esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Lazy Loading

- Mise en scène
- Présentation de Lazy loading
- Mise en pratique de Lazy loading
- Exemple d’application

Mise en scène

Introduction

- La vitesse à laquelle s'affiche un site web est l'un des critères les plus essentiels pour l'utilisateur. Et cette vitesse s'apprécie en secondes. Au-delà de 3 secondes 57% des utilisateurs quittent purement et simplement le site.
- Il faut comprendre alors comment fonctionne notre application afin d’améliorer la performance de son chargement.
- Le premier script exécuté dans une application angular est main.js

Introduction

- La commande ng build permet de compiler un projet angular et générer un dossier dist. Ce dossier contient des fichier qu'on peut les dépolyer sur un serveur web.
- La compilation de notre code source génère notamment un fichier **main.js** qui contient le code de tous les composants et leurs dépendances qui sont chargés au niveau du module racine AppModule.
- Ce fichier et d'autres seront appelés lors de l'affichage du site Web. Plus le nombre de pages sera grand, plus le fichier sera volumineux.

Problématique

Pour une application qui contient plusieurs modules avec leurs composants, services…. , la taille du fichier main.js devient grande et par la suite le temps de son chargement devient assez lent.

Et par conséquent les utilisateurs qui utilise cette application vont partir plutôt que d'utiliser cette application !

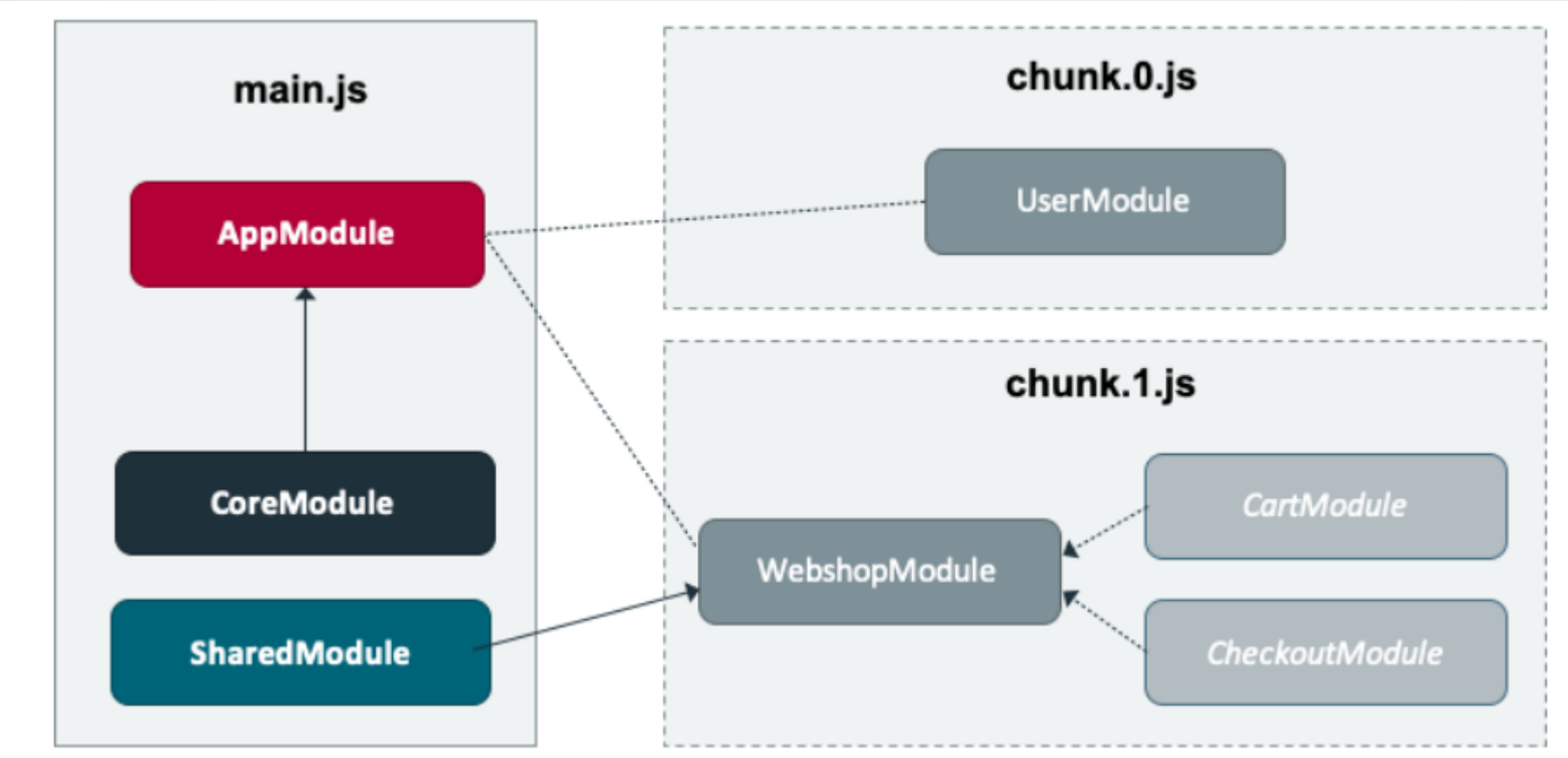
- Que faut il faire pour diminuer le temps de chargement de l'application?

