ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

Université de Rennes

# Rapport Final : Projet SI

Auteurs :

Théo LAMINIE
Antoine RAULT
Fanny SHEHABI

2023 - 2024

# Table des matières

# Rapport d’analyse

## Rappel du besoin et critères du succès

Le besoin du projet est de déterminer les emplacements stratégiques pour aménager des infrastructures cyclables à Rennes tout en estimant les risques d'accidents associés.

Les éléments clés du projet sont les suivants :

- Identification des zones nécessitant des aménagements cyclables.
- Évaluation des risques d'accidents liés à la circulation des cyclistes.
- Fournir une aide à la décision pour les élus et les citoyens.

Les critères de succès sont les suivants :

- Un réduction du nombre d’accidents impliquant des cyclistes
- Une adoption positive de notre solution par les élus et les citoyens

## Modèle du domaine métier

Détaillons les différents cas d’utilisation.

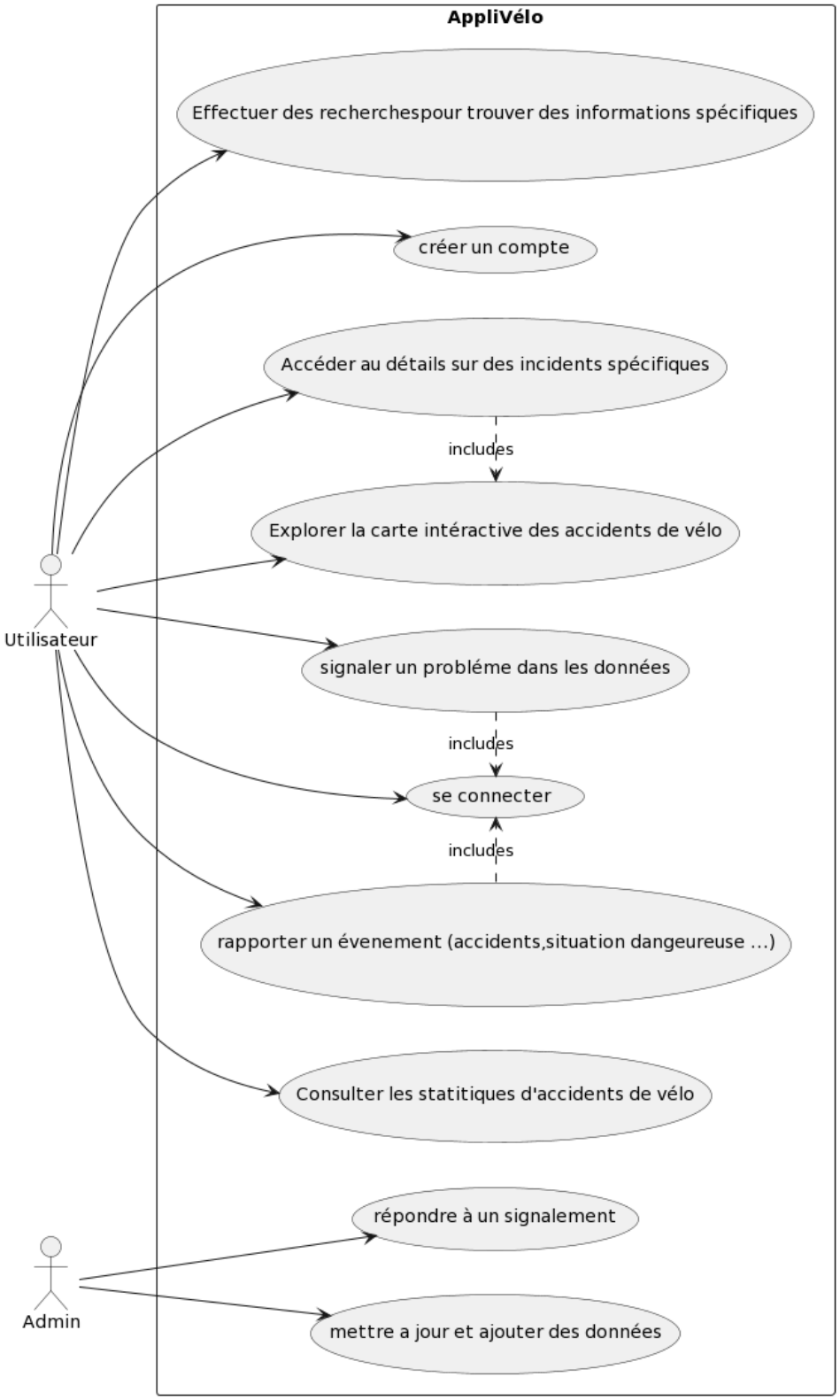



Figure 1 : Diagramme de cas d'utilisation

**Cas d'utilisation: Consulter les statistiques d'accidents de vélo**

- **But:** Permettre à l'utilisateur de consulter les statistiques d'accidents de vélo
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur a consulté les statistiques
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur accède au site
    2. L'utilisateur accède aux différents catégories d'accidents
    3. Le système affiche les statistiques sélectionnées
- **Alternatives:** -
- **Exceptions:** -

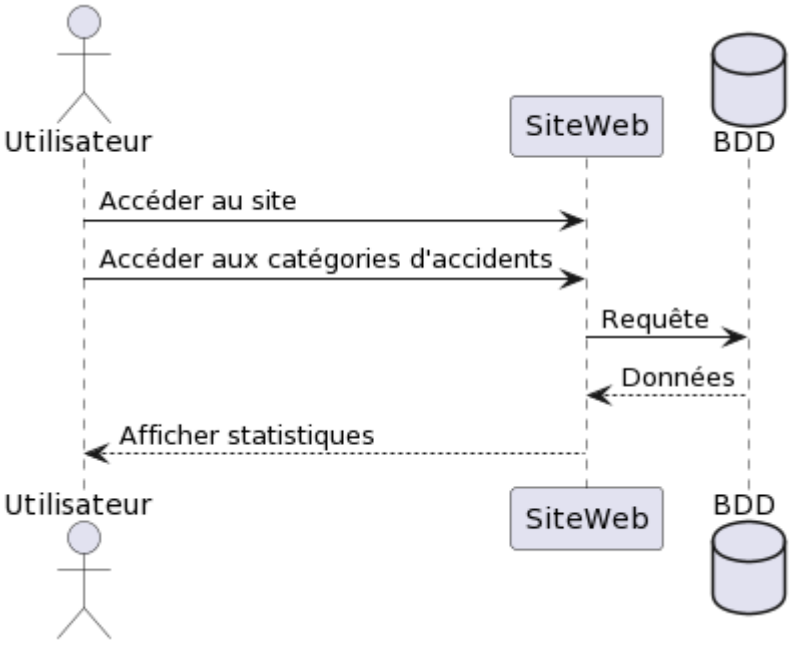



Figure 2 : Diagramme de séquence contextuel du use case *Consulter les statistiques d'accidents de vélo*

**Cas d'utilisation: Explorer la carte intéractive des accidents de vélo**

- **But:** Permettre à l'utilisateur d'explorer la carte interactive des accidents de vélo
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur a exploré la carte
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur accède au site
    2. L'utilisateur accède à la carte
    3. Le système affiche la carte
- **Alternatives:** -
- **Exceptions:** -

