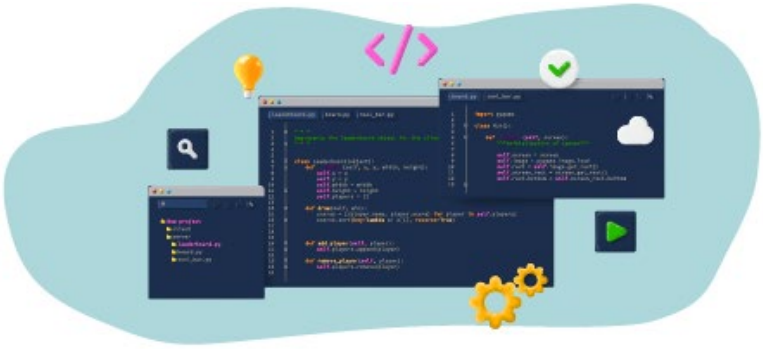
ORION



Justification des choix techniques
Projet MDD

Auteur : [BENSEGHIR Sabrina]
Version 0.0.1

1. Aperçu / Synthèse

Après avoir fait une veille autour des technologies Java Angular pour la réalisation de l'application MDD en version Minimum Viable Product, mon choix s'est orienté sur les librairies, frameworks et bonnes pratiques les plus utilisées dans les projets par des développeurs.

Cela permet de répondre aux attentes des utilisateurs orientés sur l'accessibilité du site, sa rapidité et ses interfaces faciles d'utilisation etc. Et également de faciliter le développement de l'application pour les développeurs car ce sont des technologies connues et documentées permettant d'assurer l'évolutivité de l'application tout en réduisant les coûts de développement.

Ce document tient compte des exigences particulières mentionnées dans les spécifications fonctionnelles, et également des contraintes techniques.

Tableau des choix portées sur l'application :

	Front-End	Back-End
Architecture logicielle	En couches	
Framework	Angular CLI	Spring-Boot
Librairies	Angular Material RxJS	Spring Data JPA, Spring Security avec JWT, Lombok
Bonnes pratiques	SOLID	

2. Choix techniques

2.1 Choix architecture logicielle

Au travers de ce tableau, j'ai pu effectuer mon choix concernant l'architecture à choisir pour l'application.

Tableau comparatif entre différentes architectures logicielles

	Avantages	Inconvénients
Client-Serveur	1. Encapsulation du matériel, des logiciels et des fonctionnalités. 2. Combinaison fluide de clients et de serveurs sur différentes plateformes.	1. Si tous les clients demandent simultanément des données au serveur, celui-ci peut être surchargé. 2. Si le serveur échoue pour une raison quelconque, aucune demande client ne peut être satisfaite.
Pilotée par les événements	1. Capable de gérer des millions d'événements en même temps. 2. Capable de faire communiquer différentes technologies et plateformes avec le même bus d'événement.	1. Si le bus d'événements tombe en panne, le système ne fonctionne plus. 2. Le bus d'événements peut être surchargé et subir des problèmes de performance.
En couches	1. Encapsulation du matériel, des logiciels et des fonctionnalités. 2. Si une couche est modifiée, les autres couches restent les mêmes.	1. Pour les petites applications, de nombreuses couches créent un problème de performance et sont très difficiles à maintenir.

De ce fait, mon choix s'est porté sur l'architecture en couche

Tableau choix de l'architecture logicielle

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
Architecture en couches	https://openclassrooms.com/fr/courses/7210131-definissez-votre-architecture-logicielle-grace-aux-standards-reconnus/7371866-revisez-ce-que-vous-avez-appris	Conception de l'application

Justification du choix technique :

- Chaque couche a des responsabilités distinctes au sein de l'application. Lorsqu'une couche est modifiée, cela n'affecte pas les autres car elles sont indépendantes les unes des autres. On peut par exemple faire en sorte qu'une application qui ne fonctionne que sur les PC fonctionne sur les téléphones et les tablettes, sans avoir à réécrire toute l'application.

- Les couches permettent une meilleure personnalisation du système.

- Cela permet une meilleure maintenabilité de l'application, et une meilleure gestion des composants.

- Les services sont connectés via les interfaces de programmation dites API

2.2 Choix frameworks

Dans le cadre du développement de l'application, mes choix se sont orientés sur les frameworks fréquemment utilisés, et mentionnés dans les contraintes techniques :

Tableau choix des frameworks

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
Back-End : Spring-Boot	https://openclassrooms.com/fr/courses/6900101-creez-une-application-java-avec-spring-boot/7077993-structurez-et-configurez-votre-projet	Le développement de l'application pour la partie logique métier permettant l'accès aux données, aux services et à d'autres systèmes.
Front-End : Angular CLI	*https://blog.angular.io/schematics-an-introduction-dc1dfbc2a2b2*	Le développement de l'application pour la partie visuelle concernant l'expérience utilisateur.

