Université de Pau et des Pays de l'Adour
UFR Sciences et Techniques

Projet Académique

Master Informatique, parcours TI

Développement d'une Application Web pour Favoriser l'Échange Hyperlocal d'Objets et de Compétences entre Voisins

Réalisé par :
- Missipsa Bouhadad
- Loïc Quessette

Encadrant :
M. Jawher Jerray

Année universitaire 2025–2026

Table des matières

Chapitre 1

Introduction Générale

1.1 Contexte et Motivation du Projet

Avec l'évolution des modes de vie urbains et la montée des préoccupations environnementales et sociales, de nouvelles formes de consommation et de collaboration ont émergé. Parmi elles, l'économie du partage occupe une place centrale en proposant des alternatives à la consommation individuelle excessive, en favorisant le réemploi des ressources et en renforçant les liens sociaux au sein des communautés locales.

Dans ce contexte, de nombreux objets du quotidien (outils de bricolage, appareils électroniques, équipements occasionnels) restent sous-utilisés, tandis que certaines compétences utiles (jardinage, bricolage, aide informatique, soutien scolaire, etc.) pourraient être partagées facilement entre voisins. Cependant, l'absence de plateformes adaptées à un échange véritablement local et basé sur la confiance constitue un frein à ce type de collaboration.

Ce projet s'inscrit donc dans une démarche visant à répondre à ces enjeux en proposant une application web dédiée à l'échange hyperlocal de biens et de compétences entre voisins. L'objectif est de créer un espace numérique favorisant la solidarité de proximité, la réduction de la consommation inutile et le renforcement des interactions sociales dans le monde réel.

Au-delà de l'aspect sociétal, ce projet représente également un exercice complet de génie logiciel. Il mobilise plusieurs notions abordées durant la formation, notamment l'ingénierie des exigences, la conception logicielle, l'architecture logicielle, ainsi que les tests et le deploiement. La réalisation de cette application constitue ainsi une opportunité de mettre en pratique ces connaissances dans un cas concret et réaliste.

1.2 Objectifs du Projet

1.2.1 Objectif Principal

L'objectif principal de ce projet est de concevoir et développer une application web permettant aux utilisateurs d'échanger, de manière sécurisée et locale, des objets et des compétences entre voisins. Cette application vise à faciliter la mise en relation des utilisateurs à proximité, tout en instaurant un climat de confiance grâce à des mécanismes de profil, de messagerie et de réputation.

1.2.2 Objectifs Secondaires

En complément de l'objectif principal, ce projet vise plusieurs objectifs secondaires qui définissent les orientations générales de l'application et les bénéfices attendus pour les utilisateurs et la communauté locale.

- favoriser le développement de communautés locales solidaires en facilitant les échanges directs entre voisins ;
- encourager le partage et la réutilisation des biens et des compétences afin de limiter le gaspillage et la surconsommation ;
- renforcer la confiance entre les utilisateurs grâce à un cadre structuré, transparent et sécurisé pour les échanges ;
- proposer une plateforme simple, intuitive et accessible permettant une adoption rapide par des utilisateurs aux profils variés ;
- offrir un espace numérique favorisant les interactions réelles et la coopération locale plutôt qu'une consommation purement transactionnelle ;

Chapitre 2

Analyse et Conception

2.1 Analyse des Besoins

2.1.1 Besoins Fonctionnels

Les besoins fonctionnels sont les besoins spécifiant un comportement d'entrée/sortie du système. (Contenu détaillé de la liste inchangé, mais sans la balise “ pour une meilleure

intégration dans les listes)

- Gestion de l'authentification des utilisateurs
  - Le système doit permettre à un nouvel utilisateur de s'inscrire avec son email, son nom, son prénom et sa localisation générale.
  - Le système doit authentifier un utilisateur enregistré pour lui donner accès à ses données privées.
  - Le système doit permettre à l'utilisateur de modifier son nom, sa localisation, d'ajouter une photo de profil et une courte biographie.
  - Le système doit permettre à l'utilisateur de vérifier son numéro de téléphone.

- Gestion des Annonces d'Objets
  - Le système doit permettre à un utilisateur de créer une annonce avec un titre, description, catégorie, images, disponibilité, et le besoin d'échange (objet/compétence).
  - Le système doit afficher toutes les annonces d'objets disponibles.
  - Le système doit permettre de rechercher des annonces par mots-clés et de filtrer par catégorie.
  - Le système doit afficher la page détaillée d'une annonce, incluant toutes les informations de l'objet et le profil du créateur.
- Gestion des Échanges de Compétences
  - Le système doit permettre à un utilisateur de créer une annonce de compétence avec un titre, description, disponibilités, et le besoin d'échange (objet/compétence).
  - Le système doit afficher toutes les annonces de compétences disponibles (avec une possible restriction géographique).
  - Le système doit permettre de rechercher des annonces de compétences par mots-clés.
- Flux de Transaction d'Échange

    - Le système doit permettre au bénéficiaire de répondre à une annonce en précisant l'offre en retour (objet/compétence) et une date d'échange.
    - Le système doit permettre au donateur de visualiser les demandes d'échange reçues.
    - Le système doit permettre au donateur d'accepter ou de refuser une demande d'échange.
    - Le système doit activer un canal de messagerie privée (chat) entre les deux parties uniquement après l'acceptation d'une demande.
- Système de Réputation et de Confiance
    - Le système doit permettre aux deux utilisateurs de laisser une note (1-5 étoiles) et un avis public après la finalisation d'un échange.
- Système de Notification
    - Le système doit générer des notifications pour les nouvelles demandes, les nouveaux messages et les changements de statut des demandes.

2.1.2 Besoins Non Fonctionnels

Performance, sécurité, qualité de service.

