Stage assistant ingénieur

03 septembre 2018 au 15 février 2019

Développement d'une API First

Étudiant :
Iker TARDIO

Semestre :
Automne 2018

Tuteur :
Florent Cholet

Suiveur :
Indira Thouvenin

Remerciements

Je tiens à remercier Vincent BENOIS, Frédéric BENOIS ainsi que Alban TEXIER, pour m'avoir permis d'effectuer mon stage d'assistant ingénieur au sein de l'entreprise Je Pilote.

Je remercie tout particulièrement mon responsable de stage Florent CHOLET pour l'attention et les nombreux conseils qu'il a m'a porté tout au long du stage.

Je remercie également les stagiaires que j'ai côtoyés ainsi que l'ensemble du personnel de Je pilote pour leur accueil chaleureux lors de mon arrivée et leur bonne humeur au quotidien, ce qui a permis de rendre mon stage agréable.

Table des matières

Introduction

Actuellement en 4ème année à l'université de technologie de Compiègne, il nous est demandé de réaliser un stage d'assistant ingénieur dans le domaine de l'informatique afin de découvrir le milieu professionnel dans ce domaine. J'ai eu l'opportunité d'effectuer ce stage d'assistant ingénieur avec l'équipe développement informatique de Je Pilote, une Start-Up experte dans les domaines de la comptabilité et de la gestion commerciale en ligne.

Dans un premier temps ma tâche consistait à découvrir et m'approprier deux Frameworks : Symfony et API-Platform. Dans un second temps j'ai utilisé ces Frameworks afin de développer une nouvelle API pour JePilote.

Dans ce rapport je vais présenter l'entreprise Je Pilote, ma mission et enfin le travail que j'ai réalisé.

1 Je Pilote

1.1 Présentation générale

La société « Je Pilote mon entreprise » a été fondée en 2014 par deux experts comptables, Vincent Benois et Alban Texier. Désormais abrégée en Je Pilote, l'entreprise a mis en place un logiciel en ligne qui permet de facilité la comptabilité et la gestion d'entreprise pour les auto-entrepreneurs, les Start-Up et les TPE [1]. Le logiciel Je Pilote est gratuit pour les utilisateurs afin de permettre à ces derniers d'accélérer leur passage vers la gestion comptable numérique.

Le logiciel Je Pilote a pour fonctionnalités majeurs :

- Comptabilité intuitive et experte (pour experts comptables)
- Intégration et consultation des données sociales de son entreprise
- Gestion électronique de documents
- Intégration automatique des relevés bancaires
- Des devis et facturations automatiques et personnalisable
- Des tableaux de bords de gestion
- Préparation de la déclaration de TVA

La spécificité de Je pilote est la mise en relation des utilisateurs avec des experts comptables. En effet bien que le logiciel soit gratuit, il est possible aux utilisateurs de faire appel à des experts comptable partenaire de Je Pilote qui seront eux rémunéré par les utilisateurs. En contre partie les réseaux de cabinets d'expertise-comptable contribue financièrement au développement de Je Pilote ce qui représente le modèle économique de Je Pilote.
Ces partenariats repose sur la vente des licences, qui sont les dossiers des utilisateurs de Je Pilote souhaitant être suivi par un expert-comptable, qui donneront lieu à un forfait mensuel pour l'expert-comptable. De plus Je Pilote propose également ce logiciel directement aux experts-comptables, en marque blanche marque blanche [2] . Ici aussi, c'est le nombre de dossiers utilisateurs Je Pilote traités par le cabinet qui modèlent le montant forfaitaire mensuel.

Aujourd'hui il y a plus de 80 cabinets d'expertise-comptable numérisés chez Je Pilote et ACOM audit qui est un réseau d'agences reparties sur l'ensemble du grand sud-ouest en est le partenaire principal. Il y a aussi à ce jour plus de 20.000 utilisateurs inscrits chez Je Pilote et il y a en moyenne plus de 5000 connexions différentes mensuels.

1. Très petites entreprises.
2. Il s'agit d'un mécanisme commercial de mise à disposition d'outils ou de produits, sans citer la marque.

1.2 Structure

La société Je Pilote basée à Bordeaux, en Nouvelle Aquitaine se compose de 4 salariés permanents. Vincent Benois et Alban Texier les fondateurs de Je Pilote, ainsi que Frédéric Benois et Florent Cholet.
Cependant une partie du développement informatique de la société est pour le moment réalisée en Ukraine par des prestataires.
Lors de mon stage j'ai également côtoyé deux étudiants en informatique de EPITECH Bordeaux, Guillaume Lenoir et Morgan Simala.

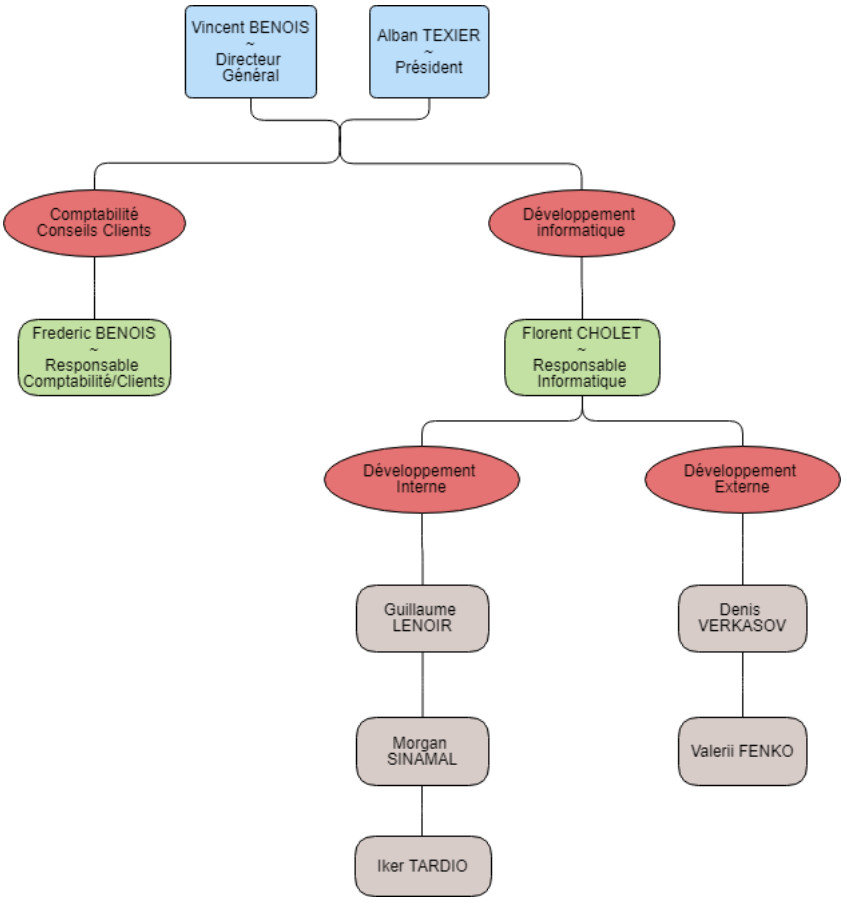


FIGURE 1 – Organigramme de JePilote

1.3 Gestion et management des projets

1.3.1 Méthodologie SCRUM

Afin de gérer ses différents projets, l'équipe de Je Pilote utilise une approche agile et notamment la méthodologie SCRUM. Cette méthodologie a été spécifiquement mise en place pour l'équipe de développeurs afin de maintenir une liste totalement transparente des demandes d'évolutions ou de corrections à implémenter. La méthodologie SCRUM fait intervenir 3 rôles distincts.

— Product Owner : Il s'agit du responsable du produit. C'est lui qui va définir et prioriser la liste des fonctionnalités du produit et définir les contenus de chaque sprint selon les réalisations effectuées par l'équipe. Chez JePilote , Vincent Benois tient ce rôle.
— Scrum Master : Il s'agit du chef de projet, il veille à au bon dynamisme l'équipe et de manière générale au respect du processus SCRUM. Florent Cholet, mon responsable de stage a cette responsabilité en plus d'être le chef de l'équipe de développement.
— Équipe développement : Elle regroupe tous les développeurs nécessaire à la réalisation du projet. Il s'agit de la fonction que j'occupe avec les autres stagiaires.

Cependant bien que la méthodologie SCRUM est dite agile, elle nécessite une grande organisation et une méthodologie précise. L'équipe de développement est organisé en taches de travail quasi-hebdomadaires appelé des sprints. Ces sprints qui se sont étendus de 1 à 2 semaines sont des suites de listes de taches sur laquelle l'équipe va devoir travailler.

— Réunion de planification de sprint : Lors de chaque début de sprint,en général le lundi matin, une réunion de planification est organisé afin de présente le contenu du sprint et les différents points clés.
— Mêlée quotidienne : Les mêlées quotidiennes sont des courtes réunions hebdomadaires avec le SCRUM master afin de lui faire part de l'avancement des taches et nous permette de lui faire part de nos éventuelles difficultés afin d'avoir un éclaircissement pour la suite du développement.
— Revue de Sprint : Les revue de Sprint s'effectuent à la fin de chaque sprint et est l'occasion de vérifier le bon fonctionnement du travail réalisé précédemment mais aussi d'analyser ce qui c'est bien ou mal passé durant ce sprint, ainsi que les difficultés rencontrées afin de savoir sur quoi travailler par la suite.

1.3.2 Outil de management de projet : Taiga

Afin de pouvoir gérer plus efficacement notre projet, le responsable informatique Florent Cholet a décidé d'utiliser Taiga qui est un outil de gestion de projet collaboratif. Cet outil nous permet de découper notre projet en sprint et de découper chaque sprint en taches afin d'avoir un meilleur visuel du projet. De plus pour chaque tache il est possible d'affecter une ou plusieurs personnes, ainsi que de définir le progrès de chaque tache (nouvelle, en progrès, prêt à être testé, fini) ce qui permet de facilement de voir l'avancement de notre sprint ainsi que l'avancement global de notre projet. Un exemple de sprint Taiga realisé pendant mon stage est transmis en annexe.

