# Documentation Complète du Projet

## Plateforme d'Incubation Start-ups avec Intelligence Artificielle

---

## 📋 CONTEXTE GÉNÉRAL

### Vue d'Ensemble

Ce projet est une plateforme web destinée aux start-ups marocaines pour les aider à structurer leur activité, générer des pitchs professionnels via IA, et se connecter avec des investisseurs potentiels.

### Équipe & Contraintes

- **Équipe** : 2 développeurs (YOUSSEF HOUSSAM - Backend/IA, HAJAR NIYYA - Frontend)
- **Durée** : 4 semaines
- **Contexte académique** : Projet de fin de semestre avec d'autres projets en parallèle
- **Technologies imposées** : Spring Boot (Backend), React.js (Frontend), PostgreSQL (BDD)

### Objectifs du Projet

1. Permettre aux start-ups de créer un profil professionnel complet
2. Générer automatiquement des pitchs d'investissement via Intelligence Artificielle (Google Gemini API)
3. Matcher les start-ups avec des investisseurs compatibles selon leur secteur
4. Fournir un tableau de bord avec statistiques et suivi de progression

---

## 🎯 FONCTIONNALITÉS PRINCIPALES

### 1. Authentification & Gestion Utilisateurs

- Inscription avec email/mot de passe
- Connexion sécurisée avec JWT tokens
- Gestion de profil utilisateur
- Rôles : STARTUP, INVESTOR, ADMIN

### 2. Gestion des Profils Start-ups

- Création et édition du profil entreprise
- Informations : nom, secteur d'activité, description, tags
- Gestion de l'équipe fondatrice (membres avec rôles)
- Suivi des jalons/milestones (tâches à accomplir)
- Calcul automatique du score de complétion du profil (0-100%)

### 3. Générateur de Pitch IA (CŒUR DU PROJET)

**Fonctionnalité phare** : Génération automatique de pitchs professionnels

- Formulaire avec 4 questions clés :
  - Quel problème résolvez-vous ?
  - Quelle est votre solution ?
  - Qui sont vos clients cibles ?
  - Quel est votre avantage concurrentiel ?
- Appel à l'API Google Gemini pour générer un pitch de 100-150 mots
- Sauvegarde automatique de tous les pitchs générés
- Historique consultable
- Export en PDF (optionnel)

**4. Matching Investisseurs**

- Base de données d'investisseurs avec leurs critères (secteurs, montants)
- Algorithme de matching simple basé sur :
  - Secteurs d'activité compatibles (70% du score)
  - Montants d'investissement recherchés (20%)
  - Localisation géographique (10%)
- Affichage des investisseurs classés par score de compatibilité
- Système de demande de connexion

**5. Tableau de Bord & Analytics**

- Dashboard avec métriques clés :
  - Score de complétion du profil
  - Nombre de pitchs générés
  - Nombre d'investisseurs matchés
  - Jalons complétés
- Graphiques de progression
- Historique des activités récentes