- Sécurité
    - Gestion sécurisée des mots de passe et des sessions
    - Assurer la confidentialité et l'intégrité des données utilisateur
- Performance
    - Le temps de réponse pour la recherche, le filtrage et l'affichage des annonces doit être optimisé.
    - L'application doit supporter un nombre croissant d'utilisateurs et d'annonces (Scalabilité).
- Utilisabilité
    - L'interface doit être intuitive, notamment pour la création d'annonces, l'envoi de demandes et la gestion du chat.
- Fiabilité et Robustesse
    - Les informations des annonces, des profils et des transactions doivent être stockées de manière cohérente et ne pas être perdues.

2.2 Méthode d'Analyse et de Conception

2.2.1 Choix de la Méthodologie

Pour la réalisation de ce projet, nous avons choisi d'adopter une approche Agile inspirée du cadre Scrum, adaptée à une équipe réduite de deux développeurs. Ce choix est justifié par la nature évolutive du projet, la diversité des fonctionnalités à implémenter (authentification, annonces, messagerie, système de notification) et la nécessité de s'adapter rapidement aux ajustements et imprévus techniques.

L'Agile est particulièrement pertinent pour un projet académique car elle permet :

- d'organiser les tâches en incréments courts et réalisables ;
- de développer en parallèle frontend et backend grâce à un découpage clair ;
- de réviser régulièrement les priorités en fonction de l'avancement ;
- de faciliter la collaboration et la communication au sein du binôme ;
- d'intégrer naturellement un workflow Git basé sur des branches de fonctionnalités.

2.2.2 Application de la Méthodologie Agile dans le Projet

Pour adapter Scrum à une équipe réduite, nous avons simplifié certains éléments tout en conservant les principes essentiels :

- **Découpage du travail en user stories** : chaque fonctionnalité (ex : création d'annonce, login, chat) a été décrite sous forme de besoins utilisateurs.
- **Itérations courtes** : le développement s'est déroulé en mini-sprints d'une durée d'environ une semaine.
- **Réunions de synchronisation** : de courtes discussions quotidiennes ou tous les deux jours permettaient de faire le point sur l'avancement et les difficultés rencontrées.
- **Priorisation continue** : les fonctionnalités critiques (authentification, annonces,chat) ont été réalisées en premier, puis les modules secondaires (notifications).
- **Revue à la fin de chaque sprint** : test local de ce qui avait été développé, corrections rapides, planification du sprint suivant.

2.2.3 Rôle du Contrôle de Version dans la Méthodologie

Le workflow Git s'intégrait directement dans cette approche Agile :

- chaque fonctionnalité correspondait à une branche distincte (feature branch) ;
- les pull requests servaient de point de validation entre nous ;
- les commits fréquents permettaient de suivre l'évolution technique du projet ;
- l'historique Git servait également d'outil de documentation du processus.

2.2.4 Livrables Produits par Phase

À chaque itération, différents éléments ont été produits :

- **Analyse** : identification des acteurs, définition des besoins.
- **Conception** : diagrammes UML, schémas d'architecture.
- **Implémentation** : composants React, routes Express, modèles MongoDB.
- **Tests** : vérification des endpoints, tests fonctionnels manuels.
- **Revue** : réajustement des priorités pour le sprint suivant.

Ainsi, l'utilisation d'une méthodologie Agile simplifiée nous a permis de maintenir un rythme de développement constant, de collaborer efficacement en binôme et d'aboutir à une application cohérente et évolutive.

2.2.5 Le Langage de Modélisation (UML)

Justification du Choix d'UML

Les raisons sont les suivantes :

- UML est avant tout un support de communication performant, qui facilite la représentation et la compréhension de solutions objet.
- L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en fait un langage universel.
- Il cadre l'analyse.
- Il permet également de générer automatiquement une partie de code, par exemple en langage Java, grâce aux outils de modélisation UML.

2.2.6 Le Modèle MVC

Définition

Le modèle MVC (Model, View, Controller) est un patron d'architecture qui sépare une application en trois couches distinctes afin d'améliorer la structuration du code, la maintenabilité et la réutilisabilité.

- **Model** : représente les données et la logique métier (dans notre cas : schémas MongoDB/Mongoose, règles métier liées aux annonces, utilisateurs, demandes d'échange, etc.).
- **View** : représente l'affichage présenté à l'utilisateur. Comme notre application utilise React, la couche View est déportée dans le frontend.
- **Controller** : contient la logique qui traite les requêtes entrantes, applique les règles métier et renvoie des réponses au client (routes express, contrôleurs).

Application du MVC dans un projet MERN

Dans une application MERN, le modèle MVC n'est pas appliqué de manière traditionnelle, car le frontend React fonctionne selon une architecture à base de composants. Toutefois, la partie backend (Node.js / Express) suit une structure inspirée de MVC :

- **Model** : les modèles Mongoose définissent les collections MongoDB.
- **Controller** : chaque fonctionnalité possède un contrôleur qui regroupe la logique métier (user.controller, ad.controller, chat.controller...).
- **Routes** : elles jouent le rôle de « point d'accès » et délèguent le traitement aux contrôleurs.

Pourquoi MVC est pertinent dans notre projet

L'utilisation d'une structure proche du MVC dans le backend permet :

- de séparer clairement les responsabilités ;
- d'organiser proprement les fichiers et dossiers ;
- de faciliter la maintenance et les tests des différentes parties du backend ;
- d'avoir une architecture cohérente, même avec un frontend React indépendant.

Ainsi, même si l'application MERN ne suit pas un MVC complet comme une application monolitique traditionnelle, elle adopte les principes du modèle pour la partie backend tout en s'appuyant sur une architecture de composants pour le frontend.

