Cours accéléré

Durée estimée : 2 heures

1. Introduction à Angular

- Expliquez brièvement ce qu'est Angular et son utilité dans le développement web.
  - Angular est un framework de développement d'applications web basé sur TypeScript, qui permet de créer des applications web dynamiques et interactives.
  - Mettez en évidence les avantages d'Angular, tels que la facilité de création de composants réutilisables, la gestion de l'état de l'application, le routage, etc.

2. Prérequis

- Présentez brièvement Node.js et expliquez son rôle dans le développement d'applications Angular.
  - Node.js est une plateforme permettant d'exécuter du code JavaScript côté serveur. Il est utilisé dans le développement d'applications Angular pour gérer les dépendances, exécuter des scripts de construction, etc.
- Montrez comment installer Node.js sur le système de la personne.
  - Rendez-vous sur le site officiel de Node.js (https://nodejs.org) et téléchargez la version recommandée pour votre système d'exploitation.
  - Suivez les instructions d'installation fournies par le programme d'installation de Node.js.
- Expliquez les bases de npm (Node Package Manager) et comment l'utiliser pour installer les dépendances.
  - Expliquez que npm est le gestionnaire de paquets par défaut de Node.js, utilisé pour installer des bibliothèques tierces et gérer les dépendances d'un projet.
  - Montrez comment ouvrir une invite de commande ou un terminal, puis comment utiliser la commande `npm install` pour installer des dépendances dans un projet.

3. Environnement de développement

- Expliquez les options disponibles pour configurer un environnement de développement Angular, notamment l'utilisation d'Angular CLI (Command Line Interface).
  - Angular CLI est un outil en ligne de commande qui facilite la création et la gestion de projets Angular.
- Montrez comment installer Angular CLI via npm.
  - Ouvrez une invite de commande ou un terminal et exécutez la commande suivante : `npm install -g @angular/cli`.
- Créez un nouveau projet Angular en utilisant la commande `ng new`.

- Utilisez la commande `ng new nom-du-projet` pour générer un nouveau projet Angular.
- Répondez aux questions posées par Angular CLI, telles que le style CSS à utiliser et si vous souhaitez configurer le routage.

4. Structure d'un projet Angular

La structure d'un projet Angular généré par Angular CLI est organisée de manière logique et suit des conventions établies. Explorons les différents dossiers et fichiers importants :

- **node_modules** : Ce dossier contient toutes les dépendances du projet installées via npm. Il est généré automatiquement lorsque vous exécutez la commande `npm install`.
- **src** : Ce dossier contient le code source de l'application Angular.
  - **app** : Ce sous-dossier contient le code spécifique à votre application. Il contient notamment les composants, les services, les modèles et les styles.
    - **app.component.ts** : Ce fichier contient la classe TypeScript du composant racine de votre application. Il est responsable du comportement du composant.
    - **app.component.html** : Ce fichier contient le template HTML du composant racine. Il détermine la structure et la mise en page de votre application.
    - **app.module.ts** : Ce fichier définit le module racine de votre application, qui est un conteneur pour tous les composants, services, directives et autres fonctionnalités utilisées dans votre application.
  - **assets** : Ce dossier est utilisé pour stocker des ressources statiques telles que des images, des fichiers de configuration, etc.
  - **environments** : Ce dossier contient des fichiers de configuration pour différents environnements de déploiement (par exemple, développement, production). Ces fichiers peuvent contenir des variables spécifiques à chaque environnement, telles que les URL des API, les clés d'accès, etc.
  - **styles.css** : Ce fichier contient les styles globaux de votre application. Vous pouvez y définir des styles qui s'appliqueront à tous les composants.
- **angular.json** : Ce fichier de configuration contient des informations sur le projet Angular, telles que les chemins des fichiers sources, les options de compilation, les dépendances, etc. Vous pouvez le modifier si nécessaire.
- **tsconfig.json** : Ce fichier contient la configuration du compilateur TypeScript pour votre projet Angular.
- **package.json** : Ce fichier définit les métadonnées du projet et les dépendances du package utilisées dans votre application. Vous pouvez y ajouter des scripts personnalisés, des dépendances supplémentaires, etc.

