

# # P6 - FullStack app - Document d'architecture

## Repository

Repo sur Github →

Meilleur expérience de ce document → <https://docs.maxdlr.com/s/6bee81d8-1ddc-4c52-badc-b1d2b342f1cc>

## Arborescence des fichiers

```
1 .
2 |— README.md
3 |— back
4 |— client-resources
5 |— front
6 |— render
```


## Architecture

 3-tiers

L'architecture en 3 couches, 3 micro-services, permet entre autre de séparer les responsabilités de chaque aspect de l'application, favoriser la maintenabilité et la facilité au développement.


### Couche DATA

Base de donnée **Mariadb**.

 La DB à été déployée dans un conteneur Docker pour éviter les problèmes de configuration d'un ordinateur à un autre.

### Base de donnée

Figma

 Open

## Couche Logique

 Back-end développé en Java, sur le Framework Spring Boot.

### Contents

#### Repository

Arborescence des fichiers

#### Architecture

Couche DATA

Base de donnée

Couche Logique

Choix techniques

Authentification

Métier

Test Driven Development

Couche Présentation

Organisation

Choix techniques

Méthode de développement

Inspiré Agile

User Stories

Sprints

Voici la réponse réécrite sous forme de tableau selon les en-têtes demandés :

Choix techniques

Différents aspect de la couche logique ont utilisé different librairie de classes, services et packages.

Tâche	Choix	Version	Liens	But du choix	Justification du choix
Création et gestion de l'application	Spring Boot	3.3.4	Spring Boot	Simplifier le développement et la configuration des applications.	Spring Boot est plus opinionné que Dropwizard, ce qui réduit la configuration manuelle, tout en offrant une meilleure intégration avec d'autres outils du stack Spring.
Validation des données	Hibernate Validator	8.0.0.Final	Hibernate Validator	Assurer l'intégrité des données grâce à des annotations.	Plus intégré avec JPA que Bean Validation d'Apache BVal, Hibernate Validator est une solution mature qui utilise les spécifications Java EE tout en bénéficiant de mises à jour fréquentes.
Mapping entre objets (DTO ↔ Entity)	MapStruct	1.6.3	MapStruct	Automatiser les conversions d'objets pour réduire le code répétitif.	MapStruct est plus performant que ModelMapper car il génère du code statique au lieu de faire des conversions à l'exécution, ce qui améliore la vitesse et la fiabilité.
Sécurisation de l'application	Spring Security	N/A	Spring Security	Gérer l'authentification et l'autorisation.	Plus flexible qu'Apache Shiro, Spring Security s'intègre parfaitement avec Spring Boot, permettant une configuration et une personnalisation rapides pour différents niveaux de sécurité.
Stockage persistant	MySQL Connector	8.0.32	MySQL Connector	Permettre la communication avec une base de données MySQL.	MySQL Connector est spécifiquement optimisé pour MySQL, tandis qu'une alternative comme PostgreSQL JDBC est adaptée pour PostgreSQL, mais moins compatible dans ce contexte précis.
Environnement de test	H2 Database	N/A	H2 Database	Fournir une base de données légère en mémoire pour les tests.	H2 est plus rapide à configurer que SQLite pour des tests, avec des fonctionnalités adaptées aux scénarios en mémoire et une compatibilité avec les tests unitaires dans un environnement Java.
Analyse de la couverture de tests	Jacoco Maven Plugin	0.8.12	Jacoco	Mesurer la couverture des tests unitaires.	Jacoco est plus léger et plus facile à intégrer à Maven que Cobertura, tout en générant des rapports lisibles et compatibles avec plusieurs outils de CI/CD.
Réduction du boilerplate code	Lombok	1.18.36	Lombok	Automatiser la génération de getters, setters et autres méthodes.	Lombok est plus facile à utiliser que Immutables, car il nécessite moins de configuration tout en offrant des annotations simples pour générer des classes concises et lisibles.

