



# RAPPORT DE STAGE

## Concepteur Développeur Informatique

Steve MORGEAT accueillit par OTODO.

Du 11 décembre 2017 au 6 mars 2018.

Version du document : 1.7.0

Date de création : 2 décembre 2017

Date de dernière mise à jour : 15 février 2018

## SOMMAIRE

Chapitre 1 - Présentation générale .....	4
1.1 - Avant-propos .....	5
1.2 - Abstract.....	6
1.3 - Remerciements .....	7
1.4 - Présentation du stagiaire.....	8
1.5 - Démarche générale.....	9
Chapitre 2 - Cahier des charges .....	10
2.1 - Présentation de l'entreprise .....	11
2.2 - Index du site OTODO .....	12
2.3 - Attente de l'entreprise .....	13
2.4 - Technologie imposée .....	14
2.5 - Contexte de développement .....	16
Chapitre 3 - Analyse.....	17
3.1 - Le Diagramme de Cas d'Utilisation .....	18
3.1.1 - Fiche de description textuelle d'un cas d'utilisation.....	20
3.2 - Les maquettes .....	21
3.3 - Le Diagramme de Navigation .....	24
3.4 - Les Diagrammes DE SYSTEMES .....	26
3.5 - Les Diagrammes d'Activité .....	28
Chapitre 4 - Conception de la Base de Données .....	30
4.1 - Introduction .....	31
4.2 - La démarche utilisée.....	32
4.3 - Le Diagramme de Classes (UML) ou le MCL (Merise) .....	33
4.4 - Le MCD (modèle conceptuel de données) .....	36
4.5 - Le MPD (Schéma de la BD) .....	38
4.6 - MongoDB : le LDD (langage de définition des données).....	41
4.7 - MongoDB : le LMD (langage de manipulation des données) .....	43
Chapitre 5 - Conception de l'application.....	45
5.1 - Technologies utilisées.....	46
5.2 - Architecture du projet.....	55
5.3 - Les Diagrammes de Séquence Détaillés .....	57

Chapitre 6 - Développement partie front .....	59
6.1 - Développement.....	60
6.1.1 - AppComponent.html.....	61
6.1.2 - AppComponent.ts.....	62
6.1.3 - Api.service.ts.....	63
6.1.4 - Interface login .....	65
6.1.5 - Login.html.....	66
6.1.6 - Login.ts .....	67
6.1.7 - Authentification.service.ts.....	68
6.1.8 - Interface Dashboard (Home).....	69
6.1.9 - Dashboard.html .....	70
6.1.10 - Dashboard.ts.....	71
6.1.11 - Interface Tutoriaux.....	72
6.1.12 - Tuto.html .....	74
6.1.13 - Tuto.ts.....	78
6.1.14 - Api.service.ts.....	84
6.2 - les controls de saisies .....	88
Chapitre 7 - Développement partie back.....	90
7.1 - Développement.....	91
7.1.1 - Création du server .....	92
7.1.2 - Route login (contrôleur).....	94
7.1.3 - UtilisateursApi.js (entités) .....	96
7.1.4 - Route tutoAppairage (contrôleur) .....	97
7.1.5 - DaoApi.js (DAO) .....	99
Chapitre 8 - Déploiement.....	104
8.1 - Le Diagramme de Déploiement .....	105
8.2 - Le déploiement.....	107
Chapitre 9 - La sécurité .....	108
9.1 - La sécurité de l'application .....	109
Chapitre 10 - Conclusion.....	110
10.1 - Retour sur l'expérience acquise.....	111
Chapitre 11 - Annexes .....	112
11.1 - Correspondances Projet/Reac.....	113
11.2 - Glossaire/Lexique ou/et Liste de mots-clés et sigles .....	115

## ***CHAPITRE 1 - PRESENTATION GENERALE***

## 1.1 - AVANT-PROPOS



### LA DOMOTIQUE

Le mot « **domotique** » vient de « **domus** » qui signifie « **domicile** », et du suffixe « **tique** » qui fait référence à la **technique**.

La domotique ne date pas d'hier, puisqu'on en parlait déjà **dans les années 1970**. En 1974, en banlieue de Bruxelles, la maison de **Pierre SARDA** était déjà équipée à l'époque de bon nombre d'automatismes et de périphériques qui font toujours rêver même encore aujourd'hui. Son système reposait sur un ordinateur « pas plus grand qu'une petite valise », dont le rôle était de permettre de connaître l'environnement et de le commander : la lumière, le chauffage, l'audio/vidéo, la sécurité.

La domotique est là pour apporter confort, économie d'énergie et sécurité.

## 1.2 - ABSTRACT

A smart home is a residence featuring ambient intelligence technologies in order to help its dwellers in different situations of common life by trying to manage their comfort and security through the execution of actions over the effectors of the house. Detection of abnormal situations is paramount in the development of surveillance systems. These situations can be detected by the analysis of the traces resulting from audio processing and the data provided by the network of sensors installed in the smart home. For instance, detection of cries along with thuds (fall of a heavy object) in a short time interval can help to infer that the resident fell.

The problem today is that the number of languages and protocols of connected objects requires a fairly sharp data processing. Only one person cannot handle that.

So the purpose of this project and to create an API allowing distribution customers to manage some of the information.

