Portfolio - Étape 4

Développement et Exécution du MVP

Carte interactive de l'univers du Seigneur des Anneaux & Créations artistiques SDA

0. Planification et Définition des Sprints

0.1. Vue d'ensemble du développement

Le développement du MVP est organisé en 5 sprints de 1 semaine chacun, couvrant les semaines 7 à 11 du projet. Cette organisation permet une progression itérative et des ajustements réguliers basés sur les retours d'équipe.

0.2. Priorisation MoSCoW

La priorisation des fonctionnalités suit le framework MoSCoW établi dans la documentation technique :

- Must Have : Carte interactive, fiches détaillées, système d'authentification, publication de créations, modération
- Should Have : Recherche et filtres, gestion des utilisateurs
- Could Have : Messagerie temps réel, édition/suppression de créations
- Won't Have : E-commerce, arbre généalogique interactif

0.3. Planification détaillée des sprints

Sprint 1 - Semaine 7 : Mise en Place du Back-End et de la Base de Données

Objectifs :

- Définir et structurer la base de données du projet
- Mettre en place les premières API et façades (User, Auth, Admin, Reviews, etc.)
- Poser les fondations du back-end avec Flask et PostgreSQL
- Réaliser les premiers tests fonctionnels via Postman

Tâches réalisées :

- Création de la structure initiale du projet (branche feature/structure)
- Construction de la base de données à partir du document de conception, avec ajustements techniques :
  - Passage à `BIGSERIAL` pour les IDs (auto-incrémentation PostgreSQL)
  - Utilisation de `SMALLINT` pour `start_year` et `death_year`
  - Conversion de `x` et `y` en `GEOMETRY(POINT, 0)` dans `map_marker`
  - Ajout d'une table `relation_type` pour gérer les liens entre personnages et histoires
    • Insertion de données initiales (utilisateurs, races, types de relations)
    • Configuration de l'application Flask et création des premières classes (User)
    • Implémentation des modèles reviews et image_post
    • Mise en place des premières façades, repositories et routes dans __init__.py
    • Développement et tests des API User, Auth et Admin avec Postman (GET, POST,

PUT validés)
• Ajout des API et façades pour Race, Character, History et Reviews
• Refactorisation avec facade2 pour centraliser la gestion des entités
• Modification du modèle image_post pour stocker les images directement en base (BYTEA + image_mime_type)
• Mise en place d'environnements locaux Flask + PostgreSQL distincts pour Robin et Timi

Critères d'acceptation :

- Base de données stable, cohérente et connectée à Flask
- APIs principales fonctionnelles et testées avec Postman
- Authentification opérationnelle (token récupéré)
- Environnements de développement indépendants configurés et prêts pour les itérations suivantes

Sprint 2 - Semaine 8 : Tests Back-End et Structuration des API

Objectifs :

- Finaliser et tester l'ensemble des routes API (GET, POST, PUT, DELETE)
- Structurer proprement les modèles, façades et routes pour toutes les entités principales
- Mettre en place une base de données de test et renforcer la fiabilité du back-end
- Premiers apprentissages sur React et exploration des outils de cartographie

Tâches réalisées :

- Validation des tests API Postman pour toutes les routes principales (GET_id, GET_ALL, POST, PUT, DELETE)
- Implémentation et test de la Search API (race, personnage, histoire)
- Ajout des modèles et façades pour Place, Marker et Région
- Tests API Map Marker et Map Région validés
- Création et séparation des fichiers schema.sql (tables) et data.sql (inserts)
- Amélioration des routes API Reviews (filtrage par utilisateur ou image)
- Correction du modèle Image_Post et mise à jour de la base locale
- Utilisation de QGIS pour le géoréférencement et la visualisation des points/polygones
- Mise en place des tests unitaires complets (modèles, façades, API) via Pytest sur une base de test dédiée
- Réalisation d'un script Postman pour automatiser les tests API
- Début d'apprentissage de React en prévision de la refonte front-end

Critères d'acceptation :

- Toutes les routes API validées et testées
- Base de données stable et structurée
- Environnement de test fonctionnel
- Fondations solides pour le passage à la partie front-end

Sprint 3 - Semaine 9 : Refonte Front-End et Intégration React

Objectifs :

