RAPPORT DE STAGE

Etudiant : ETCHEPARE Cédric

Tuteur : FUSADE Benoît

Enseignant référent : BRUYERE Marie

Enseignant lecteur : CHBEIR Richard

Sommaire

1. Contextualisation
   1.1 Introduction
   1.2 Présentation de l'entreprise
   1.3 Présentation de YellowBox

2. Environnement
   2.1 Gestion de projet
   2.2 Outils
      2.2.1 Développement
      2.2.2 API

3. Mission
   3.1 Présentation de l'outil utilisé
   3.2 Intérêt de l'entreprise pour Zapier
   3.3 Contenu de la mission

4. Résultats
   4.1 Architecture technique
   4.2 Choix de développement
   4.3 Présentation du produit final

5. Bilan du stage
   5.1 Bilan des compétences
   5.2 Bilan humain
   5.3 Bilan organisationnel
   5.4 Perspectives

6. Bibliographie
7. Glossaire
8. Annexe
9. Résumé en Anglais

Tables des figures

1. Contextualisation
   1.1 Introduction
   1.2 Présentation de l'entreprise (général)
   1.3 Présentation de YellowBox

10. Mission
   2.1 Objectifs
   2.2 Public ciblé
   2.3 Cahier des charges fonctionnel

11. Environnement
   3.1 Méthodes de travail
   3.2 Choix et utilisation d'outils
      3.2.1 Technologies
      3.2.2 Gestion de projet
   3.3 Problèmes rencontrés
   3.4 Solutions proposées

12. Résultats
   4.1 Architecture technique
   4.2 Présentation du produit final

13. Bilan du stage
   5.1 Compétences confortés
   5.2 Compétences acquises
   5.3 Bilan technique, organisationnel, humain
   5.4 Évolutions prévues / Perspectives

14. Bibliographie
15. Glossaire
16. Annexe
17. Résumé en Anglais

Ce rapport contient divers acronymes et termes techniques qui peuvent nécessiter une clarification. Un glossaire a été créé afin de fournir des définitions pour chaque mot indexé dans le document. Lorsque vous rencontrez un terme en italique ou un acronyme pour la première fois dans le rapport, veuillez vous référer à cette section pour trouver sa définition complète.

Remerciements

Je tiens à exprimer ma gratitude à ma référente de stage Madame Bruyère qui a pris le temps de venir me voir dans les locaux de mon entreprise afin de voir si tout se passait bien.

Je tiens également à remercier l'équipe de DIMO Software à Bidart pour son accueil chaleureux et sa collaboration. Chaque membre m'a aidé en partageant ses connaissances et en m'impliquant sur les réunions et les moments collectifs.

Je souhaite également exprimer ma reconnaissance envers mon tuteur de stage Benoît Fusade et celui qui m'a accompagné tout au long de ma mission Timothée Picard. J'ai pu acquérir et conforté plusieurs compétences grâce aux conseils et à l'attention que vous m'avez portés.

Je reste à votre disposition pour toute discussion supplémentaire concernant mon rapport de stage et je me réjouis de pouvoir vous présenter plus en détail mes observations et mes réflexions.

Cordialement,

ETCHEPARE Cédric

1. Contextualisation

1.1 Introduction

Je m'appelle Cédric ETCHEPARE, j'ai 19 ans et je suis étudiant en 2ème année de BUT Informatique à l'IUT de Bayonne et du Pays Basque. J'ai intégré l'IUT à la suite d'un baccalauréat général avec les spécialités Mathématiques et Numérique et Science de l'informatique, c'est à ce moment-là que j'ai pu voir un peu les métiers que l'on pouvait trouver dans l'informatique que ce soit dans le développement, le réseau, les données…

C'est après cela que j'ai souhaité me diriger vers le BUT qui me paraissait être une formation abordant plusieurs aspects des métiers de l'informatique. Après une première année qui s'est bien déroulé, c'est à la fin de celle-ci que j'ai décidé de me spécialiser dans un domaine particulier avec le parcours Parcours A « Réalisation d'applications : conception, développement, validation » qui me paraissait être le plus adapté pour moi avec un noyau axé sur le développement.

Grâce à celui-ci, j'ai pu approfondir les notions acquises en première année, mais aussi en découvrir d'autres notamment avec le développement d'applications mobiles, le développement web et l'utilisation de framework.

Cela m'a conforté dans l'idée que j'avais choisi un parcours qui me correspond, je suis donc tout naturellement partie dans la recherche d'un stage sur le développement.

C'est donc début mars que j'ai eu une réponse positive de la part de DIMO Software qui proposait un poste de Stagiaire Développeur CRM Full-Stack sur des technologies qui m'étaient familières telles que Java et JavaScript.

Après un entretien téléphonique suivi d'un entretien présentiel qui se sont bien déroulés, la société m'a confirmé son intérêt et m'a proposé le poste. J'ai directement accepté puisque le cadre, la mission, et les premiers contacts me paraissaient bons donc aucune raison de refuser.

Je suis donc en stage chez DIMO Software dans les locaux de Bidart du 10 avril au 16 juin 2023 en tant que Stagiaire Développeur CRM Full-Stack.

1.2 Présentation de l’entreprise

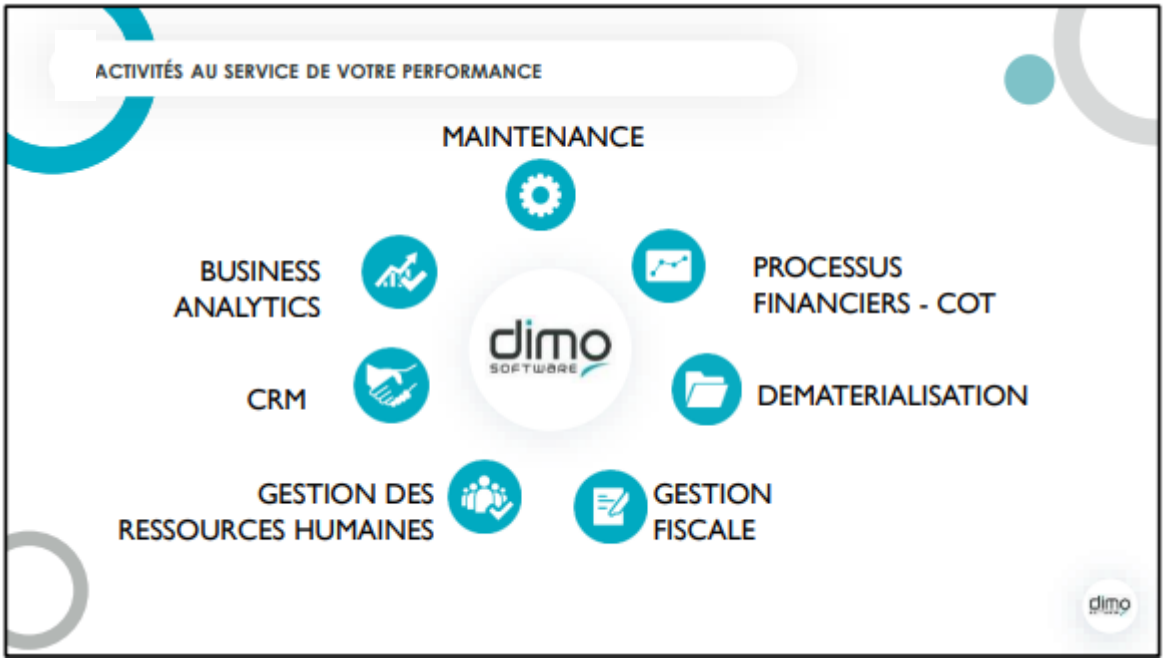
DIMO Software, éditeur et intégrateur de logiciels de gestion en mode SAAS

Le Software as a Service (SaaS) est un modèle de livraison de softwares cloud dans lequel le fournisseur de softwares cloud développe et maintient des logiciels d’applications cloud, fournit des mises à jour software automatiques et met des softwares à la disposition de ses clients via Internet sur une base de paiement à l’utilisation.

DIMO Software conçoit et accompagne la mise en œuvre de solutions de gestion innovantes qui complètent le système d’information de votre entreprise et répondent à vos besoins.

Basé sur Bidart, Lyon, Nantes, Paris, Kuala-Lumpur, Limerick, Mexico, Munich et Toronto, DIMO Software est divisé en plusieurs équipes dédiées dans chacun de ses domaines d’expertise, épaulées par des équipes transverses. Les « Dimosiens » assurent ainsi à nos clients compétences métiers, expérience, expertises produits et agilité sur plus de 30 solutions.

On retrouve donc 7 activités :



J’ai intégré la branche CRM de DIMO Software en tant que Stagiaire Développeur CRM Full-Stack, cette branche est centrée sur une application.

Avant de passer à la présentation de cet outil, qu’est ce qu’un logiciel CRM ?

Un CRM est caractérisé par la gestion de la relation client, le but principale d'approfondir ses relations avec ses clients, utilisateurs de services, collègues, partenaires et fournisseurs.

Vous bénéficiez d'une vue globale depuis un emplacement centralisé : un tableau de bord simple et personnalisable, qui retrace l'historique des interactions clients, le statut de leurs commandes, tout problème de service non résolu, et bien plus encore.

1.3 Présentation de YellowBox

La branche CRM de DIMO Software est donc axée sur le produit YellowBox, c'est une application CRM qui perdure depuis bientôt 20 ans.

Quant on s'intéresse aux grandes fonctionnalités, on retrouve les bases d'un CRM :

- Gestion des contacts
- Suivi des leads et des affaires commerciales
- Création d'actions et de rappels
- Mise en place et analyse des campagnes marketing
- Traitement des demandes clients et des réclamations
- Gestion des appels d'offres Catalogue produits
- Reporting et tableaux de bord

Vous pouvez trouver la définition des termes important d'un CRM comme un lead, une action… dans le **Glossaire**

YellowBox CRM

- **Evolutif** et personnalisable
- 100 % **français** (conception et hébergement)
- Ergonomique, **intuitif** et convivial
- Accessible partout, tout le temps (**PC, tablette et mobile**)
- Conforme au **RGPD**
- Disponible en mode **SaaS**
- Une solution **connectée** (interfaces et plugins)

Personnalisez facilement votre application selon vos besoins

Vous pouvez gérer les droits et les accès de vos collaborateurs et créer des règles pour vos différentes catégories d'utilisateurs. Le paramétrage illimité de vos écrans vous permet d'ajouter des tables, des champs, des listes ou encore des onglets pour répondre parfaitement aux besoins de chacun.

Le requêtage de la base peut se faire en mode partenaire ou avancé (SQL). Vous avez également la possibilité d'importer et d'exporter la totalité des données de façon manuelle ou automatique, ou encore de modifier vos données en masse.

La généricité est donc la force de ce CRM puisqu'il permet de toucher toutes sortes d'entreprises de secteurs différents.

Il est accessible depuis un navigateur donc sur n'importe quelle plateforme, ce qui réduit les coûts en termes de matériel et d'infrastructures, l'application est donc disponible sur toute plateforme (ordinateur, mobile, tablette…).

L'hébergement et l'application sont 100 % français et avec plus de 15 ans d'expérience, YellowBox est évolutif et performant pour répondre et s'adapter aux différentes sociétés avec plus de 1500 projets menés à bien.

Pour ce qui est de l'organisation de l'équipe, celle-ci est en plusieurs parties.

On retrouve l'équipe de développement basé à Bidart avec laquelle je travaille :

- **FUSADE Benoi**t, Responsable développement
- **SALERNO Christine**, Responsable qualité
- **DARRIEULAT Patrice**, Architecte développeur CRM
- **TORRES Julien**, Développeur CRM
- **PICARD Timothée**, Développeur CRM
- **DELLA LIBERA Vincent**, Developpeur CRM

Cet équipe intervient sur le coeur de YellowBox et tout son développement

On retrouve aussi une équipe du support basé à Lyon qui est chargée d'aider le client en cas de problème et de transmettre d'éventuels bugs qui seront ensuite testés puis corrigés par l'équipe de développement.

Afin d'accompagner les sociétés, on trouve la TMA (Tierce Maintenance Applicative) qui permet, après le déploiement du projet, de solliciter l'intervention d'un expert CRM afin de faire évoluer l'outil, son utilisation, et de corriger ses potentiels dysfonctionnements.

Tous ces services sont reliés par le responsable produit BALZAN Hervé qui est chargé de faire la traduction entre le besoin client qu'il récupère avec les clients ou depuis la TMA à l'équipe de développement qui elle développe la / les fonctionnalitée(s) demandée(s).

Enfin, il existe aussi une partie commerciale chargée de trouver des prospect afin de vendre le produit.

Commerce	Support	TMA
Nicolas BEGULE Pauline NOËL Samuel JUNOD CdC : Delphine PARISOT	**Pierre –Jérôme LUBAC** Isabelle MEYER Carine BUJALANCE Lauriane VIGNIEU Dominique BONNET	**CENTRE DE SERVICE** **Pierre-Jérôme LUBAC** Julien BRULLE Askia DABAL Hadrien DESCHET Estelle SIMONET

Développement	Produit + Qualité
Hervé BALZAN **Benoît FUSADE (B)** Patrice DARRIEULAT (B) Timothée PICARD (B) Julien TORRES (B) Vincent DELLA LIBERA (B) Christine SALERNO (B)	**PRODUIT** **Hervé BAZLAN** **TESTS** **Christine SALERNO** Nadine BAUTRY

2. Environnement

2.1 Gestion de projet

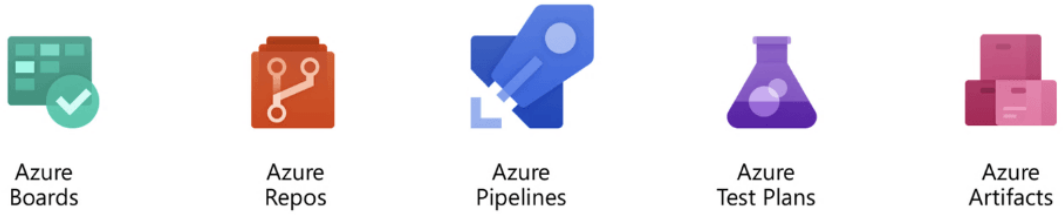
Vous l'aurez compris, l'équipe de développement intervient principalement sur YellowBox ainsi que sur quelques outils développés par eux-même.

Étant donné qu'ils travaillent donc sur le même produit depuis plus de 20 ans, la mise en place d'une gestion de projet agile leur a paru être le plus adapté.

En effet, lorsque qu'on pense à l'agilité on pense aux **sprint**, au **rétrospective**, ou encore au livraison, donc quoi de mieux que d'être toujours sur le même produit afin de mettre en place cette méthode.

Afin de mener à bien cette méthode, l'équipe de développement s'organise autour d'une application nommé "Azur DevOps"

C'est quoi Azur DevOps ?



