# Dossier principal

**Dossiers :**

1. **.angular/** - Ce dossier contient des fichiers et des configurations spécifiques à Angular. Il est généralement généré automatiquement par Angular CLI lors de la création d'un projet Angular.
2. **.git/** - Ce dossier est utilisé pour stocker les données de contrôle de version Git, y compris l'historique des commits, les branches, etc.
3. **.vscode/** - Ce dossier peut contenir des configurations spécifiques à Visual Studio Code, telles que des fichiers de configuration pour l'éditeur.
4. **node\_modules/** - Ce dossier contient toutes les dépendances externes (bibliothèques JavaScript, packages Node.js) nécessaires à votre projet. Il est généralement généré par npm ou yarn.
5. **src/** - Ce dossier est probablement le cœur de votre projet Angular. Il contient le code source de votre application, y compris les composants, les services, les modules, les fichiers HTML, CSS, etc.

**Fichiers :**

1. **.gitignore** - Ce fichier contient la liste des fichiers et dossiers à exclure de la gestion de version Git. Il est utilisé pour ignorer les fichiers générés, les fichiers de build, les fichiers temporaires, etc.
2. **angular.json** - Ce fichier de configuration est utilisé par Angular CLI pour définir la configuration de votre projet Angular, y compris les paramètres de build, les alias de chemins, etc.
3. **package.json** - Ce fichier est utilisé pour définir les dépendances de votre projet, les scripts de build, les commandes npm, et d'autres informations liées au projet.
4. **package-lock.json** - Ce fichier est généré par npm et enregistre la version précise de chaque dépendance pour garantir la reproductibilité des installations.
5. **README.md** - Ce fichier contient généralement des informations sur le projet, des instructions pour l'installation et l'utilisation, et d'autres informations utiles pour les contributeurs ou les utilisateurs du projet.
6. **tsconfig.app.json** - Ce fichier de configuration est spécifique à Angular et définit les options de compilation TypeScript pour l'application Angular.
7. **tsconfig.json** - Ce fichier de configuration TypeScript définit les options globales pour le projet TypeScript, y compris la configuration du compilateur.

Si vous souhaitez explorer davantage un dossier ou un fichier spécifique, ou si vous avez des questions sur une partie spécifique de votre projet, n'hésitez pas à me le faire savoir, et nous pouvons approfondir davantage.

## Contenu de src

**Dossier "app/" :** Ce dossier "app/" est probablement l'endroit principal où vous avez développé votre application Angular. Il contient la logique de votre application, y compris les composants, les modules, les services, les vues, etc. Si vous le souhaitez, nous pouvons explorer plus en détail le contenu du dossier "app/" en examinant les fichiers et dossiers à l'intérieur.

**Dossier "assets/" :** Le dossier "assets/" est généralement utilisé pour stocker des fichiers statiques tels que des images, des fichiers CSS, des polices, etc. Il s'agit de ressources statiques que votre application peut utiliser. Si vous avez des ressources statiques dans ce dossier que vous souhaitez explorer ou discuter, veuillez les mentionner.

**Fichier "favicon.ico" :** Ce fichier est l'icône de favicon de votre application. Il s'affiche généralement dans l'onglet du navigateur lorsque l'application est ouverte. Si vous avez des questions ou des besoins particuliers concernant cet icône, n'hésitez pas à le mentionner.

**Fichier "index.html" :** Le fichier "index.html" est la page HTML principale de votre application Angular. C'est le point d'entrée de votre application et il contient généralement la balise **<app-root>** qui représente le point de montage de l'application Angular. Si vous avez des questions sur le contenu de ce fichier ou des éléments spécifiques à discuter, veuillez les mentionner.

**Fichier "main.ts" :** Le fichier "main.ts" est un point d'entrée TypeScript de votre application. Il est responsable de l'amorçage de l'application Angular. Si vous avez des questions sur le contenu de ce fichier ou des aspects particuliers à explorer, n'hésitez pas à le signaler.

**Fichier "styles.css" :** Le fichier "styles.css" contient les styles globaux de votre application. C'est l'endroit où vous pouvez définir des styles CSS globaux qui s'appliquent à toute l'application. Si vous souhaitez explorer davantage les styles ou discuter de certains aspects spécifiques de ce fichier, veuillez le faire savoir.

### Contenu de app/

**Dossiers :**

#### login/ -

Ce dossier pourrait contenir des composants, des fichiers de modèle (HTML) et des fichiers CSS spécifiques à la fonctionnalité de connexion de votre application. Si vous le souhaitez, nous pouvons explorer davantage le contenu de ce dossier pour comprendre comment la fonctionnalité de connexion est mise en œuvre.

##### Fichier "login.component.html" :

Le fichier "login.component.html" est le modèle (HTML) associé au composant de la page de connexion de votre application. Il contient généralement la structure de la page de connexion, y compris les formulaires, les champs de saisie, les boutons, etc. Il est responsable de l'affichage de la page de connexion pour les utilisateurs de votre application.

Si vous le souhaitez, vous pouvez partager le contenu de ce fichier ici, et je pourrai examiner son code pour voir comment la page de connexion est conçue et mise en forme.

###### Voici un résumé de son contenu :

Ce fichier HTML est la vue associée au composant "LoginComponent" et il est utilisé pour afficher la page de connexion de votre application Angular.

1. La page de connexion est enveloppée dans une structure HTML de base qui inclut une balise **<div>** avec la classe "row" pour la mise en page Bootstrap.
2. À l'intérieur de cette structure, il y a une **<div>** avec la classe "card", qui est utilisée pour afficher le contenu de la page de connexion dans une carte.
3. Le titre "Page de connexion" est affiché en utilisant la balise **<span class="card-title">**. Il s'agit du titre de la carte qui décrit la page.
4. Juste en dessous du titre, il y a un élément **<p>** qui affiche le contenu du message en utilisant la syntaxe de liaison de données Angular **{{ message }}**. Le contenu de ce message est dynamique et dépend de l'état de connexion de l'utilisateur.
5. Ensuite, il y a un formulaire **<form>** avec une référence locale **#loginForm**. Ce formulaire est utilisé pour collecter les informations de connexion de l'utilisateur.
6. À l'intérieur du formulaire, il y a deux champs de saisie de texte : "Name" et "Password", avec des balises **<input>**. Ces champs sont liés à des propriétés du composant "LoginComponent" à l'aide de **[(ngModel)]**. Cela signifie que les valeurs saisies par l'utilisateur sont liées aux propriétés **name** et **password** du composant.
7. En dessous des champs de saisie, il y a une section "card-action" qui affiche les boutons "Se connecter" et "Se déconnecter" en fonction de l'état de connexion de l'utilisateur. Les boutons sont liés à des méthodes du composant "LoginComponent" (**login()** et **logout()**).

En résumé, le fichier "login.component.html" définit la structure et la présentation visuelle de la page de connexion de votre application. Il utilise des directives Angular pour lier les éléments de la vue aux propriétés et méthodes du composant, ce qui permet une interaction fluide avec l'utilisateur.

1. **page-not-found/** - Ce dossier pourrait contenir des composants, des fichiers de modèle (HTML) et des fichiers CSS pour gérer les pages introuvables ou les erreurs 404 dans votre application. Nous pouvons explorer davantage ce dossier si nécessaire.
2. **sesamien/** - Ce dossier pourrait être lié à une fonctionnalité spécifique de votre application. Si vous avez des questions ou si vous souhaitez explorer davantage ce dossier, veuillez fournir plus de détails sur son contenu ou son objectif spécifique.

