|  |
| --- |
| Concepteur Développeur d’application |
| Dossier de projet |

Une image contenant Graphique, Police, graphisme, capture d’écran

Description générée automatiquement

PAC Pierre

CDA – Titre de niveau 6

Février 2023 – octobre 2023

Table des matières

[I. Remercîments 4](#_Toc147134840)

[II. Entreprise 5](#_Toc147134841)

[A. Présentation de l’entreprise 5](#_Toc147134842)

[III. Mise en place du projet 6](#_Toc147134843)

[A. Résumé du projet 6](#_Toc147134844)

[B. Compétences couvertes par le projet 7](#_Toc147134845)

[C. Expressions des besoins 8](#_Toc147134846)

[Interface Candidat 8](#_Toc147134847)

[Interface Administrateur 8](#_Toc147134848)

[Sécurité et Authentification Back-End 9](#_Toc147134849)

[D. Gestion de projet 10](#_Toc147134850)

[E. Spécifications techniques 12](#_Toc147134851)

[IV. Réalisations du projet 14](#_Toc147134852)

[A. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité 14](#_Toc147134853)

[Maquetter une application 14](#_Toc147134854)

[Développer la partie frontend d’une interface utilisateur web 15](#_Toc147134855)

[Développer la partie backend d’une interface utilisateur web 28](#_Toc147134856)

[Développer des composants d’accès aux données 37](#_Toc147134857)

[B. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité 38](#_Toc147134858)

[Concevoir une base de données 38](#_Toc147134859)

[Mettre en place une base de données 38](#_Toc147134860)

[Développer des composants dans le langage d’une base de données 38](#_Toc147134861)

[C. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité 38](#_Toc147134862)

[Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement 38](#_Toc147134863)

[Concevoir une application 38](#_Toc147134864)

[Développer des composants métier 38](#_Toc147134865)

[Construire une application organisée en couches 38](#_Toc147134866)

[Préparer et exécuter les plans de tests d’une application 39](#_Toc147134867)

[Préparer et exécuter le déploiement d’une application 39](#_Toc147134868)

[V. Annexe 40](#_Toc147134869)

# Remercîments

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à faire de mon stage au Mipih une expérience inestimable et enrichissante. Cette opportunité a été une étape clé dans mon parcours professionnel et académique, et je suis sincèrement reconnaissant pour l'accueil et le soutien que j'ai reçu.

Tout d'abord, un grand merci à l'entreprise Mipih pour m'avoir offert cette précieuse opportunité de stage. Cette expérience m'a permis de me plonger dans un environnement professionnel stimulant et formateur, qui a significativement élargi mes compétences et ma vision du monde du travail dans le secteur de la santé numérique.

Un immense merci à Wilfried Barreaux, Manager en "Construction Gestion du Patrimoine Applicatif RH & GTA", pour m'avoir confié ce projet et m'avoir fait confiance dès le début. Son soutien continu et son professionnalisme exemplaire ont joué un rôle clé dans mon développement professionnel au fil de ce stage. Sa bienveillance a été un pilier de mon évolution et de ma réussite au sein de l'équipe.

Je souhaite aussi exprimer toute ma gratitude à Loïc Kebouët, Nicolas NOWINSKI et Gabriel Roget pour leur précieux soutien technique. Grâce à leur savoir-faire et à leur disponibilité pour répondre à mes questions, j'ai pu approfondir mes compétences et élargir mon horizon professionnel. Leur contribution a véritablement enrichi la qualité de mon apprentissage et de mon expérience globale au sein de l'entreprise.

Je souhaite également souligner l'accueil chaleureux et l'ambiance positive apportés par toute l'équipe de Prodige 2 : Johan Moreaux, le Product Manager, Claudia Mercier, la Product Owner, ainsi que Amaury Mozin, Cédric Wargnier, Grégory Dujardin et Jean Widcoq. Votre esprit d'équipe et votre ouverture m'ont permis de me sentir intégrée et valorisée dès le premier jour.

Enfin, un merci spécial aux équipes de Prodige 1 et d'Agir pour leur accueil au sein de l'entreprise. La bienveillance et la collaboration entre les équipes ont créé un environnement de travail des plus agréables.

Cette expérience au Mipih a été pour moi une véritable aventure humaine et professionnelle, et c'est grâce à chacun d'entre vous que cela a été possible. Merci de tout cœur.

# Entreprise

## Présentation de l’entreprise

Le Mipih est une entité de type GIP (Groupement d'intérêt public) qui se focalise sur les technologies de l'information dans le domaine médical. En sa qualité de concepteur et intégrateur de systèmes, il propose des innovations technologiques spécifiques pour les hôpitaux publics. Fondé à Toulouse en France, il s'efforce de répondre à la demande croissante en matière de gestion des systèmes d'information pour les institutions de santé. L'accent est mis sur le développement de solutions personnalisées aux enjeux de la médecine.

L’entreprise du Mipih, dispose de bureaux à Toulouse, Amiens, Bordeaux et Reims et est un acteur clé dans le paysage sanitaire français. Elle développe des solutions sur mesure pour chaque institution, abordant divers aspects les dossiers médicaux, le parcours patients, les ressources humaines et financières.

Reconnu pour son professionnalisme et la qualité de son service client, le Mipih comprend les défis auxquels le secteur de la santé est confronté et vise à améliorer les services de soins et l'efficacité opérationnelle grâce à des applications spécialisées.

En contribuant à la numérisation des services et infrastructures de santé, l'entreprise offre des applications qui optimisent la gestion des centres de santé.

En outre, le Mipih se concentre sur l'amélioration de l'expérience patient en mettant en place des bornes d'accueil qui simplifient et accélèrent le processus d'admission (Annexe N° 1).

Les centres de données situés à Amiens et Toulouse permettent un stockage sécurisé des applications et des données médicales. Le Mipih est certifié comme Hébergeur de données de santé (HDS), attestant ainsi de sa compétence à gérer des informations médicales sensibles (Annexe N° 2).

En perpétuelle évolution, le Mipih ajuste continuellement ses services pour répondre aux exigences changeantes du domaine médical public. Grâce à son savoir-faire et à son engagement, il joue un rôle important dans l'optimisation des institutions de santé publique.

# Mise en place du projet

## Résumé du projet

During my 12-week internship period, which took place from July 17 to October 6, 2023, I had the opportunity to join the "Prodige 2" team within the DOGRH division. This internship was part of the broader context of the Human Resources Development (HRD) division, led by Wilfried Barreaux. The main objective of the project was to modernize the way candidates were evaluated for various roles within the DOGRH division, ranging from internships and apprenticeships to fixed-term and permanent contracts (CDD and CDI).

To achieve this goal, the back-end of the application was developed using the Java programming language and the Spring Boot framework. This choice was motivated by the advantages these technologies offer in terms of scalability and robustness. On the front-end side, Angular was the technology of choice to ensure a reactive and user-friendly interface.

The application had several key components. On one hand, it allowed each candidate to take a multiple-choice test (QCM) to evaluate their skills in relation to the targeted position. On the other hand, it offered a programming test to assess technical skills. In addition to candidate-oriented features, the application included a comprehensive administrator interface. This offered recruitment managers the ability to manage the question repository, to create or modify multiple-choice tests, to track applications, and to review results, all within a secure environment.

Particular attention was given to the design and ergonomics of the user interface. We placed special emphasis on creating an interface that was not only pleasant to look at but also intuitive to use. In terms of security, several measures were put in place, including two-factor authentication and password encryption, to ensure that sensitive information was handled with the utmost care.

At the end of my internship, the application was not only functional but also well-received by the DOGRH department. It succeeded in streamlining and improving the efficiency of the evaluation process, thereby significantly reducing the time required to evaluate candidates while simultaneously increasing the quality of recruitment.

## Compétences couvertes par le projet

Chacune des compétences citées ci-dessous, est tirée du « Référentiel Emploi Activités Compétences du titre professionnel Concepteur développeur d’applications ».

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité
   1. Maquetter une application
   2. Développer des composants d’accès aux données
   3. Développer la partie frontend d’une interface utilisateur web
   4. Développer la partie backend d’une interface utilisateur web
2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité
   1. Concevoir une base de données
   2. Mettre en place une base de données
   3. Développer des composants dans le langage d’une base de données
3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité
   1. Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement
   2. Concevoir une application
   3. Développer des composants métier
   4. Construire une application organisée en couches
   5. Préparer et exécuter les plans de tests d’une application
   6. Préparer et exécuter le déploiement d’une application

## Expressions des besoins

Le service DOGRH du Mipih souhaite optimiser son processus de recrutement grâce au développement d'un outil en ligne spécialisé. Cet outil aura deux composantes majeures : une interface dédiée aux candidats et une plateforme d'administration pour gérer les différents aspects du processus d'évaluation.

### Interface Candidat

L'objectif est de fournir aux candidats une expérience utilisateur intuitive et efficace. Ainsi, après avoir été préalablement enregistré par un administrateur, le candidat pourra accéder à son espace de test simplement en saisissant son nom et son prénom. Il n'aura pas besoin de s'inscrire ni de créer un compte, ce qui facilite la démarche.

#### Fonctionnalités pour le candidat :

Test de Type QCM Chronométré : Le candidat devra répondre à un QCM dans un temps imparti. Ce test évaluera ses compétences sur des technologies spécifiques comme JAVA, Hibernate ou Angular.

Test de Code : Après le QCM, l'interface proposera un test de programmation. Le candidat pourra coder en temps réel, permettant ainsi d'évaluer ses compétences pratiques.

### Interface Administrateur

La partie administrateur se doit d'être complète et polyvalente pour gérer divers aspects du processus de recrutement.

#### Fonctionnalités pour l’administrateur :

Gestion des Questions : Les administrateurs pourront ajouter, modifier ou supprimer des questions. Ils pourront également les trier par langue/technologie (JAVA, Hibernate, Angular, etc.), par niveau de difficulté et par catégorie (Logique, Algorithme, etc.).

Création et Gestion des QCMs : L'interface permettra de créer des tests QCM en choisissant des questions en fonction de critères tels que la technologie, le niveau et la catégorie. Une fois les paramètres sélectionnés, les questions seront choisies aléatoirement pour constituer le QCM.

Gestion des Sandboxes de Code : Les administrateurs auront la possibilité d'ajouter des tests de code qui sont directement liés à un dépôt GitHub. Cela permet d'évaluer les compétences des candidats dans un environnement de développement réaliste.

Gestion des Entretiens et des Candidats : Les administrateurs pourront enregistrer des candidats, leur associer un QCM et une sandbox de code, et suivre leurs performances. Ils auront accès à des indicateurs comme le pourcentage de réussite au QCM et à la partie de code.

### Sécurité et Authentification Back-End

#### Fonctionnalités de Sécurité :

Authentification : Un processus d'authentification robuste est en place pour les administrateurs. L'authentification à deux facteurs peut être envisagée pour une couche de sécurité supplémentaire.

Autorisation : Une fois authentifié, le système contrôlera le niveau d'accès de l'utilisateur pour déterminer les fonctionnalités auxquelles il peut accéder.

Cryptage des Mots de Passe : Conformément aux meilleures pratiques de sécurité, seuls les mots de passe seront cryptés avant leur stockage dans la base de données. Ce cryptage utilise des algorithmes modernes pour assurer la confidentialité et l'intégrité des mots de passe.

#### Gestion des Données et Logique Métier

Une fois les données collectées via l'interface front-end, elles doivent être transmises de manière sécurisée vers la base de données. Le back-end joue ici un rôle crucial en appliquant la logique métier appropriée.

Fonctionnalités de Gestion des Données :

Validation des Données : Avant l'enregistrement dans la base de données, le système validera les données pour s'assurer qu'elles respectent les critères prédéfinis, tels que le format des questions, le niveau de difficulté, etc.

Stockage des Résultats : Le back-end stockera les résultats des tests des candidats d'une manière structurée, permettant des analyses détaillées ultérieures.

Indicateurs de Performance : Le système calculera automatiquement divers indicateurs, tels que les pourcentages de réussite, pour aider les administrateurs à évaluer les performances des candidats plus efficacement.