2.3 Modélisation UML

2.3.1 Diagramme de Cas d'Utilisation

Présentation des acteurs et des interactions principales avec le système.

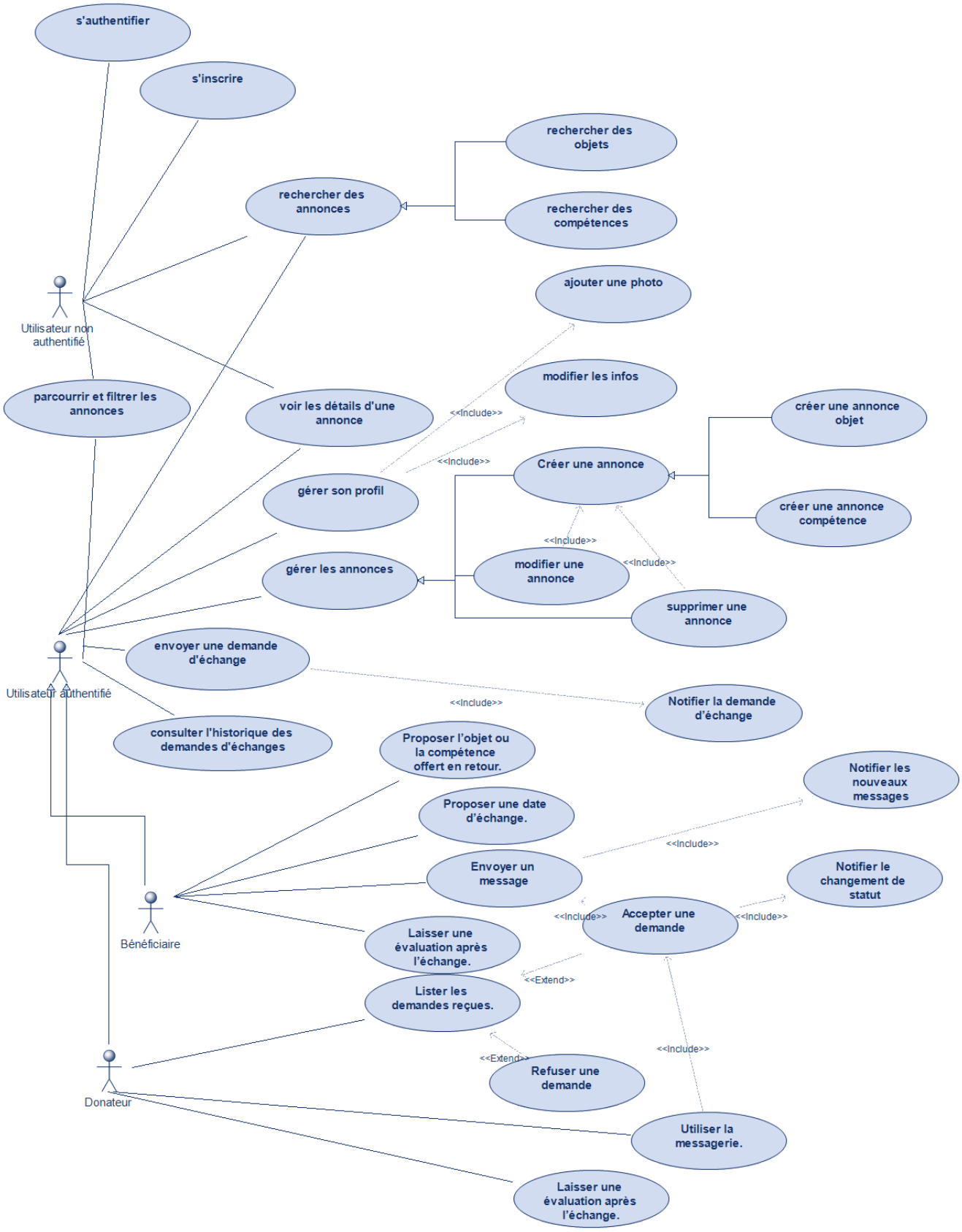

FIGURE 2.1 – Diagramme de Cas d'Utilisation

2.3.2 Diagramme de Classes

Le diagramme de classes exprime la structure statique du système en termes de classes et de relations entre ces classes. L'intérêt du diagramme de classes est de modéliser les entités du système d'information. Ces informations sont regroupées ensuite dans des classes.