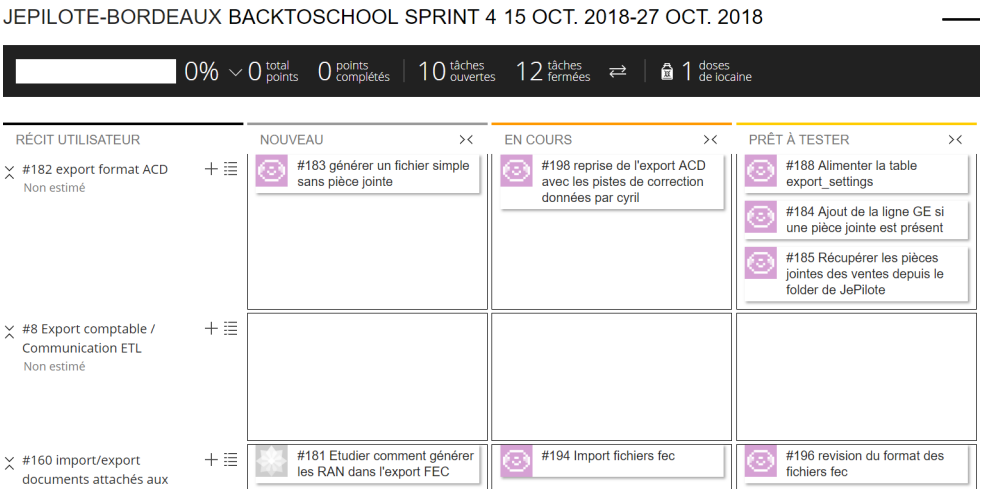


FIGURE 2 – Sprint JePilote - Taiga

1.3.3 Outil de communication de projet : Slack

Afin d'organiser la communication au sein de JePilote, l'ensemble des équipes utilisent l'outil de communication collaboratif Slack. Il nous permet de communiquer facilement avec n'importe quel membre de l'équipe et d'avoir un historique des conversations afin de retrouver les messages importants. De plus Slack permet d'organiser nos conversations en canaux correspondant a des sujets de discussions et nous avions notamment un canal pour l'équipe de développement.

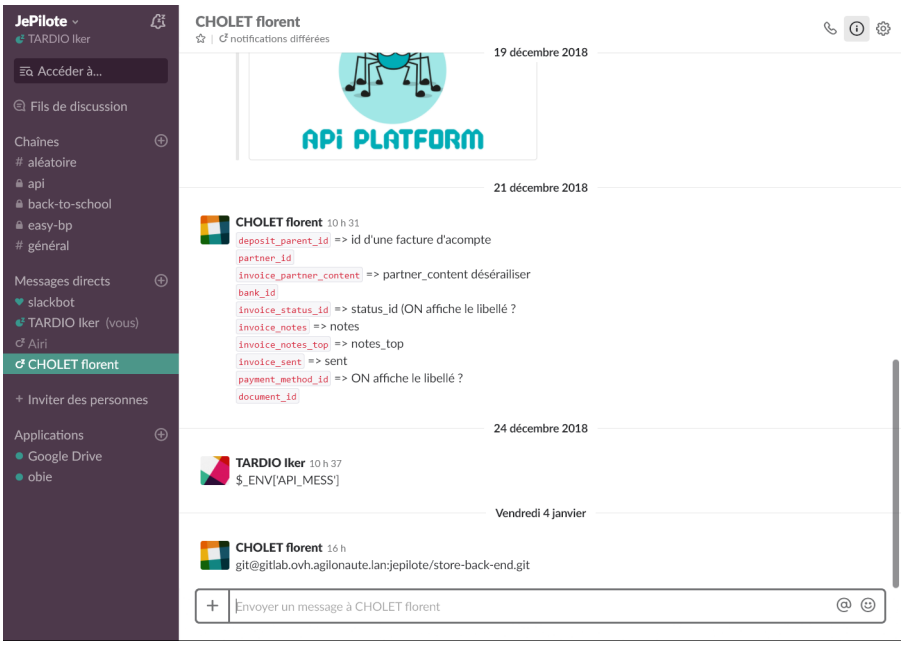


FIGURE 3 – Espace de travail JePilote - Slack

1.3.4 Git

Afin de pouvoir participer au développement informatique collaboratif de l'entreprise j'ai du m'initier rapidement au logiciel de versionnage Git. JePilote ayant un dépôt Gitlab ou sont stockés tous les codes sources des projets, j'ai dû vite m'adapter à leur façon d'utiliser cet outil.

Ainsi chaque projet est développer sur la branche principale et lorsque l'on souhaite ajouter du code à notre projet on effectue un "commit" qui doit comporter que des légers changements et avoir un titre qui résume les modifications apportées. Avant de commit on doit effectuer une re-vérification de code afin de s'assurer que tout fonctionne avant de modifier le projet. Après chaque commit il faut envoyé les modifications sur le dépôt distant et ensuite les modifications sur le serveur distant afin de les rendre visible les modifications à tout le monde. Cependant, il faut faire attention à éviter les conflits avec les autres collaborateurs en comparant le code rajouter à l'ancien. Une représentation du Gitlab de JePilote est transmise en annexe.

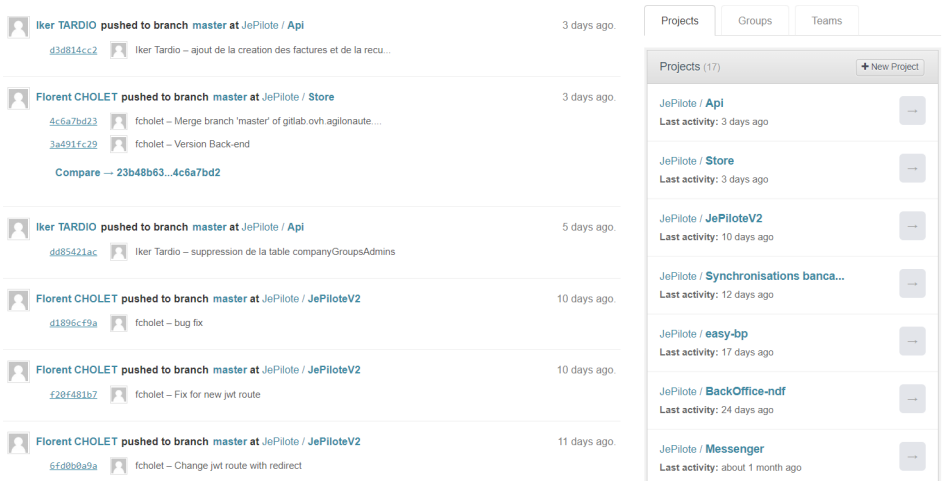


FIGURE 4 – Dépôt Gitlab JePilote

2 Descriptif de ma mission

2.1 Sujet de stage

A mon arrivé chez JePilote, l'entreprise avait prévu 3 sujets de stages pour les 3 nouveaux stagiaires, mais elle n'avait pas défini les rôles au préalable. Ces 3 sujets correspondaient à la création d'une API [3] au centre du système d'information de l'entreprise. La mise en place d'un site gérant la création de business plan, et le développement de nouveaux traitement dans le Back-Office de JePilote avec l'utilisation d'API tiers (Budget Insight, Stripe). Après réflexion, le responsable informatique Florent cholet a décidé de m'assigner le premier sujet, à savoir la création d'une API avec les Frameworks [4] Symfony et API Platform.

La mise en place de ce projet API a été décidé à la suite d'un constat clair. Le code de JePilote est devenu dense, complexe et basé sur ancien Framework PHP. Ainsi JePilote souhaite réalise une refonte du code, mais la migration d'un système à un autre est complexe car il y a beaucoup de fonctionnalités et le service est utilisé en permanence, on ne peut donc pas l'arrêter pour réaliser une refonte. La mise en place d'une API a été la solution, permettant de s'affranchir des interfaces en découpant le coté client du coté serveur dans le but de réaliser une refonte progressive et permettant aux clients d'utiliser directement l'API. L'objectif étant par la suite de réalisé de nouvelles interfaces avec de nouvelles technologies qui communiqueront directement avec l'API

Cette API devra avoir une approche First, et donc être au centre du système d'information de JePilote, avec des fonctionnalités essentielles au domaine de l'expertise comptable. Ces fonctionnalités sont notamment la gestion des utilisateurs, des factures, des documents et des exports comptables. Bien qu'une API First soit une interface de programmation qui permet à deux programmes ou applications de communiquer en utilisant le même langage et de se partager des données. Elle se distingue d'une simple API car elle se place au centre du système d'information d'une entreprise.

Ainsi cela permettra à JePilote de passer d'un site monolithique lourd à corriger, à une API centralisant la logique métier de l'entreprise garantissant à ses sites et applications un cycle de vie indépendant. De plus l'API permettra de faciliter les interactions des partenaires avec les données et le système d'information de l'entreprise. Enfin, la mise en place de l'API et la future refonte de JePilote permettra peu à peu à la société de s'affranchir du développement ukrainien.

3. Application Programming Interface.

4. Désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel

2.2 Planning du stage

Au début de mon stage je n'avais pas vraiment de planning à suivre avec des dates clés et des deadlines. Cependant le projet a bien été planifié d'une certaine manière. Les premières semaines du stage ont pour moi étaient des semaines de formations sur les différents outils et technologies a utilisé tout au long de ce stage. Suite à cette exploration, j'ai pu commencer la réalisation de l'API à la mi-octobre. Les bases de l'API étant ensuite réalisé j'ai pu commencer la réalisation de fonctionnalités plus complexes fin à la moitié du moi de novembre. Ce travail s'est déroulé jusqu'à la fin de l'année, qui c'est suivant par des tests de consommation l'API réalisé jusqu'à la fin du stage. Le projet n'ayant pas de cahier des charges précis sur l'application. Certains objectifs étaient émis au fur et à mesure du développement, provoquant par conséquent des "retours en arrière".

2.3 Contributions

Lors de mon arrivée, le projet d'API venait d'être mis en place par l'équipe JePilote. Le projet n'était pas commencé mais une base était déjà définie, avec les technologies Symfony et API Platform à utiliser et les différentes fonctionnalités à réaliser. Ce projet a été réalisé principalement en autonomie mais encadré par le responsable informatique Florent Cholet, restant disponible si je rencontrais des problèmes. Cependant le projet API étant déjà utilisé dans le développement de JePilote, il m'est arrivé de travailler avec d'autres développeurs. A ce jour, l'API JePilote est donc en place et est déjà utilisée pour le développements de JePilote. Cependant elle n'est pas totalement terminé et des fonctionnalités seront rajoutées au fur à mesure.

2.4 Outils et technologies utilisées

2.4.1 Symfony

Le Framework Symfony a été l'une des technologies les plus utilisées durant ce stage. Il s'agit de l'un des meilleurs Framework PHP qui lui permet d'être le plus utilisé au monde. Ce Framework a été choisi de part ses qualités mais aussi car il était déjà connu des équipes de développement de JePilote, ainsi le code sera plus lisible et plus facilement maintenable et renouvelable.
Symfony dispose de nombreux exécutable à écrire en ligne de commande permettant de simplifier la mise en place du code, mais aussi de nombreux composants qui sont des ensembles de librairies [5] PHP, téléchargeables grâce au logiciel de gestion de dépendances PHP Composer. De plus Symfony dispose d'une console de debug performante essentielle pour la réalisation d'un projet PHP complexe.
La grande notoriété de Symfony ainsi que sa grande communauté lui permettent d'avoir une documentation étoffée et régulièrement mise à jour.