---

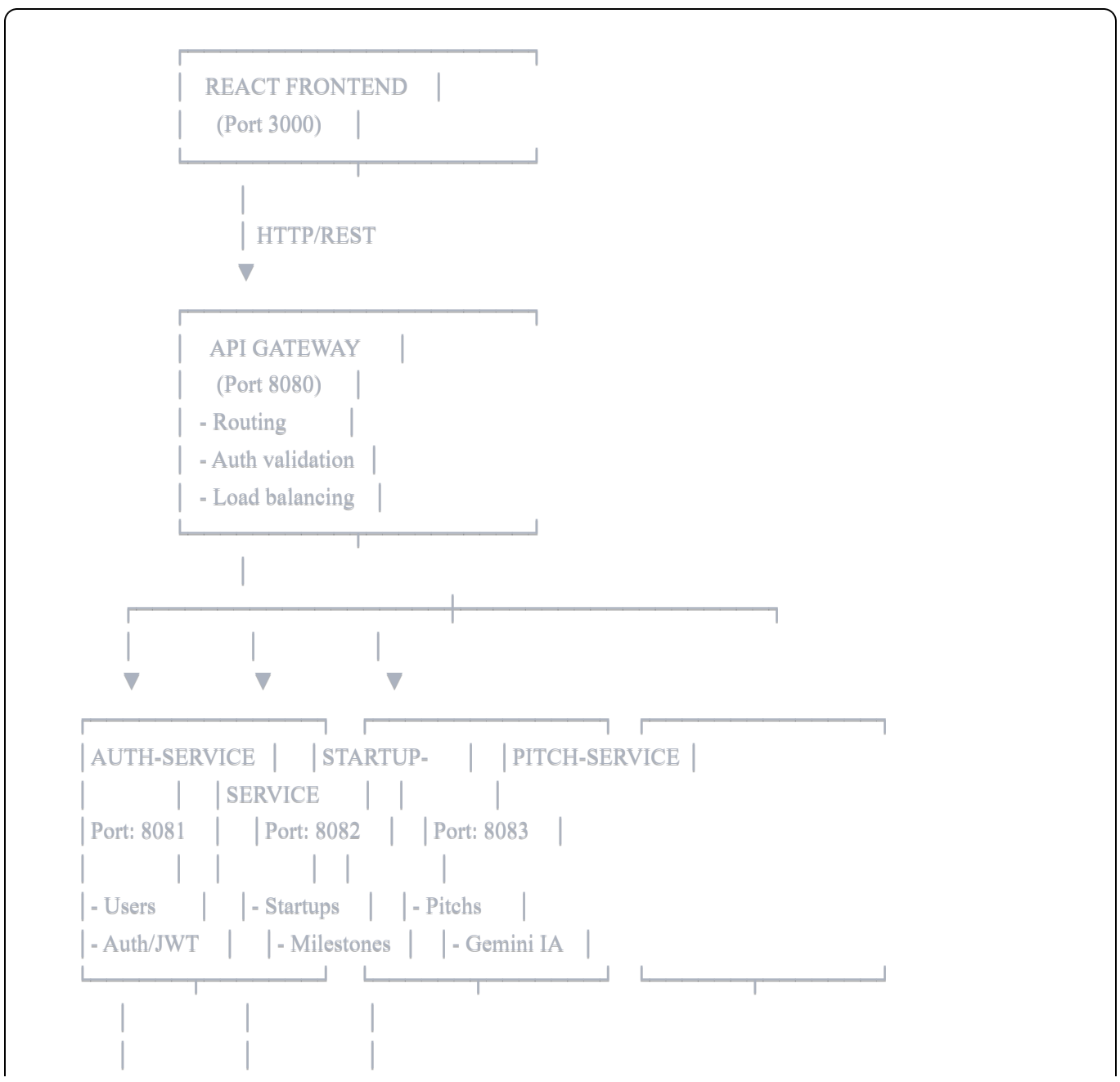
## 🧬 ARCHITECTURE MICROSERVICES

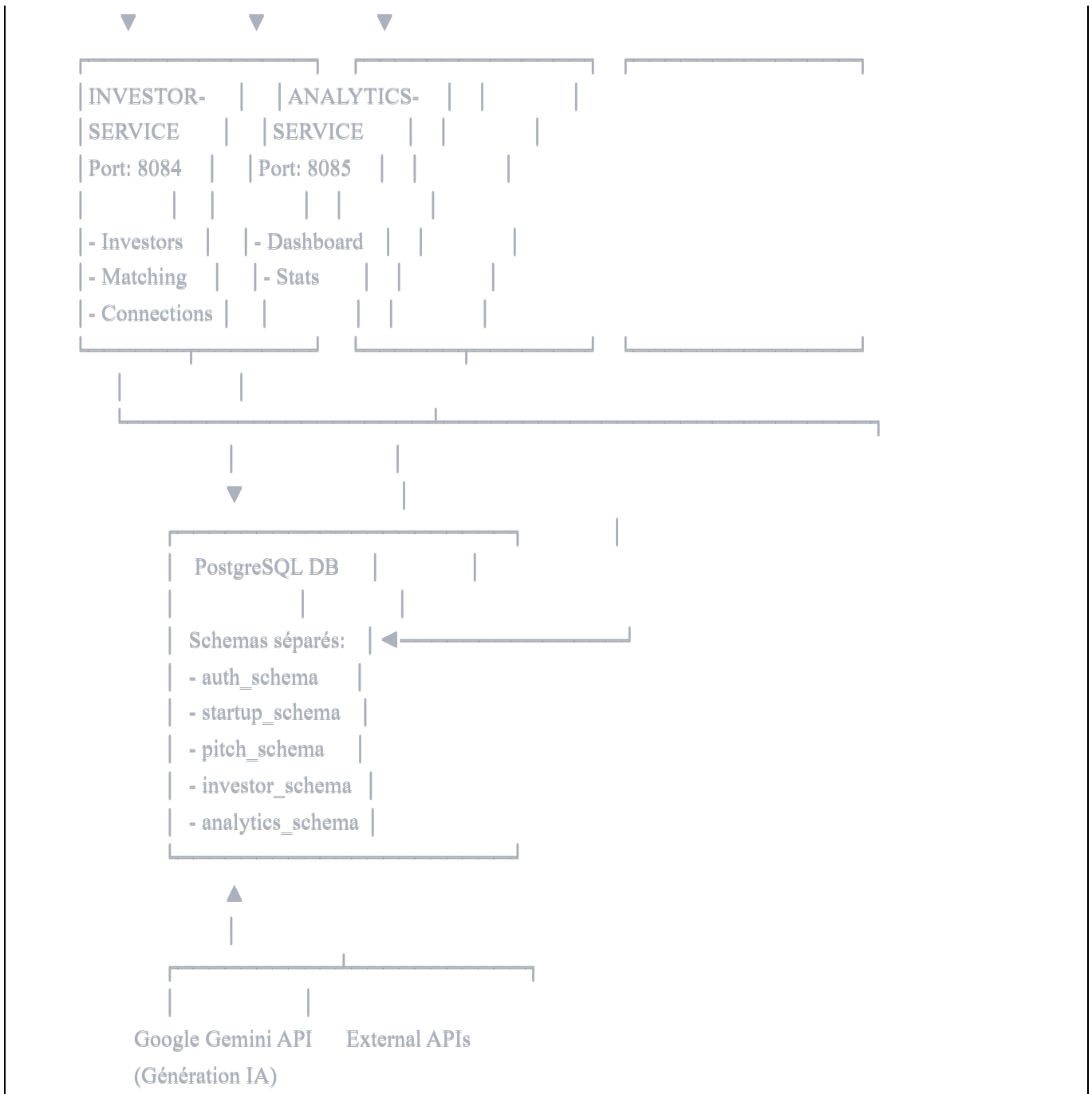
**Pourquoi des Microservices ?**

L'architecture microservices sépare l'application en plusieurs services indépendants, chacun responsable d'un domaine métier spécifique. Cela permet :

- Scalabilité indépendante de chaque service
- Maintenance plus facile (chaque service est petit)
- Technologies différentes possibles par service
- Déploiement indépendant

**Schéma Global de l'Architecture**






---

## 📦 DÉTAIL DES MICROSERVICES

**SERVICE 1 : AUTH-SERVICE (Port 8081)**

**Responsabilité** : Authentification et gestion des utilisateurs

**Base de Données (Schema: auth_schema)**

**Table: users**

```
- id (UUID, Primary Key)
- email (String, Unique)
- password (String, Hashed with BCrypt)
- role (Enum: STARTUP, INVESTOR, ADMIN)
- isActive (Boolean)
- createdAt (Timestamp)
- updatedAt (Timestamp)
```

