CAMPUS
CARLO ACUTIS

BTS SIO

RAPPORT DE STAGE

FRAMEWORK : ANGULAR ET SYMFONY

Romain CANIAUX

Stage du 18 mai au 26 juin 2026

ALTAMEOS

altameos MULTIMEDIA

Maître de stage : SCHNEKENBURGER Lucas

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué au bon déroulement de mon stage et à la réalisation de ce rapport.

Je remercie tout particulièrement Monsieur SCHNEKENBURGER, chef de l'entreprise ALTAMEOS et mon maître de stage, pour son accueil, sa disponibilité, sa confiance et les précieux conseils qu'il m'a apportés tout au long de cette expérience. Son accompagnement et son professionnalisme m'ont permis de développer mes compétences et de mieux appréhender les réalités du monde professionnel.

Je souhaite également adresser mes sincères remerciements à l'ensemble de mes professeurs pour la qualité de leur enseignement, leur soutien et leur accompagnement durant ma formation. Les connaissances et les méthodes de travail qu'ils m'ont transmises ont été essentielles à la réussite de ce stage.

Je remercie également le Campus Carlo Acutis pour son encadrement, son accompagnement pédagogique et les opportunités offertes tout au long de mon parcours. Grâce à son soutien, j'ai pu effectuer ce stage dans les meilleures conditions et enrichir mon expérience professionnelle.

Enfin, je remercie toutes les personnes qui ont participé, directement ou indirectement, à la réussite de mon stage et à la réalisation de ce rapport.

À tous, j'adresse mes plus sincères remerciements.

Sommaire

Introduction

Dans le cadre de ma formation BTS SIO (Services Informatiques aux Organisations) au Campus Carlo Acutis, j'ai effectué un stage de six semaines du 18 mai au 26 juin 2026 au sein de l'agence web ALTAMEOS, située à Évreux.

Ce stage avait pour objectif de mettre en pratique les compétences acquises durant ma formation, notamment en développement web. Il m'a été confié la réalisation d'une application de suivi d'activité physique reposant sur les frameworks Angular et Symfony, deux technologies modernes que j'avais abordées en cours mais que je n'avais pas encore utilisées dans un contexte professionnel réel.

Ce rapport présente le déroulement de ce stage : l'entreprise d'accueil, le projet réalisé, les solutions techniques mises en œuvre, ainsi qu'un bilan des compétences acquises.

1. Présentation de l'entreprise

1.1 Identité de l'entreprise

Nom de l'entreprise : *ALTAMEOS*

Secteur d'activité : *Agence web et communication digitale (création de sites internet, référencement SEO, développement web et mobile, hébergement, stratégie digitale).*

Statut juridique : *Société indépendante*

Adresse du siège : *9B Rue du Puits Carré, 27700 Evreux*

Site web : *Altameos.com*

Effectif : *1 → Monsieur SCHNEKENBURGER*

1.2 Présentation générale

ALTAMEOS Multimédia est une agence web située à Évreux et fondée en 2003. Dirigée par Monsieur Lucas SCHNEKENBURGER, elle est spécialisée dans la création de sites internet, le référencement SEO, l'hébergement web et la communication digitale.

L'entreprise accompagne ses clients dans le développement de leur présence en ligne en proposant des solutions numériques adaptées à leurs besoins. Grâce à son expertise, ALTAMEOS aide les professionnels à améliorer leur visibilité et leur image sur Internet.

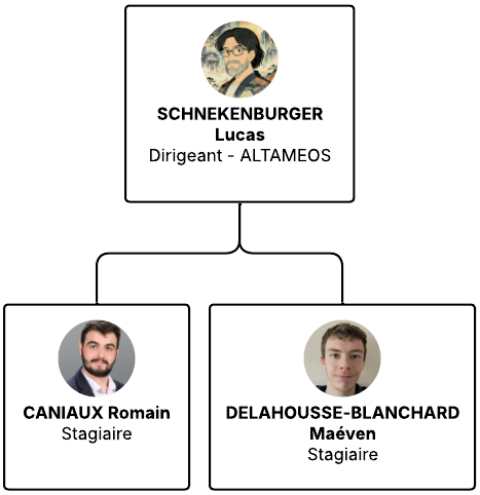
2. Présentation du service d'accueil et des moyens informatiques

2.1 Le service d'accueil

J'ai effectué mon stage au sein d'ALTAMEOS, une agence spécialisée dans le développement web et la communication digitale. L'entreprise accompagne ses clients dans la création de sites internet, le référencement SEO et la mise en place de solutions numériques adaptées à leurs besoins.

Mon maître de stage, Monsieur Lucas SCHNEKENBURGER, dirige l'entreprise et intervient sur l'ensemble des projets. Il assure la gestion de l'activité, le suivi des clients ainsi que le développement des solutions web proposées par l'entreprise.

2.2 Moyens humains

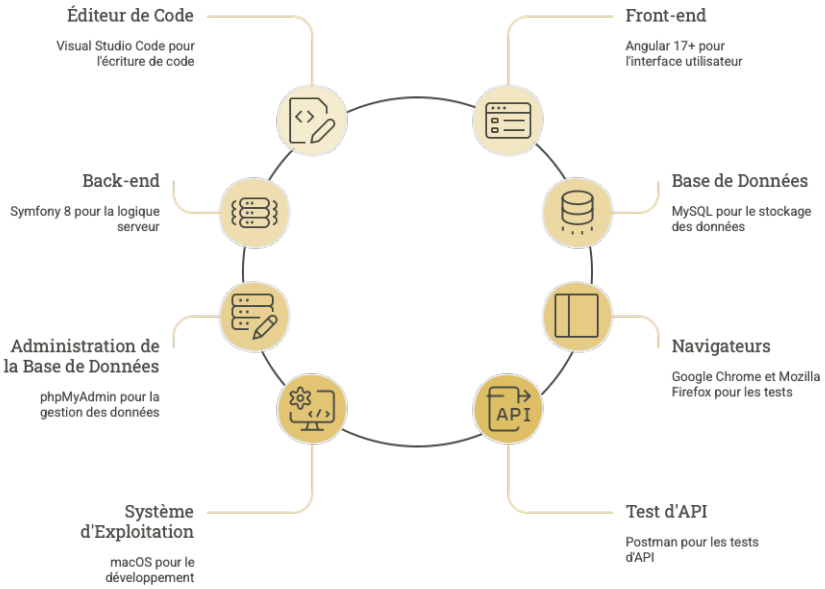


L'entreprise étant une structure unipersonnelle, j'ai travaillé en collaboration directe et exclusive avec mon maître de stage, qui assurait le suivi technique et pédagogique du projet.

2.3 Moyens matériels

Nous devions travailler avec nos ordinateurs personnels car nous étions en distanciel ; afin de travailler dans les meilleures conditions, nous nous rendions au Campus.

2.4 Moyens logiciels



L'environnement de développement utilisé repose sur **Visual Studio Code** comme éditeur de code. La partie front-end est développée avec **Angular**, tandis que la partie back-end s'appuie sur **Symfony 8**. Les données sont stockées dans une base de données **MySQL**, administrée via **phpMyAdmin**. Les tests et l'utilisation de l'application sont réalisés principalement sur les navigateurs. Le développement est effectué sous **macOS**, et l'outil **Postman** est utilisé pour tester et valider les API.

3. Présentation du sujet de stage

3.1 Quoi ? — Description du sujet

Durant mon stage chez ALTAMEOS, j'ai participé au développement d'une application web de suivi d'activité physique reposant sur les frameworks ANGULAR et SYMFONY.

Cette application permet à un utilisateur de collecter automatiquement ses positions GPS depuis un ordinateur ou un appareil mobile afin de reconstituer les trajets effectués. Elle calcule ensuite différentes statistiques telles que la distance parcourue, la vitesse moyenne et les calories dépensées.

L'objectif est de proposer un système simple de suivi d'activité inspiré des applications sportives utilisées pour la marche, la course à pied ou le vélo. Les fonctionnalités attendues sont décrites dans le cahier des charges fourni par l'entreprise.

3.2 Pourquoi ? — Motivations et contexte

L'entreprise souhaitait développer une application permettant d'exploiter les données de géolocalisation afin de fournir des indicateurs d'activité physique à l'utilisateur.

Le projet répond à plusieurs besoins :

- Suivre les déplacements réalisés ;
- Calculer automatiquement les distances parcourues ;
- Déterminer le type d'activité pratiquée ;
- Estimer les calories dépensées ;
- Proposer une interface moderne et accessible depuis un navigateur web.

Ce projet constitue également un excellent exercice technique permettant de mettre en œuvre une architecture moderne basée sur un front-end Angular et une API REST Symfony.

3.3 Pour qui ? — Le demandeur et les bénéficiaires

Le projet a été confié par ALTAMEOS dans le cadre de ses activités de développement web.

Les bénéficiaires sont les utilisateurs souhaitant suivre leur activité physique quotidienne. Grâce à cette application, ils peuvent consulter l'historique de leurs trajets, analyser leurs performances et suivre leur dépense énergétique.

L'application est également utile aux administrateurs qui disposent d'une architecture facilement maintenable et évolutive.

3.4 Cahier des charges / Étude de l'existant

Le cahier des charges définit plusieurs fonctionnalités principales :

Gestion des utilisateurs

- Création de compte ;
- Authentification sécurisée par JWT ;
- Gestion du profil utilisateur ;
- Mise à jour des informations personnelles.

Collecte des données GPS

- Récupération des coordonnées GPS via l'API Geolocation HTML5 ;
- Enregistrement de la latitude, longitude, date et heure ;
- Envoi des données vers l'API Symfony.

Gestion des trajets

- Détection automatique des trajets ;
- Calcul des distances ;
- Calcul des vitesses moyennes ;
- Génération de statistiques détaillées.

Calcul des calories

Le calcul des calories repose sur la méthode MET (Metabolic Equivalent of Task) :

$$Calories = MET \times Poids\ (kg) \times Durée\ (h)$$

Cette formule permet d'estimer la dépense énergétique en fonction du poids de l'utilisateur et du type d'activité pratiquée.

4. Développement du travail réalisé

4.1 Outils utilisés

Angular :

Angular a été utilisé pour développer l'interface utilisateur de l'application.

Les principales pages prévues étaient :

- Connexion / Inscription ;
- Profil utilisateur ;
- Suivi GPS en temps réel ;
- Liste des trajets ;
- Détail d'un trajet avec statistiques.

Angular permet de développer une application SPA (Single Page Application) offrant une navigation fluide et une meilleure expérience utilisateur.

Symfony :

Symfony a été utilisé pour développer l'API REST.

L'API gère :

- L’authentification ;
- Les utilisateurs ;
- Les positions GPS ;
- Les trajets ;
- Les calculs associés.

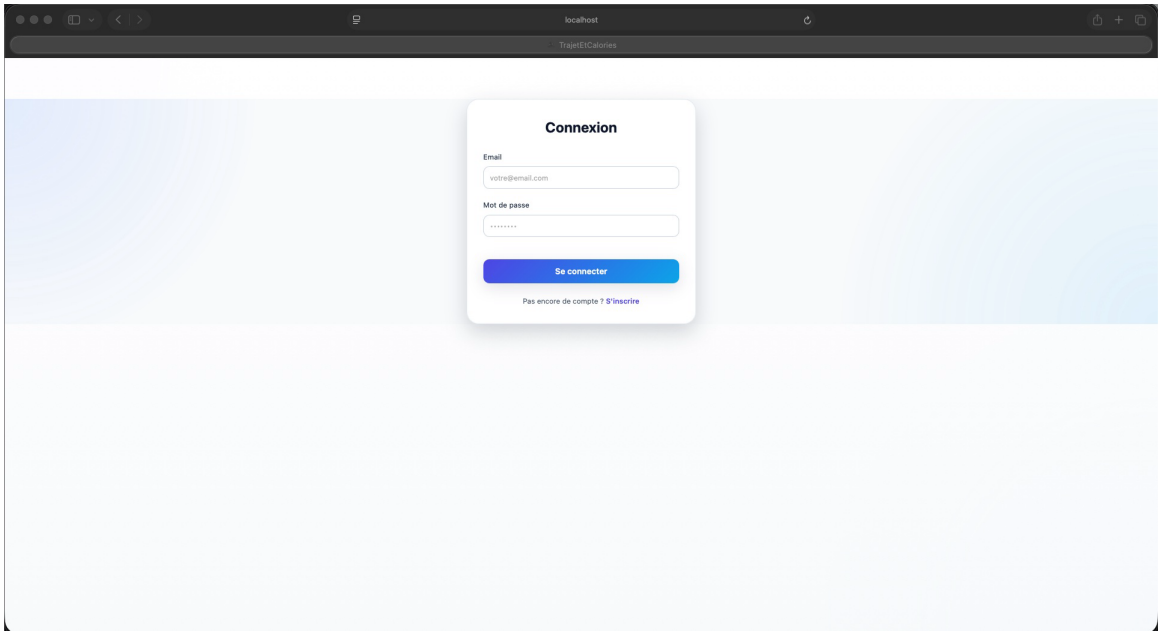
4.2 Interface utilisateur

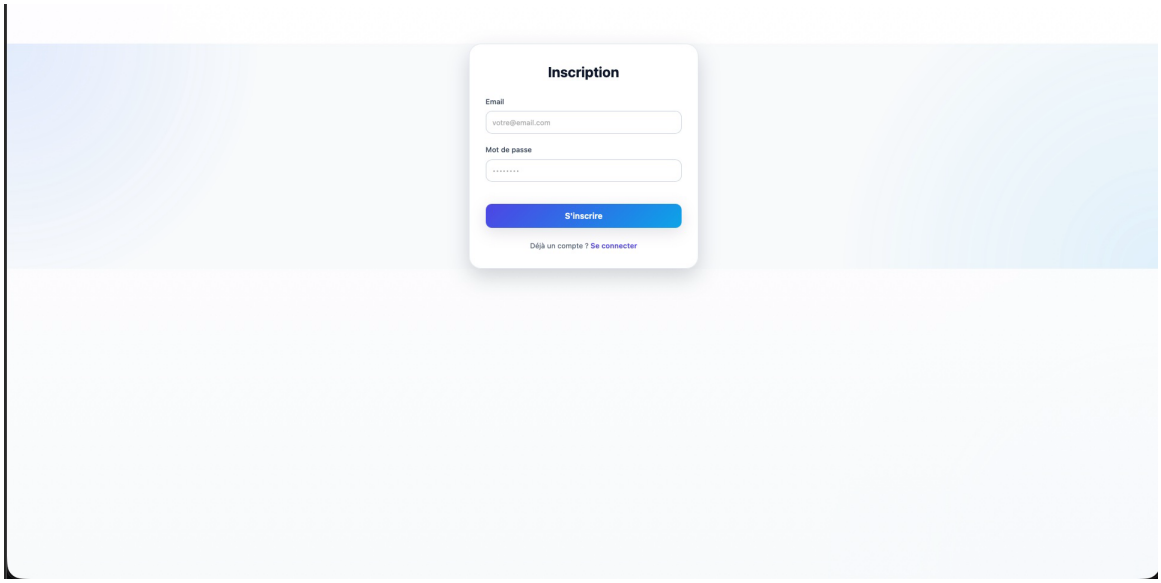
Une interface a été développée afin de permettre aux utilisateurs de renseigner leurs informations personnelles :

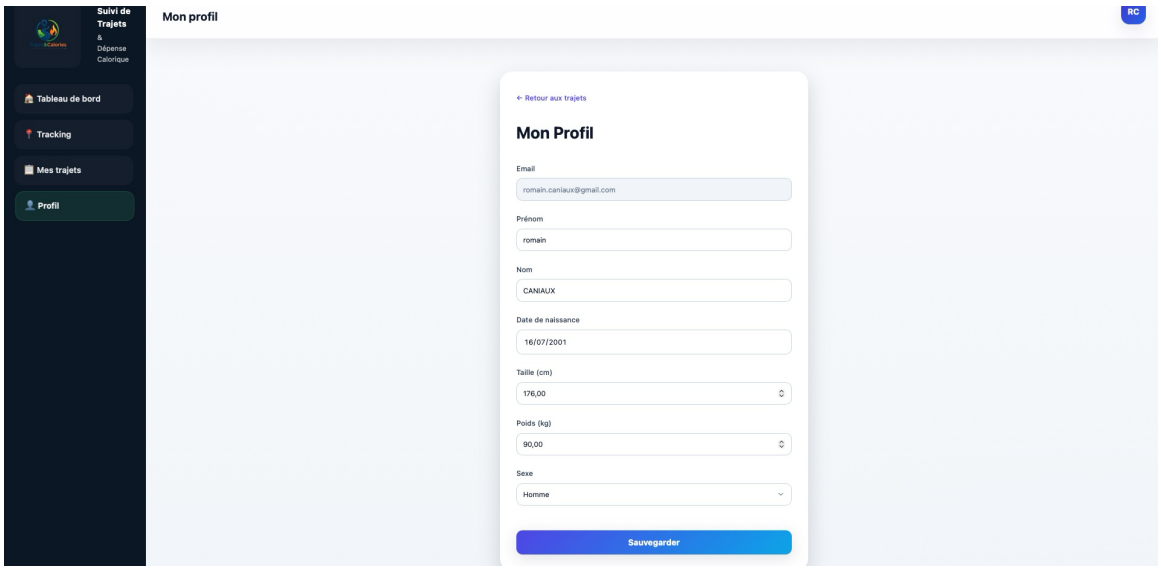
- Prénom ;
- Nom ;
- Date de naissance ;
- Poids ;
- Taille ;
- Sexe ;

Ces informations sont nécessaires pour le calcul des calories dépensées.

