



# Rapport de Projet

## Gallia

### Équipe Thémis - Q2

Réalisé par

Jihan **HAJEM**

Léo **MICHEL**

Ugo **MICHEL**

Théodore **PICOT**

Projet encadré par

**Lebreton Romain, Gasquet Malo**

Pour l'obtention du diplôme du BUT Informatique

# Remerciement

Tout d'abord, nous tenons à remercier tout particulièrement les personnes suivantes pour leur dévouement d'avoir permis la réalisation de ce projet :

- M. Malo GASQUET
- M. Romain LEBRETON
- Mme Anita MESSAOUI
- M. Simon ROBILLARD
- M. Matthieu ROSENFELD

# Résumé

Ce rapport concerne le déroulement du développement de **Gallia**, une application web de calcul d'itinéraire de plus court chemin, qui permet aux utilisateurs de trouver le chemin le plus court entre deux villes en France. Le site offre plusieurs fonctionnalités aux utilisateurs, telles qu'une liste de ses itinéraires préférés.

Le projet Gallia a été réalisé en utilisant différentes technologies. Tout d'abord, le développement a été effectué avec le langage de programmation PHP pour le côté serveur. Nous avons également utilisé le framework Symfony(\*) ainsi que l'outil Composer(\*). Ensuite, le côté client a été développé en utilisant le langage de programmation JavaScript. Pour stocker les données, le système de gestion de base de données Postgres a été utilisé.

This report is dedicated to the development of **Gallia**, a website whose purpose is to calculate the shortest path between two cities within France. This website offers to its users several functionalities, such as a list of their favorite itineraries.

The Gallia project was developed using different technologies. The server-side was developed using PHP, whilst the client-side was developed with JavaScript. We also used the open-source relational database management system PostgreSQL.

# Sommaire

<b>Introduction</b>	<b>6</b>
<b>1. Analyse</b>	<b>8</b>
1.1. Les outils utilisés pour l'identification des axes d'amélioration	8
1.2. Une première analyse des performances	9
1.3. Analyse de la base de données	9
1.3.1. Une requête peu performante	9
1.3.2. Une structure de la base de données non optimale	10
1.4. Analyse de l'algorithme initial : Dijkstra	10
1.5. Analyse de l'ergonomie	11
1.6. Analyse de la structure du projet	12
<b>2. Résultats</b>	<b>13</b>
2.1. Optimisation de la base de données	13
2.1.1. Indexage et clé primaires	14
2.1.2. Optimisation de la requête	15
2.2. Optimisation de l'algorithme côté PHP	19
2.2.1. Optimisation de l'algorithme de Dijkstra	19
2.2.2. Un nouvel algorithme : A*	21
2.3. Optimisation de la structure du projet	22
<b>3. Méthodologie et Organisation du Projet</b>	<b>32</b>
3.1 Méthode de développement et outils	32
3.2 Planification des tâches	33
<b>4. Impacts environnementaux</b>	<b>35</b>
4.1 - Analyse du cycle de vie	35
4.1.1 - Définition du champ de l'analyse	35
4.1.2 - Inventaire du cycle de vie	35
4.2 - Réflexion sur les usages	37
4.3 - Leviers d'action	38
<b>Conclusion</b>	<b>39</b>
<b>Bibliographie</b>	<b>40</b>

# Glossaire

**Xdebug** : Outil qui permet d'identifier les parties du code qui prennent le plus de temps à s'exécuter et qui ralentissent donc le fonctionnement de notre application.

**Profiling** : une méthode de débogage qui consiste à collecter des informations sur l'exécution d'une application

**Cachegrind** : type de fichier utilisé pour stocker les données obtenues suite au profiling

**PHPStorm** : environnement de développement intégré pour PHP, HTML, CSS et JavaScript, édité par JetBrains.

**SGBD** : système de gestion de base de données, permet de stocker, manipuler et gérer des données

**table** : dans une base de données, une table représente un ensemble de données organisées sous forme de tableau

**clés** : dans une table, la clé de la table est un fragment de données qui permet d'identifier l'objet de la table

**index** : dans une base de donnée, c'est une structure de données utilisée qui permet, lors de requêtes, de retrouver plus rapidement les données

**tronçon\_route** : dans notre base de donnée, un tronçon route représente un morceau de route

**nœud routier** : représente l'endroit où deux tronçons de route qui s'intersectent (intersection/carrefour).

**noeud\_commune** : représente une commune de France

**Composer** : logiciel de gestionnaire de dépendances, permet d'installer des librairies PHP

**Symfony** : ensemble de composants PHP, fournit des fonctionnalités qui permettent de faciliter le développement

**PHPUnit** : librairie PHP qui permet de faire des test unitaire

**API** : (application programming interface), permet d'utiliser des services Web externe

**PostGIS** : extension de Postgres, permet de manipuler des informations géographiques

**API REST** : standard d'API qui implique des contraintes à un service Web

**PhpFastCache** : PhpFastCache est une bibliothèque de mise en cache open-source pour PHP

**Jointures latérales** : Les jointures latérales (ou lateral joins en anglais) sont une extension des jointures classiques en SQL qui permettent d'inclure des sous-requêtes dans la clause FROM d'une requête principale. Elles permettent de récupérer des données liées à une table dans une sous-requête et de les intégrer directement dans les résultats de la requête principale.

**Union** : L'opérateur UNION en SQL permet de combiner les résultats de deux requêtes SELECT en une seule table résultante. Cette table résultante contient toutes les lignes des deux requêtes SELECT. Les doublons éventuels sont supprimés.

**Hash** : Hash est un algorithme mathématique qui transforme une entrée de données en une valeur de hachage unique.

**B-Tree** : Le B-Tree est une structure de données utilisée pour stocker et organiser des données dans un arbre équilibré, où chaque nœud peut avoir plusieurs clés et enfants. Cette structure est souvent utilisée pour stocker des données sur disque ou sur d'autres périphériques de stockage, car elle minimise le nombre d'accès aux données nécessaires pour trouver une clé donnée.

**Eager Loading** : Eager loading est une technique de programmation qui permet de charger en avance toutes les données associées à un objet ou à une requête, afin d'optimiser les performances de l'application en évitant des requêtes supplémentaires à la base de données.

**Heap** : L'heap est une structure de données en informatique utilisée pour stocker et organiser des données en mémoire. C'est une forme de tableau ordonné où chaque élément est organisé selon une priorité.

**PriorityQueue** : PriorityQueue est une structure de données qui permet de stocker un ensemble d'éléments avec une priorité associée à chacun d'entre eux, et de les récupérer en ordre de priorité croissante ou décroissante.

**Caching** : Le caching est une technique de stockage temporaire de données souvent utilisée en informatique pour améliorer les performances et réduire le temps de traitement en évitant de refaire des calculs ou des accès coûteux à une source de données.

# Introduction

Dans le cadre du semestre quatre de notre seconde année de BUT Informatique, nous avons eu à réaliser le développement d'une application complexe. Ce projet nous permet avant tout de mettre en pratique nos connaissances et nos compétences en développement Web, en qualité logicielle, mais aussi en ergonomie et sécurité. Ainsi, en partant d'un code existant, notre objectif est d'améliorer les performances d'une application Web qui calcule l'itinéraire le plus court entre deux communes.

Les applications de calculs d'itinéraires sont omniprésentes dans notre quotidien, et nous aurions du mal à nous en passer. Grâce aux différentes avancées technologiques et scientifiques, nous avons le moyen d'obtenir la distance entre deux points dans le monde, et cela instantanément.

Ce projet de développement a de multiples enjeux qui sont au centre de nos préoccupations.

En premier lieu, sur le plan technique, l'enjeu majeur est d'optimiser les performances de l'application pour fournir une réponse rapide et fiable à l'utilisateur. La rapidité d'exécution est un critère essentiel pour les utilisateurs, qui ont souvent besoin d'informations sur l'itinéraire à suivre en temps réel. Ainsi, l'optimisation des performances est au cœur de notre projet.

Ensuite, sur le plan humain, l'ergonomie de l'application est également un enjeu important. Nous devons concevoir une interface utilisateur intuitive qui permettra à chacun de trouver facilement l'information dont il a besoin, tout en offrant une expérience utilisateur agréable.

Finalement, sur le plan écologique, nous sommes conscients de l'impact environnemental des déplacements et de la nécessité de favoriser les modes de transport doux, tels que la marche ou le vélo, pour réduire les émissions de gaz à effet de serre. Dans ce contexte, l'application que nous développons peut jouer un rôle important en intégrant des informations sur l'empreinte carbone d'un utilisateur.

L'objectif principal de ce projet est d'optimiser les performances de l'application et d'intégrer une interface utilisateur comprenant différentes fonctionnalités. Les fonctionnalités principales de l'application comprennent le calcul du plus court chemin entre deux communes de France, la visualisation de l'itinéraire sur une carte, la possibilité de créer un compte utilisateur pour sauvegarder les itinéraires préférés et les consulter ultérieurement. L'application doit être sécurisée et offrir une expérience utilisateur fluide et agréable.

Dans un premier temps, notre rapport porte sur l'analyse approfondie de l'application et de ses différents axes d'amélioration. Nous avons examiné les performances de l'application pour en optimiser la rapidité et la fiabilité en effectuant des tests de vitesse. Nous avons également analysé la base de données, l'ergonomie ainsi que la sécurité de l'application.

Dans un second temps, la partie rapport technique recense le détail des différents changements et optimisations que nous avons réalisées pour chacun des points

précédents. Nous avons également présenté les tests que nous avons effectués pour mesurer les gains obtenus.

Dans la suite de ce rapport, nous présentons en détail la méthodologie que nous avons adoptée ainsi que l'organisation de notre projet. Nous mettons en avant notre choix d'adopter les méthodes agiles pour la gestion de projet, ce qui nous a permis d'optimiser notre collaboration et notre efficacité. Nous présenterons également les différents outils que nous avons utilisés, tels que Trello, Github et Discord.

Finalement, nous avons pris en compte l'impact environnemental de l'application en proposant une analyse du cycle de vie de notre application afin d'évaluer son empreinte carbone, ainsi qu'une réflexion sur ses différents usages.

# 1. Analyse

Dans cette partie, nous allons aborder l'analyse de la performance de notre application et les mesures que nous avons mises en place pour identifier les axes d'amélioration. Pour ce faire, nous avons utilisé plusieurs outils, notamment un logiciel de profiling et un outil pour analyser notre base de données. Grâce à ces outils, nous avons pu identifier les parties de l'application qui nécessitent une amélioration en termes de performance et nous avons pu déterminer les causes principales des manques de performances. Suite à cette partie, nous allons détailler les mesures que nous avons prises pour optimiser la performance de notre application et les résultats que nous avons obtenus.

## 1.1. Les outils utilisés pour l'identification des axes d'amélioration

L'identification des axes d'amélioration est une étape cruciale dans l'optimisation des performances d'un algorithme. Dans le cadre de notre projet, notre encadrant nous a recommandé l'utilisation de l'outil de débogage [Xdebug](#), pour identifier les parties du code qui prennent le plus de temps à s'exécuter et qui ralentissent donc le fonctionnement de notre application.

Après avoir installé l'extension Xdebug, nous avons été amenés à configurer cet outil pour qu'il fasse du profiling(\*). Le [profiling de Xdebug](#) permet de mesurer le temps d'exécution des différentes parties du code pour identifier les sections critiques qui consomment le plus de ressources. Il nous permet également de collecter des données sur la mémoire utilisée par le code. Ces données sont utiles pour détecter les fuites de mémoire et les problèmes de performance.

Les données obtenues grâce au profiling sont stockées dans un fichier Cachegrind(\*) qui peut être analysé ultérieurement. Nous avons ensuite utilisé un outil fourni par [PhpStorm](#)(\*), un IDE, pour visualiser ces fichiers et identifier les éléments problématiques dans notre code.

L'identification des points à améliorer est la première étape qui nous a permis d'optimiser notre algorithme de calcul du plus court chemin et d'améliorer les performances de notre application. Cette étape est essentielle pour tous les projets qui visent à améliorer les performances d'un algorithme ou d'une application.