DevOps est le processus d'intégration des équipes de développement et d'exploitation afin d'améliorer les collaborations et la productivité. Cela est rendu possible par l'automatisation des flux de travail et de la productivité pour une mesure continue des performances des applications.

Sur Microsoft Azure, il existe plusieurs outils permettant de réaliser les différentes tâches appliquées aux méthodes DevOps. Je vais détailler l'intérêt de certains de ces outils (ci-dessus) que j'ai eu à voir durant le stage.

Azure Boards : Azure Boards est un outil de gestion qui permet de planifier et de suivre le travail au sein d'un projet. On y retrouve les PBI (product backlog item) qui sont les tâches à réaliser et qui sont définis par certains points comme la priorité et l'effort définissant si le PBI est urgent et / ou difficile avec une valeur allant de 1 à 4 (du moins au plus).

Un PBI peut-être de différent type comme de l'analyse, du développement, de la sécurité et bien d'autres.

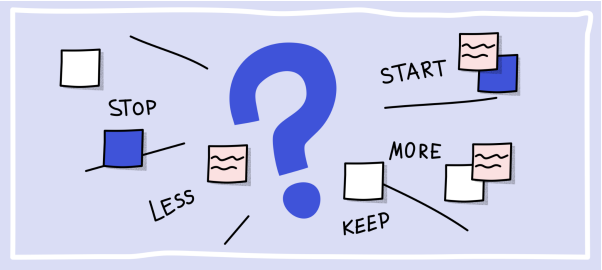
Azure Repos : Azure Repos permet le contrôle des sources et prend en charge les dépôts privés utilisant soit Git, soit Team Foundation Version Control. On y retrouve le système de branche comme présenté dans la partie **3.2.1.6** qui permet la collaboration entre tous les membres de l'équipe de développement. Et bien évidemment un système de version qui recense les versions de YellowBox au cour de son évolution

C'est donc un outil qui rassemble pas mal d'options afin de travailler en collaboration.

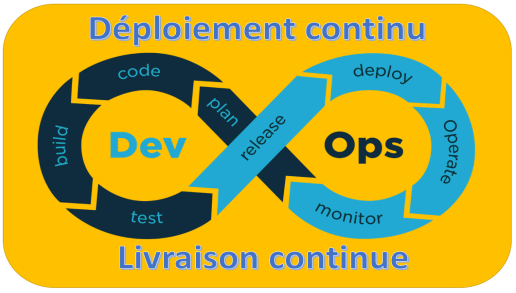
Durant mon stage, j'ai donc eu à participer / assister à plusieurs événements que l'on retrouve en agilité, je vais vous les présenter :

Daily meeting

Sprint Poker



Rétrospective



Livraison

Le Daily meeting : C'est une réunion que toute l'équipe effectue tous les matins à 9h00 que ce soit en distanciel ou en présentiel. Chacun vient détailler ce qu'il a fait hier, ce qu'il va faire aujourd'hui et si quelque chose le bloque, cela permet de faire un point rapide sur l'avancement de chacun.

Comme les autres membres de l'équipe, je répondais à ces questions afin d'avoir des retours et de savoir comment aller s'orienter ma journée .

Le Sprint poker : Cet étape est essentielle avant chaque début de sprint puisqu'elle permet de planifier les tâches qui seront à réaliser, de les attribuer et de les estimer que ce soit en temps ou en difficulté. J'ai eu à assister et à participer à deux d'entre elles où j'ai été assigné à certaines des tâches même si ce n'était pas aussi appuyé que le reste de l'équipe.

La Rétrospective : C'est un point de fin de sprint, cela permet de recenser ce qui a été bon et au contraire moins bon afin de s'améliorer pour le prochain sprint. Ce n'est pas un incontournable du moins pour l'équipe dans laquelle je travaille puisqu'il n'est pas à chaque fin de sprint mais il reste utile de le réaliser après plusieurs sprint par exemple afin d'avoir une vue générale.

Pour celui-ci j'ai eu à y assister une fois en étant pas réellement acteur puisque je venais d'arriver dans l'équipe et j'étais encore en phase de découverte de l'environnement de travail.

Livraison : Pour YellowBox il s'agit de livraison continue, c'est à dire que le produit est loué par des entreprise mais il est mis à jour et modifié tous les jours.

Bien évidemment à des degrés différents avec un système de version qui permet de spécifier la consistance du changement.

Actuellement le produit est en version 9.6, un troisième chiffre existe mais il est incrémenté tous les jours puisqu'il correspond aux corrections de bugs et aux modifications qui n'impactent pas le client. Le deuxième chiffre n'est incrémenté que lorsqu'une nouvelle version avec des nouvelles fonctionnalités arrive, environ tous les mois. Enfin, le premier chiffre n'est que très rarement changé et il n'y a pas de règle spécifié si ce n'est que le changement doit être assez conséquent.

C'est un processus continue auquel j'assiste puisque la responsable qualité est chargée de tester les fonctionnalités développées, de les injecter dans la nouvelle version avant d'enfin la publier. Ce n'est donc pas une sorte de réunion comme les autres rituels, néanmoins il est un incontournable auquel l'équipe se confronte tous les jours.

2.2 Outils

2.2.1 Développement

Plusieurs outils ont été à ma disposition afin de mener à bien ma mission de stage :

Eclipse

Eclipse est l'environnement de développement qui est utilisé par DIMO Software pour développer toutes sortes de fonctionnalités sur YellowBox.

Il est assez polyvalent étant donné que l'on peut intervenir dessus avec toute sorte de langages tels que JavaScript, PHP, C++, Java… afin de produire des applications web, mobile, logiciel… C'est un IDE que j'ai eu à utiliser durant ma formation donc dès le départ j'étais déjà assez familiarisé avec celui-ci.

Java

Java est le langage de programmation sur lequel repose en grande majorité YellowBox et est donc utilisé sous Eclipse (présenté précédemment).

Java est aujourd'hui l'un des langages les plus utilisés au monde, je n'étais donc pas étonné de le rencontrer, d'autant plus que j'ai aussi eu à l'utiliser durant ma formation

z

Apache Tomcat

Apache Tomcat ou simplement Tomcat est un serveur d'applications, plus précisément un conteneur web libre de servlets et JSP.

Tomcat est un serveur web également développé par Apache Software Foundation. Son nom officiel est Apache Tomcat. C'est un serveur HTTP mais il alimente les applications Java au lieu des sites web statiques comme le fait Apache Server que j'ai eu à manipuler

Visual Studio Code

Visual Studio Code est un environnement de développement tout comme Eclipse mais avec moins de polyvalence notamment sur les possibilités de développement avec celui-ci.

Durant mon stage j'ai eu à l'utiliser afin de produire du code JavaScript avant de le pousser sur l'interface Zapier en ligne, j'ai choisi personnellement d'utiliser cet outil car il est celui que j'ai le plus côtoyé durant ma formation et c'est celui ou je me sens le plus à l'aise

JavaScript

JavaScript est un langage de programmation qui permet de créer du contenu mis à jour de façon dynamique, de contrôler le contenu multimédia, d'animer des images, et presque tout ce à quoi on peut penser. Afin d'interagir avec l'API Zapier, il est nécessaire d'utiliser du code JavaScript afin de personnaliser ce que l'on souhaite mettre en place.

Git

Git est de loin le système de contrôle de version le plus largement utilisé aujourd'hui.

Il permet de travailler en équipe sur un même projet en évitant se marcher dessus grâce à la création de branches qui permet à chaque développeur de travailler sur sa branche en laissant la branche principale intacte afin qu'elle soit fonctionnelle. Ce n'est qu'à certaines étapes de projet que l'on rassemble le contenu des différentes branches sur la branche principale.

Cet outil est énormément utilisé par toute l'équipe de développement de YellowBox étant donné qu'ils sont plusieurs à travailler sur les mêmes fonctionnalités.

Pour ma part cet outil m'a permis de bien gérer les versions sur le code que j'ai produit que ce soit côté Zapier ou côté YellowBox. J'ai été le seul à intervenir sur le code donc je n'ai pas eu à gérer d'éventuelles conflits sur les branches, mais j'ai eu à revenir sur d'anciennes versions afin de retrouver quelque chose de fonctionnelle lorsque je touchais au code.

Jenkins

Jenkins est un outil logiciel d'intégration continu. Il permet de tester et de rapporter les changements effectués sur une large base de code en temps réel. En utilisant ce logiciel, les développeurs peuvent détecter et résoudre les problèmes dans une base de code et rapidement. Ainsi les tests de nouveaux builds peuvent être automatisés, ce qui permet

d’intégrer plus facilement des changements à un projet, de façon continue. L’objectif de Jenkin est en effet d’accélérer le développement de logiciels par le biais de l’automatisation.

Cet outil est aussi beaucoup utilisé par l’équipe de développement afin de tester le code qu’ils sont produit afin de fournir un code fonctionnelle en production.

Kibana

Kibana est une application frontend gratuite et ouverte qui s'appuie sur la Suite Elastic. Concrètement cet outil m’a permis de corriger les problèmes / bugs que j’ai pu rencontrer durant le développement côté YellowBox puisqu’il m’a permis de **“log”**; c’est à dire contrôler mon travail et mon avancé et vérifiez étape par étape que mon code fonctionne. Ce genre d’outil est essentiel lorsqu’on développe sinon on peut rapidement se retrouver dans un problème dont one ne connaît pas les causes

Sonar

Sonar permet d’inspecter du code afin de détecter d'éventuels bugs, vulnérabilités ou duplications.

Cela m’a permis de rendre mon code qualitatif et conforme aux bonne pratiques que choisissent de mettre en place l’équipe de développement

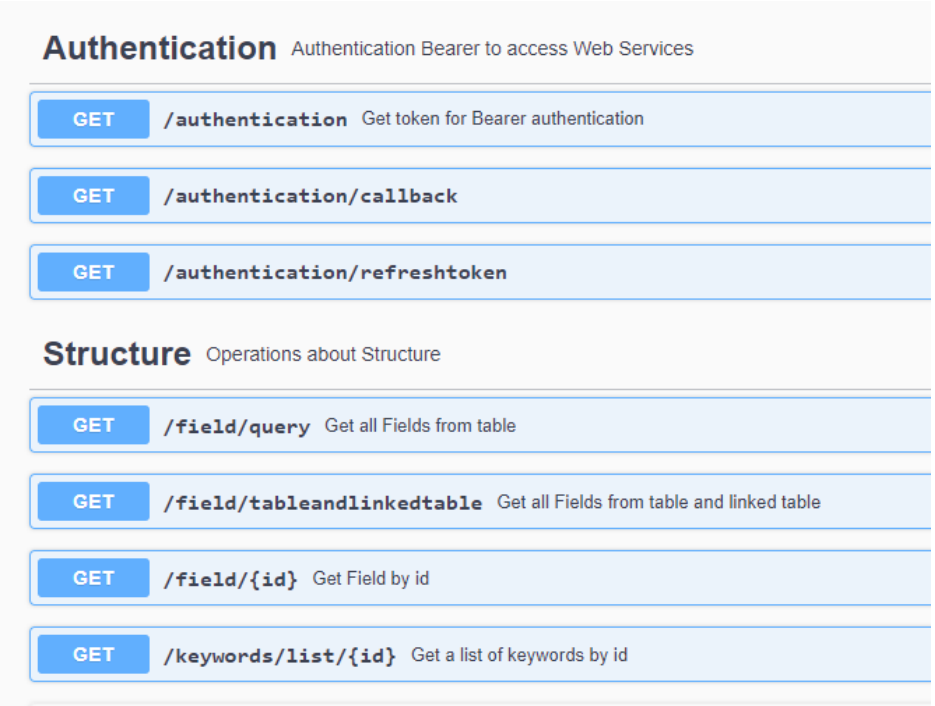
2.2.2 API

Les API ont été le coeur du sujet auquel s'oriente mon stage, je vais présenter celles que j'ai eu à manipuler

YellowBox

La première est tout simplement YellowBox, c'est une *API REST* qui manipule du JSON et qui est utilisée à travers des Web Services.

Son utilisation est facilitée par la documentation qu'on met à disposition l'équipe de développement sur Open API.



On retrouve donc différentes possibilités comme la récupération des différents champs d'une table par exemple. Cela permet de récupérer certaines données facilement afin de pouvoir par la suite les transformer, ce que j'ai beaucoup eu à faire côté Zapier.

Name	Description
body * required *(body)*	ImportRecord contains record to create c Example Value | Model


```
{
  "record": {
    "id": "68",
    "table": {
      "id": "68",
      "name": "contact"
    },
    "values": [
      {
        "field": {
          "id": "72",
          "name": "nom",
          "dbName": "NOM",
          "keywordListId": "string",
          "fieldType": "ALPHANUMERIQUE",
          "linkedTableId": "string",
          "linkedFieldId": "string",
          "dateType": "string"
        },
        "value": "Etchepare"
      }
    ],
    "linkedRecords": [
      {
        "fieldId": "string",
        "idRecord": "string",
        "calculated": false
      }
```


Parameter content type

application/json

La plupart des données renvoyées sont au format JSON, cela permet de les manipuler facilement.

Postman

Postman est une application gratuite qui permet de tester des requêtes vers une API directement depuis une interface graphique. Elle m'a été très utile afin de prendre en main le fonctionnement des web services de YellowBox. Elle permet de tester si les requêtes que j'envois sur YellowBox passent avant de les produire sur Zapier notamment

Microsoft Teams

Teams est un outil de communication et de visioconférence très utilisé par DIMO Software. Il permet de garder collaborer avec les équipes à Lyon avec des appels ou des messages, mais aussi ici à Bidart lorsque certains sont en télétravail.