Les captures d’écran suivantes illustrent les différentes pages de l’interface développée avec Angular :

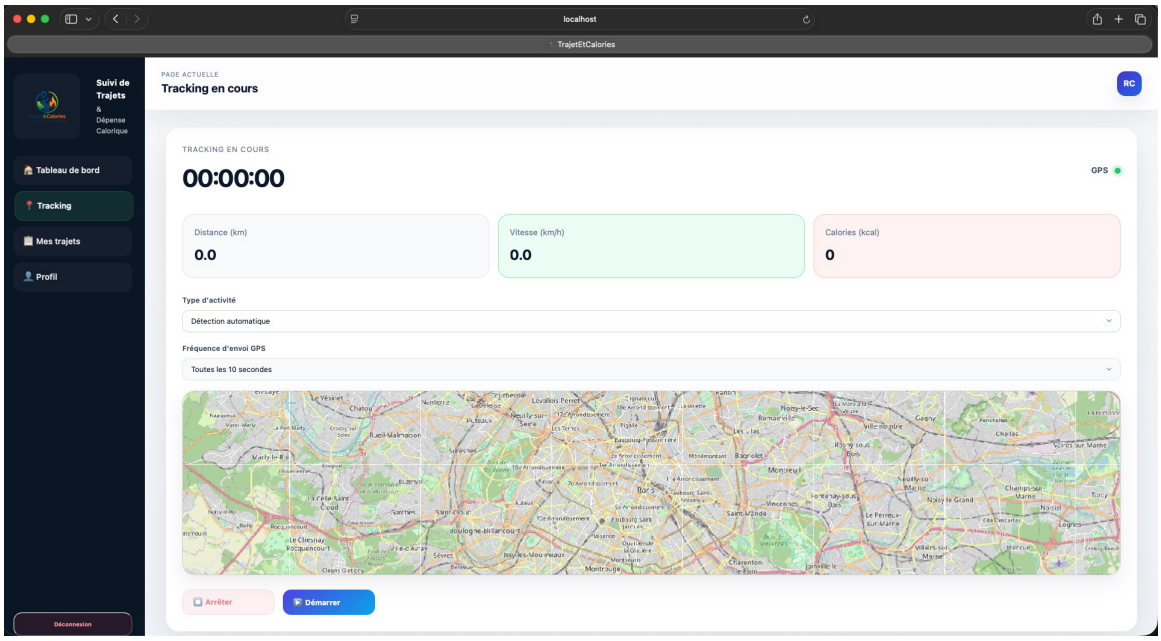

localhost
TrajetEtCalories
Connexion
Email
votre@email.com
Mot de passe
Se connecter
Pas encore de compte ? S'inscrire

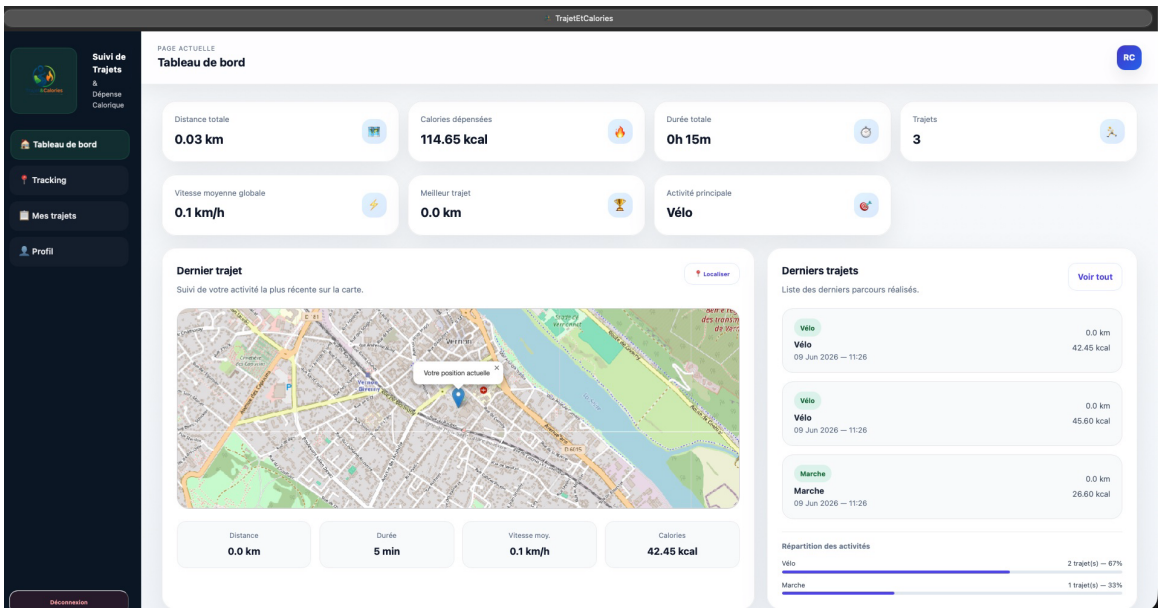

Inscription
Email
votre@email.com
Mot de passe
S'inscrire
Déjà un compte ? Se connecter

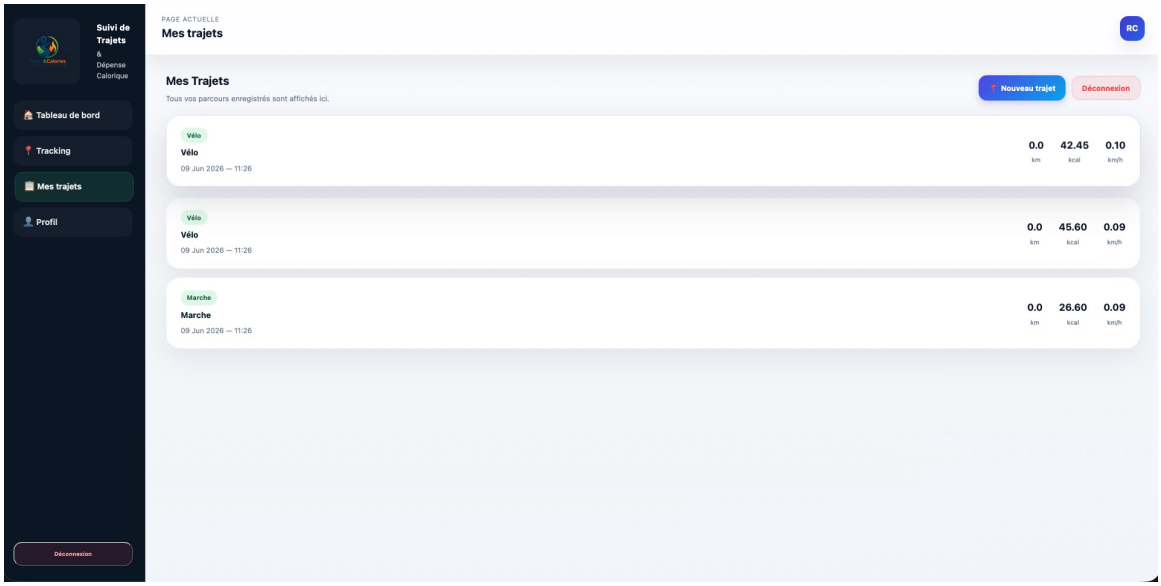

Suivi de Trajets & Dépense Calorique
Tableau de bord
Tracking
Mes trajets
Profil
Mon profil
RC
← Retour aux trajets
Mon Profil
Email
romain.caniaux@gmail.com
Prénom
romain
Nom
CANIAUX
Date de naissance
16/07/2001
Taille (cm)
176,00
Poids (kg)
90,00
Sexe
Homme
Sauvegarder

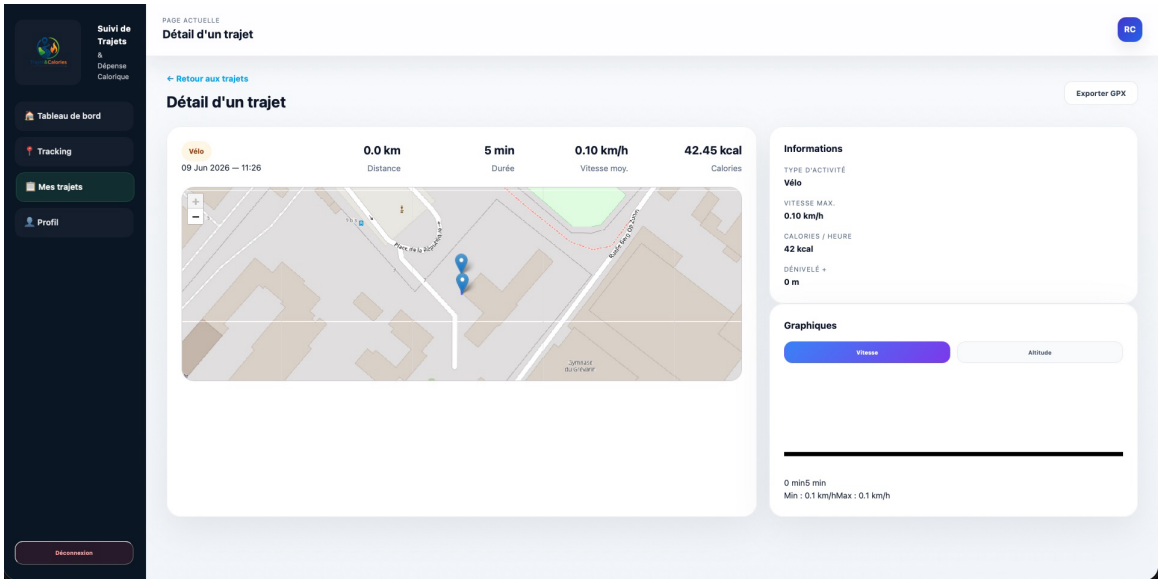
4.3 Collecte des données GPS

L'application utilise l'API Geolocation intégrée aux navigateurs web.









Les coordonnées GPS sont récupérées à intervalles réguliers puis transmises à l'API Symfony.

Les données enregistrées sont :

- latitude ;
- longitude ;
- date et heure ;
- précision GPS ;
- source de collecte.

4.4 Calcul des distances (Haversine)

Afin de calculer la distance entre deux positions GPS, l'application utilise la formule de Haversine.

$$d = 2R\arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Cette formule permet d'obtenir une estimation fiable de la distance entre deux points sur la surface terrestre.

4.5 Gestion des trajets

Une fonctionnalité a été développée afin de regrouper automatiquement les positions GPS en trajets cohérents.

Un trajet débute lorsqu'un nouveau point GPS est reçu après une période d'inactivité.

Il se termine lorsqu'aucune nouvelle position n'est reçue pendant plusieurs minutes ou lorsque la vitesse devient quasi nulle pendant une durée prolongée.

Pour chaque trajet, les informations suivantes sont calculées :

- heure de début ;
- heure de fin ;
- durée ;
- distance ;
- vitesse moyenne ;
- calories dépensées.

5. Bilan du sujet traité

Difficulté 1 — Apprentissage d'Angular

N'ayant jamais utilisé Angular dans un projet complet, la prise en main du framework a nécessité un temps d'adaptation important. J'ai dû me former en autonomie via la documentation officielle et des tutoriels, notamment sur la gestion des services, le routing et les appels HTTP vers l'API.

Difficulté 2 — Apprentissage de Symfony

De la même façon, Symfony et son système d'entités Doctrine m'étaient peu familiers. La configuration de la sécurité JWT a notamment été complexe à mettre en place, entre la génération des clés, la configuration du firewall et l'adaptation des contrôleurs.

Difficulté 3 — Liaison entre le back-end et le front-end

La communication entre l'API Symfony et l'application Angular a posé des problèmes de CORS (Cross-Origin Resource Sharing) que j'ai dû résoudre en configurant correctement les en-têtes HTTP côté Symfony. La gestion du token JWT dans les requêtes Angular a également demandé la mise en place d'un intercepteur HTTP.

Difficulté 4 — Intégration de Leaflet

L'ajout de la carte interactive via le composant Leaflet dans Angular n'était pas documenté de façon claire pour cette version du framework. J'ai dû adapter les imports et gérer l'initialisation de la carte de façon asynchrone pour éviter les erreurs de rendu.

5.3 Pistes d'amélioration

Plusieurs axes d'amélioration ont été identifiés pour faire évoluer l'application :

- Classement et statistiques globales entre utilisateurs ;
- Mode hors-ligne via une Progressive Web App (PWA) ;
- Notifications push pour les objectifs quotidiens ;
- Calcul BMR/TDEE pour affiner l'estimation des calories (intégration de l'âge et de la taille).

5.4 Amélioration implémentée

5.4.1 Export des données RGPD

Permet à l'utilisateur de télécharger toutes ses données au format ZIP.

Contenu : trajets GPX

Bénéfice : Respect du RGPD et transparence utilisateur.

5.5 Bugs corrigés

Quatre bugs importants ont été identifiés et corrigés :

Bug 1 : Doublons de trajets

Symptôme : Création de trajets dupliqués lors de la reconstruction.

Cause : Paramètre de date « depuis » non utilisé dans la requête.

Solution : Ajout du filtre de date dans la base de données.

Résultat : Plus aucun doublon, tous les trajets sont uniques.

Bug 2 : Positions GPS disparues

Symptôme : Points GPS disparaissaient après reconstruction.

Cause : Positions supprimées lors du processus de reconstruction.

Solution : Suppression de la ligne de suppression des données.

Résultat : Tous les points GPS sont conservés et visibles.

Bug 3 : Carte vide

Symptôme : Carte affichant un trajet restait vide.

Cause : Coordonnées GPS mal codées en URL.

Solution : Utilisation de encodeURIComponent() pour encoder correctement les coordonnées.

Résultat : Carte s'affiche correctement avec tous les points.

Bug 4 : Absence de bouton supprimer

Symptôme : Utilisateurs ne pouvaient pas supprimer leurs trajets.

Cause : Route DELETE et confirmation manquaient.

Solution : Ajout d'une route DELETE protégée avec confirmation modale.

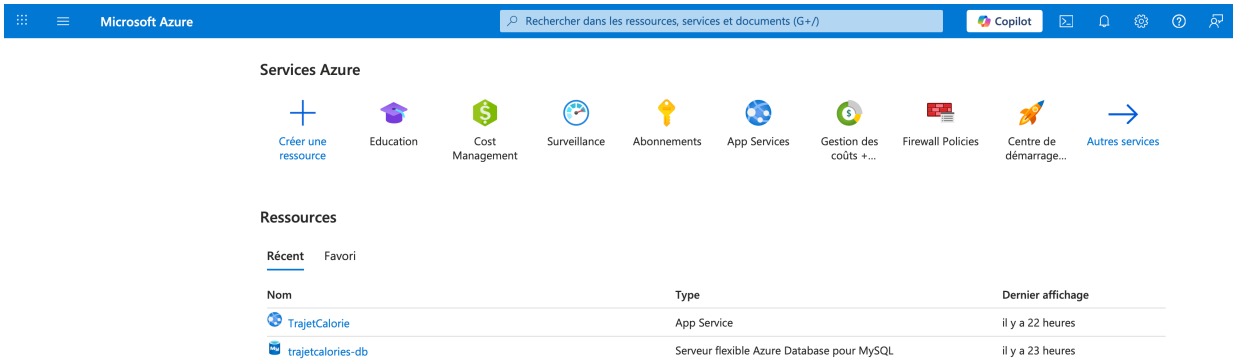
Résultat : Utilisateurs peuvent supprimer leurs trajets en toute sécurité.

5.6 Architecture et déploiement

L'application fonctionne en production sur un serveur VPS (Microsoft Azure Etudiant) avec :

Frontend : Application Angular compilée, servie par Nginx
Backend : API Symfony déployée via Docker
Base de données : MySQL avec backups quotidiens
Sécurité : HTTPS obligatoire, authentification JWT

Cette architecture garantit performance, sécurité et maintenabilité.



5.7 Prochaines étapes recommandées

Pour améliorer l'application, les prochaines étapes seraient :

Court terme : Mode hors-ligne (PWA), notifications push.

Moyen terme : Partage de trajets, challenges entre utilisateurs, système de badges.

Long terme : Application mobile, smartwatches, coaching personnalisé par IA.

Ces améliorations rendront l'application encore plus attractive et pertinente.

6. Bilan général du stage

6.1 Compétences acquises

Ce stage m'a permis de renforcer et d'acquérir les compétences suivantes :

Angular	Développement d'une SPA, gestion des services, routing, formulaires réactifs, appels HTTP
Symfony	Création d'une API REST, sécurisation JWT, entités Doctrine, migrations
MySQL	Conception du schéma de base de données, optimisation des requêtes
Deploiement	Apprendre la partie DevOps afin de publier une application et base de données via Microsoft Azure
Géolocalisation	Utilisation de l'API Geolocation HTML5, formule de Haversine
Méthodologie	Lecture et application d'un cahier des charges, gestion de projet en autonomie

Au-delà des aspects purement techniques, ce stage m'a appris à lire et appliquer un cahier des charges de façon autonome, à organiser mon travail par fonctionnalités, et à tester méthodiquement chaque endpoint avec Postman avant de passer à l'étape suivante.

La pratique d'Angular m'a permis de comprendre concrètement le fonctionnement d'une application SPA, la séparation des responsabilités entre composants et services, et la gestion des états d'authentification côté client.

Symfony m'a quant à lui donné une vision claire de ce qu'est une API REST sécurisée : gestion des rôles, hachage des mots de passe, signature des tokens, et sérialisation des réponses JSON.

6.2 Apport personnel et professionnel

Ce stage chez ALTAMEOS m'a permis de découvrir le fonctionnement concret d'une entreprise spécialisée dans le développement web. J'ai pu mettre en pratique les connaissances acquises en BTS SIO et développer mes compétences sur les frameworks Angular et Symfony.

Cette expérience m'a également permis de gagner en autonomie, en rigueur et en confiance dans mon travail. Enfin, elle a confirmé mon intérêt pour le développement informatique et m'a apporté une meilleure compréhension du monde professionnel.

6.3 Apport pour l'entreprise

À l'issue du stage, j'ai livré une base fonctionnelle comprenant l'API Symfony complète (authentification JWT, gestion des utilisateurs, collecte des positions GPS, reconstruction et calcul des trajets) ainsi qu'une interface Angular permettant à un utilisateur de s'inscrire, se connecter, enregistrer ses déplacements et consulter ses statistiques.

Ce travail constitue une base exploitable pour ALTAMEOS, qui pourra faire évoluer l'application selon les besoins futurs : ajout de nouvelles activités, intégration d'une carte interactive complète, ou déploiement en production. Mon implication a permis de gagner un temps significatif sur la phase de développement initiale du projet.

6.4 Conclusion

Ce stage chez ALTAMEOS m'a permis de découvrir le fonctionnement d'une entreprise spécialisée dans le développement web et de mettre en pratique les connaissances acquises au cours de ma formation. La réalisation d'un projet complet avec Angular et Symfony m'a permis de développer mes compétences techniques tout en gagnant en autonomie et en rigueur.

Cette expérience a été très enrichissante et a confirmé mon intérêt pour le développement informatique. Elle constitue une étape importante dans mon parcours professionnel et me sera utile pour la poursuite de mes études et de ma future carrière.

