

# Software Requirements Specification (SRS)

## Application échange local de biens et de compétences

Version : 1.0  
Auteur : Simon  
Date : —

---

## 1. Introduction

### 1.1 Objectif du document

Décrire les exigences fonctionnelles et non fonctionnelles pour le serveur Java monolithique, implémenté avec Jetty et Hibernate ORM, compatible SQLite et PostgreSQL, en appliquant les principes SOLID.

### 1.2 Portée du système

Serveur backend monolithique : API REST pour le frontend, logique métier, persistance via Hibernate, notifications, messagerie, gestion des échanges, système d'avis et d'authentification.

### 1.3 Définitions, acronymes et abréviations

- API : Application Programming Interface
- SGBD : Système de gestion de base de données
- SOLID : principes de conception objet
- JWT : JSON Web Token
- ORM : Object Relational Mapping
- ACID : Atomicité, Cohérence, Isolation et Durabilité

### 1.4 Références

Projet de synthèse : Application échange local de biens et de compétences.

## 2. Description générale

### 2.1 Perspective du produit

Monolithe Java, exécution sur JVM 17+, serveur embarqué Jetty, persistance via Hibernate ORM. L'application expose une API REST et persiste les données dans une base relationnelle (PostgreSQL en production, SQLite possible pour démo et/ou tests).

### 2.2 Fonctions du produit

- Authentification (JWT), gestion des profils.
- CRUD annonces (objets, compétences).
- Gestion des demandes d'échange et messagerie.
- Avis et notation.
- Notifications (in-app \_\_+ email\_\_).

## **2.3 Caractéristiques des utilisateurs**

- Visiteur (lecture limitée).
- Utilisateur authentifié (création/modification d'annonces, échanges).
- Administrateur (gestion, suppression abusive).

## **2.4 Contraintes générales**

- Application monolithique.
- Persistance via Hibernate.
- Serveur embarqué Jetty.
- Compatibilité : SQLite (local et/ou dev) et PostgreSQL (pour déploiement).
- Respect minimal de la confidentialité (pas d'adresse/localisation exacte exposée).

## **2.5 Hypothèses et dépendances**

- Frontend respecte conventions des API REST.
- Serveur SMTP disponible pour envois dé-mails (non-implémenté).
- Environnements Java 17+ disponibles.

# **3. Exigences fonctionnelles**

## **3.1 Authentification**

- RF-AUTH-001 : Inscription (email, nom, localisation approximative).
- RF-AUTH-002 : Connexion et émission d'un JWT.
- RF-AUTH-003 : Stockage des mots de passe hachés (BCrypt).

## **3.2 Gestion des utilisateurs**

- RF-USR-001 : Édition du profil (bio, photo).
- RF-USR-002 : Calcul et retour de la note moyenne.

## **3.3 Annonces (objets)**

- RF-OBJ-001 : Création d'annonce (titre, description, catégorie, images, disponibilité).
- RF-OBJ-002 : Recherche/filtrage par mots-clés/catégorie.
- RF-OBJ-004 : Consultation d'une annonce.

## **3.4 Annonces (compétences)**

- RF-COMP-001 : Création, recherche, consultation d'annonces de compétences.

## **3.5 Échanges**

- RF-ECH-001 : Envoi de demande d'échange.
- RF-ECH-002 : Acceptation/refus par le propriétaire.
- RF-ECH-004 : Historique des échanges.

### 3.6 Messagerie

- RF-MSG-001 : Création de conversation à l'acceptation.
- RF-MSG-002 : Envoi et persistance de messages.

### 3.7 Avis

- RF-AVIS-001 : Notation 1–5 étoiles.
- RF-AVIS-002 : Commentaire public.

### 3.8 Notifications

- RF-NOTIF-001 : Notification sur nouvelle demande.
- RF-NOTIF-002 : Notification de message.
- RF-NOTIF-003 : Envoi d'e-mail (si configuré).

## 4. Exigences non fonctionnelles

### 4.1 Performance

Temps de réponse visé : < 300 ms (opérations standards en conditions normales).

### 4.2 Sécurité

- JWT pour authentification.
- Password4j pour mots de passe.
- Validation côté serveur.
- HTTPS obligatoire en production.

