

Rapport PTI

Covélo tage



Réalisé par

BEN RAIES Sabri
DUBOIS Christophe

Spécialité : **Systemes Numériques**
Promotion : **2024**

Année Universitaire : 2023 / 2024

Table des matières

Table des matières	1
Liste des figures	3
Contexte général du projet.....	5
I. Contexte du projet	6
II. Présentation de l'organisme d'accueil.....	6
III. Présentation du projet.....	7
1. Problématique :.....	7
2. Objectifs du projet :.....	7
3. Phases du projet :.....	7
IV. Conclusion	8
Analyse et spécifications	9
I. Ressources logicielles et Frameworks	10
1. Gestionnaire de version Git :.....	10
2. React JS :.....	11
3. Node JS	11
4. Express JS	12
5. Flask :.....	12
6. OpenRouteService.....	13
7. Bootstrap.....	14
8. MongoDB	14
9. MongoDB Compass	15
II. Conclusion.....	15
Recherche et Développement	16
I. Algorithme de recherche du chemin le plus court :.....	17
1. Architecture :.....	17
2. Choix d'OpenRouteService	17
3. Configuration de l'OpenRouteService	18
4. Analyse de la Réponse JSON de l'API	20
5. Script Python Backend	20
6. Frontend Web	22
7. Interaction Python et Page Web via Flask.....	23
8. Extrait Explicatif du Code de la page WEB:	24
II. Algorithme de correspondance de chemin :	26
1. Introduction à l'algorithme LCS :.....	26
2. Étapes de l'Algorithme LCS :.....	26
3. Exemple illustratif :.....	27
4. Importance de LCS dans l'Identification des Segments Partagés :.....	28
5. Calcul du coefficient de similarité :	28
6. Implémentation :.....	29
III. Hébergement local de l'API OpenRouteService :	33

1.	Objectif de l'Hébergement Local de l'API	33
2.	Étapes :	33
3.	Configuration Initiale :	34
4.	Démarrage de Docker et Lancement de l'Instance OpenRouteService :	34
5.	Test Initial:	34
6.	Personnalisation des itinéraires :	35
7.	Test de la Nouvelle Configuration	36
IV.	<i>Bootstrap</i>	36
1.	Introduction :	36
2.	Intégration de CSS avec Bootstrap	36
3.	Illustration par des Captures d'Écran :	38
Conception et Développement du Site		43
I.	<i>Généralités</i>	44
II.	<i>Technologies</i>	44
1.	React JS.....	44
2.	Node.js	47
3.	Express JS	48
4.	MongoDB	49
III.	<i>Arborescence du projet</i>	49
1.	Frontend.....	49
2.	Backend	50
IV.	<i>Gestion des comptes utilisateur</i>	50
1.	Modules frontend récurrents.....	51
2.	Création de compte.....	52
3.	Connexion	57
4.	Modification du profil utilisateur	61
5.	Récupération de mot de passe.....	62
V.	<i>Gestion des trajets</i>	65
1.	Carte interactive.....	65
2.	Création de trajet	66
3.	Sauvegarde des trajets	68
4.	Chargement des trajets.....	69
5.	Correspondance des trajets	70
Conclusion générale et perspectives :		73
Bibliographie :		74

Liste des figures

Figure 1.1 : Une des entrées du technopôle de Brabois, côté hippodrome de Brabois.

Figure 2.1 : Système de contrôle de version distribué GIT

Figure 2.2 : Logo de React JS

Figure 2.3 : Logo Node JS

Figure 2.4 : Logo Express JS

Figure 2.5 : Logo Flask

Figure 2.6 : Logo OpenRouteService

Figure 2.7 : Logo Bootstrap

Figure 2.8 : Logo de MongoDB

Figure 2.9 : Interface MongoDB Compass

Figure 3.1 : Page Web de test pour l'algorithme du chemin le plus court

Figure 3.2 : Affichage de l'algorithme de correspondance

Figure 3.3 : Amélioration Design page d'accueil

Figure 3.4 : Amélioration Design Page de login

Figure 3.5 : Amélioration Design page Password

Figure 3.6 : Amélioration Design page Profile

Figure 3.7 : Amélioration Design page Register

Figure 3.8 : Amélioration Design page Recover

Figure 3.9 : Amélioration Design Page de Reset

Figure 4.1 : Architecture d'une application web moderne avec MERN Stack

Figure 4.3 : Fenêtre pop-up

Figure 4.4 : Page /Register

Figure 4.6 : Document utilisateur MongoDB

Figure 4.7 : Création d'une adresse mail avec Ethereum

Figure 4.8 : Page /login complétée

Figure 4.9 : Page /password

Figure 4.10 : Page /profile

Figure 4.11 : Page /Recovery

Figure 4.12 : Mail Ethereum

Figure 4.13 : Page /Reset

Figure 4.14 : Carte Interactive avec Leaflet

Figure 4.15 : Affichage d'un Trajet sur la carte

Figure 4.16 : Création et sauvegarde des trajets

Figure 4.17 : Document route MongoDB

Figure 4.18 : Chargement des Trajets

Figure 4.19 : Affichage des Trajets correspondants

1

Contexte général du projet

Ce premier chapitre commence par présenter le contexte de notre projet. L'organisme d'accueil au sein duquel nous avons effectué le projet. Ensuite, nous poserons la problématique tout en présentant l'objectif du projet et sa solution proposée. Nous décrirons aussi les méthodes de travail adaptée pour planifier ce dernier, le déroulement du projet pour finir avec une conclusion.

I. Contexte du projet

Le projet de développement du site internet vise à promouvoir le covélochage, une initiative visant à rendre l'utilisation du vélo plus accessible au quotidien en offrant des solutions aux problèmes de sécurité routière (en utilisant les trajets les plus utilisés et les plus sûrs) tout en encourageant les nouveaux cyclistes grâce à la mise en relation avec d'autres utilisateurs partageant des itinéraires similaires.

II. Présentation de l'organisme d'accueil

Le Technopôle de Nancy-Brabois est un important pôle technologique situé dans les communes de Vandœuvre-lès-Nancy et de Villers-lès-Nancy, au sud-ouest de la ville de Nancy, en France. Il a été aménagé à la fin des années 1970 et constitue l'un des premiers technopôles français, rejoignant ainsi d'autres centres tels que Sophia Antipolis près de Nice, Villeneuve-d'Ascq près de Lille et Inovallee près de Grenoble.

Ce technopôle a été créé dans le but de stimuler l'innovation, la recherche et le développement technologique en favorisant la collaboration entre les entreprises, les institutions académiques et les centres de recherche. Il s'étend sur une superficie importante et abrite des entreprises, des laboratoires de recherche, des institutions académiques, et diverses structures dédiées à la promotion de l'innovation.

Le Technopôle de Nancy-Brabois joue un rôle crucial dans le dynamisme économique et technologique de la région. Il favorise les synergies entre les entreprises et les institutions académiques, créant ainsi un environnement propice à l'émergence de nouvelles technologies et à la croissance des entreprises innovantes. Ce site contribue également à renforcer la compétitivité régionale en attirant des talents, des investissements et en favorisant le transfert de connaissances entre le monde académique et industriel.

En résumé, le Technopôle de Nancy-Brabois est un hub technologique majeur en France, jouant un rôle essentiel dans le développement économique et scientifique de la région.



Figure 1.1 : Une des entrées du technopôle de Brabois, côté hippodrome de Brabois.

III. Présentation du projet

Le projet consiste en l'élaboration d'un prototype sous la forme d'un site internet qui permettrait à des personnes novices, et peu habituées aux déplacements à vélo au sein de la Métropole du Grand Nancy, d'entrer leur itinéraire et d'être mis en relation avec des cyclistes plus aguerris ayant le même trajet.

Ce site WEB permettrait également de proposer d'autres services tels que de l'information sur le parcours (parkings, points d'eau, outils en libre-services, etc..) ou de faire le trajet seul sur le conseil des itinéraires les plus utilisés.

1. Problématique :

La principale problématique du projet est le développement d'un site WEB qui propose l'itinéraire le plus adapté possible entre celui demandé par l'utilisateur novice et les trajets disponibles dans la base de données, en utilisant un algorithme de correspondance.

2. Objectifs du projet :

Les objectifs du projet sont les suivants :

- Développer un algorithme permettant de rechercher le plus court chemin entre deux points d'itinéraires.
- Développer un algorithme permettant la correspondance des trajets entre utilisateurs.
- Développer un site WEB contenant les fonctionnalités suivantes :
 - Possibilité de créer un profil utilisateur et d'interagir avec d'autres utilisateurs.
 - Permettre la saisie d'un itinéraire sur une carte en ligne à partir de coordonnées GPS
 - Affichage des informations pertinentes pour les cyclistes le long du trajet, telles que les emplacements de parkings, les points d'eau, les stations d'outils en libre-service, etc.
 -

3. Phases du projet :

Le projet se déroulera en trois phases distinctes :

Phase 1 : Recherche et Développement

- Effectuer des recherches pour déterminer la méthode de géolocalisation, la recherche du plus court chemin et la correspondance des trajets.
- Élaborer des méthodes algorithmiques pour comparer les trajets demandés par les utilisateurs novices aux données de la base de données.
- Identifier des critères de correspondance et de sélection du trajet optimal.
- Mettre en place un système de géolocalisation via GPS ou entrée manuelle des utilisateurs.
- Concevoir un système d'affichage des itinéraires.
- Tester et valider l'algorithme de correspondance.

Phase 2 : Conception et Développement du Site

- Définir les fonctionnalités du site internet, y compris la planification d'itinéraires, la recherche d'utilisateurs expérimentés, et l'affichage d'informations pertinentes.
- Créer des maquettes du design et de la structure du site internet.
- Soumettre les maquettes au client pour validation.
- Développer le site web en utilisant les technologies appropriées.
- Mettre en place les bases de données nécessaires pour stocker les informations des utilisateurs et les données de trajet.
- Permettre au client d'assurer la maintenance des bases de données.
- Tester la sécurité du site et mettre en place des mesures de protection des données.

Phase 3 : Intégration des Fonctionnalités et Tests

- Intégrer les fonctionnalités permettant aux utilisateurs de créer des profils et de communiquer entre eux.
- Mettre en place les fonctionnalités définies lors de la phase 1, notamment la correspondance des trajets.
- Ajouter d'autres fonctionnalités jugées nécessaires pour améliorer l'expérience utilisateur.
- Effectuer des tests approfondis pour garantir le bon fonctionnement du site internet.
- Assurer la formation du client sur l'utilisation du site.
- Valider le bon fonctionnement du site avec le client.

IV. Conclusion

Ce chapitre a présenté l'organisme d'accueil Technopôle de Nancy-Brabois et son secteur d'activité. Ensuite il a décrit la problématique, les objectifs généraux du projet. Finalement, il a mis le point sur les phases suivies pour le développement de ce projet. Le chapitre suivant décrira l'étude préliminaire pour mettre le projet dans son contexte.

2

Analyse et spécifications

Dans cette section, nous nous penchons sur les logiciels et frameworks clés utilisés dans notre projet, offrant une vue d'ensemble de chaque technologie et son rôle dans la réalisation de nos objectifs. De React JS à Node.js, en passant par Flask, Bootstrap, et OpenRouteService, nous explorons la contribution unique de chaque outil au développement et à la performance de notre solution. Cette analyse vise à mettre en lumière la synergie entre ces technologies avancées et leur impact direct sur l'efficacité et la réussite de notre projet, soulignant ainsi l'importance d'une sélection judicieuse des outils dans le processus de développement logiciel.

I. Ressources logicielles et Frameworks

Cette partie contient les ressources logicielles utilisées dans ce projet :

- Gestionnaire de version Git
- React JS
- Node JS
- Express JS
- Flask
- OpenRouteService (ORS)
- Bootstrap
- MongoDB

1. Gestionnaire de version Git :

Le gestionnaire de version est largement utilisé aujourd'hui par les développeurs. Cela tient tout d'abord à sa capacité à archiver et à conserver les différentes étapes de développement d'un projet. Il offre aussi l'avantage de revenir à une version antérieure à tout moment. Il est également possible de consulter les différences entre les révisions. Enfin, il est beaucoup trop utilisé pour le travail collaboratif. Chaque développeur a un projet local et peut pousser des modifications vers le référentiel principal à tout moment afin que d'autres développeurs puissent bénéficier de leur travail. Ce processus est illustré sur la figure 3.5.

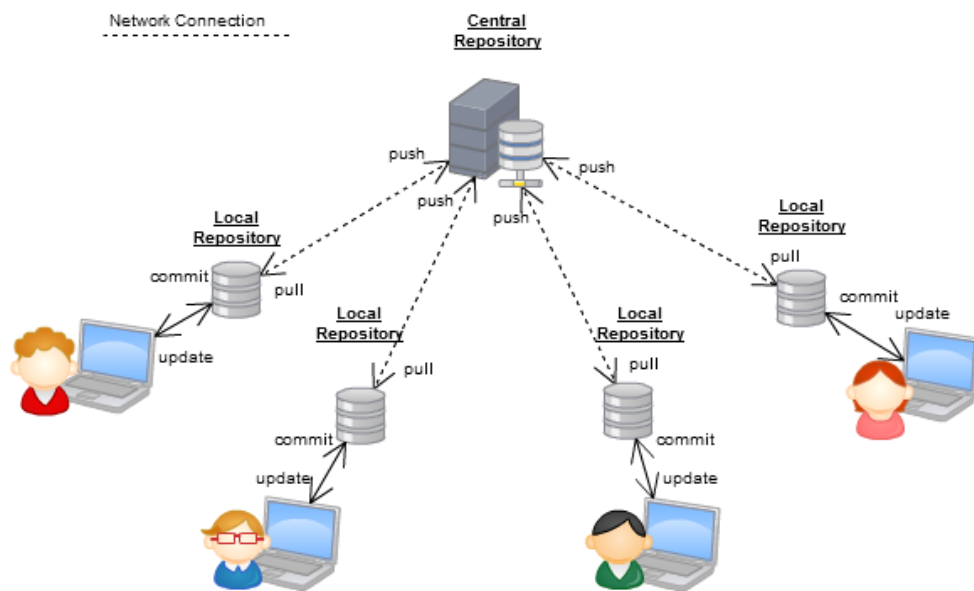


Figure 2.1 : Système de contrôle de version distribué GIT

Git a été créé par Linus Torvalds en 2005 pour le développement du noyau Linux, et d'autres développeurs du noyau ont contribué à son développement initial. Depuis 2005, Sumio Hamano en est le superviseur en

chef. Comme la plupart des systèmes de contrôle de version distribués, contrairement à la plupart des systèmes client-serveur, chaque répertoire Git sur chaque machine est indépendant du serveur, accessible par le réseau ou centralisé, avec un historique complet et des capacités de suivi des versions complètes. Git est un logiciel open source gratuit distribué sous la licence publique générale GNU version 2. La liste suivante décrit les principales fonctionnalités de Git.

- Faire un historique des modifications apportées au fichier, comparer les versions, et revenir à la version précédente si nécessaire.
- Création d'une ou plusieurs branches indépendantes pouvant être développées en parallèle de la branche principale (branche master) qui contient une version validée du code.
- La possibilité de diviser les modifications en parties distinctes. Cette option est très utile si vous souhaitez corriger l'erreur lors de l'implémentation d'une nouvelle fonction.

2. React JS :

React JS est une bibliothèque JavaScript de premier plan pour la construction d'interfaces utilisateur. Développée et maintenue par Facebook (maintenant Meta Platforms, Inc.), React JS a révolutionné la manière dont les développeurs créent des applications web interactives et dynamiques. Grâce à son modèle basé sur les composants, React permet aux développeurs de construire des interfaces utilisateur complexes et réactives avec une efficacité remarquable. Un des principaux avantages de React est son système de Virtual DOM, qui optimise les mises à jour de l'interface utilisateur en minimisant les manipulations du DOM réel, rendant ainsi l'application plus rapide et plus réactive. En outre, React JS favorise le développement rapide d'applications grâce à sa large communauté de développeurs et à un écosystème riche en bibliothèques complémentaires. Son approche déclarative rend le code plus prévisible et plus facile à déboguer, ce qui améliore significativement l'expérience de développement. La popularité croissante de React JS auprès des entreprises de toutes tailles témoigne de sa robustesse, de sa flexibilité et de sa capacité à répondre aux exigences des projets web modernes.

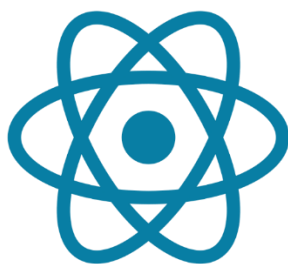


Figure 2.2 : Logo de React JS

3. Node JS

Node.js est une plateforme logicielle révolutionnaire pour le développement côté serveur utilisant JavaScript, un langage traditionnellement réservé au développement frontal (front-end) des applications web. Sa particularité réside dans son moteur d'exécution basé sur Chrome V8, qui permet d'exécuter le code JavaScript directement sur le serveur, offrant ainsi une performance exceptionnelle grâce à une exécution non bloquante et orientée événements. Cette caractéristique fait de Node.js une solution idéale pour le

développement d'applications web à haute performance, notamment pour gérer des opérations d'entrée/sortie intensives et dans des contextes en temps réel. En outre, Node.js favorise la productivité des développeurs en leur permettant d'utiliser le même langage de programmation côté client et serveur, facilitant ainsi le développement d'applications full-stack JavaScript. Grâce à son écosystème riche, incarné par le gestionnaire de paquets npm, Node.js offre une vaste bibliothèque de modules open-source, permettant aux développeurs d'ajouter facilement des fonctionnalités à leurs applications. La popularité croissante de Node.js au sein de la communauté de développeurs témoigne de sa robustesse, de sa flexibilité et de son efficacité dans la construction de solutions web modernes et évolutives.



Figure 2.3: Logo Node JS

4. Express JS

Express.js est un framework web rapide, non-opinionated et minimaliste pour Node.js, largement reconnu pour sa capacité à faciliter le développement rapide d'applications web et d'API. En offrant une couche d'abstraction élégante sans masquer les fonctionnalités essentielles de Node.js, Express simplifie la gestion des routes, des requêtes, et des réponses, rendant le développement backend plus intuitif et efficace. Sa structure modulaire, soutenue par un riche écosystème de middleware, permet aux développeurs de construire des applications robustes et performantes avec une flexibilité remarquable. Le routage avancé, la facilité de configuration des middleware, et la prise en charge intégrée des applications RESTful sont parmi les caractéristiques qui font d'Express.js un choix privilégié pour les développeurs souhaitant créer des solutions web évolutives. Sa popularité et son adoption généralisée sont soutenues par une large communauté, offrant une vaste gamme de ressources, de bibliothèques complémentaires, et un support continu, ce qui en fait un pilier incontournable du développement moderne en JavaScript.