Symfony dispose d'une architecture MVC « Modèle / Vue / Contrôleur ». C'est un découpage très répandu pour développer les sites Internet, car il sépare les couches selon leur logique propre :

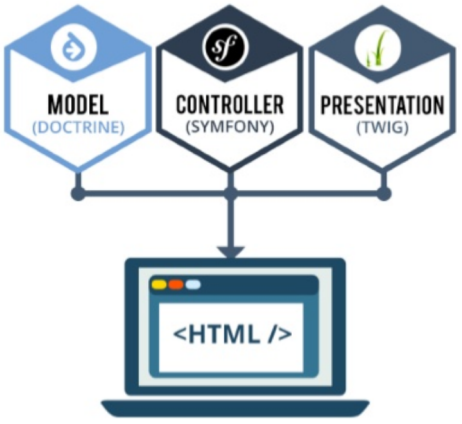


FIGURE 5 – Architecture traditionnelle Symfony

5. Bibliothèque logicielle.

La couche Modèle représente le traitement logique des données. Chez Symfony, c'est l'ORM (mapping objet-relationnel) Doctrine qui assure cette fonction. Cet ORM permet de simplifier l'accès à une base de données en proposant des « objets » plutôt que d'accéder directement à des données relationnelles.

La Vue est la couche où interagit l'utilisateur (un moteur de templates fait parti de cette couche). Twig est un moteur de templates [6] pour le langage de programmation PHP, utilisé par défaut par le framework Symfony.

La Contrôleur est un morceau de code qui appelle le modèle pour obtenir certaines données qu'il passe à la Vue pour le rendu au client.

2.4.2 MySQL et SQLyog

Les différentes bases de données utilisées lors de mon stage étaient des bases de données MySQL. MySQL est un SGBDR [7]. Les bases de données relationnelles sont des bases de données où l'information est organisée dans des tableaux à deux dimensions appelés des tables. MySQL fait partie des SGBD les plus utilisés au monde, autant par le grand public que par des professionnels.

Durant ce stage l'ensemble de l'équipe de développement a décidé d'utiliser le logiciel SQLyog, qui est un client graphique de base de données et qui offre la possibilité de se connecter à une base de données MySQL, de l'explorer, d'exécuter des requêtes SQL et de manipuler les données intuitivement.

6. Patron de mise en page.
7. Système de gestion de base de données relationnelle

2.4.3 API Platform

API Platform est un Framework PHP permettant de créer facilement et de personnaliser complètement des API Web modernes. En effet, cet outil va nous permettre de construire rapidement une API riche et facilement utilisable. De plus il respecte les standards PHP et Symfony ce qui rend ces deux outils très complémentaires. API-Platform est à ce jour un outil récent datant de 2015, qui se fait de plus en plus connaître dans le monde du développement Web.

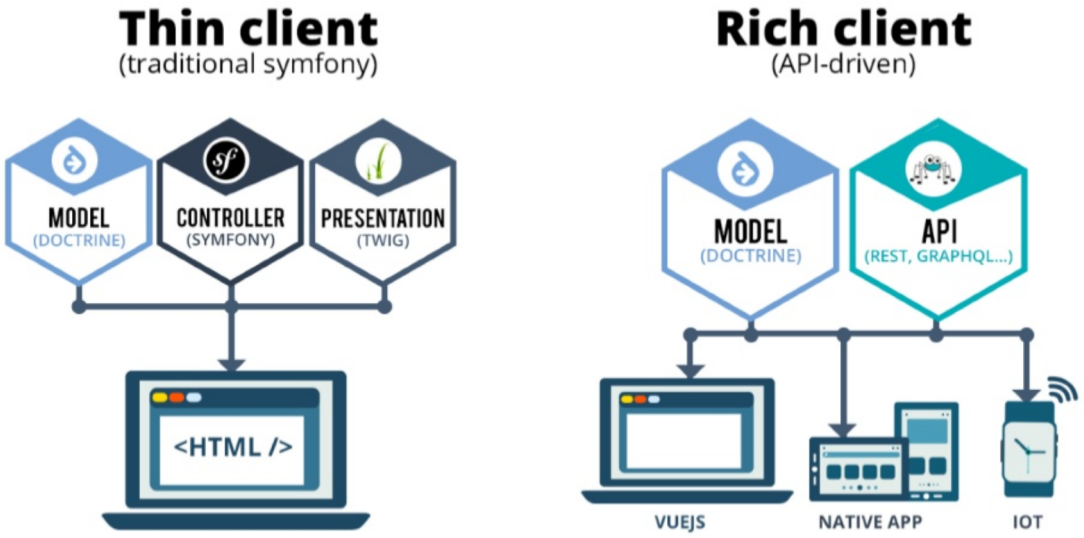


FIGURE 6 – Architecture API

2.4.4 Postman

Postman est un client [8] HTTP [9] pour API Web utilisé par de nombreux développeurs pour interroger ou tester des API. Il dispose d'un environnement graphique complet et permet notamment de construire et d'exécuter des requêtes HTTP afin de pouvoir gérer l'ensemble des interactions avec notre API. Ainsi Postman nous permet d'avoir un rendu de nos requêtes ainsi qu'une console de debug lors d'éventuelles erreurs.

8. Logiciel conçu pour se connecter à un serveur HTTP.
9. Protocole de communication client-serveur.

2.4.5 Atom

Durant tout mon mon stage, j'ai utilisé l'éditeur de texte Atom.io. Atom est un éditeur de texte open-source développé par GitHub. En plus d'un design épuré et de nombreuses fonctionnalités de bases, il dispose de nombreux packages afin de le rendre personnalisable et plus performant.

2.5 Prise de recul

Tout au long de mon stage j'ai vu mes responsabilités et la confiance de mon équipe se renforcer. En effet lors de mes débuts avec les nouvelles technologies on a beaucoup insisté sur le fait de bien communiquer mes choix et mon travail afin qu'on puisse m'aider à m'adapter au mieux aux besoins de l'entreprise. La mise en place de cette API à pour but de se placer au centre du système d'information de JePilote et d'être utilisé par les futurs développeurs de JePilote. Cependant l'API n'est pas finalisée et des fonctionnalités devraient être ajoutées par la suite. De plus une revue du code pourrait être nécessaire, car étant donné que c'était la première fois que j'utilisais ces différentes technologies, JePilote pourrait faire le choix de simplifier ou améliorer quelques-unes des fonctionnalités mises en place.

3 Activités réalisées pendant le stage

A mon arrivée chez JePilote j'ai du tout d'abord prendre connaissances des différents projets et de l'architecture informatique afin d'avoir une vue globale et donc une meilleur approche de ma mission. Les semaines qui suivent ont été principalement de l'exploration des Frameworks Symfony et API Platform. Pendant plusieurs semaines j'ai donc exploité la documentations des Frameworks et j'ai suivi de nombreux tutoriels afin de me former à ces derniers. Ayant par la suite acquis les connaissances et méthodologie nécessaire à la réalisation de cette API j'ai pu me lancer dans la phase de développement du projet d'API Comptable.

3.1 Exploration

3.1.1 Premiers pas avec les Frameworks

Les premières semaines de mon stage ont été principalement de l'exploration de Symfony et API Platform. Une fois ces Frameworks installés, la première étape de cette exploration a été la création d'entités [10], qui correspondent à des objets de mon projet. Suivant différents tutoriels et notamment la documentation de Symfony, j'ai pu réaliser un mini-projet et mettre en place la création de deux entités, une correspondant à des utilisateurs et une autre correspondant à des entreprises.

Ainsi pour réaliser ce mini-projet, j'ai du crée un nouveau dossier à l'intérieur duquel j'ai généré un squelette de projet Symfony à l'aide la console de commande :

```
$ composer create-project symfony/skeleton demo-api
```

J'ai ainsi pu commencer à créer les entités à l'intérieur de ce projet toujours avec l'aide de la console de commande et le Framework Symfony :

```
$ php bin/console make:entity
```

La console nous demande ensuite de donner un nom à l'entité crée, d'indiquer les noms des attributs [11] de cette dernière ainsi que leur type, leur longueur maximale, s'ils peuvent être nuls et s'ils sont uniques. Cette opération se répète tant que l'on souhaite ajouté des attributs.

Lorsque l'on a fini d'ajouter les attributs à l'entité, l'entité se créee. Ainsi on peut vérifier dans le fichier la création de notre entité.

10. Objet dont on confie l'enregistrement à l'ORM.
11. Champ décrivant la structure interne d'une entité.

FIGURE 7 – Entité Symfony

Une fois que le modèle de données a été crée, la deuxième étape consiste en l'utilisation du Framework API Platform. il faut ajouter l'annotation [12] d'API Platform au sein notre entité :

```
* @ApiResource()
```

Cette annotation permet au Framework de détecter les propriétés à exposer et autoriser les opérations CRUD liées à l'entité. Les opérations CRUD (Create, Read, Update, Delete) permettent d'effectuer des requêtes [13] sur une collection ou bien sur un seul élément de la collection.

— Opérations sur une collection

Méthode	Obligatoire	Description
GET	oui	Récupère une liste d'éléments
POST	non	Crée un nouvel élément

— Opérations sur un élément

Méthode	Obligatoire	Description
GET	oui	Récupère un élément
PUT	non	Met à jour un élément
DELETE	non	Supprime un élément

Ensuite pour effectuer une de ses opérations sur une entité, il faut appeler l'URL de notre API à laquelle on ajoute le nom de notre entité. Il faut ensuite choisir une méthode. Dans le cas d'une méthode GET sur l'entité Companies, il faut effectuer une des requêtes suivantes en fonction de si l'on veut récupérer la collection ou bien un élément :

```
http://localhost/api-demo/public/Companies
```

```
http://localhost/api-demo/public/Companies{id}
```

12. Élément permettant d'ajouter des méta-données à un code source.
13. Interrogation de la base de données.

3.1.2 Gestion d'une base de données avec Symfony et doctrine

Une fois que le modèle de données a été crée, il faut cependant intégrer une base de données SQL à l'ORM de Symfony, doctrine afin d'avoir des données à exploiter. Pour cela il faut insérer dans les paramètres de notre projet les informations de notre base, notamment son adresse, son identifiant et son mot de passe.

Maintenant qu'on a bien configuré la base de données pour Doctrine, on peut l'alimenter en ajoutant des informations sur les deux entités créées précédemment. On peut notamment vérifier que les tables sont bien alimentées en données grâce à l'outil graphique de base de données SQLyog.

Une fois que nos données ont bien été enregistrées, on peut donc effectuer la requête vue précédemment afin d'obtenir notre collection.