### 4.3 Fiabilité

Transactions ACID grâce à JPA/Hibernate pour les opérations critiques.

### 4.4 Maintenabilité (SOLID)

L'architecture applicative doit suivre SOLID : SRP, OCP, LSP, ISP, DIP.

### 4.5 Scalabilité

Couches claires, contracts API, repository faiblement couplés pour un futur découpage.

### 4.6 Compatibilité

API JSON, UTF-8. ORM Hibernate configuré pour PostgreSQL et SQLite (dialects configurables).

## 5. Interfaces externes

### 5.1 API REST

Liste complète des routes exposées :

- GET /test
- GET /users
- GET /users/id
- POST /users
- PUT /users/id
- DELETE /users/id
- GET /adverts
- GET /adverts/id
- POST /adverts
- PUT /adverts/id
- DELETE /adverts/id
- GET /adverts/search
- GET /applications
- GET /applications/id
- POST /applications
- POST /applications/id/accept
- POST /applications/id/reject
- DELETE /applications/id
- GET /messages
- GET /messages/id
- POST /messages
- GET /messages/conversation/userId
- GET /notifications
- GET /notifications/id
- POST /notifications/id/read
- POST /notifications

### 5.2 Base de données

Tables principales : utilisateur, annonce, echange, message, avis, notification.

### 5.3 Services externes

- SMTP (envoi d'e-mails) (Non implémenté).
- (Optionnel) service d'hébergement d'objets multimédias (Non implémenté).