Annexes

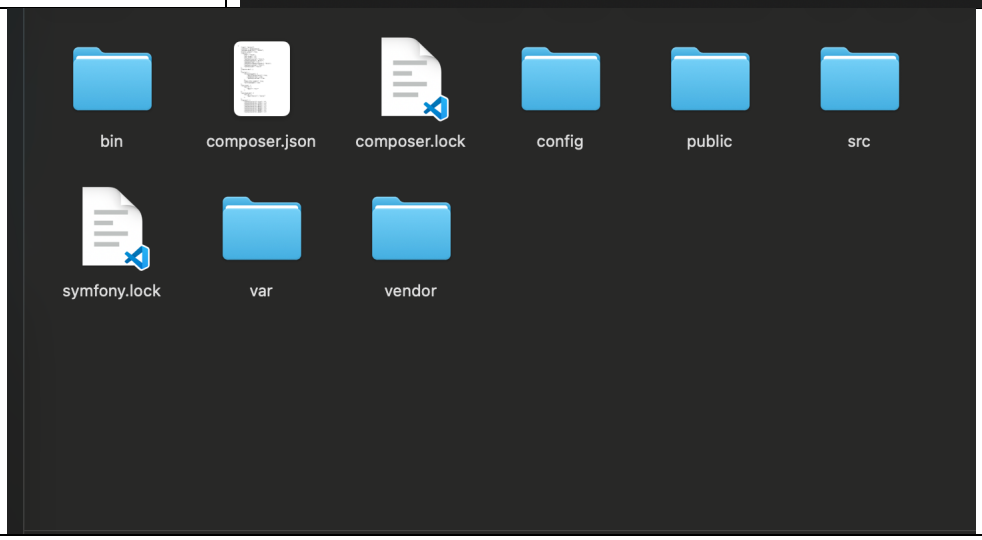
Annexe 1 — Documentation API SYMFONY

Création d'une API SYMFONY :

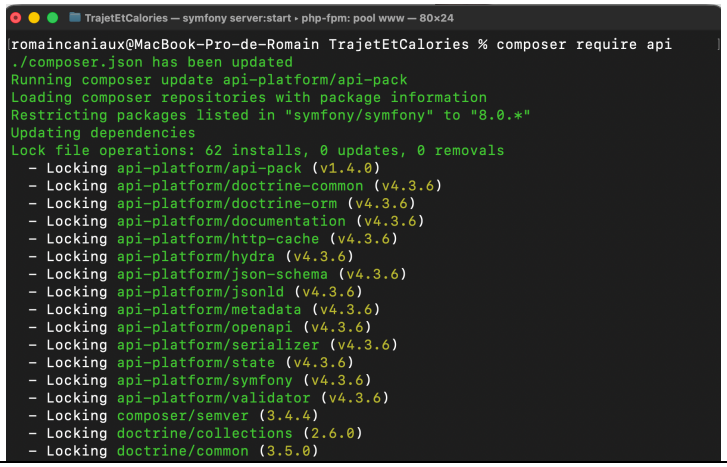
Il faut créer un « dossier » avec la commande suivante dans le terminal ->

symfony new trajetetcalories

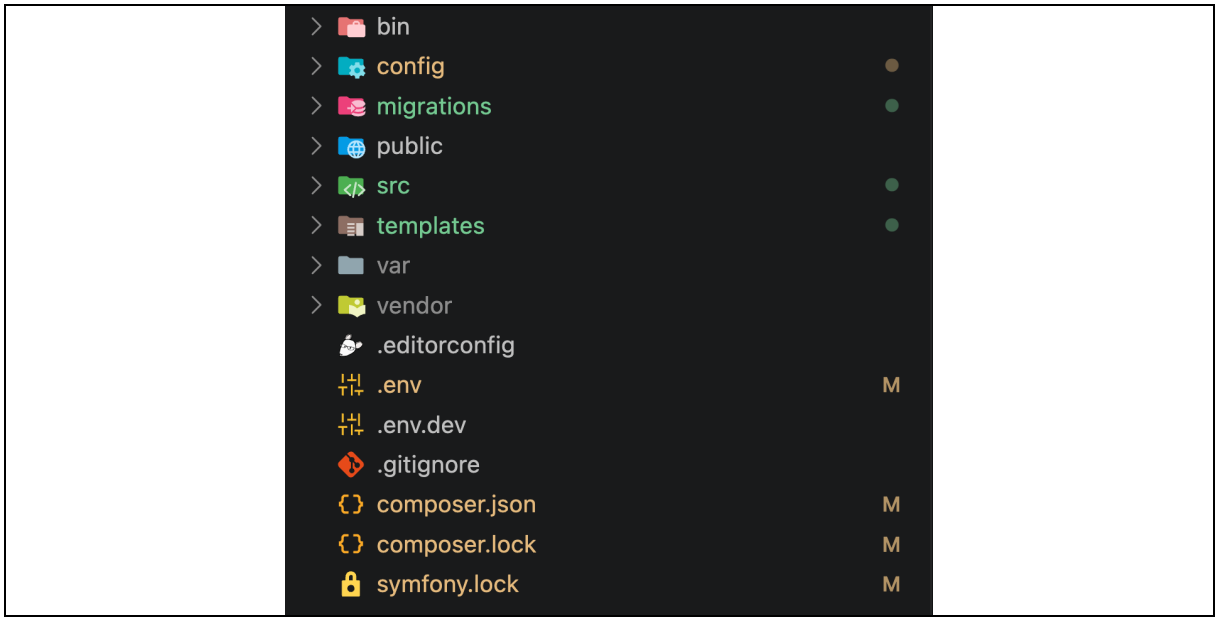
```
romaincaniaux@MacBook-Pro-de-Romain API % symfony new TrajetEtCalories

[WARNING] The configuration location for the Symfony CLI has changed in v5.17
.0.
Your configuration is still stored in the legacy directory.
Please follow these instructions:
 * symfony server:stop --all
 * symfony proxy:stop
 * move "/Users/romaincaniaux/.symfony5" to
"/Users/romaincaniaux/Library/Application Support/symfony-cli"

* Creating a new Symfony project with Composer
* Setting up the project under Git version control
  (running git init /Users/romaincaniaux/Desktop/PROJET SANS IA/API/TrajetEtCalo
ries)

[OK] Your project is now ready in /Users/romaincaniaux/Desktop/PROJET SANS IA
/API/TrajetEtCalories
```



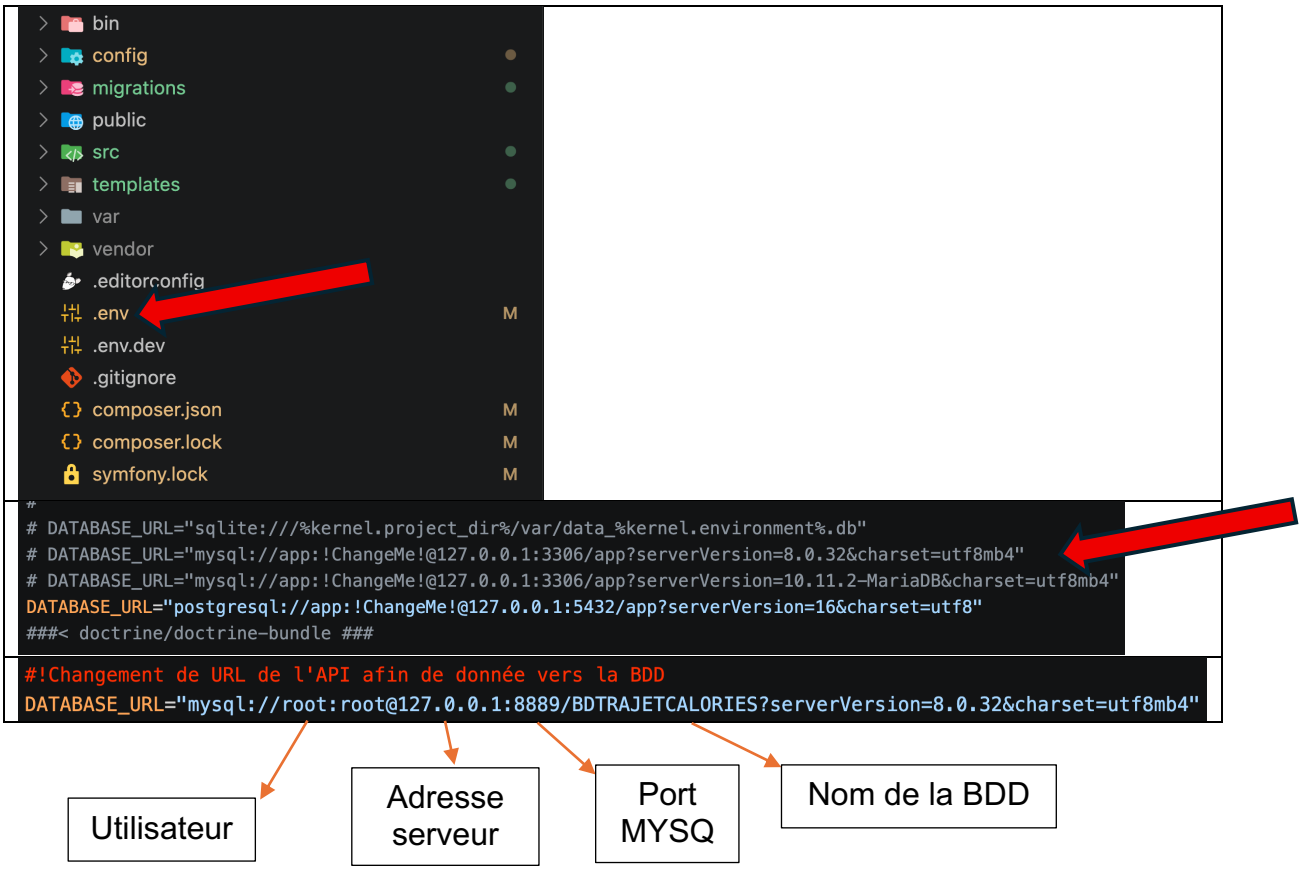
Une fois le dossier créé il faut vérifier que la création s'est bien passée il faut donc faire cette commande pour lancer le server Symfony ->

SYMFONY SERVER :START	
composer require api	Cette commande permet d'installer les dépendances API 

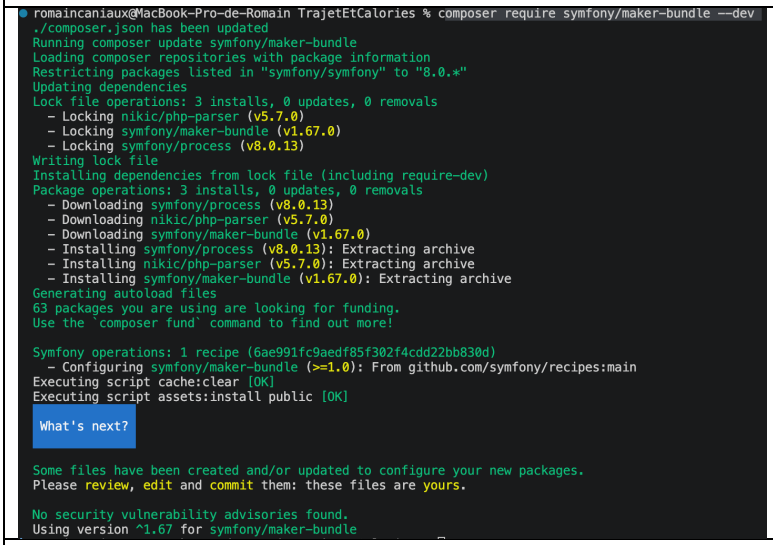
Une fois cette vérification faite nous pouvons ouvrir Visual Studio Code pour ouvrir le dossier que nous avons créé.



Une fois ouvert nous pouvons changer l'URL dans le fichier « .env » qui représente le chemin vers notre base de données relationnel « MySql ».

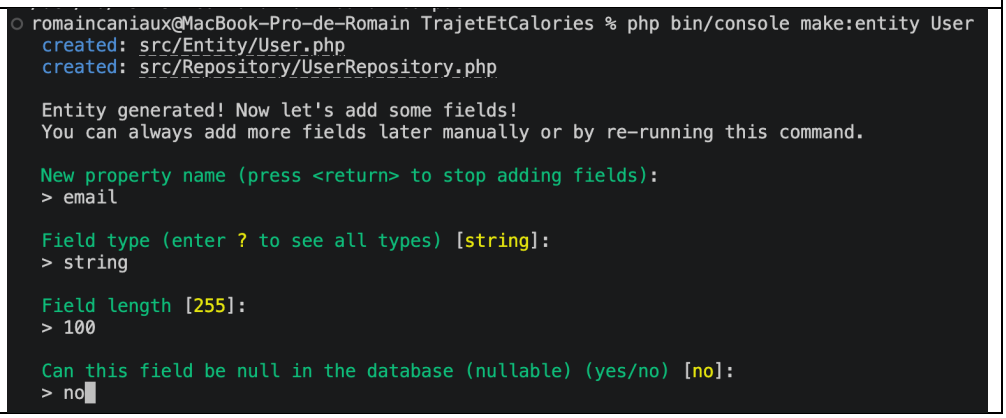


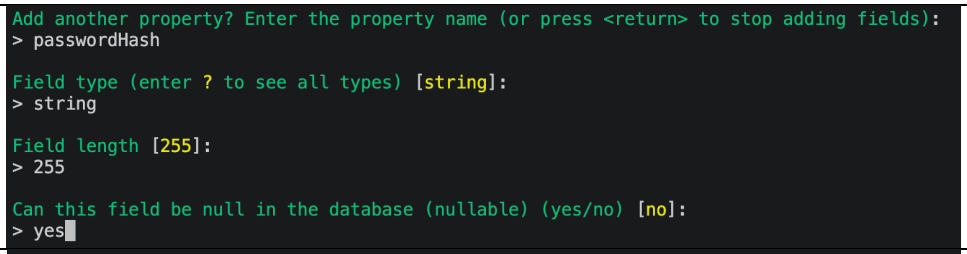
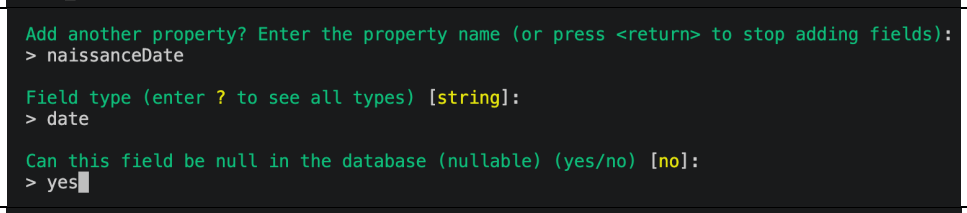
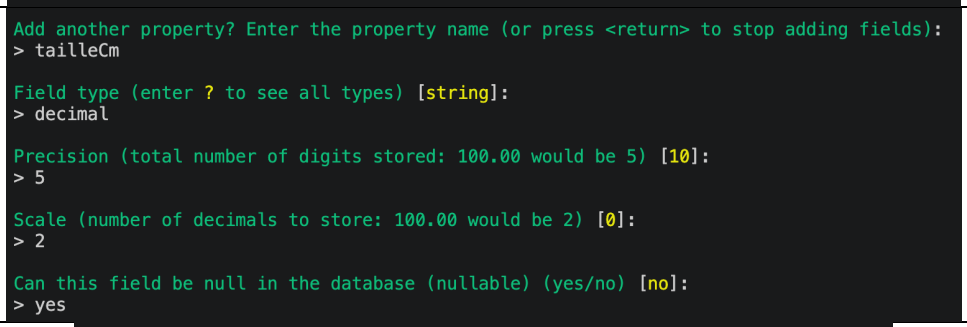
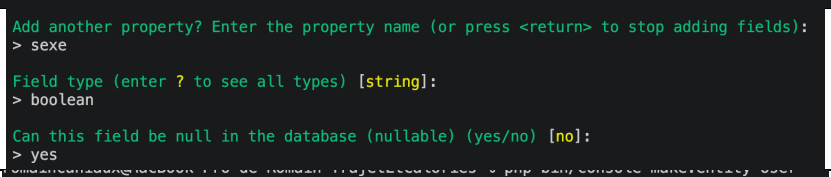
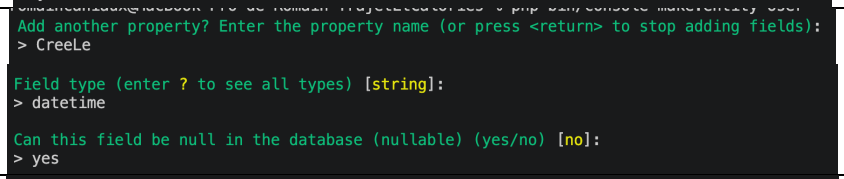
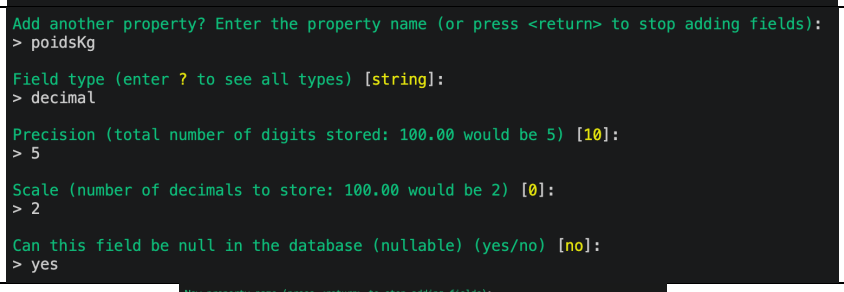
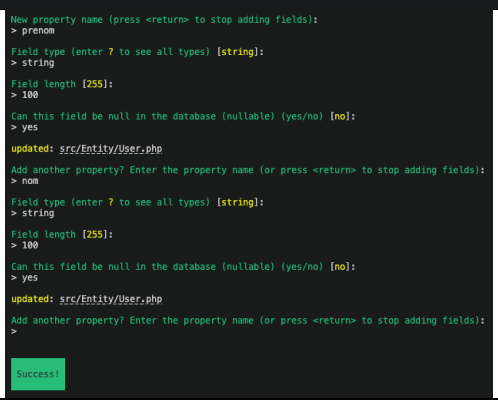
Si la base de données n'est pas créée avant voilà quoi faire :

composer require symfony/maker-bundle --dev	Il faut ensuite faire là cette commande pour installer MakeBundle
	Une fois l'installation terminée nous pouvons passer aux commandes suivantes pour la création des entity
php bin/console make:entity	Cette commande permet de créer les entités il faut ensuite suivre ce que nous dit VSCode

Si la base de données est créée en amont il faut créer les entity manuellement et faire les choses suivantes :

User :