#### Bonnes Pratiques de Codage en Java

La qualité du code est une priorité. Le développement du back-end respectera les bonnes pratiques de codage en Java, comme l'encapsulation, l'héritage, et le polymorphisme, pour assurer un code propre, modulaire et maintenable.

Respect des Principes SOLID : Ces principes de la programmation orientée objet seront suivis pour garantir un code de haute qualité et facile à gérer. (Annexe N° 3)

Tests Unitaires : Une série de tests unitaires sera implémentée pour s'assurer que chaque composant du système fonctionne comme prévu.

## Gestion de projet

Pendant mon stage de trois mois au Mipih, j'ai eu la chance d'être intégré dans une équipe qui suit les principes de la méthodologie agile SCRUM. Cette expérience m'a permis de constater combien cette approche favorise la flexibilité, la réactivité et la collaboration entre les membres de l'équipe. Ce fut surtout utile dans la gestion de projets qui nécessitent une adaptation constante et rapide aux changements.

Au sein de ce cadre agile, j'ai travaillé sur un projet qui avait deux objectifs principaux. Le premier était le développement d'un test sous forme de questionnaire à choix multiples (QCM). Le second objectif était l'intégration d'une section dédiée au codage. Afin d'aligner les efforts de l'équipe et de maintenir un rythme de travail soutenu, nous avons organisé des réunions de planification et de rétrospective à chaque fin de sprint, qui durait habituellement trois semaines. Ces réunions nous ont permis d'évaluer nos progrès et de redéfinir les objectifs pour les étapes suivantes.

Un aspect central de notre routine quotidienne était le stand-up meeting qui avait lieu chaque matin. Ce rituel incarnait parfaitement l'esprit agile en encourageant la communication ouverte et en identifiant rapidement les obstacles potentiels. Lors de ces réunions, chaque membre de l'équipe partageait ce qu'il avait accompli la veille et ce qu'il prévoyait de faire pendant la journée, ce qui nous aidait à aligner nos efforts et à aborder proactivement les défis.

Au sein du Mipih, nous employons divers outils de haute technicité pour optimiser notre flux de travail et respecter les principes agiles qui guident nos projets.

GitLab :

Nous nous appuyons sur GitLab en tant que plateforme de gestion de version de code source. Ce choix n'est pas anodin. GitLab ne se limite pas à héberger le code ; il offre une série de fonctionnalités collaboratives essentielles. Par exemple, la possibilité de créer des branches pour chaque version ou sous-version du projet permet à chaque membre de l'équipe de travailler sur des éléments distincts sans crainte de conflits de code. Ce système de branches est particulièrement efficace lors des revues de code, car il donne une vue d'ensemble des modifications apportées, facilitant ainsi le processus de validation et d'intégration.

Discourse :

Pour aller au-delà des simples forums de discussion techniques, nous avons mis en place Discourse comme une sorte de "StackOverflow" interne. Ce choix résonne parfaitement avec notre objectif de promouvoir une culture d'apprentissage et de partage. Grâce à Discourse, chaque membre de l'équipe peut non seulement poser des questions techniques mais également contribuer à résoudre les problèmes rencontrés par les autres. L'outil dispose également d'une fonction de recherche puissante, réduisant la redondance des questions et permettant un accès rapide à des solutions antérieurement discutées et validées.

Microsoft Teams :

En tant qu'outil de communication principal, Microsoft Teams est notre hub pour toutes les interactions en temps réel. Il offre une solution intégrée pour la messagerie instantanée, la planification des réunions, et même le partage de documents, ce qui élimine le besoin de jongler entre différentes applications. La fonction de statut en temps réel est particulièrement utile pour connaître la disponibilité des membres de l'équipe, ce qui permet d'ajuster notre approche communicationnelle selon que la personne est occupée, en réunion, ou disponible pour une interaction.

En somme, l'emploi combiné de ces outils sophistiqués nous permet de toucher à chaque facette de la gestion de projet, de la planification au développement et à la communication. Cette synergie d'outils a manifestement amélioré notre efficacité et a eu un impact positif sur le développement des solutions que nous proposons au Mipih.

## Spécifications techniques

Pour la réalisation de ce projet, nous avons mis en œuvre une panoplie de technologies modernes et d'outils de développement afin de garantir des normes élevées en termes d'efficacité, de robustesse et d'évolutivité.

Le cœur du système, le backend de l'application, a été développé en utilisant Java 17, une version récente et performante du langage Java. Pour faciliter la création d'une interface de programmation (API) robuste et flexible, nous avons choisi d'intégrer Spring Boot. Cette technologie a été un pilier essentiel pour construire une API REST qui répond de manière efficace aux demandes des utilisateurs.

En matière de gestion de données, nous avons adopté Hibernate, un framework de persistance très puissante. Hibernate simplifie considérablement les interactions avec la base de données, ce qui nous a permis de gagner du temps et de réduire la complexité de notre code. Cela a également contribué à améliorer la performance globale de l'application.

Pour simplifier davantage le développement Java, nous avons fait usage de Lombok, une bibliothèque qui automatise la création de code répétitif tel que les getters, setters et les constructeurs. Cette approche a permis de réduire la verbosité du code et d'améliorer la lisibilité, tout en minimisant le risque d'erreurs humaines.

Le frontend de l'application a été construit en utilisant Angular, une technologie robuste qui s'appuie sur des normes web modernes telles qu'HTML5, SCSS et TypeScript. Pour enrichir l'interface utilisateur et offrir une expérience utilisateur fluide, nous avons intégré la bibliothèque PrimeNG.

Afin de garantir une cohérence totale entre les environnements de développement, de test et de production, nous avons adopté Docker. Cette technologie de conteneurisation nous a permis de mettre en place une approche uniforme pour le déploiement de nos applications et services, éliminant ainsi les problèmes potentiels liés aux différences d'environnement.

Pour la gestion de nos données cruciales, nous avons opté pour PostgreSQL, un système de gestion de base de données robuste, reconnu pour sa capacité à gérer d'énormes volumes de données avec fiabilité. Pour la conception de la base de données, nous avons utilisé l'outil Looping, et pour la création d'interfaces utilisateur intuitives, Whimsical a été notre choix pour le maquettage et la conception.

La gestion globale du projet, y compris le suivi des versions, la gestion des tâches et la mise en place de l'intégration continue, a été assurée par GitLab. Cette plateforme collaborative nous a permis de coordonner efficacement notre équipe de développement.

Nous avons employé des environnements de développement spécifiques pour chaque aspect de notre projet. Le développement frontend s'est déroulé dans l'environnement VSCode, tandis qu'Eclipse a été notre environnement de prédilection pour le développement backend. DBeaver a été notre outil de choix pour visualiser et gérer la base de données PostgreSQL, tandis que Postman nous a aidés à tester et à valider les fonctionnalités de l'API.

Enfin, pour garantir un déploiement sans faille sur les serveurs internes du Mipih, nous avons utilisé Docker, qui a simplifié et rationalisé le processus de mise en production, tout en contribuant à la cohérence de notre infrastructure interne.

# Réalisations du projet

## Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

### Maquetter une application

Le maquettage d'une application occupe une place centrale dans le développement de logiciels, notamment dans des environnements qui adoptent des méthodes agiles comme Scrum. Cette méthode encourage l'itération rapide et la collaboration étroite entre les différentes parties prenantes, y compris les développeurs, les designers et les clients. Dans ce contexte, le maquettage sert à plusieurs fins. Il permet tout d'abord de visualiser rapidement les fonctionnalités et l'interface utilisateur, ce qui facilite la communication et la prise de décision. Le maquettage permet également de tester des idées sans avoir à développer des fonctionnalités complètes, ce qui correspond parfaitement à l'approche agile d'itération et de livraison continue. En somme, le maquettage constitue un outil indispensable pour aligner l'équipe de développement et les besoins du client de manière rapide et efficace.

Lors de mon stage au Mipih, j'ai eu l'opportunité de travailler sur le maquettage d'une application. L'entreprise m'a laissé une grande liberté en termes de conception, ce qui m'a permis de prendre des initiatives. Je me suis inspiré de leur site internet pour reprendre la charte de couleur ainsi que les polices et le style, afin de garantir une cohérence avec l'identité visuelle déjà en place.

Pour la réalisation de la maquette, j'ai opté pour l'outil en ligne Whimsical, qui offre un large panel de fonctionnalités, facilitant ainsi la création de maquettes à la fois fonctionnelles et esthétiques. Puisque le projet était spécifiquement destiné à être utilisé sur des ordinateurs, je n'ai pas eu à me soucier de la responsivité pour des appareils comme les smartphones ou les tablettes. Cela m'a permis de me concentrer entièrement sur l'expérience utilisateur sur ordinateur, en accord avec les besoins du projet.

Une fois cette charte établie, j'ai entamé la réalisation des pages clés de l'application. La première étape a été de développer une page de connexion qui sert à la fois les administrateurs et les candidats. Cette page constitue la porte d'entrée de l'application et a été conçue pour être aussi intuitive que sécurisée.

Par la suite, j'ai créé la page de test destinée aux candidats, axée sur une expérience utilisateur fluide et fonctionnelle. Cette étape était cruciale pour assurer que les utilisateurs puissent accomplir leurs tâches de manière efficace. En parallèle, plusieurs pages de gestion ont été développées du côté administrateur. Ces dernières offrent une vue d'ensemble et permettent une gestion aisée des différents modules et fonctionnalités (maquette en annexe N°3).

L'objectif de ce travail de maquettage était de donner une vision claire de l'esthétique et de l'ergonomie que je souhaitais apporter à cette application web. Chaque élément a été soigneusement pensé pour s'intégrer de manière harmonieuse, tout en répondant aux besoins spécifiques des différents utilisateurs.

### Développer la partie frontend d’une interface utilisateur web

#### Architecture d’Angular

Suite à l’installation d'Angular, la première étape a consisté à définir l’architecture de l’application. Angular, étant un framework structuré autour de composants qui incarnent différentes fonctionnalités de l’application, requiert une organisation soignée pour assurer une lisibilité et une maintenabilité optimales du code. Pour ce faire, les composants sont regroupés en modules. Un module Angular permet de regrouper des composants, services, directives, pipes, etc., de définir leurs dépendances et de réguler leur visibilité. Cinq modules ont ainsi été créés pour l’application :

CoreModule : Ce module, caractérisé par le fait qu’il n’est importé qu’une seule fois dans l’application, abrite des fichiers spécifiques, notamment les composants du header et du footer, et les services. Ces derniers établissent une connexion bidirectionnelle entre l’API et Angular et implémentent des logiques métier. Les intercepteurs, qui interceptent toutes les requêtes sortantes pour y ajouter des tokens, y sont également inclus.

SharedModule : À l’opposé du CoreModule, le SharedModule est importé dans chaque module de fonctionnalités nécessitant des composants partagés. Il convient d’éviter d’y inclure des services pour prévenir la multiplication d’instances de ce service. Ce module est idéal pour importer et exporter des modules d'interface utilisateur et des composants fréquemment utilisés, garantissant ainsi une meilleure lisibilité et maintenabilité du code. Dans notre cas, il permet l’utilisation de pipes, models, Dtos, enums, et d’un SharedComponent.

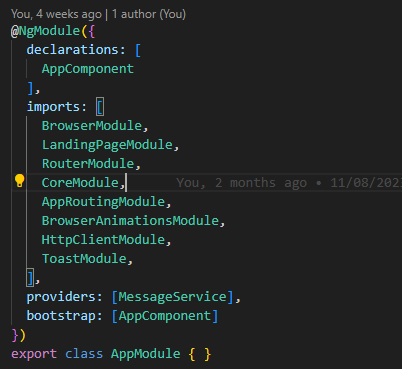
AdminModule : Il s’agit du module le plus sophistiqué, consacré à la gestion de l’application par un administrateur. Il intègre diverses fonctionnalités, allant de la création de questions à la visualisation des résultats d’un candidat, et chaque fonctionnalité repose généralement sur trois composants : un pour l'affichage, un pour les opérations de création, modification et suppression, et un dernier pour l’affichage des résultats.

