Auteur : Régereau Julie
Version : 1.0.2
Date : 20/01/2026

Version 1.0.2

Documentation et rapport du projet MDD

1. Présentation générale du projet

1.1 Objectifs du projet

Contexte
Monde de Dév (MDD) est un réseau social pensé pour les développeurs, offrant un espace de partage d'articles techniques et de suivi de thématiques spécialisées. Le projet s'inscrit dans une logique d'apprentissage full-stack, mêlant développement front-end avec Angular et back-end avec Spring Boot.

Besoins métiers

- *Permettre aux développeurs de créer un compte et d'accéder à la plateforme de manière sécurisée.*
- *Proposer un espace dédié à la publication d'articles techniques, classés par thèmes.*
- *Offrir la possibilité de s'abonner à des thématiques afin de personnaliser le fil d'actualité.*
- *Favoriser les échanges grâce à un système de commentaires.*
- *Assurer la gestion des profils utilisateurs, avec options de modification.*

Valeur ajoutée
La plateforme centralise les contenus techniques et propose un filtrage par thèmes, permettant aux développeurs de suivre uniquement les sujets qui les intéressent. Cela limite la surcharge d'informations et encourage des échanges professionnels ciblés.

Fonctionnalités principales

1. *Inscription et authentification sécurisée via des tokens JWT signés en HMAC SHA-256.*
2. *Gestion du profil utilisateur, avec consultation et modification de l'email, du nom d'utilisateur et du mot de passe.*
3. *Publication et lecture d'articles, avec un CRUD complet et une gestion des auteurs.*
4. *Abonnement et désabonnement à des thèmes pour personnaliser le fil d'actualité.*
5. *Fil d'actualité personnalisé, affichant les articles des thèmes suivis, triés par date.*
6. *Système de commentaires permettant d'ajouter et de consulter les réactions liées à chaque article.*

1.2 Périmètre fonctionnel

Fonctionnalités	Description	Statut
Inscription d'un compte	Formulaire avec validation (email, username, password 8+ chars avec majuscule, minuscule, chiffre, caractère spécial)	Terminée
Connexion (login)	Authentification via email/username + password, retour JWT	Terminée
Déconnexion (logout)	Suppression du token et redirection vers home	Terminée
Consultation du profil	Affichage des infos utilisateur (email, username, abonnements)	Terminée
Modification du profil	Édition email, username, password avec validation	Terminée
Consultation de la liste des thèmes	Affichage de tous les thèmes (abonné ou non)	Terminée
S'abonner à un thème	Association utilisateur ↔ thème, mise à jour UI	Terminée
Se désabonner d'un thème	Suppression association, mise à jour UI	Terminée
Consultation du fil d'actualité	Affichage des articles des thèmes suivis, triés par date (DESC par défaut)	Terminée
Trier le fil d'actualité	Toggle récent→ancien ou ancien→récent	Terminée
Créer un article	Formulaire (thème, titre, contenu), auto author/date	Terminée
Consulter un article	Page détail avec thème, titre, auteur, date, contenu, commentaires	Terminée
Ajouter un commentaire	Formulaire contenu, auto author/date	Terminée
Lire les commentaires	Affichage des commentaires d'un article	Terminée

Responsive design	Interface adaptée mobile, tablette, desktop	Terminée
Sécurité JWT	Tokens signés HMAC SHA-256, expiration 24h	Terminée

2. Architecture et conception technique

2.1 Schéma global de l'architecture

Diagramme de séquence voir annexe 6

Organisation technique

Front-end (Angular 14, TypeScript)

- Structure modulaire organisée par fonctionnalités : home, login, register, feed, article, themes, profile
- Services dédiés pour centraliser les appels API
- Guards pour sécuriser les routes nécessitant une authentification
- Interceptors chargés d'ajouter automatiquement le token JWT aux requêtes
- Stockage du token dans le localStorage

Back-end (Spring Boot 2.7.3, Java 11)

- Architecture en trois couches : Controllers → Services → Repositories
- Utilisation de DTO pour séparer les entités internes des données échangées
- Accès aux données via Spring Data JPA
- Sécurisation avec Spring Security et JWT
- Validation des données avec `@Valid` et les annotations JPA

Base de données (MySQL)

- Modèle relationnel composé de cinq tables
- Relation many-to-many entre les utilisateurs et les thèmes
- Contraintes d'intégrité : clés primaires, clés étrangères, unicité