- Refonte complète du front-end avec React
- Structuration claire de l'architecture du projet
- Mise en place des premiers composants et des routes React
- Familiarisation avec Leaflet pour la future carte interactive

Tâches réalisées :

- Suppression des fichiers et dossiers obsolètes
- Installation et configuration de React via Vite (fichiers main.jsx et app.jsx)
- Création et début d'intégration des composants Header, Footer et Navigation
- Ajout et stylisation des pages Mentions légales, Sources et Accueil
- Mise en place et maîtrise des routes React (test réussi sur la page Login)
- Restructuration complète du README et des dossiers front-end
- Découverte et prise en main de Leaflet-React pour la carte interactive
- Début des pages Character, Histoire et Races

Critères d'acceptation :

- Architecture React fonctionnelle et propre
- Routage opérationnel entre toutes les pages principales
- Premiers composants et pages visibles et stylisés
- Base solide pour l'intégration des futures fonctionnalités interactives

Sprint 4 - Semaine 10 : Connexions et Intégrations

Objectifs :

- Connexion complète des pages principales à la base de données (carte, personnages, races, histoires)
- Intégration des contenus et images dans la DB
- Mise en place du système de login (back/front)
- Début du module d'envoi d'images utilisateur et de la section créations artistiques

Tâches réalisées :

- Connexion avec la DB pour les pages Character, Histoire, Races et Map (marqueurs + polygones)
- Intégration des contenus bruts supplémentaires dans la base
- Finalisation et stylisation des cartes pour races/personnages/histoires
- Stylisation et interaction des marqueurs sur la carte interactive
- Implémentation du login et gestion partielle des cas d'erreurs
- Début de la section créations artistiques
- Envoi d'image fonctionnel pour un utilisateur connecté
- Page de publication stylisée et quasi finalisée
  Début du responsive design et finalisation du style des pages "Détail lieu"

Critères d'acceptation :

- Données correctement reliées à la DB
- Authentification opérationnelle
- Envoi d'images fonctionnel
- Début du responsive sur toutes les pages principales

Sprint 5 - Semaine 11 : Modération et Finalisation

Objectifs :

- Optimisation responsive (mobile/tablette)
- Corrections de bugs et améliorations UX
- Intégration des contenus bruts supplémentaires dans la base

Tâches assignées :

- Robin : implémentation d'une nouvelle table plus simple pour du texte supplémentaire, corrections UI/UX, documentation partie 4
- Thérèse-Marie : Responsive sur toutes les pages sauf map, documentation partie 4

Critères d'acceptation :

- Site responsive au maximum, MVP complet et testé

0.4. Tableau récapitulatif des sprints

Sprint	Fonctionnalités clés
Sprint 1 - Semaine 7	Configuration Flask + PostgreSQL, structure base de données (BIGSERIAL, GEOMETRY, SMALLINT), modèles User, Reviews, Image_Post, Race, Character, History, API User/Auth/Admin, façades et repositories, environnements dev Flask + PostgreSQL
Sprint 2 - Semaine 8	Tests API complets (GET, POST, PUT, DELETE), Search API (race, personnage, histoire), modèles Place, Marker, Région, API Map Marker et Map Région, QGIS géoréférencement, tests unitaires Pytest, script Postman automatisé, séparation schema.sql/data.sql
Sprint 3 - Semaine 9	Refonte front-end React + Vite (main.jsx, app.jsx), composants Header/Footer/Navigation, pages Accueil/Mentions légales/Sources/Login/Character/Histoire/Races, routage React, intégration Leaflet-React, restructuration README
Sprint 4 - Semaine 10	Connexion DB pages Character/Histoire/Races/Map (marqueurs + polygones), système login complet, envoi images utilisateurs, section créations artistiques, page publication stylisée, début responsive design, page détail lieu
Sprint 5 - Semaine 11	Optimisation responsive (mobile/tablette), table texte supplémentaire, corrections UI/UX, finalisation documentation partie 4, tests MVP complet

1. Exécution des Tâches de Développement

1.1. Méthodologie de développement

Organisation du travail :

- Présentiel : Lundi au vendredi, 8h30-19h30 maximum
- Distanciel : Week-ends et jours fériés selon disponibilités
- Stand-up quotidien : Chaque matin, première heure
- Décisions : Validation par les deux membres de l'équipe

