ITÉRATION 2

Introduction à Angular

Modalités

- Travail individuel
- 2 jours en présentiel

Objectifs

Aborder les principales notions du framework Angular.

Compétences

- Installer l'environnement de développement
- Créer un composant Angular
- Faire un affichage conditionnel
- Faire une boucle sur un affichage à partir de données
- Passer des données à un composant
- Récupérer des données depuis un composant
- Mettre en place le routing Angular
- Effectuer des requêtes vers une api
- Utiliser les formulaires Angular
- Mettre en place des tests unitaires

1.1 — Introduction

Le principe des Single Page Applications (SPA) est de construire des applications web qui fonctionnent sur une seule page "technique", en chargeant dynamiquement le contenu au lieu de charger une nouvelle page à chaque interaction de l'utilisateur. Cela permet une expérience utilisateur plus fluide et rapide car le chargement des ressources est optimisé. En utilisant une SPA, les développeurs peuvent créer des applications web avec une architecture claire et modulaire, facilitant la maintenance et l'évolutivité des projets.

Il existe différents frameworks pour la construction de SPA, Angular est un choix populaire car il offre une architecture solide et un ensemble très complet d'outils déjà intégrés. C'est un framework bien établi et documenté, qui est régulièrement mis à jour.

Vous allez découvrir les bases de ce framework à travers la création d'un exemple de site très simple promouvant un festival de musique, avec une page d'accueil et une page contenant une liste d'artistes.

1.2 — Installation et création d'un projet Angular

1h — Présentiel

Pour cela vous allez devoir créer votre première application Angular !

Angular fournit une interface en ligne de commandes pour créer et gérer des projets. Cette interface contient diverses commandes pour générer un nouveau projet, ajouter de nouveaux éléments à l'application, lancer les tests unitaires et bien plus encore.

Vous pouvez installer la CLI Angular avec le gestionnaire de paquet *npm*.

TODO

➔ Consulter les ressources.

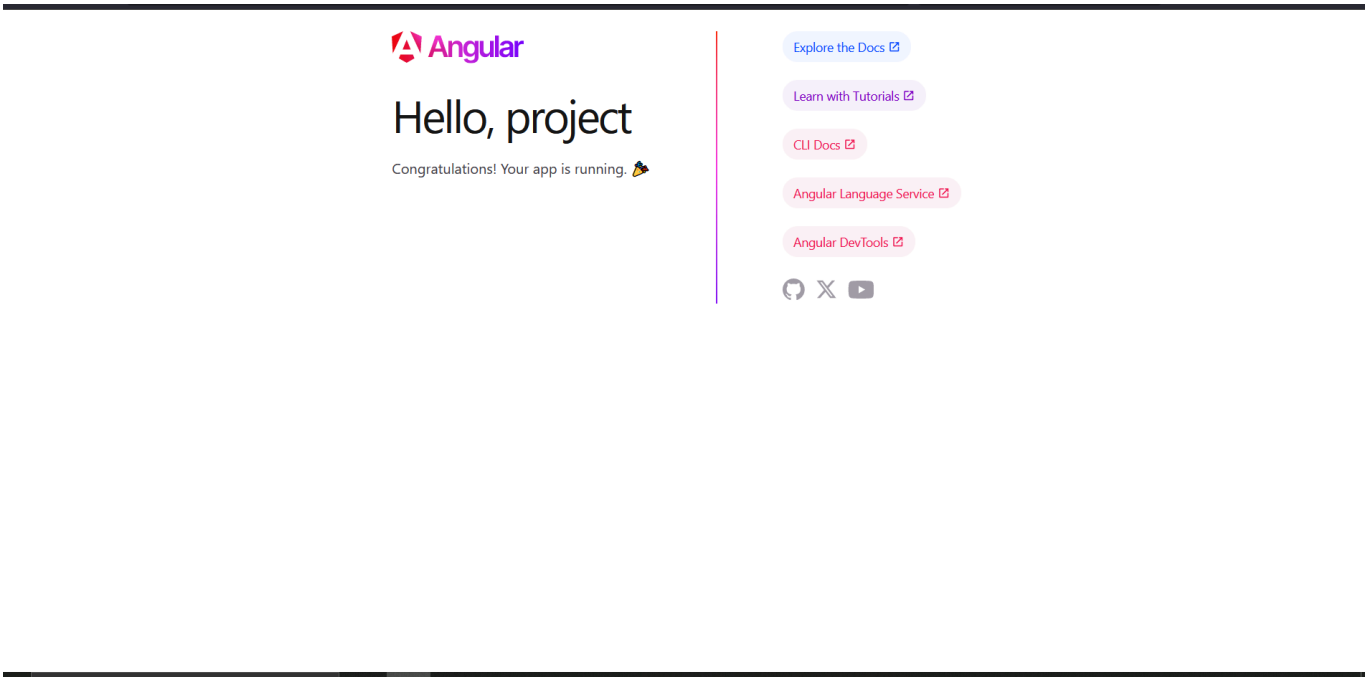
➔ Installez la CLI Angular 18.