Sécurité

- JWT signé en HMAC SHA-256 avec une durée de validité de 24 heures

- Mots de passe hashés avec BCrypt
- Configuration CORS limitée au front-end
- Authentification via le header `Authorization: Bearer`

2.2 Choix techniques

Éléments choisis	Type	Lien documentation	Objectif du choix	Justification
Angular 14	Framework front-end	angular.io] (https:// angular.io)	Structuration de l'application SPA et gestion de la réactivité de la réactivité	Framework moderne, components standalone, RxJS intégré, écosystème complet et présent dans le starter
TypeScript 4.7.4	Langage front-end	typescriptlang.org](https:// www.typescriptlang.org)	Typage statique et détection des erreurs précoces	Sécurité au build, meilleure expérience développement, lisibilité du code, améliore l'expérience développeur
Angular Material 14	Composants UI	material.angular.io](https:// material.angular.io)	Composants UI cohérents, accessibles, pré-stylisés	Composants UI cohérents, accessibles et prêts à l'emploi. Respect du design Material, responsive par défaut
RxJS 7.5.6	Programmation réactive	rxjs.dev](https:// rxjs.dev)	Gestion des streams et des appels asynchrones	Operators puissants pour gérer les flux asynchrones (map, filter, takeUntil), gestion mémoire (unsubscribe)
Jest 28.1.3	Framework de tests front-end	jestjs.io](https:// jestjs.io)	Typage statique et détection des erreurs précoces Tests unitaires rapides et isolés	Tests unitaires rapides et isolés, snapshots intégrés, mocking simple et couverture native. Très performant pour les tests front-end.
Spring Boot 2.7.3	Framework back-end	spring.io] (https://spring.io)	API REST robuste, sécurisée, scalable	Écosystème mature, Spring Security intégré, auto-configuration et présent dans le starter.

Java 11	Langage back-end	java.com](https://www.java.com)	Langage typé, performant, JVM fiable	LTS stable, support long terme, performance production
Spring Data JPA	ORM	typescriptlang.org](https://www.typescriptlang.org) spring.io/projects/spring-data-jpa](https://spring.io/projects/spring-data-jpa)	Accès simplifié aux données relationnelles	Simplifie l'accès aux données relationnelles. Génération automatique du CRUD, requêtes SQL gérées par le framework.
Spring Security	Framework sécurité	typescriptlang.org](https://www.typescriptlang.org) spring.io/projects/spring-security](https://spring.io/projects/spring-security)	Authentification et autorisation des endpoints	Standard de sécurité dans l'écosystème Spring. Gestion complète de l'authentification/autorisation, filtres intégrés, protection CSRF
JWT (JJWT 0.11.5)	Authentification	jwt.io](https://jwt.io)	Authentification stateless sécurisée	Standard moderne, tokens signés, revocation facile, scalabilité
BCrypt	Hachage password	spring.io](https://spring.io)	Sécurisation irréversible des mots de passe	Hachage robuste et irréversible, avec salts aléatoires. Coût de calcul élevé pour contrer les attaques par force brute
MySQL 8.0+	Base de données	mysql.com](https://dev.mysql.com)	Stockage relationnel fiable et performant	BD relationnelle mature, bonnes performances

Maven	Build tool	maven.apache.org](https://maven.apache.org)	Gestion des dépendances et build Java	Standard industrie, plugin riche, reproducibilité
JUnit 5	Framework tests back-end	typescriptlang.org](https://www.typescriptlang.org) junit.org/junit5] (https://junit.org/junit5)	Tests unitaires et d'intégration back-end	Annotations expressives, @DisplayName, Mockito intégré
Mockito	Mocking library	typescriptlang.org](https://www.typescriptlang.org) mockito.org] (https://mockito.org)	Création de mocks pour les tests	Syntax intuitive, verification d'interactions, when/then
Git & GitHub	Version control	github.com] (https://github.com)	Collaboration et historique de code	Gestion de versions fiable, collaboration facilitée, branches, pull requests et intégration CI/CD
VS Code	IDE	typescriptlang.org](https://www.typescriptlang.org) code.visualstudio.com](https://code.visualstudio.com)	Développement optimisé	IDE léger et performant, autocomplétion avancée, debugging intégré et large catalogue d’extensions.

