URBANIZE

urbanize.contact@gmail.com

1, rue de Kerampont
22300 — LANNION

# Documentation technique

## Projet Calc-Road

Rédigé le 30 novembre par :

- Baptiste PRIEUR
- Sylvan LE DEUNFF

# Table des figures



# Table des tableaux

# Architecture technique

Nous avons mis en place une architecture modulaire, séparant les fonctionnalités métiers, de l'interface utilisant ces fonctionnalités. Elles se découpent en deux parties, l'une est le coeur de l'application, nommée *Backend*, permettant d'assurer la communication et la sécurité de l'utilisation de l'outil. L'autre est l'interface finale pour l'utilisateur, nommée *Frontend*. Le backend s'appuie sur un package python que nous développerons, et qui contiendra les fonctionnalités nécessaires à la simulation.

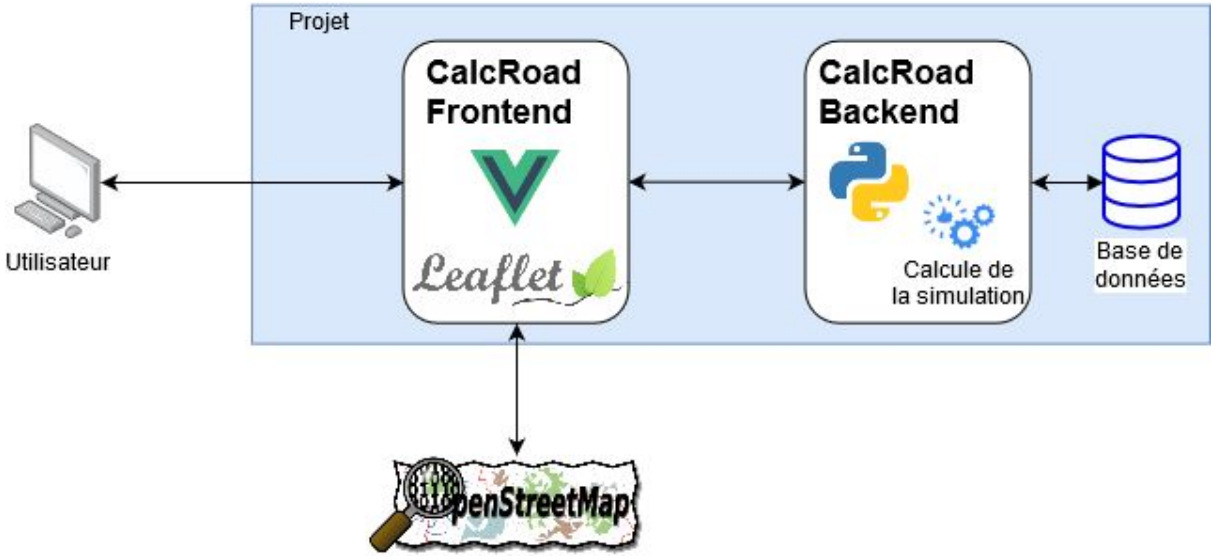



*Figure 1 : représentation fonctionnelle de l'application*

| | |
|---|---|
| Frontend | https://gitlab.com/urbanize/calcroad/ |
| Backend | https://gitlab.com/urbanize/backend |
| Simulation | https://gitlab.com/urbanize/simulation |

*Tableau 1 : Liste des liens publiques des différents projets prochainement détaillés*

# Le coeur de l'application

Dans cette partie, nous ne décrirons pas les fonctionnalités détaillées du backend (disponibles dans les lots qualification et simulation). Nous nous concentrerons sur les choix de technologies ainsi que la configuration possible de l'API.

## Backend

Le backend de l'application est écrit en *Python 3.6*.