**APIs Exposées**

**AuthController**

- `POST /api/auth/register` - Inscription d'un nouvel utilisateur
  - Input: email, password, role
  - Output: User object + JWT token
- `POST /api/auth/login` - Connexion utilisateur
  - Input: email, password
  - Output: JWT token (valide 24h)
- `POST /api/auth/logout` - Déconnexion
  - Input: JWT token
  - Output: Success message
- `POST /api/auth/refresh-token` - Rafraîchir le token expiré

- Input: Refresh token
- Output: Nouveau JWT token

**UserController**

- `GET /api/users/me` - Récupérer profil utilisateur connecté
- `PUT /api/users/me` - Modifier profil utilisateur
- `DELETE /api/users/me` - Supprimer son compte
- `GET /api/users/{id}` - Récupérer un utilisateur (admin only)

**Sécurité Implémentée**

- Passwords hashés avec BCrypt (force 10)
- JWT tokens avec expiration 24h
- Refresh tokens avec expiration 7 jours
- Validation email format
- Rate limiting sur login (max 5 tentatives/5min)

---

**SERVICE 2 : STARTUP-SERVICE (Port 8082)**

**Responsabilité** : Gestion des profils start-ups et équipes

**Base de Données (Schema: startup_schema)**

**Table: startups**

```
- id (UUID, Primary Key)
- userId (UUID, Foreign Key → auth_schema.users.id)
- nom (String, Not Null)
- secteur (String: 'FinTech', 'EdTech', 'HealthTech', 'E-commerce', etc.)
- description (Text, Max 500 chars)
- tags (String, Comma-separated)
- profileCompletion (Integer, 0-100)
- logo (String, URL)
- siteWeb (String, URL)
- dateCreation (Date)
- createdAt (Timestamp)
- updatedAt (Timestamp)
```

**Table: founder_members**

```
- id (UUID, Primary Key)
- startupId (UUID, Foreign Key → startups.id)
- nom (String)
- role (String: 'CEO', 'CTO', 'CMO', etc.)
- linkedIn (String, URL)
- photo (String, URL)
```

**Table: milestones**

```
- id (UUID, Primary Key)
- startupId (UUID, Foreign Key → startups.id)
- titre (String)
- description (Text)
- statut (Enum: TODO, IN_PROGRESS, COMPLETED)
- dateEcheance (Date)
- completedAt (Timestamp)
```

**APIs Exposées**

**StartupController**

- `POST /api/startups` - Créer un profil start-up

- Input: nom, secteur, description, tags
- Output: Startup object avec profileCompletion calculé

- `GET /api/startups/me` - Récupérer ma start-up
  - Output: Startup complète avec membres et milestones

- `PUT /api/startups/me` - Modifier ma start-up
  - Input: Champs à modifier
  - Output: Startup mise à jour + nouveau profileCompletion

- `GET /api/startups` - Liste toutes start-ups (paginated)
  - Query params: page, size, secteur
  - Output: Page<Startup>

- `GET /api/startups/search?secteur={secteur}` - Recherche par secteur

**MilestoneController**

- `POST /api/startups/me/milestones` - Créer un jalon
- `GET /api/startups/me/milestones` - Liste mes jalons
- `PUT /api/milestones/{id}` - Modifier un jalon
- `PATCH /api/milestones/{id}/complete` - Marquer comme complété
- `DELETE /api/milestones/{id}` - Supprimer un jalon

**TeamController**

- `POST /api/startups/me/team` - Ajouter membre équipe
- `GET /api/startups/me/team` - Liste membres équipe
- `PUT /api/team/{id}` - Modifier membre
- `DELETE /api/team/{id}` - Supprimer membre

**Logique Métier Importante**

**Calcul du profileCompletion** :

```
Score = 0
Si nom renseigné → +20 points
Si secteur renseigné → +20 points
Si description renseignée → +20 points
Si tags renseignés (min 2) → +20 points
Si équipe ajoutée (min 2 membres) → +20 points
Total = Score / 100
```

**Communication avec autres services**

- **Auth-Service** : Vérifier que userId existe via Feign Client
- **Pitch-Service** : Fournir infos startup pour génération pitch

