

# La Semeuse de Mobile

*Document titre CDA*

*Version 1.0 • Mai 2021*

# Abréviations

<b>API</b>	<b>A</b> pplication <b>P</b> rogramming <b>I</b> nterface
<b>BDD</b>	<b>B</b> ase <b>D</b> e <b>D</b> onnées
<b>BtoB</b>	<b>B</b> usiness <b>to</b> <b>B</b> usiness
<b>BtoC</b>	<b>B</b> usiness <b>to</b> <b>C</b> ustomer
<b>HTTP</b>	<b>H</b> ypertext <b>T</b> ransfer <b>P</b> rotocol
<b>HTTP</b>	<b>H</b> yper <b>T</b> ext <b>T</b> ransfer <b>P</b> rotocol
<b>IDE</b>	<b>I</b> ntegrated <b>D</b> evelopment <b>E</b> nvironment
<b>JWT</b>	<b>J</b> son <b>W</b> eb <b>T</b> oken
<b>OS</b>	<b>O</b> perating <b>S</b> ystem
<b>TDD</b>	<b>T</b> est <b>D</b> riven <b>D</b> evelopment

# Table des matières

<b>1. PREAMBULE.....</b>	<b>1</b>
1.1. REMERCIEMENTS.....	1
1.2. ENTREPRISE.....	1
1.2.1. Les missions.....	2
1.3. ENGLISH PROJECT SYNTHESIS.....	4
<b>2. PROJET.....</b>	<b>5</b>
2.1. CONTEXTE.....	5
2.1.1. Hiboutik : Les clients (API n° 1) :.....	5
2.1.1. Wordpress : Le détail des pierres (API n° 2) :.....	5
2.1.1. L'interfaçage (API n° 3) :.....	5
2.1.2. L'application mobile :.....	6
2.2. COMPETENCES COUVERTES.....	6
2.3. ÉLABORATION DES TESTS.....	7
2.3.1. TDD.....	7
2.4. GESTION DE PROJET.....	7
2.4.1. Planning.....	7
2.4.1.1. La recherche.....	8
2.4.1.2. La documentation.....	8
2.4.1.3. Le développement.....	9
2.4.2. Versionning (GITLAB).....	9
2.4.3. Validations.....	10
2.5. SPECIFICATIONS TECHNIQUES.....	10
2.5.1. Objectif.....	10
2.5.2. Fonctionnalité.....	10
2.5.2.1. Module NodeJS ?.....	11
2.5.2.2. Cron ?.....	12
2.5.2.3. Pourquoi un planificateur de tâches ?.....	12
2.5.3. Étude et choix technologique.....	13
2.5.3.1. Backend.....	13
2.5.3.2. Frontend.....	14
2.5.3.3. BDD.....	14
2.5.3.4. IDE.....	14
2.5.3.5. Autre.....	14
2.5.4. Patron de conception.....	15
2.5.5. Base de données.....	16
2.5.5.1. User.....	16
2.5.5.2. Rock.....	17
2.5.5.3. Lithotherapie.....	18
2.5.5.4. Les IDs Hiboutik et Wordpress ?.....	19
2.5.6. Contraintes.....	19
2.5.6.1. Connexion.....	19
2.5.6.2. Mot de passe oublié.....	19
2.5.6.3. Inscription.....	19
2.5.6.4. Modifier profil.....	20
2.5.6.5. Modifier mot de passe.....	21
2.6. SECURITE.....	21
2.6.1. IOS Keychain.....	21
2.6.2. ANDROID Keystore.....	21

2.6.3.	JWT: .....	21
2.6.4.	Flutter secure storage .....	23
2.7.	REALISATION .....	24
2.7.1.	Parcours UX.....	24
2.7.1.1.	Global .....	24
2.7.1.2.	Connexion/Inscription .....	24
2.7.2.	Maquette .....	25
2.7.2.1.	Page d'accueil .....	25
2.7.2.2.	Page Favori et Achat.....	26
2.7.2.3.	Page description d'une fiche .....	27
2.7.2.4.	Page connexion .....	28
2.7.3.	Extrait de code .....	29
2.7.3.1.	Les appels de dépendances .....	30
2.7.3.2.	Les instances.....	30
2.7.3.3.	Les fonctions.....	32
2.7.3.4.	Le Cron.....	34
2.7.4.	Jeux d'essai (test unitaire, test intégration, TDD) .....	34
2.7.4.1.	Mocha.....	34
2.7.4.2.	Chai.....	35
2.7.4.3.	Test module Hiboutik .....	36
<b>3.</b>	<b>RECHERCHE.....</b>	<b>39</b>
3.1.	VEILLE TECHNOLOGIQUE.....	39
3.2.	SITUATION DE TRAVAIL.....	39
3.3.	CONTRAINTES ET POTENTIELLES EVOLUTIONS.....	41
3.3.1.	Contraintes.....	41
3.3.2.	Évolutions.....	41
<b>4.</b>	<b>CONCLUSION .....</b>	<b>42</b>

# 1. Préambule

---

## 1.1. Remerciements

Dans un premier temps, je tiens à remercier Aurélie FERRARI pour m'avoir accueilli au sein de la société « la Semeuse de Pierres » pour cette année de Bachelor en développement Web et application mobile. Merci de m'avoir fait confiance, et de m'avoir fait découvrir le domaine de la Lithothérapie. Beaucoup de procédés étaient à numériser et je suis fier d'avoir pu y contribuer.

Je tiens par la même occasion, à remercier mes formateurs Vincent BRAY et Pierre TELLIER-ROBERT pour leurs accompagnements, leur dévouement et leurs conseils des plus qualitatifs.

Pour finir, merci à toute l'équipe pédagogique de MyDigitalSchool pour son accompagnement durant cette année.

## 1.2. Entreprise

Fondée en 2020 par Aurélie FERRARI, seule fondatrice, la boutique de La Semeuse de Pierres est spécialisée dans le domaine de la lithothérapie avec des pierres naturelles. Située à Hérouville-Saint-Clair, le type de clientèle est BtoC. Elle s'est fait connaître en commençant dans des salons et en communiquant régulièrement sur les réseaux sociaux. Aujourd'hui, l'équipe est composée d'Aurélie FERRARI CEO et de Greg DILON apprenti développeur WEB.



*Devanture du magasin*

Beaucoup de clients ont passé la porte et se sont laissé séduire par le charme inéluctable des pierres présentes. Ces pierres sont toutes différentes, il en existe plus d'un millier, leurs compositions chimiques sont de sorte à donner des pierres uniques, apportant leurs lots de bien-être aux personnes qui en font l'acquisition. Elles existent aussi sous plusieurs formes :

- Forme libre
- Forme géométrique
- Pierre plate
- Pierre roulée
- Bijoux
- Etc...

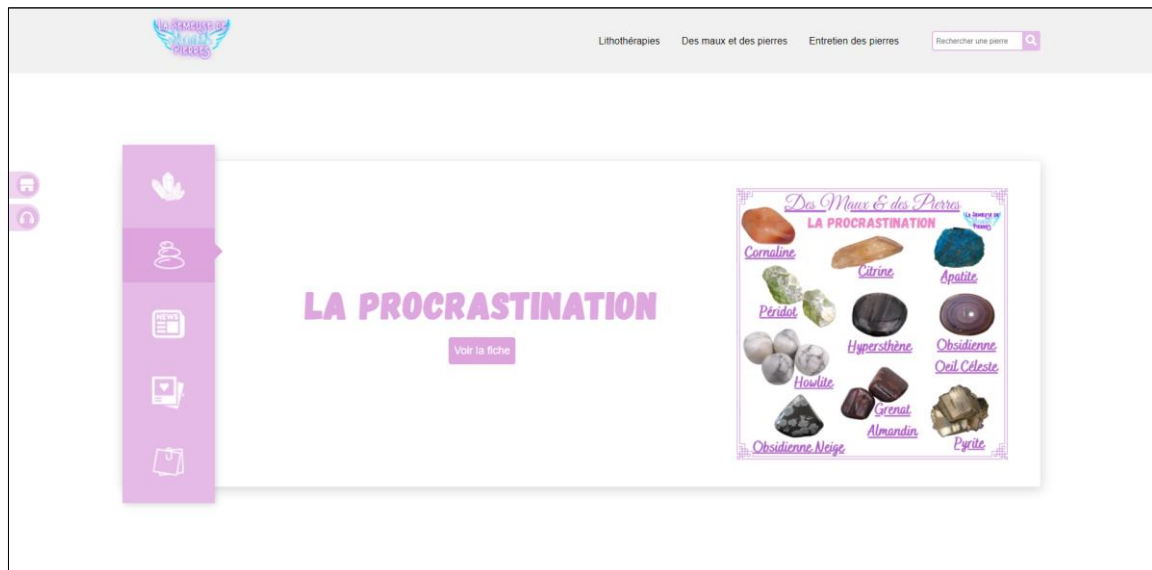
Toutes ces formes sont faites pour combler le maximum de personnes.

Durant la crise sanitaire de 2020, Aurélie FERRARI a pris conscience de l'ampleur de l'impact du numérique dans le monde professionnel. Elle cherche à gagner en visibilité grâce aux réseaux sociaux et aux sites WEB. C'est en septembre 2020 qu'elle m'a recruté en tant qu'alternant en développement WEB afin que je puisse l'accompagner dans ses démarches de numérisation.

### **1.2.1. Les missions**

Durant ma période d'alternance, j'ai été amené à travailler sur plusieurs points en rapport avec le numérique. Ma principale mission a été de concevoir un site WEB répertoriant toutes les Pierres avec leurs caractéristiques, avec les astuces pour les entretenir et les utiliser au maximum de leurs capacités.

Afin que le site puisse être maintenu sans compétences dans le développement WEB, nous avons opté pour Wordpress. En premier lieu, et pour qu'il sorte rapidement, le site était intégralement sous « Elementor ». Une fois le site mis en ligne et accessible, j'ai proposé une version 2 en créant un thème Wordpress à part entière en gardant la première version comme modèle tout en simplifiant au maximum l'espace d'administration.



[lasemeusedepierres.com](http://lasemeusedepierres.com)

Initialement, le site était un e-commerce, mais la Lithothérapie est un domaine physique, les clients ont besoin de voir et ressentir les pierres.

À la suite de cette révélation, il a été convenu de supprimer la partie boutique et d'améliorer le contenu et la présentation (passage de la version 1 à la version 2).

En parallèle j'étais chargé d'alimenter ses réseaux sociaux (Facebook, Instagram et Pinterest) à raison de deux publications par semaine. Dans un document Excel, je reçois les informations relatives à la publication que je mets en forme sur la plateforme Canva.



Exemple de design pour les publications designer sur Canva

La photo de gauche ci-dessus est une fiche Lithothérapie, elle a pour but de présenter précisément les pierres et leurs effets.

La photo de droite ci-dessus est une fiche des Maux et des Pierres, elle a pour but de présenter un groupe de pierre qui peut être combiné pour renforcer et répondre plus efficacement à une problématique. Dans le cas de cette image, nous avons un groupe de pierres qui aide à soulager de l'asthme.

En octobre 2020, le magasin a fait l'acquisition d'un outil de caisse du nom de Hiboutik. Nous l'avons configuré et fait l'inventaire afin de répertorier les pierres disponible en stock.

### **1.3. English project synthesis**

In February 2021, Aurélie came up with the idea of a mobile application that would list the details of each stone in the shop in a dedicated customer area.

Each customer can consult the list of stones and their details that they have purchased or bookmarked.

