Cahier des Charges : Application de Traçage d'Objets Volés

AZANGUE LEONEL : 22P206 3GI ENSPY

8 avril 2025

1 Introduction

1.1 Contexte

Ce projet vise à développer une application Java complète pour faciliter le signalement des objets volés, la vérification de leur statut, et la mise en relation entre les propriétaires légitimes et les potentiels acheteurs. L'application cible principalement les particuliers et les professionnels du marché de l'occasion, leur offrant un outil fiable et simple pour lutter contre le recel et le commerce illicite.

2 Spécifications Fonctionnelles

2.1 Gestion des Utilisateurs

- **Inscription :**
  - Champs obligatoires : Nom d'utilisateur (unique), adresse e-mail (valide et unique), mot de passe (sécurisé), numéro de téléphone (optionnel).
  - Validation :
    - Format de l'e-mail : Vérification via une expression régulière.
    - Complexité du mot de passe : Minimum 8 caractères, incluant au moins une lettre majuscule, une lettre minuscule, un chiffre et un caractère spécial.
    - Unicité du nom d'utilisateur et de l'e-mail : Vérification lors de la soumission du formulaire.
  - Notification : Envoi d'un e-mail de confirmation après l'inscription.
- **Authentification :**
  - Méthode : Authentification basée sur le nom d'utilisateur et le mot de passe.
  - Sécurité : Hachage du mot de passe avec un sel unique pour chaque utilisateur, stockage du hash en base de données. ((Utile pour la version web))
  - Oubli du mot de passe : Procédure de réinitialisation du mot de passe via l'envoi d'un lien de réinitialisation par e-mail.

2.2 Signalement de Vol

- **Formulaire :**
  - Champs obligatoires : Type d'objet, marque, modèle, numéro de série (si applicable), numéro IMEI (si applicable), description détaillée du vol, date du vol.
  - Champs optionnels : Photos de l'objet, preuve d'achat.
- **Géolocalisation :**

- Automatique : Utilisation des API de géolocalisation pour obtenir la position au moment du signalement (avec consentement de l'utilisateur).
- Manuelle : Possibilité pour l'utilisateur d'indiquer manuellement le lieu du vol.

