Rapport de stage

Remerciements

Je tiens à remercier chaleureusement M. SCHNEKENBURGER, fondateur d'Altameos Multimédia, de nous avoir accueillis dans le cadre de notre stage de BTS SIO option SLAM.

Je le remercie particulièrement pour le temps qu'il nous a consacré, sa disponibilité et les conseils qu'il nous a apportés tout au long de cette expérience. Malgré le déroulement du stage en distanciel, il a su assurer un suivi de notre travail et nous transmettre son expérience professionnelle dans les domaines du développement web, de la gestion de projet et du numérique.

Je souhaite également remercier mon camarade de binôme Romain CANIAUX avec qui j'ai collaboré durant toute la durée du stage. Notre coopération, nos échanges et notre entraide ont contribué à la réalisation du projet qui nous a été confié.

Enfin, j'adresse mes remerciements à l'équipe pédagogique du BTS SIO ainsi qu'à l'ensemble de nos enseignants pour leur accompagnement et leurs conseils tout au long de cette période de formation.

Sommaire

Table des matières

1. Présentation de l'entreprise

1.1 Histoire

Altameos Multimédia est une agence web et digitale créée en 2003 par Lucas SCHNEKENBURGER, conçue pour accompagner les entreprises, artisans et PME dans leur transformation numérique. Cette entreprise a progressivement élargi ses compétences pour répondre aux évolutions du digital. Elle met en avant une démarche fondée sur la proximité avec ses clients et l'adaptation aux nouvelles technologies.

1.2 Secteur d'activité

Altameos Multimédia évolue dans le secteur des technologies de l'information, du web et de la communication digitale. Ses principales activités sont :

- ✓ Création de sites internet et e-commerce
- ✓ Développement web et mobile
- ✓ Hébergement et maintenance de solutions numériques
- ✓ Référencement naturel et optimisation de visibilité
- ✓ Stratégie de communication digitale
- ✓ Conception d'interfaces centrées sur l'expérience utilisateur

1.3 Organisation de l'entreprise

Cette entreprise a une organisation centrée sur l'accompagnement personnalisé des clients, avec une prise en charge complète des projets, de la conception à la mise en ligne et au suivi. L'entreprise s'appuie également sur une veille technologique continue afin d'intégrer les innovations numériques à ses solutions.

2. Présentation du service d'accueil et des moyens informatiques

2.1 Présentation du service d'accueil

Mon stage s'est effectué au sein de l'entreprise Altameos Multimédia, une entreprise spécialisée dans le développement de solutions numériques sur mesure.

Le service d'accueil correspond directement à l'activité de développement de l'entreprise. Il est chargé de la conception, du développement et de l'évolution des applications web réalisées pour les clients. Ce service intervient tout au long du cycle de vie des projets informatiques, depuis l'analyse des besoins jusqu'à la mise en production des solutions.

Les projets développés par Altameos sont principalement des applications web orientées métier, nécessitant une forte adaptation aux besoins spécifiques des utilisateurs finaux. Le

service assure ainsi la réalisation de solutions fonctionnelles, évolutives et adaptées aux contraintes techniques et organisationnelles des clients.

Dans le cadre de mon stage en distanciel, j'ai intégré ce service afin de participer à un projet de développement web lié à l'enregistrement et la reconstitution de trajets, ainsi que différentes statistiques selon le profil utilisateur. L'environnement de travail est organisé de manière flexible, permettant un suivi des tâches à distance tout au long du projet.

2.2 Moyens humains

L'entreprise Altameos Multimédia est une structure composée d'une seule personne. Cette personne occupe l'ensemble des fonctions liées à l'activité de l'entreprise, à savoir le rôle de développeur, chef de projet et interlocuteur client. Elle assure donc à la fois l'analyse des besoins, la conception technique, le développement des applications, ainsi que leur mise en production et leur maintenance.

Dans ce contexte, les échanges liés au projet de stage se faisaient directement avec cette personne, ce qui permettait une communication rapide et un suivi précis de l'avancement des tâches. Le fonctionnement en distanciel renforçait cette organisation centralisée, tout en nécessitant une bonne autonomie dans la réalisation des missions confiées.

2.3 Moyens techniques

2.3.1 Moyens techniques matériels

Dans le cadre de mon stage, le travail s'est effectué en distanciel, ce qui impliquait l'utilisation d'un équipement personnel. J'ai donc utilisé un ordinateur portable permettant de réaliser l'ensemble des tâches de développement et de test. Une connexion internet stable était également indispensable afin de garantir les échanges avec l'entreprise, l'accès aux ressources en ligne et la bonne utilisation des outils collaboratifs.

2.3.2 Moyens techniques logiciels

Les développements réalisés dans le cadre du projet reposaient principalement sur les frameworks Angular et Symfony, utilisés respectivement pour la création de l'interface utilisateur des applications web et pour la connexion avec la base de données, Symfony étant utilisé comme API. L'environnement de développement comprenait également l'éditeur de code Visual Studio Code, permettant l'écriture et la gestion du code source, ainsi que l'environnement de développement local WampServer permettant de simuler un serveur web et un environnement de base de données afin de tester les applications en local.

3. Présentation du projet

3.1 Contexte

Dans le cadre de la numérisation croissante des activités sportives et du développement des objets connectés, le suivi des déplacements et de l'activité physique est devenu un besoin courant pour de nombreux utilisateurs. De nombreuses applications permettent aujourd'hui

d'enregistrer des données de géolocalisation afin d'analyser les trajets réalisés et d'obtenir des informations sur l'effort physique fourni. C'est dans ce contexte que nous a été demandé ce projet, qui vise à mettre en place une application de suivi d'activité capable d'exploiter les données de position GPS collectées depuis un appareil mobile ou un ordinateur. Cette solution s'inspire des applications de tracking sportif en proposant une gestion simplifiée du suivi des déplacements et de l'activité physique de l'utilisateur.

3.2 Objectifs

L'objectif du projet est alors de concevoir et développer une application web dédiée au suivi des activités sportives. Cette application a pour principaux de :

- Collecter automatiquement ses positions GPS depuis un mobile ou un ordinateur
- Détecter et reconstituer les trajets effectués
- Calculer les distances parcourues
- Estimer les calories dépensées en fonction de son âge, poids, taille et du type d'activité détectée ou déclarée

4. Développement du projet

4.1 Présentation des outils utilisés

Avant de développer sur les outils nouvellement découverts lors de ce stage, il paraît essentiel d'expliquer les outils indispensables au bon déroulement du projet.

Tout d'abord, nous avons utilisé tout au long de celui-ci l'outil Visual Studio Code afin de pouvoir développer notre projet. Il s'agit d'un IDE (Environnement de Développement Intégré) développé par Microsoft qui permet d'éditer du code dans de nombreux langages de programmation tels que ceux qui vont nous intéresser dans notre projet : le HTML, le CSS, le PHP et le TypeSript.

Ensuite, pour héberger et exécuter l'application web sur un poste de développement, nous avons utilisé WampServer. Il s'agit d'un environnement de développement web destiné aux systèmes Windows. Il regroupe plusieurs composants indispensables au fonctionnement d'une application web : le serveur web Apache, le système de gestion de base de données MySQL et l'interpréteur PHP.

Enfin, pour organiser notre travail et noter le travail effectué chaque jour pendant notre stage, nous avons utilisé Trello. Il s'agit d'un outil de gestion de projets et d'organisation du travail permettant de suivre l'avancement des tâches de manière visuelle à l'aide de tableaux et de cartes. Ainsi il servait également à noter le travail effectué et surtout de retracer le processus nous amenant à la réussite de chaque tâche.

4.1.2 Angular 

Dans ce stage, nous avons appris à utiliser de nouveaux outils, nécessaires à la complétion de notre projet. Parmi ces nouveaux outils, on retrouve Angular. Il s'agit d'un framework open source développé par Google, il permet la création d'applications web dynamiques et interactives. Basé sur le langage TypeScript, il offre une architecture modulaire qui facilite l'organisation du code et la maintenance des projets de grande envergure. Angular repose sur le principe des composants, qui permettent de découper l'interface utilisateur en éléments

réutilisables et indépendants. Dans le cadre de ce projet, Angular a été utilisé pour développer l'interface web de l'application, gérer l'affichage des données, les interactions avec l'utilisateur ainsi que la communication avec l'API du serveur. Son utilisation a permis de concevoir une application performante, structurée et offrant une expérience utilisateur fluide. Cette expérience utilisateur est assurée par le fonctionnement en SPA (Single Page Application), qui permet une navigation fluide et quasi instantanée au sein de l'application, sans rechargement complet des pages.

4.1.3 Symfony

Nous avons aussi découvert l'outil Symfony, il s'agit d'un framework open source développé en PHP, largement utilisé pour la conception d'applications web robustes et structurées. Il repose sur une architecture modulaire et suit le modèle MVC (Modèle-Vue-Contrôleur), ce qui facilite la séparation des responsabilités et la maintenance du code. Dans le cadre de ce projet, Symfony a été utilisé principalement comme API back-end, chargée de gérer la logique métier et les échanges de données avec l'application front-end développée en Angular. Il permet notamment de traiter les requêtes http envoyées depuis Angular, d'interagir avec la base de données et de fournir des réponses structurées au format JSON. Ainsi, Symfony joue un rôle central dans la communication entre l'interface utilisateur et les données du système, garantissant la fiabilité et la cohérence des informations transmises.

4.1.4 Leaflet

Leaflet est une bibliothèque JavaScript open source spécialisée dans la création de cartes interactives pour des applications web. Légère et performante, elle permet d'intégrer facilement des cartes issues de différents fournisseurs de tuiles (comme OpenStreetMap) et d'y ajouter des éléments dynamiques tels que des marqueurs, des tracés de parcours ou encore des zones géographiques. Dans le cadre de ce projet, Leaflet a été utilisée pour afficher et exploiter les données de géolocalisation collectées. Elle permet notamment de visualiser les trajets effectués par l'utilisateur sous forme de lignes sur la carte, ainsi que de positionner des points correspondant aux positions GPS enregistrées. Grâce à ses fonctionnalités, Leaflet contribue à rendre l'application plus intuitive et visuelle en facilitant la représentation des activités de l'utilisateur sur une carte interactive.

4.2 Méthodes utilisées

Pour réaliser ce projet, j'ai utilisé Trello, ce qui m'a permis de suivre étapes par étapes le déroulement du projet, de bénéficier d'une progression concrète de ce dernier et de rédiger le processus de réalisation de chaque tâche.

Ainsi, l'avancement du projet s'effectuait généralement de manière progressive, tâche par tâche. Par exemple, chaque page de l'application constituait une tâche à part entière. La majorité des tâches suivait alors le même processus : la réalisation de l'interface front-end avec Angular, puis le développement du back-end avec Symfony afin de gérer l'envoi et la récupération des données nécessaires à l'affichage de la page concernée.

4.3 Planification des tâches confiées

4.3.1 Organisation du stage

Tout au long du stage, les échanges avec le maître de stage se sont effectués à distance, ce qui a nécessité une bonne organisation et une communication régulière afin d'assurer le bon déroulement du projet.

Le stage a été réalisé en binôme avec un autre étudiant de BTS SIO : Romain CANIAUX. Bien que nous travaillions chacun sur nos propres développements, nous restions en contact afin d'échanger sur nos avancées respectives. Lorsqu'un problème technique se présentait, nous nous entraidions en partageant nos connaissances et en proposant des solutions, ce qui nous a permis de progresser plus efficacement tout au long du stage.

4.3.2 Chronologie des travaux

Lors de la première semaine, l'objectif principal était de découvrir les technologies utilisées dans le projet. J'ai ainsi pris en main Angular pour le développement de l'interface utilisateur et Symfony pour le développement côté serveur. Cette phase a permis de comprendre le fonctionnement général de ces frameworks ainsi que leur rôle dans l'architecture de l'application.

La deuxième semaine a été consacrée à la mise en pratique de ces technologies. Après la phase d'apprentissage, j'ai commencé à développer différentes fonctionnalités du projet afin de me familiariser avec les bonnes pratiques associées à Angular et Symfony.

Au cours de la troisième semaine, un nouvel outil a été introduit dans le projet : Leaflet. Cette bibliothèque JavaScript spécialisée dans l'affichage de cartes interactives a nécessité une phase de découverte et d'expérimentation. J'ai exploré ses principales fonctionnalités et réalisé plusieurs essais afin de comprendre son fonctionnement.

La quatrième semaine a été consacrée à la poursuite du développement du projet en intégrant davantage les fonctionnalités offertes par Leaflet. Cette période a permis de mettre en place différentes fonctionnalités cartographiques tout en consolidant les connaissances acquises lors des semaines précédentes.

La cinquième semaine a été consacrée à l'amélioration du système de tracking avec la mise en place de la détection de l'inactivité de l'utilisateur et d'un mode automatique détectant l'activité de l'utilisateur sans son intervention. J'ai également développé l'affichage de la liste des trajets avec la possibilité de consulter le détail et les statistiques de chacun, avant de sécuriser les communications de l'application en HTTPS.

La sixième semaine a principalement été dédiée au perfectionnement de l'application à travers diverses améliorations et corrections. J'ai notamment ajouté la suppression des trajets depuis la liste des trajets, puis terminé le stage en réalisant les premières étapes du déploiement de l'application.

4.3.3 Outils de suivi utilisés

Afin d'organiser le travail et de suivre l'avancement du projet, plusieurs outils de communication et de gestion ont été utilisés durant le stage.

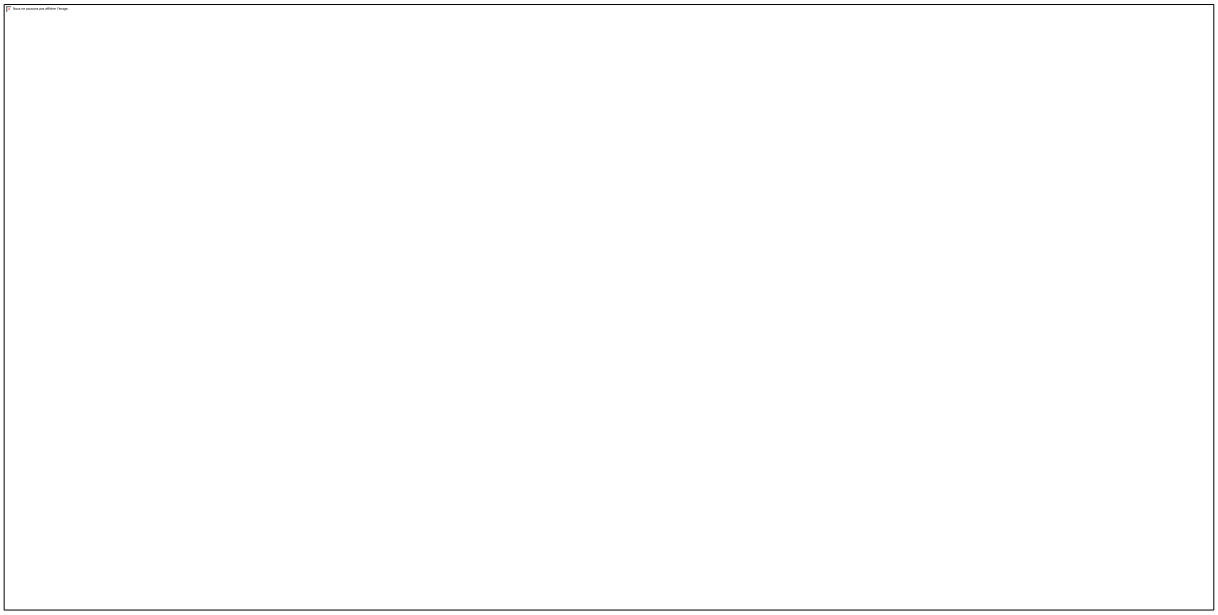
L'outil Trello a servi à planifier et suivre les différentes tâches à réaliser. Il permettait de visualiser l'état d'avancement du projet et de garder une vue d'ensemble sur les objectifs à atteindre.

Les échanges avec le maître de stage se faisaient principalement via Discord. Cet outil a facilité la communication à distance et le partage d'informations.

Enfin, Teams a été utilisé pour communiquer avec les enseignants du BTS SIO. Ces échanges ont permis d'obtenir des conseils pour répondre aux éventuelles difficultés rencontrées.

4.4 Déroulement du projet

Les différentes étapes de réalisation de notre projet concernent les 3 premières étapes du schéma ci-après, soit l'étude / l'initialisation, la conception ainsi que l'implémentation / prototype / test.



