

---

## Shayp, Tackling water loss

---

*Auteur :*

M. Saïkou Ahmadou BARRY

*Superviseur :*

Pr. Sébastien COMBEFIS

*Maître de stage :*

M. Grégoire DE  
HEMPTINNE





# Remerciements

Remerciements à l'équipe de SHAYP qui m'a accueillie dans leurs bureaux et à toute personne ayant contribué au bon déroulement de ce projet.

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 L'entreprise</b>	<b>2</b>
1.1 SHAYP . . . . .	2
1.2 Organisation de l'entreprise . . . . .	2
1.3 Rôle du stagiaire . . . . .	3
<b>2 Contexte du stage</b>	<b>4</b>
2.1 Pile logicielle . . . . .	4
2.1.1 Frontend . . . . .	4
2.1.2 Backend . . . . .	4
2.2 Cycle de développement . . . . .	5
2.3 Expression fonctionnelle du besoin . . . . .	5
2.3.1 Fonctions principales . . . . .	5
<b>3 Conception des fonctionnalités</b>	<b>6</b>
3.1 Autocomplete . . . . .	6
3.2 I18N . . . . .	7
3.2.1 React-intl . . . . .	9
3.3 OAuth . . . . .	11
3.3.1 La problématique . . . . .	11
3.3.2 OAuth Roles . . . . .	12
3.3.3 En tant que client - application . . . . .	12
3.3.4 En tant que serveur ressource - serveur d'autorisation . . . . .	15

Table des matières	iii
Conclusion	19



# Introduction

La deuxième année d'études du master en Ingénieur industriel s'achève par un stage en entreprise de six semaines. Ce stage a pour but d'enrichir la formation par le biais d'une expérience professionnelle. J'ai donc effectué un stage au sein de la société Shayp dont l'activité principale est le développement d'une plateforme de surveillance de l'eau et détection des fuites.

Ce rapport décrit le travail que j'ai effectué au sein de l'entreprise. Il débutera par une présentation de celle-ci. Nous verrons ensuite les différentes tâches qui m'ont été confiées suivi de la conception et de la présentation des implémentations réalisées avant d'aboutir à la partie dédiée à la conclusion de ce stage.

# Chapitre 1

## L'entreprise

### 1.1 SHAYP

Fondée en 2017, Shayp, start-up basée à Bruxelles, a développé un compteur d'eau intelligent offrant une surveillance en temps réel de la consommation d'eau et la détection de fuites d'eau. La solution s'appuie sur un écosystème composé d'un compteur intelligent pour surveiller la consommation, d'un algorithme de détection de fuites d'eau, d'une plateforme web et mobile offrant des informations sur la consommation d'eau et d'un système de messages SMS/e-mail en cas de fuite.

### 1.2 Organisation de l'entreprise

L'équipe de SHAYP se compose principalement des personnes suivantes :

- Alex McCormack : CEO
- Zineddine Wakrim : CTO
- Grégoire de Hemptinne : COO
- Ingrid Nolet : Communication Marketing Manager
- Dugagjin Lashi : Full-Stack Software Engineer
- Cécilia Masut : Head of Sales Partnerships

S'ajoute à cela, l'équipe des stagiaires dans laquelle je me situe :

- Alfonso Ciaran : Financial management intern
- Les coaches :



## **1.3 Rôle du stagiaire**

Mon rôle dans l'entreprise durant ces six semaines de stages fut de développer de nouvelles fonctionnalités et maintenir et améliorer l'existant pour l'application web utilisée par les clients.

# Chapitre 2

## Contexte du stage

### 2.1 Pile logicielle

#### 2.1.1 Frontend

Shayp utilise *UmiJS*, un framework chinois basé sur React pour entreprise. Il offre un grand assortiment de fonctionnalité par défaut.

Pour la partie UI, le choix s'est posé sur Ant Design. Ant Design est un framework qui fournit des composants prédéfinis. En bref, les composants sont des blocs de construction pour une application Web qui ont une logique autonome et n'ont pas besoin de dépendances externes. Ant.design nous offre donc un UI pour entreprise et un ensemble de composants React de haute qualité prêts à l'emploi qui forment ensemble un ensemble de ressources et d'outils de développement et de conception.

#### 2.1.2 Backend

Shayp utilise le langage Python et le framework Pyramid pour son API Rest. Pyramid est un framework web léger pour les applications Python. Il permet de mettre en place une application web de base et d'exécuter rapidement celle-ci.

Pour la base de données, la base de données NoSQL MongoDB a été choisie. Dans MongoDB, les données sont modélisées sous forme de document sous un style JSON. On ne parle plus de tables, ni d'enregistrements, mais de collections et de documents. Ce système de gestion de données nous évite ainsi de faire des jointures de tables, car toutes les informations propres à une certaine donnée sont stockées dans un même document et nous permet d'itérer plus facilement sur le schéma des données.