### Packages utilisés

| | |
|---|---|
| Flask | Flask est un framework open-source de développement web en Python. Son but principal est d'être léger. Il permet de développer des applications WSGI*.<br><br>Pour les lancer nous faisons le choix d'utiliser le serveur WSGI **gunicorn**, mais il est possible d'opter pour n'importe quel serveur respectant cette norme.<br><br>***WSGI**: https://fr.wikipedia.org/wiki/Web_Server_Gateway_Interface |
| flask-restplus | Plugin pour Flask permettant de développer de structurer une application sous forme d'API en respectant les normes swagger (namespaces, models, resources, endpoints).<br>Il offre la possibilité de documenter directement la documentation à la racine de l'API sous forme d'une documentation SwaggerUI. |
| Flask-SQLAlchemy | SQLAlchemy est un ORM*. Il permet de stocker des objets python dans base de données (de préférence relationnelle). L'extension Flask-SQLAlchemy permet la gestion des sessions de connexion à la base de données dans un environnement multi-threads.<br><br>***ORM**: https://fr.wikipedia.org/wiki/Mapping_objet-relationnel. |
| Flask-Marshmallow | Marshmallow est une librairie permettant la sérialisation / désérialisation d'objets (ayant une structure complexe) en JSON. |
| Flask-Bcrypt | Flask-Bcrypt est une extension permettant un hashage fort des mots de passe des utilisateurs. Il est intentionnellement lent pour réduire les risques d'attaques par dictionnaire. |
| Flask-CORS | Plugin Flask permettant de gérer la politiques d'autorisations des requêtes provenant d'origines (domaines) tierces. |
| Flask-JWT-Extended | Plugin permettant de protéger des endpoints en utilisant des tokens de type JWT. Il offre |
| calcroad_simulation | Package développé par l'équipe Urbanize pour simuler les trajets des véhicules. |

## Configuration du backend

La configuration de l'application se fait dans le répertoire settings/. Nous avons défini pour l'instant 4 environnements à savoir

- development
- test
- integration
- production

Pour sélectionner un l'environnement de l'application. Il faut déclarer une variable d'environnement système FLASK_ENV avec pour valeur l'environnement choisi. Par défaut si elle n'est pas définie, l'environnement sera "development". Exemple sous linux :

```
export FLASK_ENV=development
```

Dans le répertoire settings, un fichier common comprend la configuration communes à chacun de ces environnements et un fichier de configuration plus spécifique pour chaque environnement. Les variables définies dans le fichier common seront écrasée par une variable de même nom qui serait défini dans un fichier d'environnement plus spécifique.

| | |
|---|---|
| **APP_CONTACT** | Email de l'équipe en charge de l'application |
| **APP_DESCRIPTION** | Description de la doc pour la doc Swagger |
| **APP_NAME** | Nom de l'application pour la doc Swagger |
| **APP_VERSION** | Chaîne représentant la version de l'application |
| **APP_SEND_WELCOME** | Définit s'il faut envoyer un mail de bienvenue aux utilisateurs. |
| **BCRYPT_LOG_ROUNDS** | Nombre de tours dans l'algorithme de hachage sha256.<br>(Utilisé pour le hachage des mots de passe) |
| **MAIL_XXX** | Configuration de Flask-Mail<br>(voir https://pythonhosted.org/Flask-Mail/) |
| **SQLALCHEMY_DATABASE_URI** | URI de la base de données de la base de données à utiliser<br>https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/ |
| **SECRET_KEY** | Chaîne de caractères complexe utilisée par les protocoles de chiffrement et algorithmes de hachage. |

Pour plus de configuration, se référer aux documentations des packages utilisés.

## Contribuer au projet

Votre environnement doit comprendre le langage *Python (>=3.6)*. Pour utiliser les commandes définies dans le fichier Makefile (recommandé), il faut également avoir installé le package *virtualenv* comme suit:

```
pip install virtualenv
```

Les commandes utilisables via la commande `make` sont les suivantes.

```
$ make help
install         (re)create virtualenv and install project dependencies
lazy-install    install project dependencies in existing virtualenv
lazy-test       run test using existing test virtualenv
serve           run project in debug mode
test            run a complete test
```

# Simulation

La simulation est distribuée sous forme d'un package *Python* qui est disponible à cette url: https://pypi.org/project/calcroad-simulation/

## Installation du package

Elle peut être installée via un gestionnaire de package comme pip en utilisant la commande `pip install calcroad_simulation`.

*Note: aucune action n'est requise pour l'utiliser dans le backend de l'application car la simulation y est déjà installée.*

## Déploiement d'une nouvelle version

Le déploiement d'une nouvelle version se fait en utilisant le package python twine. Il faut donc installer celui-ci au préalable. Pour cela

```
pip install twine
```

Pour en *déployer une nouvelle version*, il faut d'abord incrémenter le numéro de version se trouvant dans le fichier "*setup.py*" (configuration du package), puis exécuter les commandes:

```
python3 setup.py sdist bdist_wheel
python3 -m twine upload dist/*
```

*ou en utilisant le fichier Makefile*

```
make dist
```

Une authentification est requise.

| | |
|---|---|
| **Username** | urbanize |
| **Password** | urbanize_enssat_2019 |

*Tableau 2 : identifiants du compte pypi (https://pypi.org/)*

# L'interface utilisateur

Dans cette partie, nous ne décrirons pas les fonctionnalités détaillées du frontend (disponibles dans la partie suivante). Nous nous concentrerons sur les choix de technologies ainsi que les restrictions et les différents moyens d'adaptations.