Authentication

L'authentification sera g  r   par `jwt`, cette m  thode offrira une s  curit   robuste et flexible. Ces informations seront stock   dans un cookie, cot   client. La validit   du token sera v  rifi   a chaque requ  te.

M  tier

Apr  s compr  hension du m  tier de la demande client, une liste d'endpoints est   tablie.

Fonctionnalit��	Endpoint	Methode	Request Payload	Response Body
<div>Register</div> <div><ul style="list-style-type: none"><li>Cr��ation d'un compte utilisateur</li></ul></div>	<code>/api/auth/register</code>	POST	<pre>1 { 2   email: 3   string, 4   username: 5   string, 6   password: 7   string 8 }</pre>	<pre>1 { 2   message: string 3 }</pre>
<div>Authenticate</div> <div><ul style="list-style-type: none"><li>Authentification d'un utilisateur par <code>Bearer</code> <code>jwt</code>.</li></ul></div>	<code>/api/auth/login</code>	POST	<pre>1 { 2   email: 3   string, 4   password: 5   string 6 }</pre>	<pre>1 { 2   token: string, 3   id: number, 4   email: string, 5   username: string 6 }</pre>
<div>Read user information</div> <div><ul style="list-style-type: none"><li>Consulter les informations d'un utilisateur</li></ul></div>	<code>/api/users/{id}</code>	GET		<pre>1 { 2   id: number, 3   email: string, 4   username: string, 5   subscriptions: 6   Subscription[] 7   created_at: Date, 8   updated_at: Date 9 }</pre>
<div>Edit user information</div> <div><ul style="list-style-type: none"><li>Editer les informations d'un utilisateur</li></ul></div>	<code>/api/users/{id}</code>	PUT	<pre>1 { 2   email:string, 3   username: 4   string, 5   password: 6   string 7 }</pre>	<pre>1 { 2   id: number, 3   email: string, 4   username: string, 5   subscriptions: 6   Subscription[] 7   created_at: Date, 8   updated_at: Date 9 }</pre>

<div>Browse themes</div> <ul style="list-style-type: none"><li>Consulter la liste de tous les thème.</li></ul>	/api/themes	GET		<pre>1 [ 2   { 3     name: string, 4   } 5 ]</pre>
<div>Subscribe</div> <ul style="list-style-type: none"><li>Associer un utilisateur à des thèmes</li></ul>	/api/subscriptions/subscribe	POST	<pre>1 { 2   userId: 3   number, 4   themeId:    number }</pre>	<pre>1 200</pre>
<div>Unsubscribe</div> <ul style="list-style-type: none"><li>Dissocier un utilisateur d'un thème.</li></ul>	/api/subscriptions/unsubscribe	POST	<pre>1 { 2   userId: 3   number, 4   themeId:    number }</pre>	<pre>1 200</pre>
<div>Create article</div> <ul style="list-style-type: none"><li>Créer un article</li></ul>	/api/articles/	POST	<pre>1 { 2   title: 3   string, 4   themeId: 5   number, 6   authorId:    number,    content:    string }</pre>	<pre>1 200</pre>
<div>Read article</div> <ul style="list-style-type: none"><li>Consulter un article</li></ul>	/api/articles/{id}	GET		<pre>1 { 2   id: number, 3   title: string, 4   themeId: number, 5   authorId: number, 6   content: string, 7   createdAt: Date, 8   updatedAt: Date 9 }</pre>
<div>Browse articles by user subscriptions</div> <ul style="list-style-type: none"><li>Consulter la liste de tous les articles correspondants aux abonnement de l'utilisateur.</li></ul>	/api/articles	GET		<pre>1 [ 2   { 3     id: number, 4     title: string, 5     themeId: number, 6     authorId: 7   number, 8     content: string, 9   }</pre>

				<pre>9      createdAt: Date, 10      updatedAt: Date 11    }     ]</pre>
<div>Comment</div> <ul style="list-style-type: none"><li>Ajouter un commentaire a un article</li></ul>	/api/comments	POST	<pre>1 { 2   content: 3 string, 4   authorId: 5 number,    articleId:    number }</pre>	<pre>1 200</pre>

Test Driven Development

Pour s’assurer de la robustesse des fonctionnalités développées, le projet sera développé en **TDD**.