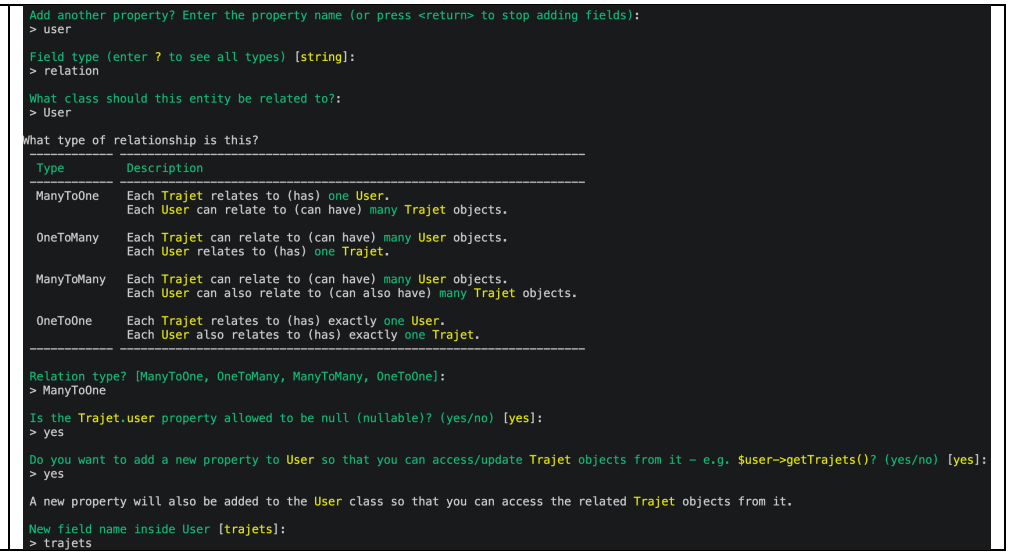
php bin/console make:entity User	Il faut suivre le type de donnée
Email	

Mot de passe crypté	
Date de naissance	
Taille en Cm	
Genre	
Date de création	
Poids en Kg	
Nom et Prenom	

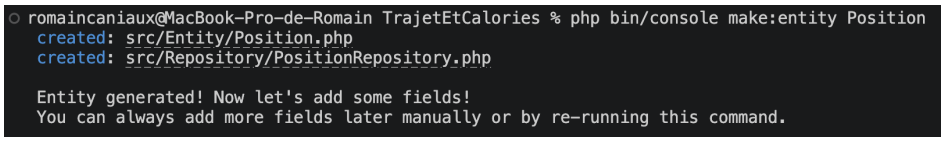
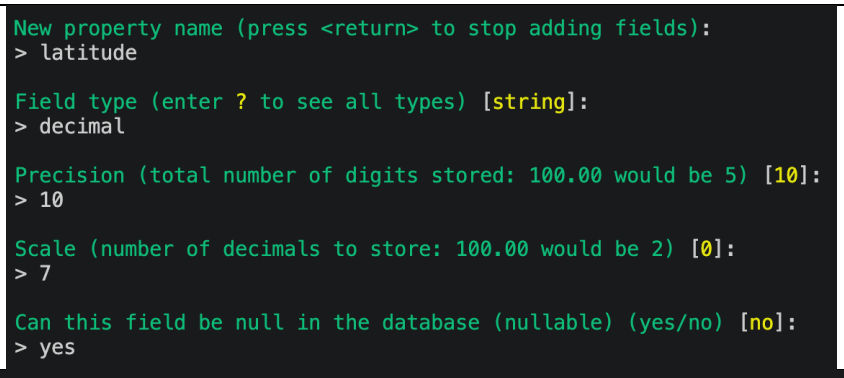
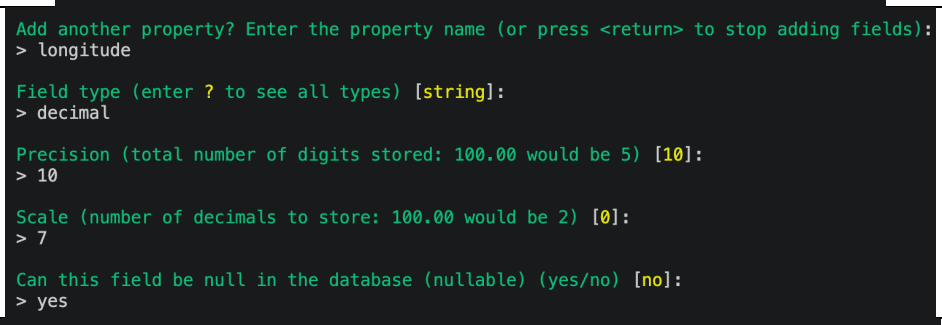
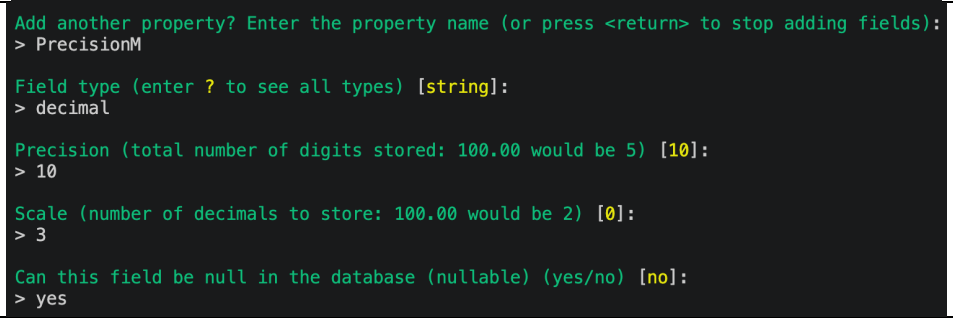
On répète l’opération pour les autres tables :

Trajet :

php bin/console make:entity Trajet	``` romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console make:entity Traje created: src/Entity/Trajet.php created: src/Repository/TrajetRepository.php Entity generated! Now let's add some fields! You can always add more fields later manually or by re-running this command. ```
Date de début	``` New property name (press <return> to stop adding fields): > debutTime Field type (enter ? to see all types) [string]: > datetime Can this field be null in the database (nullable) (yes/no) [no]: > yes ```
Date de fin	``` Add another property? Enter the property name (or press <return> to stop adding fields): > finTime Field type (enter ? to see all types) [string]: > datetime Can this field be null in the database (nullable) (yes/no) [no]: > yes ```
Distance en Km	``` Add another property? Enter the property name (or press <return> to stop adding fields): > distanceKm Field type (enter ? to see all types) [string]: > decimal Precision (total number of digits stored: 100.00 would be 5) [10]: > 10 Scale (number of decimals to store: 100.00 would be 2) [0]: > 2 Can this field be null in the database (nullable) (yes/no) [no]: > yes ```
Vitesse Moyenne	``` Add another property? Enter the property name (or press <return> to stop adding fields): > vitesseMoyKmh Field type (enter ? to see all types) [string]: > decimal Precision (total number of digits stored: 100.00 would be 5) [10]: > 10 Scale (number of decimals to store: 100.00 would be 2) [0]: > 2 Can this field be null in the database (nullable) (yes/no) [no]: > yes ```
Type d'activité	``` Add another property? Enter the property name (or press <return> to stop adding fields): > typeActivite Field type (enter ? to see all types) [string]: > string Field length [255]: > 100 Can this field be null in the database (nullable) (yes/no) [no]: > yes ```
Calories	``` Add another property? Enter the property name (or press <return> to stop adding fields): > caloriesKcal Field type (enter ? to see all types) [string]: > decimal Precision (total number of digits stored: 100.00 would be 5) [10]: > 10 Scale (number of decimals to store: 100.00 would be 2) [0]: > 2 Can this field be null in the database (nullable) (yes/no) [no]: > yes ```

Relation avec User	

Position :

Php bin/console make :entity Position	
Latitude	
Longitude	
Précision	

Source	Add another property? Enter the property name (or press <return> to stop adding fields): > source Field type (enter ? to see all types) [string]: > string Field length [255]: > 20 Can this field be null in the database (nullable) (yes/no) [no]: > yes
Date	Add another property? Enter the property name (or press <return> to stop adding fields): > date Field type (enter ? to see all types) [string]: > datetime Can this field be null in the database (nullable) (yes/no) [no]: > yes
Relation avec User	Add another property? Enter the property name (or press <return> to stop adding fields): > user Field type (enter ? to see all types) [string]: > relation What class should this entity be related to?: > User What type of relationship is this? Type — Description ManyToOne — Each Position relates to (has) one User. Each User can relate to (can have) many Position objects. OneToMany — Each Position can relate to (can have) many User objects. Each User relates to (has) one Position. ManyToMany — Each Position can relate to (can have) many User objects. Each User can also relate to (can also have) many Position objects. OneToOne — Each Position relates to (has) exactly one User. Each User also relates to (has) exactly one Position. Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]: > ManyToOne Is the Position.user property allowed to be null (nullable)? (yes/no) [yes]: > no Do you want to add a new property to User so that you can access/update Position objects from it - e.g. $user->getPositions()? (yes/no) [yes]: > yes A new property will also be added to the User class so that you can access the related Position objects from it. New field name inside User [positions]: > positions Do you want to activate orphanRemoval on your relationship? A Position is "orphaned" when it is removed from its related User. e.g. $user->removePosition($position) NOTE: If a Position may *change* from one User to another, answer "no". Do you want to automatically delete orphaned App\Entity\Position objects (orphanRemoval)? (yes/no) [no]: > no

Une fois les entités créées il faut créer le fichier de synchronisation des entity avec la base de données avec la commande suivante :

php bin/console doctrine:migrations:diff	romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console doctrine:migrations:diff [WARNING] You have 1 previously executed migrations in the database that are not registered migrations. Are you sure you wish to continue? (yes/no) [yes]: > yes Generated new migration class to "/Users/romaincaniaux/Desktop/PROJET SANS IA/API/TrajetEtCalories/migrations/Version20260528103053.php" To run just this migration for testing purposes, you can use migrations:execute --up "DoctrineMigrations\\Version20260528103053" To revert the migration you can use migrations:execute --down "DoctrineMigrations\\Version20260528103053"

Une fois terminé il faut synchroniser les entity avec la base de données

php bin/console doctrine:migrations:migrate	romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console doctrine:migrations:migrate WARNING! You are about to execute a migration in database "bdtrajetcalories" that could result in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]: > yes [WARNING] You have 1 previously executed migrations in the database that are not registered migrations. >> 2026-05-27 13:28:38 (DoctrineMigrations\Version20260527132802) Are you sure you wish to continue? (yes/no) [yes]: > yes [notice] Migrating up to DoctrineMigrations\Version20260528103053 [notice] finished in 47.9ms, used 20M memory, 1 migrations executed, 10 sql queries [OK] Successfully migrated to version: DoctrineMigrations\Version20260528103053 romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories %

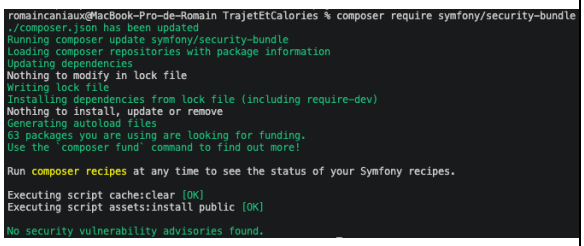
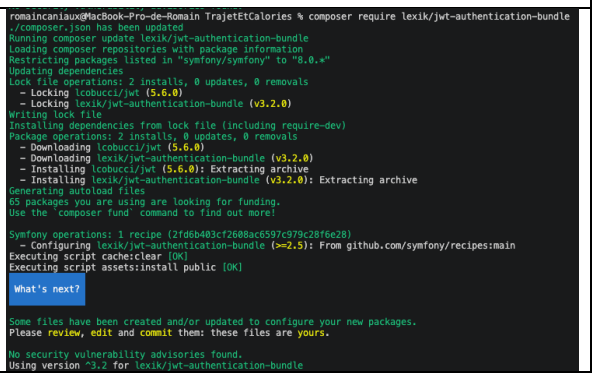
Pour faire simple, les entités créées servent à représenter les tables de la base de données

Une fois cette notion finis, passons au « Controller »

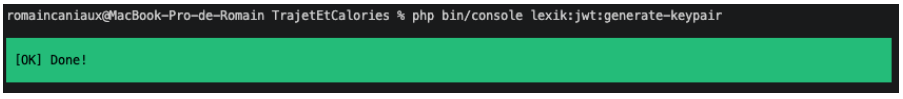
Il faut créer les contrôleurs via les commandes suivantes :

Php bin/console make :controller UserController	romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console make:controller UserControlle Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]: > n created: src/Controller/UserController.php created: templates/user/index.html.twig Success! Next: Open your new controller class and add some pages! romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories %
Php bin/console make :controller TrajetController	romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console make:controller TrajetControll Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]: > n created: src/Controller/TrajetController.php created: templates/trajet/index.html.twig Success! Next: Open your new controller class and add some pages!
Php bin/console make :controller PositionController	romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories % php bin/console make:controller PositionController Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]: > n created: src/Controller/PositionController.php created: templates/position/index.html.twig Success! Next: Open your new controller class and add some pages! romaincaniaux@MacBook-Pro-de-Romain TrajetEtCalories %

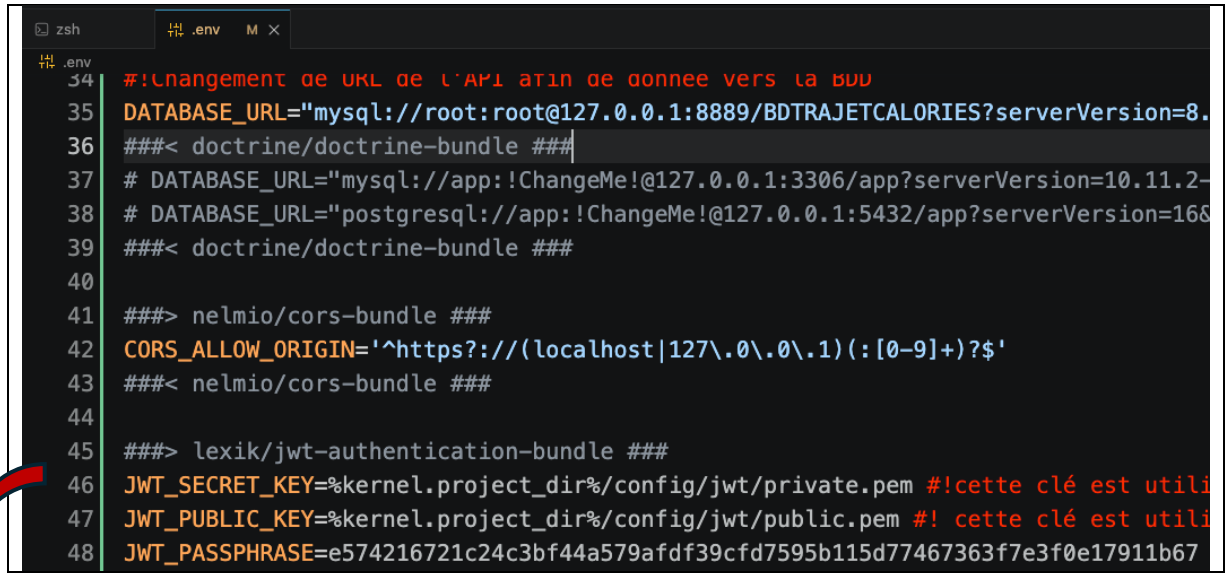
Dès que les contrôleurs sont créés nous devons installer les dépendances suivantes :

composer require symfony/security-bundle	• Gère les utilisateurs • Protège les routes • Chiffre les mots de passe • Définit les rôles (utilisateur, administrateur)	
composer require lexik/jwt-authentication-bundle	• Génère un Token à la création d'un utilisateur	

Une fois les dépendances installées nous devons configurer les paires de clés publiques et privé pour signer les Tokens avec la commande suivante :

php bin/console lexik:jwt:generate-keypair	

Pour vérifier que cela est bien fonctionné allons voir le fichier « .env »



- JWT_SECRET_KEY → Chemin vers la clé privée (Signe le TOKEN)
- JWT_PUBLIC_KEY → Chemin vers la clé publique (vérification du TOKEN)
- JWT_PASSPHRASE → Mot de passe qui protège la clé privée

Nous allons ensuite dans le fichier « security.yaml » dans config/packages/security.yaml, afin de le configurer :

```yaml
security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'

    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        users_in_memory: { memory: null }

    firewalls:
        dev:
            # Ensure dev tools and static assets are always allowed
            pattern: ^/(_profiler|_wdt|assets|build)/
            security: false
        main:
            lazy: true
            provider: users_in_memory

            # Activate different ways to authenticate:
            # https://symfony.com/doc/current/security.html#the-firewall

            # https://symfony.com/doc/current/security/impersonating_user.html
            # switch_user: true

    # Note: Only the *first* matching rule is applied
    access_control:
        # - { path: ^/admin, roles: ROLE_ADMIN }
        # - { path: ^/profile, roles: ROLE_USER }
```

```yaml
security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        App\Entity\User: 'auto' #! Modification pour utiliser le hachage automatique vers User #

    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        app_user_provider: #! Modification pour utiliser le provider de l'entité User #
            entity: #! Entity sert à identifier l'utilisateur #
                class: App\Entity\User #! Class sert à identifier l'utilisateur #
                property: email #! Property sert à identifier l'utilisateur #

    firewalls:
        dev:
            # Ensure dev tools and static assets are always allowed
            pattern: ^/(_profiler|_wdt|assets|build)/
            security: false
        #!Suppression du firewall main pour utiliser le firewall login#
        login:
            pattern: ^/api/login #! pattern sert à identifier les routes qui nécessitent une authentification #
            stateless: true #! stateless sert à indiquer que l'authentification

        api:
            pattern: ^/api #! pattern sert à identifier les routes qui nécessitent une authentification #
            stateless: true #! stateless sert à indiquer que l'authentification
            provider: app_user_provider #! provider sert à identifier le provider de l'utilisateur #
            jwt: ~ #! jwt sert à indiquer que l'authentification se fait via un token JWT #

            # Activate different ways to authenticate:
            # https://symfony.com/doc/current/security.html#the-firewall

            # https://symfony.com/doc/current/security/impersonating_user.html
            # switch_user: true

    # Note: Only the *first* matching rule is applied
    #! Modification pour utiliser les access_control pour sécuriser les routes de l'API#
    access_control:
    - { path: ^/api/login, roles: PUBLIC_ACCESS } #! path sert à identifier les routes qui nécessitent une authe
    - { path: ^/api/register, roles: PUBLIC_ACCESS } #! path sert à identifier les routes qui nécessitent une au
    - { path: ^/api/, roles: ROLE_USER }
```

La configuration de la sécurité de l'API est donc faite :

Password_hashers	Permet de dire à Symfony d'utiliser l'algorithme « auto » pour hacher les mots de passe de l'entité User
Providers	Permet de dire à Symfony de chercher un utilisateur via le champ email de la table User
Firewall	Définit deux zones de sécurité avec login et api
Access-control	Définit les règles d'accès

Sans cela, l'API est ouverte à tous ou bloqué.