5. Composants et modèles

Dans Angular, les composants jouent un rôle central dans la construction de l'interface utilisateur. Ils encapsulent la logique et le rendu d'une partie de l'application. Les composants sont composés d'une classe TypeScript et d'un template HTML qui détermine la structure et l'apparence du composant. Voici comment créer et utiliser des composants dans Angular :

1. Créez un nouveau composant en utilisant la commande `ng generate component` :

- Ouvrez une invite de commande ou un terminal.
- Exécutez la commande suivante : `ng generate component nom-du-composant`.
- Cela générera automatiquement les fichiers nécessaires pour le nouveau composant, tels que la classe TypeScript, le template HTML et les styles CSS.

2. Utilisez le composant dans un autre composant :
   - Dans le composant parent où vous souhaitez utiliser le nouveau composant, importez la classe du composant en utilisant `import { NomDuComposantComponent } from './chemin-vers-le-composant'`.
   - Ajoutez le composant à la liste des déclarations dans le module qui contient le composant parent. Cela permet à Angular de reconnaître et de rendre le composant disponible.
   - Dans le template HTML du composant parent, utilisez la balise du composant en utilisant `<nom-du-composant></nom-du-composant>`.
3. Liez des données du composant au modèle et affichez-les dans le template :
   - Dans la classe du composant, définissez des propriétés qui représentent les données que vous souhaitez afficher.
   - Dans le template HTML, utilisez la syntaxe double accolade `{{ }}` pour lier les propriétés du composant au modèle et les afficher. Par exemple, `<h1>{{ titre }}</h1>`.
   - Vous pouvez également utiliser des directives structurales comme `*ngFor` et `*ngIf` pour afficher dynamiquement des éléments basés sur des listes ou des conditions.
4. Interagissez avec les événements et les actions utilisateur :
   - Dans le template HTML du composant, utilisez les événements intégrés tels que `(click)`, `(input)`, `(submit)`, etc., pour réagir aux actions de l'utilisateur.
   - Dans la classe du composant, définissez des méthodes pour gérer les événements et effectuer des actions en réponse à ceux-ci.
   - Utilisez la liaison de données bidirectionnelle avec `[(ngModel)]` pour lier des éléments de formulaire à des propriétés du composant et mettre à jour les valeurs lorsque l'utilisateur interagit avec le formulaire.

6. Services

Dans Angular, les services sont utilisés pour encapsuler la logique métier et partager des données entre les composants. Les services sont des classes TypeScript qui fournissent des fonctionnalités spécifiques et peuvent être injectés dans les composants ou d'autres services. Voici comment créer et utiliser des services dans Angular :

1. Créez un nouveau service en utilisant la commande `ng generate service` :
   - Ouvrez une invite de commande ou un terminal.
   - Exécutez la commande suivante : `ng generate service nom-du-service`.
   - Cela générera automatiquement les fichiers nécessaires pour le nouveau service, tels que la classe TypeScript.
2. Injectez le service dans un composant :

- Dans le composant où vous souhaitez utiliser le service, importez la classe du service en utilisant `import { NomDuService } from './chemin-vers-le-service'`.
- Ajoutez le service à la liste des fournisseurs dans le module qui contient le composant. Cela permet à Angular de reconnaître et d'injecter le service lorsque le composant en a besoin.
- Dans le constructeur du composant, ajoutez une dépendance du service en utilisant `private nomDuService: NomDuService`.

3. Utilisez le service pour effectuer des opérations :
   - Dans la classe du composant, appelez les méthodes fournies par le service pour effectuer des opérations métier ou récupérer des données.
   - Vous pouvez également utiliser les propriétés du service pour accéder aux données partagées entre les composants.
4. Optionnel : Utilisez l'injection de dépendances pour configurer le service :
   - Vous pouvez injecter d'autres services ou dépendances dans votre service en utilisant l'injection de dépendances.
   - Utilisez le décorateur `@Injectable()` sur la classe du service pour indiquer à Angular que le service peut être injecté avec des dépendances.

En utilisant des services, vous pouvez séparer la logique métier de vos composants et favoriser la réutilisabilité du code. Les services sont un excellent moyen de centraliser la logique de données et de fournir des fonctionnalités cohérentes à travers votre application Angular.

