API REST Covoiturage

Date de mise à jour : 01/03/2023
Version : 0.5 alpha

Nom	DEBUSSCHER Camille	**Numéro d'auditeur**	

Table des matières

1. Objectif du document

L'objectif de ce document est de décrire la conception d'une API REST pour une application de covoiturage, réalisée avec Symfony. Cette API permettra aux utilisateurs de rechercher, réserver et annuler des trajets de covoiturage. Les fonctionnalités de l'API seront décrites en détail, ainsi que les routes et les paramètres nécessaires pour utiliser l'API.

Fonctionnalités :
L'API fournira les fonctionnalités suivantes :

- Recherche de trajets : Les utilisateurs pourront rechercher des trajets de covoiturage en spécifiant leur point de départ et leur destination, ainsi que la date souhaitée.
- Réservation de trajets : Les utilisateurs pourront réserver des trajets en indiquant les informations de leur trajet, telles que le point de départ et la destination, ainsi que les informations du conducteur et les détails du trajet.
- Annulation de réservation : Les utilisateurs pourront annuler leur réservation de trajet à tout moment avant le départ du trajet.

2. Architecture

2.1.Contraintes techniques

Les contraintes architecturales sont les caractéristiques qu'une architecture doit respecter pour être considérée comme RESTful. Voici les 6 contraintes : Stateless (sans état), modèle client-serveur, mise en cache, interface uniforme, modèle de système multicouche et code à la demande (facultatif). Ces contraintes sont nécessaires pour que l'architecture puisse répondre aux règles spécifiques qui forment la base de REST et pour garantir une communication uniforme entre le client et le serveur, avec des ressources identifiables, représentables et auto-descriptives.

2.2.Packages et dépendances

Crée avec symfony en skeleton afin d'avoir les essentiels à l'utilisation de ce projet.

Quelques dépendances essentielles au projet :
- symfony/orm : Permet de faire les requêtes avec Doctrine
- doctrine/doctrine-bundle, doctrine/doctrine-migrations-bundle : Permet de faire les migrations qui génèrent les tables par rapport aux entités créées dans Symfony.

2.3.Déploiement

Le déploiement est fait en local, pour des raisons techniques.

3. Technologies utilisées

3.1.Serveur web

Le serveur web utilisé est Apache directement avec la ligne de commande :

- symfony server:start

3.2.Stockage des données

Pour le stockage des données, MariaDB sous une image docker.
Pour lancer l'image docker :

- docker-compose up -d

Il y a également une image docker en phpmyadmin.

3.3.Couche de persistance

Le stockage des données est géré en DQL (Doctrine Query Language), c'est à dire en utilisant Doctrine qui est un ORM (Object Relationnel Mapper). C'est une bibliothèque PHP crée par SensioLabs pour le framework Symfony.

3.4.Couche métier

Pour définir la couche métier, il y a 6 entités qui définissent les informations que va recevoir la base de données :

- Marque (voiture) :

Définit les marques de voiture afin d'éviter les répétitions et de proposer un large choix de marque de voiture.

- Nom : Nom de la marque

- Personne (informations utilisateur) :

Définit les informations d'un utilisateur en temps que personne.

- Nom : Nom de la personne
- Prenom : Prenom de la personne
- Tel : Son numéro de téléphone
- Email : Adresse mail pour la contacter

- Trajet :

Définit les informations sur les trajets crées par des conducteurs pour le covoiturage.

- kms: Nombre de kilomètres estimés par le conducteur
- Heure de départ
- Heure d'arrivée
- Nombre de places disponibles
- Ville de départ
- Ville d'arrivée

- Utilisateur :

Définit les informations nécessaires à la connexion sur l'application et l'utilisation de celle-ci.

- nom : aurait pu également s'appeler pseudo
- password : mot de passe afin de se connecter
- token_api : token qui permet d'accéder à certaines sections de l'application
- roles : pour savoir le accès auxquels vous avez accès

- Ville :

Les informations des différentes villes en France.

