ORION

Justification des choix techniques
Projet MDD

Auteur : [LE NOM DE L'ÉTUDIANT]
Version 0.0.1

Aperçu / Synthèse **3**

Choix techniques **3**

Choix XXX 3

Choix XXX 3

Aperçu / Synthèse

Les choix effectués pour ce projet concernant les *frameworks ont était les suivants :*
Pour la partie backend

- Spring Data JPA
- Spring boot
- Spring Security
- SGBD MySQL / Hibernate
- Docker / Docker-compose
- Lombok
- MapStruct
- Swagger

Coté backend je suis partie sur une logique de microservice, cad avec des repositorys services implémentation de services et contrôleurs. Je suis partie sur une architecture hexagonale pour ce projet, le but ce cette approche et d'isolé le cœur de l'application afin de n'avoir qu'un point d'entrer a modifier comme les interfaces sans toucher au cœur de la partie métier.

Choix techniques

Choix Spring

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot	How to use SpringBoot	*Démarrage rapide via auto-configuration et starters, gestion de la configuration externalisée (profils), endpoints d'observabilité (Actuator). Réduit fortement le boilerplate et homogénéise l'application.*

choix technique	lien vers le site / la documentation / une ressource	but du choix
SpringWeb (Rest)	How to use SBW	Exposer des API REST robustes avec @RestController, gestion des erreurs centralisée, sérialisation JSON, CORS, etc.

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Security	*Spring Security* *Spring security OC bealccdung*	*Atm Spring security est très bien documenter, le but est de mettre facilement en place une couche d'accès au endpoint ainsi qu'a une sécurité de mot de passe et gestion de token de sécurisation JWT*

Spring data JPA

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring data jpa	*SpringDataJpa bealdung*	*par exemple : la sécurisation, la gestion de données...*

Lombok

choix technique	lien vers le site / la documentation / une ressource	but du choix
Lombook	Lombok how to use	Réduire le *boilerplate* (getters/setters, constructeurs, *builder*, equals/hashCode). Améliore la lisibilité.

Swagger

choix technique	lien vers le site / la documentation / une ressource	but du choix
Swagger	*How to impl Swagger*	Générer la spécification OpenAPI et **Swagger UI** automatiquement depuis le code. Facilite la découverte, le test manuel des endpoints et la contractualisation avec le front.

Swagger-UI

choix technique	lien vers le site / la documentation / une ressource	but du choix
Swagger-ui	*How to impl Swagger*	Interface web pour essayer les endpoints, renseigner les schémas d'entrée/sortie et les codes de réponse. Utile pour les PO etc

MapScruct

choix technique	lien vers le site / la documentation / une ressource	but du choix
MapStruct	*How to use MapStruct*	Mappings DTO ⇄ Entités **générés au build** (pas de réflexion), performants et sûrs à la compilation. Centralise la transformation des modèles.

jsonwebtoken

choix technique	lien vers le site / la documentation / une ressource	but du choix
JsonWebToken	*How to use JWT*	Authentification **stateless** : le serveur ne garde pas d'état de session. Intégration simple avec Spring Security, signature/expiration, transport dans l'en-tête Authorization.

Choix techniques – Conteneurisation & exécution

Docker

choix technique	lien vers le site / la documentation / une ressource	but du choix
Docker	*https://docs.docker.com /*	Environnements reproductibles et isolés (app + MySQL). Parité dev/staging/prod, *image immuable*.

Docker Compose

choix technique	lien vers le site / la documentation / une ressource	but du choix
Docker Compose	*https://docs.docker.com/compose/*	Démarrage multi-services en une commande (docker-compose up), variables d'environnement par profil, volumes pour la persistance locale.

Choix d'architecture

Architecture hexagonal

Architecture Hexagonal **Découpage couches** *(controller → service → repository)*		**Isoler le domaine** des technologies (web, DB, messaging). Tests plus simples, remplaçabilité (ex : changer MySQL → Postgres) sans impacter le cœur métie Responsabilités nettes, séparations des préoccupations, lisibilité et testabilité accru (mock repositories/services).*

Choix techniques – Front-end Angular

Angular (v14) – Framework	**https://angular.io/docs**	**Cadre complet et structurant (modules, DI, templates) adapté aux SPA. Standard industriel, excellente intégration TypeScript, outillage mature.**

Angular CLI	Générer, tester et construire le front de façon homogène (ng generate, ng build, ng test). Simplifie la CI/CD et la standardisation des projets.
RxJS	Modéliser l'asynchrone (HTTP, UI events) par des *streams* composables. Facilite l'annulation, le retry, la gestion d'erreurs et l'orchestration complexe.
Angular Router	Navigation déclarative, *guards* (auth/roles), *resolvers*, *lazy-loading* des modules pour réduire le bundle initial.
Reactive Forms	Formulaires testables et typés, validations synchrones/asynchrones, structure claire pour des UIs complexes.
Angular Material	Bibliothèque de composants accessible, responsive et cohérente visuellement. Accélère le *time-to-market* et réduit le CSS sur-mesure.

Données, Auth & Navigation

HttpClient	https://angular.io/guide/http	Appels REST typés (HttpClient + HttpInterceptor) : centraliser le header Authorization, la gestion d'erreurs et le *retry/backoff*.
Interceptors (Auth/Errors)	https://angular.io/guide/http#intercepting-requests-and-responses	Ajouter automatiquement le JWT aux requêtes, transformer les réponses d'erreur (ex : 401 → redirection login), journaliser, *caching* si besoin.
Route Guards (Auth/Role)	https://angular.io/guide/router#milestone-5-route-guards	Protéger des routes selon l'état de session/les rôles. Couplé à un service AuthService exposant un BehaviorSubject de l'utilisateur courant.

State par défaut : Services + RxJS		**Simple et suffisant** pour une app moyenne : BehaviorSubject/ReplaySubject pour l'état global (user, panier, filtres), sélecteurs comme méthodes de service. Faible courbe d'apprentissage.