1.2. Architecture du projet

Backend (Flask) & Frontend (React + Vite) :

```
Projet-Portfolio-/                  # Racine du projet (fullstack Flask + React)
|
├── app/                            # Backend - Application Flask
|   ├── __init__.py                 # Configuration Flask et initialisation
|   |
|   ├── api/                        # Routes de l'API REST
|   |   ├── __init__.py
|   |   └── V1/                     # Version 1 de l'API
|   |       ├── __init__.py
|   |       ├── api_admin.py        # Endpoints administration (gestion users)
|   |       ├── api_auth.py         # Endpoints authentification (login, JWT)
|   |       ├── api_characters.py   # Endpoints personnages
|   |       ├── api_histories.py    # Endpoints événements historiques
|   |       ├── api_image_post.py   # Endpoints upload images utilisateurs
|   |       ├── api_map_data.py     # Endpoints marqueurs et régions carte
|   |       ├── api_races.py        # Endpoints races (univers/lore)
|   |       ├── api_reviews.py      # Endpoints commentaires sur images
|   |       ├── api_search.py       # Endpoints barre de recherche
|   |       └── api_users.py        # Endpoints gestion utilisateurs
|   |
|   ├── database/                   # Scripts SQL
|   |   ├── schema.sql              # Création des tables PostgreSQL/PostGIS
|   |   └── data.sql                # Insertions initiales (données de base)
|   |
|   ├── models/                     # Modèles ORM (représentation des tables)
|   |   ├── __init__.py
|   |   ├── basemodel.py            # Classe mère (id, created_at, updated_at)
|   |   ├── character.py            # Modèle Personnage
|   |   ├── history.py              # Modèle Événement historique
|   |   ├── image_post.py           # Modèle Image postée par utilisateur
|   |   ├── map_marker.py           # Modèle Point/Marqueur sur carte (PostGIS)
|   |   ├── map_region.py           # Modèle Région/Polygone sur carte (PostGIS)
|   |   ├── place_map.py            # Modèle Lieu/Carte
|   |   ├── race.py                 # Modèle Race (Elfes, Nains, etc.)
|   |   ├── review.py               # Modèle Commentaire sur image
|   |   └── user.py                 # Modèle Utilisateur
|   |
|   ├── persistence/                # Repositories (accès base de données CRUD)
|   |   ├── __init__.py
|   |   ├── image_post_repository.py  # CRUD pour ImagePost
|   |   ├── repository.py           # Repository générique (template CRUD)
|   |   ├── review_repository.py    # CRUD pour Review
|   |   └── user_repository.py      # CRUD pour User
|   |
|   └── services/                   # Couche métier (logique business)
|       ├── __init__.py
|       ├── facade.py               # Façade pour entités liées à l'utilisateur
|       └── facade2.py              # Façade pour entités non liées à l'utilisateur
```

```
├── frontend/                          # Frontend - Application React
│   ├── src/                           # Code source React
│   │   ├── assets/                    # Fichiers statiques
│   │   │   ├── fonts/                 # Polices personnalisées
│   │   │   └── images/                # Images, icônes, illustrations
│   │   │
│   │   ├── components/                # Composants React réutilisables
│   │   │   ├── Header.jsx             # En-tête du site
│   │   │   ├── Footer.jsx             # Pied de page
│   │   │   ├── Navigation.jsx         # Menu de navigation
│   │   │   └── ...
│   │   │
│   │   ├── pages/                     # Pages principales de l'application
│   │   │   ├── HomePage.jsx           # Page d'accueil
│   │   │   ├── MapPage.jsx            # Carte interactive Leaflet
│   │   │   ├── CharacterPage.jsx      # Liste/détails personnages
│   │   │   ├── RacePage.jsx           # Liste/détails races
│   │   │   ├── HistoryPage.jsx        # Liste/détails événements
│   │   │   ├── GalleryPage.jsx        # Galerie créations artistiques
│   │   │   ├── LoginPage.jsx          # Connexion/Inscription
│   │   │   ├── LegalPage.jsx          # Mentions légales
│   │   │   └── SourcesPage.jsx        # Crédits et sources
│   │   │
│   │   ├── services/                  # Services frontend (API calls)
│   │   │   └── api.js                 # Configuration axios/fetch
│   │   │
│   │   ├── styles/                    # Styles CSS
│   │   │   ├── global.css             # Styles globaux
│   │   │   └── ...
│   │   │
│   │   ├── App.jsx                    # Composant racine React
│   │   └── main.jsx                   # Point d'entrée React (mount dans DOM)
│   │
│   ├── public/                        # Assets publics statiques
│   ├── index.html                     # HTML principal (injection React)
│   ├── vite.config.js                 # Configuration Vite
│   ├── package.json                   # Dépendances npm
│   └── eslint.config.js               # Configuration ESLint
├── tests/                             # Tests automatisés
│   ├── unit/                          # Tests unitaires (pytest)
│   │   ├── __init__.py
│   │   ├── test_api.py                # Tests des endpoints API
│   │   ├── test_facade1.py            # Tests façade 1
│   │   ├── test_facade2.py            # Tests façade 2
│   │   ├── test_model_statique.py     # Tests modèles non liés user
│   │   ├── test_model_dynamique.py    # Tests modèles liés user
│   │   └── test_routes.py             # Tests supplémentaires routes
│   │
│   ├── postman/                       # Tests API Postman
│   │   └── script_postman.json        # Collection complète requêtes
│   │
│   ├── __init__.py
│   └── conftest.py                    # Configuration pytest (fixtures)
│
├── .env                               # Variables d'environnement (ignoré git)
├── .env.test                          # Variables pour environnement de test
├── .gitignore                         # Fichiers ignorés par git
├── .prettierrc                        # Configuration Prettier
├── config.py                          # Configuration Flask (DB, JWT, etc.)
├── run.py                             # Point d'entrée application Flask
├── requirements.txt                   # Dépendances Python
└── README.md                          # Documentation projet
```

