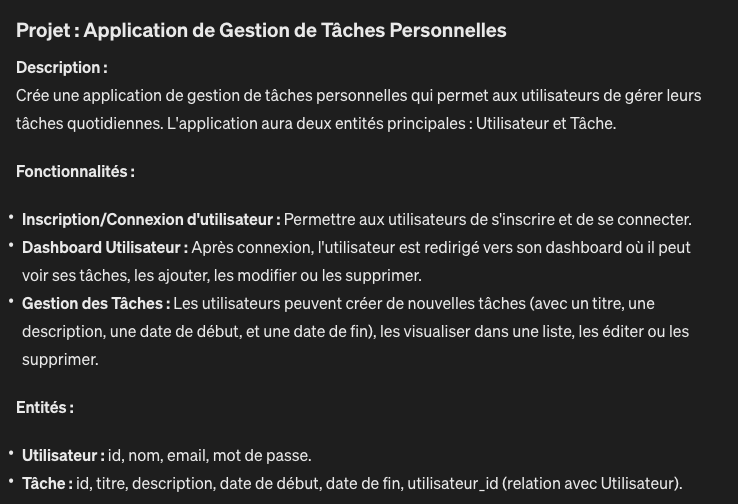
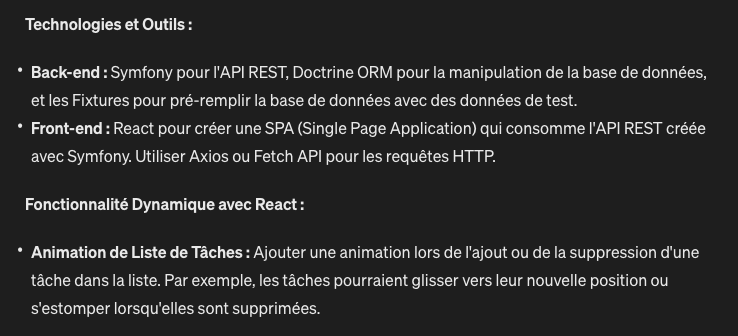
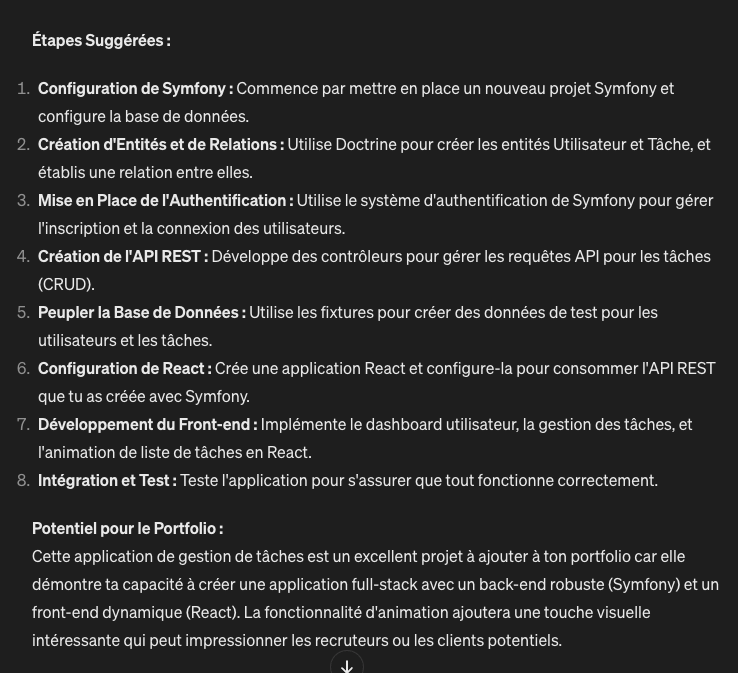
**TashPistache**