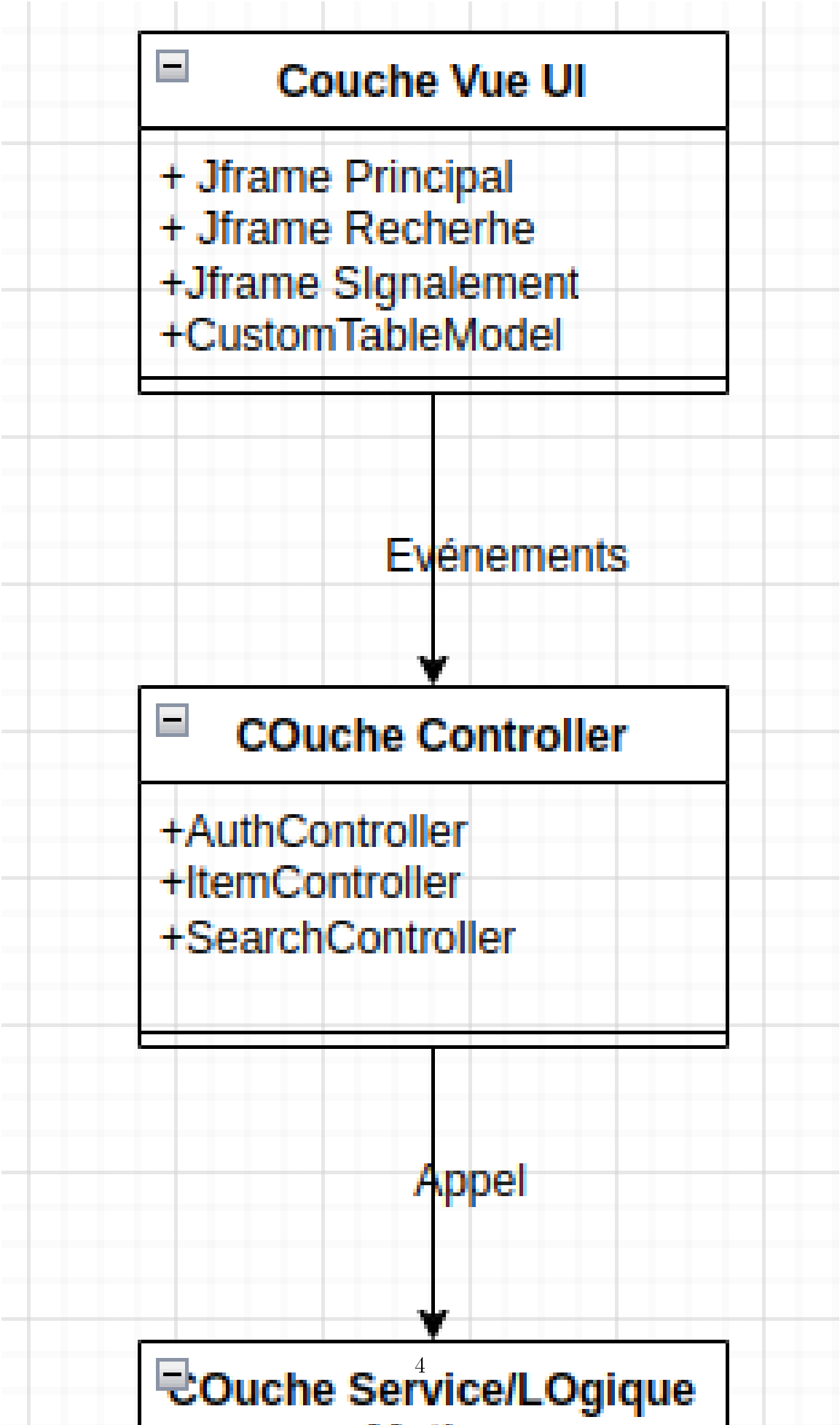
- **Confirmation :**
  - Envoi d'un e-mail de confirmation au propriétaire après le signalement.
  - Affichage d'un message de confirmation sur l'interface utilisateur.

2.3 Recherche d'Objets

- **Critères de Recherche :**
  - Par IMEI : Recherche exacte par numéro IMEI.
  - Par numéro de série : Recherche exacte par numéro de série.
  - Par description : Recherche par mots-clés dans la description de l'objet.
- **Affichage des Résultats :**
  - Liste des objets correspondant aux critères de recherche, avec :
    - Type d'objet, marque, modèle.
    - Statut (volé/non volé).
    - Date de signalement du vol.
    - Possibilité de contacter le propriétaire (si l'objet est signalé comme volé).
  - Filtrage des résultats : Par type d'objet, par date de signalement.