On récupère ici, l'ensemble de des données de notre collection. Le format dans lequel ces données sont partagées est appelé JSON.


```
{
    "@context": "/testapiv2/public/index.php/api/contexts/Users",
    "@id": "/testapiv2/public/index.php/api/users/1",
    "@type": "Users",
    "id": 1,
    "firstName": "Iker",
    "lastName": "Tardio",
    "companies": [
        "/testapiv2/public/index.php/api/companies/1"
    ]
}
```


FIGURE 8 – Collection de l'entité utilisateurs

3.1.3 JSON

JSON est l'acronyme de JavaScript Object Notation. Il s'agit d'un format de fichier open-standard, c'est à dire qu'il ne dépend d'aucun langage, permettant de stocker des données de différents types, de manière organisée et lisible.

Il existe d'autres formats de fichier comme notamment le XML, mais nous utiliserons dans ce projet seulement le format JSON afin de récupérer et d'envoyer des données.

Un fichier JSON est toujours structuré d'une manière précise :

— ... : les accolades définissent un objet.
— Les guillemets (double-quotes) et les double-points définissent un couple clé/-valeur (on parle de membre).
... : Les crochets définissent un tableau (ou array en anglais).
— Les virgules permettent de séparer les membres d'un tableau ou d'un objet .

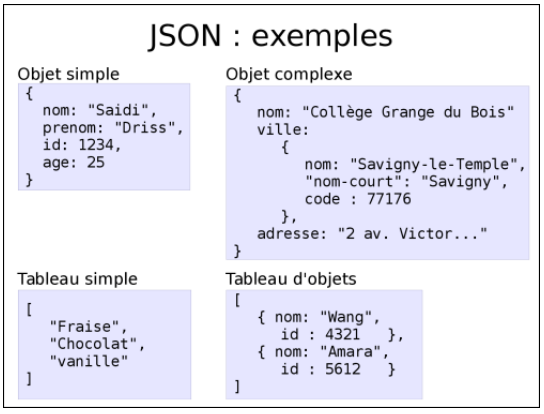


FIGURE 9 – Exemples de JSON

3.1.4 Architecture REST

L'architecture utilisé pour la création de notre API est appelé une architecture REST [14]. Cette architecture est défini par 5 règles différentes :

- l'URI comme identifiant des ressources, c'est à dire que les URL sont construite de manière précise. Il est nécessaire de prendre en compte la hiérarchie des ressources et la sémantique des URL pour les éditer.
- les verbes HTTP comme identifiant des opérations (GET, POST, PUT, DELETE)
- les réponses HTTP comme représentation des ressources. Une ressource peut avoir plusieurs représentations dans des formats divers, ici nous utiliseront principalement le format JSON.
- les liens comme relation entre ressources. Les liens d'une ressource vers une autre ont tous une chose en commun : ils indiquent la présence d'une relation.
- un paramètre comme jeton d'authentification. Le jeton d'authentification permet d'authentifier une requête, ce dernier sera explicité dans la partie sécurité.

14. Representational State Transfert

3.2 Développement API comptable

3.2.1 Présentation

Le projet d'API comptable est un projet majeur dans le développement informatique de JePilote. En plus des nombreux avantages qu'elle va amener, l'API disposera de fonctionnalités complexes.

Cette API va permettre :

— La gestion des utilisateurs, des sociétés et des partenaires.
— La génération des factures, des articles et leur comptabilisation
— L'import et l'export des écritures comptables.

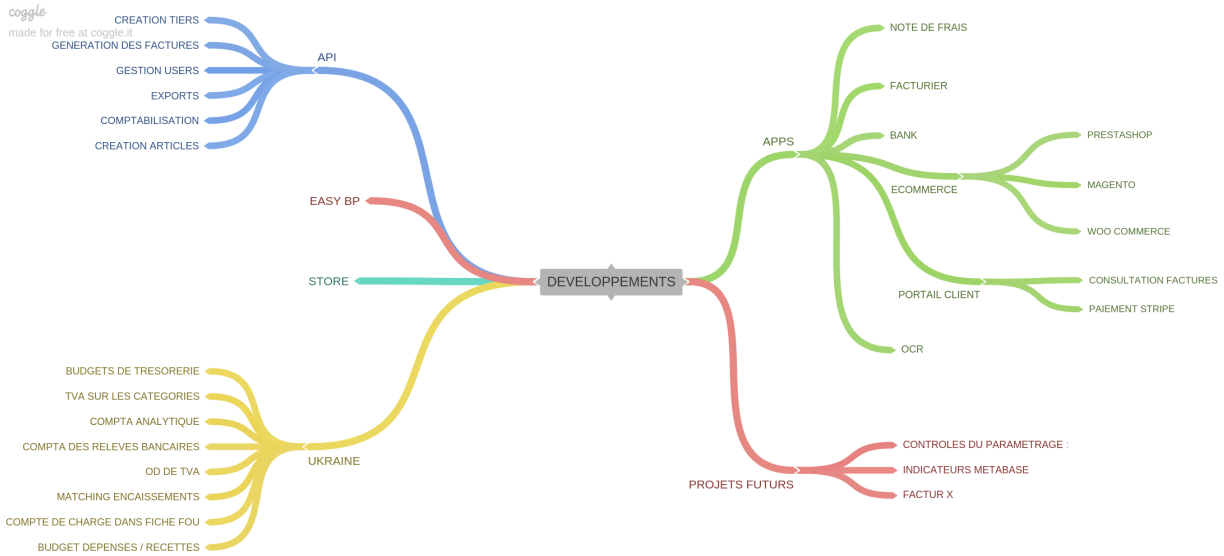


FIGURE 10 – Place du projet API au sein de JePilote

3.2.2 Architecture de la base de données

L'API comptable de JePilote ayant pour but de se placer au centre du système d'information, il faut donc utiliser le même modèle de données actuellement. La compréhension de ce système de base de données permet d'avoir une meilleur approche du produit JePilote et donc de la mise en place de l'API comptable. Ainsi la société JePilote possède une base de données principale complexe, avec plus de 100 classes différentes. La société dispose aussi de plusieurs bases de données dites de "test", similaire à celle-ci avec lesquelles nous travaillerons tout au long du stage. De plus du fait de la dimension internationale de l'équipe avec l'équipe ukrainienne, la base de données est entièrement en anglais.

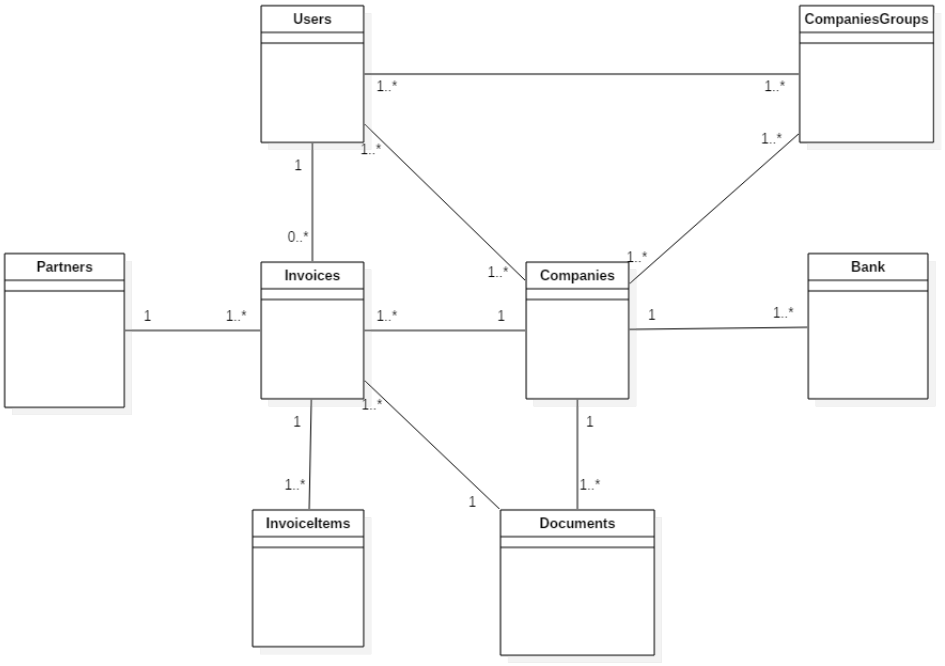


FIGURE 11 – Diagramme simplifié regroupant les tables de données principales

— La classe Users contient les données utilisateurs.
— La classe Companies contient les données des sociétés enregistrées.
— La classe CompaniesGroups contient les données des cabinets d'expertise-comptable.
— La classe Invoices contient les données des factures enregistrées sur JePilote.
— La classe InvoiceItems contient les données des articles de chaque factures.
— La classe Partners contient les données des clients ou fournisseurs.
— La classe Documents contient les données des documents comptable de chaque sociétés.

3.2.3 Mise en place API JePilote

Une fois les objectifs ainsi que le modèle de données présentés, on peut désormais commencer la réalisation de notre projet API. De la même manière que dans la partie exploration, on crée un nouveau projet correspondant à notre API. On implémente ensuite dans ce projet l'ensemble des tables de la base de données vu précédemment grâce à Symfony. Une fois la création des entités effectuée, on peut annoter chaque entité avec l'annotation spécifique d'API Platfom. On réalise ensuite la connexion entre notre projet d'API et la base de données de JePilote.

Nous avons donc crée un nouveau projet d'API contenant une centaine d'entités différentes, ayant chacune les caractéristiques CRUD d'API Platform. Il est donc possible d'effectuer des requêtes GET,POST,PUT,DELETE, sur chaque entités de notre projet en appelant le nom du serveur sur le quelle se trouve notre projet, suivie du nom de l'entité.

Nous avons ainsi crée une API sur la base du modèle de données de JePilote, Cependant l'API demeure simpliste. On va devoir utiliser les différentes librairies d'API Platform et Symfony afin d'ajouter des fonctionnalités clés à notre API, notamment concernant la sécurité de l'application et des données, ainsi que la récupération de ces dernières.

3.2.4 Authentification

Afin de rendre notre application sécurisée, il est important d'ajouter un module d'authentification afin de s'assurer de l'identité des utilisateurs. La première étape pour la mise en place de l'authentification consiste en la création d'une classe utilisateurs dans laquelle on implémente la fonctionnalité Users Interface. La mise en place de cette classe se différencie légèrement de la méthode vu précédemment :

```
$ php bin/console make:user
```

La création de cette entité utilisateur particulière, permet la mise en place d'un identifiant au choix, dans notre cas il s'agira de l'adresse mail, ainsi que d'un mot de passe pour nos utilisateurs.