Une fois cela fait, nous allons devoir modifier l'entity « User.php » :

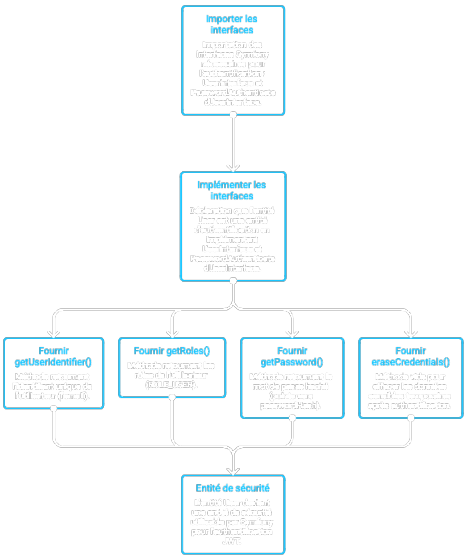
Ajout et des « use » afin de pouvoir utiliser les fonctionnalités de Symfony Sécurité	

```php
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface; //! Ajout de l'inter

#[ORM\Entity(repositoryClass: UserRepository::class)]
//! Modification de l'entité pour ajouter "implements UserInterface, Passwor
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
```

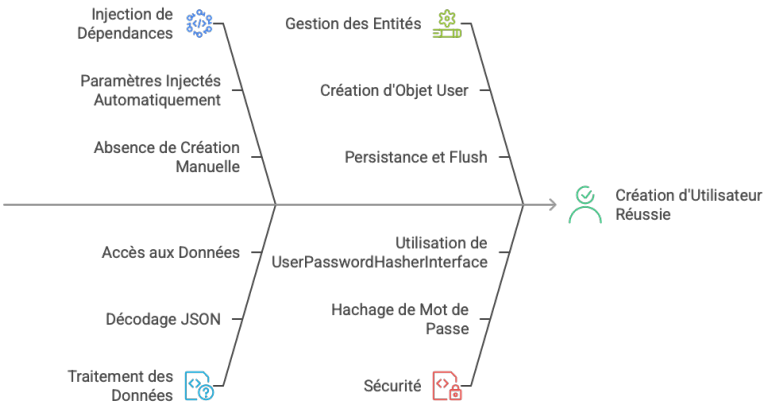
Ajout des méthodes pour identifier les utilisateurs via les emails, retourne les rôles des utilisateurs, retourne le mot de passe haché de l'utilisateur et une méthode pour supprimer les données sensibles comme mot de passe en clair	

```php
// * Méthodes requises par UserInterface et PasswordAuthenticatedUserInterface

//! Méthode pour identifier l'utilisateur, ici on utilise l'email comme identifiant unique #
    public function getUserIdentifier(): string
    {
        return (string) $this->email;
    }

// ! Retourne les rôles de l'utilisateur, tous les utilisateurs ont ROLE_USER par défaut
    public function getRoles(): array
    {
        return ['ROLE_USER'];
    }

    public function getPassword(): string
    {
        return (string) $this->passwordHash;
    }

    public function eraseCredentials():void
    {
        // $this->plainPassword = null; //! Permet de supprimer les données sensibles de l'uti
    }
```

Schéma du processus d'un utilisateur :



Une fois cela compris et mis en place nous pouvons adapter le Controller User à cela :



Ajout des nouveaux use pour ajouter JsonReponse, Request, EntityManagerInterface, userPasswordHasherInterface et User

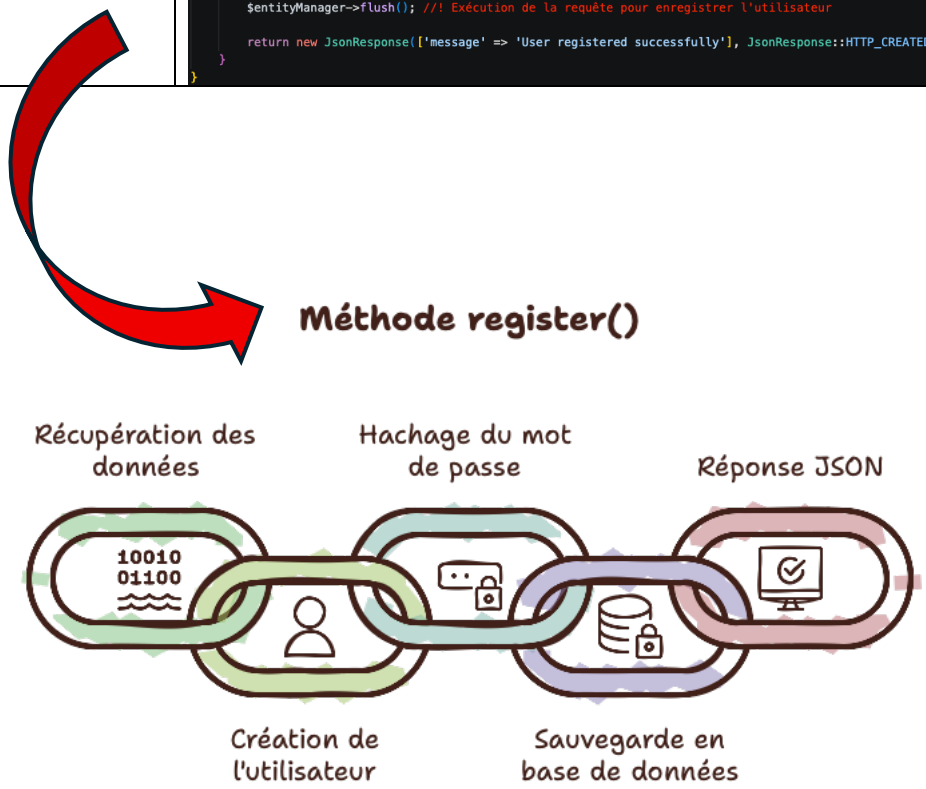
```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\HttpFoundation\JsonResponse; //! Ajout de JsonRespons
use Symfony\Component\HttpFoundation\Request; //! Ajout de Request pour gé
use Doctrine\ORM\EntityManagerInterface; //! Ajout de EntityManagerInterfac
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface; //
use App\Entity\User; //! Ajout de l'entité User pour manipuler les utilisat
```

Modification de la route pour que Symfony puisse la gérer

```php
// ! Modification de /user par /api pour indiquer que ce contrôleur gère des routes d'API
    #[Route('/api')]
final class UserController extends AbstractController
{
```

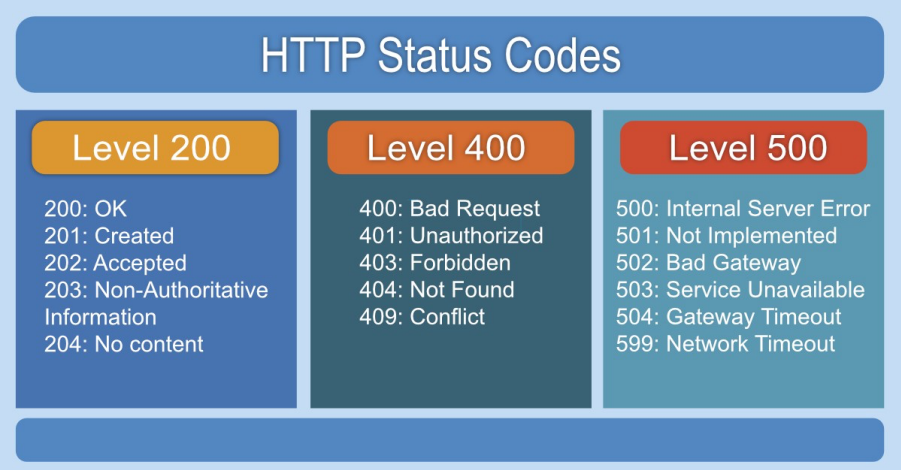
Ajout de la fonction register() afin de pouvoir enregistrer un utilisateur

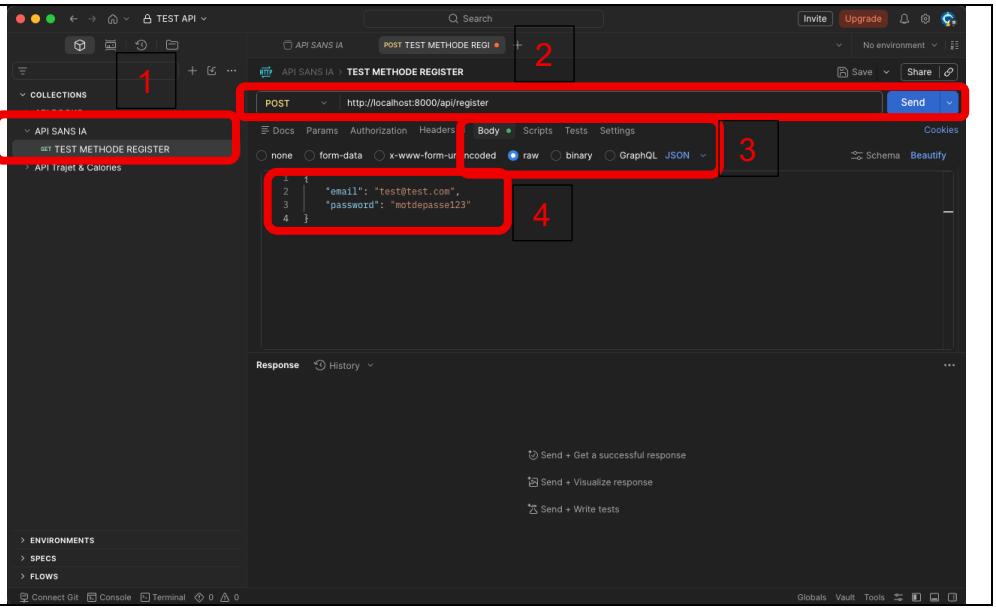
```php
final class UserController extends AbstractController
{
    //! Ajout d'une méthode register pour récupérer tous les utilisateurs et les retourner au format JSON
    #[Route('/register', name: 'app_register', methods: ['POST'])] //! Route pour l'inscription des utilisateurs, utilisant la méthode POST pour
    public function register(Request $request, EntityManagerInterface $entityManager, UserPasswordHasherInterface $passwordHasher): JsonResponse
    {
        $data = json_decode($request->getContent(), true); //! Récupération des données d'inscription envoyées en JSON dans la requête

        $user = new User(); //! Création d'une nouvelle instance de l'entité User
        $user->setEmail($data['email']); //! Définition de l'email de l'utilisateur à partir des données reçues
        $user->setPasswordHash($passwordHasher->hashPassword($user, $data['password'])); //! Hachage du mot de passe et définition du mot de pass

        $entityManager->persist($user); //! Persistance de l'utilisateur dans la base de données
        $entityManager->flush(); //! Exécution de la requête pour enregistrer l'utilisateur

        return new JsonResponse(['message' => 'User registered successfully'], JsonResponse::HTTP_CREATED); //! Retour d'une réponse JSON indiqu
    }
}
```

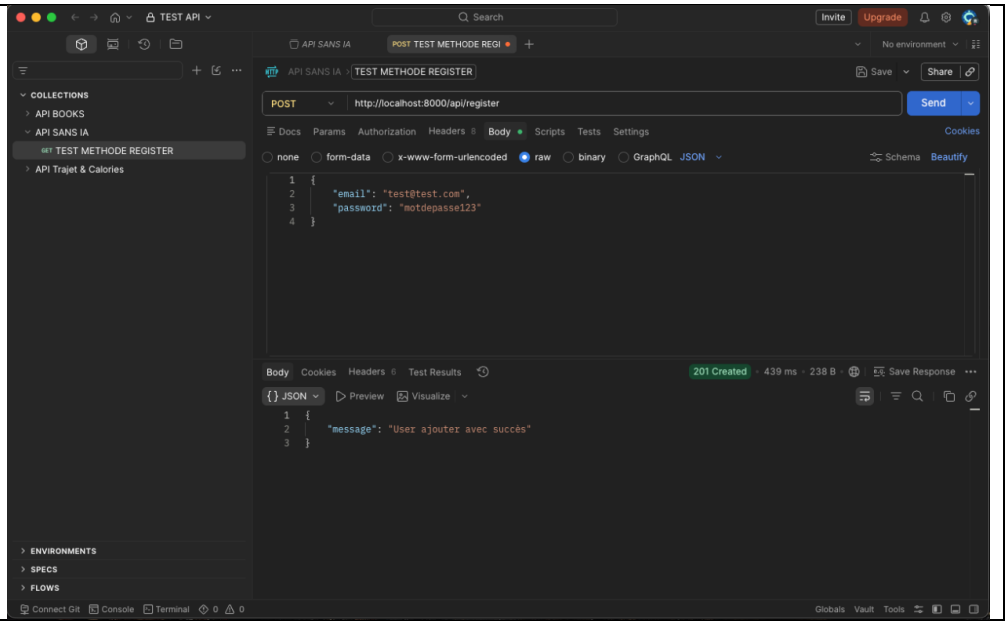


Une fois cela terminé nous devons tester si aucun problème n'est rencontré grâce à « POSTMAN » :

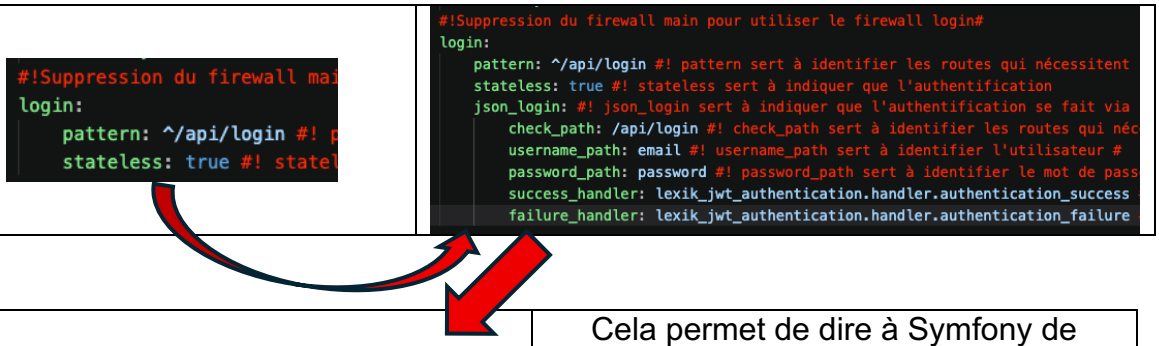
Schéma des codes http :



Nous devons faire la configuration suivante :	

Si l'inscription se passe pour le mieux voilà le résultat	

Une fois cela fait passons au Login en modifiant le login dans « security.yaml » :



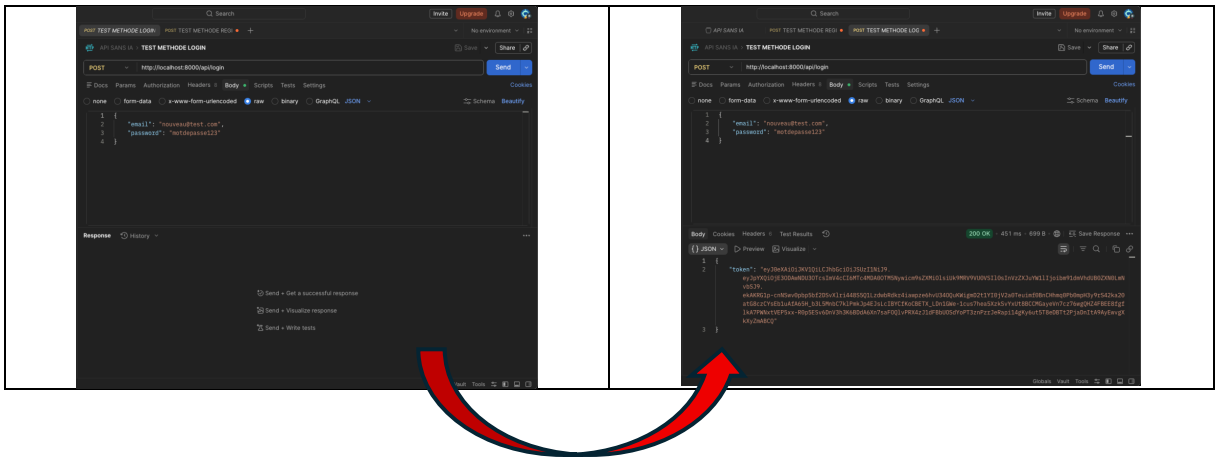
```
#!Suppression du firewall main pour utiliser le firewall login#
login:
    pattern: ^/api/login #! pattern sert à identifier les routes qui nécessitent
    stateless: true #! stateless sert à indiquer que l'authentification
    json_login: #! json_login sert à indiquer que l'authentification se fait via
        check_path: /api/login #! check_path sert à identifier les routes qui néc
        username_path: email #! username_path sert à identifier l'utilisateur #
        password_path: password #! password_path sert à identifier le mot de pass
        success_handler: lexik_jwt_authentication.handler.authentication_success
        failure_handler: lexik_jwt_authentication.handler.authentication_failure
```

Json_login	Cela permet de dire à Symfony de prendre un email et mot de passe en JSON
Username_path	Identifiant avec le nom « email »
Success-handler	Génère le token de l'utilisateur
Failure-handler	Retourne une erreur si la connexion a échoué

Avant de tester nous allons ajouter la route dans le fichier « routes.yaml » afin que Symfony comprenne le chemin qu'il doit emprunter

```yaml
# yaml-language-server: $schema=../vendor/symfony/routing/Loader/schema/routing.schema.json

# This file is the entry point to configure the routes of your app.
# Methods with the #[Route] attribute are automatically imported.
# See also https://symfony.com/doc/current/routing.html

# To list all registered routes, run the following command:
#   bin/console debug:router

controllers:
    resource: routing.controllers


#! Ajout de la route pour le login de l'API#
api_login:
    path: /api/login #! path sert à identifier les routes qui nécessitent une authentification #
    methods: ['POST'] #! methods qui sert à identifier les méthodes HTTP autorisées pour la route #