Solution

- La solution proposée par Angular est de ne charger, au départ, que les ressources essentiels pour le démarrage de l'application. Le reste des ressources seront chargées à la demande de l'utilisateur.

Il s'agit du Lazy Loading

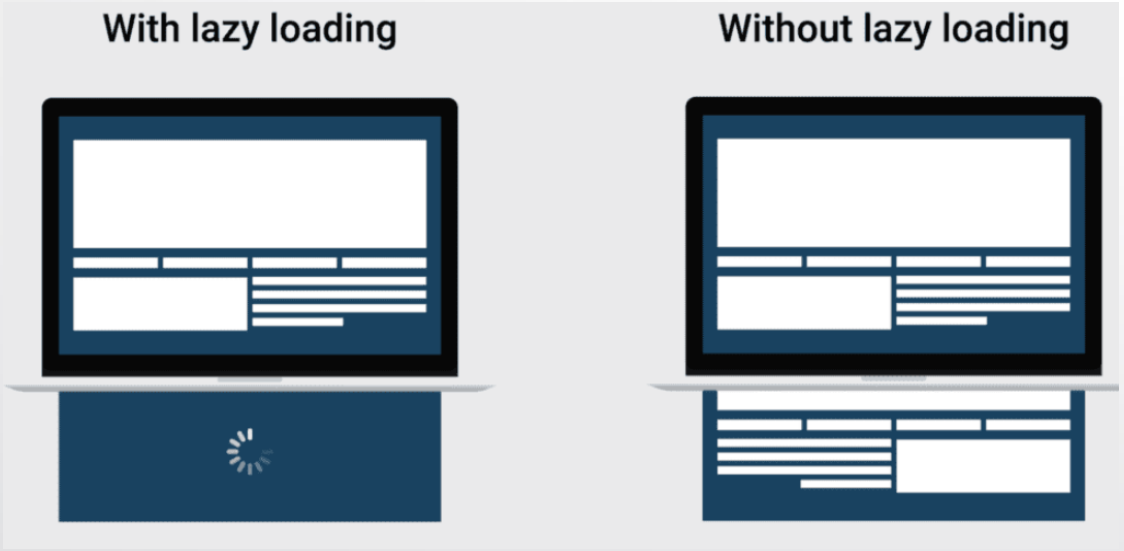
Solution

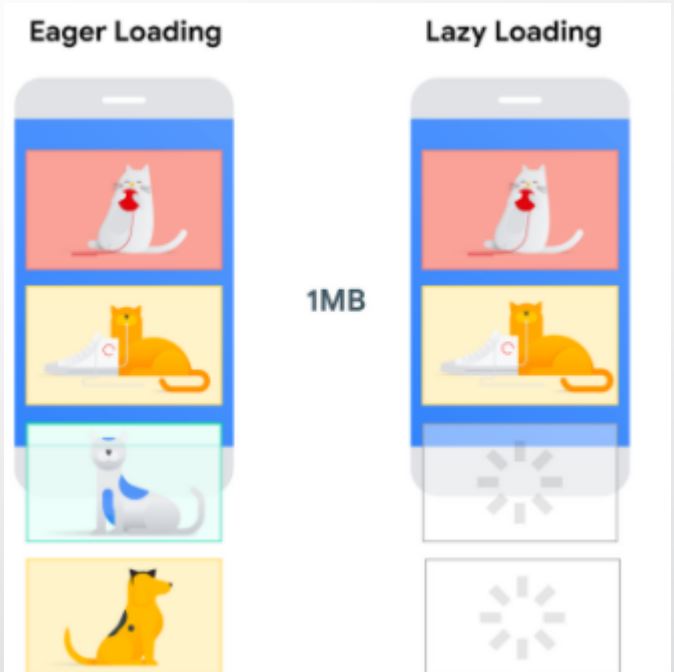

main.js
AppModule
CoreModule
SharedModule
chunk.0.js
UserModule
chunk.1.js
WebshopModule
CartModule
CheckoutModule