The aim of the application is to allow customers in the shop to know the information of the different stones or to establish a purchase list thanks to the favorites. It also allows customers to know the details of the stones they have purchased and to know everything about a stone quickly and easily.

The project is divided into two main parts:

A mobile application which is used as a graphical interface to the project. This is the part used by all registered users.

An API used as for data processing, rights, and user login. It can be accessed through a graphical interface or through tools that make HTTP requests such as POSTMAN

For the list of stones, La Semeuse de Pierres's website has a Wordpress site listing them, it provides an API to retrieve all the necessary information.

To know if a user has bought a stone, La Semeuse de Pierre has a cash register application called Hiboutik. On it, all customers are listed along with their purchases made in the shop.



## 2. Projet

---

### 2.1. Contexte

L'application « La semeuse de mobiles » est une application dédiée à la clientèle de « La Semeuse de Pierres ». Cette application mobile communique avec une API permettant la connexion et la gestion des utilisateurs, le listing des pierres. Cette API est en communication constante avec deux autres APIs. Ce sont ces API qui fournissent toutes les données utiles au projet. L'API principale sert d'interface à ses deux API qui sont Hiboutik-API et Wordpress-API.

#### 2.1.1. Hiboutik : Les clients (API n° 1) :

Hiboutik est un outil de caisse répertoriant tous les clients ainsi que leurs achats. Un client achète une ou plusieurs pierres en magasin, au moment du paiement, il renseigne ses coordonnées (nom, prénom, âge, adresse email, adresse postale, etc...). Ce client est maintenant enregistré dans l'outil de caisse numérique, lors d'un prochain achat, il est donc possible de consulter les anciens achats et en faire de nouveaux.

Une API est disponible et livrée par le constructeur d'Hiboutik, permettant d'avoir accès à la liste des clients ainsi que la liste des produits (des pierres).

#### 2.1.1. Wordpress : Le détail des pierres (API n° 2) :

Aujourd'hui, il existe un site WEB développé en interne sous WORDPRESS répertoriant toutes les pierres existantes avec leurs descriptions, leurs caractéristiques et leurs spécificités. Le CMS fournit de base une API donnant l'accès au contenu des pages, des articles ou des customs post type. C'est grâce à cela qu'il est possible de récupérer la liste des fiches lithothérapies.

#### 2.1.1. L'interfaçage (API n° 3) :

Développée sous NodeJS, cette API sert à associer et gérer les données provenant des deux précédents API pour les envoyer à de multiples interfaces graphiques par le biais de requêtes HTTP. Grâce à cela, il est possible d'avoir la même gestion et la même base de données pour une application WEB, une application mobile ou encore une application DESKTOP.

Cette API utilise une base de données NoSQL du nom de MongoDB afin de conserver les informations provenant des autres APIs.

### **2.1.2. L'application mobile :**

À travers une application mobile, un client peut avoir la possibilité de consulter les pierres qu'il a achetées, et de consulter ses détails. Il peut aussi consulter la liste de toutes les pierres présentes sur le site internet et les mettre en favori.

Un profil client peut être créé directement par l'utilisateur dans une inscription basique en entrant toutes les coordonnées ainsi qu'un mot de passe. Dans ce cas présent, il ne possèdera aucune pierre d'achetée ni aucun favori.

Un profil client peut aussi être créé lors de l'achat en magasin, les informations sont saisies sur l'application Hiboutik. À ce moment, si un utilisateur n'a pas validé son compte, un mail est envoyé sur l'adresse email du client lui proposant de télécharger l'application et d'activer son compte client et obtiendra par la suite un mot de passe provisoire. Durant la première connexion, l'utilisateur sera amené à modifier son mot de passe.

## **2.2. Compétences couvertes**

- ☒ Maquetter une application.
- ☐ Développer une interface utilisateur de type desktop.
- ☒ Développer des composants d'accès aux données.
- ☐ Développer la partie front end d'une interface utilisateur WEB.
- ☒ Développer la partie back end d'une interface utilisateur WEB.
- ☒ Concevoir une base de données.
- ☒ Mettre en place une base de données.
- ☒ Développer des composants dans le langage d'une base de données.
- ☐ Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement.
- ☒ Concevoir une application.
- ☒ Développer des composants métier.
- ☒ Construire une application organisée en couches.
- ☒ Développer une application mobile.
- ☒ Préparer et exécuter les plans de tests d'une application.
- ☒ Préparer et exécuter le déploiement d'une application.

## **2.3.Élaboration des tests**

Dans tout projet numérique, il est important de vérifier qu'un projet remplisse ses fonctions. L'élaboration des tests peut être effectuée sous plusieurs niveaux :

- Les tests unitaires afin de tester le bon déroulement de chaque fonction.
- Les tests de validations afin de tester si toutes les exigences sont bien remplies.
- Les tests d'intégrations afin de s'assurer de la mise en production de l'application.

### **2.3.1. TDD**

Le TDD est une manière d'organiser le développement d'application. Cela consiste à rédiger les tests avant le code applicatif. Cela permet d'éviter de développer des fonctionnalités ne pouvant être testées.

Grâce au TDD, c'est au code de s'adapter aux tests, en travaillant de manière itérative jusqu'à arriver à un développement finalisé, testé à 100%.

## **2.4.Gestion de projet**

### **2.4.1. Planning**

Réaliser un planning permet de connaître le cycle de vie d'un projet, celui-ci étant réalisé sous plusieurs phases (validation, réalisation, etc...).

Planifier un projet consiste à :

- Identifier les différentes tâches le constituant
- Hiérarchiser ces tâches
- Définir leur durée et leur échéance
- Déterminer les coûts et ressources humaines nécessaires à la réalisation du projet
- Planifier toutes les étapes de manière claire

Grâce à cela, il est possible de suivre avec précision un projet, de savoir s'il y a du retard ou de l'avance et de connaître l'état d'avancement de chaque tâche en pourcentage.

La semeuse d'API		Début du projet MAD Demandée : 24/08/2021		26/04/2021	
Développement Application Mobile					
Commentaires :					

Planning Gant réalisé sous Excel

Dans le cadre du projet. Le planning est divisé en 3 grandes étapes :

#### 2.4.1.1. La recherche

Cette étape est l'idée du projet. Durant cette période, il faut analyser les besoins et savoir si le projet est faisable.

Une étude technologique est effectuée afin de savoir quels langages de programmation correspondraient au mieux aux besoins du projet

#### 2.4.1.2. La documentation

La documentation est la phase d'initialisation du projet. Cela permet d'avoir une idée claire de l'orientation que le développement de l'application doit emprunter.

Le parcours UX sert à deviner les actions qu'un utilisateur pourrait effectuer pour arriver sur un résultat voulu. Cela permet de connaître les différentes fonctionnalités nécessaires.

La maquette est la partie graphique du projet. Cela permet d'avoir un aperçu visuel du front de l'application mobile et ce à quoi pourrait ressembler le produit final.

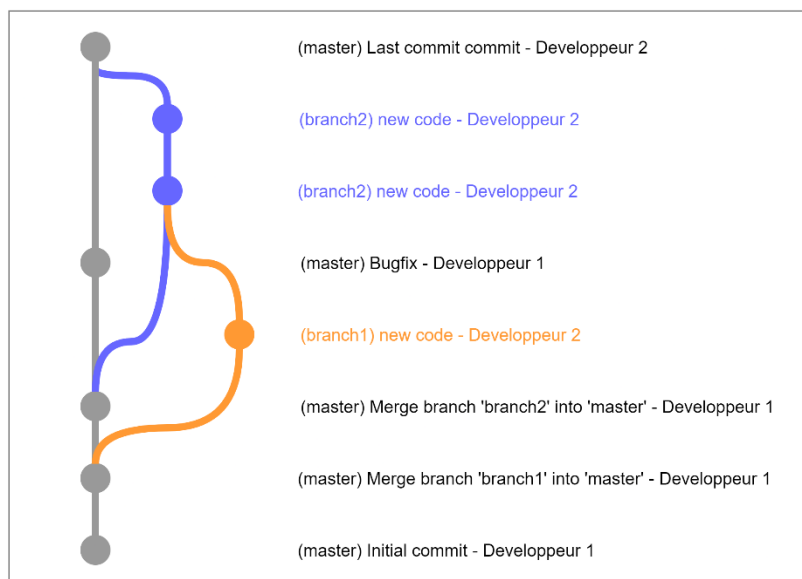
### 2.4.1.3. Le développement

Le développement est la réalisation du projet. Il doit être effectué en tenant compte des deux parties précédentes.

### 2.4.2. Versionning (GITLAB)

GITLAB est un outil de versionning, qui permet d'avoir un suivi précis d'un projet numérique quelle que soit sa nature. Durant le développement, il est possible de créer une fonctionnalité spécifique et le sauvegarder sur GITLAB avec un message précisant le but du morceau de code (commit). Cela permet en cas de problèmes de pouvoir remonter sur d'anciens commits et de pouvoir régler le problème en amont. Plus les commits sont fréquents et réguliers, plus l'historique est précis et plus il sera facile de revenir sur d'anciennes versions.

Cet outil permet aussi de gérer un projet en équipe. Il permet de travailler sur un système de branches ou chaque développeur se situe pour développer une fonctionnalité sans corrompre le code de ses collaborateurs.



*Exemple de projet GIT avec plusieurs développeurs*

Le projet « La semeuse de mobiles » est développé par un seul développeur, tout le code a été sauvegardé sur la branche principale (master).

### **2.4.3. Validations**

Les validations des étapes du projet sont gérées par le développeur du projet à l'exception de la partie visuelle qui doit être confirmée par Madame FERRARI afin que le produit corresponde au mieux à ses besoins et ses envies.

## **2.5. Spécifications techniques**

### **2.5.1. Objectif**

L'objectif du projet est de créer une API qui répertorie toutes les pierres présentes sur le site internet de la semeuse de pierre ainsi qu'une liste de clients existants (comptes utilisateurs) présente dans l'outil de caisse Hiboutik. Cette liste ne nécessite pas de connexion, elle peut être consultée à tout moment par toutes les personnes ayant téléchargé l'application.

Elle permet à un utilisateur à travers une interface mobile de se connecter par le biais d'une adresse email et d'un mot de passe. À ce moment-là, l'utilisateur a toujours la possibilité de consulter la liste complète des pierres, mais aussi de consulter les pierres qu'il a achetées (si le client a effectué un achat en magasin) ainsi que les pierres qu'il a mises en favoris.

### **2.5.2. Fonctionnalité**

Si un utilisateur est inexistant dans la base de données d'Hiboutik, il a la possibilité de créer un compte qui sera enregistré dans l'outil de caisse en ligne. Comme cela, quand le client effectuera un achat en magasin, celui-ci sera d'ores et déjà enregistré.

Après un achat, le client recevra un email contenant son identifiant et son mot de passe provisoire. Lors de la première connexion, le client reçoit un message lui conseillant de modifier son mot de passe avec un lien de redirection vers la modification du mot de passe.

Une fois l'utilisateur connecté, il a la possibilité de consulter ses achats et ses favoris. Il a aussi la possibilité de consulter son profil, de le modifier et de changer son mot de passe.

L'API communique avec deux modules NodeJS créés pour le projet. Ces modules permettent la connexion avec l'API de Wordpress et l'API d'Hiboutik. Toutes les données sont dupliquées

dans la base de données interne. Cela permet en cas de panne d'un des deux services d'avoir toujours accès aux informations et aux comptes utilisateurs.