**Fichiers :**

1. **app.component.html** - Ce fichier est associé au composant principal de votre application. Il contient généralement le modèle (HTML) de la mise en page globale de votre application. Si vous avez des questions spécifiques sur le contenu de ce fichier, n'hésitez pas à le mentionner.
2. **app.component.ts** - Ce fichier est associé au composant principal de votre application. Il contient généralement la logique JavaScript/TypeScript pour le composant principal. Si vous avez des questions spécifiques sur la logique de ce composant, veuillez les mentionner.
3. **app.module.ts** - Ce fichier est le module principal de votre application Angular. Il déclare les composants, configure les routes, et gère les dépendances de votre application. Si vous avez des questions sur la configuration du module principal, n'hésitez pas à le signaler.
4. **app-routing.module.ts** - Ce fichier est le module de routage de votre application Angular. Il configure les routes et les règles de navigation de votre application. Si vous avez besoin de détails spécifiques sur la configuration des routes, veuillez le mentionner.
5. **auth.guard.ts** - Ce fichier semble être un garde (guard) d'authentification. Les gardes sont utilisés pour protéger les routes et restreindre l'accès en fonction de l'authentification. Si vous avez des questions sur la mise en œuvre de la sécurité et de l'authentification, veuillez fournir plus de détails.
6. **auth.service.ts** - Ce fichier semble être lié à la gestion de l'authentification. Il pourrait contenir des fonctions pour l'inscription, la connexion, la déconnexion, etc. Si vous avez besoin d'informations spécifiques sur la logique de gestion de l'authentification, veuillez le signaler.
7. **in-memory-data.service.ts** - Ce fichier semble être un service utilisé pour simuler un backend ou une source de données en mémoire. Si c'est le cas, il pourrait être utilisé pour le développement et les tests locaux. Si vous avez des questions spécifiques sur son utilisation, veuillez les mentionner.

##### Fichier "login.component.ts" :

Le fichier "login.component.ts" est le fichier TypeScript associé au composant de la page de connexion. Il contient la logique de ce composant, y compris la gestion des données du formulaire, la validation, la soumission du formulaire, etc. Il interagit généralement avec le service d'authentification (peut-être "auth.service.ts") pour gérer l'authentification des utilisateurs.

###### Interpretation du Fichier "login.component.ts" Code :

1. Le composant "LoginComponent" est annoté avec **@Component** et déclaré en tant que composant Angular. Il a un sélecteur **app-login** qui peut être utilisé pour l'inclure dans les modèles d'autres composants.
2. Dans le constructeur du composant, il reçoit deux services en injection de dépendance : **AuthService** et **Router**. Ces services seront utilisés pour gérer l'authentification de l'utilisateur et la navigation de l'application.
3. Le composant a des propriétés telles que **message**, **name**, **password**, et **auth**. **message** est utilisé pour afficher des messages à l'utilisateur, tandis que **name** et **password** stockent les informations saisies par l'utilisateur.
4. La méthode **ngOnInit** est utilisée pour initialiser la propriété **auth** en injectant le service **AuthService**.
5. La méthode **setMessage** est appelée pour définir le message en fonction de l'état de connexion de l'utilisateur. Si l'utilisateur est connecté, le message indique "Vous êtes connecté". Sinon, il indique "Identifiant ou mot de passe incorrect."
6. La méthode **login** est appelée lorsque l'utilisateur tente de se connecter. Elle appelle la méthode **login** du service **AuthService**, qui renvoie un observable représentant l'état de connexion. Si l'utilisateur est connecté avec succès, il est redirigé vers la page '/sesamiens'. Sinon, il reste sur la page '/login'.
7. La méthode **logout** est appelée pour déconnecter l'utilisateur en appelant la méthode **logout** du service **AuthService**. Elle réinitialise également le message pour indiquer que l'utilisateur est déconnecté.

Ce composant semble être responsable de la gestion de la page de connexion de votre application Angular. Il gère les interactions utilisateur, l'authentification, et les redirections en fonction de l'état de connexion.

#### page-not-found/

##### Fichier " page-not-found.component.ts" :

1. Le composant "PageNotFoundComponent" est annoté avec **@Component** et déclaré en tant que composant Angular. Il a un sélecteur **app-page-not-found**.
2. Le modèle de ce composant est défini directement dans le décorateur **@Component** sous forme de modèle en ligne (inline template). Le modèle est composé de plusieurs éléments HTML, notamment une image, un titre et un lien de retour.
3. Dans le modèle, il y a une image qui est chargée à partir de l'URL spécifiée. Dans cet exemple, une image Pokémon est utilisée pour illustrer que la page n'existe pas.
4. En dessous de l'image, un titre "Hey, cette page n'existe pas !" est affiché pour informer l'utilisateur de l'erreur.
5. Il y a également un lien "Retourner à l'accueil" qui utilise la directive **routerLink** pour rediriger l'utilisateur vers la page "/sesamiens" lorsque le lien est cliqué.

Ce composant "PageNotFoundComponent" est utilisé pour améliorer l'expérience utilisateur en affichant une page d'erreur conviviale lorsque l'utilisateur accède à une URL incorrecte. Il est généralement utilisé avec un gestionnaire de routes pour capturer les URL non valides et rediriger l'utilisateur vers cette page d'erreur.

#### sesamin/

**Sous-dossiers :**

##### add-sesamien/

Ce sous-dossier pourrait contenir des composants et des fichiers spécifiques à l'ajout d'un "sesamien" (peut-être un étudiant). Nous pouvons explorer davantage ce dossier si nécessaire.

###### add-sesamien.component.ts

* + Le fichier "add-sesamien.component.ts" définit le composant "AddSesamienComponent", qui est responsable de l'affichage du formulaire pour ajouter un nouveau "Sesamien" dans votre application Angular.

Voici un résumé de son contenu :

* Le composant est annoté avec **@Component**, ce qui le rend un composant Angular. Il a un sélecteur **"app-add-sesamien"**.
* Dans le template du composant, il y a un titre "Ajouter un Sesamien" affiché en tant qu'en-tête de la page.
* Le composant utilise le sous-composant "SesamienFormComponent" pour afficher le formulaire d'ajout. Il passe l'objet **sesamien** au sous-composant en utilisant la liaison de données **[sesamien]="sesamien"**.
* Dans la méthode **ngOnInit()**, le composant initialise la propriété **sesamien** en créant une nouvelle instance de la classe "Sesamien". Cela permet d'avoir un objet vide prêt à être rempli dans le formulaire.

En résumé, le rôle principal de ce composant est d'afficher le formulaire d'ajout de "Sesamien" en utilisant le sous-composant "SesamienFormComponent". Il initialise également un objet "Sesamien" vide pour que l'utilisateur puisse saisir les détails du nouveau "Sesamien". Ce composant est probablement utilisé dans le contexte de l'ajout d'étudiants à votre application.

##### detail-sesamien/ -

Ce sous-dossier pourrait être lié à l'affichage détaillé d'un "sesamien". Nous pouvons explorer son contenu pour voir comment les détails d'un "sesamien" sont affichés. Le dossier "detail-sesamien/" contient deux fichiers : "detail-sesamien.component.html" et "detail-sesamien.component.ts"

###### detail-sesamien.component.html

* + Ce fichier contient le modèle (template) pour le composant "DetailSesamienComponent". Il est responsable de l'affichage des détails d'un "Sesamien" spécifique.

Voici un résumé de son contenu :

* Utilisation de l'expression **\*ngIf="sesamien"** : Cette directive structurelle conditionnelle permet d'afficher le contenu du modèle uniquement si la variable **sesamien** est définie.
* Le contenu est disposé en utilisant la grille CSS Bootstrap, avec des classes telles que "row", "col", et "card" pour créer une mise en page réactive.
* Le nom du "Sesamien" est affiché dans un titre de niveau 2.
* Une carte horizontale est utilisée pour afficher les informations du "Sesamien", y compris une image, le nom, les points de vie, les dégâts, la mention, et la date de création.
* La mention du "Sesamien" est affichée avec une couleur spécifique en fonction de la pipe "sesamienMentionColor" qui gère la mise en forme en fonction de la mention.
* Des actions sont affichées en bas de la carte pour retourner à la liste des "Sesamiens", éditer le "Sesamien", ou le supprimer.
* Si la variable **sesamien** n'est pas définie, un composant "Loader" est affiché à la place.