```

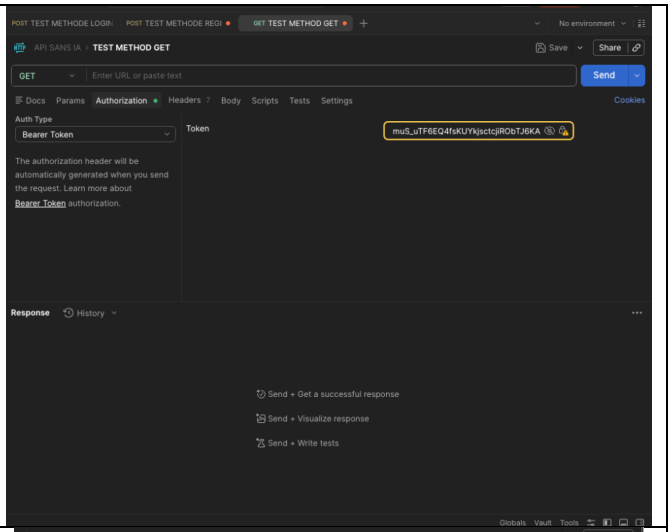
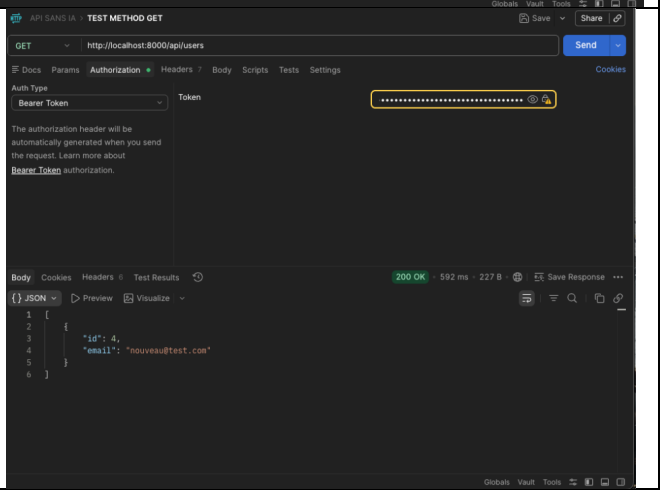
Une fois cela fait testons le sur Postman :



Pour vérifier que le Token fonctionne allons le tester avec une méthode GET via « POSTMAN » :

Mais avant il faut créer la méthode GET dans le controller user :

```php
    //! Ajout d'une méthode getUsers pour récupérer tous les utilisateurs et
#[Route('/users', name: 'app_users', methods: ['GET'])] //! Route pour récupé
public function getUsers(EntityManagerInterface $entityManager): JsonResponse
{
    $users = $entityManager->getRepository(User::class)->findAll(); //! Récup
    $usersData = []; //! Initialisation d'un tableau pour stocker les données
    foreach ($users as $user) { //! Parcours de chaque utilisateur pour extra
        $usersData[] = [
            'id' => $user->getId(), //! Ajout de l'ID de l'utilisateur au tab
            'email' => $user->getEmail(), //! Ajout de l'email de l'utilisate
        ];
    }

    return new JsonResponse($usersData, JsonResponse::HTTP_OK); //! Retour d'
```

Nous devons créer une nouvelle requête et aller dans « authorization » et choisir le type « Bearer Token » ensuite collée le TOKEN dans le champ qui s'affiche	
Si le Token est valide alors voilà le résultat	

Nous allons devoir faire la même chose pour Trajet :

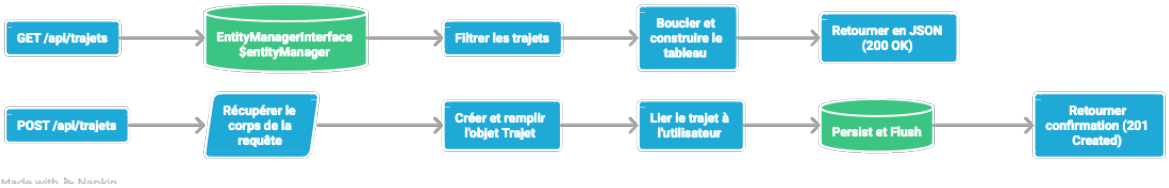
Changement et implémentation des bons	`use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;`
	`use Symfony\Component\Routing\Attribute\Route;`
	`use Symfony\Component\HttpFoundation\JsonResponse; //! Ajout de J`
	`use Symfony\Component\HttpFoundation\Request; //! Ajout de Reques`
	`use Doctrine\ORM\EntityManagerInterface; //! Ajout de EntityManag`
	`use App\Entity\Trajet; //! Ajout de l'entité User pour manipuler`

Ajout de la route et de la méthode GET	`// ! Modification de /trajet par /api pour indiquer que ce contrôleur`
	`#[Route('/api')]`
	`final class TrajetController extends AbstractController`
	`{`
	`#[Route('/trajets', name: 'api_trajets_list', methods: ['GET'])]`

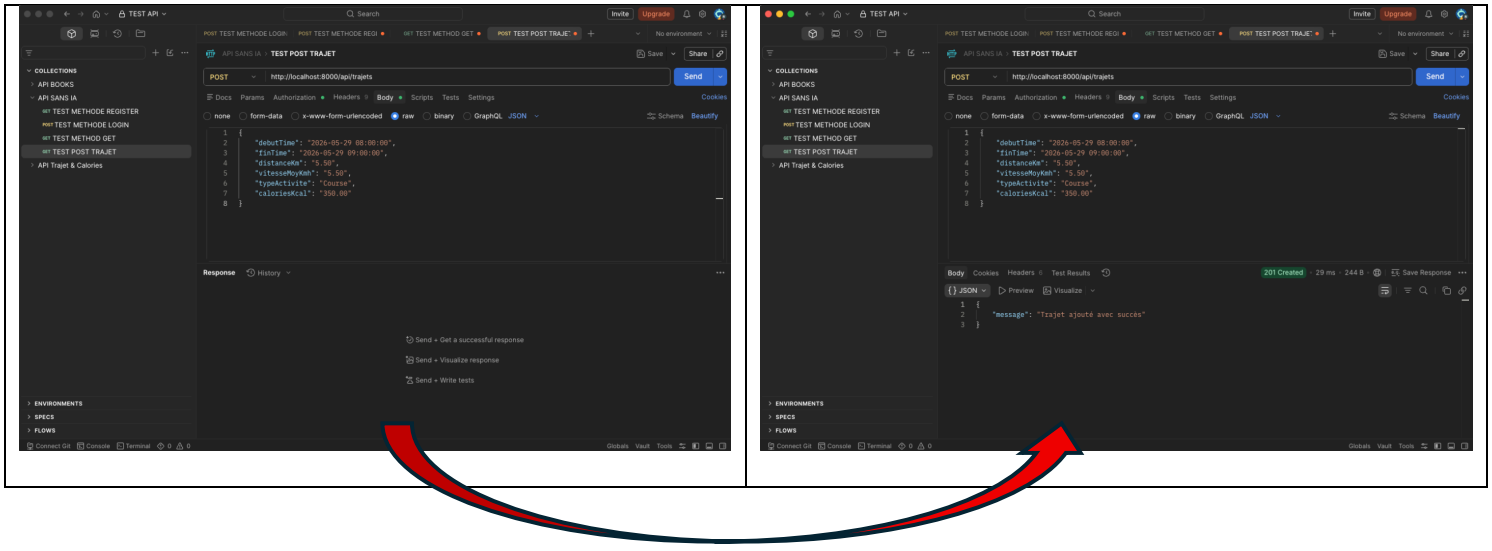
Création de la méthode getTrajet()		`public function getTrajets(EntityManagerInterface $entityManager): JsonResponse //! Retour d'une r`
		`{`
		`  //! Récupération de tous les trajets de l'utilisateur actuellement connecté à partir de la bas`
		`  $trajets = $entityManager->getRepository(Trajet::class)->findBy(['user' => $this->getUser()]);`
		`  $trajetsData = []; //! Initialisation d'un tableau pour stocker les données des trajets`
		`  foreach ($trajets as $trajet)`
		`  { //! Parcours de chaque trajet pour extraire les données nécessaires`
		`  $trajetsData[] =`
		`  [`
		`  'id' => $trajet->getId(), //! Récupération de l'identifiant du trajet`
		`  'debutTime' => $trajet->getDebutTime()?->format('Y-m-d H:i:s'), //! Récupération de la`
		`  'finTime' => $trajet->getFinTime()?->format('Y-m-d H:i:s'), //! Récupération de la dat`
		`  'distanceKm' => $trajet->getDistanceKm(), //! Récupération de la distance du trajet en`
		`  'vitesseMoyKmh' => $trajet->getVitesseMoyKmh(), //! Récupération de la vitesse moyenne`
		`  'typeActivite' => $trajet->getTypeActivite(), //! Récupération du type d'activité du t`
		`  'caloriesKcal' => $trajet->getCaloriesKcal(), //! Récupération des calories brûlées pe`
		`  ];`
		`  }`
		`  return new JsonResponse($trajetsData, JsonResponse::HTTP_OK); //! Retour d'une réponse JSON co`
		`}`

Création de la méthode createTrajet()		`#[Route('/trajets', name: 'api_trajets_create', methods: ['POST'])]`
		`public function createTrajet(Request $request, EntityManagerInterface $entityManager): JsonResponse`
		`{`
		`  $data = json_decode($request->getContent(), true); //! Récupération des données du trajet envoyé`
		`  $trajet = new Trajet(); //! Création d'une nouvelle instance de l'entité Trajet`
		`  $trajet->setDebutTime(new \DateTime($data['debutTime'])); //! Définition de la date de début du`
		`  $trajet->setFinTime(new \DateTime($data['finTime'])); //! Définition de la date de fin du trajet`
		`  $trajet->setUser($this->getUser()); //! Association du trajet à l'utilisateur actuellement conne`
		`  $trajet->setDistanceKm($data['distanceKm']); //! Définition de la distance du trajet à partir de`
		`  $trajet->setVitesseMoyKmh($data['vitesseMoyKmh']); //! Définition de la vitesse moyenne du traje`
		`  $trajet->setTypeActivite($data['typeActivite']); //! Définition du type d'activité du trajet à p`
		`  $trajet->setCaloriesKcal($data['caloriesKcal']); //! Définition des calories du trajet à partir`
		`  $entityManager->persist($trajet); //! Persistance du trajet dans la base de données`
		`  $entityManager->flush(); //! Exécution de la requête pour enregistrer le trajet`
		`  return new JsonResponse(['message' => 'Trajet ajouté avec succès'], JsonResponse::HTTP_CREATED);`
		`}`

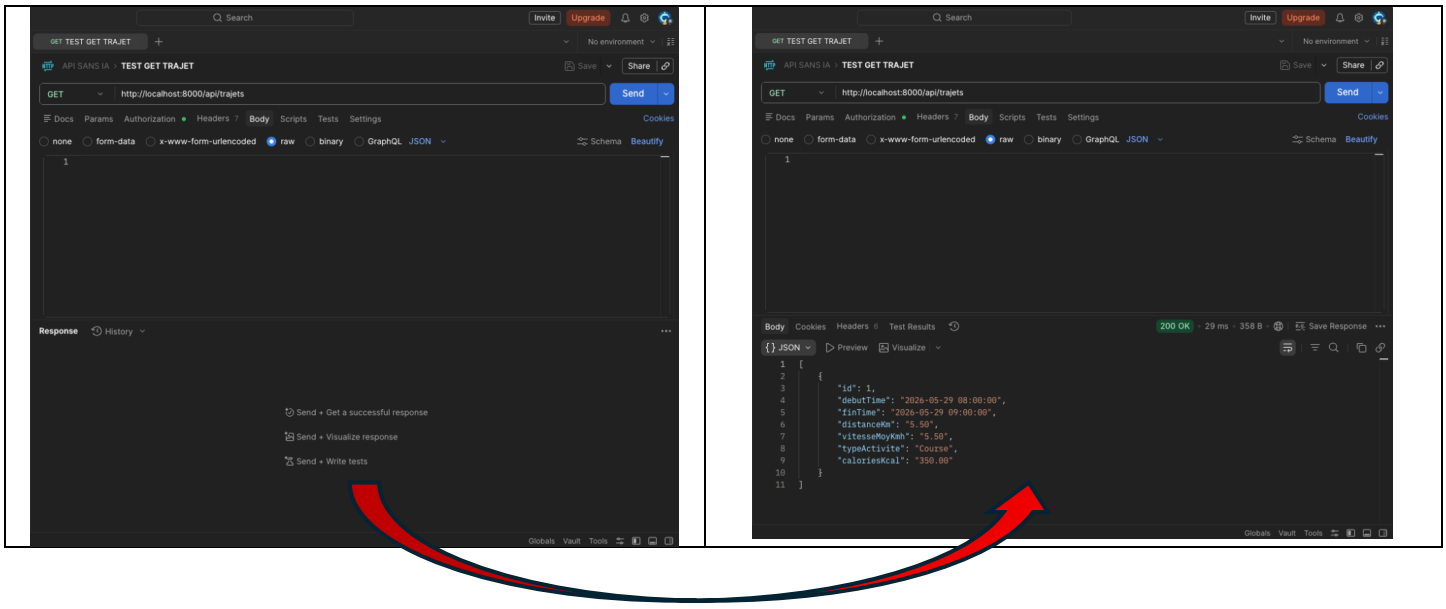
Schéma du code des fonctions :



Nous pouvons tester si cela fonctionne, en ajoutant le TOKEN que nous avons créé précédemment :



Maintenant testons la méthode GET en utilisant le Token précédent :



Position :

On va donc modifier le « PositionController » pour l'adapter à la demande des informations pour la base de données :

On ajoute les bonnes dépendance :	`use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;`
	`use Symfony\Component\Routing\Attribute\Route;`
	`use Symfony\Component\HttpFoundation\JsonResponse; //! Ajout de Js`
	`use Symfony\Component\HttpFoundation\Request; //! Ajout de Request`
	`use Doctrine\ORM\EntityManagerInterface; //! Ajout de EntityManage`
	`use App\Entity\Position; //! Ajout de l'entité Trajet pour manipul`

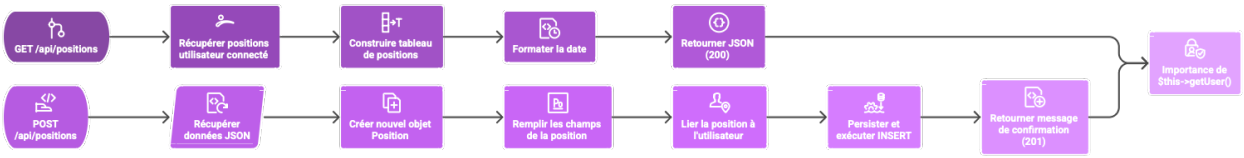
On crée la route de base	`// ! Modification de /position par /api pour indiquer que ce cont`
	`#[Route('/api')]`
	`final class PositionController extends AbstractController`

On crée la première méthode GET	`final class PositionController extends AbstractController`
	`{`
	`#[Route('/positions', name: 'api_positions_list', methods: ['GET'])] //! Route pour récupérer tous les`
	`public function getPositions(EntityManagerInterface $entityManager): JsonResponse //! Retour d'une rép`
	`{`
	`//! Récupération de tous les trajets de l'utilisateur actuellement connecté à partir de la base de`
	`$positions = $entityManager->getRepository(Position::class)->findBy(['user' => $this->getUser()]);`
	`$positionData = []; //! Initialisation d'un tableau pour stocker les données des trajets`
	`foreach ($positions as $position)`
	`{ //! Parcours de chaque trajet pour extraire les données nécessaires`
	`$positionData[] =`
	`[`
	`'id' => $position->getId(), //! Récupération de l'identifiant du trajet`
	`'latitude'=>$position->getLatitude(), //! Récupération de la latitude`
	`'longitude'=>$position->getLongitude(), //! Récupération de la longitude`
	`'precisionM'=>$position->getPrecisionM(), //! Récupération de la présicion`
	`'source'=>$position->getSource(), //! Récupération de la source`
	`'date'=>$position->getDate()?->format('Y-m-d H:i:s') //! Récupération de la date sous le f`
	`];`
	`}`
	`return new JsonResponse($positionData, JsonResponse::HTTP_OK); //! Retour d'une réponse JSON conte`
	`}`

On crée la deuxième méthode POST	

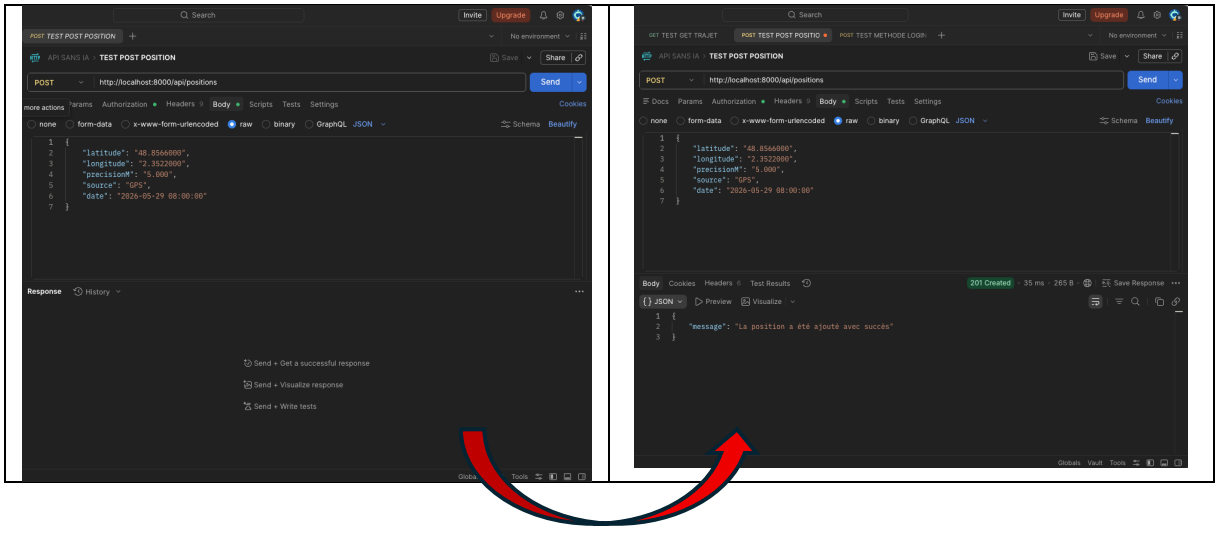
```php
    //! Liste des données récoltés pour ajout en Json et mise dans la BDD
{
    $data = json_decode($request->getContent(), true);

    $position = new Position(); //! Création d'une nouvelle instance de l'entité Trajet
    $position->setLatitude($data['latitude']); //! Définition de la latitude dans Postion à partir des données
    $position->setLongitude($data['longitude']); //! Définition de la longitude dans Postion à partir des donné
    $position->setPrecisionM($data['precisionM']); //! Définition de la présicion dans Position à partir des do
    $position->setSource($data['source']); //! Définition de la Source dans position à partir des données reçue
    $position->setDate(new \DateTime($data['date'])); //! Définition de la date
    $position->setUser($this->getUser()); //! Association du trajet à l'utilisateur actuellement connecté


    $entityManager->persist($position); //! Persistance de la position dans la base de données
    $entityManager->flush(); //! Exécution de la requête pour enregistrer le trajet

    return new JsonResponse(['message' => 'La position a été ajouté avec succès'], JsonResponse::HTTP_CREATED);
}
```