Figure 2.4 : Logo Express JS

5. Flask :

Flask est un micro-framework web léger écrit en Python, conçu pour rendre le développement d'applications web simple et rapide. Il fournit les outils, les bibliothèques et les technologies nécessaires pour construire une application web, tout en offrant aux développeurs la flexibilité de choisir comment les assembler. Contrairement à d'autres frameworks web plus volumineux qui imposent une structure spécifique et de nombreuses dépendances, Flask adopte une approche minimaliste et modulaire, permettant aux

développeurs de n'utiliser que les composants dont ils ont besoin, évitant ainsi la surcharge et facilitant la maintenance. Flask est particulièrement apprécié pour sa simplicité d'apprentissage, sa documentation exhaustive, et sa communauté active, ce qui le rend accessible aux débutants tout en étant suffisamment robuste pour les applications d'entreprise complexes. Il intègre un serveur de développement et un débogueur, supporte le routage d'URL, et propose une intégration facile avec les bases de données. De plus, Flask peut être étendu avec une variété d'extensions disponibles pour ajouter des fonctionnalités telles que l'authentification des utilisateurs, la gestion des sessions, et l'interaction avec des bases de données ORM. Cette combinaison de simplicité, de flexibilité et de puissance fait de Flask un choix populaire pour le développement web rapide, de la création de simples services web à la construction de systèmes complexes et hautement personnalisés.



Figure 2.5: Logo Flask

6. OpenRouteService

OpenRouteService est une plateforme avancée de planification d'itinéraires et d'analyses géospatiales basée sur des données OpenStreetMap (OSM), offrant une alternative puissante et flexible aux services de cartographie traditionnels. Elle fournit une gamme étendue de services API, incluant le calcul d'itinéraires pour différents modes de transport (comme la marche, le vélo, la conduite, et les déplacements en fauteuil roulant), l'optimisation de trajets, la génération d'isochrones (zones accessibles dans un intervalle de temps donné), ainsi que la géocodification inverse et directe. OpenRouteService se distingue par son engagement envers l'open source et l'utilisation de données librement accessibles, permettant aux développeurs et aux chercheurs de créer des applications personnalisées et des analyses spatiales précises. Grâce à sa capacité à traiter de grandes quantités de données géospatiales et à fournir des résultats en temps réel, OpenRouteService est devenu un outil indispensable dans le domaine de la planification urbaine, du tourisme, de la logistique et au-delà, en facilitant la prise de décisions éclairées basées sur la localisation. Sa facilité d'intégration dans des projets de développement logiciel, couplée à une documentation riche et à une communauté d'utilisateurs active, en fait une ressource précieuse pour quiconque cherche à exploiter des données géographiques complexes de manière intuitive et efficace.



Figure 2.6: Logo OpenRouteService

7. Bootstrap

Bootstrap est un framework HTML, CSS, et JavaScript de premier plan conçu pour accélérer et simplifier le développement de sites web réactifs et mobiles premiers. Depuis son lancement par Twitter en 2011, Bootstrap a révolutionné le développement frontal en fournissant un ensemble riche et extensible de feuilles de style et de composants réutilisables qui facilitent la création d'interfaces utilisateur élégantes et cohérentes. Avec son système de grille flexible, ses composants préconçus (tels que des boutons, des formulaires, des modales, et des carrousels), et ses plugins JavaScript puissants, Bootstrap permet aux développeurs de construire des sites adaptatifs rapidement sans sacrifier la qualité ou la compatibilité entre navigateurs. L'adoption de Bootstrap par des millions de sites web à travers le monde témoigne de sa facilité d'utilisation, de son efficacité et de sa capacité à s'adapter aux tendances changeantes du design web. En outre, une vaste communauté de développeurs contribue régulièrement à son développement, assurant que le framework reste à la pointe des technologies web et accessible à tous, des novices en codage aux professionnels expérimentés.



Figure 2.7: Logo Bootstrap

8. MongoDB

MongoDB est développé depuis 2007, est un système de gestion de base de données orienté documents, répartitionnable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données.



Figure 2.8 : Logo de MongoDB

9. MongoDB Compass

MongoDB Compass est une interface graphique pour MongoDB, fournie par les développeurs pour MongoDB.

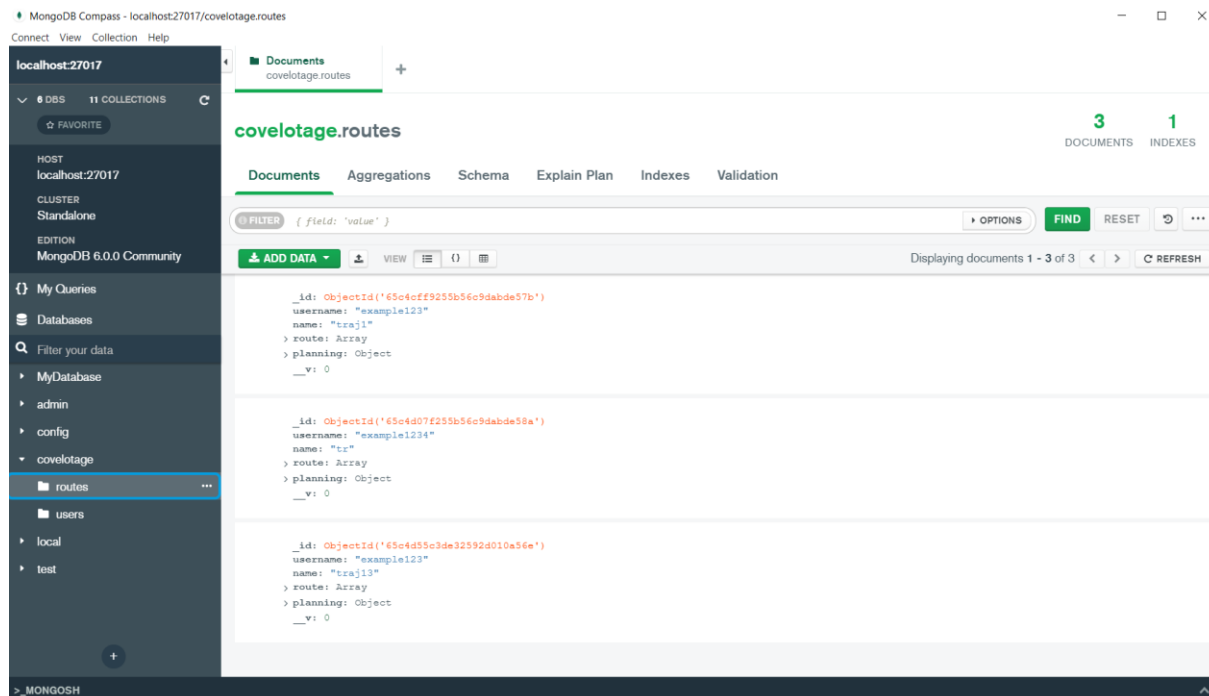


Figure 2.9 : Interface MongoDB Compass

II. Conclusion

En conclusion, l'examen approfondi des frameworks et logiciels utilisés dans notre projet révèle leur importance cruciale dans le développement d'une solution logicielle robuste et efficace. De la conception d'interfaces utilisateur engageantes avec React JS à la facilitation du développement serveur avec Node.js, en passant par la rapidité et la légèreté de Flask, la réactivité de Bootstrap, et la précision géospatiale d'OpenRouteService, chaque outil a joué un rôle indispensable dans la matérialisation de nos objectifs. Cette exploration souligne non seulement la polyvalence et la puissance de ces technologies mais aussi la manière dont leur intégration harmonieuse peut mener à la réussite d'un projet. À travers ce chapitre, nous avons non seulement détaillé les fonctionnalités et avantages de chaque logiciel mais aussi illustré l'impact de la synergie technologique sur l'avancement et l'innovation dans le développement logiciel.

3

Recherche et Développement

I. Algorithme de recherche du chemin le plus court :

1. Architecture :

Cette partie est conçue pour faciliter la communication entre un client web interactif et un serveur backend robuste, tout en utilisant une API externe pour la navigation et le routage.

Frontend (Interface Utilisateur) :

Le frontend est développé en HTML, CSS et JavaScript, avec l'intégration de la bibliothèque Leaflet.js pour la cartographie. Il offre une interface utilisateur graphique permettant aux utilisateurs de sélectionner des points de départ et d'arrivée directement sur la carte.

Les coordonnées des points sélectionnés sont capturées et envoyées au backend via des requêtes AJAX.

Backend (Serveur Flask) :

Le backend est un serveur Flask qui reçoit les données des points de départ et d'arrivée du frontend. Il est responsable de l'interaction avec l'API OpenRouteService en utilisant la clé API obtenue après l'inscription. Puis, il traite la réponse de l'API, la convertit en un format approprié et la renvoie au frontend.

Communication avec l'API OpenRouteService :

Le serveur Flask envoie les requêtes à l'API OpenRouteService et reçoit des données de routage sous forme de réponse JSON. Ces données sont ensuite parsees et structurées pour être compatibles avec la bibliothèque Leaflet.js pour l'affichage sur la carte.

Communication entre le serveur Flask et le client web :

La communication est établie via des requêtes HTTP, où AJAX est utilisé pour des requêtes asynchrones permettant une expérience utilisateur fluide sans rechargement de page. Les données d'itinéraire reçues du backend sont utilisées pour mettre à jour la carte en temps réel, affichant le chemin calculé.

2. Choix d'OpenRouteService

Dans le domaine des services de cartographie et de navigation, il existe plusieurs options d'API telles que Google Maps API, Mapbox, Here Maps, et OpenRouteService. Chacune de ces API offre des ensembles uniques de fonctionnalités qui peuvent être plus ou moins adaptées à des projets spécifiques. Pour ce projet en particulier, OpenRouteService a été préféré pour plusieurs raisons clés.

Comparaison avec d'autres :

Google Maps API est l'une des solutions les plus populaires et les plus complètes pour les services de cartographie. Elle offre une large gamme de fonctionnalités, une intégration facile, et une vaste base

d'utilisateurs. Cependant, Google Maps API est un service payant avec un modèle de tarification basé sur l'utilisation, ce qui peut devenir coûteux à grande échelle. De plus, les politiques de personnalisation et de confidentialité peuvent être restrictives pour certains projets.

Avantages spécifiques d'OpenRouteService pour le projet :

- **Flexibilité** : OpenRouteService, basé sur des données OpenStreetMap, offre une grande flexibilité en termes de personnalisation de l'itinéraire, permettant aux développeurs de créer des solutions sur mesure adaptées à leurs besoins spécifiques.
- **Coûts** : Contrairement à Google Maps API, OpenRouteService est gratuit, ce qui en fait une option attrayante pour les projets avec un budget limité ou pour une utilisation expérimentale.
- **Documentation et Support** : OpenRouteService fournit une documentation complète et bénéficie du soutien d'une communauté active, ce qui est essentiel pour résoudre les problèmes et partager les meilleures pratiques.
- **Conformité aux données ouvertes** : Pour les projets qui valorisent les principes de l'open-source et la transparence des données, OpenRouteService est aligné avec ces valeurs, utilisant des données de carte ouvertes et permettant une modification et une distribution libres.
- **Hébergement en local** : un avantage majeur de OpenRouteService que celui-ci puisse être hébergé localement, offrant ainsi un contrôle total sur les données et permettant un accès illimité aux requêtes. Cette caractéristique est souvent cruciale pour des organisations ou des projets qui ont des exigences spécifiques en matière de confidentialité, de sécurité ou de volume de requêtes.

Contraintes et limitations prises en compte :

- **Fonctionnalités** : Bien que riche en fonctionnalités, OpenRouteService peut ne pas avoir certaines fonctionnalités spécialisées disponibles dans d'autres services comme Google Maps API.
- **Complexité de mise en œuvre** : Certaines fonctionnalités avancées peuvent nécessiter une compréhension plus approfondie de la manipulation des données géospatiales, ce qui peut augmenter la courbe d'apprentissage pour les développeurs.

En conclusion, OpenRouteService a été choisi pour ce projet en raison de sa nature gratuite, de sa flexibilité, et de son alignement avec les principes de données ouvertes. Ce choix a été effectué en tenant compte des contraintes de budget, des besoins de personnalisation, et des objectifs à long terme du projet, tout en évaluant soigneusement les compromis par rapport aux limitations de service.

3. Configuration de l'OpenRouteService

L'utilisation de l'API OpenRouteService nécessite une configuration initiale qui inclut l'inscription pour obtenir une clé API, l'intégration de cette clé dans l'application, et la mise en place de mesures pour assurer la sécurité et la confidentialité de la clé.

Procédure de création de compte et acquisition de la clé API :

Pour bénéficier des fonctionnalités d'OpenRouteService, tout développeur doit préalablement s'inscrire pour obtenir une clé API. Les étapes généralement impliquées dans ce processus sont les suivantes :

1. **Inscription sur le site OpenRouteService** : Accès au site web d'OpenRouteService et localisation de la page d'inscription.
2. **Fourniture des informations requises** : Complétion du formulaire d'inscription en indiquant le nom, l'adresse e-mail, et autres détails requis par le service.
3. **Validation de l'e-mail** : Réception d'un e-mail de confirmation à l'adresse fournie. Suivi du lien inclus dans l'e-mail pour activer le compte.
4. **Connexion au tableau de bord OpenRouteService** : Une fois le compte activé, connexion au tableau de bord personnel sur OpenRouteService.
5. **Création d'une clé API** : Recherche dans le tableau de bord de l'option permettant de générer une nouvelle clé API. Suivi des instructions pour créer une clé.
6. **Enregistrement de la clé API** : Une fois générée, copie et stockage de la clé API en lieu sûr.

Configuration de la clé API dans l'application :

Après l'obtention de la clé API, la prochaine étape consiste à la configurer dans l'application Flask. Généralement, cette configuration se déroule comme suit :

1. **Stockage sécurisé de la clé** : Placement de la clé API dans un fichier de configuration.
2. **Accès à la clé dans l'application** : Utilisation de la fonction "open" de Python pour l'importation de la clé API dans l'application Flask à partir du fichier.
3. **Utilisation de la clé dans les requêtes API** : Inclusion de la clé API dans les en-têtes ou les paramètres de requête lors de l'appel à OpenRouteService.

Aspects de sécurité et de confidentialité liés à la clé API :

La clé API est essentielle pour l'accès aux services OpenRouteService et doit être manipulée avec prudence afin d'éviter les abus et les accès non autorisés :

1. **Non-partage public de la clé API** : Évitement du stockage de la clé dans des dépôts de code source publics ou du partage dans des forums publics.
2. **Surveillance de l'activité** : Contrôle régulier de l'activité de la clé API via le tableau de bord OpenRouteService pour détecter tout usage suspect.
3. **Rotation des clés** : Modification périodique de la clé API pour réduire le risque d'exploitation en cas de compromission de la clé.

En adoptant ces pratiques, garantie que l'intégration d'OpenRouteService dans l'application est à la fois sécurisée et conforme aux meilleures pratiques en matière de sécurité informatique.

4. Analyse de la Réponse JSON de l'API

L'API OpenRouteService renvoie les données de routage sous la forme d'une collection de caractéristiques GeoJSON. La structure de la réponse JSON est organisée comme suit :

- **FeatureCollection** : Le niveau le plus élevé de la structure, qui contient l'ensemble des données de l'itinéraire.
- **Metadata** : Fournit des métadonnées telles que l'attribution, le service utilisé, le timestamp, et les détails de la requête, y compris les coordonnées des points et le profil de routage.
- **Bbox** : Représente la boîte englobante de l'itinéraire qui délimite l'étendue spatiale des caractéristiques de l'itinéraire.
- **Features** : Un tableau qui contient l'itinéraire, où chaque élément définit une partie de l'itinéraire avec ses propres métadonnées.

À l'intérieur de **Features**, on trouve :

- ✓ **Properties** : Comprend des détails comme la distance totale, la durée, et les segments qui comprennent les différentes étapes de l'itinéraire.
- ✓ **Geometry** : Décrit la forme géométrique de l'itinéraire, généralement sous forme de "LineString" avec des coordonnées associées.

Le backend Flask parse cette structure pour en extraire les informations nécessaires pour le traçage de l'itinéraire. Le champ « **geometry** » est essentiel pour déterminer le tracé visuel de l'itinéraire sur la carte. Les « **steps** » dans chaque « **segment** » fournissent des instructions détaillées pour chaque partie du trajet, qui peuvent être utilisées pour afficher des directives à l'utilisateur.

5. Script Python Backend

Le script backend écrit en Python utilise le micro-framework Flask pour gérer les requêtes et les réponses HTTP. Ce script est crucial pour la logique de traitement des coordonnées fournies par l'utilisateur et pour l'interaction avec l'API OpenRouteService. Voici une analyse détaillée du code du serveur Flask (algo.py):

Gestion des Requêtes POST :

Le serveur Flask est configuré pour écouter sur l'endpoint pour les requêtes POST. Lorsqu'une requête POST est reçue, la fonction « `receive_coordinates()` » est appelée. Cette fonction extrait les données JSON de la requête, qui contiennent les coordonnées de départ et d'arrivée fournies par l'utilisateur.

Traitement des Réponses de l'API OpenRouteService :

Avec les coordonnées reçues, le script configure le client OpenRouteService en utilisant la clé API lue depuis le fichier « `OpenRouteService.txt` ». Il effectue ensuite une requête à l'API OpenRouteService en demandant un itinéraire pour le cyclisme entre les coordonnées de départ et d'arrivée au format GeoJSON. Une fois la réponse reçue, elle est stockée dans un dictionnaire « `route` ».

Communication des Données d'Itinéraire au Frontend :

La réponse de l'API, contenue dans le dictionnaire « route », est renvoyée au frontend en tant que réponse JSON via la fonction « jsonify() ». La réponse inclut un message de confirmation et les données de l'itinéraire qui seront utilisées par le script frontend pour afficher l'itinéraire sur la carte.

Points Clés du Script :

- Importation des modules nécessaires tels que Flask, openrouteservice et json.
- Initialisation de l'application Flask et configuration du CORS pour permettre les requêtes cross-origin.
- Lecture de la clé API depuis un fichier et initialisation du client OpenRouteService.
- Récupération des coordonnées du JSON de la requête, traitement des données et construction de la réponse.