De plus cela engendre la création d'un fichier sécurité dans notre projet, c'est dans ce fichier qu'est gérer l'authentification grâce à l'identifiant mail et le mot de passe. Par défaut le mot de passe est enregistré dans la base de données de manière cryptée. Cet encodage des mots de passes dans la base de données est essentiel afin de protéger les utilisateurs d'un potentiel vol d'identité et/ou de données.

3.2.5 Json Web Token

Lors de l'authentification, les identifiants utilisateurs sont en général enregistrés sur le Web sous forme de cookie [15]. Il est cependant important de crypter ces identifiants afin d'éviter les failles de sécurité.

Une librairie de Symfony, JWTAuthentifcation, nous permet de crypter ces données sous la forme d'un token d'authentification. En effet grâce à cette fonctionnalité, une fois l'authentification effectuée, l'information est échangée sous la forme d'un jeton signé afin de pouvoir en vérifier la légitimité. Ce type de token d'authentification permettant d'échanger des informations au format JSON de manière sécurisé est appelé un JSON Web Token(JWT).

Chaque JWT est composé de trois parties, chacune contenant des informations différentes :

— Le header identifie quel algorithme a été utilisé pour générer la signature, ainsi que le type de token dont il s'agit (souvent JWT, mais le champ a été prévu dans le cas où l'application traite d'autres types d'objet qu'un JWT).
— Le payload est la partie du token qui contient les informations que l'on souhaite transmettre, notamment les informations de nos utilisateurs. Ces informations doivent être non sensible afin d'éviter le transit de données importantes tel un mot de passe.
— La signature est la dernière partie du token. Elle est créée à partir du header et du payload générés et d'un secret. Une signature invalide implique systématiquement le rejet du token.

Ces trois parties sont chacunes encodées en base64url, puis concaténées en utilisant des points (“.”).

FIGURE 12 – Exemple de JWT

15. Suite d'informations envoyée par un serveur HTTP à un client HTTP.

3.2.6 Contrôles d'accès

Une fois l'utilisateur identifié, il faut analyser les rôles des différents utilisateurs. Pour distinguer ces deux types d'utilisateurs un champ administrateur a été rajouté dans l'entité User, cela permet d'envoyer l'information relative au rôle de l'utilisateur. On distingue 2 types d'utilisateurs, les Administrateurs et les utilisateurs. En effet on ne donne pas les mêmes accès a tous les utilisateurs, et l'annotation "AccesControl" d'API Platform qui est à ajouté après chaque définition de requête sur nos entité permet de gérer les différents accès aux routes de notre API.

```
* @ApiResource(
*  collectionOperations={
*  "get"={   "method"="GET",
*            "access_control"="is_granted('ROLE_ADMIN') or is_granted('ROLE_USER')"
*                                     },
*  "post"={   "method"="POST",
*             "access_control"="is_granted('ROLE_ADMIN') or is_granted('ROLE_USER')"
*                                     },
*  itemOperations={
*
*  "get"={   "method"="GET",
*            "access_control"="is_granted('ROLE_ADMIN') or is_granted('ROLE_USER')"
*                                     },
*  "put"={   "method"="PUT",
*            "access_control"="is_granted('ROLE_ADMIN') or is_granted('ROLE_USER')"
*                                     },
*  "delete"={"method"="DELETE",
*            "access_control"="is_granted('ROLE_ADMIN')"
*                                     }
*     })
```

Par exemple ici, les utilisateurs ayant un rôle utilisateur n'ont pas accès à la requête de suppression, au contraire des administrateur qui y ont accès. De plus il est possible de filtrer différents utilisateurs qui ont un même rôle utilisateur, en fonction des données relative aux utilisateurs enregistré dans la base de données.

```
*    itemOperations={
*    "get"={  "method"="GET",
*    "access_control"="is_granted('ROLE_ADMIN') or (is_granted('ROLE_USER') and object.accessRight(user))"
*          }
```

Ici la requête GET de l'entité utilisateur sur un élément, nous permet de donner seulement l'accès aux utilisateurs répondant à certains critères selon la fonction de droit d'accès. Par exemple ici cela donne accès aux utilisateurs appartiennent à un même cabinet comptable ou à la même entreprise que l'utilisateur recherché. Ainsi cela permet de données des droits aux utilisateurs en fonctions de leur relation entre entités.

3.2.7 Validation de données

La validation des données, c'est le fait de vérifier que les données qu'on envoie à notre API, sont bien des données valides. En effet lors de l'envoie de données à l'API avec les méthodes POST et PUT il est important de vérifier la validité de ces données avant de les insérer dans la base de données. L'annotation Assert de Symfony devant un attribut d'une entité, nous permet de vérifier la valité d'un attribut. On peut notamment vérifier qu'un email entré par un utilisateur correspond bien aux normes, que le code postal contienne seulement des chiffres et même que l'IBAN de l'utilisateur soit correct. Il suffit pour cela d'ajouter devant l'annotation Assert le type de vérification.

```
/**
 * @ORM\Column(name="email_address", type="string", length=100, nullable=false, unique=true)
 * @Assert\Email(
 * message = "choisissez une adresse email valide."
 * )
 */
private $emailAddress = '';
```

Ainsi si la donnée envoyé n'est pas valide, le message d'erreur associé à cette validation sera renvoyé.

3.2.8 Processus de sérialisation

Le processus de sérialisation se compose de deux contextes qui existent de base : la normalisation et la dénormalisation. La normalisation, c'est ce qui provient de la base de données, et que l'on récupère avec le verbe "GET" au format JSON. La dénormalisation, c'est lorsque l'on envoie avec le verbe « POST» ou «PUT» des données qui vont être au format JSON et qui vont être transformées et remises en place au niveau de l'entité et dans la base de données.

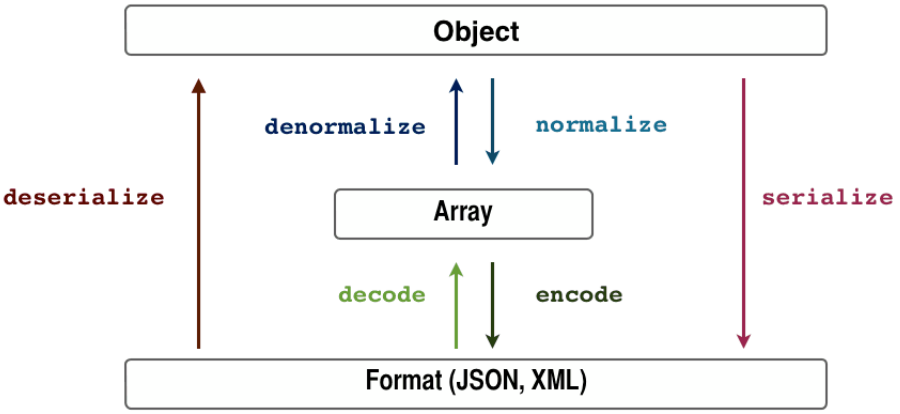


FIGURE 13 – Processus de sérialisation

Par défaut la sérialisation est présente dans API Platform et nous permet d'utiliser nos méthodes CRUD sur l'ensemble des attributs de chaque entité. Cependant le Framework nous permet de créer des groupes de sérialisation et de les associer aux routes de notre API afin de pouvoir gérer la sérialisation de celles-ci. Ainsi on peut les placer en annotation de notre ressource API Platform.

```
 * @ApiResource(attributes={
 *                          "normalization_context"={"groups"={"companies_read"}, "enable_max_depth"="true"},
 *                          "denormalization_context"={"groups"={"companies_write"}, "enable_max_depth"="true"}
```

On peut ensuite ajouter l'annotation du groupe de normalisation et dénormalisation aux attributs que l'on souhaite, afin de gérer ainsi leur affichage dans les méthodes GET, et leur insertion avec les méthodes PUT et POST.

```
/*
 * @Groups({"dossiers", "companies_write", "companies_read", "liste_companies", "invoice_read", "createSociety"})
 */
private $companyName = '';
```

Ainsi il est possible de gérer l'affichage et l'insertion des méthodes CRUD de notre API à l'aide des outils de sérialisation.

3.2.9 Filtres

Les librairies d'API Platform nous permettent aussi d'utiliser les filtres dans nos requêtes, pour filtrer les informations qu'on souhaite récupérer. Il existe différents types de filtres, les filtres de recherche, les filtres de date, les filtres booléens, les filtres numériques,les filtres de rang et des filtres d'ordre.

Pour effectuer une requête avec l'un de ses filtres il suffit d'ajouter l'annotation du filtre dans notre entité et l'attribut sur lequel il s'applique.

```
 * @ApiFilter(SearchFilter::class, properties={
 *     "companyName": "exact",
 * })
```

On peut par la suite requêter avec notre filtre sur le nom de la sociétés en ajoutant à notre requête GET, `"/companyName?=Nom"`
Ainsi ici, plutôt que de récupérer toute la collection, on va récupérer que les items dont le nom de société correspond au filtre.

Cependant si les filtres ne sont pas définis au préalable, il n'y aura pas de possibilités de filtrer, on récupérera donc l'ensemble de la collection. De plus il est également possible d'effectuer une requête sur plusieurs filtres qui s'effectuera de la même manière

3.2.10 Extension utilisateur

API Platform dispose d'une extension « CurrentUserExtension » qui comme son nom l'indique, permet de prendre en compte l'utilisateur qui s'est authentifié à l'API, et ainsi de lui permettre d'accéder seulement aux items qui lui appartiennent et non pas à toute la collection. En plus d'être très pratique pour l'utilisateur, ceci est essentiel au niveau de la sécurité afin d'éviter à l'utilisateur d'avoir accès à des données qui ne le concerne pas.

Cette extension est réalisable grâce à un code présent sur la documentation d'API Platform permettant de récupérer les collections et les items des ressources, à laquelle il faut ajouter pour chaque entité une requête en DQL afin de récupérer les données appartenant à l'utilisateur. De plus une partie de requête à été ajouté dans le cas ou l'utilisateur est l'administrateur d'un cabinet comptable. Ainsi il aura accès à toutes les données appartenant aux utilisateurs du cabinet comptable. Cette fonctionnalité a notamment été ajoutée suite à la demande du principal client de JePilote.

3.3 API Comptable

Nous avons ainsi crée une API First fonctionnelle et sécurisée. Cependant nous n'avons pas encore répondu à l'ensemble de nos objectifs. En effet certains objectifs requièrent des fonctionnalités plus complexes que des requêtes CRUD basique sur une seule entité. Afin de pouvoir répondre à ces objectifs, il va falloir créer nos propres requêtes customisées au sein des Controllers de Symfony qui seront reliées directement à notre API grâce aux annotations d'API Platform. Tout d'abord, pour créer des requêtes, il faut créer une classe dans un Controller dans lequel on va pouvoir renseigner nos différentes fonctions. Les fonctions sont ensuite annotées par l'URL de la route, puis reliées à une entité et au fichier sécurité afin de faire le lien avec l'API.