Schéma de ce que nous venons de faire :

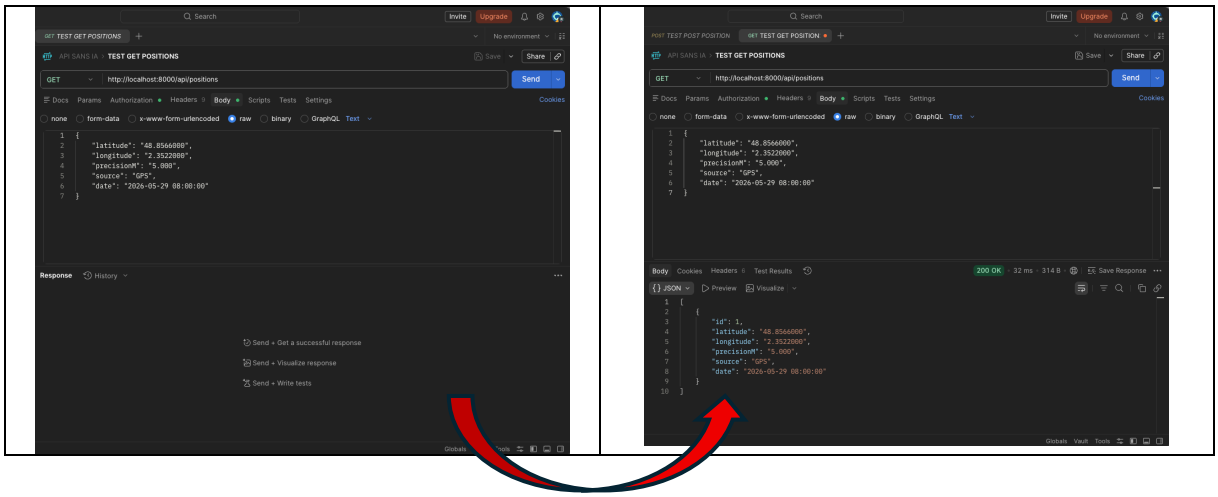


Une fois cela fait testons le sur POSTMAN :

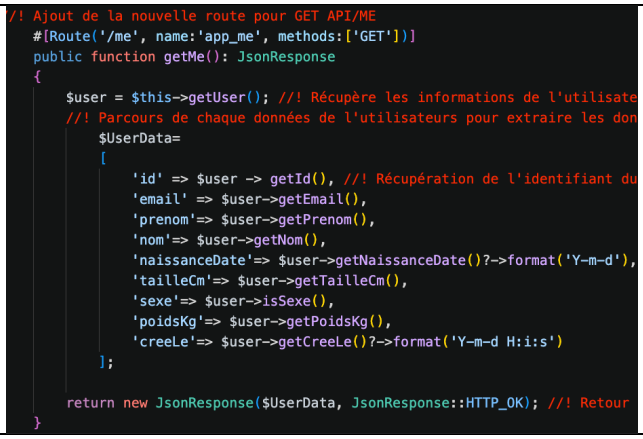
POST :



GET :

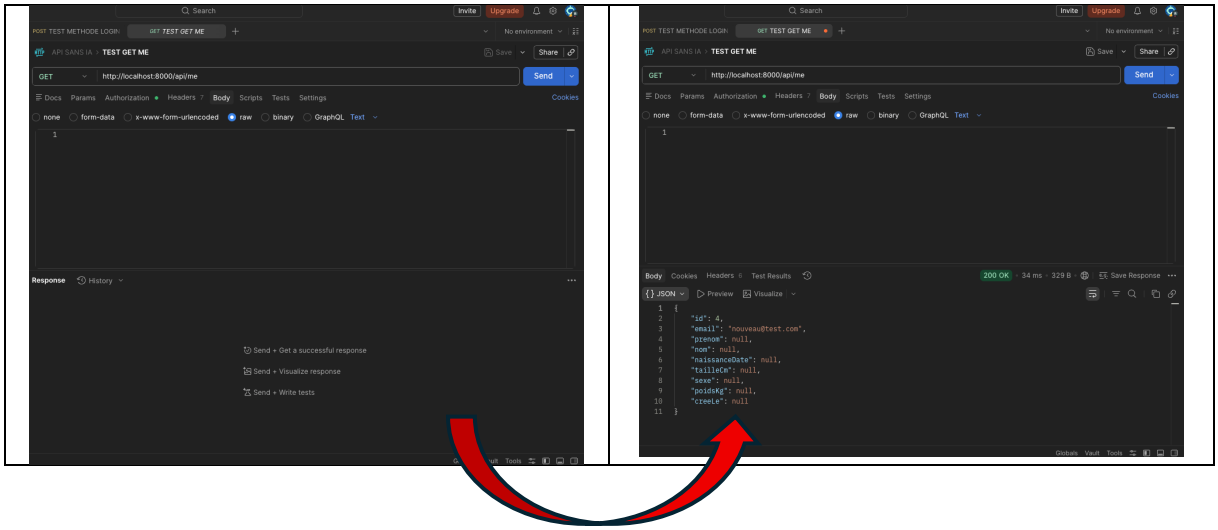


Nous pouvons à la suite de cela créer la méthode GETME() dans UserController:

Méthode qui ressemble à getTrajet() mais qui n'a pas de foreach dedans elle est donc plus simple, de plus elle récupère le Token de l'utilisateur connecté avec le « $user = $this->getUser() ; »	

```
/! Ajout de la nouvelle route pour GET API/ME
    #[Route('/me', name:'app_me', methods:['GET'])]
    public function getMe(): JsonResponse
    {
        $user = $this->getUser(); //! Récupère les informations de l'utilisate
        //! Parcours de chaque données de l'utilisateurs pour extraire les don
            $UserData=
            [
                'id' => $user -> getId(), //! Récupération de l'identifiant du
                'email' => $user->getEmail(),
                'prenom'=> $user->getPrenom(),
                'nom'=> $user->getNom(),
                'naissanceDate'=> $user->getNaissanceDate()?->format('Y-m-d'),
                'tailleCm'=> $user->getTailleCm(),
                'sexe'=> $user->isSexe(),
                'poidsKg'=> $user->getPoidsKg(),
                'creeLe'=> $user->getCreeLe()?->format('Y-m-d H:i:s')
            ];

        return new JsonResponse($UserData, JsonResponse::HTTP_OK); //! Retour
    }
```

Nous pouvons donc le tester via POSTMAN :

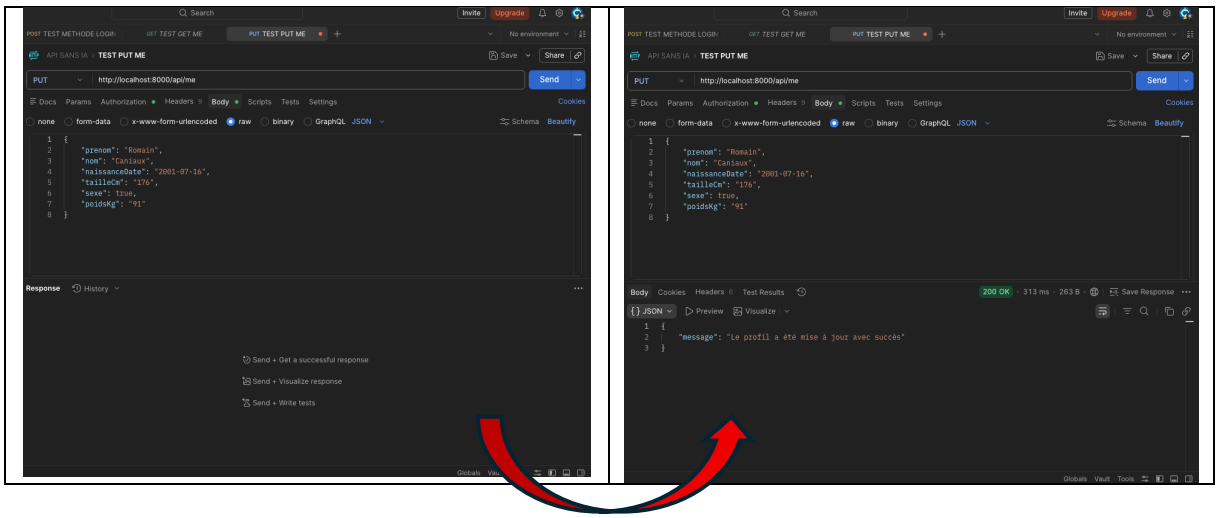


Passons à la méthode PUT :

Ajout d'une nouvelle fonction updateMe() qui permet de récupérer le Token de l'utilisateur pour mettre à jour ses données :

```php
//! Ajout de la route de la méthode PUT pour l'utilisateur'
#[Route('/me', name:'api_updateMe', methods:['PUT'])]
public function updateMe(Request $request, EntityManagerInterface $entityManager):JsonResponse //!

    //! Liste des données récoltés pour ajout en Json et mise dans la BDD
{
    $data = json_decode($request->getContent(), true);

    $user = $this->getUser(); //! Récupération de l'utilisateur connecté

    //! Mise à jour des données de l'utilisateur connecté
    $user->setPrenom($data['prenom']);
    $user->setNom($data['nom']);
    $user->setNaissanceDate(new \DateTime ($data['naissanceDate']));
    $user->setTailleCm($data['tailleCm']);
    $user->setSexe($data['sexe']);
    $user->setPoidsKg($data['poidsKg']);



    $entityManager->persist($user); //! Persistance de la position dans la base de données
    $entityManager->flush(); //! Exécution de la requête pour enregistrer le trajet

    return new JsonResponse(['message' => 'Le profil a été mise à jour avec succès'], JsonResponse::HTTP_OK);
```

Testons si cela fonctionne sur POSTMAN :



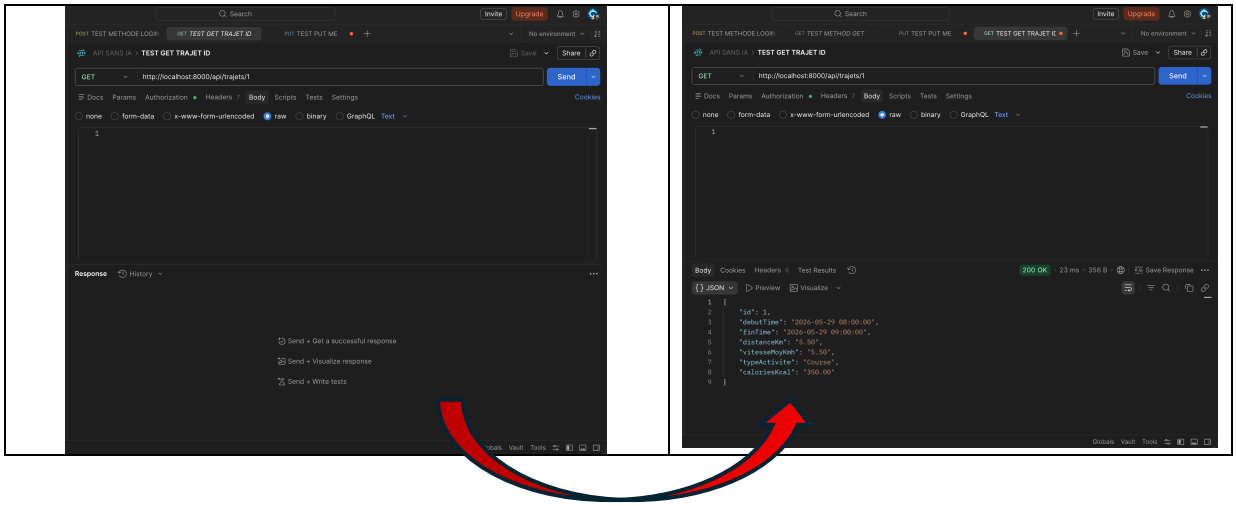
Passons à la méthode pour rechercher un trajet par son « ID »

On crée la méthode permettant de cibler le trajet en prenant en paramètre « {id} »

```php
//! Ajout de la route pour ciblé un trajet
#[Route ('/trajets/{id}', name:'api_trajet_show', methods:['GET'])]
public function getLeTrajet(int $id, EntityManagerInterface $entityManager): JsonResponse
{
    //! Récupération de l'id du trajet
    $trajet = $entityManager -> getRepository(Trajet::class) -> findOneBy(['id'=>$id]);

    //! On vérifie que le trajet existe dans la bdd avec un "SI"
    if(!$trajet) //! Si il est different de $trajet alors message disant qu'il n'existe pas
        {
            return new JsonResponse(['messsage'=>'Le trajet est introuvable'], JsonResponse::HTTP_NOT_FOUND);
        }
    if($trajet->getUser() !== $this->getUser()) //! Si l'id ne corresponds pas à l'utilisateur l'accès est refusé
        {
            return new JsonResponse(['message' => 'Vous ne pouvez pas y accedez'], JsonResponse::HTTP_FORBIDDEN);
        }
    return new JsonResponse
    (
    [
        'id' => $trajet -> getId(),
        'debutTime' => $trajet -> getDebutTime()?->format('Y-m-d H:i:s'),
        'finTime' => $trajet -> getFinTime()?->format('Y-m-d H:i:s'),
        'distanceKm'=> $trajet -> getDistanceKm(),
        'vitesseMoyKmh'=> $trajet -> getVitesseMoyKmh(),
        'typeActivite'=> $trajet -> getTypeActivite(),
        'caloriesKcal'=> $trajet -> getCaloriesKcal()
    ],JsonResponse::HTTP_OK

    );
```

Dans cette fonction on cherche à rechercher un trajet par son « id », on ajoute donc à la suite de $trajet des conditions « SI » afin de filtrer les résultats et obtenir que le trajet désiré, la particularité est que nous voulons que les id correspondant à l'utilisateur connecté.

Nous pouvons donc le tester avec POSTMAN :



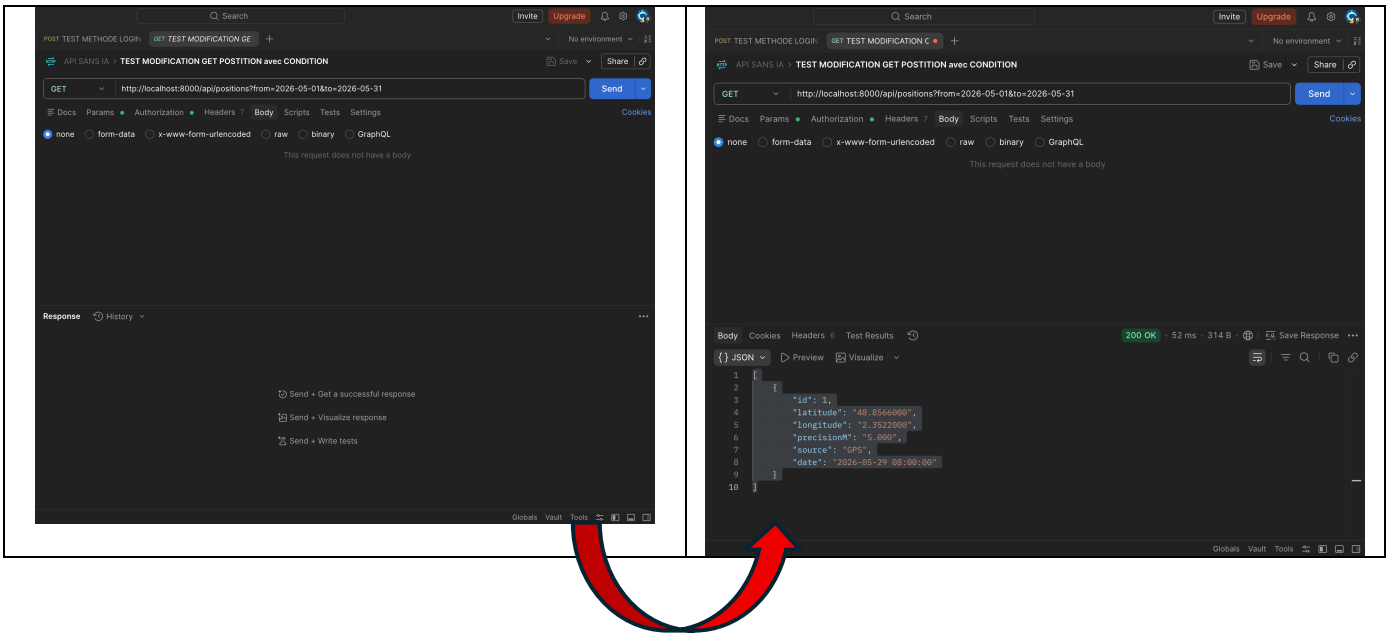
Nous allons maintenant modifier la méthode « getposition() » pour filtrer et donc répondre à la demande qui est d'afficher les données sous conditions :

Nous utilisons « createQueryBuilder » qui nous permet de récupérer les informations en utilisant l'équivalent d'une requête SQL

```php
final class PositionController extends AbstractController
{
    #[Route('/positions', name: 'api_positions_list', methods: ['GET'])] //! Route pour récupérer tous les
    public function getPositions(EntityManagerInterface $entityManager): JsonResponse //! Retour d'une rép
    {
        //! Récupération de tous les trajets de l'utilisateur actuellement connecté à partir de la base de
        $positions = $entityManager->getRepository(Position::class)->findBy(['user' => $this->getUser()]);
        $positionData = []; //! Initialisation d'un tableau pour stocker les données des trajets
        foreach ($positions as $position)
        { //! Parcours de chaque trajet pour extraire les données nécessaires
            $positionData[] =
            [
                'id' => $position->getId(), //! Récupération de l'identifiant du trajet
                'latitude'=>$position->getLatitude(), //! Récupération de la latitude
                'longitude'=>$position->getLongitude(), //! Récupération de la longitude
                'precisionM'=>$position->getPrecisionM(), //! Récupération de la présicion
                'source'=>$position->getSource(), //! Récupération de la source
                'date'=>$position->getDate()?->format('Y-m-d H:i:s') //! Récupération de la date sous le f
            ];
        }

        return new JsonResponse($positionData, JsonResponse::HTTP_OK); //! Retour d'une réponse JSON conte
    }
```

```php
#[Route('/positions', name: 'api_positions_list', methods: ['GET'])] //! Route pour récupérer
public function getPositions(Request $request, EntityManagerInterface $entityManager): JsonRes
{
    //! Recupération des valeurs pour filtrer avec "from" et "to"
    $from = $request -> query->get('from');
    $to=$request -> query->get('to');;

    //! Préparation d'une requête ditent "QueryBuilder" ce qui permet de recupèrer une donnée
    //! Cela revient à faire une requête SQL
    $QueryBuilder = $entityManager ->getRepository(Position::class) -> createQueryBuilder('p')
                    -> where('p.user = :user')
                    -> setParameter('user', $this -> getUser());

    //! Ajout des conditions pour filtrer avec le paramètre "$from"
    if($from)
        {
            $QueryBuilder -> andWhere('p.date >= :from')
                          -> setParameter('from', new\DateTime($from));
        }

    //! Ajout des conditions pour filtrer avec le paramètre "$to"
    if($to)
        {
            $QueryBuilder -> andWhere('p.date <= :to')
                          -> setParameter('to', new\DateTime($to));
        }

    $positions = $QueryBuilder -> getQuery() -> getResult();
```