### 1.3 - REMERCIEMENTS

Je tiens à remercier :

- Monsieur Éric DENOYER et Monsieur Éric SOUCHAUD pour m'avoir permis d'effectuer mon stage au sein leur équipe et surtout pour leur disponibilité pour répondre à mes questions.
- Monsieur Anaël DUVIGNEAU et Monsieur Benjamin FOLLET mes tuteurs de stage pour leurs patience et leurs aides.
- Monsieur Pascal BUGUET pour sa grande disponibilité, sa patience et son accompagnement tout le long de cette formation sans qui l'aventure aurait été des plus sinueuse.
- Monsieur Sébastien MALORON pour son expertise des nouvelles technologies, son enseignement tout le long de cette formation
- M2I Formation pour avoir tout mis en œuvre pour qu'on puisse suivre cette formation sans problème.
- A mes collègues et amis avec qui j'ai passé ces 6 mois de formation, pour leur aide, soutien et fou-rires passés. En particulier, à Quentin Enard mon binôme complémentaire de 6 mois et à Jessica Lassie le trinôme des 3 derniers mois.
- A Mélanie ma femme qui m'a permis de créer ce projet de reconversion. A Pauline ma fille pour son sourire qui m'a donné force et courage !
- Et bien évidemment au Fongecif IDF sans qui ce projet n'existerait pas.

## 1.4 - PRESENTATION DU STAGIAIRE

En 2012, pour la société Richter, de manière autodidacte, j'ai créé et mis en ligne leur site internet. Le domaine m'a plu. Un peu plus tard, également de manière autodidacte et, je me suis lancé dans la programmation avec l'Arduino (langage basé sur C, C++). J'étais intéressé par le monde de la domotique. J'aimais cet aspect de « bidouillage », c'est-à-dire à partir d'un vieux micro, de vieilles enceintes, un pc laissé dans un coin arriver à créer un système de commande vocale afin d'allumer/éteindre une lampe dans une autre pièce.

Après des nuits de recherche, de documentation, de vidéos tutoriels, l'aspect programmation demandait un savoir pointu et un apprentissage approfondi. Au travers de tutoriels sur le site OpenClassrooms, j'ai eu un déclic et me suis dit que je devais en faire mon métier.

C'est via un financement CIF auprès du Fongecif IDF que j'ai entrepris une formation de concepteur développeur informatique chez M2i Formation Picpus.

## 1.5 - DEMARCHE GENERALE

La démarche choisie correspond à une démarche préconisée par Pascal Roques dans son livre « UML2 - Modéliser une application web » publié chez Eyrolles.

C'est celle que j'ai suivi avec quelques aménagements, dû au choix des technologies et au fait que nous développons une API.