4.4.1 Etude / Initialisation

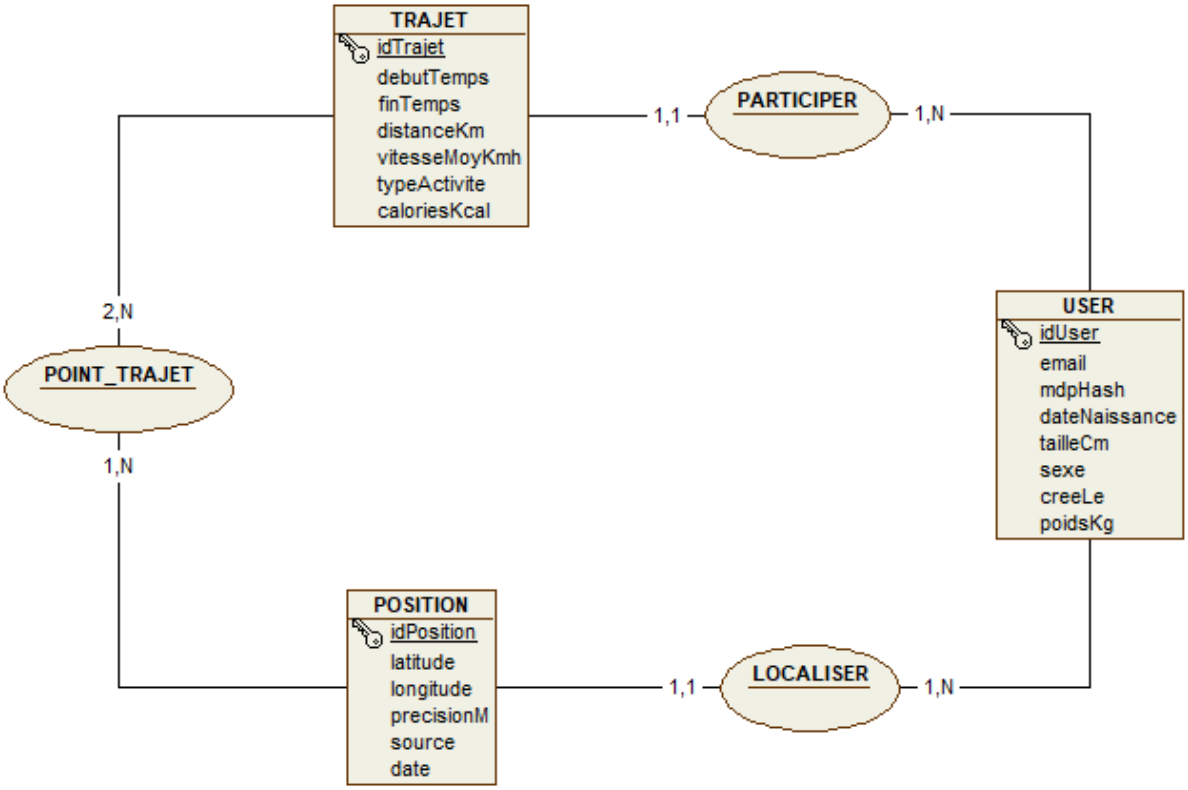
La phase d'étude et d'initialisation a consisté à analyser le besoin exprimé dans le cahier des charges. Ainsi, les objectifs du projet sont :

- Authentifier un utilisateur
- Créer un compte
- Collecter les positions GPS de l'utilisateur
- Détecter et reconstituer les trajets effectués
- Calculer les distances parcourues
- Mode simple : choisir un type d'activité (marche / course / vélo)
- Mode automatique : déduction du type d'activité en fonction de la vitesse moyenne
- Estimer les calories dépensées en fonction de son âge, poids, taille et du type d'activité détectée ou déclarée.

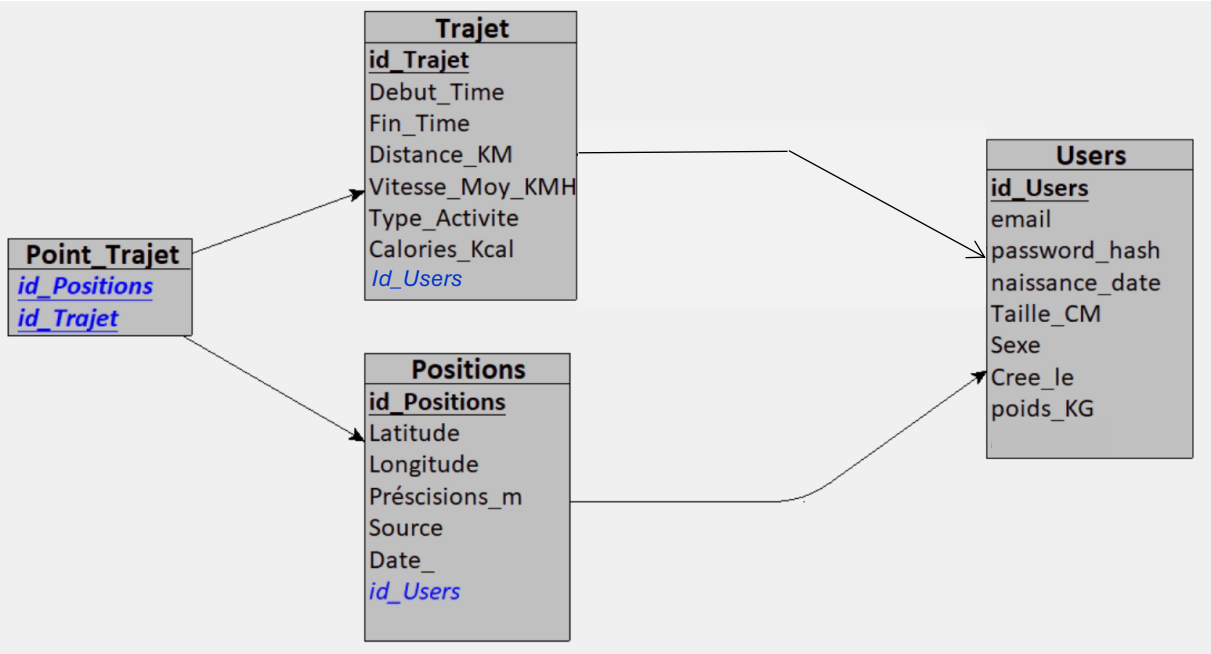
4.4.2 Conception

La phase de conception a permis de définir l'architecture globale de l'application. Celle-ci s'articule autour d'une interface utilisateur développée avec Angular pour la partie front-end, d'un back-end reposant sur une API REST conçue avec Symfony pour la gestion de la logique métier, et d'une base de données MySQL pour le stockage des données.

Nous avons créé un MCD afin de représenter la structure des données :



Nous l'avons ensuite transformé en MLD :



Nous avons ensuite terminé par la création de la base de données.

Les routes de l'API ont également été définies afin de permettre la gestion de l'authentification, la création d'un compte, l'affichage des données de l'utilisateur, la modification de ses données, la récupération des positions de l'utilisateur, l'affichage des positions lors d'un trajet allant d'un point A à un point B, recalculer un trajet, afficher la liste des trajets réalisés ainsi qu'afficher un trajet choisi.

De plus, des mécanismes de sécurité ont été implémenté, l'utilisation de mots de passe chiffrés, l'authentification par jeton JWT et la vérification du token sur chaque page de l'application.

4.4.3 Implémentation / Prototype / Test

4.4.3.1 Début

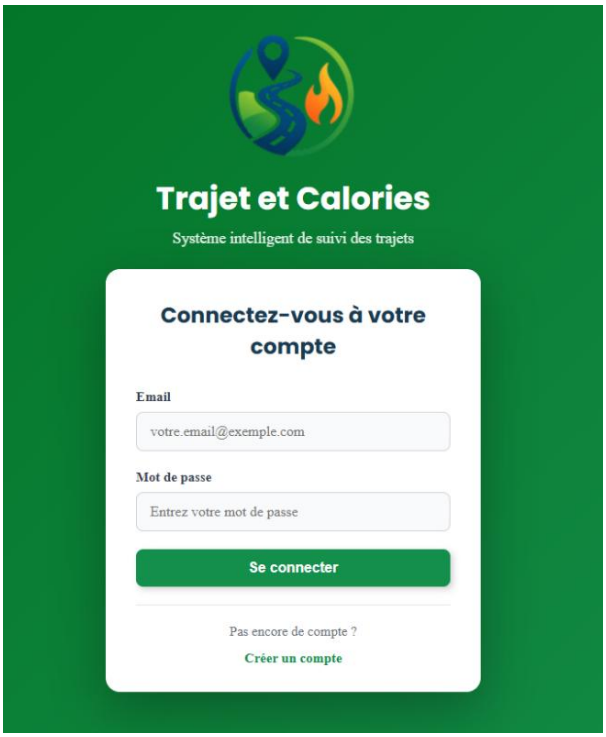
En premier lieu, il fallait créer un projet Angular avec la commande "ng new [nomProjet]"

Maintenant, l'objectif était de commencer à manipuler Angular. Avant de s'imposer l'utilisation du backend avec Symfony, je voulais tout d'abord créer l'interface d'une page d'authentification avec un lien vers une autre page, afin de commencer à manipuler les composants et les routes.

Ainsi, j'ai commencé par créer un nouveau composant en inscrivant dans le terminal la commande "ng g c Auth" (c'est l'abréviation de "ng generate component Auth"). La création de chaque composant permet d'ajouter un dossier dans /src/app avec 4 nouveaux fichiers à l'intérieur, en l'occurrence ici ces fichiers sont :

- auth.css : Ce fichier accueillera le CSS du composant créé
- auth.html : Ce fichier accueillera le HTML du composant créé
- auth.spec.ts : Ce fichier sert à vérifier que le composant fonctionne correctement, nous n'aurons jamais besoin de modifier ce genre de fichier à l'avenir
- auth.ts : Ce fichier accueillera la logique métier du composant

Dans cette page d'authentification j'ai créé 2 zones de texte où l'on doit saisir son email et son mot de passe, ainsi qu'un bouton « Se connecter ». Pour le moment, je ne m'occupais pas de la logique métier, ainsi on pouvait remplir ce qu'on voulait dans ces zones de texte, et appuyer sur le bouton « Se connecter » ne faisait rien.



Cependant j'ai commencé l'utilisation des routes. De cette manière, j'ai créé un nouveau composant "Créer" qui me servira pour la page de création de compte sur l'application. Dans le fichier app.routes.ts, j'ai créé 2 nouveaux chemins qui servent à faire le lien entre les 2 composants :

Cela m'a permis d'afficher sur la page d'authentification le lien "Créer un compte" qui m'amène sur le HTML du composant Créer (page de création de compte sur l'application).

Après la création de la page d'authentification, j'ai étudié le fonctionnement de Symfony, ce qui m'a permis d'élaborer un schéma théorique sur le voyage des données au sein de notre projet.

Pour prendre un exemple, quand l'utilisateur créera un compte, les données entrées par l'utilisateur seront récupérées par Angular, qui les enverra via une requête HTTP sur Symfony ; Symfony reçoit les données sur le contrôleur qui correspond, les données reçues sont *sérialisées*. A leur arrivée, les données reçues sont sous la forme d'objet en TypeScript, ce qui est incompréhensible par Symfony, par conséquent la *sérialisation* permet de transformer les données reçues de cet état illisible pour Symfony à un objet PHP, de sorte à pouvoir réaliser diverses opérations sur les données côté Symfony. Les données sont ensuite insérées dans la base de données et une réponse est renvoyée à Angular pour savoir si l'opération s'est bien réalisée ou s'il celle-ci a échoué.

4.4.3.2 Création des entités

Il faut maintenant créer un projet Symfony qui jouera le rôle d'API entre Angular et la base de données. Pour ce faire, il faut créer le projet, ainsi dans le terminal on exécute la commande "symfony new [nomProjet]". D'ailleurs je précise que pour utiliser Angular et Symfony, il faut lancer le serveur de chaque projet, le serveur de Angular se lance avec la commande "ng serve" et le serveur de Symfony se lance avec la commande "symfony server:start".

Avec ce nouveau projet Symfony, l'objectif est désormais de créer les différentes entités du projet qui seront ensuite migrées vers la base de données, afin de créer cette dernière avec les entités présentes dans le projet Symfony pour qu'il n'y est pas d'incohérence entre le projet et la base de données.

Pour créer une entité sur Symfony, il faut écrire dans le terminal la commande "php bin/console make:entity". Ensuite des questions nous sont posées pour connaître le nom de l'entité, les différentes propriétés de celle-ci, le type de chaque propriété ainsi que les éventuelles relations entre les entités. J'ai donc créé les entités « user », « trajet », « position » et « point_trajet » en suivant le même processus à chaque fois.

Après la création de toutes ces entités j'ai exécuté dans le terminal les commandes suivantes :

php bin/console make:migration (cette commande compare l'état actuel de la base de données avec les entités nouvellement créées et génère un fichier dans le dossier migrations/)

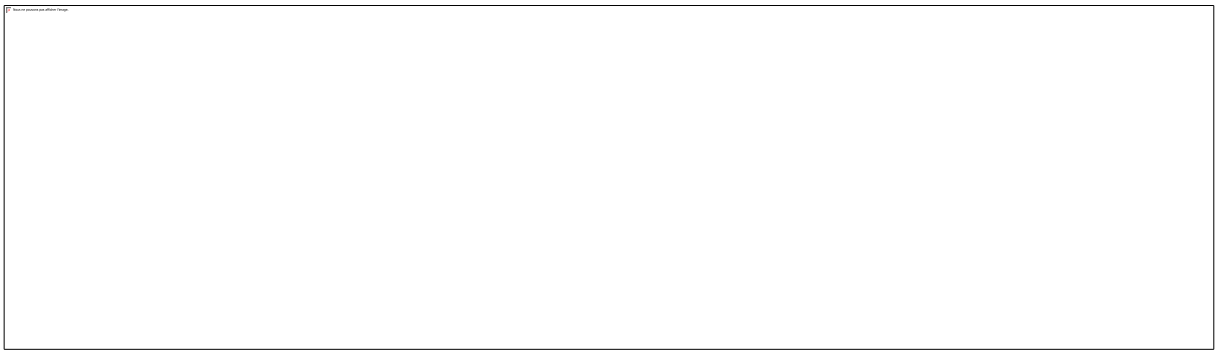
php bin/console doctrine:migrations:migrate (cette commande applique les changements à la base de données)

**Voir entité User Annexe 1*

4.4.3.3 Authentifier un utilisateur

Maintenant que j'ai un projet Angular, un projet Symfony et une base de données fonctionnels, on peut mettre en place un système d'authentification avec jeton JWT. Il faut pouvoir envoyer les informations entrées par l'utilisateur dans l'API, afin que celles-ci soient vérifiées et que l'accès à l'application soit autorisé ou non en fonction de la validité des informations entrées par l'utilisateur.

–A cet effet, il faut dans un premier temps créer un formulaire dans le fichier auth.ts sur Angular afin de récupérer à l'aide de la balise <form> les informations entrées par l'utilisateur lors de son authentification.

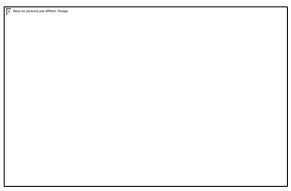


Ensuite, on crée un dossier "services" qui centralisera la logique métier et les accès aux données tout au long du projet afin ne pas submerger les composants. Puis, on crée au sein de ce dossier le fichier auth.service.ts où je vais écrire une fonction qui va envoyer une requête HTTP avec la méthode POST à Symfony. Cette fonction sera appelée chaque fois que l'utilisateur appuiera sur le bouton « Se connecter », ainsi la requête HTTP envoie l'email et le mot de passe entrés par l'utilisateur.

Sur Symfony, je crée un dossier « Api » dans le dossier « Controller » afin que les routes sur Symfony commencent toujours par /api/... . Ce dossier Api accueillera l'ensemble des contrôleurs de l'application. Je crée le contrôleur « AuthController » à l'aide de la commande « php bin/console make:controller [nomContrôleur] » dans le terminal. Dans celui-ci je commence par créer une route /api/login dans laquelle se réaliseront les opérations qui vont suivre. Dans la fonction « authenticate() » associée à cette route, je récupère les données envoyées depuis Angular concernant l'authentification de l'utilisateur et les transforme en objet PHP. Je commence ensuite par vérifier si l'email existe dans la base de données, s'il n'existe pas, je renvoie en réponse une erreur 401 (HTTP_UNAUTHORIZED). Si l'email existe, je vérifie que le mot de passe entré par l'utilisateur correspond bien au mot de passe relié à l'email correspondant dans la base de données. S'il n'est pas valide, je renvoie une erreur 401, sinon je renvoie une réponse 200 (HTTP_OK) pour montrer que l'authentification est valide.

**Voir fonction authenticate() Annexe 2*

Dorénavant, il faut renvoyer un jeton JWT pour vérifier ultérieurement dans l'application l'authentification de l'utilisateur sur chaque page de l'application. Avant tout, il faut installer les dépendances JWT à l'aide de la commande « composer require lexik/jwt-authentication-bundle », on crée ensuite un dossier « jwt » dans config/ pour ensuite générer les clés avec la commande « php bin/console lexik:jwt:generate-keypair » qui apparaissent dans le dossier nouvellement créé.



