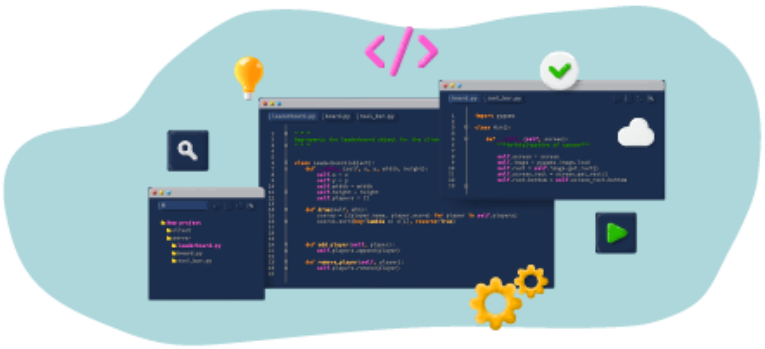
ORION



Justification des choix techniques
Projet MDD

Auteur : PERNOT Dorian
Version 0.0.1

Aperçu / Synthèse

Brièvement, exposer vos décisions principales concernant les librairies, les frameworks, les design patterns et les outils à utiliser pour ce projet.

Les décisions principales pour ce projet s'articulent autour d'une architecture full-stack moderne respectant les contraintes imposées. Pour le backend, j'ai opté pour l'écosystème Spring avec Spring Boot comme framework principal, Spring Security pour la sécurisation, Spring Data JPA pour la persistance et JWT pour l'authentification stateless. Pour le frontend, Angular 14 avec TypeScript constitue le choix principal, complété par Angular Material pour l'interface utilisateur et RxJS pour la programmation réactive. Les design patterns implémentés incluent le Repository Pattern, le DTO Pattern, l'injection de dépendances, le pattern MVC et l'Observer Pattern. Enfin, dans les outils déjà sélectionnés pour mener à bien ce projet, il y a Git/GitHub pour la gestion de versions, Maven pour le build backend et Angular CLI pour le développement frontend.

Montrer que vos choix s'alignent avec les contraintes techniques imposées.

Mes choix se sont alignés pour respecter strictement les contraintes imposées dans le document des contraintes techniques ainsi que dans la réunion de pré-projet. L'architecture backend distincte du frontend est réalisée via une API REST Spring Boot communiquant avec une SPA Angular (Single Page Application). L'interaction sécurisée est assurée par JWT et Spring Security. Les principes SOLID sont respectés grâce à l'injection de dépendances Spring et la séparation des responsabilités en couches.

Le backend implémente une architecture en couches (Layered Architecture) avec Spring Boot, structurée en quatre couches distinctes : la couche de présentation avec les contrôleurs REST (@RestController), la couche de logique métier avec les services (@Service), la couche de persistance avec les repositories (@Repository) utilisant Spring Data JPA, et la couche de données avec les entités JPA (@Entity). Cette architecture respecte le pattern MVC et facilite la maintenabilité grâce à la séparation claire des responsabilités. Le backend utilise Java/Spring avec Spring Core obligatoire pour l'Inversion de Contrôle (IoC) et l'Injection de Dépendances (DI), permettant au container Spring de gérer automatiquement la création et l'injection des objets métier (services, repositories, controllers). Spring Boot fournit l'auto-configuration qui détecte automatiquement les dépendances du projet et configure les composants nécessaires (base de données, sécurité, web) sans configuration manuelle. Spring Data JPA est choisi comme solution de persistance en priorité sur EclipseLink par exemple, conformément à la contrainte de privilégier les modules Spring.

Le frontend adopte une architecture modulaire Angular basée sur le pattern Component-Service, où les fonctionnalités sont organisées en modules dédiés (AuthModule, PostsModule, ThemesModule, ProfileModule) contenant chacun leurs composants, services et routes spécifiques. Cette architecture modulaire facilite la réutilisabilité du code, la maintenabilité et le lazy loading. Le frontend utilise TypeScript/Angular conformément aux exigences, avec respect des bonnes pratiques Angular notamment en matière de sécurité. Angular Material est intégré pour fournir une interface utilisateur cohérente et responsive respectant les standards d'accessibilité et le Material Design de Google. RxJS est utilisé pour la programmation réactive native à Angular, facilitant la gestion des appels HTTP asynchrones et des événements utilisateur grâce au pattern Observer. Les Route Guards et HTTP Interceptors Angular assurent la sécurisation des routes et l'injection automatique des tokens JWT. Angular CLI est utilisé pour le développement, conformément aux bonnes pratiques recommandées. La gestion de code utilise Git et GitHub avec un repository unique contenant frontend et backend ainsi qu'un dossier "spécifications" contenant ce document, les contraintes techniques du projet, et les spécifications fonctionnelles.