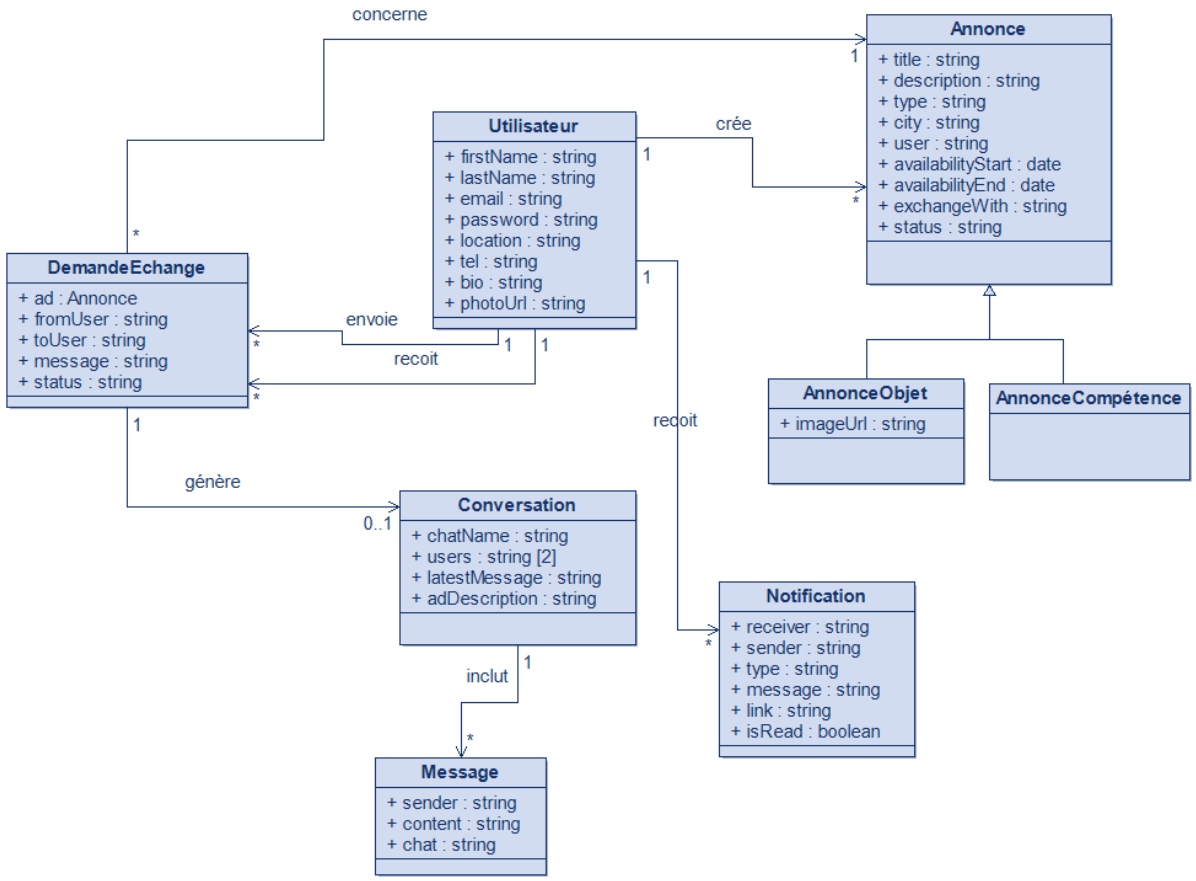

FIGURE 2.2 – Diagramme de classe

2.3.3 Diagrammes de Séquence

Les diagrammes de séquence sont des diagrammes UML qui permettent de représenter les interactions entre les acteurs et les différents composants du système au cours du temps. Ils illustrent, de manière chronologique, l'enchaînement des messages échangés pour la réalisation d'un scénario précis, tel que l'authentification d'un utilisateur, la création d'une annonce ou une demande d'échange. Ces diagrammes facilitent la compréhension du fonctionnement dynamique du système et permettent de valider la logique des traitements entre les différentes parties de l'application. Voici quelques diagrammes de séquence illustrant les scénarios principaux de l'application :

