

**Gaspard Lonchampt**

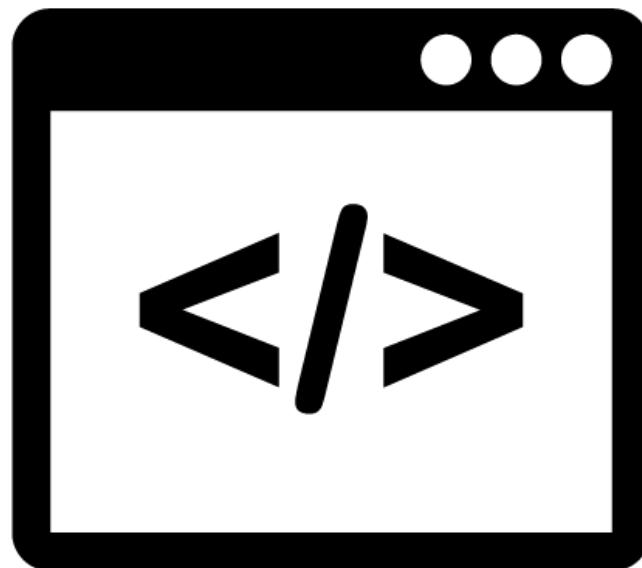
---

## Dossier de Projet Professionnel

*Projet application mobile : Radio grenouille*

*Gestion de la facturation manuelle : Jaguar-Network*

**Titre : Concepteur Développeur d'Applications**



<b>Introduction</b>	<b>4</b>
<b>Liste des compétences du référentiel couvert par les projets :</b>	<b>4</b>
<b>Résumé du dossier de projet en anglais</b>	<b>5</b>
<b>Présentations</b>	<b>6</b>
Présentation Gaspard Lonchampt	6
Présentation de La Plateforme_	6
Présentation Radio-Grenouille	6
Présentation Jaguar-Network	7
<b>Préparation du projet application mobile : radio-grenouille</b>	<b>8</b>
Descriptions des besoins	8
Analyse de l'existant	8
Organisation de travail	9
<b>Réalisation du cahier des charges et de la maquette</b>	<b>9</b>
Fonctionnalités à développer	9
Structure de l'application mobile	10
Design & Maquette	10
<b>Spécificités techniques</b>	<b>13</b>
Base de données et back-end	13
Création d'une API : framework API Platform	15
Front-end et API Transistor	19
Composant et cycle de vie	20
<b>Réalisation et extrait de code</b>	<b>23</b>
Navigation	23
Services GetPodcast	26
Contourner CORS avec un middleware	29
Déploiement :VirtualBox & Docker	31
Conclusion du projet Application Mobile : Radio Grenouille	38
<b>Projet Facturation : Jaguar-Network</b>	<b>39</b>
Présentation de l'entreprise	39
Mon rôle dans l'entreprise	42
Organisation de travail	43
<b>Spécificités techniques</b>	<b>44</b>
Technologie et modèle de données	44

<b>Réalisation et extrait de code</b>	<b>46</b>
Description et analyse de la tâche	46
SQL-ADMIN	48
Webservice	50
Intranet	54
Test du code	56
Conclusion du projet facturation : Jaguar-Network	56
Recherche en anglais	57
<b>Annexe</b>	<b>59</b>
Cahier de test	59

## I. Introduction

### A. Liste des compétences du référentiel couvert par les projets :

Les projets couvrent les compétences ci-dessous :

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :

- Maquetter une application
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web

2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :

- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données

3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :

- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches
- Développer une application mobile
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application

## B. Résumé du dossier de projet en anglais

This file summarizes two projects, one is front-end oriented and concerns the design and implementation of a mobile application on the model of an existing radio in web format: radio-grenouille. This includes a work of realization of specifications, mock-up, database design, creation of component, link to an existing API that hosts the podcast and deployment inside a virtual machine. I used React-Native framework along WSL environment for this project. The second project is back-end oriented and deals with the management of manual invoicing in the company where I am working as a student. This includes a rework of the intranet, a modification of the web service, creation of database functions and import integration into an accounting software. I used PHP, Perl and PostgreSQL along Linux environment for this project.

## II. Présentations

### A. Présentation Gaspard Lonchampt

Je m'appelle Gaspard Lonchampt, j'ai 28 ans et je vis à Marseille depuis maintenant 2 ans. J'ai travaillé comme auditeur financier grands comptes pendant 3 ans à Paris, Besançon et Marseille chez Deloitte et Mazars. J'ai souhaité faire une reconversion professionnelle en juillet 2020, c'est alors que j'ai pu intégrer le cursus Coding School de deux ans à La Plateforme. L'année scolaire 2021/2022 a donc été ma deuxième année en alternance chez Jaguar-Network consacrée à l'apprentissage de la conception et développement d'une application mobile à l'école ainsi que le travail en système d'information au travail. Je souhaite poursuivre au sein de l'entreprise en tant que salarié à plein temps.

## **B. Présentation de La Plateforme**

L'école la plateforme est une nouvelle école du numérique, qui vise à former un public venant de tous horizons, tout âge, et sans condition d'accès. Parmi les différentes formations proposées, celle de la coding school s'oriente sur le développement web et la conception d'applications mobile. La formation se déroule au rythme des apprenants, la pédagogie est orientée sur une autonomie des étudiants qui doivent rendre des projets permettant d'en débloquer des suivants, demandant plus de compétences. Ainsi, chacun peut évoluer à son rythme, et selon ses capacités/rapidité.

## **C. Présentation Radio-Grenouille**

Radio Grenouille est une radio associative culturelle diffusant ses programmes dans la région de Marseille.

Fondée en 1981, elle émet 24h/24 en modulation de fréquence, par voie numérique terrestre et en streaming sur Internet. La programmation est composée d'émissions musicales et d'éditoriaux de proximité axés sur les arts, des questions de citoyenneté et plus largement sur la culture.

Radio Grenouille est constituée d'une équipe de dix permanents et d'environ cent bénévoles. Elle est affiliée à Radio Campus France.

## **D. Présentation Jaguar-Network**

Appartenant au groupe Iliad, Jaguar Network met à disposition de ses clients, de la TPE aux grands comptes en passant par les marchés publics, son expertise en Hébergement, Cloud, Réseau, Téléphonie d'entreprise et Services Managés.

Son datacenter design Tiers IV, implanté à Marseille, répond sur 8000 m<sup>2</sup> aux spécificités les plus exigeantes en colocation. Jaguar Network produit des

solutions à forte valeur ajoutée en s'appuyant sur son réseau privé européen entièrement supervisé et une présence dans plus de 30 datacenters interconnectés. A partir de briques de solutions développées par son laboratoire R&D, Jaguar Network est en mesure d'offrir efficacité, robustesse et rapidité de déploiement, avec une proximité technique qui fait la différence.

Mon alternance s'est effectué au sein de l'équipe RFC (Ressources humaines, finance et commerce) où j'ai pu travailler sur ces sujets pendant 1 an.

### III. Préparation du projet application mobile : radio-grenouille

#### A. Descriptions des besoins

Radio Grenouille souhaite rendre sa radio accessible depuis un terminal mobile, en effet aujourd'hui il n'existe qu'un site internet avec un design responsive peu poussé. De plus, il souhaite améliorer l'accessibilité aux contenus recherchés par les auditeurs et auditrices via une organisation claire.

Le concept global de cette application était de proposer une première solution totalement responsive aux smartphones de type "preuve de concept" afin d'étudier la possibilité d'une V2.

#### B. Analyse de l'existant

- Le nom de domaine est à passer sous certificat SSL ;
- Charte graphique existante à base de vert ;

- Faire un nettoyage des sites web inactifs présents sur l'abonnement à l'hébergeur ;
- Faire un diagnostic des capacités serveurs : trafic/bande passante, capacité d'accueillir 3 flux radios, espace disque ;
- L'espace disque disponible pose 2 questions :
  - o Quelles pages et articles doivent faire la transition sur l'application mobile ;
  - o Quel est la capacité d'auto-héberger les podcasts pour ne plus dépendre d'hébergeur tiers spécialisé qui pourrait augmenter ses coûts (Transistor) ;

## C. Organisation de travail

Nous avons travaillé avec trois autres camarades de promotion sur le projet, nous avons réparti les tâches sous forme de cartes avec des statuts “en cours”, “à relire” et “terminé” dans un tableau de type kanban sur le logiciel Notion, qui nous a servis également de base de documentation avec le détail du plan de site, la charte graphique, police et maquette. Pour le versionnage, nous avons utilisé Git et GitHub avec des consignes sur les messages de commit, par exemple “[Features] - Exemple de features” lorsqu'il s'agit d'une nouvelle fonctionnalité ou bien “[Hotfix] - exemple de bug réglé” lorsqu'il s'agit d'un bug corrigé, nous avons travaillé chacun sur des branches et nous faisions des “pull request” avec des revues de code par un autre membre du groupe. Tout cela en “squashant” nos commits afin d'identifier clairement le travail de chacun dans l'historique grâce à la commande “git rebase”. Concernant l'écriture du code, nous avons adopté les conventions suivantes :

- Composant en PascalCase ;
- Nom des functions et variable en camelCase ;

## IV. Réalisation du cahier des charges et de la maquette

### A. Fonctionnalités à développer

Avant toutes choses, nous avons défini les fonctionnalités à développer à partir de l'existant :

- Créer un compte et se connecter ;
- Ajouter un programme et un épisode à ses favoris ;
- Écouter un podcast ;
- Consulter les programmes, épisode et la grille horaire des programmes ;
- Écouter le flux radio en direct ;

### B. Structure de l'application mobile

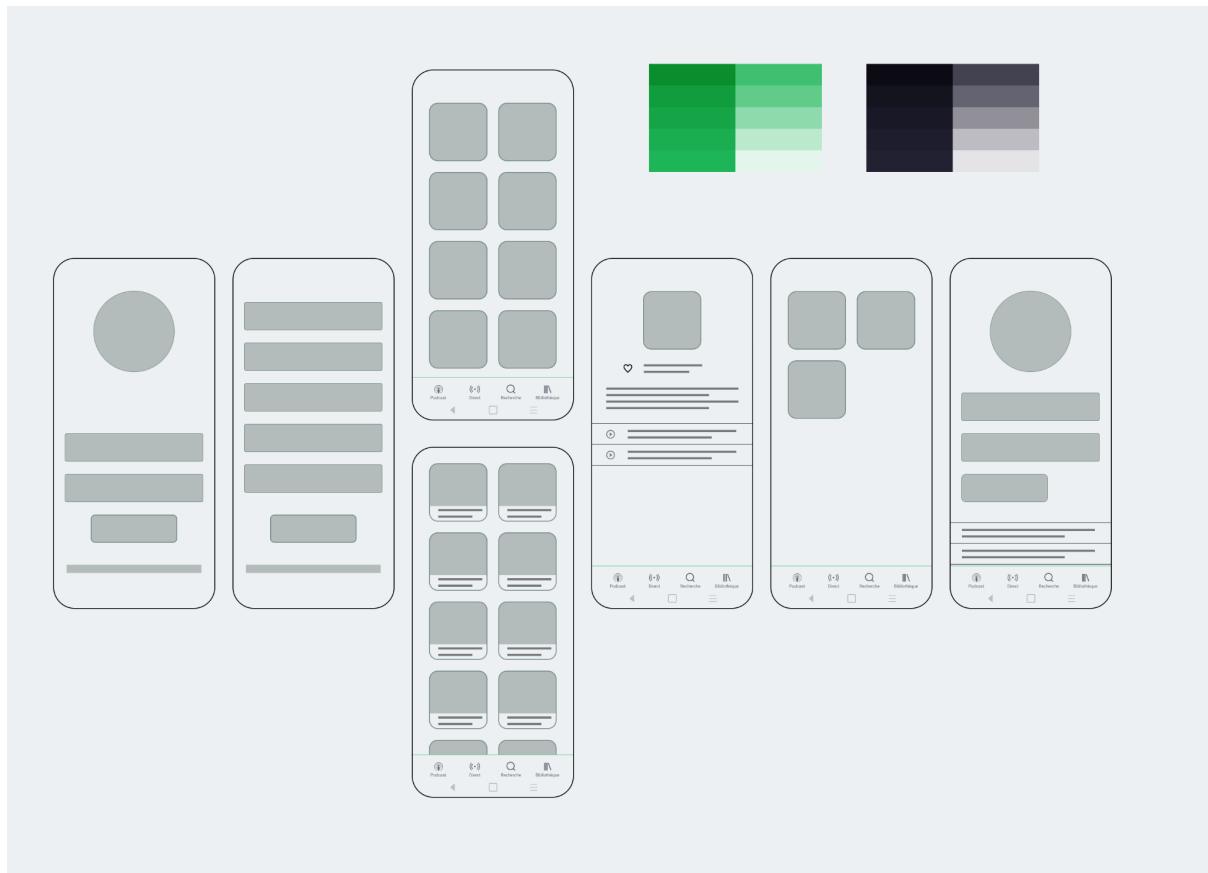
Après définition des fonctionnalités, le choix pour la structure du site a été le suivant :