Il est assez efficace puisqu'il permet donne accès aux différents emplois du temps de chacun afin de planifier facilement des réunions par exemple. Ce n'était pas une découverte pour moi puisque j'ai déjà utilisé cet outil à l'IUT lors de différents projets

3. Mission

Intitulé : Développer un plugin d'interface de données (via API de Web Services nommé Zapier)

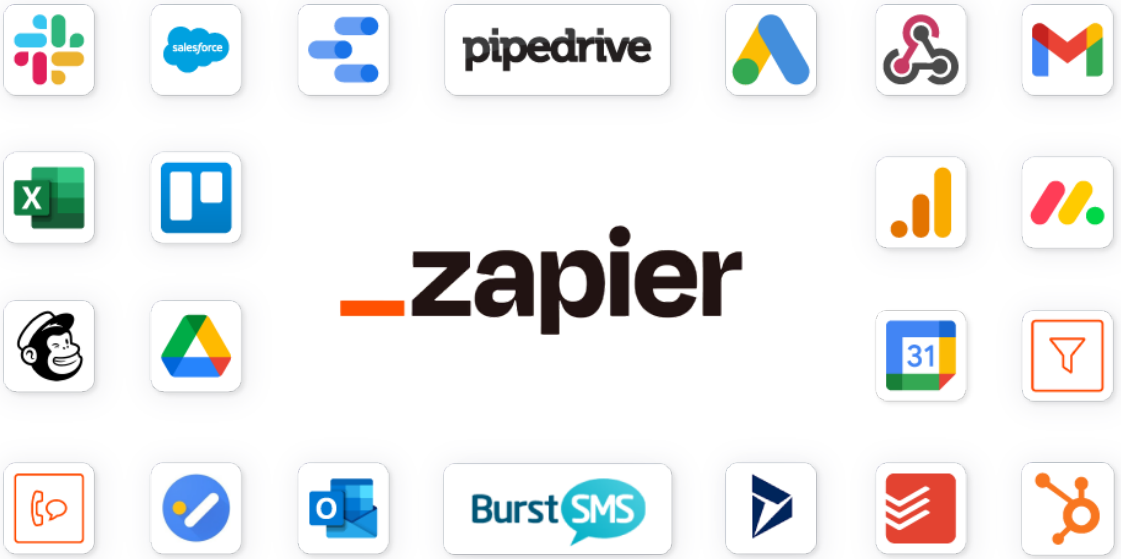
3.1 Présentation de l'outil utilisé

Le produit sur lequel je travaille est donc YellowBox, on retrouve sur celui-ci la notion de plugin.

Cela peut être comparé à des sortes d'extensions qui peuvent être développées de manière assez séparées sans toucher à l'architecture du produit. Parmi les extensions déjà présentes on retrouve des produits développés pour se connecter à des produits déjà existants comme Google Maps, ou Office 365. Mais aussi des plugins qui ont été développés entièrement par l'équipe de développement comme Yellow Scancard permettant de scanner des cartes de visites.

Aujourd'hui il existe des milliers d'applications dans le monde et intégrer celles-ci sur un même produit est une tâche interminable.

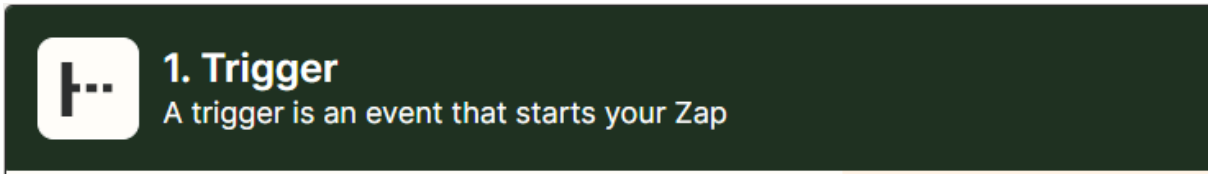
C'est la qu'intervient **Zapier**, un service web capable d'automatiser des actions entre plusieurs applications Web que vous utilisez tous les jours



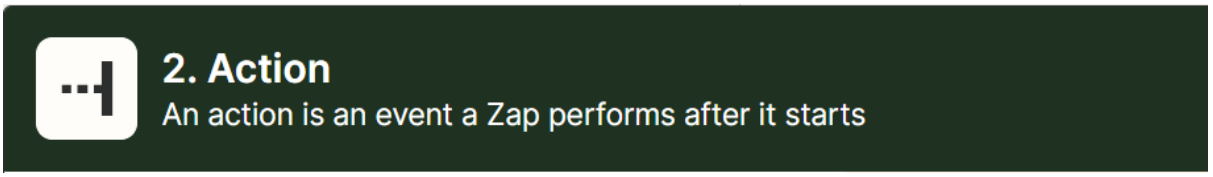
Concrètement, comment peut-on s'en servir ?

Quelques termes sont à définir sur l'utilisation de Zapier afin de bien comprendre :

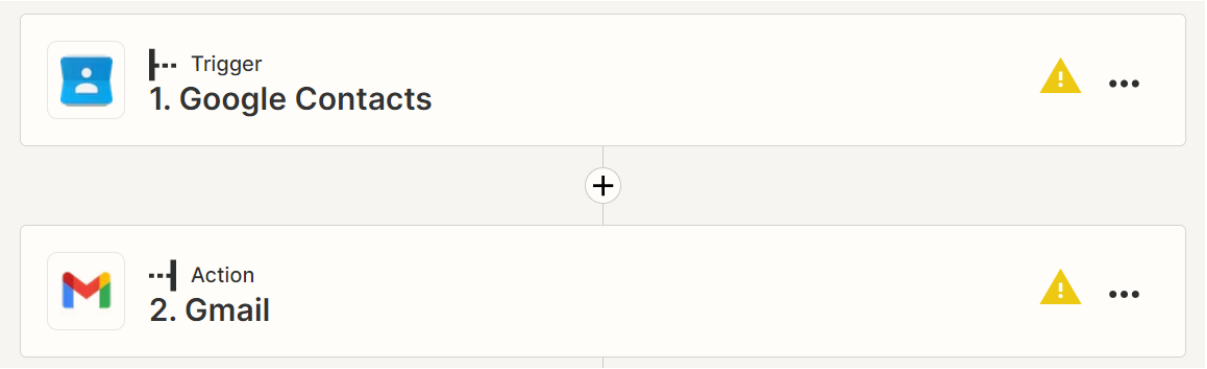
- **Le déclencheur :** Celui-ci constitue la base de ce que l'on veut mettre en place, il est chargé de réagir lorsque ce qu'on lui a demandé de surveiller ce produit. Par exemple, la réception de mail peut-être un déclencheur ou alors la création d'un contact, une fois qu'un de ces événements se produit le déclencheur transmet les données à l'action



- **Les actions :** Celles-ci sont donc appelées par le déclencheur lorsqu'un événement s'est produit, elles peuvent aussi de nature divers et varié. Allant de l'envoie d'un mail, à la création d'un fichier ou encore la planification dans un calendrier. Il est possible d'enchaîner plusieurs de ces actions si on en a le besoin



- **Les Zaps :** Celui-ci constitue l'association d'un déclencheur et une action. Il est possible d'en créer plusieurs selon nos besoins et resteront fonctionnels et prêts à s'exécuter de jour comme de nuit.



3.2 Intérêt de l’entreprise pour Zapier

Parlons maintenant de l'intérêt de l’équipe sur l'interfacement de YellowBox sur Zapier, il y a 2 raisons principales à ce que l’entreprise souhaite s’interfacer avec cet outil.

3.2.1 Infinité de possibilités

Comme je l’ai dis précédemment, il n’est pas possible de s’interfacer avec tous les outils de la planète. Mais grâce à un outil comme Zapier, il suffit de mettre à disposition les déclencheurs et / ou des actions que vous jugez intéressantes pour que Zapier interconnecte votre application avec les 5000 applications qu’il recense.

Évidemment tout n’est pas possible et cet outil limite principalement à des actions simples comme la création d’un contact mais cela peut déjà se relever utile pour un logiciel CRM

3.2.2 Visibilité

Sur Zapier on retrouve globalement toutes les plus grosses entreprises mondiales avec leur applications telles que Gmail, Outlook, Drive, Excel…

Avoir son logo à côté de telles entreprises n’est pas anodin et peut pourquoi pas faire connaître l’application à un utilisateur régulier de Zapier.

On peut très bien aussi imaginer un potentiel client qui hésite entre 2 logiciels CRM qui lui paraissent cohérent, il va chercher les extensions possibles avec d’autres logiciels et c’est là que Zapier peut être une valeur ajoutée.

3.3 Contenu de la mission

Vous l'aurez compris, mon sujet de stage est donc partagé entre YellowBox et Zapier mais la prise en main de ces outils s'est faite assez différemment.

3.3.1 Veille technologique

YellowBox

Il faut savoir que j'avais très peu de connaissance sur ce qu'était un CRM avant d'arriver dans cette entreprise, j'ai donc découvert l'application durant le début du stage. Avec des présentations résumés du responsable produit et d'un membre de l'équipe de développement j'ai pu rapidement comprendre l'utilité d'un produit CRM et plus particulièrement YellowBox vis à vis d'une entreprise.

Pour la partie développement, celle-ci a été facilité puisque je n'ai pas eu à prendre en main l'application étant donné que j'ai seulement travaillé sur la partie plugin (expliqué précédemment).

De plus, j'ai utilisé une documentation développeur faite par l'équipe de développement permettant d'entrer dans le développement de l'application assez simplement avec des exemples de fonctions, plugins… pour comprendre un peu l'architecture et le fonctionnement.

Lien de la documentation :

https://ybhub.mydimosoftware.com/documentation/utilisateur/prod/public/

Zapier

Pour Zapier c'était assez différent, à l'exception d'un membre de l'équipe qui avait eu à essayer Zapier il y a 2 ans dans le but d'envisager un futur développement sur celui-ci, aucun membre de l'équipe ne connaissait concrètement Zapier et son utilisation, son paramétrage…

J'ai donc eu à faire une petite analyse sur comment marche l'outil, quelles sont ses possibilités / limites, les documentations disponibles afin de comparer avec ce qui est possible sur YellowBox.

Après réunion, avec Benoît Fusade (tuteur) et Timothée Picard (développeur qui effectue mon suivi), l'idée d'effectuer une veille concurrentielle sur les concurrents CRM disponible sur Zapier est naturellement née

3.3.2 Veille concurrentielle

L'objectif était donc d'analyser ce que proposent les concurrents à YellowBox sur Zapier afin de se caler si possible sur les mêmes fonctionnalités.

Le principale concurrent CRM sur lequel j'ai porté mon regard a été SalesForce :

Le choix de ce concurrent a été porté par le fait que c'est la plus grosse solution CRM de France et de plus, DIMO Software accompagne certains clients dans la mise en place de ce produit.

En effet ils sont partenaires intégrateur avec Salesforce, c'est donc une source d'inspiration permettant de connaître ce qu'il se fait chez les concurrents et possiblement s'en inspirer.

J'ai donc pu regarder les actions / déclencheurs qu'ils proposaient sur Zapier

3.3.2.1 Déclencheurs

On retrouve 4 types de déclencheurs :

New Outbound Message Instant

Triggers when a new outbound message is received in Salesforce.

New Record

Triggers when a record of the specified Salesforce object (ie. Contact, Lead, Opportunity, etc.) is created.

Updated Field on Record

Triggers when a field of your choosing (ie. email address, status) is updated on any record of a specified Salesforce object (ie. Contact, Lead, Opportunity, etc.).

Updated Record

Triggers when any record of the specified Salesforce object (ie. Contact, Lead, Opportunity, etc.) is updated.

Globalement, ces déclencheurs surveillent le trafic sur tout ce qui est création et modification de fiche. Il y aussi la notion de “Outbound Message” qui caractérise un retour lorsqu’un appel à un point de terminaison comme une API a été fait

3.3.2.2 Actions

Create Attachment

Creates a new attachment (max 25MB).

Create Contact

Creates a new contact in Salesforce.

Create File

Creates a new File (max 25MB).

Create Lead

Creates a new lead in Salesforce.

Create Record

Creates a new record of a specified Salesforce object (ie. Contact, Lead, Opportunity, etc.).

Create Record (UTC)

Creates a new record of a specified Salesforce object (ie. Contact, Lead, Opportunity, etc.). Date inputs will be treated as if they are in UTC

Update Contact

Updates an existing contact in Salesforce.

Update Lead

Updates an existing lead in Salesforce.

Update Record

Updates an existing record for a specified Salesforce object (ie. Contact, Lead, Opportunity, etc.).

Update Record (UTC)

Updates an existing record for a specified Salesforce object (ie. Contact, Lead, Opportunity, etc.). Date inputs will be treated as if they are in UTC

SEARCH

Find Record

Finds a record of a specified Salesforce object by a field and value you choose. Optionally, create a record if none are found.

Find Record by Query

Finds a record of a Salesforce object using a Salesforce Object Query (SOQL) WHERE clause.

Find Record(s)

Finds a record of a specified Salesforce object by a field and value you choose (with line-item support).

On retrouve globalement les principales actions de création sur les contacts, les leads mais aussi une création générique : “Create Record” qui permet de choisir quel *fiche* on veut impacter

La même chose pour les modifications avec des actions prédéfinis sur les contacts et les leads mais aussi une modification générique : “Update Record” qui permet de choisir quelle *fiche* on veut impacter.

Enfin la notion de “Find” très important pour mettre à jour des *fiches* puisqu’il permettent de trouver selon des critères (nom, prénom pour un contact par exemple) quel est la / les fiche(s) précise(s) que l’on doit modifier

3.3.3 Travail à réaliser

A la suite des veilles technologiques et concurrentielles, j’ai eu a faire une réunion un point avec mon tuteur de stage Benoit FUSADE et Timothée PICARD qui effectue mon suivi afin de voir quelles sont les tâches à réaliser.

3.3.2.1 Choix des déclencheurs

Pour ce qui est des déclencheurs, on a décidé de fusionner un peu l’existant afin de créer un seul déclencheur qui gère 2 cas.

Il va permettre de surveiller constamment si une fiche est créée mais aussi si une fiche est modifié, en résumé, le moindre changement est détecté et avertit Zapier

L'utilisateur peut donc associer toutes sortes d'actions dès que quelque chose bouge sur son YellowBox.