NOTE: Il est important de bien installer la dernière version **Angular 18** pour avoir accès à toutes les dernières fonctionnalités. Vous pouvez le vérifier avec la commande ng --version qui devrait afficher « Angular CLI: 18.0.1 ».

➔ Créez un nouveau projet Angular utilisant les feuilles de style SCSS et avec le SSR activé.

➔ Lancez votre projet Angular et affichez la page d'accueil.

Après avoir lancé votre projet, vous devriez arriver sur une page de ce type :



Prenez un temps pour explorer les différents fichiers et vous familiariser avec l'arborescence. N'hésitez pas aussi à lire un peu de documentation sur Angular.

TODO

➔ Expliquer dans un mémo la différence entre NodeJS et Angular.

➔ Expliquer dans votre mémo à quoi servent les fichiers "package.json" et "angular.json" .

➔ Lister dans votre mémo les commandes de base de la CLI Angular.

RESSOURCES

- https://angular.dev/overview
- https://angular.dev/tools/cli#

1.3 — Page d'accueil et liste des profils

4h — Présentiel

Nous allons commencer par créer nos deux pages « accueil » et « liste des artistes » qui, dans un premier temps, ne fera qu' un affichage. Nous ajouterons la possibilité d'éditer la liste dans un second temps.

Il nous faudra aussi une barre de navigation pour naviguer d'une page à l'autre.

Points importants :

➔ Pensez bien à typer vos variables, retours de fonctions, etc... , comme vu précédemment dans l'introduction à Typescript.

➔ Votre application Angular utilise les feuilles de style SCSS, une surcouche du CSS qui apporte de nouvelles fonctionnalités (variables, imbrication de sélecteurs, etc...). N'hésitez pas à vous renseigner et à utiliser les possibilités offertes par cette surcouche (la doc est dans les ressources).

➔ Il est essentiel de découper et de cloisonner au maximum les différentes parties de votre application. Pensez donc à bien séparer les différentes parties de votre application en différents composants. Par exemple, dans cette première partie vous devriez avoir au minimum un composant par page et un pour la barre de navigation. Vous pouvez générer de nouveaux composants avec la commande ng generate component .

➔ Une application Angular est composée de plusieurs types de fichiers : modules, composants, modèles de données, services, etc... . Il est donc important de bien séparer les différentes parties de votre application et d'organiser le rangement des différents fichiers pour pouvoir continuer à s'y retrouver au fur et à mesure que votre application grandira. Voici un exemple d'arborescence que vous pouvez suivre :

```
├── app
│   ├── app.module.ts
│   ├── app.routing.ts
│   ├── components
│   │   ├── homepage
│   │   │   ├── homepage.component (ts,html,scss,spec.ts)
│   ├── models
│   ├── services
│   ├── artists
│   │   ├── artists.module.ts
│   │   ├── artists.routing.ts
│   │   ├── artists.component (ts,html,scss,spec.ts)
│   │   ├── artist-list
│   │   │   ├── artist-list.component (ts,html,scss,spec.ts)
│   │   ├── artist
│   │   │   ├── artist.component (ts,html,scss,spec.ts)
│   │   ├── models
│   │   │   ├── artist.model.ts
│   │   ├── services
│   │   │   ├── artists.web-service.ts
```

TODO

➔ Une page « **Accueil** » contenant à minima un **titre**, une **image** et un **paragraphe** qui apparaît/disparaît à l'appui d'un bouton « **plus** ».

➔ Une page « **Liste des artistes** » de quelques artistes contenant au moins pour chacun un **nom** et une **photo**. Stockez la donnée en dur dans le code dans un premier temps. Vous pouvez découper cette page en deux composants : un composant « **ArtistList** » pour gérer la liste et un composant « **Artist** » pour gérer l'affichage individuel d'un artiste.

➔ Une **barre de navigation** permettant de passer d'une page à l'autre grâce au routing Angular .

Questions :
⇒ Comment marche la détection des changements du DOM d'une liste utilisée dans une boucle *ngFor ?
⇒ Comment marche la détection des changements du DOM d'une variable récupérée depuis un @Input passé par le parent ? Quelle est la particularité de la détection dans le cas où la variable récupérée depuis un @Input est un objet ?

À partir de ces questions et de vos recherches personnelles, ajoutez à votre mémo une explication de la détection des changements Angular.