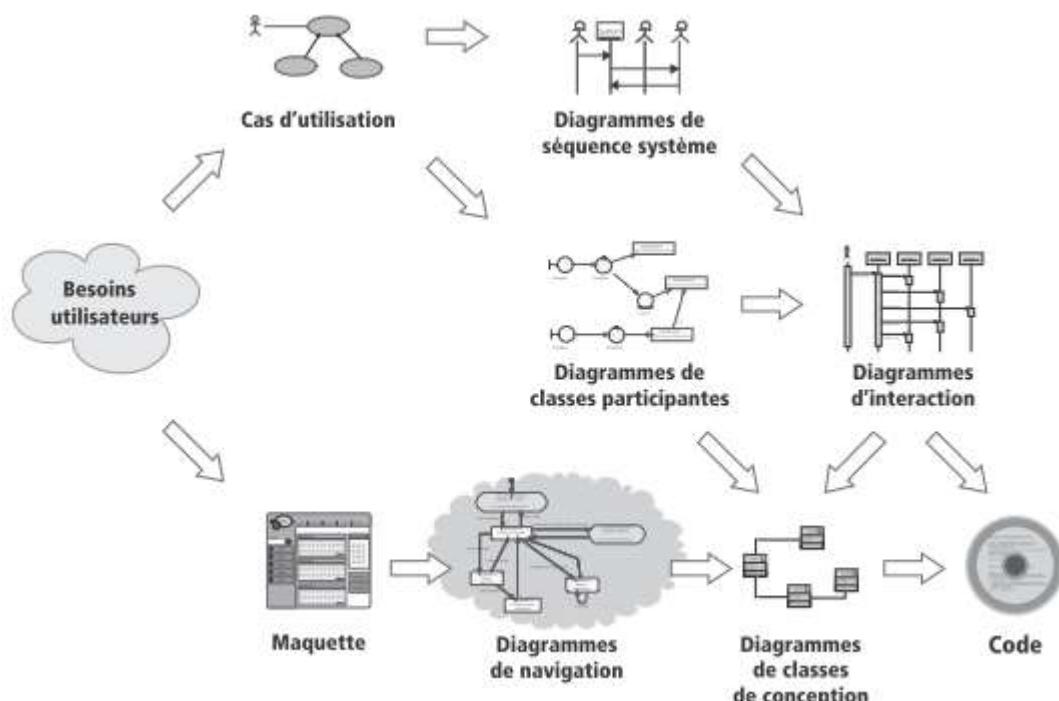


Figure 1–20 Schéma complet du processus de modélisation d'une application web

## **CHAPITRE 2 - CAHIER DES CHARGES**

## 2.1 - PRÉSENTATION DE L'ENTREPRISE

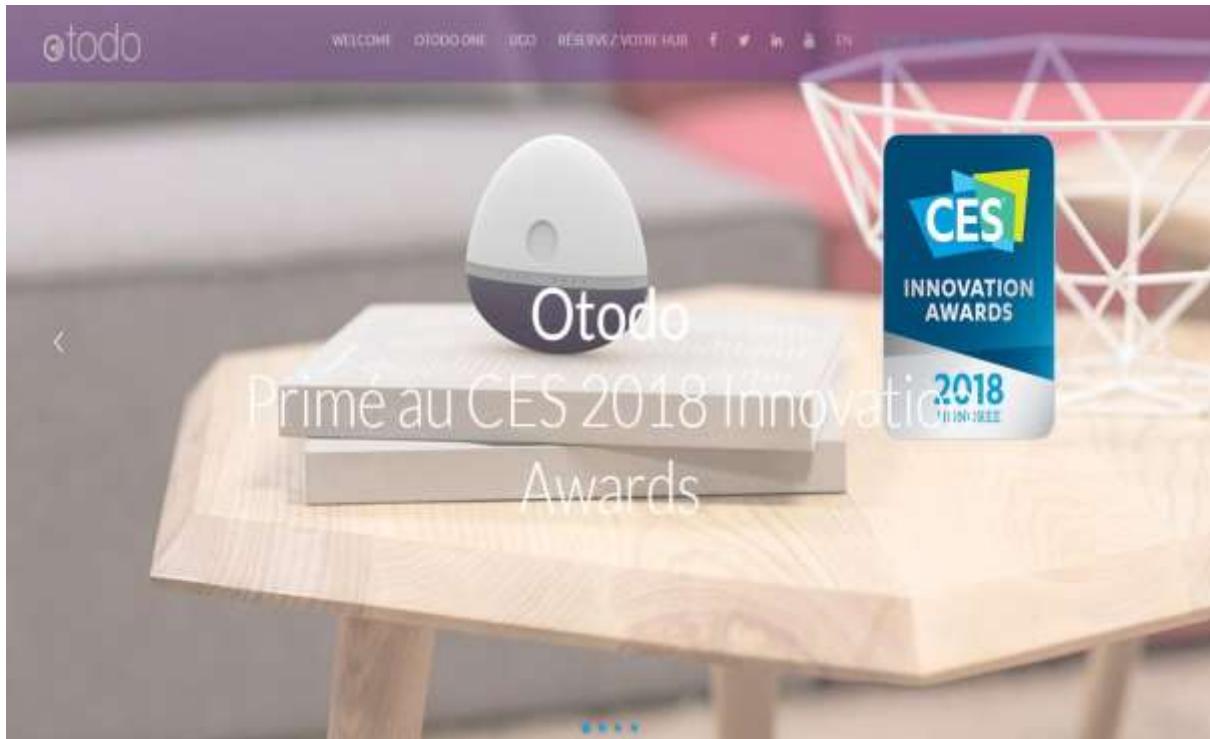
L'aventure Otodo commence en juin 2016 lorsqu'Eric Denoyer, ex directeur-général de SFR-Numéricable et passionné de technologie, se fixe pour objectif de démocratiser la Smart Home, à l'instar de la révolution de l'internet haut débit.

Pour faire décoller le marché foisonnant de la Smart Home, sur lequel de plus en plus d'entreprises du net se positionnent (Amazon Echo, Google Home, ...), Eric Denoyer est persuadé qu'il faut casser les silos et simplifier les usages. Il s'associe avec Eric Souchaud, expert du logiciel cloud pour opérateurs, qui a notamment développé des marques blanches pour de grands acteurs des télécoms. Avec Otodo, ils veulent renouer avec la mission première de la Smart Home : apporter confort, simplicité et sécurité... pour tous !

Aujourd'hui, les offres sont souvent propriétaires, incompatibles entre elles et relativement chères. Les solutions existantes s'adressent à des utilisateurs avertis ; elles n'englobent pas les équipements en petite domotique déjà largement déployés. La rupture que propose Otodo, c'est la création d'une solution universelle capable de piloter l'ensemble des objets connectés de la maison, des plus récents, équipés d'une technologie de pointe, aux plus anciens tout simplement télécommandés. Un hub interopérable, compatible avec tous types de langages numériques. Pour assurer une diffusion rapide de « la révolution Smart Home », les co-fondateurs d'Otodo sont persuadés qu'il faut passer par les opérateurs télécoms afin de proposer une offre de services « maison connectée » complémentaire à leur cœur métier.

Pour quelques euros supplémentaires par mois, les millions d'abonnés haut débit pourront piloter l'ensemble de leur maison à travers une application unique ! C'est la rencontre du monde de la technologie avec le monde grand public de l'internet haut débit. La révolution Smart Home est prête à démarrer ...

## 2.2 - INDEX DU SITE OTODO



« Je suis très heureux que notre objet compagnon Ugo soit récompensé par le CES Innovation Awards 2018. Otodo était le chaînon manquant pour universaliser la Smart Home. Nous sommes désormais prêts pour conquérir le CES de Las Vegas en janvier. »

Déclare Eric Denoyer, Président d'Otodo.

## 2.3 - ATTENTE DE L'ENTREPRISE

Créer une API (Application Programming Interface) permettant l'authentification, la gestion des rôles des utilisateurs authentifiés, le CRUD sur des collections de la base de données MongoDB (create, read, update et delete).

Il y a au 3 types d'utilisateurs :

- Administrateur OTODO qui a un accès à toutes les fonctionnalités.
- Client Fournisseur (type SFR, Chacon, etc..) qui pourra créer des sous-clients, créer des tutos, opérer sur les données des clients finaux.
- Et accessoirement un sous-client (type un opérateur chez un fournisseur) qui aura accès aux données de son parc. C'est-à-dire qu'il n'aura pas de vue sur le parc d'un autre sous-client.

Créer en parallèle une interface venant effectuer des requêtes vers cette API.

Ces 2 applications se veulent indépendante l'une de l'autre.

## 2.4 - TECHNOLOGIE IMPOSEE

Le projet en partie Back sera une API Rest développé en **Node.js** car ils veulent que l'API soit utilisable par plusieurs workflows. Ils souhaitent utiliser **Mongoose**, un **ODM** de Node.js qui permet structurer les données avec **MongoDB** et le module Express pour sa flexibilité et sa robustesse. Pour la partie front, la technologie n'est pas imposée. Il y avait un choix entre **Angular** et **React.js**. Donc je me tourne vers le framework **Angular** de Google pour son aspect Modèle-Vue-Contrôleur, sa gestion des guards, et son nombre incalculable de modules.