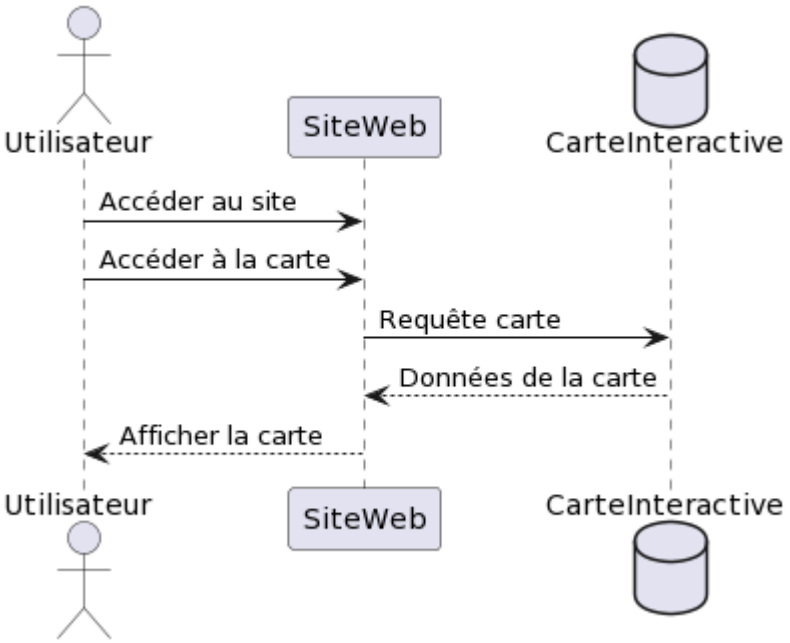



Figure 3 : Diagramme de séquence contextuel du use case *Explorer la carte interactive des accidents de vélo*

**Cas d'utilisation: Accéder aux détails d'un accident spécifique**

- **But:** Permettre à l'utilisateur d'accéder aux détails d'un accident choisi
- **Début:** L'utilisateur a exploré à la carte
- **Fin:** L'utilisateur a accédé aux détails de l'accident
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur accède à la carte
    2. L'utilisateur consulte les détails de l'accident
    3. Le système affiche les détails
- **Alternatives:**
    - L'accident n'a pas de détails

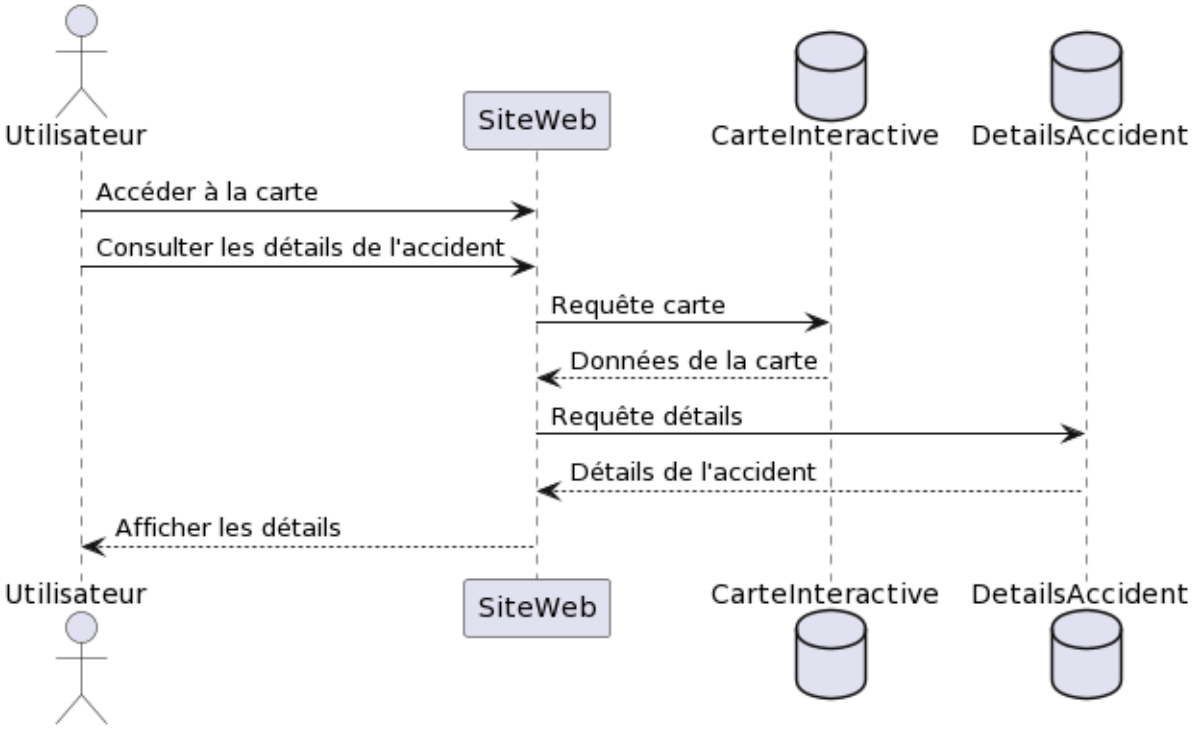



Figure 4 : Diagramme de séquence contextuel du use case *Accéder aux détail d'un accident spécifique*

**Cas d'utilisation: Effectuer des recherches pour trouver des informations spécifiques**

- **But:** Permettre à l'utilisateur d'effectuer des recherches pour trouver des informations spécifiques
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur a accédé aux informations qu'il cherchait
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur effectue une recherche à l'aide de filtres
    2. Le système affiche le résultat de la recherche en fonctions des filtres appliqués
- **Alternatives:**
    - La recherche n'a pas de résultat
- **Exceptions:** -

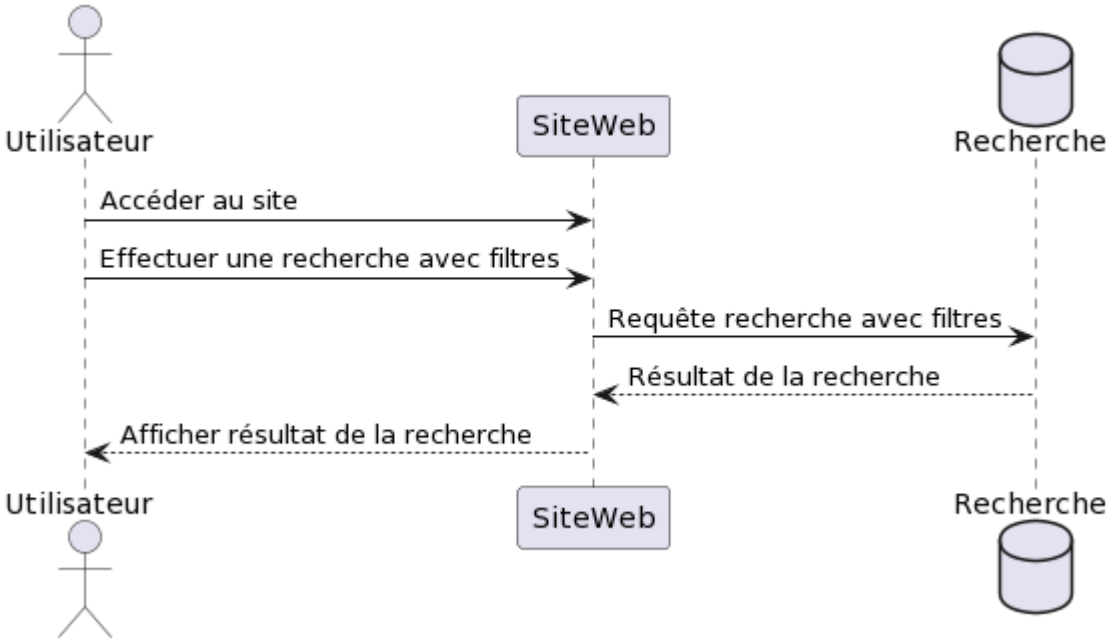



Figure 5 : Diagramme de séquence contextuel du use case *Effectuer des recherches*