---

**SERVICE 3 : PITCH-SERVICE (Port 8083)**

**Responsabilité** : Génération de pitchs avec Intelligence Artificielle

**Base de Données (Schema: pitch_schema)**

**Table: pitchs**

- id (UUID, Primary Key)
- startupId (UUID, Foreign Key → startup_schema.startups.id)
- probleme (Text, Max 500 chars)
- solution (Text, Max 500 chars)
- cible (Text, Max 300 chars)
- avantage (Text, Max 300 chars)
- pitchGenere (Text, Résultat de l'IA)
- type (Enum: ELEVATOR, DECK, VALUE_PROP)
- rating (Integer, 1-5, nullable)
- isFavorite (Boolean, default false)
- createdAt (Timestamp)
- updatedAt (Timestamp)

**Table: pitch_templates**

- id (UUID, Primary Key)
- nom (String)
- prompt (Text, Template du prompt pour l'IA)
- secteur (String, nullable)
- isActive (Boolean)

**APIs Exposées**

**PitchController**

- `POST /api/pitchs/generate` - **ENDPOINT PRINCIPAL**
  - Input JSON:

```
json

{
 "probleme": "Les entrepreneurs perdent du temps...",
 "solution": "Une plateforme IA qui...",
 "cible": "Start-ups marocaines en phase d'amorçage",
 "avantage": "Génération automatique en 2 minutes"
}
```

- Process:
  1. Valide les inputs
  2. Récupère infos startup via StartupServiceClient
  3. Construit le prompt pour Gemini
  4. Appelle Google Gemini API
  5. Sauvegarde le pitch en BDD
  6. Retourne le pitch généré
- Output:

```
json

{
 "id": "uuid",
 "pitchGenere": "Dans un écosystème entrepreneurial...",
 "createdAt": "2024-11-22T10:00:00"
}
```

- `GET /api/pitchs/me` - Historique de mes pitchs
  - Output: Liste triée par date (plus récent en premier)
- `GET /api/pitchs/{id}` - Récupérer un pitch spécifique
- `PUT /api/pitchs/{id}` - Modifier un pitch manuellement
- `DELETE /api/pitchs/{id}` - Supprimer un pitch
- `PATCH /api/pitchs/{id}/favorite` - Toggle favori

- `POST /api/pitchs/{id}/rate` - Noter le pitch (1-5 étoiles)

**AIController**

- `POST /api/ai/generate-elevator` - Générer pitch elevator (30 sec)
- `POST /api/ai/generate-deck` - Structure complète pitch deck
- `POST /api/ai/improve` - Améliorer un pitch existant
- `GET /api/ai/suggestions` - Suggestions d'amélioration

**Intégration Google Gemini API**

**Configuration**

```yaml
gemini:
  api-key: ${GEMINI_API_KEY}
  model: gemini-pro
  max-tokens: 500
  temperature: 0.7
```

**Prompt Engineering (Exemple)**

```
Tu es un expert en pitchs de start-ups et en levées de fonds.

Contexte de la start-up :
- Secteur : {secteur}
- Nom : {nom}

Informations fournies :
- Problème : {probleme}
- Solution : {solution}
- Cible : {cible}
- Avantage : {avantage}

Génère un elevator pitch professionnel de 120-150 mots maximum qui :
1. Accroche dès la première phrase
2. Présente clairement le problème et la solution
3. Met en avant la proposition de valeur unique
4. Est orienté bénéfices pour les clients
5. Se termine par un call-to-action implicite

Ton : Professionnel, confiant, concis
Format : Un seul paragraphe fluide sans bullet points
```

**Gestion des Erreurs IA**

- Timeout après 15 secondes
- Retry automatique (max 2 fois)
- Fallback : Message d'erreur user-friendly
- Logging de toutes les requêtes IA

**Communication avec autres services**

- **Startup-Service** : Récupérer infos startup (nom, secteur) via Feign
- **Google Gemini** : Appel API externe pour génération