TODO

➔ Ajoutez à votre mémo une partie sur la détection des changements d'Angular.

RESSOURCES

- https://angular.dev/guide/components
- https://sass-lang.com/documentation/
- https://angular.dev/guide/directives#adding-or-removing-an-element-with-ngif

- https://angular.dev/guide/templates/binding
- https://angular.dev/guide/directives#listing-items-with-ngfor
- https://angular.dev/guide/components/inputs
- https://angular.dev/guide/routing/common-router-tasks

1.4 — Page d'accueil et liste des profils

5h — Présentiel

Il est maintenant temps de mettre en place la possibilité d'ajouter ou de supprimer un artiste de notre liste.

NOTE: La **liste des artistes** se réinitialisera au rechargement de l'application / de la page car elle est stockée en dur dans l'un de vos composants. Ce n'est pas grave pour l'instant car nous souhaitons nous concentrer sur les mécaniques de suppression et d'ajout.

TODO

➔ Un bouton « **supprimer** » à côté de chaque artiste qui permet de supprimer l'artiste de la **liste des artistes**.

➔ Un nouveau composant « **ArtistForm** » qui permet de renseigner les informations d'un profil. Le nouvel artiste est ajouté à la **liste des artistes** au clic d'un bouton « **ajouter** ». Ce composant doit utiliser les reactive forms Angular.

RESSOURCES

- https://angular.dev/guide/components/output-fn
- https://angular.dev/guide/forms/reactive-forms

1.5 — Implémentation de tests unitaires

4h — Présentiel

Maintenant que nous avons une base fonctionnelle, il est important de s'assurer qu'elle reste stable et cohérente au fil des mises à jour et des modifications. Pour cela, nous allons ajouter des tests unitaires à notre application. Les tests unitaires permettent de vérifier le comportement attendu de chaque fonction ou composant individuellement, afin de détecter rapidement les éventuels problèmes ou régressions lors de la modification de code.

Angular inclut par défaut Jasmine pour l'écriture, et Karma pour l'exécution automatique, des tests unitaires. Vous avez d'ailleurs peut-être déjà remarqué les fichiers “spec.ts” créés à la génération d'un composant.

Écrivez des tests unitaires pour les composants liés à votre liste d’artistes.

NOTE: Il est important de ne pas tester que les cas où tout se déroule comme prévu. Il faut aussi prendre en compte le cas d’un mauvais input de l'utilisateur ou d’une erreur dans une autre partie de l'application.

TODO

- ➔ Un test pour vérifier que la liste des artistes est initialisée correctement au démarrage.
- ➔ Un ou plusieurs tests pour vérifier l’affichage correct d’un artiste dans le html.
- ➔ Un ou plusieurs tests pour vérifier la suppression d’un artiste.
- ➔ Un ou plusieurs tests pour vérifier l’ajout d’un artiste.

RESSOURCES

- https://angular.dev/guide/testing

1.6 — Appel à une API

3h — Présentiel

Une bonne pratique essentielle en Angular est la séparation de certaines opérations de l'application dans des fichiers « **service** ». Les services peuvent par exemple être utilisés pour y mettre de la logique métier pour un ou plusieurs composants, pour partager de la donnée entre plusieurs composants qui ne sont pas liés, ou encore pour accéder à de la donnée extérieur à l'application.

Nous allons pouvoir créer un **service** pour requêter une api « **artists-api** » qui gérera pour nous le stockage et la mise à jour de la liste des artistes. Il est d'usage de nommer ce type de services « **webservice** ».

Pour cette partie, l'api « **artists-api** » est fournie et accessible à l'url suivante : https://artists-api-ndhd.onrender.com/ . Vous avez accès à un fichier "artists api documentation" dans le drive avec toutes les informations permettant de consommer cette api.

NOTES:

- La **liste des artistes** de l'API est unique et donc partagée pour tous.
- L'API est hébergée sur un site gratuit et se relance si personne ne l'utilise pendant un certain temps (il y a petit délai quand on la rappelle pour la première fois, et la liste reprend ses valeurs d'origine)

TODO

➔ Créez un **webservice** qui fait appel aux différentes routes de l'api fournie.

➔ Utilisez ce **webservice** pour récupérer et mettre à jour votre liste des artistes.

➔ Pensez à mettre à jour vos **tests unitaires** si besoin.

RESSOURCES

- Documentation de l'API dans le drive
- https://angular.dev/guide/di
- https://angular.dev/guide/http

- https://reqbin.com/req/adf8b77i/authorization-bearer-header

---

Livrables

Voici les livrables attendus à la fin de ce kit :

➔ Un site Angular affichant une page d'accueil et une liste d'artistes.

➔ Un mémo contenant les principales notions d'Angular.