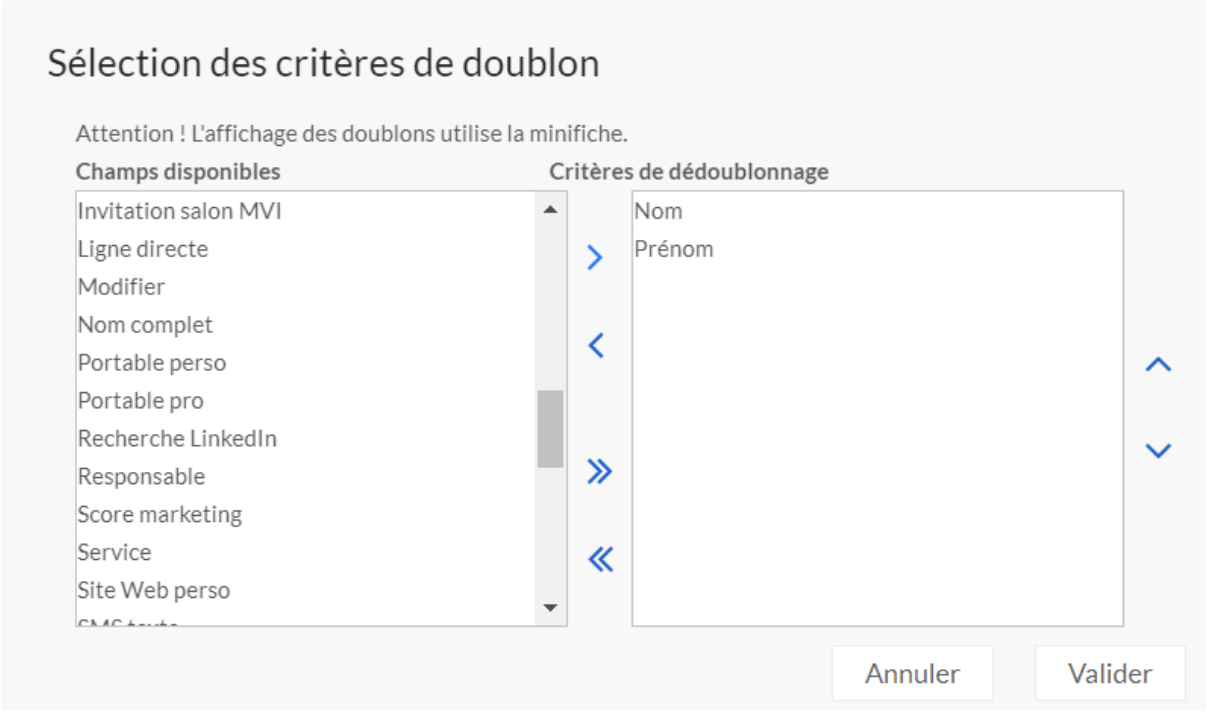
A noter que la notion de déclencheurs est déjà présente sur YellowBox, il est possible de paramétrer certains événements et de définir quand est ce que l'on veut le déclencher que ce soit à la création, suppression ou modification d'une fiche

3.3.2.2 Choix des actions

Pour ce qui est des déclencheurs, on a aussi décidé un seul qui réunit 2 cas :

Elle va permettre de paramétrer la création d'une fiche sur YellowBox en fonction des données récupérées sur le déclencheur paramétré auparavant

Grâce aux critères de dédoublonnage présent sur YellowBox, si la fiche que l'on crée est déjà présente, elle sera mise à jour.



Lorsque je vais créer un contact qui aura le même nom et prénom qu'un contact existant, alors, c'est ce contact existant qui sera mis à jour avec les nouvelles informations comme le numéro de téléphone ou autre

3.3.2.3 Documentation

Afin qu'un utilisateur soit capable de paramétrer ses déclencheurs / actions sur Zapier, il m'a été demandé de produire une documentation détaillant quelques cas d'utilisations qui sont amenés à être traités.

De plus, au-delà de la partie utilisateur, j'ai aussi eu à produire une documentation sur la maintenance. Pour ce qui est de la partie faite côté YellowBox, cela n'a pas été nécessaire car l'équipe de développement est habituée à travailler sur les plugins.

En revanche, pour la partie Zapier j'ai détaillé dans une documentation les ressources requises, quelques termes importants et quelques règles afin de comprendre rapidement l'environnement si quelqu'un vient à reprendre mon travail

3.3.2.4 Récapitulatif

Tâches à réaliser	
Avoir un déclencheur permettant d'avertir Zapier lorsqu'une fiche d'un type choisi est créé ou modifié sur YellowBox	
Avoir une action permettant la création ou la modification d'une fiche du type que l'on souhaite	
Avoir une doc de maintenance sur l'utilisation de Zapier (CLI…)	
Avoir une doc utilisateur sur le paramétrage des zaps	

4. Résultats

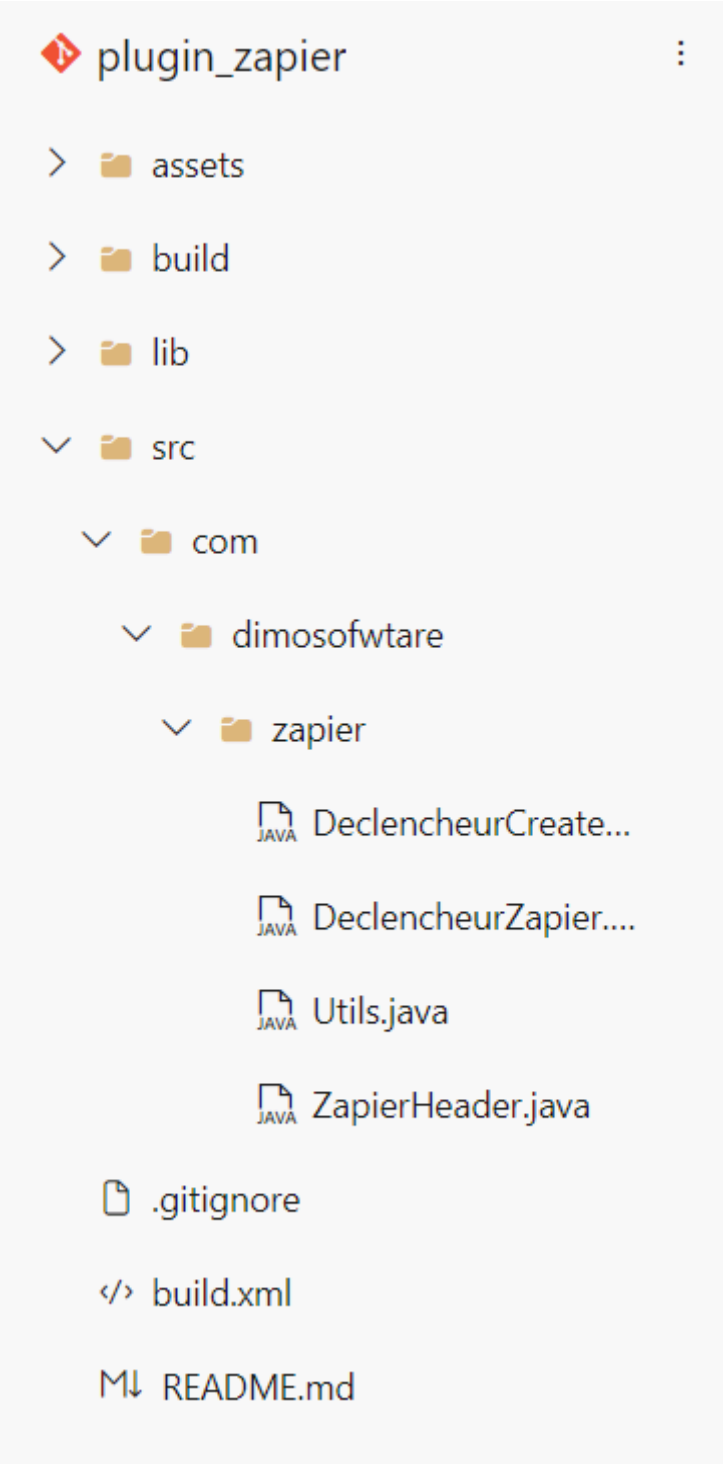
Tâches réalisées	
Avoir un déclencheur permettant d'avertir Zapier lorsqu'une fiche d'un type choisi est créé ou modifié sur YellowBox	X
Avoir une action permettant la création ou la modification d'une fiche du type que l'on souhaite	X
Avoir une doc de maintenance sur l'utilisation de Zapier (CLI…)	X
Avoir une doc utilisateur sur le paramétrage des zaps	X

La documentation utilisateur et la documentation de maintenance sont disponibles en **Annexe**

4.1 Architecture technique

J'ai donc eu à produire du code sur Zapier mais aussi sur YellowBox

4.1.1 Côté YellowBox



4.1.1.1 Actions

Pour ce qui est des actions qui envoient des données depuis Zapier à YellowBox, absolument rien n'est a paramétré ici

4.1.1.2 Déclencheurs

Les 4 fichiers présents dans le dossier Zapier sont important afin que le déclencheur fonctionne correctement

---

ZapierHeader.java

Objectif : Ce fichier sert à décrire le plugin que j'ai créé sur YellowBox à travers plusieurs fonctions

Plusieurs méthode sont à définir :

- getName() : doit retourner le nom du package tel qu'il sera visible dans le store
- getDescription() doit retourner une description du plugin telle qu'elle sera visible dans le store
- getId() doit retourner une référence UNIQUE du package, cette valeur sert de référence dans le store. Son unicité sera validée ou non lors de l'upload du package dans le store.
- getVersion() doit retourner la version du package à l'aide d'un objet Version(X,X,X) comprenant trois niveaux : principale, majeur et mineur.
- getCodeVersion() doit retourner un numéro de version qui s'incrémente à chaque modification permettant ainsi de connaître quelle version est plus récente qu'une autre.
- getMinYellowboxVersion() doit retourner la version de YellowBox minimale pour exécuter ce package. L'objet de retour est également un objet Version(X,X,X).
- getLogo() doit retourner le chemin vers une image qui sera utilisée comme logo dans le store. Ce chemin est relatif par rapport au dossier "assets", par exemple si mon image se nomme "logo.png" et qu'elle se trouve à la racine du dossier assets, la méthode doit retourner le String suivant : "logo.png".

- getProvider() doit retourner le nom du créateur du plugin
- geteMail() doit retourner l'email de contact du développeur du plugin

---

DeclencheurZapier.java

Vous pouvez retrouver la définition d'un webhook dans le glossaire

Plusieurs méthode sont à définir :

- call() : exécute le traitement voulu à chaque appel de la fonction
- getName() : nom de la fonction sans espace ni caractères spéciaux
- getTitle() : traduction de la fonction, présenté lors de la sélection à l'utilisateur
- getDescription() : description de la fonction

Afin de bien paramétrer le webhook, 2 informations sont nécessaires :

- L'URL que YellowBox doit appeler pour avertir Zapier
- L'id de la table que YellowBox doit surveiller

Ces informations arrivent depuis le premier contact entre Zapier et YellowBox afin de paramétrer le déclencheur.

Objectif :

1) Récupérer l'id de la table choisi par l'utilisateur ainsi que l'url qu'a créé Zapier afin de l'avertir sur des changements intervenus sur cette table précisément

```
// Récupère les données de l'url
String hookUrl = param.getStringValue(HOOKURL);
String idTable = param.getStringValue(IDTABLE);
```

   Ces 2 informations arrivent dans une URL qu'a transmis Zapier lors du paramétrage du déclencheur côté Zapier

2) Ensuite YellowBox se rend dans un fichier nommé "data.json", il vérifie si l'id qu'il vient de récupérer existe. A partir de la se dégage plusieurs scénarios :

   Si le fichier n'existait pas car pour une raison ou une autre il a été supprimé la fonction l'aurait créé

- Si l'id existe déjà, il vérifie que l'url est aussi identique à ce qu'il vient de récupérer. Si c'est égal il ne fait rien, sinon il l'a remplacé
- Si l'id n'existe pas, il l'ajoute avec l'url associé

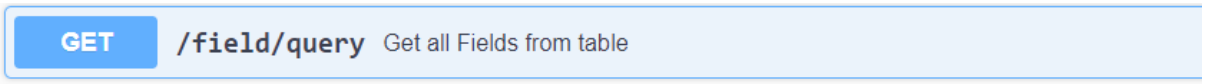
Si le fichier n'existait pas car pour une raison ou une autre il a été supprimé la fonction l'aurait créé

```
// Écriture de l'objet JSON dans un fichier


String repertoireFichier = getYellowboxEntryPoint().getCheminDocjoint() +
Utils.CHEMINDATAJSON;
        File dir = new File(repertoireFichier);
        File f = new File(repertoireFichier + Utils.NOMDATAJSON);
        try {
                dir.mkdirs();
                retourCreate = f.createNewFile();
                ajouterObjetJson(f, hookUrl, idTable);
                if (retourCreate) {
                      getYellowboxEntryPoint().getLog().putDebug("Le fichier a
été créé avec succès.");
                } else {
                    getYellowboxEntryPoint().getLog().putDebug("La création du
fichier a
                    échoué ou le fichier existe déjà.");
                }

        } catch (IOException e) {
                getYellowboxEntryPoint().getLog().putError("Erreur lors de la
création du fichier : "
            + e.getMessage());
        }
```

Extrait du fichier JSON :


```
[{"idTable":"69705","hookUrl":"https://hooks.zapier.com/hooks/stand
ard/15073428/8d95e826c12b4c24a232ae7603fd6145/"},
{"idTable":"245","hookUrl":"https://hooks.zapier.com/hooks/standard
/15073428/600117a539ca462bb33d161fbcc20f70/"},
{"idTable":"334","hookUrl":"https://hooks.zapier.com/hooks/standard
/15073428/82a81df2708945358d87715c5d807ba4/"},
{"idTable":"216","hookUrl":"https://hooks.zapier.com/hooks/standard
/15073428/6a0dc2449b634e29806e5aaf85b2543d/"},
{"idTable":"8437","hookUrl":"https://hooks.zapier.com/hooks/standar
d/15073428/d3eb0784ff774518bad7f94ce346b443/"},
{"idTable":"45","hookUrl":"https://hooks.zapier.com/hooks/standard/
15073428/1af2020be9454d23b3bde6a37ab9d1d0/"},
{"idTable":"contact","hookUrl":"https://hooks.zapier.com/hooks/stan
dard/15073428/fb9e794cbedb4c419395b1b22879d0c5/"},
{"idTable":"68","hookUrl":"https://hooks.zapier.com/hooks/standard/
15073428/1998a1a948e843e4a6477b426cfe3270/"}]
```


3) Maintenant ce data.json a été crée ou modifié ou bien simplement consulté, YellowBox va récupérer tous les champs de la table que l'utilisateur a choisi grâce au web service suivant :




```
//Recupère les champs de la table choisi
List<Field> fieldOfTable = getYellowboxEntryPoint().getServices().
getFieldService().getFieldsFromTable(idTable);
```

4) Il les stocke dans un objet JSON et les envoie à la bonne URL qu'il trouve dans le fichier JSON

```java
// Conversion de la liste JSON en tableau JSON
JSONArray arrayFieldOfTable = new JSONArray(fieldOfTable);

// Création de l'objet résultat
JSONObject resultField = new JSONObject();

// Parcours de chaque élément dans la liste
for (int i = 0; i < arrayFieldOfTable.length(); i++) {
    JSONObject jsonObject = arrayFieldOfTable.getJSONObject(i);

    // Récupération du nom du champ et ajout dans l'objet résultat
    String fieldName = jsonObject.getString("name");
    resultField.put(fieldName, fieldName);
}


// Envoi des données à Zapier
Utils.sendJsonData(hookUrl, jsonTable, getYellowboxEntryPoint().getLog());
```

Grâce à cela, l'utilisateur pourra associer les champs YellowBox avec les champs d'une action comme la création d'un contact Outlook.