###### detail-sesamien.component.ts

Voici un résumé de son contenu :

* Le composant est annoté avec **@Component** et a un sélecteur **"app-detail-sesamien"**.
* Dans le constructeur du composant, il y a une injection des dépendances (**route**, **router**, **sesamienService**). Ces dépendances sont nécessaires pour accéder aux données de l'URL, effectuer des navigations, et obtenir des informations sur le "Sesamien" depuis le service.
* La méthode **ngOnInit()** est appelée lors de l'initialisation du composant. Elle récupère l'ID du "Sesamien" depuis les paramètres de l'URL à l'aide de **this.route.snapshot.paramMap.get('id')**, puis appelle le service pour obtenir les détails du "Sesamien" en utilisant **this.sesamienService.getSesamienById(+sesamienId)**.
* Le composant a également des méthodes pour supprimer un "Sesamien" (**deleteSesamien(sesamien: Sesamien)**) et pour naviguer vers la liste des "Sesamiens" (**goToSesamienList()**) ou la page d'édition d'un "Sesamien" (**goToEditSesamien(sesamien: Sesamien)**).

En résumé, ce composant est responsable de l'affichage des détails d'un "Sesamien" spécifique en utilisant le modèle HTML correspondant. Il récupère les données du "Sesamien" à afficher en fonction de l'ID passé dans l'URL et offre des fonctionnalités pour la navigation et la suppression.

##### edit-sesamien/

Ce sous-dossier pourrait être lié à la modification d'un "sesamien". Nous pouvons explorer davantage ce dossier pour comprendre comment la modification des données d'un "sesamien" est gérée.

###### edit-sesamien.component.ts

Responsable de l'édition des informations d'un "Sesamien".

Voici un résumé de son contenu :

* Le composant est annoté avec **@Component** et a un sélecteur **"app-edit-sesamien"**.
* Dans le constructeur du composant, il y a une injection des dépendances (**route** et **sesamienService**). Ces dépendances sont nécessaires pour accéder aux données de l'URL et au service qui fournit les informations sur le "Sesamien".
* La méthode **ngOnInit()** est appelée lors de l'initialisation du composant. Elle récupère l'ID du "Sesamien" depuis les paramètres de l'URL à l'aide de **this.route.snapshot.paramMap.get('id')**. Ensuite, elle appelle le service pour obtenir les détails du "Sesamien" en utilisant **this.sesamienService.getSesamienById(+sesamienId)**.
* Le modèle (template) du composant affiche le titre "Editer {{ sesamien?.name }}" avec le nom du "Sesamien" actuellement édité (s'il est défini).
* L'image du "Sesamien" est actuellement une balise **<img>** avec un attribut "src" vide. Vous devrez ajouter l'URL de l'image appropriée pour afficher l'image du "Sesamien" en édition.
* Le composant affiche également le formulaire de modification du "Sesamien" en utilisant **<app-sesamien-form>**. Le formulaire est affiché conditionnellement uniquement si la variable **sesamien** est définie, ce qui signifie que les détails du "Sesamien" ont été chargés avec succès.

En résumé, ce composant est responsable de l'édition des informations d'un "Sesamien". Il récupère les données du "Sesamien" à éditer en fonction de l'ID passé dans l'URL et affiche un formulaire de modification conditionnellement. Vous devrez ajouter l'URL de l'image du "Sesamien" pour qu'elle soit correctement affichée.

##### list-sesamien/

Ce sous-dossier pourrait contenir des composants et des fichiers associés à la liste des "sesamiens". Cela peut inclure la liste de tous les "sesamiens" enregistrés dans votre application.

###### list-sesamien.component.ts

Voici un résumé de son contenu :

* Vous avez une balise **div** avec la classe "container", qui est une classe de Bootstrap pour la mise en page.
* À l'intérieur de cette balise, vous avez une autre **div** avec la classe "row" pour afficher une grille.
* Vous avez également un composant **app-search-sesamien** qui semble être utilisé pour la recherche de "Sesamiens".
* Ensuite, vous utilisez une boucle **\*ngFor** pour parcourir la liste des "Sesamiens" (**sesamienList**) et afficher chaque "Sesamien" sous forme de carte (**card**) dans une disposition horizontale.
* Chaque carte "Sesamien" est cliquable grâce à **(click)="goToSesamien(sesamien)"**, ce qui signifie que lorsque vous cliquez sur une carte, vous serez redirigé vers la page de détails du "Sesamien".
* La carte contient des informations telles que le nom, la date de création et les mentions, qui sont extraites de l'objet "Sesamien".

###### list-sesamien.component.ts

Voici un résumé de son contenu :

* Dans le composant TypeScript associé, vous avez une variable **sesamienList** pour stocker la liste des "Sesamiens".
* Dans la méthode **ngOnInit**, vous appelez **getSesamienList** du service **SesamienService** pour récupérer la liste des "Sesamiens" à partir de votre API (ou de toute source de données) et vous la stockez dans **sesamienList**.
* Vous avez également une méthode **goToSesamien(sesamien)** qui est appelée lorsqu'un "Sesamien" est cliqué. Elle redirige l'utilisateur vers la page de détails du "Sesamien" en utilisant la route "/sesamien/:id", où ":id" est l'identifiant unique du "Sesamien".

Dans l'ensemble, ce code semble être la page de liste des "Sesamiens" dans votre application. Lorsqu'un utilisateur clique sur un "Sesamien" dans la liste, il est redirigé vers la page de détails du "Sesamien" correspondant.

##### loader/

Ce sous-dossier pourrait être lié à la gestion du chargement de données ou d'éléments dans votre application. Il peut contenir des composants ou des services liés à cette fonctionnalité.

###### loader-sesamien.component.ts

Ce composant est utilisé pour afficher une animation de chargement sous forme d'un spinner lorsque des données sont en cours de traitement. Le code HTML dans ce fichier définit la structure de l'animation du spinner.

Voici un résumé de son contenu :

Le code HTML du fichier crée un spinner avec des cercles en rotation pour indiquer visuellement que des données sont en cours de chargement. C'est une composante visuelle qui peut être utilisée pour améliorer l'expérience de l'utilisateur lorsqu'il attend le chargement de données asynchrones.

En résumé, ce fichier et son code sont dédiés à la création d'un composant de chargement visuel pour votre application Angular.

##### search-sesamien/

Ce sous-dossier pourrait être lié à la fonction de recherche des "sesamiens" dans votre application. Il peut contenir des composants ou des services pour la recherche.

###### search-sesamien.component.html

Ce fichier HTML définit la structure visuelle du composant "SearchSesamienComponent". Il s'agit d'un formulaire de recherche de Sesamiens avec un champ de texte où les utilisateurs peuvent saisir leur requête de recherche.

Voici un résumé de son contenu :

* Il utilise la directive **(keyup)** pour détecter les événements de frappe de clavier dans le champ de recherche, et chaque fois que l'utilisateur tape une touche, la méthode **search(searchBox.value)** est appelée.
* Dans la partie inférieure du formulaire, il y a une liste de résultats de recherche affichée à l'aide de la directive **\*ngFor**. Cette liste est générée à partir des résultats de recherche obtenus à partir du composant TypeScript correspondant.
* Chaque résultat de recherche est rendu sous forme de lien cliquable avec le nom du Sesamien. Lorsque l'utilisateur clique sur l'un de ces liens, la méthode **goToDetail(sesamien)** est appelée pour naviguer vers les détails du Sesamien sélectionné.

###### search-sesamien.component.ts

Ce fichier TypeScript contient la logique du composant "SearchSesamienComponent".

Il utilise Angular et RxJS pour gérer la recherche asynchrone des Sesamiens en fonction des termes de recherche saisis par l'utilisateur.

Voici un résumé de son contenu :

* Le composant déclare une variable **searchTerms**, qui est un objet **Subject<string>**. Cet objet est utilisé pour suivre les termes de recherche saisis par l'utilisateur.
* La méthode **ngOnInit** est appelée lors de l'initialisation du composant. À l'intérieur de cette méthode, le composant utilise des opérateurs RxJS pour surveiller les changements de texte dans le champ de recherche.
* Lorsque l'utilisateur saisit du texte, la méthode **search(term: string)** est appelée, ce qui émet le terme de recherche vers l'observable **searchTerms**.
* L'observable **searchTerms** est ensuite passé à travers une chaîne d'opérateurs RxJS (**debounceTime**, **distinctUntilChanged**, **switchMap**) pour effectuer la recherche asynchrone.
* Les résultats de la recherche sont stockés dans la variable **sesamiens$**, qui est un observable de tableau de Sesamiens.
* Enfin, la méthode **goToDetail(sesamien: Sesamien)** est utilisée pour naviguer vers la page de détails du Sesamien lorsque l'utilisateur clique sur un résultat de recherche.