---

**SERVICE 4 : INVESTOR-SERVICE (Port 8084)**

**Responsabilité** : Gestion investisseurs et système de matching

**Base de Données (Schema: investor_schema)**

**Table: investors**

- id (UUID, Primary Key)
- userId (UUID, Foreign Key → auth_schema.users.id)
- nom (String)
- type (Enum: VC, BUSINESS_ANGEL, INCUBATOR, CORPORATE_VC)
- secteursInterets (String, JSON Array ex: ["FinTech","EdTech"])
- montantMin (Decimal, Montant minimum d'investissement)
- montantMax (Decimal, Montant maximum d'investissement)
- description (Text)
- localisation (String)
- portfolio (Text, Startups déjà investies)
- siteWeb (String, URL)
- email (String)
- createdAt (Timestamp)
- updatedAt (Timestamp)

**Table: matching_results**

- id (UUID, Primary Key)
- startupId (UUID, Foreign Key)
- investorId (UUID, Foreign Key)
- score (Integer, 0-100)
- criteria (JSON, Détail du calcul)
- createdAt (Timestamp)
- isViewed (Boolean)

**Table: connection_requests**

- id (UUID, Primary Key)
- startupId (UUID, Foreign Key)
- investorId (UUID, Foreign Key)
- message (Text, Message personnalisé)
- statut (Enum: PENDING, ACCEPTED, REJECTED)
- createdAt (Timestamp)
- respondedAt (Timestamp, nullable)

**APIs Exposées**

**InvestorController**

- `POST /api/investors` - Créer profil investisseur
- `GET /api/investors/me` - Mon profil investisseur
- `PUT /api/investors/me` - Modifier profil
- `GET /api/investors` - Liste tous investisseurs (paginated)
- `GET /api/investors/{id}` - Détails d'un investisseur
- `GET /api/investors/search?secteur={secteur}` - Recherche

**MatchingController**

- `GET /api/matching/for-me` - **ENDPOINT CLÉ**
  - Process:
    1. Récupère la startup de l'utilisateur connecté
    2. Récupère tous les investisseurs
    3. Pour chaque investisseur, calcule le score de matching
    4. Trie par score décroissant
    5. Retourne Top 20
  - Output:

```
json
```

```
[
  {
    "investor": {...},
    "score": 85,
    "criteria": {
      "secteurMatch": true,
      "montantCompatible": true,
      "localisationMatch": false
    }
  }
]
```


- `GET /api/matching/startups` - Start-ups matchées (vue investisseur)
- `POST /api/matching/calculate` - Force le recalcul du matching
- `GET /api/matching/score/{investorId}` - Score pour un investisseur spécifique

**ConnectionController**

- `POST /api/connections/request` - Demander une connexion
  - Input: investorId, message personnalisé
  - Crée une demande avec statut PENDING
  - Envoie notification email à l'investisseur
- `GET /api/connections/received` - Demandes reçues (investisseur)
- `GET /api/connections/sent` - Demandes envoyées (startup)
- `PUT /api/connections/{id}/accept` - Accepter connexion
- `PUT /api/connections/{id}/reject` - Rejeter connexion
- `GET /api/connections/active` - Connexions actives établies

**Algorithme de Matching**

**Calcul du Score (sur 100)**

```
Score Total = 0

1. Match Secteur (70 points max)
  - Si secteur startup IN secteursInterets investisseur ⟶ +70
  - Sinon ⟶ +0

2. Match Montant (20 points max)
  - Si startup a un "montant_recherche" défini:
    - Si montant_recherche BETWEEN montantMin AND montantMax ⟶ +20
    - Sinon ⟶ +10 (partiellement compatible)
  - Si pas de montant défini ⟶ +10 (par défaut)

3. Match Localisation (10 points max)
  - Si localisation startup == localisation investisseur ⟶ +10
  - Si même région ⟶ +5
  - Sinon ⟶ +0

Score Final = Score Total (arrondi)
```

**Exemple Concret**

Startup:
- Secteur: "FinTech"
- Localisation: "Casablanca"
- Montant recherché: 500,000 MAD