Code :

```
• from flask import Flask, request, jsonify
• from flask_cors import CORS
•
• import openrouteservice as ors
• import folium
• import json
•
• app = Flask(__name__)
• CORS(app)
•
• @app.route('/endpoint', methods=['POST'])
• def receive_coordinates():
•     data = request.get_json()
•
•     # Access the coordinates from the JSON data
•     start_latitude = data.get('start_latitude')
•     start_longitude = data.get('start_longitude')
•     end_latitude = data.get('end_latitude')
•     end_longitude = data.get('end_longitude')
•
•     # Process the coordinates as needed
•     print(f'Start Latitude: {start_latitude}')
•     print(f'End Latitude: {end_latitude}')
•     print(f'Start Longitude: {start_longitude}')
•     print(f'End Longitude: {end_longitude}')
•
•     # Perform operations and generate the 'route' dictionary
•
•     API=open("OpenRouteService.txt","r")
•     APIKey=API.read()
•     client=ors.Client(key=APIKey)
•     route={}
•     coords = [[start_longitude,start_latitude], [end_longitude,end_latitude]]
```

```

•     print(coords)
•     route = client.directions(coordinates=coords,
•                               profile='cycling-regular',
•                               format='geojson')
•
•     print(len(route))
•
•
•     # send a response back to the JavaScript code with the 'route' dictionary
•     return jsonify({'message': 'Coordinates received successfully', 'route':
route})
•
•
• if __name__ == '__main__':
•     # Run the Flask app on localhost:5000
•     app.run(debug=True)
•

```

Ce script est l'épine dorsale de l'application, car il effectue l'opération clé de calcul d'itinéraire et de transmission des résultats au client. Il montre comment Flask peut être utilisé pour créer des applications web interactives et dynamiques qui intègrent des services tiers.

6. Frontend Web

Le frontend de l'application web est la partie avec laquelle l'utilisateur interagit directement. Il est composé d'une structure HTML, stylisée avec CSS, et rendue dynamique grâce à JavaScript en conjonction avec la bibliothèque Leaflet pour les fonctionnalités de cartographie.

Description de la structure HTML/CSS de la page :

HTML : Le corps principal de « index.html » contient une <div> avec la classe « wrap » qui englobe le formulaire pour les entrées de latitude et longitude ainsi que la « <div> » de l'id « map » où la carte est affichée.

CSS : Les styles sont définis dans « style.css » et sont responsables de la mise en page visuelle de la page web. Ils déterminent la disposition des éléments du formulaire, la taille et la position de la carte, ainsi que d'autres styles visuels pour rendre l'interface utilisateur attrayante et accessible.

Explication du rôle du JavaScript dans l'interaction avec la carte et le backend :

JavaScript : Le fichier « scripts.js » gère les interactions de l'utilisateur avec la carte et coordonne la communication entre le frontend et le backend. Lorsque l'utilisateur clique sur la carte, le script place des marqueurs pour le point de départ et le point d'arrivée, et récupère les coordonnées de ces points. Ces

coordonnées sont ensuite envoyées au backend via une requête POST asynchrone AJAX, en utilisant la méthode « fetch ».

Une fois que le backend Flask renvoie l'itinéraire calculé, le script JavaScript le reçoit sous forme de données GeoJSON. Il utilise ensuite Leaflet pour tracer l'itinéraire sur la carte en utilisant ces données, en mettant à jour la carte avec une polyligne représentant l'itinéraire calculé.

Ce script joue un rôle vital en rendant la page web réactive et interactive. Il permet une mise à jour dynamique de la carte sans avoir besoin de recharger la page, offrant ainsi une expérience utilisateur fluide et réactive.

7. Interaction Python et Page Web via Flask

La communication entre le frontend et le backend dans une application web Flask est un processus clé qui permet l'échange de données de manière fluide et sécurisée. Flask joue un rôle central dans cette interaction, en agissant comme le serveur qui reçoit les requêtes du client (frontend), traite ces requêtes, et renvoie des réponses appropriées.

Comment Flask Facilite la Communication :

1. **Gestion des Requêtes HTTP** : Flask est conçu pour gérer les requêtes HTTP, y compris GET, POST, PUT, et DELETE. Dans notre cas, les coordonnées sélectionnées par l'utilisateur sur la carte sont envoyées au serveur Flask via une requête POST.
2. **Routing**: Flask utilise des décorateurs pour définir des routes, ce qui permet de mapper les URL aux fonctions Python qui doivent être exécutées quand ces URL sont visitées. Chaque route peut accepter des données envoyées par le client, comme les coordonnées dans notre application.
3. **Traitement des Données** : Une fois que le serveur Flask reçoit les coordonnées via la requête POST, il les utilise pour faire une requête à l'API OpenRouteService. Flask gère le traitement de ces données, y compris la sérialisation et la désérialisation des données JSON.

Transfert des Données de l'Itinéraire vers le Client :

1. **Récupération de l'Itinéraire** : Après avoir reçu les données de l'itinéraire de l'API OpenRouteService, le serveur Flask les traite, potentiellement en filtrant ou en reformatant les données selon les besoins de l'application frontend.
2. **Envoi des Données au Frontend** : Flask envoie ensuite ces données au client sous forme de réponse JSON à la requête POST initiale. Ceci est réalisé en utilisant la fonction « jsonify() » de Flask, qui convertit les dictionnaires Python en JSON.
3. **Affichage sur la Carte** : Côté client, le JavaScript reçoit la réponse JSON et utilise les données de l'itinéraire pour les afficher sur la carte. Ceci est généralement accompli en utilisant la bibliothèque Leaflet pour ajouter une couche d'itinéraire ou une polyligne basée sur les coordonnées de l'itinéraire reçu.

En conclusion, Flask sert de pont entre le frontend et le backend, permettant une interaction fluide et sécurisée. Il gère la réception des données de l'utilisateur, communique avec des services tiers comme OpenRouteService, et retourne les données nécessaires pour enrichir l'expérience utilisateur côté client.

8. Extrait Explicatif du Code de la page WEB:

Initialisation de la Carte

```
let mapOptions = {
  center: [48.659260, 6.173090],
  zoom: 13
};
let map = new L.map('map', mapOptions);
let layer = new L.TileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
map.addLayer(layer);
```

Cette section du code initialise la carte Leaflet, la centre sur une position spécifique et ajoute une couche de tuiles OpenStreetMap pour l'affichage.

Gestion des Clics sur la Carte

```
map.on('click', (event) => {
  // Logique pour placer les marqueurs de départ et d'arrivée
});
```

Ici, nous écoutons les clics sur la carte pour placer alternativement les marqueurs de départ et d'arrivée, et enregistrons les coordonnées dans des variables.

Envoi des Coordonnées au Serveur

```
function sendCoordinatesToServer(coords) {
  fetch(url, {
    method: 'POST',
    body: JSON.stringify(coords),
  })
  .then(response => response.json())
  .then(data => {
    // Logique pour afficher le trajet sur la carte
  });
}
```

Cet extrait montre comment les coordonnées sont envoyées au serveur via une requête POST. La réponse est ensuite traitée pour afficher le trajet sur la carte.

Affichage du Trajet

```
.then(data => {
  coordinates = data.route['features'][0]['geometry']['coordinates'].map(coord =>
  coord.reverse());
  polyline = L.polyline(coordinates, {color: 'blue'}).addTo(map);
```

```
});
```

Après avoir reçu le trajet du serveur, cet extrait montre comment le trajet est converti en une ligne (polyline) et affiché sur la carte.

Stylisation avec CSS

```
#map{  
  width: 100%;  
  height: 500px;  
  border-radius: 10px;  
}
```

Cet extrait de CSS définit la taille de l'élément de la carte, essentiel pour s'assurer que la carte est visible et correctement dimensionnée sur la page.

Page Web finale :

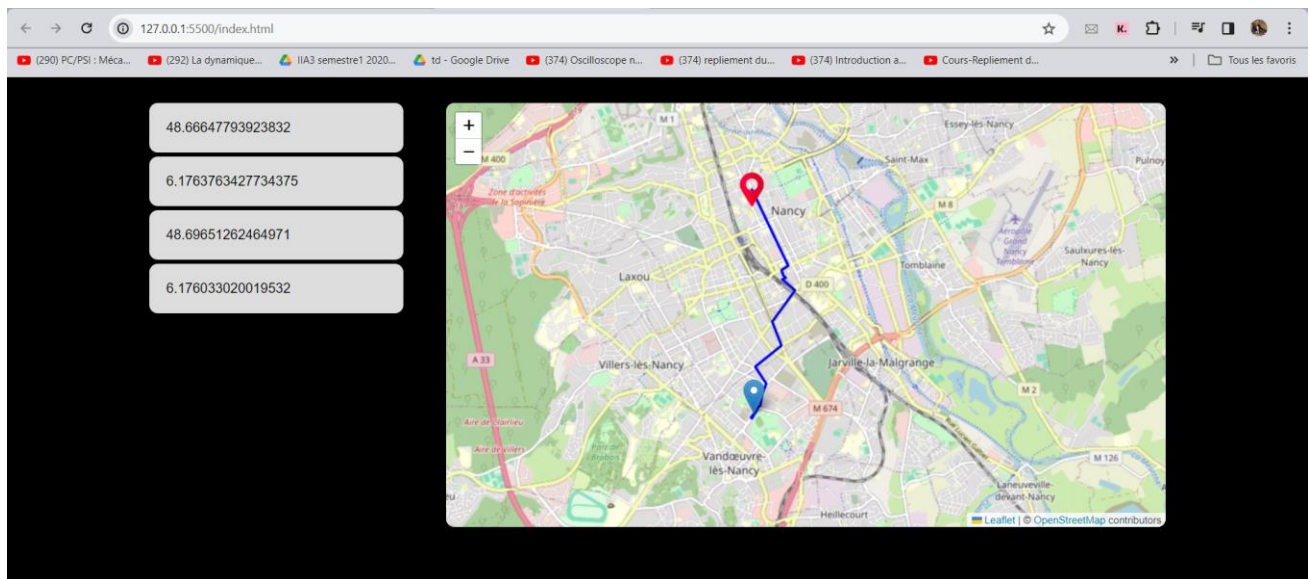


Figure 3.1 : Page Web de test pour l'algorithme du chemin le plus court

II. Algorithme de correspondance de chemin :

1. Introduction à l'algorithme LCS :

L'algorithme LCS est une méthode de programmation dynamique utilisée pour trouver la plus longue sous-séquence commune entre deux séquences. Une sous-séquence est une séquence qui apparaît dans le même ordre relatif, mais pas nécessairement de manière contiguë dans les deux séquences.

2. Étapes de l'Algorithme LCS :

1. **Initialisation** : Création d'une matrice « **dp** » (pour dynamic programming) de taille $(m+1) \times (n+1)$, où « **m** » est la longueur de la première séquence et « **n** » est la longueur de la deuxième séquence. Toutes les cellules de cette matrice sont initialisées à 0. Cette matrice est utilisée pour stocker les longueurs des sous-séquences communes les plus longues à différentes étapes.
2. **Remplissage de la Matrice** : Parcourir chaque élément des deux séquences. Pour chaque paire d'éléments (i, j) :
 - Si les éléments sont identiques ($list1[i - 1] == list2[j - 1]$), cela signifie que nous avons trouvé un élément commun aux deux séquences. Nous ajoutons 1 à la valeur de la cellule diagonale précédente $dp[i - 1][j - 1]$ et stockons le résultat dans $dp[i][j]$.
 - Si les éléments ne sont pas identiques, nous prenons le maximum entre $dp[i - 1][j]$ et $dp[i][j - 1]$ et le stockons dans $dp[i][j]$. Cela reflète la plus longue sous-séquence commune jusqu'à présent sans inclure les éléments actuels.
3. **Reconstitution de la LCS** :

Une fois la matrice **dp** entièrement remplie, la valeur en $dp[m][n]$, qui représente la longueur de la LCS pour les séquences complètes A et B, est identifiée.

Pour reconstituer la LCS, suivez ces étapes :

 1. **Commencez à partir de $dp[m][n]$** : Positionnez-vous à la cellule qui correspond à la fin des deux séquences
 2. **Recherchez les correspondances** : Si les caractères correspondants à la position actuelle dans les séquences A et B sont identiques, cela signifie que ce caractère fait partie de la LCS. Notez ce caractère et déplacez-vous diagonalement vers $dp[i-1][j-1]$.
 3. **Gérez les non-correspondances** : Si les caractères ne correspondent pas, vous devez décider de la direction à prendre : vers le haut ($dp[i-1][j]$) ou vers la gauche ($dp[i][j-1]$). Le choix dépend de la valeur la plus élevée des deux cellules, indiquant le chemin pris pour obtenir la LCS optimale. Si les valeurs sont égales, vous pouvez choisir n'importe quelle direction, mais généralement, on suit une règle constante pour maintenir la cohérence.
 4. **Continuez jusqu'à atteindre le début** : Répétez les étapes 2 et 3 jusqu'à ce que vous atteigniez la cellule $dp[0][0]$. Le chemin emprunté indique les éléments de la LCS, mais en ordre inverse puisque vous remontez de la fin au début.

4. **Résultat** : La sous-séquence commune la plus longue est obtenue en inversant l'ordre des éléments collectés pendant la phase de reconstitution, car nous remontons de la fin au début.

3. Exemple illustratif :

Pour illustrer comment la matrice **dp** est remplie étape par étape dans le cadre de l'algorithme du plus long sous-séquence commune, nous utiliserons l'exemple des séquences « ACBA » et « ABCBA ». Ce processus démontre comment l'algorithme évalue les séquences pour déterminer la longueur de la LCS à chaque intersection des séquences, permettant ainsi de reconstruire la sous-séquence à la fin.

Séquence A : ACBA

Séquence B : ABCBA

Étape 1 : Initialisation

La matrice **dp** est initialisée avec des dimensions $(m+1) \times (n+1)$, où « $m=4$ » et « $n=5$ » sont les longueurs des séquences A et B, respectivement. Toutes les valeurs sont initialisées à zéro. Pour les séquences « ACBA » et « ABCBA », la matrice initiale (avec une ligne et une colonne supplémentaire pour la base zéro) serait :

	-	A	B	C	B	A
-		0	0	0	0	0
A	0					
C	0					
B	0					
A	0					

Étape 2 : Remplissage de la Matrice

La matrice est remplie en parcourant chaque caractère de A et B. Pour chaque paire (i, j) :

- Si $A[i] == B[j]$, on ajoute 1 à $dp[i-1][j-1]$ et on stocke le résultat dans $dp[i][j]$.
- Si $A[i] \neq B[j]$, on prend le maximum entre $dp[i-1][j]$ et $dp[i][j-1]$ et on le place dans $dp[i][j]$.

En appliquant ces règles, voici le remplissage progressif pour notre exemple :

Après la première comparaison (A contre A) :

	-	A	B	C	B	A
-	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0					
B	0					
A	0					

Continuant avec C contre B, puis A contre C, et ainsi de suite, jusqu'à remplir toute la matrice :

	-	A	B	C	B	A
-	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
B	0	1	2	2	3	3
A	0	1	2	2	3	4

Étape 3 : La LCS

La valeur en $dp[m][n]$ (ici, 4) indique la longueur de la LCS. En partant de $dp[4][5]$, on peut retracer le chemin qui a mené à cette LCS, en remontant à travers la matrice selon les règles établies.

En partant de $dp[4][5]$, on trouve que les séquences se terminent toutes les deux par « A », donc « A » est une partie de la LCS. On se déplace diagonalement à $dp[3][4]$, où on trouve une correspondance avec « B », et ainsi de suite, jusqu'à reconstituer la LCS « ACBA ».

4. Importance de LCS dans l'Identification des Segments Partagés :

- **Détection de Similarités** : En identifiant la plus longue sous-séquence commune, on peut déterminer si deux utilisateurs partagent une portion significative de leur trajet.
- **Flexibilité** : La méthode LCS n'exige pas que les itinéraires soient identiques dans leur intégralité pour reconnaître les correspondances. Elle permet de détecter des similitudes même lorsque les itinéraires diffèrent partiellement, offrant ainsi une flexibilité appréciable dans l'analyse des trajets.
- **Optimisation des Ressources** : En identifiant les segments d'itinéraire partagés, les utilisateurs peuvent optimiser leurs déplacements en termes de coûts, de temps, et d'impact environnemental.

5. Calcul du coefficient de similarité :

La mesure de similarité fournie permet d'évaluer de manière quantitative à quel point deux itinéraires sont similaires. Ce processus de comparaison est crucial pour identifier des opportunités de mise en relation entre utilisateurs partageant des intérêts de trajet communs, favorisant ainsi la collaboration et l'optimisation des déplacements.

Le score de similarité est calculé en divisant la longueur de la sous-séquence commune par la longueur du plus court des deux itinéraires comparés. Ce ratio offre une mesure normalisée de la similarité, allant de 0 (aucune similarité) à 1 (itinéraires identiques).

6. Implémentation :

Configuration de l'API OpenRouteService :

- **Chargement de la clé API** : La clé API est lue depuis un fichier « OpenRouteService.txt ».
- **Création d'un client ORS** : Un client OpenRouteService est initialisé avec la clé API.

Définition des coordonnées des utilisateurs :

- **Coordonnées des utilisateurs** : Des paires de coordonnées sont définies pour simuler les points de départ et d'arrivée de différents utilisateurs.

Obtention des itinéraires :

- **Demande d'itinéraires** : Pour chaque paire de coordonnées, une requête est envoyée à l'API OpenRouteService pour obtenir un itinéraire cyclable. Ces itinéraires sont stockés dans des variables distinctes.

Comparaison des itinéraires :

- **Longest Common Subsequence (LCS)** : La fonction « **longest_common_subsequence** » (LCS) est un composant essentiel de l'algorithme, conçu pour identifier les segments d'itinéraire partagés entre deux ensembles de coordonnées.
- **Calcul de similarité** : La fonction « **compare_routes** » constitue un pivot dans l'analyse de la similarité entre deux itinéraires, exploitant la méthode Longest Common Subsequence (LCS) pour quantifier la similarité de manière objective. Cette fonction est essentielle pour l'application, permettant de déterminer les utilisateurs ayant des segments de trajet en commun significatifs. Voici une exploration détaillée de son fonctionnement et de son rôle dans l'analyse des résultats.

Identification des utilisateurs similaires :

- **Boucles de comparaison** : Tous les itinéraires sont comparés deux à deux. Pour chaque paire d'itinéraires ayant une similarité supérieure à zéro, les informations pertinentes sont stockées dans un dictionnaire « **similar_users** ».

Affichage des résultats :

- **Impression des paires similaires** : Les paires d'utilisateurs ayant des itinéraires partiellement communs sont affichées avec leur coefficient de similarité et les coordonnées de début et de fin de leur sous-séquence commune.

Visualisation des itinéraires sur une carte :

- **Création de la carte** : Une carte Folium est initialisée.
- **Ajout des itinéraires à la carte** : Chaque itinéraire est ajouté à la carte avec une couleur spécifique. Des marqueurs sont placés pour indiquer le début et la fin de chaque itinéraire.