## 1.2. Une première analyse des performances

Après avoir bien configuré Xdebug en mode profiling, nous avons effectué un test sur l'algorithme visant à trouver la distance la plus courte entre les communes de Billère et Cirès. Cette distance est de 30 mètres. Le résultat de l'exécution donné par Xdebug est disponible ci-dessous.

Callable	Time	Own Time	Memory (B)	Own Memory (B)	Calls
App\PlusCourtChemin\Controller\ControleurNœudCommune::plusCourtChemin	67,391 275.7%	0 0.0%	104,552	0	1 0.2%
php::PDOStatement->execute	67,125 274.6%	67,125 99.6%	8,800	8,800	8 1.8%
App\PlusCourtChemin\Modele\Repository\NœudRoutierRepository->construireDepuisTableau	66,411 271.7%	0 0.0%	5,064	288	3 0.7%
App\PlusCourtChemin\Modele\DataObject\NœudRoutier->__construct	66,411 271.7%	0 0.0%	4,776	0	3 0.7%
App\PlusCourtChemin\Modele\Repository\NœudRoutierRepository->getVoisins	66,411 271.7%	48 0.1%	15,264	0	3 0.7%
App\PlusCourtChemin\Modele\Repository\AbstractRepository->récupérerPar	53,887 220.5%	61 0.1%	32,672	0	4 0.9%
C:\Program Files\XAMPP\htdocs\plus-court-chemin-code-de-base\web\contrôleurFrontal.php	24,443 100.0%	0 0.0%	133,840	13,256	1 0.2%
App\PlusCourtChemin\Lib\PlusCourtChemin->calculer	13,496 55.2%	0 0.0%	1,880	0	1 0.2%
App\PlusCourtChemin\Modele\Repository\AbstractRepository->récupérerParCléPrimaire	13,496 55.2%	15 0.0%	5,728	0	1 0.2%
App\PlusCourtChemin\Modele\Repository\ConnexionBaseDeDonnées::getPdo	131 0.5%	0 0.0%	8,384	0	8 1.8%
App\PlusCourtChemin\Modele\Repository\ConnexionBaseDeDonnées::getInstance	131 0.5%	0 0.0%	8,384	480	8 1.8%
App\PlusCourtChemin\Modele\Repository\ConnexionBaseDeDonnées->__construct	131 0.5%	0 0.0%	7,992	0	1 0.2%
php::PDO->__construct	129 0.5%	129 0.2%	208	208	1 0.2%
App\PlusCourtChemin\Lib\Psr4AutoloaderClass->loadClass	7 0.0%	3 0.0%	116,576	0	16 3.7%
App\PlusCourtChemin\Lib\Psr4AutoloaderClass->loadMappedFile	5 0.0%	0 0.0%	113,144	1,400	40 9.2%
App\PlusCourtChemin\Lib\Psr4AutoloaderClass->requireFile	5 0.0%	2 0.0%	111,704	109,600	16 3.7%
App\PlusCourtChemin\Controller\ControleurGénérique::afficherVue	2 0.0%	0 0.0%	30,616	2,296	1 0.2%
require:C:\Program Files\XAMPP\htdocs\plus-court-chemin-code-de-base\src\Modele\Repository\NœudRoutierRepository	1 0.0%	0 0.0%	29,120	1,296	1 0.2%
App\PlusCourtChemin\Lib\Message\Dispatcher\TousMessages	1 0.0%	0 0.0%	11,272	376	1 0.2%

Capture d'écran du Cachegrind de Xdebug avec PHPStorm

Avec la capture d'écran ci-dessus, nous avons pour une ligne la méthode concernée, puis sur la deuxième colonne, le temps que cette méthode occupe. [Reformuler]

En analysant les résultats obtenus grâce à Xdebug, nous avons constaté que la majeure partie du temps d'exécution était occupée par les requêtes à la base de données. Pour cette raison, notre priorité a été d'optimiser les requêtes utilisées par le script pour récupérer les données, afin de réduire le temps nécessaire pour effectuer ces opérations.

Après avoir fait des recherches, nous constatons également que l'algorithme initial n'est pas le plus optimal dans notre contexte, nous reviendrons sur ce sujet après l'analyse de la base de données.

## 1.3. Analyse de la base de données

Pour aller plus loin dans l'analyse de la base de données, nous avons utilisé DBeaver, une puissante interface nous permettant de visualiser et gérer des bases de données. De plus, le SGBD(\*) PostgreSQL que nous utilisons, nous permet d'analyser le coût et le temps qu'une requête consomme avec la fonctionnalité [explain](#). Ainsi, nous avons pu comparer différentes requêtes et déterminer laquelle est la plus optimisée.

### 1.3.1. Une requête peu performante

La requête à la base de données permettant de récupérer les voisins d'un nœud routier(\*) représente une partie essentielle et centrale dans notre algorithme du plus court

chemin. Les voisins d'un nœud routier représentent tous les nœuds routiers (intersection) qui sont connectés par une route.

Node Type	Entity	Cost	Rows	Time	Condition
▼ Unique		8416175513892.71 - 8418812393644.37	2	23970.027	
▼ Sort		8416175513892.71 - 8416834733830.62	2	23970.008	
▼ Gather		1000.42 - 8325596374504.16	2	23969.985	
▼ Parallel Append		0.42 - 8299227575987.56	1	20480.086	
▼ Nested Loop		0.42 - 4147636128180.04	0	14699.324	
▼ Nested Lo		0.42 - 18288500.52	0	12162.960	
Parallel troncon_route		0.00 - 58025.16	651379	1049.981	
Index S noeud_routier		0.42 - 8.44	1	0.008	(nr.gid = 1)
Seq Scan noeud_routier		0.00 - 22122.33	910833	1072.761	
▼ Nested Loop		0.42 - 4147636128180.04	0	16020.788	
▼ Nested Lo		0.42 - 18288500.52	0	13027.373	
Parallel troncon_route		0.00 - 58025.16	651379	949.783	
Index S noeud_routier		0.42 - 8.44	1	0.008	(nr_1.gid = 1)
Seq Scan noeud_routier		0.00 - 22122.33	910833	1168.791	

Capture d'écran du résultat de Xdebug avec PHPStorm

Il est important de prendre en compte l'impact de cette requête sur les performances de l'application. En effet, un appel répété à cette requête peut ralentir considérablement le fonctionnement de l'algorithme si elle n'est pas optimisée.

Avec les résultats d'analyse obtenus grâce à DBeaver, nous avons constaté que la requête permettant de récupérer les voisins d'un nœud routier prenait plus de 20 secondes, suggérant ainsi qu'elle n'était pas du tout optimisée. Si cette requête était appelée plusieurs milliers de fois pour calculer la distance entre deux points éloignés, le temps de calcul serait considérable.

### 1.3.2. Une structure de la base de données non optimale

Il est important de noter que la base de données utilisée pour cet algorithme n'a pas été conçue de manière optimale, avec des tables(\*) ne contenant ni clés(\*) ni index(\*). Ces éléments sont pourtant cruciaux pour améliorer le temps d'exécution des requêtes, et sont aujourd'hui considérés comme essentiels dans la conception d'une base de données performante. En outre, une optimisation supplémentaire pourrait être apportée en ajoutant des index.

## 1.4. Analyse de l'algorithme initial : Dijkstra

Pour trouver le plus court chemin entre deux communes, l'algorithme de Dijkstra nous a été fourni. Il s'applique sur un graphe connexe avec des arêtes possédant un poids positif.

En résumé, l'algorithme de Dijkstra consiste à construire progressivement un sous-graphe en classant les nœuds par ordre croissant de leur distance minimale par rapport au sommet de départ. La distance correspond à la somme des poids des arêtes empruntées pour atteindre chaque sommet.

La première étape consiste à définir la distance entre le sommet de départ et les autres sommets en mettant de côté le sommet de départ. Si un arc relie le sommet au sommet de départ et que le poids le plus faible (s'il y en a plusieurs) est de n, la distance est de n.

Ensuite, la seconde étape consiste à trouver le sommet qui a la distance la plus courte au sommet de départ et à le mettre de côté. Pour tous les sommets restants, on compare la distance précédemment trouvée à celle obtenue en passant par le sommet sélectionné, en ne gardant que la plus petite valeur. On continue ainsi jusqu'à épuisement des sommets ou jusqu'à atteindre le sommet d'arrivée.

Ce dernier s'avère, dans la plupart des cas, une option efficace pour résoudre ce problème. Malheureusement, dans notre contexte, Dijkstra n'est pas la méthode optimale. Dijkstra effectue une recherche aveugle et peut donc consommer énormément de ressources sur des graphes de grande taille.

Cependant, il existe des algorithmes similaires offrant des améliorations significatives qui nous offraient une résolution plus efficace de ce problème. Nous pourrions alors envisager de tester différentes alternatives et comparer les résultats pour déterminer s'il est possible d'améliorer l'efficacité de notre algorithme.

## 1.5. Analyse de l'ergonomie

En partant du code fourni initialement, l'interface utilisateur fournit est la suivante :

Nom	id_rte	COMMUNE	nomCommune
Lourties-Monbrun	1	COMMUNE_000000009760754	nomCommune Lourties-Monbrun
Noeud routier	2	COMMUNE_000000009756744	nomCommune Boudy-de-Beauregard
Noeud routier	3	COMMUNE_000000009760252	nomCommune Autrans-Méaudre-en-Vercors
Noeud routier	4	COMMUNE_000000009754403	nomCommune Autrans-Méaudre-en-Vercors
Noeud routier	5	COMMUNE_000000009727903	nomCommune Willeman
Noeud routier	6	COMMUNE_000000009732421	nomCommune Arvieu-de-Maurouard
Noeud routier	7	COMMUNE_000000009759331	nomCommune Cravencières
Noeud routier	8	COMMUNE_000000009741242	nomCommune Rigny-le-Ferron
Noeud routier	9	COMMUNE_000000009758465	nomCommune Pissos
Noeud routier	10	COMMUNE_000000009751252	nomCommune Cordelle
Noeud routier	11	COMMUNE_000000009749135	nomCommune Sivignon
Noeud routier	12	COMMUNE_000000009740248	nomCommune Saint-Pierre-des-Landes
Noeud routier	13	COMMUNE_000000009755238	nomCommune Roissand
Noeud routier	14	COMMUNE_000000009752658	nomCommune Trou-Palis
Noeud routier	15	COMMUNE_000000009738405	nomCommune Gazeille
Noeud routier	16	COMMUNE_000000009733716	nomCommune Fugères
Noeud routier	17	COMMUNE_000000009737191	nomCommune Chavoy
Noeud routier	18	COMMUNE_000000009746928	nomCommune Septfontaines
Noeud routier	19	COMMUNE_000000009733847	nomCommune Poët-Saint-Pier
Noeud routier	20	COMMUNE_000000009760450	nomCommune Seysses
Noeud routier	21	COMMUNE_000000009754981	nomCommune Porcherès
Noeud routier	22	COMMUNE_000000009737377	nomCommune

Capture d'écran de la page "Communes" de l'interface fournit

Nous pouvons relever plusieurs points en ce qui concerne l'ergonomie du site web. L'interface présentée ici est très basique, avec une présentation sous forme de blocs de texte. Elle présente une liste de communes de France obtenue à partir de la base de

données, accompagnée de liens pour afficher plus d'informations sur chaque commune. Le texte est peu aéré et semble difficile à lire. Cela peut être décourageant pour les utilisateurs.

La page présente aussi un lien, qui mène vers la page "Calculer un plus court chemin", fonctionnalités pourtant principales du site. Le lien est relativement petit, il peut être difficile pour les utilisateurs de le repérer, surtout s'ils recherchent spécifiquement cette fonctionnalité.

De plus, aucune carte n'est présente sur le site, accentuant le manque de contexte. Sans une carte ou une autre forme de contexte visuel, les utilisateurs peuvent avoir du mal à comprendre comment la fonctionnalité de calcul de plus court chemin est censée fonctionner et comment elle s'intègre dans le site web. Cela peut rendre l'expérience utilisateur moins fluide et moins intuitive.