La mise en forme est gérée principalement par **Bootstrap**, une librairie de HTML/CSS/JS, ainsi que des feuilles de styles complémentaires propre au projet.

Les avantages du développement d'applications en couche sont :

- ✓ **Multiplateforme** : Le server back peut être utilisé par plusieurs client front via des requêtes Http.
- ✓ **Sécurité** : plusieurs couches permettent d'avoir plusieurs couches de contrôles.
- ✓ **Maintenabilité** du code du server back : il n'y a qu'un seul code à modifier en cas de mise à jour.

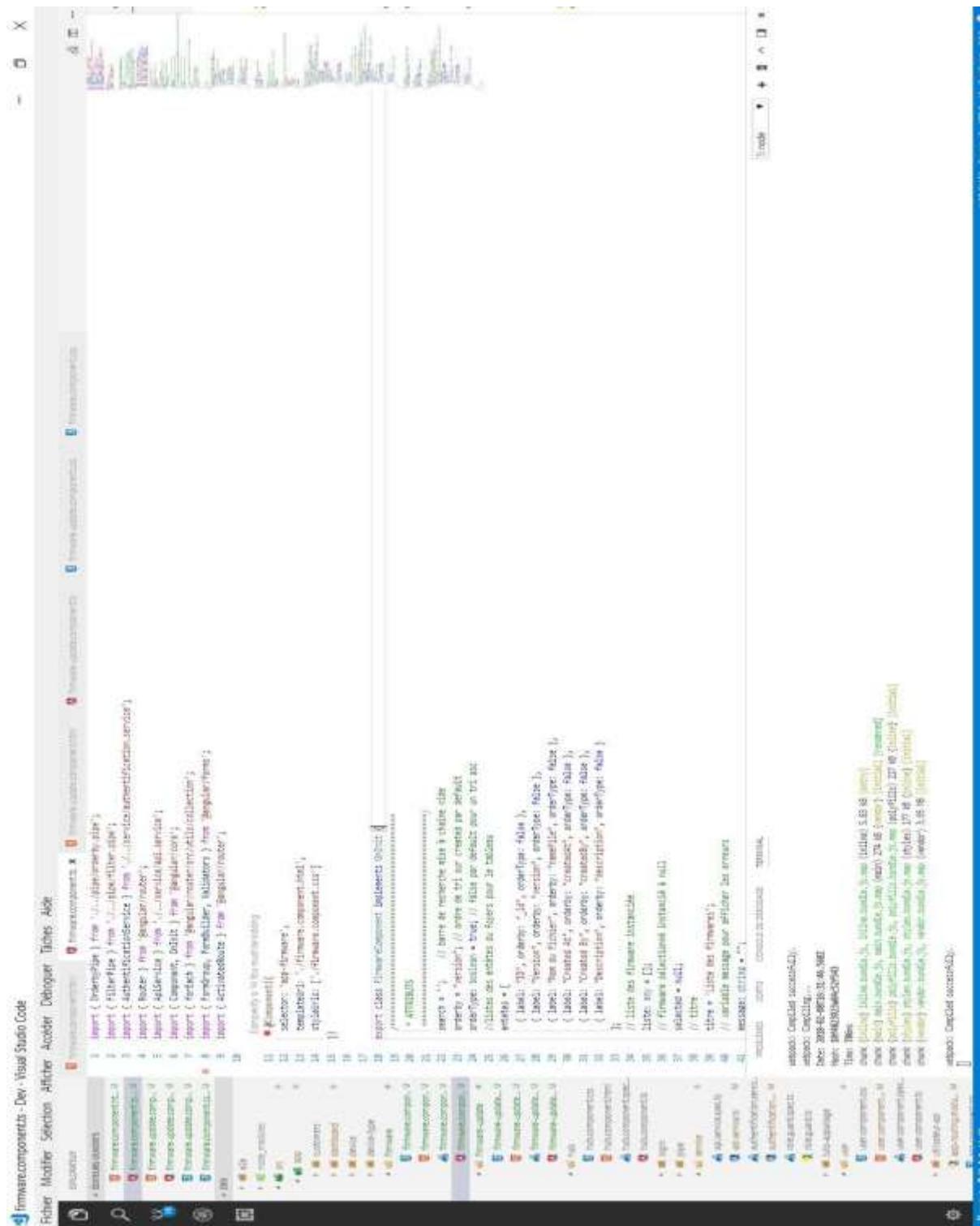
Les Inconvénients :

- Le debugge peut-être parfois difficile.
- Le développement est plus long.

Toute la gestion du projet s'est faite grâce à **Git**, un outil de gestion de version du code source décentralisé, permettant une cohésion de travail exemplaire, ainsi qu'une traçabilité du code source grâce à un historique permettant de revenir très facilement sur une version antérieure.

J'ai pris le pas d'utiliser l'éditeur de code Visual Studio Code. Très léger et étant open source, il offre une bibliothèque de plug-in importante et qui augmente la productivité du développeur de façon significative.