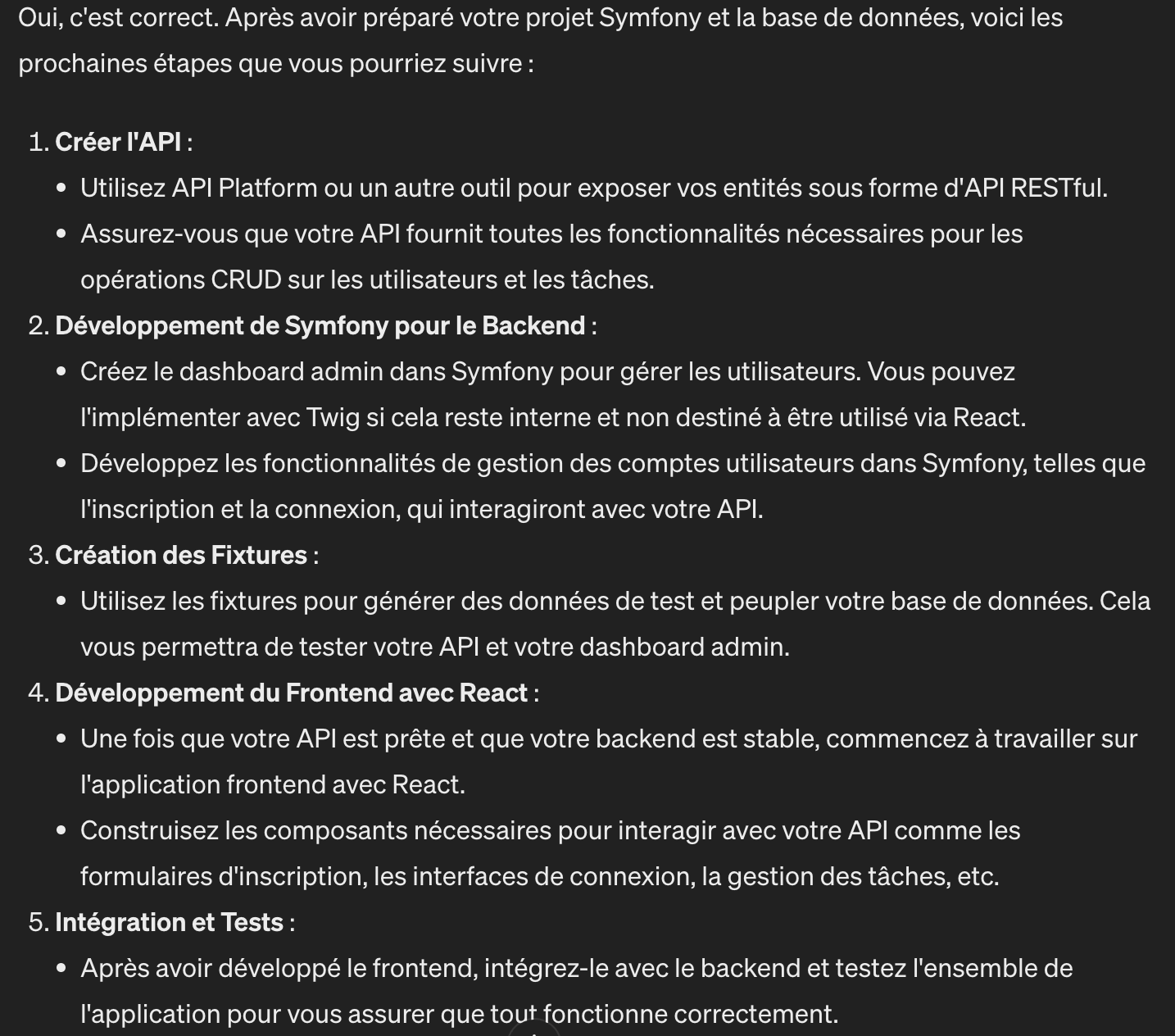
Donc j’ai :

* Créé mon projet Symfony avec symfony new --webapp my\_project
* Créé mon repo GitHub et configuré mon .gitignore
* Installé Composer avec composer install
* Configuré mon fichier .env en commentant/décommentant les lignes nécessaires à la BDD ( <https://www.youtube.com/watch?v=5rhRDv6eAP4> )
* Créé un utilisateur tashpistacheadmin sur PhpMyAdmin en modifiant la ligne dans .env avec les infos de connexion. Attention, s’il y a des caracteres spéciaux dans le MDP il faut les encoder.
* Ensuite, j’ai créé mes entités (= modèles = classes) (avec une majuscule et au singulier)