Code :

```
import openrouteservice as ors
import folium
import json

# Charger la clé API depuis un fichier
API = open("OpenRouteService.txt", "r").read()
client = ors.Client(key=API)
# si hosted en local
# client = ors.Client(base_url='localhost:8080/ors')

# Coordonnées des utilisateurs
coords_user1 = [[6.183264, 48.693684], [6.172724, 48.667496]]
coords_user2 = [[6.183264, 48.693684], [6.172724, 48.667496]]
coords_user3 = [[6.175689, 48.697079], [6.142902, 48.679287]]
coords_user4 = [[6.171398, 48.712372], [6.204700, 48.710333]]
coords_user5 = [[6.156463, 48.686314], [6.178264, 48.672939]]
coords_user6 = [[6.165390, 48.686767], [6.169338, 48.666137]]

# Obtenir les itinéraires pour chaque utilisateur
route_user1 = client.directions(coordinates=coords_user1, profile='cycling-regular',
format='geojson')
route_user2 = client.directions(coordinates=coords_user2, profile='cycling-regular',
format='geojson')
route_user3 = client.directions(coordinates=coords_user3, profile='cycling-regular',
format='geojson')
route_user4 = client.directions(coordinates=coords_user4, profile='cycling-regular',
format='geojson')
route_user5 = client.directions(coordinates=coords_user5, profile='cycling-regular',
format='geojson')
route_user6 = client.directions(coordinates=coords_user6, profile='cycling-regular',
format='geojson')

# Fonction pour trouver la plus longue sous-séquence commune entre deux listes
def longest_common_subsequence(list1, list2):
    m = len(list1)
    n = len(list2)

    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if list1[i - 1] == list2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
```

```

lcs_length = dp[m][n]
lcs = []
i, j = m, n

while i > 0 and j > 0:
    if list1[i - 1] == list2[j - 1]:
        lcs.append(list1[i - 1])
        i -= 1
        j -= 1
    elif dp[i - 1][j] > dp[i][j - 1]:
        i -= 1
    else:
        j -= 1

return lcs[::-1]

# Fonction pour comparer deux tracés de chemin et déterminer la similarité
def compare_routes(route1, route2):
    coords1 = route1['features'][0]['geometry']['coordinates']
    coords2 = route2['features'][0]['geometry']['coordinates']

    lcs = longest_common_subsequence(coords1, coords2)
    similarity = len(lcs) / min(len(coords1), len(coords2))

    return lcs, similarity

# Créer un dictionnaire pour stocker les paires d'utilisateurs similaires
similar_users = {}

# Comparer tous les utilisateurs entre eux
for i in range(len([route_user1, route_user2, route_user3,
route_user4, route_user5, route_user6])):
    for j in range(i + 1, len([route_user1, route_user2, route_user3,
route_user4, route_user5, route_user6])):
        user1 = [route_user1, route_user2, route_user3,
route_user4, route_user5, route_user6][i]
        user2 = [route_user1, route_user2, route_user3,
route_user4, route_user5, route_user6][j]

        lcs, similarity = compare_routes(user1, user2)
        if similarity > 0: # Seuil de similarité
            key = f"User{i + 1} et User{j + 1}"
            similar_users[key] = {"similarity": similarity, "lcs": lcs}

# Afficher les paires d'utilisateurs similaires
print("Utilisateurs avec des parties de chemin communes :")
for pair, data in similar_users.items():
    similarity = data["similarity"]

```



```

lcs = data["lcs"]

start_coords = lcs[0]
end_coords = lcs[-1]

print(f"{pair} - Similarité : {similarity}, Début : {start_coords}, Fin : {end_coords}")

# Créer la carte avec les tracés de chemin
m = folium.Map(location=list(reversed([6.183264, 48.693684])), zoom_start=13)

# Ajouter les tracés de chemin à la carte
for i, route in enumerate([route_user1, route_user2, route_user3,
route_user4, route_user5, route_user6]):
    color = ["blue", "red", "green", "purple", "orange", "black"][i]
    folium.PolyLine(locations=[list(reversed(coord)) for coord in
route['features'][0]['geometry']['coordinates']],
                    color=color).add_to(m)

# Ajouter des marqueurs au début et à la fin de chaque chemin
start_marker =
folium.Marker(location=list(reversed(route['features'][0]['geometry']['coordinates']
[0])),
              popup=f"Début - User {i + 1}",
              icon=folium.Icon(color=color))

end_marker =
folium.Marker(location=list(reversed(route['features'][0]['geometry']['coordinates']
[-1])),
              popup=f"Fin - User {i + 1}",
              icon=folium.Icon(color=color))

m.add_child(start_marker)
m.add_child(end_marker)

# Afficher la carte
m

```

Résultat :

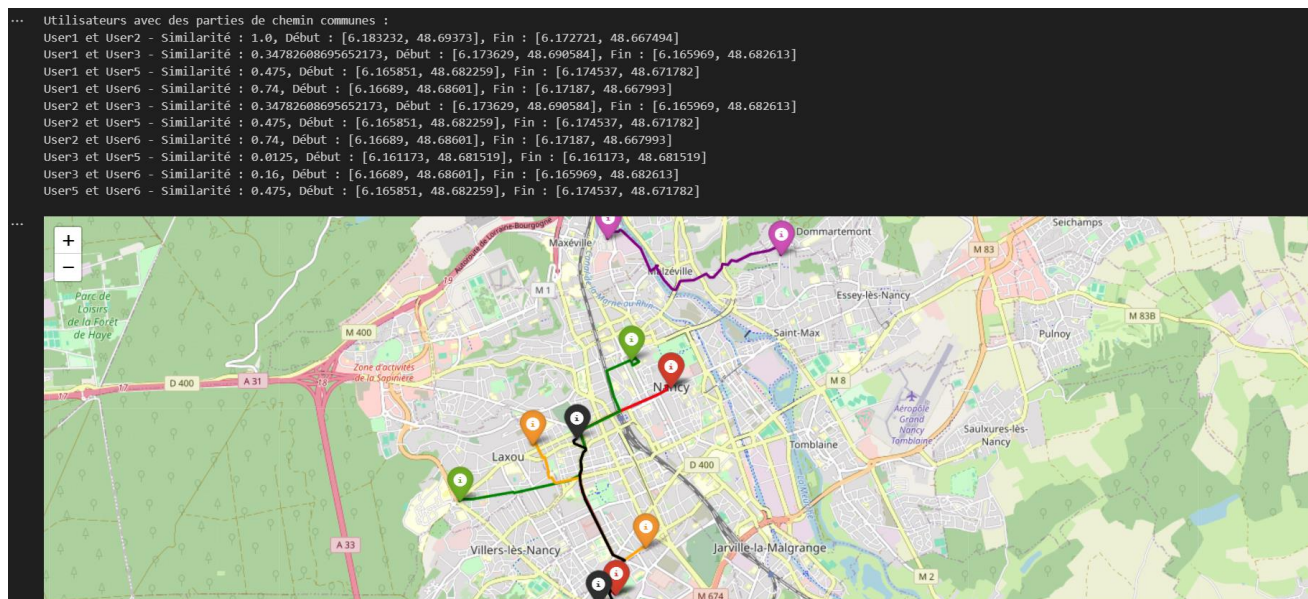


Figure 3.2: Affichage de l'algorithme de correspondance

III. Hébergement local de l'API OpenRouteService :

1. Objectif de l'Hébergement Local de l'API

L'hébergement local de l'API OpenRouteService offre plusieurs avantages significatifs pour notre projet :

- **Performance et réactivité** : Un serveur local réduit la latence associée aux requêtes API, garantissant des réponses plus rapides et une meilleure expérience utilisateur pour les applications qui dépendent de données de routage en temps réel.
- **Contrôle des données** : Héberger l'API sur nos serveurs nous permet de gérer directement les mises à jour des données OSM, assurant que notre application utilise les informations les plus actuelles et précises disponibles.
- **Gestion illimitée des requêtes** : En déployant l'API localement, nous bénéficions de la capacité de générer un nombre illimité de requêtes, éliminant ainsi les restrictions potentielles de quotas imposées par des serveurs distants. Cette flexibilité nous offre une liberté totale dans la gestion des requêtes, permettant d'optimiser les performances et d'adapter notre application aux besoins évolutifs, tout en garantissant une expérience utilisateur continue et réactive.

2. Étapes :

1. Installation de docker
2. Configuration Initiale :
3. Démarrage de Docker :
4. Test Initial
5. Personnalisation des Itinéraires
6. Test de la Nouvelle Configuration

3. Configuration Initiale :

Clonage du dépôt GitHub :

Pour obtenir la version spécifique d'OpenRouteService adaptée à nos besoins, nous utilisons la commande « git clone » suivie de l'URL du dépôt GitHub et de la spécification de la branche souhaitée. Cette commande permet de télécharger le code source complet d'OpenRouteService directement sur notre machine locale.

```
"git clone --branch v7.1.1 https://github.com/GIScience/openrouteservice.git"
```

Préparation de l'Environnement Docker :

Une fois le dépôt cloné, la préparation de l'environnement Docker nécessite la création de plusieurs dossiers essentiels au bon fonctionnement d'OpenRouteService. Ces dossiers stockeront les configurations, les caches d'élévation, les graphes de routage, ainsi que les logs d'OpenRouteService et de Tomcat (le serveur web utilisé par ORS).

```
"cd openrouteservice"
```

```
"cd docker"
```

```
"mkdir -p conf elevation_cache graphs logs/ors logs/tomcat"
```

4. Démarrage de Docker et Lancement de l'Instance OpenRouteService :

Commande de Démarrage :

Pour démarrer le service Docker sur une machine Linux, nous utilisons la commande suivante :

```
"systemctl start docker"
```

Lancement de l'Instance Docker :

Une fois le service Docker en fonction, l'instance OpenRouteService peut être initialisée en utilisant Docker Compose. Cela est réalisé à l'aide de la commande suivante dans le répertoire où se trouve le fichier `docker-compose.yml` d'OpenRouteService :

```
"docker compose up"
```

5. Test Initial:

Après avoir configuré et lancé l'instance d'OpenRouteService localement, il est essentiel de procéder à un test initial pour vérifier le bon fonctionnement de l'API. Ce test consiste à effectuer une requête simple de routage à travers un navigateur web, en utilisant une URL spécifique qui interroge l'API sur un exemple d'itinéraire en Allemagne.

Vérification Fonctionnelle :

Pour confirmer que l'API OpenRouteService fonctionne comme prévu, ouvrez un navigateur web et saisissez l'URL suivante :

```
"http://localhost:8080/ors/v2/directions/driving-car?&start=8.681495,49.41461&end=8.687872,49.420318"
```

Détails de la Requête :

- **URL de base** : « `http://localhost:8080/ors/v2/directions` » indique que nous interrogeons l'API de directions d'OpenRouteService hébergée localement.
- **Mode de transport** : « `driving-car` » spécifie que nous demandons un itinéraire pour une voiture.
- **Paramètres de l'itinéraire** : Les paramètres « `start` » et « `end` » contiennent les coordonnées géographiques (longitude, latitude) du point de départ et du point d'arrivée, respectivement.

Interprétation des Résultats :

La réussite d'une requête à cette API confirme le bon fonctionnement de l'OpenRouteService. L'observation d'un résultat au format JSON dans le navigateur révélera les détails de l'itinéraire calculé, comprenant des informations telles que la distance, le temps estimé et les étapes du parcours.

6. Personnalisation des itinéraires :

Lorsque l'OpenRouteService (ORS) est déployé localement, il est limité à fournir uniquement les itinéraires d'une ville en Allemagne en raison des données initiales.

La personnalisation des itinéraires dans OpenRouteService (ORS) est une étape essentielle pour adapter l'API aux besoins spécifiques du projet, en particulier lorsqu'une couverture géographique précise et étendue est requise. Cette section détaille le processus de téléchargement et d'intégration de données d'itinéraires personnalisées, en utilisant la région de Lorraine, en France.

Téléchargement des Données d'Itinéraires :

Pour enrichir ORS avec des itinéraires spécifiques à la région de Lorraine, il est nécessaire de télécharger les données géographiques au format PBF (Protocolbuffer Binary Format) depuis une source fiable. OpenStreetMap (OSM) offre des extraits régionaux qui peuvent être téléchargés à partir de divers sites, tels que :

"`https://download.openstreetmap.fr/extracts/europe/france/lorraine/`"

Il faut Accéder au site web et sélectionner le fichier `.pbk` correspondant à la région de Lorraine. Télécharger le fichier « `lorraine.osm.pbf` » sur le système local. Le fichier est mis dans le dossier `openRouteService`.

Intégration des Données dans OpenRouteService :

Modification du fichier « `docker-compose.yml` » :

Pour intégrer les données téléchargées dans ORS, la réalisation de modifications est nécessaire dans le fichier « `docker-compose.yml` », utilisé pour configurer et lancer l'instance Docker d'ORS. Voici les ajustements spécifiques à apporter :

- Localisation de la section « `volumes` » dans le fichier « `docker-compose.yml` ».
- Ajout du chemin vers le fichier « `lorraine.osm.pbf` » téléchargé en tant que volume, en le mappant à « `/home/ors/ors-core/data/osm_file.pbf` » dans le conteneur. Ceci est illustré par la ligne suivante dans le fichier de configuration :

"- `./lorraine.osm.pbf:/home/ors/ors-core/data/osm_file.pbf`"

Puis effacer le contenu des dossiers « `graphs` », « `elevation cache` » et « `data` »

Relancement de Docker :

Relancez docker avec la nouvelle configuration en exécutant *"docker compose up "* dans le même répertoire.

7. Test de la Nouvelle Configuration

Validation Fonctionnelle :

Pour tester le bon fonctionnement de l'API avec les nouvelles données d'itinéraires, effectuez une requête spécifique à la région de Lorraine. Cette requête peut être exécutée directement dans un navigateur web ou à l'aide d'outils comme Postman ou cURL. Voici un exemple de requête visant à calculer un itinéraire de conduite entre deux points situés en Lorraine :

"http://localhost:8080/ors/v2/directions/driving-car?start=6.167967,48.674437&end=6.148211,48.652941 "

IV. Bootstrap

1. Introduction :

Dans la dernière partie du projet, après avoir développé l'ensemble du site web et intégré les fonctionnalités essentielles telles que l'inscription, la connexion, la récupération du mot de passe, et d'autres pages liées à la gestion du compte utilisateur, il est important de discuter de l'aspect stylistique et de l'interface utilisateur du site. L'utilisation de CSS avec Bootstrap pour améliorer l'esthétique et la convivialité du site mérite une attention particulière. Voici comment vous pourriez aborder cette section dans votre rapport :

2. Intégration de CSS avec Bootstrap

Choix de Bootstrap :

Bootstrap a été choisi comme framework CSS pour notre projet pour plusieurs raisons clés qui le distinguent comme un outil de choix pour le développement web moderne :

- **Popularité** : Bootstrap est l'un des frameworks front-end les plus utilisés et reconnus, bénéficiant d'une vaste communauté de développeurs. Cette popularité garantit un large éventail de ressources, de documentation et de plugins prêts à l'emploi, facilitant ainsi le processus de développement.
- **Facilité d'utilisation** : Avec ses composants préconçus et son système de classes intuitif, Bootstrap permet une mise en œuvre rapide et efficace de designs complexes, sans nécessiter une profonde expertise en CSS.
- **Réactivité** : Le framework est conçu avec une approche mobile-first, offrant un système de grille flexible et des utilitaires responsifs qui assurent que les sites s'adaptent harmonieusement à tous les écrans, des plus grands moniteurs aux smartphones.
- **Cohérence visuelle** : En fournissant un ensemble standardisé de composants UI (boutons, formulaires, barres de navigation, etc.), Bootstrap aide à maintenir une uniformité et une cohérence visuelle à travers toutes les pages du site, renforçant ainsi l'identité de marque et l'esthétique globale du projet.

Amélioration de l'Expérience Utilisateur :

Réactivité :

L'utilisation de Bootstrap a transformé notre site en une plateforme réellement réactive :

- La mise en page s'ajuste automatiquement pour offrir une expérience de lecture optimale sur une variété d'appareils, réduisant la nécessité de zoomer, de défiler horizontalement, ou de réajuster le contenu.
- Les images et les éléments multimédias sont également rendus de manière à s'adapter à la taille de l'écran, garantissant que le contenu est toujours présenté de la manière la plus engageante possible.

Navigation et Accessibilité :

- Barre de navigation : Bootstrap a simplifié la création d'une barre de navigation cohérente et accessible sur le site, avec des menus déroulants réactifs et une bascule pour les petits écrans, améliorant ainsi la navigabilité.
- Formulaires et Boutons : Les composants de formulaire et les boutons ont été conçus pour être esthétiquement agréables tout en restant fonctionnels et accessibles, grâce à des états de focus clairs et des indications visuelles qui améliorent l'expérience utilisateur pour tous, y compris pour les personnes utilisant des technologies d'assistance.