## 2.2 Cycle de développement



Pour ce projet, on a opté pour un cycle itératif. Tout commence par l'expression du besoin : les fonctionnalités voulues, tout en sachant que ses besoins peuvent être modifiés au courant du processus pendant le cycle. Ensuite, on se lance dans le processus itératif en lui-même avec la mise en place du planning qui est suivie par les recherches, puis le développement et enfin la phase de test et de feedback. À l'issue de la phase de test, on passe au déploiement : le livrable qui a été validé est mis en production. Après cela vient la maintenance et la résolution des possibles bogues découverts.

L'avantage qu'on a avec ce cycle est que chaque itération permet de s'adapter à ce qui a été appris dans les itérations précédentes et le projet fini peut varier du besoin qui a été exprimé à l'origine.

## 2.3 Expression fonctionnelle du besoin

### 2.3.1 Fonctions principales

3 fonctionnalités majeures devaient être implémentées :

- Autocomplete : Créer un champ de saisie automatique pour les adresses
- I18n : Traduction et mise à disposition de la plateforme web en français, néerlandais et anglais
- OAuth 2.0 : Intégrer le protocole d'autorisation d'accès OAuth 2.0 dans la plateforme.

# Chapitre 3

## Conception des fonctionnalités

### 3.1 Autocomplete

L'objectif de cette fonctionnalité est d'offrir aux utilisateurs un champ d'entrée intelligent pour les adresses. C'est une fonctionnalité assez simple pour me familiariser avec le code source de la plateforme.

Pour offrir une auto complétion pour les adresses, nous avons 3 services disponibles :

- **Google Maps**

Un service de cartographie en ligne. Il offre une API pour la suggestion d'adresses. Nous n'utilisons pas leur service à la suite d'une forte hausse de leur tarification.

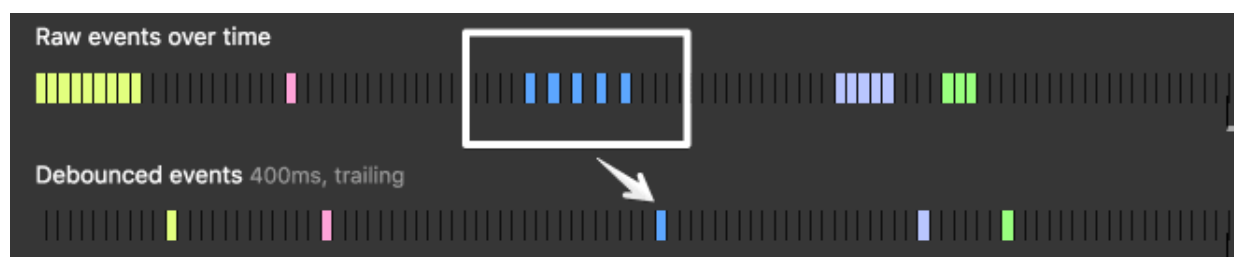
- **Mapbox**

Un service de cartographie en ligne. Il offre tout comme Google une API pour la suggestion d'adresse payant aussi, mais au-delà d'une certaine utilisation. Nous n'utilisons pas leur service à cause de leur condition d'utilisation qui ne nous autorise pas l'utilisation de leur service pour notre cas d'utilisation, localiser nos utilisateurs et enregistrer cette information dans notre base de données.

- **Algolia Places**

Algolia Places est un géocodeur offrant un moyen rapide, distribué et facile d'utiliser une recherche d'adresse automatiquement complétée sur un site web. Il est basé sur OpenStreetMap. Algolia Places a un niveau gratuit de 100 000 recherches par mois. Nous optons donc pour ce choix-là.

Chaque fois que l'utilisateur tape une nouvelle lettre, nous devons faire une requête pour mettre à jour l'autosuggestion dans la boîte de recherche. Mais que faire si l'utilisateur tape très vite ? Cela fait beaucoup de requêtes lancées en peu de temps, et il n'est pas nécessaire d'afficher la suggestion "apple" pour "a" si vous avez déjà tapé "atomic" avant que la requête ne revienne par exemple. Une façon de minimiser le nombre de requêtes rapides qu'un élément d'interface utilisateur fait est de débobiner (*debounce*) la fonction.



On enveloppe la fonction d'autosuggestion dans une autre fonction (la fonction `debounce`) que nous appelons chaque fois que l'utilisateur tape un nouveau caractère. Avant de faire la requête d'autosuggestion, la fonction `debounce` attend quelques centaines de millisecondes pour voir si d'autres appels arrivent. Si ce n'est pas le cas, `debounce` lance la requête d'autosuggestion et la boîte de recherche est mise à jour. Mais si de nouveaux appels sont reçus dans ce délai, il se débarrasse des anciens appels et recommence à attendre.

Avec cette solution, nous ne pouvons faire qu'une seule requête dont nous avons réellement besoin ! Cette approche n'élimine pas toutes les requêtes inutiles ; si l'utilisateur prend plus que le nombre de ms choisi pour taper le caractère suivant, nous ferons quand même une requête supplémentaire. Mais ce n'est peut-être pas la mauvaise chose à faire. Peut-être ont-ils fait une pause parce qu'ils n'arrivaient pas à trouver l'orthographe du terme dont ils avaient besoin.