##### sesamien-form/ -

Ce sous-dossier pourrait contenir des composants ou des fichiers liés à la création ou à la modification d'un "sesamien". Cela peut inclure les formulaires de saisie de données.

###### sesamien-form.component.ts

Ce fichier CSS est spécifique au composant "SesamienFormComponent" et définit les styles qui lui sont associés.

Voici un résumé de son contenu :

* Il commence par définir des styles pour les éléments de formulaire en utilisant des sélecteurs CSS spécifiques au composant, par exemple, **.ng-valid[required]** et **ng-invalid:not(form)**.
* La règle **.ng-valid[required]** définit le style des éléments de formulaire valides (comme les champs requis) en ajoutant une bordure verte à gauche (**border-left: 5px solid #42a948**).
* La règle **ng-invalid:not(form)** définit le style des éléments de formulaire non valides en ajoutant une bordure rouge à gauche (**border-left: 5px solid #A94442**).
* Ces styles permettent de donner un retour visuel immédiat à l'utilisateur sur la validité de ses saisies dans le formulaire.
* En utilisant ces règles, le fichier CSS personnalise également l'apparence des messages d'erreur qui s'affichent lorsque les champs du formulaire ne sont pas valides.
* Par exemple, il définit un style pour les messages d'erreur qui sont affichés en rouge avec un fond gris (**class="card-panel red accent-1"**).
* Ces styles contribuent à améliorer l'expérience utilisateur en rendant les erreurs de saisie plus visibles et compréhensibles.
* Le fichier CSS peut être utilisé en conjonction avec le fichier HTML du composant pour fournir une présentation visuellement attrayante et conviviale pour le formulaire de gestion des Sesamiens.

###### sesamien-form.component.ts

* Ce fichier HTML définit le modèle de vue pour le composant "SesamienFormComponent".
* Il s'agit d'un formulaire permettant de créer ou de modifier un objet "Sesamien".
* Le formulaire contient plusieurs champs, y compris un champ de saisie de texte pour le nom, un champ d'URL pour l'image, des champs numériques pour les points de vie (hp) et les dégâts (cp), ainsi que des cases à cocher pour sélectionner les mentions du Sesamien.

Voici un résumé de son contenu :

* Les champs du formulaire sont associés à des propriétés du composant via [(ngModel)] et portent des attributs HTML tels que "required", "pattern", et "name" pour la validation et la gestion des erreurs.
* Lorsqu'un champ est invalide, un message d'erreur approprié est affiché sous le champ, utilisant la classe "card-panel red accent-1" pour une présentation visuelle.
* Le formulaire contient également un bouton "Valider" qui est désactivé tant que le formulaire n'est pas valide (attribut [disabled]="!sesamienForm.form.valid").
* En bas du formulaire, s'il n'y a pas encore de Sesamien sélectionné, il affiche le composant "LoaderComponent" pour une indication visuelle du chargement.
* Le fichier HTML est bien structuré avec des balises HTML sémantiques pour faciliter la compréhension et la maintenance.

###### Sesamien-form.component.ts

Ce fichier TypeScript définit la classe du composant "SesamienFormComponent".

Voici un résumé de son contenu :

* Le composant reçoit un objet "Sesamien" en entrée via la directive **@Input()**.
* Il initialise des propriétés telles que "mentions" pour la liste des mentions possibles et "isAddForm" pour indiquer s'il s'agit d'un formulaire d'ajout ou de modification.
* Le composant implémente plusieurs méthodes pour gérer les mentions sélectionnées, vérifier leur validité et gérer la soumission du formulaire.
* Par exemple, la méthode "selectMention" est appelée lorsqu'une case à cocher de mention est modifiée, permettant d'ajouter ou de supprimer la mention correspondante de la liste des mentions du Sesamien.
* La méthode "isMentionsValid" vérifie la validité des mentions en fonction de certaines règles définies.
* La méthode "onSubmit" gère la soumission du formulaire, appelant soit "addSesamien" soit "updateSesamien" du service "SesamienService" en fonction du type de formulaire.
* Les dépendances vers le service "SesamienService" et le routeur sont injectées via le constructeur.

En résumé, ce composant définit la logique du formulaire de gestion des Sesamiens, notamment la gestion des mentions, la validation des champs et la soumission du formulaire. Il est associé au modèle de vue défini dans le fichier HTML pour créer une expérience utilisateur complète.

**Fichiers :**

##### border-card.directive.ts

- Ce fichier semble être une directive personnalisée appelée "border-card". Les directives personnalisées sont utilisées pour ajouter des comportements ou des styles spécifiques à certains éléments HTML. Nous pouvons explorer son contenu pour voir comment elle est implémentée.

###### Voici un résumé de son contenu :

* La directive est annotée avec **@Directive** et utilise le sélecteur **[appBorderCard]**. Cela signifie que la directive peut être appliquée à n'importe quel élément HTML en utilisant l'attribut **appBorderCard**.
* Dans le constructeur de la directive, l'élément DOM sur lequel la directive est appliquée est injecté en utilisant **ElementRef**.
* La directive utilise des variables privées pour stocker des valeurs telles que la couleur initiale, la couleur par défaut et la hauteur par défaut de l'élément.
* Il y a un input **borderColor** qui permet de définir la couleur de la bordure de l'élément. Si aucune couleur n'est spécifiée, la couleur par défaut sera utilisée.
* La directive réagit à deux événements : "mouseenter" (souris sur l'élément) et "mouseleave" (souris hors de l'élément). Lorsque la souris entre dans l'élément, la bordure de l'élément est modifiée pour utiliser la couleur spécifiée (ou la couleur par défaut), et la hauteur de l'élément est augmentée. Lorsque la souris quitte l'élément, la bordure et la hauteur sont réinitialisées aux valeurs par défaut.
* Les méthodes privées **setHeight** et **setBorder** sont utilisées pour appliquer les styles CSS à l'élément.

Cette directive personnalisée permet donc de modifier dynamiquement l'apparence d'un élément HTML lorsqu'un utilisateur passe la souris dessus

##### mock-sesamien-list.ts –

Ce fichier semble contenir des données fictives ou de test pour une liste de "sesamiens". Il peut être utilisé pour le développement et les tests. Nous pouvons explorer son contenu si vous le souhaitez.

###### Voici un résumé de son contenu :

* Le tableau "SESAMIENS" contient plusieurs objets, chacun représentant un "sesamien". Chaque objet a des propriétés telles que "id", "name", "hp" (points de vie), "cp" (capacité de combat), "picture" (image), "mentions" (mention), et "created" (date de création).
* Les données semblent être utilisées pour simuler des "sesamiens" ou des étudiants avec des caractéristiques différentes, telles que leurs noms, points de vie, images, mentions, etc.
* Ces données fictives peuvent être utilisées pour développer et tester votre application Angular avant d'intégrer une base de données réelle.

##### sesamien.module.ts –

Ce fichier est probablement le module principal du domaine "sesamien". Il peut contenir les déclarations, les imports, et les providers liés aux fonctionnalités de gestion des "sesamiens".

###### Voici un résumé de son contenu :

* Le module importe plusieurs modules Angular, y compris **NgModule**, **RouterModule**, **CommonModule**, et **FormsModule**. Ces modules sont utilisés pour la création de composants, la gestion des routes, et la manipulation de formulaires.
* Le module déclare plusieurs composants, directives, pipes, et services liés à la gestion des "sesamiens". Ces éléments incluent **ListSesamienComponent**, **DetailSesamienComponent**, **BorderCardDirective**, **SesamienMentionColorPipe**, **SesamienFormComponent**, **EditSesamienComponent**, **AddSesamienComponent**, **SearchSesamienComponent**, et **LoaderComponent**.
* Le module définit également un tableau de routes (**sesamienRoutes**) qui spécifie les chemins URL associés à chaque composant. Par exemple, le chemin "/sesamiens" est associé au composant **ListSesamienComponent**. L'utilisation de l'AuthGuard (**canActivate: [AuthGuard]**) indique que l'accès à ces routes est protégé et nécessite une authentification.
* Le module est décoré avec **@NgModule**, ce qui le rend utilisable comme module principal ou module enfant. Il configure les déclarations, les imports, et les providers nécessaires pour le fonctionnement de la gestion des "sesamiens".
* Le service **SesamienService** est fourni dans le tableau des providers, ce qui signifie qu'il peut être injecté dans les composants du module.

