# Structure du projet

## Librairies

Le projet a été créé avec VueCli, avec comme options :

* VueJs 2
* Tests unitaires Jest
* Tests E2E avec Cypress
* VueRouter
* VueX

Les librairies suivantes ont été installées :

* bootstrap-vue (la configuration a été ajoutée dans main.js).
* axios
* leaflet
* vue2-leaflet

Les librairies de développement suivantes ont été installées :

* flush-promises
* jest-when
* json-server
* json-server-auth
* axios-mock-adapter
* concurrently
* @faker-js/faker
* proj4
* bcryptjs

## Les tests

Le dossier ./tests/unit n'est pas utilisé dans ce projet. Les tests doivent se retrouver dans les dossiers \_\_tests\_\_ des unités à tester. Par exemple : ./src/components/\_\_tests\_\_, ./src/router/\_\_tests\_\_, ./src/views/\_\_tests\_\_ et ./src/services/\_\_tests\_\_.

## Externalisation

Afin d'externaliser tout le texte qu'on retrouve dans les interfaces, un plugin d'externalisation (./src/externalization/uiTextPlugin.js) a été ajouté au projet. Pour en savoir plus sur son utilisation dans [Bonnes pratiques : Externalisation](https://ymazieres.gitlab.io/W40-H22/bonnes-pratiques/externalisation/).

## Variables d'environnement

Des fichiers de variables d'environnement, ont été créées afin de faciliter la maintenabilité du code.

## Script dans package.json

3 scripts ont été ajoutés au fichier package.json :

| **script** | **rôle** |
| --- | --- |
| backend | démarre l'API REST de développement |
| dev | exécute parallèlement le script backend et serve |
| generate | génère le fichier db.json qui est utilisé par l'API REST de développement |

# API REST (backend)

Un serveur local (API-REST), pour le développement, a été configuré dans le projet. L'API REST donne accès au fichier db.json.

### Fichier db.json

Le fichier db.json n'est pas présent avec le code de départ du projet. Il doit être généré : pnpm generate

* Dans le fichier généré db.json, l'utilisateur avec l'id 0 est considéré comme l'administrateur.
* Les données sur les utilisateurs sont générées aléatoirement.
* Le mot de passe de tous les utilisateurs générés est password.
* À noter que le fichier db.json n'est pas versionné et n'a pas à l'être (il se trouve dans le .gitignore).
* Un plus petit fichier nommé dbjson-small.json est présent dans le dossier backend, comme exemple de contenu.

### Routes et autorisations

* Des exemples de routes se trouvent dans /backend/requests/.
* L'autorisation des routes est gérée par le fichier routes.json. Ce fichier ne devrait pas être modifié. Il est utilisé par la librairie [json-server-auth](https://github.com/jeremyben/json-server-auth) qui elle permet d'ajouter l'authentification et l'autorisation aux routes de [json-server](https://github.com/typicode/json-server).

### Fichier sentieretel.json

* Emplacement dans le projet : /backend/input/sentieretel.json
* Ce fichier ne doit pas être modifié.
* Les coordonnées dans les fichiers sont dans le format **NAD 1983 Québec Lambert (EPSG:32198)**
* Les données sur les sentiers proviennent des jeux de données de [Données Québec](https://www.donneesquebec.ca/) et plus particulièrement de cette [source](https://www.donneesquebec.ca/recherche/dataset/sentiers-estivaux).

### Script generateDbJson.js

* Emplacement dans le projet : /backend/generate/generateDbJson.js
* Ce script ne doit pas être modifié. Il sert à générer le fichier db.json.
* La Librairie utilisée pour manipuler les différents formats de coordonnées géographique est [proj4js](https://github.com/proj4js/proj4js).
* Cet [outil](https://epsg.io/32198-1696) (voir dans le bas de la page) a été utilisé pour trouver la projection nécessaire par [proj4js](https://github.com/proj4js/proj4js) pour transformer les coordonnées **EPSG:32198** en **WGS84**.

# Gérer la carte

* [Vue Leaflet](https://vue2-leaflet.netlify.app/)
* [Vue Leaflet - LPolyline](https://vue2-leaflet.netlify.app/components/LPolyline.html)
* [Cette solution](https://github.com/vue-leaflet/Vue2Leaflet/issues/96) a été mise en place dans le code de départ pour corriger un bogue d'affichage.

# FAQ

## Authentification

### Comment faire pour accéder au id de l'utilisateur ?

Voir le fichier authentication.js du projet exemple-auth. Ce fichier utilise la classe tokenHelper.js.

### Est-ce que la connexion et la création d'un utilisateur peuvent se gérer sur une seule page ?

Oui, c'est possible avec le même principe que dans PostDetail.vue du projet exemples-vuex-blog pour l'ajout et la modification d'une publication.

### Est-ce qu'on doit créer le id de l'utilisateur dans le code VueJS ?

Non, le id est autogénéré par l'API REST. Dans l'exemple du fichier usersAndLikes.http (dossier /backend/requests), le id est présent lors de l'envoi de la requête, mais ce n'est que pour simplifier l'exemple. En réalité, ce id ne doit pas se retrouver dans la requête. Le id est créé par l'API REST et est présent dans le token. Pour manipuler un token, voir fichier tokenHelper.js du projet exemple-auth.

## Divers

### Est-ce qu'il y a un exemple de validation de formulaire ?

Oui, une validation de base avec l'attribut required de Bootstrap. Voir le formulaire dans Login.vue du projet exemple-auth et PostDetail.vue du projet exemples-vuex-blog.

