Portfolio - Partie **3**

1. Définir les récits d'utilisateurs et les maquettes

1.1. Récits utilisateurs

1.1.1. Must Have (indispensable)

*En tant qu'**invité**, je souhaite **explorer la carte interactive** afin de **découvrir les lieux**.*

*En tant qu'**invité**, je souhaite consulter les fiches détaillées d'un lieu ou personnage afin de m'informer sur l'univers.*

*En tant qu'**utilisateur inscrit**, je souhaite **créer un compte et me connecter** afin d'**accéder aux fonctionnalités de contribution**.*

*En tant qu'**utilisateur inscrit**, je souhaite **publier une création artistique (image + description)** afin de **partager mon oeuvre avec la communauté**.*

*En tant qu'**administrateur**, je souhaite **modérer les publications** afin de **garantir la qualité et le respect des règles.***

1.1.2. Should Have (important mais pas vital pour le MVP)

*En tant qu'**invité**, je souhaite **rechercher un contenu par filtre (lieux, personnages, créations)** afin d'**accéder rapidement à l'information souhaitée.***

*En tant qu'**administrateur**, je souhaite **gérer les utilisateurs (bloquer/supprimer)** afin de **maintenir un espace sécurisé**.*

1.1.3. Could Have (utile mais optionnel)

*En tant qu'**utilisateur inscrit**, je souhaite **utiliser une messagerie en temps réel** afin de **discuter avec la communauté**.*

*En tant qu'**utilisateur inscrit**, je souhaite **éditer ou supprimer mes créations** afin de **garder le contrôle sur mon contenu.***

1.1.4. Won't Have (pas prévu pour ce MVP)

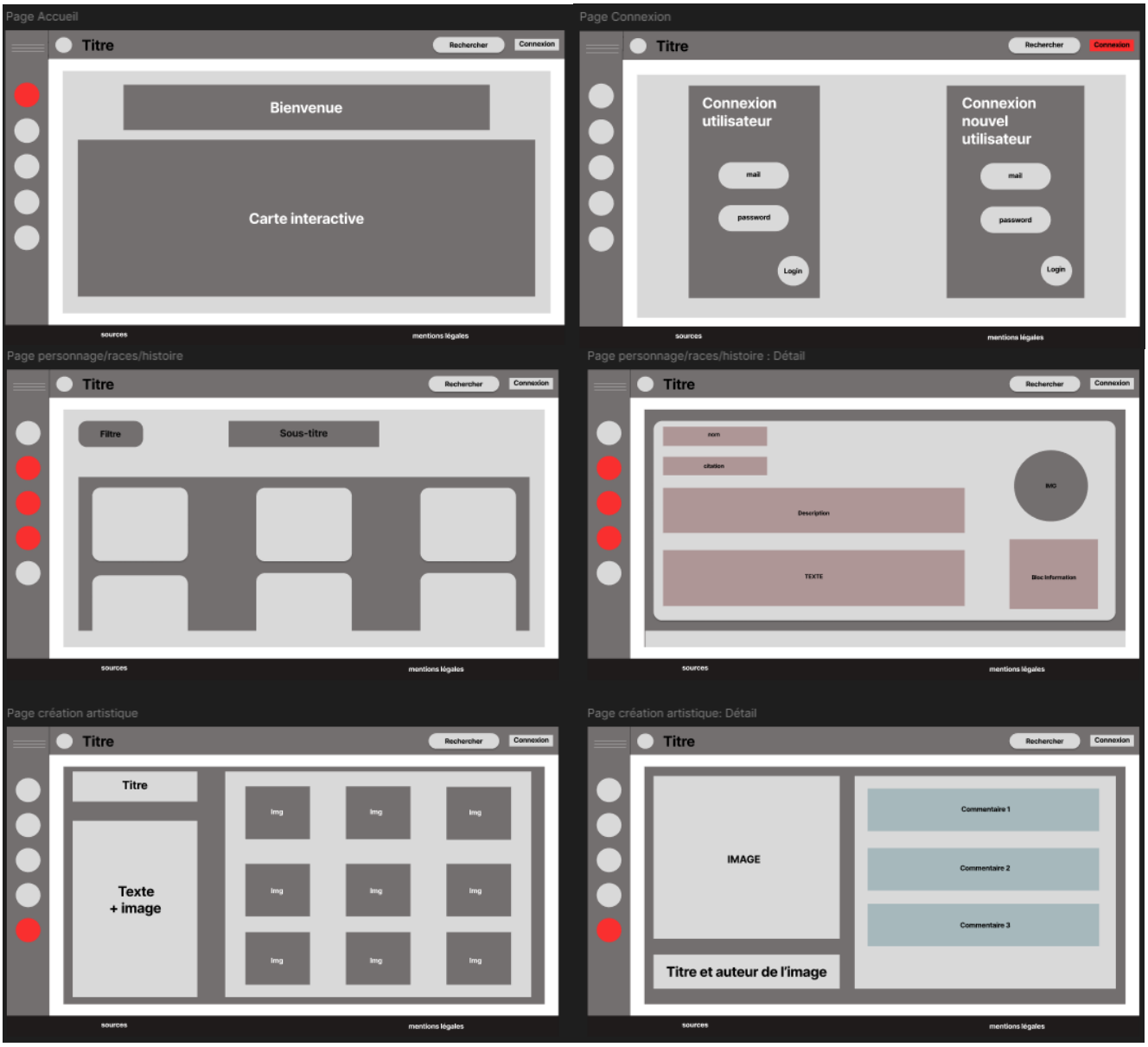
*En tant qu'**utilisateur**, je souhaite **acheter des créations ou objets dérivés** afin de **les posséder.***

*En tant qu'**utilisateur**, je souhaite **naviguer dans un arbre généalogique interactif** afin de **voir les liens entre les personnages**.*