Choix techniques

Pour chaque choix (de librairies, de frameworks, de design patterns et d'outils), fournissez un lien (dont l'accès public) vers son site officiel, sa documentation ou un ouvrage de référence le détaillant.

Préciser le but des éléments choisis (par exemple : la sécurisation, l'architecture, la gestion de données...).

Puis décrire en une ou plusieurs phrases une justification du choix. Si le choix va affecter ce que Heidi avait commencé à faire, notez-le.

Choix 1

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot 2.7.3	https://mvnrepository.com/artifact/org.springframework.boot/spring-boot/2.7.3	*Architecture backend, auto-configuration, développement rapide*

Justification du choix technique : Spring Boot respecte la contrainte Java/Spring obligatoire tout en fournissant une auto-configuration qui accélère le développement. Il intègre nativement Spring Core pour l'IoC et la DI comme exigé, et permet de créer rapidement une API REST robuste avec des conventions établies. Pour ce projet MVP, une architecture monolithique avec Spring Boot est plus appropriée qu'une architecture microservices qui ajouterait une complexité inutile sans bénéfice tangible.

Choix 2

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Security avec JWT	*https://www.sfeir.dev/back/securisez-vos-api-avec-spring-security-jwt/* ou *https://spring.io/projects/spring-security*	*Sécurisation de l'application, authentification, autorisation*

Justification du choix technique : Spring Security s'intègre parfaitement dans l'écosystème Spring et permet de sécuriser l'interaction frontend-backend comme exigé. L'utilisation de JWT assure une authentification stateless adaptée à une architecture SPA, tout en maintenant la persistance de session requise par les spécifications fonctionnelles.

Choix 3

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Data JPA avec Hibernate	*https://spring.io/projects/spring-data-jpa*	*Gestion de données, mapping objet-relationnel, abstraction de la persistance*

Justification du choix technique : Spring Data JPA respecte la contrainte de prioriser les modules Spring. Il fournit une abstraction puissante de la couche de persistance avec génération automatique des requêtes, ce qui facilite l'implémentation des fonctionnalités CRUD nécessaires pour la gestion des utilisateurs, articles et abonnements.

Choix 4

choix technique	lien vers le site / la documentation / une ressource	but du choix
JSON Web Token	https://jwt.io/introduction	*Authentification stateless, sécurisation des échanges*

Justification du choix technique : JWT permet une authentification stateless compatible avec une architecture API REST et SPA. Il assure la persistance de session entre les navigations comme requis dans les spécifications fonctionnelles, tout en maintenant la sécurité des échanges entre frontend et backend.

Choix 5

choix technique	lien vers le site / la documentation / une ressource	but du choix
BCrypt	https://laconsole.dev/blog/hacher-mot-de-passe-js-bcrypt#:~:text=Bcrypt%20est%20aujourd'hui%20tr%C3%A8s,sel%20unique%20pour%20chaque%20hachage	*Sécurisation des mots de passe, protection des données utilisateurs*

Justification du choix technique : BCrypt est intégré à Spring Security et fournit un hachage sécurisé avec salt automatique. Il permet de respecter les exigences de sécurité tout en s'intégrant naturellement dans l'écosystème Spring choisi pour le projet.

Choix 6

choix technique	lien vers le site / la documentation / une ressource	but du choix
Angular 14	https://v14.angular.io/docs	*Architecture frontend, développement SPA, typage statique*

Justification du choix technique : Angular 14 respecte strictement la contrainte TypeScript/Angular imposée. Il fournit une architecture modulaire robuste avec injection de dépendances native, facilitant l'implémentation des fonctionnalités utilisateurs, d'abonnements et d'articles. TypeScript apporte la sécurité du typage statique recommandée pour des projets d'envergure.