Exemple :



Écran de Visual Studio Code montrant un fichier Java component configuration. Le code est un JSON avec les suivantes sections :

```

{
    "name": "mycomponent",
    "version": "1.0.0",
    "description": "My first component",
    "dependencies": {
        "java": [
            "com.google.guava:guava:18.0"
        ]
    },
    "outputs": [
        {
            "name": "output1",
            "path": "output/output1"
        }
    ],
    "processors": [
        {
            "name": "processor1",
            "path": "processor/processor1"
        }
    ],
    "run": {
        "mainClass": "com.mycomponent.MyMainClass",
        "args": [
            "arg1"
        ]
    },
    "configuration": {
        "processors": [
            {
                "id": "processor1",
                "order": 1,
                "processor": "MyProcessor"
            }
        ],
        "environments": [
            {
                "id": "environment1",
                "order": 1,
                "environment": "MyEnvironment"
            }
        ],
        "executors": [
            {
                "id": "executor1",
                "order": 1,
                "executor": "MyExecutor"
            }
        ],
        "registries": [
            {
                "id": "registry1",
                "order": 1,
                "registry": "MyRegistry"
            }
        ],
        "repositories": [
            {
                "id": "repository1",
                "order": 1,
                "repository": "MyRepository"
            }
        ],
        "services": [
            {
                "id": "service1",
                "order": 1,
                "service": "MyService"
            }
        ],
        "listeners": [
            {
                "id": "listener1",
                "order": 1,
                "listener": "MyListener"
            }
        ]
    },
    "imports": [
        {
            "name": "import1",
            "path": "import/import1"
        }
    ],
    "script": [
        {
            "id": "script1",
            "order": 1,
            "script": "MyScript"
        }
    ],
    "api": [
        {
            "name": "api1",
            "path": "api/api1"
        }
    ],
    "dashboards": [
        {
            "name": "dashboard1",
            "path": "dashboard/dashboard1"
        }
    ]
}

```

## 2.5 - CONTEXTE DE DEVELOPPEMENT

### ENVIRONNEMENT DE DEVELOPPEMENT

<b>WINDOWS 10</b>	Pour l'entièrre partie du projet
<b>VISUAL STUDIO CODE</b>	Pour la partie Programmation, un IDE idéale pour la gestion de projet JS
<b>GIT</b>	Pour la partie sauvegarde et la gestion des versions et historiques du projet
<b>STARUML</b>	Pour la création de diagramme simple
<b>POWERAMC</b>	Pour la modélisation des traitements informatiques et des bases de données associées
<b>POSTMAN</b>	Pour effectuer des requêtes sur les url de l'APIRest en get/post/put/delete.

## **CHAPITRE 3 - ANALYSE**

### 3.1 - LE DIAGRAMME DE CAS D'UTILISATION

*Recueillir, analyser et organiser les besoins, voilà l'objectif d'un Diagramme de Cas d'Utilisation.*

Un **DCU** est un moyen simple d'exprimer le besoin d'un utilisateur, en illustrant les fonctionnalités de l'application dont il aura besoin, via des schémas très compréhensibles.

Ces derniers débutent généralement par un diagramme de cas d'utilisation, qui est un diagramme **UML** qui sert à donner une vision simple et globale du comportement d'une application ou d'un logiciel.

J'ai d'abord défini les différents acteurs représentant les utilisateurs intervenant sur le système, puis j'en ai déduit les différents **rôles** qui leur seront attribués.

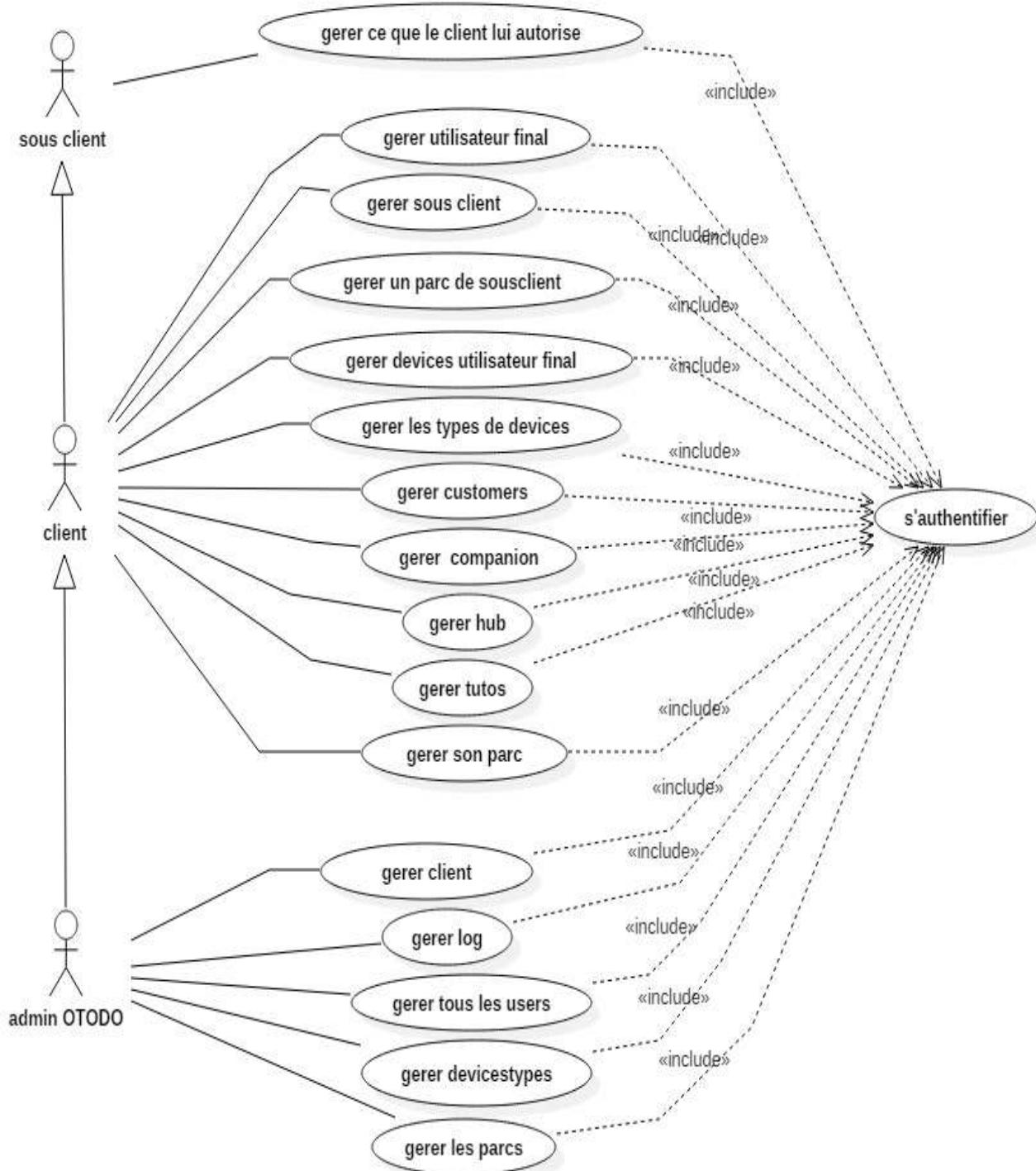
Ce diagramme met en avant 3 acteurs :