3 Architecture Technique

3.1 Diagramme d'Architecture Globale

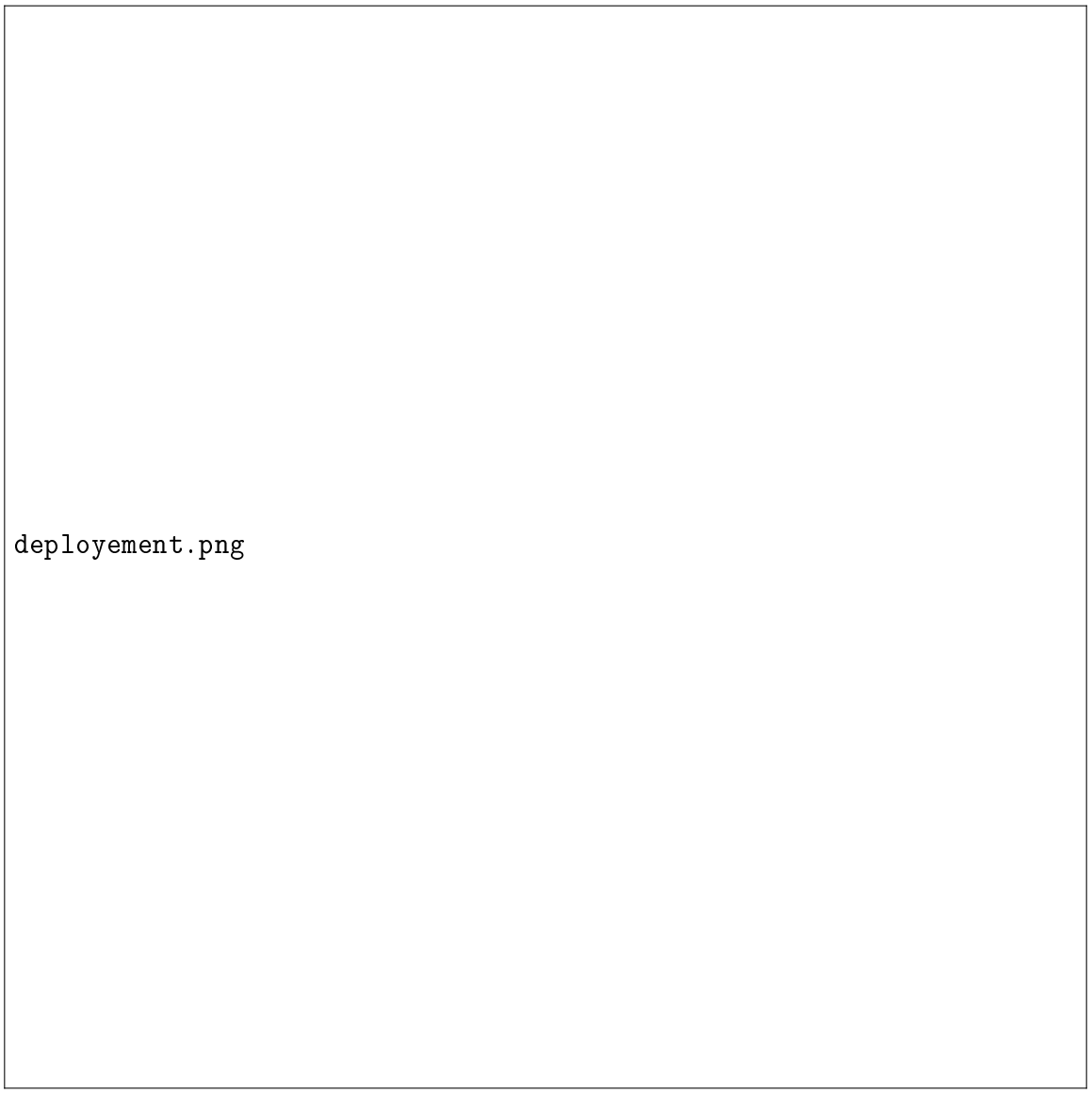


Description : Ce diagramme illustre l'architecture globale de l'application, mettant en évidence les différentes couches (présentation, logique métier, persistance) et leurs interactions. Il montre également les principaux composants de chaque couche et les technologies utilisées.

3.2 Stack Technologique

Couche	Technologies	Justification
Présentation	JavaFX	Framework UI moderne et riche, supportant les interfaces graphique
Logique Métier	Java 23	Langage robuste, mature et largement utilisé pour les applications c
Persistance	SQLite + JDBC	Base de données légère et facile à intégrer, idéale pour les applicatio
Sécurité	SHA-256 + Salage	Algorithme de hachage robuste pour le stockage sécurisé des mots d