Ensuite, on va dans le fichier « .env » pour ajouter une passphrase secrète qui sécurise la clé privée utilisée pour signer les tokens.

Puis, on configure le fichier « lexik_jwt_authentication.yaml » avec la durée de vie du token que l'on souhaite :

Ensuite on configure le fichier security.yaml comme dans l'image ci-dessus

Une fois que tout cela est bien mis en place, je peux créer le token et l'envoyer dans la réponse de Symfony en cas d'authentification valide. Celui-ci est récupéré par Angular et est stocké temporairement dans localStorage (mémoire du navigateur) pour vérifier continuellement dans l'utilisation de l'application l'authentification de l'utilisateur, il s'agit d'une couche de sécurité supplémentaire.

4.4.3.4 Créer un compte

L'objectif de cette partie est de créer la page de création de compte afin qu'un utilisateur ne disposant pas encore de compte puisse créer un compte et se connecter ensuite avec ce compte.

Pour l'instant, inutile de créer de nouveau composant étant donné que j'avais déjà créé le composant « Creer » précédemment.

Je commence par créer l'interface de ce composant avec une zone de texte pour les éléments suivants : email, mot de passe, nom, prénom, date de naissance, genre, taille et poids. J'ajoute également un bouton « Valider » qui servira à appliquer la logique métier liée à ce composant. Pour le moment, je procède de la même manière que pour l'authentification de l'utilisateur, je crée un formulaire dans le fichier creer.ts qui récupère les informations entrées par l'utilisateur. Je crée ensuite un fichier creer.service.ts dans le dossier « services » pour séparer la requête HTTP du composant Créer. Lorsque l'utilisateur clique sur « Valider », les informations du formulaire sont envoyées via la requête HTTP, présente dans la fonction du fichier creer.service.ts, avec la méthode POST qui permet l'envoi de données.

**Voir Interface « Créer un compte » Annexe 3*

Je crée ensuite côté Symfony un nouveau contrôleur : RegisterController que je crée toujours dans le dossier Api. Dans ce contrôleur, je vérifie la validité de l'email afin de s'assurer que le format de celui-ci correspond bien à un format d'email. Dans le cas où l'email est invalide, on renvoie une erreur 400 (HTTP_BAD_REQUEST) pour montrer qu'il y a bien une erreur, et dans le cas où les informations sont toutes correctes, on procède à l'ajout des informations de l'utilisateur dans la base de données et on renvoie le code 201 (HTTP_CREATED) pour dire que le compte a bien été créé. Lors de l'arrivée de la réponse, un message est affiché sur l'interface en fonction du code retourné par Symfony, afin de prévenir l'utilisateur de l'échec ou de la réussite de l'opération. En cas de réussite, l'utilisateur peut alors s'authentifier avec son nouveau compte.

**Voir fonction register() Annexe 4*

4.4.3.5 Création page d'accueil et Menu

Maintenant que la création d'un compte et que l'authentification d'un utilisateur est possible, il faut créer la page d'accueil de l'application.

Pour cela, j'ai créé un nouveau composant Accueil. J'ai ensuite mis en place la route qui permet d'accéder à l'interface liée à ce nouveau composant lorsque l'utilisateur clique sur « Se connecter », en ajoutant notamment un chemin dans app.routes.ts et en faisant les imports nécessaires entre les différents composants.

Il faut maintenant créer quelque chose de primordial : le menu. Pour y parvenir, je commence par créer un nouveau composant que j'appelle « Sidebar », car il s'agit d'un menu que j'ai décidé de placé sur la partie gauche de l'interface. Ce dernier composant abritera tout le code lié au menu. Dans ce menu, j'ai mis les éléments suivants :

- Accueil
- Session
- Parcours
- Profil
- Déconnexion

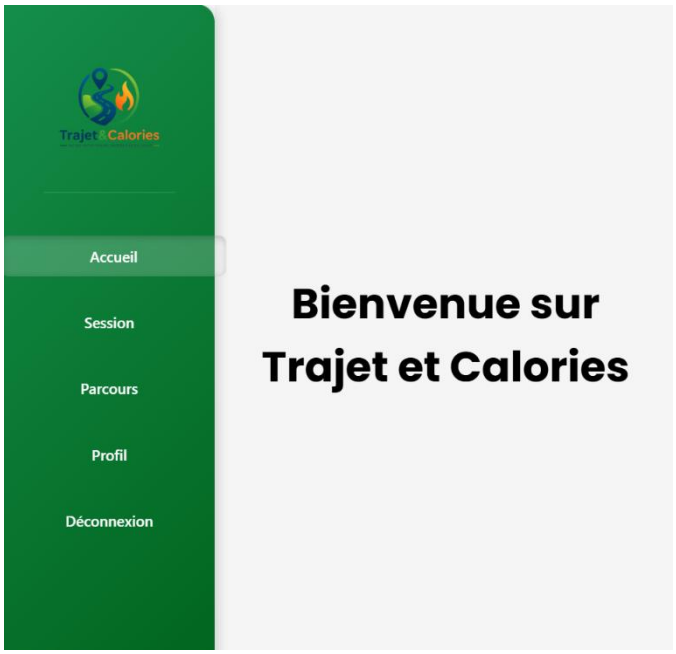
J'ai également ajouté en haut de ce menu le logo de notre application, en le rendant cliquable pour qu'à l'instar de l'élément « Accueil », il nous ramène à la page d'accueil.

Pour chaque élément présent dans ce menu, il faut créer une route qui va lui être associée dans le fichier app.routes.ts.

D'ailleurs, je me suis occupé directement de l'élément « Déconnexion » qui nous renvoie simplement sur la page d'authentification, donc inutile de créer ni un composant, ni une route concernant cet élément du menu.

Le composant lié au menu s'appelle alors <app-sidebar/>, je l'appelle donc dans le HTML de mon composant « Accueil » afin d'afficher le menu, je ferai de même pour toutes les autres pages de l'application où le menu sera visible en permanence.

Voici à quoi ressemble la page d'accueil :

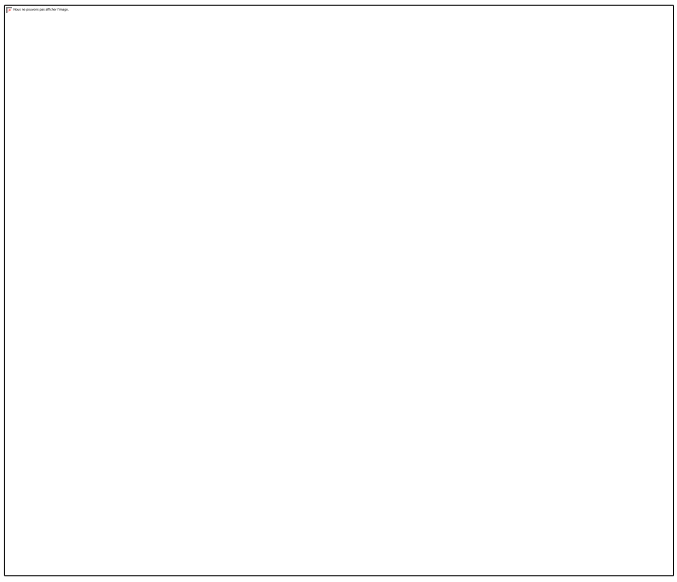


4.4.3.6 Création page profil

4.4.3.6.1 Affichage des informations

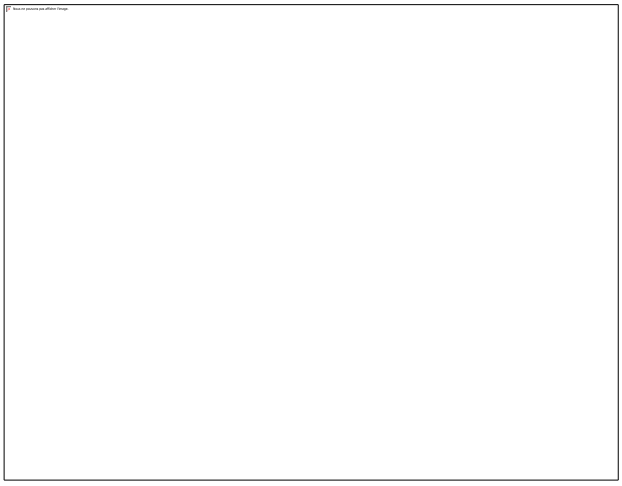
Après avoir créé le menu, j'ai créé un nouveau composant : Profil. Ce composant permettra d'afficher toutes les informations liées à l'utilisateur connecté, avec la possibilité s'il le souhaite, de modifier ses informations.

Je commence donc par faire le lien entre l'élément « Profil » du menu et le composant correspondant grâce à un nouveau chemin dans app.routes.ts. Une fois le lien effectué, je crée l'interface de la page Profil où je crée une zone de texte pour chaque information de l'utilisateur, sauf pour le sexe qui est inscrit dans une liste déroulante. Il faut maintenant que lorsque j'arrive sur la page, les informations de l'utilisateur connecté s'affichent dans les zones de textes.



Pour réaliser cela, je crée un nouveau fichier « profil.service.ts » dans le dossier « services » où je vais, dans une fonction appelée lors de l'initialisation du composant Profil (c'est-à-dire dans la fonction ngOnInit() qui permet d'exécuter du code directement lors de l'appel du composant) écrire une requête HTTP avec la méthode GET afin de recevoir les informations de l'utilisateur et les afficher sur la page Profil. Je précise que dans le header de la requête HTTP, j'ajoute le token envoyé auparavant lors de l'authentification de l'utilisateur afin de vérifier qu'il s'agit toujours du bon utilisateur, dans le cas contraire, un message d'erreur 401 (HTTP_UNAUTHORIZED) sera envoyé dans la console.

Je crée donc dans Symfony un nouveau contrôleur : ProfileController. Dans celui-ci je crée une route /api/me qui récupère les informations de l'utilisateur connecté dans la base de données. Ces données sont alors renvoyées en réponse vers Angular.



```
//Formulaire de modification du profil (initialement vide)
this.modifForm = new FormGroup({
  nom: new FormControl("", Validators.required),
  prenom: new FormControl("", Validators.required),
  email: new FormControl("", [Validators.required, Validators.email]),
  birthdate: new FormControl("", [Validators.required, Validators.pattern(/^\d{4}-\d{2}-\d{2}$/)]),
  height: new FormControl(0, [Validators.required, Validators.min(0)]),
  sex: new FormControl("", [Validators.required]),
  weight: new FormControl(0, [Validators.required, Validators.min(0)])
});
```

J'avais précédemment créé dans profil.ts un formulaire vide (voir ci-dessus) qui va maintenant servir à récupérer les données envoyées par Symfony. Je récupère donc les données dans ce formulaire et j'affiche les données de ce formulaire au chargement de la page. Par conséquent,

lorsqu'un utilisateur se rend sur la page profil, toutes les informations liées à son profil sont affichées.

4.4.3.6.2 Modification des informations

Maintenant que j'ai créé le composant profil et que j'ai rempli les données avec celles de l'utilisateur authentifié, il faut que l'utilisateur puisse modifier ses données à sa guise.

Pour y parvenir, j'ai affiché les informations de l'utilisateur dans des zones de texte afin que celles-ci puissent être directement modifiées. J'ai alors ajouté en bas de la page un bouton « Valider les modifications » et créé dans le fichier profil.service.ts une nouvelle fonction « saveChanges() » appelée lorsque l'utilisateur clique sur le bouton, cette fonction envoie une requête HTTP contenant toutes les informations de l'utilisateur (y compris celles modifiées) avec la méthode PUT grâce au formulaire.

```
saveChanges(data: { nom: string, prenom: string, email: string, birthdate: string, height: number, sex: boolean, weight: number
  const token = localStorage.getItem('token');

  return this.http.put<{ message: string }>('https://localhost:8000/api/me', data, {
    headers: {
      Authorization: `Bearer ${token}`
    }
  });
}
```

Je ne crée pas de nouveau contrôleur dans Symfony étant donné qu'il s'agit toujours de la page Profil, en revanche, je crée une nouvelle route : /api/me (avec la méthode PUT cette fois-ci) avec une nouvelle fonction. Dans cette fonction, je vérifie la validité des données envoyées. En cas d'invalidité des données, je renvoie une erreur 400 (HTTP_BAD_REQUEST), en cas de succès, je renvoie le code 200 (HTTP_OK).

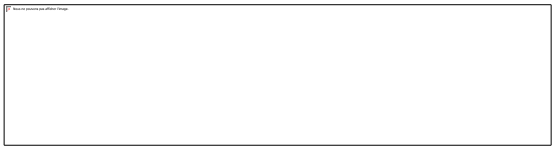
S'il y a une erreur lors de l'opération, un message est affiché en bas de la page pour signaler une erreur lors de l'opération, et dans le cas contraire, un message de succès est affiché : « Modifications enregistrées ».

**Voir fonction « updateProfile » Annexe 5*

4.4.3.7 Création page parcours (début Leaflet)

Maintenant, l'objectif est de créer la page Parcours. J'ai ainsi créé le composant Parcours et j'ai ajouté un chemin dans app.routes.ts afin de relier l'élément « Parcours » du menu au composant correspondant. Dans cette page, je souhaitais afficher une carte sur la partie gauche de la page (à droite du menu) pour afficher ultérieurement des trajets, ainsi qu'à droite de la page, des détails concernant les futurs trajets affichés.

Pour afficher une carte, il faut exécuter la commande « npm install leaflet » ainsi que la commande « npm install --save-dev @types/leaflet » pour installer les types TypeScript dans le terminal. Il faut aussi importer le css Leaflet dans le fichier angular.json :



Après ces configurations, j'ai créé le composant « Map » qui affichera une carte lorsqu'il sera appelé. J'ai ensuite appelé ce composant dans mon composant Parcours afin d'afficher la carte à gauche comme je le souhaitais. En récupérant la position GPS de l'utilisateur, à ce moment-là, j'affichais simplement sur la carte la position de l'utilisateur avec un marqueur notant « Vous êtes ici ».

Par la suite, j'ai essayé de mettre ne place en une seule fois tout le processus d'enregistrement et d'affichage d'un trajet, or, en procédant de cette manière, je n'ai pas réussi. De ce fait, j'ai décidé d'aborder le problème d'une autre manière en commençant par la fin et en procédant étapes par étapes :

- Etape 1 : Afficher un trajet sur la carte avec des valeurs codées en dur dans Angular (sans utiliser Symfony)
- Etape 2 : Enregistrer manuellement des données dans la table « Position » de la base de données et afficher un trajet en récupérant ces positions via la méthode GET.
- Etape 3 : Envoyer des positions depuis Angular vers la base de données afin de les enregistrer, pour ensuite afficher le trajet correspondant à ces données initialement envoyée depuis Angular
- Etape 4 : Enregistrer des positions à l'aide de la géolocalisation dans la base de données et afficher le trajet.

Malheureusement, je n'ai une nouvelle fois pas réussi à atteindre la dernière étape. Les 2 premières étaient un succès, en revanche j'ai éprouvé, une fois de plus, des difficultés pour enregistrer des positions dans la base de données. Cela m'a conduit à remettre en question ma vision de la fonction du composant « Parcours » dans l'application.

Ainsi, je ne vais plus enregistrer des trajets depuis cette page comme initialement souhaité, mais je vais simplement afficher, lors de l'arrivée de l'utilisateur sur la page, le dernier trajet réalisé par celui-ci, en affichant sur la partie gauche de la page le tracé de son trajet, et sur la partie droite les informations concernant ce dernier trajet.

J'ai créé un nouveau composant « InfosParcours » qui sera appelé dans le composant « Parcours » pour afficher les détails du trajet affiché sur la page (la partie droite de la page).

Dans le "parcours.ts", j'ai ajouté la fonction ngOnInit() qui permet de réaliser des opérations instantanément lors de l'appel du composant.