3. Illustration par des Captures d'Écran :

Page d'accueil :



Page de login :

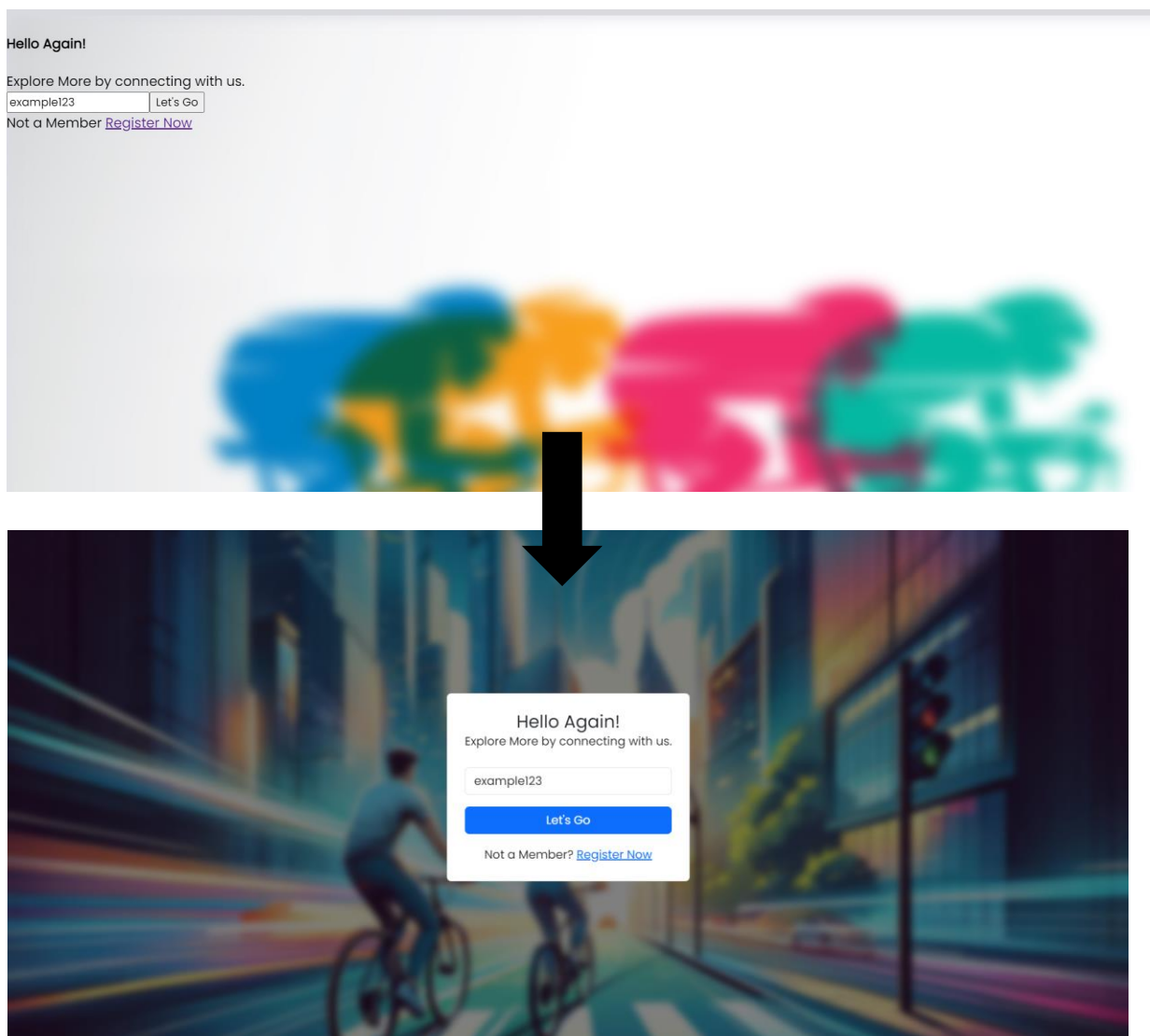


Figure 3.4: Amélioration Design Page de login

Page du Password :

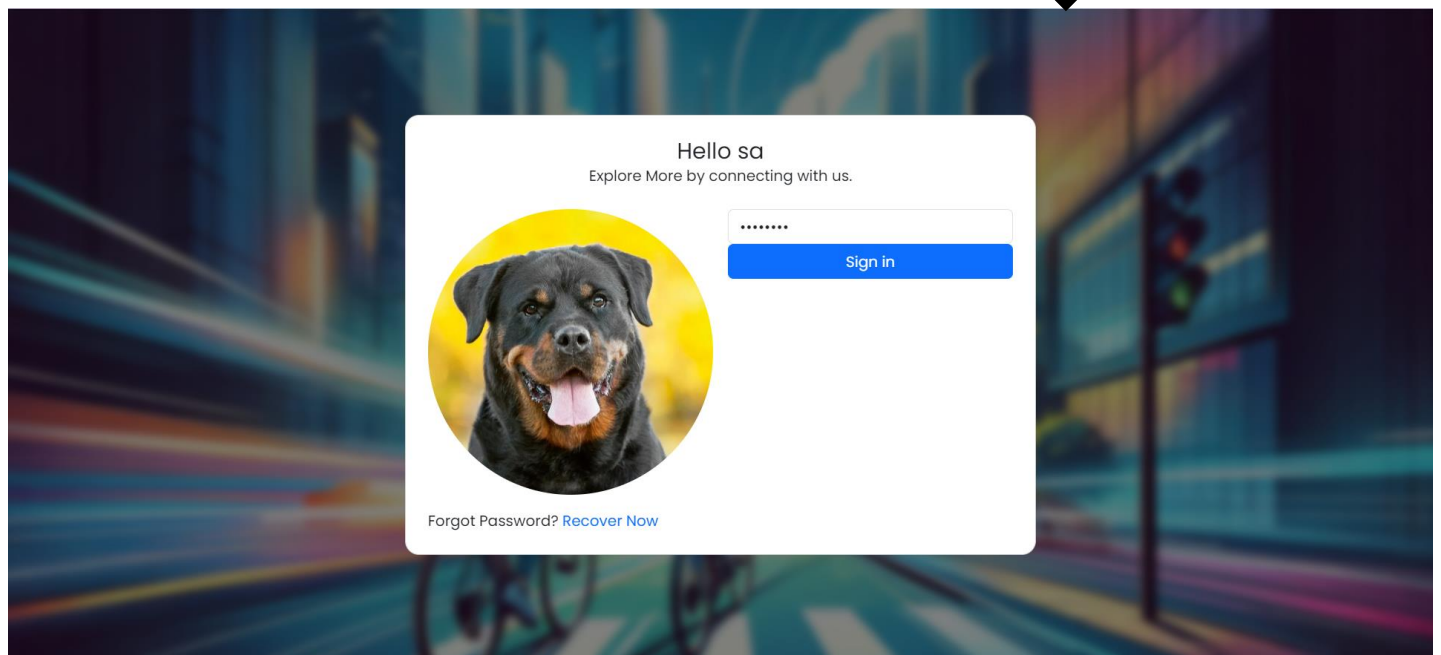
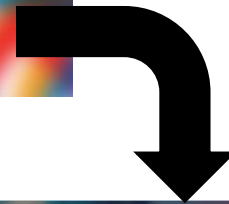
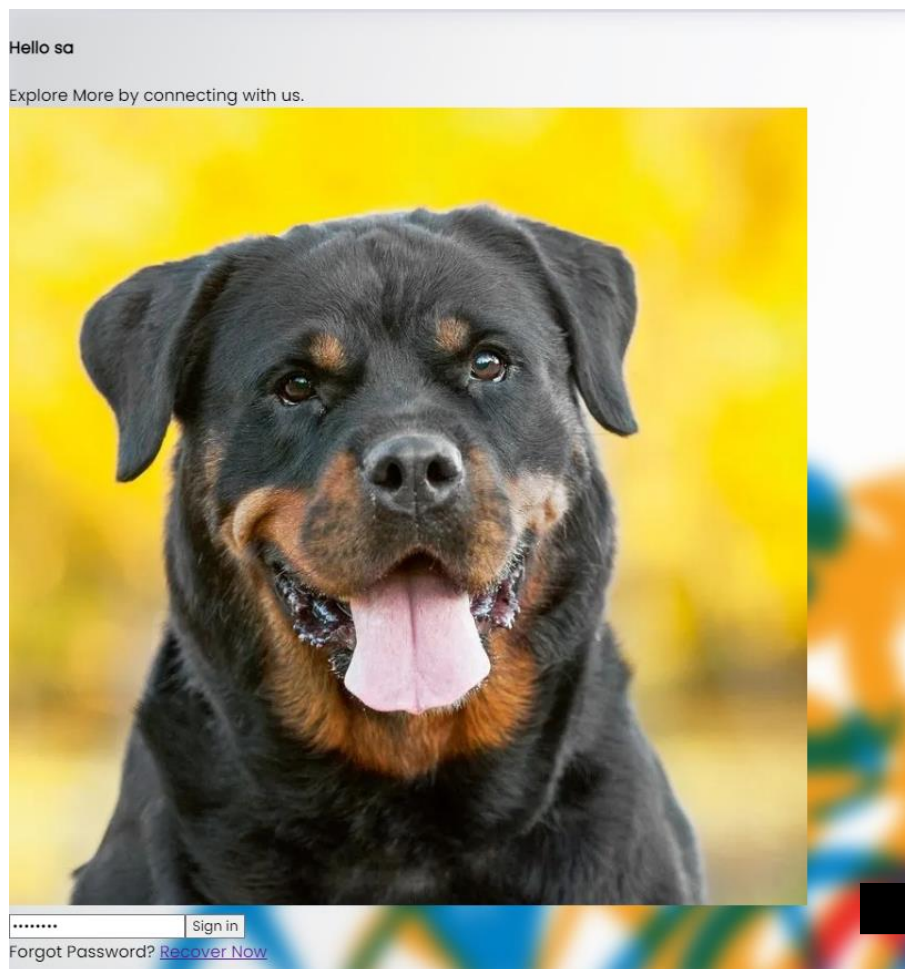



Figure 3.5: Amélioration Design page Password

Page de Profil :

Profile

You can update the details.



Choirir un fichier | Aucun fichier choisi


sa	be
9999	c666@gmail.com
42	Update

come back later? [Logout](#)



Profile

You can update the details.



sa

be

9999

c666@gmail.com

42

Update

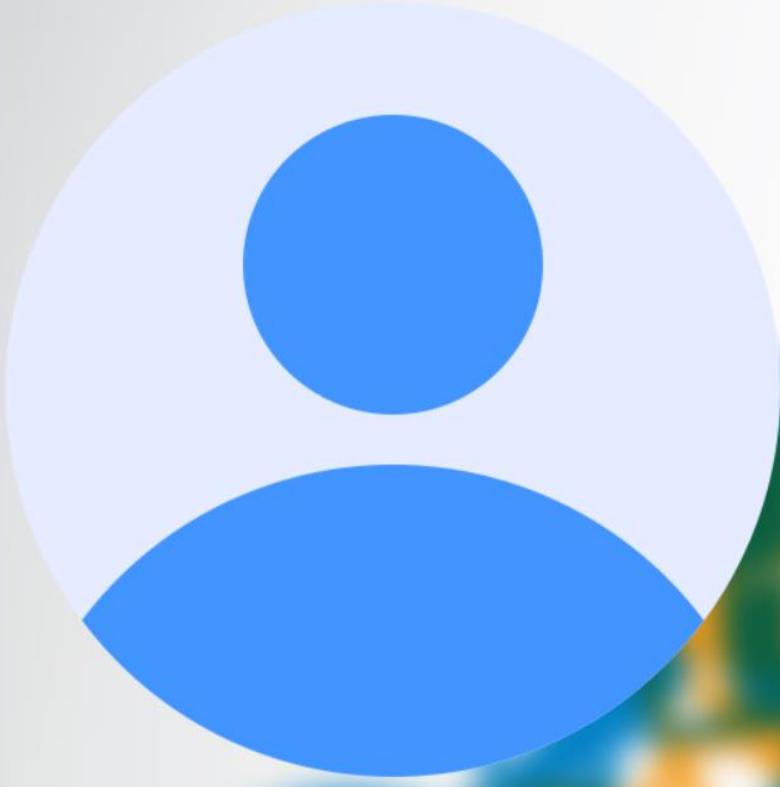
Come back later? [Logout](#)

Figure 3.6: Amélioration Design page Profile

Page de « Register »

Register

Happy to join you!



Choir un fichier | Aucun fichier choisi


c666@gmail.com	example123	admin@12	Register
----------------	------------	----------	----------

Already register? [Login now](#)



Register

Happy to join you!



Choir un fichier | Aucun ...r choisi

Already registered? [Login now](#)

c666@gmail.com
example123

Register

Figure 3.7: Amélioration Design page Register

Page de « Recover » :

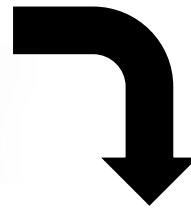
Recovery

Enter OTP to recover password.

Enter 6 digits OTP sent to your email address:

Can't get OPT?

Figure 3.8: Amélioration Design page Recover



Recovery

Enter OTP to recover password.

Enter 6 digits OTP sent to your email address.

Can't get OTP? [Resend](#)

Page de « Reset » du mot de passe :

Reset

Enter new password.

Figure 3.9 : Amélioration Design Page de Reset



Reset

Enter new password.

4

Conception et Développement du Site

I. Généralités

Une application web est composée de deux parties principales : le frontend et le backend.

- Le **frontend**, également connu sous le nom de partie côté client, est la partie de l'application web avec laquelle l'utilisateur interagit directement. Cela inclut l'interface utilisateur visible dans le navigateur web, comprenant la disposition, le design, le contenu et les fonctionnalités interactives telles que les formulaires, les boutons, etc.
- Le **backend**, ou partie côté serveur, est la partie invisible qui gère le fonctionnement interne du site web. Cela inclut le traitement des requêtes des utilisateurs, l'accès aux bases de données, l'authentification, la logique de l'application.

L'interaction entre le frontend et le backend est essentielle pour créer un site web fonctionnel et accessible en ligne. Ces deux parties travaillent de concert pour fournir une expérience utilisateur complète.

II. Technologies

Pour développer un site web, une variété d'outils et de bibliothèques sont disponibles. Ces outils sont souvent regroupés en ce qu'on appelle des piles technologiques, qui offrent une cohérence et une compatibilité entre les différentes technologies utilisées dans le développement d'une application web.

La pile MERN est un choix bien connu, largement documenté et entièrement open source dans laquelle j'ai déjà acquis une certaine expérience. C'est pourquoi elle a été choisie pour le développement du site. MERN est un acronyme représentant les technologies MongoDB, Express, React et Node.js.

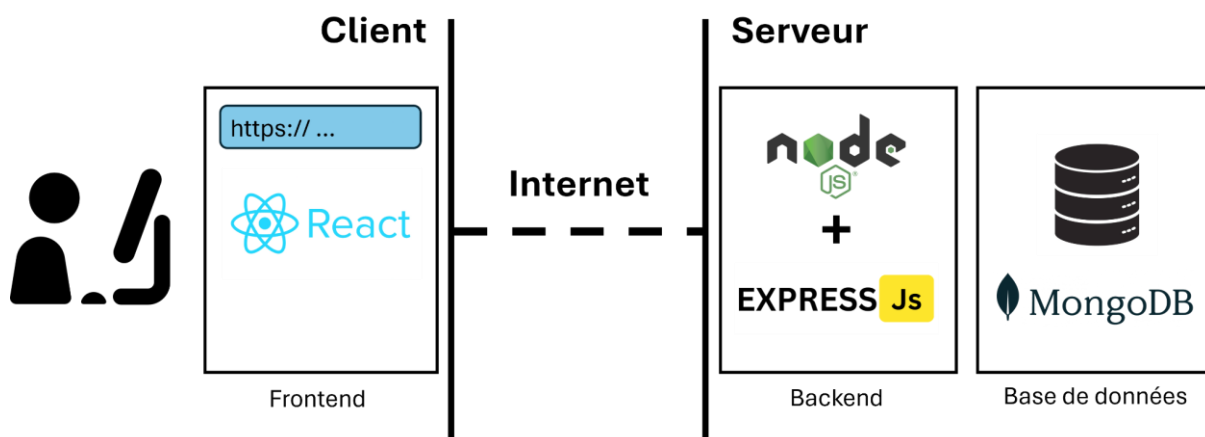


Figure 4.1 : Architecture d'une application web moderne avec MERN Stack

1. React JS

1.1 Fonctionnement général

React est un framework frontend, mieux décrit comme une bibliothèque, qui offre un cadre et de nombreuses fonctionnalités pour le développement côté client. Il se distingue par sa capacité à créer simplement des interfaces utilisateur interactives et réactives.

Une caractéristique clé de React est son système de composants, qui permet de diviser l'interface utilisateur en blocs modulaires réutilisables.

Un composant type est composé de 3 parties :

- Import des librairies
- Comportement de la page
- Code HTML à afficher

Dans le code, cela se traduit sous la forme suivante :

```
// importation de tous les librairies et composants nécessaires
import React from 'react';

// exportation d'un composant
export default function nom_du_composant() {

  // comportement de la page :
  // ... code JavaScript qui sera exécuté dans le navigateur

  // retourne le code HTML qui sera affiché dans le navigateur
  return (
    <div>
      {/** ... code HTML */}
    </div>
  )
}
```

1.2 Sous-composant

Dans ce projet, on établit la distinction entre les composants principaux, chargés d'afficher des pages et les sous-composants qui sont des parties intégrées dans des pages.

Par exemple, comme il sera détaillé plus loin dans le rapport, on peut créer un sous-composant "Bouton Déconnexion" qui gère la fonctionnalité de déconnexion.

```
/** logout button component */
export const LogoutButton = () => {
```

```

// déconnexion de l'utilisateur et redirection vers la page d'accueil
// lorsqu'on clique sur le bouton

return (
  <div>
    {/** Bouton */}
  </div>
);
};

```

Il suffit ensuite de l'inclure dans une ou plusieurs pages :

```

// importation du bouton de déconnexion
import { LogoutButton } from '../LogoutButton'

/** Une page du site */
export const page = () => {

  // déconnexion de l'utilisateur et redirection vers la page d'accueil
  // lorsqu'on clique sur le bouton

  return (
    <div>
      {/** Bouton de déconnexion */}
      <LogoutButton />
    </div>
  );
};

```

1.3 Fonctions hooks

Une fonction hook, élément important des dernières versions de React, est une fonction qui permet à un composant de gérer son propre état et d'accéder à d'autres fonctionnalités de React. Le hook le plus utilisé, `useState`, permet de créer et de mettre à jour des variables d'état à l'intérieur d'un composant.

Par exemple, on peut enregistrer et visualiser le nombre de clics effectués sur un bouton.

```
export function Composant() {  
  
  // Déclaration d'une nouvelle variable d'état  
  // pour le compteur de clics  
  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
        
      {/** Affiche le nombre de clics sur le bouton */}  
  
      <p>Vous avez cliqué {count} fois</p>  
  
        
      {/** Un bouton pour ajouter 1 au compteur */}  
  
      <button  
          
        // Mise à jour de la variable d'état lorsqu'on clique sur le bouton  
        onClick={() => setCount(count + 1)}  
      >  
        Ajouter 1 au compteur  
      </button>  
    </div>  
  );  
}
```

2. Node.js

Node.js est un environnement d'exécution JavaScript côté serveur, idéal pour le développement backend, offrant des performances élevées pour les applications web en temps réel et hautement interactives.

De plus, grâce à son système de gestion de modules, il permet d'ajouter facilement des fonctionnalités supplémentaires à une application, que ce soit des packages intégrés à Node.js ou des packages externes disponibles sur le registre npm, le gestionnaire de packages par défaut. Il permet d'installer, de partager et de gérer les dépendances des projets Node.js de manière simple et efficace.

3. Express JS

3.1 Fonctionnement général

Express.js est un framework web pour Node.js qui simplifie la création d'applications web en fournissant des fonctionnalités pour la gestion des routes et des requêtes HTTP. Il permet également de créer des API RESTful, qui sont des interfaces de programmation d'applications conçues pour être simples, évolutives et faciles à comprendre.

Les API RESTful permettent aux clients d'effectuer des opérations sur les ressources via des requêtes HTTP bien définies, comme la création (POST), la lecture (GET), la mise à jour (PUT) et la suppression de données (DELETE).

