# **Mosaïque Capital - Plan de développement**

## **Vue d'ensemble du projet**

Mosaïque Capital est une application de gestion patrimoniale personnelle qui permet aux utilisateurs de centraliser, suivre et optimiser leur patrimoine à travers différentes classes d'actifs. L'objectif est de créer une plateforme complète où les utilisateurs peuvent avoir une vision consolidée de leurs actifs, suivre leur évolution et prendre des décisions financières éclairées.

## **Objectifs principaux**

* Offrir une vision globale et en temps réel du patrimoine
* Permettre le suivi et l'analyse de différentes classes d'actifs
* Automatiser l'agrégation des données financières
* Protéger les données sensibles des utilisateurs
* Proposer des analyses et visualisations pertinentes

## **Fonctionnalités clés**

### **Phase 1 - Fondations (MVP)**

* **Gestion des utilisateurs**
  + Inscription et authentification avec JWT
  + Authentification à deux facteurs (2FA)
  + Gestion des profils et préférences
* **Gestion des actifs**
  + CRUD complet pour différents types d'actifs
  + Catégorisation (immobilier, actions, obligations, etc.)
  + Suivi des valorisations dans le temps
* **Tableaux de bord basiques**
  + Calcul du patrimoine total
  + Visualisation de la répartition des actifs
  + Historique d'évolution

### **Phase 2 - Extensions**

* **Connectivité bancaire**
  + Intégration avec Budget Insight
  + Synchronisation des comptes bancaires
  + Import automatique des transactions
* **Portefeuille d'investissements**
  + Connexion avec courtiers financiers
  + Suivi des positions en titres
  + Analyse de performance
* **Gestion immobilière**
  + Suivi des revenus locatifs
  + Gestion des dépenses par bien
  + Calcul de rentabilité

### **Phase 3 - Avancées**

* **Support des cryptomonnaies**
* **Optimisation fiscale**
* **Rapports personnalisés**
* **Système de notifications**
* **API mobile avancée**

## **Approche technique**

### **Architecture**

* **Backend**: Java 21 + Spring Boot 3.x
* **Base de données**: MySQL 8 (prod) / H2 (tests)
* **Sécurité**: Spring Security + JWT + TOTP (2FA)
* **Persistence**: JPA/Hibernate
* **API**: REST avec documentation OpenAPI

### **Structure du projet**

com.master.mosaique\_capital

├── config // Configuration Spring

├── controller // Contrôleurs REST

├── dto // Objets de transfert

├── entity // Entités JPA

├── enums // Énumérations

├── exception // Gestion des exceptions

├── mapper // Mappers DTO<->Entity

├── repository // Repositories JPA

├── security // Configuration sécurité

├── service // Services métier

└── util // Classes utilitaires

### **Approche de développement**

En tant que développeur unique sur le backend:

1. **Itérative**: Développement par incréments successifs
2. **Tests en priorité**: Assurer la qualité du code avec des tests solides
3. **Sécurité native**: Intégrer les bonnes pratiques de sécurité dès le début
4. **Documentation**: Documenter au fur et à mesure du développement
5. **Revues personnelles**: Mettre en place des routines de revue de code

## **Plan de mise en œuvre (Roadmap)**

### **Sprint 1-2: Infrastructure et sécurité (4 semaines)**

* Configuration du projet Spring Boot
* Mise en place de la sécurité (JWT, 2FA)
* Structure de base de données
* CI/CD basique

### **Sprint 3-4: Gestion des actifs (4 semaines)**

* Implémentation des entités de base
* CRUD pour les actifs et valorisations
* Tests unitaires et d'intégration
* Documentation API

### **Sprint 5-6: Tableaux de bord (4 semaines)**

* Services d'analyse patrimoniale
* Calculs de répartition et évolution
* Optimisation des requêtes
* Refactoring et nettoyage

### **Sprint 7-8: Intégrations externes (4 semaines)**

* Intégration avec Budget Insight
* Synchronisation des données externes
* Gestion des imports/exports
* Tests de charge

### **Maintenance et évolution continue**

## **Méthodologie**

* **Planning**: Planification hebdomadaire avec objectifs clairs
* **Suivi**: Utilisation d'outils simples (Trello/GitHub Projects)
* **Revue**: Auto-revue du code tous les vendredis
* **Apprentissage**: 2-4 heures par semaine dédiées à la veille technique

## **Principes de développement**

* **Code propre**: Respecter les principes SOLID et les conventions Java
* **Documentation**: JavaDoc pour les classes/méthodes importantes
* **Tests**: Coverage minimum de 80% sur le code critique
* **Sécurité**: Audit régulier avec OWASP
* **Performances**: Optimiser les requêtes et les traitements

## **Risques identifiés**

1. **Charge de travail**: Étant seul sur le projet, risque de sous-estimation des efforts → *Solution*: Découper en petites tâches, prioritiser strictement
2. **Dette technique**: Accumulation rapide sans équipe de revue → *Solution*: Sessions de refactoring planifiées, outils d'analyse de code
3. **Intégrations externes**: Complexité des APIs tierces → *Solution*: Commencer par des mocks, puis intégrer progressivement
4. **Sécurité**: Protection des données financières sensibles → *Solution*: Audits de sécurité réguliers, chiffrement des données sensibles