## 3.2 I18N

Shayp étant une société proposant ses services sur plusieurs pays de l'Europe, l'application se doit d'être disponible dans plusieurs langues. Proposer une application dans plusieurs langues c'est l'internationaliser. L'internationalisation ou `i18n` consiste à préparer son adaptation à des langues différentes. Le but est donc de produire un programme qui peut être immédiatement déployé dans différentes langues en ajoutant simplement un nouveau fichier de traduction.

React étant écrit en Javascript, j'ai cherché les bibliothèques qui permettent de traduire et maintenir facilement une application internationale. Il y a 3 bibliothèques qui se sont démarquées :

- `Polyglot.js` : Le framework créé par Airbnb qui a leur plateforme traduite dans 30 langues différentes.
- `I18next` : Un framework créé par un groupe de développeurs.
- `FormatJS` : Le framework développé et maintenu par Yahoo.

J'ai fait une comparaison basée sur un certain nombre de fonctionnalités et aspects importants.

## Gestion de la pluralisation et de l'interpolation de phrases

Les trois frameworks gèrent la pluralisation appropriée et peuvent interpoler le contenu variable en traductions. Alors que polyglot et i18next utilisent un formatage similaire - formatjs utilise le format de message ICU (International Components for Unicode). C'est une question de préférences ici, car tous les trois obtiennent de bons résultats.

### Environnements / frameworks supportés

Il est préférable si le framework choisi est directement supporté par la librairie.

- Polyglot : js, node.js.

Il faut directement appeler la librairie. Il n'y a pas d'intégration officielle avec React.

- I18next : js, react-i18next, ng-i18next, vue-i18next...

i18next a une intégration avec énormément de frameworks web.

- FormatJS : js, react-intl, ember-intl,...

Tout comme I18next, formatjs a une intégration avec plusieurs frameworks web.

Pour ce point-là, formatjs et i18next sortent du lot.

### Prise en charge du formatage (date, nombres...)

Formatjs est livré avec un formatage intégré utilisant l'API intl du navigateur.

I18next est livré avec une fonction de formateur personnalisée que les développeurs peuvent définir pour gérer les formats (en utilisant l'API intl ou moment.js selon les besoins des développeurs).

Polyglot, vous devrez formater le contenu avant l'interpolation - pas de support intégré.

Le grand gagnant est donc FormatJS.

### Comment les traductions sont-elles chargées ?

Formatjs et polyglot laissent cela aux développeurs. Ils auront besoin d'écrire du code pour la détection de la langue et pour charger ou regrouper les traductions dans leur application.

I18next est livré avec plusieurs plugins qui permettent de charger des traductions à partir du serveur, du système de fichiers ou de regrouper des traductions avec webpack. De plus, il existe des plugins pour détecter la langue de l'utilisateur.

Le grand gagnant est donc I18next.

## Comment extraire les traductions de notre code ?

Polyglot et Formatjs laissent cela aux développeurs. Pour React-intl (formatjs) il y a un plugin d'extraction qui existe et extrait la traduction de la base de code.

Pour i18next, un plugin d'extraction existe.

Dans notre cas vu que nous utilisons React, on a deux possibilités alors React-intl et i18next.

## Le choix

React-intl (FormatJS) et I18next ont chacun leurs avantages et désavantages. Pour départager les deux, j'ai regardé celui qui s'intégrait le plus facilement avec UmiJS, le framework basé sur React que nous utilisons. Il se fait qu'UmiJS vient avec React-intl par défaut. J'avais donc besoin de ne faire aucune installation et configuration supplémentaire.

### 3.2.1 React-intl

Cette librairie fournit des composants React et une API pour formater les dates, les nombres et les chaînes de caractères, y compris la pluralisation et la gestion des traductions. React Intl utilise et s'appuie sur l'API d'internationalisation intégrée à JavaScript. Plus précisément, l'API intégrée est utilisée pour formater les dates/heures et les nombres dans React Intl. React Intl enveloppe ces API de manière cohérente, ce qui les rend plus faciles à utiliser et plus performantes.

J'ai dû mettre à jour tout le code source afin d'extraire les phrases à traduire et faire appel aux fonctions de React-intl.

```
function Example() {  
  const name = "Saikou";  
  return <p>{name}, welcome to umi's world</p>;  
}
```

FIGURE 3.1 – Avant la mise en place de React-intl

```
import intl from "umi-plugin-locale";

function Example() {
  const name = "Saikou";
  return (
    <p>
      {intl.formatMessage(
        { id: "welcome", defaultMessage: "Welcome to umi's world" },
        { name }
      )}
    </p>
  );
}
```