DeclencheurCreateRecord.java

Plusieurs méthode sont à définir :

- call() : exécute le traitement voulu à chaque appel de la fonction
- getName() : nom de la fonction sans espace ni caractères spéciaux
- getTitle() : traduction de la fonction, présenté lors de la sélection à l'utilisateur
- getDescription() : description de la fonction

Objectif :

1) Récupérer la référence de la fiche qui a été créé ou modifié

```
String idFiche = (String) getYellowboxEntryPoint().get("REF_FICHE");
```

2) Regarder son type de fiche et en fonction allez chercher dans le data.json la bonne url à appeler

```
// Parcourir le JSON pour trouver la bonne hookUrl
JSONArray jsonArray = new JSONArray(json);
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    String idTableFichier = jsonObject.getString("idTable");
    hookUrl = jsonObject.getString("hookUrl");
    if (idTableFichier.equals(idTable)) {
        // La bonne hookUrl a été trouvée
        getYellowboxEntryPoint().getLog().putError("La hookUrl correspondant à
        l'idFiche " + idFiche + " est " + hookUrl);
        break;
    }
}
```

3) Envoyer à cette URL la, tous les champs de la fiche qui a été créé ou modifié

```
RecordParam recordParam = new RecordParam();
recordParam.setIdRecord(idFiche);
recordParam.setIncludeAllFields(true);

Record record = getYellowboxEntryPoint().getServices()
.getRecordServiceV3().getRecordFromId(recordParam);

List<Value> listeRecord = record.getValues();

JSONObject resultRecord = new JSONObject();

for (Value value : listeRecord) {

    resultRecord.put(value.getField().getName(), value.getValue());
}


Utils.sendJsonData(hookUrl, resultRecord, getYellowboxEntryPoint().getLog());
```

Zapier recevra toutes ses données et saura par rapport aux champs qu'a choisi l'utilisateur lequel faire passer à l'action suivante.

Utils.java

Objectif : Ce fichier sert de classe commune à DeclencheurZapier.java et DeclencheurCreateRecord.java, il comprend 2 méthodes qui concernent l'envoi des données à Zapier.

La première fonction se nomme sendJsonData()

Elle prend en paramètre l'URL qui a été récupéré depuis le fichier JSON, l'objet JSON à transmettre à Zapier ainsi qu' un loger permettant de gérer les erreurs.

Cette fonction se charge de préparer et paramétrer la requête Http à envoyer

```
public static void sendJsonData(String hookUrl, JSONObject jsonData, ILog
loger) {

            // URL Zapier
            URL url = null;
            try {
                url = new URL(hookUrl);
            } catch (MalformedURLException e) {
                loger.putError(e.getMessage());

            }

            // Connexion HTTP
            HttpURLConnection connexion = null;
            if (url != null) {
```

```
                try {
                    connexion = (HttpURLConnection) url.openConnection();
                } catch (IOException e) {
                    loger.putError(e.getMessage());
                }
            } else {
                loger.putDebug("L'url ne peut pas être nul");
            }

            // Paramétrage connexion
            if (connexion != null) {
                connexion.setDoOutput(true);
                try {
                    connexion.setRequestMethod("POST");
                } catch (ProtocolException e) {
                    loger.putError(hookUrl);
                }
                                connexion.setRequestProperty("Content-Type",
"application/json");
            } else {
                loger.putDebug("La connexion ne peut pas être nul");
            }

         // Conversion de la chaîne JSON en tableau d'octets
                                                byte[]     input     =
jsonData.toString().getBytes(StandardCharsets.UTF_8);

            sendDataAndCheckResponse(connexion, input, loger);
        }
```

La deuxième fonction se nomme sendDataAndCheckResponse()

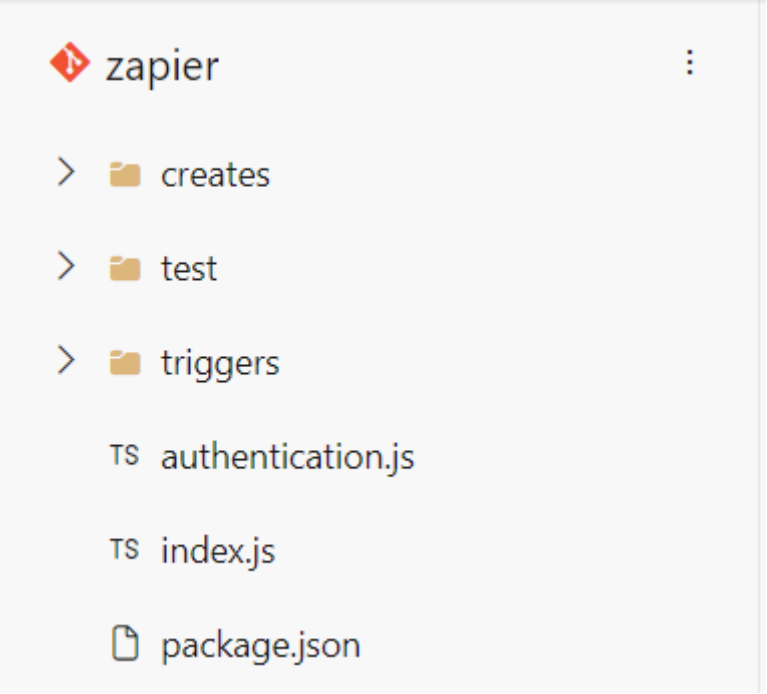
Elle prend en paramètre une requête http préparée dans la fonction précédente, l'objet JSON qui a été transformé afin d'être passé dans une requête et un loger, toujours pour gérer les erreurs.

Cette fonction envoie donc les données à Zapier tout en vérifiant qu'elles arrivent bien grâce au loger.

```
private static void sendDataAndCheckResponse(HttpURLConnection connexion,
byte[] input, ILog loger) {
            int statusCode = 0;
             // Envoi des données via la connexion HTTP
            if (connexion != null) {
                try (OutputStream os = connexion.getOutputStream()) {
                    os.write(input, 0, input.length);
                } catch (IOException e) {
                    loger.putError(e.getMessage());
                }
            } else {
                loger.putDebug("La connexion ne peut pas être nul");
            }



         // Vérification de la réponse HTTP
            if (connexion != null) {
                try {
                    statusCode = connexion.getResponseCode();
                } catch (IOException e) {
                     loger.putError("Erreur lors de la récupération du code de
réponse HTTP : " + e.getMessage());
                }
            } else {
```

```
                    loger.putDebug("La connexion HTTP est nulle, impossible de
récupérer le code de réponse.");
            }
            if (statusCode == HttpURLConnection.HTTP_OK) {
                loger.putDebug("Données postées avec succès.");
            } else {
                    loger.putDebug("Erreur lors de la post des données : " +
statusCode);
            }

        }
```

4.1.2 Côté Zapier



4.1.1.2 Actions

Dossiers qui va recenser les actions YellowBox :

create_record_selection.js

Ici on retrouve un seul fichier JavaScript qui s'occupe de la création d'une nouvelle fiche sachant que la mise à jour est géré côté YellowBox

Sur ce fichier, on va retrouver 2 constantes importantes : **perform** et **getInputFields**

perform :

Objectif : Elle est donc de transformer ce qu'elle reçoit afin de le fournir à YellowBox via un web service dans le bon format. Celle-ci stocke ce que retourne la méthode "async ()". Elle en paramètre "bundle" qui regroupe les données renseignées par l'utilisateur dans les formulaires Zapier.

```
const perform = async (z, bundle) => {
  const selectedOption = bundle.inputData.table;
  const inputDataKeys = Object.keys(bundle.inputData);
  const inputDataValues = inputDataKeys.map((key) => bundle.inputData[key]);
  const urlUser = bundle.authData.url;
  const createRecord = "/ws/record/v3";
  const options = {
    url: `${urlUser}${createRecord}`,
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Accept: 'application/json',
    },
    params: {},
    body: {
      record: {
```

```
        table: {
          id: selectedOption,
        },

        values: extractKeysAndValues(inputDataKeys, inputDataValues),
      },
        fieldIdDuplicateManager: "0",
    },
  };

  return z.request(options).then((response) => {
    response.throwForStatus();
    const results = response.json;



    var obj = { id: results };
    return obj;
  });
};
```

Il faut savoir j'effectue une requête au service web suivant :

POST	/record/v3	Create or update a record with specific description for integration

En se référant à la documentation OpenApi, on constate qu'un format précis doit-être respecté pour que la demande soit traitée.

Tout n'a pas à être rempli mais dans notre cas plusieurs lignes sont importantes,

les voici :

```
{
  "record": {
    "id": "string",
    "table": {
      "id": "string",
      "name": "string",
      "traduction": "string"
    },
    "values": [
      {
        "field": {
          "id": "string",
          "name": "string",
          "dbName": "string",
          "keywordListId": "string",
          "fieldType": "ALPHANUMERIQUE",
          "linkedTableId": "string",
          "linkedFieldId": "string",
          "dateType": "string"
        },
        "value": "string",
        "operator": "EQUAL"
      }
    ],
    "linkedRecords": [
      {
        "fieldId": "string",
        "idRecord": "string",
        "calculated": false
      }
    ],
    "editable": false
  },
  "fieldIdDuplicateManager": "string",
  "typeImport": 0,
  "applyRules": false
}
```

Ils permettent de savoir quelle table cibler et quel champ avec la valeur a lui mettre.

Pour arriver à transformer dans le bon format, on retrouve donc une fonction qui est appelée depuis perform : extraxtKeysAndValues

Elle prend donc en paramètre le tableau des différents champs de la table choisi et y associe la valeur qu'a saisi l'utilisateur avec

```
function extractKeysAndValues(jsonKeys, jsonData) {
  var extractedJson = [];
  for (var i = 0; i < jsonData.length; i++) {
    if (jsonKeys[i] != 'table') {
      extractedJson.push({
        field: { id: jsonKeys[i] },
        value: jsonData[i],
      }
      );
    }
  }
  return extractedJson;
}
```

getInputFields :

Celle-ci stocke ce que retourne la méthode “async ()”.

Elle en paramètre “bundle” qui regroupe les données renseignées par l’utilisateur dans les formulaires Zapier.

Objectif : Elle est chargée d’effectuer une requête à YellowBox pour récupérer les différents champs d’une table afin de créer le formulaire associé pour l’utilisateur avec le bon typage.

```
const getInputFields = async (z, bundle) => {
  const selectedOption = bundle.inputData.table;
  const urlUser = bundle.authData.url;
  const options = {
    url: `${urlUser}/ws/field/query?idTable=${selectedOption}`,
    method: 'GET',
    headers: {
      Accept: 'application/json',
    },
    params: {},
  };
```

```
return z.request(options).then((response) => {
  response.throwForStatus();
  const results = response.json;



  const newJson = [];
  var transformation;

  for (const cle in results) {
    if (results.hasOwnProperty(cle)) {
      transformation = results[cle].fieldType;
      switch (transformation) {
        case 'ALPHANUMERIQUE':
          transformation = 'string';
          break;
        case 'DATE':
          transformation = 'dateTime';
          break;
        case 'BOOLEEN':
          transformation = 'boolean';
          break;
        case 'TELEPHONE':
          transformation = 'number';
          break;
        case 'ZONETEXTE':
          transformation = 'text';
          break;
        case 'ENTIER':
          transformation = 'number';
```

```
            break;
          default:
            transformation = 'string';
            break;
        }
        const obj = {
          key: results[cle].id,
          type: transformation,
          label: results[cle].name,
        };
        newJson.push(obj);
      }
    }

    return newJson;
  });
};
```

Cette méthode fait appel au web service suivant :

GET	/field/query	Get all Fields from table

En lui passant l'id de la table que l'utilisateur a choisi, je suis capable de récupérer tous les champs d'une table.

J'ai mis en place un switch afin de récupérer le typage des champs YellowBox pour l'adapter au typage Zapier.

4.1.1.3 Déclencheurs

Dossiers qui va recenser les actions YellowBox :

- declencheur_create_update.js
- tables_ajout.js

Ici on retrouve deux fichiers JavaScript qui s'occupent de la création d'une nouvelle sachant que la mise à jour d'une fiche est gérée côté YellowBox.

---

declencheur_create_update.js

On retrouve 2 constantes importantes au bon fonctionnement du déclencheur :

- performSubscribe : permet d'exploiter le webhook mis en place côté YellowBox en lui envoyant plusieurs paramètres saisis par l'utilisateur. Son URL YellowBox, l'id de la table qu'il a choisi, l'autorisation du plugin ainsi que l'url hook fournis par Zapier pour que YellowBox puisse l'appeler lorsque qu'un changement intervient
- performUnsubscribe : exactement le même contenu que le performSubscribe afin que Zapier puisse faire l'inverse, c'est à dire de desabonner du webhook si il en a le besoin

performSubscribe / performUnsubscribe

Objectif : Ces 2 fonctions récupère donc les données saisis par l'utilisateur afin de les envoyer à YellowBox, c'est à dire :

- L'url de son YellowBox
- L'autorisation du plugin YellowBox associé
- L'id de la table que l'utilisateur a choisi de surveiller

Elle envoie aussi l'url hook que YellowBox devra appeler si un événement s'est produit.

```
const performSubscribe = async (z, bundle) => {
  const hookUrl = bundle.targetUrl
```

```
  const authorization = bundle.inputData.authorization;
  const urlUser = bundle.authData.url;
  const idTable = bundle.inputData.trigger;
  const options = {

url:`${urlUser}/plugin/callwebhook.do?token=${authorization}&idTable=${idTable
}&hookUrl=${hookUrl}`,
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Accept: 'application/json',
    },
    params: {},
  };

  return z.request(options).then((response) => {
    response.throwForStatus();
    const results = response.json;
    return results;
  });
};
```

Tout cela est envoyé directement à une url qui appelle le webhook paramétré sur YellowBox comme expliqué précédemment.

tables_ajout.js

On retrouve une constante importante dans ce fichier, c'est le "perform"

perform

Objectif : Elle est chargée de créer un menu déroulant reprenant toutes les tables de Yellow afin que l'utilisateur puisse choisir lors de la création.

Celle-ci stocke ce que retourne la méthode "async ()".

Elle en paramètre "bundle" qui regroupe les données renseignées par l'utilisateur dans les formulaires Zapier.

```
const perform = async (z, bundle) => {
  const urlUser = bundle.authData.url;
  const getTables = "/ws/table";
  const options = {
    url: `${urlUser}/ws/table`,
    method: 'GET',
    headers: {
      'Content-Type': 'application/json',
      Accept: 'application/json',
    },
    params: {},
  };

  return z.request(options).then((response) => {
    response.throwForStatus();
    const results = response.json;

    const transformedData = results.map(item => {
```

```
      return {
        traduction: item.name,
        id: item.id,
        name: item.traduction

      };
    });

    return transformedData;
  });
};
```

Cette fonction fait appel au web service suivant :

GET /table Get all tables

Je récupère toutes les tables sous cette forme :

```
[
  {
    "id": "string",
    "name": "string",
    "traduction": "string"
  }
]
```

Je transforme ensuite la structure afin de mettre le champ traduction avec la clé “name” car Zapier prend obligatoirement name comme nom d’affichage.

Cela permet à l'utilisateur d’avoir : “Appels d’offres” à l’écran et non pas “c69705appeldoffre" qui n’est pas très parlant.

