Département de génie logiciel et des T.I.

Numéro du laboratoire	2
Étudiant(s)	Jimmy Mai Nick-Karl Chao Nikola Perotic Ai-Vi Nguyen
Code(s) permanent(s)	MAIJ09079505 CHAN91090005 PERN06049400 NGUA16569307
Cours	GTI-525 - Technologies de développement Internet
Session	Hiver 2022
Groupe	01
Date de remise	9 mars 2022

Introduction

Durant la première partie du laboratoire de GTI525, nous avions conçu une application web avec le but d'afficher des données plutôt simples sous forme de tableau. Toutefois, ce n'était que le début de l'application frontale. Maintenant, pour notre deuxième laboratoire, nous pouvons étendre la partie frontale tout en développant la partie dorsale afin d'implémenter des fonctionnalités additionnelles. Ces dernières incluent une présentation d'informations supplémentaires sur une carte interactive, une vue des statistiques de comptages de vélos par graphique ainsi qu'un API pour le traitement de requêtes AJAX et pour la recherche de compteur de vélo. Pour accomplir ces tâches, nous utiliserons quatre fichiers supplémentaires de données, compteur_stats_[année], qui sont de format CSV.

Évolution de l'architecture logicielle et de la structure du code

Avant tout, pour séparer la partie frontale de la partie dorsale de l'application, nous avons maintenant séparé le code en deux dossiers principaux: *client*, pour la partie frontale, et *server*, pour la partie dorsale. Cela permet principalement de différencier les fichiers sources ainsi que les fichiers de configurations JavaScript d'une partie à l'autre.

Du côté *client*, les dossiers source *assets*, *component* et *views* gardent pratiquement les mêmes fonctions et rôles qu'ils avaient lors du premier livrable. Précisément, *assets* comprend les feuilles de style ainsi que les fichiers de données compteurs.csv et fontaines.csv, *component* contient les balises et composantes qui constituent le *front-end*, alors que *views* possède encore certaines fonctions algorithmiques de tri et d'affichage de données. Dans ce deuxième laboratoire, en parallèle à ces trois dossiers se trouve maintenant le dossier *utils* contenant le fichier Services.js. Ce fichier nous permet d'encapsuler les routes spécifiques tout en composant le point d'accès de l'API /gti525/. Pour cela, on utilise la librairie Redaxios afin de faciliter les appels à l'API. Cela inclut toutes les requêtes entre le frontal et le dorsal. Sur ce sujet, le *front-end* est compilé par les ressources du serveur.

Du côté serveur, le *back-end* se repose sur le cadriciel Express.js qui nous permet d'établir les routes entre *client* et *server*. Ces routes communiquent les données des compteurs et des fontaines, du côté frontal au côté dorsal (et vice-versa). Autre que cela, le fichier parseData.js cherche, lit et filtre les statistiques des fichiers de format .csv du deuxième livrable de ce laboratoire avant qu'elles soient envoyées aux routes de l'API /gti525/ lors des requêtes du *front-end*. Puisque nous utilisons encore le cadriciel Vue.js qui offre une application web d'une seule page, un fichier unique *index* est servi par le dorsal. Pour afficher la bonne page sur le DOM, l'application emploie le paquet *vue-router* pour mettre à jour le DOM sans changer de page.

Implémentation de la couche modale et de la carte

L'affichage de la carte de compteurs vélos se résume à quatre fichiers *front-end* principaux: BikeCounterView.vue, MapModal.vue, MyModal.vue, et Map.vue, qui se retrouvent tous dans le dossier source. De plus, deux librairies importantes s'intègrent à l'application, c'est-à-dire vue-universal-modal et vue-leaflet.

Premièrement, la vue BikeCounterView contient la composante MapModal, puisque l'application se sert de la librairie vue-universal-modal. Cette dernière nous permet d'implémenter une modale contenant la carte qui sera utilisable avec Vue 3. BikeCounterView communique donc à MapModal les coordonnées (id, nom, longitude et latitude) de tous les compteurs ainsi que celles du compteur sélectionné manuellement par l'utilisateur.

Deuxièmement, MyModal contient un *slot* dans lequel MapModal insère le gabarit Map. Ce dernier est un composant séparé qui facilite la réutilisation et l'organisation du code. Ainsi, au besoin, un autre modal peut être inséré dans le *slot* pour un nouvel affichage. Comme mentionné précédemment, on implémente vue-universal-modal pour sa compatibilité et sa simplicité à personnaliser.

Troisièmement, le fichier MapModal est l'instance spécialisée de MyModal qui est appelée par BikeCounterView lorsque le pointeur rouge d'un compteur est sélectionné. Il s'occupe de filtrer la liste de compteurs reçue pour y ajouter une couleur de marqueurs (rouge pour celui sélectionné et bleu pour tous les autres) et un attribut pour indiquer si c'est le compteur sélectionné. Ces ajouts vont aider à l'affichage des lieux dans Map.