Diagramme pour l'authentification

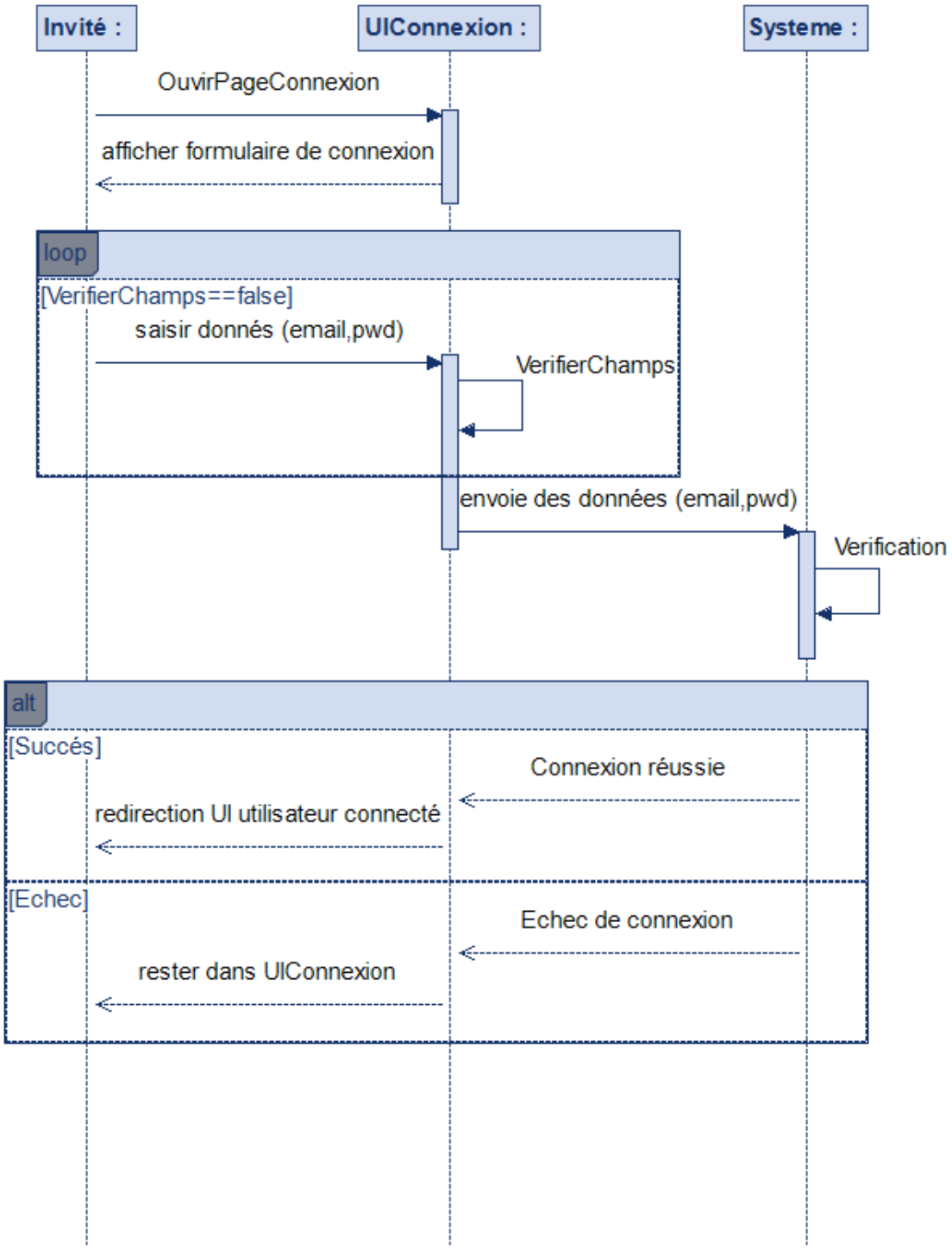

FIGURE 2.3 – Diagramme de séquence UML pour le processus d'authentification

Diagramme pour la création d'annonce

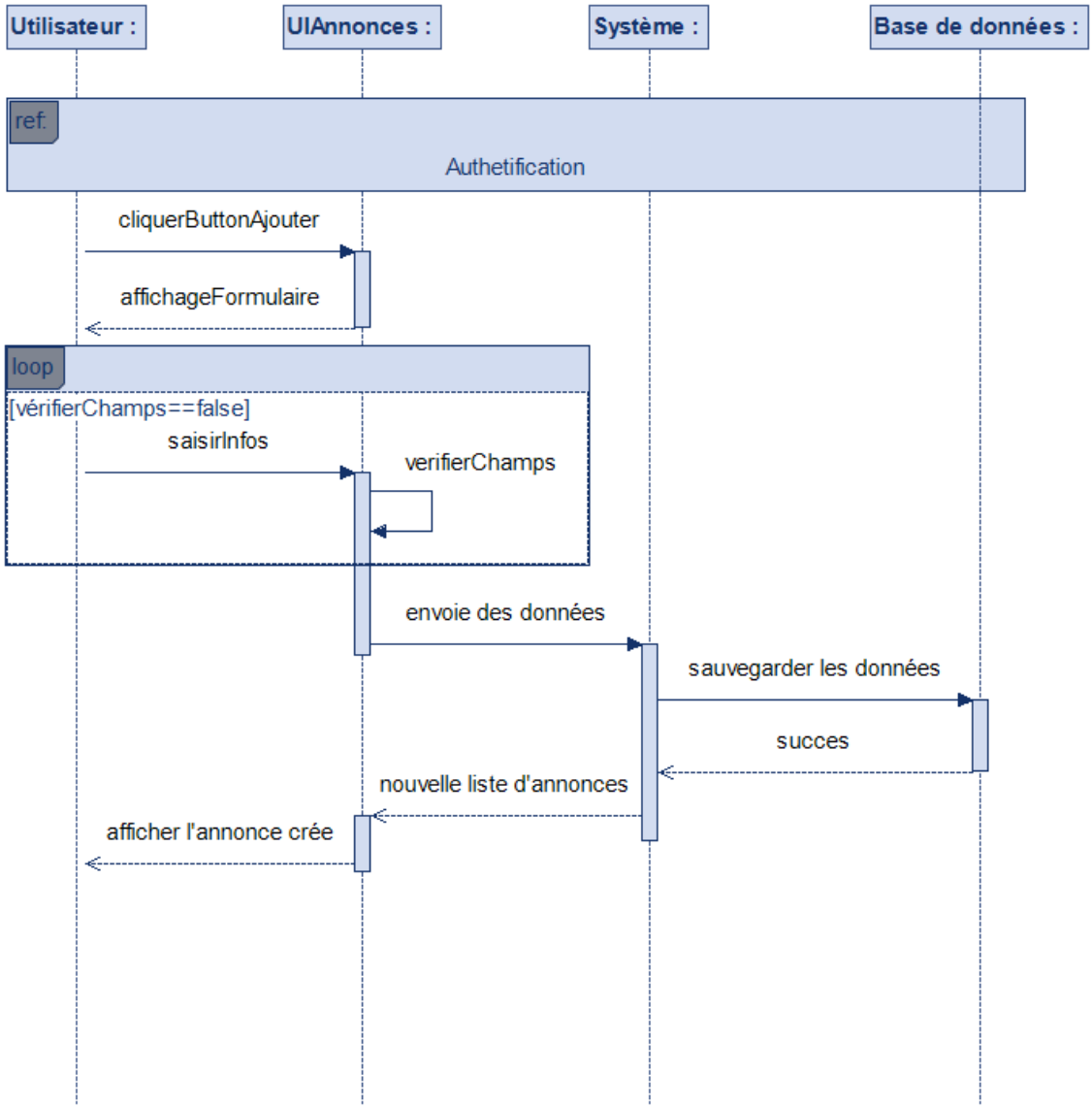

FIGURE 2.4 – Diagramme de séquence UML pour la création d'annonce.