La base de données interne est alimentée par les deux modules dits précédemment. Grâce à un système de Cron, tous les soirs à 2h du matin, heure à faible fréquentation numérique, l'API remet à jour toutes les données de la base de données avec les données des services. Cela permet d'ajouter les nouveaux utilisateurs et les nouvelles pierres enregistrés dans la journée.

Lors d'un oubli de mot de passe, un utilisateur a la possibilité de demander un nouveau mot de passe. Il saisit son adresse mail et reçoit un mail contenant un lien personnalisé de redirection vers une page WEB. L'utilisateur saisit son mot de passe deux fois et l'envoie au serveur. Suite à cela, il recevra un email de confirmation pour son changement de mot de passe.

#### 2.5.2.1. Module NodeJS ?

Node.js inclut la notion de module. Un module est un ensemble de fonctions et d'objets JavaScript qui peut être utilisé par des applications NodeJS externes. Tout fichier ou ensemble de fichiers Node.js peuvent être considérés comme un module si ses fonctions et ses données sont rendues utilisables par des programmes externes.

L'installation d'un module se fait à l'aide de NPM, gestionnaire de paquets officiel de Node.js. Grâce à lui, il est possible d'installer/désinstaller des modules avec une commande dans le terminal (npm install LE\_MODULE – save).

Une fois installé, il suffit d'indiquer au fichier que l'on veut pouvoir utiliser les fonctions du module par un « require » et utiliser ses fonctions.

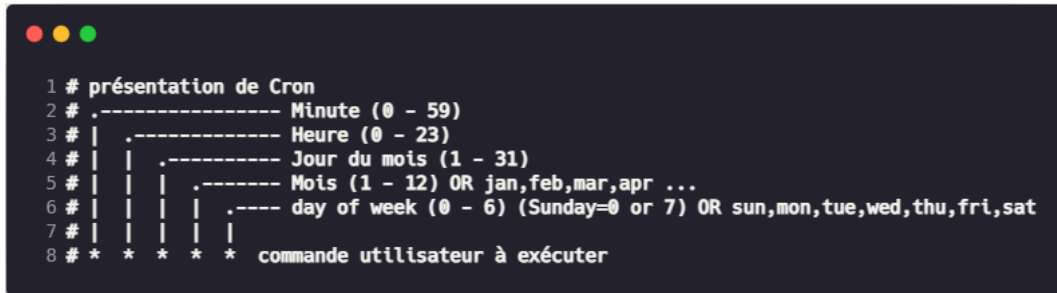


```
1 //commande : npm install module
2
3 const module = require('module');
4
5 module.fonctionModule();
```

*Exemple d'intégration d'un module*

### 2.5.2.2. Cron ?

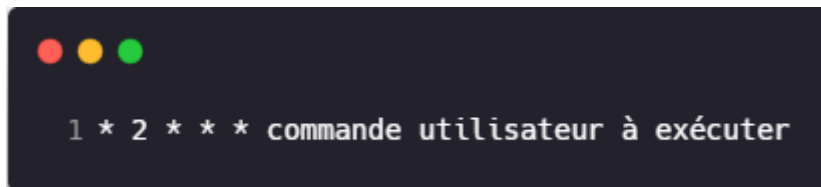
Cron ou Crontab est un outil de planification de tâches régulières. Cela permet de planifier des scripts tels que des scripts de sauvegarde ou de récupération de données sur un tableau Excel.



*Schéma d'utilisation de Cron*

Une commande CRON est divisée en 2 étapes :

1. La planification : Présenter sous 5 étoiles qui correspondent chacune à une tranche horaire/jour
2. La commande à exécuter



*Exemple de commande Cron pour exécuter un programme tous les jours à 2 heures du matin*

### 2.5.2.3. Pourquoi un planificateur de tâches ?

Les APIs Hiboutik et Wordpress ont été développés par leur constructeur respectif.

Dans le cas d'Hiboutik, pour connaître le nom d'une pierre, chaque article possède une marque. Cela permet d'éviter les doublons. Or, cette API ne permet pas de demander une seule marque, seulement la liste. C'est pourquoi il est préférable de stocker la liste des marques dans la base de données interne.



Même problématique chez Wordpress. Les fiches Lithothérapie sont des articles personnalisés nommés « Custom post Type ». Les CPT peuvent être visibles avec l'API, mais seulement sous forme d'une liste et non à l'unité.

Cette solution permet aussi d'effectuer des sauvegardes des APIs. Grâce à cela, en cas de panne d'un des deux services, l'application sera toujours accessible avec des données récentes.

### **2.5.3. Étude et choix technologique**

#### **2.5.3.1. Backend**

##### **NodeJS:**

NodeJS est une plateforme logicielle libre basée sur JavaScript qui permet d'utiliser le langage JavaScript initialement prévu pour le Front pour construire des logiciels cotés serveur.

Elle se distingue des autres plateformes grâce à une approche non bloquante permettant d'effectuer des tâches et des fonctions de manière asynchrone.

C'est un outil très populaire dans la création d'API, qui est simple d'utilisation et très facile à prendre en main.

##### **Dépendances principales :**

- Express : est « le » Framework NodeJS standard pour la construction d'applications WEB.
- Mongoose : est un Framework permettant l'accès à une base de données MongoDB (cf.

Flutter est basé sur le langage Dart, lui aussi développé par Google en 2011.

- BDD) qui supporte à la fois les promesses et les callbacks.
- Node-cron : est un Framework de planification de tâches basé sur Cron (cf.Cron ?)
- Mocha : est un Framework pour effectuer des tests en JavaScript

### **2.5.3.2. Frontend**

Flutter est un kit de développement de logiciel d'interface utilisateur open source créé par Google en 2011. Il permet de faire du développement cross-platform, sur une seule base de code pour des interfaces comme Android, IOS, etc...

Flutter est basé sur le langage Dart, lui aussi développé par Google en 2011.

### **2.5.3.3. BDD**

MongoDB est une base de données NoSQL et non relationnel développé par la société MongoDB en 2009. Les données sont stockées sous forme d'objet. Cette technologie se distingue des bases de données SQL/relationnelles par sa flexibilité et ses performances.

### **2.5.3.4. IDE**

Visual Studio Code est un IDE développée par Microsoft en 2015. Il supporte un large éventail de technologies telles que C++, PHP, JavaScript, Dart, etc.. ;

La force de cette IDE est sa communauté qui développe des fonctionnalités afin de simplifier ou d'automatiser certaines tâches et de rendre l'expérience meilleure.

### **2.5.3.5. Autre**

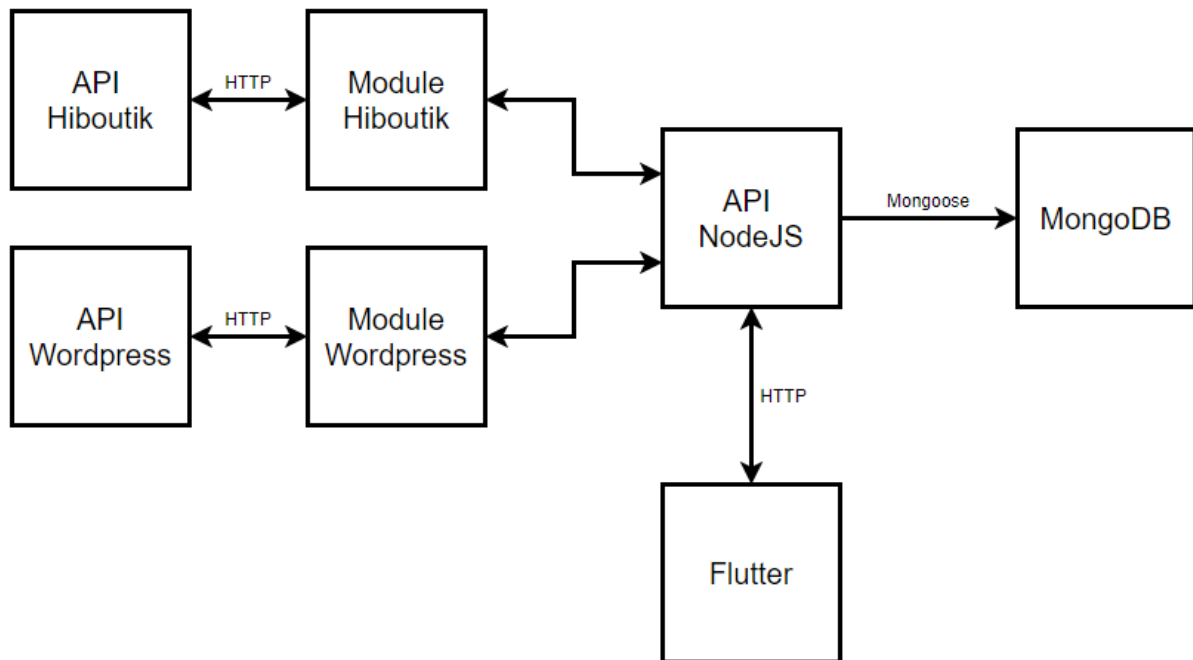
#### **API Hiboutik :**

Cette API est proposée par le constructeur de la plateforme de caisse en ligne Hiboutik. Elle permet de lister les clients ayant effectué des achats en magasin.

#### **API Wordpress :**

Cette API est proposée par défaut par Wordpress depuis 2012. Elle permet de lister les fiches lithothérapie présentes sur le site internet de la semeuse de pierres.

#### 2.5.4. Patron de conception



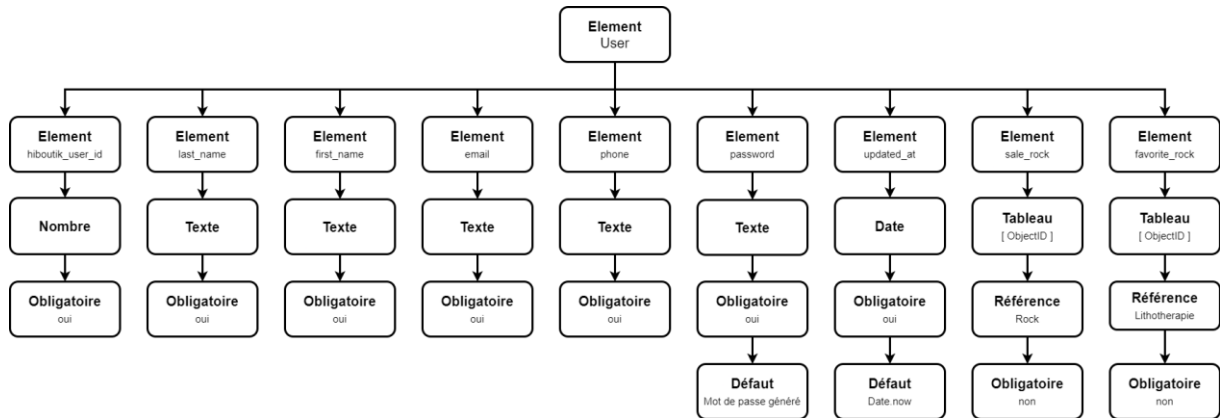
L'API NodeJS récupère les informations des APIs de Wordpress, d'Hiboutik et de sa base de données MongoDB dans le but de les envoyer à l'application mobile.

Pour cela, l'application mobile requête l'API NodeJS. L'API NodeJS demande à l'API d'Hiboutik la liste des achats effectués par l'utilisateur connecté. L'API NodeJS récupère la liste des pierres achetées, les filtre pour éviter les doublons et demande à L'API de Wordpress les détails de chaque contenu dans la liste nouvellement obtenue. Pour finir, l'API NodeJS transmet les données obtenues précédemment pour afficher la liste des pierres achetées par le client.

