

ORION



Justification des choix techniques ***Projet MDD***



Auteur : [BADRI Sameh]
Version 0.0.1

Sommaire

APERÇU / SYNTHESE	3
<i>Architecture logicielle (client-serveur) :.....</i>	<i>3</i>
<i>Paradigmes.....</i>	<i>3</i>
<i>Utilisation du framework Spring Security et de JSON Web Token (JWT)</i>	<i>3</i>
<i>Gestion des tokens et de la sécurité.....</i>	<i>3</i>
<i>Utilisation des design patterns</i>	<i>4</i>
CHOIX TECHNIQUES	5
1. BACK-END (API).....	5
<i>Frameworks :</i>	<i>5</i>
Choix 1 (Spring Boot Starter Data JPA)	5
Choix 2 (Spring Boot Starter Web).....	5
Choix 3 (Spring Boot Starter Security)	6
Choix 4 (Hibernate Core).....	7
<i>Librairies:</i>	<i>7</i>
Choix 1 (MySQL Connector/J)	7
Choix 2 (Java JWT)	8
Choix 3 (Project Lombok)	8
Choix 4 (Jakarta Validation API)	9
Choix 5 (Jackson Databind et Jackson Core).....	9
Choix 6 (MapStruct).....	10
<i>Design Patterns.....</i>	<i>10</i>
Choix 1 (Inversion of Control / Dependency Injection).....	10
Choix 2 (Pattern Repository)	10
Choix 3 (Decorator)	11
Choix 4 (Unit of Work & Identity Map).....	11
Choix 5 (MVC)	11
Choix 6 (Builder)	11
Choix 7 (Adapter).....	12
Choix 8 (Strategy)	12
Choix 9 (Factory).....	12
2. FRONT-END (UI)	13
Choix 1 (Framework Angular)	13
Choix 2 (Material)	13

Architecture logicielle (client-serveur) :

- Client : Application Angular (UI réactive avec RxJS et Angular Material).
- Serveur : API Spring Boot (gestion des requêtes, sécurité, persistance des données via JPA/Hibernate).

⇒ L'application Angular (client) communique avec l'application Spring Boot (serveur) pour échanger des données et effectuer des opérations.

avantage	inconvénient
✓ Centralisation des données.	✗ Dépendance au serveur.
✓ Scalabilité et maintenance simplifiée.	✗ Risque de surcharge du serveur.

Paradigmes :

- **Front-end : Programmation réactive (RxJS)** pour une UI dynamique. Le front-end de l'application est développé avec Angular et intègre plusieurs modules d'Angular Material pour offrir une interface utilisateur dynamique et ergonomique. L'utilisation de la librairie RxJs permet une gestion réactive des données, tandis que les services et intercepteurs Angular assurent la gestion des sessions et la sécurité via JWT.
- **Back-end : Programmation orientée objet (POO)** Le côté API (serveur) utilise Spring Boot pour gérer les requêtes HTTP, la sécurité, et les opérations de persistance avec JPA et Hibernate pour l'ORM. avec ORM ,
- **Data-base : Paradigme ORM (Object-Relational Mapping)** : pour mapper les données MySQL en objets Java. L'ORM permet de manipuler les données de la base comme des objets dans le code, simplifiant ainsi le développement en réduisant l'écriture de requêtes SQL complexes et répétitives. Le stockage des données est assuré par une base de données MySQL, tandis que des bibliothèques comme Jackson et MapStruct facilitent la conversion et la manipulation des objets. Cette architecture garantit une séparation claire entre le front-end et le back-end, rendant le projet plus facile à maintenir et à faire évoluer.

Utilisation du framework Spring Security et de JSON Web Token (JWT)

L'application utilise Spring Security associé à JSON Web Token (JWT) pour assurer l'authentification et l'autorisation. Le design pattern Strategy est employé pour encapsuler cette logique dans des composants distincts, intégrés au sein de Spring Security.

Gestion des tokens et de la sécurité

Les paramètres de sécurité sont configurables dans le fichier `application.properties` :

- `oc.app.jwtSecret` : clé secrète pour signer les tokens JWT.
- `oc.app.jwtExpirationMs` : durée de validité des tokens JWT, définie par défaut à 24 heures pour le MVP, sans persistance entre les sessions.

Utilisation des design patterns

- Inversion of Control (IoC) et Dependency Injection : Spring Boot gère la création et l'injection des dépendances de manière flexible.
- Repository : standardise l'accès aux données.
- Decorator : enrichit la sécurité, notamment pour la protection des routes et API.
- Unit of Work & Identity Map : optimisent la gestion des transactions et sessions.
- Model-View-Controller (MVC) : structure l'application en séparant les responsabilités.
- Builder : réduit le code répétitif.
- Adapter : facilite le mappage objet-objet.
- Strategy (via JWT) : gère l'authentification.
- Factory : assure la connexion à la base de données MySQL.