*Lorsque vous utilisez la commande make:entity de Symfony pour créer une entité, Symfony gère automatiquement la création de la clé primaire pour vous. Par défaut, une propriété id sera ajoutée à votre entité, qui sera utilisée comme clé primaire auto-incrémentée dans la base de données.*

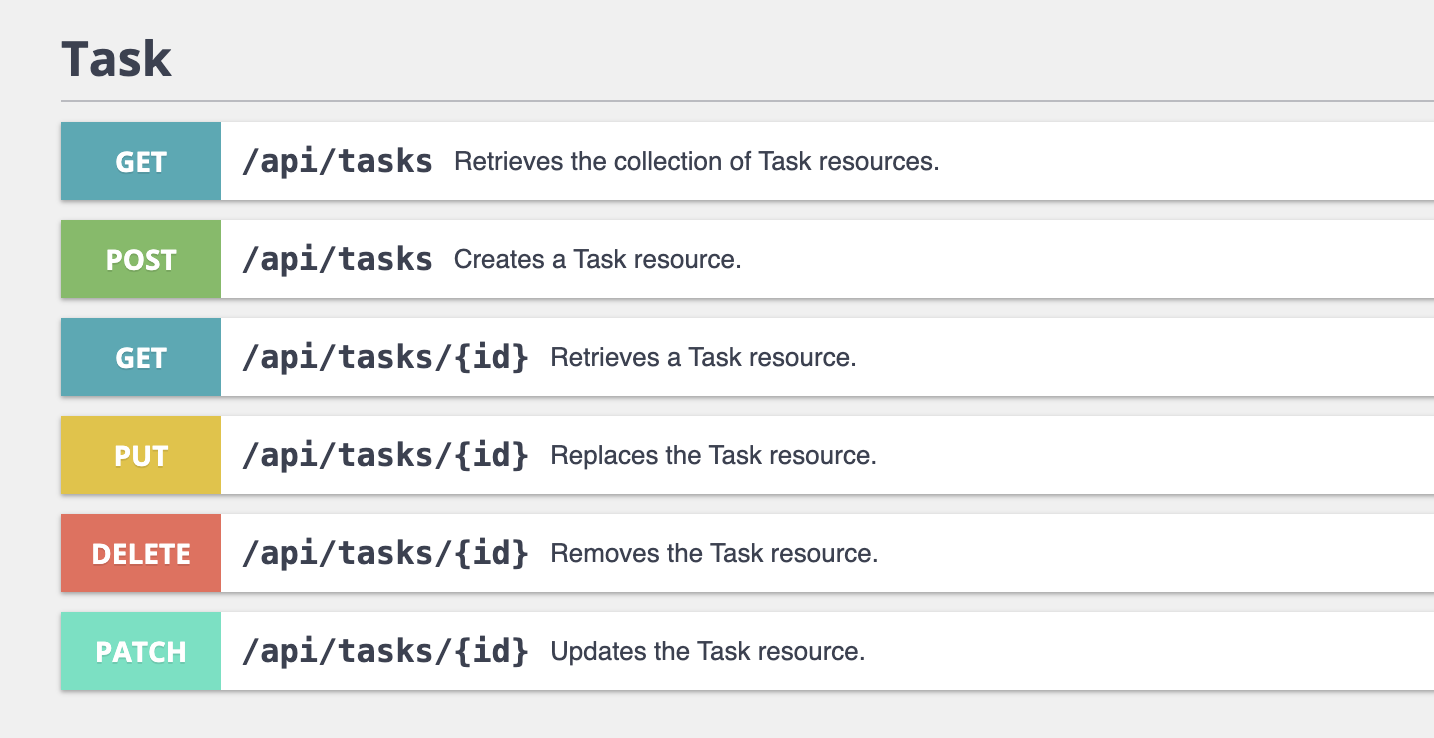
* J’ai lié les entités entre-elles en ajoutant la clé étrangère (donc en créant la colonne de la clé étrangère, sans la préfixer de id\_).[[1]](#footnote-1)
* Ensuite, j’ai fait la migration de la BDD, donc c’est la 1ere cela a créé les tables et leurs colonnes dans PhpMyAdmin.