Nous avons décidé d'analyser l'interface de différentes applications qui offrent la possibilité de visualiser des trajets entre deux villes, notamment Google Maps. Du point de vue ergonomique, nous constatons plusieurs points. Premièrement, la barre de recherche est située en haut à gauche de la page et permet aux utilisateurs de saisir une adresse. Ensuite, la carte est la partie centrale de l'interface de Google Maps, située sur la page principale, et montre la région sélectionnée par l'utilisateur. Les utilisateurs peuvent zoomer, dézoomer, faire pivoter et incliner la carte pour afficher différentes vues et détails. De plus, le menu de navigation est situé sur le côté gauche de l'interface de Google Maps et contient différentes options telles que la possibilité de consulter les adresses enregistrées par l'utilisateur.

Dans l'ensemble, l'interface de Google Maps est conçue pour offrir une expérience de navigation facile et intuitive aux utilisateurs, tout en leur fournissant une grande quantité d'informations sur les lieux et les itinéraires qu'ils recherchent.

## 1.6. Analyse de la structure du projet

Initialement, la structure du projet était assez mauvaise, car elle laissait peu de possibilité de test, que ce soit au niveau des algorithmes de recherche ou encore sur les différentes fonctionnalités utilisateurs.

Il est donc possible de réorganiser la structure et de séparer certaines classes pour pouvoir améliorer la testabilité.

Pour pouvoir bien réaliser les tests, nous serons probablement amenés à installer une librairie de Composer, comme PHPUnit(\*) .

Par ailleurs, le projet n'utilisait pas d'API(\*) alors qu'elles permettent une meilleure sécurité et elles sont faciles à implémenter. De plus, elles contribuent à créer des applications robustes et intuitives. Nous pouvons décider d'implanter certaines API pour ajouter plusieurs fonctionnalités utilisateur. Ainsi, nous pourrions par exemple ajouter la météo de la ville destination. Cela permettrait à l'utilisateur de possiblement repousser son voyage s'il y a un mauvais temps.

## 2. Résultats

Dans cette partie, nous allons aborder les différentes optimisations que nous avons pu effectuer durant notre projet. Nous avons optimisé dans un premier temps la base de donnée dans l'optique de récupérer les données plus rapidement. Par la suite, nous avons choisi d'optimiser l'algorithme PHP pour récupérer plus rapidement le trajet entre deux communes. Enfin, nous avons optimisé la structure du projet pour qu'elle respecte principalement les principes SOLID.

### 2.1. Optimisation de la base de données

Voici la requête SQL qui permet de récupérer les voisins d'un nœud routier :

```
select nr2.gid as noeud_routier_gid, tr.gid as troncon_gid, tr.longueur
from noeud_routier nr,
     troncon_route tr,
     noeud_routier nr2
where (st_distancesphere(nr.geom, st_startpoint(tr.geom)) < 1
      and st_distancesphere(nr2.geom, st_endpoint(tr.geom)) < 1
      and nr.gid = :gidTag)
union
select nr2.gid as noeud_routier_gid, tr.gid as troncon_gid, tr.longueur
from noeud_routier nr,
     troncon_route tr,
     noeud_routier nr2
where (st_distancesphere(nr2.geom, st_startpoint(tr.geom)) < 1
      and st_distancesphere(nr.geom, st_endpoint(tr.geom)) < 1
      and nr.gid = :gidTag);
```

Et voici le résultat d'exécution de cette requête avec DBeaver sur le serveur de l'IUT :

Node Type	Entity	Cost	Rows	Time	Condition
▼ Unique		9839331752130.76 - 9842412760125.44	2	6526.919	
▼ Sort		9839331752130.76 - 9840102004129.43	2	6526.918	
▼ Gather		1000.00 - 9727885070379.78	2	6526.883	
▼ Parallel Append		0.00 - 9697074989432.98	1	4389.633	
▼ Nested Loop		0.00 - 4846226738720.48	0	3272.991	
▼ Nested Lo		0.00 - 38366075.52	0	2305.798	
Parallel noeud_routier		0.00 - 17757.92	0	437.197	(nr.gid = 1)
Seq Scn troncon_route		0.00 - 76881.81	1532758	958.472	
Seq Scan noeud_routier		0.00 - 22122.33	910833	436.983	
▼ Nested Loop		0.00 - 4846226738720.48	0	3311.448	
▼ Nested Lo		0.00 - 38366075.52	0	2402.791	
Parallel noeud_routier		0.00 - 17757.92	0	728.278	(nr_1.gid = 1)
Seq Scn troncon_route		0.00 - 76881.81	1532758	617.242	
Seq Scan noeud_routier		0.00 - 22122.33	910833	305.226	

résultat requête SQL de base

Six secondes pour récupérer les deux voisins du nœud routier de gid 1. Il serait préférable que cette requête ne prenne que quelques millisecondes pour s'exécuter pour remarquer un changement des performances de l'algorithme.

### 2.1.1. Indexage et clé primaires

Premièrement, nous avons constaté que les attributs **gid** des tables **noeud\_routier**, **noeud\_commune(\*)** et **troncon\_route(\*)** ne sont pas définis en tant que clé primaire. En les définissant en tant que clé primaire, un index est automatiquement ajouté. Cela optimise le temps d'exécution de la requête getVoisins, car **gid** est utilisé dans la clause "**where**" de la requête pour récupérer les voisins.

```
alter table troncon_route
    add constraint troncon_route_pk
        primary key (gid);
alter table noeud_routier
    add constraint noeud_routier_pk
        primary key (gid);
alter table noeud_commune
    add constraint noeud_commune_pk
        primary key (gid);
```

Il est également possible d'indexer l'attribut **geom** qui figure lui aussi dans la clause **WHERE** de la requête qui nous permet de récupérer les voisins d'un nœud routier. Pour cela, nous pouvons utiliser le type d'index "**gist**" qui est un index spatial.

```
CREATE INDEX idx_gid_geom_noeud_routier
    ON noeud_routier
        USING gist (geom);
CREATE INDEX idx_gid_geom_troncon_route
    ON troncon_route
        USING gist (geom);
```

## 2.1.2. Optimisation de la requête

Après avoir effectué des recherches, nous avons constaté que la méthode `st_distancesphere` de l'extension PostGIS(\*) nous retourne la distance minimale, en mètre, entre deux points possédant une longitude et latitude. Puis avec cette longueur, nous regardons si la distance est plus petite que 1 mètre.

```
st_distancesphere(nr.geom, st_startpoint(tr.geom)) < 1
```

Cette condition nous retourne vraie si un certain nœud routier est à moins d'un mètre du début ou de la fin d'un tronçon. Cependant, il y a une méthode plus optimisée permettant de nous retourner le même résultat `st_dwithin`. Cette méthode est plus optimisée, car elle utilise des index sur **geom**.

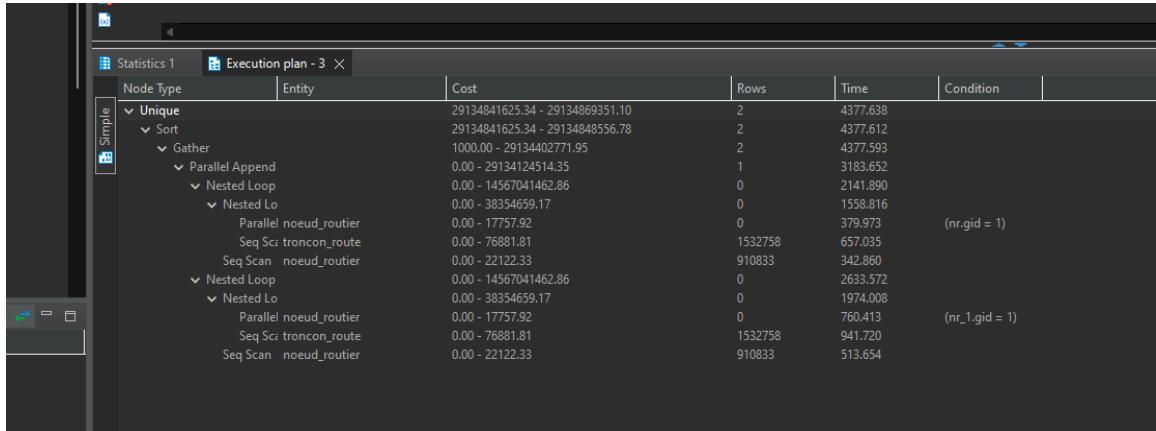
Voici la nouvelle requête et le résultat sur DBeaver :

```
-- La distance est en kilomètre donc 0.001 = 1 mètre
select nr2.gid as noeud_routier_gid, tr.gid as troncon_gid, tr.longueur
from noeud_routier nr,
     troncon_route tr,
     noeud_routier nr2
where (st_dwithin(nr.geom, st_startpoint(tr.geom), 0.001)
and st_dwithin(nr2.geom, st_endpoint(tr.geom), 0.001)
and nr.gid = :gidTag)
union
select nr2.gid as noeud_routier_gid, tr.gid as troncon_gid, tr.longueur
from noeud_routier nr,
     troncon_route tr,
     noeud_routier nr2
where (st_dwithin(nr2.geom, st_startpoint(tr.geom), 0.001)
and st_dwithin(nr.geom, st_endpoint(tr.geom), 0.001)
and nr.gid = :gidTag);
```

Node Type	Entity	Cost	Rows	Time	Condition
Simple					
✓ Unique		29134841625.34 - 29134869351.10	2	4377.638	
✗ Sort		29134841625.34 - 29134848556.78	2	4377.612	
✗ Gather		1000.00 - 29134402771.95	2	4377.593	
✗ Parallel Append		0.00 - 29134124514.35	1	3183.652	
✗ Nested Loop		0.00 - 14567041462.86	0	2141.890	
✗ Nested Lo		0.00 - 38354659.17	0	1558.816	
Parallel noeud_routier		0.00 - 17757.92	0	379.973	(nr.gid = 1)
Seq Scan troncon_route		0.00 - 76881.81	1532758	657.035	
Seq Scan noeud_routier		0.00 - 22122.33	910833	342.860	
✗ Nested Loop		0.00 - 14567041462.86	0	2633.572	
✗ Nested Lo		0.00 - 38354659.17	0	1974.008	
Parallel noeud_routier		0.00 - 17757.92	0	760.413	(nr_1.gid = 1)
Seq Scan troncon_route		0.00 - 76881.81	1532758	941.720	
Seq Scan noeud_routier		0.00 - 22122.33	910833	513.654	

résultat requête SQL `st_dwithin`

Le résultat de cette requête avec les index ci-dessus :



Node Type	Entity	Cost	Rows	Time	Condition
Unique		29134841625.34 - 29134869351.10	2	4377.638	
Sort		29134841625.34 - 29134848556.78	2	4377.612	
Gather		1000.00 - 29134402771.95	2	4377.593	
Parallel Append		0.00 - 29134124514.35	1	3183.652	
Nested Loop		0.00 - 14567041462.86	0	2141.890	
Nested Lo		0.00 - 38354659.17	0	1558.816	
Parallel noeud_routier		0.00 - 17757.92	0	379.973	(nr_gid = 1)
Seq Scan troncon_route		0.00 - 76881.81	1532758	657.035	
Seq Scan noeud_routier		0.00 - 22122.33	910833	342.860	
Nested Loop		0.00 - 14567041462.86	0	2633.572	
Nested Lo		0.00 - 38354659.17	0	1974.008	
Parallel noeud_routier		0.00 - 17757.92	0	760.413	(nr_1_gid = 1)
Seq Scan troncon_route		0.00 - 76881.81	1532758	941.720	
Seq Scan noeud_routier		0.00 - 22122.33	910833	513.654	

résultat requête SQL st\_dwithin avec index

Nous avons gagné environ 2 secondes sur le temps d'exécution. Ce qui est déjà important. Mais il est possible de pousser cette optimisation davantage.