1.2. Synthèse MoSCoW

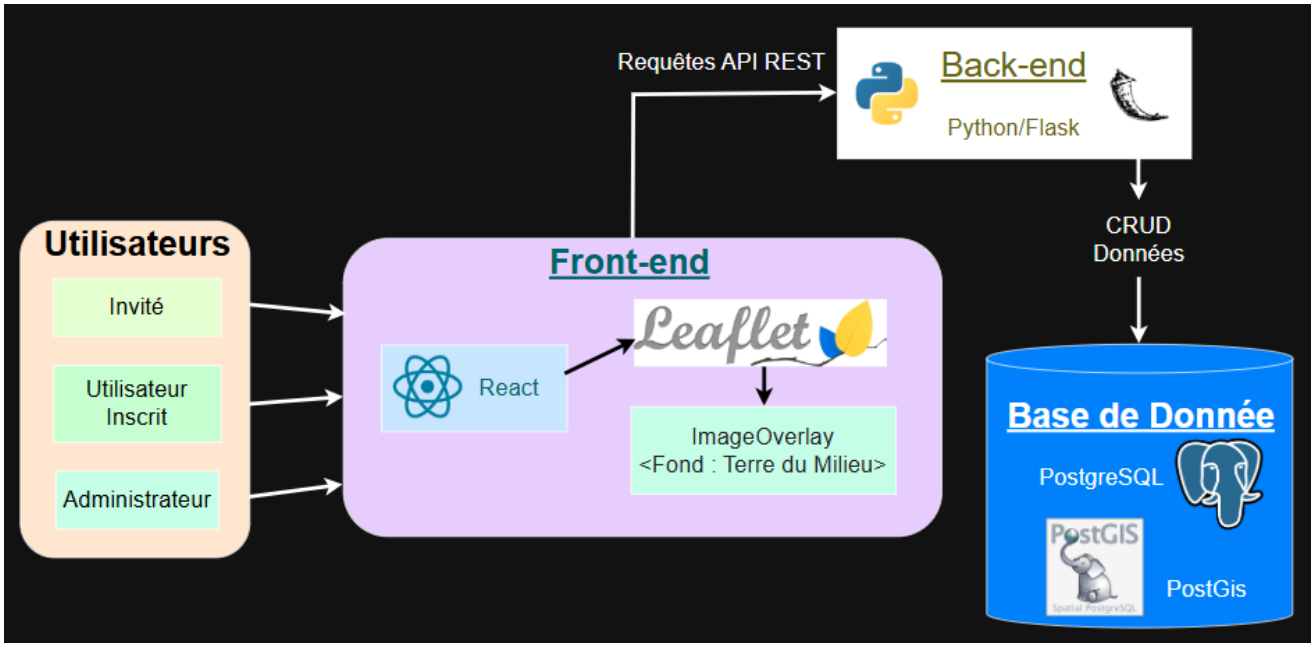
Priorité	Fonctionnalités retenues
Must Have **(Indispensable)**	- Carte interactive pour découvrir les lieux - Fiches détaillées des lieux/personnages - Création de compte et connexion - Publication de créations artistiques (image + description) - Modération des publications par l'administrateur
Should Have **(Important mais non vital)**	- Recherche et filtres (lieux, personnages, créations) - Gestion des utilisateurs par l'administrateur (bloquer/supprimer)
Could Have **(Optionnel)**	- Messagerie en temps réel entre utilisateurs inscrits - Édition/suppression de ses propres créations
Won't Have **(Hors périmètre du MVP)**	- Achat de créations ou objets dérivés (e-commerce) - Arbre généalogique interactif des personnages

1.3. Wireframe



2. Architecture du système de conception

2.1. Diagramme d'Architecture Haut Niveau



2.2. Explications

2.2.1. Front-end (React + Leaflet)

- **Langage :** JavaScript (avec JSX pour combiner HTML et JavaScript) + CSS pour le style.
- **Technologies principales :**
  - **React** : structure l'interface utilisateur (UI) et gère l'état de l'application.
  - **Leaflet** : bibliothèque JavaScript de cartographie interactive.
  - **ImageOverlay** : sert d'image de fond (ex. la carte de la Terre du Milieu).
- **Composants clés :**
  - `App` (point d'entrée React)
  - `Header` (entête)
  - `Sidebar` (navigation entre les sections)
  - `Map` (intègre Leaflet pour afficher zones et lieux au format GeoJSON)
  - `InfoGroupe` (affiche personnages, histoires, créatures)
  - `InfoSolo` (affiche la fiche d'un personnage/créature)
  - `ArtWorkGallery` (galerie des créations artistiques)
  - `ArtWorkSolo` (affiche une création avec commentaire)