EntretienModule : Ce module est dédié à la réalisation des entretiens, qu’ils soient sous forme de QCM ou de tests de code.

LandingPageModule : Il sert de module de connexion à l’application.

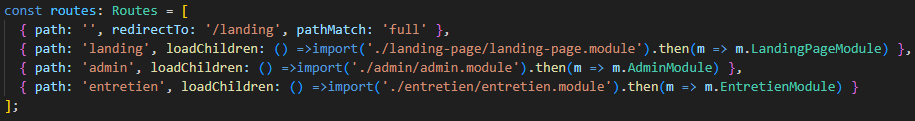
Chacun de ces modules a été pensé et développé avec attention pour répondre aux exigences du projet et assurer une expérience utilisateur optimale, tant pour les candidats que pour les administrateurs de l’application.

Une fois les modules créés, il est impératif de les déclarer dans les imports de l'app.module (à l’exception du SharedModule qui est, lui, importé directement dans chaque composant concerné) afin de les intégrer à l’ensemble de l’application. De surcroît, il est nécessaire de définir pour chaque module une adresse ou 'path', ainsi qu’une méthode de chargement.



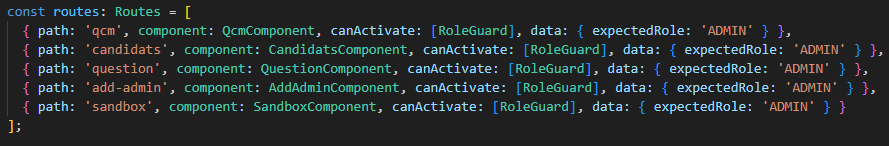
1 - Déclaration des modules dans l'app.module

Pour respecter les bonnes pratiques d'Angular, les différents composants sont chargés en utilisant la technique du Lazy Loading. Angular générant une SPA (Single Page Application), l’intégralité de ses composants sont normalement chargés simultanément. Ceci peut entraîner le chargement de nombreuses bibliothèques ou modules superflus. Le Lazy Loading en Angular permet de contourner ce problème en ne chargeant les composants, modules, ou autres, que lorsqu’ils sont réellement nécessaires, optimisant ainsi la performance de l’application.



2 - Chargement des modules en Lazy Loading

Par la suite, chaque module doit intégrer un routing-module, qui sert à aiguiller vers le bon composant en fonction de la navigation de l'utilisateur. Aucun besoin de recourir de nouveau au Lazy Loading à ce niveau, celui-ci ayant déjà été appliqué au niveau des modules, assurant ainsi une gestion optimale des ressources et une navigation fluide et réactive pour l’utilisateur, qu’il soit un candidat ou un administrateur de l’application.



3 - Déclaration des routes de chaque composant d'un module

#### La page de connexion

Chaque profil doit s'identifier pour accéder à l'application web. L'administrateur, désigné dans la suite de ce dossier comme « admin », se connectera avec son nom d'utilisateur et son mot de passe, tandis que le candidat s'identifiera avec son nom et son prénom. Rappelons que l'application web est destinée, dans un premier temps, à une utilisation interne uniquement. Si l'application devait être déployée sur Internet pour administrer des tests à distance, le moyen d'authentification des candidats serait révisé pour renforcer la sécurité.

À cette fin, un formulaire de type ReactiveForm est employé pour transmettre les données via un service. De plus, nous écoutons la réponse de notre requête qui, en cas de succès, nous transmettra les données de l'utilisateur ainsi qu'un token et un refreshToken que nous enregistrons en session. Les informations de l'utilisateur, et plus spécifiquement son rôle – soit ADMIN, soit CANDIDAT – sont également enregistrées en session. Ces rôles seront utilisés par un guard d'Angular.

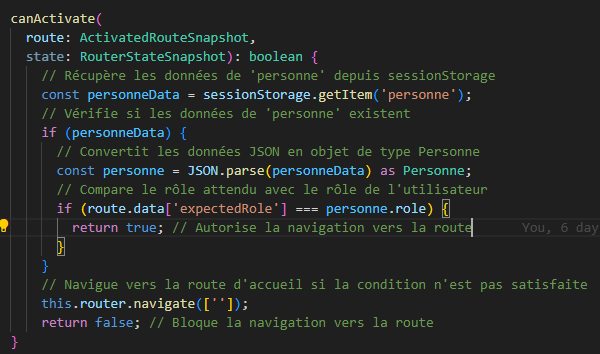


4 - Création des deux formulaires de connexion



5 - Envoi du formulaire de connexion et réaction suivant la réponse reçue

Les Route Guards d'Angular sont des interfaces qui, une fois implémentées, nous permettent de contrôler l'accès à une route selon les conditions spécifiées dans la classe implémentant l'interface. Angular propose différents types de guards tels que CanActivate, CanActivateChild, CanLoad, CanDeactivate et Resolve. Dans notre application, nous mettons en œuvre le CanActivate qui vérifie le rôle de l'utilisateur afin d'autoriser ou non l'accès à une page, garantissant ainsi que chaque utilisateur accède uniquement aux fonctionnalités qui lui sont pertinentes et autorisées.

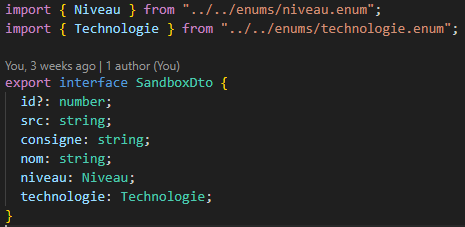


6 - mise en place du Guard basé sur le rôle de l'utilisateur

#### Le share module

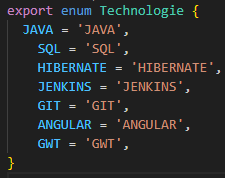
Comme mentionné précédemment, le SharedModule est conçu pour être utilisé à plusieurs endroits dans l'application. C’est pour cela qu'il inclut les DTO (Data Transfer Object), les composants partagés, les énumérations, les modèles et les pipes.

Les DTO : Un Data Transfer Object (DTO) sert à transporter des données entre les processus, facilitant ainsi la communication entre deux systèmes, tels qu'une API et un serveur, tout en préservant la confidentialité des données. Il permet également d'appliquer une logique métier sur un objet sans manipuler directement les entités. Voice un exemple d’un DTO pour une Sandbox avec une valeur optionnelle « id » signalé avec un « ? », différents « String » et deux enumérations « niveau » et « technologie » :



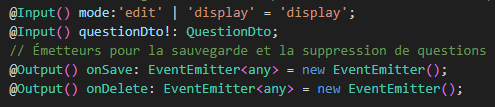
7 - Dto Sandbox

Les énumérations : Elles servent à définir une liste de constantes nommées, permettant ainsi de créer un code plus lisible et documenté pour différents cas spécifiques. TypeScript permet la création d'énumérations numériques et basées sur des chaînes.



8 - Exemple Enum

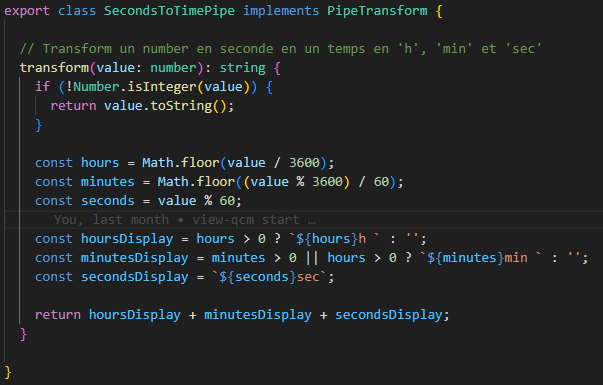
Les composants partagés : Ces composants sont destinés à être invoqués plusieurs fois et à différents endroits dans l'application. Dans cette application, le composant partagé est à la fois un formulaire et un affichage, selon l'état qui lui est attribué. Il devient alors un composant enfant, et la logique métier est appliqué dans ce composant, car, selon l'endroit où le composant partagé est invoqué, la logique métier peut varier. De ce fait, nous utilisons des input et output pour récupérer et envoyer des valeurs vers un composant qui appel ce « share component » comme ceci :



9 - input et output du share-component

Comme on le constate, pour afficher le composant il faut fournir un mode, ‘edit’ pour le formulaire et ‘display’ pour visualiser la question. Ainsi qu’un questionDto pour afficher potentiellement des questions existantes (annexe N° 5).

Les pipes : Un pipe est une fonction utilisée dans les templates pour transformer des données. Angular propose de nombreux pipes intégrés, tels que date pour formater les dates, uppercase pour convertir un texte en majuscules et decimal pour formater les nombres. Ces pipes peuvent être très utiles pour traiter et présenter des données de manière cohérente dans toute l'application. Dans l’exemple ci-dessous, ce pipe permet de récupérer une valeur en nombre et de retourner un temps en ‘h’ : ‘min’ : ‘sec’ ce qui servira lors de l’affichage du chrono du test.



10 - transformation d'un nombre en temps h:min:sec

Ces éléments du SharedModule contribuent à la modularité et à la maintenabilité du code en évitant la redondance et en favorisant la réutilisation des composants et des fonctionnalités à travers l'application.

#### Les behaviorSubject

Un BehaviorSubject fait partie de la bibliothèque RxJS, largement utilisée avec Angular pour gérer les données de manière réactive. Un BehaviorSubject est un type particulier d'Observable, qui permet de représenter une valeur qui change dans le temps, avec la particularité de pouvoir accéder à la dernière valeur émise, même après que la valeur ait été émise.

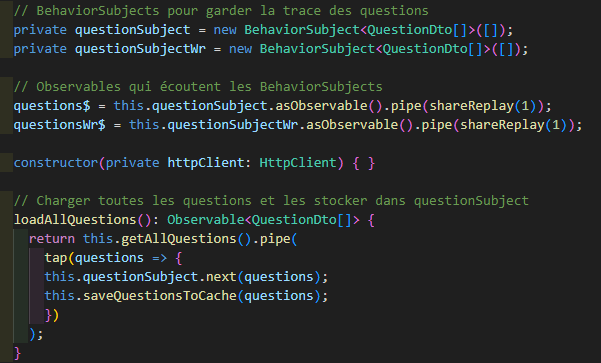
Un BehaviorSubject a deux caractéristiques principales:

* Il a toujours une valeur courante. À l'initialisation, un BehaviorSubject requiert une valeur initiale, qui sera immédiatement disponible comme valeur courante.
* Il permet de multicaster ou de diffuser des valeurs à plusieurs Observers. Cela signifie qu'il peut émettre des valeurs à tous ses consommateurs/abonnés simultanément.

Le BehaviorSubject est particulièrement utile dans les situations suivantes:

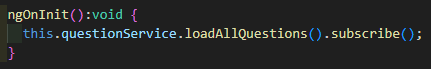
* Partage de Valeurs entre Composants: Lorsque plusieurs composants dépendent d'une valeur commune, par exemple, les détails de l'utilisateur connecté ou un thème global.
* Gestion d'État: Il peut être utilisé pour gérer l'état de l'application de manière réactive, en particulier dans les services Angular.
* Obtention de la Dernière Valeur: Il est pratique lorsque vous avez besoin de la dernière valeur émise par l'Observable.

La mise en place d’un Behavior Subject ce fait en deux partie, la première ce fais dans le service où l’on va récupérer déclarer un Behavior Subject et une méthode afin de récupérer et instancier le behavior Subject.



11 - Behavior Subject

Comme on peux le voir, la méthode prépare une requête mais ne l’exécute pas. Cette méthode ne sera exécutée seulement au chargement des composants qui aurons besoin de ce Behavior Subject. Pour cela il suffira de souscrire à la méthode comme suit :



12 - Souscription à un Behavior Subject

#### Les envois optimistes et pessimistes

Dans le cadre de la gestion de communication entre le client (l’application Angular) et le serveur (L’API Spring Boot), les mises à jour peuvent être gérées de manière optimiste ou pessimiste. Ces méthodes déterminent comment une application gère les actualisations de l'état et répond aux interactions de l'utilisateur, particulièrement lors de la manipulation de données et de l'intégration avec les API.