- L'administrateur OTODO (Admin)
- Le client (opérateur principale chez SFR, Chacon...etc)
- Le sous-client (opérateur chez un client qui aura certains rôles)

Voici une représentation simple d'un DCU basé qui représente tous les acteurs et leurs cas d'utilisation de l'API

OTODO 2018

Diagramme de cas d'utilisation



### 3.1.1 - Fiche de description textuelle d'un cas d'utilisation

La fiche de description textuelle d'un cas d'utilisation est très utile pour clarifier le déroulement et décrire les actions qui seront réalisées par une fonctionnalité.

La fiche ci-dessous vous décrit le scénario nominal de « modifier un customer » pour un Admin (la partie modifiée car « gérer customers » contient tout le CRUD).

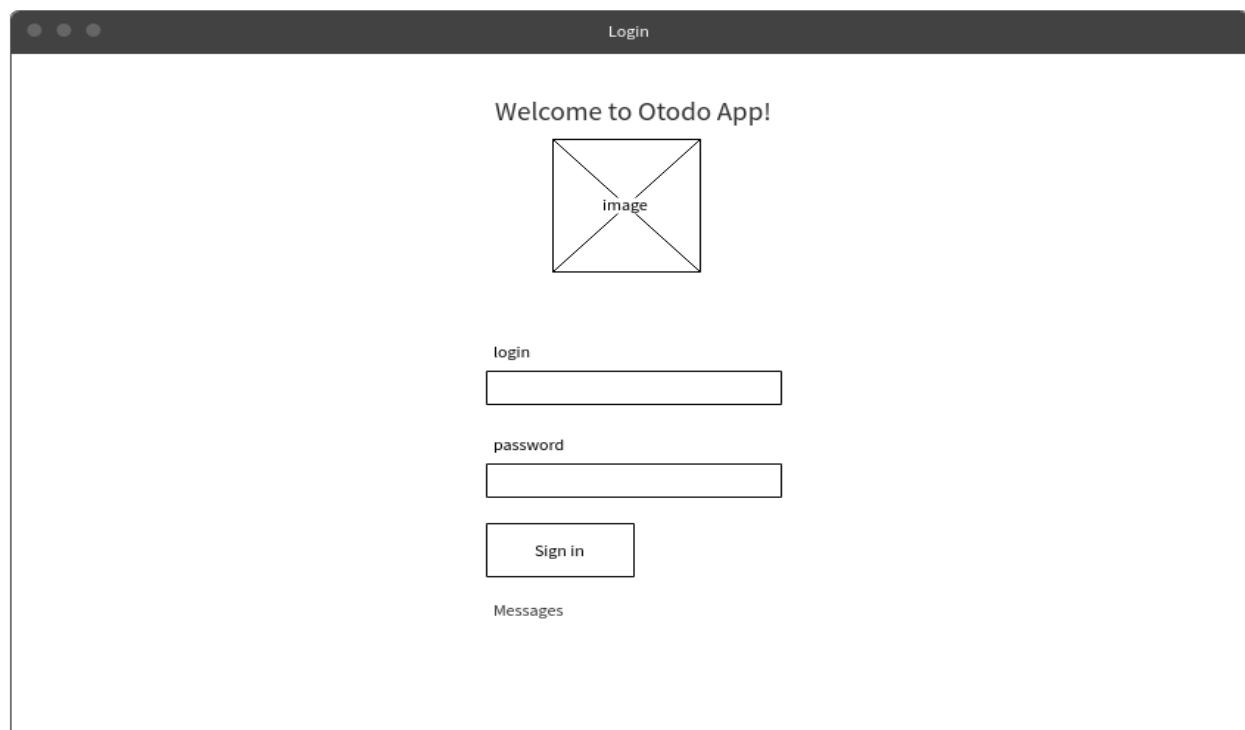
ETAPES	DESCRIPTION
<b>Identification du CU</b>	Titre : modifier un customer Résumé : une action du CRUD sur la collection MongoDB « customer » Acteurs : Administrateurs, client Date de création : 02/01/2018 Version : 1.0 Auteur : Steve Morgeat
<b>Préconditions</b>	L'utilisateur doit s'authentifier et avoir les droits requis Avoir les droits de faire cette opération Le server back end est disponible et La base de données est disponible
<b>Scenario nominal</b>	1 – Cliquer sur la navigation : « customers » 2 – Sélectionner un customer dans la liste 3 – Modifier des valeurs 4 – Valider le formulaire 5 – Afficher le message de validation
<b>Scenarii d'erreurs du cas nominal</b>	1 – Un ou plusieurs champs du formulaire vide 1 – Erreur lors de l'envoi des données en BD 1 – Erreur de l'api ou de l'application 1 – Droits requis absents 3 – Afficher le message d'erreur correspondant
<b>Scenarii alternatif</b>	1 – Cliquer sur la navigation : « customers » 2 – Sélectionner un customer dans la liste 3 – Supprimer un customer sélectionné
<b>Post-conditions</b>	Les informations ont été persistées en base de données. Un message confirme la persistance.
<b>IHM (interface homme-machine)</b>	Liste + Formulaire