Nous avons également créé des vues pour les tronçons et le nœud possédant strictement les données nécessaires. Cela nous permettra de légèrement améliorer le coût de la requête. Voici le code SQL :

```
-- Vue noeudRoutier GID GEOM

CREATE MATERIALIZED VIEW view_gid_geom_routier AS
SELECT gid, geom
FROM noeud_routier;

-- Vue troncon Route GID GEOM

CREATE MATERIALIZED VIEW view_gid_geom_troncon AS
SELECT gid, geom, longueur
FROM troncon_route;
```

De même, nous avons remarqué qu'il était possible d'améliorer la structure de la requête en utilisant des jointures latérales(\*) au lieu de faire un **union**. Voici la requête :

```
select nr2.gid as noeud_routier_gid, tr.gid as troncon_gid, tr.longueur
      from view_gid_geom_troncon tr
      join lateral ( select nr.gid, nr.geom
                     from view_gid_geom_routier nr
                    where nr.gid = :gidTag) as nr
      on st_dwithin(nr.geom, st_startpoint(tr.geom), 0.001)
         or st_dwithin(nr.geom, st_endpoint(tr.geom), 0.001)
      join lateral (select nr2.gid, nr2.geom
                     from view_gid_geom_routier nr2
                    where nr2.gid != :gidTag) as nr2
      on st_dwithin(nr2.geom, st_endpoint(tr.geom), 0.001)
         or st_dwithin(nr2.geom, st_startpoint(tr.geom), 0.001);
```

Cette nouvelle requête a considérablement réduit le coût et le temps d'exécution :

Node Type	Entity	Cost	Rows	Time	Condition
▼ Gather		1002.98 - 42298097.75	2	840.292	
▼ Nested Loop		2.98 - 41823155.75	1	797.119	
▼ Nested Loop		0.42 - 31888733.80	1	797.093	
Parallel Seq Scan	view_gid_geom_troncon	0.00 - 37226.93	434253	51.787	
Index Scan	view_gid_geom_routier	0.42 - 8.44	1	0.001	(nr.gid = 1)
▼ Bitmap Heap Scan	view_gid_geom_routier	2.55 - 9154.33	1	0.032	((nr2.gid <> 1) ...)
▼ BitmapOr		2.55 - 2.55	0	0.030	
Bitmap Index	idx_geom_view_noeud_rou...	0.00 - 1.23	1	0.015	(nr2.geom && ...)
Bitmap Index	idx_geom_view_noeud_rou...	0.00 - 1.23	1	0.009	(nr2.geom && ...)

résultat requête SQL join lateral

Grâce à cette requête, nous avons pu créer une table contenant tous les voisins de chaque noeud routier. Voici la requête :

```
CREATE TABLE voisins AS
select nr.gid as noeud_routier_base, nr2.gid as noeud_routier_gid, tr.gid as
troncon_gid, tr.longueur
from view_gid_geom_troncon tr
join lateral ( select nr.gid, nr.geom
                     from view_gid_geom_routier nr) as nr
      on st_dwithin(nr.geom, st_startpoint(tr.geom), 0.001)
         or st_dwithin(nr.geom, st_endpoint(tr.geom), 0.001)
      join lateral (select nr2.gid, nr2.geom
                     from view_gid_geom_routier nr2
                    where nr2.gid != nr.gid) as nr2
      on st_dwithin(nr2.geom, st_endpoint(tr.geom), 0.001)
         or st_dwithin(nr2.geom, st_startpoint(tr.geom), 0.001);
```

Cette table est constituée d'environ 4,7 millions de lignes, le temps d'exécution pour la création est d'une minute en localhost :

Node Type	Entity	Cost	Rows	Time	Condition
▼ Gather		1005.11 - 9482290538889.37	4772910	60185.076	
▼ Nested Loop		5.11 - 9050584966520.27	1590970	61256.785	
▼ Nested Loop		2.55 - 4967843686.52	1024726	18471.116	
Parallel Seq Scan	view_gid.geom.troncon	0.00 - 37226.93	434253	99.613	
▼ Bitmap Heap Scan	view_gid.geom_routier	2.55 - 9153.87	2	0.040	(st_dwithin(nr.g...
▼ BitmapOr		2.55 - 2.55	0	0.037	
Bitmap In idx_geom_view_noeud_rou...		0.00 - 1.23	1	0.018	(nr.geom && s...
Bitmap Inv idx_geom_view_noeud_rou...		0.00 - 1.23	1	0.018	(nr.geom && s...
▼ Bitmap Heap Scan	view_gid.geom_routier	2.55 - 9153.87	3	0.039	(st_dwithin(nr....
▼ BitmapOr		2.55 - 2.55	0	0.037	
Bitmap Index idx_geom_view_noeud_rou...		0.00 - 1.23	1	0.018	(nr2.geom && ...
Bitmap Index idx_geom_view_noeud_rou...		0.00 - 1.23	1	0.018	(nr2.geom && ...

résultat création de la table des voisins

Voici la nouvelle requête nous permettant de récupérer les voisins d'un nœud routier et les détails d'exécution de celle-ci :

```
select noeud_routier_gid,
       troncon_gid,
       longueur
  from voisins
 where noeud_routier_base = :gidTag;
```

Node Type	Entity	Cost	Rows	Time	Condition
▼ Gather		1000.00 - 74788.58	2	181.176	
Parallel Seq Scan	voisins	0.00 - 73787.38	1	166.650	(voisins.noeud_...

résultat nouvelle requête voisins

Et voici le nouveau temps et coût d'exécution pour trouver les voisins du nœud de gid 1 avec cette nouvelle table et un index utilisant la méthode hash(\*) sur l'attribut noeud\_routier\_base de la nouvelle table :

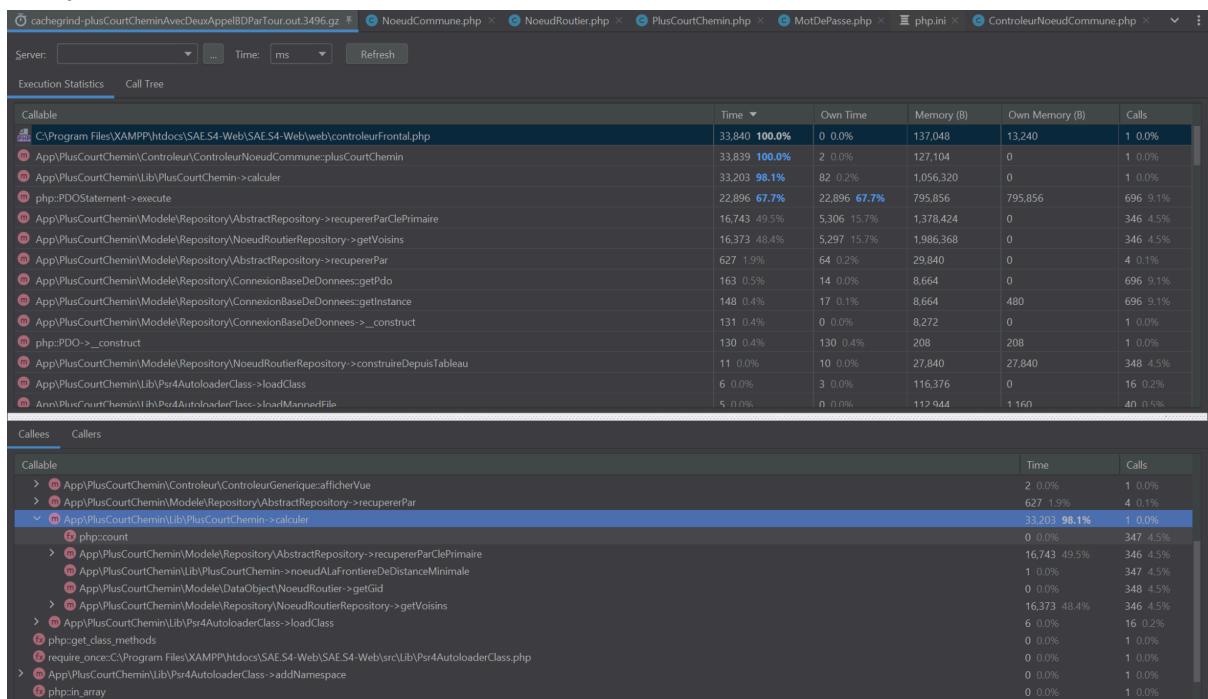
Node Type	Entity	Cost	Rows	Time	Condition
▼ Bitmap Heap Scan	voisins	4.08 - 43.66	2	0.025	
Bitmap Index Scan	idx_gid_voisins	0.00 - 4.08	2	0.011	(voisins.noeud_...

Nous avons choisi d'utiliser l'index de type hash car celui-ci a été prouvé 40% plus rapide sur des entiers uniques et une égalité que les index de types b-tree. Depuis la version 9.6 de PostgreSQL.

## 2.2. Optimisation de l'algorithme côté PHP

### 2.2.1. Optimisation de l'algorithme de Dijkstra

Pour calculer le plus court chemin entre les deux communes Dieppe → Varengeville-sur-Mer (environ 8 km) nous avons le résultat suivant grâce au profiling de XDebug avec l'algorithme de Dijkstra et la nouvelle requête SQL optimisée :



Ici, nous pouvons constater que le temps d'exécution de l'algorithme est d'environ 30 secondes. La majorité du temps est occupée par la méthode **PDOStatement->execute**. Cela représente les appels à la base de données. Nous avons rapidement remarqué que nous faisons un appel à la base de données non nécessaire.

```

while (count($this->noeudsALaFrontiere) !== 0) {
    $noeudRoutierGidCourant = $this->noeudALaFrontiereDeDistanceMinimale();

    // Fini
    if ($noeudRoutierGidCourant === $this->noeudRoutierArriveeGid) {
        return $this->distances[$noeudRoutierGidCourant];
    }

    // Enlève le noeud routier courant de la frontière
    unset($this->noeudsALaFrontiere[$noeudRoutierGidCourant]);

    /** @var NoeudRoutier $noeudRoutierCourant */
    $noeudRoutierCourant = $noeudRoutierRepository->recupererParClePrimaire($noeudRoutierGidCourant);
    $voisins = $noeudRoutierCourant->getVoisins();
}

foreach ($voisins as $voisin) {
    $noeudVoisinGid = $voisin["noeud_routier_gid"];
    $distanceTroncon = $voisin["longueur"];
    $distanceProposee = $this->distances[$noeudRoutierGidCourant] + $distanceTroncon;

    if (!isset($this->distances[$noeudVoisinGid]) || $distanceProposee < $this->distances[$noeudVoisinGid]) {
        $this->distances[$noeudVoisinGid] = $distanceProposee;
        $this->noeudsALaFrontiere[$noeudVoisinGid] = true;
    }
}
}

```

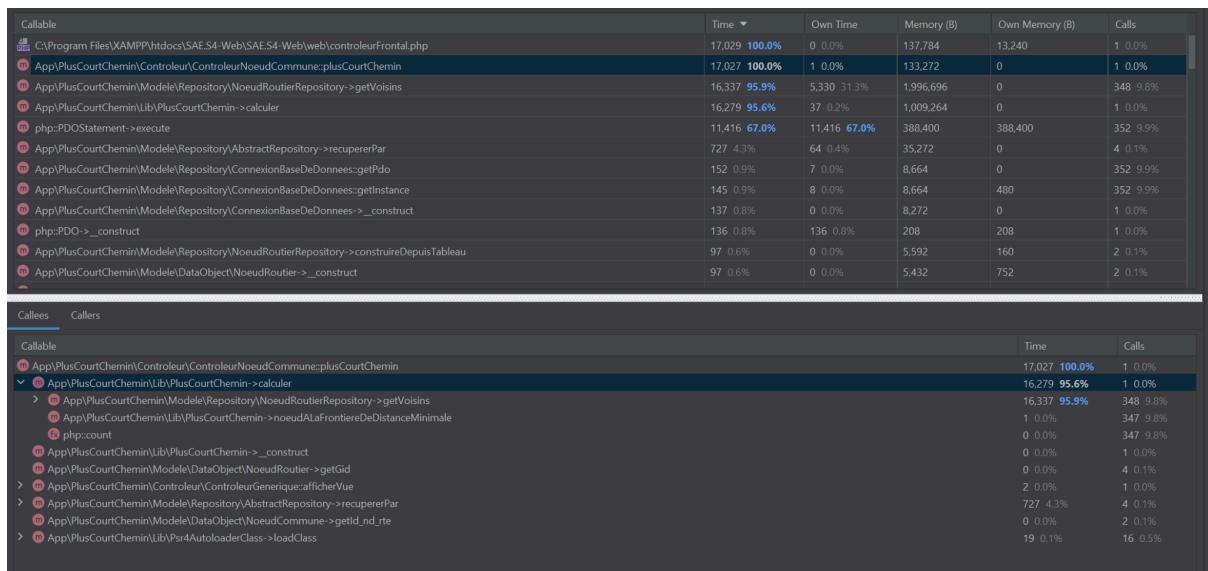
En effet, à chaque itération de l'algorithme, nous faisons ces deux appels à la base de données alors qu'il serait possible d'en faire uniquement un :

```

/** @var NoeudRoutier $noeudRoutierCourant */
$noeudRoutierCourant = $noeudRoutierRepository->recupererParClePrimaire($noeudRoutierGidCourant);
$voisins = (new NoeudRoutierRepository)->getVoisins($noeudRoutierGidCourant);