Mise à Jour Pessimiste :

Dans cette approche, l'application suspend toute mise à jour de l'UI jusqu'à l'obtention d'une confirmation du serveur. Autrement dit, lorsqu'une action (telle que la sauvegarde d'une nouvelle donnée) est initiée, l'application se repose sur la réponse du serveur pour déterminer si l'action a été exécutée avec succès ou non avant de matérialiser tout changement dans l'interface utilisateur.

Avantages :

* Fiabilité :

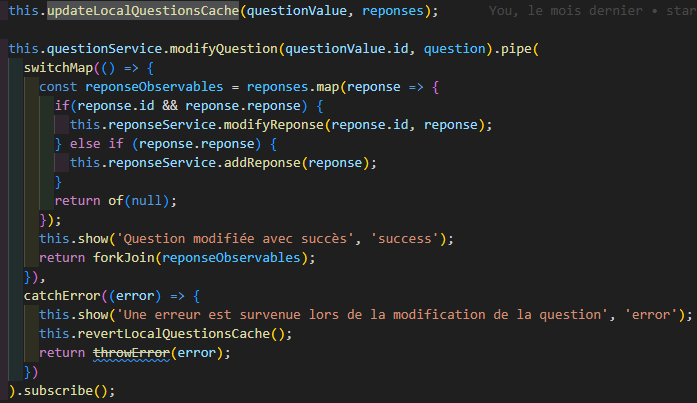
Cette méthode est idéale dans des contextes où la précision et la cohérence des données sont primordiales.

Prévention des Erreurs :

La validation systématique par le serveur minimise le risque d'états incorrects et d'erreurs.

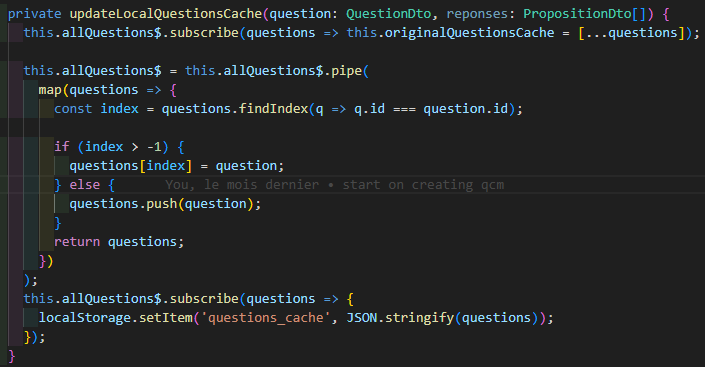
Mise à Jour Optimiste :

À l'opposé, une mise à jour optimiste actualise instantanément l'UI en anticipant une réussite de la requête au serveur. En cas d'échec, l'application doit rectifier et restaurer l'état antérieur. Pour cela, dans l’exemple ci-dessous la méthode permet de mettre à jour une question et ses réponses :



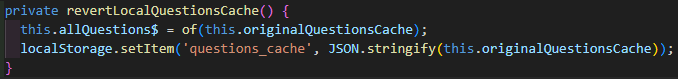
13 - Méthode optimiste

La première chose que fais la méthode est un appel à la méthode updateLocalQuestionsCache qui prend la question et les réponses comme paramètre et permet de les rajouter au Behavior Subject comme suit :



14 - mise à jour du Behavior Subject

Nous pouvons aussi observer qu’en cas d’erreur, une autre méthode est appelée afin de faire marche arrière et de retirer la question et ses réponses du Behavior Subject.



15 - Annulation de la mise à jour du Behavior Subject

Avantages :

* Réactivité :

Les utilisateurs bénéficient d'une expérience plus fluide et réactive, car l'application semble plus rapide.

* Économie de Temps :

La confirmation du serveur n'étant pas requise pour visualiser les modifications, le temps d'attente de l'utilisateur est réduit.

En résumé, le choix entre une mise à jour optimiste ou pessimiste doit être guidé par les exigences spécifiques du projet, en équilibrant soigneusement la fiabilité des données avec une expérience utilisateur optimale.

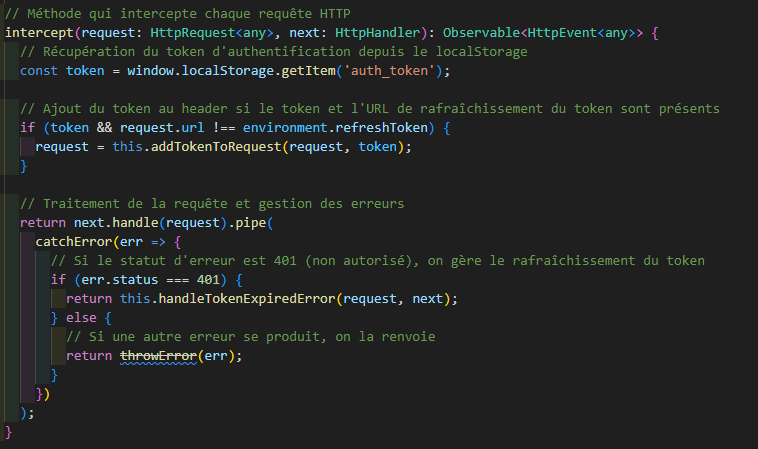
En somme, l'envoi optimiste permet d'actualiser l'affichage côté client de manière plus rapide, effectuant le traitement subséquent. Cela suppose également une certitude quant au succès de la requête pour éviter toute régression. Pour ce faire, cette méthode sera privilégiée pour des requêtes associées à des formulaires qui contraignent au maximum l'utilisateur, dans le but de minimiser les erreurs et les retours en arrière. De surcroît, son utilité est décuplée lorsqu'elle est appliquée sur des Behavior Subject renfermant une quantité importante de données, car sa mise à jour s'effectuera avec plus de rapidité.

#### L’intercepteur

Dans une application Angular, un intercepteur (interceptor) permet de traiter les requêtes HTTP avant leur envoi ou après leur réception. L’un des usages les plus courants de cet outil concerne l’ajout automatique de tokens, éléments clés de l’authentification.

Fonctionnement :

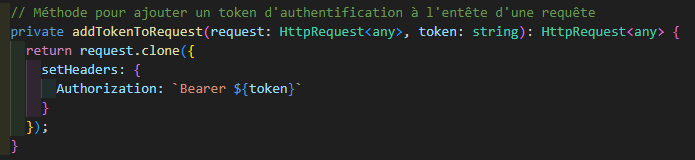
Lorsqu'une requête HTTP est initiée, l'intercepteur Angular permet d'injecter des informations, telles qu'un token, dans les en-têtes de la requête. Ce token est souvent utilisé pour authentifier l'utilisateur auprès du serveur. Si le token est invalide ou expiré, un refresh token peut être utilisé pour en obtenir un nouveau sans nécessiter une nouvelle connexion de l'utilisateur.



16 - Interception de chaque requête Http

Ajout du Token :

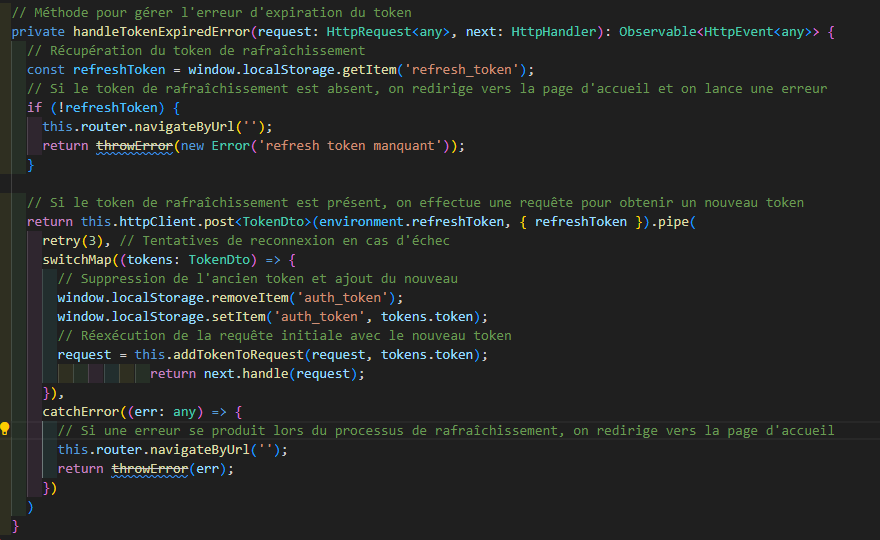
Chaque fois qu'une requête est effectuée vers le serveur, l'intercepteur ajoute le token d'authentification à l'en-tête de la requête. Ceci assure que chaque requête est authentifiée, permettant ainsi une communication sécurisée entre le client et le serveur.



17 - Ajout du Token à la requête

Gestion du Refresh Token :

En cas d'expiration du token, l'intercepteur Angular peut intercepter la réponse d'erreur, utiliser le refresh token pour obtenir un nouveau token d'accès et réexécuter la requête initiale, le tout de manière transparente pour l'utilisateur. Ceci permet de maintenir l’authentification de l’utilisateur et d’assurer une expérience utilisateur fluide, même sur des sessions prolongées.



18 - Réaction en cas d'erreur 401

Sécurité :

L’ajout automatique de tokens renforce la sécurité de l’application en assurant que toutes les requêtes sont authentifiées et autorisées.

Efficacité :

La gestion automatique des tokens et des refresh tokens réduit la complexité du code en centralisant la logique d’authentification.

Expérience Utilisateur :

La possibilité de rafraîchir les tokens de manière transparente permet d’éviter des déconnexions inutiles et contribue à une expérience utilisateur plus agréable et sans interruption.

Maintenabilité :

Centraliser la gestion des tokens dans un intercepteur permet de faciliter la maintenance du code, en évitant la répétition de logique à travers différents composants et services.

En conclusion, les intercepteurs Angular, en manipulant les tokens et les refresh tokens, jouent un rôle crucial pour sécuriser et optimiser les applications web modernes, tout en améliorant la qualité globale de l'expérience utilisateur.

### Développer la partie backend d’une interface utilisateur web

Pour la partie backend de l'application, il a été décidé d'utiliser Spring Boot en conjonction avec Java et Maven. Spring Boot offre de nombreux avantages pour le développement d'APIs Java. Il simplifie considérablement la configuration, réduit le temps de mise en place et facilite l'intégration de divers composants, grâce à sa convention en surcouche. Il offre également une gamme vaste de plugins, permettant une personnalisation accrue et une adaptabilité aux besoins spécifiques du projet. L'utilisation de Java assure la portabilité, la robustesse et une performance optimale du système, et permet de bénéficier d'un écosystème riche et d'une communauté active. Maven, quant à lui, gère efficacement la gestion de dépendances et le cycle de vie du projet, assurant ainsi une uniformité et une stabilité du build, ce qui est essentiel dans un environnement d'entreprise où la cohérence et la maintenance sont cruciales. Ensemble, ces technologies créent une fondation solide et flexible pour le développement d'applications backend robustes et évolutives.

#### Configuration de Spring Boot

Pour structurer et télécharger un squelette de l’API, j’ai utilisé l’application web Spring Initializr (annexe N°6). Pour les versions de Java et de Spring Boot, nous avons opté pour les dernières versions LTS (Long Term Support), car mon projet est un projet transverse qui sera encapsulé dans une image Docker. Concernant le choix des dépendances, plusieurs se sont avérées indispensables au projet.

* Spring-boot-starter-web est crucial pour construire des applications web, y compris les applications RESTful, en simplifiant les processus d’inclusion de dépendances et de configuration.
* Spring-boot-starter-data-jpa facilite l’interaction avec les bases de données par le biais de Java Persistence API, automatisant ainsi l'accès et la gestion des données.
* Spring-boot-starter-tomcat permet d’intégrer le serveur Tomcat pour déployer des applications web avec Spring Boot, offrant un serveur web intégré pour l'application.
* Spring-boot-starter-security qui, bien que non activé initialement pour simplifier la mise en place de l’application, sert à intégrer des mécanismes de sécurité comme l'authentification et l'autorisation.