Présentation de Lazy Loading

Lazy Loading - Définition

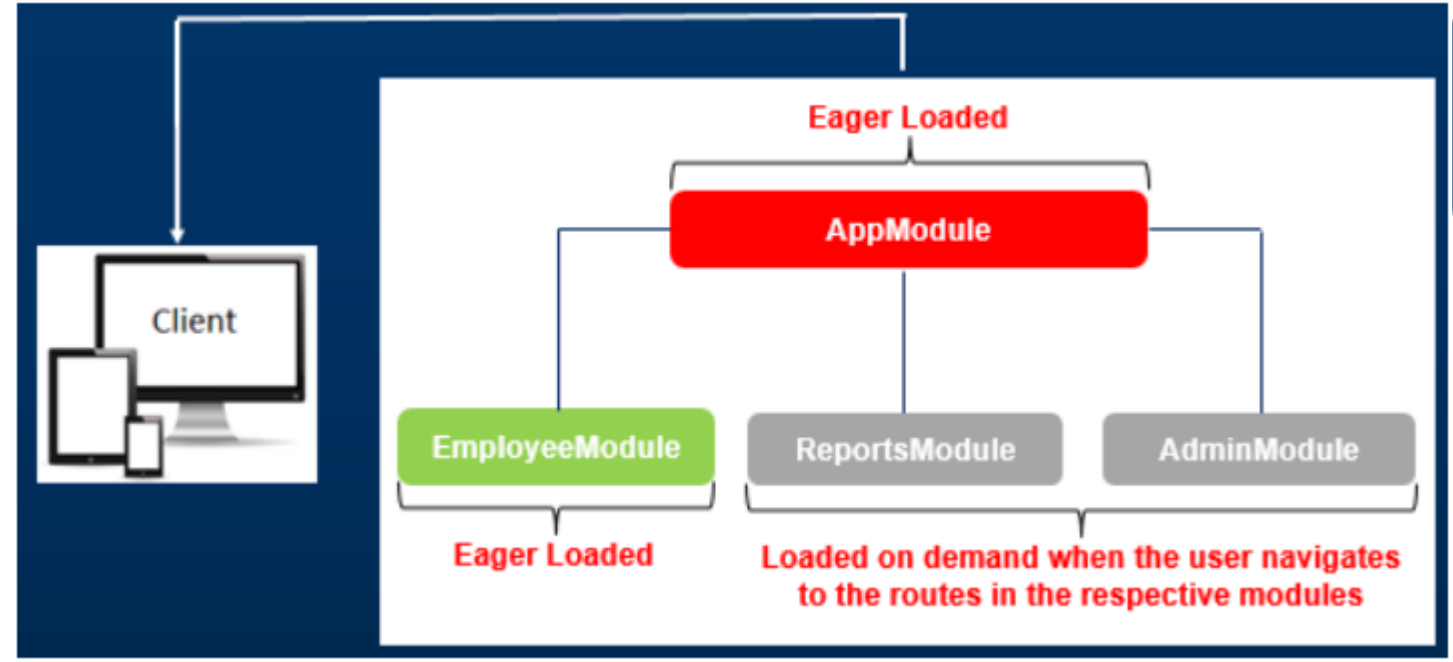
- « Lazy loading » ou « chargement paresseux » est la pratique consistant à retarder le chargement ou l'initialisation des ressources ou des objets jusqu'à ce qu'ils soient réellement nécessaires pour améliorer les performances et économiser les ressources système.