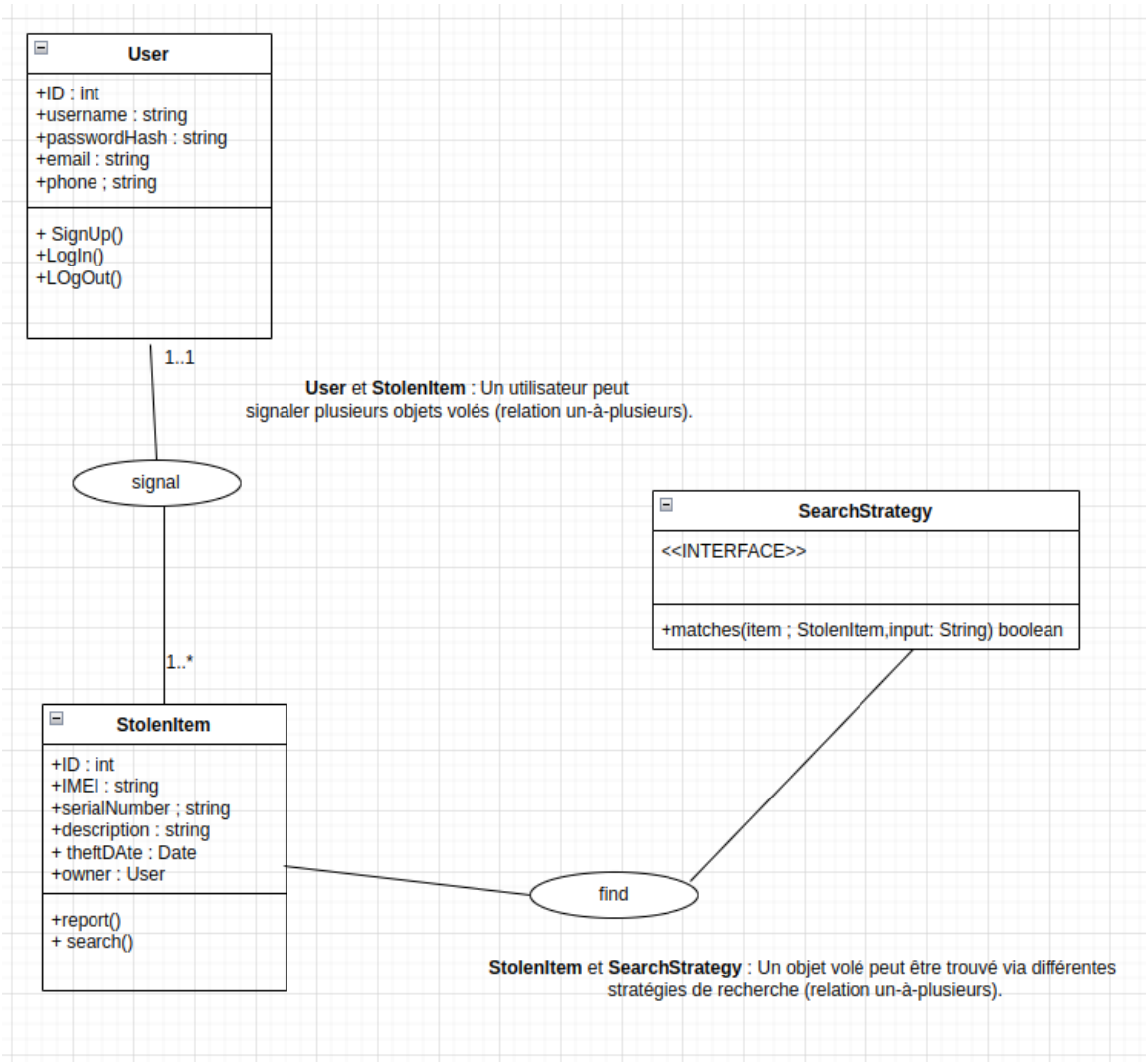
3.3 Diagramme de Déploiement



Description : Ce diagramme illustre l'environnement de déploiement de l'application, montrant les serveurs, les bases de données et les connexions réseau.