2.3 API et schémas de données

L'API respecte les principes REST et est organisée par domaines fonctionnels :

- **Authentification** (/api/auth) : gestion de l'inscription et de la connexion (endpoints publics).
- **Utilisateurs** (/api/users) : consultation et modification des profils (endpoints protégés par JWT).
- **Thèmes** (/api/themes) : consultation des thèmes et gestion des abonnements (endpoints protégés par JWT).
- **Articles** (/api/articles) : création, lecture, mise à jour, suppression et récupération du fil d'actualité (endpoints protégés par JWT).
- **Commentaires** (/api/articles/{id}/comments) : ajout et consultation des commentaires liés à un article (endpoints protégés par JWT).

Sécurité

Tous les endpoints, à l'exception de ceux situés sous /api/auth/*, exigent un token JWT valide transmis dans le header Authorization: Bearer .

Format des échanges

Les requêtes et réponses utilisent systématiquement le format JSON encodé en UTF-8.

Codes HTTP utilisés

200 (OK), 201 (Created), 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), 500 (Internal Server Error).

Endpoint	Méthode	Description	Corps / Réponse
/api/auth/register`	POST	Inscription d'un nouvel utilisateur (public)	JSON – RegisterRequest → AuthResponse (201)
/api/auth/login	POST	Connexion, retourne un JWT	JSON – LoginRequest → AuthResponse (200)
/api/users/me	GET	Récupère le profil de l'utilisateur connecté (JWT requis)	JSON – UserResponse (200)
/api/users/me	PUT	Modifie le profil de l'utilisateur connecté (JWT requis)	JSON – UserUpdateRequest → UserResponse (200)
`/api/themes`	GET	Liste de tous les thèmes disponibles (JWT requis)	JSON – liste de ThemeResponse (200)
/api/themes/{id}/	POST	S'abonner à un thème par	JSON – Message de

subscribe		son ID (JWT requis)	confirmation (200)
/api/themes/{id}/ unsubscribe	DELETE	Se désabonner d'un thème par son ID (JWT requis)	JSON – Message de confirmation (200)
/api/articles	GET	Récupère le fil d'actualité (articles des thèmes abonnés, triés par date) (JWT requis)	JSON – liste d'ArticleResponse (200)
/api/articles/{id} `	GET	Récupère le détail d'un article avec ses commentaires (JWT requis)	JSON – ArticleDetailResponse (200)
/api/articles/{id}	*DELETE*	*upprime un article par son ID (JWT requis)*	*JSON – Message de confirmation (200)*
/api/articles/{id}/ comments	*GET*	*Liste les commentaires d'un article (JWT requis)*	*JSON – liste de CommentResponse (200)*
/api/articles/{id}/ comments	POST	Ajoute un commentaire à un article (JWT requis)	JSON – CreateCommentRequest → CommentResponse (201)

Exemples de requêtes et réponses JSON :

POST /api/auth/register

```json
// Requête
{
 "username": "julie",
 "email": "julie@example.com",
 "password": "MonPassword123!"
}

// Réponse (201 Created)
{
 "id": 1,
 "username": "julie",
 "email": "julie@example.com",
 "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzAzMDg..."
}
```

```

**POST /api/auth/login**
```json
// Requête
{
 "emailOrUsername": "julie@example.com",
 "password": "MonPassword123!"
}