## 2.5.5. Base de données

### 2.5.5.1. User

Schéma :



*Modélisation de l'objet User pour MongoDB*

Modèle JSON :

```
1 {
2   "sale_rock": ["6109faa0e44a5d53c00a52fz", "60febbc4615f110e5c87ac9f"], // Liste des pierres
   acheté par le client
3   "favorite_rock": ["60febbc4615f110e5c87aca6"], // Liste des pierres mis en favori par le client
4   "_id": "6109faa0e44a5d53c00a286d", // ID de l'objet auto-généré
5   "hiboutik_user_id": "1229", // ID du client Hiboutik
6   "last_name": "Doe", // Nom du client
7   "first_name": "John", // Prénom du client
8   "email": "doe.john@demo.fr", // Email du client
9   "phone": "0606060606", // Numéro de téléphone du client
10  "password": "$2b$10$.lgcgkQ0JACKAcrSaoFkx.Mza4K/8Tj9pjep3CGKbsxIqgTvbGkVW", // Mot de passe
   encrypté du client
11  "updated_at": "2021-08-04T02:25:36.997Z" // Date de dernière mise à jour du client
12 }
```

*Exemple de données MongoDB au format JSON pour les utilisateurs (User)*

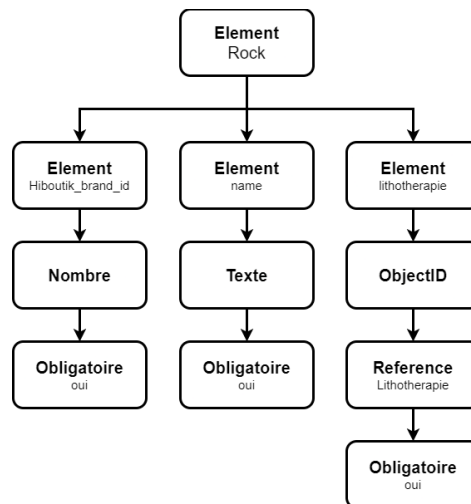
```
1 {
2   "hiboutik_user_id" : 1229,
3   "last_name": "Doe",
4   "first_name": "John",
5   "email": "doe.john@demo.fr",
6   "phone": "0606060606",
7   "password": "Password123."
8 }
```

*Exemple de données d'entrées pour la création d'un utilisateur (User)*

L'ajout de pierres dans les tableaux « sale\_rock » et « favorite\_rock » est effectué ultérieurement afin de vérifier qu'il n'y a pas deux fois la même. Elles sont donc vérifiées et ajoutées une par une dans leur liste respective.

### 2.5.5.2. Rock

Schéma :



*Modélisation de l'objet Rock pour MongoDB*

Modèle JSON :

```

1 {
2   "_id": "60febbc4615f110e5c87ac9f", // ID de la pierre
3   "hiboutik_brand_id": "1", // ID de la pierre d'Hiboutik
4   "name": "Améthyste", // Nom de la pierre
5   "lithotherapie_id" : "61024f3ee889ea41ac2b225e" // ID de la fiche Lithothérapie
6 }

```

*Exemple de données MongoDB au format JSON pour les pierres (Rock)*

```

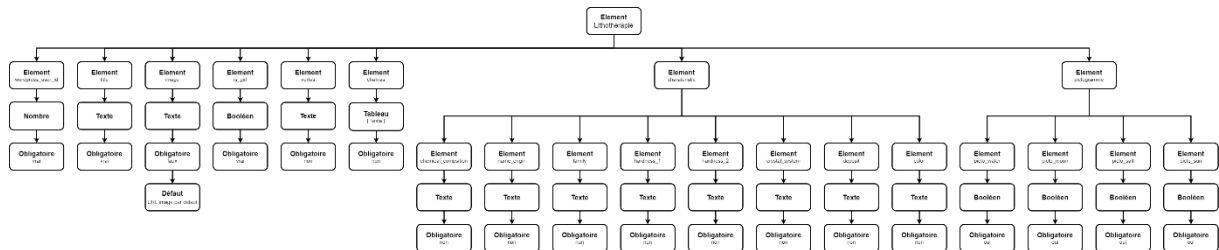
1 {
2   "hiboutik_brand_id": "1",
3   "name": "Améthyste",
4   "lithotherapie_id" : "61024f3ee889ea41ac2b225e"
5 }

```

*Exemple de données d'entrées pour la création d'une pierre (Rock)*

### 2.5.5.3. Lithothérapie

Schéma :



Modélisation de l'objet Lithothérapie pour MongoDB

Modèle JSON :

```
1 {
2   "id": "61024ed81667c72d74791fd1", // ID de la fiche lithothérapie
3   "wordpress_id": 19977, // ID de la fiche lithothérapie de Wordpress
4   "title": "Agate Mousse", // Nom de la fiche lithothérapie
5   "image": "URL IMAGE", // URL de l'image vers le site de la Semeuse de pierre
6   "is_girl": true, // Savoir si la pierre est féminine ou masculine
7   "vertus": "Les vertus de la pierre",
8   "chakras": [], // Liste des chakras de la pierre
9   "properties": {
10    "physical": "Description des biens physique",
11    "mental": "Description des biens mental"
12  },
13  "characteristic": {
14    "chemical_composition": "Dioxyde de silicium, SiO2.", // Composition chimique de la pierre
15    "name_origin": "Elle devrait plutôt s'appeler Calcedoine Mousse car c'est une variété de Calcedoine.", // L'origine du nom de la pierre
16    "family": "Quartz", // groupe d'appartenance de la pierre
17    "hardness_1": "6.5", // Valeur de résistance de la pierre
18    "hardness_2": "7", // second valeur de résistance de la pierre si la résistance est comprise entre deux valeurs
19    "crystal_system": "Rhomboédrique", // Système cristallin de la pierre
20    "deposit": "partout dans le monde", // Gisement de la pierre
21    "color": "L'agate mousse est incolore avec des stries ou des fils vert et des structures faisant penser à de la mousse." // Les
couleurs de la pierre
22  },
23  "pictogramme": { // Si vrai, la pierre ne supporte pas l'élément. Si faux, la pierre supporte l'élément
24    "picto_water": false,
25    "picto_moon": false,
26    "picto_salt": false,
27    "picto_sun": false
28  }
29 }
```

Exemple de données MongoDB au format JSON pour les fiches lithothérapies (Lithothérapie)

```
1 {
2   "wordpress_id": 19977,
3   "title": "Agate Mousse",
4   "image": "URL IMAGE", // Si non renseigné, c'est l'URL par défaut qui est pris en compte
5   "is_girl": true,
6   "vertus": "Les vertus de la pierre",
7   "chakras": ["chakras_1", "chakras_2"],
8   "properties": {
9     "physical": "Description des biens physique",
10    "mental": "Description des biens mental"
11  },
12  "characteristic": {
13    "chemical_composition": "Dioxyde de silicium, SiO2.",
14    "name_origin": "L'agate mousse devrait plutôt s'appeler Calcedoine Mousse car c'est une variété de Calcedoine. Et on l'appelle mousse
car sa structure, fait penser à la mousse.",
15    "family": "Quartz",
16    "hardness_1": "6.5",
17    "hardness_2": "7",
18    "crystal_system": "Rhomboédrique",
19    "deposit": "partout dans le monde",
20    "color": "L'agate mousse est incolore, bleu clair, brune, avec des stries ou des fils vert et des structures faisant penser à de la
mousse."
21  },
22  "pictogramme": {
23    "picto_water": false,
24    "picto_moon": false,
25    "picto_salt": false,
26    "picto_sun": false
27  }
28 }
```

Exemple de données d'entrées pour la création d'une fiche lithothérapie (Lithothérapie)

#### 2.5.5.4. Les IDs Hiboutik et Wordpress ?

Les formats des objets précédents sont présents pour la base de données interne. Nous y retrouvons les lignes « hiboutik\_user\_id », « hiboutik\_brand\_id » et « wordpress\_id » qui sont les références (ID) aux anciennes bases de données d'Hiboutik et de Wordpress. Elles sont toujours présentes afin d'éviter de doubler les informations.

Durant une tâche Cron (cf. Cron ?), le programme vérifie que l'objet existe pour savoir s'il doit le créer ou non. Dans le cas où l'objet est existant, le script vérifie si les champs sont différents. Si tel est le cas, l'objet est mis à jour.

#### 2.5.6. Contraintes

##### 2.5.6.1. Connexion

**Email :**

- 1 '@' et un nom de domaine (demo.fr)
- Pas de caractères spéciaux (!?.,@\* etc..)

**Mot de passe :**

- 8 caractères minimum | 25 maximum
- 1 majuscule minimum
- 1 caractère spécial minimum (!?.,@\* etc..)
- 1 chiffre minimum [0-10]

##### 2.5.6.2. Mot de passe oublié

**Email :**

- 1 '@' et un nom de domaine (demo.fr)
- Pas de caractères spéciaux (!?.,@\* etc..)

##### 2.5.6.3. Inscription

**Nom :**

- 2 caractères minimum
- Pas de caractères spéciaux
- Pas de chiffres [0-10]

**Prénom :**

- 2 caractères minimum
- Pas de caractères spéciaux (!?.,@\* etc..)
- Pas de chiffres [0-10]

**Email :**

- 1 '@' et un nom de domaine (demo.fr)
- Pas de caractères spéciaux (!?.,@\* etc..)

**Téléphone :**

- 10 caractères minimum | 10 caractères maximum
- 10 chiffres minimum [0-10]
- Pas de lettres [A-Z]
- Pas de caractères spéciaux (!?.,@\* etc..)

**Mot de passe :**

- 8 caractères minimum | 25 maximum
- 1 majuscule minimum
- 1 caractère spécial minimum (!?.,@\* etc..)
- 1 chiffre minimum [0-10]

**Confirmation du mot de passe :**

- Identique au « Mot de passe »

#### 2.5.6.4. Modifier profil

**Nom :**

- 2 caractères minimum
- Pas de caractères spéciaux
- Pas de chiffres [0-10]

**Prénom :**

- 2 caractères minimum
- Pas de caractères spéciaux (!?.,@\* etc..)
- Pas de chiffres [0-10]

**Email :**

- 1 '@' et un nom de domaine (demo.fr)
- Pas de caractères spéciaux (!?.,@\* etc..)

**Téléphone :**

- 10 caractères minimum | 10 caractères maximum
- 10 chiffres minimum [0-10]
- Pas de lettres [A-Z]
- Pas de caractères spéciaux (!?.,@\* etc..)



### 2.5.6.5. Modifier mot de passe

**Mot de passe :**

- 8 caractères minimum | 25 maximum
- 1 majuscule minimum
- 1 caractère spécial minimum (!?.,@\* etc..)
- 1 chiffre minimum [0-10]

**Confirmation du mot de passe :**

- Identique au « Mot de passe »

## 2.6. Sécurité

### 2.6.1. IOS Keychain

Les appareils sous IOS disposent d'une base de données sécurisée nommée « trousseau de clés » ou encore « keychain ». Cette base de données a pour principe de stocker des données sensibles telles que des mots de passe, des informations bancaires ou des notes personnelles.

### 2.6.2. ANDROID Keystore

Au même titre que Keychain (cf.**IOS Keychain**IOS Keychain), les appareils sous Android disposent d'un conteneur sécurisé interne rendant l'extraction de données le plus difficile possible.