3.3.1 Manipulation des données depuis le Controller

La récupération et l'enregistrement des données depuis un Controller ne se fait pas de manière automatique, contrairement aux requêtes CRUD basique sur les entités. Cependant la méthodologie est toujours la même que ce soit pour la récupération ou l'enregistrement, et nous allons les utiliser tout le long du développement de ces requêtes customisées.

- La récupération des données se fait en accédant aux données envoyées au format JSON, puis en les décodant afin de rendre ces données utilisables dans notre code.
- L'enregistrement des données se fait en grâce à l'ORM Doctrine. On utilise pour cela une l'entité manager de doctrine qui réalise l'enregistrement en deux temps. On utilise pour cela des fonctions de Doctrine qui permettent de signaler à l'ORM que l'objet doit être enregistré et qui ensuite met à jour la base à partir des objets signalés à Doctrine.

3.3.2 Enregistrement des sociétés

L'enregistrement des sociétés est l'une des fonctionnalités importantes de l'API. Avec plus de 23000 sociétés enregistrés sur JePilote, cette fonction va permettre aux clients de faciliter les inscriptions à JePilote. L'enregistrement des sociétés diffère d'une simple requête POST sur la table sociétés. En effet lors de l'enregistrement de société de JePilote, on fait aussi appel à plusieurs tables pour lesquels on va enregistrer des données lors de l'inscription. Plus d'une dizaine de tables de donnés sont concernées par l'enregistrement des sociétés dont notamment les tables des sociétés, des banques, des experts comptables et des exercices comptables.

La création d'une entreprise se fait en plusieurs étapes :

— Enregistrement dans les tables les données comptables relative au type de sociétés
— Récupération des données de la société et enregistrement
— Récupération des données bancaires et enregistrement.
— Récupération des données de l'expert-comptable et enregistrement
— Récupération des données de l'exercice comptable et enregistrement

Lors de l'enregistrement de la société, l'utilisateur doit spécifier le nom et le type de celle-ci (sociétés commerciale, start-up, associations etc...). La base de données de JePilote dispose justement de sociétés modèles pour chaque type de société. Ainsi grâce à ses sociétés modèles, une partie du processus d'enregistrement est automatisé. En effet, pour enregistrer dans la base les données correspondantes au type de la société, il suffit de récupérer les données de la société modèle et de les enregistrer de nouveau, au nom de la société crée. C'est un ainsi que sont initialisés pour chaque sociétés ses journaux et documents comptables ainsi que ses partenaires.

Ainsi une fois les données de l'utilisateur récupéré au format JSON, on réalise ce clonage de données modèles. Une fois cette étape réalisé, les enregistrements liés à la sociétés, à sa banquet et à son expert-comptables sont effectués directement en utilisant les données récupérés.

Cependant, pour enregistrer un exercice comptable, il faut effectuer un enregistrement sur deux tables, la table correspondant à l'exercice comptable, et la table période qui découpe cette exercice de manière mensuelle. Ainsi on récupère les dates de début et de fin de l'exercice en vérifiant que celui ne dépasse pas la durée maximale de deux ans et on les insère dans la table correspondante. On découpe ensuite chaque mois composant l'exercice avec la date de début et de fin correspondant au premier et au dernier du mois, excepté pour le premier mois de l'exercice qui peut réalise en cours de mois.

A chaque étape on récupère les données au format JSON, puis on les enregistre dans la base de données, tout en vérifiant la validité des données grâce à l'annotation Assert placée dans les attributs de nos entités dans la partie validation des données. Ainsi si une donnée n'est pas valide, elle ne sera enregistrées et un message d'erreur sera renvoyé, expliquant les modifications à effectuer. L'utilisateur pourra ensuite re-effectuer son enregistrement correctement.

L'enregistrement des sociétés se fait ici via une seule route de notre API. Au début du stage une inscription en quatre étapes correspondantes à quatre routes avaient été réalise, mais celle-ci a été modifié en milieu de stage afin de permettre aux partenaires de JePilote d'insérer directement la route de notre API sur leur site Web en une étape.

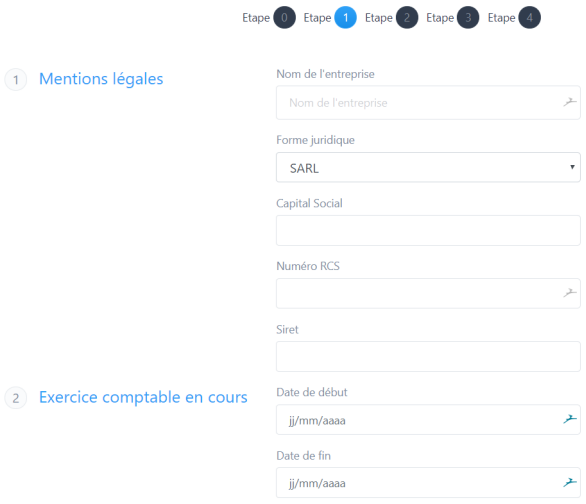


FIGURE 14 – Aperçu de l'interface de création de sociétés appelant l'API

3.3.3 Génération des factures

Enregistrement des factures

Avec plus de 25000 factures enregistrées par mois, l'enregistrement des factures est une fonctionnalité importante de notre API. Lors de la génération des factures il faut manipuler cinq tables de données différentes :

— La table des partenaires
— La table des factures
— La table des articles de facture
— La table des montants des articles
— La table des montants des factures

L'enregistrement des données du partenaires n'est cependant pas systématique. En effet un utilisateur peut choisir d'ajouter un nouveau partenaire et ses données. Cependant lors de l'enregistrement d'une facture, l'utilisateur peut aussi choisir un partenaire qu'il avait déjà enregistré au préalable dans quel cas on a juste en récupérer l'identifiant de ce partenaire afin de le lier à notre facture. Ainsi on vérifie qu'un même partenaire n'est pas enregistré plusieurs fois.

L'enregistrement de la facture se fait en récupérant les données et permet notamment de lier une facture a une société, un utilisateur de cette dernière, un partenaire et lui associer un type de facture et un numéro de facture qui est générer de manière précise en fonction de la date et de sa place dans le dossier comptable.

Les articles sont eux enregistré à l'aide d'une boucle enfin d'enregistrer un nombre illimité d'articles sur une facture. On insère notamment ici le nom des articles et leur prix HT. De plus chaque article est identifié avec son nom et l'entreprise qui l'enregistre. Ainsi cela permet d'enregistrer les mêmes articles enregistrés plusieurs fois afin de mettre en place des statistiques comptables.

Les deux dernières tables correspondant aux montant sont calculés à partir de la facture et de ses articles. Dans la table des montants des articles ont récupère le prix HT et on auquel on ajoute le pourcentage de la taxe en fonction du type de taxe pour la société afin d'obtenir le montant TTC. Le total des articles est ensuite récupéré dans la table liés au montant de la facture auquel on ajoute le montant déjà payé par l'utilisateur et le montant restant à payer, afin d'avoir accès à sa balance des paiements.

Lecture des factures

Une fois que toutes ces données ont été enregistrés dans la table de données. On peut récupérer ses données afin de réaliser une lecture de la facture. Cependant ses données se trouvant dans cinq tables différentes, on ne peut pas réaliser une requête GET basique de la table des ventes. Afin de récupérer l'ensemble de ses données, nous allons utiliser les groupes de sérialisation vu précédemment. En effet on a vu que les groupes de normalisation permettaient de gérer l'affichage d'une entité, cependant les groupes peuvent aussi s'utiliser sur une entité liée par une relation à celle possédant le groupe de normalisation. Ainsi il est possible à l'aide de ces groupes de récupérer toutes les données nécessaire à la lecture de nos factures à l'intérieur d'une seule requête. De cette manière on récupère toutes ces données en effectuant une requête GET sur nos ventes. Par la suite ces données vont être récupérer coté client afin de réaliser un PDF de cette facture.

3.3.4 Export des écritures comptables

En comptabilité, une écriture comptable est l'opération consistant à enregistrer les flux financier à l'intérieur de comptes comptables. Ces écritures sont enregistrés dans un journal comptable qui est composé de documents et de mouvements comptables. Ainsi cela représente l'opération de base sur laquelle s'appuie toute la chaîne de production des résultats comptables.
Cependant les entreprises qui tiennent leur comptabilité au moyen de systèmes informatisés doivent pouvoir la présenter sous forme de fichiers dématérialisés pour que les experts puissent analyser les écritures comptables à l'aide de logiciels spécifiques nécessitant un export avec un format particulier.
Chez JePilote ces exports se font de deux manières différentes, une société peut choisir d'exporter ses écritures au format FEC et au format ACD. Cette fonctionnalités va notamment permettre au premier partenaire de JePilote qui a plus de 900 dossiers, d'automatiser les exports de manière régulière en s'affranchissant de toutes interfaces.

Pour réaliser ces types d'export, il faut au préalable que l'utilisateur ait rentré des paramètres qui seront insérés dans une table dédiée aux exports comptables. Ces données récupérées correspondent au numéro de la société intéressé par l'export, le format de l'export ainsi que l'intervalle de dates entre lesquelles on souhaite réalisé notre export. En plus de ces données, un identifiant crypté est ajouté à la table afin de récupérer l'export une fois ce dernier réalisé.

Une fois la demande d'export enregistrée, une vérification a lieu dans la base de données afin de voir si il y a bien des écritures comptables enregistrés entre l'intervalle de date sélectionné par l'utilisateur. Dans le cas contraire un message d'erreur est renvoyé à l'utilisateur.

Une fois la vérification effectuée favorablement, les exports se font ensuite de manière différente selon les formats.

Format ACD

Pour un format ACD, toutes les données comptables sont récupérées à l'aide d'une vue ,ou table virtuelle, écrite dans notre système de base de données en SQL. Cette vue est ensuite intégrée à notre projet afin d'avoir accès à ces données. Ces données sont ensuite insérées dans un fichier avec une extension précise (.IN) et sont reparties avec des zones d'espaces et de saut à la ligne de manière précise selon les normes ACD. Une fois le fichier d'export réalisé, ce dernier est inséré dans un dossier Zip auquel on va ajouter tous les documents PDF correspondant aux écritures comptables présentes dans cet export. Ces documents PDF sont récupérés en se connectant au serveur de JePilote puis en récupérant tous les documents correspondant présent dans la GED de JePilote.