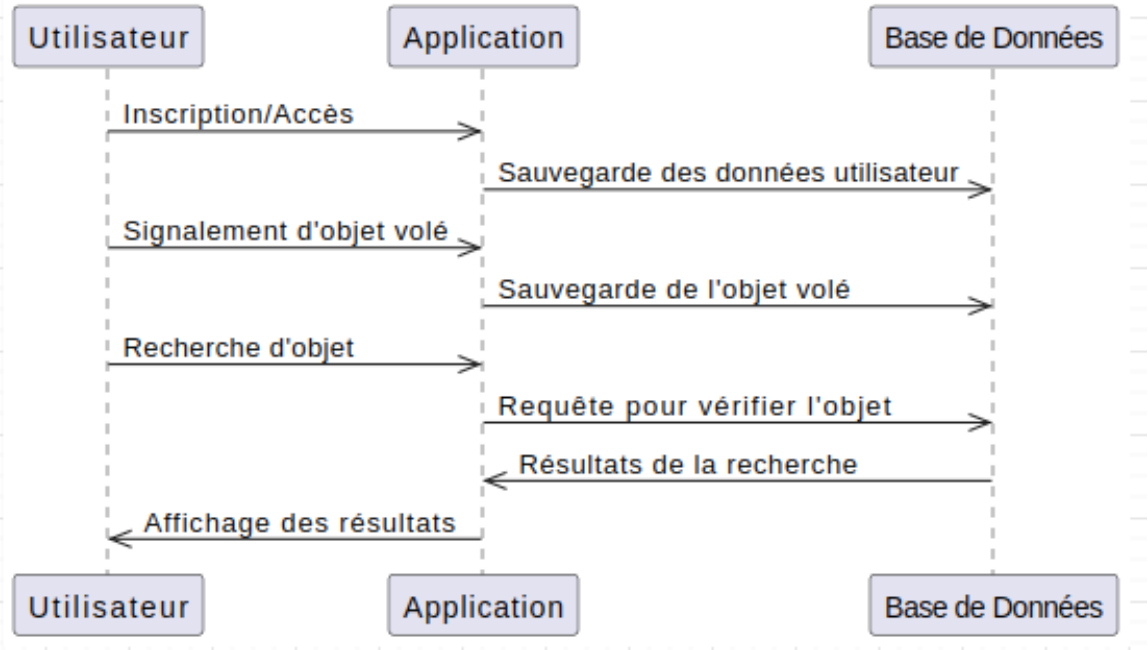
4 Modèle de Données

4.1 Diagramme Entité-Relation



Description : Ce diagramme présente les entités principales de l'application (Utilisateur, ObjetVolé) et leurs relations. Il détaille les attributs de chaque entité et les clés primaires/étrangères.

4.2 Diagramme d'Activité



Description : Ce diagramme décrit les flux de travail importants de l'application, tels que le signalement d'un objet volé, la recherche d'un objet, et la gestion des utilisateurs.

