Portfolio - Task 4

Développement and Execution of MVP

Interactive map of the lord of the rings & Artistic creations LOTR

0. Planification and Sprints Definition

0.1. Overview of development

The development of the MVP is organised into five sprints of one week each, covering weeks 7 to 11 of the project. This organisation allows for iterative progress and regular adjustments based on team feedback.

0.2. Priorisation MoSCoW

Feature prioritisation follows the MoSCoW framework established in the technical documentation:

- Must Have: Interactive map, detailed information sheets, authentication system, publication of creations, moderation
- Should Have: Search and filters, user management
- Could Have: Real-time messaging, editing/deleting creations
- Won't Have: E-commerce, interactive family tree

0.3. Detailed sprint planning

Sprint 1 - Week 7: Setting Up the Back End and Database

Objectives :

- Define and structure the project database
- Set up the initial APIs and interfaces (User, Auth, Admin, Reviews, etc.)
- Lay the foundations for the back end with Flask and PostgreSQL
- Perform initial functional tests using Postman

Tasks completed :

- Creation of the initial project structure (feature/structure branch)
- Construction of the database based on the design document, with technical adjustments:
    - Switch to BIGSERIAL for IDs (PostgreSQL auto-increment)
    - Use SMALLINT for start_year and death_year
    - Conversion of `x` and `y` to `GEOMETRY(POINT, 0)` in `map_marker`
    - Addition of a `relation_type` table to manage links between characters and stories
      • Insertion of initial data (users, races, relationship types)
      • Configuration of the Flask application and creation of the first classes (User)
      • Implementation of the reviews and image_post models
      • Implementation of the first facades, repositories and routes in __init__.py
      • Development and testing of the User, Auth and Admin APIs with Postman (GET, POST, PUT validated)
      • Adding APIs and facades for Race, Character, History, and Reviews
      • Refactoring with facade2 to centralise entity management

• Modifying the image_post model to store images directly in the database (BYTEA + image_mime_type)
• Setting up separate local Flask + PostgreSQL environments for Robin and Thérèse-Marie

Acceptance criteria:

- Stable, consistent database connected to Flask
- Main APIs functional and tested with Postman
- Authentication operational (token retrieved)
- Independent development environments configured and ready for subsequent iterations

Sprint 2 - Week 8: Back-End Testing and API Structuring

Objectives :

- Finalise and test all API routes (GET, POST, PUT, DELETE)
- Properly structure models, facades and routes for all main entities
- Set up a test database and strengthen back-end reliability
- • Initial learning about React and exploration of mapping tools

Tasks completed :

- Validation of Postman API tests for all main routes (GET_id, GET_ALL, POST, PUT, DELETE)
- Implementation and testing of the Search API (race, character, history)
- Addition of models and facades for Place, Marker and Region
- Map Marker and Map Region API tests validated
- Creation and separation of schema.sql (tables) and data.sql (inserts) files
- Improvement of API Reviews routes (filtering by user or image)
- Correction of the Image_Post model and update of the local database
- Use of QGIS for georeferencing and visualisation of points/polygons
- Implementation of comprehensive unit tests (models, facades, API) via Pytest on a dedicated test database
- Creation of a Postman script to automate API tests
- Start of React training in preparation for front-end redesign

Acceptance criteria:

- All API routes validated and tested
- Stable and structured database
- Functional test environment
- Solid foundations for transition to front-end development

Sprint 3 - Week 9: Front-End Redesign and React Integration

Objectives :

- Complete redesign of the front end with React
- Clear structuring of the project architecture
- Implementation of the first components and React routes
- Familiarisation with Leaflet for the future interactive map

Tasks completed:

- Removal of obsolete files and folders
- Installation and configuration of React via Vite (main.jsx and app.jsx files)
- Creation and initial integration of Header, Footer and Navigation components
- Addition and styling of Legal Notices, Sources and Home pages
- Implementation and mastery of React routes (test successful on the Login page)
- Complete restructuring of the README and front-end folders
- Discovery and familiarisation with Leaflet-React for the interactive map
- Start of the Character, History and Races pages

Acceptance criteria:

- Functional and clean React architecture
- Operational routing between all main pages
- First components and pages visible and styled
- Solid foundation for integrating future interactive features