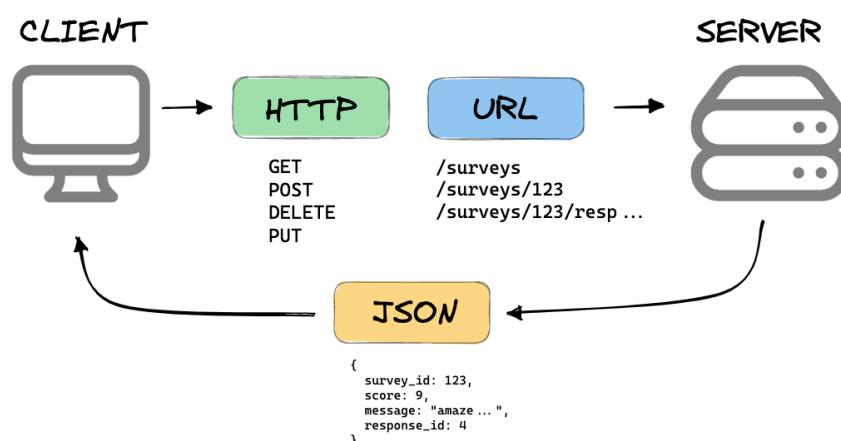


Figure 4.2 : Fonctionnement de la Communication Client-Serveur dans une Application Web

2.2 Structure d'un serveur

Le fichier principal, typiquement nommé server.js, est l'endroit où l'on déclare et lance l'application Express. Dans ce fichier, on configure notre application Express et on lui spécifie un ou plusieurs routeurs.

Un router, dans le contexte d'Express.js, est un composant qui permet de définir des routes pour l'application. Les routes sont des points d'entrée pour les requêtes HTTP et sont associées à des méthodes HTTP spécifiques. Un routeur peut être considéré comme un mini-ensemble de routes regroupées logiquement ensemble.

A chaque route sont associés un ou plusieurs contrôleurs. Les contrôleurs, dans le contexte d'une application Express.js, sont des fonctions qui contiennent la logique métier de notre application. Ils sont responsables de traiter les requêtes HTTP reçues par l'application, d'interagir avec la base de données si nécessaire, et de renvoyer les réponses appropriées aux clients.

Par exemple, la ligne de code ci-dessous spécifie que lorsque le serveur est questionné sur la route “/register” via une méthode POST, le contrôleur “register” s’occupe de traiter la requête.

```
router.route('/register').post(controller.register); // register user
```

4. MongoDB

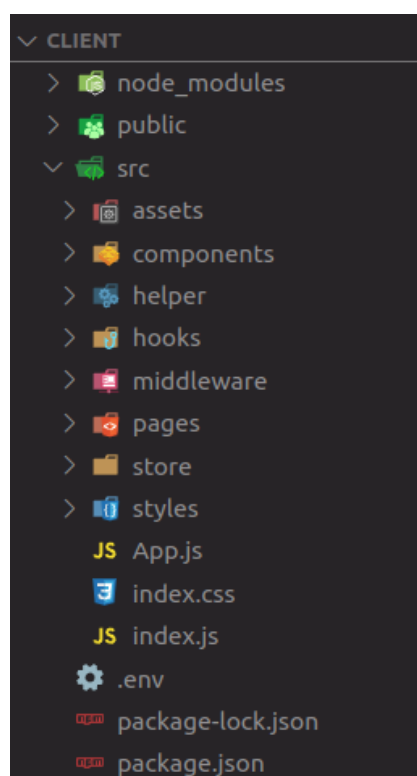
MongoDB est une base de données NoSQL qui stocke les données sous forme de documents JSON. Elle utilise des collections pour organiser les données similaires, un peu comme des dossiers. Par exemple, une collection "User" peut stocker les informations sur les utilisateurs, offrant ainsi une gestion flexible et évolutive des données.

III. Arborescence du projet

Ci-dessous, nous détaillons l'arborescence du projet afin de fournir une meilleure compréhension globale des différents éléments qui le composent.

1. Frontend

La partie client de l’application Web contient beaucoup de dossiers et de fichier :

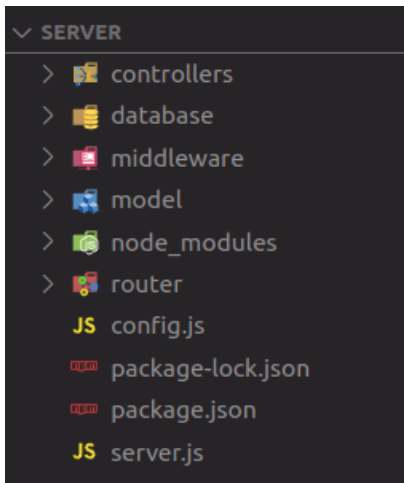


Voici le détail de certains d’entre eux :

- **node_modules** : contient tous les fichiers essentiels au bon fonctionnement des frameworks Node.js et Express, ainsi que les modules supplémentaires installés. Ces modules sont répertoriés avec leurs versions dans le fichier package.json.
- **pages** : regroupe tous les fichiers correspondant à la gestion des pages, telles que "profile" ou "login".

- **components** : rassemble divers éléments destinés à être affichés dans les pages.
- **helper** : comprend un ensemble de fichiers permettant d'exécuter diverses fonctions, comme la gestion des requêtes avec le serveur.
- **app.js** : associe les URL aux pages correspondantes. Par exemple, il définit que la page "login" est affichée lorsque l'URL se termine par "/login".

2. Backend



Voici un détail de certains des dossier et fichiers de cette partie :

- **server.js** : fichier permettant de configurer et lancer l'application côté serveur.
- **route.js** : définit les routes de l'API et les contrôleurs associés à chacune d'elles.
- **controllers** : contient les fichiers qui traitent les requêtes du client en effectuant différentes opérations.
- **config.js** : contient les informations importantes comme le chemin d'accès vers la base de données.
- **database** : contient des fichiers implémentant des fonctions relatives à la base de donnée comme la connexion à celle-ci
- **model** : dans ce dossier sont définis les modèles de données attendus pour les documents, en fonction des collections. Par exemple, les documents de la collection "User" doivent tous posséder le champ "username". Cela assure une approche plus rigoureuse de la gestion des données.

IV. Gestion des comptes utilisateur

Dans cette partie, le but est d'assurer à l'utilisateur de pouvoir créer un profil. Les fonctionnalités suivante sont donc à développer :

- Création d'un nouveau compte
- Connexion à un compte existant via un nom d'utilisateur et un mot de passe
- Modification du profil utilisateur
- Récupération de mot de passe

Afin de simplifier la gestion des données, en particulier lors des requêtes ultérieures dans la base de données, on impose que les noms d'utilisateur et les adresses e-mail des utilisateurs soient uniques.

1. Modules frontend récurrents

Dans le frontend, il y a 2 modules utilisés dans la plupart des pages :

- react-hot-toast
- formik

1.1 react-hot-toast

Un “toast” dans le développement web fait référence à une petite fenêtre pop-up qui va généralement afficher l'état d'une opération.

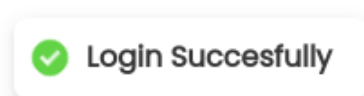


Figure 4.3: Fenêtre pop-up

Le module react-hot-toast facilite l'affichage des “toasts” et prend notamment en charge les promesses.

Une promesse (promise en anglais) en JavaScript est un objet représentant le résultat d'une opération asynchrone. Elle peut être dans 3 états, en attente, résolue ou rejetée.

Dans l'exemple ci-dessous, on affiche différents messages à l'utilisateur en fonction de l'état de la promesse “myPromise”.

```
toast.promise(myPromise, {  
  loading : 'chargement ...',  
  success : <b>Opération réussie</b>,  
  error : <b>Une erreur est survenue</b>  
});
```

1.2 formik

Les formulaires en JavaScript permettent aux utilisateurs de saisir et d'envoyer des données dans les applications web. La bibliothèque Formik simplifie la gestion des formulaires dans les applications React en fournissant des utilitaires pour gérer l'état, la validation des données, la soumission du formulaire et la gestion des erreurs.

Dans cet exemple, Formik est utilisé pour gérer le formulaire en stockant et en mettant à jour les valeurs des champs “firstName” et “lastName”. La fonction “donneesValides”, spécifiée dans le champ “validate”, est une opération facultative qui permet de vérifier les données avant de les envoyer. Par exemple, si le nom de famille

contient des chiffres, on pourrait choisir de ne pas envoyer le formulaire. Lorsque le formulaire est soumis, les valeurs sont envoyées via la fonction "onSubmit".

```
import { useFormik } from 'formik';

export default function MyForm() {

  // Utilisation de formik pour gérer le formulaire

  const formik = useFormik({

    initialValues: {

      firstName: '',

      lastName: ''

    },

    validate : donneesValides,

    onSubmit: values => {

      // Envoi des données du formulaire

    }

  });

  return (

    <form onSubmit={formik.handleSubmit}>

      <input {...formik.getFieldProps('firstName')} type="text" />

      <input {...formik.getFieldProps('lastName')} type="text" />

      <button type="submit">Envoyer</button>

    </form>

  );

};
```

2. Création de compte

Dans cette section, l'utilisateur a la possibilité de fournir un nom d'utilisateur, une adresse e-mail, un mot de passe et une photo de profil. Une fois l'opération de création de compte réussie, un e-mail de confirmation lui est envoyé.

2.1 Frontend

La page "register.js", accessible à l'adresse /register, permet à l'utilisateur de saisir ses informations.

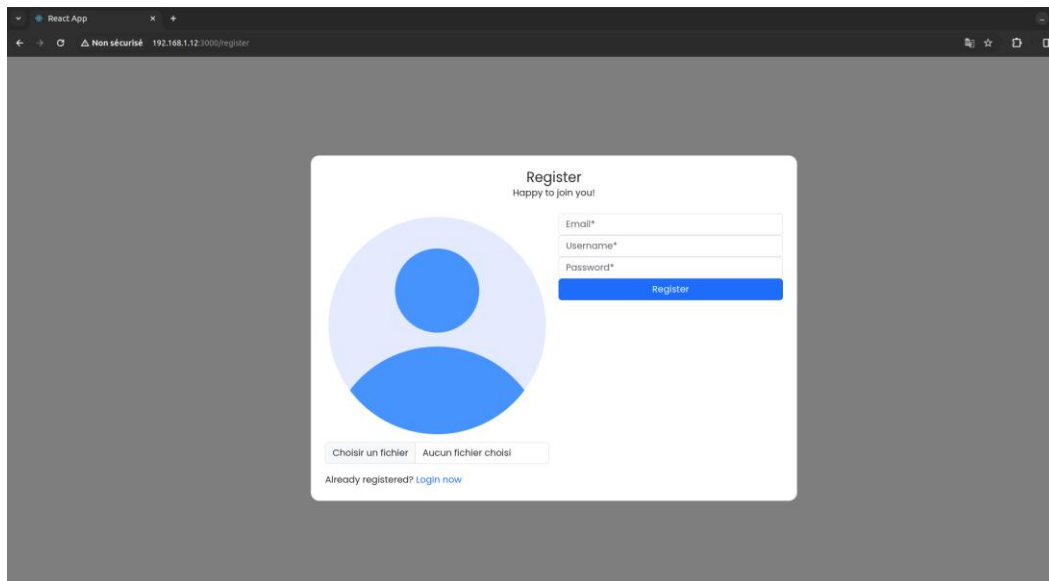


Figure 4.4: Page /Register

Les champs “Email”, “Username” et “Password” sont gérés sous la forme d’un formulaire avec formik.

Avec la propriété “validate” de ce module, la fonction “registerValidate” s’assure que :

- le nom d’utilisateur ne contient pas d’espace et n’est pas vide
- le format de l’adresse e-mail est correct (xxx@xx.xx)
- le mot de passe ne contient pas d’espaces et possède au moins 4 caractères dont un caractère spécial

L’image de profil est recadrée dans un format carré puis convertie en base 64. Autrement dit, l’image est transformée en texte au format ASCII, une méthode couramment utilisée pour transmettre des données sur Internet.

Voici le profil qui servira d’exemple tout au long de la partie gestion des comptes utilisateur.

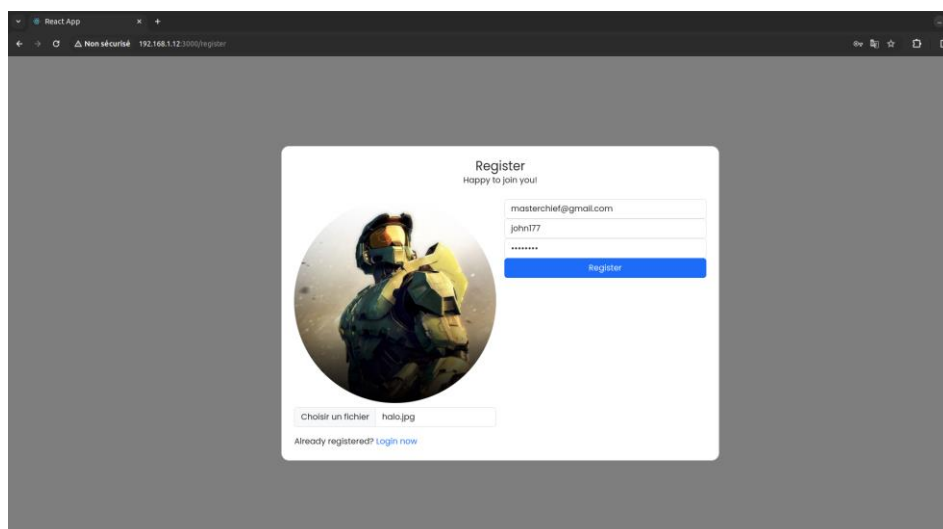


Figure 4.5: Page /Register complétée

Lorsque l'utilisateur appuie sur le bouton "register", la fonction "registerUser" située dans le fichier userHelper.js du dossier helper prend le relais. Si elle s'exécute correctement, l'utilisateur est ensuite redirigé vers la page de connexion.

La fonction "registerUser" effectue deux actions principales :

- Elle envoie les données au serveur pour demander la création d'un nouveau compte sur la route /register.
- Elle demande également au serveur d'envoyer un e-mail de confirmation sur la route /registerMail.

Pour effectuer ces tâches, le module "axios" est utilisé. Axios est un client HTTP qui permet d'effectuer des requêtes, notamment les requêtes POST, GET, PUT et DELETE nécessaires dans ce contexte.

2.2 Backend

Pour enregistrer un utilisateur dans la base de données MongoDB, il est nécessaire de définir un modèle. Ce modèle détermine la structure des données que doivent avoir les documents de la collection utilisateur dans la base de données. En d'autres termes, il définit les champs et les types de données attendus pour chaque utilisateur enregistré.

Ainsi dans le fichier User.model.js dans le dossier model, avec l'aide du module mongoose, on impose la structure de donnée suivante :

```
import mongoose from "mongoose";

export const UserSchema = new mongoose.Schema({

  username : {

    type : String,

    required : [true, "Please provide unique Username"],

    unique : [true, "Username Exist"]

  },

  password : {

    type : String,

    required : [true, "Please provide a password"],

    unique : false

  },

  email : {
```

```

    type: String,
    required : [true, "Please provide a unique email"],
    unique: true
  },
  firstName : { type: String},
  lastName : { type: String},
  mobile : { type : Number},
  address : { type: String},
  profile : { type: String}
});

```

Comme spécifié précédemment, le nom d'utilisateur et l'adresse e-mail sont bien uniques. De plus, hormis le mot de passe, les autres champs sont optionnels. Avec cette approche, il est facile de modifier ultérieurement la structure de la base de données.

a) Enregistrement d'un compte

```

router.route('/register').post(controller.register);

```

Le contrôleur register associé à la route /register s'occupe de la création d'un compte utilisateur. En premier lieu, cette fonction interroge la base de données pour vérifier que le nom d'utilisateur et l'adresse e-mail sont uniques. Si c'est le cas, les données reçues via la requête sont enregistrées via le modèle UserSchema.

Il est cependant essentiel de hacher un mot de passe avant de le stocker dans une base de données pour des raisons de sécurité. Le hachage transforme le mot de passe en une chaîne de caractères aléatoire, rendant impossible de le récupérer en clair même si la base de données est compromise.

Pour accomplir cela, il existe bcrypt, une bibliothèque de hachage de mots de passe couramment utilisée. Elle utilise un algorithme sécurisé et lent, offrant une protection supplémentaire contre les attaques de force brute.

```

bcrypt.hash(password, 10)

```

Ainsi, le résultat du point de vue de la base de données est le suivant :


```
_id: ObjectId('65c88d6ff3767e632136012e')
username: "john177"
password: "$2b$10$d4hIpcsX6g328ryDA86EuE7xG29dn0FixtElaoPjyYlHGBdRTc0"
email: "masterchief@gmail.com"
profile: "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAQABAAD/4gHYSUNDX1BST0ZJTEU..."
```

Figure 4.6: Document utilisateur MongoDB

b) Mail de confirmation


Le contrôleur registerMail, associé à la route /registerMail, gère l'envoi d'un e-mail à l'utilisateur à l'adresse e-mail qu'il a fournie.

L'e-mail est généré avec le module Mailgen et il est envoyé avec le module Nodemailer.

```
import nodemailer from 'nodemailer';
import Mailgen from 'mailgen';

let transporter = nodemailer.createTransport(nodeConfig);
let MailGenerator = new Mailgen()
```

Pour simplifier le test de cette fonctionnalité, il existe un outil en ligne appelé Ethereal. Il s'agit d'un faux service SMTP parfaitement compatible avec nodemailer. Ethereal permet de générer une adresse e-mail temporaire, qui remplace l'adresse e-mail réelle lors de l'envoi d'e-mails, ce qui permet de simuler l'envoi d'e-mails aux adresses e-mail des utilisateurs pour les tests.


Ethereal
Home
FAQ
Help
Messages

Logout

Account created

Here you can find all the details needed to access your new Ethereal test account. Remember that if sending messages through SMTP then no message is actually delivered, all messages are caught and you can see these in the [Messages](#) page or by using your favorite IMAP/POP3 client.

Account credentials

Account details generated using [Faker.js](#)

NB! these credentials are shown only once. If you do not write these down then you have to create a new account.

Name	Jacques Miller
Username	jacques.miller31@ethereal.email
Password	GaW3UcDmJhrRNwhhf8

Download as CSV

Open Mailbox

Nodemailer configuration

```
const transporter = nodemailer.createTransport({
  host: 'smtp.ethereal.email',
  port: 587,
  auth: {
    user: 'jacques.miller31@ethereal.email',
    pass: 'GaW3UcDmJhrRNwhhf8'
  }
});
```

Figure 4.7: Création d'une adresse mail avec Ethereal

3. Connexion

Dans cette section, l'utilisateur peut se connecter à un compte entrant dans un premier temps le nom d'utilisateur, puis si le compte existe, le mot de passe.

3.1 Frontend

a) Nom d'utilisateur

La page "username.js", accessible à l'adresse /login, permet à l'utilisateur de saisir son nom d'utilisateur.