Investisseur A:
- Secteurs: ["FinTech", "EdTech"]
- Montants: 300,000 - 1,000,000 MAD
- Localisation: "Casablanca"

Calcul:
- Secteur match ⟶ +70
- Montant compatible ⟶ +20
- Localisation identique ⟶ +10
⟶ Score Total: 100/100 ✅

Investisseur B:
- Secteurs: ["HealthTech", "AgriTech"]
- Montants: 100,000 - 500,000 MAD
- Localisation: "Rabat"

Calcul:
- Secteur NO match ⟶ +0
- Montant compatible ⟶ +20
- Localisation différente ⟶ +0
⟶ Score Total: 20/100

**Communication avec autres services**

- **Startup-Service** : Récupérer infos startup pour matching
- **Auth-Service** : Vérifier rôle utilisateur (INVESTOR vs STARTUP)

---

**SERVICE 5 : ANALYTICS-SERVICE (Port 8085) - OPTIONNEL**

**Responsabilité** : Statistiques, métriques et tableaux de bord

**Base de Données (Schema: analytics_schema)**

**Table: dashboards**

- id (UUID, Primary Key)
- userId (UUID, Foreign Key)
- profileCompletion (Integer)
- pitchsGenerated (Integer)
- matchingInvestors (Integer)
- connectionsActive (Integer)
- milestonesCompleted (Integer)
- lastUpdated (Timestamp)

**Table: activities**

- id (UUID, Primary Key)
- userId (UUID, Foreign Key)
- type (Enum: LOGIN, PITCH_GENERATED, CONNECTION_REQUEST, etc.)
- description (String)
- metadata (JSON)
- createdAt (Timestamp)

**APIs Exposées**

**DashboardController**

- `GET /api/dashboard/me` - Tableau de bord complet
  - Agrège données de tous les services
  - Output:

```json
{
  "profileCompletion": 75,
  "pitchsGenerated": 5,
  "matchingInvestors": 12,
  "connectionsActive": 3,
  "milestonesCompleted": 8,
  "recentActivities": [...]
}
```

- `GET /api/dashboard/stats` - Statistiques détaillées
- `GET /api/dashboard/activities` - Activités récentes (timeline)
- `GET /api/dashboard/progress` - Progression hebdomadaire

**AnalyticsController (Admin)**

- `GET /api/analytics/overview` - Vue d'ensemble plateforme
  - Total startups, pitchs générés, matching success rate
- `GET /api/analytics/startups-stats` - Stats par secteur
- `GET /api/analytics/pitchs-stats` - Stats génération IA
- `GET /api/analytics/matching-stats` - Taux de conversion matching

**Communication avec autres services**

Ce service appelle **TOUS les autres services** pour agréger les données :

- **Startup-Service** : Récupérer completion, milestones
- **Pitch-Service** : Nombre de pitchs
- **Investor-Service** : Matching et connexions

---

**API GATEWAY (Port 8080)**

**Responsabilité** : Point d'entrée unique de l'application

**Rôles du Gateway**

1. **Routing** : Diriger chaque requête vers le bon microservice
2. **Authentification** : Valider le JWT avant de transférer
3. **Load Balancing** : Répartir la charge (si plusieurs instances)
4. **Rate Limiting** : Limiter le nombre de requêtes par utilisateur
5. **CORS** : Gérer les autorisations cross-origin pour le frontend
6. **Logging** : Logger toutes les requêtes entrantes

**Configuration des Routes**

```yaml
```

```yaml
spring:
  cloud:
    gateway:
      routes:
        # Auth Service
        - id: auth-service
          uri: http://localhost:8081
          predicates:
            - Path=/api/auth/**, /api/users/**

        # Startup Service
        - id: startup-service
          uri: http://localhost:8082
          predicates:
            - Path=/api/startups/**, /api/milestones/**, /api/team/**
          filters:
            - JwtAuthenticationFilter

        # Pitch Service
        - id: pitch-service
          uri: http://localhost:8083
          predicates:
            - Path=/api/pitchs/**, /api/ai/**
          filters:
            - JwtAuthenticationFilter

        # Investor Service
        - id: investor-service
          uri: http://localhost:8084
          predicates:
            - Path=/api/investors/**, /api/matching/**, /api/connections/**
          filters:
            - JwtAuthenticationFilter

        # Analytics Service
        - id: analytics-service
          uri: http://localhost:8085
          predicates:
            - Path=/api/dashboard/**, /api/analytics/**
          filters:
            - JwtAuthenticationFilter
```