1.3. Standards de codage

Back-end (Flask/Python) :

- Respect des conventions PEP 8
- Docstrings pour toutes les fonctions et classes
- Validation des données en entrée
- Gestion appropriée des erreurs et logging

Front-end (React) :

- Composants fonctionnels avec hooks (useState, useEffect)
- Nommage en PascalCase pour les composants
- Props explicites et documentation des composants

1.3. Processus de revue de code

Chaque fonctionnalité fait l'objet d'une pull request avant intégration. Le processus de revue comprend :

- Vérification de la qualité du code et respect des standards
- Tests de la fonctionnalité
- Suggestions d'amélioration ou d'optimisation
- Validation par l'autre membre avant merge

2. Surveillance de la Progression et Ajustements

2.1. Stand-ups quotidiens

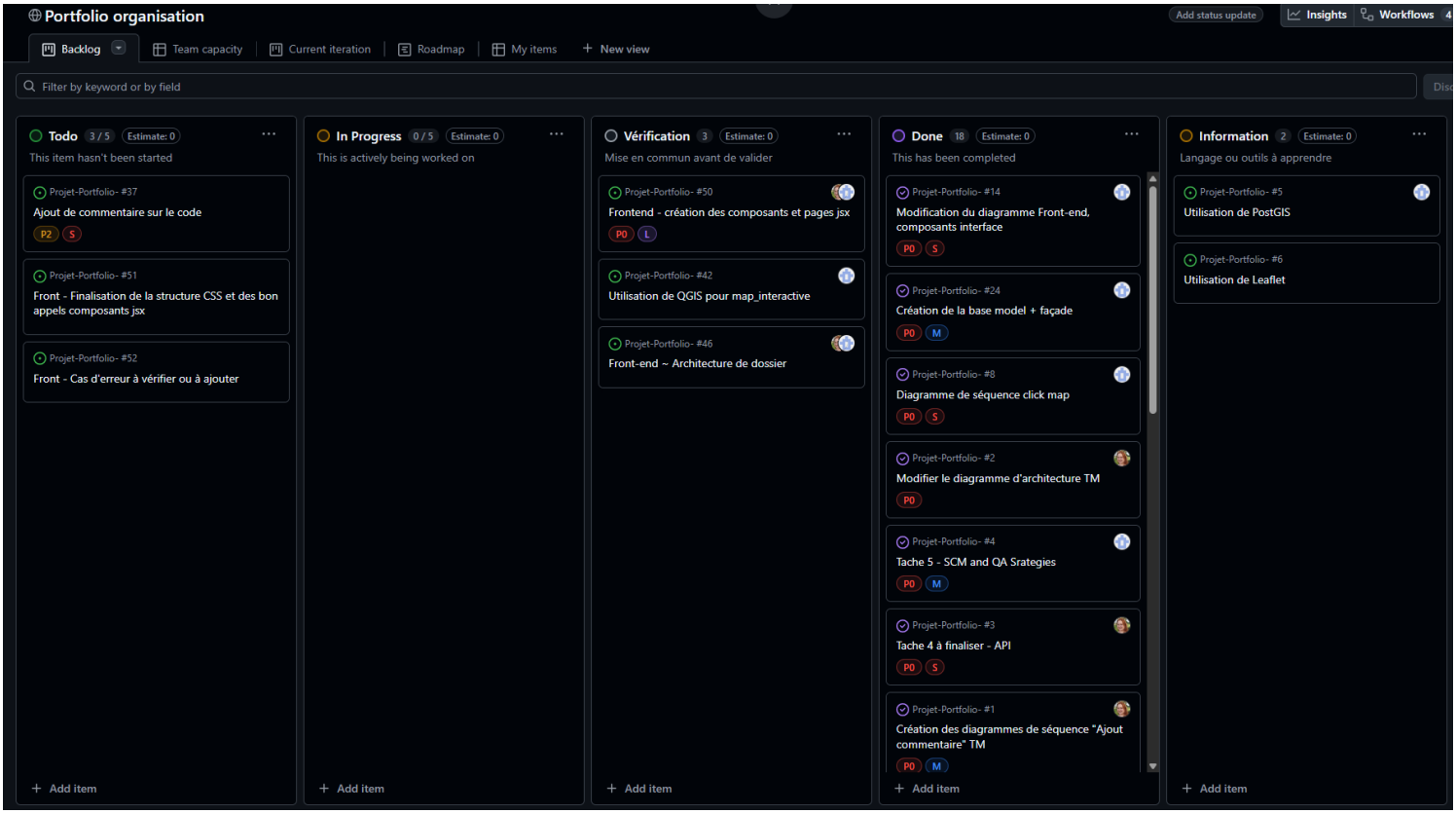
Format des stand-ups :

- Stand-up le midi d'une dizaine de minutes avec tous les élèves sur le déroulement de la journée et les obstacles rencontré.
- Que vais-je faire aujourd'hui ?
- Quels obstacles rencontré-je ?

2.2. Outils de gestion de projet

GitHub Projects :

- Kanban board avec colonnes : To Do, In Progress, Vérification, Done, Information, None
- Issues liées aux user stories et tâches
- Labels pour meilleur compréhension (documentation, dev, information, bug)



Discord :

- Communication quotidienne
- Partage d'informations
- Partage d'écran pour pair programming si travail en distanciel