2.2.2. Back-end (Flask API en Python)

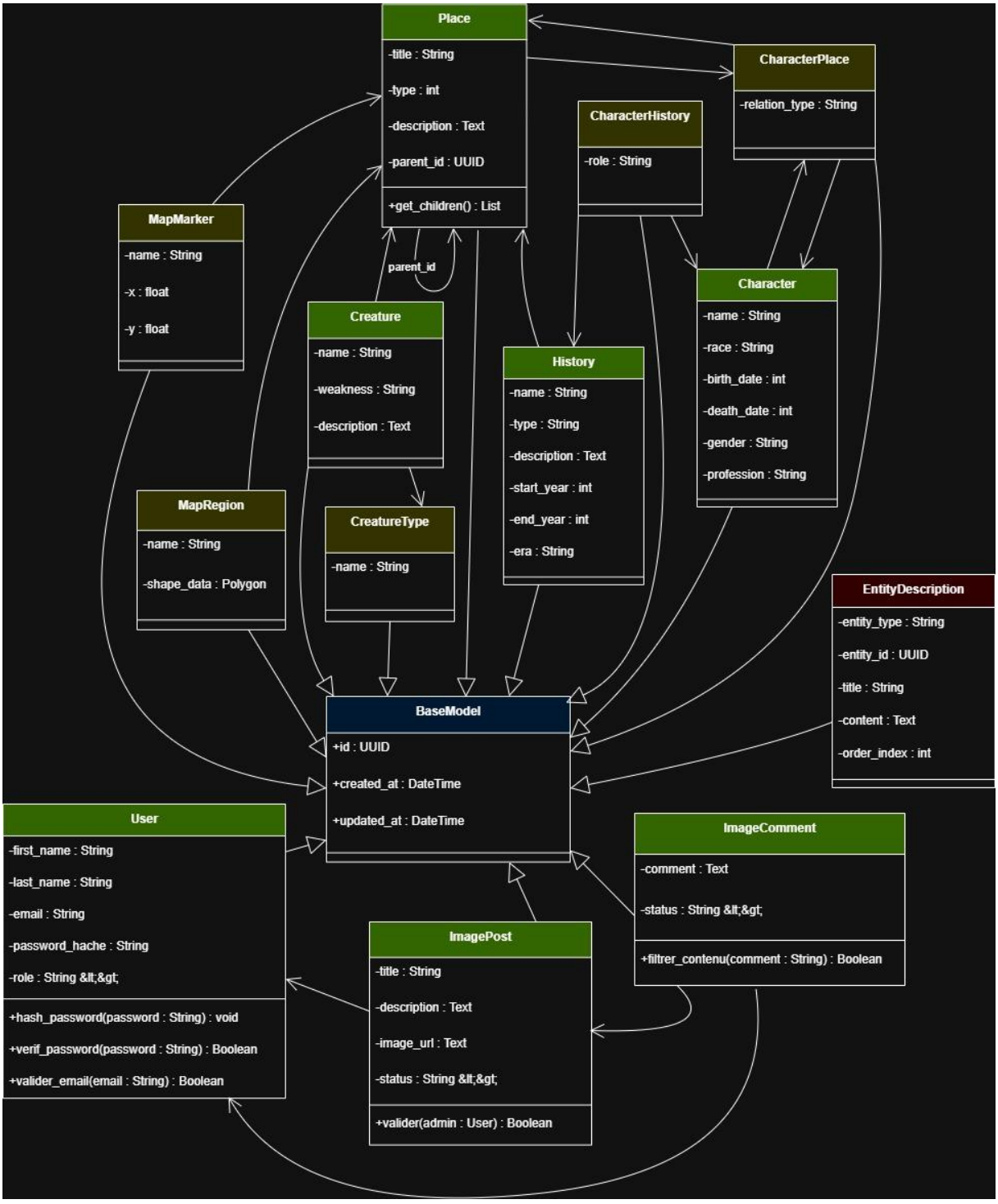
- **Langage :** Python.
- **Framework :** Flask (micro-framework web).
- **Rôle :**
    - Sert de **façade unique** entre le front-end et la base de données.
    - Fournit une **API REST** (endpoints `/characters`, `/stories`, `/regions`, `/locations`, `/artworkgallery`, `/artworksolo` ...).
    - Retourne les données au format **JSON** (contenu textuel) ou **GeoJSON** (données géographiques).
    - Contient la **logique métier** : validation, contrôleurs CRUD (Create, Read, Update, Delete), règles d'application.

2.2.3. Base de données (PostgreSQL + PostGIS)

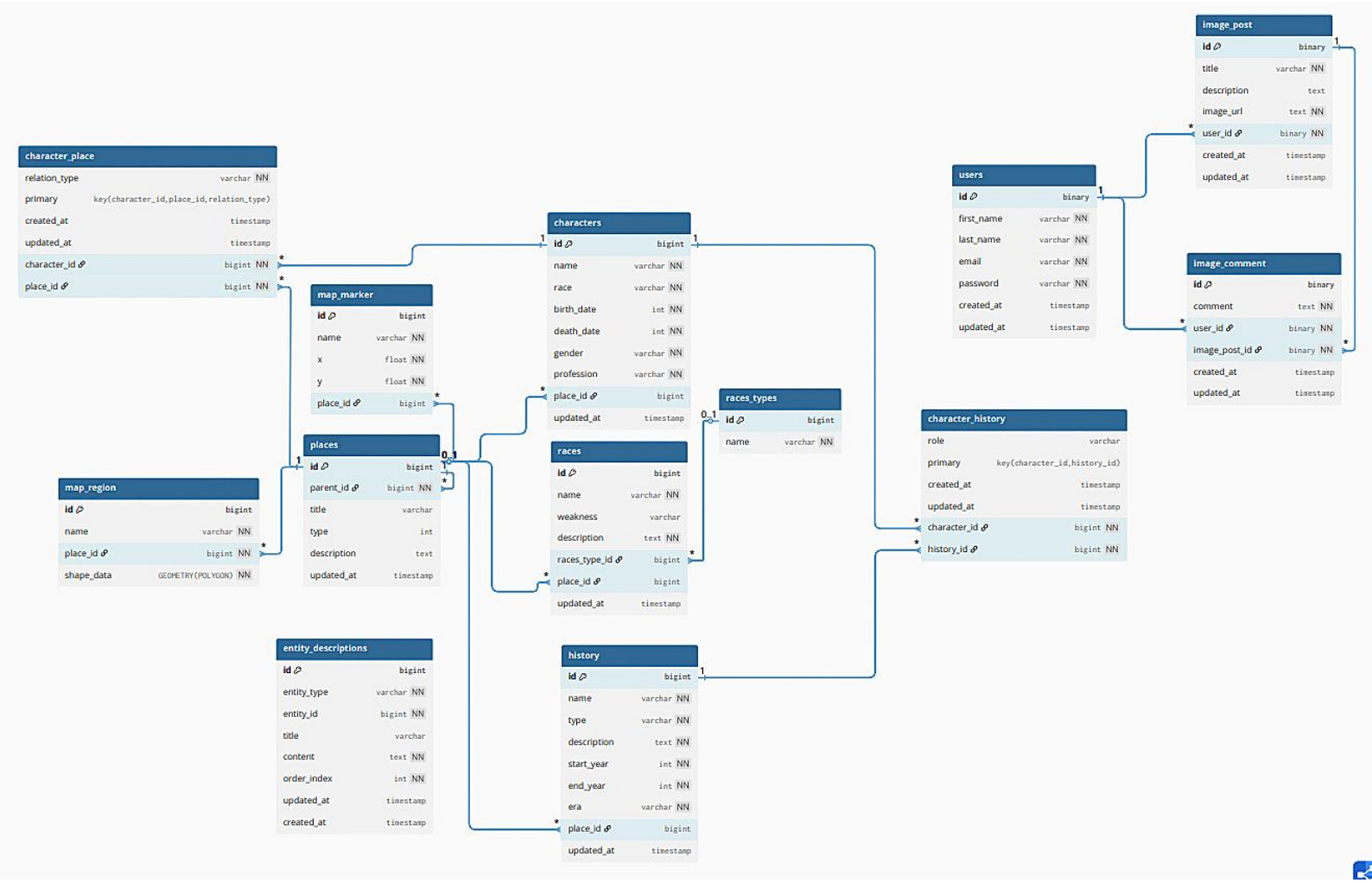
- **Langage :** SQL.
- **Technologie :** PostgreSQL (base relationnelle) avec l'extension **PostGIS** pour gérer les données spatiales.
- **Stockage :**
    - Données **structurées** : utilisateurs, personnages, histoires, créatures, créations artistiques, commentaires.
    - Données **géographiques** :
        - Polygones (zones : Gondor, Mordor, etc.).
        - Points (lieux précis : villes, montagnes, forteresses).
- **Relations gérées** via clés primaires et étrangères.