Le **TDD** impose une méthode qui se décrit avec quelques points simple. Bien sûr, ces méthodes sont adaptées au compétence du développeur et de la pertinence des tests.

- Ecrire le test avant la fonctionnalité
  - Dans un premier temps, le test doit échouer.
  - La fonctionnalité est développée dans le but de faire passer le test uniquement.
- La fonctionnalité doit contenir uniquement le code faisant passer le test.
  - Si du code n’est pas couvert par test, c’est qu’il n’a pas lieu d’exister.
  - Si du code n’étant pas couvert par le test est obligatoire, le test doit être mis à jour avant la fonctionnalité.

Ce choix émane d’une certaine incertitude concernant mes compétences en Java/Spring boot. N’étant pas sur à 100% que mon code fonctionnera du premier coup, j’ai décidé d’écrire des test d’intégrations avant de codé ladite intégration. Les tests me permettront d’analyser les besoins métier et réfléchir tout de suite aux retours logiques que sont sensés renvoyer mes controllers et services. Dans un second temps, je coderai uniquement le code nécessaire pour faire passer mes tests pré-rédigés.

Je prévois de générer un rapport de couverture de code avec **Jacoco** et viser une couverture minimal de 65%, avant de m’attaquer à la couche Front.

Couche Présentation

**i** La couche de front-end est développé en Typescript sur le framework **Angular**.

## Organisation

Des composants, interceptors et autres services généraux seront créés pour faciliter l'intégration d'éléments de base.

Composants	Services	Autres
<ul style="list-style-type: none"><li>DialogComponent</li></ul>	<ul style="list-style-type: none"><li>AuthService</li><li>AuthGuard</li><li>NavigationService</li><li>SnackService</li><li>UserService</li><li>ArticleService</li><li>CommentService</li><li>SessionService</li><li>SubscriptionService</li><li>ThemeService</li></ul>	<ul style="list-style-type: none"><li>JwtInterceptor</li><li>AuthInterceptor</li></ul>

Les composants et services spécifiques de l'application front-end seront organisé en modules.

Module	Composants
AuthModule	<ul style="list-style-type: none"><li>AuthBaseComponent</li><li>LoginComponent</li><li>RegisterComponent</li><li>MeComponent</li></ul>
NavigationModule	<ul style="list-style-type: none"><li>BaseComponent</li><li>DesktopMenuComponent</li><li>MobileMenuComponent</li></ul>
ArticleModule	<ul style="list-style-type: none"><li>ArticleCardComponent</li><li>ArticleListComponent</li><li>ArticleReadComponent</li><li>ArticleFormComponent</li></ul>
ThemeModule	<ul style="list-style-type: none"><li>ThemeCardComponent</li><li>ThemeListComponent</li></ul>

## Choix techniques

Différents aspect de la couche logique ont utilisé different librairie de classes, services et packages.

Tâche	Choix	Version	Liens	But du choix	Justification du choix
-------	-------	---------	-------	--------------	------------------------