Enfin, j’ai également utilisé Lombok pour fluidifier le développement en réduisant la verbosité du code Java.

Le fichier pom.xml est un élément essentiel dans les projets Maven. Il définit le projet et ses configurations, telles que les dépendances, les plugins, les versions et d'autres paramètres du cycle de vie du projet. C'est grâce à ce fichier que Maven sait comment construire le projet et gérer les dépendances. C’est donc directement dans ce fichier que l’on va pouvoir rajouter les différentes dépendances qui vont être utilisé tout au long de ce projet.

D'autre part, le fichier application.properties est un fichier de configuration qui permet de définir les propriétés de l'application. Il va contenir des informations telles que les paramètres de connexion à la base de données, les configurations du serveur, les chemins des fichiers, la clés secrète du token (que nous verrons dans le chapitre sur la sécurité), etc. C’est aussi dans ce fichier que l’on pourra configurer les différents logs qui servirons au court du développement de l’application. Dans Spring Boot, ce fichier est utilisé pour surcharger les valeurs par défaut de configuration afin de personnaliser le comportement de l'application selon les besoins spécifiques du projet.

Ces deux fichiers sont donc indispensables pour définir le comportement et les configurations de l'application Spring Boot, l'un au niveau du build et de la gestion des dépendances, et l'autre au niveau de l'exécution et du comportement en runtime de l'application.

#### Modèle-Vue-Contrôleur

Pour respecter les bonnes pratiques d'une API RESTful, j'ai structuré l'application en suivant le modèle MVC (Modèle-Vue-Contrôleur). Le modèle MVC est un patron de conception qui sépare une application en trois composants interconnectés :

**Modèle** : Gère la logique métier, l'accès et la manipulation des données.

**Vue** : Présente les données à l'utilisateur sous une forme appropriée.

**Contrôleur** : Interagit à la fois avec le modèle et la vue, recevant les entrées de l'utilisateur et effectuant les appels correspondants au modèle.

Dans ce contexte, différents packages ont été créés pour isoler les responsabilités, à savoir :

Controllers : Ce package est lié, comme son nom l’indique à la composante Contrôleur du MVC. Il contient les classes qui reçoivent et traitent les requêtes HTTP, délèguent la logique métier et renvoient les réponses HTTP.

DTOs (Data Transfer Objects) : Ce sont, comme dans la partie Angular, des objets qui transportent des données entre les processus. Ils sont généralement utilisés pour regrouper les attributs que vous voulez partager avec l'interface utilisateur ou l'API externe. De plus, ils permettent d’appliquer des logique métier et être donc diffèrent des modèles.

Mappers : Ils sont responsables de la conversion des objets, comme la transformation des entités en DTOs et inversement, afin de garantir l'indépendance entre les modèles de données internes et la représentation externe des données.

Models : Ils correspondent au composant Modèle dans le pattern MVC. Ils représentent les structures de données de l'application et contiennent la logique pour accéder à la base de données.

Repositories : Ils sont utilisés pour gérer la communication entre l'application et la base de données. Ils jouent un rôle crucial dans la manipulation des données du Modèle.

Services : Ils contiennent la logique métier et les règles de l'application. Les services interagissent avec les modèles et les repositories pour traiter les données et renvoyer les résultats au contrôleur.

Chacun de ces packages a une fonction distincte, mais tous interagissent de manière cohérente pour former une application structurée, conformément aux principes du modèle MVC, offrant ainsi une base solide pour le développement d'une API RESTful.

#### Spring security

Dans le cadre du développement de notre application, une attention particulière a été accordée à la sécurité. Ainsi, il a été décidé d’intégrer Spring Security, une solution éprouvée et robuste, pour gérer l’authentification et les autorisations. Cette framework de sécurité veillera à ce que seul l’administrateur puisse s’authentifier via un nom et un mot de passe, tandis que le candidat le fera par le biais de son nom et prénom. En matière d'autorisation, l'approche adoptée repose sur l'utilisation de tokens. Pour mettre en œuvre cette stratégie, la dépendance java-jwt a été ajoutée au projet, offrant ainsi une couche supplémentaire de sécurité à travers la gestion des tokens JWT (JSON Web Token). Ce choix stratégique permet non seulement de sécuriser l’accès aux ressources de l’application mais sert également de garantie pour maintenir l'intégrité et la confidentialité des données échangées.

La méthode securityFilterChain(HttpSecurity http) (annexe N°7) est une illustration concrète de l'application de cette stratégie de sécurité. Elle est essentielle pour capter toutes les URL entrantes et pour définir les URL qui doivent être exemptées du processus d'authentification comme les URL liées à la connexion et à la régénération de tokens, car elles peuvent être accompagnées d'un token invalide ou dépourvues de token. Pour cela, trois instances de MvcRequestMatcher sont définies pour correspondre aux URL /login/admin, /login/candidat, et /login/refresh-token, toutes associées à la méthode HTTP POST. Chaque matcher est configuré avec HandlerMappingIntrospector pour obtenir des informations détaillées sur les mappages de gestionnaire dans l'application. Un OrRequestMatcher est ensuite construit avec la liste de ces matchers pour autoriser toute requête correspondant à l'un d'eux sans authentification.

Le reste de la configuration de HttpSecurity inclut l'établissement d'un authenticationEntryPoint personnalisé qui sert principalement à gérer la façon dont votre application va répondre lorsqu’un utilisateur non authentifié tente d'accéder à une ressource sécurisée, l'intégration d'un filtre JwtAuthFilter personnalisé qui sert à récupérer et vérifier le token, et la configuration de la gestion de session pour être STATELESS ce qui implique que le serveur ne va pas créer ou utiliser de session entre les requêtes, et ainsi chaque requête sera isolée et indépendante.

#### Authentification et Génération de Tokens

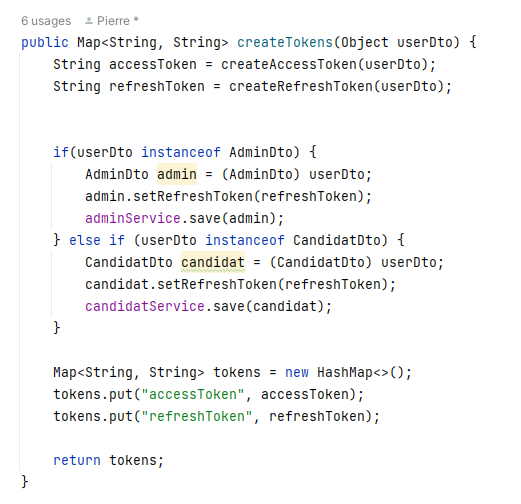
Dans le cadre de notre projet, nous avons élaboré un système d'authentification robuste et sécurisé. Les administrateurs s'authentifient à l'aide de leurs noms et mots de passe, tandis que les candidats utilisent leurs noms et prénoms. À la suite d'une authentification réussie, un token d'accès et un refresh token sont générés et attribués à l'utilisateur, assurant ainsi la sécurisation et la fluidité des interactions ultérieures avec l'application.

Le processus d'authentification inclut également un mécanisme de chiffrement des mots de passe grâce à l'utilisation de PasswordEncoder, renforçant ainsi la protection des données sensibles des utilisateurs.

#### Méthodologie de Création des Tokens

La génération des tokens se décompose en trois phases essentielles, orchestrées par la méthode createTokens. Cette méthode reçoit en paramètre un objet UserDto, pouvant être un AdminDto ou un CandidatDto, et procède à l'invocation de deux méthodes subalternes responsables de la création de l'access token et du refresh token.

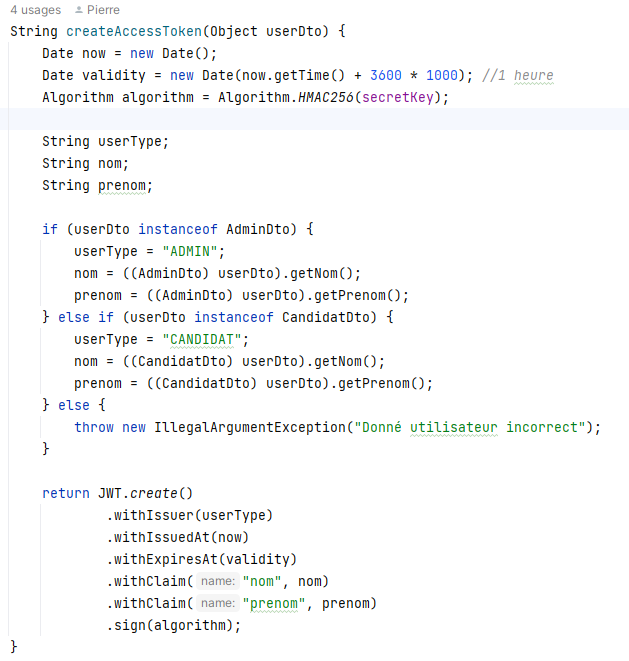
Le refresh token produit est ensuite associé à l'instance de AdminDto ou CandidatDto correspondante et est persisté dans notre base de données, permettant une gestion optimisée des sessions utilisateurs.



19 - Création des tokens

#### Elaboration de l'Access Token

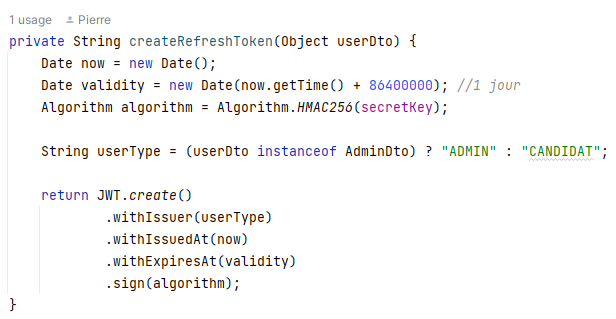
La méthode createAccessToken engendre l'access token, un composant crucial pour l'intégrité et la sécurisation de notre application. Ce token, doté d'une durée de vie d'une heure, encapsule des informations utilisateur telles que le nom et le prénom et est signé via un algorithme HMAC256, assurant ainsi l'authenticité et la confidentialité des informations transmises.



20 - Création de l'access Token

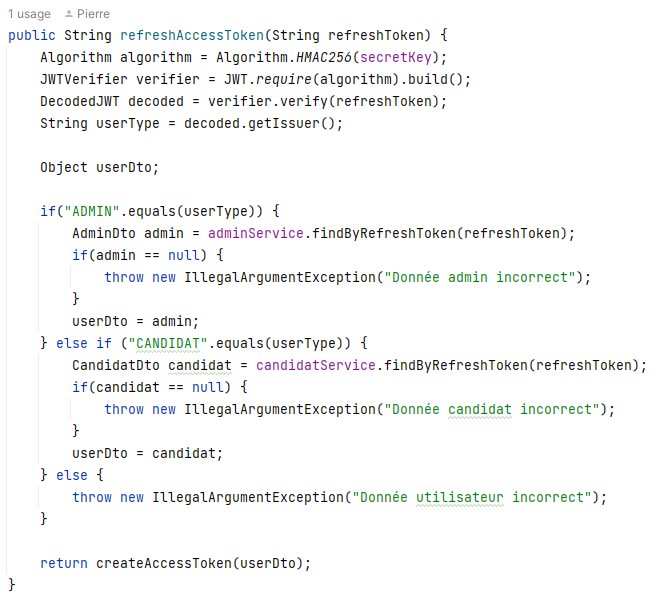
#### Conception du Refresh Token

En parallèle, la méthode createRefreshToken se charge de formuler un refresh token. Ce token, bien que disposant d'une durée de validité supérieure, s'échelonnant sur une journée, embarque moins d'informations relatives à l'utilisateur pour minimiser d'éventuelles expositions d'informations sensibles.