- Une page affichant les podcasts ;
- Une page affichant les programmes ;
- Une page inscription et connexion ;
- Une page listant les épisodes du programme ;
- Un workflow “favoris” ;

### C. Design & Maquette

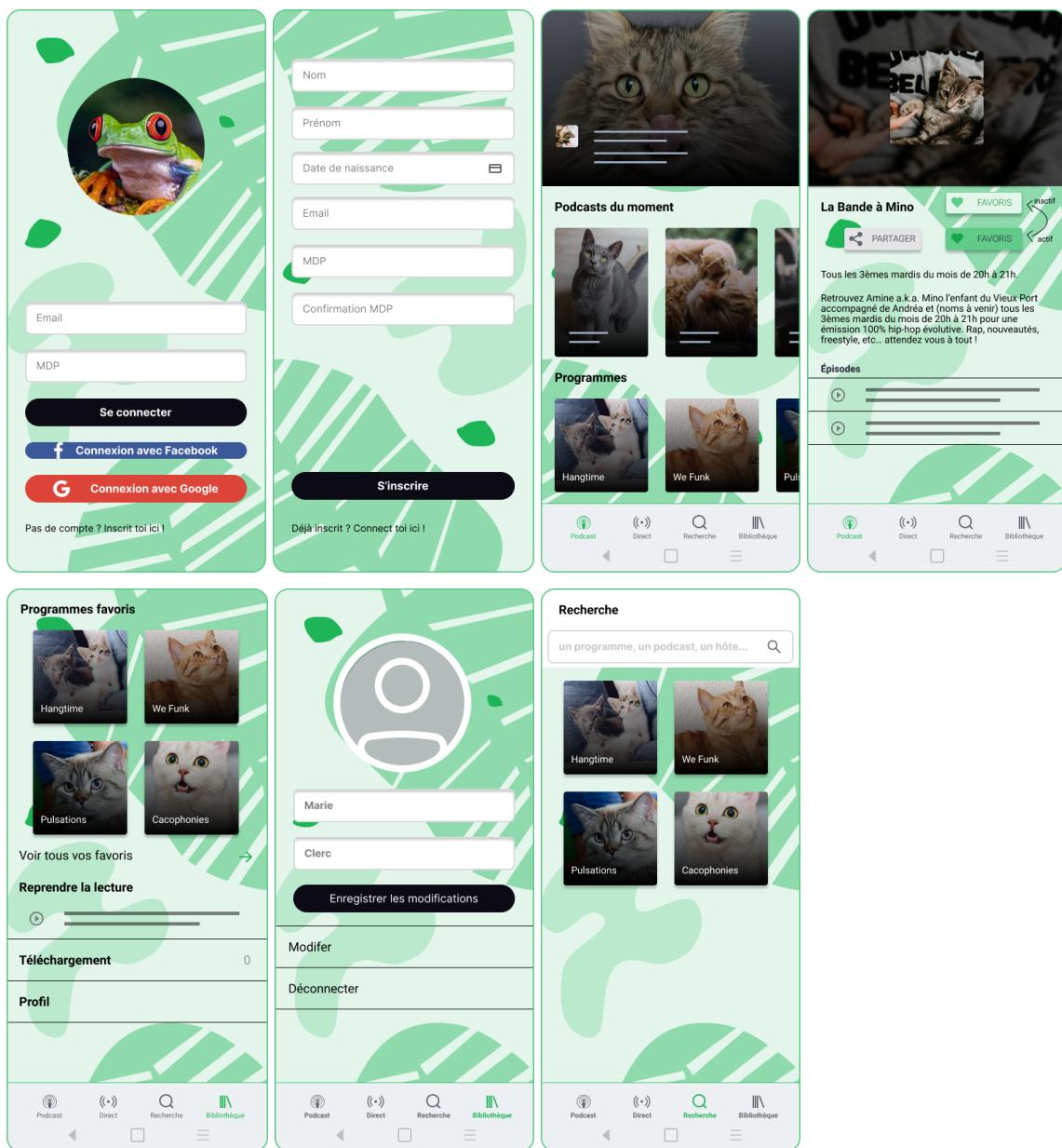
Nous avons utilisé Figma, un outil de maquettage et prototypage gratuit avec une version web.

En premier lieu, nous avons défini des wireframes avec un code couleur plutôt simple :



Les deux premières frames correspondent aux pages d'inscriptions et de connexion, puis les autres correspondent à la navigation parmi les programmes, la lecture d'un programme sélectionné, le profil, la recherche de programme ainsi qu'une navigation. Ces wireframes nous ont permis d'avoir une première ébauche du projet et d'affiner notre design en fonction des fonctionnalités dans la deuxième étape de conception : la maquette.

Dans l'ordre : une page connexion, inscription, accueil, détail programme, programme favoris, profil et recherche.



Cette maquette nous a permis de développer de façon efficiente l'application puisque nous avons tout de suite pu déterminer quelle serait les composants, image et code couleur en communs et donc le code réutilisable lors du développement front-end.

Après avoir conçu la maquette, nous avons entamé le développement de l'application mobile, nous nous sommes réparties les tâches selon les appétences de chacun. Faisant déjà beaucoup de back-end au travail, j'ai choisi de me concentrer principalement sur le front-end en communiquant avec l'API existante

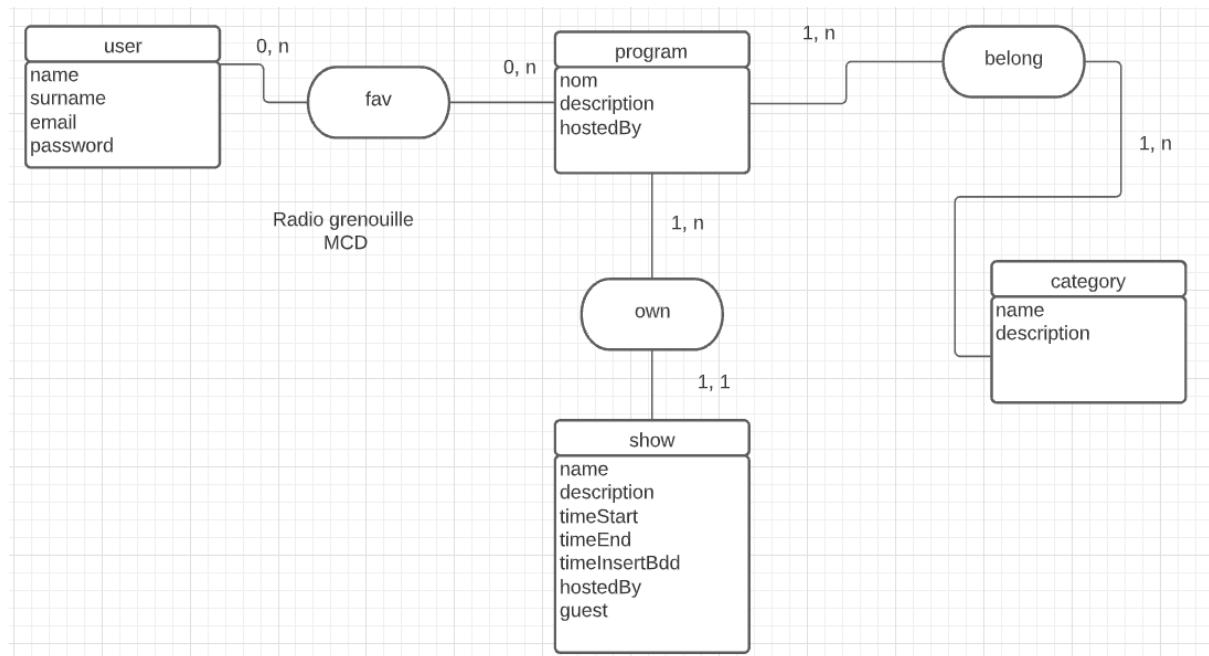
de Transistor après avoir mis en place la base de données et participer à la création de notre propre API.

## V. Spécificités techniques

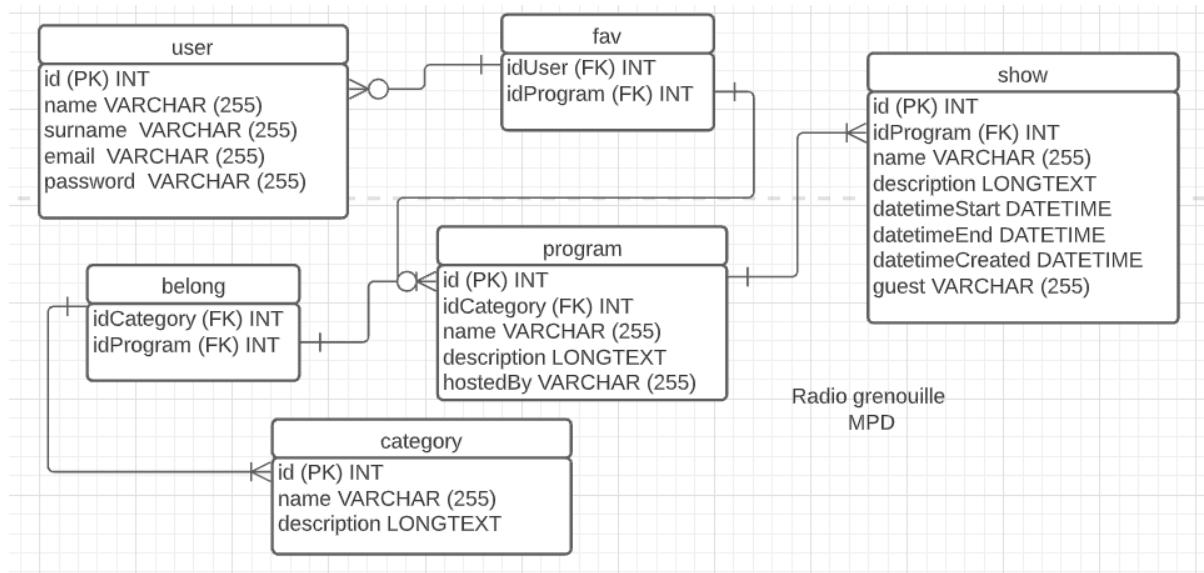
### A. Base de données et back-end

Dans le cadre du projet d'application mobile de Radio Grenouille, nous avons conçu et mis en place une base de données MySQL avec PhpMyAdmin pour l'interface graphique de gestion de la base de données. Durant le projet, nous avons géré l'interface administrateur via Symfony et la base de données avec l'ORM doctrine.

Tout d'abord, nous avons créé un modèle conceptuel de données (MCD) qui permet de classer les programme par catégorie, d'avoir plusieurs émissions par programme et la possibilité d'ajouter un programme en favoris par l'utilisateur. Ci-dessous illustré par un schéma :



Puis, nous avons créé un modèle physique de données (MPD) qui fait bien apparaître les deux tables de liaisons “fav” et “belong” puisque nous sommes dans le cas des cardinalités maximum N, autrement dit, si la relation est de type « many to many ». Elle ne contient pas à proprement parler des données : son rôle est d'organiser les rapports entre les éléments des tables qui, elles, les contiennent. Une table de liaison contiendra uniquement des propriétés correspondant aux clés primaires des deux entités, qu'elle associera deux à deux. De plus, ce modèle intègre le typage des données inséré dans les tables. Ci-dessous illustré par un schéma.



Une fois ces schémas de conceptions définis, nous avons créé les tables directement depuis l'ORM Doctrine intégrée dans Symfony après avoir configuré l'environnement MySQL.

Concernant la sécurité de la base de données, en premier lieu, nous avons installé le **SecurityBundle** de Symfony qui génère un fichier de configuration security.yaml. Ensuite, nous avons créé la classe “user” via l'extension **MakerBundle** avec le chiffrement de mot de passe. Pour protéger les inputs où il est possible de faire une tentative d'injection SQL qui sont ceux des formulaires de connexion et de l'inscription, il est recommandé de mettre en place des requêtes préparées via **PHP Data Objects (PDO)**.

## B. Création d'une API : framework API Platform

Le framework API Platform nous permet de créer une API avec Symfony. Nous avons plusieurs possibilités pour paramétrier notre API. Tous les paramètres se font directement sur l'entité via l'annotation API Ressource. Automatiquement, API Plateforme nous génère un CRUD pour toutes les entités. Nous avons la possibilité de paramétrier ce que nous renvoie l'API sur plusieurs critères. Nous pouvons effectuer des opérations sur les items (user/{id} par exemple) et sur les collections ( /users listes de tous les utilisateurs).

Concernant sur les données que nous renvoie l'API nous pouvons définir quels attributs recevoir en créant des groupes.