**Filtres Implémentés**

- **JwtAuthenticationFilter** : Vérifie le token JWT dans le header Authorization
- **LoggingFilter** : Log method, path, user, timestamp
- **RateLimitingFilter** : Max 100 requêtes/min par utilisateur

---

## 🔗 COMMUNICATION INTER-SERVICES

**Méthode : OpenFeign (REST Synchrone)**

**Principe** : Un service appelle un autre service via son API REST

**Exemple : Pitch-Service appelle Startup-Service**

```java
```

```java
// Dans pitch-service/client/StartupServiceClient.java
@FeignClient(name = "startup-service", url = "http://localhost:8082")
public interface StartupServiceClient {

    @GetMapping("/api/startups/{id}")
    StartupDTO getStartup(@PathVariable("id") UUID startupId);
}

// Utilisation dans PitchService.java
@Service
public class PitchService {

    @Autowired
    private StartupServiceClient startupClient;

    public Pitch generatePitch(...) {
        // Récupérer infos startup
        StartupDTO startup = startupClient.getStartup(startupId);

        // Utiliser ces infos dans le prompt IA
        String prompt = buildPrompt(startup.getSecteur(), ...);
        ...
    }
}
```

**Schéma des Communications**

```
PITCH-SERVICE
    │
    ├──(Feign)──► STARTUP-SERVICE (récupérer secteur, nom)
    │
    └──(HTTP)──► GOOGLE GEMINI API (génération IA)


INVESTOR-SERVICE
    │
    └──(Feign)──► STARTUP-SERVICE (récupérer startups pour matching)


ANALYTICS-SERVICE
    │
    ├──(Feign)──► STARTUP-SERVICE (stats profils)
    ├──(Feign)──► PITCH-SERVICE (stats pitchs)
    └──(Feign)──► INVESTOR-SERVICE (stats matching)


TOUS LES SERVICES
    │
    └──(Feign)──► AUTH-SERVICE (valider tokens, récupérer user)
```

**Gestion des Erreurs de Communication**

**Cas : Service cible indisponible**

java

```
@FeignClient(name = "startup-service",
             url = "http://localhost:8082",
             fallback = StartupServiceFallback.class)
public interface StartupServiceClient {
    @GetMapping("/api/startups/{id}")
    StartupDTO getStartup(@PathVariable UUID id);
}

// Classe fallback
@Component
public class StartupServiceFallback implements StartupServiceClient {
    @Override
    public StartupDTO getStartup(UUID id) {
        // Retourner données par défaut ou null
        return new StartupDTO("Unknown", "Default");
    }
}
```

---

## 💾 SCHÉMA COMPLET BASE DE DONNÉES

**PostgreSQL Database: `startup_platform`**

```
sql

-- Schema 1: auth_schema
CREATE SCHEMA auth_schema;

CREATE TABLE auth_schema.users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL CHECK (role IN ('STARTUP', 'INVESTOR', 'ADMIN')),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Schema 2: startup_schema
CREATE SCHEMA startup_schema;

CREATE TABLE startup_schema.startups (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES auth_schema.users(id) ON DELETE CASCADE,
    nom VARCHAR(255) NOT NULL,
    secteur VARCHAR(100),
    description TEXT,
    tags VARCHAR(500),
    profile_completion INTEGER DEFAULT 0 CHECK (profile_completion BETWEEN 0 AND 100),
    logo VARCHAR(500),
    site_web VARCHAR(500),
    date_creation DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE
```