Justification des choix technique :

- *Spring-Boot* : est un outil qui accélère et simplifie le développement d'applications. C'est un composant Spring au service des autres composants. Afin de respecter les bonnes pratiques, l'architecture Spring-Boot se veut être une architecture en couche comme celle choisi pour la conception de l'application, voici les principales :
  - *couche Controller* : gestion des interactions entre l'utilisateur de l'application et l'application.
  - *couche Service* : implémentation des traitements métiers spécifiques à l'application.
  - *couche Repository* : interaction avec les sources de données externes.
  - *couche Model* : implémentation des objets métiers qui seront manipulés par les autres couches.

- *Angular CLI* : est un outil d'interface de ligne de commande qui permet d'initialiser, développer, et maintenir des applications Angular directement à partir d'un terminal de commande. Sans cela, la création et la construction d'une application nécessite l'utilisation et la maîtrise de nombreux outils.

2.3 Choix librairies

Pour cette partie, je me suis orientée sur différents aspects à prendre en compte pour le développement de l'application, notamment la sécurité, l'accès aux données, la gestion des données, l'UI design etc.

De ce fait, j'ai dressé des tableaux distincts pour mes choix techniques du côté Back-End (Java), et du côté Front-End (Angular) afin d'en faciliter la lecture.

J'ai défini mes choix sur les librairies les plus couramment utilisées en projet, en faisant un comparatif avec leurs avantages et inconvénients.

Front-End : Angular

Tableau comparatif entre les différentes librairies pour l'UI Design

	Avantages	Inconvénients
Angular Material	1. Créé par Angular 2. Composants tout prêt 3. Composants customisables 4. Responsive, facile d'utilisation 5. Forte communauté de dev	1. Difficile de créer des composants complexes
PrimeNG	1. Composants tout prêt 2. Composants customisables 3. Créations de composants complexes	1. Lenteur de la librairie 2. Utilisation des composants limités
NG-ZORRO	1. Composants tout prêt 2. Composants customisables	1. Faible communauté de dev 2. Peu utilisé en projet

De ce fait, mon choix s'est porté sur la librairie Angular Material pour l'UI Design. J'ai ajouté à mon tableau une autre librairie pour la gestion des flux s'appuyant sur le paradigme de programmation réactive.

Tableau choix des librairies pour Angular

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
Angular Material	*https://material.angular.io/* *https://www.reddit.com/r/Angular2/comments/ovvnug/which_is_the_best_primeng_vs_material_vs_ngzorro/*	Interface de l'application, UI design

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
RxJS	*https://angular.io/guide/rx-library*	La gestion des flux de données

Justification des choix technique :

- *Angular Material* : est une librairie créée par l'équipe Angular, permettant de créer des interfaces simples et élégants, et ce de manière responsive. Sa documentation permet d'intégrer des composants déjà mis à disposition et également de personnaliser ses propres composants.

- *RxJS* : est une bibliothèque JavaScript puissante qui permet aux développeurs d'écrire du code asynchrone et événementiel. Elle fournit un large éventail d'opérateurs et de fonctions qui facilitent la création de modèles de programmation réactifs en se basant sur le patron de conception Observable. De ce fait, cela améliore l'expérience utilisateur avec le système en obtenant des réponses plus rapides à leurs requêtes.

Back-End : Java

Tableau comparatif entre les différentes librairies pour l'accès aux données

	Avantages	Inconvénients
Spring Data JPA	1. Créer des programmes Java basés sur des bases de données en utilisant une sémantique orientée objet 2. Usages des annotations 3. Indépendants de la base de données 4. Simple à comprendre, développement de l'application réalisé plus rapidement	1. Si un développeur ne comprend pas le fonctionnement interne du framework JPA ou la conception de la base de données, il sera incapable d'écrire du bon code
Spring Data JDBC	1. Permet d'écrire des requêtes SQL 2. Alternative préférable si une application utilise une base de données simple	1. Performances de JDBC peuvent être extrêmement lentes si les requêtes sont mal rédigées