### Comment faire pour accéder à un "getter" d'un autre module ?

Comme [dans cet exemple](https://vuex.vuejs.org/guide/getters.html#property-style-access), on peut ajouter le paramètre getters après state dans les paramètres du getter. Ce deuxième paramètre donne accès à tous les getters du module.

Pour avoir accès à tous les getters, modules confondus, il faut utiliser le quatrième paramètre rootGetters (voir [la liste des paramètres](https://vuex.vuejs.org/api/#getters)). La section des paramètres du getter sera donc (state, getters, rootState, rootGetters) et pour accéder à un getter d'un autre module l'instruction sera rootGetters['module/getterName'] comme dans [cet exemple](https://vuex.vuejs.org/guide/modules.html#accessing-global-assets-in-namespaced-modules).

### Faut-il calculer le centre du sentier pour le centrer sur la carte ?

Le centre de la carte peut être la première coordonnée du sentier. Par contre, si vous désirez relever un beau défi, vous pouvez essayer de calculer le vrai centre du sentier… mais à faire seulement si vous avez le temps. Sinon, utilisez un l-feature group pour mettre tous les l-polyline dans un div et ensuite prendre les bounds (zoom et center, essentiellement) de tout le sentier.

## Tests unitaires

### Est-ce qu'on peut utiliser le fichier db.json dans nos tests ?

Non, car certains tests ne passeront plus si le script "generate" est exécuté de nouveau. Les coordonnées gps et les sentiers resteront les mêmes, mais pas les données des utilisateurs (qui elles sont aléatoires). Il faut créer ses propres fichiers json pour les tests et les mettre dans /tests/units/data.

### Est-ce qu'on doit faire les tests du code provenant du projet exemple-auth ?

L'application doit être testée, mais vous n'avez pas à faire les tests unitaires des fichiers qui se trouvent dans le dossier /src/shared.

## API REST

### J'ai un problème avec le port de l'API REST

Assurez-vous que le port dans le fichier .env.development est celui de l'API REST (3000, dans le TP3).

### Lorsque je fais un DELETE d'un like, tous les likes associés à l'utilisateur s'effacent. Est-ce normal ?

Non, mais si c'est le cas, voici la raison : tous les ids dans les likes (trailId et userId) doivent être associés à une entité existante dans la db.json. Lors d'une requête HTTP DELETE, les likes qui ne respectent pas cette contrainte sont supprimés.

## Cypress

### Dans la section "before each", Cypress roule à l'infini ou me donne l'erreur "timed out waiting"

1. Est-ce que le backend (API REST ) est démarré ? Voir le script test:e2e dans package.json) et comparer avec celui de votre projet.
2. Voir la question ci-dessous : "Pour les tests avec Cypress, est-ce que je dois générer le fichier db.json à chaque fois ?"

### Pour les tests avec Cypress, est-ce que je dois générer le fichier db.json à chaque fois ?

Pour les tests d'interfaces, l'API REST ne devrait pas utiliser le fichier db.json qui a été est généré en début de projet avec "pnpm generate". Vous pouvez utiliser, comme dans l'exemple du fichier dbHelper.js de exemple-cypress, une méthode reset qui cré un fichier db.json utilisé par l'API REST. Mais, contrairement à l'exemple de la méthode reset, vous pouvez mettre les données désirées dans la constante data.

### Cypress n'utilise pas le bon port pour de l'API REST

Assurez-vous que le port dans le fichier .env.e2e est celui de l'API REST (3000, dans le TP3)

## Déploiement

### J'ai l'erreur 404

Assurez-vous que, sur GitHub dans settings, la source de votre page GitHub est la branche sur laquelle vous avez déployée (dans l'exemple le nom est gh-pages)

Aussi, le script deploy.sh ne s'exécute pas automatiquement. C'est vous qui devez l'exécuter sur votre ordi pour déployer le site. Si vous n'arrivez pas à exécuter le script deploy.sh, alors exécuté chacune des commandes manuellement.

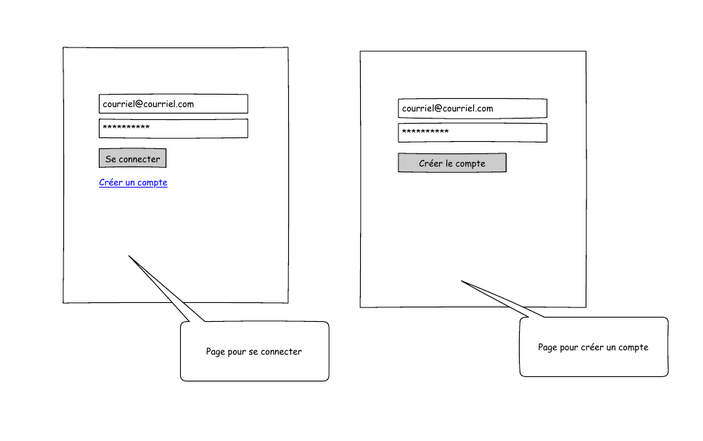
### J'ai une erreur avec la command git push

Vous devez [générer une clé ssh](https://docs.github.com/en/github-ae@latest/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent#about-ssh-key-generation) et ensuite [l'ajouter sur GitHub](https://docs.github.com/en/github-ae@latest/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account).

# Maquettes

Ci-dessous les maquettes des pages de l'application. Il n'y a aucune obligation à suivre ce design, mais l'utilisation de l'application doit rester intuitif. Soyez créatif !

## Connexion / Créer un compte



## Accueil

Une image contenant carte

Description générée automatiquement