## Versions compatibles

L'interface utilisateur est une interface web basé sur le Framework *Vue.js*. De ce fait, il est nécessaire que l'utilisateur utilise un navigateur web compatible. Voici la liste des navigateurs acceptés et supportés par notre application.

| IE* | Edge* | Firefox* | Chrome* | Opera* |
|---|---|---|---|---|
| **X** | **42.17134** et + | **65.0** et + | **76.0.3809** et + | **63.0** et + |

**tests des navigateurs non contractuels.*

*Tableau 3 : Liste des navigateurs compatibles*

## Aperçu des Frameworks

Plusieurs frameworks ont été utilisés pour aboutir à une application fonctionnelle. Notamment, *Vuetify (2.1.0)*, une librairie UI embarquant la charte graphique *Material Design* de Google. Également, *Leaflet (1.5.1)*, la librairie permettant de créer une carte, et interagir avec elle. Il y a des extensions à cette librairie, comme *Leaflet-Geoman (2.3.0)* et *Leaflet-Geosearch (1.0.6)* qui permettent respectivement d'éditer les objets créés et de centrer la carte à un lieu donnée. *OpenStreetMap* est utilisé pour récupérer le fond de carte.

## Contribuer au projet

Ce projet est versionné sur Gitlab afin d'assurer le développement, mais aussi le déploiement continu de l'application. On retrouve dans un premier temps, ces dossiers et fichiers pour valider et déployer l'application via une image Docker.

**Fichiers**

| | |
|---|---|
| *package.json* | Contient les commandes exécutés pour la création de l'application lors de sa construction, dont les frameworks cités précédemment et d'autres. |
| *Dockerfile* | C'est le fichier principal pour le déploiement continue. Il permet de construire, et d'embarquer tout le nécessaire pour le fonctionnement de l'application. |
| *.dockerignore* | C'est un fichier qui ignore les dossiers ou fichiers lourds afin de ne pas impacter et alourdir le container Docker lors de l'étape de construction. |
| *gitlab-ci.yml*. | C'est le fichier principale qui permet de déployer dans un hébergement Gitlab public ou privé, et d'exécuter la construction de l'application. |

*Tableau 4 : Liste d'une partie des fichiers du répertoire Frontend*

**Dossiers**

| | |
|---|---|
| *nginx* | Ici se trouvent les configurations NGINX utilisés par le container Docker. |

L'application, elle, est développé grâce au framework *Vue.js*. L'environnement de développement de notre application nécessite d'utiliser un serveur *JavaScript*. L'application a été développé et testé sur la version de *nodejs (v12.10.0)*.

**Dossiers**

| | |
|---|---|
| *src* | Ici se trouvent les fichiers et dossiers de l'application. On y trouve majoritairement des fichiers *Typescript* et *Vue*. |
| *public* | Ce dossier regroupe tous les fichiers non JavaScript, tels que les images ou le fichier *index.html*. |

*Tableau 5 : Liste d'une partie des dossiers du répertoire Frontend*

## Limitations et contournements

Comme indiqué précédemment, l'application utilise des framework pour son fonctionnement graphique. Nous avons donc dû adapter certaines fonctionnalités à nos besoins..

### La connexité des points lors d'un nouveau tracé

Pour le framework *geoman-leaflet*, lors de la création d'une route (aka. Polyline ou Line), nous avons besoin de pouvoir fermer un tracer sur lui-même, avec exactitude sur le point de celui-ci, ou sur la ligne entre deux points.

Ce comportement n'est pas disponible par le framework. Cependant, le comportement de placer un point sur les routes existantes existe.

Pour pouvoir placer un nouveau point sur la route qui est en cours d'être dessinée, et ainsi de créer ainsi nouvelles intersections, nous avons choisie de dessiner une nouvelle route temporaire en dessous de celle-ci. Ainsi, la route qui est tracé obtient la possibilité de créer une nouvelle intersection sur elle-même.