3. Définir les composants, les classes et la conception de la DB

3.1. Back-end, Classes, attributes et méthodes



3.2. Base de données, Diagramme ER (entité-relation)



3.3 Front-end, composants interface et intéractions

App

- **Rôle** : Point d'entrée de l'application, gère le **routing** et le contexte global (authentification, thème…).

- **Interactions** :
  - Contient le **Header**, le **Menu de navigation**, le **contenu principal** et le **footer**.
  - Passe les props (paramètre) nécessaires aux composants enfants (utilisateur, données chargées depuis le back-end).

Header

- **Rôle** : Barre de navigation et informations utilisateur.

- **Contenu** : Logo, menu de connexion/inscription,(bouton profil), bouton recherche.

- **Interactions** :
    - Permet de se connecter à son compte
    - Recherche sur le site

Sidebar

- **Rôle** : Onglets pour accéder aux différentes sections :
    - **Carte interactive**
    - **Personnages / Histoire / Créatures**
    - **Créations artistiques (photos et commentaires)**

- **Interactions** :

    - Change le composant affiché dans le **contenu principal**.
    - Permet de naviguer entre les onglets principaux : Carte, personnages / Histoire / Créatures, Créations Artistiques.
    - Permet d'agrandir ou de réduire la sidebar

Map

- **Rôle** : Affiche la carte de la Terre du Milieu.

- **Fonctionnalités** :
    - Points cliquables pour régions
    - Contour sur région avec effet possible au survol

- **Interactions** :
    - Clic sur un marqueur (lieu) → affiche une nouvelle page avec le lieu rattaché.
    - Possibilité de filtrer les marqueurs, selon les types de lieux (forêt, forteresse, ville…)

InfoGroupe

- **Rôle** : Affiche tous les éléments du groupe sélectionné (personnage ou créature ou histoire)

- **Contenu** :
  - Nom + images (illustration).
  - Liens vers d'autres personnages ou événements liés.

- **Interactions** :
  - Reçoit les données du back-end via API REST.
  - Peut passer des filtres ou critères de sélection aux composants enfants.
  - Composant réutilisable React pour l'affichage de chaque groupe

InfoSolo

- **Rôle** : Affiche le détail de l'élément sélectionné dans infoGroupe

- **Contenu** :
  - Nom, description, image (illustration).
  - Bloc d'information
  - Liens possibles vers d'autres personnages ou événements liés.

- **Interactions** :
  - Reçoit les données du back-end via API REST.
  - Composant réutilisable React pour l'affichage de chaque élément sélectionné

ArtWorkGallery

- **Rôle** : Affiche les créations des utilisateurs (images, illustrations, fan art).

- **Contenu** :
  - Miniatures, titre, auteur.
  - Possibilité de cliquer pour voir l'image en grand et les commentaires.

- **Interactions** :
  - Chargement des données depuis le back-end (images et commentaires).