```

Ceci divise pratiquement le temps d'exécution de l'algorithme par deux :



Nous avons également trouvé qu'il est possible d'améliorer la méthode permettant de trouver le noeud à la frontière de distance minimale. Pour l'instant nous utilisant le code suivant :

```

private function noeudALaFrontiereDeDistanceMinimale()
{
    $noeudRoutierDistanceMinimaleGid = -1;
    $distanceMinimale = PHP_INT_MAX;
    foreach ($this->noeudsALaFrontiere as $noeudRoutierGid => $valeur) {
        if ($this->distances[$noeudRoutierGid] < $distanceMinimale) {
            $noeudRoutierDistanceMinimaleGid = $noeudRoutierGid;
            $distanceMinimale = $this->distances[$noeudRoutierGid];
        }
    }
    return $noeudRoutierDistanceMinimaleGid;
}

```

Il se trouve qu'il y a de meilleures solutions pour stocker les nœuds de distance minimale dans la frontière. Les structures de données telles que la Heap(\*) ou la PriorityQueue(\*) pourraient des très bonnes solutions pour optimiser cette partie de l'algorithme.

Après quelques recherches, nous avons trouvé des librairies natives PHP qui nous donnent accès à la PriorityQueue. Nous avons donc choisi d'utiliser cette structure de donnée pour obtenir le nœud de distance minimale dans la frontière.

### 2.2.2. Un nouvel algorithme : A\*

Dans la partie analyse, nous avons identifié que l'algorithme de Dijkstra n'était pas le plus optimal pour notre contexte. Nous avons donc exploré d'autres options et avons trouvé que la variante A\* de Dijkstra était plus efficace dans la majorité des cas. Lajout d'une fonction heuristique, qui calcule la distance à vol d'oiseau entre le nœud routier courant et le nœud routier d'arrivée, nous a permis d'estimer la distance restante et de choisir le prochain nœud à explorer en fonction de cette distance théorique.

Cependant, nous avons remarqué que les appels à la base de données prenaient toujours un temps considérable malgré ces optimisations. C'est pourquoi nous avons décidé d'implémenter la méthode d'eager loading(\*) pour récupérer plus rapidement les voisins d'un certain nœud routier. Cette méthode permet de charger en une seule requête tous les éléments nécessaires à la recherche et d'éviter ainsi des appels répétés à la base de données. Cette amélioration a considérablement accéléré notre algorithme et a contribué à une meilleure expérience utilisateur.

Voici le résultat du temps d'exécution pour calculer le plus court chemin entre Menton→Porospoder (les deux communes les plus éloignées) après avoir implémenté ces fonctionnalités :

	Time	Own Time	Memory (B)	Own Memory (B)	Calls
Callable					
C:\Program Files\XAMPP\htdocs\SAE-S4-Web\SAE-S4-Web\web\contrroleurFrontal.php	19,800 <b>100.0%</b>	0 0.0%	5,136,992	800	1 0.0%
App\PlusCourtChemin\Contrroleur\RouteurURL::traiterRequete	19,791 <b>100.0%</b>	5 0.0%	4,518,240	0	1 0.0%
# php:call_user_func_array(C:\Program Files\XAMPP\htdocs\SAE-S4-Web\SAE-S4-Web\src\Contrroleur\RouteurURL.php:291)	19,606 <b>99.0%</b>	0 0.0%	1,909,344	0	1 0.0%
# App\PlusCourtChemin\Contrroleur\ContrroleurNoeudCommune->plusCourtChemin	19,605 <b>99.0%</b>	779 3.9%	971,489,656	969,563,768	1 0.0%
# App\PlusCourtChemin\Lib\PlusCourtCheminAStar->_construct	17,664 <b>89.2%</b>	0 0.0%	0	0	1 0.0%
# App\PlusCourtChemin\Lib\PlusCourtCheminAStar->calculer	17,637 <b>89.1%</b>	2,066 10.5%	871,760	679,640	1 0.0%
# App\PlusCourtChemin\Service\NoueudRouterService->getTousLesVoisinsV2	13,392 <b>67.6%</b>	0 0.0%	0	0	1 0.0%
# App\PlusCourtChemin\Modelle\Repository\NoueudRouterRepository->getTousLesVoisinsV2	13,392 <b>67.6%</b>	137 0.7%	1,304	0	1 0.0%
# php:PDO->query	10,799 <b>54.5%</b>	10,799 <b>54.7%</b>	1,304	1,304	1 0.0%
# php:PDOStatement->fetchAll	2,454 12.4%	2,454 12.4%	0	0	1 0.0%
# php:SpriorityQueue->extract	1,175 5.9%	1,174 5.9%	0	0	5,480 2.7%
# App\PlusCourtChemin\Contrroleur\ContrroleurGenerique->afficherTwig	816 4.1%	0 0.0%	1,885,032	160	1 0.0%
# Twig\Environment->render	813 4.1%	0 0.0%	1,663,472	0	1 0.0%
# Twig\Environment->loadTemplate	800 4.0%	1 0.0%	1,604,104	84,432	2 0.0%
# Twig\Environment->compileSource	797 4.0%	0 0.0%	1,787,488	0	2 0.0%
# App\PlusCourtChemin\Lib\PlusCourtCheminAStar->calculDistanceHeuristique	714 3.6%	714 3.6%	0	0	6,256 3.1%
# Twig\Environment->parse	592 3.0%	0 0.0%	907,360	952	2 0.0%
# Twig\Parser->parse	591 3.0%	1 0.0%	862,520	3,032	2 0.0%
# Twig\TemplateWrapper->render	557 2.8%	0 0.0%	385,392	0	1 0.0%
# Twig\Template->render	557 2.8%	0 0.0%	386,016	384	1 0.0%
# Twig\Template->display	557 2.8%	0 0.0%	369,704	320	2 0.0%
# Twig\Template->displayWithErrorHandling	557 2.8%	0 0.0%	369,008	0	2 0.0%
# __TwigTemplate_6ce0313f5374b91ea50f4a992bd624->doDisplay	557 2.8%	0 0.0%	369,384	0	1 0.0%
# Twig\Template->loadTemplate	546 2.8%	0 0.0%	338,784	0	1 0.0%
Callers					
Callable					
<All scripts>					
> # C:\Program Files\XAMPP\htdocs\SAE-S4-Web\SAE-S4-Web\web\contrroleurFrontal.php	19,800 <b>100.0%</b>	201,994 <b>100.0%</b>	19,800 <b>100.0%</b>	1 0.0%	

Nous remarquons que ceci prend environ 20 secondes. Nous avons donc réussi à considérablement optimiser le temps d'exécution de cet algorithme.

Il est également possible de faire du caching(\*) du tableau stockant tous les voisins, ceci va nous permettre de récupérer ces données plus rapidement. Voici une fonction qui nous permet de cacher les données d'une requête et récupérer ces données si elles sont déjà dans le cache. Tout cela avec PhpFastCache(\*) :

```
public function getTousLesVoisins(): array
{
    $defaultDriver = 'Files';
    $Psr16Adapter = new Psr16Adapter($defaultDriver);
    if (!$Psr16Adapter->has('queried')) {
        $Psr16Adapter->set('queried', 'query', 300); // 5 minutes
        $requeteSQL = <<<SQL
            select noeud_routier_base, noeud_routier_gid,st_x(coordonnees_base) as longitude_base, st_y(coordonnees_base) as latitude_base, troncon_gid, st_x(coordonnees_voisins) as longitude_voisin, st_y(coordonnees_voisins) as latitude_voisin
        SQL;
        $pdoStatement = $this->connexionBaseDeDonnees->getPdo()->query($requeteSQL);
        $fetchResult = $pdoStatement->fetchAll(PDO::FETCH_GROUP|PDO::FETCH_ASSOC);
        $Psr16Adapter->set('tousLesNoeudsRoutiers', $fetchResult, 300);
        return $fetchResult;
    } else {
        // Getter action
        return $Psr16Adapter->get('tousLesNoeudsRoutiers');
    }
}
```

## 2.3. Optimisation de la structure du projet

Dans l'optique de développer une API REST(\*), nous avons eu besoin que les URL des pages de notre site n'utilisent plus le *query string*. Pour cela, nous avons utilisé une bibliothèque PHP existante, et donc un gestionnaire de bibliothèques appelé Composer.

Composer est utilisé dans le cadre du développement d'applications PHP pour installer des composants tiers. Composer gère un fichier appelé `composer.json` qui référence toutes les dépendances de votre application.

Composer nous a fourni un autoloader, c'est-à-dire un chargeur automatique de classe, qui satisfait la spécification PSR-4. En effet, cet autoloader était très pratique pour utiliser les paquets que nous avons installés via Composer.

Quand nous avons installé Composer, ce dernier place les librairies téléchargées dans un dossier `vendor`.

Notre objectif était de faire un nouveau routeur par Url. Nous avons d'abord installé le composant `HttpFoundation` défini une couche orientée objet pour la spécification HTTP. Puis cela nous a permis d'utiliser la classe `Request` de `HttpFoundation` pour représenter une requête HTTP. Cette classe était très intéressante pour récupérer les chemins qui nous intéressaient. Après nous avons eu besoin d'installer le composant `Routing` de Symfony qui nous a permis de faire l'association entre une URL et une action du contrôleur. Nous avons notamment utilisé la classe `Route` du composant `Routing`. Nous avons changé l'URL de nos pages par exemple "`web/controleurFrontal.php/connexion`" vers une URL plus classique "`web/connexion`".

```
web/controleurFrontal.php/  
web/controleurFrontal.php/connexion  
web/controleurFrontal.php/inscription
```

exemple d'URL possible

Ensuite, nous avons utilisé le composant `HttpKernel` de Symfony qui fournit un processus structuré pour convertir une `Request` en `Response`. Les classes qui nous ont intéressés dans ce composant sont les classes `ControllerResolver` et `ArgumentResolver`. La classe `ControllerResolver` a pour rôle de déterminer le contrôleur et la méthode à appeler en fonction de la requête. Puis la classe `ArgumentResolver` construit une liste des arguments des actions du contrôleur. L'avantage de ce mécanisme est qu'il permet de récupérer beaucoup de types d'arguments dans le contrôleur.