Lazy Loading dans Angular

- En parlant de lazy loading dans Angular, nous parlons de **Lazy loaded modules**. Ces derniers sont des modules qui sont chargés à la demande lorsque l'utilisateur navigue vers les routes de ces modules.



eager : le navigateur charge l'objet directement lors de la consultation de la page ;

Lazy Loading - Avantages

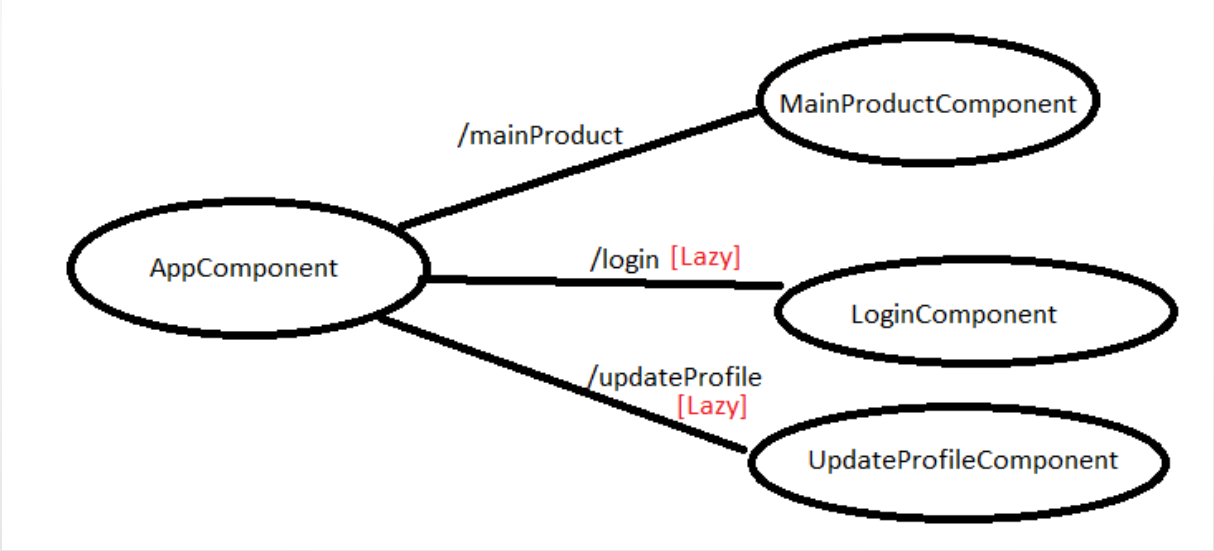
- Le chargement initial de l'application est plus rapide car un seul module sera chargé lors de lancement de l'application
- Charger les modules demandées par l'utilisateur
- Continuer à implémenter de nouveaux modules sans alourdir votre application
- Les nouveaux modules sont chargés à la volée du client
- le Lazy Loading nous oblige d'organiser le code source de l'application en regroupant les fonctionnalités par modules.

Lazy Loading - Inconvénients

- Code un peu compliqué
- Le retour en arrière peut ne pas être possible si la structure de la page n'est pas optimale.
- L'application peut planter ou ne plus répondre si l'appareil se déconnecte pendant quelques secondes : par exemple, réfléchissez à ce qui se passe lorsque l'utilisateur perd sa connexion Internet.
- Le chargement paresseux peut parfois affecter le classement du site Web sur les moteurs de recherche, en raison d'une mauvaise indexation du contenu déchargé.

