Projet : Polygon

Modules développés :

**Module "AUTH"**

**LOG IN (temps de développement 45min)**

Fichier HTML

Affichage du formulaire de connexion avec des champs pour le nom d'utilisateur et le mot de passe.

Ajout de validations aux champs du formulaire, par exemple, le nom d'utilisateur et le mot de passe sont requis.

Ajout d'un bouton de soumission pour envoyer le formulaire.

Inclusion d'une option pour afficher/masquer le mot de passe.

Ajout d'une case à cocher pour se souvenir de l'utilisateur.

Fourniture d'un lien pour réinitialiser le mot de passe.

Fichier TypeScript (ts) :

Création d'un formulaire réactif à l'aide de FormGroup et FormControl, avec des validations pour les champs requis.

Définition de la méthode submitLogin() pour gérer la soumission du formulaire de connexion.

Activation de l'indicateur isLoading pour afficher une icône de chargement pendant le traitement de la requête de connexion.

Envoi d'une requête HTTP au service d'authentification pour effectuer la connexion.

Gestion de différentes réponses de la requête :

Si la réponse est null, affichage d'une erreur.

Si la réponse est une erreur, affichage d'un message d'erreur.

Si la connexion réussit, redirection vers la page utilisateur, stockage de l'information de connexion dans les cookies et sessionStorage.

Gestion des erreurs lors de la requête de connexion, avec affichage d'un message d'erreur approprié.

Désactivation de l'indicateur isLoading après le traitement complet de la requête.

**REGISTER (temps de développement 1h10 min)**

Fichier HTML

Le formulaire d'inscription est affiché avec des champs pour le nom complet, le nom d'utilisateur, le numéro de téléphone et le mot de passe.

Des validations sont appliquées aux champs du formulaire, par exemple, le nom d'utilisateur est requis.

Un bouton de soumission est présent pour envoyer le formulaire.

Un lien est fourni pour rediriger vers la page de connexion.

Fichier TypeScript (ts) :

Un formulaire réactif est créé à l'aide de FormGroup et FormControl, avec des validations pour les champs requis.

La méthode newUser() est définie pour traiter l'inscription d'un nouvel utilisateur.

Lorsque l'utilisateur soumet le formulaire, la méthode newUser() est appelée.

Dans newUser(), l'indicateur isLoading est activé pour afficher une icône de chargement pendant le traitement de la requête.

Les données du formulaire sont extraites à partir du formulaire réactif.

Une requête HTTP est envoyée au service d'authentification pour enregistrer le nouvel utilisateur.

En cas de succès de la requête, l'utilisateur est redirigé vers la page de connexion, et une notification de succès est affichée à l'aide de SweetAlert2.

En cas d'erreur lors de la requête, un message d'erreur est affiché dans la console.

Dans tous les cas (succès, erreur ou achèvement), l'indicateur isLoading est désactivé pour masquer l'icône de chargement.

**Module "USER"**

**AGGREGATE (temps de developpement 30 minutes)**

Fichier Html :

Affichage d'un titre "Aggregate".

Affichage des informations sur la date, le ticker et le prix d'ouverture dans des groupes d'entrée.

Affichage d'un tableau pour afficher les données agrégées.

Utilisation d'une boucle \*ngFor pour parcourir les éléments de aggregateDate et afficher les données dans le tableau.

Fichier TypeScript (ts) :

Importation des dépendances nécessaires, notamment Component, OnInit, ActivatedRoute et AggregateService.

Définition de variables pour stocker les informations sur la date, le ticker, le prix d'ouverture et les données agrégées.

Création d'une méthode resAggregate() pour récupérer les données agrégées à partir du service AggregateService.

Utilisation de la méthode subscribe() pour effectuer une requête HTTP et gérer les réponses de manière asynchrone :

Gestion de la réponse réussie (next) : Affichage des données agrégées dans la console, assignation des données à aggregateDate, puis initialisation du plugin DataTable après un délai de 1 seconde.

Gestion des erreurs (error) : Affichage des erreurs dans la console.

Complétion de la requête (complete) : Affichage d'un message indiquant que la requête est terminée.

Implémentation de OnInit pour exécuter le code lors de l'initialisation du composant :

Souscription aux paramètres de l'URL à l'aide de ActivatedRoute pour obtenir les valeurs de la date, du ticker et du prix d'ouverture.

Appel de la méthode resAggregate() pour récupérer les données agrégées une fois les paramètres obtenus.

**HOME ((temps de developpement 10 minutes)**

Fichier Html :

Utilisation de la classe container-fluid pour créer un conteneur fluide qui s'étend sur toute la largeur de la fenêtre.