21 - Création d'un refresh token

Le rôle de ce refresh token est crucial, il permet de réduire les demandes répétitives d'authentification en facilitant la régénération de nouveaux access tokens, sans requérir de nouvelles saisies des identifiants de l'utilisateur. Comme vue précédemment avec l’intercepteur d’Angular, lorsque le token est périmé, l’API va renvoyer une erreur 401 ce qui va déclencher la création d’un nouveau token en utilisant le refresh token. La méthode refreshAccessToken prend donc en argument le refresh token, retrouve l’utilisateur associé à ce refresh token et génère un nouveau token.



22 - Création d'un nouveau token à partir d'un refresh token

Ces mécanismes de création de tokens, en conjugaison avec les méthodes d'authentification et de gestion de tokens, consolident la structure de sécurité de notre application. Ils garantissent une interaction authentifiée et sécurisée, tout en préservant une expérience utilisateur cohérente et sécurisée.

#### Mécanisme de vérification du token

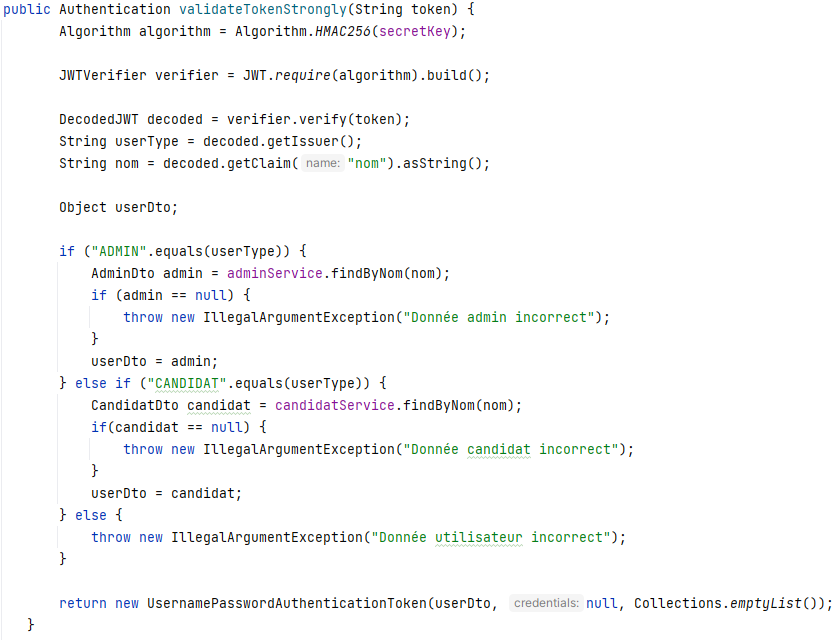
Une fois le token généré, chaque requête provenant du côté client est interceptée par l'intercepteur d'Angular, lequel, à travers ledit intercepteur, va attacher le token dans l'en-tête de la requête. Cette requête est ensuite interceptée par la méthode doFilterInternal, pour une vérification minutieuse de la présence du token.



23- la méthode doFilterInternal pour vérifier la présence token

Si un token est effectivement présent, deux méthodes supplémentaires sont déployées pour valider ce token, en fonction du type de requête : une méthode spécifique pour les requêtes GET et une autre pour tous les autres types de requêtes.

Pour les requêtes de type GET, la vérification du token se centre principalement sur sa validité afin d'approuver la requête. En revanche, pour toutes les autres requêtes, une étape de vérification supplémentaire est nécessaire. Il est impératif de confirmer non seulement la validité du token, mais également l'existence de l'utilisateur en base de données avant d'approuver la requête. Cette stratégie permet d'assurer une sécurité renforcée et de prévenir d'éventuelles requêtes malveillantes ou non autorisées, garantissant ainsi l'intégrité et la fiabilité de l'application.



24 - méthode pour valider le token

#### Gestion des CORS origin

Les CORS (Cross-Origin Resource Sharing) sont une politique de sécurité implémentée par les navigateurs web pour maîtriser les requêtes HTTP effectuées entre différents domaines. Cette politique vise à protéger l'utilisateur final contre les requêtes malveillantes qui pourraient être effectuées à son insu. En utilisant des en-têtes spécifiques, CORS permet aux serveurs d'indiquer quelles origines sont autorisées à lire les informations de la ressource. Ainsi, CORS joue un rôle crucial dans la sécurisation des communications entre le navigateur et le serveur, en n'autorisant les échanges que lorsque les critères de sécurité sont satisfaisants.

Pour ce faire, j'ai mis en place une méthode destinée à autoriser toutes sortes de requêtes provenant du localhost de mon application Angular. Ultérieurement, il s'agira de l'adresse de l'application Angular située sur le serveur. Cette méthode est configurée pour accepter les en-têtes portant le titre "Authorization", nécessaire pour le token, et "Content\_Type", pour le contenu. Elle a également été adaptée pour accepter les requêtes "GET", "POST", "PUT", et "DELETE". Cette approche assure une interaction souple et sécurisée entre différents services et composants de l'application, tout en préservant l'intégrité et la confidentialité des échanges de données. Toutes les autres requêtes sortant de ce cadre seront automatiquement refusé.

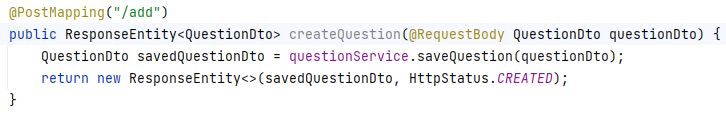


25 - Gestion des CORS Origin

#### CRUD

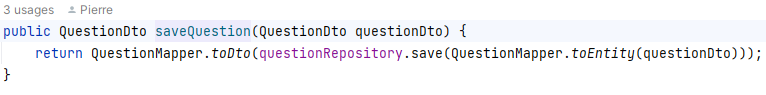
Suite à la configuration de l'authentification, de l'autorisation, et du CORS origin, l'établissement de requêtes entre mon application Angular et mon API est désormais opérationnel. Afin d'adhérer aux meilleures pratiques et au modèle MVC, après les vérifications de sécurité, la requête est dirigée vers le contrôleur. Ce dernier joue le rôle de répartiteur, acheminant la demande vers un ou plusieurs services où une logique métier spécifique est appliqué avant d'envoyer une réponse en retour.

Dans l'exemple traité ci-après, une requête d'ajout de question est reçue. Le corps de cette requête contient un objet JSON incorporant les éléments d'une question, ces derniers correspondent à un QuestionDto. Si la requête contient bel et bien un objet QuestionDto conforme, l'intégralité est transmise au QuestionService où la logique métier associée sera mis en œuvre.



26 - controller pour l'ajout d'une question

Dans ce cas d'usage particulier, le DTO ne subit pas de modifications ; il est converti en entité au moyen d'un mapper (Annexe N°8), puis persisté dans la base de données. Cette approche méticuleuse permet de maintenir une architecture propre et structurée, facilitant ainsi l'intégration, la compréhension, et l'évolutivité du code, tout en garantissant une gestion efficace et sécurisée des données. Les morceaux de code associés au contrôleur, au service et au mapper illustrent de manière explicite la mécanique et la fluidité des opérations effectuées.



27 - persistance des données

### Développer des composants d’accès aux données

Spring fournit un panel de solutions afin d'optimiser la gestion des composants d'accès aux données dans une application Java. JPA (Java Persistence API) en est un exemple phare. En conjonction avec des fournisseurs de persistance tels qu'Hibernate, JPA permet une abstraction efficiente de la couche de base de données. Dans cette configuration, les "Repositories" sont des acteurs clés, offrant une interface sophistiquée pour les opérations CRUD (Create, Read, Update, Delete) (Annexe N°9).

Spring Boot, de son côté, offre un mécanisme de configuration centralisé et aisée à travers le fichier application.properties. Pour notre projet, ce fichier héberge des paramètres cruciaux pour établir la connexion à la base de données PostgreSQL, incluant l'URL, le nom d'utilisateur, et le mot de passe, et renferme également divers attributs déterminants pour configurer Hibernate. Ceux-ci spécifient le dialecte SQL à employer et les réglages relatifs à la mise à jour automatique du schéma de la base de données.

Le fichier application.properties de notre projet est configuré comme suit :



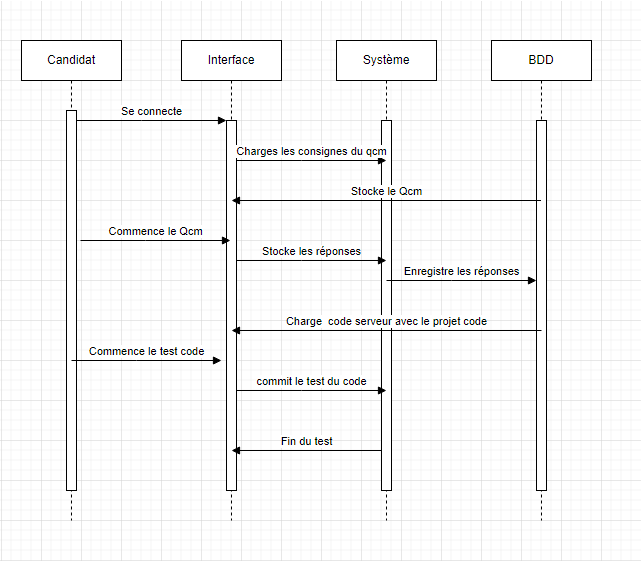
28 - configuration base de données application properties

L'intérêt majeur de cette centralisation réside dans la facilitation de la gestion et de l'évolution de l'application, permettant une maintenance plus agile et une adaptation rapide aux modifications de l'environnement de l'application, comme par exemple lors du déploiement de l’application sur les serveurs .

## Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

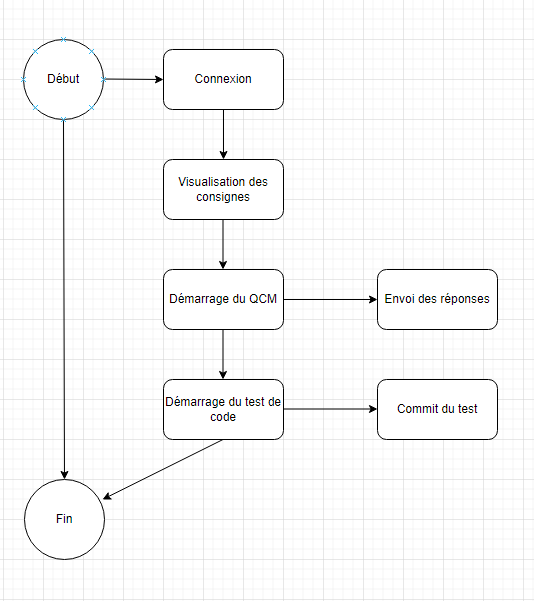
### Concevoir une base de données

Avant de concevoir une base de données, il est crucial de conceptualiser et de visualiser le flux de fonctionnalités et d'interactions au sein du système. Pour ce faire, des outils tels que les diagrammes de séquence et d'activité sont indispensables.



29 - Diagramme de séquence du candidat

Les diagrammes de séquence, illustrés ci-dessus pour un candidat, détaillent les interactions entre différents objets dans un système en mettant l'accent sur l'ordre des interactions. Ils offrent une vue claire et détaillée des fonctionnalités et des responsabilités de chaque composant, permettant ainsi de comprendre précisément le flux de processus et les interactions au sein du système.



30 - Diagramme d'activité Candidat

D'autre part, les diagrammes d'activité, comme montré ci-dessus pour un candidat, mettent en lumière le flux de travail au sein du système. Ils illustrent les étapes et les activités réalisées par les utilisateurs ou les systèmes, les conditions et les décisions qui structurent le parcours de l'utilisateur.

En illustrant les étapes, les activités, et les décisions prises par un administrateur, ces diagrammes jouent un rôle crucial dans la transmission d'une compréhension unifiée des exigences et comportements souhaités de l'application entre les différentes parties prenantes du projet.

Ces diagrammes préparent le terrain pour la conception de la base de données en offrant une fondation solide pour élaborer une architecture cohérente et réaliser une implémentation sans faille.