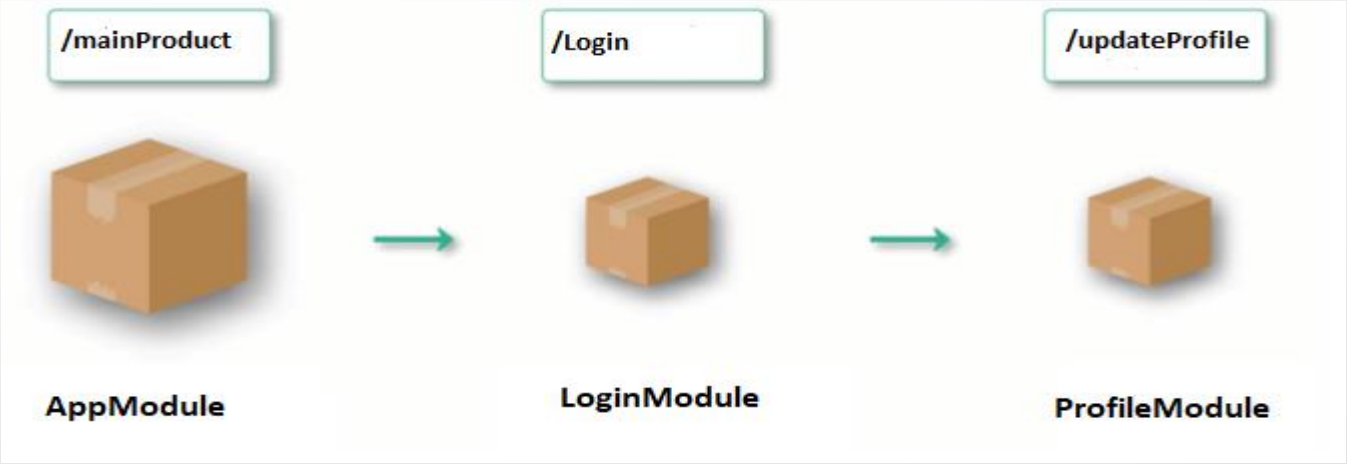
Exemple

- Soit une application E-commerce dont la page home contient une liste de produit contenu dans MainProductComponent. L'utilisateur, en consultant cette page, n'est pas forcément authentifié. Une fois l'utilisateur s'authentifie il peut éditerulter son profil.



Exemple

- En appliquant le Lazy Loading on ne charge au départ que le module racine contenant MainProductComponent.
- Puis si l'utilisateur va accéder à la page d'authentification, on va charger seulement le module Login.
- Enfin si on va accéder à la page web update profile , on va charger le module correspondant



Mise en pratique de Lazy Loading

Etapes

Pour mettre en pratique le lazy loading, il faut:

1. Créer des modules ainsi que leurs modules de routage
2. Associer pour chaque modules les composants correspondants
3. Alimenter les modules de routages des modules crées
4. Mettre à jour le module de routage du module racine en définissant les routes vers ses composants **ainsi que les routes vers les modules lazy loaded**

Création de modules

- Pour créer un module dans angular vous pouvez lancer la commande: **ng generate module nom_du_module** ou bien son raccourci **ng g m nom_du_module**
- Pour créer un module avec son module de routing la commande devient: **ng g m nom_du_module --routing**
- Par défaut le module crée n'est pas attaché au module racine. Pour attacher le module créé automatiquement au module racine lors de sa création la commande devient

ng g m nom_du_module --routing --module=app

Création d'un composant dans un module

Il existe deux méthodes pour créer un composant dans un sous-module

- **La première méthode:**Se positionner dans le dossier du module concerné puis lancer la commande suivante dans le terminale:

ng g c nom_du_composant

- **La deuxième méthode:**

ng g c nom_du_module/nom_du_composant

Exemple : ng g c Product/MainProduct

Peu importe la méthode utilisée uniquement le sous-module concerné est mis à jour. Autrement dit le composant créé est ajouté dans la liste des déclrations du module en question, le module racine n'est pas mis à jour

CommonModule

On remarque que dans les modules générés le module **CommonModule** est importé. Ce module contient les pipes et les directives qu'on va utiliser dans les composants du sous-module crée.

Question : Est ce que ce module 'CommonModule' existe dans le module racine?

Réponse : Non

Pourquoi?

Car le module racine importe le module **BrowserModule** et ce dernier importe le module CommonModule

.

Routing du module racine - Syntaxe

Pour charger les modules « paresseux », utilisez loadChildren (au lieu de component) dans la configuration de vos routes dans AppRoutingModule comme suit.

```
const routes: Routes = [
  {
    path: 'items',
    loadChildren: () => import('./items/items.module').then(m => m.ItemsModule)
  }
];
```

Nom fichier du module

Nom dossier

Nom du module

Exemple d’application

Application

- Créer le module LoginModule : **ng g m loginModule**
- Créer les composant Authenticate: **ng g c Login/authenticate**
- Créer les composant Forgotpassword: **ng g c Login/forgotpassword**
- Ajouter les routes pour les deux composants Authenticate et Forgotpassword dans LoginRoutingModule

```
const routes: Routes = [
  {path:'', component:AuthenticateComponent},
  {path:'forgotpwd', component:ForgotpasswordComponent},
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class LoginRoutingModule { }
```

Composant chargé par défaut

Application

- On va faire une référence du fichier de routing du module LoginModule dans le fichier de routing principale à l'aide de **loadChildren**

```
const routes: Routes = [
{path:'admin', loadChildren:
()=>import('./login/login.module').then(m=>m.LoginModule)} ]
```

Chemin vers le fichier du module

Nom du module

Néthographie

- https://sortable.com/blog/how-to-guide/how-to-optimize-user-experience-with-lazy-loading/

Merci de votre attention