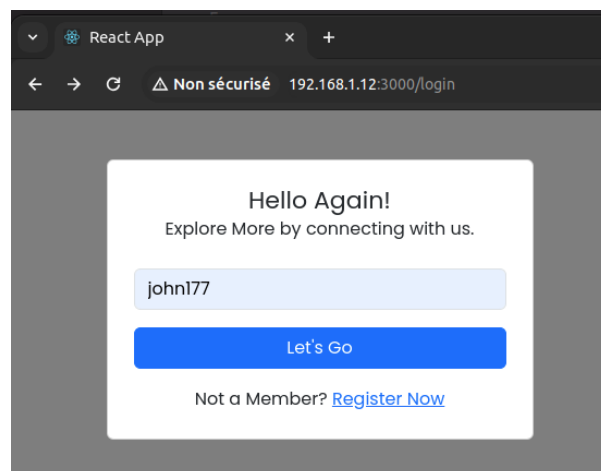


Figure 4.8: Page /login complétée

Ce champ est géré via un formulaire formik. Avec la propriété “validate” de ce module, la fonction “usernameValidate” s’assure qu’en envoyant une requête au serveur que le nom d’utilisateur est déjà enregistré dans la base de données.

Si c’est le cas, le nom d’utilisateur est stocké localement et l’utilisateur est redirigé vers la page permettant d’entrer le mot de passe.

Pour stocker le mot de passe, le module Zustand de Node.js, une bibliothèque légère pour la gestion de l’état dans les applications React, est utilisé. Il permet de créer facilement des “stores” pour stocker des données localement.

b) Mot de passe

La page “password.js”, accessible à l’adresse /password, permet à l’utilisateur de saisir son mot de passe.

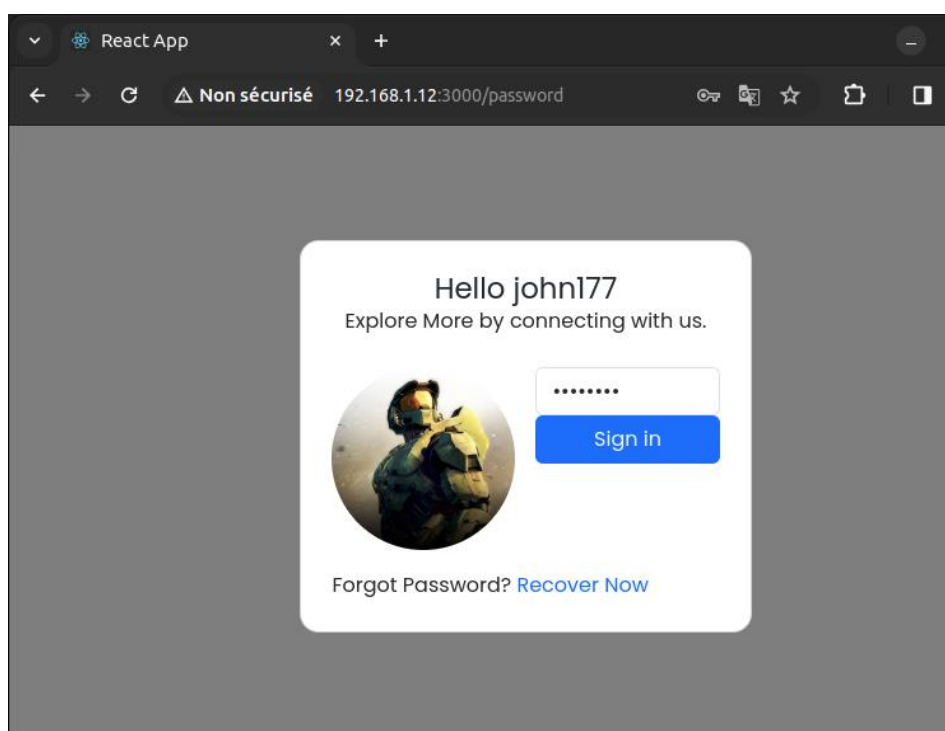


Figure 4.9: Page /password

Ce champ est géré via un formulaire formik. Une requête contenant le mot de passe et le nom d'utilisateur récupéré sur cette page via le “store” est envoyé au serveur sur la route /login.

Si le mot de passe est correct, le serveur retourne un token, c'est-à-dire une petite unité de données utilisée pour l'authentification et l'autorisation. Il est émis après une connexion réussie et est utilisé pour prouver l'identité de l'utilisateur dans les requêtes ultérieures.

```
let { token } = response.data;  
localStorage.setItem('token', token);
```

L'utilisateur est ensuite redirigé vers la page principale de l'application avec la carte.

c) Déconnexion

Pour déconnecter l'utilisateur, il suffit de supprimer le token stocké localement. Cela peut être réalisé en créant un composant avec un simple bouton qui effectue cette action. Ce composant peut ensuite être inclus dans toutes les pages où la déconnexion est nécessaire, offrant ainsi une solution réutilisable.

```
export const LogoutButton = () => {  
  
  const navigate = useNavigate();  
  
  const userLogout = () => {  
  
    // Suppression du token  
  
    localStorage.removeItem('token');  
  
    // Retour à la page d'accueil  
  
    navigate('/');  
  
  };  
  
  return (  
  
    <div className="text-center">  
  
      <span>  
  
        Come back later?{' '}  
  
        <button onClick={userLogout}>Logout</button>  
  
      </span>  
  
    </div>  
  
  );  
};
```

3.2 Backend

```
router.route('/login').post(controller.login);
```

Le contrôleur login, associé à la route /login, gère la connexion à un compte utilisateur. Cette fonction récupère le document dans la base de données correspondant au nom d'utilisateur transmis dans la requête, puis vérifie que les mots de passe correspondent.

Si tout coïncide, un JWT est créé. Un JWT (JSON Web Token) est un format de token sécurisé utilisé pour transférer des réclamations entre deux parties. Il se compose de trois parties : l'en-tête, la charge utile et la signature, et est couramment utilisé pour l'authentification et l'autorisation dans les applications web.

Le token, valide 24 heures, est généré avec la bibliothèque jsonwebtoken.

```
const token = jwt.sign({
  userId : user._id,
  username : user.username
}, ENV.JWT_SECRET, {expiresIn : "24h"});
```

Ici, la variable JWT_SECRET correspond à une clé secrète préalablement générée, utilisée pour signer et vérifier l'intégrité du JWT.

Ainsi à partir du moment où l'utilisateur est connecté, toutes les requêtes effectuées passeront par la vérification du token. Le middleware Auth.js permet de réaliser cette authentification. Un middleware est une fonction intermédiaire utilisée dans les applications web pour intercepter et traiter les requêtes HTTP avant qu'elles n'atteignent leur destination finale.

```
// Récupération du token dans la requête
const token = req.headers.authorization.split(" ")[1];
// Décodage du token
const decodedToken = await jwt.verify(token, ENV.JWT_SECRET);
// Mise à jour de la requête avec le token décodé
req.user = decodedToken;
// Passer au contrôleur suivant
next()
```

4. Modification du profil utilisateur

4.1 Frontend

La page “Profile.js”, accessible à l’adresse /profile, permet à l'utilisateur de mettre à jour son profil. Les champs sont gérés via un formulaire formik.

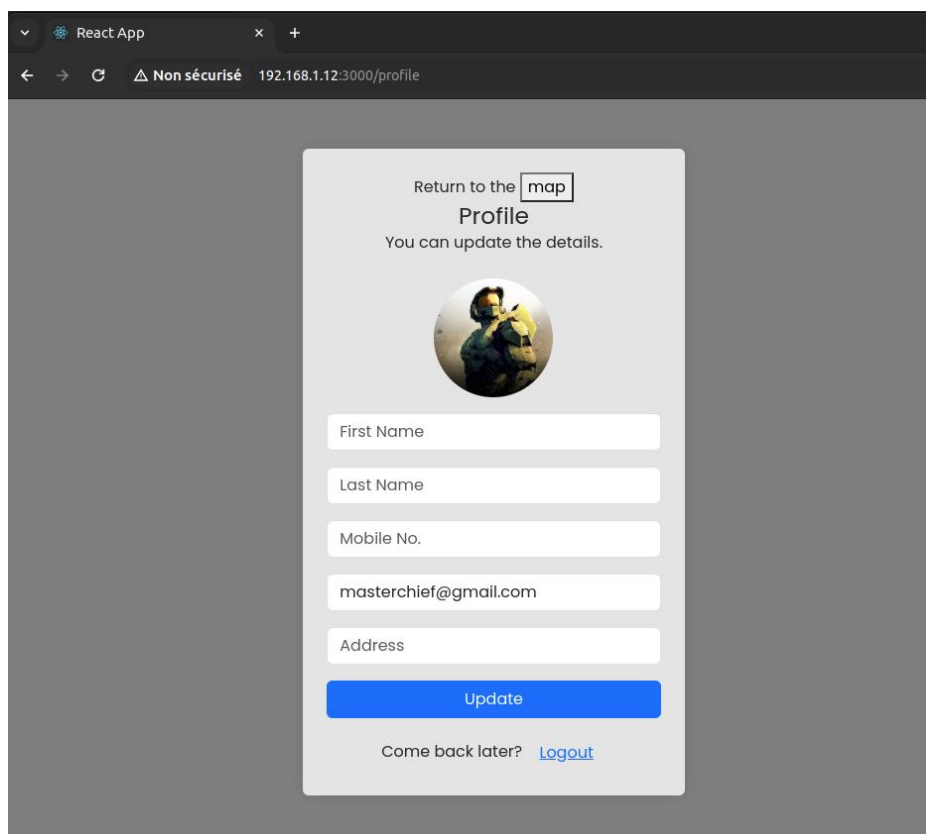


Figure 4.10 : Page /profile

Au chargement de la page, une requête est envoyée au serveur pour récupérer les données relatives à l'utilisateur.

Un custom Hook est utilisé afin d'envoyer la requête et connaître l'état de la demande. Cela permet, si la base de données n'est pas correctement remplie pour certaines raisons ou si l'utilisateur a été supprimé entre temps, de gérer les différents cas plus efficacement.

```
const [{ isLoading, apiData, serverError }] = useFetch()
```

Pour l'instant, seul un message est affiché à la place de la page :

```
if (isLoading) {  
  return <h1>isLoading</h1>  
};
```

```
if (serverError) {
  return <h1>{serverError}</h1>
}
```

De plus, dans la requête faite au serveur, le token est envoyé.

```
const data = await axios.put('/api/updateuser', response, { headers : {
  "Authorization" : `Bearer ${token}`}}});
```

4.2 Backend

```
router.route('/updateUser').put(Auth, controller.updateUser);
```

Le contrôleur updateUser, associé à la route /updateUser, gère la mise à jour des informations de l'utilisateur. Étant donné que l'utilisateur est censé être déjà connecté, l'authentification est vérifiée préalablement à l'aide du token

5. Récupération de mot de passe

5.1 Frontend

a) Démarrage de la procédure

La page “Recovery.js”, accessible à l’adresse /recovery, permet à l'utilisateur de démarrer la procédure de récupération de mot de passe en entrant un code reçu par mail.

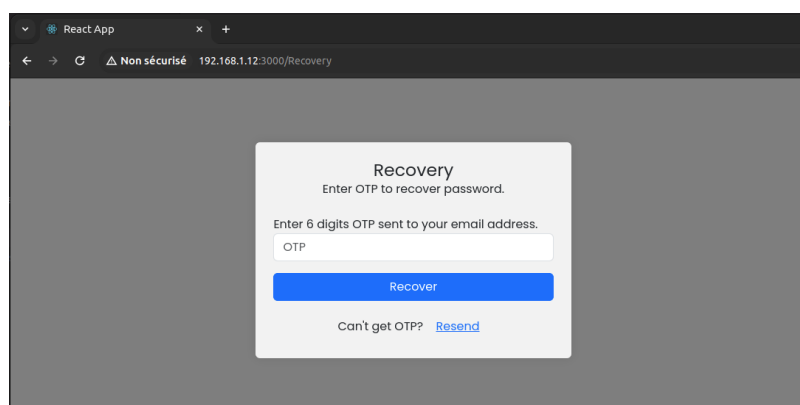


Figure 4.11: Page /Recovery

Au chargement de la page, une requête est envoyée au serveur pour demander la génération d’un OTP (one time password) et de son envoi par mail sur la route /generateOTP. Il y a également un bouton Resend qui effectue la même action sur demande de l'utilisateur.

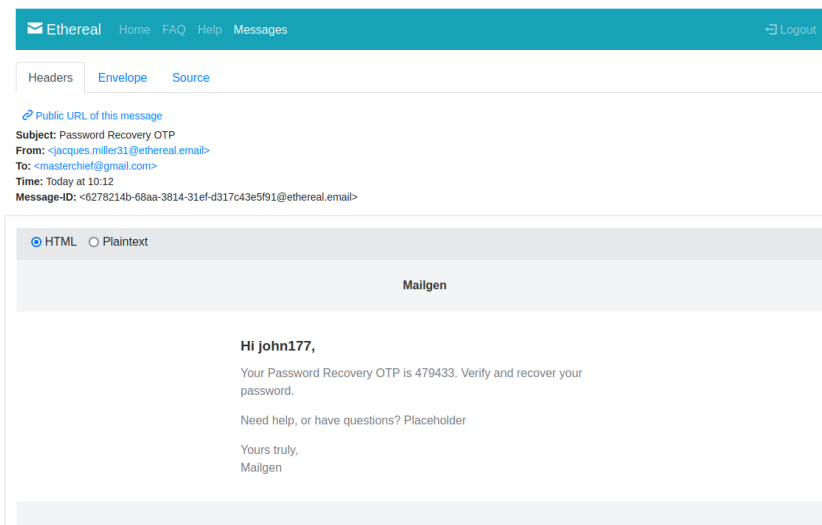


Figure 4.12: Mail Ethereal

Une fois l’OTP reçu par mail et rempli, une nouvelle requête est envoyée au serveur sur la route `/verifyOTP` pour vérifier que le code entré est le bon. Si c’est bon, l’utilisateur est redirigé vers la page de réinitialisation de mot de passe.

b) Réinitialisation du mot de passe

La page “Reset.js”, accessible à l’adresse `/reset`, permet à modifier le mot de passe de l’utilisateur.

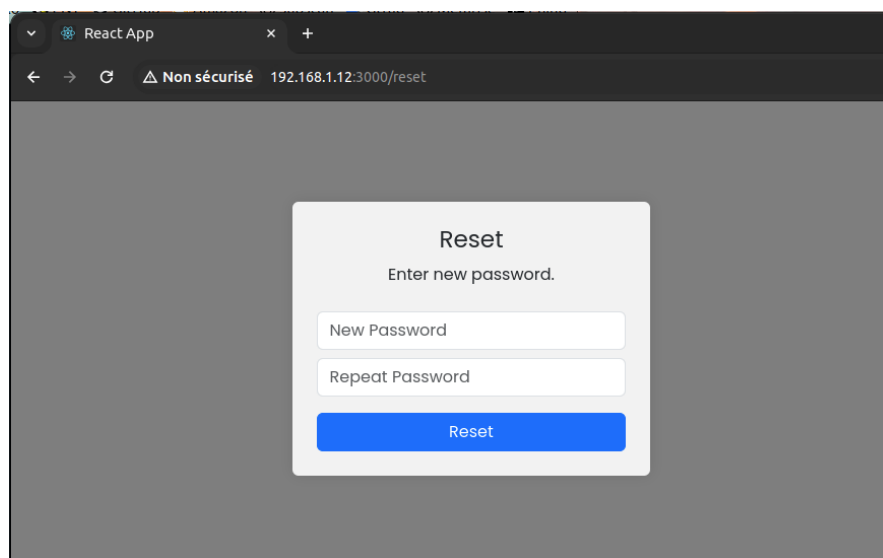


Figure 4.13: Page /Reset

Ces champs sont gérés via un formulaire formik. Avec la propriété “validate” de ce module, la fonction “resetPasswordValidate” s’assure que les mots de passes entrés sont identiques et qu’ils respectent les mêmes critères que lors de la création de compte.

Le nouveau mot de passe est ensuite envoyé au serveur sur la route /resetPassword.

5.2 Backend

a) Génération de l'OTP

```
router.route('/generateOTP').get(localVariables, controller.generateOTP);
```

Le contrôleur generateOTP, associé à la route /generateOTP, permet de générer un OTP via la bibliothèque otp-generator.

L'OTP doit être disponible jusqu'à que l'utilisateur demande la vérification du code qu'il a reçu par mail. Le middleware localVariables permet de créer des variables locales, une pour accueillir l'OTP et l'autre pour agir comme un flag pour savoir quand peut commencer la session de réinitialisation de mot de passe.

```
export function localVariables(req, res, next) {  
  req.app.locals = {  
    OTP : null,  
    resetSession : false  
  }  
  
  next()  
}
```

Ainsi l'OTP est stocké localement côté serveur lors de sa génération :

```
req.app.locals.OTP = await otpGenerator.generate(...);
```

b) Vérification de l'OTP

```
router.route('/verifyOTP').get(controller.verifyOTP);
```

Le contrôleur verifyOTP, associé à la route /verifyOTP, permet de vérifier si l'OTP entré par l'utilisateur est correcte.

Si le code est correct, l'OTP enregistré en mémoire est supprimé et le drapeau autorisant le changement de mot de passe est activé.

```
req.app.locals.OTP = null;  
req.app.locals.resetSession = true;
```

c) Mise à jour du mot de passe

```
router.route('/resetPassword').put(controller.resetPassword);
```

Le contrôleur `resetPassword`, associé à la route `/resetPassword`, permet de mettre à jour le mot de passe de l'utilisateur.

Une fois le mot de passe modifié dans la base de données, le drapeau autorisant le changement de mot de passe est désactivé, obligeant ainsi l'utilisateur à refaire toute la procédure s'il souhaite à nouveau modifier le mot de passe.

```
req.app.locals.resetSession = false;
```

V. Gestion des trajets

La page "Map.js", accessible à l'adresse `/map`, permet de gérer les trajets. Dans cette partie, le but est de permettre à l'utilisateur de :

- Implémenter une carte interactive
- Créer des trajets
- Sauvegarder des trajets et leur associer des horaires et des dates
- Charger et modifier les trajets enregistrés
- Afficher le trajet similaire
- Implémenter la correspondance des trajets

Étant donné que cette partie contient beaucoup de code lié à la manipulation des données et des éléments d'affichage, elle se concentrera sur l'architecture et les principes de fonctionnement généraux.

1. Carte interactive

Le projet utilise la bibliothèque open source `react-leaflet` qui permet d'intégrer facilement une carte interactive `OpenStreetMap` dans une application `react`. `OpenStreetMap` est une carte du monde libre et collaborative, constamment mise à jour par une communauté mondiale de contributeurs.

Voici les composants essentiels utilisés pour afficher la carte :

- `<MapContainer>` : crée un conteneur pour afficher la carte
- `<TitleLayer>` : fournit le fond de la carte
- `<MapClickHandler>` : composant personnalisé qui gère les clics sur la carte

```
<MapContainer  
  
  center={[48.65, 6.15]}  
  
  zoom={17}  
  
  style={{height: '700px', width: '700px'}}>
```

```
>

<TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"/>

  { /* Insérer tous les éléments à afficher sur la carte */ }

<MapClickHandler />

</MapContainer>
```

Avec cette base il est possible de se déplacer sur la carte et de modifier le niveau de zoom.