FIGURE 3.2 – Après la mise en place de React-intl

Pour chaque langue, on a un fichier de traduction **[lang].json** contenant l'ensemble des phrases de notre application sous forme de id et messages.

```
// en-US.json
{
  welcome: "{name}, welcome to Umi's world",
}

// fr-FR.json
{
  welcome: "{name}, bienvenue dans le monde d'Umi",
}

// nl.json
{
  welcome: "{name}, welkom in de wereld van Umi",
}
```

FIGURE 3.3 – Exemple de fichier de traduction

Par la suite, il fallait mettre en place une gestion simple des fichiers de traductions et une extraction automatique des phrases depuis le code source. Pour se faire j'ai écrit un script js qui va :

- Compiler notre projet UmiJS
- Extraire les traductions du code source



Le script extrait que les phrases qui sont appelées depuis une fonction `React-intl` qui respecte la signature `intl.formatMessage` (voir figure 3.2). Il n'extrait pas les phrases écrites dans de simples variables ou les phrases données en tant que paramètre à une fonction ou un component `React`.

- Créer des fichiers temporaires contenant toutes les traductions extraites.
- Met à jour les fichiers de traductions en fusionnant ceux-ci avec les fichiers temporaires.
- S'il y a des duplicata en ce qui concerne les traductions, il supprime les clés dupliquées et met à jour le code source
- Il formate le code source suivant nos directives.

```
> npm run translate

✓ Compiling Shayp project
✓ Extracting translations
✓ Generating translation files
✓ Updating translation files
✓ Updating Shayp code
✓ Formatting files
All duplicated keys have been changed to one of the following keys:
  - p.userInstallation.connectivity.step2
Please choose a more representative general name such as common.name or common.example
```

FIGURE 3.4 – Exemple d'appel du script d'extraction

L'application est depuis disponible dans 4 langues : français, néerlandais, anglais et allemand. Si nous voulons ajouter une nouvelle langue, il suffit de modifier le script d'extraction pour qu'il génère aussi les fichiers pour la langue en question. En terme d'amélioration, plutôt que d'hardcoder dans le script les langues voulues, on pourrait les donner en entrée au script au lancement.

## 3.3 OAuth

### 3.3.1 La problématique

Pouvoir donner un accès sécurisé et sous certaines conditions de notre API à un service tiers dans d'autres cas par exemple une assurance. Plusieurs options s'offrent à nous pour résoudre ce problème. Implémenter le protocole oauth 2.0 nous-mêmes ou utiliser un service tiers.

### 3.3.2 OAuth Roles

OAuth définit quatre rôles :

- Propriétaire de la ressource
- Client
- Serveur de ressources
- Serveur d'autorisation

Nous détaillerons chaque rôle dans les sous-sections suivantes.

#### Propriétaire de la ressource : Utilisateur

Le propriétaire de la ressource est l'utilisateur qui autorise une application à accéder à son compte. L'accès de l'application au compte de l'utilisateur est limité à la "portée" de l'autorisation accordée (par ex. accès en lecture ou en écriture).

#### Ressource / Serveur d'autorisation : API

Le serveur de ressources héberge les comptes utilisateurs protégés et le serveur d'autorisation vérifie l'identité de l'utilisateur puis émet des jetons d'accès à l'application.

#### Client : Application

Le client est l'application qui veut accéder au compte de l'utilisateur. Avant de pouvoir le faire, il doit être autorisé par l'utilisateur et l'autorisation doit être validée par l'API.

À Shayp, on doit agir en tant que Client (Application) si nous voulons intégrer Shayp avec un service tiers et on doit agir en tant que serveur de ressource / serveur d'autorisation si des applications tierces veulent interagir avec notre plateforme.

### 3.3.3 En tant que client - application

Avant d'utiliser OAuth avec notre application, nous devons enregistrer notre application auprès du service en question. Chaque service a son procédé, mais généralement il faut fournir les informations suivantes (et probablement des détails sur la demande) :

- Nom de l'application
- Site Web de l'application
- Rediriger l'URI ou l'URL de rappel

La redirection URI est l'endroit où le service redirigera l'utilisateur après avoir autorisé (ou refusé) notre application, et donc la partie de notre application qui traitera les codes d'autorisation ou les clés d'accès.

Ensuite, nous pouvons intégrer oauth à notre application.