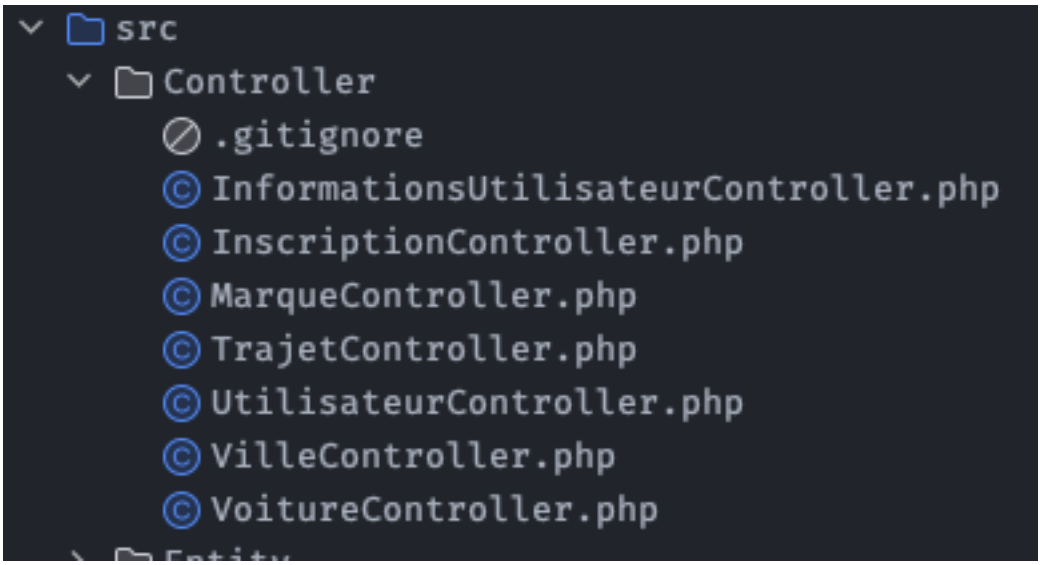
- nom : nom de la ville
- code postal : identifiant français
- latitude
- longitude

- Voiture :

Les informations sur les différentes voitures des personnes afin que les possibles passagers puissent repérer facilement la voiture.

- immatriculation
- couleur : permet l'identification rapide du véhicule sur le lieu de rdv
- places : nombre de places maximum dans le véhicule
- conducteur : lien direct avec la personne qui enregistre le véhicule
- marque : qui fait lien avec l'entité marque pour récupérer les informations des marques.

3.5.Couche service



La couche service est définit par les Controllers qui gèrent la partie API REST.

- InformationsUtilisateurController :

Permet l'affichage et la gestion de l'entité Personne.

- insertionPersonne() : POST / permet l'ajout d'une personne, on peut également dans le cas où il souhaite être conducteur ajouter directement une voiture
- listePersonnes(): GET / affiche les personnes enregistrées
- listeUnePersonne(): GET / affiche une personne par rapport à son id
- miseajourPersonne(): PUT / met à jour une personne par rapport à son id
- supprimerPersonne(): DELETE / supprime une personne par rapport à son id

- InscriptionController :

Permet l'ajout à des trajets et l'affichage de ses trajets.

- listeInscription() : GET / affiche toute les références trajets et personnes liées.
- insertionInscription(): POST / inscription à un trajet en temps que personne

- MarqueController :

Permet l'affichage des marques.

- listeMarques(): GET / affiche toutes les marques

- TrajetController :

Permet l'affichage et la gestion des trajets.

- insertionTrajet(): POST / création d'un trajet
- listeTrajet(): GET / affichage des trajets
- rechercheTrajet(): GET / affichage des trajets en lien avec les villes et la date
- supprimeTrajet(): DELETE / suppression d'un trajet

- UtilisateurController

Permet l'ajout d'un utilisateur qui donne l'accès à l'application

- creationCompte(): POST / création d'un compte avec un login et mot de passe
- authentification(): POST / se connecter à l'application (login et mot de passe)
- deconnexion(): POST / se déconnecter de l'application

- VilleController :

Permet l'affichage des villes.

- listeVilles(): GET / affiche toutes les villes

- VoitureController

Permet l'affichage et la gestion d'une voiture.

- insertionVoiture(): POST / enregistrement d'une voiture
- listeVoiture(): GET / affiche toutes les voitures
- supprimerVoiture(): DELETE / supprime une voiture par rapport à son id