En résumé, ce module "SesamienModule" encapsule toutes les fonctionnalités liées à la gestion des "sesamiens" dans votre application Angular, y compris la définition des routes, des composants, des directives, et des pipes associés.

##### sesamien.service.ts –

Ce fichier semble être un service lié à la gestion des "sesamiens". Les services sont souvent utilisés pour interagir avec des données et effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer). Nous pouvons explorer son contenu pour voir quelles opérations il gère.

###### Voici un résumé de son contenu :

* Le service utilise le module **HttpClient** d'Angular pour effectuer des requêtes HTTP vers le serveur. Il est injecté dans le constructeur du service.
* Le service définit plusieurs méthodes pour interagir avec les données des "sesamiens" :
  + **getSesamienList()** : Récupère la liste des "sesamiens" depuis le serveur.
  + **getSesamienById(sesamienId: number)** : Récupère un "sesamien" spécifique par son ID depuis le serveur.
  + **searchSesamienList(term: string)** : Recherche des "sesamiens" en fonction d'un terme de recherche.
  + **updateSesamien(sesamien: Sesamien)** : Met à jour un "sesamien" sur le serveur.
  + **addSesamien(sesamien: Sesamien)** : Ajoute un nouveau "sesamien" sur le serveur.
  + **deleteSesamienByID(sesamienId: number)** : Supprime un "sesamien" par son ID depuis le serveur.
  + **getSesamienMentionList()** : Renvoie une liste de mentions possibles pour les "sesamiens".
* Le service utilise également les opérateurs **pipe**, **tap**, et **catchError** pour gérer les réponses HTTP et les erreurs.
* Les méthodes **log** et **handleError** sont utilisées pour afficher des messages de journalisation dans la console et gérer les erreurs de manière centralisée.
* Le service est décoré avec **@Injectable()**, ce qui permet de l'injecter ailleurs dans l'application.

Ce service joue un rôle essentiel dans la gestion des données des "sesamiens" et assure la communication avec le serveur pour effectuer les opérations CRUD (Create, Read, Update, Delete).

##### sesamien.ts –

Ce fichier pourrait contenir une classe ou une interface représentant un "sesamien" avec ses propriétés. Nous pouvons explorer sa structure si nécessaire.

###### Voici un résumé de son contenu :

* La classe "Sesamien" a plusieurs propriétés, notamment **id**, **name**, **hp** (points de vie), **cp** (capacité de combat), **picture** (image), **mentions** (mention), et **created** (date de création).
* Le constructeur de la classe permet de créer une instance de "Sesamien" avec des valeurs par défaut. Il prend en charge des paramètres tels que le nom, les points de vie, la capacité de combat, l'image, les mentions, et la date de création.
* Cette classe sert de modèle pour représenter les données des "sesamiens" dans l'application.

En résumé, le fichier "sesamien.ts" définit la structure de données pour les "sesamiens", tandis que le fichier "sesamien.service.ts" fournit les méthodes pour interagir avec ces données en utilisant des requêtes HTTP vers le serveur. Ces deux éléments sont essentiels pour la gestion des "sesamiens" dans votre application Angular.

##### sesamien-mention-color.pipe.ts –

Ce fichier semble être une pipe personnalisée appelée "sesamien-mention-color". Les pipes sont utilisées pour transformer les données dans les modèles. Nous pouvons explorer son contenu pour comprendre son rôle.

###### Voici un résumé de son contenu :

* La pipe est annotée avec **@Pipe** et porte le nom "sesamienMentionColor".
* La pipe implémente l'interface **PipeTransform**, ce qui signifie qu'elle doit fournir une méthode **transform**.
* La méthode **transform** prend en argument une mention (une chaîne de caractères) et retourne une classe CSS qui définit la couleur de la mention. La méthode utilise une instruction **switch** pour déterminer la couleur en fonction de la mention. Par exemple, si la mention est "S", la couleur "red lighten-1" est renvoyée.
* La classe CSS est préfixée avec "chip" pour indiquer qu'elle est destinée à être utilisée pour afficher des mentions de "sesamiens" dans l'interface utilisateur.

Cette pipe personnalisée "SesamienMentionColorPipe" est utile pour appliquer des styles de couleur aux mentions de "sesamiens" en fonction de leur valeur.

#### app-routing.module.ts

Ce fichier configure les routes de l'application Angular, ce qui signifie qu'il définit comment les différentes vues de l'application sont accessibles via l'URL. Voici une description détaillée :

##### Voici un résumé de son contenu :

1. Importations :
   * Le fichier commence par les importations nécessaires depuis le module '@angular/core' et '@angular/router'. Ces importations sont essentielles pour définir et configurer les routes de l'application.
2. Constante "routes" :
   * Une constante nommée "routes" est définie, qui est un tableau d'objets de type "Routes". Chaque objet de route spécifie le chemin d'URL correspondant à un composant Angular.
3. Redirection par défaut :
   * La première route déclare une redirection par défaut. Lorsque l'utilisateur accède à la racine de l'application ('/'), il est automatiquement redirigé vers le chemin 'login'. Cela signifie que la page de connexion sera la première page affichée lors de l'accès au site.
4. Route vers le composant "LoginComponent" :
   * La deuxième route spécifie que lorsque l'utilisateur accède à l'URL '/login', le composant "LoginComponent" doit être affiché. Cela signifie que lorsque l'utilisateur visite '/login', le formulaire de connexion sera affiché.
5. Route wildcard (toutes autres routes) :
   * La dernière route utilise un "wildcard" '\*\*' pour capturer toutes les autres URL qui n'ont pas été définies précédemment. Elle redirige vers le composant "PageNotFoundComponent". Ainsi, si un utilisateur entre une URL incorrecte, il sera redirigé vers une page d'erreur "PageNotFoundComponent".
6. NgModule "AppRoutingModule" :
   * Enfin, le module "AppRoutingModule" est défini. Il importe le module "RouterModule.forRoot(routes)" pour configurer les routes de l'application et l'exporte pour que d'autres parties de l'application puissent l'utiliser.

En résumé, ce fichier configure les routes de l'application Angular, définissant comment chaque URL correspond à un composant spécifique. Il gère également la redirection par défaut vers la page de connexion et la gestion des URL incorrectes en redirigeant vers une page d'erreur "PageNotFoundComponent".

#### app.component.html

Le fichier "app.component.html" représente le modèle de votre composant principal "AppComponent" dans une application Angular.

##### Voici un résumé de son contenu :

1. Balise **<nav>** :
   * Cette balise **<nav>** représente la barre de navigation de votre application. Elle sera généralement utilisée pour afficher des liens de navigation ou d'autres éléments de navigation.
2. **<div>** dans la balise **<nav>** :
   * À l'intérieur de la balise **<nav>**, il y a une balise **<div>** avec la classe CSS "nav-wrapper teal". Cette balise **<div>** est utilisée pour styliser la barre de navigation. "teal" est une classe CSS qui définit la couleur de fond de la barre de navigation en turquoise.
3. Lien du logo :
   * À l'intérieur de la balise **<div>**, il y a un lien **<a>** avec la classe CSS "brand-logo center". Ce lien représente le logo de votre application et est centré dans la barre de navigation. Le texte "Sesamex" à l'intérieur du lien est le nom de votre application.
4. **<router-outlet>** :
   * Enfin, en dehors de la balise **<nav>**, il y a une balise **<router-outlet>**. Cette balise est un emplacement réservé où les composants associés aux routes seront insérés dynamiquement. En d'autres termes, lorsque l'utilisateur navigue dans votre application en utilisant les routes définies, les vues correspondantes seront affichées à cet endroit.