// Réponse (200 OK)
{
 "id": 1,
 "username": "julie",
 "email": "julie@example.com",
 "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzAzMDg..."
}
```

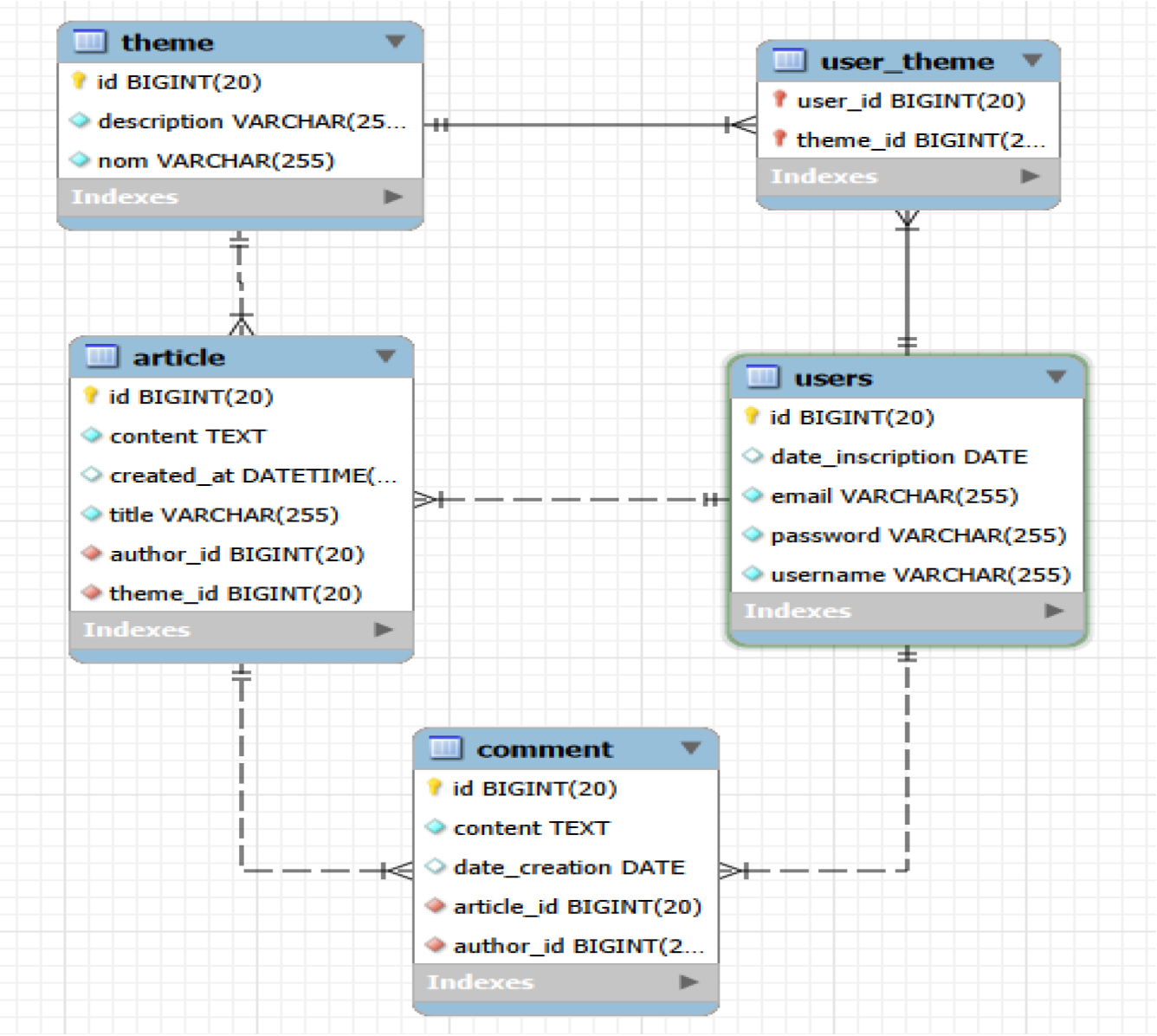
GET /api/articles (avec header `Authorization: Bearer <token>`)
```json
// Réponse (200 OK)
[
 {
  "id": 1,
  "title": "Démarrer avec Angular 14",
  "content": "Angular 14 est sorti avec de nouvelles features...",
  "author": {
   "id": 2,
   "username": "dev_pro"
  },
  "theme": {
   "id": 1,
   "name": "Angular"
  },
  "createdAt": "2026-01-15T10:30:00Z",
  "comments": [
   {
    "id": 1,
    "content": "Merci pour cet article !",
    "author": {
     "id": 3,
     "username": "john_dev"
    },
    "createdAt": "2026-01-15T11:00:00Z"
   }
  ]
 }
```
```

]
```

**POST /api/articles** (Créer un article)
```json
// Requête
{
 "title": "Spring Boot 3.x Migration",
 "content": "Guide complet de migration de Spring Boot 2.x à 3.x...",
"themeId": 2
}

// Réponse (201 Created)
{
 "id": 5,
 "title": "Spring Boot 3.x Migration",
 "content": "Guide complet de migration...",
 "author": {
  "id": 1,
  "username": "julie"
 },
 "theme": {
  "id": 2,
  "name": "Spring Boot"
 },
 "createdAt": "2026-01-20T14:25:00Z",
 "comments": []
}
```
```

chéma relationnel de la base de données (ERD)