On trouve cette modification dans le fichier *EditMap.vue* (ligne 193 à 220) situé dans *src/components/map* du projet.

# Conception de l'interface utilisateur

Nous passons maintenant à la présentation de l'interface utilisateur. Il y a trois grandes parties distinctes de l'application CalcRoad.

## Connexion et enregistrement

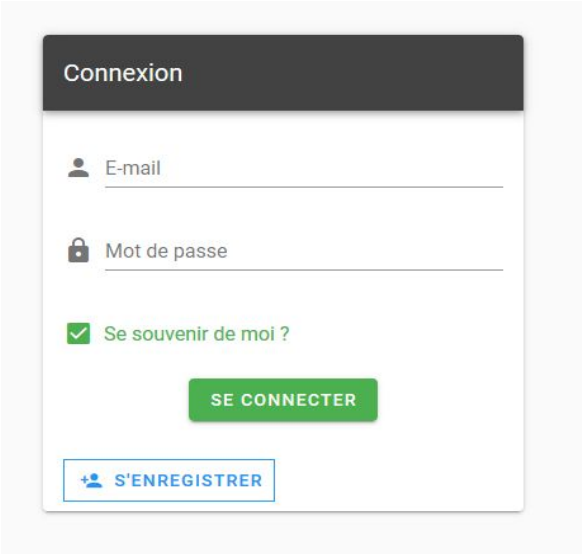



*Figure 2 : Page de connexion*

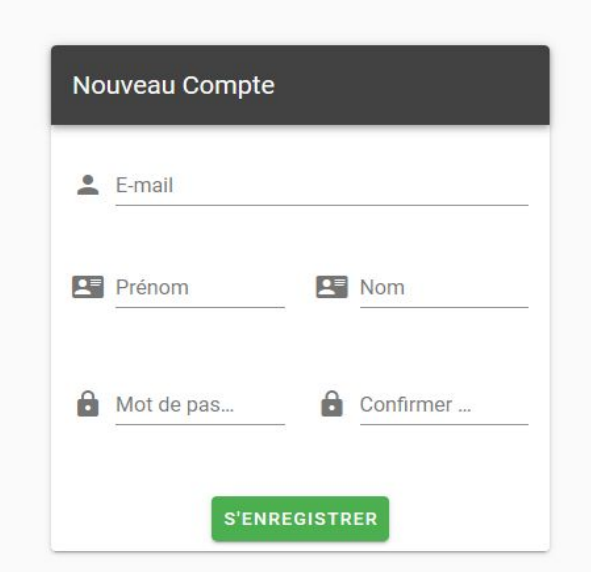



*Figure 3 : Page d'enregistrement*

L'utilisateur arrive sur la page web de l'application. Il est redirigé automatiquement sur la page de connexion (Figure 2).

Il a la possibilité de se connecter avec ses identifiants, ou bien de s'enregistrer avec un nouveau compte (Figure 3).

Après qu'il ait saisi les champs et appuyé sur le bouton, il est redirigé sur la page d'édition si la création ou la connexion a été un succès. Le cas échéant, un message d'erreur lui est notifié.

## Édition d'une carte


*Figure 4 : Interface de l'édition, aucune carte est sélectionnée.*

L'utilisateur peut maintenant utiliser l'application. Il a le choix entre Créer ou Ouvrir une carte à éditer. Ou bien, il faut accéder à la simulation de l'application.