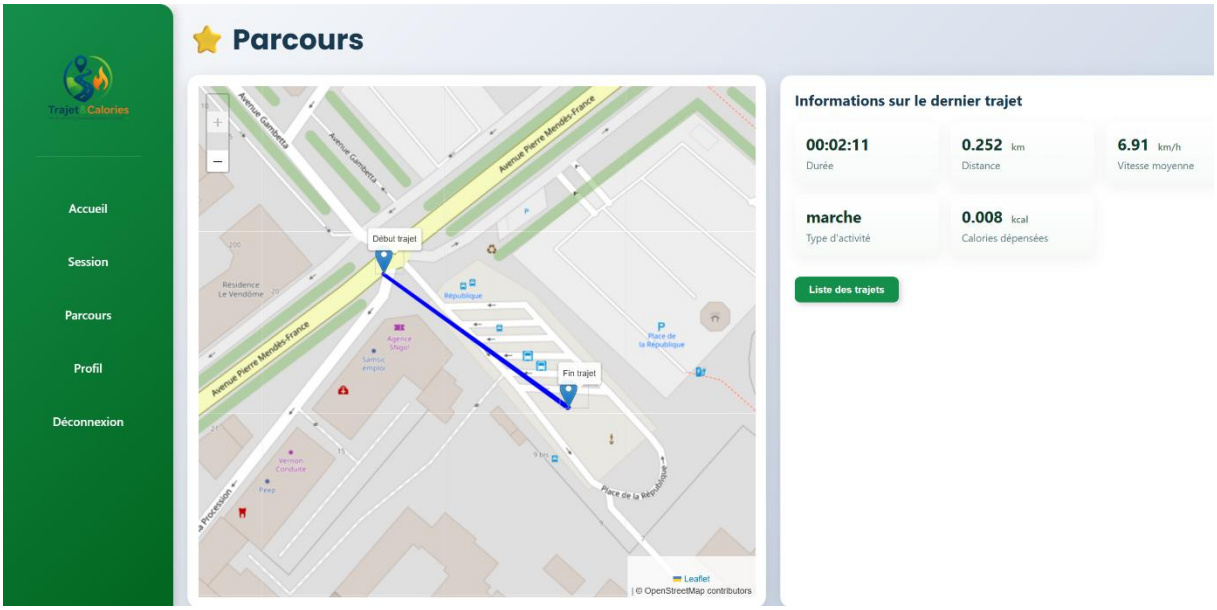
**Voir fonction ngOnInit() Annexe 6*

Dans cette fonction, je crée un formulaire qui me permettra ensuite de récupérer les informations du dernier trajet de l'utilisateur. J'appelle la fonction « getInfos » présente dans le fichier "parcours.service.ts" qui exécute une requête HTTP vers Symfony avec la méthode GET.

Dans le contrôleur TripController, je récupère les données liées au dernier trajet de l'utilisateur connecté et les renvoie en réponse. Je peux ainsi avoir accès aux données attendues depuis Angular, or je les récupère dans le composant « Parcours », et je veux les afficher dans le HTML du composant enfant « InfosParcours », c'est pourquoi j'envoie ces données vers le composant enfant en faisant dans le HTML du composant « InfosParcours » une liaison de propriété transmettant le formulaire. Je récupère dans une variable ce formulaire sur le composant « InfosParcours » à l'aide de @Input et affiche les données dans le HTML.

Maintenant que j'ai les informations liées au dernier trajet de l'utilisateur actuellement connecté, je souhaite afficher le tracé du trajet en question. De cette façon, dans la fonction ngOnInit() du composant « Parcours », après les opérations précédemment listées, j'ai appeléune fonction pour récupérer les positions du trajet dont les données ont été récupérées auparavant, cela est possible car lorsque je renvoie les données du trajet depuis Symfony, j'ai ajouté l'id du trajet afin de pouvoir déterminer le trajet à tracer sur la carte. J'appelle donc la fonction « getPosition() » présente dans le service "parcours.service.ts" qui fait une requête HTTP vers Symfony avec la méthode POST. Je reçois cette requête sur Symfony dans le contrôleur « TripController » et récupère toutes les positions liées au trajet souhaité à l'aide de l'id du trajet envoyé en paramètre. Les positions me sont alors envoyées dans le composant « Parcours », or, je souhaite les envoyer dans le composant « Map » pour les afficher sur la carte, ainsi je procède de la même façon que pour les informations du trajet, je réalise une liaison de propriété dans le HTML du composant « Parcours », je récupère les données à l'aide de @Input dans le composant « Map » et peux tracer le trajet correspondant sur la carte en ajoutant des marqueurs pour affiché le point de départ et le point d'arrivée du trajet.

Voici le rendu de la page Parcours :



4.4.3.8 Création page Session

4.4.3.8.1 Interface

L'objectif de cette partie est de créer l'interface de la page Session de l'application, cette page permettra à l'utilisateur d'enregistrer ses sessions de sports à l'aide du tracking. Celle-ci sera composée du menu à gauche, d'une carte en bas de la page montrant la position actuelle de l'utilisateur, d'une partie de texte en haut à droite qui montrera, à la fin d'une session, les statistiques liées à celle-ci, et enfin en haut entre le menu et cette dernière partie, les différents boutons pour permettant le choix du mode et de l'activité souhaités par l'utilisateur (*Voir rendu ci-après*).

Je commence par créer le composant Session. Je veux ensuite afficher une carte montrant la position GPS actuelle de l'utilisateur. Pour cela, je vais dans session.ts et je récupère la position actuelle de l'utilisateur dans la fonction ngOnInit() :

```
ngOnInit() {
  navigator.geolocation.getCurrentPosition(
    (position) => {
      this.position = {
        lat: position.coords.latitude,
        lng: position.coords.longitude
      };

      this.cdr.detectChanges();
    },
    (error) => {
      console.error('Erreur de géolocalisation :', error);
      alert('Impossible de récupérer votre position. Veuillez vérifier les permissions de géolocalisation et réessayer.');
    },
    {
      enableHighAccuracy: true,
      maximumAge: 0,
      timeout: 20000
    }
  );
}
```

J'envoie les données liées à la position de l'utilisateur vers le composant Map à l'aide d'une liaison de propriété dans session.html. Ensuite je récupère ces données dans le composant Map à l'aide de @Input. J'ajoute une condition dans la fonction ngOnChanges() où je vérifie la validité des données reçues, puis si ces données sont valides, j'appelle la fonction afficherPointDepart() qui permet d'afficher une carte centrée sur les coordonnées de l'utilisateur. J'ai par ailleurs ajouté un marqueur affichant « Point de départ ».

Par la suite j'ai recréé le CSS de la page afin de structurer la page et de présenter quelque chose de plus moderne et harmonieux.

Maintenant on va s'intéresser à la partie de la page qui concerne les choix de l'utilisateur quant à l'activité qu'il souhaite exercer. Dans cette partie, j'ai créé une liste déroulante qui permet de choisir entre 2 modes :

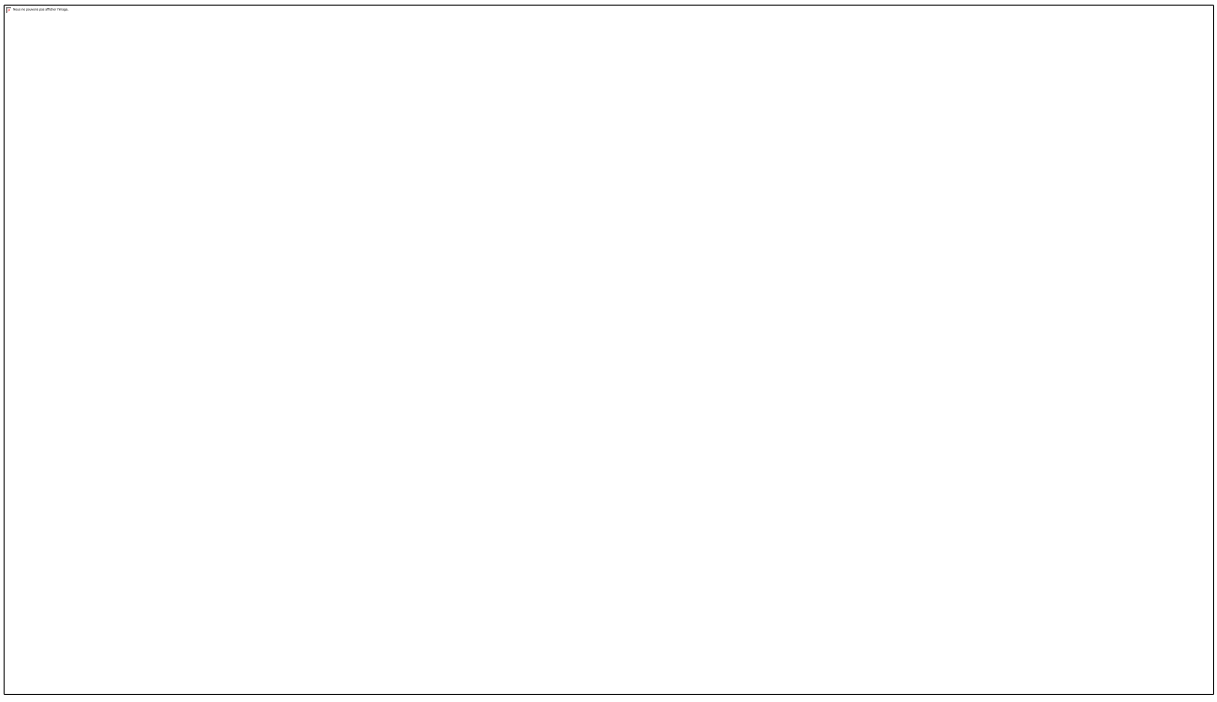
- Mode Simple : L'utilisateur choisit lui-même l'activité qu'il souhaite exercer
- Mode Automatique : Le programme comprend par lui-même l'activité que l'utilisateur exerce

En dessous j'ai mis 3 boutons qui permettent à l'utilisateur de choisir son activité :

- Marche
- Course
- Vélo

Et enfin, en bas j'ai mis un bouton Démarrer et un bouton Arrêter qui serviront plus tard lorsque le tracking sera mis en place.

Voici le rendu final de la page Session (j'ai rajouté en plus le choix de l'intervalle de temps entre les prises de positions GPS de l'utilisateur) :



L'étape suivante était la gestion des boutons, puisque le bouton Démarrer ne doit par exemple pas être accessible si l'utilisateur n'a pas encore choisi d'activité ou activé le mode automatique.

Pour mettre en place la gestion des boutons, j'ai commencé par le cas où l'utilisateur choisit le mode Simple (choix par défaut). Dans ce cas, l'utilisateur doit choisir une activité parmi les 3 proposées pour activer le bouton Démarrer. J'ai commencé par créer une fonction selectActivity() qui est appelée lorsque l'utilisateur clique sur un des boutons liés aux activités, et lors de ce clique une variable « activity » prend la valeur de l'activité sélectionnée par

l'utilisateur (à l'aide de la fonction). Ensuite j'ai créé une fonction selectMode() qui permet, en suivant un principe similaire, d'enregistrer le choix du mode de l'utilisateur dans une variable. J'ai aussi créé la fonction demarrerSession() qui permettra de démarrer une session mais qui ne permet pour l'instant que d'activer le bouton Arrêter et de désactiver le bouton Démarrer. J'ai créé ensuite la fonction arreterSession() associée au bouton Arrêter qui débloque le bouton Démarrer et bloque le bouton Arrêter. De plus, j'ai créé 2 getters : get activityDisabled() et get actionButtonsDisabled() qui permettent respectivement, de renvoyer un booléen s'adaptant à l'activité choisie ainsi qu'adapté au statut de la session (lancée ou non), et de renvoyer un booléen en fonction du mode choisit par l'utilisateur et en fonction de la sélection d'une activité. Concrètement, le getter get activityDisabled() permet d'activer ou non les types d'activité, et le getter get actionButtonsDisabled() permet d'activer ou désactiver le bouton Démarrer.

Je précise que pour griser les boutons lorsqu'ils sont désactivés, il fallait simplement rajouter du CSS.

Voir l'utilisation de toutes ces fonctions Annexe 7

4.4.3.8.2 Tracking

Maintenant, on va mettre en place le tracking, c'est-à-dire la récupération des positions GPS de l'utilisateur à intervalle régulier afin de les envoyer dans la base de données et de créer un trajet associé à ces positions.

Avant toute chose je souhaitais afficher un timer qui se lance lorsque je clique sur Démarrer et qui s'arrête lors du clic sur Arrêter. Pour implémenter cela, j'ai ajouté de nouvelles fonctions dans session.ts :

- startTimer() qui appelle la fonction updateTimer() et qui se charge de lancer le timer
- stopTimer() qui se charge d'arrêter le timer
- updateTimer() qui met à jour l'affichage du timer afin que l'utilisateur puisse voir en temps réel le timer.

Voici les méthodes liées au timer :

Dans la fonction demarrerSession(), j'ai ajouté cette ligne qui initialise le timer à 0 : this.dureeLabel = '00:00:00';

Dans arreterSession(), j'ai ajouté this.stopTimer(); afin de stopper le timer.

J'ai également rajouté cette ligne dans le HTML afin d'afficher le timer :

```
<h3>Temps : {{dureeLabel}}</h3>
```

L'objectif est désormais de mettre en place le suivi de position GPS de l'utilisateur et d'enregistrer les positions. Pour l'enregistrement dans les tables Trajet et Point_trajet dans la base de données, je m'en suis occupé après, je voulais y aller petit à petit et commencer par enregistrer dans la base de données mes positions uniquement. J'ai décidé d'aborder les choses ainsi car j'avais déjà essayé de mettre en place le tracking 2 fois, mais ce fut 2 échecs, c'est pour cela que mon approche est de faire les choses progressivement.

Dans le fichier session.ts, j'ai commencé par créer l'ensemble des variables qui me serviront pour réaliser le processus d'enregistrement des positions GPS de l'utilisateur. Je précise que j'ai pris mes captures d'écran après avoir terminé toute la partie, pour simplement enregistrer les positions de l'utilisateur, je n'avais pas besoin de toutes les variables de la capture :

```
position!: {
  lat: number;
  lng: number;
};

//Initialisation des variables qui déterminant le choix de l'activité de l'utilisateur
mode: 'simple' | 'automatique' = 'simple';
activity: 'marche' | 'course' | 'velo' | null = null;
actionEnabled = false;

message: any = null;
//Variables qui déterminent l'état de la session
sessionState: 'idle' | 'activity-selected' | 'ready' | 'running' = 'idle';
sessionStarted: boolean = false;

//Variables du timer
timer: any = null;
startTime = 0;
dureeLabel = '00:00:00';

//Variables du timer d'inactivité
messageInactivity: string = "";
inactivityTimer: any = null; //id du timer lancé
inactivityTimeOut = 60 * 1000;

//Variables pour le processus de session
distance: number = 0;
distanceTotale: number = 0;
listePoints: any[] = [];
nbPositions: number = 0;
dateDebut: Date = new Date();
dateFin: Date = new Date();
poids: number = 0;

//Pour stocker l'id du trajet créé au début
trajetId: number | null = null;

//Variables pour les calculs
MET: number = 0;
vitesseMoy: number = 0;
calories: number = 0;

//Variable pour l'intervalle de temps
intervalle: number = 5;
gpsInterval: any = null;

//Variable pour déterminer la cause de l'arrêt du trajet (volontaire ou inactivité)
arreteParInactivite: boolean = false;
```

J'ai ensuite modifié la fonction demarrerSession() dans laquelle je réinitialise toutes les variables nécessaires au processus afin de ne pas avoir de problème si l'utilisateur enregistre plusieurs trajets d'affilée (chaque variable devant être réinitialisée au début de l'enregistrement de chaque de trajet).

Par ailleurs, je rajoute l'appel de la fonction startTracking() qui va me permettre de lancer l'enregistrement des positions GPS de l'utilisateur à intervalle régulier (j'ai commencé par enregistrer les positions toutes les 5s).

Dans la fonction startTracking(), donc une fois que le tracking est lancé, je commence par appeler une 1ère fois la fonction recupererPosition() afin de récupérer la position de l'utilisateur au départ du trajet. Ensuite, à intervalle régulier, j'appelle une nouvelle fois cette fonction afin d'enregistrer la position de l'utilisateur toutes les 5s.

```
// Méthode permettant de lancer le tracking
startTracking(): void {
  if (!navigator.geolocation) {
    console.error('Géolocalisation non supportée');
    return;
  }

  // Première récupération immédiate
  this.recupererPosition();

  // On récupère la position de l'utilisateur toutes les 5 secondes
  this.gpsInterval = setInterval(() => {
    this.recupererPosition();
  }, 5000);
}
```

Dans la fonction recupererPosition() :

- Je récupère la position de l'utilisateur en stockant les infos liées à celle-ci dans une variable nouveauPoint
- Si ma variable listePoints contenant la liste des positions de l'utilisateur compte au moins un point avant l'enregistrement du nouveau point créé, je calcule la distance entre le dernier point enregistré et le nouveau point créé afin d'incrémenter la variable distanceTotale de la nouvelle distance calculée. Pour cela j'appelle la fonction calculDistance() qui utilise la formule de Haversine pour calculer la distance parcourue par l'utilisateur entre 2 positions GPS :

```
// Méthode calculant la distance entre 2 points
//avec la formule de Haversine
private calculDistance(
  lat1: number,
  lng1: number,
  lat2: number,
  lng2: number
  ): number {

  //Rayon de la Terre en mètres
  const R = 6371000;
  const dLat = (lat2 - lat1) * Math.PI / 180;
  const dLon = (lng2 - lng1) * Math.PI / 180;

  //Formule de Haversine
  const a =
      Math.sin(dLat / 2) * Math.sin(dLat / 2) +
      Math.cos(lat1 * Math.PI / 180) *
      Math.cos(lat2 * Math.PI / 180) *
      Math.sin(dLon / 2) *
      Math.sin(dLon / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

  return R * c;
}
```

- Ensuite j'ajoute le nouveau point créé à la liste des points enregistrés et incrémente de 1 le nombre total de positions enregistrées
- Ensuite, si la variable listePoint contient au moins 3 positions (il s'agit d'une valeur totalement arbitraire qui pourra changer ultérieurement), je fais une copie de cette variable que j'envoie en paramètre lors de l'appel de la fonction envoyerPosition() juste après. La copie de cette fonction me permet de réinitialiser directement le contenu de la variable listePoint afin de ne pas avoir de problèmes liés au manque de synchronisation des différentes opérations du programme. → J'envoie les positions par groupe afin de ne pas devoir toutes les envoyer d'un seul coup à la fin et pour ne pas non-plus les envoyer une par une, ce qui pourrait poser des problèmes de maintenance de l'application.