### 2.6.3. JWT:

Le JWT est une méthode sécurisée d'échange d'informations sous la forme d'un jeton signé afin d'en vérifier la légitimité. Ce jeton peut être intégré dans l'en-tête d'une requête HTTP.

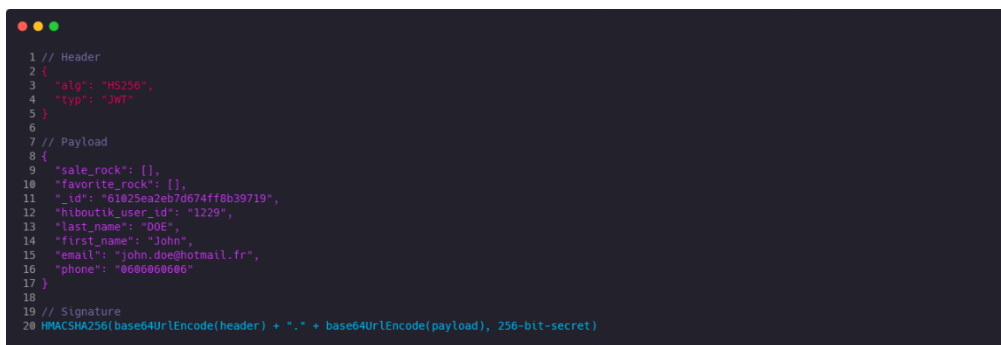
Un JWT se compose de 3 chaînes de caractères séparées par un point.

### Exemple de token

La première partie est l'« en-tête » du token. C'est un objet JSON encodé en base64 qui indique le type de token et l'algorithme utilisé pour la signature.

La seconde partie est le « payload » du token. C'est un objet JSON encodé en base64 contenant les informations à transmettre. Il peut contenir toutes les informations ainsi que des clés prédéfinies appelées « claims » qui donnent la possibilité d'ajouter des informations supplémentaires sur le token tels que le sujet, la date de validité, la date d'expiration, etc...

La dernière partie, la plus importante, est la « Signature » du token. Elle permet de s'assurer de l'authenticité du token. Contrairement aux deux précédentes parties, elle n'est pas encodée en base64. La génération de la signature est effectuée à partir des deux parties précédentes ainsi que d'une clé secrète en utilisant l'algorithme indiqué dans la première partie (exemple : HS256).



```
1 // Header
2 {
3   "alg": "HS256",
4   "typ": "JWT"
5 }
6
7 // Payload
8 {
9   "sale_rock": [],
10  "favorite_rock": [],
11  "id": "61025aa2eb7d674ff8b39719",
12  "hiboutik_user_id": "1229",
13  "last_name": "DOE",
14  "first_name": "John",
15  "email": "john.doe@hotmail.fr",
16  "phone": "0606060606"
17 }
18
19 // Signature
20 HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), 256-bit-secret)
```

*Exemple avant encodage (les couleurs correspondent avec la photo précédente)*

La base64 peut être encodée et décodée par n'importe qui à tout moment. Il est donc simple de décoder la partie « payload », récupérer la valeur, la modifier, l'encoder et remettre le résultat dans le JWT.

C'est ainsi que la partie « signature » vérifie que le payload correspond à la valeur qu'il avait initialisée avec les données de base. Si la comparaison est identique, le token est valide et peut être utilisé.

```
1 {
2   "sale_rock": [],
3   "favorite_rock": [],
4   "id": "61025ea2eb7d674ff8b39719",
5   "hiboutik_user_id": "1229",
6   "last_name": "DOE",
7   "first_name": "John",
8   "email": "john.doe@hotmail.fr",
9   "phone": "0606060606"
10 }
```

### Payload initial JSON



```
1 eyJzYXNjLXN3YjY2Si01tdlCjMYXZvcml0ZV9yb2
  NriJpbXSwiX2klJioiNjEwMjVlYjY2JyJkdNjc0
  ZmY4YjM5NE5iWiAGLlb3V0awtfdXNlclpZC
  I6IjEyMjkiLXN3YXNj025hbWU0iJET0UilCjM
  aXZzdF9uYWllIjoiSm9obiIsImVYbWVsIjoiam
  9ob25kb2VAAg90bWpCbC5mcilSiInBob25lIjoi
  MDYwNjA2MDYwNiJ9
```

*Payload Initial encodé (base64)*  
*Correct*

```
1 {
2   "sale_rock": [],
3   "favorite_rock": [],
4   "_id": "61025ea2eb7d674ff8b39719",
5   "hiboutik_user_id": "1229",
6   "last_name": "DOE",
7   "first_name": "John",
8   "email": "admin@hotmail.fr",
9   "phone": "0606060606"
10 }
```

### Modification de l'adresse email

→

```
1 eyJzYXNjLXN3Y2Y2Si0tdCJmYXZvcml0ZVY9b2
  Nr1jpbXSwiX2lkIjoInJEWmJlVlYlJlYjdnNjc0
  ZmY4YjYMSnZ5EiIwIaglib3V0awtfdXNlc1pZC
  I6IjEYmjkIJCjSYXN0X25hbWUoiEjE0UilCjMj
  aXZzdF9uYWllIjoImS9obiIsImVhYiYwLjoiIjYw
  RtaW5AaG90bW9wcm5kIjE2InB0b251IjoIMDYw
  NjA2MDYwNii39
```

Résultat encodé (base64) avec nouvelle  
adresse email  
**INCORRECT**

#### 2.6.4. Flutter secure storage

Flutter secure storage est une librairie pour Flutter qui permet par des données paires clé-valeur de stocker des informations sensibles directement dans la Keychain (cf.**IOS Keychain**) ou le Keystore (cf.**ANDROID Keystore**) selon l'OS courant de manière automatique.

La librairie permet en quelques lignes de :

- Stocker de nouvelles données
- Consulter toutes les données
- Consulter une donnée spécifique
- Supprimer toutes les données
- Supprimer une donnée spécifique

```
1 var storage = new FlutterSecureStorage();
2 storage.write(key : "clé", value : "valeur")
```

### Exemple de stockage d'une nouvelle donnée

```
1 var value = await storage.read(key: "clé");
2 var allValues = await storage.readAll()
```

### Exemple de consultation de données

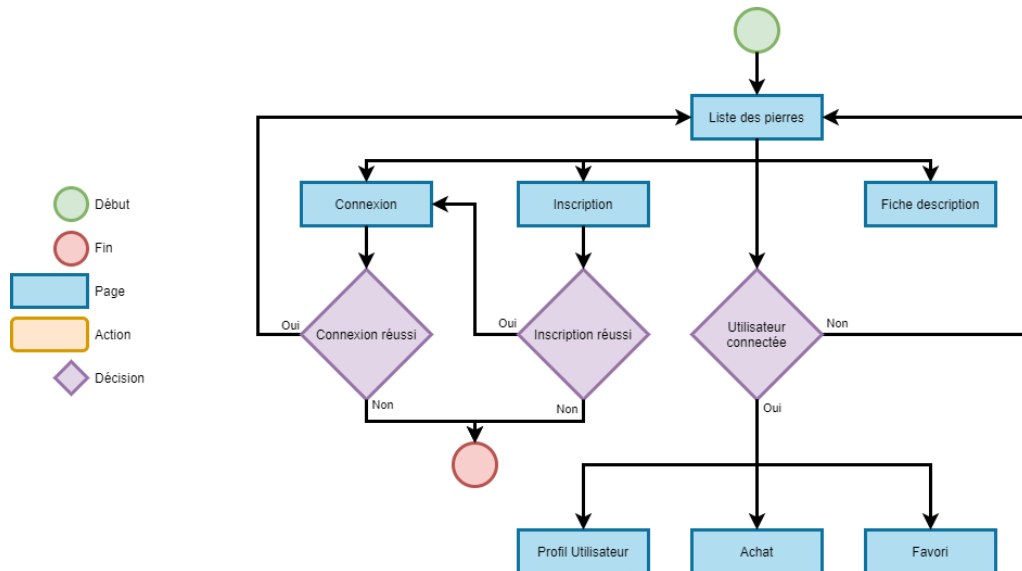
```
1 storage.delete(key: "clé");
2 storage.deleteAll();
```