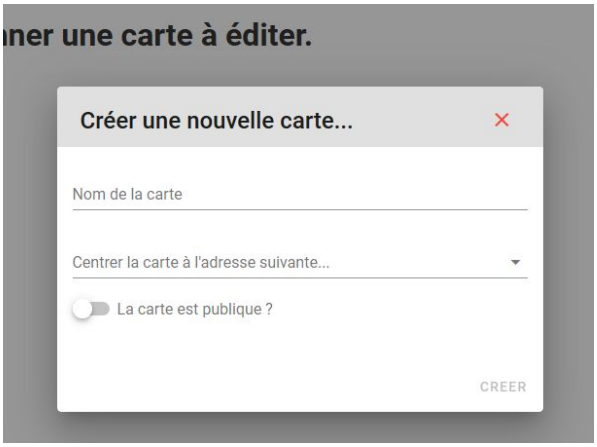



*Figure 5 : Créer une nouvelle carte*

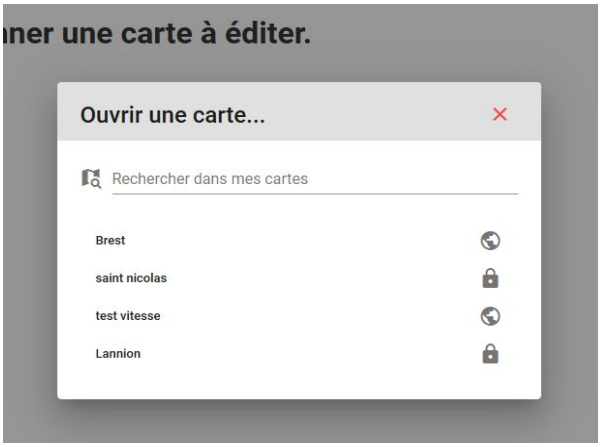
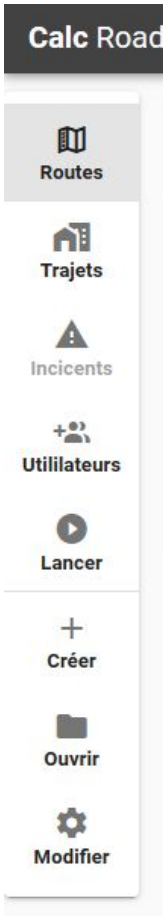



*Figure 6 : Ouvrir une carte*

Si l'utilisateur souhaite ouvrir une carte (Figure 6), il a la possibilité de sélectionner les cartes dont il a le droit d'écriture. Il a également la possibilité de chercher dans ses cartes grâce à une barre de recherche.

Si l'utilisateur souhaite créer une nouvelle carte, il lui suffit de rentrer un nom et une localisation pour centrer la carte au lieu indiqué.

Après qu'il ait sélectionné ou validé la création, il est redirigé sur l'interface d'édition d'une carte.




**Description respective des onglets :**

C'est un bouton qui permet à l'utilisateur d'afficher la liste des routes tracées, existantes et prises en compte durant la simulation.

C'est un bouton qui permet à l'utilisateur d'afficher la liste des trajets existants.

La notion d'incidents n'a pas encore été intégrée mais sera prochainement réalisée.

C'est un bouton qui permet au propriétaire de la carte d'ajouter ou de supprimer un utilisateur. Il peut également restreindre ou accorder des permissions.

C'est un raccourci pour lancer la simulation de la carte en cours d'édition.

C'est le bouton qui permet de créer une carte (Figure 5)

C'est le bouton qui permet d'ouvrir une carte (Figure 6)

C'est le bouton qui permet à l'utilisateur d'afficher les modifications de la carte, comme le nom de celle-ci.