**Cas d'utilisation: Création de compte**

- **But:** Permettre à l'utilisateur de créer un compte
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur a un compte
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur souhaite créer un compte
    2. L'utilisateur rentre les données personnelles nécessaires
    3. Le système crée le compte utilisateur
- **Alternatives:**
    - Les identifiants correspondent déjà à un compte
- **Exceptions:** -

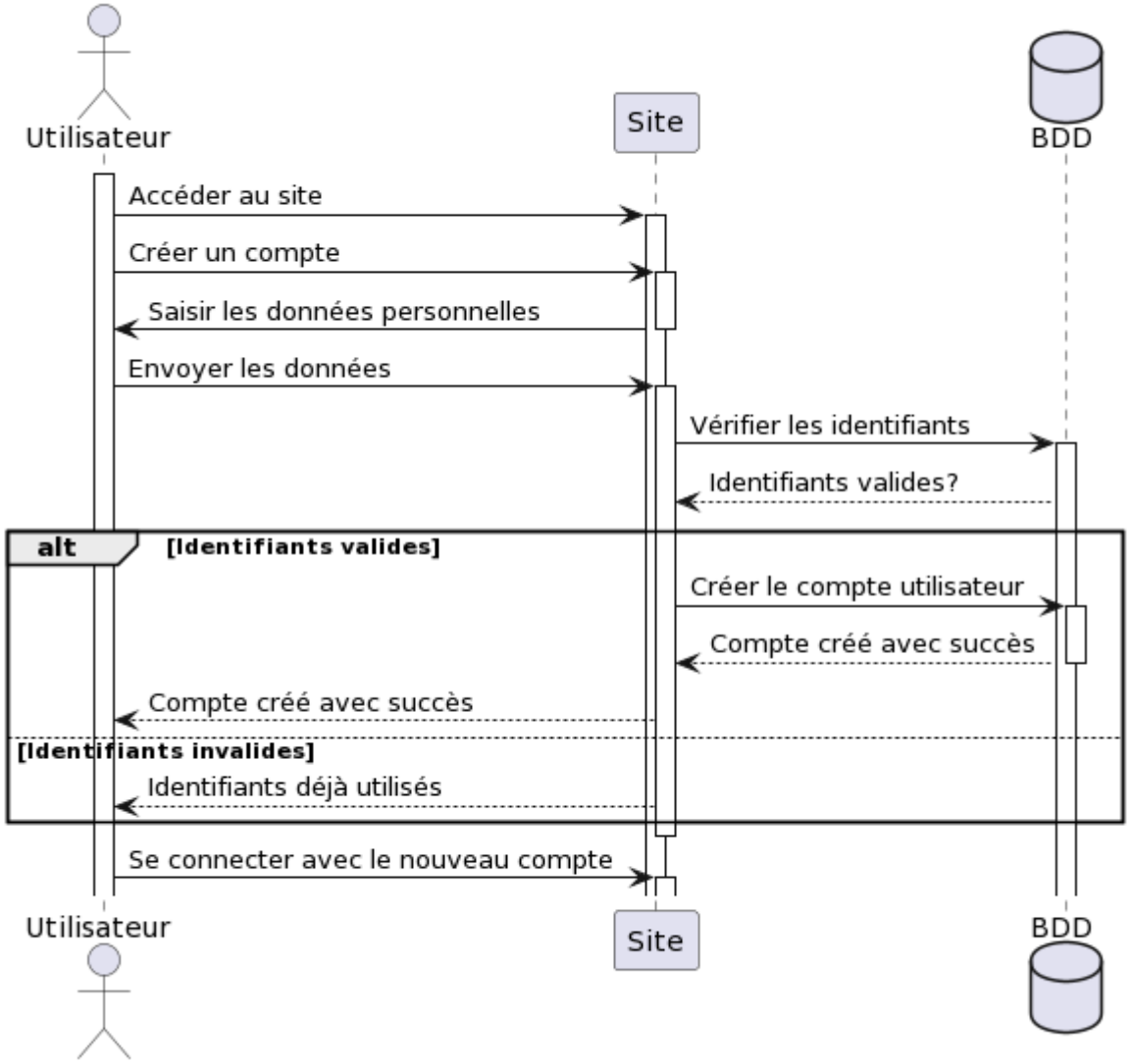



Figure 6 : Diagramme de séquence contextuel du use case *Créer un compte*

**Cas d'utilisation: Se connecter**

- **But:** Permettre à l'utilisateur de se connecter
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur est connecté
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur rentre ses identifiants
    2. Le système connecte l'utilisateur
- **Alternatives:**
    - Les identifiants sont faux
- **Exceptions:** -

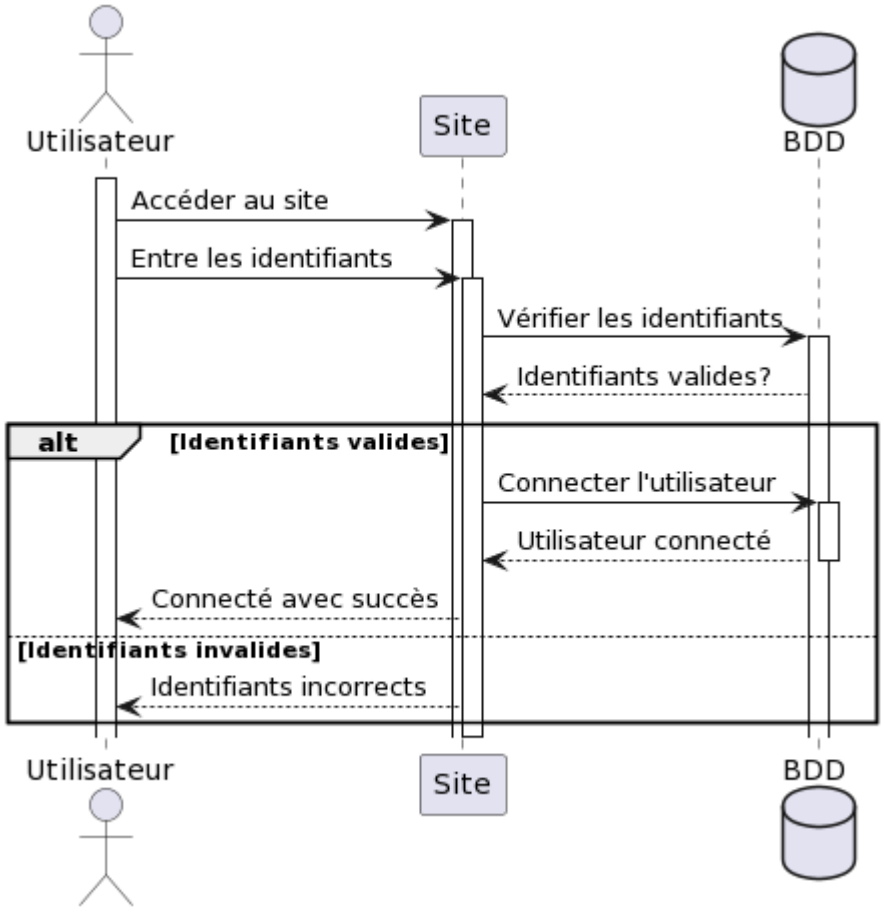



Figure 7 : Diagramme de séquence contextuel du use case *Se connecter*

**Cas d'utilisation: Rapporter un événement**

- **But:** Permettre à l'utilisateur de rapporter un accident, accrochage ou zone dangereuse
- **Début:** L'utilisateur accède au site
- **Fin:** L'utilisateur a rapporté le problème
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur choisi le type de problème
    2. L'utilisateur place le point sur la carte
    3. L'utilisateur remplit un léger formulaire
- **Alternatives:** -
- **Exceptions:** -