Voir fonction recupererPosition() Annexe 8

Voici la fonction envoyerPositions() qui envoie les positions par groupe (ne pas faire attention à la variable trajetid, il s'agit d'une variable rajoutée ultérieurement permettant de remplir aussi la table PointTrajet faisant le lien entre les tables Position et Trajet) :

```
envoyerPositions(positions: any[], trajetId: number): Observable<any> {
  const token = localStorage.getItem('token');

  const headers = {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`
  };

  return this.http.post(
      'http://localhost:8000/api/positions',
      { positions: positions, trajetId: trajetId },
      { headers }
  );
}
```

Dans cette fonction, j'envoie la liste des positions de l'utilisateur via la méthode POST sur la route créée sur Symfony /api/positions

Dans le contrôleur PositionController, je crée la fonction d'ajout des positions dans la base de données sur Symfony (idem, elle remplit aussi la table PointTrajet mais cela à été fait plus tard)

Voir fontion addPositions() Annexe 9

Dans celle-ci, je récupère les données, je procède à un certain nombre de vérifications afin de s'assurer de la validité des données reçues, puis j'enregistre pour chaque position sa latitude, sa longitude, sa date d'enregistrement, sa source et l'id de l'utilisateur connecté.

Je reviens ensuite sur Angular afin de modifier la fonction arreterSession() qui va appeler la fonction stopTimer() qui va arrêter le timer, puis appeler la fonction stopTracking() qui va arrêter le tracking, voici cette dernière fonction :

```
// Méthode qui permet d'arrêter le tracking
stopTracking(): void {
    if (this.gpsInterval) {
      clearInterval(this.gpsInterval);
      this.gpsInterval = null;
    }
}
```

```
//Au cas où des positions ne seraient pas envoyées
if (this.listePoints.length > 0) {
  const batch = [...this.listePoints];
  this.listePoints = [];

  this.trackingService.envoyerPositions(batch, this.trajetId!).subscribe({
    next: () => console.log('Dernier batch envoyé'),
    error: err => console.error(err)
  });
}
```

S'il reste des positions non envoyées dans la variable listePoint, je les envoie en suivant le même processus qu'expliqué précédemment.

A ce stade, le programme enregistre en continu les positions de l'utilisateur lorsqu'il clique sur Démarrer et arrête le processus de tracking lorsque l'utilisateur clique sur Arrêter. Désormais, l'objectif est d'enregistrer les données liées au trajet réalisé, pour cela il ne faut en plus d'enregistrer les positions liées à l'utilisateur connecté dans la table Positions, enregistrer un nouveau trajet dans la table Trajet et remplir la table PointTrajet avec chaque position de l'utilisateur associée au trajet nouvellement créé.

J'ai séparé cette partie en 3 étapes :

1) Créer, lors du clic sur Démarrer, un nouveau trajet en ne remplissant que les informations strictement nécessaires, soit l'id de l'utilisateur, le type d'activité et la date de début du trajet.

2) Remplir la table PointTrajet en même temps que la table PointTrajet liées grâce à l'id du trajet créé au début renvoyé lors de la création du trajet afin de pouvoir le réutiliser ensuite.

3) Mettre à jour le trajet créé au début avec les données statistiques liées au trajet réalisé.

Etape 1 : Créer un trajet au clic sur Démarrer

Pour réussir cela, je commence par modifier la fonction demarrerSession() en appelant la fonction creerTrajet() pour créer un nouveau trajet :

```
creerTrajet(dateDebut: Date){
    const token = localStorage.getItem('token');
    const headers = {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`
    };

    return this.http.post(
      'https://localhost:8000/api/trips',
      { dateDebut},
      { headers }
    );
}
```

Celle-ci envoie les informations à enregistrer dans la base de données liée au nouveau trajet, je fais donc une requête HTTP avec la méthode POST sur la route /api/trips de Symfony. Je crée donc la fonction createTrajet() dans le contrôleur TripController afin de créer un nouveau trajet dans la base de données :

Voir fonction createTrajet Annexe 10

Ensuite, dans demarrerSession() je récupère le poids de l'utilisateur en appelant la fonction getPoids() qui envoie une requête HTTP via la méthode GET sur la route /api/me/weigth de Symfony. Récupérer le poids de l'utilisateur permettra de calculer les calories dépensées lors du trajet.

```
getPoids() {
  const token = localStorage.getItem('token');
  return this.http.get<{ weight: number }>(
      'https://localhost:8000/api/me/weight',
      {
          headers: {
              Authorization: `Bearer ${token}`
          }
      }
  );
}
```

Ainsi, dans le contrôleur ProfileController, je crée la fonction getWeight() qui se charge de récupérer le poids de l'utilisateur connecté dans la base de données.

```
//Retourne le poids de l'utilisateur connecté
#[Route('/api/me/weight', name: 'get_weight', methods: ['GET'])]
public function getWeight(): JsonResponse {
    $user = $this->getUser();

    if (!$user) {
        return $this->json([
            'message' => 'Utilisateur non connecté'
        ], Response::HTTP_UNAUTHORIZED);
    }

    return $this->json([
        'weight' => $user->getPoidsKg()
    ], Response::HTTP_OK);
}
```

Ensuite, toujours dans demarrerSession(), je regarde le choix du mode de l'utilisateur en créant une première condition, celle où l'utilisateur choisit le mode simple, soit le mode où l'utilisateur choisit de lui-même l'activité qu'il souhaite. De cette manière, en fonction de l'activité choisie par l'utilisateur, j'initialise la variable du MET en conséquence. Cela permettra aussi de calculer les calories dépensées. Je ne mets pas de capture pour cela car plus tard j'ai déplacé ces conditions pour les mettre à la fin, ce qui est plus judicieux.

Etape 2 : Remplir table PointTrajet

Pour ajouter ces données dans la base de données, je rajoute en paramètre lors de l'appel sur Angular de la fonction envoyerPosition() l'id du trajet créé au début afin de pouvoir enregistrer les données dans la table PointTrajet.

```
envoyerPositions(positions: any[], trajetId: number): Observable<any> {
  const token = localStorage.getItem('token');

  const headers = {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`
  };

  return this.http.post(
      'https://localhost:8000/api/positions',
      { positions: positions, trajetId: trajetId },
      { headers }
  );
}
```

De cette manière, dans la fonction addPositions() sur Symfony, je rajoute quelques lignes pour enregistrer les positions de l'utilisateurs aussi dans la table PointTrajet, en liant chaque position au trajet créé au début à l'aide de son id envoyé. Cette table me permet de faire le lien entre mes tables Trajet et Position.

Voici les quelques lignes rajoutées dans la fonction addPositions() :

```
//Maintenant qu'on a rajouté une position, il faut aussi compléter la table Point_Trajet
$pointTrajet = new PointTrajet();
$pointTrajet->setTrajet($trajet);
$pointTrajet->setPosition($position);

$entityManager->persist($pointTrajet);
```

Etape 3 : Mettre à jour le trajet afin d'enregistrer les nouvelles informations

Pour cela, je modifie la fonction arreterSession() afin de récupérer la date de fin du trajet, de récupérer une valeur arrondie de la distance totale parcourue, de calculer la vitesse moyenne de l'utilisateur et de calculer les calories. Je mets ensuite une ligne permettant de rafraîchir la page automatiquement après le calcul des informations, afin de les afficher aux yeux de l'utilisateur.

```
//Calcul des infos indépendantes du choix du mode de l'utilisateur
this.dateFin = new Date();

//On calcule la durée en heures, celle-ci est calculée de 2 manières différentes en fonction de la cause de l'arrêt du trajet (volontair
let dureeMs = this.dateFin.getTime() - this.dateDebut.getTime();
if (this.arreteParInactivite) {
  dureeMs -= this.inactivityTimeOut;
}
const dureeHeures = dureeMs / (1000 * 3600);

this.vitesseMoy = Math.round(((this.distanceTotale) / dureeHeures) * 100) / 100; //2 chiffres après la virgule
```

Enfin, j'appelle la fonction terminerTrajet() en envoyant en paramètres toutes les nouvelles informations relatives au trajet créé au début. Cette fonction envoie une requête HTTP via la méthode PUT vers Symfony sur la route /api/trips/${trajetId} afin d'envoyer les données à modifier pour le trajet créé au début du processus de tracking.

```
terminerTrajet(trajetId: number, activity: string | null, distance: number, vitesseMoy: number,
  calories: number, dateFin: Date) {
  const token = localStorage.getItem('token');
  const headers = {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`
  };

  return this.http.put(
    `https://localhost:8000/api/trips/${trajetId}`,
    {
      activity,
      distance,
      vitesseMoy,
      calories,
      dateFin
    },
    { headers }
  );
}
```

Je crée donc la fonction updateTrajet() dans TripController sur Symfony qui se charge de rajouter les nouvelles données pour le trajet créé au début du processus dans la base de données.

Mon programme est désormais capable, lors du clic sur Démarrer, de créer un nouveau trajet, de récupérer les positions GPS de l'utilisateur toutes les 5 secondes, en les enregistrant dans la base de données par groupe, et d'arrêter le processus lors du clic sur Arrêter afin de calculer des statistiques liées au trajet effectué par l'utilisateur et de les enregistrer dans la BDD.

4.4.3.9 Amélioration page Session

L'objectif de cette partie est de continuer à perfectionner la page de Session afin de faire fonctionner le Mode Automatique et d'ajouter une liste déroulante permettant de choisir l'intervalle de temps entre la prise de chaque position GPS.

J'ai commencé par mettre en place le Mode Automatique, pour ce faire, j'ai arrêté d'envoyer l'activité lors de la création du trajet lors du clic sur Démarrer. Maintenant je n'envoie que la date de début de trajet, seulement à la fin afin de rendre le code plus optimisé et de ne pas copier des lignes de codes déjà écrites, et également pour ne pas devoir recréer de méthodes ou de routes simplement pour enregistrer une information en plus avec le Mode Automatique.

De cette manière, lorsque je crée un trajet lors du clic de l'utilisateur sur Démarrer, je n'enregistre plus l'activité sélectionnée, même pour le Mode Simple. J'ai donc déplacé dans arreterSession() la condition qui vérifie le choix du mode de l'utilisateur, et j'ai complété cette condition avec un *else* dans le cas où l'utilisateur choisit le Mode Automatique.

Dans celui-ci je détermine l'activité de l'utilisateur en calculant la vitesse moyenne avant les conditions. Si la vitesse moyenne de l'utilisateur est inférieure à 6km/h, celui-ci exerce de la marche, si elle est comprise entre 6 et 12km/h, il exerce de la course et si elle est supérieure, il fait du vélo. Cela me permet également de déterminer le MET afin de calculer les calories dépensées juste après.

De plus j'ai déplacé à la fin la réinitialisation de l'activité et du clic sur une activité dans le cas où l'utilisateur avait choisi le Mode Simple, afin de ne pas envoyer une valeur null dans l'activité, quel que soit le choix de l'utilisateur :

```
//Pour enlever le clic s'il y en avait un sur une des activités
this.activity = null;
this.actionEnabled = false;
```

J'ai rajouté dans le HTML une liste déroulante afin de proposer plusieurs intervalles de temps dans une liste déroulante :

```
<div class="modeBlock">
    <label for="modeSelect">Choisissez l'intervalle de prise de position :</label>
    <select id="modeSelect" class="modeSelect"
        [(ngModel)]="intervalle">
        <option [ngValue]="5" selected>5s</option>
        <option [ngValue]="10">10s</option>
        <option [ngValue]="15">15s</option>
        <option [ngValue]="20">20s</option>
        <option [ngValue]="30">30s</option>
    </select>
</div>
```

De cette manière, je récupère dans la variable « intervalle » la valeur associée à l'intervalle choisi et l'applique au setInterval :

Je précise que je multiplie par 1000 la valeur sélectionnée parce que le temps est compté en millisecondes.

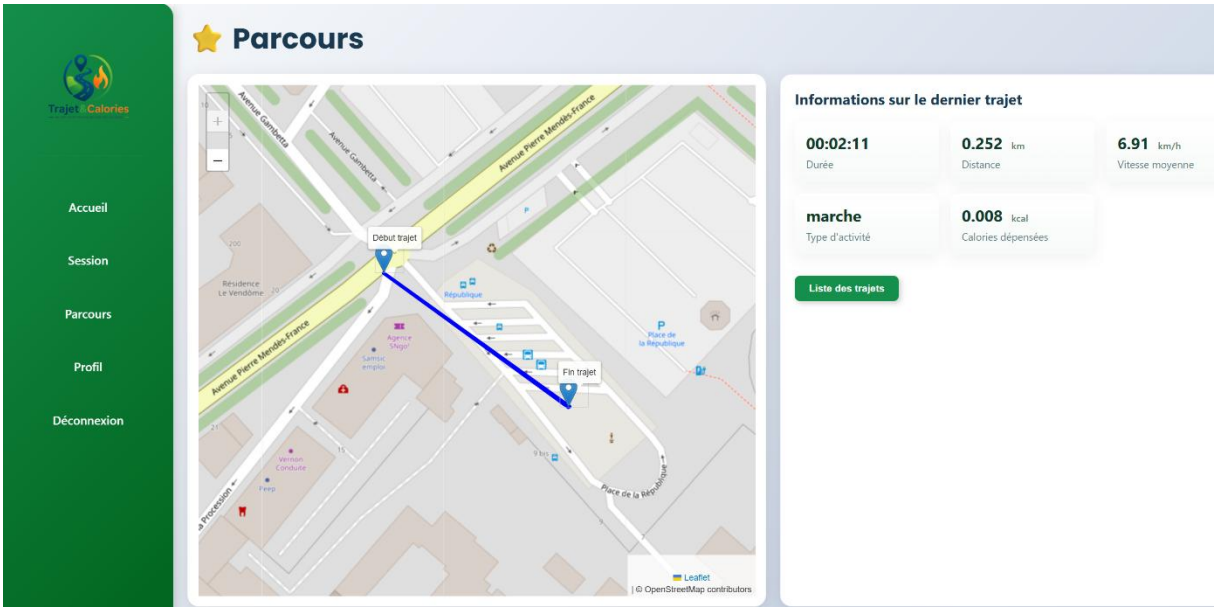
4.4.3.10 Afficher liste des trajets

L'objectif de cette partie est de pouvoir mettre en place sur la page Parcours un bouton qui nous permette d'accéder à la liste de tous les trajets réalisés par l'utilisateur.

```
<div class="buttonGroup">
    <button type="button" class="btnPrimary" (click)="openList()">Liste des trajets</button>
</div>
```

Pour cela, il fallait tout d'abord créer un bouton dans le HTML du fichier infos-parcours.html :

Voici le rendu de la page :



Ensuite, je suis retourné dans le composant Parcours et j'ai ajouté 2 variables :

```
//Variable pour changer l'affichage de la page
viewMode: 'current' | 'list' = 'current';
//Variable pour récupérer la liste de tous les trajets
trajets: any[] = [];
```

L'idée est de basculer sur 'list' quand on clique sur le bouton "Liste des trajets" ou sur 'current' quand on clique sur "Retour". La variable « trajets » permettra de stoker tous les trajets réalisés par l'utilisateur.

Actuellement, quand j'arrive sur la page Parcours de l'application, au sein du composant Parcours, j'affiche 2 composants : Map et InfosParcours. L'objectif est que lorsque je clique sur le bouton que "Liste des trajets", j'arrête d'afficher les composants Map et InfosParcours et que je bascule sur un autre composant : ListeTrajet, ce composant permettra d'afficher la liste de tous les trajets effectués par l'utilisateur.

Je commence donc par créer un nouveau composant : ListeTrajet.