### 3.2 - LES MAQUETTES

Pour les maquettes de mes IHM, j'ai utilisé le site web <https://mockflow.com/>. Un site simple, efficace et rapide qui renvoie un téléchargeable des maquettes. Gratuit jusqu'à trois maquettes

Voici la suite de trois maquettes qui montrera le déroulement de la navigation de l'utilisateur :

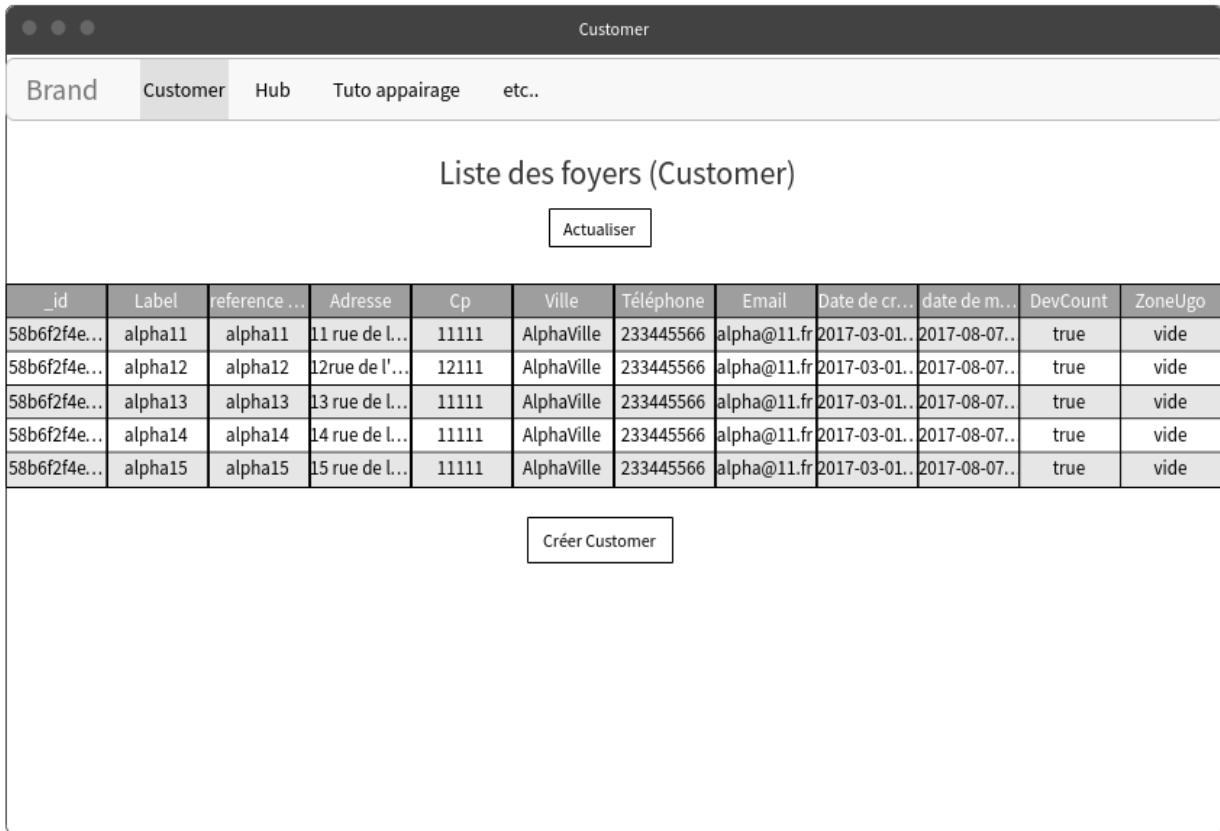
- Maquette de l'entrée du site, la page de « login » :



Maquette de la page accueil, la page du « dashboard » :



- Maquette de la page des foyers, la page du « customer » :



The screenshot shows a web-based application interface titled "Customer". At the top, there is a navigation bar with tabs: "Brand", "Customer" (which is selected and highlighted in grey), "Hub", "Tuto appairage", and "etc..". Below the navigation bar, the main content area has a title "Liste des foyers (Customer)" and a "Actualiser" button. A table displays 15 rows of data, each representing a customer record with columns for: \_id, Label, reference ..., Adresse, Cp, Ville, Téléphone, Email, Date de cr..., date de m..., DevCount, and ZoneUgo. The data in the table is as follows:

_id	Label	reference ...	Adresse	Cp	Ville	Téléphone	Email	Date de cr...	date de m...	DevCount	ZoneUgo
58b6f2f4e...	alpha11	alpha11	11 rue de l...	11111	AlphaVille	233445566	alpha@11.fr	2017-03-01..	2017-08-07..	true	vide
58b6f2f4e...	alpha12	alpha12	12rue de l'...	12111	AlphaVille	233445566	alpha@11.fr	2017-03-01..	2017-08-07..	true	vide
58b6f2f4e...	alpha13	alpha13	13 rue de l...	11111	AlphaVille	233445566	alpha@11.fr	2017-03-01..	2017-08-07..	true	vide
58b6f2f4e...	alpha14	alpha14	14 rue de l...	11111	AlphaVille	233445566	alpha@11.fr	2017-03-01..	2017-08-07..	true	vide
58b6f2f4e...	alpha15	alpha15	15 rue de l...	11111	AlphaVille	233445566	alpha@11.fr	2017-03-01..	2017-08-07..	true	vide

At the bottom center of the table area is a "Créer Customer" button.

### 3.3 - LE DIAGRAMME DE NAVIGATION

Un diagramme de navigation sert à représenter le cheminement de l'application entre les différentes vues.