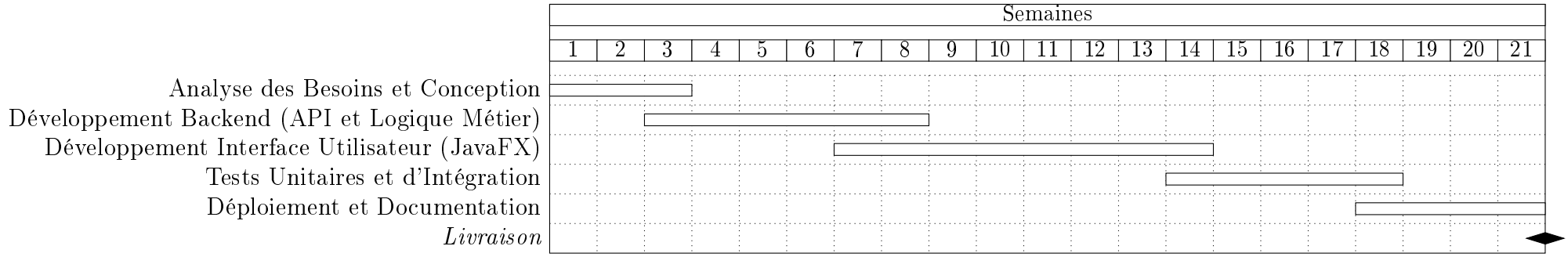
4.3 Schéma des Tables

Listing 1 – Schéma de la table Users

```sql
CREATE TABLE Users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(64) NOT NULL, -- SHA-256
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    creation_date DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE StolenItems (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    imei VARCHAR(15) UNIQUE,
    serial_number VARCHAR(50),
    description TEXT NOT NULL,
    theft_date DATETIME NOT NULL,
    owner_id INTEGER NOT NULL,
    report_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(owner_id) REFERENCES Users(id)
);
```

5 Plan de Développement


Semaines
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
Analyse des Besoins et Conception
Développement Backend (API et Logique Métier)
Développement Interface Utilisateur (JavaFX)
Tests Unitaires et d'Intégration
Déploiement et Documentation
Livraison

6 Sécurité

6.1 Vulnérabilités Potentielles

- Injections SQL : Attaques visant à manipuler les requêtes SQL pour accéder ou modifier des données non autorisées.
- Attaques par force brute : Tentatives répétées de deviner les mots de passe.
- Cross-Site Scripting (XSS) : Injection de scripts malveillants dans les pages web.
- Cross-Site Request Forgery (CSRF) : Exploitation de la session d'un utilisateur authentifié pour effectuer des actions non autorisées.

6.2 Mesures de Prévention

- Prévention des injections SQL : Utilisation systématique de Prepared Statements pour échapper les données utilisateur.
- Protection contre les attaques par force brute : Implémentation d'un système de verrouillage des comptes après un nombre limité de tentatives de connexion infructueuses, utilisation de CAPTCHA.
- Protection contre le XSS : Validation et nettoyage des données utilisateur avant de les afficher sur les pages web.
- Protection contre le CSRF : Utilisation de tokens CSRF pour vérifier l'authenticité des requêtes.
- Hachage des mots de passe : Utilisation de l'algorithme SHA-256 avec un sel unique pour chaque utilisateur, stockage des mots de passe hachés en base de données.
- Chiffrement des données sensibles : Chiffrement AES-256 des données sensibles stockées en base de données (par exemple, les informations personnelles).
- Gestion des sessions : Utilisation de sessions HTTP sécurisées avec expiration automatique après une période d'inactivité, invalidation des sessions lors de la déconnexion.

7 Implementation

7.1 Langage et Outils

- Langage : Java 23
- IDE : IntelliJ IDEA
- Base de données : SQLite
- Framework UI : JavaFX
- Outils de gestion de version : Git
- Outils de build : Maven

7.2 Environnement de Développement

- Système d'exploitation : Windows 10/11 ou Linux
- JDK : OpenJDK 23 ou supérieur
- SQLite : Version 3.35 ou supérieure
- JavaFX : Version 23 ou supérieure
- IDE : IntelliJ IDEA 2023 ou supérieur

- Outils de gestion de version : Git 2.30 ou supérieur
- Outils de build : Maven 3.6 ou supérieur
- Outils de test : JUnit 5, Mockito
- Outils de documentation : Javadoc

7.3 Gestion de Version

- Utilisation de Git pour le contrôle de version
- Branches : master, develop, feature/nom_feature
- Pull requests pour la révision de code
- Intégration continue avec GitHub Actions

7.4 Application du Principe SOLID

- **S : Single Responsibility Principle (Principe de Responsabilité Unique)**
  - *Description :* Chaque classe ou module doit avoir une seule responsabilité.
  - *Exemple :*