2.3. Gestion des obstacles

Process de résolution des blocages :

- Identification immédiate lors du stand-up
- Pair programming sur les difficultés techniques
- Recherche documentaire et tutoriels ciblés
- Utilisation de l'intelligence artificielle
- Prise de renseignement avec d'autre élèves

2.4. Métriques de suivi

- **Vélocité** : Vitesse de sprint pour la partie développement : 5 semaines.
- **Sprint 1 (semaine 1-2) :** 14 tâches complétées pour la mise en place du backend
- **Sprint 2 (semaine 3-4) :** 9 tâches complétées pour la mise en place du frontend
- **Sprint 3 (semaine 5) :** 6 tâches complétées pour la finalisation du frontend et correctif du code (doublon / syntaxe).
- **Pourcentage :** 90% des features prévues ajoutés pour 10% finalement non implémentés car fonctionnalité trop compliquée à implémenté à notre niveau ou manque de temps.

3. Revues de Sprint et Rétrospectives

3.1. Sprint Review

À la fin de chaque sprint (lundi matin) :

- Démonstration des fonctionnalités terminées
- Validation par rapport aux critères d'acceptation
- Discussion sur les fonctionnalités non terminées

3.2. Sprint Rétrospective

Template de rétrospective

- Qu'est-ce qui s'est bien passé pendant le sprint ?
  - Le travail d’équipe, l’investissement en travail/temps sur le projet

- Quels défis avons-nous rencontrés ?
  - Des bugs par dizaine, des conflits de merge et des features compliqué à implémenter.