Voici la suite à réaliser une fois toutes ces étapes accomplies :



Donc j’ai :

* Installé API Platform[[2]](#footnote-2), on s’y rend via le lien <http://127.0.0.1:8000/api>.
* Créé les points d’entrées qui permettent d’interagir avec l’API, en ajoutant la ligne #[ApiResource][[3]](#footnote-3) dans les entités, ce qui génère les opérations CRUD automatiquement :



* + GET /api/tasks: C'est une opération de lecture qui permet de récupérer la collection de toutes les ressources Task disponibles.
  + POST /api/tasks: C'est une opération de création qui permet d'ajouter une nouvelle ressource Task à la collection. Vous devez envoyer les données de la tâche (comme le titre et la description) en tant que partie du corps de la requête HTTP.
  + GET /api/tasks/{id}: C'est une opération de lecture pour une ressource Task spécifique. En remplaçant {id} par l'identifiant réel d'une tâche, vous pouvez obtenir les détails de cette tâche en particulier.
  + PUT /api/tasks/{id}: C'est une opération de mise à jour qui permet de remplacer une ressource Task existante par de nouvelles données. Cela écrase toutes les données précédentes de la tâche spécifiée par l'identifiant.
  + DELETE /api/tasks/{id}: C'est une opération de suppression qui permet de supprimer une ressource Task spécifiée par son identifiant.
  + PATCH /api/tasks/{id}: C'est également une opération de mise à jour, mais contrairement à PUT, PATCH permet de mettre à jour partiellement une ressource Task. Vous pouvez envoyer seulement les champs que vous souhaitez modifier.
* Utilisé le JWT pour identifier l’utilisateur plutôt que d’avoir à l’identifier via la BDD, pour qu’un utilisateur ne puisse avoir accès qu’à ses tâches à lui.

Il faut installer les composants nécessaires avec Composer :

* + composer require symfony/security-bundle =

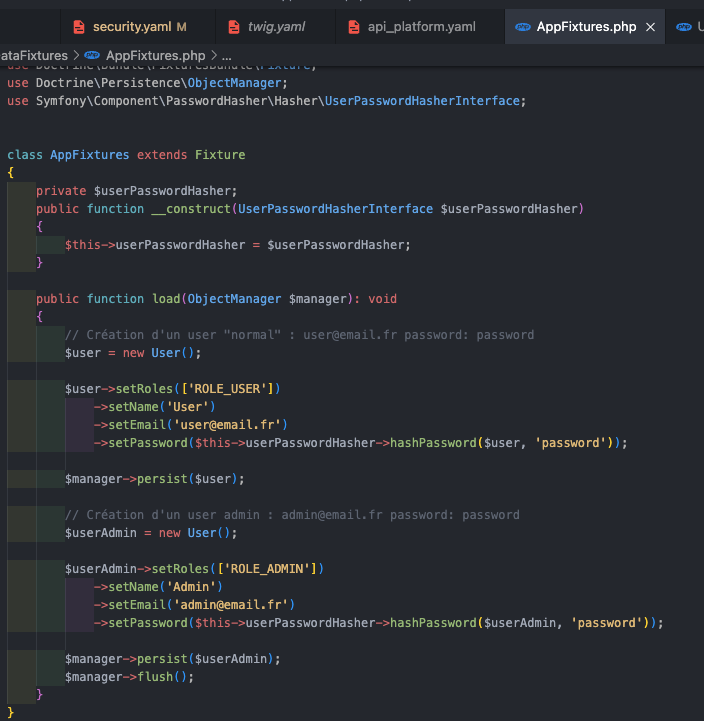
Composant essentiel pour s’occuper de la sécurité de l’appli (gère l’identification des utilisateurs, détermine leurs droits, protège contre les failles de sécurités comme XSS/CSRF, hache les MDP…)