ArtWorkSolo

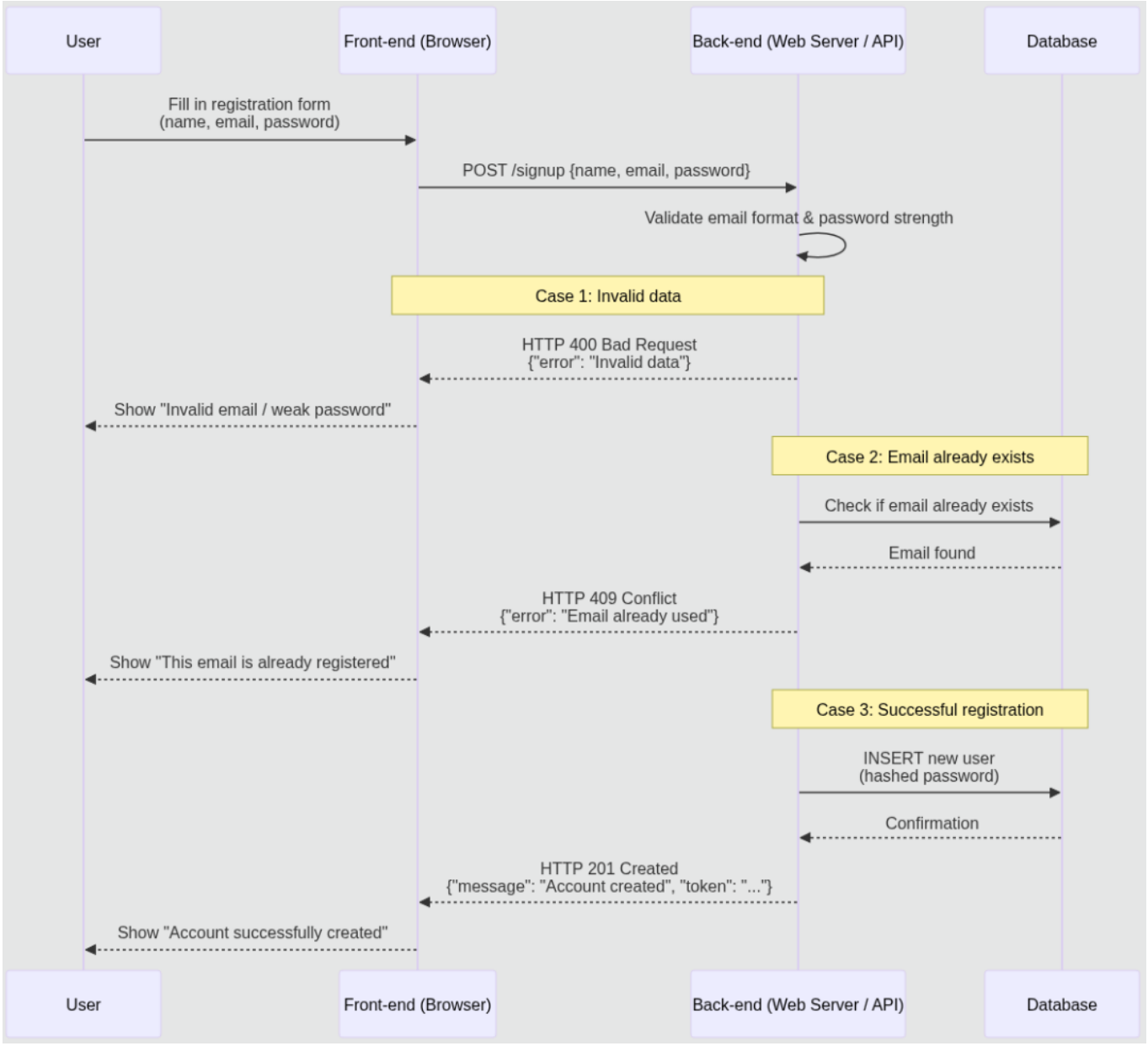
- **Rôle** : Gestion des commentaires pour chaque création artistique.

- **Contenu** : Liste des commentaires, champ pour en ajouter un nouveau.

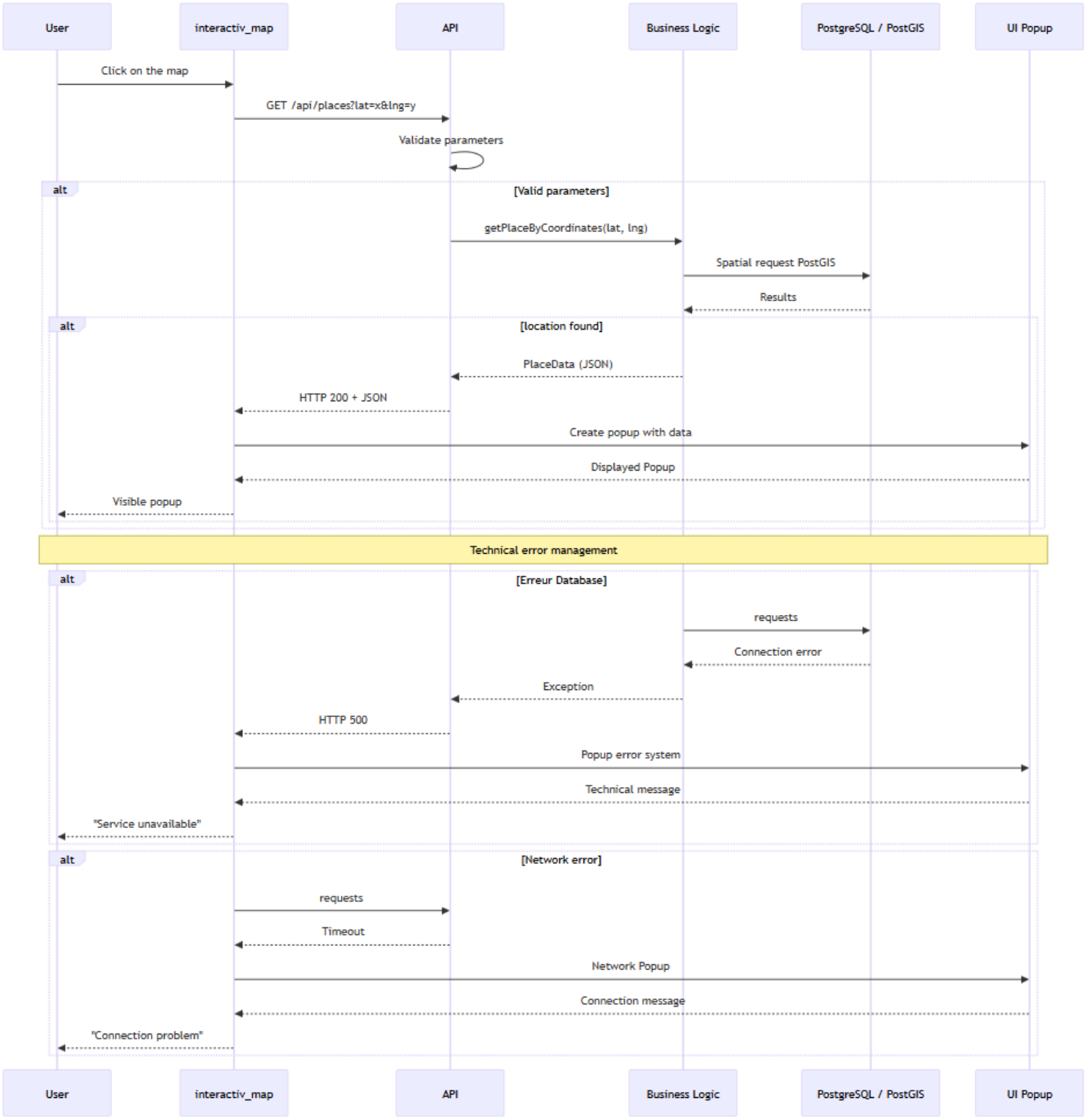
- **Interactions** :
  - POST pour ajouter un commentaire, GET pour récupérer la liste.
  - Mise à jour en temps réel possible via WebSocket ou rafraîchissement.
  - Composant React pour les commentaires