### Exemple de suppression de données

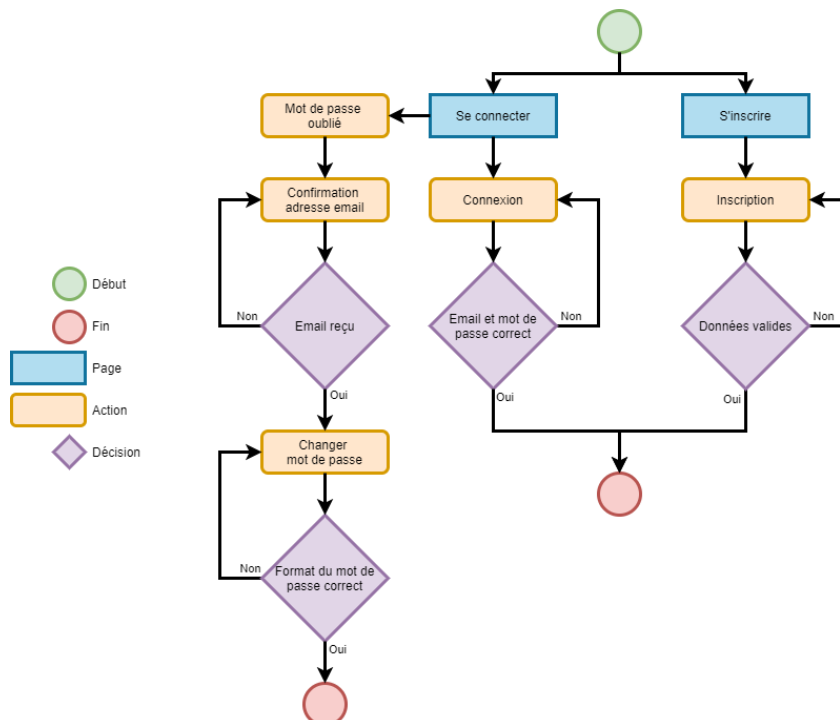
## 2.7. Réalisation

### 2.7.1. Parcours UX

#### 2.7.1.1. Global

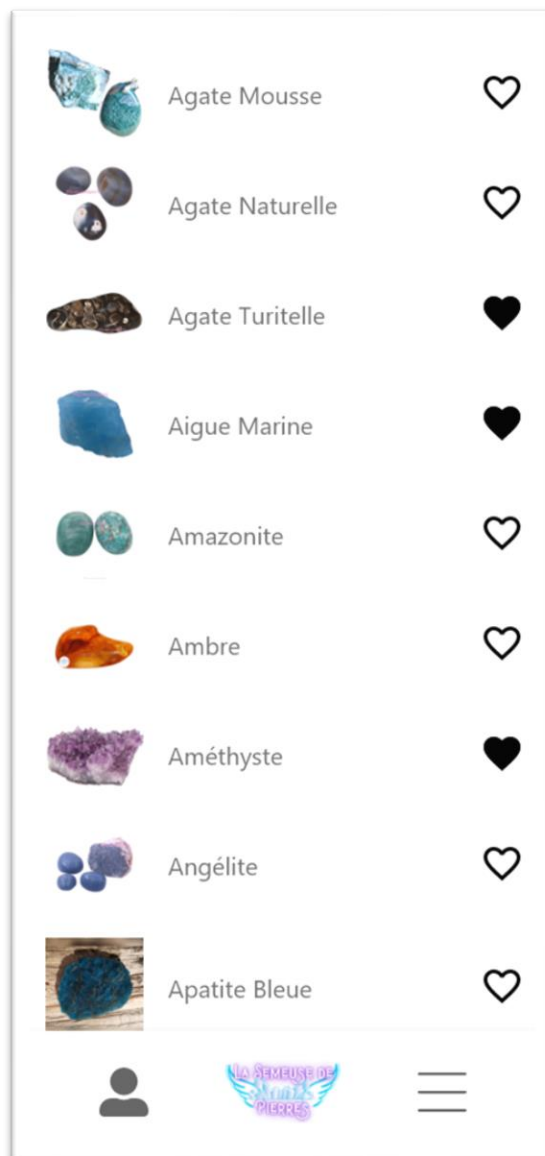


#### 2.7.1.2. Connexion/Inscription



## 2.7.2. Maquette

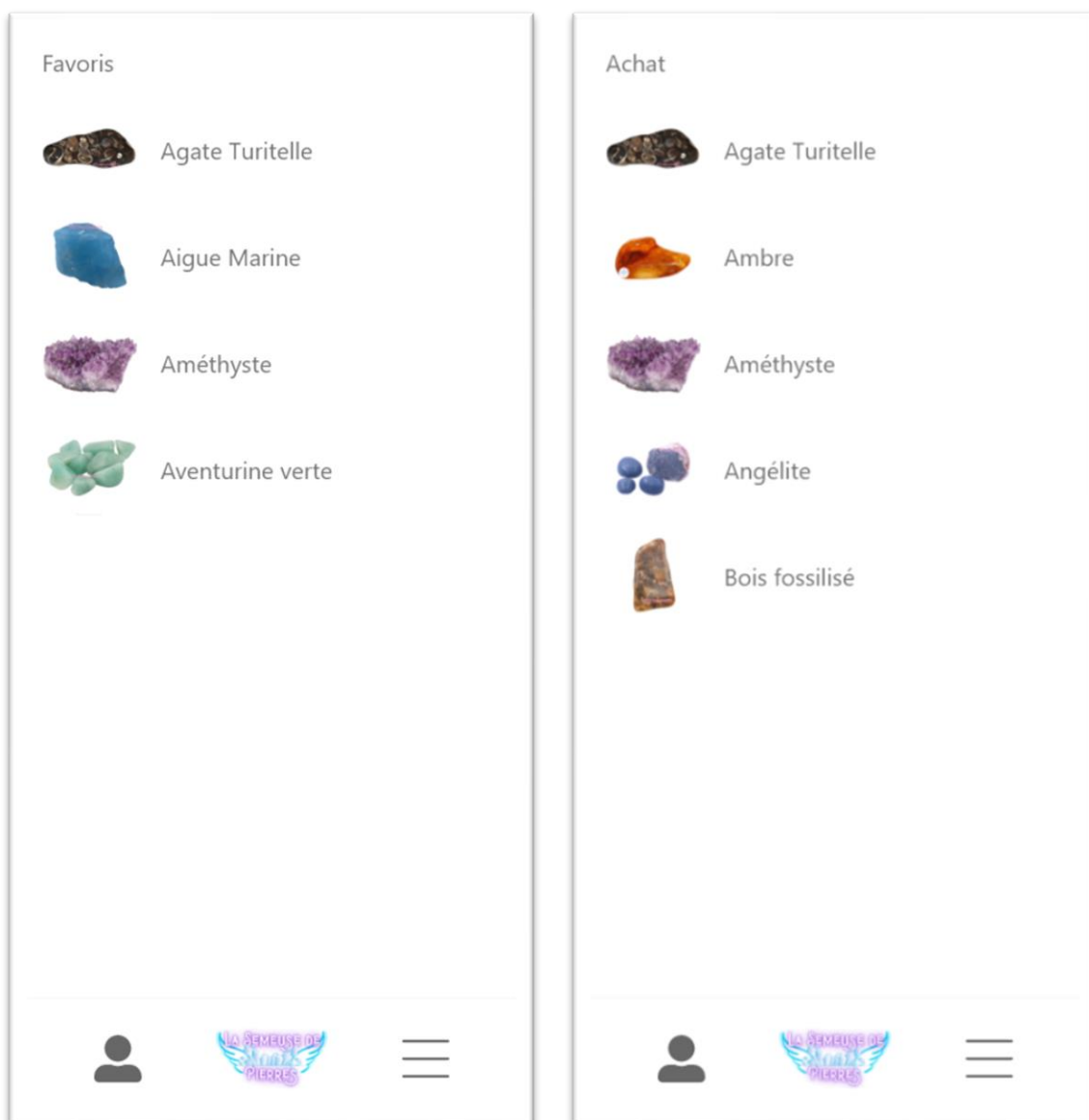
### 2.7.2.1. Page d'accueil



La page d'accueil liste les pierres présentes sur le site de la semeuse de pierre. Chaque cellule correspond à une fiche lithothérapie. Sélectionner une cellule pour accéder au descriptif complet de la fiche. (cf. **Page description d'une fiche**). Sélectionner le petit cœur sur la droite d'une cellule afin de l'ajouter au favori de l'utilisateur connecté.

L'icône du cœur pour l'ajout au favori n'est pas présent si l'utilisateur n'est pas connecté.

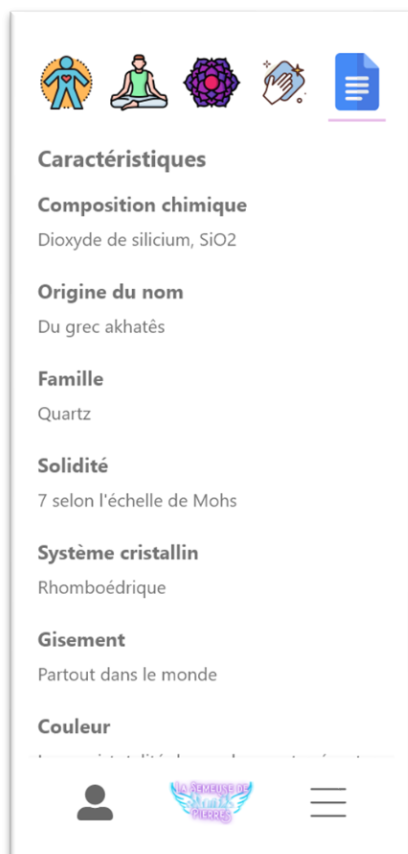
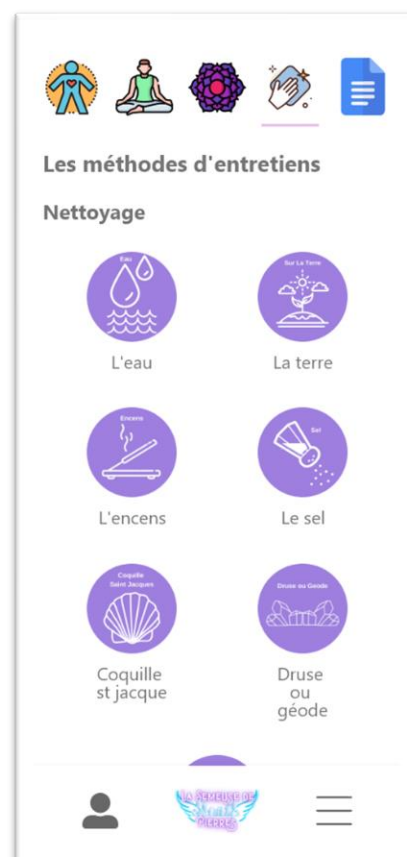
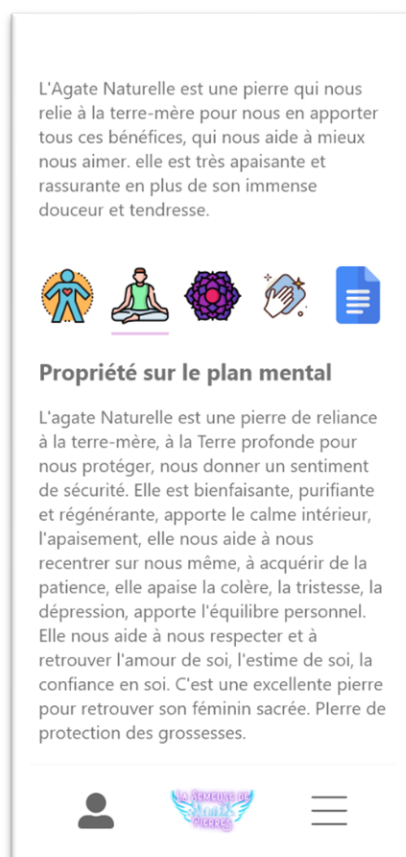
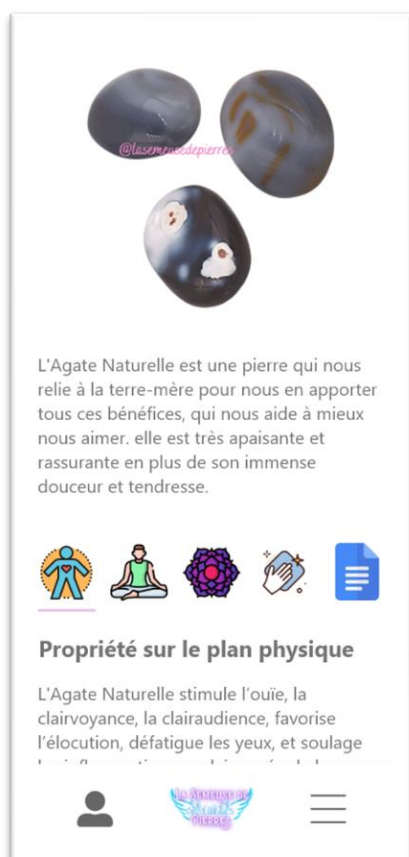
### 2.7.2.2. Page Favori et Achat



La page des favoris liste les pierres que l'utilisateur a sélectionnées. Comme sur la page d'accueil (cf. **Page d'accueil**), chaque cellule est cliquable afin d'accéder à la description complète de la fiche.

La page des achats liste les pierres achetées par le client. Comme sur la page d'accueil (cf. **Page d'accueil**) chaque cellule est cliquable afin d'accéder à la description complète de la fiche.

### 2.7.2.3. Page description d'une fiche



La page de description d'une fiche lithothérapie contient toutes les informations relatives à une pierre sous plusieurs sections :

- La photo de la pierre
- Une brève description (vertus)
- Une zone d'onglet renfermant davantage de descriptions

La zone d'onglet est composée de différentes parties :

- Les propriétés sur le plan physique
- Les propriétés sur le plan mental
- Les chakras
- Les méthodes d'entretiens
- Les caractéristiques

#### 2.7.2.4. Page connexion

Connexion




Email

Mot de passe

Envoyer

[S'inscrire](#)

[Mot de passe oublié](#)





### 2.7.3. Extrait de code

Démonstration de la partie Cron pour la planification de tâches. Cette partie est intéressante, car elle associe les modules pour Hiboutik et Wordpress créée à l'occasion ainsi que le module Cron.