7. Routage

Le routage est un élément clé d'une application Angular, permettant de naviguer entre différentes vues (ou pages) de l'application. Angular fournit un système de routage intégré qui vous permet de configurer et de gérer facilement les routes. Voici comment configurer le routage dans votre application Angular :

1. Créez un module de routage :
   - Dans le dossier `app`, créez un nouveau fichier `nom-du-module-routing.module.ts` pour votre module de routage.
   - Importez les modules nécessaires depuis Angular et déclarez un nouveau module de routage en utilisant `@NgModule`.
   - Configurez les routes dans le tableau `routes` en définissant chaque route avec un chemin et un composant associé.
2. Importez le module de routage dans le module principal :
   - Dans le fichier `app.module.ts`, importez le module de routage en utilisant `import { NomDuModuleRoutingModule } from './chemin-vers-le-module-routing.module'`.
   - Ajoutez le module de routage à la liste des imports dans le décorateur `@NgModule` du module principal.
3. Configurez les liens de navigation dans les templates des composants :
   - Utilisez la directive `routerLink` pour créer des liens de navigation dans les templates des composants. Par exemple, `<a routerLink="/chemin-de-la-route">Lien</a>`.
   - Vous pouvez également utiliser des liens relatifs en utilisant `[routerLink]` avec un tableau pour spécifier les segments de l'URL. Par exemple, `<a`

`[routerLink]="['/chemin-parent', id, 'chemin-enfant']">Lien</a>`.

4. Affichez les composants de route dans le template principal :
   - Dans le template principal (généralement `app.component.html`), utilisez la directive `router-outlet` pour spécifier où afficher les composants correspondant aux routes.
   - Lorsque vous naviguez vers une route, le composant associé sera automatiquement chargé et affiché dans l'emplacement spécifié par `router-outlet`.
5. Utilisez la navigation programmée :
   - Vous pouvez également effectuer une navigation programmée en utilisant le service `Router` fourni par Angular.
   - Importez `Router` dans votre composant et injectez-le dans le constructeur.
   - Utilisez les méthodes `navigate` ou `navigateByUrl` du service `Router` pour effectuer des navigations programmées vers des routes spécifiques.

8. Gestion de l'état avec NgRx

NgRx est une bibliothèque de gestion d'état inspirée par Redux, conçue spécifiquement pour les applications Angular. Elle facilite la gestion de l'état global de l'application en utilisant un flux de données unidirectionnel. Voici comment configurer NgRx dans votre projet Angular et implémenter un exemple simple :

1. Installez les dépendances nécessaires via npm :
   - Ouvrez une invite de commande ou un terminal à la racine de votre projet Angular.
   - Exécutez la commande suivante pour installer les dépendances NgRx : `npm install @ngrx/store @ngrx/effects @ngrx/entity`.
2. Créez une action :
   - Dans le dossier `app`, créez un nouveau fichier `nom-de-l-action.actions.ts`.
   - Définissez une classe pour représenter votre action et utilisez le décorateur `@ngrx/store` pour définir le type d'action.
   - Par exemple, vous pouvez créer une action pour charger des données :

```
import { createAction } from '@ngrx/store';


export const loadDonnees = createAction('[Donnees] Charger les données');
```

3. Créez un reducer :
   - Dans le dossier `app`, créez un nouveau fichier `nom-du-reducer.reducer.ts`.
   - Définissez une fonction reducer qui prend en paramètre l'état actuel et une action, et retourne le nouvel état.
   - Par exemple, vous pouvez créer un reducer pour gérer les données chargées :

```
import { createReducer, on } from '@ngrx/store';
import { loadDonnees } from './nom-de-l-action.actions';


export interface State {
```

```
  donnees: any[];
}

export const initialState: State = {
  donnees: [],
};

export const reducer = createReducer(
  initialState,
  on(loadDonnees, (state) => {
    // Effectuez les opérations nécessaires pour charger les données depuis une
API ou une source de données
    return { ...state, donnees: /* Données chargées */ };
  })
);
```

4. Configurez le store :
   - Dans le dossier `app`, créez un nouveau fichier `nom-du-store.module.ts`.
   - Importez les modules nécessaires depuis NgRx et configurez le store en utilisant `StoreModule.forRoot({})`.
   - Enregistrez le reducer dans la configuration du store.

```
import { NgModule } from '@angular/core';
import { StoreModule } from '@ngrx/store';
import { reducer } from './nom-du-reducer.reducer';

@NgModule({
  imports: [StoreModule.forRoot({ donnees: reducer })],
})
export class NomDuStoreModule {}
```

5. Utilisez le store dans un composant :
   - Importez `Store` depuis `@ngrx/store` dans le composant où vous souhaitez utiliser le store.
   - Injectez `Store` dans le constructeur du composant.
   - Dispatchez une action en utilisant `store.dispatch(nom-de-l-action())` pour effectuer des changements d'état.