Diagramme pour faire une demande d'échange

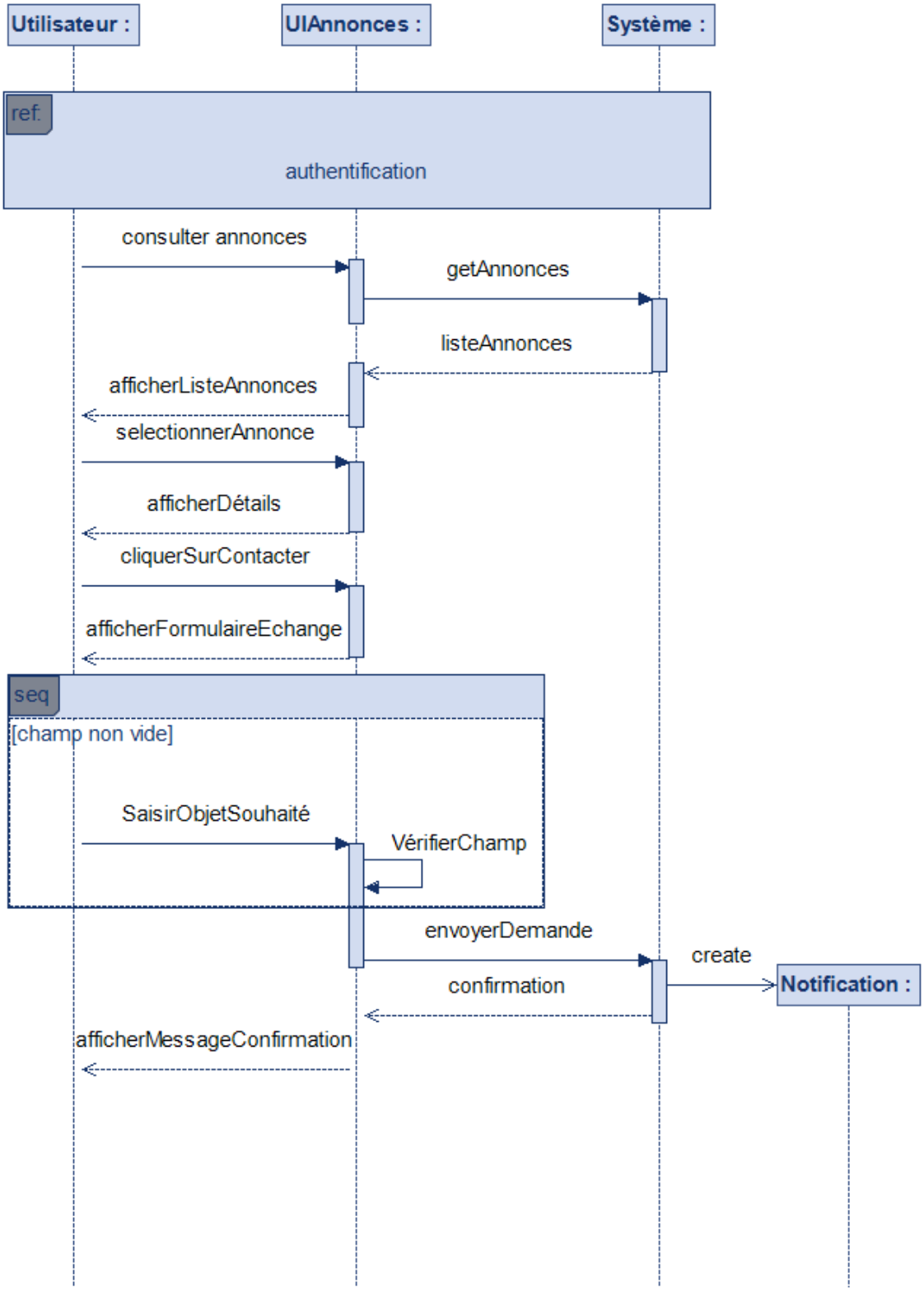

FIGURE 2.5 – Diagramme de séquence UML pour le flux de demande d'échange.

Chapitre 3

Architecture Technique

3.1 Présentation de la Stack MERN

L'architecture technique de l'application repose sur la stack MERN, un ensemble de technologies JavaScript modernes largement utilisées pour le développement d'applications web robustes et évolutives. Ce choix garantit une cohérence technologique entre le frontend et le backend, tout en facilitant le développement et la maintenance.

3.1.1 MongoDB

MongoDB est une base de données NoSQL orientée documents. Elle permet de stocker les données sous forme de documents JSON, ce qui offre une grande flexibilité dans la gestion des structures de données. Cette caractéristique est particulièrement adaptée à une application évolutive où les entités (utilisateurs, annonces, échanges) peuvent être amenées à évoluer au fil du temps. MongoDB facilite également la scalabilité horizontale et s'intègre naturellement avec Node.js via Mongoose.

3.1.2 Express.js

Express.js est un framework backend léger et performant basé sur Node.js. Il permet de structurer efficacement l'API REST de l'application en définissant des routes claires et des contrôleurs associés. Express facilite également l'intégration de middlewares pour la gestion de l'authentification, de la validation des données, de la sécurité et de la gestion des erreurs.

3.1.3 React.js

React.js est une bibliothèque JavaScript dédiée à la création d'interfaces utilisateur dynamiques et réactives. Son architecture basée sur des composants favorise la réutilisabilité du code et la séparation des responsabilités. React permet de gérer efficacement l'état de l'application et d'offrir une expérience utilisateur fluide, notamment lors de la navigation, de la création d'annonces et de l'utilisation de la messagerie.

3.1.4 Node.js

Node.js constitue l'environnement d'exécution JavaScript côté serveur. Il permet de gérer un grand nombre de requêtes de manière asynchrone et performante. Son intégration

avec Express et MongoDB en fait un choix naturel pour le développement du backend de l'application.

3.2 Architecture Générale du Système

3.2.1 Schéma Architectural (Vue 3-Tiers)

L'application suit une architecture de type trois tiers :

— **Client** : le frontend React, exécuté dans le navigateur, gère l'interface utilisateur et les interactions.

— **Serveur** : l'API backend Node.js / Express assure la logique métier, la gestion des utilisateurs et des échanges.

— **Base de données** : MongoDB stocke les données persistantes de l'application.

Les échanges entre le client et le serveur se font via des requêtes HTTP REST sécurisées.