Listing 2 – Séparation des responsabilités dans les services

```
// Service dédié à l'authentification
class AuthService {
    public User authenticate(String username, String password) { ... }
}

// Service dédié aux opérations CRUD sur les objets volés
class StolenItemCrudService {
    public void createItem(StolenItem item) { ... }
}
```

  - *Avantages :* Facilite la maintenance, les tests et la réutilisation du code.
- **O : Open/Closed Principle (Principe Ouvert/Fermé)**
  - *Description :* Une classe doit être ouverte à l'extension, mais fermée à la modification.
  - *Exemple :*

Listing 3 – Extension par interfaces pour les stratégies de recherche

```
interface SearchStrategy {
    boolean matches(StolenItem item, String input);
}

class ImeiSearch implements SearchStrategy {
    public boolean matches(StolenItem item, String input) { ... }
}
```

  - *Avantages :* Permet d'ajouter de nouvelles fonctionnalités sans modifier le code existant, réduisant le risque d'introduction de bugs.
- **L : Liskov Substitution Principle (Principe de Substitution de Liskov)**
  - *Description :* Les sous-types doivent être substituables à leurs types de base sans altérer le comportement du programme.

- *Exemple :*

Listing 4 – Héritage contrôlé pour les types d'objets

```
abstract class ElectronicItem extends StolenItem {
    abstract String getPowerInfo();
}

class Phone extends ElectronicItem { ... } // Doit implémenter
   getPowerInfo()
```

- *Avantages :* Garantit la cohérence et la prédictibilité du code, facilite la maintenance et les tests.

- **I : Interface Segregation Principle (Principe de Ségrégation des Interfaces)**
  - *Description :* Une classe ne doit pas être forcée à implémenter des méthodes qu'elle n'utilise pas.
  - *Exemple :*

Listing 5 – Interfaces spécifiques pour les opérations sur les objets volés

```
interface StolenItemRepository {
    void add(StolenItem item);
    List<StolenItem> findByIMEI(String imei);
}

interface UserRepository {
    void add(User user);
    User findByUsername(String username);
}
```

Listing 6 – Interfaces spécialisées pour la persistance

```
interface ReadableDao<T> {
    T findById(String id);
}

interface WritableDao<T> {
    void save(T entity);
}
```

  - *Avantages :* Réduit les dépendances inutiles, améliore la flexibilité et la maintenabilité du code.

- **D : Dependency Inversion Principle (Principe d'Inversion des Dépendances)**
  - *Description :* Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre d'abstractions.
  - *Exemple :*

Listing 7 – Abstractions pour la couche de notification

```
interface NotificationService {
    void sendAlert(User user);
}

class EmailNotification implements NotificationService { ... }
```

  - *Avantages :* Permet de découpler les composants, améliore la testabilité et la flexibilité du code.

8 Testing

8.1 Stratégie

- Tests unitaires : Pour vérifier le bon fonctionnement de chaque composant individuellement (classes, méthodes). Utilisation de JUnit 5 et Mockito pour l'isolation des dépendances.
- Tests d'intégration : Pour vérifier l'interaction entre les différents composants (par exemple, la communication entre la couche de présentation et la couche de logique métier).
- Tests de performance : Pour évaluer la capacité de l'application à gérer une charge importante (nombre d'utilisateurs simultanés, volume de données). Utilisation de JMeter.
- Tests de sécurité : Pour identifier les vulnérabilités potentielles (injections SQL, XSS, etc.).

9 Documentation

- Guide d'installation : Pour expliquer comment installer et configurer l'application sur différents environnements.
- Manuel utilisateur : Pour expliquer comment utiliser l'application (inscription, signalement d'objets volés, recherche d'objets).
- Javadoc technique : Pour documenter le code source de l'application (classes, méthodes, interfaces).

10 Budget et Délais

Élément	**Estimation (heures)**
Analyse des besoins et conception	20
Développement Backend	60
Développement Interface Utilisateur	50
Tests	30
Documentation	20
Gestion de projet	10
Total	190

Date de livraison : 8 avril 2025