Ensuite, je modifie le html de parcours.html de sorte à pouvoir basculer de l'affichage d'un trajet à la liste des trajets sur la page Parcours (en fonction de la valeur de viewMode) :

```
<app-sidebar/>
<div class="mainContent">
    <h1>⭐ Parcours</h1>
    @if (viewMode === 'current') {
        <div class="contentArea">
            <div class="mapSection">
                <app-map [trajetId]="idTrajet"> </app-map>
            </div>
            <div class="infoSection">
                <!-- Informations sur le parcours iront ici -->
                @if (idTrajet){
                    <app-infos-parcours [trajet]="infosParcours" (showList)="showTrajetsList()"> </app-infos-parcours>
                } @else {
                    <p class="PasTrajet">{{message}}</p>
                }
            </div>
        </div>
    }
    @if (viewMode === 'list') {
        <app-liste-trajets
            [trajets]="trajets"
            (trajetSelected)="selectTrajet(id: $event)"
            (retour)="showCurrent()">
        </app-liste-trajets>
    }
</div>
```

Je crée donc dans parcours.ts plusieurs fonctions qui serviront à faire ce basculement :

- showTrajetsList() qui change la variable viewMode en ‘list’ pour faire le basculement et qui récupère à l’aide de la fonction getAllTrajets() la liste de tous les trajets réalisés par l’utilisateur connecté.

```
//Méthode pour basculer de mode de vue (current -> list) et donc afficher tous les trajets
showTrajetsList() {
  this.parcoursService.getAllTrajets().subscribe({
    next: (res: any) => {
      this.trajets = res.trips;
      this.viewMode = 'list';
      console.log('Tableau trajets reçus :', this.trajets);
      this.cdr.detectChanges();
    },
    error: (err: any) => console.error(err)
  });
}
```

Je crée donc dans le fichier parcours.service.ts une fonction getAllTrajets() qui fait une requête HTTP via la méthode GET sur la route /api/trips/all vers Symfony :

```
getAllTrajets() {
  const token = localStorage.getItem('token');
  return this.http.get('http://localhost:8000/api/trips/all', {
      headers: {
        Authorization: `Bearer ${token}`
      }
  });
}
```

Je crée ensuite une fonction getTrips() relative à cette route dans Symfony afin de récupérer la liste de tous les trajets de l’utilisateur connecté.

Voir fonction getTrips Annexe 11

- showCurrent() qui permet de revenir dans le mode ‘current’ (affichage du dernier trajet réalisé par l’utilisateur) :

```
//Méthode pour basculer de mode de vue (list -> current)
showCurrent() {
  this.viewMode = 'current';
}
```

Ensuite je retourne dans le composant InfosParcours où je crée dans le fichier infos-parcours.ts une variable “showList” avec @Output afin d’envoyer lors du clic sur “Liste des trajets” l’information au composant parent “Parcours” qu’il faut activer la fonction pour changer l’affichage de la page afin d’afficher la liste des trajets de l’utilisateur. Je crée donc une fonction openList() qui me permet de faire cela au clic sur le bouton :

```
@Output() showList = new EventEmitter<void>();
```

```
//Fonction pour avertir le composant parent que le bouton a été cliqué
openList() {
  this.showList.emit();
}
```

Ainsi, lors du clique sur “Liste des trajets”, cette dernière fonction est appelée afin de créer un événement qui est envoyé au composant parent Parcours qui repère que l'événement a été créé et qu’il faut activer la fonction associée showTrajetList() permettant de basculer l’affichage de la page :

```
@if (viewMode === 'current') {
    <div class="contentArea">
        <div class="mapSection">
            <app-map [trajetId]="idTrajet"> </app-map>
        </div>
        <div class="infoSection">
            <!-- Informations sur le parcours iront ici -->
            @if (idTrajet){
                <app-infos-parcours [trajet]="infosParcours" (showList)="showTrajetsList()" </app-infos-parcours>
            } @else {
                <p class="PasTrajet">{{message}}</p>
            }
        </div>
    </div>
}
```

Je vais ensuite dans le composant ListeTrajets où je crée le HTML me permettant, à l’aide d’une boucle for, d’afficher l’entièreté des trajets réalisés par l’utilisateur sous forme de boutons :

```
@for (trajet of trajets; track trajet.id; let i = $index) {

    <button type="button" class="trajetItem" [attr.data-activite]="trajet.activite | lowercase" (click)="selectTrajet(trajet.id)">
        <div class="trajetItemHeader">
            <span class="activityBadge">
                <span class="activityIcon">{{ getActivityIcon(trajet.activite) }}</span>
                {{ trajet.activite || 'Autre' }}
            </span>
            <div class="trajetTitle">
                <strong>Trajet {{ trajets.length - i }}</strong>
                <span class="trajetMeta">{{ trajet.distance | number:'1.3-3'}} km</span>
            </div>

        </div>
        <div class="trajetItemFooter">
            <span class="trajetVitesse">{{ trajet.vitesseMoy || '-- ' }} km/h</span>
```

Par ailleurs, j’ai créé une fonction getActivityIcon() permettant d’afficher un icône en fonction de l’activité exercée durant le trajet :

Voir fonction getActivityIcon() Annexe 12

Voici le rendu de la page Parcours après clic sur « Liste des trajets » pour un utilisateur ayant exercé beaucoup de trajets :

Maintenant, l'objectif est que lorsque l'utilisateur clique sur un des trajets, on revienne sur l'interface d'avant mais avec les informations du trajet sélectionné.

Tout d'abord, voyons pour afficher les données statistiques du trajet sélectionné. Pour faire cela, je commence par créer une variable « trajetSelected » avec un @Output dans liste-trajets.ts :

```
@Output() trajetSelected = new EventEmitter<number>();
```

Ensuite, je crée la fonction selectTrajet() qui crée un événement envoyé au composant parent Parcours (à l'aide de @Output).

parcours.html :

```
@if (viewMode === 'list') {
    <app-liste-trajets
        [trajets]="trajets"
        (trajetSelected)="selectTrajet(id: $event)"
        (retour)="showCurrent()">
    </app-liste-trajets>
}
```

Le composant Parcours reçoit ensuite l'événement et appelle une autre fonction appelée aussi selectTrajet() :

```
selectTrajet(id: number) {
  this.idTrajet = id;
  this.viewMode = 'current';
  //On récupère les informations liées au trajet sélectionné
  this.parcoursService.getInfosById(id).subscribe({
    next: (res: any) => {
      this.data = res;
      console.log(this.data);
      this.infosParcours.patchValue(this.data);
      this.cdr.detectChanges();
    },
    error: (err) => {
      console.error('Erreur lors de la récupération des informations', err);
      this.message = "Erreur lors de la récupération des informations";
      this.cdr.detectChanges();
    }
  });
}
```

Cette fonction permet de repasser dans me mode ‘current’ et donc de repasser à l’ancienne interface. Ensuite, elle appelle la fonction getInfosById() qui fait une requête http sur la route /api/trips/${trajetId}/infos via la méthode GET vers Symfony :

```
getInfosById(trajetId: number) {
  const token = localStorage.getItem('token');
  return this.http.get(`https://localhost:8000/api/trips/${trajetId}/infos`, {
      headers: {
        Authorization: `Bearer ${token}`
      }
  });
}
```

La fonction getInfosTrajetById() permet de récupérer toutes les informations relatives au trajet sélectionné :

```
#[Route('/api/trips/{id}/infos', name: 'api_trip_id_infos', methods:['GET'])]
public function getInfosTrajetById(int $id, EntityManagerInterface $entityManager): JsonResponse
{
    $trajet = $entityManager->getRepository(Trajet::class)->find($id);
    if (!$trajet) {
        return $this->json([
            'error' => 'Trajet introuvable'
        ], Response::HTTP_NOT_FOUND);
    }

    $duree = $trajet->getDuree();

    return $this->json([
        'duree' => $duree ? $duree->format('%H:%I:%S') : null,
        'distance' => $trajet->getDistanceKm(),
        'vitesseMoy' => $trajet->getAvgSpeedKmh(),
        'typeActivite' => $trajet->getActivityType(),
        'caloriesDepensees' => $trajet->getCaloriesKcal(),
        'id' => $trajet->getId()
    ]);
}
```

Ensuite, il faut afficher le tracé du trajet sélectionné, pour cela, il suffit de rajouter quelques lignes de code qui permettent d’afficher le trajet en fonction de l’id du trajet :

```
if (this.trajetId) {
  console.log("Appel après init map :", this.trajetId);
  this.afficherTrajet(this.trajetId);
}
```

Ces lignes sont rajoutées dans le composant Map dans la fonction ngAfterViewInit(), car dans l'ordre d'exécution des différentes fonctions, on est certain que cette fonction est exécutée après que les @Input soient récupérés, ce qui n'est pas le cas avant. De cette manière, après avoir cliqué sur un trajet dans la liste, on affiche non seulement les informations liées à ce trajet mais aussi le tracé du trajet correspondant.

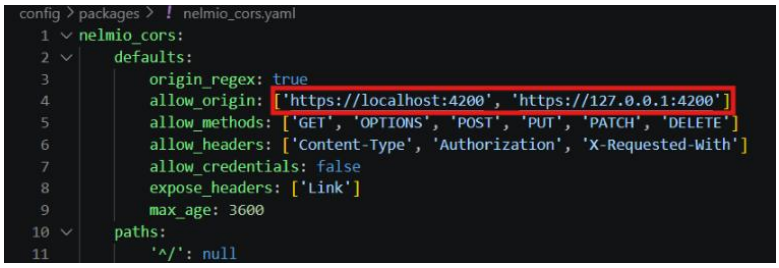
4.4.3.11 Communication HTTPS

L'objectif de cette partie est de mettre en place une communication HTTPS au sein du projet afin de sécuriser les échanges entre les utilisateurs et le serveur.

Je me place dans mon projet Angular et je vais dans le terminal. Je crée un dossier "ssl" à l'aide de la commande « mkdir ssl ». Ensuite je tape la commande « openssl req -x509 -newkey rsa:4096 -keyout ssl/localhost.key -out ssl/localhost.crt -days 365 -nodes ». Cette dernière a pour effet de créer un certificat et une clé privée. Ensuite je peux lancer le serveur avec cette commande : « ng serve –ssl ».

Je vais ensuite dans tous mes services Angular, afin de modifier les requêtes HTTP et remplacer les "http" par "https".

Ensuite, j'ouvre un autre terminal et me place sur mon projet Symfony et je tape la commande « symfony ca:install ». Ensuite, je vais dans le fichier nelmio_cors.yaml du projet Symfony où je remplace les 'http' en 'https' afin d'autoriser l'origine HTTPS de l'application Angular :



Ainsi, les échanges entre l'application Angular et l'API Symfony sont effectués via une connexion HTTPS sécurisée.

4.4.3.12 Détecter l'inactivité de l'utilisateur

4.4.3.12.1 Détecter l'inactivité v1

L'objectif de cette partie est de déterminer l'inactivité de l'utilisateur lorsqu'il a démarré un trajet, afin de le terminer.

Pour ce faire, il faut tout d'abord établir un seuil d'inactivité. J'ai choisi de considérer que l'utilisateur est inactif s'il parcourt en 5s moins de 4m, ce qui correspond à peu près à 3 km/h.

Par conséquent, il y a proportionnalité entre l'intervalle de temps de la prise de positions GPS choisi par l'utilisateur et le seuil d'inactivité :

Intervalle	Distance maximale d'inactivité
5s	4m
10s	8m
15s	12m
20s	16m
30s	24m

Ainsi, comme il y a proportionnalité entre l'intervalle choisi par l'utilisateur et la distance maximale d'inactivité. J'ai créé une formule pour calculer la distance max d'inactivité de l'utilisateur en fonction de l'intervalle qu'il choisit :

distance = intervalle - (intervalle / 5)

Ainsi, dans session.ts je crée la variable "inactivite" correspondant au temps d'inactivité de l'utilisateur (en secondes)

```
//Variable pour calculer le temps d'inactivité de l'utilisateur
inactivite: number = 0;
```

Ensuite, au démarrage du trajet (dans la fonction demarrerSession()), j'initialise la variable à 0. Puis, je vais dans la fonction recupererPosition(), juste après le calcul de la distance, je mets une condition qui va regarder si la distance parcourue par l'utilisateur entre les 2 derniers points est inférieure à la distance max d'inactivité, si elle l'est, on incrémente la variable "inactivite" de l'intervalle choisi au départ par l'utilisateur. Dans le cas contraire, on réinitialise la variable "inactivite" à 0, car si l'utilisateur est actif, c'est que le trajet n'est pas terminé, et s'il ne l'est pas, on réinitialise la variable :

```
//On regarde si l'utilisateur a été inactif
if (distance < (this.intervalle - (this.intervalle / 5))){
  //Si l'utilisateur est inactif, on incrémente le compteur mesurant le temps d'inactivité
    this.inactivite += this.intervalle;
}
else{
  this.inactivite = 0;
}
```

Je vais ensuite dans startTracking() pour ajouter dans le setInterval() une condition pour regarder si l'utilisateur est inactif depuis 5 minutes (300 secondes). S'il ne l'est pas, on continue de récupérer les positions à intervalle régulier, s'il l'est, on arrête le tracking :

```
// On récupère la position de l'utilisateur toutes les 5 secondes
this.gpsInterval = setInterval(() => {
  console.warn("Inactivité :", this.inactivite, "s");
  //Si l'utilisateur est inactif depuis 5min, on arrête automatiquement le tracking
  if (this.inactivite >= 300){
      this.stopTracking();
      return;
  }
  else{
      this.recupererPosition();
  }
}, this.intervalle * 1000);
```

Par la suite, j'ai rajouté un message qui s'affiche lorsque le trajet s'arrête à cause de l'inactivité de l'utilisateur.

4.4.3.12.2 Détecter l'inactivité v2

Par la suite, je me suis rendu compte que la variable d'inactivité n'était pas la meilleure option pour déterminer le temps d'inactivité de l'utilisateur. En raison du manque de synchronisation entre les différentes opérations réalisées par le programme, l'inactivité n'est pas vérifiée en temps réel, ainsi si je mets que le trajet doit d'arrêter au bout de 15 secondes d'inactivité, le temps que la variable "inactivite" atteigne 15 et que la condition arrêtant le programme soit exécutée, il se passe généralement 40s.

C'est pourquoi j'ai mis en place une autre façon de déterminer l'inactivité de l'utilisateur : un autre timer. Je commence par créer 2 nouvelles variables :

```
//Variables du timer d'inactivité
inactivityTimer: any = null; //id du timer lancé
inactivityTimeOut = 20 * 1000;
```

"inactivityTimer" sert à générer le timer d'inactivité et "inactivityTimeOut" représente le temps au bout duquel le programme doit s'arrêter suite à l'inactivité de l'utilisateur (j'ai mis 20s dans l'exemple).

J'ai ensuite créé une méthode resetInactivityTimer() qui me permet de réinitialiser le timer d'inactivité et d'arrêter le programme au bout du temps de la variable "inactivityTimer" (ici 20s) :

```
private resetInactivityTimer(): void {
  //On reset le timer
  if (this.inactivityTimer) {
    clearTimeout(this.inactivityTimer);
  }

  //On lance le timer qui s'arrête au bout d'une durée prédéfinie dans la variable "inactivityTimeOut"
  this.inactivityTimer = setTimeout(() => {
    this.messageInactivity = "Suite à votre inactivité, le trajet s'est arrêté";
    this.arreterSession();
  }, this.inactivityTimeOut);
}
```

Ensuite, dans demarrerSession(), je rajoute l'appel de cette fonction afin de détecter l'inactivité de l'utilisateur dès le début.

Je vais ensuite dans recupererPosition() et je modifie mon ancienne condition qui vérifiait si la distance parcourue entre les 2 dernières positions était inférieure au seuil d'inactivité, et qui, dans le cas contraire, réinitialisait la variable. Maintenant je ne mets que la condition où je vérifie si la distance est supérieure au seuil, et si elle l'est, je reset mon timer, car cela signifie que l'utilisateur est actif.

```
//On regarde si l'utilisateur a été inactif
if (distance > (this.intervalle - (this.intervalle / 5))){
  //Si l'utilisateur est actif, on reset le timer d'inactivité
  this.resetInactivityTimer();
}
```

Et enfin, à la fin de arreterSession(), je rajoute cette condition qui arrête correctement le timer d'inactivité :

```
//On reset le timer d'inactivité
if (this.inactivityTimer) {
  clearTimeout(this.inactivityTimer);
  this.inactivityTimer = null;
}
```

Désormais, le processus de tracking s'arrête en temps réel si l'utilisateur atteint la durée d'inactivité maximale.

Je vais maintenant faire en sorte que l'inactivité de l'utilisateur soit prise en compte dans le calcul de la vitesse moyenne, afin que celle-ci ne soit pas faussée.

Pour cela, il me suffit de modifier la ligne correspondant au calcul de la durée en heures du trajet, et de remplacer la variable utilisée précédemment par la nouvelle variable représentant le temps maximal d'inactivité de l'utilisateur :

```
//Calcul des infos indépendantes du choix du mode de l'utilisateur
this.dateFin = new Date();
const dureeHeures = ((this.dateFin.getTime() - this.dateDebut.getTime()) - this.inactivityTimeOut) / (1000 * 3600);

this.vitesseMoy = Math.round(((this.distanceTotale) / dureeHeures) * 100) / 100; //2 chiffres après la virgule
```

4.4.3.13 Connexion HTTPS sécurisée

Avec la connexion HTTPS que j'ai mis en place auparavant dans le stage, à chaque fois que je vais sur https://localhost:4200, j'ai le message "Votre connexion n'est pas privée" qui s'affiche, et je dois forcer pour accéder à mon application. Cela est dû au fait que le certificat que j'ai créé pour mettre en place la connexion HTTPS est un certificat local non-reconnu par le navigateur.