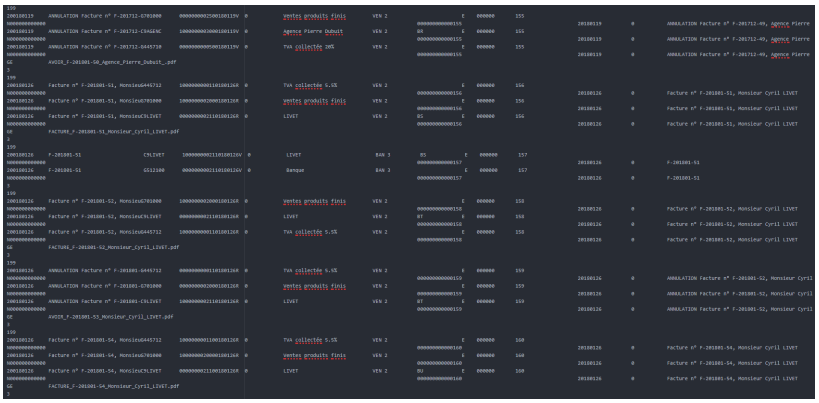
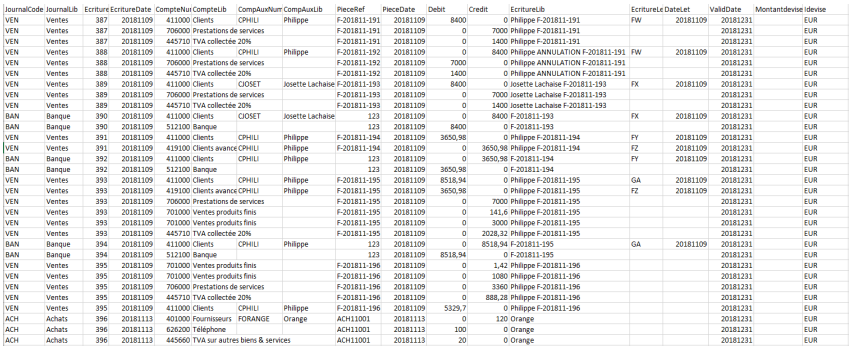


FIGURE 15 – Export comptable au format ACD

Format FEC

Pour un format FEC, la méthodologie est presque similaire. Les données des écritures comptables sont récupérées dans la base de données et sont enregistrées dans un tableau de données. Chaque ligne de ce tableau correspondant aux écritures comptables sont ensuite copiés dans un fichier puis insérées dans un dossier Zip.



JournalCode	JournalLib	Ecriture	EcritureDate	CompteNui	CompteLib	CompAuxNum	CompAuxLib	PieceRef	PieceDate	Debit	Credit	EcritureLib	EcritureLe	DateLet	ValidDate	Montantdevise	Idevise
VEN	Ventes	387	20181109	411000	Clients	CPHILI	Philippe	F-201811-191	20181109	8400	0	Philippe F-201811-191	FW	20181109	20181231		EUR
VEN	Ventes	387	20181109	706000	Prestations de services			F-201811-191	20181109	0	7000	Philippe F-201811-191			20181231		EUR
VEN	Ventes	387	20181109	445710	TVA collectée 20%			F-201811-191	20181109	0	1400	Philippe F-201811-191			20181231		EUR
VEN	Ventes	388	20181109	411000	Clients	CPHILI	Philippe	F-201811-192	20181109	0	8400	Philippe ANNULATION F-201811-191	FW	20181109	20181231		EUR
VEN	Ventes	388	20181109	706000	Prestations de services			F-201811-192	20181109	7000	0	Philippe ANNULATION F-201811-191			20181231		EUR
VEN	Ventes	388	20181109	445710	TVA collectée 20%			F-201811-192	20181109	1400	0	Philippe ANNULATION F-201811-191			20181231		EUR
VEN	Ventes	389	20181109	411000	Clients	CIOSET	Josette Lachaise	F-201811-193	20181109	8400	0	Josette Lachaise F-201811-193	FX	20181109	20181231		EUR
VEN	Ventes	389	20181109	706000	Prestations de services			F-201811-193	20181109	0	7000	Josette Lachaise F-201811-193			20181231		EUR
VEN	Ventes	389	20181109	445710	TVA collectée 20%			F-201811-193	20181109	0	1400	Josette Lachaise F-201811-193			20181231		EUR
BAN	Banque	390	20181109	411000	Clients	CIOSET	Josette Lachaise		123	20181109	0	8400 F-201811-193	FX	20181109	20181231		EUR
BAN	Banque	390	20181109	512100	Banque				123	20181109	8400	0 F-201811-193			20181231		EUR
VEN	Ventes	391	20181109	411000	Clients	CPHILI	Philippe	F-201811-194	20181109	3650,98	0	Philippe F-201811-194	FY	20181109	20181231		EUR
VEN	Ventes	391	20181109	419100	Clients avance	CPHILI	Philippe	F-201811-194	20181109	0	3650,98	Philippe F-201811-194	FZ	20181109	20181231		EUR
BAN	Banque	392	20181109	411000	Clients	CPHILI	Philippe		123	20181109	0	3650,98 F-201811-194	FY	20181109	20181231		EUR
BAN	Banque	392	20181109	512100	Banque				123	20181109	3650,98	0 F-201811-194			20181231		EUR
VEN	Ventes	393	20181109	411000	Clients	CPHILI	Philippe	F-201811-195	20181109	8518,94	0	Philippe F-201811-195	GA	20181109	20181231		EUR
VEN	Ventes	393	20181109	419100	Clients avance	CPHILI	Philippe	F-201811-195	20181109	3650,98	0	Philippe F-201811-195	FZ	20181109	20181231		EUR
VEN	Ventes	393	20181109	706000	Prestations de services			F-201811-195	20181109	0	7000	Philippe F-201811-195			20181231		EUR
VEN	Ventes	393	20181109	701000	Ventes produits finis			F-201811-195	20181109	0	141,6	Philippe F-201811-195			20181231		EUR
VEN	Ventes	393	20181109	701000	Ventes produits finis			F-201811-195	20181109	0	3000	Philippe F-201811-195			20181231		EUR
VEN	Ventes	393	20181109	445710	TVA collectée 20%			F-201811-195	20181109	0	2028,32	Philippe F-201811-195			20181231		EUR
BAN	Banque	394	20181109	411000	Clients	CPHILI	Philippe		123	20181109	0	8518,94 F-201811-195	GA	20181109	20181231		EUR
BAN	Banque	394	20181109	512100	Banque				123	20181109	8518,94	0 F-201811-195			20181231		EUR
VEN	Ventes	395	20181109	701000	Ventes produits finis			F-201811-196	20181109	0	1,42	Philippe F-201811-196			20181231		EUR
VEN	Ventes	395	20181109	701000	Ventes produits finis			F-201811-196	20181109	0	1080	Philippe F-201811-196			20181231		EUR
VEN	Ventes	395	20181109	706000	Prestations de services			F-201811-196	20181109	0	3360	Philippe F-201811-196			20181231		EUR
VEN	Ventes	395	20181109	445710	TVA collectée 20%			F-201811-196	20181109	0	888,28	Philippe F-201811-196			20181231		EUR
VEN	Ventes	395	20181109	411000	Clients	CPHILI	Philippe	F-201811-196	20181109	5329,7	0	Philippe F-201811-196			20181231		EUR
ACH	Achats	396	20181113	401000	Fournisseurs	FORANGE	Orange	ACH11001	20181113	0	120	Orange			20181231		EUR
ACH	Achats	396	20181113	626200	Téléphone			ACH11001	20181113	100	0	Orange			20181231		EUR
ACH	Achats	396	20181113	445660	TVA sur autres biens & services			ACH11001	20181113	20	0	Orange			20181231		EUR

FIGURE 16 – Export comptable au format FEC

Envoi des exports

Une fois l'export au format désiré effectué, il est stocké sur les serveurs de JePilote afin que l'utilisateur y ait accès. On a ensuite fait appel à l'API Messenger de JePilote qui permet d'envoyer un mail à notre utilisateur, dans lequel on lui envoie le lien de téléchargement du dossier export. Afin de préserver la securité des données, l'URL de téléchargement est accessible que par l'utilisateur qui à reçu le mail puisque l'URL utilise l'identifiant crypté généré lors de l'enregistrement de la demande d'export.

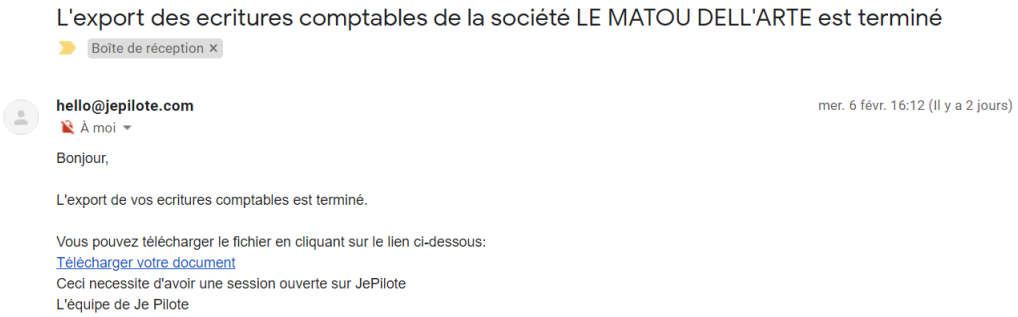


FIGURE 17 – E-mail envoyé lors d'un export

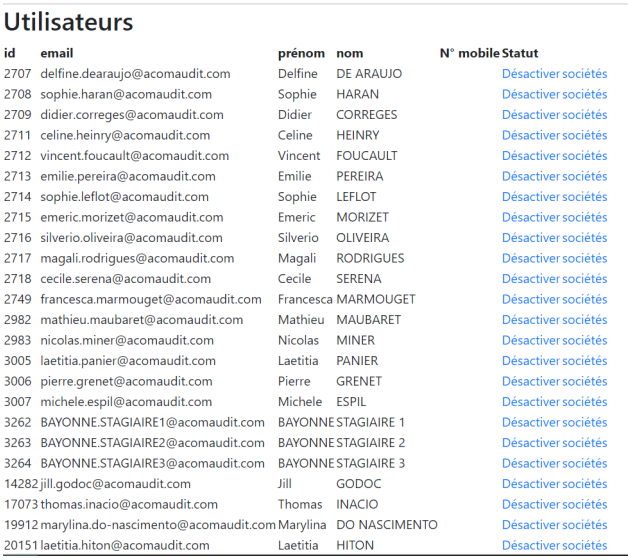
3.4 Mise en pratique de l'API comptable

3.4.1 Démonstration API

Une fois la mise en place de l'API comptable effectuée, j'ai pu mettre cette dernière en pratique à travers deux cas d'utilisation. La première a été la mise en place d'une interface Web basique de démonstration permettant l'export d'écritures comptables pour le partenaire principal de JePilote, ACOM audit.