3.2.2 Technologies Complémentaires

Plusieurs technologies complémentaires ont été intégrées :

— **JWT (JSON Web Tokens)** pour l'authentification sécurisée ;

— **Bcrypt** pour le hachage des mots de passe ;

— **API de géolocalisation** pour limiter les échanges à une zone géographique pertinente.

Chapitre 4

Développement et Implémentation

4.1 Organisation de l'Architecture Technique du Backend

4.1.1 Structure des Répertoires et Flux des Données

L'application serveur est organisée autour de trois composants principaux : les Modèles, les Contrôleurs et les Routes. Ce découpage reflète la manière dont les requêtes sont traitées, du point d'entrée API à la base de données.

- /**models** : Contient les schémas **Mongoose** définissant la structure des collections **MongoDB** (User, Ad, Exchange, Message). Il incarne la couche d'accès aux données.
- /**routes** : Définit les URL (endpoints) de l'API REST et délègue l'exécution des requêtes HTTP aux contrôleurs correspondants.
- /**controllers** : Regroupe la logique métier de l'application. Il manipule les données via les modèles Mongoose et prépare la réponse HTTP pour le client.
- /**middlewares** : Gère les fonctions de traitement intermédiaires, notamment pour l'authentification (vérification du token JWT) et la validation des données.

4.1.2 Mise en œuvre des Modèles Mongoose

Chaque entité modélisée dans le Diagramme de Classes (Chapitre 2) a été implémentée comme un schéma Mongoose. Par exemple, le modèle d'Annonce (`Ad`) inclut des champs pour la description, les catégories...

Cela permet des requêtes spécifiques de proximité via **MongoDB GeoSpatial Queries**.

4.2 Implémentation des Fonctionnalités de Sécurité et d'Authentification

La sécurité des données utilisateurs et la protection des routes ont été des priorités majeures.

4.2.1 Gestion Sécurisée des Mots de Passe (Bcrypt)

Lors de l'inscription d'un utilisateur, le mot de passe n'est jamais stocké en clair. Le middleware **bcrypt** est utilisé pour **hacher** le mot de passe avant son enregistrement en base de données, empêchant toute compromission en cas de fuite de données. La vérification lors de la connexion utilise le même mécanisme de hachage.

4.2.2 Authentification sans État (JWT)

Pour gérer l'accès aux ressources privées, nous avons implémenté un système d'authentification basé sur les **JSON Web Tokens (JWT)**.

— **Génération :** Lors de la connexion réussie, un JWT est généré côté serveur, contenant l'identifiant utilisateur (payload).

— **Vérification :** Ce token est transmis par le client dans l'en-tête de chaque requête subséquente. Un middleware dédié intercepte le token, vérifie sa validité et son expiration, et autorise ou rejette l'accès (`401 Unauthorized` si invalide).

4.3 Développement des Modules Fonctionnels

4.3.1 Gestion des Annonces et Médias

La création d'annonces est un flux utilisateur essentiel.

— **Validation :** Le contrôleur s'assure que les données soumises respectent les contraintes (présence d'un titre, catégorie valide, etc.) avant de les enregistrer.

— **Hébergement des Images :** Pour décharger le serveur principal et garantir la performance, les images d'annonces sont téléversées et hébergées sur le service tiers **Cloudinary**. Seules les URL de ces images sont stockées dans le modèle Mongoose.

4.3.2 Mise en Place de la Messagerie Interne

Afin de permettre aux voisins d'organiser concrètement l'échange après l'acceptation d'une demande, un système de messagerie interne a été mis en place. Ce mécanisme permet une communication privée et sécurisée entre les utilisateurs concernés.

— **Communication Client–Serveur :** La messagerie repose sur une architecture basée sur des services *REST*. Les messages sont transmis entre le client et le serveur via des requêtes HTTP, en utilisant la librairie **Axios** côté frontend et **Express.js** côté backend.

— **Gestion et Persistance des Messages :** Chaque message échangé est enregistré dans la base de données **MongoDB**, permettant ainsi la conservation de l'historique des conversations et leur consultation ultérieure.

— **Contrôle d'Accès :** La création et l'accès à une session de discussion sont strictement conditionnés à l'état de l'objet (*accepté*), garantissant que seules les parties directement impliquées dans l'échange peuvent communiquer.

4.4 Développement du Frontend (React.js)

Le client est une application monopage (SPA) construite avec **React.js**.

4.4.1 Architecture par Composants

L'application est découpée en composants modulaires (ex : `CardAnnonce`, `FormLogin`, `ChatBox`) pour faciliter le développement parallèle et la maintenance. L'utilisation du DOM Virtuel de React assure une interface utilisateur rapide et réactive, essentielle pour une expérience utilisateur fluide lors de la navigation ou du filtrage des annonces.

4.4.2 Gestion de l'État de l'Application

L'état global (informations utilisateur, annonces filtrées, statut d'authentification) est géré via les **React Hooks** (`useState`, `useEffect`) et le contexte (`useContext`) pour permettre le partage d'informations à travers l'arbre de composants sans recourir à des props drilling excessifs. Les requêtes asynchrones vers l'API **Express** sont gérées via des librairies comme **Axios**.

Chapitre 5

Tests et Validation

5.1 Stratégie de Test

Afin de garantir la robustesse et la fiabilité de la plateforme d'échange, nous avons adopté une stratégie de test orientée vers l'intégration continue (CI). L'objectif principal est de détecter les régressions le plus tôt possible dans le cycle de développement.

Nous avons créé des tests unitaires ainsi que des tests d'intégration automatisés, exécutés à chaque modification du code source via une pipeline GitHub Actions. Cette approche permet de valider systématiquement le backend avant tout déploiement, en isolant l'environnement de test de la base de données de production.