Dans le cadre de ce projet, la conceptualisation et la réalisation de diagrammes, notamment les diagrammes de séquence et d'activité, étaient indispensables avant de s'avancer dans la conception de la base de données. Ils permettent de visualiser les interactions et le flux de travail entre les différents utilisateurs du système, qu'il s'agisse des candidats ou des administrateurs, et facilitent une compréhension holistique du comportement attendu et des exigences fonctionnelles de l'application. Les diagrammes d'activité, tels qu'illustrés pour le parcours du candidat, sont particulièrement utiles pour dépeindre de manière structurée les différents scénarios d'utilisation, les conditions et les décisions qui informent l'expérience utilisateur au sein du système.

En regardant à travers le prisme de ces diagrammes, les équipes projet peuvent acquérir une vision commune et unifiée des fonctionnalités requises et des interactions entre les divers composants du système, jetant ainsi les bases d'une conception de base de données rationnelle et cohérente.

Intégrer ces éléments préliminaires avec une structure de base de données bien pensée était donc crucial. Pour ce faire, après une série d'échanges approfondis avec l'équipe projet et une analyse détaillée du cahier des charges, j'ai opté pour l'utilisation de Looping pour construire cette base de données. Cela a été réalisé en dépit de l'absence de certaines fonctionnalités dans Looping, telles que l'intégration d'énumérations ou de classes abstraites, qui auraient pu optimiser davantage la représentation visuelle de la base de données (Annexe N°10).

La nécessité de gérer deux types d'utilisateurs, à savoir les administrateurs et les candidats, a engendré la création d'une classe abstraite, évitant ainsi la redondance et regroupant les champs communs entre ces deux entités distinctes. Les classes administrateur et candidat en héritent, promouvant la modularité et facilitant les modifications futures, tout en maintenant une cohérence structurelle.

En outre, pour satisfaire à des relations complexes, des tables de liaison ont été incorporées pour établir des relations « Many-to-Many » entre divers éléments comme les candidats et les propositions ainsi qu'entre les QCM et les questions, avec des champs supplémentaires pour assurer une flexibilité et une organisation optimales.

Ainsi, cette structure de base de données robuste et rationalisée sert de fondement à l'application, permettant des interactions fluides entre les différentes entités, et garantissant l'intégrité et la durabilité des données manipulées. En alignant rigoureusement la conception de la base de données sur les besoins définis par l'équipe et le cahier des charges, un environnement de données efficace et évolutif a été mis en place, prêt à supporter les évolutions et adaptations futures du projet.

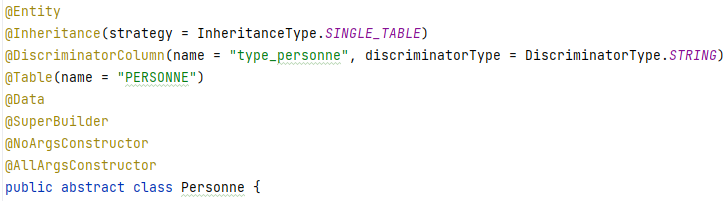
### Mettre en place une base de données

Dans le développement d'applications avec Spring Boot, la mise en place d'une base de données est une étape cruciale, et pour ce faire, l'utilisation de Java Persistence API (JPA) et Hibernate s'avère souvent indispensable. JPA est une interface de programmation standard qui sert de spécification pour la gestion relationnelle des données dans les applications Java. Elle permet de définir la manière dont les données doivent être persistées, gérées et récupérées, offrant une abstraction élevée par rapport à la base de données sous-jacente.

Hibernate, quant à lui, est l'un des fournisseurs de persistance les plus populaires pour JPA. Il implémente les spécifications de JPA et offre des fonctionnalités supplémentaires, tels que la gestion de cache de second niveau et des mécanismes de requêtage avancés, qui optimisent les performances de l'application.

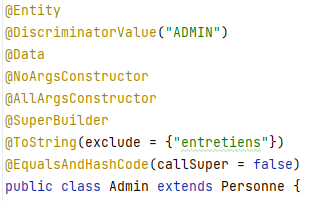
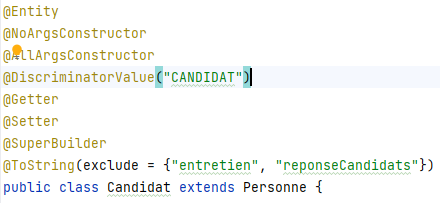
En utilisant JPA et Hibernate dans un contexte Spring Boot, les développeurs peuvent bénéficier d'une intégration fluide et d'une configuration simplifiée. Cette combinaison permet de manipuler les entités de base de données de manière intuitive et orientée objet, en minimisant le besoin d'écrire des requêtes SQL explicites. Elle encourage ainsi le développement rapide, tout en garantissant la qualité et la robustesse de l'accès aux données dans l'application, rendant la gestion de la persistance des données plus flexible et moins sujette à erreurs.

La mise en place d'une architecture orientée objet efficace est cruciale dans un tel environnement. Une classe abstraite comme Personne a été définie dans notre cas pour représenter les éléments communs à différentes entités, les class Admin et Candidat qui sont en réalité des spécialisations de la classe Personne. Cette approche d'utilisation de classes abstraites et d'héritage, combinée à l'abstraction des données offerte par JPA et Hibernate, permet une représentation cohérente et logique du modèle de données.



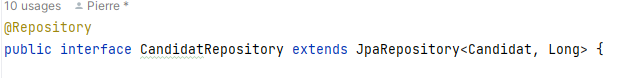
31 - class abstraite Personne

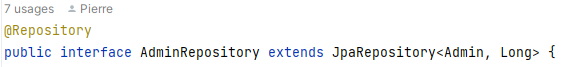
Les classes Admin et Candidat, en étendant la classe Personne, héritent de ses attributs et méthodes, mais peuvent également déclarer des propriétés et des comportements qui leur sont propres. La stratégie d'héritage SINGLE\_TABLE implique que toutes les entités de cette hiérarchie sont représentées dans une seule table, et la colonne type\_personne sert à discriminer entre les différentes instances de sous-classes.

32 - class admin qui étend Personne class Candidat qui étend Personne

Pour interagir avec la base de données, des repositories JPA, tels que AdminRepository et CandidatRepository, sont définis. Ces repositories fournissent une interface pour effectuer des opérations CRUD sur les entités correspondantes et définissent des méthodes pour effectuer des requêtes sur la base de données selon divers critères.





L'utilisation de classes abstraites et de l'héritage, en harmonie avec JPA et Hibernate dans Spring Boot, permet de créer un modèle de données structuré et extensible. Cela facilite le développement et la maintenance du code tout en optimisant l'interaction avec la base de données grâce à une interface de repository intuitive, minimisant ainsi les erreurs potentielles et favorisant un développement plus rapide et plus robuste.

## Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

### Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement

Klaxoon

### Concevoir une application

Diagram de séquence …

### Développer des composants métier

Détail d’un service

### Construire une application organisée en couches

MVC

### Préparer et exécuter les plans de tests d’une application

Junit

### Préparer et exécuter le déploiement d’une application

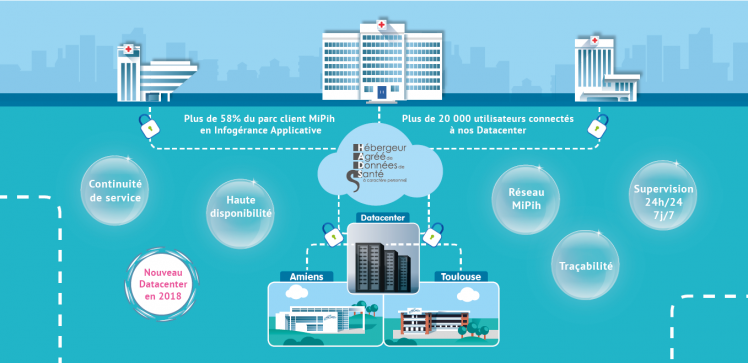
Docker

# Annexe

Annexe 01 : Bornes admissions patients



Annexe 02 : Datacenter du MiPih



Annexe 03 : Les principes SOLID

Les principes SOLID sont un ensemble de cinq principes de conception de logiciels destinés à rendre les programmes plus efficaces, faciles à comprendre et à maintenir. Ils sont largement reconnus dans le domaine de la programmation orientée objet. Le terme SOLID est un acronyme, où chaque lettre correspond à l'un des principes :

#### S - Single Responsibility Principle (Principe de Responsabilité Unique)

Ce principe stipule qu'une classe doit avoir une seule responsabilité ou raison de changer. En d'autres termes, une classe ne doit effectuer qu'une seule "tâche" ou avoir une seule fonctionnalité. Cela simplifie la maintenance et facilite la compréhension du code.

#### O - Open/Closed Principle (Principe Ouvert/Fermé)

Ce principe suggère que le code doit être ouvert à l'extension mais fermé à la modification. Cela signifie que vous pouvez ajouter de nouvelles fonctionnalités sans changer le code existant, généralement en utilisant l'héritage et/ou l'implémentation d'interfaces.

#### L - Liskov Substitution Principle (Principe de Substitution de Liskov)

Ce principe énonce que les objets d'une classe mère doivent être en mesure d'être remplacés par des objets d'une classe fille sans affecter la correction du programme. Il s'assure que la classe fille respecte le contrat établi par la classe mère.

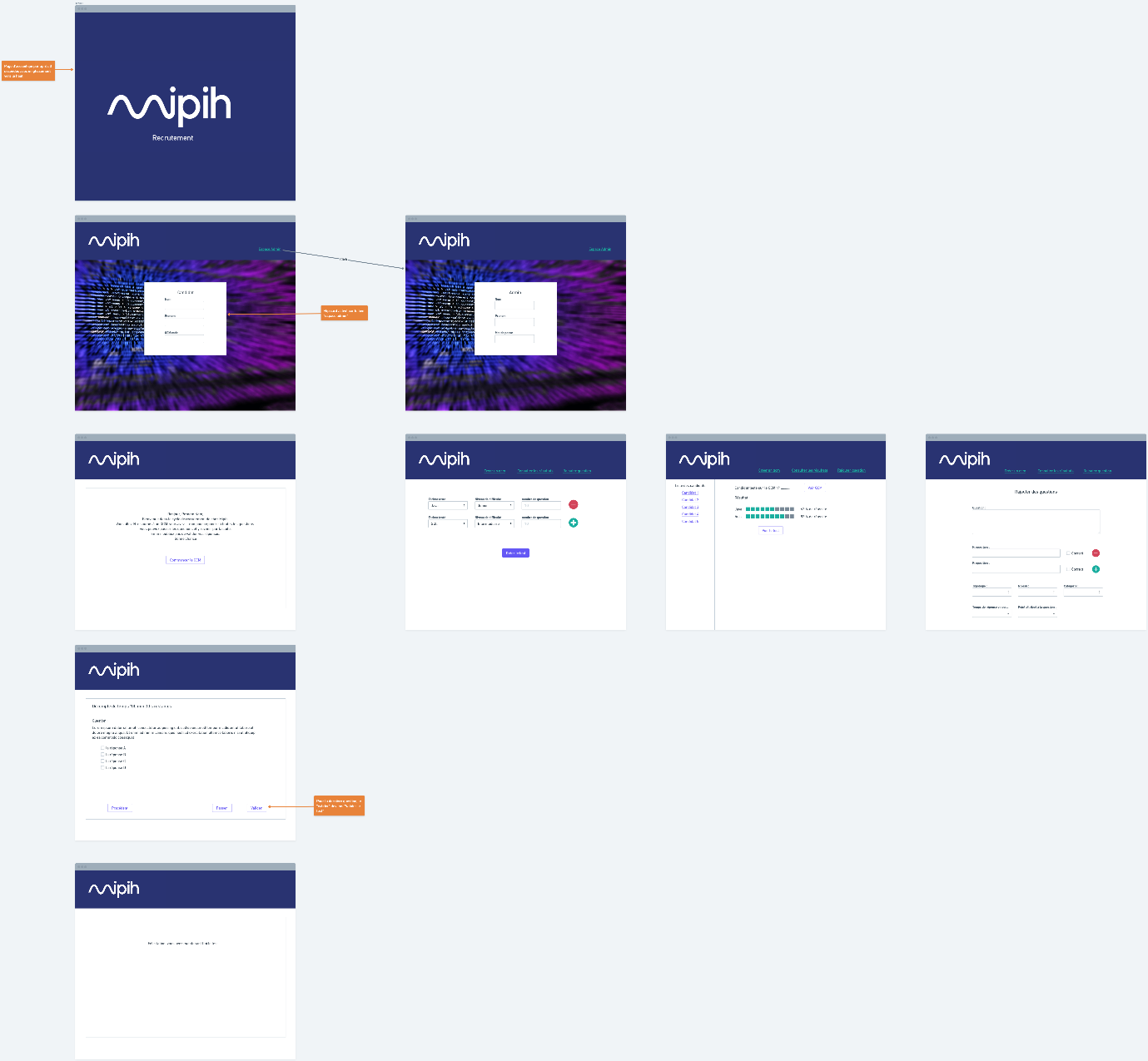
#### I - Interface Segregation Principle (Principe de Ségrégation des Interfaces)

Ce principe suggère que les classes ne doivent pas être forcées de dépendre d'interfaces qu'elles n'utilisent pas. En d'autres termes, une interface devrait avoir uniquement les méthodes qui sont spécifiquement nécessaires pour la classe qui l'implémente.