Cette architecture modulaire garantit une meilleure organisation du code, une sécurité renforcée et une évolutivité optimisée.

Choix techniques

1. BACK-END (API)

Frameworks :

Choix 1 (Spring Boot Starter Data JPA)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Data JPA	https://spring.io/guides/gs/accessing-data-jpa	La persistance, la gestion des données.

Justification du choix technique :

La persistance des données est simplifiée grâce à l'intégration automatique des dépendances nécessaires et la gestion des versions. Elle génère des requêtes automatiquement, réduisant ainsi le besoin de code SQL personnalisé. La gestion des transactions est automatique, minimisant les erreurs et garantissant la cohérence des données. L'abstraction du repository facilite l'accès aux données, tandis que le support de la pagination et du tri améliore les performances, notamment grâce au cache.

avantage	inconvénient
<ul style="list-style-type: none">✓ Intégration transparente avec Spring Boot.✓ Réduction du code boilerplate.✓ Facilité d'utilisation des repositories.	<ul style="list-style-type: none">✗ Surcouche supplémentaire.✗ Limitations des méthodes de requête générées.

Choix 2 (Spring Boot Starter Web)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Web	https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web	Démarrer rapidement le développement web avec Spring.

Justification du choix technique :

Ce choix technique permet de simplifier le développement d'applications web et RESTful en Java en encapsulant automatiquement les configurations et les dépendances nécessaires. Il offre un démarrage rapide, sans nécessiter de gestion des détails de configuration sous-jacents. Le starter inclut **Tomcat**

comme serveur web embarqué par défaut, facilitant ainsi le déploiement et la mise en service. De plus, il offre un support intégral pour la création de services web RESTful grâce à **Spring MVC**, ainsi qu'une sérialisation et désérialisation JSON automatiques avec **Jackson**, la validation des données et la gestion des erreurs, ce qui améliore la robustesse et la qualité du code.

avantage	inconvénient
<ul style="list-style-type: none"> ✓ Configuration simplifiée : Démarrage rapide avec des configurations par défaut. ✓ Intégration facile : Fonctionne bien avec d'autres composants Spring. ✓ Communauté active : Support et documentation abondants. 	<ul style="list-style-type: none"> ✗ Abstraction élevée : Moins de contrôle sur les détails. ✗ Dépendances lourdes : Peut augmenter la taille de l'application.

Choix 3 (Spring Boot Starter Security)

Choix technique	Lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Security	https://docs.spring.io/spring-security/reference/getting-spring-security.html	<i>En utilisant Spring Security, sécuriser l'application contre les menaces communes.</i>

Justification du choix technique :

Ce starter facilite l'intégration des mécanismes d'authentification et d'autorisation, offrant une protection efficace contre les menaces courantes. Il gère de manière transparente la sécurité des sessions, le cryptage des mots de passe et la sécurisation des requêtes HTTP avec une configuration minimale. De plus, il est conforme aux normes de sécurité, garantissant ainsi une solution fiable et robuste.

avantage	inconvénient
<ul style="list-style-type: none"> ✓ Configuration simplifiée : Intégration rapide avec des configurations par défaut. ✓ Sécurité robuste : Prend en charge les meilleures pratiques de sécurité. ✓ Extensibilité : Facile à personnaliser et à étendre. 	<ul style="list-style-type: none"> ✗ Courbe d'apprentissage : Peut être complexe. ✗ Dépendances lourdes : Peut augmenter la taille de l'application.

Choix 4 (Hibernate Core)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Hibernate Core	https://hibernate.org/orm/documentation/getting-started/	<i>faciliter la cartographie des objets Java aux tables de base de données et vice versa.</i>

Justification du choix technique :

En tant que framework ORM (Object-Relational Mapping), il simplifie la cartographie des objets Java aux tables de bases de données, rendant la manipulation des données plus intuitive et orientée objet. Il propose également des fonctionnalités avancées telles que le caching, la gestion des transactions et l'optimisation des requêtes, ce qui contribue à améliorer la performance et l'efficacité de l'application.

Avantage	Inconvénient
<ul style="list-style-type: none">✅ ORM puissant : Simplifie la gestion des bases de données.✅ Support HQL : Langage de requête orienté objet.✅ Cache intégré : Améliore les performances.	<ul style="list-style-type: none">❌ Courbe d'apprentissage : Peut-être complexe.❌ Configuration lourde : Peut nécessiter beaucoup de configurations.❌ Performance : Peut-être moins performant pour des requêtes très complexes.