Quatrièmement, Map contient tous les éléments visibles qui constituent la carte dans le modal: l'image de la carte, le bouton pour fermer le modal, les marqueurs et les indices contextuels. Ce gabarit va afficher un marqueur pour chaque compteur en utilisant leurs coordonnées (latitudes et longitudes) dans la couleur attribuée en plus de créer un *popup* qui va afficher leurs noms. Il va ensuite créer le marqueur et le *popup* pour le compteur sélectionné. C'est ainsi que dès l'ouverture du modal, le marqueur sélectionné apparaît avec son nom. Les autres noms de marqueurs apparaissent si on les sélectionne par la suite. Ce composant entier requiert l'usage de la librairie *open-source* Leaflet, conçue pour rapidement intégrer une carte interactive contenant des éléments personnalisables, comme les marqueurs. Pour être plus exacts, nous implémentons la librairie vue-leaflet, qui emballe Leaflet en la rendant compatible avec Vue 3, facilitant donc son utilisation pour notre application web. Finalement, Leaflet contient une fonction *fitBounds* qui prend en paramètre une liste de coordonnées et un *padding* désiré. C'est avec celle-ci que l'affichage des points apparaît dans les limites de la map.

Cadriciels de l’application dorsale et de l’API

Le cadriciel majeur intégré au côté dorsal de notre application est Express. C’est un cadriciel de l’environnement Node.js qui permet au projet de principalement garder le code en JavaScript. Avec nos aptitudes en ce langage, le développement d’un *back-end* pour un projet de notre grandeur requiert donc moins d’effort et de complications en employant Express. En effet, l’implémentation des routes et de l’API reste simple à coder et facile à réutiliser.

Parallèlement, une librairie à souligner serait l’emballeur Redaxios. On l’utilise pour sa compatibilité avec notre compilateur et groupeur de modules Vite.js l’outil que nous employons pour *build* l’application frontale. Redaxios est une librairie beaucoup plus légère et petite que Axios. Elle utilise l’API Fetch du navigateur pour faire ses requêtes HTTP tout en simplifiant certains fonctions en utilisant la même syntaxe que Axios.

Distributions des tâches

Les tâches ont été attribuées en fonction des habiletés techniques en programmations dorsale et frontale. Ce deuxième livrable offre une plus vaste diversité de tâches, signifiant qu’il est plus facile de diviser le mandat dépendamment des connaissances de chacun. La distribution nous a permis de mieux adapter l’implémentation de chaque mandat du laboratoire. Par exemple, avec son expérience, Nikola a pu s’assurer que les nouvelles fonctions et librairies conçues pour l’application frontale concordent bien avec Vue. Il a aussi suggéré l’emploi de Redaxios pour sa compatibilité avec Vue et Vite.

Jimmy	Jimmy s’est occupé de la nouvelle implémentation du côté dorsal de l’application communicant par requêtes de type AJAX, principalement en raison de son expérience plus avancée en programmation dorsale.
Ai-Vi	Ai-Vi avait un grand intérêt pour continuer de travailler sur la partie graphique et visuelle du *front-end*, alors elle a implémenté l’affichage de la modale et de la carte interactive. De plus, elle a ajouté les éléments manquants lors de la première itération.
Nick-Karl	Autre que la mise en page et correction du rapport de laboratoire, Nick-Karl a travaillé sur le *back-end* en paire avec Jimmy.
Nikola	Étant donné ses connaissances, Nikola a fortement assisté à l'implémentation des côtés frontal et du dorsal, notamment avec la compréhension générale de Vue, la recherche et implémentation des librairies, et l'optimisation du code.

Conclusion

Pour résumer, notre application web est désormais plus complète maintenant qu'un côté dorsal a été additionnellement implémenté. Parallèlement, l'ajout d'une carte ainsi qu'un graphique pour le comptage de vélos nous permettent à présent de plus clairement et facilement visualiser les statistiques fournies. Sur ce sujet, nous croyons que l'information à analyser aurait pu être sous un différent format qu'un fichier de format CSV. Par exemple, au lieu de "manuellement" lire un fichier, nous pourrions utiliser et accéder à une base de données afin d'offrir une plus grande flexibilité et maintenabilité au site web livré. De plus, nous n'avons actuellement aucune implémentation de système de sécurité pour protéger notre application. Malgré que ce n'était pas une priorité lors de l'ajout de fonctionnalités initiales pour les deux côtés de l'application, il serait logique de maintenant y intégrer une gestion des requêtes HTTP par jetons ou par *cookies*. De cette façon, nous assurons l'intégrité de notre application web pour le futur.