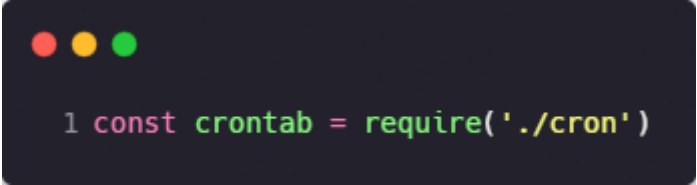
Dans le cadre du projet, la planification de tâches agit sur les parties suivantes :

- Hiboutik
  - Liste des marques (pierre)
  - Liste des clients
- Wordpress
  - Liste des fiches lithothérapie

Dans un projet NodeJS, un fichier principal (index.js ou server.js) est initialisé en premier. C'est à travers ce fichier que le serveur NodeJS va démarrer. Ce fichier a pour principe d'appeler tous les autres fichiers du projet nécessaires.

Dans le cadre du projet, un fichier cron.js a été créé à la racine. Ce fichier contenant toutes les fonctions utiles pour la mise à jour des éléments indiqués précédemment.

Une fois le fichier créé, par un appel dans le fichier principal, le fichier cron.js est appelé au démarrage du serveur.



```
1 const crontab = require('./cron')
```

*Appel du fichier cron.js*

### 2.7.3.1. Les appels de dépendances

```
1 const { Rock } = require("./src/db/model/rocks")
2 const { Lithotherapie } = require("./src/db/model/lithotherapies")
3 const { User } = require("./src/db/model/users")
4
5 // customs modules
6 const Hiboutik = require('hiboutik-api')
7 const Wordpress = require('wordpress-api')
```

Au début du fichier se trouvent les appels des dépendances. Le but étant de mettre à jour les éléments indiqués ci-dessus, les trois modèles Mongoose pour la base de données sont appelés. Cela permet d'interagir avec la base de données en consultation, en création, en modification et en suppression pour ces éléments précis.

C'est aussi à ce moment-là que les modules d'Hiboutik et de Wordpress sont appelés permettant l'utilisation de leurs fonctionnalités.

Chaque appel de dépendance est associé à une constante permettant de les utiliser par la suite.

### 2.7.3.2. Les instances

```
1 // Hiboutik settings
2 const { host, username, password } = {
3   host : process.env.HIBOUTIK_HOST,
4   username : process.env.HIBOUTIK_USERNAME,
5   password : process.env.HIBOUTIK_PASSWORD
6 }
7
8 // Wordpress settings
9 const { URL } = {
10   URL : process.env.WORDPRESS_URL
11 }
12
13 // Hiboutik instance
14 const hiboutik = new Hiboutik({ host, username, password });
15
16 // Wordpress instance
17 const wordpress = new Wordpress({ URL });
```

Cette section de code contient les instances des classes contenues dans les modules d'Hiboutik et de Wordpress. Dans un premier temps, nous préparons toutes les configurations relatives aux instanciations.

Hiboutik a besoin de trois paramètres :

- **Host** : URL de l'API
- **Username** : Identifiant d'accès à l'API
- **Password** : Mot de passe d'accès à l'API

Ces trois informations ont pour principe de générer le JWT donnant les autorisations ainsi que l'accès à l'API d'Hiboutik.

Contrairement à Hiboutik, Wordpress est une API libre-service et accessible par toute personne ayant connaissance de l'URL. Wordpress prend donc une URL en seul argument.

Les morceaux de code contenant « `process.env.###` » sont des variables d'environnement stockées dans un fichier « `.env` ». Cela permet d'avoir accès à ses variables n'importe où dans le projet et d'éviter de les écrire plusieurs fois.

Une fois les configurations terminées, nous créons des constantes contenant l'instance des classes d'Hiboutik et de Wordpress avec les paramètres d'entrées.

Il est à noter que les morceaux de code « `new Hiboutik()` » et « `new Wordpress` » sont possibles grâce aux appels de dépendances (cf. Les appels de dépendances) faits précédemment.

### 2.7.3.3. Les fonctions

Mettre à jour la liste des pierres :

```
1 /**
2  * hiboutik.brand = Rock
3  *
4  * Checks that a stone is saved in the internal database.
5  * If the stone is not present, it is created and saved.
6  * If it is present, nothing happens.
7  * If it is present but the content is different, it is updated.
8  */
9 updateRockByBrand = async () => {
10   const brands = await hiboutik.all_brands()
11
12   brands.forEach(brand => {
13     Rock.find({"hiboutik_brand_id" : brand.brand_id}, (err, current_rock) => {
14       if (current_rock[0] === undefined) {
15         Rock.create({"hiboutik_brand_id" : brand.brand_id, "name" : brand.brand_name});
16       } else {
17         if (current_rock[0].name !== brand.brand_name) {
18           current_rock[0].name = brand.brand_name
19           current_rock[0].save()
20         }
21       }
22     })
23   });
24 }
```

Cette fonction a pour principe de mettre à jour les pierres présentes dans la base de données avec les données contenues sur Hiboutik. Cela permet en cas de mise à jour sur Hiboutik qu'elle se fasse aussi pour l'application mobile.

En premier lieu, nous listons sous la forme d'un tableau les pierres présentes dans Hiboutik grâce à la fonction « `hiboutik.all_brands()` » .

Une fois la liste initiée, nous parcourons chaque ligne (chaque ligne correspond à une « itération ») du tableau. À chaque itération, nous demandons à la base de données interne si l'ID de la pierre sur Hiboutik (`brand_id`) existe.

Si elle n'existe pas, la pierre est sauvegardée en base de données. Si elle existe, rien ne se passe. Si elle existe, mais qu'un champ autre que l'ID Hiboutik est différent, elle est mise à jour et sauvegardée en base de données.

```

1 /**
2  * Checks that a lithotherapy card is saved in the internal database.
3  * If the record is not present, it is created and saved.
4  * If it is present, nothing happens.
5  * If it is present but the content is different, it is updated.
6  */
7 updateLithotherapie = async () => {
8   const lithotherapies = await wordpress.all_posts('/lithotherapie')
9
10  lithotherapies.forEach(lithotherapie => {
11    Lithotherapie.find({"wordpress_id" : lithotherapie.id}, (err, current_post) => {
12      if (current_post[0] === undefined) {
13        Lithotherapie.create({
14          "wordpress_id" : lithotherapie.id,
15          "title" : lithotherapie.title,
16          "is_girl" : lithotherapie.is_girl,
17          "vertus" : lithotherapie.vertus,
18          "chakras" : lithotherapie.chakras,
19          "properties" : {
20            "physical" : lithotherapie.properties.physical,
21            "mental" : lithotherapie.properties.mental
22          },
23          "characteristic" : {
24            "chemical_composition" : lithotherapie.characteristic.chemical_composition,
25            "name_origin" : lithotherapie.characteristic.name_origin,
26            "family" : lithotherapie.characteristic.family,
27            "hardness_1" : lithotherapie.characteristic.hardness_1,
28            "hardness_2" : lithotherapie.characteristic.hardness_2,
29            "crystal_system" : lithotherapie.characteristic.crystal_system,
30            "deposit" : lithotherapie.characteristic.deposit,
31            "color" : lithotherapie.characteristic.color
32          },
33          "pictogramme" : {
34            "picto_water" : lithotherapie.pictogramme.picto_water,
35            "picto_moon" : lithotherapie.pictogramme.picto_moon,
36            "picto_salt" : lithotherapie.pictogramme.picto_salt,
37            "picto_sun" : lithotherapie.pictogramme.picto_sun
38          }
39        });
40      } else {
41        //TODO Find an efficient way to update the lithotherapy cards if there is at least one difference
42        // (between the database and wordpress) in 1 field
43      }
44    })
45  })
46 }

```

Cette fonction a pour principe de mettre à jour les fiches lithothérapies présentes dans la base de données avec les données contenues sur Wordpress. Cela permet en cas de mise à jour sur Wordpress qu'elles se fassent aussi pour la BDD interne.

En premier lieu, nous listons sous la forme d'un tableau les pierres présentes dans Wordpress grâce à la fonction « `wordpress.all_posts('/lithotherapie')` ».

Une fois la liste initiée, nous parcourons chaque ligne (chaque ligne correspond à une « itération ») du tableau. À chaque itération, nous demandons à la base de données interne si l'ID de la fiche lithothérapies sur Wordpress (`wordpress_id`) existe.

Si elle n'existe pas, la pierre est sauvegardée en base de données. Si elle existe, rien ne se passe. Si elle existe, mais qu'un champ autre que l'ID Wordpress est différent, elle est mise à jour et sauvegardée en base de données.

#### 2.7.3.4. Le Cron

```
1 /**
2  * Executes functions every 2 hours
3  */
4 cron.schedule('* * 2 * * *', async () => {
5     updateRockByBrand()
6     updateLithotherapie()
7 })
```

Cette partie correspond à la planification de tâches. Toutes les 2 heures, l'application exécutera les fonctions présentées précédemment.

#### 2.7.4. Jeux d'essai (test unitaire, test intégration, TDD)

Les tests ont été réalisés grâce aux modules NodeJS « Mocha » et « Chai »

Le projet est développé dans une logique TDD. Le jeu de test est implémenté avant le code. Les tests ont été rédigés en JavaScript dans le cadre d'un projet NodeJS, complété des modules « Mocha » et « Chai ». Ces deux modules sont complémentaires et permettent une bonne rédaction grâce à une syntaxe simple et épurée.

##### 2.7.4.1. Mocha

Mocha est un module permettant d'exécuter des tests en JavaScript grâce à des « assertions ». C'est un module léger auquel nous pouvons y intégrer d'autres modules.

```
1 describe('example test', () => {
2     it('should be an example test', () => {
3         // verification
4     })
5 })
```


*Exemple de la syntaxe Mocha*

### 2.7.4.2. Chai

Chai est un module qui donne la possibilité d'écrire les tests avec d'autres syntaxes selon les préférences.

Le module possède au total trois syntaxes qui permettent d'écrire des assertions :

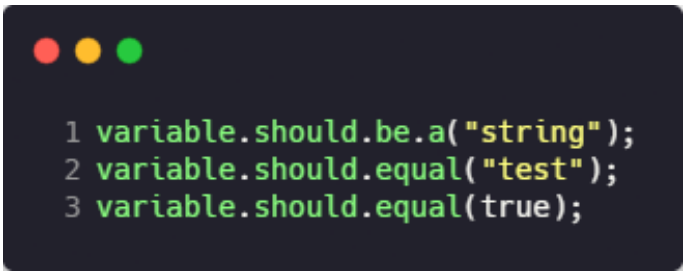
- **Assert**



```
1 assert.isTrue(variable);
2 assert.isTrue(variable, "Message en cas d'erreur");
3 assert.isBelow(variable, 5);
```

*Exemple de syntaxe Assert*

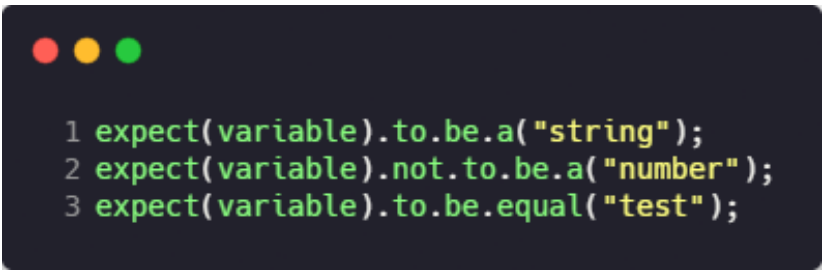
- **Should**



```
1 variable.should.be.a("string");
2 variable.should.equal("test");
3 variable.should.equal(true);
```

*Exemple de syntaxe Should*

- **Expect**



```
1 expect(variable).to.be.a("string");
2 expect(variable).not.to.be.a("number");
3 expect(variable).to.be.equal("test");
```

*Exemple de syntaxe Expect*

Dans le cadre du projet, la syntaxe « Expect » est utilisée pour l'élaboration des tests. Ce choix a été fait, car sa syntaxe est relativement simple à comprendre, donnant l'impression de lire une phrase comme : s'attendre à ce que la variable soit une chaîne de caractères (Exemple de syntaxe Expect : Ligne 1).