*Figure 7 : Le panneau d'édition avec sa description.*

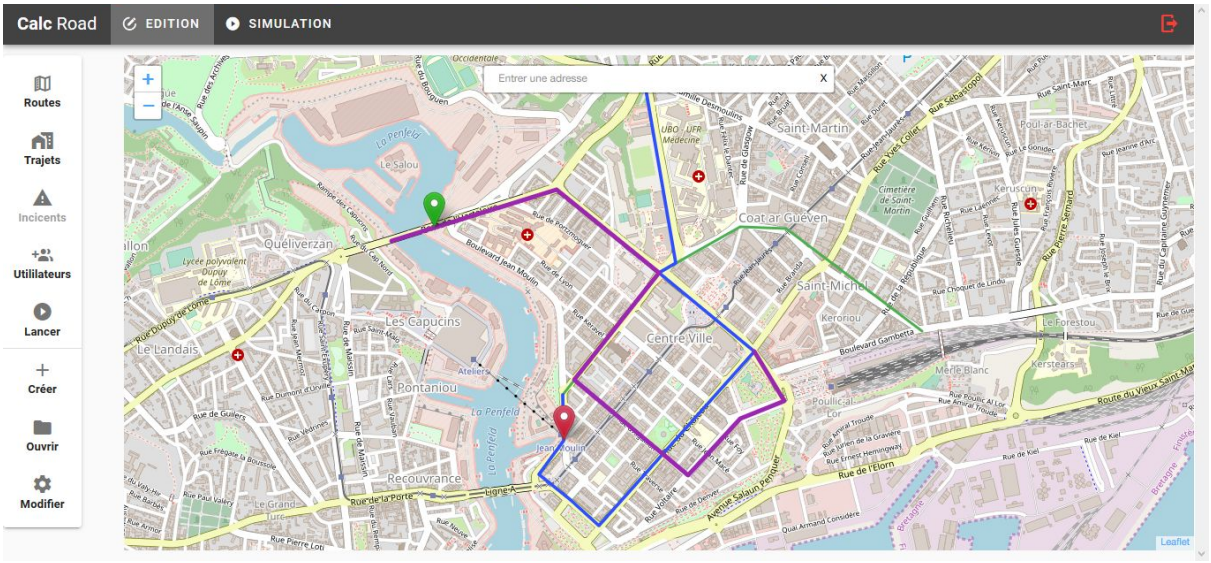



*Figure 8 : Vue d'ensemble de l'édition avec la carte et un exemple de routes quand aucun onglet n'est sélectionné*

L'utilisateur peut interagir avec la carte après qu'il l'ait sélectionné.

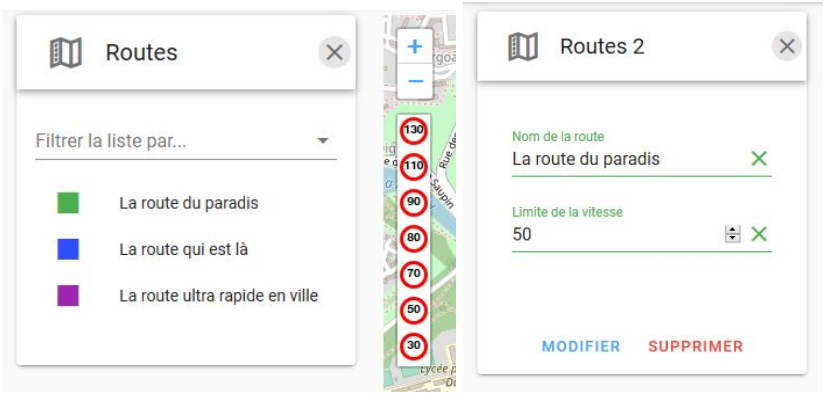



*Figure 9 : un exemple d'une liste des routes, et du détail d'une de ces routes.*

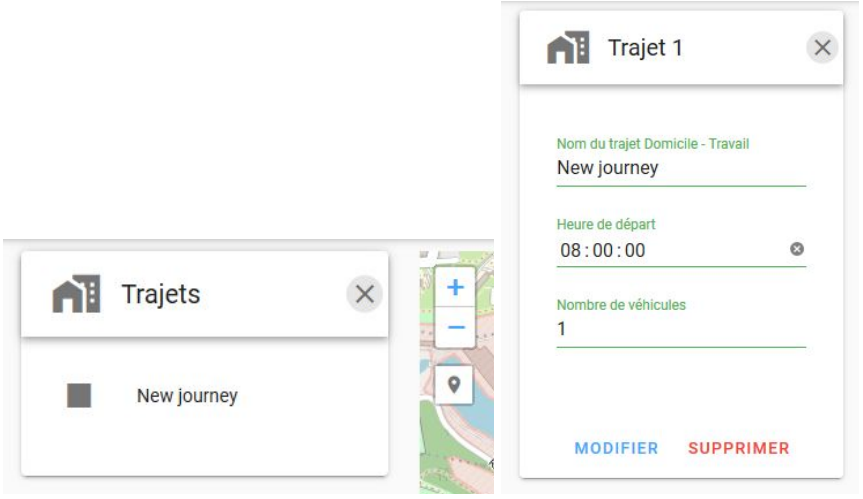



*Figure 10 : un exemple d'une liste des trajets, et du détail d'un de ces trajets.*

Les routes (Figure 8) et les trajets (Figure 9) dessinés par l'utilisateur sont affichés dans un panneau d'affichage comme ci-dessus. Il pourra cliquer sur la route ou sur le trajet dessiné sur la carte pour la modifier.

L'ajout de nouvelles routes ou d'un nouveau trajet se fait avec les icônes apparus sur la cartes en fonction du panneau (Routes ou Trajets) sélectionné.