Sprint 4 - Week 10: Connections and Integrations

Objectives:

- Complete connection of main pages to the database (map, characters, races, stories)
- Integration of content and images into the DB
- Implementation of the login system (back/front)
- Start of the user image submission module and the artistic creations section

Tasks completed:

- Connection to the database for the Character, History, Races and Map pages (markers + polygons)
- Integration of additional raw content into the database
- Finalisation and styling of maps for races/characters/stories
- Styling and interaction of markers on the interactive map
- Implementation of login and partial error handling
- Start of the artistic creations section
- Image sending functional for a logged-in user
- Stylised and almost finalised publication page
- Start of responsive design and finalisation of the style of the ‘Location details’ pages

Acceptance criteria:

- Data correctly linked to the database
- Authentication operational
- Image sending functional
- Responsive design implemented on all main pages

Sprint 5 - Week 11: Moderation and Finalisation

Objectives:

- Responsive optimisation (mobile/tablet)
- Bug fixes and UX improvements
- Integration of additional raw content into the database

Assigned tasks:

- Robin: implementation of a new, simpler table for additional text, UI/UX corrections, documentation part 4
- Thérèse-Marie: Responsive on all pages except map, documentation part 4

Acceptance criteria:

- Fully responsive website, complete and tested MVP

0.4. Tableau récapitulatif des sprints

Sprint	Key features
Sprint 1 - Week 7	Flask + PostgreSQL configuration, database structure (BIGSERIAL, GEOMETRY, SMALLINT), User, Reviews, Image_Post, Race, Character, History models, User/Auth/Admin API, facades and repositories, Flask + PostgreSQL dev environments
Sprint 2 - Week 8	Comprehensive API testing (GET, POST, PUT, DELETE), Search API (race, character, story), Place, Marker, Region, Map Marker and Map Region templates, QGIS georeferencing, Pytest unit testing, automated Postman script, schema.sql/data.sql separation.
Sprint 3 - Week 9	React + Vite front-end redesign (main.jsx, app.jsx), Header/Footer/Navigation components, Home/Legal notices/Sources/Login/Character/History/Races pages, React routing, Leaflet-React integration, README restructuring
Sprint 4 - Week 10	DB connection to Character/History/Races/Map pages (markers + polygons), complete login system, user image uploads, artistic creations section, stylised publication page, responsive design start, location details page
Sprint 5 - Week 11	Responsive optimisation (mobile/tablet), additional text table, UI/UX corrections, finalisation of documentation part 4, complete MVP testing

1. Execution of Development Tasks

1.1. Development methodology

Organisation of work :

- In person: Monday to Friday, 8:30 a.m. to 7:30 p.m. maximum
- Remote: Weekends and public holidays depending on availability
- Daily stand-up meeting: Every morning, first thing
- Decisions: Approval by both team members

1.2. Project architecture

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. Coding standards

Back-end (Flask/Python):

- Compliance with PEP 8 conventions
- Docstrings for all functions and classes
- Input data validation
- Appropriate error handling and logging

Front-end (React):

- Functional components with hooks (useState, useEffect)
- PascalCase naming for components
- Explicit props and component documentation

1.3. Code review process

Each feature is subject to a pull request before integration. The review process includes :

- Verification of code quality and compliance with standards
- Feature testing
- Suggestions for improvement or optimisation
- Validation by the other member before merging