4. Diagrammes de séquence de haut niveau

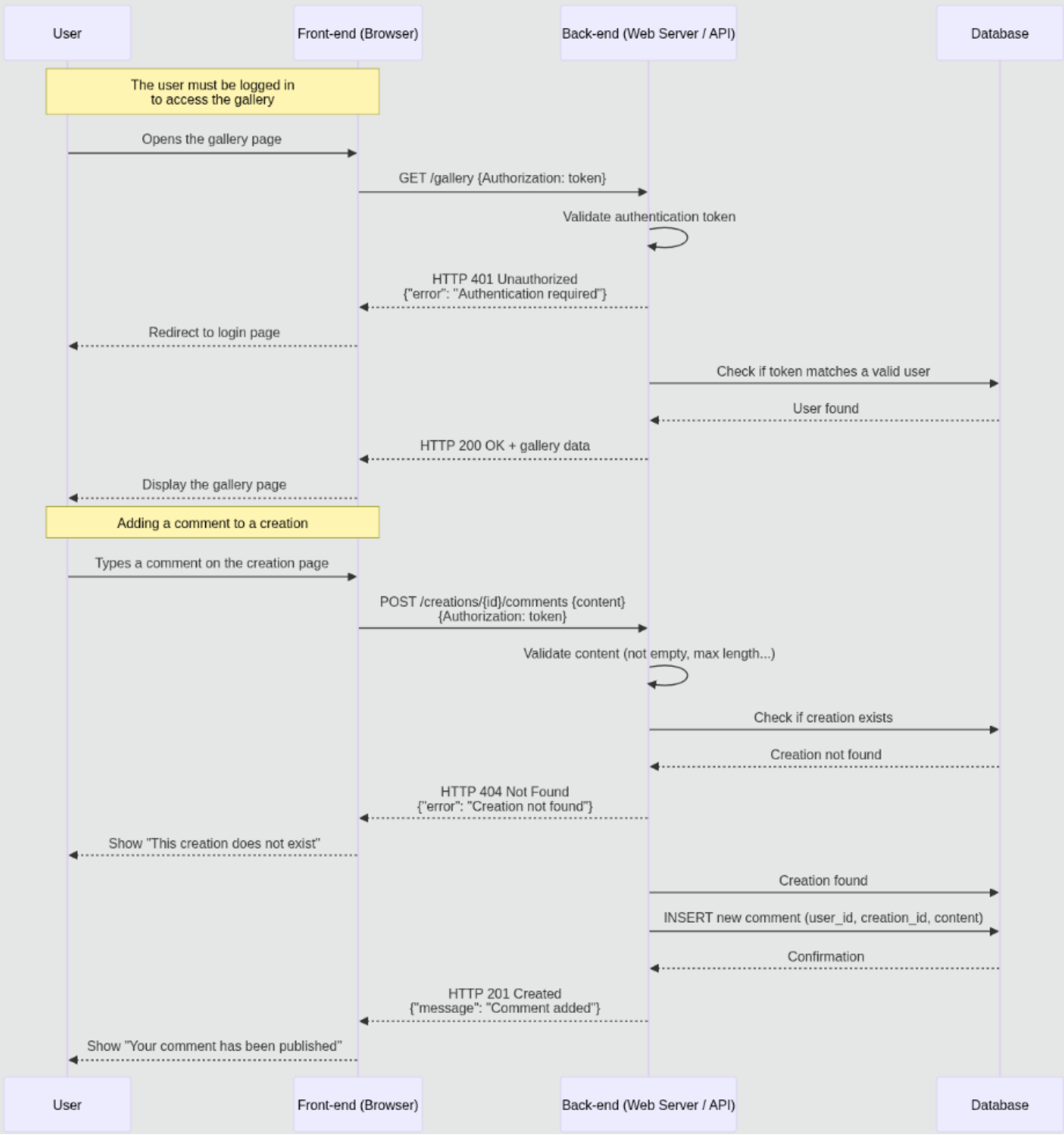
4.1. Enregistrement utilisateur



4.2. Clic sur la map



4.3. Ajout d’un commentaire



5. Documenter l'API interne

5.1. Tableau points de terminaison internes

Ressource	Méthode	Endpoint	Entrée (JSON / query param)	Sortie (JSON / GeoJSON)	Codes retour	Description
User	POST	/api/v1/users/register	{ "username", "email", "password" }	{ "id", "username", "email" }	201, 400, 409	Création d'un nouvel utilisateur
	POST	/api/v1/users/login	{ "email", "password" }	{ "token" }	200, 400, 401	Connexion utilisateur, renvoie un token JWT
	GET	/api/v1/users/{id}	—	{ "id", "username", "email" }	200, 404	Récupération d'un utilisateur par ID
	PUT	/api/v1/users/{id}	{ "username?", "email?", "password?" }	{ "id", "username", "email" }	200, 400, 404	Modification d'un utilisateur
	DELETE	/api/v1/users/{id}	—	{ "message": "User deleted" }	200, 404	Suppression d'un utilisateur
Character	GET	/api/v1/characters	—	[{ "id", "name", "race", "storyId" }]	200	Liste de tous les personnages
	POST	/api/v1/characters	{ "name", "race", "storyId" }	{ "id", "name", "race", "storyId" }	201, 400	Création d'un personnage
	GET	/api/v1/characters/{id}	—	{ "id", "name", "race", "storyId" }	200, 404	Récupération d'un personnage par ID
	PUT	/api/v1/characters/{id}	{ "name?", "race?", "storyId?" }	{ "id", "name", "race", "storyId" }	200, 400, 404	Modification d'un personnage
	DELETE	/api/v1/characters/{id}	—	{ "message": "Character deleted" }	200, 404	Suppression d'un personnage
History	GET	/api/v1/stories	—	[{ "id", "title", "summary" }]	200	Liste de toutes les histoires
Creature	GET	/api/v1/creatures	—	[{ "id", "name", "species" }]	200	Liste de toutes les créatures
Régions (zones)	GET	/api/v1/regions	—	GeoJSON FeatureCollection	200	Récupération de toutes les régions
	POST	/api/v1/regions	GeoJSON Feature (Polygon)	GeoJSON Feature	201, 400	Création d'une nouvelle région
	GET	/api/v1/regions/{id}	—	GeoJSON Feature	200, 404	Récupération d'une région par ID
	PUT	/api/v1/regions/{id}	GeoJSON Feature (Polygon modifié)	GeoJSON Feature	200, 400, 404	Modification d'une région
	DELETE	/api/v1/regions/{id}	—	{ "message": "Region deleted" }	200, 404	Suppression d'une région
Lieux (points)	GET	/api/v1/locations	—	GeoJSON FeatureCollection	200	Récupération de tous les lieux