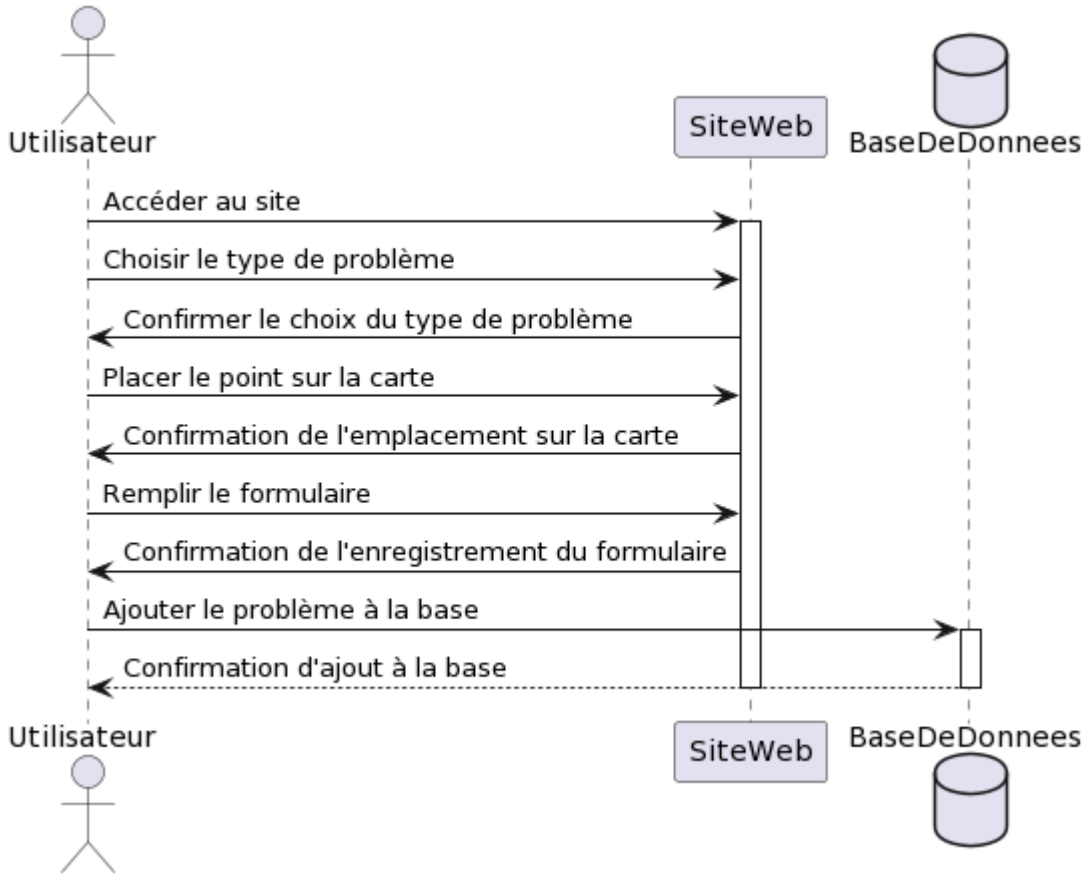



Figure 8 : Diagramme de séquence contextuel du use case *Rapporter un évènement*

**Cas d'utilisation: Signaler un problème dans les données**

- **But:** Permettre à l'utilisateur de signaler des données erronées
- **Début:** L'utilisateur souhaite signaler un problème
- **Fin:** L'utilisateur a signaler un probléme
- **Acteur:** Utilisateur
- **Enchaînement:**
    1. L'utilisateur signale un problème
    2. L'utilisateur donne les informations correctes
    3. Le système enregistre le signalement et l'envoie aux admins
- **Alternatives:** -
- **Exceptions:** -

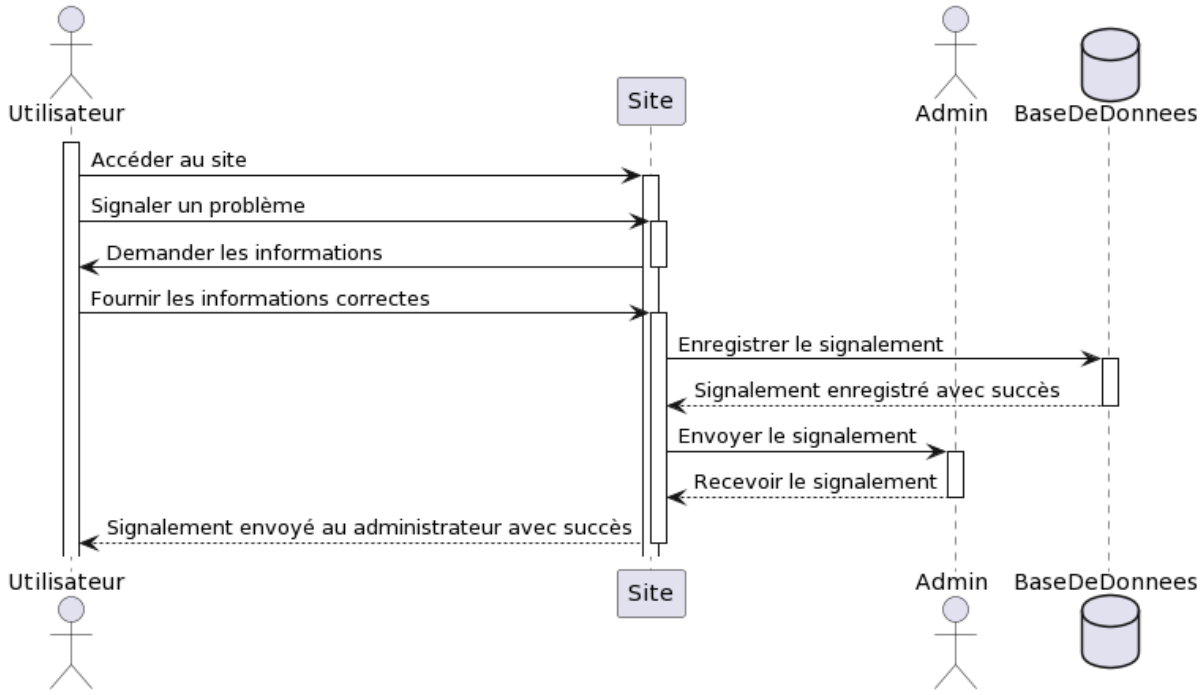



Figure 9 : Diagramme de séquence contextuel du use case *Signaler un problème dans les données*

**Cas d'utilisation: Répondre à un signalement**

- **But:** Répondre à un signalement
- **Début:** Sélectionne un signalement
- **Fin:** Le signalement à été traité
- **Acteur:** Admin
- **Enchaînement:**
    1. Sélectionne un signalement
    2. Valide le signalement
    3. Le système ajoute la modification à la base de données
- **Alternatives:**
    - Invalidation du signalement et suppression du signalement
- **Exceptions:** -