Le module react-leaflet offre également des composants à utiliser sur la carte comme des marqueurs et des lignes.

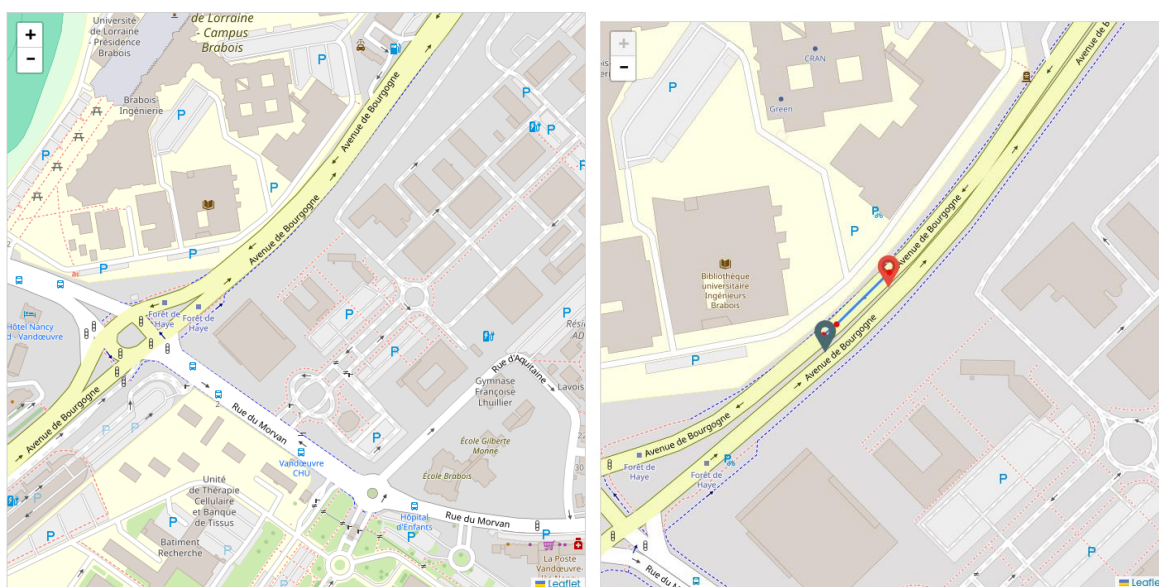


Figure 4.14: Carte Interactive avec Leaflet

2. Création de trajet

L'utilisateur a la possibilité de choisir un point d'arrivée et un point de départ.

```
const [startPoint, setStartPoint] = useState(null);
const [endPoint, setEndPoint] = useState(null);
```

Quand ces deux points sont définis, le client envoie une requête au serveur pour récupérer le plus court chemin entre ces 2 points. Comme vu dans la partie de Sabri, cela se fait via l'API de OpenRouteService.

Côté serveur, la requête est réceptionnée par la route /shortestPath.

```
router.route('/shortestPath').post(mapController.shortestPath)
```

Avec la liste de points reçus par le serveur, le chemin est affiché sur la carte.

```
const [receivedPoints, setReceivedPoints] = useState([]);
```

☑ Point de départ

□ Point d'arrivée

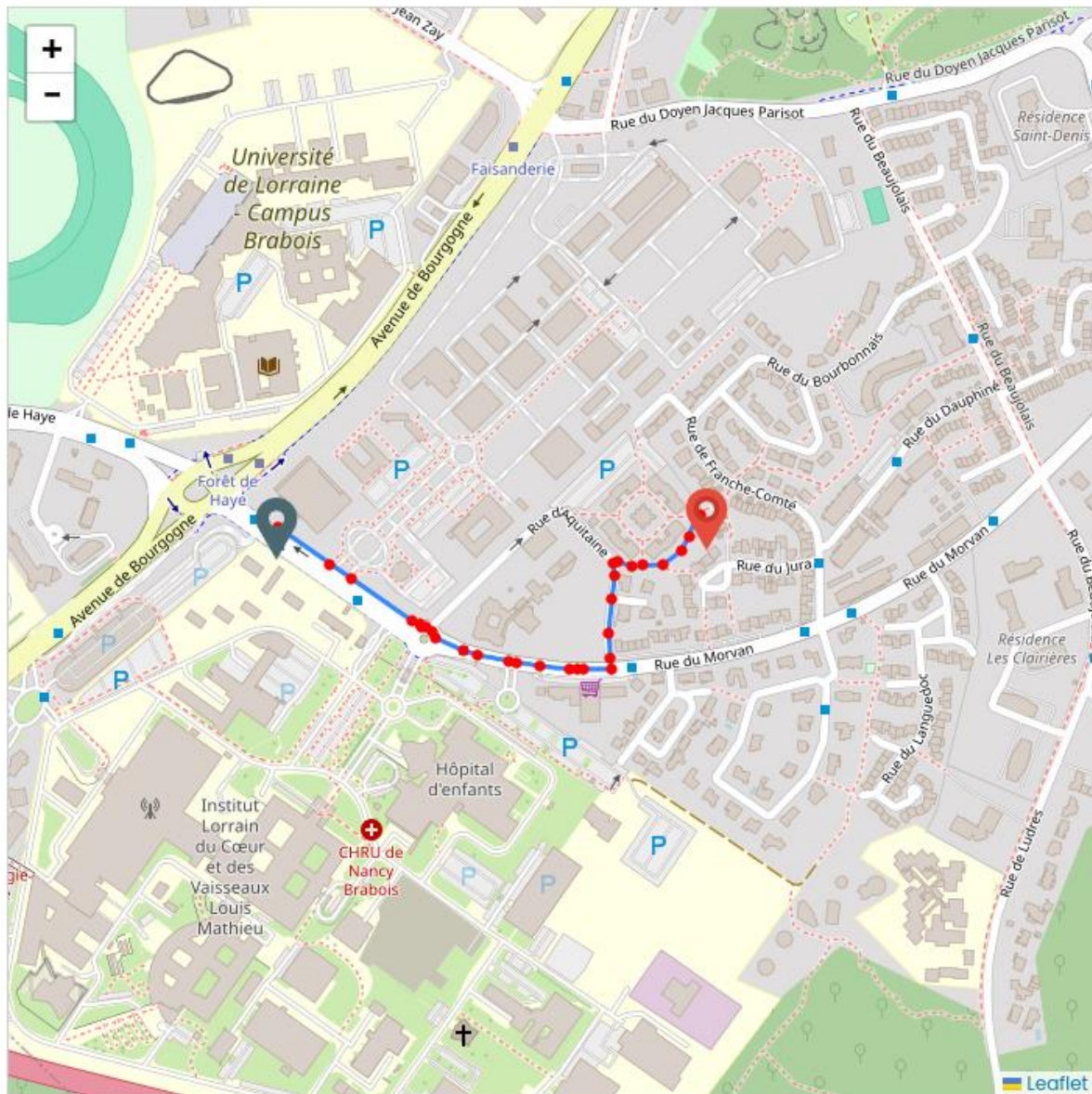


Figure 4.15: Affichage d'un Trajet sur la carte

Après l'apparition du trajet sur la carte, l'utilisateur peut déplacer les points pour personnaliser l'itinéraire selon ses préférences. Les points modifiés sont ensuite enregistrés dans une liste distincte, puis le serveur recalcule le chemin le plus court entre chaque paire de points.


```
const [intermediatePoints, setIntermediatePoints] = useState([]);
```

Quand la nouvelle liste de points arrive du serveur, les indices des points intermédiaires déjà existant sont mis à jour pour que l'ordre reste cohérent.

```
export async function updateIndex(receivedPoints, intermediatePoints)
```

3. Sauvegarde des trajets

L'utilisateur a la possibilité d'enregistrer des trajets en leur associant des dates ponctuelles ou récurrentes. Pour simplifier les opérations, les trajets enregistrés par un même utilisateur doivent avoir des noms différents.

La partie 'trajet courant', qui remplit les informations d'un trajet et permet de le sauvegarder, est gérée par un composant nommé CreateRouteForm.js.

Trajet courant

- 06/02/2024 04:35
- 21/02/2024 04:25

Horaires périodiques :

Jour de la semaine : Vendredi

Heure : 14:45

x

+

- Lundi 06:30
- Vendredi 14:45

Créer trajet

☐ Point de départ
 ☒ Point d'arrivée

Figure 4.16: Création et sauvegarde des trajets

Lorsque que l'on crée un trajet, toutes les informations sont envoyées au serveur puis stockées dans la collection "routes" de la base de données.

Côté serveur, la requête est réceptionnée par la route /shortestPath.

```
router.route('/addroute').post(Auth, appController.addRoute);
```

La structure de données est définie dans le fichier Route.model.js, et elle produit le résultat suivant :

```

    _id: ObjectId('65c991ab6958c07dc21326fc')
  username: "john177"
  name: "Trajet_1"
  route: Array (53)
  planning: Object
    dates: Array (2)
      0: 2024-02-06T03:35:00.000+00:00
      1: 2024-02-21T03:25:00.000+00:00
    periodic: Array (2)

```

Figure 4.17: Document route MongoDB

4. Chargement des trajets

L'utilisateur peut visualiser les trajets qu'il a enregistrés, de les supprimer et de les sélectionner pour les charger.

La partie "Mes Trajets" affichant la liste de tous les trajets créés par un utilisateur est gérée par un composant nommé ListRouteForm.js.

Il est possible de les supprimer en appuyant sur le bouton "supprimer". Côté serveur, la requête est réceptionnée par la route /deleteRoute.

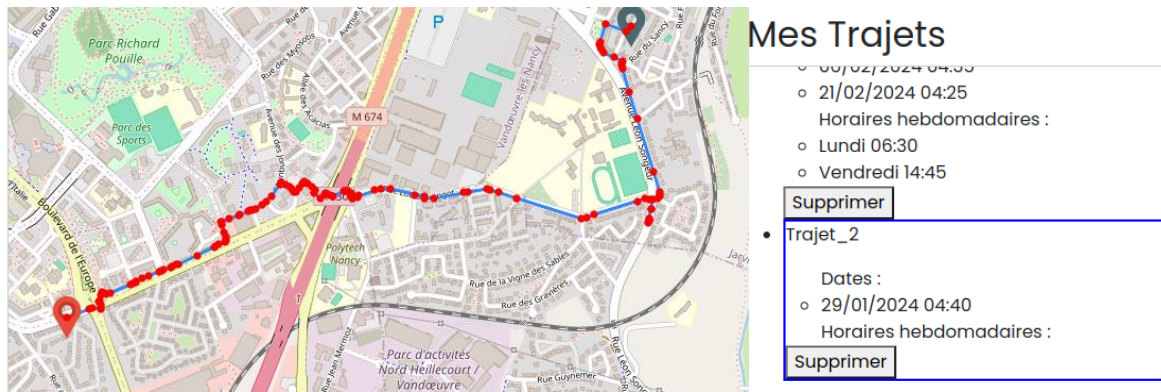
```
router.route('/deleteRoute').delete(Auth, appController.deleteRoute);
```

Il est également possible de mettre à jour un trajet. Lorsque l'utilisateur clique sur un élément de la liste, celui-ci charge le chemin sur la carte et met à jour les informations de la section "Trajet Courant". Ensuite, lorsque l'utilisateur appuie sur le bouton pour modifier le trajet et que le nom existe déjà, les informations du trajet sont envoyées au serveur pour être modifiées. Côté serveur, la requête est reçue par la route /updateRoute.

```
router.route('/updateRoute').put(Auth, appController.updateRoute);
```

A chaque fois qu'un trajet est supprimé, ajouté ou modifié, la liste se met instantanément à jour en demandant au serveur la liste de toutes les routes de l'utilisateur. Côté serveur, la requête est traitée par la route /getroutes'.

```
router.route('/getroutes').get(Auth, appController.getRoutes);
```



Trajet courant

Nom du chemin :

Dates de départ :

- 29/01/2024 04:40

Horaires périodiques :

Jour de la semaine :

Heure :

Figure 4.18: Chargement des Trajets

5. Correspondance des trajets

Cette partie doit permettre à l'utilisateur de pouvoir visualiser les chemins des autres utilisateurs qui correspondent aux mieux aux leurs. Pour cela il y a deux critères :

- La proximité temporelle
- La proximité spatiale

Pour éviter de prendre en compte l'ensemble des chemins existants, un premier tri est effectué sur les dates de chaque trajet dans la base de données. Etant donné qu'il y a 2 manières différentes d'enregistrer une date (ponctuelle et périodique), il y a 4 requêtes à effectuer dans la base de donnée pour comparer :

- Les dates ponctuelles de l'utilisateur avec les autres dates ponctuelles
- Les dates ponctuelles de l'utilisateur avec les autres dates périodiques
- Les dates périodiques de l'utilisateur avec les autres dates ponctuelles
- Les dates périodiques de l'utilisateur avec les autres dates périodiques

Par exemple, voici la requête permettant de comparer le premier point :

```
let dateDate = user_dates.map(date => {
    // convert the date string to a date object
})
```

```

    let dateObj = new Date(date);

    // set the start date

    let startDate = new Date(dateObj);

    startDate.setMinutes(startDate.getMinutes() - dt);

    // set the end date

    let endDate = new Date(dateObj);

    endDate.setMinutes(endDate.getMinutes() + dt);

    // return the date range

    return {

        'planning.dates': {

            $elemMatch: {

                $gte: startDate, // greater than equal

                $lte: endDate    // lower than equal

            }

        }

    };

});

// Request the MongoDB database

const dateMatches = await RouteModel.find({

    // Finds routes that do not belong to the user

    username: { $ne: username },

    $or: dateDate

});

```

Dans le code, on laisse une marge de différence temporelle entre les trajets de “dt” minutes (réglé à 10 minutes pour l’instant).

Les 3 autres requêtes ne sont pas totalement opérationnelles et la partie algorithmique permettant de trouver la plus longue sous séquence n'a pas pu être implémentée par manque de temps. Ainsi, la route `/findMatches` qui s'occupe de cette partie retourne pour l'instant uniquement les trajets filtrés avec la première requête.

```
router.route('/findMatches').post(Auth, appController.findMatches)
```

Cependant, il est déjà possible de visionner et sélectionner les trajets retournés par le serveur via cette route.:

Trajets correspondant

- 06/02/2024 22:30
Horaires hebdomadaires :
 - Dimanche 18:27
 - Mardi 10:40
- **Trajet_1**
Dates :
 - 06/02/2024 04:35
 - 21/02/2024 04:25Horaires hebdomadaires :
 - Lundi 06:30
 - Vendredi 14:45

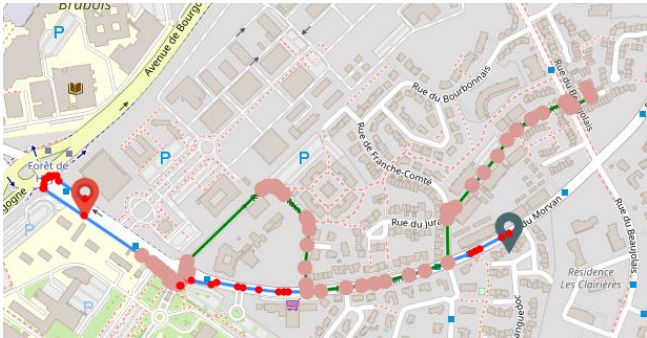


Figure 4.19: Affichage des Trajets correspondants

La partie “Trajets correspondant” affichant la liste de tous les trajets créés par un utilisateur est gérée par un composant nommé `MatchList.js`. Il est basé sur le même principe que le composant `ListRouteForm.js`.

Conclusion générale et perspectives :

Le site Web n'a pas été entièrement achevé en raison principalement de contraintes de temps et d'une méconnaissance du développement Web, bien que les fonctionnalités principales soient en place. Cependant, certaines fonctionnalités telles que le système de messagerie entre les utilisateurs et l'intégration de la correspondance des itinéraires sont encore manquantes. L'optimisation du code et l'amélioration du design du site sont également des aspects qui pourraient être améliorés. Malgré cela, le développement du projet s'est bien déroulé, ouvrant la voie à une reprise par une autre équipe qui pourra poursuivre le projet au cours du prochain semestre.

Bibliographie :

- Référence API React [en ligne]. Disponible sur : <https://fr.react.dev/reference/react>
- Node.js [en ligne]. Disponible sur : <https://nodejs.org/en>
- Express, référence de l'API [en ligne]. Disponible sur : <https://expressjs.com/fr/4x/api.html>
- MongoDB [en ligne]. Disponible sur : <https://www.mongodb.com/fr-fr>
- React hot toast [en ligne]. Disponible sur : <https://react-hot-toast.com/docs>
- Formik [en ligne]. Disponible sur : <https://formik.org/>
- Axios [en ligne]. Disponible sur : <https://axios-http.com/fr/docs/intro>
- Npm. bcrypt.js [en ligne]. Disponible sur : <https://www.npmjs.com/package/bcrypt>
- Wikipédia. Bcrypt [en ligne]. Disponible sur : <https://fr.wikipedia.org/wiki/Bcrypt>
- Npm. Mailgen [en ligne]. Disponible sur : <https://www.npmjs.com/package/mailgen>
- Daily Tuition, 2022. Complete MERN App [en ligne]. Disponible sur : <https://youtu.be/BfrJxGQEPSc>
- Nodemailer [en ligne]. Disponible sur : <https://nodemailer.com/>
- Ethereum [en ligne]. Disponible sur : <https://ethereal.email/>
- Npm. Zustand [en ligne]. Disponible sur : <https://www.npmjs.com/package/zustand>
- Npm. jsonwebtoken [en ligne]. Disponible sur : <https://www.npmjs.com/package/jsonwebtoken>
- Npm. otp-generator [en ligne]. Disponible sur : <https://www.npmjs.com/package/otp-generator>
- React leaflet [en ligne]. Disponible sur : <https://react-leaflet.js.org/>
- Mongoose [en ligne]. Disponible sur : <https://mongoosejs.com/docs/>
- Npm. React TimePicker [en ligne]. Disponible sur : <https://www.npmjs.com/package/rc-time-picker>
- Npm. React Date Picker [en ligne]. Disponible sur : <https://www.npmjs.com/package/react-datepicker>
- OpenRouteService [en ligne]. Disponible sur : <https://openrouteservice.org/>
- Bootstrap [en ligne]. Disponible sur : <https://getbootstrap.com/>
- Daniel, 2013. Longest common subsequence [en ligne]. Disponible sur : <https://youtu.be/P-mMvhfJhu8>
- Wikipédia. Technopôle de Nancy-Brabois [en ligne]. Disponible sur : https://fr.wikipedia.org/wiki/Technop%C3%B4le_de_Nancy-Brabois