Relation M:N (Many-to-Many) :

- Users ↔ Themes via user_theme → Un utilisateur s'abonne à plusieurs thèmes

Relations 1:N (One-to-Many) :

- Un utilisateur crée plusieurs articles et commentaires
- Un thème contient plusieurs articles
- Un article reçoit plusieurs commentaires

3. Tests, performance et qualité

3.1 Stratégie de test

Ma méthode de travail

J'ai écrit les tests au fil du développement, en particulier dès qu'une fonctionnalité introduisait de la logique métier. Cette approche m'a permis d'identifier rapidement les erreurs et de sécuriser progressivement le code.
Malgré cette vigilance, certains tests ont échappé aux premières itérations, ce qui m'a conduit à en ajouter un nombre plus important en fin de parcours afin d'atteindre le niveau de couverture attendu et de garantir la fiabilité globale de l'application.

Si tu veux, je peux aussi t'aider à formuler la partie sur la couverture, les outils ou les limites rencontrées.

Approche : Tests unitaires avec mocks

Principe

L'objectif est de tester chaque partie du code **de manière isolée**, sans dépendre des autres couches de l'application.
Pour cela, j'utilise des **mocks** afin de simuler les dépendances externes (base de données, services tiers, etc.).

Exemple concret

Pour tester `UserService`, je ne souhaite pas m'appuyer sur la vraie base MySQL.
J'utilise donc **Mockito** pour simuler le `UserRepository` :

```
@Test
void testFindUserByEmail() {
  // Simulation de la base de données
  User mockUser = new User();
  mockUser.setEmail("test@test.com");
  when(userRepository.findByEmail("test@test.com")).thenReturn(mockUser);

  // Appel du service
  User result = userService.findByEmail("test@test.com");

  // Vérification du résultat
  assertEquals("test@test.com", result.getEmail());
}
```

Avantages

- Tests **rapides** : aucune base MySQL à démarrer.
- Tests **fiables** : pas d'effets de bord, pas de dépendance à l'environnement.
- Tests **précis** : chaque classe est validée indépendamment.

Tests manuels

En complément des tests automatisés, j'ai également réalisé des tests manuels dans le navigateur pour valider les parcours utilisateurs :

- Inscription → Connexion → Création d'un article → Commentaire → Déconnexion
- Tests effectués sur **Chrome** et **Firefox**
- Vérification du **responsive** en mode mobile

Ces tests m'ont permis de valider l'expérience utilisateur et de détecter des comportements non couverts par les tests unitaires.

Exemples de tests

Backend – Test d'inscription utilisateur :

```
@Test
void testRegisterUser() {
  User user = new User();
  user.setUsername("test");
  user.setEmail("test@test.com");

  User saved = userService.save(user);

  assertNotNull(saved.getId());
}
```

Frontend – Test de connexion :

```
it('doit se connecter avec succès', () => {
  const mockResponse = { token: 'abc123' };

  authService.login(email, password).subscribe(result => {
    expect(result.token).toBe('abc123');
  });
});
```

Résultats obtenus :

Type de test	Outil / framework	Portée	Résultats
Back-end unitaires	JUnit 5 + Mockito	Services, SecurityUtils | 43 tests	100% passing

Front-end unitaires	Jest + @Angular/core/testing	Services, Guards, Components 82 tests	100% passing
Tests de couverture	JaCoCo (back) / Jest (front)	Code coverage	Back: 65% / Front: 82.8%

3.2 Rapport de performance et optimisation

La performance a été prise en compte dès la conception de l'application, en privilégiant des choix techniques simples et adaptés à un MVP.

Des tests manuels ont été réalisés afin d'évaluer le comportement global de l'application :
• analyse des temps de chargement via Chrome DevTools (onglet Network)
• vérification de la stabilité de l'application après plusieurs minutes d'utilisation continue
• tests sur mobile afin de valider le bon fonctionnement du responsive
• observation du temps d'exécution des tests automatisés (environ 3 secondes pour le backend et 5 secondes pour le frontend)

Ces vérifications montrent que l'application est rapide et fluide, avec un temps de chargement moyen compris entre 1 et 2 secondes et une navigation sans ralentissement notable.