Il existe deux contextes pour la sérialisation des données : le contexte de normalisation et le contexte de dénormalisation :

- Le contexte de normalisation prend en compte toutes les opérations où l'on passe d'un objet vers du JSON. C'est la plupart du temps lorsque l'on souhaite récupérer de la données et la récupérer au format json côté front (GET).
- Le contexte de dénormalisation s'applique lorsque que l'on envoie des données aux formats JSON et qu'il est nécessaire de le transformer au format objet pour les faire persister en base de données (POST, PUT, PATCH).

Suivant ces différents contextes, les groupes et les choix des items ou collections, on peut facilement définir quelles données envoyer ou récupérer selon les opérations de l'utilisateur.

Nous également avons également ajouté des filtres grâce à l'annotation API filter, ce qui nous permet d'ajouter des paramètres dans la route et ainsi récupérer un résultat selon certaines conditions.

```
#[ApiFilter(SearchFilter::class, properties: ['id' => 'exact', 'programs.id' => 'exact'])]

#[Groups(['read:User'])]
private $email;

#[ORM\Column(type: 'json')]
#[Groups(['read:User'])]
private $roles = [];

#[ORM\Column(type: 'string')]
#[Assert\NotBlank(
| message: 'Veuillez remplir ce champ.'
)]
private $password;

#[ORM\Column(type: 'string', length: 255)]
#[Assert\NotBlank(
| message: 'Veuillez remplir ce champ.'
)]
#[Groups(['read:User'])]
private $name;

#[ORM\Column(type: 'string', length: 255)]
#[Assert\NotBlank(
| message: 'Veuillez remplir ce champ.'
)]
#[Groups(['read:User'])]
private $surname;

#[ORM\ManyToMany(targetEntity: Program::class, mappedBy: 'users')]
#[ORM\JoinTable(name:'`favoriteBy`')]
#[Groups(['read:User'])]
private $programs;
```

On voit ici tous les attributs qui appartiennent au groupe read:User, ainsi si on récupère la liste des utilisateurs, le mot de passe ne sera pas retourné.

On peut également ajouter une nouvelle route pour une opération spécifique, en la reliant directement à un controller Symfony.

Nous avons réalisé un système de like sur l'application, l'utilisateur à la possibilité d'ajouter en favoris un programme en particulier.

Dans un premier temps, nous avons créé un controller et la méthode "fav" :

```
#Route('/api/add', name : 'fav')
public function fav(Request $request, UserRepository $userRepository, ProgramRepository $programRepository, EntityManager $entityManager)
{
    $contentBody = $request->getContent();
    $body = json_decode($contentBody);

    $idProgram = $body->id_program ;
    $idUser = $body->id_user;

    $entityManager = $doctrine->getManager();

    $user = $userRepository->find($idUser) ;
    $program = $programRepository->find($idProgram);

    $user->addProgram($program);

    $entityManager->flush();

    return $this->json([
        'add' => 'Votre programme à bien été ajouter !',
    ]);
}
```

Nous pouvons l'ajouter par la suite à mon API directement depuis l'entité User via l'annotation Api Ressource.

```

#[ApiResource(
    collectionOperations: [
        "me" => [
            'pagination_enabled' => false,
            'path' => 'api/me',
            'method' => 'get',
            'controller' => MeController::class,
            'read' => false
        ],
    ],
    itemOperations: [
        "addFav" => [
            'controller' => AddFavController::class,
            'path' => '/api/add',
            'method' => 'patch',
            'read' => false
        ],
    ],
    normalizationContext: ['groups' => ['read:User']]
)]

```

Cela permet également d'ajouter les différentes routes à la documentation Swagger :

The screenshot shows the Swagger UI interface for the 'User' resource. At the top, it says 'User'. Below that, there are two operation cards: one for PATCH /api/add labeled 'Updates the User resource.' and another for GET /api/me labeled 'Retrieves the collection of User resources.'

Nous avons utilisé Swagger pour documenter notre API. Il permet d'avoir la liste des différentes routes de notre application avec une description personnalisée. L'outil permet aussi de tester les différentes routes, et voir quels sont les paramètres attendus dans le body et la réponse retourner en JSON pour chacune des routes. C'est donc une partie importante du projet, car il permet à nous, ou un nouveau développeur sur le projet, d'avoir une vision globale de l'API.

Concernant la sécurité de l'API, il existe deux types principaux d'attaque potentielle :

- **Un déni de service** (Denial of Service ou DoS) est une attaque réalisée dans le but de rendre indisponible les services. En effet, une attaque DoS fonctionne en épuisant une ressource limitée dont une API a besoin pour répondre aux demandes légitimes. En inondant une API de fausses

requêtes, ses ressources sont bloquées pour répondre à ces requêtes et pas aux autres.

- **Une attaque brute force**, c'est lorsque l'attaquant utilise des outils pour envoyer un flux continu de requêtes à une application ou une API – afin de tester toutes les combinaisons possibles d'un paramètre, via un processus d'essais et d'erreurs pour espérer deviner juste. Les objectifs peuvent être multiples : brute force d'un formulaire d'authentification dans le but de voler un compte, brute force d'un identifiant pour récupérer des données sensibles, brute force d'un secret, etc.

Pour palier à ces attaques, il convient d'implémenter **des mécanismes de Rate Limiting**. Ils permettent de protéger les APIs et les autres services contre une utilisation excessive et abusive, dans le but d'assurer leur disponibilité. Il s'agit d'anticiper le fait qu'un ou plusieurs client(s) – système – peuvent utiliser plus que leur « juste » part d'une ressource, via l'envoi de requêtes. Et en limitant le nombre de requêtes qu'un utilisateur donné est autorisé à envoyer dans un laps de temps défini, on peut réduire le risque d'attaque DoS ou d'attaque brute force.

Pour cela, il existe un bundle Symfony nommé “Rate Limiter” avec une configuration personnalisable du nombre de requêtes autorisé sur une période de temps avec trois stratégies différentes : Fixed Window Rate Limiter, Sliding Window Rate Limiter et Token Bucket Rate Limiter.

## C. Front-end et API Transistor

La partie front-end de l'application a été réalisé avec le framework React Native. Une technologie qui nous permet d'utiliser React avec les fonctionnalités natives d'appareil Android et IOS.

Il nous permet donc de créer une application adaptable sur différents supports. L'application est structurée avec plusieurs dossiers :

- Un dossier **assets** dans lequel sont stockées toutes les images et les font

- Un dossier **src** dans lequel on retrouve plusieurs dossiers :
  - A. Un dossier **component** qui regroupe tous les composants de l'application
  - B. Un dossier **screens** avec les différents écrans de l'application
  - C. Un dossier **navigation** qui définit les routes vers les différents composants
  - D. Un dossier **services** pour les appels à l'API Transistor

L'application est découpée sous forme de composants. Ils sont regroupés dans le dossier src/components. Nous avons privilégié une approche fonctionnelle avec l'utilisation des Hooks.

Pour ma part, je me suis occupé de la partie navigation et podcast avec un appel à l'API existante : Transistor.

Transistor est un hébergeur de podcast qui facilite le déploiement de ceux-ci sur les différentes plateformes comme Apple Podcasts, Spotify et Google Podcasts. Il génère automatiquement une page web et un panel administrateur pour la gestion analytique des écoutes. C'est un excellent outil "No code" pour ceux qui souhaitent se lancer dans la création de Podcast. Il dispose d'une documentation technique de l'API qui m'a permis de m'adapter facilement à l'outil.

Pour la Navigation, j'ai utilisé la librairie React-Navigation qui permet l'implémentation de fonctionnalités de navigation dans React-Native.

Pour simuler un écran de téléphone, nous avons utilisé Expo.

## D. Composant et cycle de vie

Abordé précédemment, il s'agit ici d'expliquer plus en détail le fonctionnement d'un composant React. Celui-ci dispose d'un cycle de vie divisé en quatre étapes :

**Initialisation** : C'est l'étape où le composant est construit avec les "Props" et l'état par défaut donné. Cela se fait dans le constructeur d'une classe de composants.

**Montage** : le montage est l'étape de rendu du JSX renvoyé par la méthode de rendu elle-même.

**Mise à jour** : La mise à jour est l'étape où l'état d'un composant est mis à jour et l'application est repeinte.

**Démontage** : comme son nom l'indique, le démontage est l'étape finale du cycle de vie du composant où le composant est supprimé de la page.

React nous donne plusieurs méthodes au sein d'un composant permettant d'interagir à un moment précis de son cycle de vie.

En premier lieu, la méthode **componentWillMount()** est appelé juste avant que le composant ne soit monté par le DOM. D'autre part, la méthode **componentWillUpdate()** est appelé lorsque le composant se reconstruit lors de la modification d'un state. Enfin, la méthode **componentWillUnmount()** permet d'agir lorsque le composant est démonté du DOM, dès lors qu'il est supprimé de la page et marque la fin du cycle de vie.

Toutes ces méthodes sont utilisées avec React lorsque que le composant est sous forme de classe. Dans notre projet, tous les composants ont une approche fonctionnelle, c'est là qu'intervient l'utilisation des "hooks".

Le "hook **useEffect**" remplace les méthodes ci-dessus, c'est une méthode qui est appelée à chaque nouvel affichage de notre composant. On va donc généralement implémenter notre requête API à l'intérieur de la méthode pour avoir un rendu après avoir récupéré les données.

Il peut être appelé à chaque nouvel affichage, ou seulement lors du premier montage, ou selon un événement précis, comme le clic d'un bouton par exemple.

On peut implémenter cela en ajoutant un second paramètre à “useEffect”, de plus, il est directement déclaré dans le composant, ainsi il a directement accès à toutes les “props” passé au composant.



```
useEffect(() => {
  fetch(URL, requestOptions)
    .then(response => response.json())
    .then(responseJson) => {
      setData(responseJson);
      setLoading(false);
    }
    .catch(error => console.log('error', error));
}, [])
```

Dans le cas de l'image ci-dessus abordé plus en détail dans la partie “Services GetPodcast”, la méthode “fetch” représente la requête API pour récupérer toutes les données des programmes. Elle sera appelée qu'une seule fois lors du premier montage du composant, car on lui renseigne un tableau vide en paramètre.

Le “**hook useState**”, permet quant à lui de déclarer tous les états et de créer des setters associés. Dès lors que l'on appelle un setter, le composant est directement mis à jour avec un nouvel affichage.

L'utilisation des “hooks” est donc très puissante puisqu'ils nous permettent de travailler avec les états et le cycle de vie du composant sans passer par un composant sous forme de classe, ce qui fût un choix technique afin d'appréhender plus facilement le framework **React-Native** : en effet, la documentation était beaucoup plus fournie et les exemples plus nombreux sur le web.

## VI. Réalisation et extrait de code

### A. Navigation

Le développement de la navigation en premier m'a semblé pertinent puisque cela m'a permis ensuite de naviguer entre les différentes pages que je développais. Pour cela, j'ai utilisé la librairie React-Navigation qui permet de gérer la présentation et la transition entre plusieurs écrans.

En premier lieu, j'ai importé les modules nécessaires :

- Bottom-tabs de React-Navigation qui permet une navigation sous forme d'onglet en bas de l'écran ;
- Material community icons qui permet d'avoir une liste d'icône disponible ;
- Les différents composants que j'ai utilisés dans mes routes ;

```
● ○ ●  
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';  
import { MaterialCommunityIcons } from '@expo/vector-icons';  
import Podcast from '../components/Podcast';  
import Library from '../components/Library';  
import Search from '../components/Search';  
import Direct from '../components/Direct';
```

Ensuite, j'assigne le composant “createBottomTabNavigator” dans une constante “Tab” que je réutilise plus bas dans la fonction “Navigations” qui sera exporté dans le composant parent “App.js”.