Tableau comparatif entre les différentes librairies pour la sécurité

	Avantages	Inconvénients
JWT	1. Peuvent transférer les détails des utilisateurs. 2. Peuvent être vérifiés efficacement et rapidement 3. JWT ne sont stockés que côté client permettant d'économiser de l'espace de stockage de base de données. 4. Offrent de solides garanties de sécurité. Ils sont signés numériquement et ne peuvent être modifiés par des clients ou des attaquants.	1. Ne peuvent pas être révoqués car il n'y a aucun appel à la base de données lors de la validation. 2. La révocation immédiate de JWT nécessite la mise en œuvre d'une liste noire JWT, ce qui peut prendre du temps
OAuth	1. Est largement utilisé 2. Dispose de bibliothèques clientes bien testées dans presque tous les langages et frameworks Web. 3. Permet l'isolation du code, ce qui signifie que le code de l'application client n'est pas affecté par le code d'autorisation	1. Peut être compliqué à comprendre pour les débutants 2. OAuth peut être trop complexe pour certains scénarios 3. Peut créer des problèmes de confidentialité pour les utilisateurs finaux.

Suite à l'analyse de ses librairies, mon choix s'est orienté pour Spring Data JPA pour l'accès / persistance des données, et JWT accompagné de Spring Security qui est le composant utilisé pour la sécurité des applications dans le framework Spring. J'ai ajouté à mon tableau une autre librairie permettant une lisibilité du code et un gain de temps dans le développement grâce à des annotations.

Tableau choix des librairies pour Java

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
Spring Data JPA	*https://www.baeldung.com/jpa-vs-jdbc#:~:text=JDBC%20is%20database%2Ddependent%2C%20which,few%20(or%20no)%20modifications*.	L'accès / persistance des données

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
Spring Security avec JWT	*https://frontegg.com/blog/oauth-vs-jwt*	La sécurité de l'application
Lombok	*https://www.javatpoint.com/lombok-java*	Gain de temps avec l'usage d'annotations

Justification des choix technique :

- *Spring Data JPA* : est un module de Spring qui traite l'accès aux données par une couche de persistance. Ce module est indépendant des bases de données, ce qui signifie que le même code peut être utilisé dans diverses bases de données avec peu (ou pas) de modifications. Facile d'utilisation, il permet de créer des requêtes par l'usage d'annotations dans les classes. De plus Heidi, a fait ce choix lors de la création du projet.

- *Spring Security avec JWT* : Spring Security est le module utilisé dans tous projets Spring. Il est souvent accompagné soit de JWT ou OAuth. Dans mon cas j'ai choisi JWT, car il est plus simple à mettre en place qu'OAuth, les jetons peuvent être vérifiés rapidement et efficacement du côté serveur, et cela permet d'économiser de l'espace de stockage de base de données.

- *Lombok* : est une librairie créée pour les projets codés en Java. En effet, il permet d'alléger la rédaction du code par le biais d'annotations. Il se connecte automatiquement aux IDE et crée des outils pour notre application Java. Cela permet d'économiser du temps et des efforts, et d'améliorer la lisibilité du code source.

2.4 Choix des bonnes pratiques

Tableau comparatif entre les différents principes

	Avantages	Inconvénients
Principes DRY, KISS, YACNI	1. Permet d'éviter de dupliquer le code 2. Code facile à lire et à comprendre 3. Gain de temps en évitant de rédiger du code inutile	1. Principes assez abstraites, pas de ligne directrice à suivre
Principes SOLID	1. Rend la conception de logiciels facile à comprendre, à maintenir et plus flexible 2. Permet de réduire la complexité 3. Rédigé par Robert C. Martin, ingénieur logiciel reconnu dans le milieu	1. Parfois tous les principes ne peuvent être utilisés dans un projet

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
SOLID	https://workat.tech/machine-coding/tutorial/software-design-principles-dry-yagni-eytrxfhz1fla https://workat.tech/machine-coding/tutorial/solid-design-principles-8yu7bjegrxs5 https://www.baeldung.com/solid-principles	Développement application, clean code

Justification des choix technique :

- *SOLID* : ce sont des principes de conception permettant de créer des logiciels plus maintenables, plus compréhensibles et plus flexibles. Par conséquent, à mesure que la taille des applications augmente, il est possible réduire leur complexité. Ces principes sont aussi bien utilisés dans le développement back-end que le développement front-end.

Voici les cinq concepts qui constituent les principes SOLID :

- ***S****ingle Responsibility* : une classe, une responsabilité
- ***O****pen/Closed :* ouvert à l'extension, fermé à la modification
- ***L****iskov Substitution* : une sous-classe peut être utilisée à la place de sa classe parent.
- ***I****nterface Segregation* : les interfaces plus grandes doivent être divisées en interfaces plus petites.
- ***D****ependency Inversion* : les modules de haut niveau ne dépendront plus des modules de bas niveau