Cette performance est notamment liée aux choix techniques suivants :
• API REST stateless sécurisée par JWT
• validation des données côté front et côté back pour limiter les requêtes inutiles
• indexation MySQL sur les champs fréquemment recherchés
• chargement des pages Angular uniquement lors de la navigation

Points d'amélioration identifiés et actions appliquées

Je n'ai pas encore mis en place d'optimisations avancées, car je ne maîtrise pas certains outils spécialisés (Lighthouse, SonarQube, Redis, CDN, etc.).
En revanche, j'ai appliqué plusieurs ***bonnes pratiques essentielles*** *qui contribuent déjà à de bonnes performances.*

Ce que j'ai mis en place

Frontend (Angular)

- *Responsive design : utilisation de CSS pour adapter l'interface aux formats mobile, tablette et desktop*
- *Validation des formulaires : contrôle des champs côté client avant l'envoi au serveur, limitant les erreurs et les requêtes inutiles*

Backend (Spring Boot)

- *Validation avec `@Valid` : contrôle automatique des données entrantes (email valide, mot de passe minimum, etc.)*
- *Index MySQL automatiques : les colonnes `email` et `username`, définies comme `UNIQUE`, bénéficient d'index améliorant les performances de recherche*

Résultat

*Même sans optimisation avancée, l'application reste **performante** :*

- *chargement rapide*
- *navigation fluide*
- *aucun ralentissement notable*

3.3 Revue technique

Ce que j'ai réussi

✔ Application fonctionnelle

- Toutes les fonctionnalités sont opérationnelles : inscription, connexion, articles, commentaires
- Aucun bug bloquant identifié
- 125 tests réussis (43 backend + 82 frontend)

✔ Architecture 3-couches maîtrisée

- Pattern Controllers → Services → Repositories respecté
- Structure claire qui m'a aidée à organiser le code proprement

✔ Sécurité de base en place

- Authentification via JWT
- Mots de passe hashés avec BCrypt
- Validation des formulaires côté front et côté back

Mes difficultés et ce que j'ai appris

🔴 Coverage à 64 % (objectif : 70 %)

- J'ai mal suivi l'évolution du coverage pendant le développement
- Je me suis rendu compte trop tard que j'étais en dessous de la cible
- Les tests existants fonctionnent, mais il en manque pour atteindre le seuil
- **Leçon : surveiller la couverture en continu, pas uniquement en fin de projet**

🔴 Cypress abandonné

- Incompatibilité de version connue entre Cypress et mon environnement
- Après recherches, j'ai constaté que d'autres développeurs rencontraient le même problème
- Une solution aurait été de créer un dossier dédié avec une configuration spécifique

- Par manque de temps, j'ai privilégié les tests d'intégration Spring, plus stables
- **Leçon : anticiper les problèmes de compatibilité et prévoir du temps pour les résoudre**

🔴 **Discipline Git à renforcer**

- Lors de la relecture du projet, j'ai identifié un point d'amélioration concernant la clarté et la précision de certains messages de commit. Cette analyse m'a permis de mieux comprendre l'importance de commits plus ciblés et explicites, afin de faciliter la lecture de l'historique du projet.
- Cette bonne pratique a depuis été intégrée et appliquée sur mes projets suivants.

Concepts non utilisés ou utilisés différemment

Lazy loading Angular

- J'utilise bien le lazy loading via `loadComponent` (standalone components)
- Je n'utilise pas `loadChildren` car je n'ai pas de modules Angular
- Chaque page est chargée uniquement lors de la navigation → comportement attendu

Logs

- La plupart des logs ont été retirés du code final
- Utilisation uniquement en développement pour le débogage
- Pas encore de système de logs structuré pour ce MVP

Mes choix techniques pour apprendre

Au début, mes tests dépendaient trop de la base de données, ce qui les rendait lents et instables.
Pour améliorer cela :

- J'ai créé un **DataInitializer** (profil `!test`) pour charger quelques données en développement
- J'ai appris à utiliser **Mockito** pour mocker les repositories
- Résultat : des tests unitaires plus rapides, plus isolés et plus fiables

Ce que je continue d'apprendre

Le développement est un **processus d'apprentissage continu** :

- Chaque fonctionnalité m'a permis d'appronfondir les concepts
- J'ai fait des erreurs, mais elles m'ont aidée à progresser
- J'ai choisi certaines technologies pour pratiquer et monter en compétence

Conclusion honnête

L'application est fonctionnelle, les tests passent et l'architecture est propre.
Il reste des axes d'amélioration (coverage, discipline Git, logs structurés), je pense que c'est un résultat solide.
Je continue d'apprendre à chaque étape et ce projet m'a permis de progresser de manière concrète et le prochain aussi ....