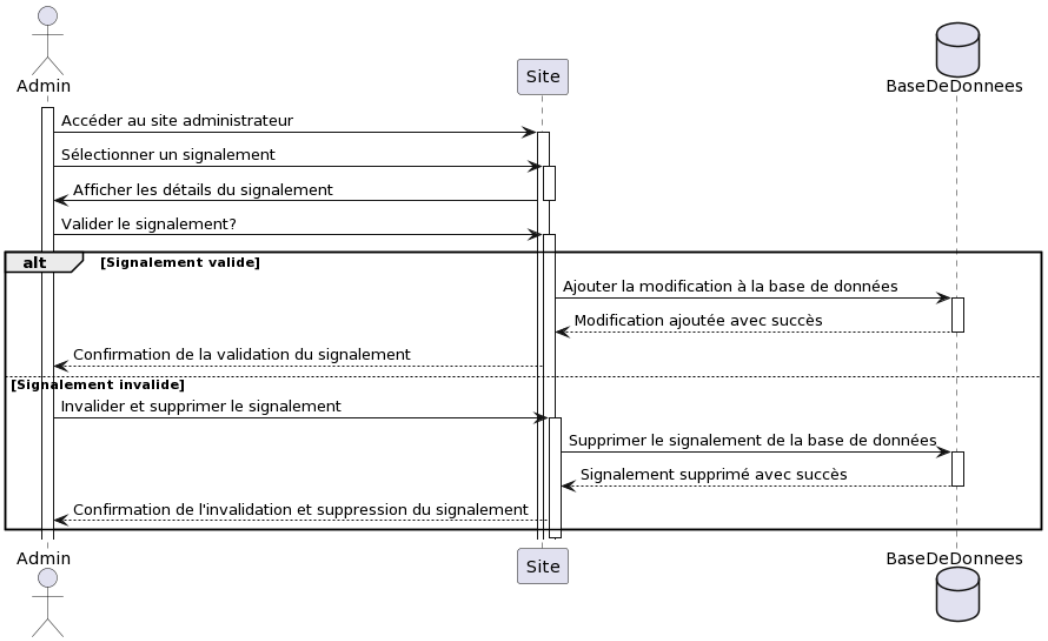



Figure 10 : Diagramme de séquence contextuel du use case *Répondre à un signalement*

**Cas d'utilisation: Mettre à jour ou ajouter des données**

- **But:** Mettre à jour ou ajouter des données
- **Début:** Connexion en admin
- **Fin:** Données mise à jour ou ajoutées
- **Acteur:** Admin
- **Enchaînement:**
    1. Choisi d'ajouter ou de modifier une donnée
    2. Donne les informations de la nouvelle donnée
    3. Le système l'ajoute à la base de données
- **Alternatives:** -
- **Exceptions:** -

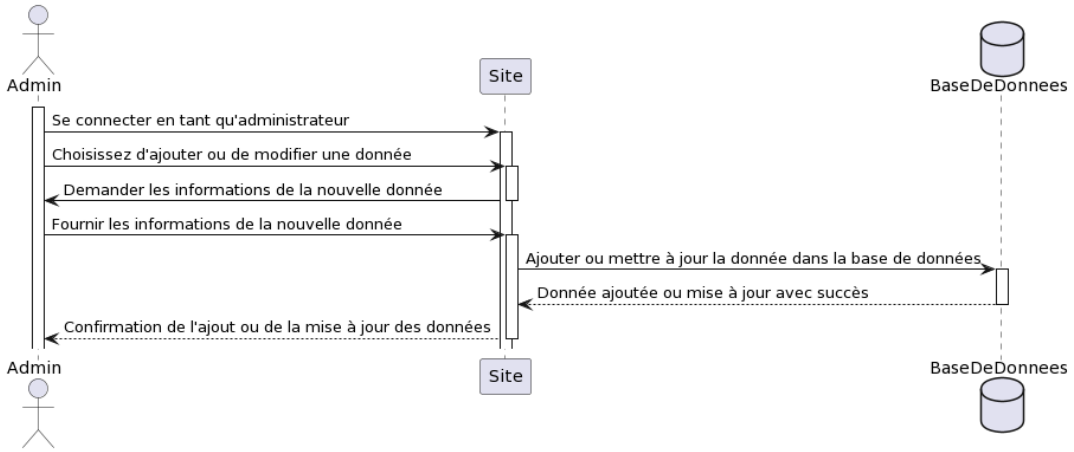



Figure 11 : Diagramme de séquence contextuel du use case *MàJ/Ajouter des données*

## Diagramme de classe

Lien PlantUML du Diagramme de Classe

## Diagramme de séquence

Lien PlantUML du Diagramme de Séquence

# Description de l’écosystème

Le système doit s'intégrer avec :

- **OpenStreetMap ou une API de cartographie similaire** pour la représentation spatiale des données.
- **La base de données Open Data** pour l’accès aux données d'accidents.

# Principe de solution

La solution proposée consiste en une application web comprenant :

- Consultation des données existantes sur les accidents de vélo à Rennes, présentées sous la forme d'une carte interactive et de statistiques.
- Collecte des données des utilisateurs permettant de signaler des accidents, des dangers potentiels ou des suggestions d'aménagements.

# Rapport d’architecture & rapport de design

## Principe de mise en oeuvre de la solution

La solution sera implémentée sous la forme de microservices interagissant à travers des API bien définies. Les services seront déployés de manière indépendante pour permettre une évolutivité facile. Les interactions entre les composants se feront de manière asynchrone lorsque cela est possible pour améliorer les performances.

## Règles d’architecture

Pour assurer la qualité et la cohérence de la solution, nous avons défini des règles d’architecture qui sont :

- Assurer la sécurité et la confidentialité des données
- Respecter les principes RESTful pour les API
- Utiliser des containers Docker pour assurer la portabilité
- Mettre en œuvre des tests unitaires et d'intégration pour garantir la fiabilité du système

## Modèle statique

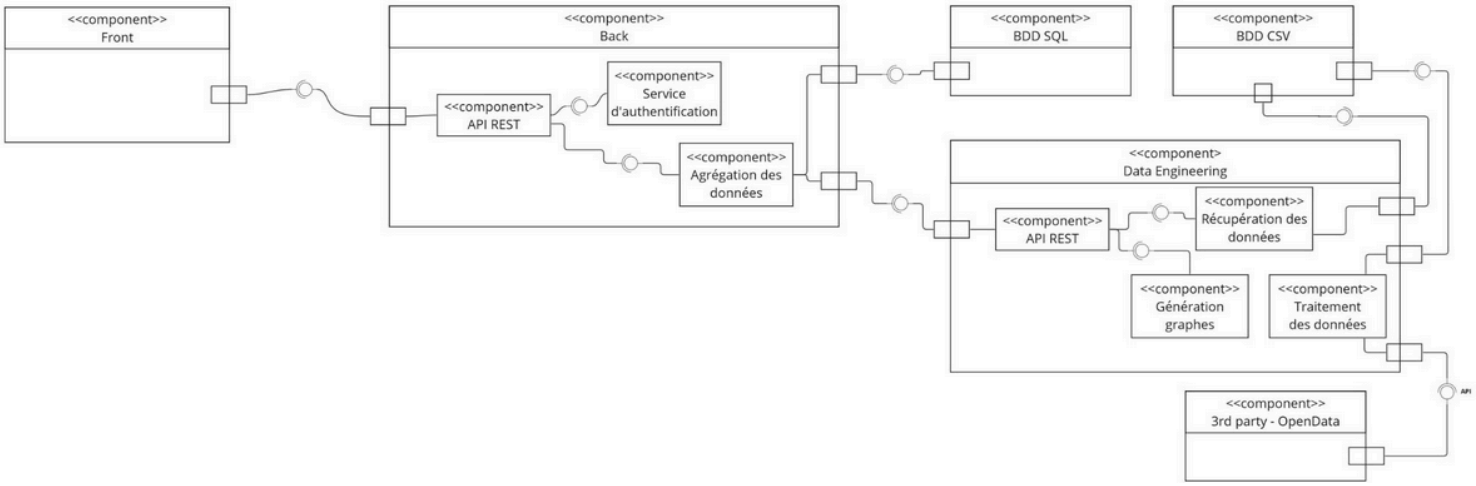



Figure 12 : Diagramme de composants

## Organisation des packages

Le projet est organisé en plusieurs services, chacun étant contenu dans son propre package. Les services principaux incluent l'ingénierie des données, la base de données, le backend, le frontend et le serveur Nginx. Voici une description des packages et de leurs interactions :

- **data-engineering** : Gère les processus d'ingénierie des données. Ce service s'occupe de la transformation et du traitement des données.
- **database** : Fournit la base de données PostgreSQL pour stocker les signalements des utilisateurs. Ce service est configuré pour redémarrer automatiquement et inclut des volumes pour persister les données et initialiser la base de données avec un script SQL.
- **backend** : Sert de couche intermédiaire entre la base de données et le frontend. Ce service gère la logique métier et expose des API pour que le frontend puisse interagir avec les données.
- **frontend** : Fournit l'interface utilisateur. Ce service est construit en utilisant des technologies web modernes pour offrir une expérience utilisateur interactive.
- **nginx** : Agit comme un serveur proxy inverse, dirigeant les requêtes vers les services appropriés (backend, frontend et data-engineering).