Choix 7

choix technique	lien vers le site / la documentation / une ressource	but du choix
Angular Material	https://material.angular.dev/components/categories	*Interface utilisateur, design system, responsive design*

Justification du choix technique : Angular Material fournit des composants UI prêts à l'emploi respectant les standards d'accessibilité et le responsive design requis par les spécifications. Il accélère le développement tout en assurant une cohérence visuelle et une expérience utilisateur optimale sur mobile et desktop.

Choix 8

choix technique	lien vers le site / la documentation / une ressource	but du choix
RxJS	https://rxjs.dev/api	*Gestion des événements asynchrones, programmation réactive*

Justification du choix technique : RxJS est intégré nativement à Angular et permet de gérer efficacement les appels HTTP vers l'API, la gestion d'état de l'application et les interactions utilisateur. Il facilite l'implémentation du fil d'actualité en temps réel et la synchronisation des données entre composants.

Choix 9

choix technique	lien vers le site / la documentation / une ressource	but du choix
MySQL	https://dev.mysql.com/doc/	*Persistance des données, gestion relationnelle*

Justification du choix technique : MySQL est un SGBD relationnel mature parfaitement adapté aux relations complexes du projet (utilisateurs, thèmes, articles, abonnements, commentaires). Il s'intègre excellemment avec Spring Data JPA et offre les performances nécessaires pour les fonctionnalités de fil d'actualité et de tri chronologique.

Choix 10

choix technique	lien vers le site / la documentation / une ressource	but du choix
Repository Pattern avec Spring Data	*https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html*	*Abstraction de la couche de données, respect des principes SOLID*

Justification du choix technique : Le Repository Pattern abstrait la logique d'accès aux données et respecte le principe d'inversion de dépendance. Spring Data JPA génère automatiquement les implémentations, réduisant le code boilerplate tout en maintenant la flexibilité pour des requêtes personnalisées nécessaires au tri et filtrage des articles.

Choix 11

choix technique	lien vers le site / la documentation / une ressource	but du choix
Data Transfer Object Pattern	*https://www.baeldung.com/java-dto-pattern*	*Sécurisation des transferts de données, découplage API-modèle*

Justification du choix technique : Le DTO Pattern permet de contrôler précisément les données exposées par l'API, évitant notamment l'exposition des mots de passe utilisateurs. Il découple le modèle de données interne de l'interface publique, facilitant l'évolution indépendante et renforçant la sécurité.

Choix 12

choix technique	lien vers le site / la documentation / une ressource	but du choix
Route Guards	*https://v14.angular.io/guide/router-tutorial-toh#milestone-5-route-guards*	*Protection des routes, contrôle d'accès*

Justification du choix technique : Les Guards Angular protègent les routes sensibles conformément aux bonnes pratiques de sécurité Angular mentionnées dans les contraintes. Ils assurent que seuls les utilisateurs authentifiés accèdent aux fonctionnalités de gestion d'articles et d'abonnements, respectant ainsi le modèle de sécurité requis.

Choix 13

choix technique	lien vers le site / la documentation / une ressource	but du choix
HTTP Interceptors Angular	*https://v17.angular.io/guide/http-interceptor-use-cases*	*Ajout automatique des tokens d'authentification*

Justification du choix technique : Les interceptors automatisent l'ajout des tokens JWT à toutes les requêtes HTTP vers l'API, simplifiant la gestion de l'authentification côté frontend. Cette approche respecte les bonnes pratiques Angular et assure une sécurisation transparente de toutes les communications avec le backend.

Choix 14

choix technique	lien vers le site / la documentation / une ressource	but du choix
Component-Service Pattern	*https://v17.angular.io/guide/architecture-services*	*Séparation des responsabilités, réutilisabilité, testabilité*

Justification du choix technique : Le pattern Component-Service sépare la logique de présentation (composants) de la logique métier (services), respectant le principe de responsabilité unique des principes SOLID. Les composants gèrent uniquement l'affichage et les interactions utilisateur, tandis que les services encapsulent la logique métier, les appels API et le partage de données entre composants. Cette architecture facilite la réutilisabilité du code, améliore la testabilité et permet une maintenance plus aisée du code frontend.