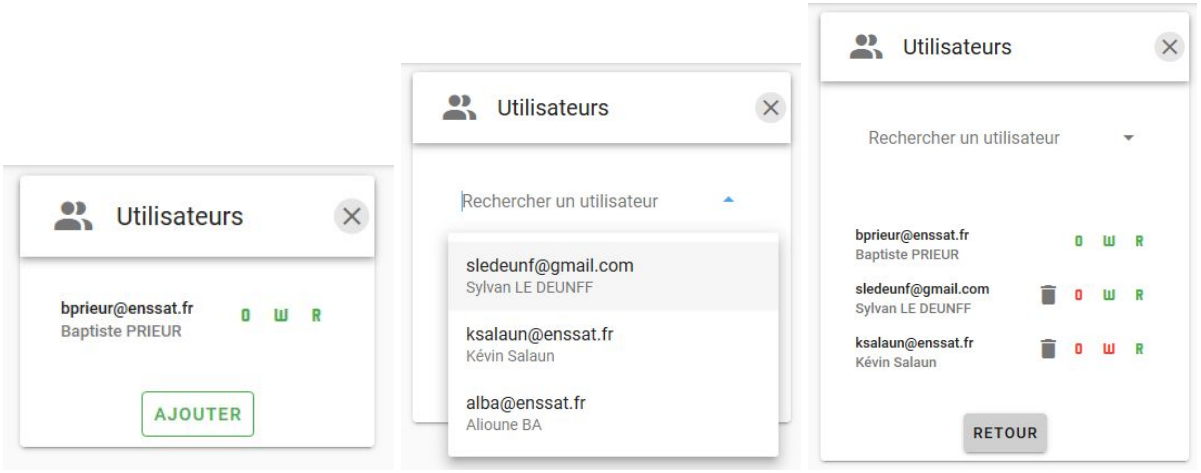



*Figure 11 : Liste des utilisateurs avant l'ajout de nouveaux utilisateurs à la carte.*

L'utilisateur peut également ajouter des collaborateurs à sa carte (Figure 10). Il suffit de rechercher l'adresse mail d'un des utilisateurs, puis de cliquer dessus pour l'ajouter. Une fois ajouté, on peut modifier ses permissions. Il peut également supprimer l'utilisateur.


*Figure 12 : Affichage des paramètres modifiables d'une carte*

Le dernier onglet d'édition permet de modifier les informations d'une carte (Figure 11) comme son nom.

# Simulation d'une ou plusieurs cartes

Si l'utilisateur clique sur le raccourci du menu d'édition, il sera redirigé sur la page de la simulation.

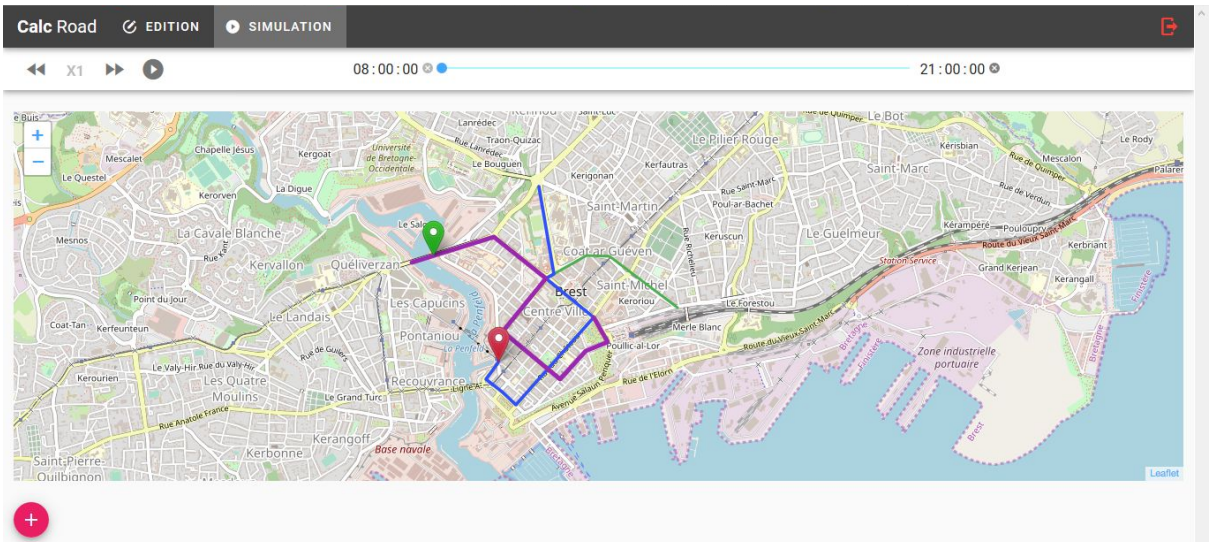



*Figure 13 : Affichage d'un exemple d'une simulation*

On peut voir en haut une barre de lecture pour démarrer la simulation (Figure 13). En partant de la gauche, il y a deux boutons permettant de contrôler la vitesse de lecture, suivi du bouton de lecture et de pause. On a enfin la barre de contrôle permettant de voir et de changer le temps actuel de la simulation en cours.