Dans la fonction Navigations j'appelle tout d'abord le nœud “Navigator” qui servira de composant parent aux différents composants enfants à la propriété “Screen”. Un nœud React est une représentation virtuelle, légère, sans état, immuable, d'un nœud DOM.

```

const Tab = createBottomTabNavigator();

function Navigations() {
  return (
    <Tab.Navigator
      initialRouteName="Podcast"
      screenOptions={{
        tabBarActiveTintColor: '#1DB558',
        tabBarLabelPosition: 'below-icon'
      }}
    >
      <Tab.Screen
        name="Podcast"
        component={Podcast}
        options={{
          tabBarLabel: 'Podcast',
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="podcast" color={color} size={size} />
          ),
        }}
      />
      <Tab.Screen
        name="Library"...
        ...
      />
      <Tab.Screen
        name="Direct"...
        ...
      />
      <Tab.Screen
        name="Search"...
        ...
      />
    </Tab.Navigator>
  );
}

export default Navigations

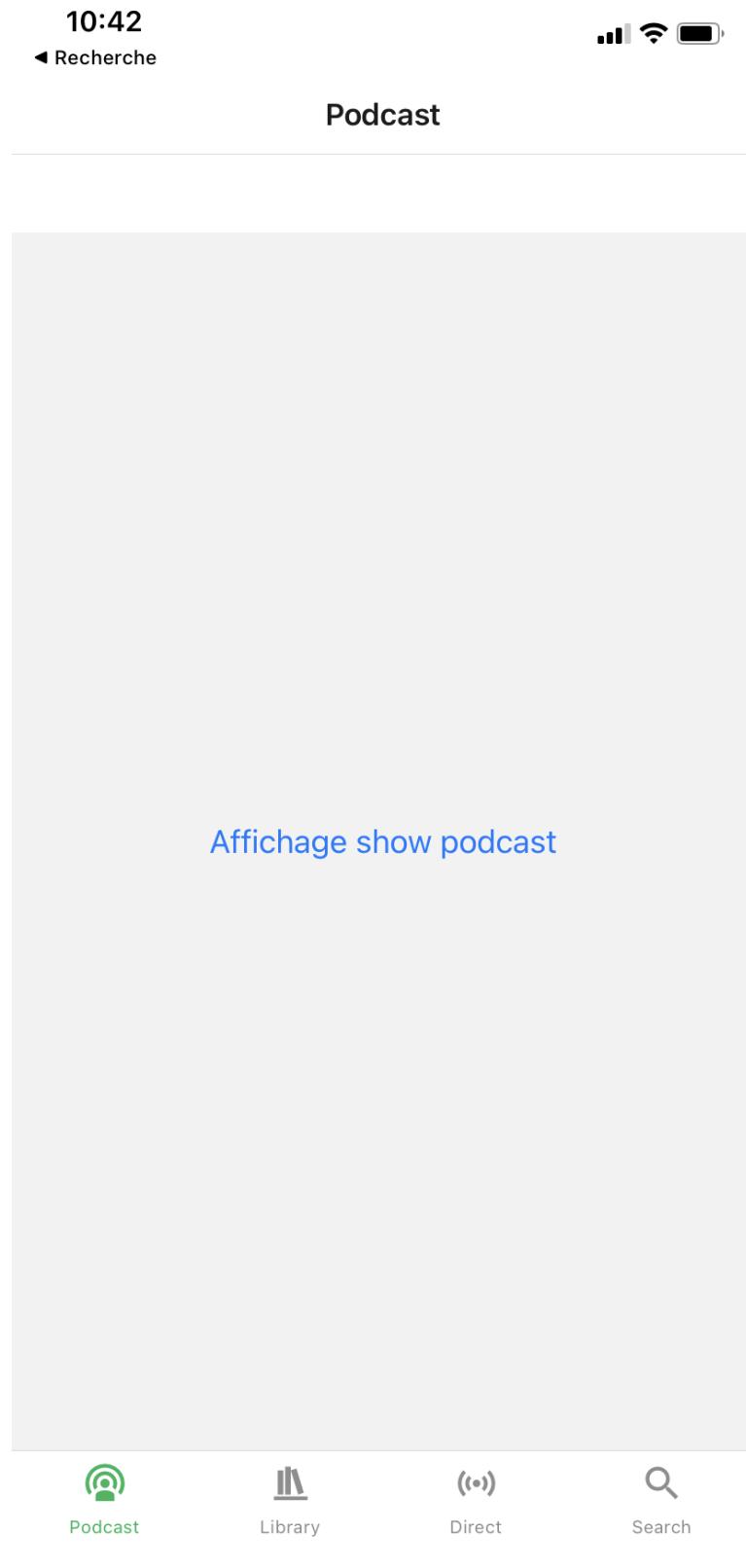
```

On constate plusieurs paramètres important pour le nœud ‘Navigator’ :

- **initialRouteName** : permet de définir la page par défaut sur laquelle l’application va s’ouvrir, ici la page Podcast.
- **screenOptions** : permet de personnaliser l’affichage des écrans, ici j’ai défini une couleur lorsque l’onglet est actif et j’ai mis le texte en dessous des icônes.

Dans les composants enfants avec la propriété “Screen”, on va importer les composants associés à l’écran, dans l’exemple, ce sera le composant “Podcast”, on va le nommer et lui passer des paramètres tel que le label et une icône représentant un podcast.

Ci-dessous le résultat de la Navigation :



## B. Services GetPodcast

Pour les appels à l'API d'hébergement de podcast Transistor, j'ai développé le service "GetPodcast" qui prend comme argument un objet "props". En effet, lorsque **React-Native** rencontre un élément représentant un composant défini par l'utilisateur, il transmet les attributs JSX et les enfants à ce composant sous la forme d'un objet unique (à la manière des arguments d'une fonction). L'API de Transistor est construite autour de la spécification JSON:API. Les points de terminaison (endpoints) acceptent des corps de requête JSON ou codés par formulaire, puis renvoient des réponses codées en JSON et utilisent des codes de réponse HTTP standard.

Dans notre cas où il s'agit de récupérer des données et de les afficher, nous utiliserons la méthode de requête HTTP "GET" et l'objet "props" correspondra à la terminaison de l'URI souhaité.

Ci-joint un extrait du code que je vais commenter :

```
function GetPodcast(props) {  
  
    var myHeaders = new Headers();  
    myHeaders.append("x-api-key", "randomkey");  
  
    var requestOptions = {  
        method: 'GET',  
        headers: myHeaders,  
        redirect: 'follow',  
    };  
  
    const host = 'https://proxy-grenouille.herokuapp.com/';  
    const type = props.route.params.route;  
  
    let podcast = type.toLowerCase().trim();  
    const URI = host.concat(podcast);  
  
    const [data, setData] = useState(null);  
    const [loading, setLoading] = useState(true);  
    var content;  
  
    useEffect(() => {  
        fetch(URI, requestOptions)  
            .then(response => response.json())  
            .then((responseJson) => {  
                setData(responseJson);  
                setLoading(false);  
            })  
            .catch(error => console.log('error', error));  
    }, [])  
  
    if(loading) {  
        content = <ActivityIndicator />  
    } else {  
        if (type == 'shows') {  
            return (  
                <PodcastShows data={data} />  
            )  
        } else if (type == 'episodes') {  
            return <PodcastEpisodes data={data} />  
        } else {  
            alert("ceci n'existe pas");  
        }  
    }  
    return (  
        <View>  
            {content}  
        </View>  
    )  
};  
  
export default GetPodcast
```

Tout d'abord, on définit les requestOptions de l'appel API :

- Méthode HTTP GET ;
- Headers avec la clé API ;
- Redirection ‘follow’ afin de rediriger le code du statut HTTP ;

Dans un second temps, on définit l'URI où l'appel va être dirigé qui est une concaténation de la “props” correspondant à la terminaison de l'URI souhaité avec l'URL du middleware que j'utilise dans le but d'éviter les problèmes de CORS dont je parlerais dans la partie suivante.

Puis, on détermine deux hooks **useState** et une variable :

- La constante **data, setData** initialement à nul afin de récupérer la réponse de Transistor ;
- La constante **loading, setLoading** initialement à true qui nous servira de repère concernant le chargement des données ;
- La variable **content** qui contiendra une icône de “spinner” qui s'affichera le temps de chargement des données ;

Viens ensuite le hook **useEffect**, celui-ci prend comme paramètre un tableau vide, ce qui signifie que les méthodes appelées en son sein seront appelée qu'une seule fois lors du premier montage du composant comme expliqué précédemment.

Pour faire la requête API j'ai décidé d'utiliser l'API Fetch qui est extrêmement bien documenté et facile à prendre en main. Celle-ci prend comme argument l'URI et les requestOptions et retourne une promesse qui résout la “response” de cette requête, qu'elle soit couronnée de succès ou non. Si la “response” est un succès, les données récupérées seront converties au format JSON et passé au useState data afin de les afficher et le useState loading passera à false.

Enfin, tant que “loading” est à “true” le spinner s'affichera, une fois passé à “false” et que les données ont bien été récupérés, celles-ci seront envoyées au

composant correspondant au type de terminaison de l'URI, soit “show” ou “episodes”.

## C. Contourner CORS avec un middleware

Lorsque notre front-end envoie des requêtes à un back-end qui se trouve dans un domaine différent, nous devons faire face au Cross-Origin Resource Sharing (CORS).

Notre interface (<http://localhost:19006>) et l'API de Transistor (<https://api.transistor.fm/v1>) se trouvent dans des domaines différents. Le navigateur applique donc la politique de “l'origine commune” et bloque toute demande allant du premier vers le second.

Selon la documentation de Mozilla, le partage des ressources entre origines (Cross-Origin Resource Sharing, CORS) est un mécanisme qui utilise des en-têtes HTTP supplémentaires pour indiquer à un navigateur qu'il doit autoriser une application Web exécutée à une origine (domaine) à accéder à des ressources sélectionnées sur un serveur à une origine différente. Une application Web effectue une requête HTTP inter-origine lorsqu'elle demande une ressource dont l'origine (domaine, protocole et port) est différente de la sienne.

Étant donné que les en-têtes CORS doivent être inclus dans la réponse que nous recevons du serveur et que nous n'avons aucun contrôle sur l'API de Transistor, nous devons construire notre propre serveur proxy qui transmet les demandes à l'API de Transistor, mais qui émet les bons en-têtes CORS afin que le navigateur ne bloque rien. Nous devrons ensuite apporter des modifications à notre client afin qu'il accède à notre serveur plutôt qu'à celui de Transistor.

Pour cela, j'ai choisi de créer un middleware en utilisant Express : un framework d'application web Node.js simple et flexible.

J'ai installé deux modules nécessaires à la création de ce serveur proxy :

- Axios qui est un client HTTP basé sur les promesses pour node.js et le navigateur ;
- Cors qui est un paquet node.js qui permet d'activer CORS avec différentes options ;

```
● ○ ●
const express = require('express')
const app = express()
const axios = require('axios')
const cors = require('cors')
```

Ensute j'ai défini un port sur lequel le serveur doit démarrer, j'ai utilisé le paquet Cors pour limiter quelle origine peut faire des requêtes à notre serveur :

- soit un port spécifique de notre ordinateur lorsque le serveur est déployé sur Heroku ;
- soit “\*”, tous, en environnement de développement ;

```
● ○ ●
let port = process.env.PORT;
if (port == null || port == "") { port = 3001; }

app.listen(port, function () {
    console.log("Server has started successfully!");
});

app.use(cors({
    origin: process.env.ORIGIN || "*"
}))
```

Pour finir, je fais ma requête API à Transistor avec son URL définis en variable d'environnement concaténé avec l'URL et la clé API récupéré à partir de la requête de mon application mobile vers le middleware.

```
app.get('★', (req, res) => {

    let endpoint = process.env.API_BASE_URL
    let uri = req.originalUrl;
    let headers = req.get('x-api-key');

    axios.get(endpoint.concat(uri), {
        headers: {
            'x-api-key': headers
        }
    }).then(response => {
        res.json(response.data)
    }).catch(error => {
        res.json(error)
    })
})
```

Concernant la sécurité, il est évident que ce middleware est à utiliser qu'en phase de développement et qu'une fois déployé sur un serveur comme Heroku cela peut poser des problèmes de failles type DDoS et Brute Force. C'est pour cela que travaillé avec ce serveur proxy uniquement en environnement local, sur mon ordinateur, je l'ai déployé sur Heroku uniquement pour m'exercer puis aussitôt désactivé.

## D. Déploiement :VirtualBox & Docker

Le déploiement est la mise en production d'un service informatique. Lorsque l'on développe un site internet sur un serveur local, il faut par la suite le déployer sur un serveur web.

La majeure partie des serveurs web sont réalisés sous le système d'exploitation Linux. C'est pourquoi j'ai choisi de déployer le back-end de l'application sous

Linux via une machine virtuelle. L'hyperviseur, l'outil VirtualBox développé par Oracle me semblait le meilleur choix pour m'exercer. Il s'agit d'un logiciel libre permettant de créer un environnement virtuel isolé sur la machine hôte. Avant de pouvoir installer une machine virtuelle sous Linux, il faut télécharger le système d'exploitation. La distribution Debian est préconisée, car stable. Il faut télécharger une version stable de Debian (pas la version testing), au format ISO. Une fois Debian installé et mis en route, il est nécessaire d'installer Grub, qui est le gestionnaire de démarrage de Linux ainsi que Xfce, un environnement de bureau léger pour les systèmes d'exploitation Linux. Il est rapide et léger, tout en étant visuellement attractif et facile d'utilisation. Il est recommandé d'utiliser Xfce pour une meilleure visualisation des tâches effectuées sur la VM.

Pour pouvoir administrer la machine virtuelle, il faut se connecter à un serveur à l'aide du protocole SSH. Pour cela je dois d'abord en changer l'IP, je rajoute donc un accès par pont pour l'adapter à la carte réseau de ma machine physique, puis je change le port de 22 à 2222 via la commande :

```
● ● ●  
sudo vim /etc/ssh/sshd_config
```

Enfin, je rajoute le port 2222 en port autorisé pour la connexion via la commande :

```
● ● ●  
sudo /sbin/iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 2222 -j ACCEPT.
```

Afin d'avoir un serveur web, il est nécessaire d'installer différents services.