	POST	/api/v1/locations	GeoJSON Feature (Point)	GeoJSON Feature	201, 400	Création d'un nouveau lieu
	GET	/api/v1/locations/{id}	—	GeoJSON Feature	200, 404	Récupération d'un lieu par ID
ArtWorks	GET	/api/v1/artworks	—	[{ "id", "title", "authorId" }]	200	Liste des créations artistiques
	POST	/api/v1/artworks	{ "title", "description", "authorId" }	{ "id", "title", "description", "authorId" }	201, 400	Création d'une nouvelle œuvre
	GET	/api/v1/artworks/{id}	—	{ "id", "title", "description", "comments" }	200, 404	Récupération d'une œuvre par ID
	PUT	/api/v1/artworks/{id}	{ "title?", "description?" }	{ "id", "title", "description" }	200, 400, 404	Modification d'une œuvre
	DELETE	/api/v1/artworks/{id}	—	{ "message": "Artwork deleted" }	200, 404	Suppression d'une œuvre
Comment	GET	/api/v1/artworks/{id}/comments	—	[{ "id", "text", "userId", "date" }]	200, 404	Liste des commentaires d'une œuvre
	POST	/api/v1/artworks/{id}/comments	{ "text", "userId" }	{ "id", "text", "userId", "date" }	201, 400, 404	Création d'un commentaire pour une œuvre
Search	GET	/api/v1/search	`query` : texte à rechercher	{ "characters": [...], "locations": [...] }	200, 400	Recherche globale sur personnages et lieux
				Characters array : [{ "id", "name", "race", "storyId" }]		Filtrage insensible à la casse
				Locations array : [{ "id", "name", "regionId", "latitude", "longitude" }]		Les lieux peuvent être affichés sur Leaflet avec marker et popup
			`query` manquant	{ "error": "Query param 'query' is required" }	400	Paramètre de recherche obligatoire
			aucun résultat	{ "characters": [], "locations": [] }	200	Aucun résultat trouvé

5.2. Matrice CRUD avec rôles et permissions

Ressource	Action	Invité	Utilisateur connecté (User)	Admin
Personnage	Create	✗	✗	✓
	Read	✓	✓	✓
	Update	✗	✗	✓
	Delete	✗	✗	✓
Histoire	Create	✗	✗	✓
	Read	✓	✓	✓
	Update	✗	✗	✓
	Delete	✗	✗	✓
Créature	Create	✗	✗	✓
	Read	✓	✓	✓
	Update	✗	✗	✓
	Delete	✗	✗	✓
Création artistique	Create	✗	✓ (seulement ses images)	✓
	Read	✓	✓	✓
	Update	✗	✓ (seulement ses images)	✓
	Delete	✗	✓ (seulement ses images)	✓
Commentaire	Create	✗	✓ (seulement ses commentaires)	✓
	Read	✓	✓	✓
	Update	✗	✓ (seulement ses commentaires)	✓
	Delete	✗	✓ (seulement ses commentaires)	✓
Lieux	Create	✗	✗	✓
	Read	✓	✓	✓
	Update	✗	✗	✓
	Delete	✗	✗	✓