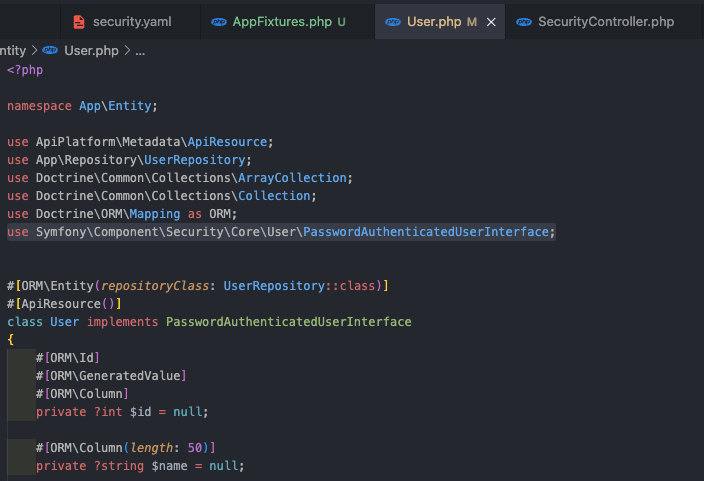
* + composer require lexik/jwt-authentication-bundle (voir lien annotation 5) =

L'utilisation de JWT est une méthode courante pour sécuriser les API car elle permet d'échanger des informations de manière sécurisée entre le client et le serveur.

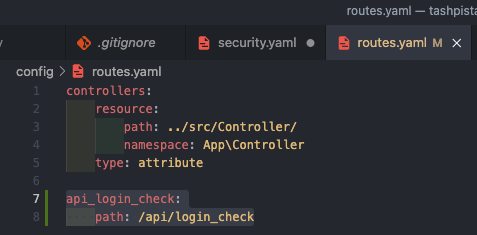
JWT est souvent utilisé dans des applications où vous voulez une authentification sans état (stateless). Cela signifie que le serveur ne conserve pas d'état de session pour les utilisateurs entre les requêtes. Au lieu de cela, chaque requête du client contient le token JWT nécessaire pour s'authentifier (=> pas besoin de cookies, sessions)

* Généré les clés privée et publique avec php bin/console lexik:jwt:generate-keypair puis configuré le fichier security.yaml [[4]](#footnote-4).
* Créé les fixtures des Users avec composer require orm-fixtures --dev puis symfony console make:fixtures . il faut créer l’objet, nourrir les setters et hacher les MDP[[5]](#footnote-5), pour cela j’ai utilisé composer require symfony/password-hasher et implémenté la méthode magique dans AppFixtures & modifié User :





* Créé les rôles si ce n’est pas déjà fait. En fait, il faut dès le départ utiliser make:user pour avoir la gestion des roles et du hachage du password intégré.
* Chargé les fixtures pour entrer les utilisateurs en BDD.
* Pour la prochaine fois :
  + vérifier que le JWT est OK avec le tuto OpenClassrooms, sachant qu’on y explique aussi comment faire les vérifications avec Postman ( <https://openclassrooms.com/fr/courses/7709361-construisez-une-api-rest-avec-symfony/7795148-authentifiez-et-autorisez-les-utilisateurs-de-l-api-avec-jwt> )
  + vérifier que la sécurité est OK ( <https://openclassrooms.com/fr/courses/7709361-construisez-une-api-rest-avec-symfony/7795190-creez-une-api-avec-api-platform> )
  + voir la vidéo de grafikart pour faire un authenticator et le configurer ( <https://grafikart.fr/tutoriels/symfony-api-authenticator-2196#autoplay> )
* Modifié le fichier routes.yaml



1. https://symfony.com/doc/current/the-fast-track/fr/8-doctrine.html [↑](#footnote-ref-1)
2. https://symfony.com/doc/6.2/the-fast-track/fr/26-api.html [↑](#footnote-ref-2)
3. https://api-platform.com/docs/core/operations/ [↑](#footnote-ref-3)
4. https://youtu.be/XPXrNI-fux4 [↑](#footnote-ref-4)
5. https://openclassrooms.com/fr/courses/7709361-construisez-une-api-rest-avec-symfony/7795148-authentifiez-et-autorisez-les-utilisateurs-de-l-api-avec-jwt [↑](#footnote-ref-5)