5.2 Tests Unitaires et d'Intégration Backend

5.2.1 Couverture et Outils Utilisés

Pour la partie serveur (API Node.js), nous avons utilisé le framework de test JavaScript **Jest**. Ce choix s'explique par sa rapidité, sa simplicité de configuration "zéro-config" et ses fonctionnalités intégrées d'assertions et de mocking.

Les outils clés de notre suite de tests comprennent :

— **Jest** : Le lanceur de tests et moteur d'assertions.

— **Supertest** : Une bibliothèque permettant de simuler des requêtes HTTP (GET, POST, PUT, DELETE) vers notre application Express sans avoir à lancer manuellement le serveur.

— **MongoDB Memory Server** : Un outil essentiel qui lance une instance de base de données MongoDB directement dans la mémoire vive (RAM) lors de l'exécution des tests. Cela garantit des tests rapides, isolés et n'impactant jamais les données réelles.

5.3 Tests Fonctionnels et Scénarios

Nous avons concentré nos efforts sur la vérification des règles métier critiques et la conformité aux besoins fonctionnels.

5.3.1 Scénarios Utilisateurs

Les tests automatisés couvrent les scénarios complets du cycle de vie des données (CRUD) et des permissions. Voici les principaux scénarios validés :

1. **Gestion des Annonces (Cycle complet) :**
   — Un utilisateur authentifié peut créer une annonce (Objet ou Compétence).
   — L'API rejette la création si des champs obligatoires (Exemple : titre, dates...) sont manquants.
   — La modification d'une annonce met à jour les informations en base de données.
   — La suppression d'une annonce la retire définitivement de la liste.
2. **Isolement des Données :**
   — Vérification qu'un utilisateur A ne peut pas modifier ou supprimer l'annonce d'un utilisateur B (Test de retour HTTP 403 Forbidden).
3. **Recherche et Filtrage :**
   — Validation que les endpoints de récupération retournent bien les listes filtrées (par utilisateur ou liste globale).

5.4 Tests Non Fonctionnels (Performance et Sécurité)

5.4.1 Tests de Performance

Bien que nous n'ayons pas réalisé de tests de charge massifs (type JMeter), l'architecture a été pensée pour la performance :

— L'utilisation de *mongodb-memory-server* dans la CI prouve la capacité du backend à traiter des requêtes en quelques millisecondes.
— Les images sont traitées et hébergées via **Cloudinary**, déchargeant le serveur principal du stockage et de la distribution des médias lourds.

5.4.2 Audit de Sécurité

La sécurité a été intégrée dès la conception :

— **Protection des Routes :** Tous les tests d'écriture (POST/PUT/DELETE) vérifient systématiquement la présence et la validité du Token JWT. Une requête sans token valide retourne une erreur 401.
— **Hashage des Mots de passe :** Validation que les mots de passe ne sont jamais stockés en clair (utilisation de *bcrypt*).
— **Injections NoSQL :** L'utilisation de l'ODM Mongoose avec des schémas stricts permet de prévenir nativement la majorité des injections.

Chapitre 6

Conclusion et Perspectives

Ce projet académique avait pour objectif de concevoir et de développer une application web dédiée à l'échange hyperlocal d'objets et de compétences entre voisins. À travers la réalisation de cette application, nous avons cherché à proposer une solution numérique favorisant la solidarité de proximité, la réutilisation des ressources et le renforcement du lien social au sein des communautés locales.

Le travail réalisé a permis de couvrir l'ensemble des étapes classiques d'un projet de génie logiciel. Une analyse approfondie des besoins fonctionnels et non fonctionnels a été menée afin de définir les fonctionnalités essentielles de l'application. La phase de conception s'est appuyée sur des diagrammes UML permettant de modéliser la structure et le comportement du système. Le développement s'est ensuite appuyé sur la stack MERN, offrant une architecture cohérente et moderne, avec un backend structuré et un frontend réactif et intuitif.

L'adoption d'une méthodologie Agile inspirée de Scrum a facilité l'organisation du travail en binôme, la priorisation des fonctionnalités et l'adaptation continue du projet face aux contraintes techniques et temporelles. Cette approche a permis d'aboutir à une application fonctionnelle intégrant l'authentification des utilisateurs, la gestion des annonces, les demandes d'échange, la messagerie en temps réel ainsi qu'un système de notifications pour les nouvelles demandes et les nouveaux messages.

Au-delà de l'aspect technique, ce projet a constitué une expérience formatrice sur le plan méthodologique et organisationnel. Il a permis de consolider des compétences en développement web , en conception logicielle, en modélisation UML, ainsi qu'en utilisation d'outils de collaboration et de gestion de version tels que Git.

Malgré les résultats obtenus, certaines limites subsistent. Le projet étant réalisé dans un cadre académique et avec un temps limité, certaines fonctionnalités pourraient être approfondies, notamment en matière de tests automatisés, de sécurité avancée et d'optimisation des performances. De plus, l'interface utilisateur pourrait être améliorée à travers des tests d'utilisabilité auprès d'un panel d'utilisateurs.

En termes de perspectives, plusieurs évolutions peuvent être envisagées. Parmi elles, le développement d'une application mobile, la mise en place d'un système de modération des annonces et des avis, ainsi que l'ajout de mécanismes de recommandation basés sur les préférences et la localisation des utilisateurs. Enfin, un déploiement en conditions réelles permettrait d'évaluer l'impact de l'application sur la dynamique des échanges locaux et d'envisager son évolution vers une plateforme à plus grande échelle.

En conclusion, ce projet représente une synthèse concrète des connaissances et compétences acquises durant la formation et constitue une base solide pour de futurs développements dans le domaine du developpement web.