Librairies:

Choix 1 (MySQL Connector/J)

Choix technique	Lien vers le site / la documentation / une ressource	But du choix
MySQL Connector/J	https://mvnrepository.com/artifact/mysql/mysql-connector-java	<i>Connecter l'API à la base de données MySQL, interaction fluide entre l'application et la base de données.</i>

Justification du choix technique :

En tant que driver JDBC officiel, il assure une compatibilité optimale et des

performances élevées pour les applications Java communiquant avec MySQL. Il offre une solution fiable, performante et facile à utiliser pour intégrer MySQL dans des applications Java.

avantage	inconvénient
<ul style="list-style-type: none"> ✓ Compatibilité : Fonctionne bien avec MySQL. ✓ Performance : Optimisé pour des performances élevées. ✓ Documentation : Bien documenté et supporté. 	<ul style="list-style-type: none"> ✗ Dépendance spécifique : Limité à MySQL. • Complexité : Peut être complexe à configurer ✗ Mises à jour : Nécessite des mises à jour régulières pour rester sécurisé.

Choix 2 (Java JWT)

Choix technique	Lien / Documentation	But du choix
Java JWT	GitHub - auth0/java-jwt	Création et vérification de tokens JWT pour sécuriser les échanges client-serveur.

Justification du choix technique :

Cette bibliothèque simplifie la gestion des tokens JWT grâce à une API intuitive, tout en respectant les standards de sécurité. Elle permet de signer, vérifier et décoder les tokens efficacement, ce qui sécurise l'authentification sans stocker de sessions côté serveur.

Avantages	Inconvénients
<ul style="list-style-type: none"> ✓ Tokens auto-contenus et interopérables. ✓ Intégration rapide avec Spring Security. ✓ Compatibilité avec les normes JWT. 	<ul style="list-style-type: none"> ✗ Gestion manuelle de l'expiration. ✗ Taille des tokens potentiellement volumineuse.

Choix 3 (Project Lombok)

Choix technique	Lien / Documentation	But du choix
Project Lombok	Site officiel Lombok	Réduire le code boilerplate (getters, setters, constructeurs).

Justification du choix technique :

Lombok génère automatiquement le code répétitif via des annotations (ex: @Data, @Builder), améliorant la productivité et la lisibilité du code. En automatisant la création de méthodes courantes telles que les getters, setters,

ainsi que les méthodes equals(), hashCode(), et toString() via des annotations simples, Lombok améliore considérablement la lisibilité et la concision du code. En outre, Lombok facilite la maintenance en réduisant les risques d'erreurs humaines dans la génération de ces méthodes.

Avantages	Inconvénients
<ul style="list-style-type: none"> ✓ Réduction de 30% du code boilerplate. ✓ Code plus concis et maintenable. ✓ Lisibilité : Rend le code plus propre et lisible. 	<ul style="list-style-type: none"> ✗ Débogage complexe (code généré invisible). ✗ Nécessite un plugin IDE.

Choix 4 (Jakarta Validation API)

Choix technique	Lien / Documentation	But du choix
Jakarta Validation API	Jakarta EE Validation	Valider les données entrantes (ex: formulaires, requêtes API).

Justification du choix technique :

Les annotations comme @NotNull ou @Size permettent de valider les données métier directement dans les entités, garantissant leur intégrité avant traitement. Permet de spécifier des contraintes de validation directement sur les modèles de données via des annotations simples comme @NotNull, @Size, @Min, et @Max. En garantissant que les données respectent ces contraintes avant leur traitement ou leur persistance, l'API Jakarta Validation aide à prévenir les erreurs de données et à renforcer l'intégrité des systèmes.

Avantages	Inconvénients
<ul style="list-style-type: none"> ✓ Validation déclarative via annotations. ✓ Intégration native avec Spring. 	<ul style="list-style-type: none"> ✗ Courbe d'apprentissage pour les validations personnalisées.

Choix 5 (Jackson Databind et Jackson Core)

Choix technique	Lien / Documentation	But du choix
Jackson	Site officiel Jackson	Sérialiser/désérialiser les objets Java en JSON

Justification du choix technique :

Jackson est la bibliothèque standard pour Spring Boot. Elle offre une conversion rapide entre objets et JSON, avec un support avancé des annotations (@JsonIgnore, @JsonProperty).

Avantages	Inconvénients
<ul style="list-style-type: none"> ✓ Performances élevées. 	<ul style="list-style-type: none"> ✗ Configuration manuelle pour les cas

Avantages	Inconvénients
✓ Personnalisation via annotations.	complexes.

Choix 6 (MapStruct)

Choix technique	Lien / Documentation	But du choix
MapStruct	Site officiel MapStruct	Convertir automatiquement les DTO en entités (et inversement).

Justification du choix technique :

MapStruct génère du code de mappage à la compilation, éliminant les erreurs manuelles et améliorant les performances.

Avantages	Inconvénients
✓ Aucun impact sur les performances runtime.	✗ Dépendance à ajouter au projet.
✓ Réduction du code boilerplate.	✗ Configuration nécessaire pour les cas complexes.

