*/routes.ts : Routes par feature

Auth (AUTH\_ROUTES)

* Routes : /login, /register
* Composants associés : LoginComponent, RegisterComponent
* Rôle : gérer l’accès à l’application (connexion / inscription).

Reservations (RESERVATIONS\_ROUTES)

* Routes : /reservations, /reservations/new, /reservations/:id
* Composants associés : ReservationListComponent, ReservationFormComponent, ReservationDetailComponent
* Rôle : gérer le cœur fonctionnel côté utilisateur (consulter, créer, voir les détails d’une réservation).

Admin (ADMIN\_ROUTES)

* Routes : /admin, /admin/users, /admin/computers
* Composants associés : AdminDashboardComponent, UserManagementComponent, ComputerManagementComponent
* Rôle : offrir une interface de gestion réservée aux administrateurs (utilisateurs et ordinateurs).

La gestion des routes dans ce projet suit une logique de modularité par feature, ce qui permet de bien séparer les responsabilités :

* Auth assure l’entrée et l’accès sécurisé à l’application,
* Reservations gère les fonctionnalités principales offertes aux utilisateurs,
* Admin fournit des outils de supervision et de gestion réservés aux administrateurs.

Cette organisation rend l’application plus claire, plus maintenable et plus évolutive, car chaque module est indépendant et peut être développé, testé ou modifié sans impacter directement les autres.

## Lazy Loading : Comment ça fonctionne