- Quels changements pouvons-nous apporter pour améliorer le prochain sprint ?
  - S’imposer des horaires de fin de travail raisonnable max 18h30
  - Veiller à push nos fichiers sans conflit de merge

- Actions concrètes à mettre en place
  - Mettre une alarme de fin de travail
  - Attention à ne pas modifier les mêmes fichiers pour conflit.

- Responsable et échéance pour chaque action
  - Les deux sont responsables

4. Intégration Finale et Tests QA

4.1. Stratégie de tests

Tests unitaires (Pytest) :

- tests/unit/test_model_statique.py : Tests des modèles non liés aux utilisateurs (character, race, history, place_map)
- tests/unit/test_model_dynamique.py : Tests des modèles liés aux utilisateurs (user, image_post, review)
- tests/unit/test_facade1.py : Tests de la facade pour utilisateurs
- tests/unit/test_facade2.py : Tests de la facade pour données statiques
- Couverture de code ciblée : > 70%

Tests d'intégration (Postman/Newman) :

- tests/postman/script_postman.json : Collection complète de tests API
- Tests des endpoints : /api/v1/auth, /api/v1/characters, /api/v1/races, /api/v1/histories, /api/v1/images, /api/v1/reviews, /api/v1/map-data, /api/v1/admin, /api/v1/search
- Vérification des codes de statut HTTP (200, 201, 400, 401, 403, 404)
- Validation du format des réponses JSON/GeoJSON
- Tests d'authentification JWT et permissions par rôle

Tests end-to-end (Manuels) :

- Parcours utilisateur complets
- Tests responsive (desktop, tablette, mobile)

4.2. Checklist de validation

Critère	Statut	Priorité
Responsive desktop/tablette/mobile sur certaines page	*À valider*	**Majeur**
Gestion d'erreur à vérifier	*À valider*	**Mineur**
Panneau de configuration pour tout type d'utilisateur	*À valider*	**Ajout à l'avenir**
Filtres par type de contenu	*À valider*	**Ajout à l'avenir**
Fonction de recherche	*À valider*	**Ajout à l'avenir**
Marqueur de région plus visible	*À valider*	**Ajout à l'avenir**
Données en DB bien appelé pour les lieux / races / characters …	*Valider*	**Terminé**
Marqueur bien ajouté pour les régions et autres types de villes	*Valider*	**Terminé**
Route API fonctionnel et tester sur postman	*Valider*	**Terminé**
Plus de test à rajouter	*À valider*	**Mineur**
Modèle et façade fonctionnel	*À valider*	**Terminé**
Post-image réussi avec un utilisateur connecté	*À valider*	**Terminé**
Commentaire possible sur une image par un utilisateur connecté	*À valider*	**Terminé**
CSS terminé à expérience utilisateur pris en compte	*À valider*	**Optionnel**
Structure cohérente des fichiers et bonne connaissance de l'architecture	*Valider*	**Terminé**

4.3. Correction des bugs critiques

Priorisation des bugs :

P0 - Critique (correction immédiate) :

- Empêche l'utilisation de fonctionnalités essentielles, crash de l'application

P1 - Majeur (correction dans les 24h) :

- Impacte significativement l'expérience utilisateur

P2 - Mineur (correction avant release) :

- Problèmes cosmétiques, bugs mineurs n'empêchant pas l'utilisation

P3 - Optionnel (peut être reporté) :

- Améliorations mineures, suggestions

5. Livrables et Documentation

5.1. Documentation technique

README.md du projet :

- Description du projet : Wiki Interactif - Univers du Seigneur des Anneaux
- Technologies utilisées : React + Vite, Flask, PostgreSQL + PostGIS, Leaflet
- Instructions d'installation avec configuration de .env
- Structure du projet avec arborescence complète
- Guide de contribution et conventions Git

Documentation API :

- Liste complète des endpoints par module (auth, characters, races, histories, images, reviews, map-data, admin, search, users)
- Paramètres d'entrée et formats de réponse JSON/GeoJSON
- Codes de statut HTTP et gestion des erreurs
- Exemples de requêtes curl et réponses attendues

Documentation technique supplémentaire :

- app/database/schema.sql : Structure complète de la base de données
- app/database/data.sql : Données d'insertion initiales
- app/database/data_place.sql : Données d’insertion conséquente de description pour les places
- tests/conftest.py : Configuration des fixtures pytest
- .env.test : Configuration pour environnement de test