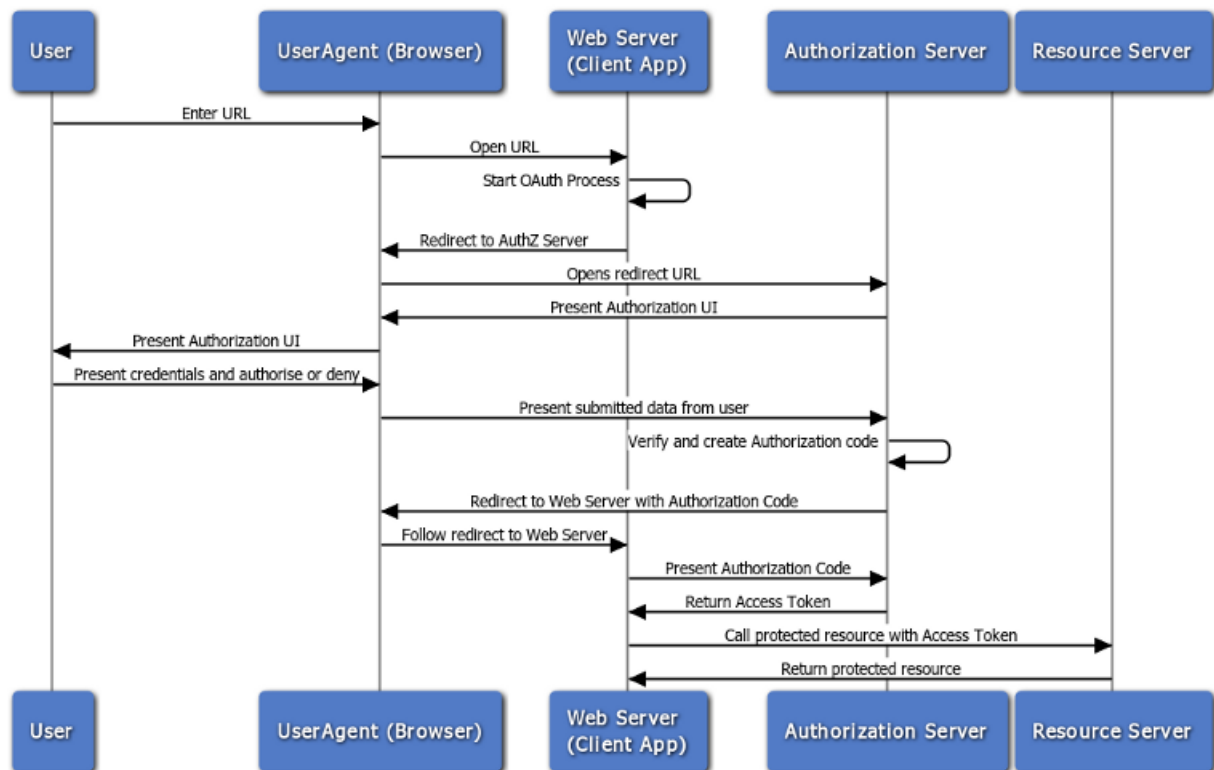


FIGURE 3.5 – Authorization Code Flow

**Authorization Code Flow** est le plus couramment utilisé parce qu'il est optimisé pour les applications côté serveur, où le code source n'est pas exposé publiquement et où la confidentialité *Client Secret* peut être maintenue. Il s'agit d'un flux basé sur la redirection, ce qui signifie que l'application doit être capable d'interagir avec l'agent utilisateur (c'est-à-dire le navigateur Web de l'utilisateur) et de recevoir les codes d'autorisation API qui sont acheminés par l'agent utilisateur.

Pour le prototype, j'ai voulu intégrer Shayp avec la plateforme EnergieID, une plateforme permettant de mesurer et surveiller l'énergie, l'eau et le transport, individuellement et collectivement.

## Prototype

L'intégration était assez simple, j'ai fait une demande auprès de leur plateforme et ils m'ont demandé les informations citées ci-dessus. Ensuite, il suffisait que je crée une nouvelle route (`/oauth/energyid/callback`) sur l'application React pour recevoir le code d'autorisation.