Tous ces services sont interconnectés via un réseau Docker interne nommé "back".

## Descriptions des classes principales et de leurs responsabilités

1. Data Engineering (data-engineering)

- Responsabilité : Traitement et transformation des données de Rennes Métropole.

2. Base de Données (database)

- Responsabilité : Stockage et gestion des données utilisateur.

3. Backend (backend)

- Responsabilité : Gestion de la logique métier et des API.
- Classes principales :
  - `AccidentMetropoleController` : Gère les requêtes HTTP liées aux accidents de Rennes Métropole.
  - `SignalementController` : Gère les requêtes HTTP liées aux signalements utilisateurs.
  - `DatabaseConnector` : Gère les interactions avec la base de données PostgreSQL.

4. Frontend (frontend)

- Responsabilité : Interface utilisateur et expérience utilisateur.
- Utilisation de Leaflet pour les différentes cartes.
- Utilisation de Geoapify pour récupérer les adresses à partir de coordonnées GPS

- Classes principales :
  - `App` : Composant principal de l'application frontend.
  - `Map` : Affiche une carte avec les accidents de Rennes Métropole.
  - `Signalement` : Formulaire pour soumettre un nouveau signalement.

5. Nginx (nginx)

- Responsabilité : Serveur proxy inverse pour distribuer les requêtes vers les services appropriés.
- Fichier principal :
  - `nginx.conf` : Fichier de configuration Nginx définissant les routes et les proxys pour les différents services.

Cette organisation modulaire permet une gestion claire et efficace des responsabilités, facilitant la maintenance et l'extensibilité du projet. Chaque service peut être développé, déployé et mis à jour indépendamment, ce qui améliore l'agilité et la résilience de l'ensemble du système.

# Modèle dynamique

Interactions d'un utilisateur avec notre application web :

Diagramme de séquence contextuel global

## Flux nominal

### Démarrage des services :

Tous les services sont démarrés via Docker Compose.
Les services suivants sont démarrés dans l'ordre : database, backend, frontend, data-engineering, puis nginx.

### Initialisation de la base de données :

Le service database initialise la base de données PostgreSQL à l'aide du script signalements.sql.

### Interactions utilisateur :

Un utilisateur accède à l'application via le service nginx (port 8080).
Les requêtes HTTP de l'utilisateur sont redirigées vers le service frontend (port 3000).

### Soumission de signalement :

L'utilisateur soumet un signalement via l'interface du frontend.
Le frontend envoie une requête HTTP au backend (port 3001) pour créer un nouveau signalement.
Le backend traite la requête et effectue une opération d'insertion dans la base de données via DataSource.

### Affichage des signalements :

Le frontend demande la liste des signalements au backend.
Le backend récupère les données de la base de données et les renvoie au frontend pour affichage.

## Flux sur erreur (Non réalisé mais pensé)

### Erreur de connexion à la base de données :

Si le backend ne peut pas se connecter à la database, il renvoie une réponse d'erreur appropriée au frontend.
Le frontend affiche un message d'erreur à l'utilisateur indiquant que le service est temporairement indisponible.

### Erreur d'insertion de données :

Si une erreur se produit lors de l'insertion d'un signalement dans la base de données, le backend renvoie une réponse d'erreur au frontend.
Le frontend informe l'utilisateur de l'échec de l'opération et suggère de réessayer plus tard.

### Erreur de traitement des données :

Si le service data-engineering rencontre une erreur lors du traitement des données, il enregistre l'erreur dans les logs pour une analyse ultérieure et arrête le pipeline concerné.
Un mécanisme de notification peut être mis en place pour informer les administrateurs du système de l'erreur.

## Démarrage

### Exécution de Docker Compose :

La commande docker-compose up est exécutée pour démarrer tous les services définis dans le fichier docker-compose.yml.

### Ordre de démarrage :

database est démarré en premier pour garantir que la base de données est prête avant que les autres services ne tentent de s'y connecter.
backend est démarré après la base de données, et il vérifie la disponibilité de celle-ci.
frontend et data-engineering sont démarrés ensuite, car ils dépendent de la disponibilité du backend et de la database.
nginx est démarré en dernier pour agir comme point d'entrée et diriger le trafic vers les services appropriés.

### Vérifications de l'état :

Chaque service effectue des vérifications de l'état pour s'assurer qu'il peut se connecter aux services dont il dépend avant de signaler son état de "prêt".

### Arrêt

#### Arrêt de Docker Compose :

La commande docker-compose down est exécutée pour arrêter tous les services.

## Explication de la prise en compte des contraintes d'analyse

### Scalabilité

#### Architecture modulaire

Le système est conçu en microservices, ce qui permet de mettre à l'échelle chaque composant individuellement en fonction des besoins. Par exemple, le backend peut être répliqué pour gérer une augmentation du nombre de requêtes.
Les services sont orchestrés via Docker Compose, ce qui facilite le déploiement et la gestion de multiples instances de services.

### Maintenabilité

#### Code modulaire

Chaque service est encapsulé dans son propre package, avec une séparation claire des responsabilités. Cela facilite la maintenance et les mises à jour du code.
Le code est documenté et suit les meilleures pratiques de développement, ce qui permet aux développeurs de comprendre et de modifier facilement le système.

## Cadre de production

Les outils de développement incluront des environnements de développement intégrés (IDE) tels que Visual Studio Code, des outils de gestion de versions comme Git. La configuration sera gérée via des fichiers de configuration spécifiques à chaque composant. La livraison se fera à travers des conteneurs Docker.

D'autre part, nous utilisons Google Drive pour la rédaction collaborative des rapports. Nous utilisons Trello pour la gestion de projet et Discord pour communiquer facilement avec les autres membres de l'équipe.