4. Documentation utilisateur et supervision

4.1 FAQ utilisateur

La FAQ complète est disponible dans le fichier **FAQ_UTILISATEUR.md**, qui regroupe 36 questions/réponses couvrant l'ensemble des besoins courants :

- Compte et authentification (inscription, connexion, mot de passe, déconnexion)
- Navigation et interface (fil d'actualité, articles, commentaires)
- Thèmes et abonnements (consultation, abonnement, désabonnement, filtrage)
- Profil utilisateur (consultation, modification, suppression du compte)
- Dépannage (erreurs fréquentes, support technique)
- Conformité et sécurité (données personnelles, RGPD, signalement de contenu)

Cette FAQ constitue la documentation utilisateur principale du MVP.

4.2 Supervision et tâches déléguées à l'IA

Copilot m'a servi d'accélérateur sur certaines tâches (auto-complétion, snippets, refactors simples, explications d'erreurs), mais **l'intégralité du code, des tests et des validations finales a été réalisée, relue et exécutée par moi-même**.

Concrètement :

- **Code et design** : génération de squelettes (contrôleurs, services, DTO, annotations JPA, composants Angular, CSS responsive), puis relecture, corrections et finalisation manuelle.
- **Tests** : aide sur la structure AAA (Arrange-Act-Assert) et quelques spécifications, puis vérification manuelle jusqu'au passage complet des tests (coverage : 64 % backend, 82.8 % frontend).
- **Débogage et documentation** : suggestions pour clarifier certains messages d'erreur et améliorer la structure de la documentation, systématiquement revues et ré-exécutées par moi.

Méthodologie de supervision

1. Revue de code

Chaque fichier généré ou modifié a été vérifié manuellement :

- Cohérence de la logique et absence de raccourcis
- Concordance entre les noms, les types et les responsabilités
- Respect des conventions (camelCase, JavaDoc, organisation des imports)

- Suppression du code inutile et des commentaires obsolètes

2. Tests

Exécution systématique des tests à chaque étape :

- Backend : `mvn test` → **43/43 tests réussis**
- Frontend : `npm test:coverage` → **82/82 tests réussis**
- Analyse de la couverture : **64 % backend**, **82.8 % frontend**

3. Linting et formatage

Contrôles automatiques :

- Prettier pour le frontend
- IntelliJ formatter pour le backend
- Aucun avertissement non résolu