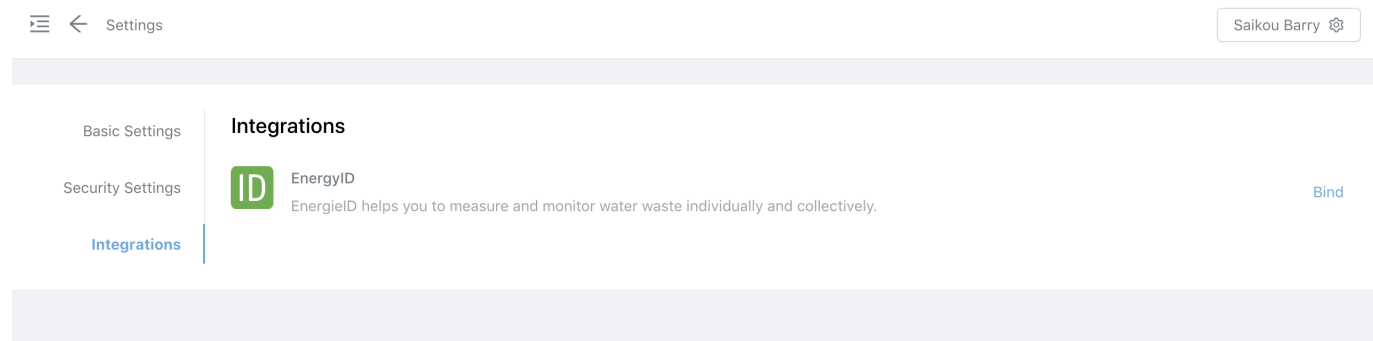


FIGURE 3.6 – Page d'intégration dans l'application

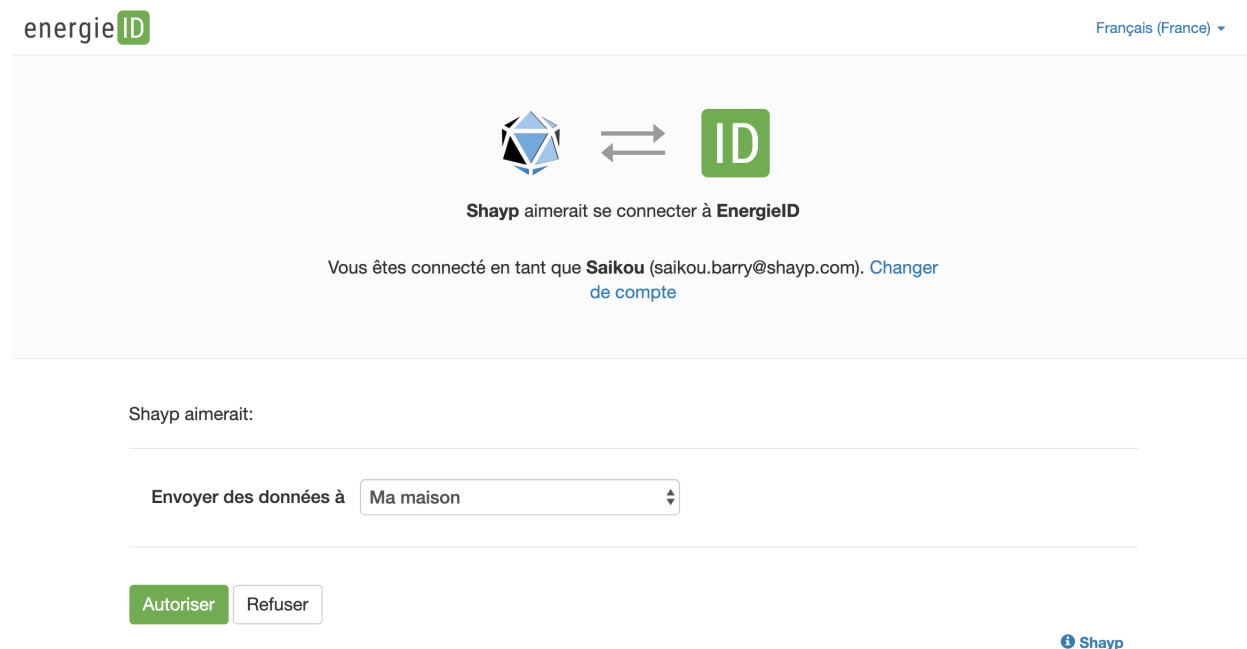


FIGURE 3.7 – Demande d'autorisation pour l'intégration

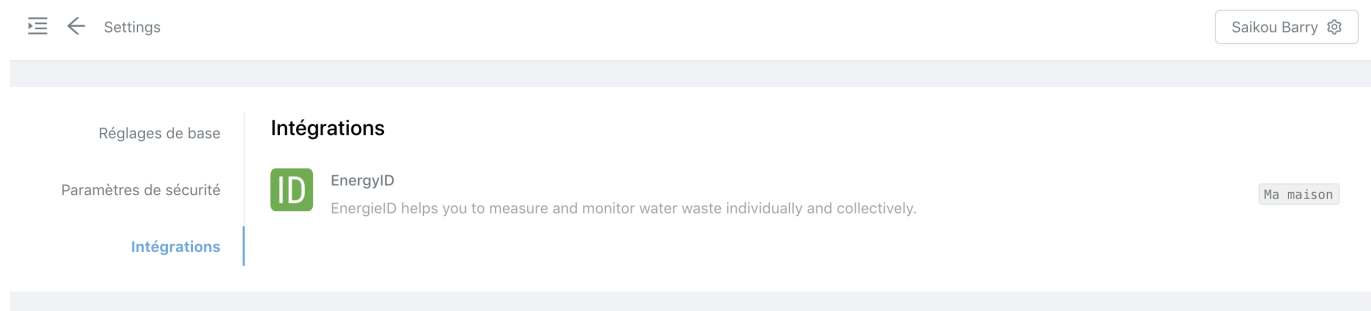


FIGURE 3.8 – Intégration acceptée

En ce qui concerne le backend, il fallait créer un nouveau modèle pour la base de données. Ce modèle reprend les informations importantes de l'intégration avec une plateforme pour un utilisateur donné.

```
class Integration(Document):
    """This model represents an integration that a user has made with a third-party application.
    """

    created: datetime = DateTimeField(required=True, default=datetime.utcnow)
    user_id: str = StringField(required=True)
    info: Optional[Dict[str, str]] = DictField(required=True)
    integration_type: str = StringField(default="energyid", choices=("energyid",),
    required=True)
```

FIGURE 3.9 – Modèle Integration pour la base de données

Nous pouvons ensuite planifier une tâche qui enverra les données vers leur service pour chaque utilisateur qui aura une intégration dans notre base de données.