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
```

```
import { loadDonnees } from './nom-de-l-action.actions';

@Component({
  selector: 'app-mon-composant',
  template: `
    <button (click)="chargerDonnees()">Charger les données</button>
  `,
})
export class MonComposantComponent {
  constructor(private store: Store) {}

  chargerDonnees() {
    this.store.dispatch(loadDonnees());
  }
}
```

En utilisant NgRx, vous pouvez gérer efficacement l'état global de votre application Angular en séparant la logique de gestion de l'état du reste de votre code. NgRx facilite le suivi et la gestion des actions et des états de votre application, ce qui contribue à une architecture plus robuste et maintenable.

9. Intégration de ng-zorro

- Présentez ng-zorro, une bibliothèque de composants UI pour Angular.
  - ng-zorro fournit des composants prêts à l'emploi pour créer une interface utilisateur attrayante et réactive.
- Montrez comment installer ng-zorro via npm.
  - Utilisez la commande `npm install ng-zorro-antd` pour installer la bibliothèque ng-zorro.
- Intégrez quelques composants ng-zorro dans l'application pour illustrer son utilisation.
  - Utilisez les composants ng-zorro tels que les boutons, les formulaires, les modales, etc., pour améliorer l'apparence et la fonctionnalité de l'application.

10. Linting avec ESLint et Prettier

Le linting est une pratique importante pour maintenir un code propre, cohérent et exempt d'erreurs de style. ESLint est un outil populaire de linting JavaScript, tandis que Prettier est un outil de formatage de code. Combiner ESLint et Prettier vous permet d'automatiser la détection et la correction des erreurs de style. Voici comment configurer ESLint et Prettier dans votre projet Angular :

1. Installez les dépendances nécessaires via npm :

- Ouvrez une invite de commande ou un terminal à la racine de votre projet Angular.
- Exécutez la commande suivante pour installer les dépendances ESLint et Prettier : `npm install eslint prettier eslint-plugin-prettier eslint-config-prettier --save-dev`.

2. Créez un fichier de configuration pour ESLint :
   - À la racine de votre projet, créez un fichier `.eslintrc.json`.
   - Configurez ESLint en utilisant les règles et les plugins souhaités. Vous pouvez commencer avec une configuration de base ou utiliser une configuration spécifique à Angular.
   - Voici un exemple de configuration de base avec le plugin Prettier :

```
{
  "extends": ["eslint:recommended", "plugin:prettier/recommended"],
  "rules": {
    // Règles personnalisées ou supplémentaires si nécessaire
  }
}
```

3. Créez un fichier de configuration pour Prettier :
   - À la racine de votre projet, créez un fichier `.prettierrc` ou `.prettierrc.json`.
   - Configurez Prettier en utilisant les options de formatage souhaitées. Vous pouvez consulter la documentation de Prettier pour connaître toutes les options disponibles.
   - Voici un exemple de configuration de base :

```
{
  "semi": true,
  "singleQuote": true,
  "trailingComma": "all"
}
```

4. Configurer ESLint avec Prettier :
   - Ouvrez le fichier `.eslintrc.json` et ajoutez les règles spécifiques à Prettier pour assurer une intégration harmonieuse.
   - Ajoutez `"plugin:prettier/recommended"` dans la liste des règles étendues (extends).
5. Exécutez le linting et le formatage :
   - Ajoutez des scripts dans votre fichier `package.json` pour exécuter ESLint et Prettier.
   - Par exemple, vous pouvez ajouter les scripts suivants :

```
"scripts": {
  "lint": "eslint .",
  "lint:fix": "eslint . --fix",
  "format": "prettier --write ."
```

```
}
```

6. Exécutez les commandes de linting et de formatage :
   - Pour exécuter ESLint et afficher les erreurs et les avertissements : `npm run lint`.
   - Pour exécuter ESLint et corriger automatiquement les erreurs : `npm run lint:fix`.
   - Pour exécuter Prettier et formater automatiquement votre code : `npm run format`.

11. Conclusion et ressources supplémentaires

- Récapituler les points clés couverts dans le tutoriel.
- Fournir des ressources supplémentaires pour continuer à apprendre Angular par sois-même, notamment des tutoriels, des blogs et la documentation officielle d'Angular.

Revision #2
Created 2023-05-24 07:52:23 UTC by Noé Larrieu-Lacoste
Updated 2023-05-24 08:04:44 UTC by Noé Larrieu-Lacoste