En résumé, ce fichier HTML définit la structure de base de votre application Angular. Il comprend une barre de navigation avec un logo au centre, et il utilise **<router-outlet>** pour afficher dynamiquement le contenu des différentes vues de l'application en fonction des routes définies.

#### app.component.ts

Le fichier "app.component.ts" définit la classe de composant principale "AppComponent" pour votre application Angular

##### Voici un résumé de son contenu :

1. Importation de modules :
   * La première ligne importe le module **Component** du package **@angular/core**. Cela permet d'utiliser le décorateur **@Component** pour définir votre composant.
2. Définition du composant :
   * La classe **AppComponent** est définie en utilisant le décorateur **@Component**. Ce décorateur configure les métadonnées du composant.
   * Le sélecteur **selector** est défini comme "app-root". Cela signifie que ce composant sera associé à la balise **<app-root>** dans votre application HTML, généralement utilisée comme composant racine.
   * Le modèle du composant est défini en utilisant **templateUrl**, qui pointe vers le fichier "app.component.html". Cela indique quel fichier HTML doit être utilisé pour afficher le contenu du composant.
3. Définition de la classe :
   * La classe **AppComponent** est vide dans ce fichier, ce qui signifie qu'elle ne contient pas de méthodes ou de propriétés personnalisées. Cependant, elle hérite des fonctionnalités de base d'un composant Angular.

En résumé, ce fichier définit la classe **AppComponent** en utilisant le décorateur **@Component**. Ce composant est associé à la balise **<app-root>** dans le fichier HTML et utilise "app.component.html" comme modèle pour afficher le contenu. C'est le composant racine de votre application Angular.

#### app.module.ts

Le fichier "app.module.ts" est un élément clé de votre application Angular. Il définit le module principal de l'application et configure les dépendances, les composants, les services, les modules, et les autres éléments nécessaires pour son bon fonctionnement.

##### Voici un résumé de son contenu :

1. Importation des modules :
   * Les différentes lignes d'importation importent les modules nécessaires depuis Angular et d'autres sources. Par exemple, **NgModule** est importé depuis **@angular/core**, et d'autres modules comme **BrowserModule**, **HttpClientModule**, **HttpClientInMemoryWebApiModule**, etc., sont importés depuis leurs packages respectifs.
   * Ces modules sont essentiels pour l'application. Par exemple, **BrowserModule** est nécessaire pour le fonctionnement d'une application Web Angular, **HttpClientModule** permet de faire des requêtes HTTP, **FormsModule** est nécessaire pour la liaison de données, etc.
2. Définition du module principal :
   * Le décorateur **@NgModule** est utilisé pour définir le module principal de l'application, **AppModule**. Cela configure les métadonnées du module.
   * Les métadonnées incluent :
     + **declarations**: Cette propriété liste les composants, les directives et les pipes qui sont définis dans ce module. Dans ce cas, les composants **AppComponent**, **PageNotFoundComponent**, et **LoginComponent** sont répertoriés.
     + **imports**: Cette propriété liste les modules importés et utilisés dans ce module. Les modules tels que **BrowserModule**, **FormsModule**, **HttpClientModule**, **HttpClientInMemoryWebApiModule**, **SesamienModule**, et **AppRoutingModule** sont importés ici.
     + **providers**: Cette propriété permet de configurer les fournisseurs de services. Dans ce cas, elle est laissée vide (**[]**).
     + **bootstrap**: Cette propriété indique quel composant doit être utilisé comme point d'entrée de l'application. Ici, c'est **AppComponent**.
3. Utilisation des modules :
   * Les modules importés sont utilisés pour configurer le fonctionnement de l'application. Par exemple, **HttpClientModule** est nécessaire pour effectuer des requêtes HTTP, **FormsModule** permet la liaison de données, **SesamienModule** est probablement un module personnalisé de votre application, etc.
4. Configuration de **HttpClientInMemoryWebApiModule** :
   * **HttpClientInMemoryWebApiModule** est configuré avec **forRoot** pour simuler une API web en utilisant le service **InMemoryDataService**. Cela peut être utile pour le développement local lorsque vous n'avez pas encore une véritable API.

En résumé, ce fichier configure le module principal de votre application Angular, déclare les composants, importe les modules nécessaires, et configure les services. Il s'agit d'un élément central de votre application qui organise les différentes parties de votre code en un tout cohérent.

#### auth.guard.ts

Le fichier "auth.guard.ts" est un gardien de route (route guard) dans une application Angular. Il est utilisé pour contrôler l'accès à certaines routes en fonction de la condition spécifiée. Dans ce cas, il semble que le gardien "AuthGuard" vérifie si l'utilisateur est authentifié avant de permettre l'accès à une route particulière.

##### Voici un résumé de son contenu :

1. Importation des modules et services :
   * Le fichier commence par l'importation des modules nécessaires depuis Angular. **Injectable** est importé depuis **@angular/core**, et **CanActivate** ainsi que **Router** sont importés depuis **@angular/router**.
   * Le service **AuthService** est également importé depuis son emplacement correspondant.
2. Définition de la classe **AuthGuard** :
   * La classe **AuthGuard** est décorée avec **@Injectable({ providedIn: 'root' })**, ce qui signifie qu'elle est injectable en tant que service et peut être fournie dans la racine de l'application. Cela permet de l'injecter automatiquement dans d'autres composants ou gardiens qui en ont besoin.
3. Constructeur :
   * Le constructeur de **AuthGuard** prend en deux arguments : **authService** et **router**.
   * **authService** est une instance du service **AuthService** qui gère l'authentification de l'utilisateur.
   * **router** est une instance du service **Router** qui permet la navigation vers différentes routes.
4. Méthode **canActivate** :
   * La méthode **canActivate** est une méthode requise par l'interface **CanActivate**. Elle est appelée lorsqu'un utilisateur tente d'accéder à une route protégée par ce gardien.
   * À l'intérieur de cette méthode, il y a une vérification de l'état de l'authentification en utilisant **this.authService.isLoggedIn**. Cela indique si l'utilisateur est authentifié ou non.
   * Si l'utilisateur est authentifié (isLoggedIn est vrai), la méthode renvoie **true**, ce qui signifie que l'accès à la route est autorisé.
   * Si l'utilisateur n'est pas authentifié (isLoggedIn est faux), la méthode utilise **this.router.navigate(['/login'])** pour rediriger l'utilisateur vers la page de connexion et renvoie **false**, ce qui signifie que l'accès à la route est refusé.

En résumé, le fichier "auth.guard.ts" définit un gardien de route "AuthGuard" qui vérifie l'authentification de l'utilisateur. Si l'utilisateur est authentifié, il est autorisé à accéder à la route. Sinon, il est redirigé vers la page de connexion. Ce gardien peut être utilisé pour protéger certaines parties de l'application qui nécessitent une authentification préalable.

#### auth.service.ts

Le fichier "auth.service.ts" définit un service Angular appelé "AuthService" qui gère l'authentification des utilisateurs dans l'application.

##### Voici un résumé de son contenu :

1. Importation des modules et décorateur **@Injectable** :
   * Le fichier commence par l'importation des modules nécessaires depuis Angular. **Injectable** est importé depuis **@angular/core**.
   * La classe "AuthService" est décorée avec **@Injectable({ providedIn: 'root' })**, ce qui signifie qu'elle est injectable en tant que service et peut être fournie dans la racine de l'application. Cela permet de l'injecter automatiquement dans d'autres composants ou services qui en ont besoin.
2. Propriétés de la classe :
   * **isLoggedIn**: Une propriété booléenne qui indique si l'utilisateur est authentifié. Par défaut, elle est définie sur **false**, ce qui signifie que l'utilisateur n'est pas authentifié.
   * **redirectUrl**: Une chaîne de caractères qui peut être utilisée pour stocker l'URL vers laquelle l'utilisateur doit être redirigé après s'être authentifié avec succès.
3. Méthode **login** :
   * La méthode **login** prend deux arguments : **name** (nom d'utilisateur) et **password** (mot de passe).
   * À l'intérieur de la méthode, il y a une vérification si les valeurs **name** et **password** correspondent à des valeurs spécifiques (dans cet exemple, "sesamien" pour les deux).
   * Si les valeurs correspondent, la variable **isLoggedIn** est définie sur **true**, ce qui signifie que l'utilisateur est authentifié avec succès.
   * La méthode retourne un observable d'un booléen (**Observable<boolean>**) en utilisant la fonction **of(isLoggedIn)**. Il y a également un délai artificiel de 1000 millisecondes (1 seconde) ajouté avec **delay(1000)** pour simuler un processus de connexion asynchrone.
   * Avant de retourner l'observable, la méthode utilise **tap** pour mettre à jour la propriété **isLoggedIn** avec la valeur obtenue.
4. Méthode **logout** :
   * La méthode **logout** est utilisée pour déconnecter l'utilisateur en définissant la propriété **isLoggedIn** sur **false**.