Design Patterns

Choix 1 (Inversion of Control / Dependency Injection)

Choix technique	But du choix
IoC/DI (Spring Boot)	Centraliser la gestion des dépendances.

Justification :

Spring injecte automatiquement les dépendances (ex: @Autowired), découplant les composants pour une meilleure testabilité.

Avantages	Inconvénients
✓ Composants réutilisables.	✗ Configuration initiale complexe.
✓ Tests simplifiés (mocks).	

Choix 2 (Pattern Repository)

Choix technique	But du choix
Repository (Spring Data JPA)	Standardiser l'accès aux données.

Justification :

Les interfaces JpaRepository génèrent automatiquement les requêtes CRUD, réduisant le code SQL manuel.




Avantages	Inconvénients
✓ Requetes générées dynamiquement.	✗ Limité pour les requêtes SQL avancées.
✓ Support de la pagination/tri.	

Choix 3 (Decorator)

Choix technique	But du choix
Decorator (Spring Security)	Ajouter des couches de sécurité (ex: logging, chiffrement).

Justification :

Enveloppe les composants existants pour ajouter des fonctionnalités sans modifier leur code (ex: OncePerRequestFilter).




Avantages	Inconvénients
 Flexibilité d'ajout de fonctionnalités.	 Risque de surcharge des décorateurs.
 Respect du principe Open/Closed.	

Choix 4 (Unit of Work & Identity Map)

Choix technique	But du choix
Hibernate	Optimiser les transactions et le cache.

Justification :

- Unit of Work : Groupe les opérations SQL en une transaction unique.
- Identity Map : Évite les doublons d'entités en mémoire.




Avantages	Inconvénients
 Cohérence des données.	 Gestion manuelle du cache nécessaire.
 Réduction des appels à la BDD.	

Choix 5 (MVC)

Choix technique	But du choix
MVC (Spring Boot)	Séparer la logique métier, l'UI et le contrôle.

Justification :

- Modèle : Entités et services métier.
- Vue : JSON via les contrôleurs REST.
- Contrôleur : Gère les requêtes HTTP.

Avantages	Inconvénients
 Code organisé et modularité.	 Taille accrue pour les petits projets.
 Facilité de maintenance.	

Choix 6 (Builder)

Choix technique	But du choix
Builder (Lombok)	Construire des objets complexes étape par étape.

Justification :

L'annotation @Builder génère un builder fluent pour instancier des objets

immuables.

Avantages	Inconvénients
✓ Lisibilité améliorée.	✗ Dépendance à Lombok.
✓ Support des paramètres optionnels.	

Choix 7 (Adapter)

Choix technique	But du choix
Adapter (MapStruct)	Convertir des objets entre couches (ex: DTO → Entité).

Justification :

MapStruct agit comme un adaptateur entre les formats de données front-end et back-end.

Avantages	Inconvénients
✓ Découplage des couches.	✗ Configuration pour les cas complexes.
✓ Code généré à la compilation.	

Choix 8 (Strategy)

Choix technique	But du choix
Strategy (JWT)	Interchangeabilité des algorithmes d'authentification.

Justification :

Permet de choisir dynamiquement entre JWT, OAuth2, etc., via des composants Spring Security.

Avantages	Inconvénients
✓ Modularité de la sécurité.	✗ Configuration avancée nécessaire.
✓ Facilité d'évolution.	

Choix 9 (Factory)

Choix technique	But du choix
Factory (MySQL Connector/J)	Créer des connexions à la BDD via DataSource.

Justification :

Spring Boot configure automatiquement le pool de connexions via application. properties.

Avantages	Inconvénients
✓ Gestion centralisée des connexions.	✗ Dépendance au driver MySQL.
✓ Performance optimisée.	

2. FRONT-END (UI)

Choix 1 (Framework Angular)

Choix technique	Lien / Documentation	But du choix
Angular	Site officiel Angular	Développer une SPA (Single Page Application) réactive.

Justification du choix technique :

Angular offre une structure solide (components, services, modules) et des outils comme RxJS pour gérer les flux de données asynchrones.

Avantages	Inconvénients
✓ Architecture modulaire.	✗ Courbe d'apprentissage élevée.
✓ Two-way data binding.	✗ Taille du bundle.

Choix 2 (Material)

Choix technique	Lien / Documentation	But du choix
Angular Material	Site officiel Material	Fournir des composants UI prêts à l'emploi.

Justification du choix technique :

Angular Material suit les principes Material Design, offrant des composants accessibles, responsives et esthétiques (ex: MatTable, MatDialog).

Avantages	Inconvénients
✓ UI cohérente et professionnelle.	✗ Personnalisation limitée sans effort supplémentaire.
✓ Documentation complète.	