2. Progress Monitoring and Adjustments

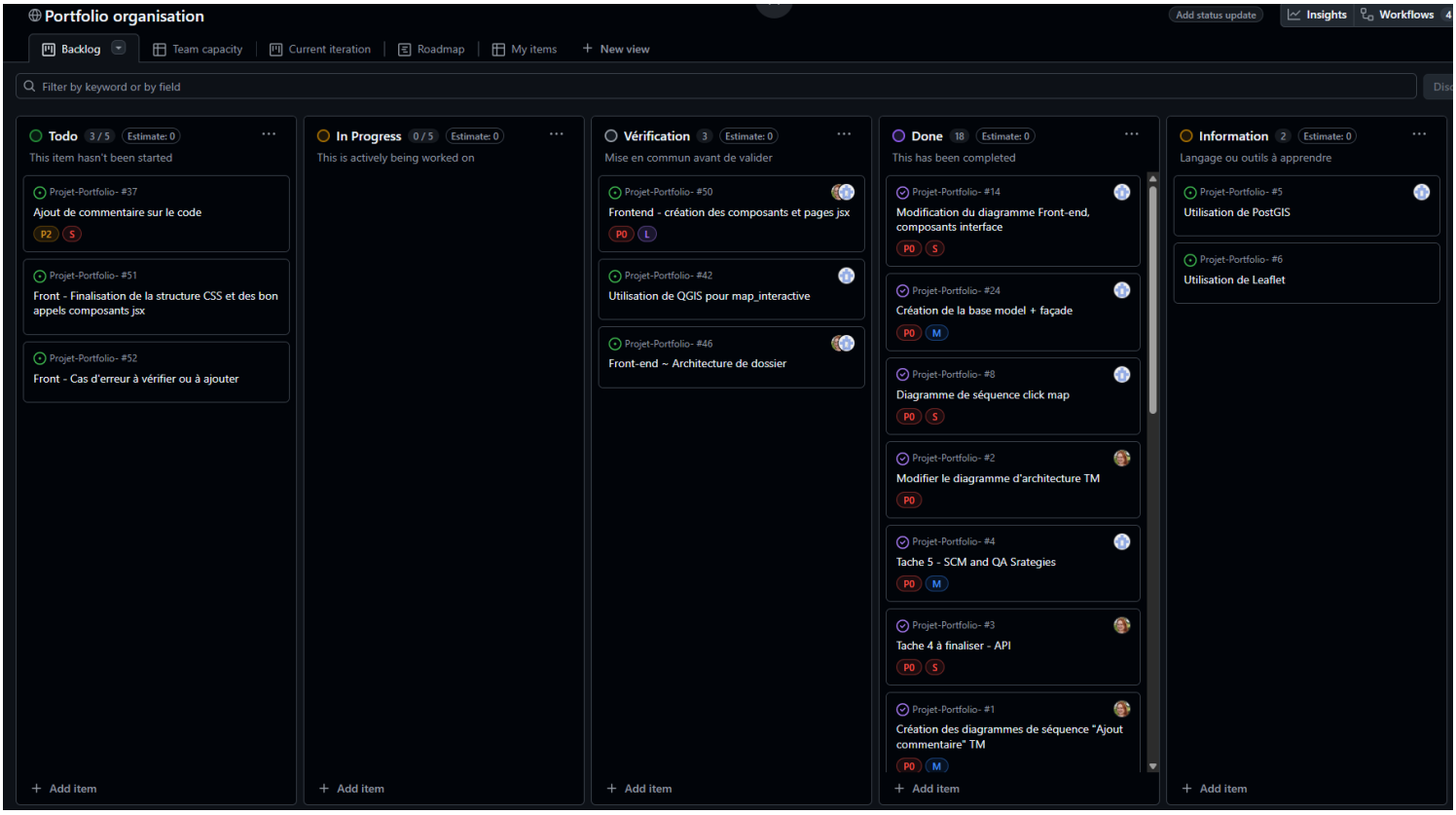
2.1. Daily stand-ups

Stand-ups format :

- Ten-minute stand-up meeting at lunchtime with all students to discuss the day's schedule and any obstacles encountered.
- What am I going to do today?
- What obstacles will I encounter?2.2. Outils de gestion de projet

GitHub Projects :

- Kanban board with columns: To Do, In Progress, Verification, Done, Information, None
- Issues related to user stories and tasks
- Labels for better understanding (documentation, dev, information, bug)



Discord:

- Daily communication
- Information sharing
- Screen sharing for pair programming when working remotely

2.3. Obstacle management

Process for resolving blockages:

- Immediate identification during stand-up meetings
- Pair programming for technical difficulties
- Targeted documentary research and tutorials
- Use of artificial intelligence
- Seeking information from other students

2.4. Tracking metrics

- **Vélocity** : Sprint speed for the development phase: 5 weeks.
- **Sprint 1 (weeks 1-2) :** 14 tasks completed for backend implementation
- **Sprint 2 (weeks 3-4) :** 9 tasks completed for frontend implementation
- **Sprint 3 (weeks 5) :** 6 tasks completed for finalising the frontend and code fixes (duplicates/syntax).
- **Percentage :** 90% of planned features added, with 10% ultimately not implemented due to functionality being too complicated to implement at our level or lack of time.