Je commence par installer mkcert avec la commande :

« winget install FiloSottile.mkcert »

Ensuite, je crée et installe l'autorité locale avec la commande

« mkcert -install »

Ensuite, dans le terminal, je me place bien dans mon projet Angular et tape la commande

« mkcert localhost 127.0.0.1 ::1 »

Elle permet de générer un certificat pour localhost.

Je peux maintenant redémarrer mon navigateur et redémarrer le serveur Angular avec la commande

« ng serve --ssl true --ssl-cert localhost.pem --ssl-key localhost-key.pem »

Pour éviter de devoir sans cesse utiliser cette longue commande, je vais dans le fichier angular.json et rajoute dans "serve" la partie "options" afin de démarrer le serveur directement en HTTPS et qu'il utilise les fichiers créés précédemment comme certificat et clé privée :

```
"serve": {
  "builder": "@angular/build:dev-server",
  "options": {
    "ssl": true,
    "sslCert": "localhost.pem",
    "sslKey": "localhost-key.pem"
  },
  "configurations": {
    "production": {
      "buildTarget": "TrajetEtCalories:build:production"
    },
    "development": {
      "buildTarget": "TrajetEtCalories:build:development"
    }
  },
  "defaultConfiguration": "development"
},
"test": {
  "builder": "@angular/build:unit-test"
}
```

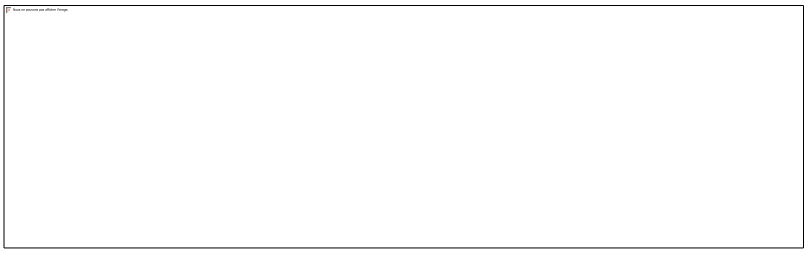
Maintenant, en tapant la commande “ng serve”, j’arrive sur la page d’authentification de mon application avec une connexion sécurisée et reconnue par le navigateur :

Voir Annexe 13

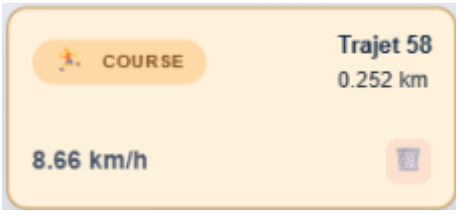
4.4.3.14 Suppression d’un trajet

L’objectif de cette étape est de pouvoir supprimer n’importe quel trajet lorsqu’on se trouve sur la liste des trajets.

J’ai commencé par ajouter un symbole de corbeille cliquable en bas à droite de chaque bouton de trajet en rajoutant du code dans liste-trajet.html juste après l’affichage de la vitesse moyenne du trajet :



Voici le rendu d’un trajet avec le bouton de suppression ajouté (corbeille) :



Au clic de cette corbeille, j’appelle la fonction deleteTrajet(), il faut maintenant la créer :

```
//Fonction permettant de supprimer un trajet de la liste
deleteTrajet(id: number, event: Event) {

    // Empêche le clic sur le bouton parent
    event.stopPropagation();

    if (!confirm('Supprimer ce trajet ?')) {
      return;
    }

    this.trajetService.deleteTrajet(id).subscribe({
      next: () => {
        this.trajets = this.trajets.filter(
          trajet => trajet.id !== id
        );
        this.cdr.detectChanges();
      },
      error: (err: any) => {
        console.error(err);
        this.cdr.detectChanges();
      }
    });

}
```

Elle appelle une fonction du fichier trajet.service.ts également appelée deleteTrajet() qui fait une requête HTTP vers Symfony via la méthode DELETE afin de supprimer le trajet sélectionné :

Il faut maintenant ajouter une fonction dans TripController sur Symfony afin de supprimer le trajet sélectionné de la base de données. J'appelle une fois de plus cette fonction deleteTrajet() et celle-ci se charge de supprimer le trajet correspondant après avoir supprimé toutes les positions liées à ce trajet juste avant. Ainsi, les informations du *trajet* sélectionné sont supprimées sur les tables Trajet, Positions et PointTrajet.

Voir fonction deleteTrajet (TripController) Annexe 14

4.4.3.15 Tentative de déploiement

Durant les 2 derniers jours de stage, j'ai essayé de déployer mon application en utilisant Microsoft Azure et à l'aide de GitHub. Malheureusement, je n'ai pas réussi, seul mon projet Symfony était déployé mais l'application ne peut pas fonctionner sans base de données et sans le projet Angular.

5. Conclusion

5.1 Bilan du projet

Je suis très content d'avoir pu réaliser ce projet en compagnie de mon camarade Romain CANIAUX, celui-ci m'a permis de découvrir et d'apprendre à utiliser de nouvelles technologies de développement : Angular, Symfony, Leaflet ainsi que la notion d'API.

Ce projet m'a permis de me rendre mieux compte de ce à quoi ressemble un projet concret dans la vie d'un développeur. Celui-ci m'a fait acquérir une grande rigueur avec l'utilisation de ces technologies.

Ce projet m'a néanmoins confronté à de nombreuses difficultés avec l'utilisation de ces technologies qui m'étaient totalement inconnues et avec la complexité de certaines tâches. Mais malgré un projet dont l'issue me semblait inatteignable au début du stage, j'ai réussi à mettre en place toutes les fonctionnalités principales du cahier des charges afin d'atteindre l'objectif visé.

5.2 Bilan général du stage

La découverte de toutes ces nouvelles technologies m'a permis d'élargir mes compétences, notamment grâce à l'apprentissage du framework Angular et du framework Symfony en tant qu'API. De plus, l'utilisation de la bibliothèque Leaflet m'a aussi permis de mieux comprendre le fonctionnement de la géolocalisation et de son utilisation dans une application web.

Les nombreuses difficultés rencontrées m’ont aussi permis de développer mon autonomie dans l’apprentissage de nouvelles technologies et dans l’utilisation de l’IA au cours du développement du projet.

Pour conclure, ce stage m'a permis d'acquérir des connaissances et des compétences essentielles dans le développement informatique, mais également de gagner en autonomie, en rigueur et en capacité d'adaptation face à de nouvelles problématiques techniques.

6. Annexes

Annexe 1 (Entité User)

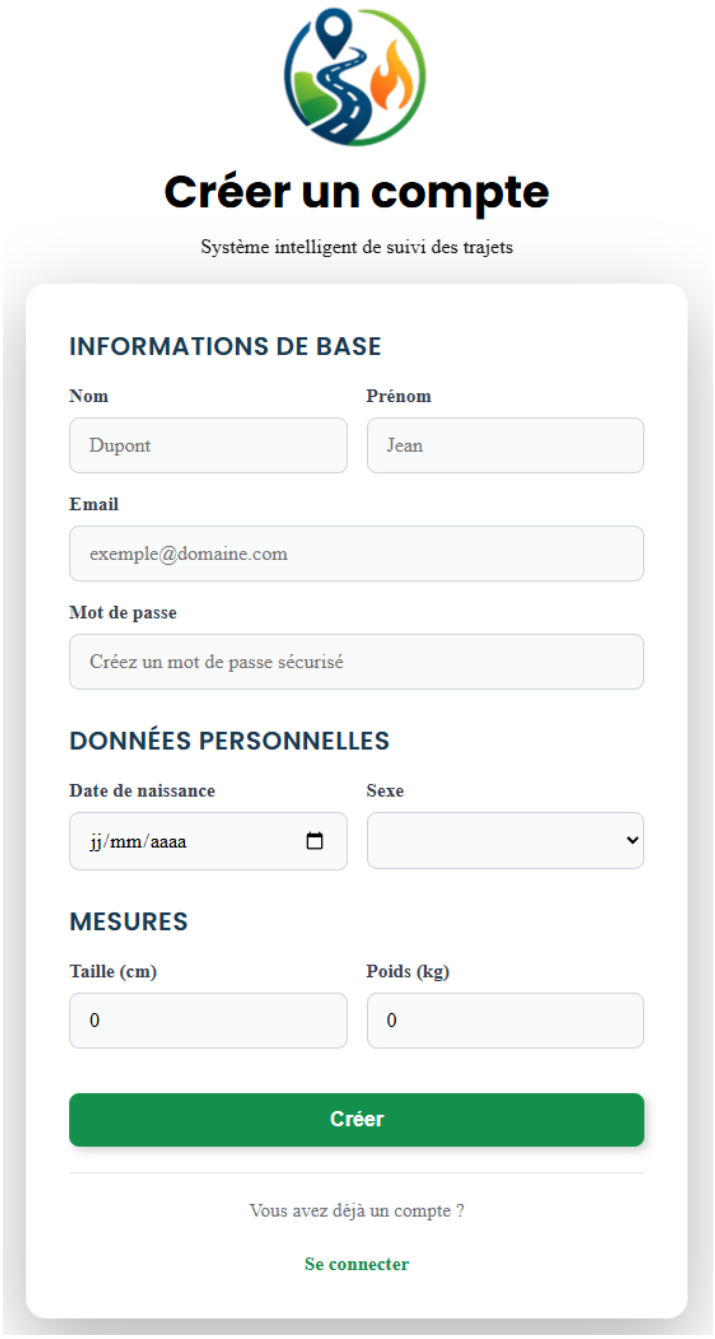
```php
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\Table(name: "users")]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $email = null;

    #[ORM\Column(length: 255)]
    private string $password = '';

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $nom = null;

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $prenom = null;

    #[ORM\Column(type: 'date_immutable', nullable: true)]
    private ?\DateTimeImmutable $naissanceDate = null;

    #[ORM\Column(type: 'integer', nullable: true)]
    private ?int $tailleCm = null;

    #[ORM\Column(nullable: true)]
    private ?bool $sexe = null;

    #[ORM\Column(type: 'date_immutable', nullable: true)]
    private ?\DateTimeImmutable $creeLe = null;

    #[ORM\Column(type: 'float', nullable: true)]
    private ?float $poidsKg = null;
```

```php
// GETTERS / SETTERS

public function getId(): ?int { return $this->id; }

public function getEmail(): ?string { return $this->email; }
public function setEmail(?string $email): static { $this->email = $email; return $this; }

public function getPassword(): string { return $this->password; }
public function setPassword(string $password): static { $this->password = $password; return $this; }

public function getNom(): ?string { return $this->nom; }
public function setNom(?string $nom): static { $this->nom = $nom; return $this; }

public function getPrenom(): ?string { return $this->prenom; }
public function setPrenom(?string $prenom): static { $this->prenom = $prenom; return $this; }

public function getNaissanceDate(): ?\DateTimeImmutable { return $this->naissanceDate; }
public function setNaissanceDate(?\DateTimeImmutable $naissanceDate): static { $this->naissanceDate = $naissanceDate; return $this; }

public function getTailleCm(): ?int { return $this->tailleCm; }
public function setTailleCm(?int $tailleCm): static { $this->tailleCm = $tailleCm; return $this; }

public function isSexe(): ?bool { return $this->sexe; }
public function setSexe(?bool $sexe): static { $this->sexe = $sexe; return $this; }

public function getCreeLe(): ?\DateTimeImmutable { return $this->creeLe; }
public function setCreeLe(?\DateTimeImmutable $creeLe): static { $this->creeLe = $creeLe; return $this; }

public function getPoidsKg(): ?float { return $this->poidsKg; }
public function setPoidsKg(?float $poidsKg): static { $this->poidsKg = $poidsKg; return $this; }

//Copilot
public function getRoles(): array
{
    return ['ROLE_USER'];
}

public function getUserIdentifier(): string
{
    return (string) $this->email;
}

public function eraseCredentials(): void
{
    // If you store any temporary, sensitive data on the user, clear it here
}
```

Annexe 2 (fonction authenticate - Symfony)

```php
#[Route('/api/login', name: 'api_login')]
public function authenticate(Request $request, UserRepository $userRepository, UserPasswordHasherInterface $passwordHasher, JWTTokenManagerInterface $jwtManager): JsonR
{
    $data = json_decode($request->getContent(), true); //On récupère les données dans un tableau associatif

    //Vérification du format
    if (!is_array($data)) {
        return $this->json([
            'status' => 'error',
            'message' => 'Le corps de la requête doit être un JSON valide.',
        ], Response::HTTP_BAD_REQUEST);
    }

    //On extrait les informations d'authentifications
    $email = trim((string) ($data['email'] ?? ''));
    $password = trim((string) ($data['password'] ?? ''));

    //Vérification que les champs ne sont pas vides
    if ($email === '' || $password === '') {
        return $this->json([
            'status' => 'error',
            'message' => 'Les champs email et password sont obligatoires.',
        ], Response::HTTP_BAD_REQUEST);
    }

    //Vérification que l'utilisateur existe dans la BDD
    $user = $userRepository->findOneBy(['email' => $email]);
    if ($user === null) {
        return $this->json([
            'status' => 'error',
            'message' => 'Les identifiants sont incorrects.',
        ], Response::HTTP_UNAUTHORIZED);
    }

    //Vérification que le mot de passe est correct
    if (!$passwordHasher->isPasswordValid($user, $password)) {
        return $this->json([
            'status' => 'error',
            'message' => 'Les identifiants sont incorrects.',
        ], Response::HTTP_UNAUTHORIZED);
    }

    //Si on arrive là, c'est que tout est ok
    //On génère un JWT via Lexik
    $jwt = $jwtManager->create($user);

    return $this->json([
        'status' => 'success',
        'message' => 'Authentification réussie',
        'token' => $jwt,
    ], Response::HTTP_OK);
}
```

Annexe 3 (Interface page Créer un compte)



Annexe 4 (fonction register – Symfony)

```php
#[Route('/api/register', name: 'app_register')]
public function register(
    Request $request,
    EntityManagerInterface $entityManager,
    ValidatorInterface $validator,
    UserPasswordHasherInterface $passwordHasher
): JsonResponse
{
    //json_decode transforme une chaine JSON en structure PHP
    $data = json_decode($request->getContent(), true);

    //On vérifie que le JSON est valide
    if (json_last_error() !== JSON_ERROR_NONE) {
        return new JsonResponse([
            'message' => 'JSON invalide'
        ], Response::HTTP_BAD_REQUEST);
    }

    //On crée un nouvel utilisateur
    $user = new User();

    $user->setNom($data['nom']) ?? '';
    $user->setPrenom($data['prenom']) ?? '';

    $user->setEmail($data['email']);

    //Hashage du mdp
    $hashedPassword = $passwordHasher->hashPassword($user, $data['password']);
    $user->setPassword($hashedPassword);
    $birthdate = $data['birthdate'] ?? null;
    if ($birthdate !== null && $birthdate !== '') {
        try {
            // Normalize to date-only; DateTimeImmutable will accept 'YYYY-MM-DD' or other formats
            $user->setNaissanceDate(new \DateTimeImmutable($birthdate));
        } catch (\Exception $e) {
            return new JsonResponse([
                'message' => 'Date de naissance invalide'
            ], Response::HTTP_BAD_REQUEST);
        }
    } else {
        $user->setNaissanceDate(null);
    }
    $user->setTailleCm($data['height']);
    $user->setSexe($data['sex']);
    $user->setCreeLe(new \DateTimeImmutable('today'));
    $user->setPoidsKg($data['weight']);
```

```php
    //On vérifie que chacune des données sont valides et correspondent bien aux normes
    //pour ne pas avoir des trucs absurdes
    $errors = $validator->validate($user);
    if (count($errors) > 0) {
        $formattedErrors = [];
        foreach ($errors as $error) {
            $formattedErrors[] = [
                'property' => $error->getPropertyPath(),
                'message' => $error->getMessage(),
            ];
        }
        return new JsonResponse([
            'message' => 'Données invalides',
            'errors' => $formattedErrors
        ], Response::HTTP_UNPROCESSABLE_ENTITY); //Erreur 422 (données valides mais incorrectes pour la BDD)
    }

    //On vérifie que l'utilisateur n'existe pas déjà
    $existingUser = $entityManager
        ->getRepository(User::class)
        ->findOneBy(['email' => $data['email']]);

    if ($existingUser) {
        return new JsonResponse([
            'message' => 'Cet email existe déjà'
        ], Response::HTTP_CONFLICT);
    }

    //Si on arrive là, c'est que normalement tout est ok
    $entityManager->persist($user);
    $entityManager->flush();

    //On envoie une réponse témoignant du succès de la requête
    return new JsonResponse([
        'message' => 'Utilisateur créé avec succès'
    ], Response::HTTP_CREATED);

}
```