Pour Linux il y a LAMP, qui est en fait un ensemble de logiciels à installer séparément.

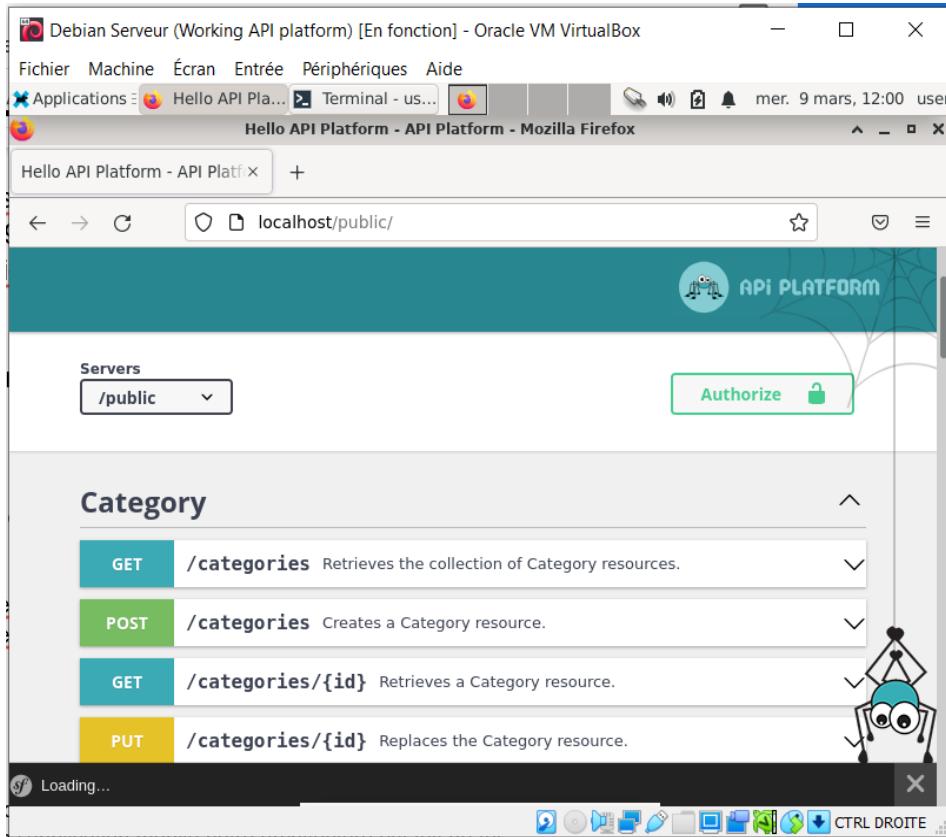
Les différents services à installer sont :

- Apache ;
- La dernière version de PHP disponible ;
- Mariadb server ;
- Phpmyadmin ;
- Les différents modules/connecteurs.

Une fois un utilisateur créé pour la base de données, il est temps de déployer le site :

- Création de la base de donnée API, git clone du projet API dans /var/www/api après avoir rajouté la clé public ssh de la VM sur mon compte github ;
- Changement du document root dans /etc/apache2/sites-available/000-default.conf en '/var/www/api' ;
- Installation de composer via le site officiel doc, car il manque le fichier composer.phar via le gestionnaire de paquet Debian ce qui provoque des erreurs pour les commandes suivantes ;
- Exécution de “composer install” pour installer les différents modules nécessaires à Symfony et API platform ;
- Commandes Symfony pour faire la migration de la base de données via l’ORM Doctrine.

Voici le résultat :



Cependant, il est nécessaire de rajouter une couche de sécurité : un certificat SSL.

Pour pouvoir utiliser un certificat SSL *quel qu'il soit*, nous devons d'abord activer mod\_ssl, un module Apache qui prend en charge le cryptage SSL.

Maintenant qu'Apache est prêt à utiliser le cryptage, nous pouvons passer à la génération d'un nouveau certificat SSL. Le certificat stockera quelques informations de base sur notre site et sera accompagné d'un fichier clé qui permet au serveur de traiter les données cryptées en toute sécurité.

Nous pouvons créer les fichiers de clés et de certificats SSL avec la commande openssl.

Après avoir entré la commande, vous serez amené à une invite où vous pourrez entrer des informations sur votre site web.

La liste complète des invites ressemblera à ceci :

Country Name (2 letter code) [XX]:FR  
State or Province Name (full name) []:Example  
Locality Name (eg, city) [Default City]:Example  
Organization Name (eg, company) [Default Company Ltd]:Example Inc  
Organizational Unit Name (eg, section) []:Example Dept  
Common Name (eg, your name or your server's hostname)  
[]:your\_domain\_or\_ip  
Email Address []:webmaster@example.com

Maintenant que nous disposons de notre certificat et de notre clé auto-signés, nous devons mettre à jour notre configuration Apache pour pouvoir les utiliser.

Applications Terminal - mar. 21 juin, 12:28 root

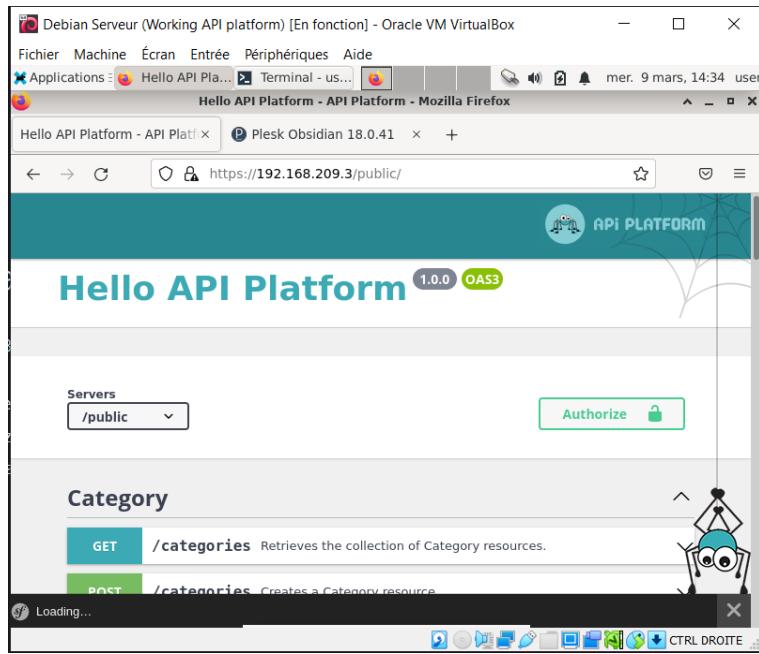
Applications Fichier Edition Affichage Terminal Onglets Aide

```
<VirtualHost *:443>
    ServerName 192.168.209.3
    DocumentRoot /var/www/api

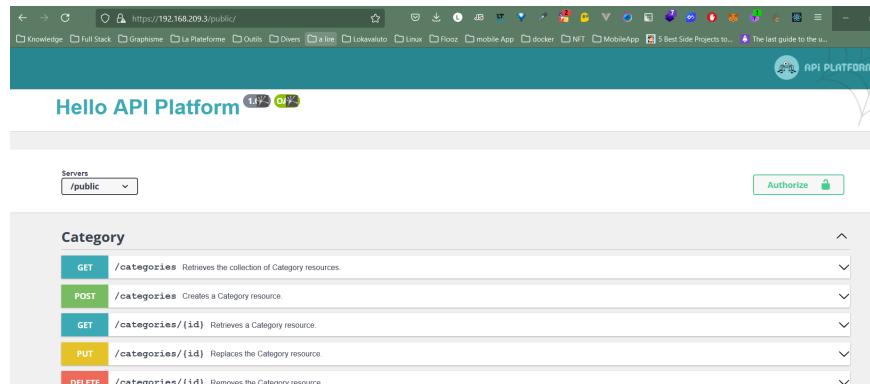
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
</VirtualHost>
~
```

"patate.conf" 8L, 232B 1,1 Tout

Voici le résultat côté machine virtuelle :



Ainsi que du côté machine hôte :



Pour ce qui est du paramétrage, notre application mobile est déjà configurée pour accéder à notre API hébergé sur Heroku, je me permets donc de passer cette étape puisque déjà réalisé dans un environnement de développement cloud sur le lien suivant : <https://intense-springs-77079.herokuapp.com/>.

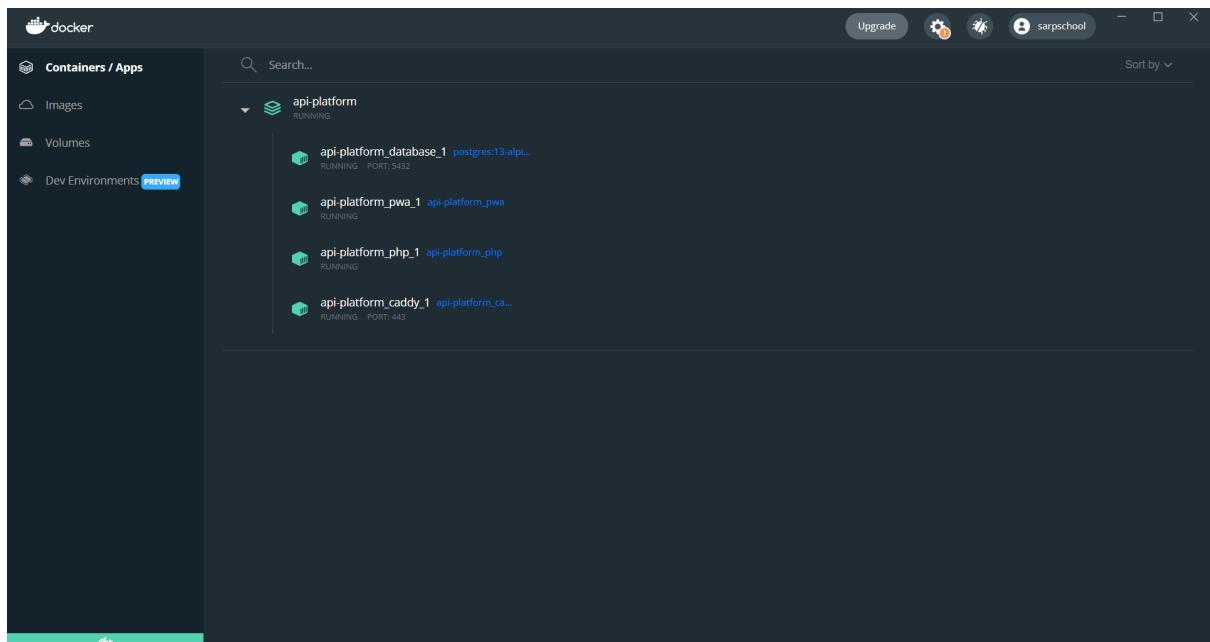
En effet, le déploiement sur une machine virtuelle étant pour l'exercice et par curiosité, car je travaille sur des "VM" quotidiennement au travail, je voulais alors mieux comprendre leur fonctionnement.

Le déploiement sur Heroku ne nécessite pas d'explications supplémentaires, il est extrêmement facile via son CLI et il suffit de suivre la documentation pour que l'opération ne prenne que quelques minutes.

Lors de la création du projet, j'ai également utilisé Docker pour créer un environnement de travail commun à tous en suivant la documentation d'API Platform avec les services suivant :

- caddy : un serveur web ;
- php : L'API avec PHP 8 et composer ;
- pwa : une application web Next.js avec un panel administrateur d'API platform pré-installé ;
- database : un serveur de base de données PostgreSQL ;

Ci-joint un exemple des microservices en état de fonctionnement :



Son principal avantage est le déploiement rapide, Docker utilise le noyau Linux et des fonctions de ce noyau pour séparer les processus afin qu'ils puissent s'exécuter de façon indépendante en créant un conteneur pour chaque processus, de ce fait, nous pouvons rapidement partager les processus similaires avec les

nouvelles applications. De plus, comme nous n'avons pas besoin de redémarrer le système d'exploitation pour ajouter ou déplacer un conteneur, le délai de déploiement s'en trouve encore réduit. Pour résumer, la technologie Docker propose une approche plus granulaire, contrôlable et basée sur des microservices, qui place l'efficacité au cœur de ses objectifs.

Malheureusement, à la création du projet, nous n'avions pas accès à un hébergeur gratuit prenant en charge Docker pour le déploiement en ligne, nous avons donc opté pour un développement en local et un déploiement de l'API sur Heroku.

## E. Conclusion du projet Application Mobile : Radio Grenouille

Durant cette réalisation, j'ai pu découvrir plusieurs éléments :

- Le framework React Native ;
- L'architecture en composant et leur cycle de vie ;
- Créer et travailler avec des API ;
- Le déploiement d'une machine virtuelle et de conteneurs ;

Même si toutes ces notions ne sont pas maîtrisées parfaitement, cela constitue un premier pas vers l'apprentissage du développement d'application mobile qui me servira dans ma vie professionnelle.