3.6. Couche présentation

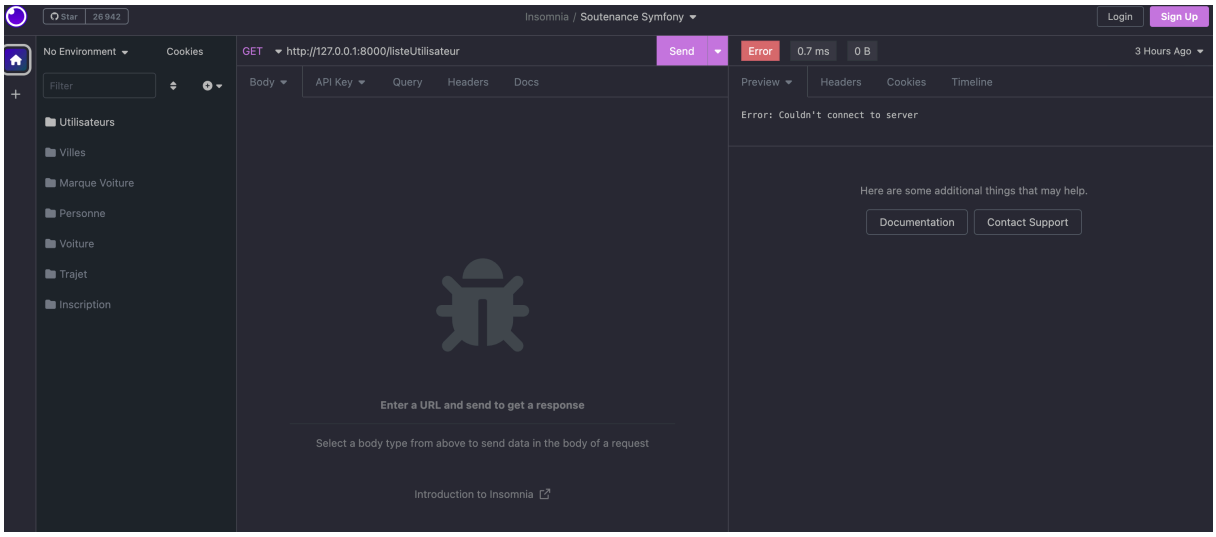
La couche présentation sera réalisée dans un deuxième temps.

3.7. Environnement de développement

Utilisation de PHPStorm v 2022.3.2

3.8. Test fonctionnel

Tous les tests fonctionnels ont été effectués sous Insomnia v2022.7.5

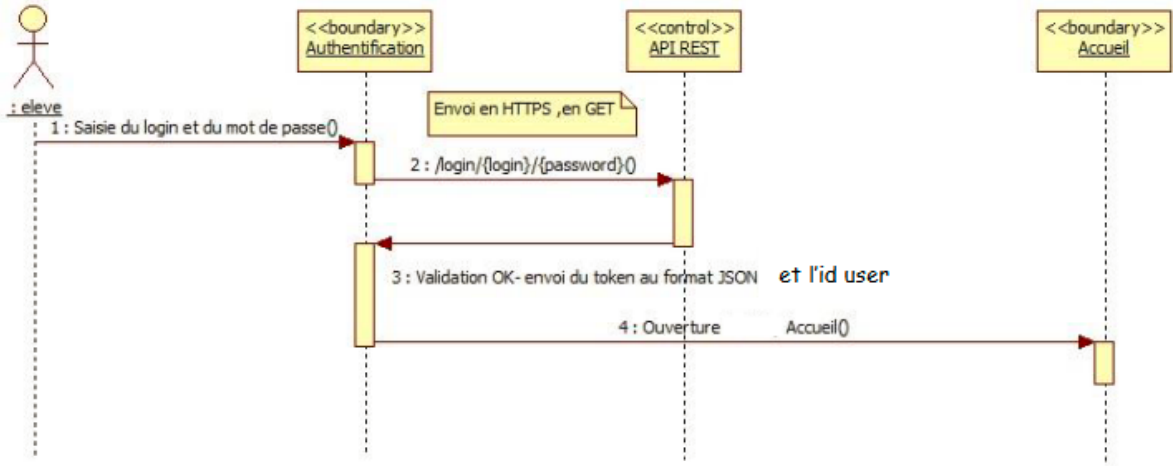


4. Cas d’utilisation

4.1.Login

4.1.1. Liste des objets candidats

- <<Entity>> : User
- <<Boundary>> : Authentification
- <<Control>> : API REST
- <<Boundary>> : Accueil



6. Annexes

6.1.Terminologie

Entity	structure composée de propriétés-types représentant un ensemble d'entités, c'est-à-dire un ensemble de composants identifiables dans un domaine fonctionnel.
Workflow	Série d''étapes liées au traitement des données. En effet, un workflow est la modélisation des processus métiers et la gestion des processus métiers.
Boundary	Encapsulation l'interaction avec des acteurs externes (utilisateurs humains ou systèmes externes)
Lifecycle	Séquence d'étapes que traverse une unité de données particulière depuis sa génération ou sa capture initiale jusqu'à son archivage et/ou sa suppression éventuelle à la fin de sa vie utile.
API REST	Système permettant d'agir sur une application back afin de communiquer sur sa base de données et agir via le protocole HTTP. REST est une normalisation comme peut l'être le CRUD.

6.2.Autre annexes