Annexe 5 (fonction updateProfile() – Symfony)

```php
#[Route('/api/me', name: 'update_profile', methods: ['PUT'])]
public function updateProfile(Request $request, EntityManagerInterface $entityManager): JsonResponse{
    $data = json_decode($request->getContent(), true);

    $user = $this->getUser();
    //Vérifications liées à l'utilisateur
    if (!$user) {
        return $this->json(['error' => 'Utilisateur non authentifié'], Response::HTTP_UNAUTHORIZED);
    }
    if (!is_array($data)) {
        return $this->json(['error' => 'Données invalides'], Response::HTTP_BAD_REQUEST);
    }

    //On récupère les valeurs
    $nom = $data['nom'];
    $prenom = $data['prenom'];
    $email = $data['email'];
    $birthdate = $data['birthdate'];
    $height = array_key_exists('height', $data) ? (int) $data['height'] : 0;
    $sexRaw = $data['sex'] ?? true;
    $weight = array_key_exists('weight', $data) ? (float) $data['weight'] : 0;

    //On met à jour le nom et prénom
    $user->setNom($nom);
    $user->setPrenom($prenom);

    //Vérification de l'email
    if (!$email || trim($email) === '') {
        return $this->json([
            'error' => 'Email obligatoire'
        ], Response::HTTP_BAD_REQUEST);
    }
    $user->setEmail($email);

    //Vérification de la date de naissance
    if ($birthdate) {
        try {
            $dt = new \DateTimeImmutable($birthdate);
            $user->setNaissanceDate($dt);
        } catch (\Exception $e) {
            return $this->json(['error' => 'Date invalide'], Response::HTTP_BAD_REQUEST);
        }
    } else {
        $user->setNaissanceDate(null);
    }

    $user->setTailleCm($height);

    //Vérification du genre
    $sex = null;
    if ($sexRaw !== null) {
        if (is_bool($sexRaw)) {
            $sex = $sexRaw;
        } else {
            $sr = strtolower((string) $sexRaw);
            $sex = in_array($sr, ['1', 'true', 'yes', 't', 'male', 'm', 'homme', 'h']);
        }
    }
    $user->setSexe($sex);

    $user->setPoidsKg($weight);

    $entityManager->flush();

    return $this->json([
        'message' => 'Profil mis à jour avec succès'
    ], Response::HTTP_OK);
}
```

Annexe 6 (fonction ngOnInit() du composant Parcours)

```
ngOnInit(){
  //! On récupère les informations du dernier trajet
  this.infosParcours = new FormGroup({
    duree: new FormControl('', Validators.required),
    distance: new FormControl('', Validators.required),
    vitesseMoy: new FormControl('', Validators.required),
    typeActivite: new FormControl('', Validators.required),
    caloriesDepensees: new FormControl('', Validators.required),
  });

  const token = localStorage.getItem('token');
  if (!token) {
      console.error('Token non trouvé');
      this.router.navigate(['/login']);
      return;
  }

  this.parcoursService.getInfos().subscribe({
    next: (res: any) => {
      this.data = res;
      console.log(this.data);
      this.infosParcours.patchValue(this.data);
      this.idTrajet = res.id;
      console.log("Dernier trajet :", this.idTrajet);
      this.cdr.detectChanges();
    },
    error: (err) => {
      console.error('Erreur lors de la récupération des informations', err);
      this.message = "Pas encore de trajets réalisés.";
      this.cdr.detectChanges();
    }
  //! Fin de la récupération des informations du dernier trajet
  });

}
```

Annexe 7 (Utilisation fonctions évaluant état des boutons)

```
<div class="modeBlock">
    <label for="modeSelect">Choisissez un mode :</label>
    <select id="modeSelect" class="modeSelect"
        [(ngModel)]="mode"
        [disabled]="sessionStarted">
        <option value="simple" selected>Mode Simple</option>
        <option value="automatique">Mode Automatique</option>
    </select>
</div>
<div class="activityButtons">

    <button type="button" class="activityButton"
        [disabled]="activityDisabled"
        [class.active]="activity === 'marche'"
        (click)="selectActivity(activity: 'marche')">Marche</button>

    <button type="button" class="activityButton"
        [disabled]="activityDisabled"
        [class.active]="activity === 'course'"
        (click)="selectActivity(activity: 'course')">Course</button>

    <button type="button" class="activityButton"
        [disabled]="activityDisabled"
        [class.active]="activity === 'velo'"
        (click)="selectActivity(activity: 'velo')">Vélo</button>

</div>
```

Annexe 8 (fonction recupererPosition())

```
//* Méthode qui récupère la position de l'utilisateur
private recupererPosition(): void {
  navigator.geolocation.getCurrentPosition(
    (position) => {

      //On récupère les données
      const nouveauPoint = {
        lat: position.coords.latitude,
        lng: position.coords.longitude,
        enregistreA: new Date(),
        source: 'GPS',
        //trajetId: this.trajetId //Pour récupérer l'id du trajet pour enregistrer dans Point_trajet
      };

      // Calcul de la distance avec le point précédent
      if (this.listePoints.length > 0) {
        const dernierPoint = this.listePoints[this.listePoints.length - 1];
        //On calcule la distance entre les 2 derniers points
        const distance = this.calculDistance(
          dernierPoint.lat,
          dernierPoint.lng,
          nouveauPoint.lat,
          nouveauPoint.lng
        );

        //On regarde si l'utilisateur a été inactif
        if (distance > (this.intervalle - (this.intervalle / 5))){
          //Si l'utilisateur est actif, on reset le timer d'inactivité
          this.resetInactivityTimer();
        }

        this.distanceTotale += distance / 1000;
        console.log(`Distance segment : ${distance.toFixed()} m`);
        console.log(`Distance totale : ${this.distanceTotale.toFixed(3)} km`);
      }

      this.listePoints.push(nouveauPoint); //On ajoute la position à la liste
      this.nbPositions++; //On incrémente le compteur de positions

      //On envoie les positions regroupées dans la requête si elles sont regroupées par 3
      if (this.listePoints.length >= 3) {
        console.log("3 positions dans la liste");

        //On crée une copie du tableau pour pouvoir le réinitialiser directement après, pour pallier les problèmes de synchronisation
        const batch = [...this.listePoints];
        this.listePoints = [];

        //On envoie les positions
        this.trackingService.envoyerPositions(batch, this.trajetId!).subscribe({
            next: () => {
              console.log('Batch de positions envoyé en BDD', batch);
            },
            error: (err: any) => {
              console.error('Erreur envoi batch', err);
            }
        });
      }
    },
    (error) => {
      console.error('Erreur GPS', error);
    },
    {
      enableHighAccuracy: true,
      maximumAge: 0,
      timeout: 10000
    }
  );
}
```

Annexe 9 (fonction addPositions - PositionController – Symfony)

```php
#[Route('/api/positions', name: 'app_positions', methods: ['POST'])]
public function addPositions(
    Request $request,
    EntityManagerInterface $entityManager,
): JsonResponse
{
    $data = json_decode($request->getContent(), true);

    if (!isset($data['trajetId'])) {
        return $this->json([
            'error' => 'Missing trajetId'
        ], 400);
    }
    //On vérifie maintenant si le trajet existe bien dans la base de données
    $trajet = $entityManager->getRepository(Trajet::class)->find($data['trajetId']);

    if (!$trajet) {
        return $this->json([
            'error' => 'Trajet not found'
        ], Response::HTTP_NOT_FOUND);
    }
    //On vérifie la validité de ce qui a été envoyé
    if (!isset($data["positions"]) || !is_array($data["positions"])) {
        return $this->json([
            'error' => 'Missing positions array'
        ], Response::HTTP_BAD_REQUEST);
    }

    //On vérifie l'utilisateur
    $user = $this->getUser();
    if (!$user) {
        return $this->json(
            ['error' => 'Unauthorized'],
            Response::HTTP_UNAUTHORIZED
        );
    }

    foreach ($data["positions"] as $p) {

        if (!isset($p["lat"]) || !isset($p["lng"])) {
            return $this->json([
                'error' => 'Missing informations'
            ], Response::HTTP_BAD_REQUEST);
        }

        $date = isset($p["enregistreA"]) ? new \DateTimeImmutable($p["enregistreA"]) : new \DateTimeImmutable();

        $position = new Positions();
        $position->setLatitude((string) $p["lat"]);
        $position->setLongitude((string) $p["lng"]);
        $position->setEnregistreA($date);
        $position->setSource($p["source"] ?? 'GPS');
        $position->setUser($user);

        //Persist écrit les requêtes sans les exécuter
        $entityManager->persist($position);

        //Maintenant qu'on a rajouté une position, il faut aussi compléter la table Point_Trajet
        $pointTrajet = new PointTrajet();
        $pointTrajet->setTrajet($trajet);
        $pointTrajet->setPosition($position);

        $entityManager->persist($pointTrajet);

    }

    //Flush exécute les requêtes
    $entityManager->flush();

    return $this->json([
        'status' => 'ok',
        'count' => count($data["positions"])
    ], Response::HTTP_CREATED);
}
```

Annexe 10 (fonction createTrajet – TripController – Symfony)

```php
#[Route('/api/trips', name: 'api_trip', methods:['POST'])]
public function createTrajet(Request $request, EntityManagerInterface $em): JsonResponse
{
    $data = json_decode($request->getContent(), true);

    if (!isset($data['dateDebut'])) {
        return $this->json([
            'error' => 'Données manquantes'
        ], Response::HTTP_BAD_REQUEST);
    }

    $user = $this->getUser();

    if (!$user) {
        return $this->json([
            'error' => 'Utilisateur non authentifié'
        ], Response::HTTP_UNAUTHORIZED);
    }

    $trajet = new Trajet();
    //On crée les infos du trajet
    $trajet->setIdUser($user);
    $trajet->setDebutTime(new \DateTime($data['dateDebut']));

    $em->persist($trajet);
    $em->flush();

    return $this->json([
        'id' => $trajet->getId()
    ]);
}
```

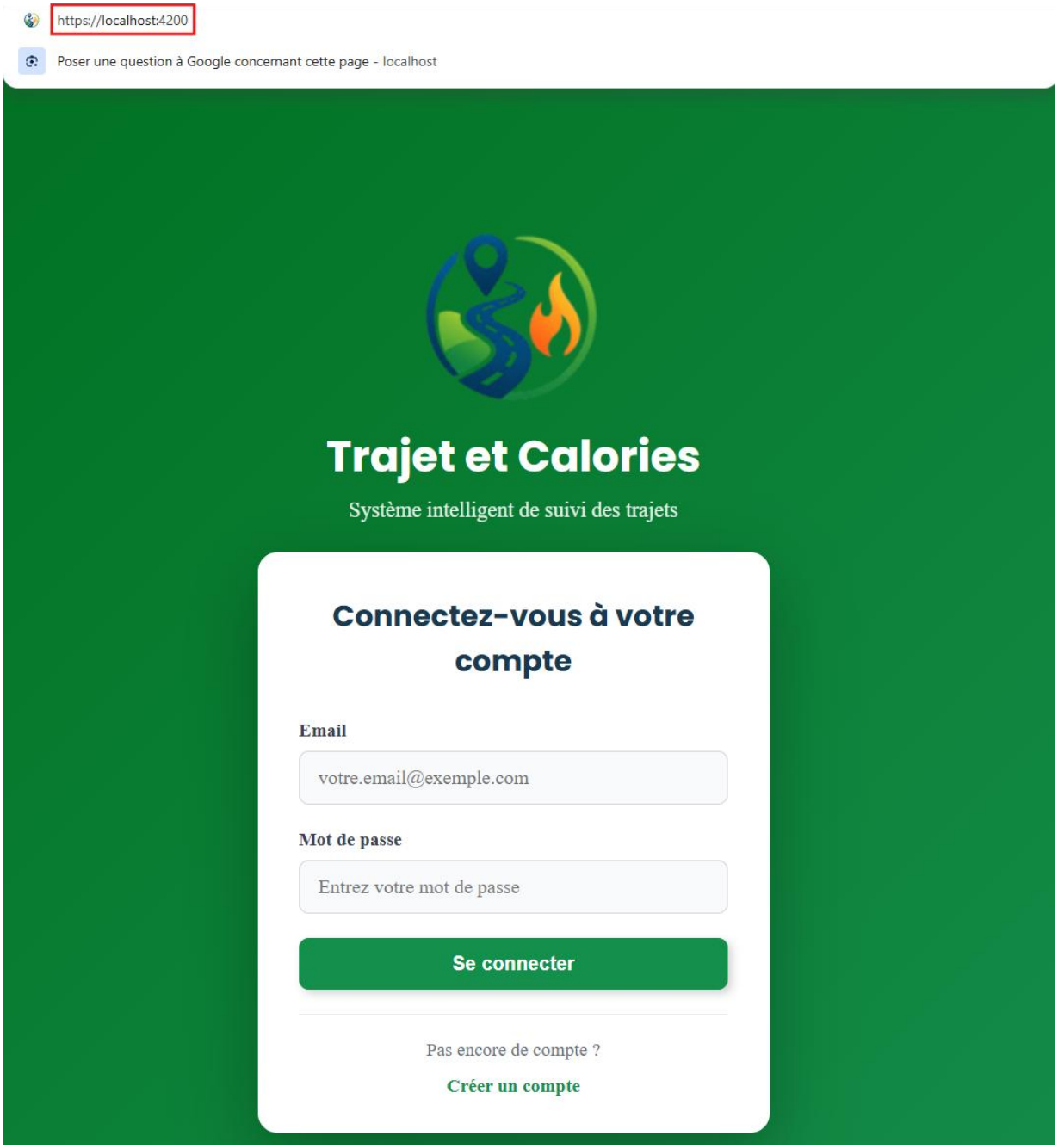
Annexe 11 (fontion getTrips() – TripController – Symfony)

```php
#[Route('/api/trips/all', name: 'api_trip_all', methods: ['GET'])]
public function getTrips(EntityManagerInterface $entityManager): JsonResponse
{
    $user = $this->getUser();
    if (!$user) {
        return $this->json([
            'error' => 'Utilisateur non authentifié'
        ], Response::HTTP_UNAUTHORIZED);
    }

    $trajets = $entityManager->getRepository(Trajet::class)
        ->findBy(['idUser' => $user], ['id' => 'DESC']);

    if (!$trajets) {
        return $this->json([
            'error' => 'Aucun trajet trouvé'
        ], Response::HTTP_NOT_FOUND);
    }

    $data = array_map(function (Trajet $trajet) {
        return [
            'id' => $trajet->getId(),
            'dateDebut' => $trajet->getDebutTime(),
            'dateFin' => $trajet->getFinTime(),
            'distance' => $trajet->getDistanceKm(),
            'vitesseMoy' => $trajet->getAvgSpeedKmh(),
            'activite' => $trajet->getActivityType(),
            'calories' => $trajet->getCaloriesKcal()
        ];
    }, $trajets);

    return $this->json([
        'trips' => $data
    ], Response::HTTP_OK);
}
```

Annexe 12 (fonction getActivityIcon())

```
//Fonction permettant de mettre l'émoji correspondant à l'activité exercée par l'utilisateur
getActivityIcon(activite: string | null | undefined) {
  const type = (activite || '').toLowerCase();
  if (type === 'marche') {
    return '🚶';
  }
  if (type === 'course') {
    return '🏃';
  }
  if (type === 'vélo' || type === 'velo') {
    return '🚴';
  }
  return '⭐';
}
```

Annexe 13 (Page d'authentification sécurisée)



Annexe 14 (fonction deleteTrajet() – TripController – Symfony)

```php
#[Route('/api/trips/{id}/delete', name: 'api_trip_delete', methods: ['DELETE'])]
public function deleteTrajet(int $id, EntityManagerInterface $entityManager, TrajetRepository $trajetRepository): JsonResponse
{
    $user = $this->getUser();
    if (!$user) {
        return $this->json([
            'error' => 'Utilisateur non authentifié'
        ], Response::HTTP_UNAUTHORIZED);
    }

    $trajet = $trajetRepository->find($id);

    if (!$trajet) {
        return new JsonResponse(['error' => 'Trajet not found'], Response::HTTP_NOT_FOUND);
    }

    //On parcorut chaque ligne de PointTrajet correspondant au bon trajet
    //Puis on supprime chaque position correspondante dans la table Positions
    foreach ($trajet->getPointTrajets() as $pointTrajet) {
        $position = $pointTrajet->getPosition();
        if ($position) {
            $entityManager->remove($position);
        }
        $entityManager->remove($pointTrajet);
    }

    //On supprime également le trajet
    $entityManager->remove($trajet);
    $entityManager->flush();

    return new JsonResponse([
        'message' => 'Trajet supprimé avec succès'
    ], Response::HTTP_OK);
}
```