Ce projet m'a apporté les compétences du référentiel ci-dessous :

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :

- Maquetter une application
- Développer la partie front-end d'une interface utilisateur web

2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :

- Concevoir une base de données

- Mettre en place une base de données

3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :

- Concevoir une application
- Construire une application organisée en couches
- Développer une application mobile
- Préparer et exécuter le déploiement d'une application

## VII. Projet Facturation : Jaguar-Network

### A. Présentation de l'entreprise

Jaguar Network a été fondée le 11 septembre 2001 par Kevin POLIZZI alors qu'il était encore étudiant.

Société aujourd'hui renommée dans le domaine des télécommunications, elle a vu son activité évoluer au fil des années.

L'entreprise opérait initialement dans l'hébergement de sites Web jusqu'à ce que son activité se précise en janvier 2006. Jaguar Network devient alors un Opérateur Télécom via l'obtention de la licence L33-I, à l'instar des plus grands opérateurs français comme Orange ou SFR. Ce changement a permis de couvrir la quasi-totalité des besoins des entreprises en proposant toute une gamme de solutions et services, dont une grande proportion d'offres sur mesure basées autour du Datacenter, de la Voix, du Cloud et du Réseau.

L'entreprise rassemble près de 300 collaborateurs rassemblés sur huit sites en France :

Marseille, Paris, Sophia Antipolis, Annecy, Lyon, Nantes, Bordeaux, Lille.

Janvier 2019 constitue une date clé pour l'entreprise puisqu'elle marque l'alliance de Jaguar

Network avec le groupe Iliad, acteur majeur des télécommunications en Europe. Le groupe, fondé par Xavier Niel, regroupe différentes filiales comme Free ou Scaleway. L'objectif de cette alliance étant de rattraper le retard d'Iliad dans les services télécoms pour les entreprises.

Cette opération permet à Xavier Niel d'hériter de nouveaux clients tels qu'Amazon, EDF, France Télévisions ou Amadeus.

Fait notable, Jaguar Network est propriétaire de Datacenters, tous situés en France. Le premier se situe à proximité du siège social de l'entreprise, à Marseille.

Inaugurée en 2013, cette infrastructure de 8000m<sup>2</sup> s'inscrit parmi les plus grands d'Europe. En plus de Marseille, deux Datacenters sont situés à Lyon.

Au-delà de ces trois structures, l'entreprise loue des espaces de stockage dans cinq Datacenters à Paris, deux à Marseille, un dans la région lyonnaise et un à Bordeaux. Ces bâtiments sont développés et gérés en interne par les services de l'entreprise.

Les forces de Jaguar Network résident en la proximité, l'écoute et l'adaptabilité et ses valeurs sont les suivantes :

- Client : accompagnement du client dans le temps, anticipation des besoins
- Expertise : Qualité des conseils, ingénierie, bonne connaissance des produits du marché et des secteurs d'activités des clients, recherche des meilleures solutions à leurs problèmes.
- Disponibilité : Un travail de qualité dans de courts délais.

L'entreprise définit son activité selon plusieurs domaines d'activités stratégiques précis :

L'hébergement Cloud : l'entreprise fournit des services d'hébergement publics et privés. Ces offres sont faites sur mesure pour permettre leur adaptation à tout type d'infrastructure. La sécurité est assurée et la protection des données n'est jamais négligée.

**Les Télécommunications :** Jaguar Network est également un fournisseur d'accès à Internet, de VoIP et de réseaux privés. La société est maître d'un réseau de fibre optique redondant sur le territoire européen. Les offres sont diverses : services d'accès à Internet, transit Internet en Datacenter, solutions VPN et Voix sur IP.

**Les services managés :** Jaguar Network dispose d'équipes d'infogérance qui permettent la supervision et le bon fonctionnement de ses infrastructures matérielles au sein des Datacenters.

**Les infrastructures logicielles (applications, API)** sont assurées par les équipes du Système d'Information (SI).

Jaguar Network adresse la globalité du marché BtoB (PME/PMI, grands comptes et marchés publics) avec des offres orientées usages, besoins clients et une distribution nécessairement multicanale (Vente directe/Vente indirecte).

L'ambition de Jaguar Network est de garantir des services à valeur ajoutée, les aidant à se concentrer pleinement sur leur activité.

Ces services peuvent se distinguer en deux grandes familles :

- Hébergement
  - Gamme s'étendant de la colocation à la gestion de plateformes critiques
- Réseau et télécommunication
  - Offres de solutions Voix (VoIP)
  - Offres d'interconnexion de sites (réseaux VPN par exemple)

L'entreprise agit sur plusieurs secteurs très prisés, elle est donc naturellement confrontée à de nombreux concurrents. On peut notamment noter Orange, SFR, OVH ou Cheops. Par conséquent et en raison de son action dans plusieurs

activités, Jaguar Network ne fait pas partie des leaders mais arrive tout de même à se démarquer et de disposer de nombreux clients dans ce marché.

En outre, le marché est très vaste et très sollicité.

De nombreuses entreprises françaises ont recours au Cloud et font appel à des services privés d'hébergement.

Par secteur d'activité, les entreprises clientes les plus présentes sont celles dans dont le secteur est le service aux entreprises, le service informatique et l'édition de logiciels. Elle compte plus de 1000 clients dont 70 % de PME.

D'autres plus gros clients souscrivent à nos offres :Amazon, France Télévisions, EDF, ou encore Amadeus.

80 % de ces clients sont centralisés en région Provence-Alpes-Côtes-d'Azur, Ile-De-France et Rhône-Alpes, région où est installée Jaguar Network.

La compagnie commercialise des offres sur mesure pour un portefeuille de clients variés :

- Acteurs du commerce en ligne : Cdiscount, Rue du commerce, Winamax
- Entreprises de services du numérique : Capgemini, PrestaShop
- Médias : France télévisions, RMC Info, Dailymotion
- Sociétés du domaine du transport et de la logistique
- Banques et assurances : Groupama, Generali
- Marchés publics : ministère de l'Intérieur, Sénat
- Ou encore des sociétés de l'industrie, de l'hôtellerie, de l'édition de logiciels, de luxe et de santé.

Jaguar Network est gage de qualité et a un réel souci d'excellence à la fois dans les conseils techniques et par l'exemplarité de son comportement.

## B. Mon rôle dans l'entreprise

Dans le cadre de mon travail chez Jaguar-Network en tant que développeur système d'information en alternance sur les projets RFC (ressource humaine,

finance et commerce), j'ai participé au développement et à la maintenance du système d'information en développant les outils de Jaguar-Network intranet/extranet. Mes missions étaient variées, allant de la conception, la mise à jour, la maintenance logicielle et des bases de données à la diffusion des données à travers des API et leur utilisation dans des applications web. L'équipe RFC regroupe des tâches allant de la gestion de nouvelle méthode de facturation, au développement de nouveaux processus d'astreinte et maintenance de la génération de bons de commande. Cette équipe s'est réellement formée peu après mon arrivé, au début composée de deux développeurs et d'un lead développeur, nous sommes aujourd'hui quatres développeurs. Nous n'avions pas de Product Owner attitré jusqu'à récemment, celui-ci a permis une meilleur organisation de travail notamment avec la mise en place de sprint issue de la méthode Agile.

## C. Organisation de travail

Pour le travail en équipe, nous fonctionnons en méthode Agile avec des sprints de deux semaines constitué d'un poker planning à son lancement et d'une rétrospective à la fin.

Le poker planning sert à estimer la difficulté des tâches sans que les membres s'influencent, puisque tout le monde doit révéler sa carte en même temps. Les personnes ayant estimé le plus bas et le plus haut prennent la parole pour justifier leur choix, puis après un court débat nous sélectionnons souvent la moyenne.

La rétrospective permet de comprendre ce qui s'est bien ou mal passé durant un sprint et de trouver des solutions pour améliorer l'efficacité de l'équipe. Ensuite nous votons pour les solutions, celles majoritaires sont sélectionnées et mises en place.

Les tâches sont organisées sous forme de ticket Jira et assignées en fonction des appétences de chacun. Si l'un des membres rencontre des difficultés, on organise des sessions de peer-programming afin de mieux appréhender le problème. Nous utilisons Gitlab et git pour le versionning, avec des branch de développement par équipe, dans notre cas “rfc-develop”, “rfc-release” pour les démonstrations client et “master” qui correspond à ce que nous avons en production.

Notre procédure pour la gestion de potentiel conflit est simple :

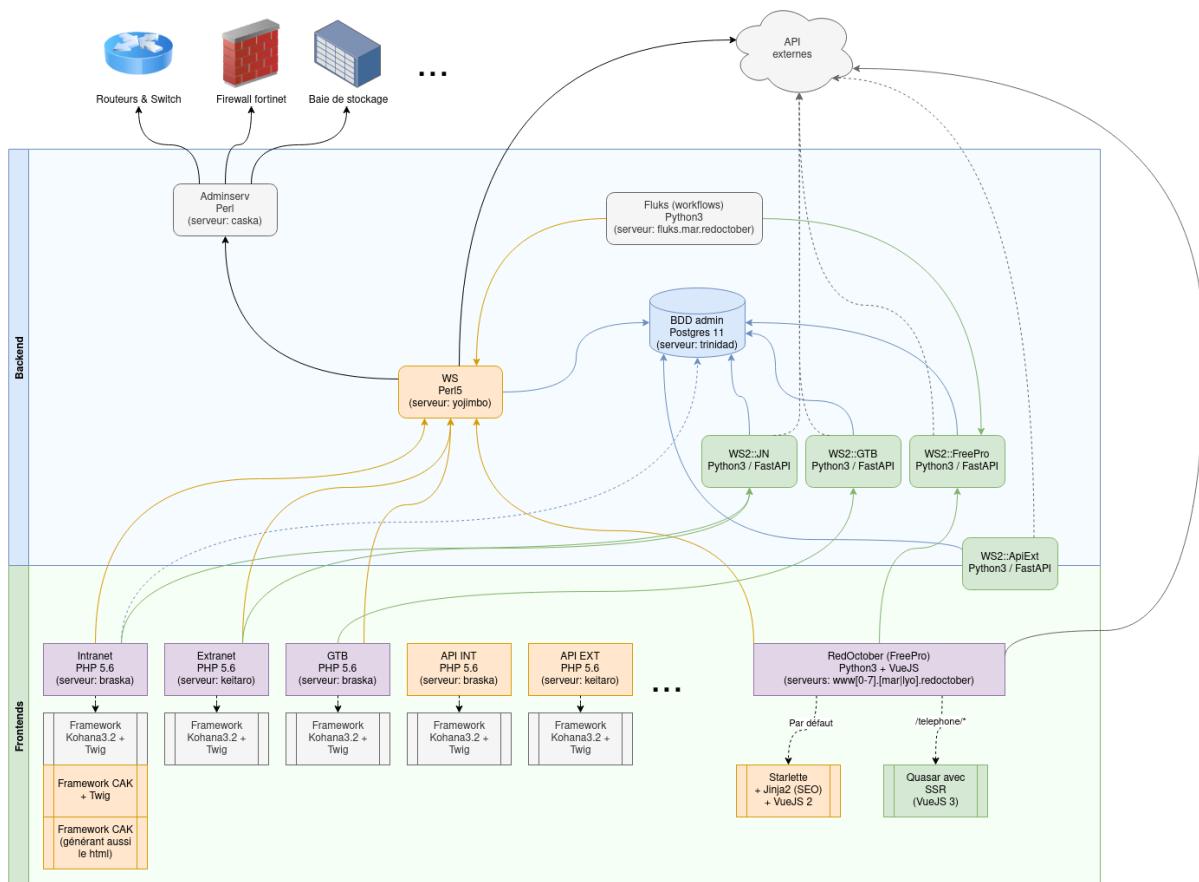
- Toujours s'assurer d'être à jour de la branch de développement avant de créer une branch qui correspond au ticket Jira, e.g.  
RFC-148\_manual\_invoice\_process ;
- Faire des “rebases” pour squash ses commits ;
- Si une branch a été mergé avec la branch de développement entre temps (rfc-develop dans l'exemple) , rebase sa branch (RFC\_148 dans notre exemple) avec rfc-develop afin d'avoir les dernières modifications ;
- Si la gestion de conflit avec le rebase est trop compliqué, créé une nouvelle branch à partir de rfc-develop à jour, puis cherry-pick les commits déjà effectués localement ;

Ainsi, nous limitons les conflits lors de la mise en production d'un sprint.