En résumé, le fichier "auth.service.ts" définit un service "AuthService" qui permet de gérer l'authentification des utilisateurs. Il fournit des méthodes pour se connecter et se déconnecter, ainsi qu'une propriété pour vérifier l'état de l'authentification. Ce service peut être utilisé pour protéger certaines parties de l'application qui nécessitent une authentification préalable.

#### in-memory-data.service.ts

Le fichier "in-memory-data.service.ts" définit un service Angular utilisé en conjonction avec le module "angular-in-memory-web-api". Ce service simule un backend pour stocker des données en mémoire et les fournir via des API HTTP simulées.

##### Voici un résumé de son contenu :

1. Importation des modules et décorateur **@Injectable** :
   * Le fichier commence par l'importation des modules nécessaires. **Injectable** est importé depuis **@angular/core**.
   * La classe "InMemoryDataService" est décorée avec **@Injectable({ providedIn: 'root' })**, ce qui signifie qu'elle est injectable en tant que service et peut être fournie dans la racine de l'application.
2. Implémentation de l'interface **InMemoryDbService** :
   * La classe "InMemoryDataService" implémente l'interface **InMemoryDbService**, ce qui signifie qu'elle doit fournir une implémentation pour la méthode **createDb()**.
3. Méthode **createDb** :
   * La méthode **createDb** est une méthode requise de l'interface **InMemoryDbService**.
   * À l'intérieur de cette méthode, un ensemble de données simulées est créé et stocké dans une variable appelée **sesamiens**. Dans cet exemple, **SESAMIENS** est importé depuis le fichier "./sesamien/mock-sesamien-list" pour fournir les données simulées.
   * La méthode retourne un objet avec une seule propriété **sesamiens**, qui contient les données simulées.

En résumé, le fichier "in-memory-data.service.ts" est utilisé pour créer une base de données en mémoire contenant des données simulées, qui peuvent être consultées via des API HTTP simulées. Cela permet de développer et de tester des fonctionnalités d'API sans nécessiter un backend réel pendant le développement de l'application. Ce service est souvent utilisé pour simuler des appels HTTP lorsque le backend réel n'est pas encore disponible.Haut du formulaire

## Contenu de angular.json

Le fichier "angular.json" est un fichier de configuration principal pour un projet Angular. Il définit la structure du projet, les paramètres de construction, les dépendances et d'autres configurations importantes.

### Résumé de son contenu :

1. **$schema**: Spécifie le schéma JSON utilisé pour valider la structure du fichier de configuration. Il pointe vers le fichier de schéma Angular CLI.
2. **version**: Indique la version de la configuration du projet Angular. Dans cet exemple, la version est définie à 1.
3. **newProjectRoot**: Indique le répertoire racine des nouveaux projets. Dans cet exemple, les nouveaux projets seront créés dans le répertoire "projects".
4. **projects**: Cette section contient la configuration principale du projet. Dans cet exemple, le projet s'appelle "ng-sesame-app" et est de type "application".
   * **root**: Spécifie le répertoire racine du projet. Dans cet exemple, il est laissé vide, ce qui signifie qu'il est situé à la racine du répertoire du projet.
   * **sourceRoot**: Indique le répertoire source du projet. Dans cet exemple, il est défini sur "src".
   * **prefix**: Définit le préfixe des sélecteurs CSS générés automatiquement pour les composants. Dans cet exemple, le préfixe est "app".
   * **architect**: Cette section contient des configurations pour les différentes étapes de la construction du projet.
     + **build**: Configuration pour la construction du projet. Il spécifie le générateur ("@angular-devkit/build-angular:browser") et les options de construction telles que le chemin de sortie, le fichier d'index, le point d'entrée principal, les polyfills, etc. Il y a également des configurations spécifiques pour "production" et "development" qui peuvent être utilisées lors de la construction pour différents environnements.
     + **serve**: Configuration pour le serveur de développement. Il utilise le générateur "@angular-devkit/build-angular:dev-server" et définit des configurations pour "production" et "development". Le serveur de développement est utilisé pour tester l'application localement.
     + **extract-i18n**: Configuration pour extraire les fichiers de traduction. Il est utilisé pour générer des fichiers de traduction à partir du code source de l'application.
5. **cli**: Cette section contient des informations sur l'analyse et la collecte de données de télémétrie Angular CLI.

En résumé, le fichier "angular.json" est essentiel pour configurer et construire un projet Angular. Il spécifie la structure du projet, les paramètres de construction et d'autres configurations importantes pour le développement et la production de l'application.

## package-lock.json

### Résumé de son contenu :

## package.json

### Résumé de son contenu :

## tsconfig.app.json

Le fichier "tsconfig.app.json" est un fichier de configuration TypeScript spécifique à une application Angular. Il est utilisé pour définir les options de compilation TypeScript pour le code source de l'application Angular.

### Résumé de son contenu :

1. **extends**: Il spécifie que ce fichier de configuration hérite des paramètres définis dans le fichier "tsconfig.json" situé à la racine du projet. Cela permet de réutiliser les options de compilation générales définies dans "tsconfig.json" et de les personnaliser pour l'application spécifique.
2. **compilerOptions**: Cette section définit les options de compilation TypeScript spécifiques à l'application.
   * **outDir**: Il indique le répertoire de sortie pour les fichiers JavaScript transpilés. Dans cet exemple, les fichiers transpilés seront placés dans le répertoire "./out-tsc/app". Cela signifie que les fichiers JavaScript générés seront stockés dans un dossier séparé par rapport au code source.
   * **types**: Cette option spécifie un tableau de types TypeScript à inclure dans le projet. Dans cet exemple, aucun type n'est spécifié (tableau vide).
3. **files**: Il spécifie les fichiers TypeScript à inclure dans la compilation. Dans cet exemple, seul le fichier "src/main.ts" est inclus. C'est le point d'entrée principal de l'application Angular.
4. **include**: Cette option indique les fichiers TypeScript à inclure dans la compilation en utilisant des motifs globaux. Dans cet exemple, tous les fichiers ".d.ts" (déclarations TypeScript) situés dans le répertoire "src" et ses sous-répertoires seront inclus.

En résumé, le fichier "tsconfig.app.json" est une configuration TypeScript spécifique à l'application Angular qui hérite des paramètres définis dans "tsconfig.json" et les personnalise pour l'application. Il spécifie le répertoire de sortie des fichiers transpilés, les fichiers à inclure dans la compilation et les types TypeScript à utiliser.

## tsconfig.json

Le fichier "tsconfig.json" est un fichier de configuration TypeScript pour un projet Angular. Il définit les options de compilation TypeScript globales pour l'ensemble du projet.