Dans un second temps, nous avons ajouté un code de réponse HTTP pour indiquer plus clairement l'état du serveur. Ceci est indispensable dans l'optique du développement d'une API REST, qui était notre objectif principal. Puis, nous avons utilisé du JSON, ce qui est aussi un des fondamentaux des API REST. Effectivement, dans le cas du développement de cette API, l'échange de données doit s'effectuer en format JSON ou XML. Enfin, nous avons appris à utiliser un template engine (`Twig`), c'est-à-dire à un langage spécifique pour la création de vues. Voici un exemple, ci-dessous, d'une vue `Twig` pour afficher les trajets favoris.

```

1  {% extends "base.html.twig" %}
2
3  {% block page_title %}Favoris{% endblock %}
4
5  {% block page_content %}
6
7      <main>
8
9          <div class="d-flex justify-content-center my-4">
10             <h1> Vos trajets favoris</h1>
11         </div>
12
13         <div class="row my-5">
14             <div class="col-lg-6 col-md-12 col-sm-12 offset-lg-3 my-5">
15                 {% for trajet in trajets %}
16                     <div class="card mx-5 h-40 my-3">
17                         <div class="card-body">
18                             <h5 class="card-title" style="...><p class="card-text" style="...>{{ trajet.comm_depart }}<br/>- {{ trajet.comm_arrivee }} | {{ trajet.date|date("H:i - d m \Y") }}</h5>
19                             <a href="{{ route("afficherTrajet", { 'idTrajet' : trajet.idTrajet}) }}" class="btn btn-primary">Voir le trajet</a>
20                             <a href="{{ route("supprimerFavoris", { 'idTrajet' : trajet.idTrajet}) }}" class="delete-trajet btn btn-primary">
21                                 Supprimer des favoris
22                             </button>
23                         </div>
24                     </div>
25                 {% endfor %}
26             </div>
27         </div>
28
29     </main>
30
31  {% endblock %}

```

### Exemple d'une vue Twig : favoris.html.twig

Tout d'abord, pour les réponses HTTP, nous avons utilisé la classe Response du composant HttpFoundation de Symfony. L'objet Response contient toutes les informations qui doivent être renvoyées au client *HTTP* : des en-têtes, un code de réponse et un corps de réponse. Par conséquent, nous avons modifié nos actions pour qu'elles retournent toutes une instance de la classe Response. Ceci nous a permis par la suite d'utiliser toutes les possibilités des réponses *HTTP*.

Le composant HttpFoundation de Symfony fournit aussi la classe RedirectResponse qui hérite de Response. Cette classe nous a permis de bénéficier automatiquement d'une redirection plus professionnelle. De plus, RedirectResponse a associé automatiquement le code de réponse 302 Found qui indique une redirection temporaire.

Ensuite, les méthodes UrlMatcher::match(), ControllerResolver::getController() et ArgumentResolver::getArguments() sont présentes dans le RouteurURL, elles peuvent lever des exceptions. Donc, nous avons traité ces exceptions en envoyant une réponse *HTTP* adéquate, en faisant particulièrement attention au code de réponse.

En ce qui concerne le langage de gabarit, Twig, nous l'avons découvert et appris lors de td. Puis nous l'avons appliquée dans ce projet. Le principe des langages de gabarit (template engines) est de fournir un langage adapté aux vues. Les langages de gabarit ont plusieurs points positifs comme sa réutilisation, l'héritage de gabarit permet de reprendre une mise en page existante (comme vueGenerale.php) en spécifiant des parties (le titre, le corps de la page, ...). Sa sécurité est activée par défaut, en particulier l'échappement des caractères spéciaux du HTML. Ceci vise à protéger les non-développeurs des menaces web courantes telles que XSS et CSRF dont ils ne sont pas nécessairement conscients. Twig est

aussi apprécié notamment grâce à sa rapidité. Twig compile les gabarits en code PHP simple, ce qui permet leur évaluation à un surcoût minimum.

Un langage de gabarit doit trouver un bon équilibre entre offrir suffisamment de fonctionnalités pour faciliter l'implémentation de la logique de présentation, et limiter les fonctionnalités avancées pour éviter l'apparition de logique métier dans les gabarits. On fournit aux vues la liste des variables qu'elles peuvent utiliser. Idéalement, la vue n'accède aux variables qu'en lecture, ce qui évite de modifier l'état du système.

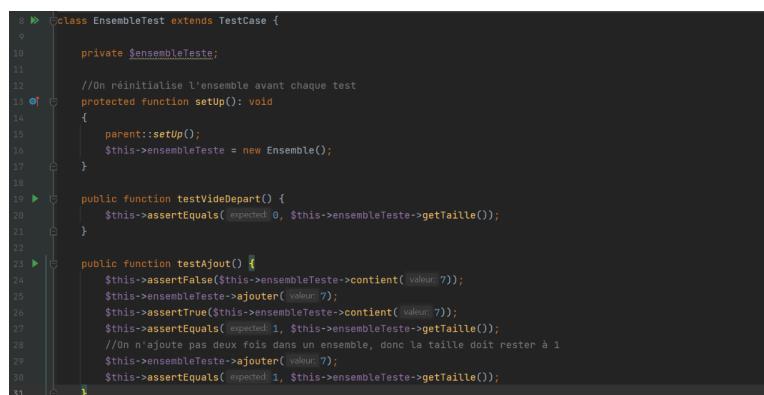
Par la suite, nous avons mis en place des tests unitaires sur une application web PHP pour tester les différentes fonctionnalités Utilisateurs. Ainsi, nous devons dans un premier temps structurer notre application par rapport aux principes SOLID que nous avions étudiés cette année, notamment dans le cours de qualité de développement.

Après avoir structuré notre projet, nous avons pu commencer à mettre en place les tests avec PHPUnit qui est une librairie PHP permettant de réaliser des tests unitaires sur une application PHP. Son fonctionnement est similaire à JUnit que nous avons déjà utilisé notamment en cours de Tests. PHPUnit intègre par défaut les outils nécessaires à l'utilisation de mocks ainsi que l'analyse de la couverture de code. Ainsi, comme toute librairie PHP, PHPUnit s'installe à l'aide de composer, cela est donc simple d'installation.

Après avoir installé PHPUnit, nous avons mis en place une classe de test qui regroupe différents tests unitaires. On peut considérer qu'un test unitaire se traduit par une fonction dans une classe dédiée qui exécute différents tests sur des objets de l'application. Il s'agit de vérifier, par exemple, si le retour d'une fonction avec un paramétrage spécifique est bien conforme aux attentes et aux spécifications. On peut aussi tester si l'exécution d'un code déclenche des exceptions. Notre classe de test étend la classe TestCase.

À partir de là, nous avions accès à une grande variété de méthodes internes pour réaliser des assertions. Une assertion est simplement une vérification qui est faite (sur un résultat, sur un comportement...). Si cette vérification échoue (résultat différent de ce qui est attendu) le test échoue alors.

Nous avons, au début, réalisé des premiers tests simples afin de comprendre le fonctionnement de PHPUnit. Puis, nous avons utilisé cet outil de manière plus concrète en testant notre application web plus en profondeur. Néanmoins, nous avons constaté que notre application n'était pas testable en l'état.



```
1 >>> <class EnsembleTest extends TestCase {
2
3     private $ensembleTeste;
4
5     //On réinitialise l'ensemble avant chaque test
6     protected function setUp(): void
7     {
8         parent::setUp();
9         $this->ensembleTeste = new Ensemble();
10    }
11
12    public function testVideDepart(): void
13    {
14        $this->assertEqual(0, $this->ensembleTeste->getTaille());
15    }
16
17    public function testAjout(): void
18    {
19        $this->assertFalse($this->ensembleTeste->contient('valeur'));
20        $this->ensembleTeste->ajouter('valeur');
21        $this->assertTrue($this->ensembleTeste->contient('valeur'));
22        $this->assertEquals(1, $this->ensembleTeste->getTaille());
23        //On n'ajoute pas deux fois dans un ensemble, donc la taille doit rester à 1
24        $this->ensembleTeste->ajouter('valeur');
25        $this->assertEquals(1, $this->ensembleTeste->getTaille());
26    }
27}
```

Une classe test simple sans l'utilisation de mocks

En effet, pour tester, nous avions besoin de faire des assertions sur des résultats spécifiques obtenus lors de l'exécution d'une fonctionnalité. Ainsi, les fonctionnalités étaient réalisées par les contrôleurs. Or, les différentes fonctions des contrôleurs renvoient un objet Response qui n'est pas bien exploitable.

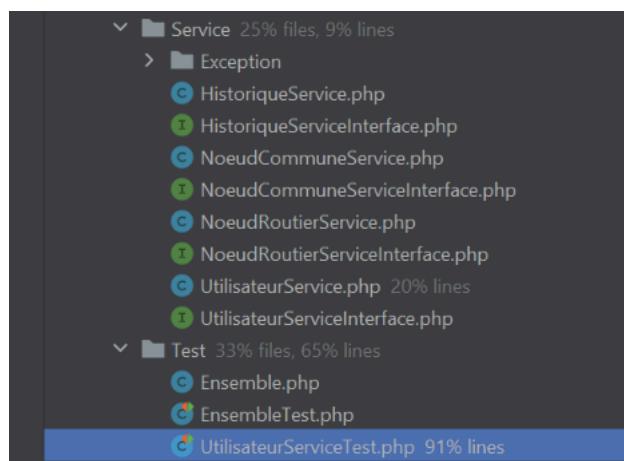
Ainsi, notre application web comme tout logiciel peut être organisée selon une architecture qui sépare de manière optimisée les classes et programmes selon leur rôle. L'architecture actuelle de notre application était une architecture MVC. Cette architecture permet de séparer les entités, les vues et les contrôleurs de l'application et de les faire communiquer.

Néanmoins, il n'est pas explicitement fait mention des services dans cette architecture. C'est pourquoi il est possible de venir placer la couche service entre les contrôleurs, les entités et la couche stockage. Ainsi, le contrôleur n'effectue pas de logique métier et on a une séparation plus forte.

Dans notre cas, notre application tend vers une architecture utilisant les services. On peut considérer qu'une telle séparation permettra alors de pouvoir tester la logique métier indépendamment au travers des tests unitaires sur les services plutôt que sur les contrôleurs. D'une part, il sera alors possible de passer des données à ces services autrement que par une requête HTTP, et d'autre part, on pourra également obtenir un résultat exploitable et pas une page web complète.

Nous avons donc créé une classe service pour un contrôleur et cela permet d'alléger les contrôleurs et de réduire la gestion des erreurs.

Après avoir réalisé une multitude de tests, nous avons utilisé la couverture de code. Cet outil permet de réaliser des statistiques sur les portions de code que nos tests permettent de tester. Après l'exécution des tests, on peut alors visualiser le pourcentage de code testé sur une classe et on peut même aller dans le détail en visualisant les lignes de code qui ont été franchies par les tests et celles qui n'ont jamais été franchies.



Un exemple de couverture de code

A ce stade, l'application a rencontré des problèmes de dépendance au niveau des tests que nous avons écrits pour tester le service utilisateur. Dans certains tests, nous avons dû préciser des utilisateurs réels. Ces tests dépendent donc de l'état actuel de l'application, des utilisateurs inscrits. Nous n'avons pas pu tester la création "normale" d'un utilisateur. Déjà, nous n'avons pas les moyens de récupérer directement l'utilisateur créé (la méthode de

création d'utilisateur ne retourne rien). Deuxièmement, le test entraîne la création réelle de l'utilisateur dans l'application. Cela n'est pas très adapté. En plus, la base de données doit obligatoirement être allumée pendant l'exécution de tests. Tout cela est dû au fait que notre classe UtilisateurService est fortement dépendante d'autres classes et notamment d'une classe repository. En fait, nous ne pouvions pas encore qualifier nos tests de tests unitaires.

Un test unitaire doit seulement porter sur une portion de code très précise, typiquement une méthode, et ne doit pas concrètement déclencher l'exécution d'autres services dans l'environnement de l'application (pas d'effet de bord). De plus, les tests ne doivent pas dépendre de l'état concret de l'application à l'instant du test, typiquement, les tests ne doivent pas dépendre de l'état de la base de données.

Pour régler ce problème nous avons utilisé deux solutions. Premièrement, le principe SOLID notamment l'inversion de contrôle afin de réaliser l'injection de dépendances. Secondement, l'utilisation de mocks a permis de simuler et configurer les dépendances d'une classe lors des tests unitaires afin de construire un scénario précis.

Lorsqu'une classe est amenée à utiliser des instances d'autres classes lors de l'exécution de ses différentes méthodes, nous pouvons dire qu'il existe une dépendance entre ces deux classes. Par conséquent, il était judicieux d'appliquer le concept d'inversion de contrôle en favorisant l'injection des dépendances de la classe plutôt que de laisser la classe instancier un objet de la classe cible ou bien utiliser un singleton. Exemple d'injection avec l'image ci-dessous.



```
13
14 class UtilisateurService implements UtilisateurServiceInterface {
15
16     private UtilisateurRepositoryInterface $coUtilisateur;
17
18     public function __construct(UtilisateurRepositoryInterface $coUtilisateur)
19     {
20         $this->coUtilisateur = $coUtilisateur;
21     }
22 }
```

Capture d'écran - Injection du repository dans UtilisateurService

Ce que nous avons retenu à propos d'une bonne architecture est que les différentes classes ne dépendent pas d'instances d'autres classes en particulier. Mais plutôt d'une interface qui pourra prendre des formes différentes grâce au polymorphisme. De plus, les instances concrètes sont injectées dans les classes qui doivent utiliser un service. Enfin, il est possible d'utiliser la même instance et de l'injecter dans différentes classes.