## VIII. Spécificités techniques

### A. Technologie et modèle de données

Le système d'information de Jaguar-Network est constitué de plusieurs couches de technologie qui s'articulent entre elles, avec deux pôles principaux : le rouge qui correspond aux projets liés aux offres FreePro et le bleu “historique” sur lequel j'ai principalement travaillé. Ci-dessous un schéma suivi d'une description de l'architecture du SI côté bleu :



- BDD admin PostgreSQL, une base de données relationnelle orientée objet open-source ayant une capacité de résoudre des problèmes complexes.
- relationnelle orientée objet open-source ayant une capacité de résoudre des problèmes complexes. Pièce maîtresse du système d'information, elle regroupe l'essentiel de la logique métier et dispose de “sanity check” intégré aux fonctions ;
- Webservice en Perl, qui fait le lien entre la base de données et les

differentes interfaces front-end ;

- Intranet, Extranet, GTB en PHP avec le framework Kohana et le moteur de template Twig ;

Le développement d'un nouveau webservice (WS2) est en cours en Python avec le framework FastAPI, notamment pour la partie FreePro, avec comme framework front-end Vue.js et Python3.

## IX. Réalisation et extrait de code

### A. Description et analyse de la tâche

Durant cette année d'alternance, j'ai travaillé sur différents tickets en lien avec la facturation, en voici un que je vais détailler ici :

#### Problème :

Lors de la saisie manuelle des encaissements des factures impayés clients, ceux-ci s'exportent de la BDD au logiciel Sage X3 à la date de l'opération (date de valeur sur le compte bancaire). Ceci pose des soucis de cut-off pour les closing. Le cut-off regroupe l'ensemble des opérations visant à clôturer les comptes d'une entreprise et ayant pour objectif de respecter le principe comptable d'image fidèle.

#### Analyse :

Il s'agit ici d'étendre les données renseignée dans la base de données qui seront ensuite exporté dans le logiciel comptable Sage X3, en effet, Jaguar-Network a migré du logiciel Quadra, adapté au petite PME , à Sage peut avant mon arrivé et de nombreuse problématique sont encore en cours de résolution, de surcroît avec l'arrivé de FreePro. La facturation est un sujet complexe, toutes les écritures

comptables sont d'abord enregistrées en base de données avant d'être exportées dans Sage X3, les opérations ainsi que les pièces comptables sont résumées dans des tables séparés, les paiements sont centralisés dans groupe Iliad via une API externe et il y a également un système de gestion des comptes de facturations, de génération de compte comptable et d'autorisation de paiement. En sommes, c'est quasiment un véritable logiciel comptable qui est maintenu au sein du système d'information.

Dans un premier temps, vient l'analyse de la partie base de données, deux tables ici sont concernées :

- “payment\_operations” : résume l'opération comptable avec ses informations les plus importantes tel que le prix, les foreigns keys du compte de facturation client et de la méthode de paiement, du code de paiement de l'API externe d'Iliad ainsi que la date de création, échéance et rejet.
- “cpt\_ecritures” : enregistre les écritures comptables proprement dites tel qu'on peut les retrouver dans les logiciels comptable avec le montant en débit, crédit, TVA, le label, le compte, le journal, la date d'écriture et d'échéance.

Il est proposé de créer un nouvelle colonne “date\_sent” pour la table “payment\_operations” comme date d'envoie du paiement à l'API d'Iliad, d'utiliser la colonne “date\_ecriture” et “date\_echeance” de “cpt\_ecritures” respectivement comme la date comptable et la date d'encaissement.

- Date d'encaissement : date à laquelle le virement a été crédité sur le relevé du compte bancaire
- Date comptable : date de l'écriture du Virement que l'on indique pour Sage X3. C'est à cette date que l'écriture comptable devra s'importer dans Sage X3.

L'objectif de ces 2 dates est de pouvoir permettre au service facturation d'être en phase avec les dates de règlements émis par le client et pour la comptabilité de figer les données remontées au groupe pour la clôture des comptes. Ces nouveaux champs ont également un impact sur plusieurs **fonctions PL/pgSQL** tel que "insert\_payment\_manual()", "insert\_payment\_transfer()", "insert\_payment\_check()" et "insert\_payment\_batch\_ecritures()" qui correspondent aux différentes méthodes de paiement autorisé chez Jaguar-Network.

Dans un second temps, nous modifierons le Webservice qui fait appel aux fonctions SQL ainsi que la partie d'export dans Sage X3 toute deux en Perl. En effet, il nous faut modifier le modèle d'export en y ajoutant un champ ainsi que les arguments de la fonction d'appel à la base de données : "sub record\_payment()".

Enfin, nous modifierons l'interface front-end de l'intranet pour que ces champs soient différenciés visuellement lors de la saisie manuel de l'encaissement des factures impayés tout en ajoutant un nouvel argument lors de l'appel à la fonction du webservice.

## B. SQL-ADMIN

Tout d'abord nous allons rajouter des commentaires associés aux colonnes de date dans les tables concernés afin de bien définir leur utilisation pour les prochains développeurs qui travaillent avec :

```
COMMENT ON COLUMN cpt_ecritures.date_ecriture IS 'is accounting date, i.e. date_posting';
COMMENT ON COLUMN cpt_ecritures.date_echeance IS 'is collecting date, i.e. date_bank';
COMMENT ON COLUMN payment_operations.date_sent IS 'SEPA batches: Date when operation is sent to Iliad78
; Otherwise it''s collecting date, i.e. date_bank';
```

Puis nous modifions les différents arguments des fonctions pour que cela corresponde au nouveau fonctionnement, ici nous prendrons comme exemple la fonction “insert\_payment\_manual” qui est appelé par les fonction “insert\_payment\_transfer()”, “insert\_payment\_check()” et “insert\_payment\_batch\_ecritures()” avec des arguments qui correspondent à la méthode de paiement :

```
CREATE OR REPLACE FUNCTION insert_payment_transfer(vpieces jsonb, vid_jn_bank_statement_line integer)
RETURNS integer AS $$  
BEGIN  
    RETURN insert_payment_manual('transfer', vpieces, vid_jn_bank_statement_line);  
END;  
$$ LANGUAGE 'plpgsql';
```

```
CREATE OR REPLACE FUNCTION insert_payment_manual(vtype text, vid_jn_bank integer, vpieces jsonb,
vdate_posting date, vdate_bank date) RETURNS integer AS $$
```

Celle-ci a deux rôles, créer un opération de paiement dans “payment\_operation” avec une date d’envoie “date\_sent” qui correspond à la date de valeur bancaire et insérer les écritures dans la table “cpt\_ecriture” les nouvelles dates décrites précédemment :

```
INSERT INTO payment_operations(id_batch, id_billing_account_bank, id_payment_method, price, label,
status, date_echeance, date_sent)
VALUES (
    vid_batch, vbank.id_billing_account, vid_payment_method, vpiece.price,
    array_to_string(array['FACTURE', vbank.jn_ent_ref, vpiece.date_payment::text,
    vpiece.jnref], ' '),
    'ok', vpiece.date_payment, vdate_bank
)
RETURNING id INTO vid_payment_op;
```

```
IF vcredit + vddebit > 0 THEN
    -- generer dans cpt_ecritures (credit) compte client, journal de banque
    -- les écritures de prélevement correspondant à chaque ligne dans payment_operations
    PERFORM insert_cpt_ecriture(
        vpiece.id_piece,
        vid_payment_op,
        vpiece.id_billing_account,
        vbank.journal,
        vpiece.compta_ref,
        vdate_posting,
        vdate_bank,
        vprefixes->>vtype || ' ' || vpiece.compta_ref || ' ' || vpiece.jnref,
        CASE WHEN vpiece.price < 0 THEN 0 - vpiece.price ELSE 0 END,
        CASE WHEN vpiece.price < 0 THEN 0 ELSE vpiece.price END,
        0,
        UPPER(vtype)
    );
END IF;
```

Une fois ces insertions en base de données faites, nous passons maintenant à la modification du webservice.

## C. Webservice

D'abord on modifie la fonction qui fait appel à la Base de données avec un nouvel argument transmis depuis l'interface intranet :

```

sub record_payment
{
    my ($self, $pieces, $date_bank, $mode, $id_bank, $date_posting) = @_;
    my ($session, $info, $admin_dbh) = check_session('edit_factu');

    if (!$pieces || (ref $pieces ne 'ARRAY') || !$date_bank || !$mode) {
        die '[record_payment] Parameter errors ; need pieces [ { jnref, price } ], dates and mode';
    }

    my $sth;

    # standard payment modes ; functions sharing the same signature
    my %std_modes = ('vir' => 'insert_payment_transfer', 'chq' => 'insert_payment_check');

    $mode //='';
    $mode = lc($mode);

    if (exists($std_modes{$mode})) {
        if (!$date_posting) {
            die sprintf('[record_payment] Parameter errors ; need date_posting for mode "%s"', $mode);
        }
        $sth = $admin_dbh->sql_query(
            sprintf("SELECT %s(?, ?, ?, ?)", $std_modes{$mode}),
            $id_bank,
            encode_json($pieces),
            $date_posting,
            $date_bank
        ) or die $admin_dbh->{'last_error_msg'};
    }
    elsif ($mode eq 'credit') {
        $sth = $admin_dbh->sql_query(
            "SELECT insert_payment_credit(?, ?)",
            $date_bank,
            encode_json($pieces)
        ) or die $admin_dbh->{'last_error_msg'};
    }
    else {
        die sprintf('[record_payment] Mode %s not implemented', $mode);
    }

    my ($id_batch) = $sth->fetchrow_array();

    return $id_batch;
}

```

Puis on modifie les fonctions qui charges les lignes depuis la base de données pour les exporter dans Sage x3 :

```

sub load_lines {
    my ($self) = @_;
    my $query;

    if (!$self->id_batch && !@{$self->id_operations}) {
        die 'Please load a valid batch or a list of operations before loading lines';
    }

    $query = <<SQL;
    SELECT po.label AS payment_label,
           po.status, po.date_creation, po.date_echeance, po.date_reject,
           ce.id AS id_cpt_ecritures,
           ce.label AS cpt_label,
           collectif.compte AS col_compte,
           collectif.code AS col_code,
           ce.account, ce.journal, ce.debit, ce.credit, ba.compta_ref,
           to_char(ce.date_ecriture, 'YYYYMMDD') AS date_posting,
           to_char(ce.date_echeance, 'YYYYMMDD') as date_bank,
           p.jnref as piece_ref
      FROM payment_operations po
     -- rejects also generate cpt_ecritures ; don't include them in generated piece
   JOIN cpt_ecritures ce ON ce.id_payment_operation = po.id AND NOT
cpt_ecritures_is_reject(ce.id)
     JOIN pieces p ON ce.id_piece = p.id
     JOIN billing_accounts ba ON p.id_billing_account = ba.id
     JOIN LATERAL (
       SELECT * FROM sage_get_ba_collectif(ba.id)
     ) AS collectif ON true
SQL

    if ($self->id_batch) {
        $query .= ' WHERE po.id_batch = ?';
    }
    else {
        $query .= sprintf(' WHERE po.id IN (%s)', join(',', map { '?' } @{$self->id_operations}));
    }
    $query .= <<SQL;
        AND po.status != 'a envoyer' AND ce.x3_ref IS NULL
        ORDER BY ce.account, p.jnref
        LIMIT ?
SQL

    my $sth = $self->_db()->sql_query($query, $self->id_batch || @{$self->id_operations},
$self->max_op_count) or die $DBI::errstr;

    while (my $rec = $sth->fetchrow_hashref()) {
        push @{$self->lines}, $rec;
    }
    $sth->finish();
}

```

```

sub generate_sage_payload
{
    my ($self) = @_;

    if (!$self->id_batch && !@$self->lines) {
        die 'Lines are empty ; please load a valid batch or list of operations before generating sage payload';
    }

    my $payload = '';

    my $site = $global_conf->{'sage'}->{'jn_sites'}->{$self->ent_ref} || '';
    # lines
    my $line_number = 1;

    # header according to ZWSGAS import model
    # G;ENCAI;CMJN132102000003;JN13;CIC;20210211;20211106;F202001000001;20211124;EUR;STD0;1
    # Section; Type de pièce; Numéro de pièce; JN/Freepro; Journal de banque; Date comptable; Id batch;
    # Référence document d'origine; Date d'encaissement; Devise; Transaction; Catégorie pièce comptable;
    $payload .= sprintf(
        'G;%s;%s;%s;%s;%s;EUR;STD0;1',
        $self->sage_type,
        $site,
        $self->lines->[0]['journal'],
        $self->lines->[0]['date_posting'],
        $self->doc_ref,
        $self->lines->[0]['date_bank']
    );

    # contrepartie
    # D;1;1;1;JN13;;51220000;;;F2020xxx;1;1792;EUR
    # D;1;4;1;JN13;;51220000;;;F2020xxx;1;1792;EUR
    foreach my $ref (qw(1 4)) {
        $payload .= $self->separator . sprintf(
            'D;%d;%d;1;%s;;%s;;%s;1;%f;EUR',
            $line_number,
            $ref,
            $site,
            $self->infos->{'x3_account'},
            $self->infos->{'label'} || '',
            $self->infos->{'batch_sum'},
        );
    }

    # load lines with accounting piece as lettering ref in field "Référence libre"
    foreach my $l (@{ $self->lines }) {
        $line_number++;

        # D;2;1;2;JN13;C0;41100000;013TRIL3;TestBatchSI013TRIL3;F2020xxxxxxxx OR A2020xxxxxxxx;-1;1792;EUR
        foreach my $ref (qw(1 4)) {
            my ($dir, $amount);
            if (substr($l->{'piece_ref'}, 0, 1) eq 'A') {
                # avoir => debit
                $dir    = 1;
                $amount = $l->{'debit'};
            }
            else {
                # default : facture => credit
                $dir    = -1;
                $amount = $l->{'credit'};
            }
            $payload .= $self->separator . sprintf(
                'D;%d;%d;2;%s;%s;%s;%s;%s;%d;%f;EUR',
                $line_number,
                $ref,
                $site,
                # should be $l->{'col_code'}, but : <ENCAI > SAC R\x{e9}f 4 C0 (Ligne 2) : \\GRP
                C0 Collectif inexistant (99)
                '',
                $l->{'col_compte'},
                $l->{'compta_ref'},
                $l->{'cpt_label'} || '',
                $l->{'piece_ref'},
                # debit / credit
                $dir,
                $amount,
            );
        }
    }

    # end of payload
    $payload .= $self->separator;
}

return $payload;
}

```

Enfin, nous pouvons passer à la partie front-end avec l'intranet.