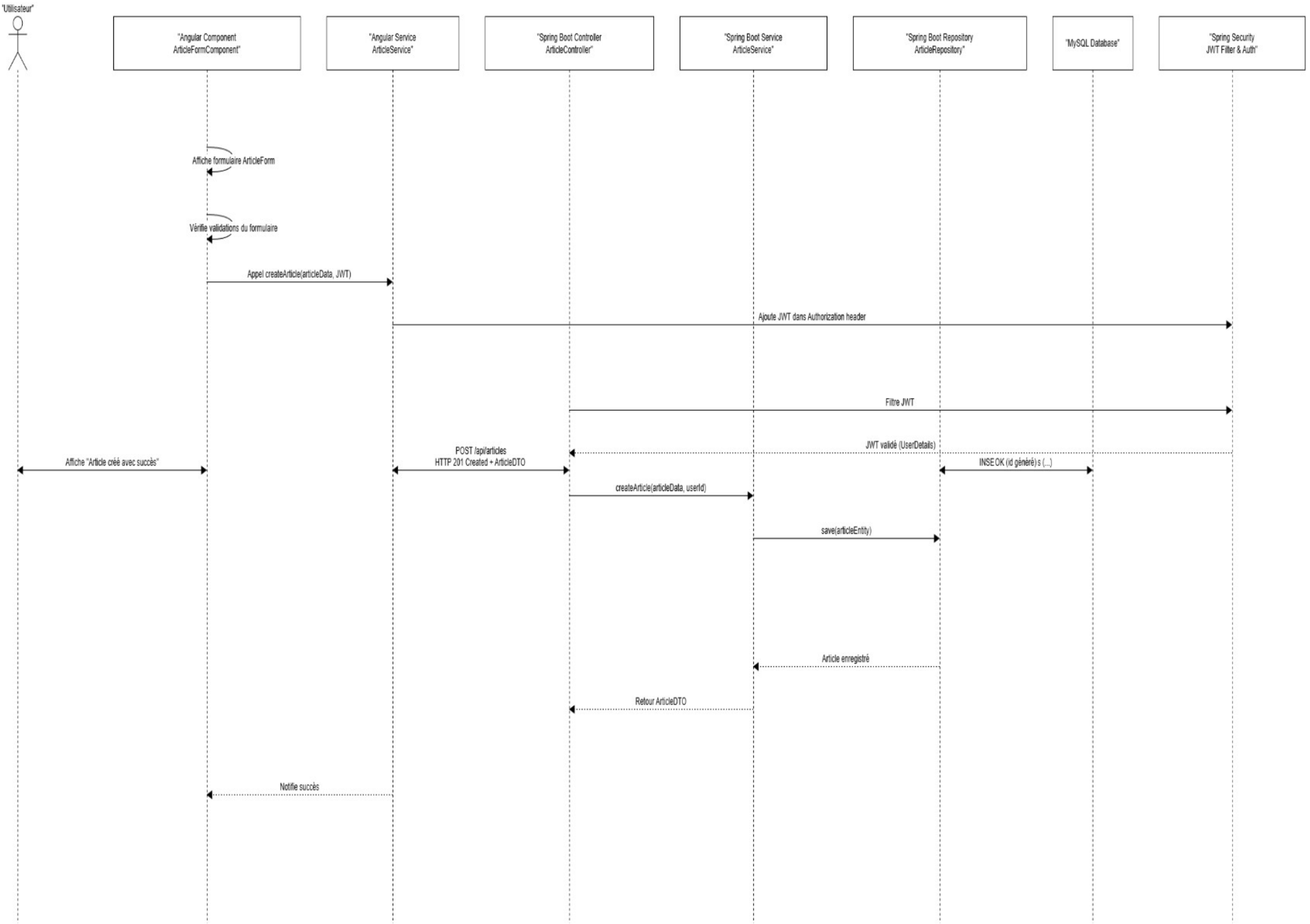
4. Tests fonctionnels manuels

Validation du parcours utilisateur complet :

- Inscription → Connexion → Consultation des articles → Abonnement aux thèmes
- Tests réalisés directement dans le navigateur
- Aucun bug fonctionnel identifié

5. Annexes

1. **Captures d’écran UI**
   Dossier : `docs/screenshots/`
   Contenu : pages principales (accueil, inscription, connexion), fil d’actualité, thèmes, profil, article + commentaires, vues mobiles.
2. **Analyse des besoins front-end**
   Fichier : `docs/Analyse_besoins_frontend.md`
   Contenu : routes, pages/composants, responsive design, services Angular, guards, lien avec les specs.
3. **Définition des données**
   Fichier : `docs/Definition_donnees.md`
   Contenu : schéma BDD (4 entités + M:N), formats JSON, règles de validation, sécurisation (hashage, contraintes).
4. **Rapports de couverture et tests**
   Fichier : `docs/Rapport de couverture et de tests/Rapport_couverture_tests.md`
   Contenu : 125 tests (43 back / 82 front), coverage JaCoCo & Jest, captures d’écran, analyse des zones non couvertes.
5. **Revue technique**
   Fichier : `docs/Rapport de revue technique/Rapport_revue_technique.md`
   Contenu : points forts/faibles, architecture, sécurité, qualité du code, axes d’amélioration.
6. **Diagramme de séquence**
   Contenu : capture du diagramme de séquence du parcours utilisateur.


"Utilisateur"
"Angular Component
ArticleFormComponent"
"Angular Service
ArticleService"
"Spring Boot Controller
ArticleController"
"Spring Boot Service
ArticleService"
"Spring Boot Repository
ArticleRepository"
"MySQL Database"
"Spring Security
JWT Filter & Auth"
Affiche formulaire ArticleForm
Vérifie validations du formulaire
Appel createArticle(articleData, JWT)
Ajoute JWT dans Authorization header
Filtre JWT
JWT validé (UserDetails)
POST /api/articles
HTTP 201 Created + ArticleDTO
INSERT OK (id généré) s (...)
Affiche "Article créé avec succès"
createArticle(articleData, userId)
save(articleEntity)
Article enregistré
Retour ArticleDTO
Notifie succès