Création d'une section avec la classe page-image pour afficher une image.

Utilisation d'une carte (<div class="card">) pour entourer l'image, ce qui peut être utile pour ajouter des styles supplémentaires ou d'autres éléments à l'avenir.

Inclusion de l'image elle-même à l'intérieur de la carte à l'aide de la balise <img>.

Utilisation de la balise <img> avec l'attribut src pour spécifier le chemin de l'image à afficher. Ici, l'image est "assets/images/logos/img1.jpg".

**SETTINGS(temps de developpement 23min)**

Fichier Html :

"Enable Daily Update" : Une case à cocher pour activer ou désactiver les mises à jour quotidiennes.

"Set Update Time" : Un champ de texte pour définir l'heure de mise à jour, avec un petit texte explicatif en dessous pour indiquer qu'il s'agit du temps de cron.

**STATS(temps de developpement 5jrs)**

Fichier Html

Définition d'un formulaire avec un groupe de contrôles (filter\_form) pour les filtres de recherche.

Utilisation de directives conditionnelles (\*ngIf) pour afficher un spinner de chargement (spinner-container) lorsque les données sont en cours de chargement.

Utilisation de la classe Bootstrap row pour organiser les éléments en lignes.

Utilisation de Bootstrap pour la mise en page, y compris les classes de grille (col-md-\*) pour organiser les éléments en colonnes.

Utilisation de champs de texte pour permettre à l'utilisateur de saisir des valeurs de filtre.

Utilisation d'une boucle \*ngFor pour afficher une liste déroulante de tickers à sélectionner.

Utilisation de directives conditionnelles (\*ngIf) pour afficher un message lorsque les données ne sont pas disponibles.

Utilisation d'un tableau HTML (<table>) pour afficher les données statistiques.

Utilisation de directives routerLink pour naviguer vers une autre page avec des paramètres spécifiques lorsqu'un lien est cliqué.

Fichier TypeScript (ts) :

Importation des dépendances nécessaires.

Définition d'un objet filter\_form de type FormGroup pour gérer les valeurs du formulaire.

Définition d'une méthode generateStats() pour générer les statistiques en fonction des valeurs du formulaire et appeler le service approprié.

Définition d'une propriété tickers contenant une liste de tickers à afficher dans la liste déroulante.

Définition d'une méthode selectForm() pour initialiser le plugin Select2 pour la liste déroulante de sélection des tickers.

Définition d'une méthode initSelectPicker() pour initialiser le plugin Flatpickr pour la sélection de la date.

Implémentation de l'interface Angular OnInit pour exécuter les méthodes d'initialisation lors du chargement du composant.

Dossiers supplémentaires :

**Services : Contient tous les services nécessaires au fonctionnement de l'application.**

**services déclarer: (temps de developpement 15min)**

**AggregateService**:

Méthode aggregate(): Cette méthode envoie une requête HTTP POST vers l'URL /api/scraper/Aggregates avec le corps de la requête fourni en paramètre (body). Les options HTTP définies dans httpOptions sont également passées lors de l'envoi de la requête.

**AuthService**:

Méthode login() :

Cette méthode envoie une requête HTTP POST vers l'API /api/polygon/Login avec le corps de la requête fourni en paramètre (body). Les en-têtes HTTP définis précédemment sont inclus dans la requête.

Méthode isLoggedinUser() :

Cette méthode vérifie si l'utilisateur est connecté en consultant les cookies du navigateur pour vérifier s'il existe un cookie nommé isLoggedIn. Si ce cookie existe, la méthode retourne true, sinon elle retourne false.

Méthode register() :

Cette méthode envoie une requête HTTP POST vers l'API /api/polygon/createUser avec le corps de la requête fourni en paramètre (body). Aucun en-tête HTTP spécifique n'est défini pour cette requête.

**StatsService**:

Méthode stats() :

Cette méthode envoie une requête HTTP POST vers l'API /api/scraper/generateStats avec le corps de la requête fourni en paramètre (body). Les en-têtes HTTP définis précédemment sont inclus dans la requête.

**Guards : Contient les guards pour l'authentification.**

**AuthGuard (temps de developpement 15min)**

Méthode canActivate :

Vérifie si l'utilisateur est authentifié en utilisant isLoggedinUser() du service AuthService.

Redirige l'utilisateur vers /tauth si non authentifié.

Méthode canActivateChild :

Similaire à canActivate, mais pour les routes enfants.

Redirige l'utilisateur vers /login si non authentifié.

Méthode canDeactivate :

Vérifie si l'utilisateur est authentifié.

Méthode canLoad :

Actuellement, retourne toujours true, permettant le chargement de modules de manière asynchrone.