### 3.3.4 En tant que serveur ressource - serveur d'autorisation

Certaines applications aimeraient pouvoir accéder directement à l'API de Shayp, nous créons donc une communication machine-to-machine. Avec les applications machine-to-machine (M2M), le système authentifie et autorise l'application plutôt que l'utilisateur. Pour ce scénario, les schémas d'authentification typiques tels que nom d'utilisateur + mot de passe ou logins sociaux n'ont pas de sens. Au lieu de cela, les applications M2M utilisent le **Client Credentials Flow**, dans lequel elles transmettent leur Client ID et leur Client secret pour s'authentifier et obtenir un token.

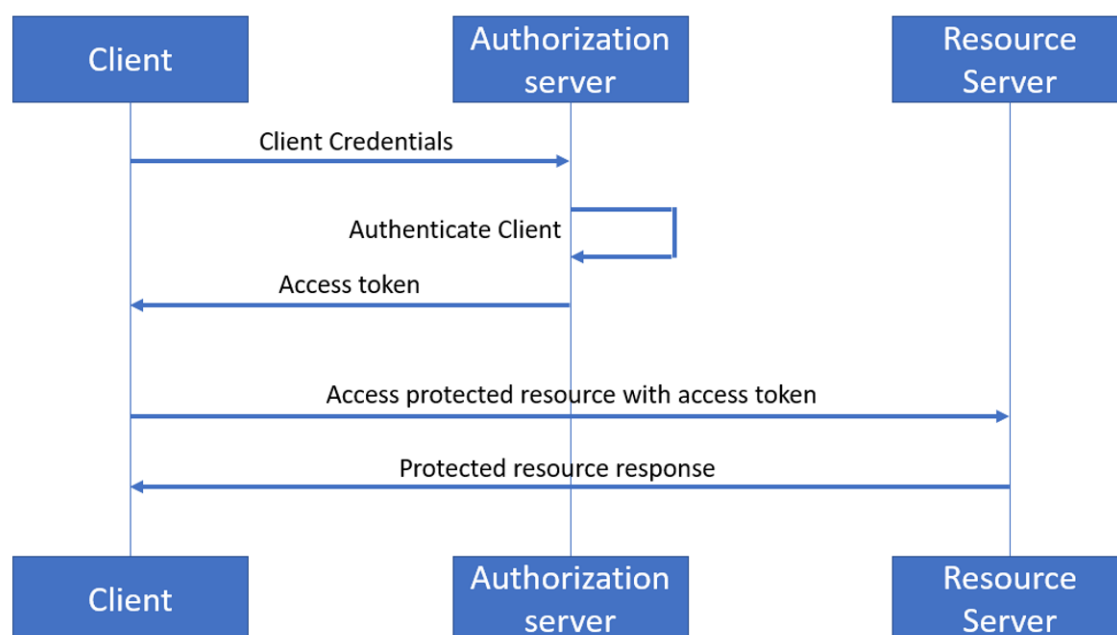


FIGURE 3.10 – Client Credentials Flow

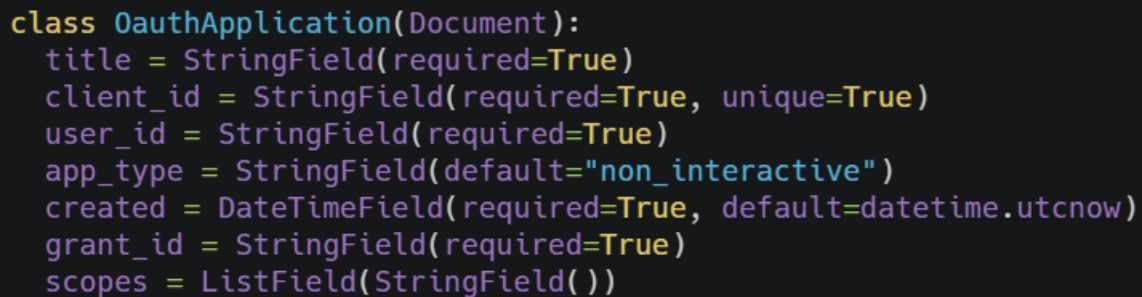
Pour implémenter le serveur d'autorisation, j'ai décidé d'utiliser un service tiers. Parmi les services tiers disponibles, 2 se distinguent, *AWS Cognito* et *Auth0*. Pour le prototype, j'ai choisi Auth0, car leur documentation est très bonne et leur prix est raisonnable pour notre cas d'utilisation.

## Prototype

Nous allons créer une API en Auth0 qui n'est qu'une représentation logique de notre propre API. Nous allons également créer toutes les permissions possibles pour notre API tel que *read :devices*, *update :devices* et *read :users*.