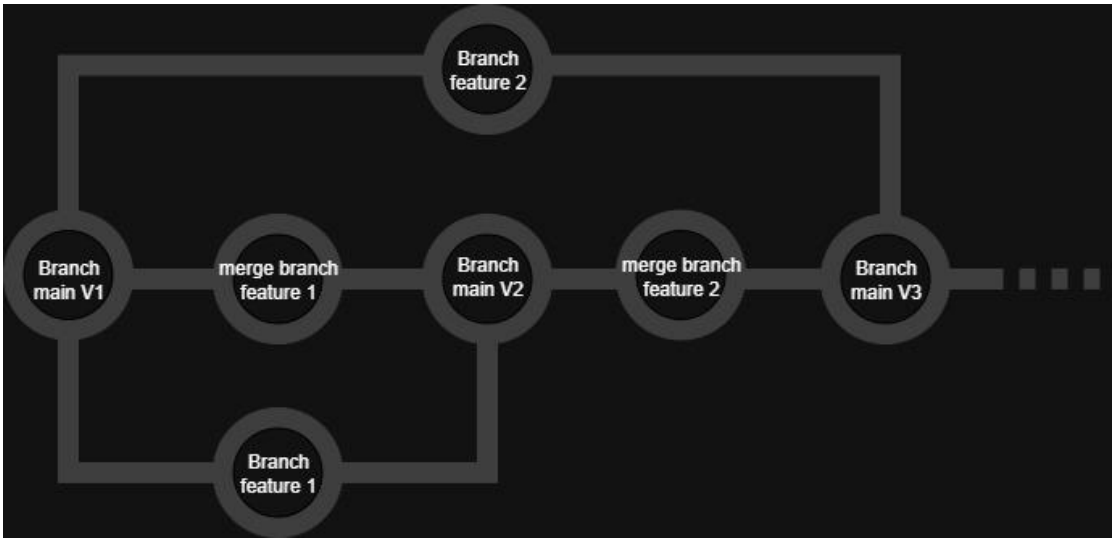
6. Planifier les stratégies SCM et QA

6.1. Stratégie SCM - Contrôle de version avec l'outil GIT

- ### Template de commit GIT:

  - Utilisation de format professionnel pour les commits, (ex: docs: update README with installation instructions or fix: resolve login button)
  - Les titres commits ne dépasseront pas 50 caractères pour se restreindre à une attention unique et claire
  - Les commits qui auront besoin d'une description détaillée, contiendront plus de caractères mais avec une segmentation pour chaque fonctionnalité ajoutée
  - Type pour définir le type de commit :
    - Docs : Documentation.
    - Feat : ajout d'une fonctionnalité.
    - Fix : correction de bogue.
    - Refactor : changement du code qui n'influe pas sur le fonctionnement.
    - Test : modification des tests
    - Style : Changement du style du code (sans changer la logique).

- ### Etablir une stratégie de branche :



- ### Processus de collaboration git :
  - Commits réguliers
    - Après chaque tâche significative (ajout d'une fonctionnalité, correctif, etc…)
  - Pull requests + Revues de code

    - On fait une demande de fusion avant de l'envoyer dans la branche main. L'étudiant a ensuite pour objectif d'analyser le code soumis par un autre étudiant afin d'en améliorer la qualité. Il peut proposer des optimisations, signaler d'éventuelles erreurs, ou suggérer d'autres améliorations si nécessaire.

6.2. Stratégie QA (assurance qualité)

6.2.1. Définir une stratégie de tests

- Tests unitaires : Vérifient le comportement des fonctions, méthodes et classe isolées.
    - Outil
        - Pytest

- Tests d'intégration : Vérifient les modules fonctionnent ensemble, endpoints, API ⇔ DB
    - Outil
        - Postman
        - Newman : pour automatiser dans pipeline CI/CD

6.2.2. Planifier un pipeline de déploiement

Un pipeline de déploiement est un processus automatisé qui fait passer votre code du développement à la production en passant par différentes étapes de validation. Il a pour but de tester l'application avant la mise en production.

Outils utilisés :

Github actions : pour lancer automatiquement les actions quand on pousse du code via github

Docker : L'application est stockée dans un conteneur pour permettre d'avoir un environnement identique sur la machine (local) + staging + production.

Plan du pipeline

Partie réalisable pour le portfolio :

- Ecriture du code.
- Push du code sur github.
- Github actions avec les tests se lancent automatiquement

Qu'est-ce que c'est ?

- Local : développement sur ma machine
- Staging : tests avant production
- Production : version finale accessible aux utilisateurs

Partie optionnelle (si temps disponible) :

- Si aucun problème rencontré, alors déploiement automatique sur staging.
- Test sur staging manuellement.
- Si le staging est validé, on déploie manuellement en production.
- On vérifie que tout fonctionne.
    - Si un problème arrive, il est possible de revenir en arrière.

Services possibles pour hébergement gratuits :

- Heroku
- Railway
- Vercel
- Autres…

Objectifs :

- **Principal** : Dockerfile + GitHub Actions avec tests automatiques
- **Bonus** : Environnement de test + mise en production

7. Justification techniques

React

- Utilisation de React car cela va permettre de gagner du temps en utilisant des composants réutilisables pour HTML + CSS

PostgreSQL / Postgis

- Utilisation intéressante car permet une meilleure gestion sur de gros volumes de données et permet d'avoir un système d'information géographique grâce à PostGIS qui va permettre l'utilisation de formes géospatiales avec des coordonnées pour permettre de placer un marqueur sur une zone spécifique

Python / Flask

- Utilisation de Python (et Flask) car nous avons une meilleure connaissance de ce langage et nous avons déjà eu l'occasion de travailler dessus sur un projet de groupe. De plus, étant donné que nous apprenons beaucoup de nouvelles technologies dans ce projet, nous préférons éviter de nous surcharger en apprenant trop de nouveaux langages simultanément afin d'avoir le temps de sortir un projet fonctionnel que nous arriverons à expliquer et maîtriser

Javascript / Leaflet

- Utilisation de JavaScript pour un affichage dynamique des pages web. Leaflet, une bibliothèque JavaScript, va nous être utile pour visualiser et styliser les données géospatiales stockées dans PostGIS