Développement du framework frontend	Angular	19.0.0	<a href="#">Angular</a>	Fournir un framework robuste et structuré pour créer des applications web SPA.	Angular, contrairement à React ou Vue.js, offre une architecture complète et opinionnée, idéale pour des projets nécessitant une forte structure et un bon typage TypeScript natif.
Gestion des animations	Angular Animations	19.0.0	Angular Animations	Ajouter des animations complexes dans l'application.	Intégré directement à Angular, il évite les dépendances tierces comme GSAP, avec une configuration native et cohérente dans le cadre Angular.
Composants UI	Angular Material	19.0.0	Angular Material	Fournir une bibliothèque de composants UI respectant les guidelines Material Design.	Angular Material est plus aligné avec Angular qu'Ant Design ou PrimeNG, offrant des performances accrues grâce à son intégration native avec Angular CDK.
Gestion des cookies	ngx-cookie-service	19.0.0	<a href="#">ngx-cookie-service</a>	Gérer facilement les cookies dans les applications Angular.	Préféré à js-cookie pour sa compatibilité avec Angular, ngx-cookie-service exploite les services Angular pour une utilisation intuitive et un typage strict.
Sécurisation JWT	jwt-decode	4.0.0	<a href="#">jwt-decode</a>	Décoder et lire les informations contenues dans un token JWT côté client.	jwt-decode est plus léger et plus spécialisé que des bibliothèques comme Auth0, ce qui le rend idéal pour des besoins simples et rapides de traitement des JWT dans un projet Angular.
Utilitaires de manipulation de données	Lodash	4.17.21	<a href="#">Lodash</a>	Faciliter les opérations complexes sur les objets et tableaux.	Lodash offre une API plus complète et performante que les fonctions utilitaires natives de JavaScript, bien que certaines alternatives comme Ramda soient plus fonctionnelles mais moins couramment utilisées.
Gestion des styles	Tailwind CSS	3.4.15	<a href="#">Tailwind CSS</a>	Ajouter un système de design utilitaire pour une personnalisation rapide des interfaces.	Tailwind CSS est plus flexible que Bootstrap, permettant une personnalisation fine sans dépendre de classes préconçues ou de structures rigides.
Tests unitaires	Jasmine + Karma	~4.2.0	Jasmine / Karma	Fournir une suite pour écrire et exécuter des tests unitaires et d'intégration.	Plus adaptés à Angular que Jest, Jasmine et Karma s'intègrent nativement avec Angular CLI et Angular Testing Utilities pour une configuration simplifiée.
Linting et qualité de code	ESLint	9.15.0	<a href="#">ESLint</a>	Vérifier et maintenir un code propre et uniforme.	ESLint, associé à @angular-eslint, offre une meilleure intégration avec Angular que TSLint, aujourd'hui déprécié, tout en offrant une flexibilité accrue pour personnaliser les règles.



Formatage de code	Prettier	3.3.3	<u>Prettier</u>	Assurer une mise en forme cohérente du code source.	Préféré à Beautify, Prettier s'intègre mieux avec les outils modernes comme VSCode et Angular CLI, offrant une configuration plus simple et des performances optimisées.
Gestion des dépendances	Angular CLI	19.0.1	Angular CLI	Simplifier le démarrage, la configuration et la gestion du projet Angular.	Angular CLI, spécifique à Angular, surpasse des outils génériques comme Create React App dans un contexte Angular, grâce à ses commandes natives pour générer des composants ou des services.

## Méthode de développement

Après lecture de la demande client et compréhension du métier, mise en place d'un backlog détaillé. Ce backlog me permettra de resté organisé tout au long de mon sprint.

**i** Le BackLog a été créé sur **Github Projects**.  
Création d'un tableau organisé, visible de plusieurs manière, étroitement reliée a mes pull-request et issues.  
Voir →

## Inspiré Agile

### User Stories

Description d'une **User Story** selon le point de vue d'un utilisateur.

### Sprints

**i** Les sprints sont représentés sous forme de *Milestone* sur dans le BackLog.

Séparation des tâches **Back-end** et **Front-end** pour permettre de ne pas changer de langage en permanence et être sûr que la couche logique est déjà développée au moment du développement du front.

### Aperçu

Liste complete des User Story à implémenter

Détails d'une issue incorporé dans le GH Project

## Conclusion

J'ai trouvé ce projet passionnant.



Beaucoup d'apprentissage a été tiré de ce projet tant sur **Java** et l'environnement **Spring** mais aussi celui d'**Angular** et **Angular Material**.

Ayant fait mes débuts sur **PHP** et **Symfony** et l'environnement front-end de **Twig**. Je constate chez moi une préférence forte pour **Java/Angular**.

---