## D. Intranet

En premier lieu on modifie la fonction qui parcours les réponses du formulaire avant de les envoyer au webservice :

```

public function doRecordPayment(&$form)
{
    $total = 0;

    if (!$this->acl['can_edit_factu']) {
        $form->setErrorMessage('Forbidden');

        return;
    }
    $pieces = [];

    // gather pieces refs and amount posted for each piece
    foreach ($form->getPostedValue('piece_refs') as $ref) {
        $price = $form->getPostedValue('amount_'.$ref);
        if (null === $price) {
            $form->setErrorMessage(sprintf('Could not find price for piece %s', $ref));
            return;
        }
        $total += $price;

        $pieces[] = [
            'jnref' => $ref,
            'price' => $price,
        ];
    }
    if (!count($pieces)) {
        $form->setErrorMessage('Need at least one piece');

        return;
    }

    // default value if not posted, needed for credit
    $iso_format = [ 'date_bank' => date('d/m/Y') ];

    // "normal" payment
    if ($total > 0) {
        foreach ([ 'mode', 'jn_bank', 'date_bank', 'date_posting' ] as $field) {
            if (!$form->getPostedValue($field)) {
                $form->setErrorMessage(sprintf('Missing %s', $field));
                return;
            }
        }

        // convert date from DD/MM/YYYY to ISO : YYYY-MM-DD
        // posting and collection date convert in string
        foreach ([ 'date_bank', 'date_posting' ] as $key => $field) {
            $dates[$key] = DateTime::createFromFormat('d/m/Y', $form->getPostedValue($field));
            $iso_format[$field] = $dates[$key]->format('Y-m-d');
        }

        if ($iso_format['date_bank'] > $iso_format['date_posting']) {
            $form->setErrorMessage('The posting date must be less than or equal to the bank date');
            return;
        }

        $mode = $form->getPostedValue('mode');
        $bank = $form->getPostedValue('jn_bank');
    }
    // sum = 0 when credit = debit ; just record A/F pieces altogether
    else {
        $mode = 'credit';
        $bank = null;
    }

    // all good, send the cash !
    try {
        // this is a real payment (VIR / CHQ)
        if ($bank) {
            $this->sponge->record_payment(
                $pieces,
                $iso_format['date_bank'],
                $mode,
                $bank,
                $iso_format['date_posting']
            );
        }
        // this is just a list of F/A with a total amount of 0 ; no bank involved
        else {
            $this->sponge->record_payment(
                $pieces,
                $iso_format['date_bank'],
                $mode
            );
        }
    } catch (Exception $e) {
        $form->setErrorMessage($e->getMessage());
        return;
    }
    $this->request->redirect('/app/si/payments/outstanding/' . $this->request->param('ent_ref'));
}
}

```

Et dans un second temps nous modifions le formulaire généré en Twig pour ajouter un champ “Date bancaire” qui correspond à la date à laquelle le montant est passé en banque :

```
● ● ●
{%
    if acls.can_edit_factu %}
    <div class="form-group toolbar">
        <label><span id="payment_label">Encaisser</span> <span id="show_count">0</span> pièces(s)</label> -
        <span id="show_amount" class="badge badge-primary">0</span><b> €</b>
        <span id="form_inputs">
            {{ form.addElement('select', 'mode').setOptions({'VIR': 'Virement', 'CHQ': 'Chèque'}).addEmptyOption('Choisir le mode d\'encaissement') }}
            {{ form.addElement('select', 'jn_bank').setArrayOptions(jn_banks, 'id', 'name').addEmptyOption('Choisir la banque') }}
            en date de valeur du {{form.addElement('calendar', 'date_bank').setClass('form-control jnform_calendar').setMaxDate('yesterday|date("Y-m-d"))}}}
            en date comptable du {{form.addElement('calendar', 'date_posting').setClass('form-control jnform_calendar').setMaxDate('yesterday|date("Y-m-d"))}}}
        </span>
        <br/>{{ form.addElement('submit', 'encaisser', 'Encaisser').setClass('btn btn-primary pull-right') }}
    </div>
{%
    endif %}
```

## E. Test du code

Pour la réalisation de test j'ai opté pour une approche fonctionnelle, avec différent scénario et profil utilisateur afin d'identifier les potentiel bug avant la mise en production dans un cahier de test sous format tableur. Par exemple, si je vérifie que la date valeur ne peut pas être supérieur à la date de saisi puisqu'il s'agit d'opérations bancaire passés déjà présent sur le relevé bancaire, je vais me mettre en profil “DAF/CDG” avec les acl correspondantes, spécifié les données entrées et données attendus dans mon fichier, puis mener le test pour indiquer son résultat. Un exemple de cahier de test est en annexe.

## F. Conclusion du projet facturation : Jaguar-Network

Durant cette réalisation, j'ai pu découvrir plusieurs éléments :

- Les base de données relationnel orienté objet et le pl/sql ;
- L'architecture applicative web ;

- Travailler avec des API ;

Cette tâche plutôt facile m'a permis d'aborder le système de facturation aussi bien côté back-end que front-end, ce qui m'a conduit aujourd'hui à travailler sur un projet bien plus conséquent que j'aurais voulu présenter mais qui était encore sous couvert de confidentialité.

Ce projet m'a apporté les compétences du référentiel ci-dessous :

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :

- Développer des composants d'accès aux données
- Développer la partie back-end d'une interface utilisateur web

2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :

- Développer des composants dans le langage d'une base de données

3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :

- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Développer des composants métier
- Préparer et exécuter les plans de tests d'une application

## G. Recherche en anglais

Lors d'une tâche de génération de facture via le langage LaTeX et template toolkit, j'ai dû mener une recherche en anglais sur le forum tex.stackexchange.com où j'ai moi même posé une question sur la possibilité de répéter un "header" sur un tableau qui se génère sur plusieurs pages :

## Add header when the table is generating a new page (PDFtolatex for invoice)

Asked 7 months ago Modified 7 months ago Viewed 40 times



I will try to explain my problem as precise as I can, feel free to ask me more details:

- 0
- I'm currently working on a invoice generator with LateX and template toolkit, I know it's not optimal but it's legacy code and we can't create a new one ;
  - I want to add a banner on top of a particular part of the pdf document which are table, when the table is long enough to create a new page, so I need this banner on top of the table second page ;
  - I got multiple template which depend of the services/type of goods, so the debug is quite complicated because I never know in which file or line the errors really occurs (yes I used -file-line-error debug but it show the compiling temporary files in var/tmp and not the files I'm currently working on) ;

**Question :** Is this even possible ? For me LaTex was only for designing scientific documents, not for invoice, and the compiling process with file included seems working badly.

**What I tried :** To make a table that incorporate the different template part, with a table header that looks like a banner. Problem is the different template part are also table and I got multiple error I didn't manage to solve due to the architecture of this invoice generator because debug point things in temporary compiling pdf and not in the working files I'm on. Errors like :

1. Underfull \hbox (badness 10000) in paragraph at lines xxx--xxx ;
2. ./tempi6163.tex:723: Too many }'s.

Any hints, work around and advice are more than welcome, I litteraly know noone who have ever use LaTe $\backslash$ X except for thesis document but in a very basic way.

Heres the code, warning it may scares you :

### The Overflow Blog

- ✓ Skilling up to architect: What land high-paying IT roles
- ✓ Exploring the interesting results from our 2022 Dev

### Featured on Meta

- 💡 Testing new traffic mana
- 💡 Duplicated votes are bei
- { } Acceptable Questions ar

### Linked

- 89 What does the phrase (badness 10000) in pa mean?

### Related

- 8 Add Table in Header w
- 4 Add header to the table
- 0 Table for Invoice
- 1 How to add the caption table, which transferred

J'ai eu la chance de pouvoir ensuite échanger avec un des membres de l'équipe

LaTeX3 : David Carlisle :

I have no idea what templating language you are using to generate the latex source, but that probably doesn't matter. In your `longtable` code if you replace ``\[-0.2cm]\HRule\` by `\[-0.2cm]\HRule\endhead\` then that part will be repeated every page, which seems to be your main question. – David Carlisle Nov 8, 2021 at 11:29`

for the underfull hbox warning see [tex.stackexchange.com/questions/334246/...](https://tex.stackexchange.com/questions/334246/) – David Carlisle Nov 8, 2021 at 11:35

You haven't said what programming language `IF NOT document_config.document_watermark ;` is (and I don't recognise it) but you would find it easier to get help here if you removed all this template code from the question and just show one generated latex file. Once you have the generated latex edited to do what you want then you should be able to go back and modify your templating script to generate that target document. – David Carlisle Nov 8, 2021 at 12:27

Hi David, thank's for your reply. Actually I said it, the programming language you mention is template toolkit (quite unknown), and I added `\endhead` at the end of my try : "And here's a sample of a template `(${grid.grid_line_type_ref}.tt)` I want to integrate inside which are also table :" but it didn't work as expected because my templates doesn't have a fixed column number. I will try to reconstruct a one LaTeX file but I don't think it's relevant because there is almost 25 different template with different construction... Anyway I'll try ! – Sarp Nov 8, 2021 at 13:19

Celui-ci m'a conseillé de déconstruire mes templates afin de voir d'où venait le problème et de modifier le code dans ma partie “longtable”. Après avoir essayé ces solutions, j'ai pu résoudre mon problème en trouvant une erreur de syntaxe dans un des templates qui était inclus dans le mien, ce qui générerait une réaction en chaîne et empêchait la détection de la nouvelle page.

## Annexe

### A. Cahier de test

N°	Description	User / Profil	Chemin	Données en entrée	Attendu	Etat Test	Résultat	Commentaire
1	L'entité doit avoir des factures impayées	N/A	/app/sip/payments/outstanding/XENTITLEXX	N/A	Btre dans l'état paiement provisoire	OK		
2	Lors de la saisie des encaissements (virement ou chèque) pour les impayées	SAP/ADV	//	Facture impayée Montant : XXX Date : XXX Valeur uniquement < Date du jour car ajout bloquage sur la sélection de la date Date unique = date comptable, date échéance => date valeur et je veux que la date valeur et la date comptable apparaissent respectivement dans les input "Date document" et "Date"	Avoir la date valeur à la place de date échéance Avoir la date valeur à la place de date comptable Avoir la date valeur dans la table pt, écritures - Avoir la date valeur dans la table pt, écritures -> date valeur - Avoir la date valeur qui apparaît dans l'input "Date document" et "Date" - Avoir la date comptable qui apparaît dans l'input "Date" dans SAGE lors de la saisie de pièces	OK	Les valeurs sont dans les colonnes attendus dans la base de données et pas dans l'input attendu dans SAGE.	Virement : SAGE ref virement : CMN132111000016 ; - Chèque : CMN132111000017 ; - SAGE ref chèque : CMN132111000017 ;
3		SAP/JUV	//	valeur < Date Comptable	Date	OK	//	Virement : SAGE ref virement : CMN132111000018 ; - Chèque : CMN132111000019 ; - SAGE ref chèque : CMN132111000019 ;
4	je veux vérifier que la date valeur ne peut pas être supérieure à la date comptable pour que ce soit possible puisque il s'agit d'opérations bancaires passées déjà présent sur le relève bancaire	SAP/CDG	//	Date valeur > Date comptable	Message d'erreur et impossibilité de saisir une date valeur > à la date comptable	OK	l'insertion est bloqué	
5	je veux vérifier quel profils ont accès à l'action sur la page des factures impayées	SAP/COMPTABLE	//	Date valeur = Date comptable	Impossibilité d'avoir l'action	Bloqué	Action est bloquée	
6	je veux vérifier quel profils ont accès à la page de impayées	SAP/PAGE	//	N/A	Impossibilité d'accès à la page	Bloqué	La page n'est pas accessible	