Une fois cela fait testons la sur POSTMAN :



On va créer la dernière méthode rebuilderTrips(), qui permet d'avoir les positions GPS et de construire le trajet :

Elle permet de prendre la position de départ en fonction du temps et d'ensuite construire le trajet dans un tableau et de

```php
//! Nouvelle méthode et Route pour les positions GPS de l'utilisateurs
#[Route('/trips/rebuild', name: 'api_trips_rebuild', methods:['POST'])]
public function rebuilderTrips(EntityManagerInterface $entityManager) : JsonResponse
{
    //! Query Builder comme getPosition() mais en ajoutant un order by
    $positions = $entityManager -> getRepository(Position::class) -> createQueryBuilder('p')
                -> where ('p.user = :user')
                -> setParameter('user', $this -> getUser())
                -> orderBy('p.date','ASC')
                ->getQuery()
                ->getResult();

    //! Regrouppemet des positions en trajet
    $groupes = []; //! Tableau de groupe des positions
    $groupesActuel = []; //! Tableau du groupe en "construction"

    //! Boucle Foreach pour parcourir les positions
    foreach($positions as $position)
        {
            //! Vérification si la positions et "vide" si c'est oui alors on l'ajoute au groupeActuelle car poin
            if(empty($groupesActuel))
                {
                    $groupesActuel [] = $position;
                }
            //! On calcul ensuite l'écart entre les deux positions (en seconde)
            else
                {
                    $derniere = end($groupesActuel); //! Récupération de la dernière position
                    //! Soustraction entre les deux date pour obtenir l'écart en secondes
                    $ecart = $position ->getDate() -> getTimestamp() - $derniere -> getDate() -> getTimestamp();

                    //! Vérification de l'écart SI > à 5 minutes alors :
                if ($ecart > 300)
                    {
                        $groupes[] = $groupesActuel;
                        $groupesActuel = [$position];
                    }
                else //! Sinon on l'ajoute au groupe actuel
                    {
                        $groupesActuel[] = $position;
                    }
                }

        }

        //! SI il n'est pas vide alors on l'ajoute en dernier au groupe
        if(!empty($groupesActuel))
            {
                $groupes[] = $groupesActuel;
            }
```

Avant de terminer rebuilderTrip() nous devons ajouter une méthode « private », afin de finaliser notre fonction rebuilderTrip() :

Elle permet de retourner une distance en Km grâce à deux point GPS, de plus on utilise « atan2 » pour faire l'arc tangente ainsi que « sqrt » pour faire la racine carrée

```php
//* Ajout d'une nouvelle methode en private afin de calculer la distance entre les deux points GPS (en Km)
private function haversine(float $lat1, float $lon1, float $lat2, float $lon2):float
{
    //! Variable pour le rayon de la terre en Km soit 6371
    $RayonTerre = 6371;
    //! Calcul de la différence de latitude et longitude
    $diffLatitude = deg2rad($lat2 - $lat1);
    $diffLongitude = deg2rad($lon2 - $lon1);

    $carreDistanceAngulaire = sin($diffLatitude/2) * sin($diffLatitude/2) +
    cos(deg2rad($lat1)) * cos(deg2rad($lat2)) *
    sin($diffLongitude/2) * sin($diffLongitude/2);

    //! Calcul de l'angle central
    $angleCentral = 2 * atan2(sqrt($carreDistanceAngulaire), sqrt(1 - $carreDistanceAngulaire));

    //! Retourne le calcul final
    return $RayonTerre * $angleCentral;
}
```

Cette fonction permet de faire la Formule d'HARVERSINE :

$$a = \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1) \times \cos(\varphi_2) \times \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)$$

$$c = 2 \times \arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right)$$

$$d = R \times c$$

Une fois la méthode HARVERSINE(), on peut terminer la méthode rebuildTrips() :

A la suite de la ligne de la méthode rebuildTrips() :

```php
//! SI il n'est pas vide alors on l'
if(!empty($groupesActuel))
    {
        $groupes[] = $groupesActuel;
    }
```

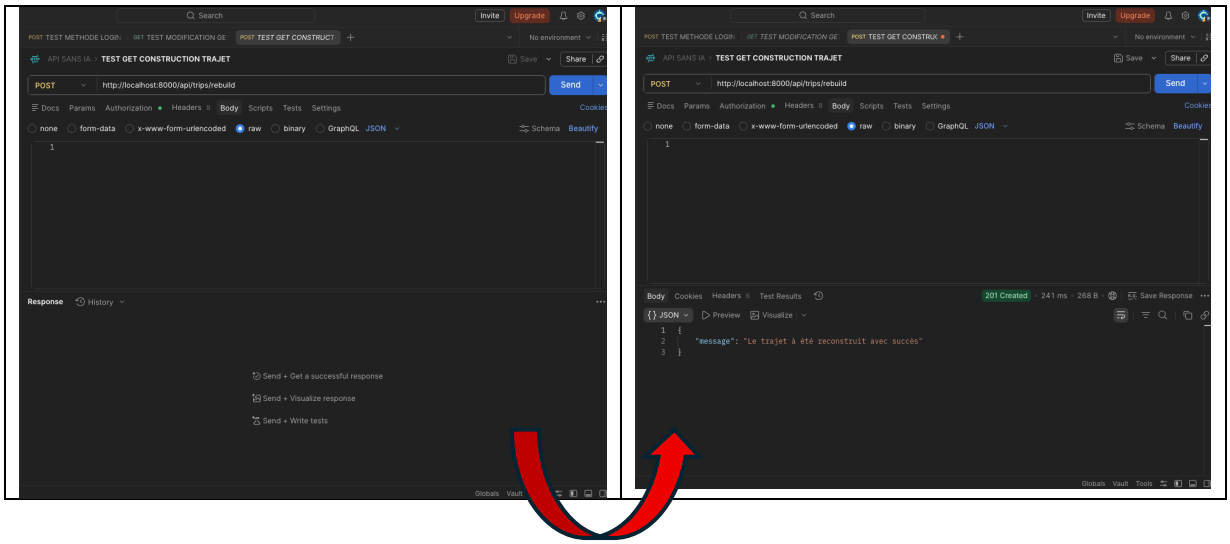
On ajoute le code suivant permettant de finir la méthode et retourner le trajet reconstruit.
Il permet de déterminer le MET via le type d'activité lui-même déterminé par la vitesse moyenne

```php
//! Vérification de de la validité du trajet
foreach($groupes as $groupe)
    {
        //! SI il n'y a pas deux points alors on passe au groupe suivant
        if(count($groupe) < 2) continue; //* "continue" permet de passer au groupe suivant

        //! Variable de début et fin de trajet
        $debut = $groupe[0] ->getDate();
        $fin = end($groupe) ->getDate();

        //!Calcul de la distance total du parcours et utilisation de la méthod haversine()
        $distanceTT = 0;
        for($i = 0; $i < count($groupe) - 1; $i++)
            {
                $distanceTT += $this -> haversine(
                    $groupe[$i] -> getLatitude(),
                    $groupe[$i] -> getLongitude(),
                    $groupe[$i+1] -> getLatitude(),
                    $groupe[$i+1] -> getLongitude()
                );
            }
        //! Convertion en heure
        $dureeHeures = ($fin -> getTimestamp() - $debut -> getTimestamp()) /3600;
        //! Calcul de la vitesse moyenne
        $vitesseMoy = $dureeHeures > 0 ? $distanceTT / $dureeHeures:0;

        //! En fonction de la vitesse moyenne afin de determiner le type d'activité
        if($vitesseMoy < 6)
            {
                $typeActivite = 'Marche';
                $MET = 3.5;
            }
        else if($vitesseMoy <= 12)
            {
                $typeActivite = 'course';
                $MET = 9;
            }
        else
            {
                $typeActivite = 'Vélo';
                $MET = 6;
            }

    //! calcul des calories avec la formule donnée MET x POIDS x DUREE
    $poids = $this -> getUser() -> getPoidsKg()?? 70; //! Si le poids n'est pas donnée alors
    $calories = $MET * $poids * $dureeHeures;
```

Une fois cela fait on sauvegarde les données et nous retournons un message confirmant la reconstruction du trajet

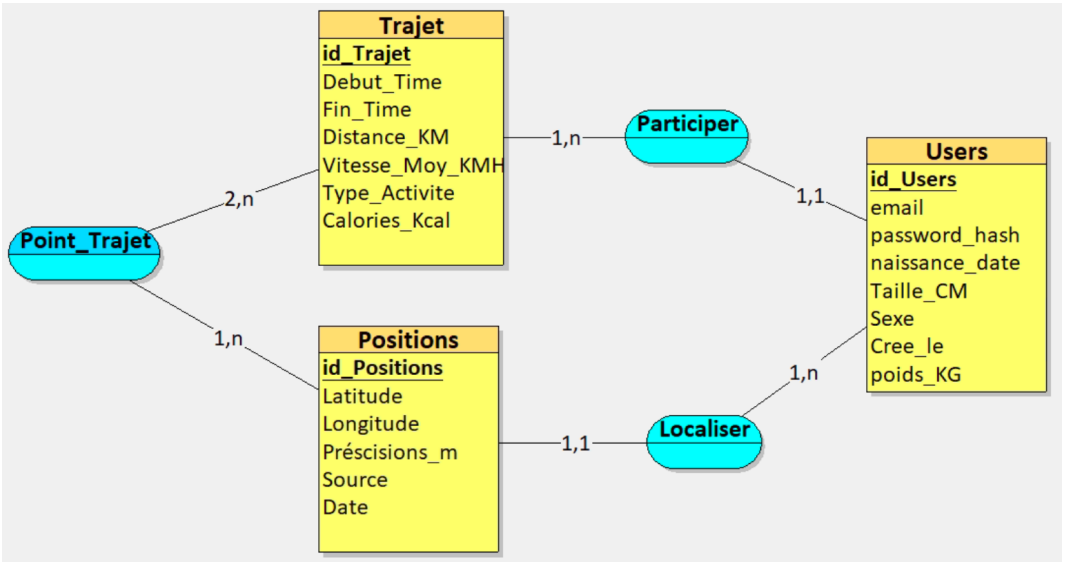
```php
//! On crée une sauvegarde du trajet
$trajet = new Trajet();
$trajet -> setDebutTime($debut);
$trajet -> setFinTime($fin);
$trajet -> setDistanceKm(round($distanceTT,2)); //! Ajoudis à deux chiffre
$trajet -> setVitesseMoyKmh(round($vitesseMoy,2));
$trajet -> setTypeActivite($typeActivite);
$trajet -> setCaloriesKcal(round($calories,2));
$trajet -> setUser($this -> getUser());
$entityManager -> persist($trajet);
}
//! On retourne la réponse que le trajet et reconstruis en entier
$entityManager -> flush();
return new JsonResponse(['message' => 'Le trajet à été reconstruit avec succès'], JsonResponse::HTTP_CREATED);
```

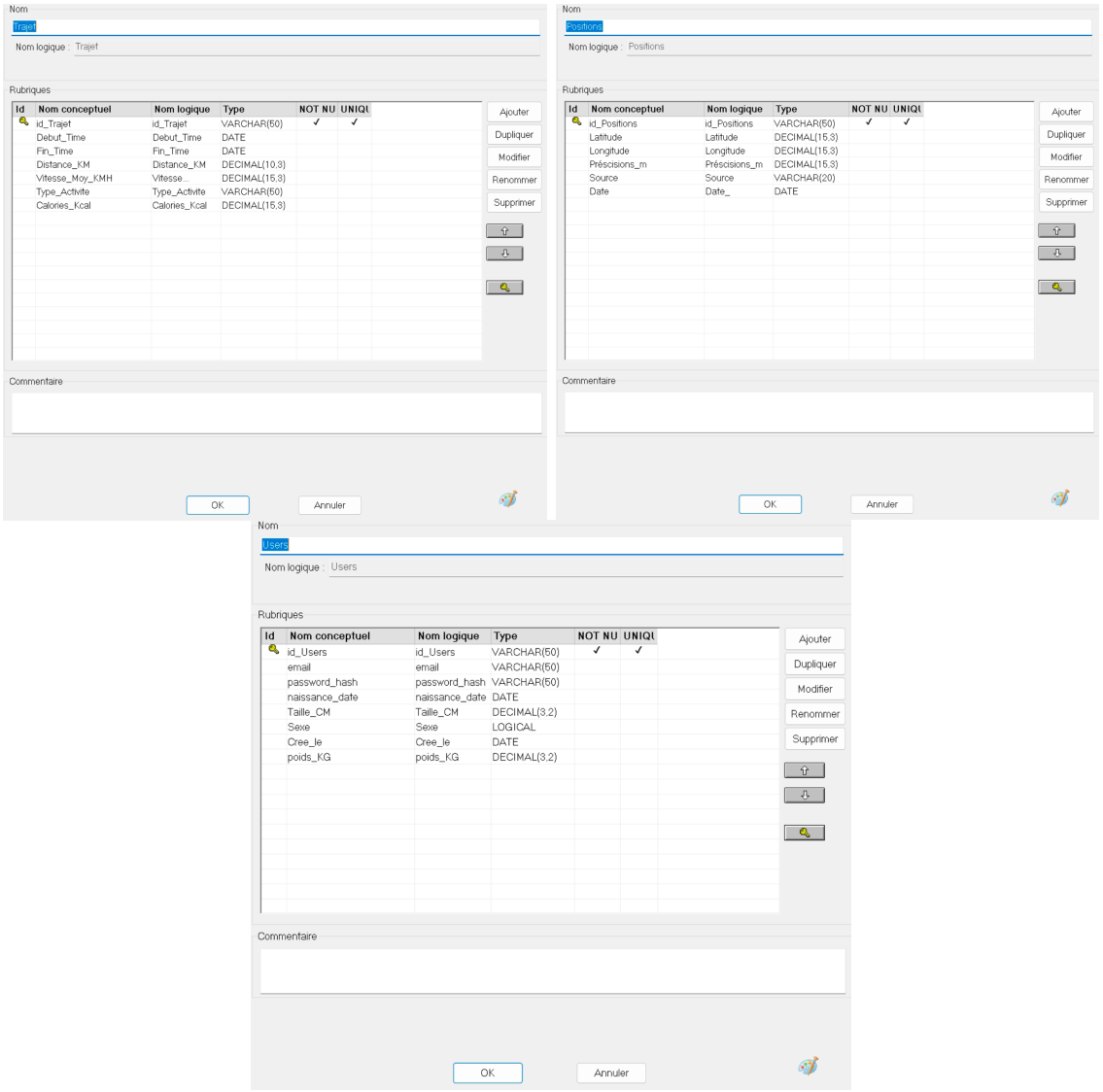
Nous pouvons donc tester la méthode via POSTMAN :



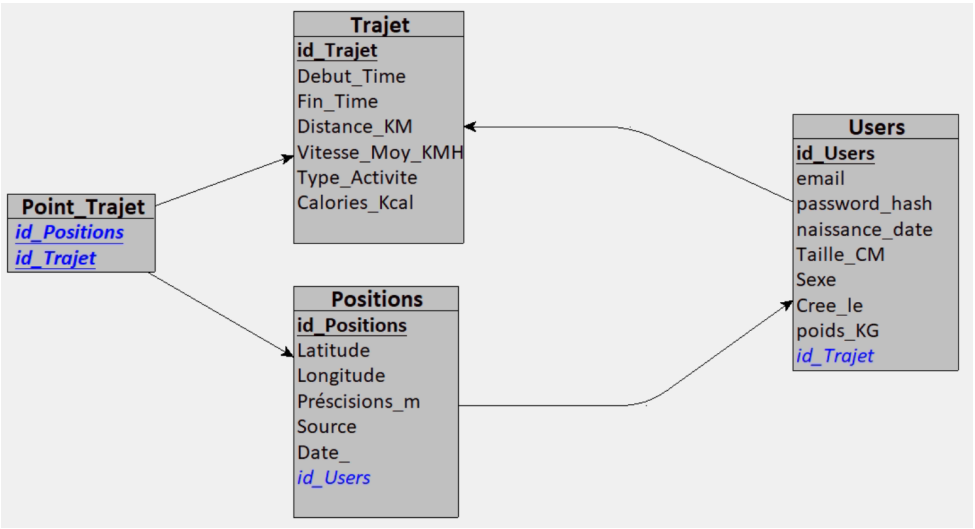
Annexe 2 — Documentation Base de données MySQL

Création du MCD via Looping





Création du MLD via Looping



Création du Script SQL

```
CREATE TABLE Trajet (
   id_Trajet       INT AUTO_INCREMENT,
   Debut_Time      DATETIME,
   Fin_Time        DATETIME,
   Distance_KM     DECIMAL(10,3),
   Vitesse_Moy_KMH DECIMAL(10,3),
   Type_Activite   VARCHAR(50),
   Calories_Kcal   DECIMAL(10,3),
   PRIMARY KEY (id_Trajet)
);

CREATE TABLE Users (
   id_Users        INT AUTO_INCREMENT,
   email           VARCHAR(100),
   password_hash   VARCHAR(255),
   naissance_date  DATE,
   Taille_CM       DECIMAL(5,2),
   Sexe            TINYINT(1),
   Cree_le         DATETIME,
   poids_KG        DECIMAL(5,2),
   id_Trajet       INT NOT NULL,
   PRIMARY KEY (id_Users),
   FOREIGN KEY (id_Trajet) REFERENCES Trajet(id_Trajet)
);

CREATE TABLE Positions (
   id_Positions  INT AUTO_INCREMENT,
   Latitude      DECIMAL(10,7),
   Longitude     DECIMAL(10,7),
   Precision_m   DECIMAL(10,3),
   Source        VARCHAR(20),
   Date_         DATETIME,
   id_Users      INT NOT NULL,
   PRIMARY KEY (id_Positions),
   FOREIGN KEY (id_Users) REFERENCES Users(id_Users)
);

CREATE TABLE Point_Trajet (
   id_Positions  INT,
   id_Trajet     INT,
   PRIMARY KEY (id_Positions, id_Trajet),
   FOREIGN KEY (id_Positions) REFERENCES Positions(id_Positions),
   FOREIGN KEY (id_Trajet)    REFERENCES Trajet(id_Trajet)
);
```