4.2 Choix de développement

Lors du développement que j'ai eu à réaliser des choix en concertation avec l'équipe car je me suis confronté à quelques contraintes.

4.2.1 Le système de déclencheur

Contrainte : Comme dit précédemment, un système de webhook est mis en place afin que Zapier puisse se connecter à YellowBox pour récupérer les alertes instantanément lorsqu'une fiche est modifiée ou créée

Néanmoins une question s'est posé lors du choix de la technologie utilisé avec le polling

Le polling est utilisé par beaucoup d'applications puisqu'il est assez simple à comprendre. C'est un système qui interroge à intervalle régulier une application afin de détecter si un changement s'est produit.

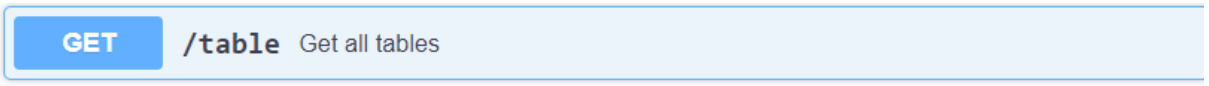
Il est possible d'utiliser ce mécanisme sur Zapier et il est assez facile à mettre en place

Solution : Néanmoins, il est certes moins compliqué qu'un webhook étant donné qu'il n'y aurait rien à paramétrer côté YellowBox mais il est bien moins efficace.

Si on prend le cas d'un utilisateur qui paramètre son zap, cela peut-être très contraignant d'attendre 2, 3, 5 minutes pour que les modifications soient prises en compte. D'autant plus quand on sait que l'autre système proposé (webhook) est instantané à l'utilisation et que l'équipe de développement a déjà eu à créer ce genre de système donc la mise en place a été facilité.

4.2.2 Des noms parlants

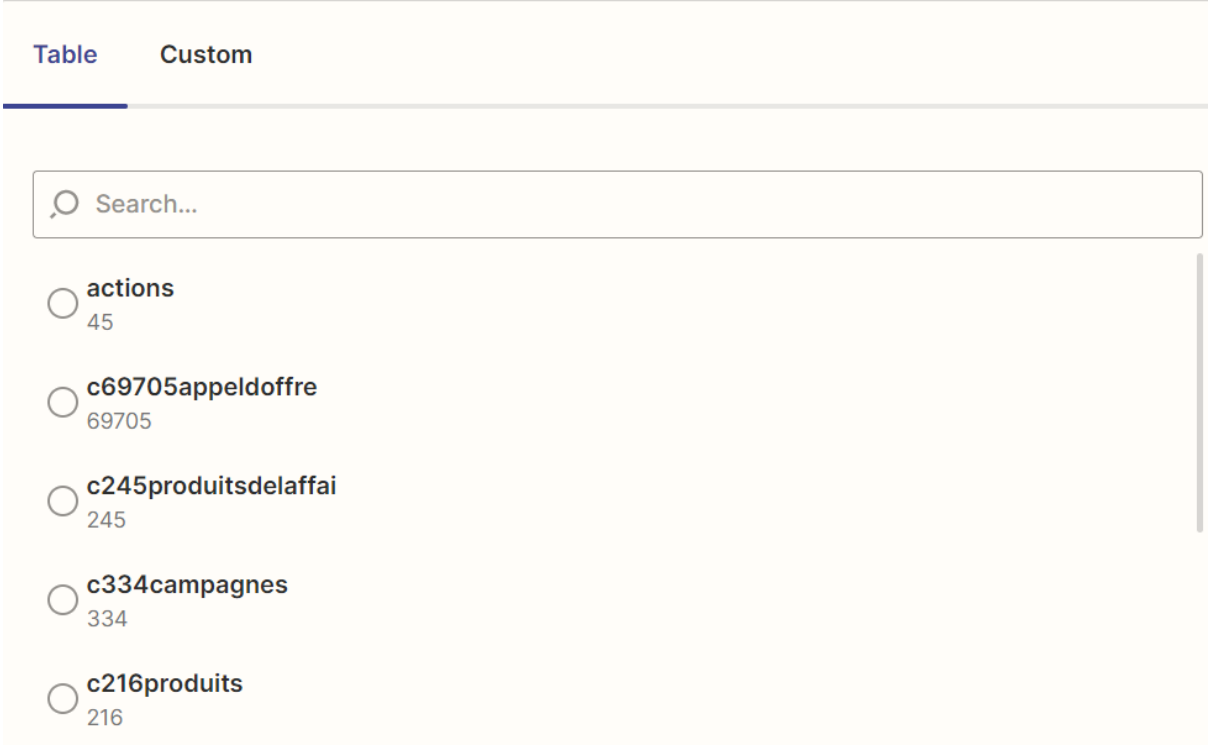
Contrainte : Partant de web services qui sont déjà établies je ne devais pas être amené à y toucher mais seulement à exploiter leurs résultats.



Néanmoins ça n'a pas été tout à fait le cas sur ce web service ci dessous puisqu'il me manquait certaines informations afin de produire quelque chose de qualitatif à l'utilisateur.

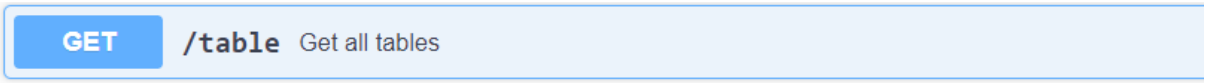
En effet, les 2 champs renvoyés par cette requête sont "id" et "name", l'id me sert évidemment à cibler la table que je souhaite et pouvoir par la suite la récupérer. Le champ

“name” n’est pas très parlant ou du moins pas correct pour être utilisé lors d’une interface utilisateur (voir ci-dessous).



Ceci est le menu déroulant permettant de choisir le type de fiche que l’on souhaite traiter lors de l’action ou du déclencheur, et comme vous le voyez le nom est caractérisé par un nom sans majuscule et avec des chiffres, ce qui n’est pas montrable à l’utilisateur.

Solution : Je me suis donc attardé sur la modification du web service afin de récupérer la traduction des types de fiches



Celui-ci renvoie maintenant le champ “traduction” qui est le champ utilisé sur le reste de YellowBox pour afficher proprement le contenu à l'utilisateur

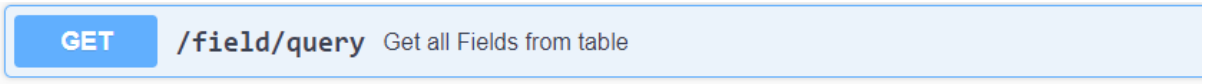
```
[
  {
    "id": "string",
    "name": "string",
    "traduction": "string"
  }
]
```

Il fallait quand même faire attention que le fait de rajouter ce champ ne casse pas l'existant qui utilise ce web service puisque qu'en effet ici, je sors du cadre du plugin et je touche a des choses beaucoup utilisées.

Cependant, j'ai vite été fixé là- dessus puisque des tests ont déjà été établis par l'équipe de développement sur ces web services à travers Jenkins. Et il s'est avéré que ces changements ne posaient pas de soucis sur l'existant

4.2.3 Les champs obligatoires

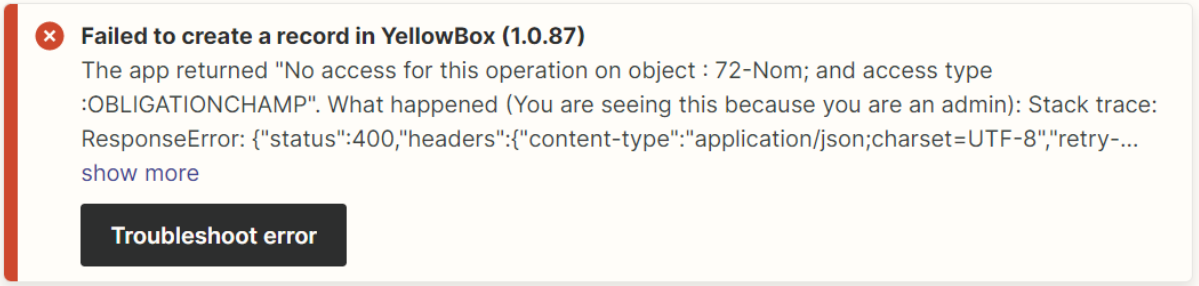
Contrainte : Lors des actions de création ou de mise à jour de fiche, l'utilisateur est amené à manipuler des formulaires créés par Zapier a partir des champs renvoyés par YellowBox via ce web service



Cette requête est exécutée avec l'id de table en paramètre et elle permet de récupérer aussi le typage afin que l'utilisateur soit averti sur ce qu'il doit saisir.

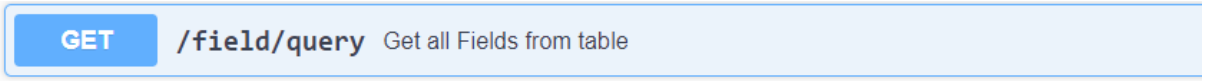
Néanmoins, une information assez importante est manquante puisque l'information permettant de savoir si le champ est obligatoire ou non n'est pas disponible.

Cela pose un problème puisque dans le cas où l'utilisateur ne renseigne pas un champ obligatoire, il se retrouve avec ce genre d'erreur :



Ce qui peut-être très perturbant pour un utilisateur

Solution : La première idée a donc été de modifier ce web service afin d'y intégrer l'information qui permet de savoir ou non si le champ est obligatoire



Après concertation avec l'équipe qui connaît bien l'architecture et qui connaît le niveau de complexité pour le mettre en place, cette idée a vite été écartée.

La modification n'entrainera en soit pas grand impact comme pour le type des fiches (fait précédemment) mais le problème vient de l'accès à l'information disant si le champ est obligatoire ou non.

La deuxième idée a donc été d'essayer de travailler cette erreur pour qu' elle renvoie un message plus agréable à l'utilisateur, en lui indiquant précisément ce qu'il ne va pas.

Néanmoins cette idée a été écartée pour le moment puisqu'il est nécessaire d'avoir un compte professionnel Zapier. Il est prévu à l'avenir et notamment lorsque YellowBox sera en phase bêta sur Zapier de souscrire à un compte professionnel et de gérer ce genre d'erreur.

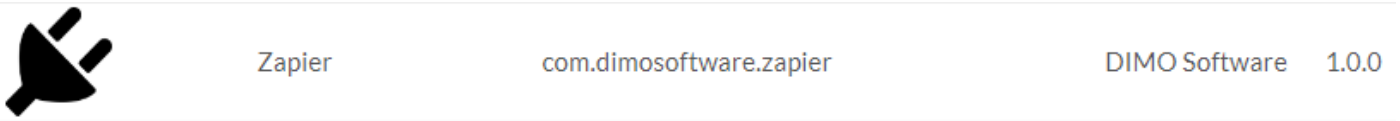
De plus, l'utilisateur qui utilisera Zapier est habitué à utiliser YellowBox régulièrement et donc à manipuler les créations mise à jour de fiche. Les champs qui sont obligatoire sont donc connu par l'utilisateur et il en va donc de sa responsabilité que de faire attention aux champs qu'il saisit.

4.3 Présentation du produit final

4.3.1 Paramétrage YellowBox

Afin de parfaitement s'interconnecter avec Zapier, il est nécessaire de paramétrer le plugin que j'ai développé côté YellowBox en administrateur.

Pour cela, mon plugin est compilé sous forme d'un fichier .jar qu'il faut importer dans un environnement YellowBox.



Sur celui-ci il faut paramétrer la fonction de webhook ainsi que la fonction qui s'occupe de déclencher (expliquées précédemment).

Nom

Zapier webhook

Plugin

zapier

Instance de

zapierwebhook (Webhook)

eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhdXRoMCJ9.C

Générer nouveau

URL d'accès : https://cedricdev.myyellowboxcrm.com/yellowbox/plugin/callwebhook.do

header : 'Authorization' = Bearer

eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhdXRoMCJ9.QnVI9MIx0GjCgG7iKANS0GIVGHQhyFztCJbt3W8jMXQ

Exécution du webhook avec l'utilisateur

Administrateur

Paramètre(s) entrant(s)

Nom

CreateRecordDeclencheur

Plugin

zapier

Instance de

Zapier create record (fonction)

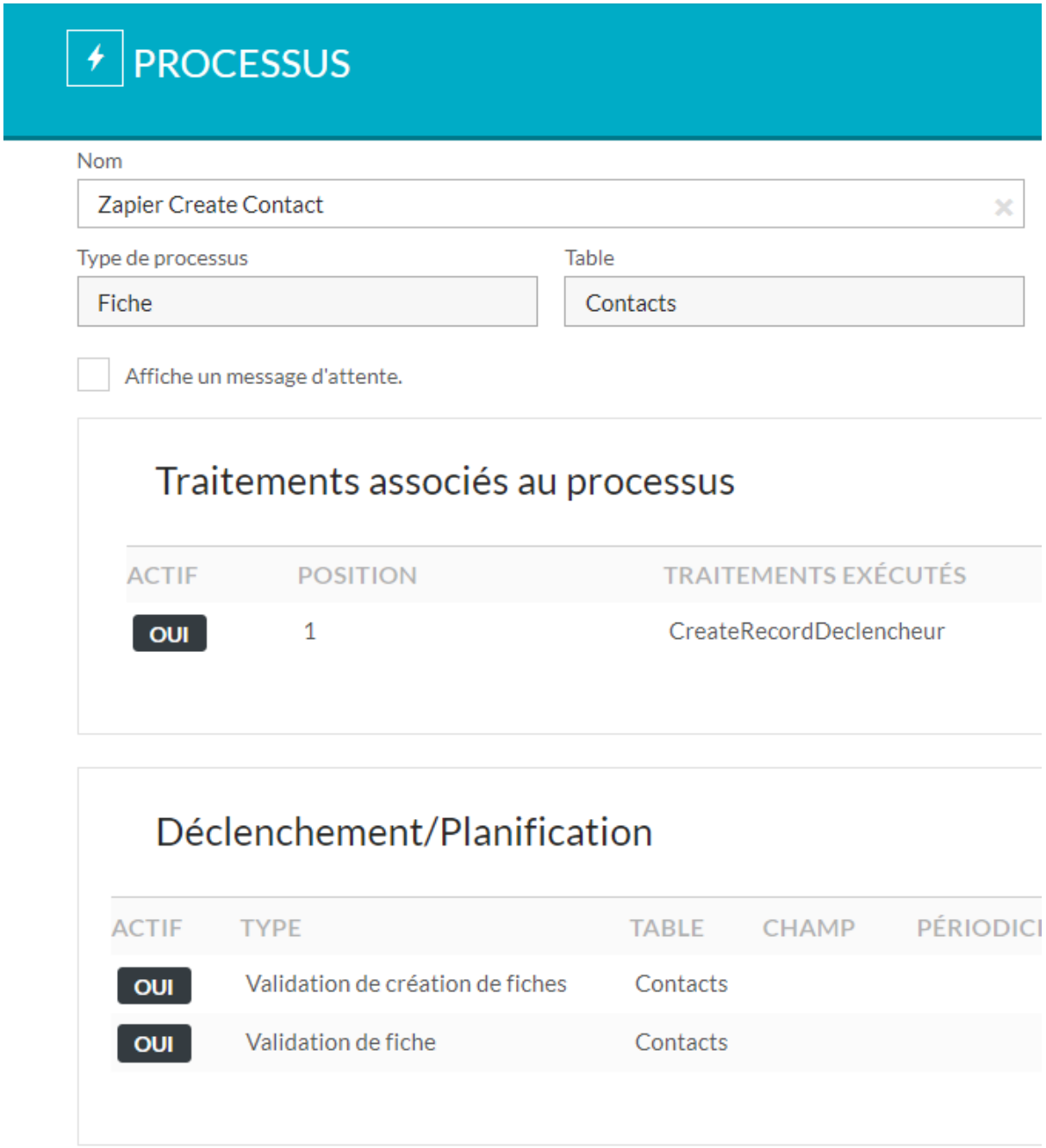
☑ dans le contexte d'une fiche

Table

Contacts

Paramètre(s) entrant(s)