### 2.7.4.3. Test module Hiboutik

Sur l'image ci-dessous (Exemple de jeu de test pour le module Hiboutik), les tests sont portés sur le Module « Hiboutik ». L'exemple vérifie que l'instance de la table Hiboutik se passe bien et génère correctement un token de type Bearer (ligne 12), utile pour communiquer avec l'API d'Hiboutik.

Il teste aussi la fonction « all\_customers » afin de vérifier si toutes les données d'entrée et de sortie sont correctes.

#### #all\_customers:

En premier lieu, le test vérifie que la méthode « hiboutik.all\_customers » existe dans la classe Hiboutik (ligne 56).

En second lieu, le test vérifie si la méthode retourne bien un tableau de valeur (ligne 62). Il vérifie par la suite si le format du tableau (ligne : 63) correspond à un format précis selon un schéma donné (ligne 19 à 53).

Le tableau en question doit être un tableau contenant des objets (exemple : {"donnée1" : "valeur", "donnée2" : "valeur"}).

Chaque objet doit contenir obligatoirement :

- **customers\_id** : valeur numérique uniquement et valeur unique
- **last\_name** : valeur alphanumérique uniquement et valeur unique
- **first\_name** : valeur alphanumérique uniquement et valeur unique
- **email** : valeur alphanumérique uniquement et valeur unique
- **phone** : valeur alphanumérique uniquement et valeur unique

Si toutes ces conditions ne sont pas remplies, le test sera considéré comme faux.



## Résultat :

Le résultat des tests est affiché dans un terminal en exécutant la commande définie dans le fichier « package.json ». Dans le cas présent, la commande est « npm test ».

Comme nous sommes dans une logique TDD, toutes les vérifications sont initialement de couleur rouge. C'est au fur et à mesure de la rédaction du code qu'elles passeront au vert.

```
Hiboutik
1) should generate the basic token
#all_customers
2) Should check if the function exist
3) Should return all hiboutik customers
#all_brands
4) Should check if the function exist
5) Should return all hiboutik brands
#sale_by_customer_id
6) Should check if the function exist
7) Should return array of sales.product_id
#product_by_id
8) Should check if the function exist
9) Should return product.product_brand

0 passing (22ms)
9 failing
```

*Avant*



```
Hiboutik
✓ should generate the basic token
#all_customers
✓ Should check if the function exist
✓ Should return all hiboutik customers (1276ms)
#all_brands
✓ Should check if the function exist
✓ Should return all hiboutik brands (96ms)
#sale_by_customer_id
✓ Should check if the function exist
✓ Should return array of sales.product_id (63ms)
#product_by_id
✓ Should check if the function exist
✓ Should return product.product_brand (71ms)

9 passing (2s)
```

*Après*

```

1 describe('Hiboutik', () => {
2     let {host, username, password} = {
3         host : "****",
4         username : "****",
5         password : "****",
6     }
7
8     const hiboutik = new Hiboutik({host, username, password});
9
10
11     it('should generate the basic token', () => {
12         expect(hiboutik.token).toEqual("****")
13     })
14
15     let customers;
16     let brands;
17
18     describe("#all_customers", () => {
19         const customerSchema = {
20             title : 'Customer schema',
21             type : 'array',
22             items : {
23                 type : 'object',
24                 required : ['customers_id', 'last_name', 'first_name', 'email', 'phone'],
25                 properties : {
26                     customers_id : {
27                         type : 'number',
28                         minItems : 1,
29                         uniqueItems : true,
30                     },
31                     last_name : {
32                         type : 'string',
33                         minItems : 1,
34                         uniqueItems : false,
35                     },
36                     first_name : {
37                         type : 'string',
38                         minItems : 1,
39                         uniqueItems : false,
40                     },
41                     email : {
42                         type : 'string',
43                         minItems : 1,
44                         uniqueItems : true,
45                     },
46                     phone : {
47                         type : 'string',
48                         minItems : 1,
49                         uniqueItems : true,
50                     }
51                 }
52             }
53         }
54
55         it('Should check if the function exist', () => {
56             expect(hiboutik.all_customers).toBe.a('function')
57         })
58
59         it('Should return all hiboutik customers', async () => {
60             customers = await hiboutik.all_customers();
61
62             expect(customers).toBe.a('array');
63             expect(customers).toBe.jsonSchema(customerSchema);
64         })
65     })
66 })

```

*Exemple de jeu de test pour le module Hiboutik*

## 3. Recherche

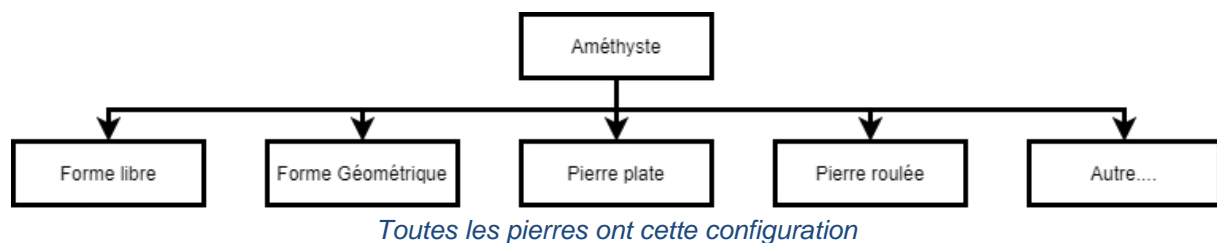
---

### 3.1. Veille technologique

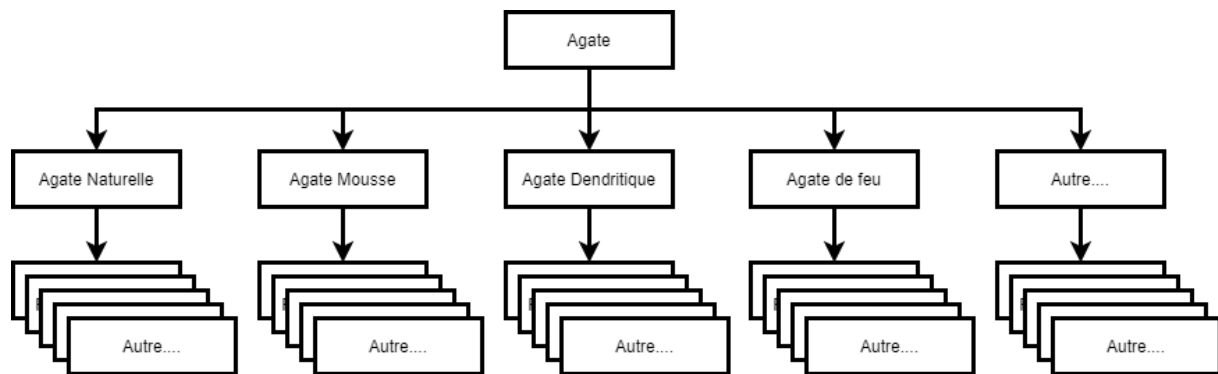
Dans le cadre du titre CDA, il était demandé d'effectuer une veille technologique sur la sécurité informatique. Cette veille a pour but de se sensibiliser aux failles présentes et de savoir comment s'y préparer lors de nos projets. Pour effectuer cette veille, j'ai mis en place un flux RSS sur la plateforme Feedly et je me suis documenté sur des sites tels que medium.com, lafermeduweb.net ou encore Owasp.

### 3.2. Situation de travail

Sur Hiboutik, chaque article présent en magasin correspond à une catégorie de pierres. Mais l'organisation adoptée par l'entreprise implique des catégories et des sous-catégories.



Dans l'illustration ci-dessus, l'Améthyste est une catégorie qui contient des catégories enfants en rapport avec les déclinaisons et les formes. Cette situation est assez simple à envisager. Il suffira de récupérer l'ID de la catégorie d'un article afin de récupérer l'ID de sa catégorie parente. Or, certaines pierres appartiennent à une « famille » de pierres.



Sur l'illustration ci-dessus, la catégorie « Agate » contient plusieurs catégories de pierres qui contiennent aussi les sous-catégories présentées précédemment.

Cette configuration en cascade est plus fastidieuse pour récupérer le nom exact de la pierre, mais elle est nécessaire pour trouver les articles rapidement durant une vente en magasin.

Il fallait donc une autre solution pour récupérer la liste des pierres. Nous avons songé à récupérer le nom de la pierre sur le libellé de l'article. Cependant, beaucoup d'articles contiennent parfois un nom abrégé. Cette solution est donc encore moins efficace que les catégories d'articles.

La solution adoptée aujourd'hui est de mettre en place le système de « marques » proposé par Hiboutik pour les articles. Nous avons attribué une marque contenant le nom de la pierre à tous les articles. C'est grâce à cela que nous pouvons lister la liste des achats d'un client.

### **3.3. Contraintes et potentielles évolutions**

#### **3.3.1. Contraintes**

L'API principale est trop structurée pour Hiboutik. Si dans le futur un autre outil de caisse était choisi en magasin, rien ne garantirait que cette application puisse disposer d'une API. Dans le cas contraire, si une API était présente, la structure des données envoyées serait sûrement différente. Une conséquente mise à jour de l'API et potentiellement de l'application mobile seraient à prévoir.

#### **3.3.2. Évolutions**

Actuellement en magasin sont disposées les pierres sur des tables avec des fiches expliquant les grandes lignes de chaque type de pierres.

L'idée serait de remplacer le contenu de ces cartes par des QR code afin qu'un client puisse le scanner pour ouvrir la fiche de la pierre en question sur l'application mobile. Cela permettrait aux clients de trouver la fiche d'une pierre plus rapidement durant leurs achats.

## 4. Conclusion

---

Au début de mon alternance, quand le contexte sanitaire le permettait, je travaillais dans l'arrière-boutique. Madame FERRARI s'occupe seule du magasin et de sa clientèle. Le temps qu'elle s'occupe de renseigner une personne, d'autres clients peuvent attendre un certain moment avant de pouvoir être pris en charge. L'intérêt de l'application mobile prend tout son intérêt à ce moment-là. Même si des fiches sont mises à disposition pour chaque pierre, il n'est pas possible d'y retranscrire toutes les informations de par leur petite taille. Grâce à cette application, le client a la possibilité de bien se documenter sur les pierres qu'il consulte. Si une pierre l'intéresse, il a juste à récupérer le nom de la pierre et à la chercher dans l'application. De plus le client peut savoir s'il a déjà fait l'acquisition de ladite pierre. L'application permet par la même occasion de mettre en favori des pierres pouvant servir de liste d'achats.

Le projet s'est déroulé uniquement en distanciel. Seules des visioconférences ont été effectuées afin de valider les différentes étapes du projet (cf. **Planning**). J'ai découvert différentes manières de développer, ainsi que Flutter, qui commence à se démocratiser. Durant ce projet, j'ai appliqué ce que j'avais appris durant mon année de bac+3.

Cette application mobile est un vrai plus pour le magasin, l'intéressement pour la lithothérapie est exponentiel. Une communauté grandit au sein de la boutique de la Semeuse de Pierre. Partager l'information d'une nouvelle application sur ses réseaux sociaux pourrait donner un réel engouement.