### Résumé de son contenu :

1. **compileOnSave**: Cette option est définie sur "false" pour désactiver la compilation automatique lors de la sauvegarde des fichiers TypeScript. Elle est généralement gérée par l'IDE ou l'éditeur de code utilisé.
2. **compilerOptions**: Cette section définit les options de compilation TypeScript globales pour le projet.
   * **baseUrl**: Il spécifie le chemin de base relatif à partir duquel les fichiers TypeScript sont résolus. Dans cet exemple, "./" signifie que le chemin de base est le répertoire racine du projet.
   * **outDir**: Il indique le répertoire de sortie pour les fichiers JavaScript transpilés. Dans cet exemple, les fichiers transpilés seront placés dans le répertoire "./dist/out-tsc".
   * **forceConsistentCasingInFileNames**: Cette option garantit une casse cohérente dans les noms de fichiers sur des systèmes de fichiers insensibles à la casse (comme Windows). Elle est définie sur "true".
   * **strict**: Cette option active un ensemble strict de vérifications TypeScript. Elle est définie sur "true", ce qui signifie que des vérifications strictes sont activées pour le projet.
   * **strictPropertyInitialization**: Cette option est définie sur "false". Lorsqu'elle est activée (à "true"), TypeScript exige que toutes les propriétés de classe soient initialisées dans le constructeur ou par une valeur par défaut. Elle est désactivée dans cet exemple.
   * **noImplicitOverride**: Cette option est définie sur "true" et signifie que TypeScript génère une erreur si une méthode de sous-classe n'annule pas explicitement une méthode héritée de la classe de base.
   * **noPropertyAccessFromIndexSignature**: Cette option est définie sur "true" pour empêcher l'accès aux propriétés via une signature d'index. Cela renforce la sécurité.
   * **noImplicitReturns**: Lorsqu'elle est définie sur "true", TypeScript génère une erreur si une fonction n'a pas de déclaration "return" explicite. Elle est activée dans cet exemple.
   * **noFallthroughCasesInSwitch**: Cette option est définie sur "true" pour détecter les cas "fallthrough" dans les instructions "switch" et générer une erreur si nécessaire.
   * **sourceMap**: Cette option est définie sur "true" pour générer des fichiers source map (.map) pour le débogage.
   * **declaration**: Lorsqu'elle est activée (à "true"), TypeScript génère des fichiers de déclaration (.d.ts) pour le projet. Elle est désactivée dans cet exemple.
   * **downlevelIteration**: Cette option est activée (à "true") pour activer la génération de code itératif descendante.
   * **experimentalDecorators**: Cette option est définie sur "true" pour activer les décorateurs expérimentaux TypeScript.
   * **moduleResolution**: Il spécifie la méthode de résolution de modules à utiliser. Dans cet exemple, "node" est utilisé, ce qui signifie que la résolution des modules suit les règles de Node.js.
   * **importHelpers**: Cette option est activée (à "true") pour générer des appels aux fonctions d'aide de module plutôt que de les copier dans chaque fichier.
   * **target**: Il spécifie la version ECMAScript cible pour la compilation. Dans cet exemple, "ES2022" est utilisé.
   * **module**: Il spécifie le format des modules pour la compilation. Dans cet exemple, "ES2022" est utilisé.
   * **useDefineForClassFields**: Lorsqu'elle est activée (à "false"), TypeScript utilise "var" plutôt que "define" pour les champs de classe. Elle est désactivée dans cet exemple.
   * **lib**: Cette option spécifie les bibliothèques TypeScript à inclure dans le projet. Dans cet exemple, "ES2022" et "dom" sont inclus.
3. **angularCompilerOptions**: Cette section contient des options spécifiques au compilateur Angular.
   * **enableI18nLegacyMessageIdFormat**: Cette option est définie sur "false". Elle contrôle le format des identifiants de message pour la localisation. Lorsqu'elle est désactivée, le nouveau format est utilisé.
   * **strictInjectionParameters**: Lorsqu'elle est activée (à "true"), cette option impose des contrôles stricts sur les paramètres d'injection de dépendances dans les constructeurs.
   * **strictInputAccessModifiers**: Lorsqu'elle est activée (à "true"), cette option impose des contrôles stricts sur les modificateurs d'accès pour les propriétés d'entrée (input) des composants.
   * **strictTemplates**: Lorsqu'elle est activée (à "true"), cette option impose des contrôles stricts sur les modèles Angular.

En résumé, le fichier "tsconfig.json" définit les options de compilation TypeScript globales pour le projet Angular, y compris les paramètres stricts, les options de résolution de modules, la version ECMAScript cible, et d'autres options. Il sert à configurer le comportement de TypeScript lors de la compilation du code source de l'application

Résumé  avec Chat GPT  à la date du : 24/09/23

**Projet : Gestion des Sesamiens**

**Structure du dossier "app/" :**

1. **add-sesamien/** : Contient le composant et le formulaire pour ajouter un Sesamien.
   * **add-sesamien.component.ts** : Définit le composant "AddSesamienComponent" pour l'ajout de Sesamiens.
   * **add-sesamien.component.html** : Modèle de vue du composant pour le formulaire d'ajout.
2. **detail-sesamien/** : Contient le composant pour afficher les détails d'un Sesamien.
   * **detail-sesamien.component.ts** : Définit le composant "DetailSesamienComponent" pour afficher les détails d'un Sesamien.
   * **detail-sesamien.component.html** : Modèle de vue du composant pour l'affichage des détails.
3. **edit-sesamien/** : Contient le composant pour l'édition d'un Sesamien.
   * **edit-sesamien.component.ts** : Définit le composant "EditSesamienComponent" pour l'édition d'un Sesamien.
   * **edit-sesamien.component.html** : Modèle de vue du composant pour l'édition.
4. **list-sesamien/** : Contient le composant pour la liste des Sesamiens.
   * **list-sesamien.component.ts** : Définit le composant "ListSesamienComponent" pour afficher la liste des Sesamiens.
   * **list-sesamien.component.html** : Modèle de vue du composant pour la liste.
5. **loader/** : Contient le composant pour l'affichage d'un indicateur de chargement.
   * **loader-sesamien.component.ts** : Définit le composant "LoaderComponent" pour l'affichage du chargement.
6. **search-sesamien/** : Contient le composant et le formulaire de recherche de Sesamiens.
   * **search-sesamien.component.ts** : Définit le composant "SearchSesamienComponent" pour la recherche de Sesamiens.
   * **search-sesamien.component.html** : Modèle de vue du composant pour le formulaire de recherche.
7. **sesamien-form/** : Contient le composant et le formulaire pour la création et la modification de Sesamiens.
   * **sesamien-form.component.ts** : Définit le composant "SesamienFormComponent" pour le formulaire de création et d'édition de Sesamiens.
   * **sesamien-form.component.html** : Modèle de vue du composant pour le formulaire de création et d'édition.
   * **sesamien-form.component.css** : Feuille de style CSS associée au formulaire.

**Services :**

* **sesamien.service.ts** : Fournit des méthodes pour interagir avec les données des Sesamiens, telles que la récupération de la liste, l'ajout, la mise à jour et la suppression.

**Modèle de données :**

* **sesamien.ts** : Définit la classe "Sesamien" avec des propriétés telles que "id", "name", "hp", "cp", "picture", "mentions", et "created". Cette classe est utilisée pour représenter les objets Sesamiens dans l'application.

**Fonctionnalités clés du projet :**

* Ajout, modification, et suppression de Sesamiens.
* Affichage des détails d'un Sesamien.
* Recherche de Sesamiens par nom.
* Liste des Sesamiens avec paginations.
* Gestion des mentions pour chaque Sesamien.
* Indicateur de chargement lors du chargement des données.

**Fonctionnalités à venir (non implémentées) :**

* Gestion des utilisateurs avec différents rôles (administrateur, professeur, visiteur).
* Restrictions d'accès aux fonctionnalités en fonction du rôle de l'utilisateur.

**Étapes réalisées jusqu'à présent :**

1. Création de la structure de dossiers et des composants.
2. Mise en place des formulaires pour l'ajout, l'édition, et la recherche de Sesamiens.
3. Création d'un service "SesamienService" pour gérer les opérations CRUD sur les Sesamiens.
4. Affichage des détails d'un Sesamien.
5. Affichage de la liste des Sesamiens avec pagination.
6. Indicateur de chargement lors du chargement des données.

**Étapes à venir :**

1. Implémentation de la gestion des mentions pour chaque Sesamien.
2. Mise en place de la gestion des utilisateurs avec différents rôles.
3. Restreindre l'accès aux fonctionnalités en fonction du rôle de l'utilisateur.

Ce résumé devrait vous permettre de comprendre le projet dans son ensemble, les fonctionnalités mises en œuvre jusqu'à présent, et les étapes à venir pour son développement. Vous pouvez maintenant poser des questions spécifiques ou discuter des parties du projet qui nécessitent plus d'attention.