Chaque application machine aura son propre *client d'identification* et son propre *client secret*. Par exemple, si nous avons une compagnie d'assurance AG qui veut accéder à notre API, nous devons créer une application pour elle et lui envoyer son identifiant et son client secret. Pour une application, c'est gérable, mais si le nombre d'applications augmente, il est préférable de l'automatiser avec l'API de Auth0. Chaque application machine est liée à l'API créée ci-dessus et ses permissions sont également définies. Elle n'a pas nécessairement toutes les permissions définies dans l'API.

Afin d'éviter d'avoir à créer une application manuelle à chaque fois via le tableau de bord et de fournir l'identifiant client et le client secret à l'utilisateur, j'ai tout automatisé et créé un nouveau modèle dans la base de données.



```
class OAuthApplication(Document):
    title = StringField(required=True)
    client_id = StringField(required=True, unique=True)
    user_id = StringField(required=True)
    app_type = StringField(default="non_interactive")
    created = DateTimeField(required=True, default=datetime.utcnow)
    grant_id = StringField(required=True)
    scopes = ListField(StringField())
```

FIGURE 3.11 – Modèle OAuthApplication pour la base de données

Chaque utilisateur peut créer, modifier et supprimer son application. J'ai adopté cette approche parce que je crois qu'avoir une seule application peut être limité pour une compagnie comme une compagnie d'assurance. Ils peuvent vouloir accéder à notre API, mais à partir de plusieurs endroits différents. Il serait donc préférable qu'ils aient une application pour chaque endroit avec son ensemble de permissions activées. Par exemple, au point A, l'application ne peut lire que les informations de l'utilisateur et au point B, les informations de l'appareil. Il vous permet également de supprimer une application à tout moment si, à tout moment, elle est compromise.

On ne conserve pas le client secret pour des raisons de sécurité. Lorsque le client crée l'application, le client secret n'est affiché qu'une seule fois. C'est à lui de le mettre en lieu sûr. À tout moment, il peut en créer un nouveau.

Par la suite, les applications tierces peuvent communiquer avec notre API. Les routes qu'ils appelleront seront protégées par deux middlewares, *valid\_oauth\_token* et *requires\_scopes*. Ils vont respectivement vérifier si le token d'authentification est correct et valide et si le token contient bien les permissions nécessaires pour accéder à la ressource en question.

```
@resource(collection_path='/oauth/test', path='/oauth/test/{id}', validators=valid_oauth_token)
class TestOauth(object):

    @view(validators=(requires_scope), permission=('read:devices', 'create:devices',
    'delete:devices', 'update:devices'))
    def collection_get(self):
        return {
            "message": "all scope needed"
        }

    @view(validators=(requires_scope), permission=('create:devices',))
    def collection_post(self):
        return {
            "message": "method:post - create device"
        }

    ...
```

FIGURE 3.12 – Exemple de route au niveau de l'API protégé par Oauth



# Conclusion

Le stage fut une expérience incroyable. J'ai appris beaucoup de choses. Durant six semaines, j'ai travaillé sur plusieurs fonctionnalités qui m'ont ouvert les yeux sur plusieurs points importants pour la bonne mise en production d'un produit final.

Une très bonne gestion du planning est important afin de pouvoir livrer les fonctionnalités dans les délais.

Une très bonne gestion de l'environnement de développement aussi. Le but n'est pas seulement de livrer les fonctionnalités à la fin de mon stage mais d'avoir tout un environnement de développement structuré afin que l'équipe de développement puisse malgré mon départ continuer à développer l'application et la maintenir à jour. Pour cela, il a fallu que je détermine une procédure de tests et de développement ce qui a été nouveau pour moi car on n'apprend pas ces aspects à l'école où l'importance est juste de délivrer le projet demandé dans les temps.

J'ai aussi appris à utiliser un ensemble de nouveaux outils et frameworks (Gitlab, UmiJS, Ant Design), c'est donc un plus en terme d'expérience technique aussi.

Pour conclure, le stage s'est très bien passé et j'en suis très ravi du résultat obtenu. Je n'oublierais pas ces six semaines intenses et je remercie encore Shayp pour son accueil.

# Bibliographie

- [1] i18next : *Comparison to others*  
<https://www.i18next.com/overview/comparison-to-others/>
- [2] Jan Mühlemann : *i18n frameworks — the unfair showdown*  
<https://medium.com/@jamuhl/i18n-frameworks-the-unfair-showdown-8d436cd6f470>
- [3] atlassian : *i18n for React*  
<https://openedx.atlassian.net/wiki/spaces/LOC/pages/946503833/i18n+for+React>
- [4] atomic object : *Improve Your Autocomplete Timing with Debouncing*  
<https://spin.atomicobject.com/2018/06/04/autocomplete-timing-debouncing/>
- [5] digitalocean : *An Introduction to OAuth 2*  
<https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>
- [6] css-tricks : *Debouncing and Throttling Explained Through Examples*  
<https://css-tricks.com/debouncing-throttling-explained-examples/>  
<https://conversionxl.com/mobile-app-analytics/>
- [7] algolia places : *Turn any <input> into an address autocomplete*  
<https://community.algolia.com/places/>