Pour finir la partie sur l'implémentation des tests, notre logique métier était indépendante de classes concrètes, nous avons pu réaliser de véritables tests unitaires sans avoir besoin ou influer sur le reste de l'application. Les mocks nous ont permis de créer de fausses classes possédant les mêmes méthodes qu'une vraie classe. Il était donc possible de configurer dynamiquement la classe mock par des lignes de code. Par exemple, nous pouvions préciser un résultat à renvoyer lors de l'appel d'une méthode précise ou bien même déclencher une exception. En plus, un autre aspect très utile des mocks était de pouvoir exécuter un callback lorsqu'une méthode est exécutée tout en récupérant les valeurs des paramètres de la méthode exécutée. Nous avons configuré tout cela grâce à la méthode

willReturnCallback lors de la configuration d'une méthode sur un mock. Enfin, pour l'API REST, nous avons modifié la méthode RouteurURL::traiterRequete afin qu'elle prenne une requête en paramètre et renvoie la réponse plutôt que de tout traiter d'un coup.

Une fois l'implémentation des tests terminés, nous avons enfin pu développer un API REST. Les aspects fondamentaux d'un service Web RESTful sont :

- adopter une convention de nommage pour les identifiants de ressources (URI) ;
- utiliser des verbes HTTP.
- utiliser les codes de réponse HTTP pour indiquer si une requête a pu être traitée avec succès.
- échanger des données au format JSON (ou XML).
- être sans état (Stateless), ou sans mémoire.
- Le fonctionnement du service doit pouvoir être découvert.

Dans l'optique de développer un API REST, nous avons échangé les données au format JSON.

Pour utiliser l'API dans la page Web avec AJAX, nous avons dû créer de nouvelles routes comme par exemple /api/supprimerFavoris/{idTrajet} qui est associée au verbe HTTP DELETE qui supprime un trajet en favori. Comme l'illustre l'image ci-dessous.

```
$route = new Route( path: "/api/supprimerFavoris/{idTrajet}", [
    "_controller" => "contrroleur_historique_api::supprimer",
]);
$route->setMethods(["DELETE"]);
$routes->add( name: "supprimerFavoris", $route);
```

Exemple d'une route qui utilise l'API

Pour tester l'API, nous avons appris à utiliser le logiciel Postman qui est un petit logiciel très pratique quand on développe des API. Ce logiciel nous a permis de paramétrier et d'envoyer des requêtes de manière interactive et de visualiser le résultat très simplement. Exemple avec l'image ci-dessous.

### Exemple d'utilisation de Postman

Ensuite, nous avons programmé la fonction `jsonSerialize()` pour créer une requête qui renvoie les détails d'un utilisateur au format JSON. Exemple de la fonction juste en dessous :

```
public function jsonSerialize(): array
{
    return [
        "login" => $this->getLogin(),
        "nom" => $this->getNom(),
        "prenom" => $this->getPrenom(),
        "email" => $this->getEmail()
    ];
}
```

Exemple de la fonction `jsonSerialize()` dans la classe Utilisateur



```
{"login":"test","nom":"test","prenom":"test","email":"martin@yopmail.com"}
```

Exemple du résultat de la route qui utilise la fonction jsonSerialize()

Ensute, nous avons dû implémenter une authentification avec des JWT pour respecter le principe Stateless. En effet, à ce stade, notre API ne respectait pas le principe Stateless car on utilisait des sessions pour garder en mémoire l'utilisateur connecté et ainsi l'autoriser à accéder à des routes sécurisées.

La solution que nous avons choisi de mettre en place consiste pour le serveur à ajouter une signature cryptographique dans le cookie. Ainsi, le client n'a plus la possibilité de modifier son cookie ; sinon il devrait falsifier la signature, ce qui est pratiquement impossible puisque seul le serveur est en capacité de signer. Ce mécanisme est fourni par les Json Web Token (JWT).

Pour implanter cette authentification nous avons utilisé une bibliothèque externe, puis nous avons créé une classe JsonWebToken.php.

```
1 <?php
2
3 namespace App\PlusCourtChemin\Lib;
4
5 use ...
6
7
8 class JsonWebToken
9 {
10     private static string $jsonSecret = "RYh14hV6x+g4wBSrdS7kTE";
11
12     public static function encoder(array $contenu) : string {
13         return JWT::encode($contenu, self::$jsonSecret, alg: 'HS256');
14     }
15
16     public static function decoder(string $jwt) : array {
17         try {
18             $decoded = JWT::decode($jwt, new Key(self::$jsonSecret, algorithm: 'HS256'));
19             return (array) $decoded;
20         } catch (\Exception $exception) {
21             return [];
22         }
23     }
24
25 }
```

Capture d'écran de la classe JsonWebToken.php

Nous avons ensuite rencontré un problème au niveau de la couche Service qui doit être indépendante de l'interface, et donc de toute la couche de transfert de donnée HTTP. Du coup, nous n'avions pas le droit d'appeler la classe ConnexionUtilisateur, qui est basée sur

les mécanismes Web cookie et session, dans les services. Par conséquent, nous avons modifié les appels à la classe ConnexionUtilisateur dans le Service UiltsateurService.

Ainsi, nous avons réussi proposer deux mécanismes d'authentification au sein de notre application web :

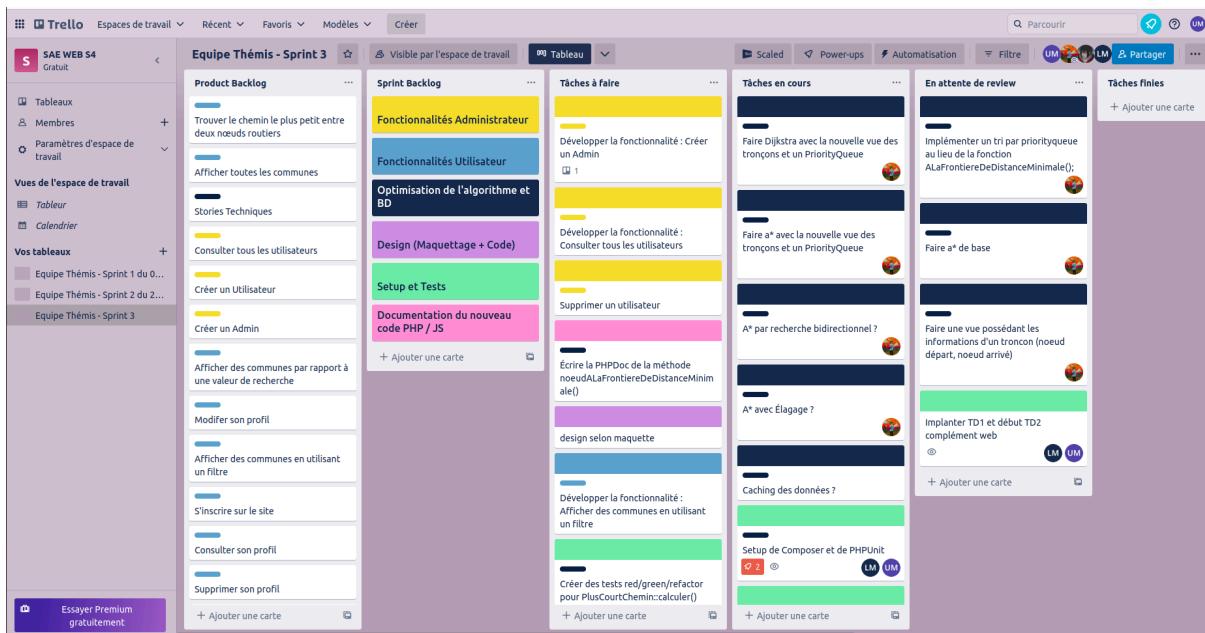
- un mécanisme basé sur les sessions, qui n'est utilisé que sur le site Web (ControleurUtilisateur)
- un mécanisme basé sur les JWT. Ce mécanisme est utilisé à la fois dans l'API REST (pour devenir Stateless), et dans le site classique pour que les fonctionnalités JavaScript puissent appeler l'API REST.

# 3. Méthodologie et Organisation du Projet

Dans cette partie, nous aborderons la méthodologie et l'organisation de notre projet. Pour ce faire, nous avons utilisé plusieurs outils pour organiser nos tâches à réaliser. Ensuite, nous détaillerons la planification des tâches que nous avons effectuée durant notre projet ainsi que les problèmes que nous avons rencontrés.

## 3.1 Méthode de développement et outils

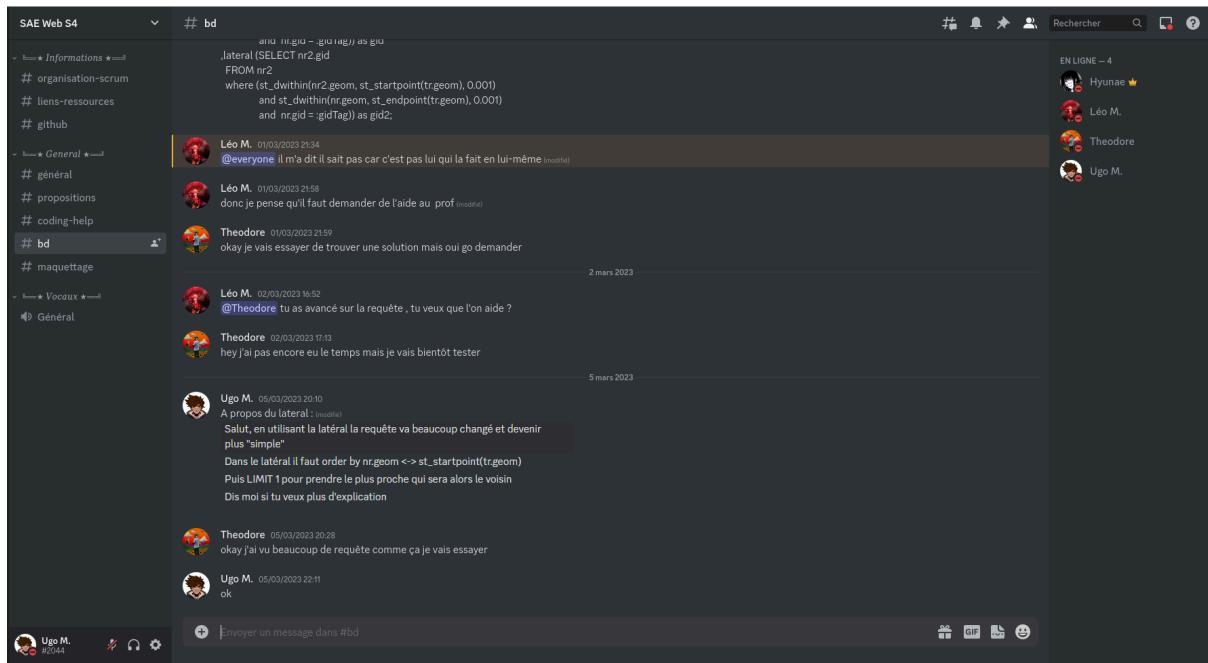
Pour mener à bien notre projet, nous avons choisi de suivre la méthode agile Scrum. ([notre product backlog et tableaux sprints](#)) Pour organiser nos tâches et suivre l'évolution de notre projet, nous avons opté pour Trello.



Capture d'écran du Trello - Sprint 3

Cette plateforme nous a permis de partager les tâches que chaque membre de l'équipe s'était attribuées, de visualiser leur état d'avancement et faciliter la communication au sein de notre équipe. Nous avons fait un tableau pour chaque sprint. Les sprints avaient une durée de 2 semaines.