Par la suite, il faut intégrer cette fonction dans un processus, c'est ce processus qui va permettre de faire tourner en continu la surveillance sur la table choisi :

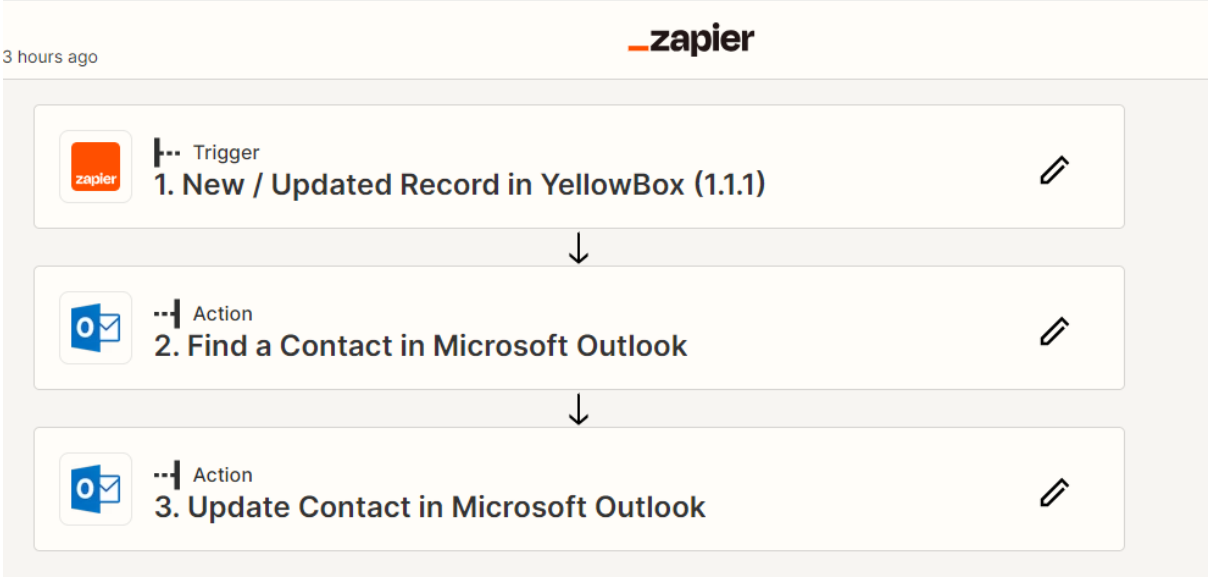


Tout ce paramétrage est détaillé précisément dans la documentation utilisateur disponible en annexe.

4.3.2 Démonstration

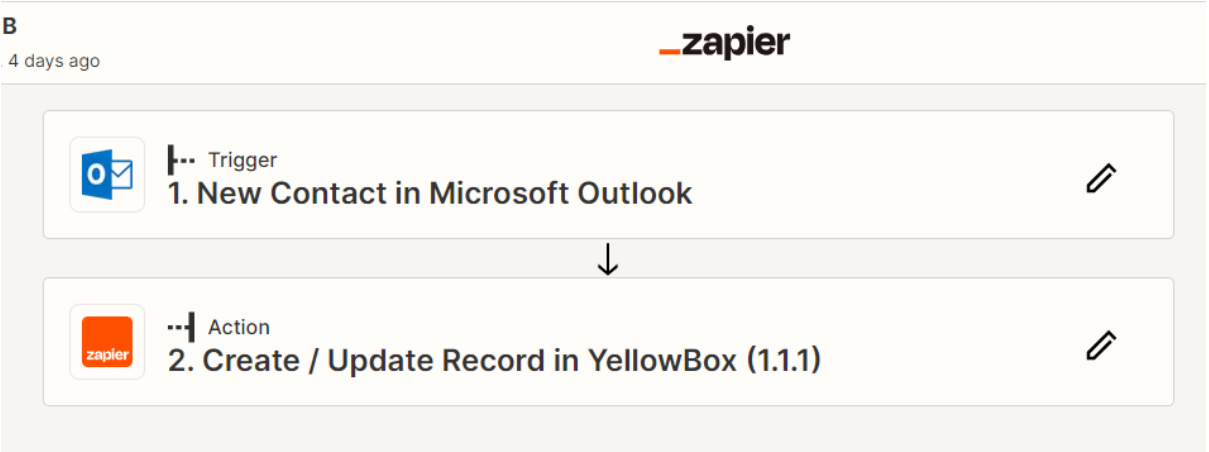
YellowBox est donc maintenant capable de se connecter à toutes sortes d'applications disponibles sur Zapier;

Voici un modèle de Zap permettant de surveiller les contacts YellowBox afin que toute modification ou ajout soit fait aussi sur Outlook de manière presque instantanée.



Exemple :

Toujours avec Outlook, le sens inverse avec la possibilité de surveiller les contacts Outlook afin que toute modification ou ajout soit aussi fait sur YellowBox de manière presque instantanée.

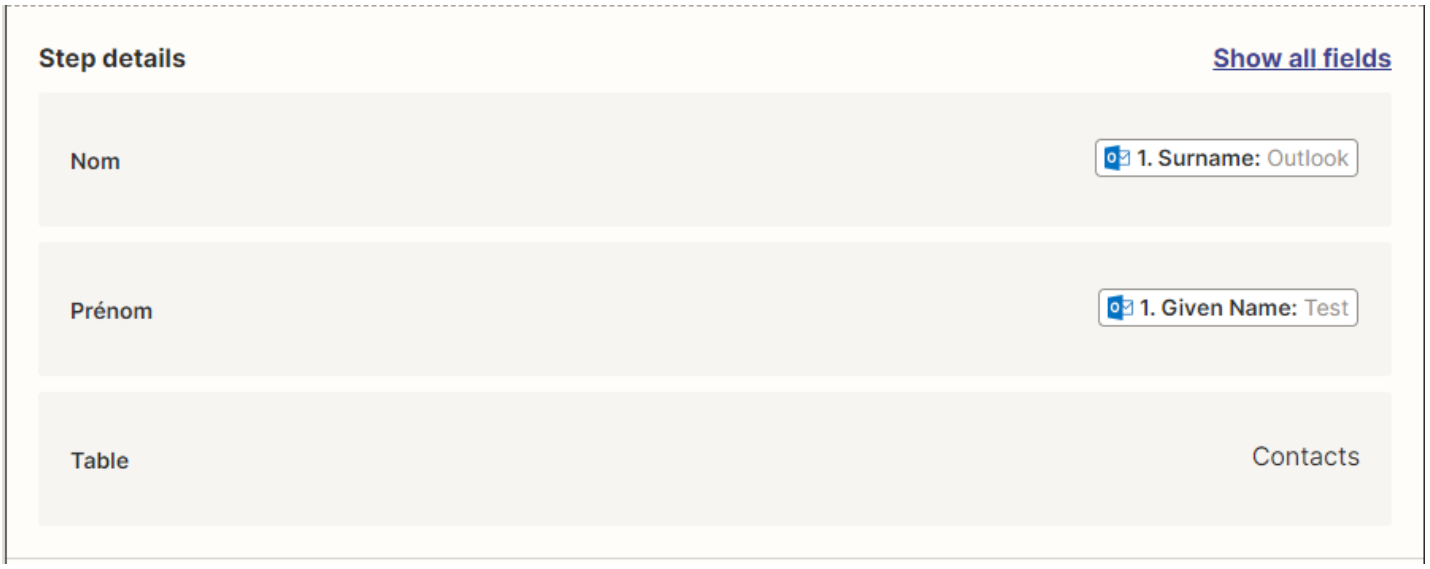


Exemple :

Ces 2 zaps peuvent facilement être créés en 5 minutes, une fois activé plus besoin d'y toucher.

Leurs paramétrage nécessitent seulement quelques étapes :

- une connexion basique pour Outlook et YellowBox.
- l'autorisation de webhook que l'on trouve dans le paramétrage de YellowBox (partie …) AAAAAAAAAAAAAAAAAAAAA
- choisir la tables et les champs surveillés



Dans le deuxième exemple, on voit que j'ai choisi de créer un contact avec le nom et le prénom renseigné par l'utilisateur. Dans le cas où ce contact existe déjà, le contact sera mis à jour avec les critères de dédoublonnage expliqués précédemment

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

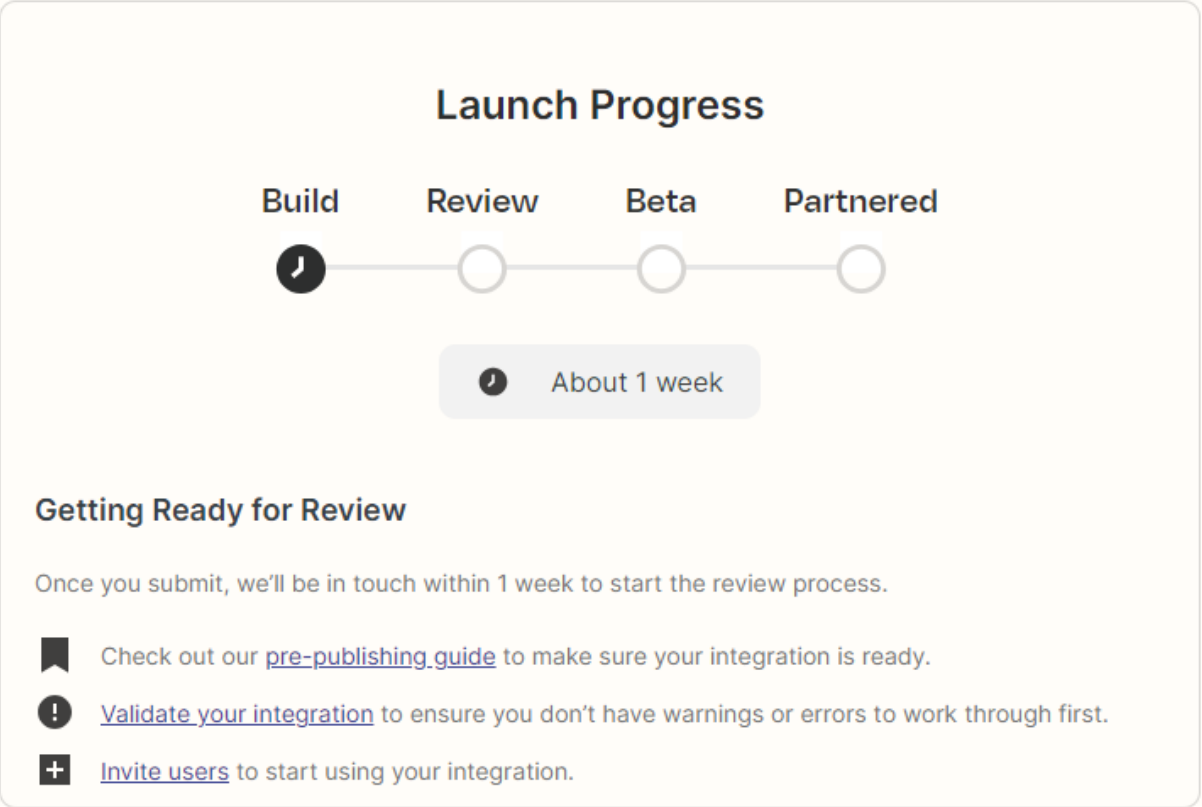
Tout le paramétrage précis est disponible dans la documentation utilisateur disponible en annexe

4.3.3 Evolutions prévues

Build

Actuellement, l'intégration YellowBox n'est pas encore accessible au public sur Zapier bien qu'elle soit fonctionnelle, elle est pour le moment en phase "Build' car il reste encore quelques ajustements.

Il reste une phase de publication de l'intégration qui sera réalisé par l'équipe de développement :



Un lien d'accès va être fourni au personnes chargées du commerce autour de YellowBox afin de pouvoir tester les différentes possibilités et récupérer quelques retours.

Notamment au niveau de la description de l'application à saisir, les noms souhaités… ainsi que d'éventuels problèmes, bugs.

Une fois cela fait l'équipe va souscrire à un compte Zapier afin de pouvoir gérer les différentes erreurs que peut causer l'intégration. C'est une étape importante car la gestion d'erreurs constitue un des critères à valider pour publier l'intégration

Lien vers les consignes d'examen Zapier :

https://platform.zapier.com/partners/integration-review-guidelines

Review

L'application va ensuite être soumise afin qu'une personne du support Zapier examine et teste l'intégration.

Cette phase peut durer avec d'éventuels échanges sur certains points pas clairs ou à requalifier.

Beta

Après vérification, l'application sera disponible au public avec le tag "Bêta"

Durant cette phase YellowBox sera disponible dans le répertoire d'applications de Zapier et il sera possible d'étoffer l'intégration avec plusieurs ajouts :

- Des documents d'aide, à l'image de la documentation fournis en Annexe
- Des modèles Zap prêts à l'emploi où l'on peut recenser les plus utilisés
- Intégrer Zapier dans YellowBox, effectuer de la pub en quelque sorte pour montrer au client de l'entreprise que Zapier est disponible
- Augmenter l'utilisation active, le but de cette phase est que le plus de personnes possibles utilisent votre intégration afin de récupérer des commentaires favorisant l'optimisation
- Gérer les bugs et les demandes de nouvelles fonctionnalités, cela permet de régler les problèmes qui n'auraient pas été détectés durant la publication et d'avoir une vue globale de ce que veulent les utilisateurs dans le futur.

Phase publique

Après 90 jours en version bêta publique, YellowBox deviendra publique :

Le statut sera mis à jour sur public et la balise bêta sera supprimée.