#### D - Dependency Inversion Principle (Principe d'Inversion des Dépendances)

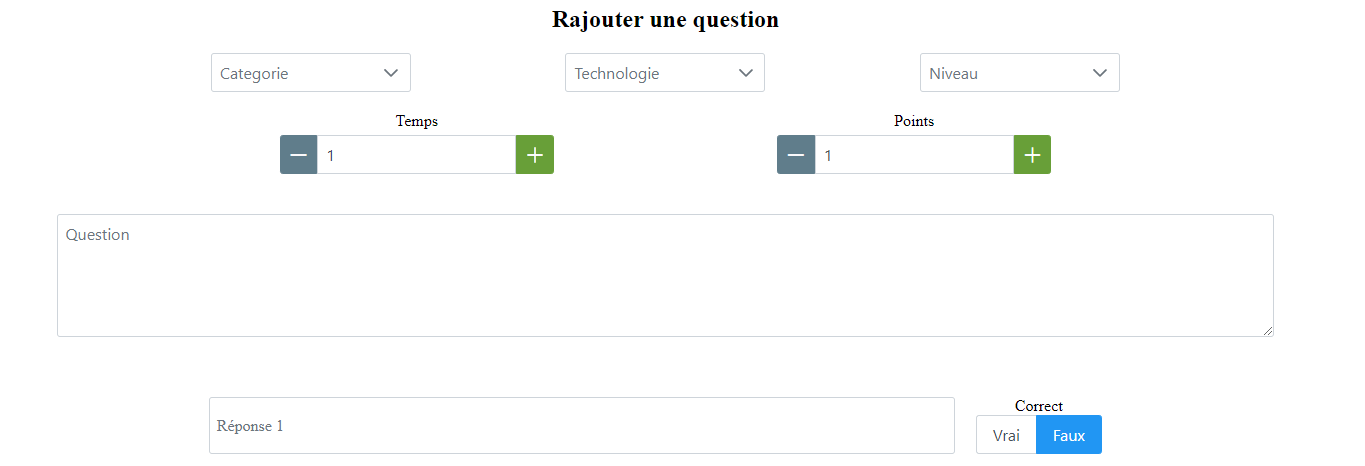
Ce principe encourage à dépendre des abstractions plutôt que des implémentations concrètes. Cela favorise le découplage entre différentes parties d'un système et facilite la maintenance et les tests.

Annexe 04 : La maquette

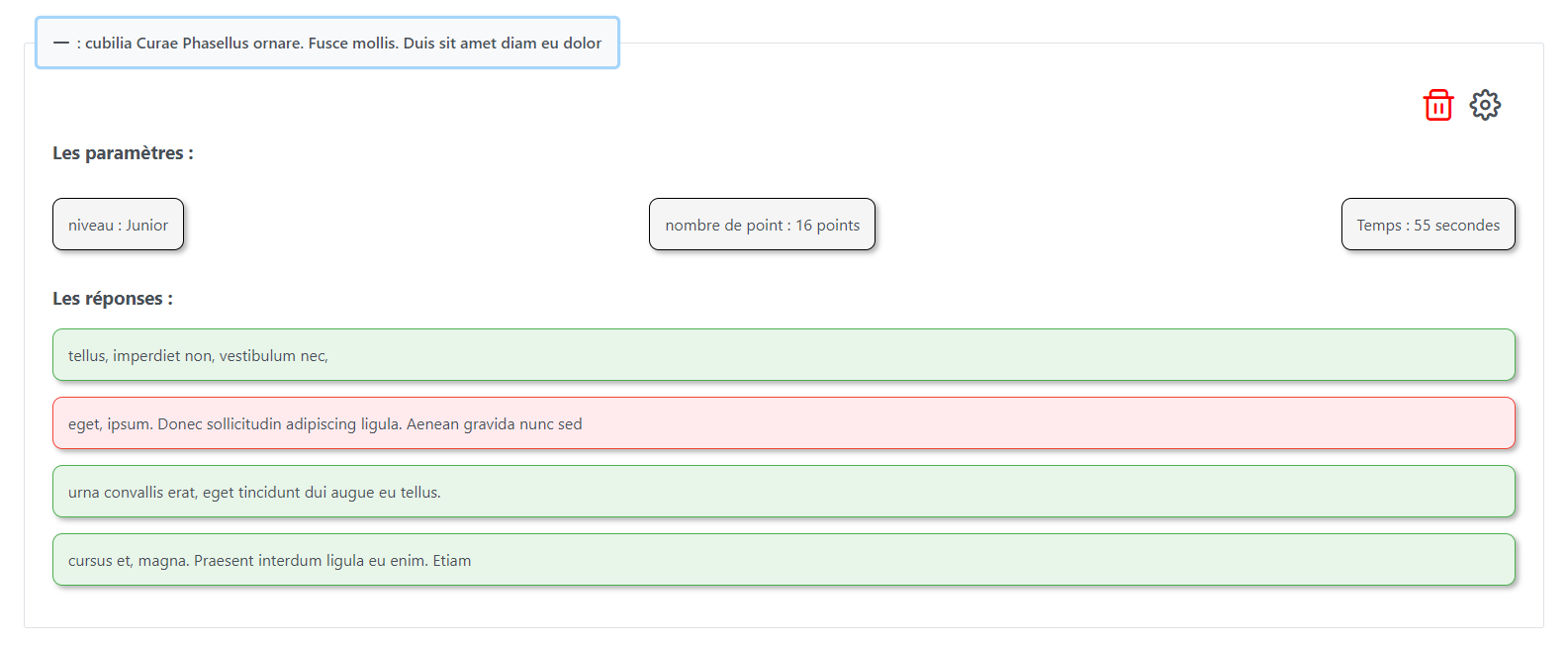


Annexe 5 : Le Share Component

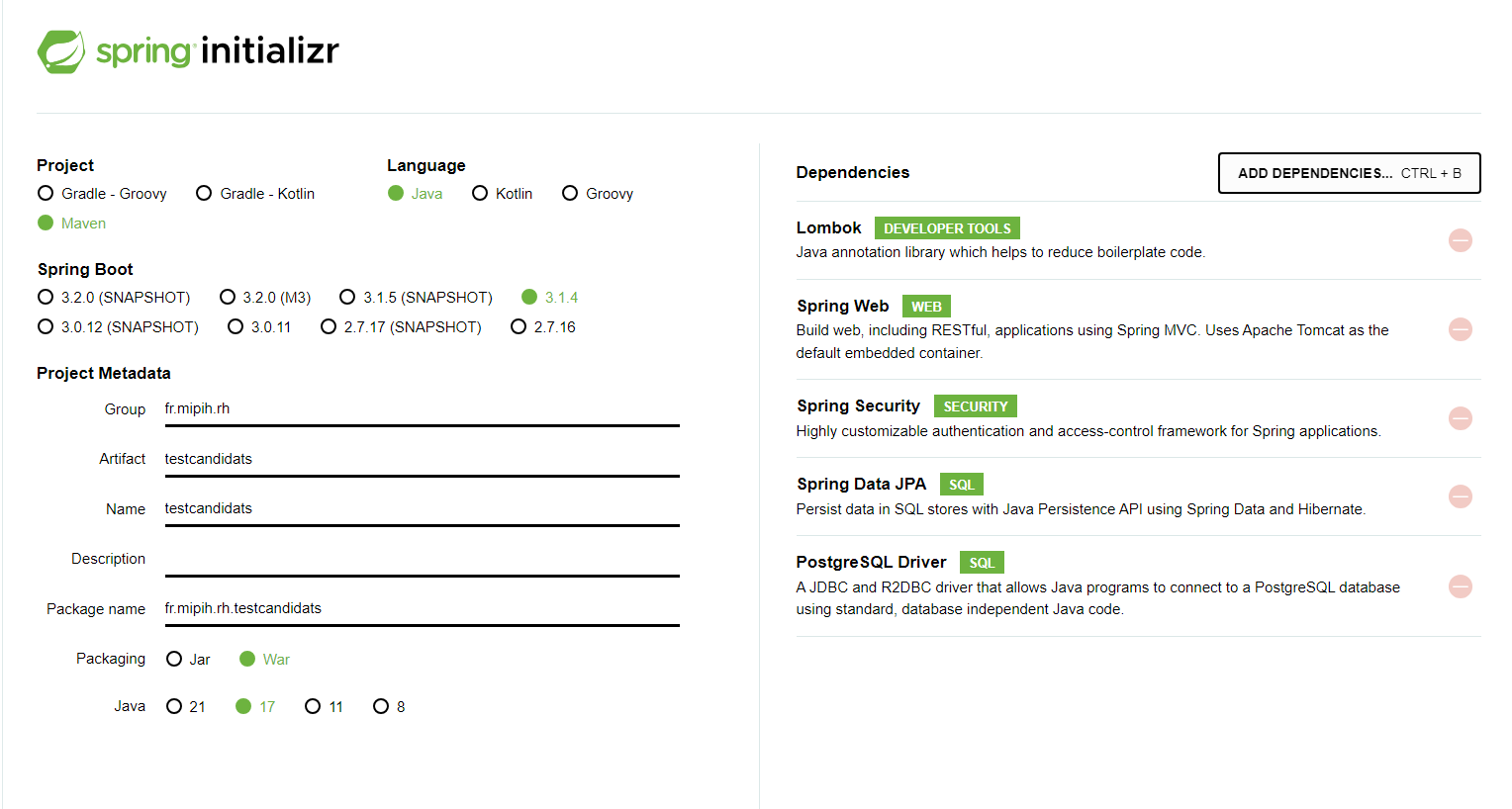
Le mode edit :



Le mode display :



Annexe 6 : Spring initializr



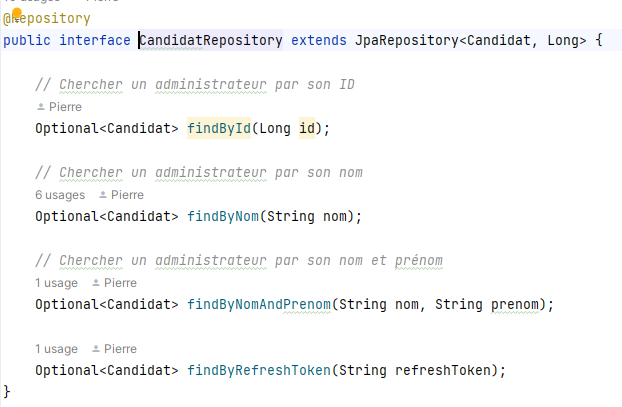
Annexe 7 : SecurityFilterChain



Annexe 8 : Mapper Question



Annexe 9 : Repository



Annexe 10 : Schéma base de données