## 6. Contraintes techniques

### 6.1 Environnement d'exécution

- Java 17+
- Javalin
- Maven

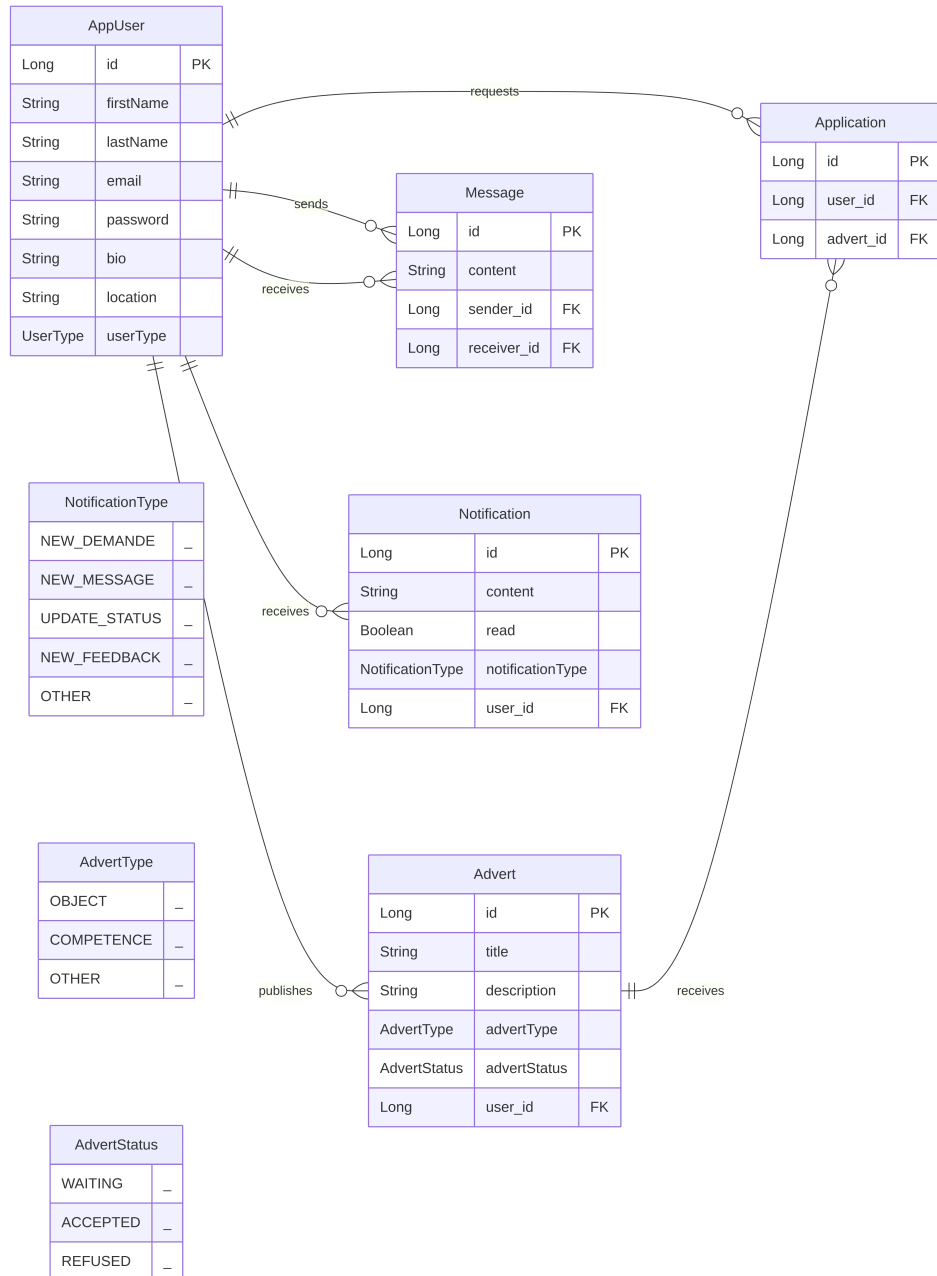


FIGURE 1 – Schéma entité relation de la base de données

## 6.2 Dépendances logicielles

- Javalin
- Hibernate ORM
- PostgreSQL ou SQLite
- JUnit
- Password4j

## 6.3 Configuration Hibernate

Par défaut : PostgreSQL. Option à passer : `-database sqlite` pour les tests locaux.

## 7. Critères d'acceptation

Fonctionnel :

- flux d'échange complet
- authentification
- autorisations vérifiées.

Non fonctionnel :

- tests CI avec SQLite et/ou préprod PostgreSQL
- temps de réponse conforme
- mots de passe hachés.

## 8. Annexes

### 8.1 Diagrammes d'architecture

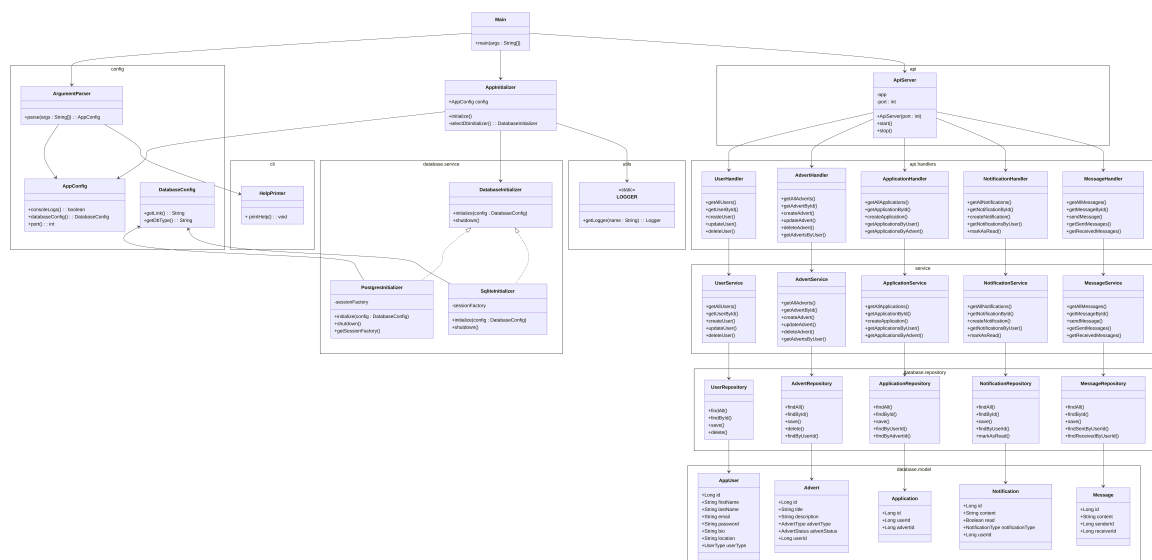


FIGURE 2 – Diagramme de classe