3. Sprint Reviews and Retrospectives

3.1. Sprint Review

At the end of each sprint (Monday morning):

- Demonstration of completed features
- Validation against acceptance criteria
- Discussion of unfinished features

3.2. Sprint Retrospective

Retrospective template

- What went well during the sprint?
  - Teamwork, investment in work/time on the project

- What challenges did we encounter?
  - Dozens of bugs, merge conflicts and features that were complicated to implement.

- What changes can we make to improve the next sprint?
  - Set reasonable working hours, ending no later than 6:30 p.m.
  - Ensure that our files are pushed without merge conflicts

- Concrete actions to implement
  - Set an end-of-work alarm
  - Be careful not to modify the same files to avoid conflicts.

- Responsible person and deadline for each action
  - Both are responsible

4. Final Integration and QA Testing

4.1. Tests strategies

Unit tests (Pytest) :

- tests/unit/test_model_static.py: Tests for models not linked to users (character, race, history, place_map)
- tests/unit/test_model_dynamique.py: Tests for user-related models (user, image_post, review)
- tests/unit/test_facade1.py: Tests for the user interface
- tests/unit/test_facade2.py: Tests for the static data interface
- Targeted code coverage: > 70%

Integration tests (Postman/Newman) :

- tests/postman/script_postman.json: Complete collection of API tests
- Endpoint tests: /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
- Verification of HTTP status codes (200, 201, 400, 401, 403, 404)
- Validation of JSON/GeoJSON response format
- JWT authentication and role-based permissions tests

Tests end-to-end (Manuels) :

- Complete user journeys
- Responsive testing (desktop, tablet, mobile)

4.2. Checklist of validation

Critère	Statut	Priority
Responsive desktop/tablet/mobile on certain pages	*To be validated*	**Major**
Error handling to be verified	*To be validated*	**Minor**
Control panel for all types of users	*To be validated*	**Future addition**
Filters by content type	*To be validated*	**Future addition**
Search function	*To be validated*	**Future addition**
More visible region marker	*To be validated*	**Future addition**
Data in DB correctly named for locations/races/characters, etc.	*Validate*	**Completed**
Marker added for regions and other types of cities	*Validate*	**Completed**
API route functional and tested on Postman	*Validate*	**Completed**
More tests to be added	*To be validated*	**Minor**
Functional model and front end	*To be validated*	**Completed**
Post image successful with a logged-in user	*To be validated*	**Completed**
Comments possible on an image by a logged-in user	*To be validated*	**Completed**
CSS completed with user experience taken into account	*To be validated*	**Optional**
Consistent file structure and good knowledge of architecture	*Validate*	**Completed**

4.3. Critical bug fixes

Bugs priorisation :

P0 - Critical (immediate fix):

- Prevents the use of essential features, application crash

P1 - Major (fix within 24h):

- Significantly impacts the user experience

P2 - Minor (fix before release) :

- Cosmetic issues, minor bugs that do not prevent use

P3 - Optional (can be postponed) :

- Minor improvements, suggestions

5. Deliverables and Documentation

5.1. Technic documentation

Project README.md :

- Project description: Interactive Wiki - The Lord of the Rings Universe
- Technologies used: React + Vite, Flask, PostgreSQL + PostGIS, Leaflet
- Installation instructions with .env configuration
- Project structure with complete tree view
- Contribution guide and Git conventions

Documentation API :

- Complete list of endpoints by module (auth, characters, races, histories, images, reviews, map-data, admin, search, users)
- Input parameters and JSON/GeoJSON response formats
- HTTP status codes and error handling
- Examples of curl requests and expected responses

Additional technical documentation:

- app/database/schema.sql: Complete database structure
- app/database/data.sql: Initial insertion data
- app/database/data_place.sql: Subsequent insertion data for place descriptions
- tests/conftest.py: Pytest fixture configuration
- .env.test: Configuration for test environment