Ajout au programme de partenariat permettant de bénéficier d'avantages marketing et d'assistance.

5. Bilan du stage

5.1 Bilan des compétences

- C1 : Réaliser un développement d'application ;
- C2 : Optimiser des applications ;
- C3 : Administrer des systèmes informatiques communicants complexes ;
- C4 : Gérer des données de l'information ;
- C5 : Conduire un projet ;
- C6 : Collaborer au sein d'une équipe informatique

Durant mon stage je pense dégager 3 compétences que j'ai eu à vraiment à approfondir et cela coïncide bien avec mon parcours "Réalisation d'applications : conception, développement, validation" qui s'axe aussi sur ces 3 compétences qui sont : C1, C2 et C6 :

Compétence 1 : Réaliser un développement d'application

J'ai eu à développer une intégration en manipulant 2 principaux langages que sont Java et JavaScript, ce sont des langages que j'ai déjà eu à manipuler durant des projets ou des travaux pratiques à l'IUT.

Je n'ai donc pas eu à apprendre un nouveau langage, néanmoins je me suis familiariser avec le JavaScript avec certes des choses d'une complexité pas excessive mais que je n'arrivais pas à traiter durant les heures à l'IUT.

De plus, développer dans une nouvelle entreprise, c'est aussi s'adapter à une architecture existante avec des pratiques qui lui sont propres.

Bien que comme expliqué précédemment j'ai travaillé sur la partie plugin qui est un peu détaché de l'application principale, c'est tout de même un environnement différent à s'approprier, ce qui m'a permis de sortir un peu de ma zone de confort à l'IUT

Ce stage m'a donc permis de conforter cela et de me rassurer quant à ma capacité à produire un code fonctionnel et de qualité.

Compétence 2 : Optimiser des applications

Ma mission de stage est parti d'un besoin concret puisque comme expliqué précédemment, être présent sur Zapier peut-être un vrai atout.

Partant de ce besoin là et de la prise en main de YellowBox et Zapier, j'ai eu à concevoir une solution capable de répondre au mieux à ce besoin avec bien entendu l'aide mon équipe pour bien m'orienter fonctionnellement.

C’est un exercice auquel j’ai déjà eu à faire face de nombreuses fois auparavant puisqu’avant tout développement, il y a une phase de conception qui est nécessaire afin de produire quelque chose qui répond à un besoin précis. C’est quelque chose que l’on apprend dans différents projets et que j’ai donc aussi conforter durant ce stage avec un côté plus professionnel. Notamment grâce aux réunions avec le responsable produit ou encore avec le responsable commercial, on sens vraiment que l’on parle de quelque chose de concret qui lie beaucoup de thèmes différent

Compétence 3 : Administrer des systèmes informatiques communicants complexes

Malgré qu’elle ne figure pas dans les compétences que j’ai le plus approfondi, j’ai tout de même pu retrouver un peu ce que j’ai pu voir à l'IUT.

On peut notamment prendre le cas de la virtualisation qui est une des ressources vu durant cette deuxième année et que j’ai vu être utilisée en entreprise.

En effet, tous les environnements YellowBox fonctionnent sur du docker, ce qui permet de maintenir bien plus facilement les différentes versions des utilisateurs, personnellement je n’ai pas eu à apporter sur cela mais j’avais quand même des environnements qui m'étaient dédiés afin d’accomplir ma mission.

De plus, j’ai eu à parler d’architecture durant mon rapport, c’est quelque chose qui peut aussi rentrer dans cette compétence puisque comme expliqué dans la partie 4.2.2 j’ai eu à améliorer un web service existant.

Compétence 4 : Gérer des données de l’information

Les données sont une partie importante de ma mission puisque qu’elles sont amenées à être manipulées entre tous les échanges entre Zapier et YellowBox.

Certes, c’était un peu différent de ce que j’ai eu à réaliser à l’IUT puisque c’est plus axé sur des bases de données classiques avec du SQL notamment. Ici j’étais amené à manipuler les données à travers une API déjà établie et utilisée par énormément de parties différentes dans l’application.

Quand je parle de manipulation, sur ma mission j’étais plus sur de l’exploitation puisque je n’ ai pas eu à concevoir une base de données ou en modifier avec des relations déjà existantes…

La structure est donc restée la même, c’est seulement sur l’utilisation de celles-ci que j’ai eu à intervenir.

Compétence 5 : Conduire un projet

Comme expliqué, j'ai intégré une équipe qui travaille sur un seul et même produit, j'ai donc eu à m'intégrer dans tout leur suivi quotidien fait grâce à l'agilité ou aux différentes réunions. J'ai donc pris part à un projet collectif en y apportant mon projet personnel qu'est ma mission avec Zapier.

Tout comme l'équipe qui effectue le suivi des différentes fonctionnalités, tests, bug à travers AzureDevOps et Teams j'ai eu à le faire à mon échelle?

Mon travail était disponible sur un dépôt Git, mes missions étaient recensées dans les tâches à réaliser afin de ne pas perdre le fil et formaliser les besoins.

J'ai aussi comme expliqué dans la partie 4.2 à faire des choix et notamment sur la faisabilité des différentes choses réalisées, ce qui fait partie de la conduite d'un projet

Compétence 6 : Collaborer au sein d'une équipe informatique

Je pense que quand on intègre une entreprise, on est forcément amené à développer cette compétence et surtout lorsque c'est une première.

J'ai déjà eu à travailler en collaboration avec d'autres étudiants via des outils de communication tels que discord, des outils de travail en groupe tels que Github…

On retrouve certaines similitudes avec bien évidemment des outils différents et tout comme l'architecture d'une application à prendre en main, une manière de fonctionner différente avec des rôles précis et des statuts.

J'ai donc été amené à collaborer avec différentes personnes que ce soit le responsable produit, le responsable commercial ou encore le responsable de développement. Ce sont des postes bien différents mais il est important de communiquer afin d'avancer dans le même sens, d'autant plus lorsqu'on est stagiaire et que l'on a pas beaucoup d'expérience.

Je pense avoir réussi à bien mettre à dispositions mes aptitudes et à situer mon rôle dans l'équipe, cela m'a permis à ma mission de bien avancer et de progresser dans le travail en équipe.

5.2 Bilan humain

Si je dois dresser un bilan humain, je pense vraiment être tombée une super équipe.

Dès le premier jour, toutes mes appréhensions / craintes ont disparu de part l'accueil et la manière dont j'ai été intégré directement dans l'équipe.

Je n'ai vraiment pas eu l'impression d'être "le stagiaire" avec l"équipe puisque j'ai eu à participer à tous les événement collectifs, que ce soit les réunions quotidiennes, les réunions avec les équipes à Lyon ou encore les moments collectifs le matin ou le midi.

Ce stage dans une équipe informatique était une première pour moi et il ne m'a laissé aucun doute sur ce que j'étais capable de faire, au contraire il m'a rassuré sur ce que je suis et ce que j'avais eu à apprendre durant ma formation. Je tiens encore à remercier toute l'équipe pour cette expérience qui me sera très utile j'en suis sur.

5.3 Bilan organisationnel

5.4 Perspectives

6. Bibliographie

Enumération des ressources documentaires utilisées au cours du stage (il peut s'agir de livres, de cours, de tutoriels trouvés sur internet, etc.)

Info sur crm :
lafinancepourtous.com
custup.com
blogdumoderateur.com
sumup.com

https://datascientest.com/azure-devops-definitions-methodes-devops

7. Glossaire

Termes	Définitions
Termes important d'un CRM	
Prospect	Un des principaux éléments du CRM contenant des informations (numéro de téléphone, e-mail, site web) d'une personne ou

	société potentiellement capable de devenir un client, c'est-à-dire de finaliser une transaction
Lead	Un lead est un contact qui a manifesté de l'intérêt pour le produit et pouvant potentiellement se transformer en client. Même si les notions de lead et de prospect sont très proches, elles ne doivent pas être confondues : un lead est un contact basique, très peu qualifié, par opposition au prospect qui est un contact plus mûr et sur lequel l'entreprise dispose de davantage d'informations. En français, lead peut être traduit par suspect. La gestion des suspects fait partie intégrante de la stratégie commerciale et marketing.
Actions	Une action représente une fraction du capital d'une entreprise. Si vous détenez des actions, vous êtes propriétaire d'une partie de la société. Cela vous donne des droits, notamment, celui de recevoir des dividendes, si la société en distribue, et d'influer sur la politique générale de l'entreprise lors des assemblées générales d'actionnaires
Appels d'offres	Une définition simple de l'appel d'offres désigne une procédure par laquelle un acheteur potentiel demande à plusieurs offreurs de lui faire une proposition commerciale.
Fiche	Une fiche généralise simplement ce que gère globalement un CRM, un contact est une fiche, un lead est une fiche, une société est une fiche… Toutes ces choses là qui sont enregistrées sur YellowBox sont des fiches
Termes techniques	
Webhook	Un webhook est un mécanisme utilisé pour envoyer des notifications ou des données en temps réel d'une application à une autre. Il s'agit d'une méthode de communication basée sur les requêtes HTTP, dans laquelle une application envoie une demande (généralement au format JSON) à une URL spécifique lorsqu'un événement se produit.

	Le fonctionnement d'un webhook est le suivant : lorsqu'un événement prédéfini se produit dans l'application source (par exemple, la création d'un nouvel article sur un blog), un message contenant des informations sur cet événement est envoyé à une URL cible spécifiée (le webhook). L'URL cible est généralement fournie par l'application cible qui souhaite recevoir les données. Le rôle de l'application cible consiste à écouter les requêtes entrantes sur cette URL spécifique et à traiter les données reçues en fonction de ses besoins. Les webhooks sont souvent utilisés pour automatiser des tâches, synchroniser des données entre différentes applications ou déclencher des actions spécifiques lorsqu'un événement se produit.
API REST	Une API REST est une interface qui permet aux applications de communiquer entre elles via le protocole HTTP en utilisant les verbes et les URLs pour effectuer des opérations sur des ressources de manière fiable et sans état. Elle est largement utilisée dans le développement d'applications web et mobiles pour fournir des services et des données aux clients de manière structurée et standardisée

8. Annexe

8.1 Documentation Zapier

LIEN VERS LA DOC DE DEV GLOBALE ZAPIER : https://zapier.github.io/zapier-platform/

Requis pour le dev :

- Editeur de code (VSCode par exemple)
- Node.js
- Cli de Zapier = `npm install -g zapier-platform-cli`

Architecture de l'application Zapier :

build	20/04/2023 12:24	Dossier de fichiers
creates	24/04/2023 10:20	Dossier de fichiers
node_modules	20/04/2023 12:18	Dossier de fichiers
test	20/04/2023 11:53	Dossier de fichiers
triggers	20/04/2023 11:53	Dossier de fichiers
.env	20/04/2023 11:53	Fichier ENV
.gitignore	20/04/2023 11:53	Document texte
.zapierapprc	20/04/2023 11:53	Fichier ZAPIERAPP...
authentication.js	21/04/2023 14:26	Fichier de JavaScri...
index.js	24/04/2023 10:21	Fichier de JavaScri...
package.json	24/04/2023 10:15	Fichier source JSON
package-lock.json	21/04/2023 09:30	Fichier source JSON

- **build** : Dossier où est généré le zip lors du "zapier push" avant d'être uploadé sur l'éditeur Zapier
- **creates** : Dossier qui contient les actions de l'application
- **node_modules** : Bibliothèque Node.js
- **test** : espace ou vous pouvez effectuer des tests
- **triggers** : Dossier qui contient les triggers (déclencheurs) de l'application
- **.env** : Fichier où l'on peut définir "USERNAME" et "PASSWORD" en dur
- **.zapierapprc** : Contient l'id de votre application
- **authentification.js** : Connexion à YellowBox
- **index.js** : Regroupe les appels aux actions, triggers, la version… toute le nécessaire qui s'exécute lors du démarrage de l'application
- **package.json** : Des informations importantes comme les numéros de version de node, zapier, npm… 50 versions maximum

- **package.json.lock** : Fichier package-lock.json dans Zapier est un fichier généré automatiquement par NPM qui enregistre la version exacte de toutes les dépendances de votre projet, y compris les sous-dépendances

Pour le code :

Trigger : Nécessaire d'avoir 2 méthodes (performSubscribe(), performUnsubscribe())
Action : Peut avoir getInputFields() qui permet de paramétrer les champs dynamiques
Methode perform() présente sur les deux

Dans le **module.exports** On retrouve le nom, la description de l'action / déclencheur.
La possibilité de masquer, de définir une clé…

Commandes intéressantes sur le CLI :

`zapier login` = Connecter son compte Zapier en ligne

`zapier push` = Permet de build et de pousser le code travaillé sur l'éditeur zapier en ligne
Peut se faire en 2 étapes avec `zapier build` et `zapier upload`

Ne pas oublier de sauvegarder et de changer le numéro de "version" dans le fichier "package.json", auquel cas la mise à jour du projet ne se fait pas

`zapier versions` = Permet de voir l'historique des versions de votre application

Version	Platform	Users	Deployment	Deprecation Date	Timestamp
1.0.18	14.0.0	0	non-production	null	2023-04-24T08:15:26+00:00
1.0.17	14.0.0	0	non-production	null	2023-04-24T08:07:42+00:00
1.0.16	14.0.0	0	non-production	null	2023-04-24T07:54:10+00:00

`zapier logs --type=console` = Permet de récupérer des logs dans le cli une fois que vous avez tester votre zap sur l'éditeur zapier.
Commande pour log : `z.console.log()`
Les logs avec une chaine de caractère en paramètre type : "test" ne s'affichent pas dans le cli, il faut lui passer une variable, un objet…

On peut faire `zapier help logs`

9. Résumé en Anglais

This document summarizes my internship at DIMO Software, a company that designs software to support companies in the various tasks they may have (Accounting, Treasury, CRM...). So I joined the CRM branch of the company working on software called YellowBox.

After a short time of adaptation where I discovered how the team worked, YellowBox... my mission was therefore transmitted to me through the intermediary of my tutor in charge of development and the product manager.

My goal is to integrate their application on an online platform called Zapier which is used to automate tasks with big apps like Gmail, Linkedin, Outlook...
Thanks to this it will be possible to associate for example the reception of an email on outlook with the creation of a contact on YellowBox.

In order to achieve this, I used languages that I knew like Java and JavaScript but also tools that I didn't know like Sonar and Jenkins.I think that this mission allowed me to consolidate both things that I knew but also to acquire skills that I did not have and it is very rewarding

At the time of writing this text, I have finished developing the integration on Zapier but it is not yet publicly available. Indeed, there is a publication phase which takes several weeks or even months for it to be complete and accessible.

Thanks to this I was able to intervene on a concrete need by interacting with different teams such as sales, the product manager...
I thank DIMO Sofwtare and the IUT of Bayonne for giving me this opportunity