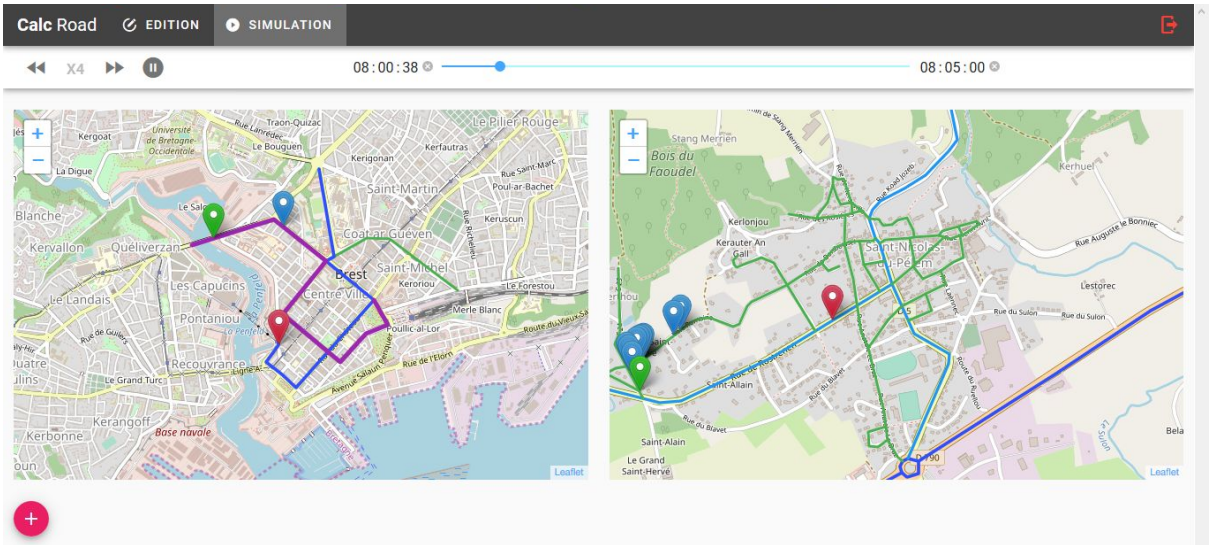



*Figure 14 : Affichage d'un exemple d'une simulation après 43 secondes*

On observe que la barre de lecture s'actualise au cours du temps de la simulation, ici accélérée quatre fois. Il y a également un bouton en bas de la page pour visualiser plusieurs simulations (Figure 14). Le bouton permet d'afficher les cartes disponibles (Figure 4).

# Déploiement

Pour faciliter l'hébergement du frontend et du backend de l'application, nous avons fait le choix de les déployer sous forme d'images docker. Ces images peuvent être testés localement en utilisant les commandes suivantes à la racine du projet :

*construction de l'image docker*

```
docker build -t <nom_image> .
```

*exécution de l'image précédemment construite*

```
docker run -p <port_machine_hote>:<port_interne_conteneur> <nom_image>
```

Ces images peuvent ensuite être stockés sur un registry docker (plateforme de stockage d'images) pour être rendues accessibles à quiconque souhaiterait déployer sa propre version de l'application.

Pour rendre automatique le processus de construction des images décrit précédemment, notre équipe utilise la chaîne d'intégration/déploiement (CI/CD) continu proposé par la plateforme GitLab. Pour cela un fichier "*gitlab-ci.yml*" défini les actions à effectuer lorsqu'une nouvelle version du projet est poussée. La configuration actuelle de la CI/CD est la suivante:

Quand un utilisateur pousse une nouvelle version du projet sur les branches master ou dev

- le jeu de test du projet est lancé pour assurer la non-régression de cette nouvelle version.
- une image docker est construite par le runner GitLab
- cette image est ensuite hébergée automatiquement dans le registry docker associé au projet (proposé par Gitlab).

Vous pouvez déployer ces images (et donc l'application CalcROAD) :

- Sur une plateforme de type PAAS* ou CAAS* publique comme AWS ou Heroku
- Dans un PAAS* ou CAAS* privé
- Sur votre propre serveur en l'exécutant directement