# Rapport de gestion de projet

## Rappel des besoins et de la solution

Les besoins du projet sont les suivants :

- Déterminer où aménager des infrastructures cyclables
- Estimer les risques d'accidents

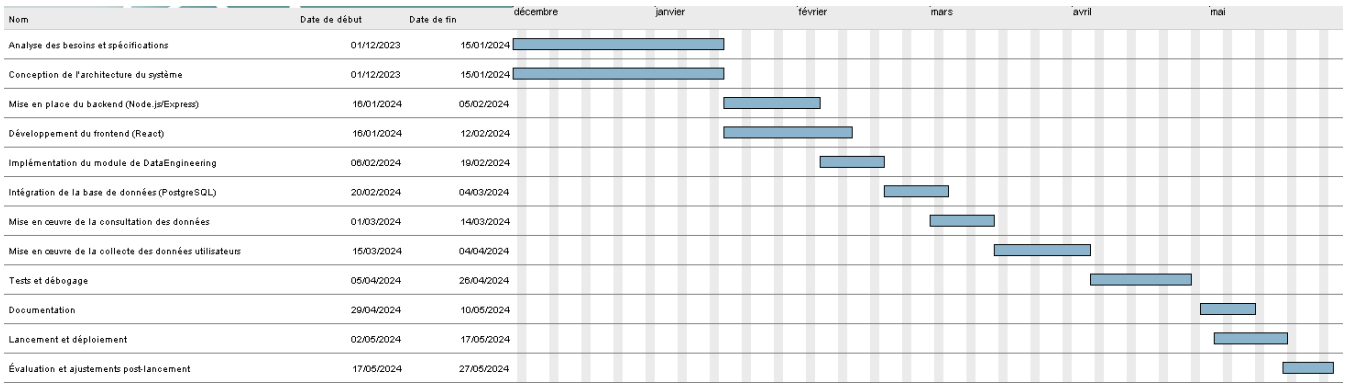
- Aide à la décision des élus et des citoyens:
    - Fournir des informations utiles pour soutenir la prise de décision des élus.
    - Impliquer les citoyens dans le processus de signalement des incidents et de suggestion d'aménagements.

Les principaux éléments de solution sont :

- Application Web Interactive :
    - Backend développé avec Node.js et Express.
    - Frontend développé avec React pour une interface utilisateur moderne.
- Base de Données :
    - Utilisation de PostgreSQL pour stocker les données relatives aux zones aménageables, aux infrastructures, aux accidents, et aux utilisateurs.
- Module de Data Engineering :
    - Développement en Python avec l'utilisation de Pandas.
    - Analyse des données d'accidents pour aider à identifier les zones à risque.
    - Mise en place d'une API avec Flask

# Estimations et déroulement du projet

Voici ci-dessous le diagramme de Gantt provisoire de notre projet. Les tâches de travail mises en parallèle sont limitées dû au nombre réduits de membres pour le S8 (équipe de 3). Ce diagramme est voué à évoluer car l'estimation des quantités de travail est approximative et débutante.




# Rôles

Les rôles et responsabilités des membres du projet sont définis comme suit :

- Product Owner : Responsable du projet et du produit
    - Zoltan Miklos
- Cheffe de projet : Responsable de l'exécution du projet

    - Fanny Shehabi
- Développeur : Responsable du développement et des tests
    - Théo Laminie
    - Antoine Rault

Les membres du projet sont responsables de la bonne communication avec les autres membres du projet. Chaque membre est responsable de la gestion de son temps et de son travail.

# Evolution du projet

Voici la planification initiale de notre projet :

| | DATE: | DATE: | DATE: | DATE: | DATE: |
|---|---|---|---|---|---|
| | **SPRINT 1** | **SPRINT 2** | **SPRINT 3** | **SPRINT 4** | **SPRINT 5** |
| A FAIRE | En tant qu'utilisateur, je souhaite explorer une carte interactive des accidents de vélo pour visualiser les zones à risque et prendre des décisions éclairées sur mes trajets ainsi que consulter les détails de cet accident. | En tant qu'utilisateur, je veux pouvoir signaler un accident, un accrochage ou une zone dangereuse sur la carte interactive pour contribuer à la sécurité routière. | En tant qu'utilisateur, je souhaite explorer une carte interactive des accidents de vélo **signalés par les utilisateurs**. | En tant qu'administrateur, je souhaite pouvoir mettre à jour ou ajouter des données pour refléter avec précision les informations les plus récentes sur la sécurité des vélos dans la région. | En tant qu'utilisateur, je souhaite pouvoir effectuer des recherches avec des filtres pour trouver des informations spécifiques sur les accidents de vélo dans ma région. |
| DELIVERABLES | Une application Web dont le front communique avec le back et dont le back communique avec une BDD | Une application Web qui contient deux onglets différents : un pour la consultation des données Rennes Métropole et un pour la signalisation | Une application Web dockerisée qui contient deux onglets différents contenant **deux cartes** : un pour la consultation des données Rennes Métropole et un pour la signalisation | Une application Web avec plusieurs Profil : un profil non connecté où l'on a accès qu'à la carte des accidents de Rennes Métropole, un profil connecté où on peut signaler et un profil Admin qui peut corriger les données | Une application Web finale où l'on peut filtrer sur la carte les différents pins (accidents ou dégradation de l' infrastructure). |

L'idée était donc de fonctionner sous forme de sprint. Chaque séance, nous faisons un point et en fin de sprint nous faisons une rétrospective du sprint ainsi que la planification pour le sprint suivant.

Afin de suivre l'évolution du projet, nous concentrerons ici les rapports de réunion et de séances.

**Sprint 0 :**

Le Lundi 4 Décembre 2023 :

- Formation des groupes
- Début de la définition des cas d'utilisation
- Recherche de données exploitables

Le Jeudi 21 Décembre 2023 :

- Traitement sur les données pour les rendre exploitables
- Premier diagramme de cas d'utilisation

Le Vendredi 22 Décembre 2023 :

- Evaluation de nos options technologiques
- Suite des traitements sur les données
- Premières représentations des données sous forme de courbes, graphiques

Le Mardi 9 Janvier 2024 :

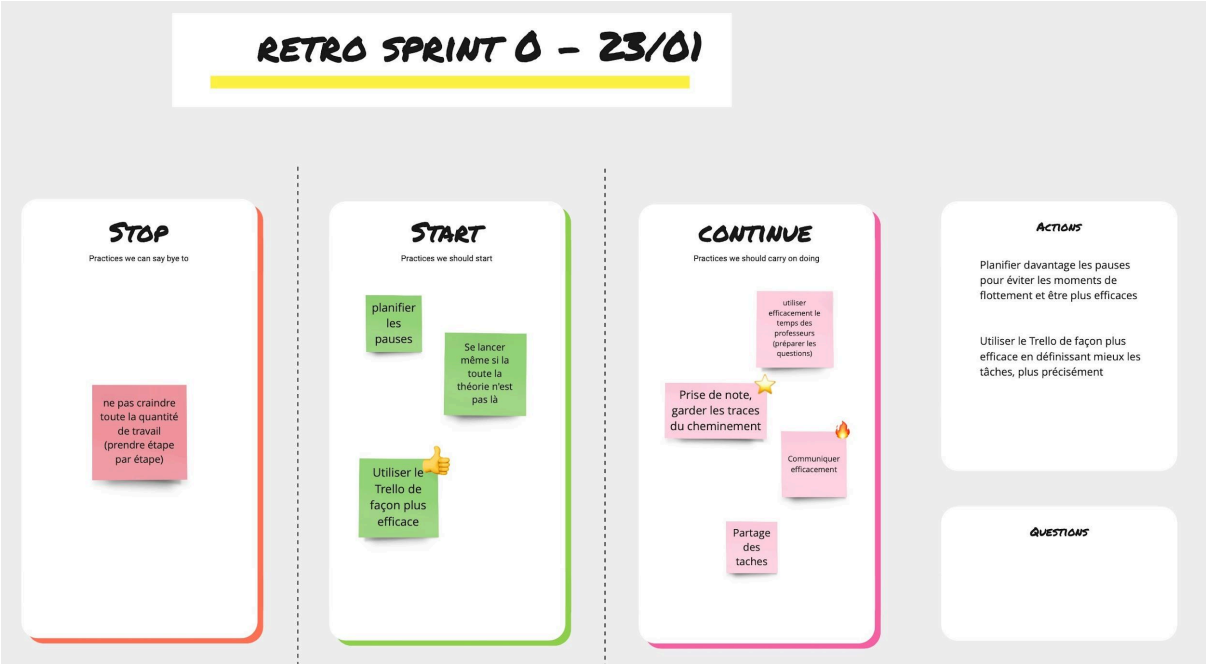
- Nouveau diagramme de cas d'utilisation avec nouveau use case : signaler un accident ou un problème dans l'infrastructure
- Début du diagramme de classe
- Diagrammes de séquence
- Choix des technologies que l'on va utiliser
- Début de MVP