De plus, nous avons également utilisé Google Docs pour synthétiser nos recherches et rédiger notre rapport de projet. Nous avons choisi d'utiliser Github pour le versionning du code. Dans le but de réaliser des daily meetings et de pouvoir communiquer rapidement, nous avons décidé de créer un serveur Discord.



Capture d'écran serveur Discord

Nous avons créé différents salons textuels afin d'organiser nos recherches et nos discussions. Par exemple, le salon #coding-help nous permettait de poser nos questions lorsqu'un des membres rencontrait une difficulté. Par ailleurs, nous avions une catégorie d'informations pour organiser les différents daily meeting ou archiver les différents liens importants à traiter. Enfin, nous avions une catégorie "vocal" pour travailler ensemble à distance.

## 3.2 Planification des tâches

Durant le semestre, nous avons réalisé un premier sprint de deux semaines, un sprint d'un mois, puis un sprint de trois semaines. Nous avons essayé de nous adapter à notre charge de travail qui était très importante, nous avons donc eu des difficultés à faire des sprints de longueurs égales. Au début de chaque sprint, nous nous réunissons pour faire un point sur les tâches que nous avions réussi à faire ou non au sprint précédent. Après avoir fait le point sur le précédent sprint, nous choisissons les nouvelles tâches ainsi que celle à poursuivre pour les deux prochaines semaines. Chaque membre du groupe se positionne sur le travail qu'il souhaite effectuer durant le sprint.

De plus, nous faisions régulièrement des réunions que nous planifiions sur un calendrier google. Durant ces réunions, nous pouvions chacun partager et expliquer les tâches que nous avions réalisées ou commencées aux autres membres du groupe. Si quelqu'un rencontrait une difficulté dans l'une de ses tâches, nous pouvions l'aider à trouver des solutions durant ces réunions. Cela nous a fait gagner beaucoup de temps. Par exemple, au cours du premier sprint, nous avons rencontré des problèmes dans l'installation de Xdebug. De plus, nous ne nous sommes pas assez bien répartis les tâches prévues dans le sprint. Par conséquent, nous avions pris un peu de retard au début du projet. C'est

notamment grâce à plus de daily meetings que nous sommes arrivés à régler nos problèmes plus rapidement.

## 4. Impacts environnementaux

### 4.1 - Analyse du cycle de vie

#### 4.1.1. - Définition du champ de l'analyse

Nous analysons l'impact environnemental lors de l'usage de notre application de calcul du plus court chemin. Pour cette analyse, nous prenons comme unité **l'impact par utilisateur** lors de l'usage de l'application. Notre unité de mesure est le kWh/par utilisateur.

Lorsqu'un utilisateur utilise notre application, il engendre différents types de consommation qui peuvent impacter l'environnement. Dans un premier temps, pour utiliser l'application, l'usager utilise un terminal (téléphone, ordinateur, tablette, etc.). De plus, il a également besoin d'être connecté à un réseau en ligne. Nous prenons aussi en compte la consommation énergétique en fonction de la puissance du CPU. Ensuite, le temps passé sur l'application entraîne aussi une consommation plus ou moins importante selon les recherches effectuées. Nous pensons aussi au [navigateur](#) [8] que l'utilisateur utilise, qui représente également un impact.

#### 4.1.2. - Inventaire du cycle de vie

Nous avons réalisé un inventaire du cycle de vie de notre application en 5 grands points.

Tout d'abord, la première étape consiste à définir les objectifs de l'analyse de cycle de vie (ACV) et à déterminer les limites de notre système. Cela implique de définir les fonctionnalités attendues du site web par rapport à l'utilisateur, les limites géographiques (la carte de la France métropolitaine), etc. Puis, nous avons regroupé les exigences et limitations en termes de données, en fonction du niveau de précision souhaité de l'algorithme de recherche et des ressources disponibles pour la collecte de données.

Par la suite, la deuxième étape consiste à quantifier tous les flux entrants et sortants de notre système, dans notre cas, le site web permet de trouver le plus court chemin. Pour cela, nous avons recueilli des données auprès de sources telles que des API de transport, des bases de données de cartographie, des fournisseurs de données météorologiques, etc. Par exemple, nous avons implanté une API qui permet de calculer les émissions de gaz à effet de serre et de polluants atmosphériques d'un trajet tracé sur une carte de la France métropolitaine.

Ensuite, la troisième étape consiste à évaluer les impacts environnementaux, sociaux et économiques de chaque flux identifié dans l'étape 2. Pour cela, nous avons utilisé des outils et des modèles appropriés, tels que des facteurs d'émission, des indicateurs d'impact environnemental, des indicateurs de coût, etc. Par ailleurs, nous aurions pu faire une analyse de sensibilité pour évaluer l'incertitude des résultats.

De plus, la quatrième étape consiste à interpréter les résultats de l'analyse et à les communiquer aux parties prenantes, donc dans notre cas, c'est vous à travers ce rapport. Cela implique la hiérarchisation des impacts environnementaux, sociaux et économiques, de mettre en évidence les sources d'incertitudes, de proposer des pistes d'amélioration, etc.

Enfin, la cinquième étape consiste à prendre des mesures pour améliorer la durabilité du site web permettant de trouver le plus court chemin. Cela implique de modifier les fonctionnalités du site web pour réduire les impacts environnementaux, de choisir des fournisseurs plus durables pour les données, d'optimiser les performances énergétiques du site web, etc.

## 4.2 - Réflexion sur les usages

On peut réfléchir sur les différents points positifs et négatifs qu'implique l'usage de notre application sur l'environnement.

Tout d'abord, il y a plusieurs points positifs qui permettent de réduire l'émission de gaz à effet de serre ou encore les différentes consommations d'énergies.

Ainsi, la recherche d'un chemin plus court permet de réduire l'émission de gaz à effet de serre, car le trajet en voiture est plus rapide.

Par ailleurs, notre application se concentre principalement sur le calcul du plus court chemin entre deux villes, ainsi, celle-ci répond à la définition d'un **produit numérique durable** (un produit optimisé, qui se concentre sur les principales problématiques auxquelles il doit répondre).

Néanmoins, on peut relever plusieurs éléments qui ont un impact négatif sur la situation énergétique actuelle et qui peuvent entraîner un changement de nos habitudes.

De plus, les terminaux utilisés ainsi que les serveurs où sont hébergées les applications consomment énormément d'eau et d'électricité, ce qui entraîne un impact négatif sur l'environnement. Comme le confirme le professeur Andrew Ellis, de l'université Aston, la consommation d'énergie mondiale de l'économie digitale va doubler d'ici à 2030, cette prédiction est confirmée par les récents rapports de Greenpeace [6].

Par ailleurs, Tim Frick, dans son ouvrage *Designing for Sustainability* estime à 40% la part du frontend sur l'empreinte écologique totale d'un produit numérique [4]. Ainsi, on peut se poser la question : le design d'une application est-il vraiment essentiel pour qu'elle soit utilisée ?

## 4.3 - Leviers d'action

Afin de limiter l'impact environnemental d'une application, nous pouvons envisager de mettre en place plusieurs points de levier d'action.

D'une part, nous pouvons utiliser les applications GPS pour proposer des trajets moins polluants, mais aussi proposer des trajets alternatifs à la voiture comme le vélo ou les transports en commun.

Dès qu'un utilisateur cherche un trajet, l'itinéraire proposé peut être accompagné d'une estimation de la pollution causée. On pourrait par exemple connaître la quantité de gaz à effets de serre émise en fonction du mode de transport choisi.

De plus, l'application pourrait être capable de proposer des trajets un peu plus longs en termes de distance, mais avec une estimation de pollution moins élevée.

Enfin, les applications GPS peuvent sensibiliser ses utilisateurs avec des messages lors de recherche d'itinéraire en voiture. L'application peut par exemple recommander à l'utilisateur de réaliser son trajet à pied ou à vélo si le trajet est court.

# Conclusion

Dans le cadre de notre projet de développement d'une application complexe de calcul d'itinéraires, notre objectif était d'optimiser les performances de l'application tout en offrant une interface utilisateur ergonomique.

Nous avons réussi à atteindre ces objectifs en effectuant une analyse approfondie de l'application, en réalisant des tests de vitesse, en optimisant la base de données et l'algorithme et en améliorant l'ergonomie de l'application. Nous avons également pris en compte l'impact environnemental de l'application en proposant une analyse du cycle de vie et en réfléchissant à ses différents usages.

Du point de vue technique, ce projet nous a permis de consolider nos compétences en développement web, notamment en PHP, Twig et JavaScript. Nous avons pu approfondir notre connaissance des architectures MVC (Modèle-Vue-Contrôleur) et des frameworks comme Symfony, ainsi que découvrir des outils tels que Composer, un gestionnaire de dépendances pour les projets PHP. Nous avons également appris à effectuer des tests de performance et à optimiser une base de données.

Surmontant les difficultés techniques, nous avons réussi à améliorer les performances de l'application en réduisant le temps de réponse et en optimisant les requêtes à la base de données. Nous avons également réussi à améliorer l'expérience utilisateur en proposant une interface plus intuitive et en ajoutant des fonctionnalités telles que la possibilité de sauvegarder des itinéraires préférés.

Finalement, d'un point de vue humain, ce projet nous a permis de travailler en équipe et de mettre en pratique les méthodes agiles, en particulier la méthode Scrum. Nous avons réussi à communiquer efficacement et à collaborer sur un même projet, en attribuant des tâches à chaque membre de l'équipe et en fixant des objectifs à court terme.

# Bibliographie

- [1]  
« Comment évaluer et réduire les impacts environnementaux des sites Web ? - Hello Future Orange », *Hello Future*, 8 janvier 2021. <https://hellofuture.orange.com/fr/comment-evaluer-et-reduire-les-impacts-environnementaux-des-sites-web/> (consulté le 3 avril 2023).
- [2]  
P. E. conception, « Définition des objectifs et du champ de l'étude (ACV) - Pôle Eco conception », [eco-conception.fr](https://www.eco-conception.fr/static/definition-des-objectifs-et-du-champ-de-l-etude-ACV.html). <https://www.eco-conception.fr/static/definition-des-objectifs-et-du-champ-de-l-etude-ACV.html> (consulté le 3 avril 2023).
- [3]  
M. Gatti, « La consommation énergétique des applications mobiles les plus populaires », *Eco CO2*, 23 mai 2019. <https://www.ecoco2.com/blog/la-consommation-energetique-des-applications-mobiles-les-plus-populaires/> (consulté le 3 avril 2023).
- [4]  
contributeur, « Les applications (aussi) peuvent être pensées pour limiter leur impact écologique », *FRENCHWEB.FR*, 6 mai 2019. <https://www.frenchweb.fr/les-applications-aussi-peuvent-etre-pensees-pour-limiter-leur-impact-ecologique/357224> (consulté le 3 avril 2023).
- [5]  
« PhpStorm: l'IDE et éditeur de code PHP de JetBrains », *JetBrains*. <https://www.jetbrains.com/fr-fr/phpstorm/> (consulté le 3 avril 2023).
- [6]  
« Pollution numérique : comment la réduire ? », *Greenpeace France*, 20 mars 2023. <https://www.greenpeace.fr/la-pollution-numerique/> (consulté le 3 avril 2023).
- [7]  
« Quand Google Maps améliore les temps de trajet grâce à l'IA ». <https://www.intelligence-artificielle-school.com/actualite/google-maps-ameliore-temps-trajet-ia/> (consulté le 3 avril 2023).
- [8]  
K. DERUDDER, « The environmental impact of search engines apps », *Greenspector*, 22 juillet 2020. <https://greenspector.com/en/search-engines/> (consulté le 3 avril 2023).
- [9]  
« Waze, Google Maps, Mappy... les applications GPS vont devoir inciter les conducteurs à rouler plus vert - Le Parisien ». <https://www.leparisien.fr/environnement/waze-google-maps-mappy-les-applications-gps-vont-devoir-inciter-les-conducteurs-a-rouler-plus-vert-08-08-2022-ZWSKZTOPK5H4ZJ4TYFF57CRI7Q.php> (consulté le 3 avril 2023).
- [10]  
« Xdebug - Debugger and Profiler Tool for PHP ». <https://xdebug.org/> (consulté le 3 avril 2023).