Cette interface Web doit permettre au responsable informatique d'ACOM de lister l'ensemble des dossiers de sociétés, des agences et des collaborateurs des cabinets comptables d'ACOM sont liés. Ainsi l'utilisateur peut par la suite choisir un collaborateur d'une agence puis sélectionner un dossier d'entreprise afin que le collaborateur reçoivent l'export de ce dernier par email.

Cette interface Web a été réalisé avec les technologie Twig, qui est le moteur de template par défaut de Symfony, et Vue.js, un Framework Javascript. L'utilisation du Javascript est nécessaire afin de permettre l'affichage dynamique des données récupérer depuis notre API. L'accès à l'API se fait se fait grâce à une librairie Js , Axios, qui sert de client permettant d'effectuer des requêtes HTTP. Ainsi les données de chaque pages sont générés en effectuant une requête HTTP contenant l'URL correspondantes, ainsi que le JWT d'identification nécessaire à l'accès à l'API. Les données récupérées sont ensuite affichées sous forme de tableau dans lesquels on peut passer d'une page Web à une autre à l'aide d'hyperliens. Le résultat de l'interface Web effectué sont disponibles en annexes.



Utilisateurs

id	email	prénom	nom	N° mobile	Statut
2707	delfine.dearaujo@acomaudit.com	Delfine	DE ARAUJO		Désactiver sociétés
2708	sophie.haran@acomaudit.com	Sophie	HARAN		Désactiver sociétés
2709	didier.correges@acomaudit.com	Didier	CORREGES		Désactiver sociétés
2711	celine.heinry@acomaudit.com	Celine	HEINRY		Désactiver sociétés
2712	vincent.foucault@acomaudit.com	Vincent	FOUCAULT		Désactiver sociétés
2713	emilie.pereira@acomaudit.com	Emilie	PEREIRA		Désactiver sociétés
2714	sophie.leflot@acomaudit.com	Sophie	LEFLOT		Désactiver sociétés
2715	emeric.morizet@acomaudit.com	Emeric	MORIZET		Désactiver sociétés
2716	silverio.oliveira@acomaudit.com	Silverio	OLIVEIRA		Désactiver sociétés
2717	magali.rodrigues@acomaudit.com	Magali	RODRIGUES		Désactiver sociétés
2718	cecile.serena@acomaudit.com	Cecile	SERENA		Désactiver sociétés
2749	francesca.marmouget@acomaudit.com	Francesca	MARMOUGET		Désactiver sociétés
2982	mathieu.maubaret@acomaudit.com	Mathieu	MAUBARET		Désactiver sociétés
2983	nicolas.miner@acomaudit.com	Nicolas	MINER		Désactiver sociétés
3005	laetitia.panier@acomaudit.com	Laetitia	PANIER		Désactiver sociétés
3006	pierre.grenet@acomaudit.com	Pierre	GRENET		Désactiver sociétés
3007	michele.espil@acomaudit.com	Michele	ESPIL		Désactiver sociétés
3262	BAYONNE.STAGIAIRE1@acomaudit.com	BAYONNE	STAGIAIRE 1		Désactiver sociétés
3263	BAYONNE.STAGIAIRE2@acomaudit.com	BAYONNE	STAGIAIRE 2		Désactiver sociétés
3264	BAYONNE.STAGIAIRE3@acomaudit.com	BAYONNE	STAGIAIRE 3		Désactiver sociétés
14282	jill.godoc@acomaudit.com	Jill	GODOC		Désactiver sociétés
17073	thomas.inacio@acomaudit.com	Thomas	INACIO		Désactiver sociétés
19912	marylina.do-nascimento@acomaudit.com	Marylina	DO NASCIMENTO		Désactiver sociétés
20151	laetitia.hiton@acomaudit.com	Laetitia	HITON		Désactiver sociétés

FIGURE 18 – Interface agences de démonstration de l'API

FIGURE 19 – Interface utilisateur de démonstration de l'API

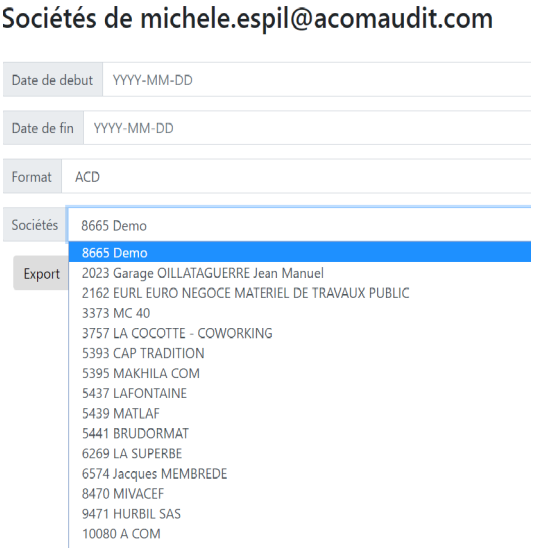


FIGURE 20 – Interface export de démonstration de l'API

3.4.2 Enregistrement des factures du Store

Le deuxième cas d'utilisation de l'API a été l'enregistrement des factures depuis le projet store de JePilote. Ce projet correspond à la mise en place d'une partie du site de JePilote ou l'utilisateur a la possibilité d'obtenir des fonctionnalités supplémentaires payantes qui ne sont pas disponibles de base sur le site Web de JePilote. Le projet store est une API qui a été mise, spécifique au store afin d'enregistrer les données de paiement obtenu par l'API de Stripe, qui est une solution de traitement de paiements en ligne pour les entreprises. Ce projet étant une API réalisé en PHP, l'accès à l'API JePilote se fait par un autre client HTTP, le client Guzzle qui est un client HTTP pour PHP. On utilise donc notre client Guzzle en lui fournissant l'URL de notre générateur de facture vu précédemment, notre JWT d'authentification ainsi qu'un tableau de données en JSON correspondant aux données générer lors de l'achat d'un module.
Ces données sont donc transmises à l'aide de notre API du projet Store à la base de données de JePilote afin de générer les factures. Un exemple de facture PDF généré à partir du store et à l'aide de l'API a été renseigné en annexe.

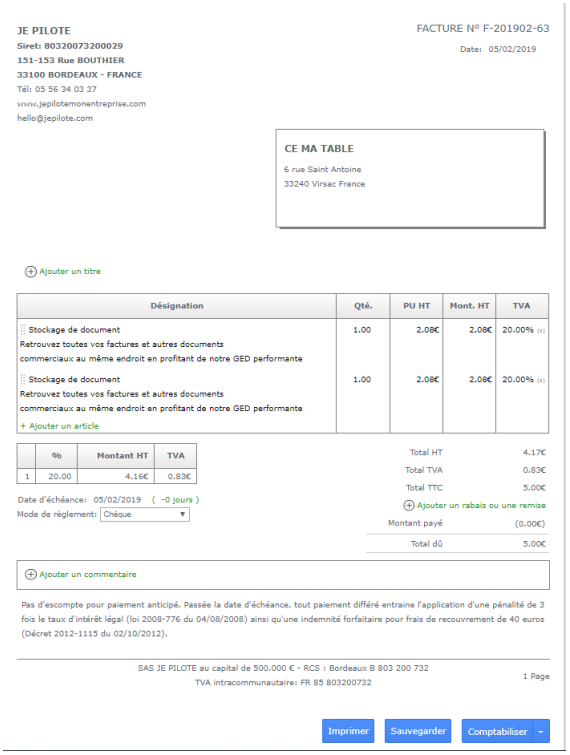


FIGURE 21 – Facture généré depuis le store

Conclusion

Ce stage a été pour moi l'occasion de gérer l'évolution et le développement d'un projet informatique sur une longue période. De plus il m'a permis d'apprendre beaucoup de concepts et de méthodologie lié aux projets Web. En effet j'ai maintenant une bonne connaissance dans le développement Web et plus particulièrement dans le développement d'API qui est un est un sujet très tendance en ce moment avec l'explosion des échanges de données. Le développement réalisé m'a notamment permis d'acquérir des compétences techniques comme l'utilisation des Frameworks Symfony, API Platform Vue.js ainsi que la manipulation de bases de données conséquentes. Au delà des compétences informatiques, ce stage m'a aussi permis de m'intéresser à la comptabilité et ainsi d'élargir ma culture d'ingénieur.

De plus ayant été le principal acteur de ce projet, cela m'a permis de développer une capacité à trouver des solutions aux problèmes auxquels j'étais confronté lorsque j'utilisais des nouvelles technologies. Bien que majoritairement autonomie sur ce projet, j'étais accompagnés par les membres de l'équipe JePilote et notamment mon responsable Florent Cholet qui a su me conseiller et dont j'ai vu la confiance en moi et mon travail s'accroître au fur à mesure du stage.

Cependant le point négatif de ce stage est que la dernière fonctionnalité de l'API correspondant à l'import des écritures comptables n'a pas pu être réalisé par manque de temps et de notions. Florent Cholet considérant cette fonctionnalité trop complexe à mettre en place d'un point de vue comptable.

Ce stage m'a aussi permis de découvrir le milieu professionnel d'une Start-up et ainsi d'agrandir ma vision du monde professionnel. Ainsi JePilote m'a fait découvrir les enjeux d'une Start-up, de la vision entrepreneuriale et la gestion de plusieurs projets en parallèles.

Enfin l'utilisation d'API et la manipulation d'un grand nombre de données m'ont permis de me conforter dans mon choix de filière. Des nouvelles opportunités d'exploiter ces données vont voir le jour et la demande de personnel qualifié pour les exploiter risque de croître également. Ayant observé ce discours a plusieurs reprises, je me suis conforté dans l'idée de choisir la filière fouille de données du génie informatique.

Glossaire

Framework : Désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.

TPE : Très petites entreprises.

Marque blanche : Il s'agit donc d'un mécanisme commercial de mise à disposition d'outils ou de produits, sans citer la marque.

API : Application Programming Interface.

Librairie : Bibliothèque logicielle.

ORM : Mapping objet-relationnel.

Template : Patron de mise en page.

SGBDR : Système de gestion de base de données relationnelle.

Client HTTP : Logiciel conçu pour se connecter à un serveur HTTP.

HTTP : Protocole de communication client-serveur.

Entité : Objet dont on confie l'enregistrement à l'ORM.

Attribut : Champ décrivant la structure interne d'une entité.

JSON : JavaScript Object Notation.

JWT : JSON Web Token.

REST : Representational State Transfert.

Cookie : Suite d'informations envoyée par un serveur HTTP à un client HTTP.

Requête : Interrogation de la base de données.

Annotation : Élément permettant d'ajouter des méta-données à un code source.

Bibliographie

JePilote : `jepilotemonentreprise.com`

Symfony : `symfony.com`

API-Platform : `api-platform.com`

Les-Tilleuls.coop : `les-tilleuls.coop`

JWT : `jwt.io`

JSON : `json.org`

Vuejs : `fr.vuejs.org`