Le Vendredi 12 Janvier 2024 :

- MVP fonctionnel sur le GitHub
- Préparation de la présentation
- Finalisation des différents rapports
- Finalisation des diagrammes (composants et classe notamment)

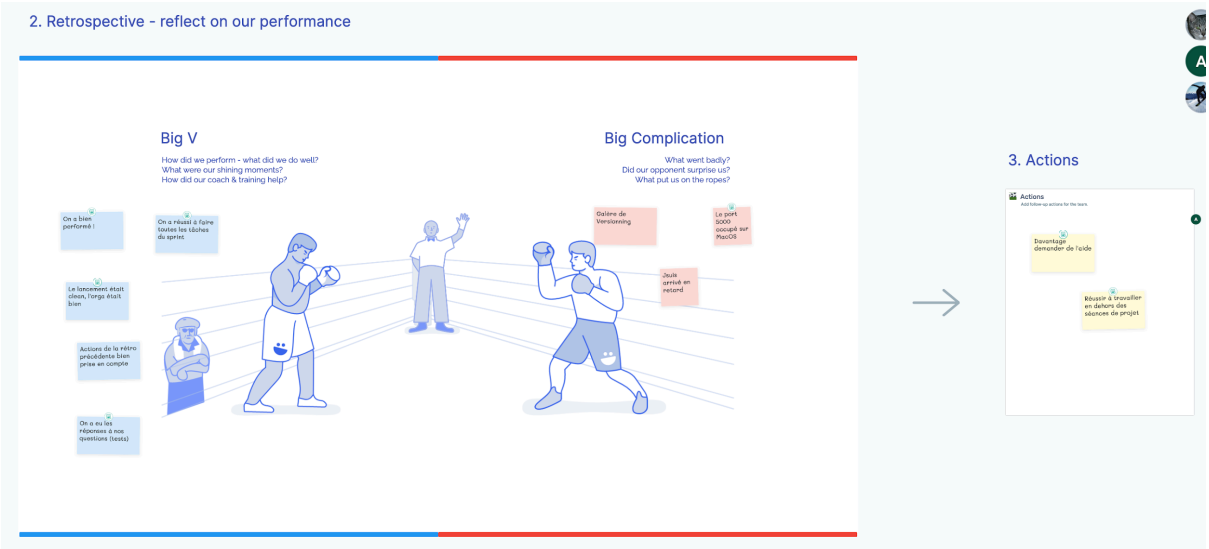
Le Mardi 23 Janvier 2024 :

- Rétrospective Sprint 0
- Redéfinition des User Story
- Planification du Sprint 1 : Ajout des tâches dans le Trello et quotation



**Sprint 1 :**

- Développement des premières features et organisation de l'architecture
- Test des premières features : Cypress, Jest, Pytest




Lien vers la rétro : https://metroretro.io/BO8UD0PQIDVW

**Sprint 2 :**

- Dockerisation
- Mise en place des signalements et du login

Lien vers la rétro : https://metroretro.io/BOU2PGQCBSIL

**Sprint 3 :**

- Mise en place des signalements et du login

**Sprint 4 :**

- Déploiement sur une VM de l'ISTIC
- Préparation de l'oral

# Rapport de test

Les tests de notre application ont été effectués sur les différents composants et avec différents outils. Nous allons développer les tests effectués pour chaque composant dans la suite.

## Tests Backend

Nous avons effectué des tests des endpoints de notre application pour vérifier le bon fonctionnement du backend. Ces tests ont été réalisés en utilisant le framework de test supertest pour Node.js.

## Tests Data Engineering

Pour garantir la qualité et la fiabilité de notre composant de data engineering, nous avons mis en place des tests automatisés à l'aide de Pytest. Ces tests vérifient la cohérence et l'intégrité des données manipulées par le composant.

## Tests Frontend

Nous avons utilisé Cypress pour effectuer des tests end-to-end sur l'interface utilisateur de notre application. Ces tests vérifient le bon fonctionnement des principales fonctionnalités accessibles par les utilisateurs. Initialement, les tests ont été réalisés avec succès lorsque l'application était en local. Cependant, après avoir dockerisé l'application, des problèmes ont été rencontrés lors de l'exécution des tests. Ces problèmes ont été attribués à des limitations matérielles et de configuration, rendant difficile l'exécution des tests dans un environnement Docker. Suite à des discussions avec notre product owner, il a été décidé de prioriser les fonctionnalités restantes du projet pour l'instant, en attendant une résolution de ces problèmes.

# Post-mortem

Au cours de ce projet, nous avons adopté les méthodes agiles pour le développement, ce qui a permis une répartition et une organisation efficaces des tâches. Grâce aux sprints et aux réunions de suivi, nous avons pu ajuster nos priorités et adapter notre planification en fonction des besoins et des obstacles rencontrés. Travaillant en équipe de trois, nous avons rencontré des contraintes, notamment un manque de temps et de personnel pour réaliser toutes les fonctionnalités initialement envisagées. En particulier, nous n'avons pas pu intégrer un module d'intelligence artificielle ni effectuer tous les tests souhaités. Malgré ces limitations, notre organisation a été suffisante pour aboutir à un produit fonctionnel.

Nous avons considérablement développé nos compétences en développement web, travaillant avec des technologies variées telles que React pour le front-end, Node.js et Flask pour le back-end. La mise en place de tests a également été un aspect crucial de notre processus de développement, bien que nous reconnaissons qu'il reste encore des

améliorations à apporter dans ce domaine. La mise en place de la base de données PostgreSQL a été une étape importante de notre projet car liée à la mise en place de Docker. De plus, nous avons réussi à déployer notre application sur une machine virtuelle, ce qui a permis de rendre notre produit accessible et opérationnel.

## Problèmes rencontrés

Bien que plusieurs objectifs aient été atteints, nous avons rencontré un certain nombre de problèmes techniques qui ont affecté notre progression et notre productivité. Dans cette partie nous détaillerons certains de ces problèmes.

Lors de l'intégration de Leaflet dans notre projet, nous avons rencontré des problèmes de compatibilité entre les versions des packages dépendants. Cela a conduit à des erreurs lors de l'exécution et a nécessité plusieurs itérations pour trouver une configuration stable.

L'absence de Docker pour la gestion de la base de données PostgreSQL a compliqué les configurations et les déploiements. Chaque membre de l'équipe devait configurer manuellement la base de données sur son environnement local, entraînant des incohérences et des pertes de temps. La mise en place de Docker tôt dans le projet nous a permis de résoudre ce problème.

L'exécution de tests Cypress dans des conteneurs Docker a souffert de limitations de mémoire, rendant alors compliqué les tests liés à la base de données.

La documentation sur l'utilisation des machines virtuelles fournies par l'ISTIC était insuffisante, ce qui a obligé les membres de l'équipe à solliciter de l'aide auprès de camarades qui avaient, eux, déjà contacté les administrateurs.

Les défis rencontrés durant ce projet ont mis en lumière plusieurs domaines nécessitant des améliorations, notamment la documentation des environnements de travail. Nous espérons que les leçons tirées de cette expérience contribueront à des projets plus fluides et plus réussis à l'avenir.