Le début de la navigation est représenté par le rond noir. Les transitions sont représentées par une flèche qui pointe vers un état, comme celle qui part du début de la navigation pour arriver à la page de login.

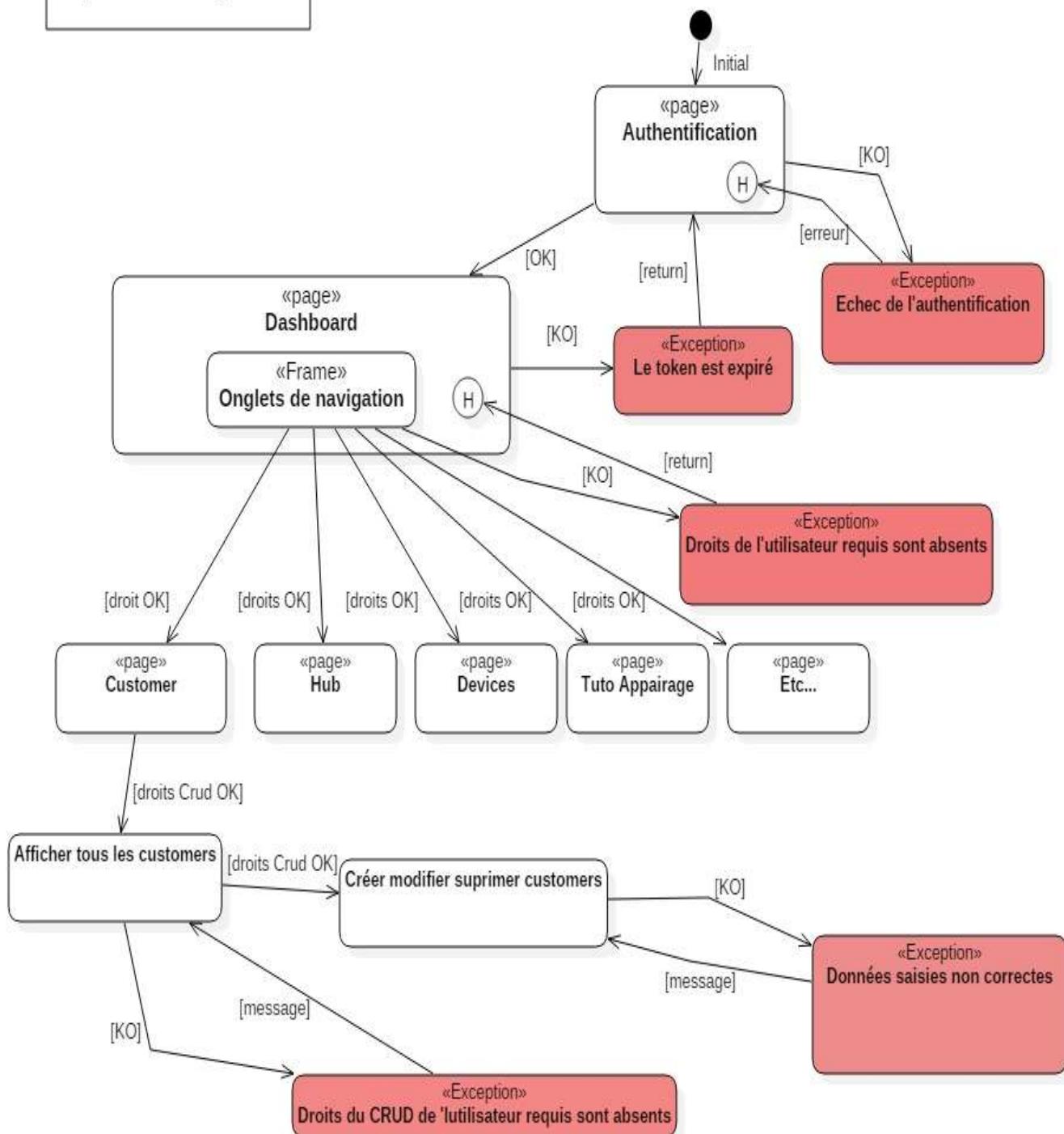
Un état est un écran, une page, qui est représenté par un rectangle avec les coins arrondis.

L'utilisateur qui arrive sur la page d'accueil, sur laquelle est présent un menu. Il clique sur le bouton du menu « customer », qui contient le listing des customers déjà ajoutées, ainsi que le bouton de création d'un customer. Il clique ensuite sur ce bouton de création pour arriver au formulaire.

Ce schéma reprend le cas d'utilisation de la « gestion des customer ». Cas relativement similaire à la gestion des hubs ou des devices.

L'utilisateur devra être authentifié. Il faudra également que le token émis lors de son authentification ne soit pas expiré lorsqu'il sera renvoyé au moment d'une requête vers l'Api Node.js et que les droits stockés dans le token lui autorisent les navigations voulu et lui permettent d'effectuer les tâches voulues.

API otodo 2018  
Diagramme de navigations



Mots clés : page, frame, exception, transition, historique.

### 3.4 - LES DIAGRAMMES DE SYSTEMES

**Le Diagramme de Séquence Système décrit le comportement du système qui découle d'un Cas d'Utilisation précédent.**

Il formalise les interactions entre les acteurs et le système ou les classes, grâce à des messages, avec un point de vue chronologique et spatial, le temps étant représenté à la verticale, et l'espace à l'horizontal. Il représente généralement deux objets : l'**utilisateur du système**, et le **système**.

Chaque objet est représenté par un rectangle, sous lequel se dessine une ligne en pointillés, qui descend vers le bas depuis son centre. On appelle cela **une ligne de vie**.

Les messages sont représentés par des flèches horizontales, pleines ou en pointillés, de l'émetteur vers le destinataire.

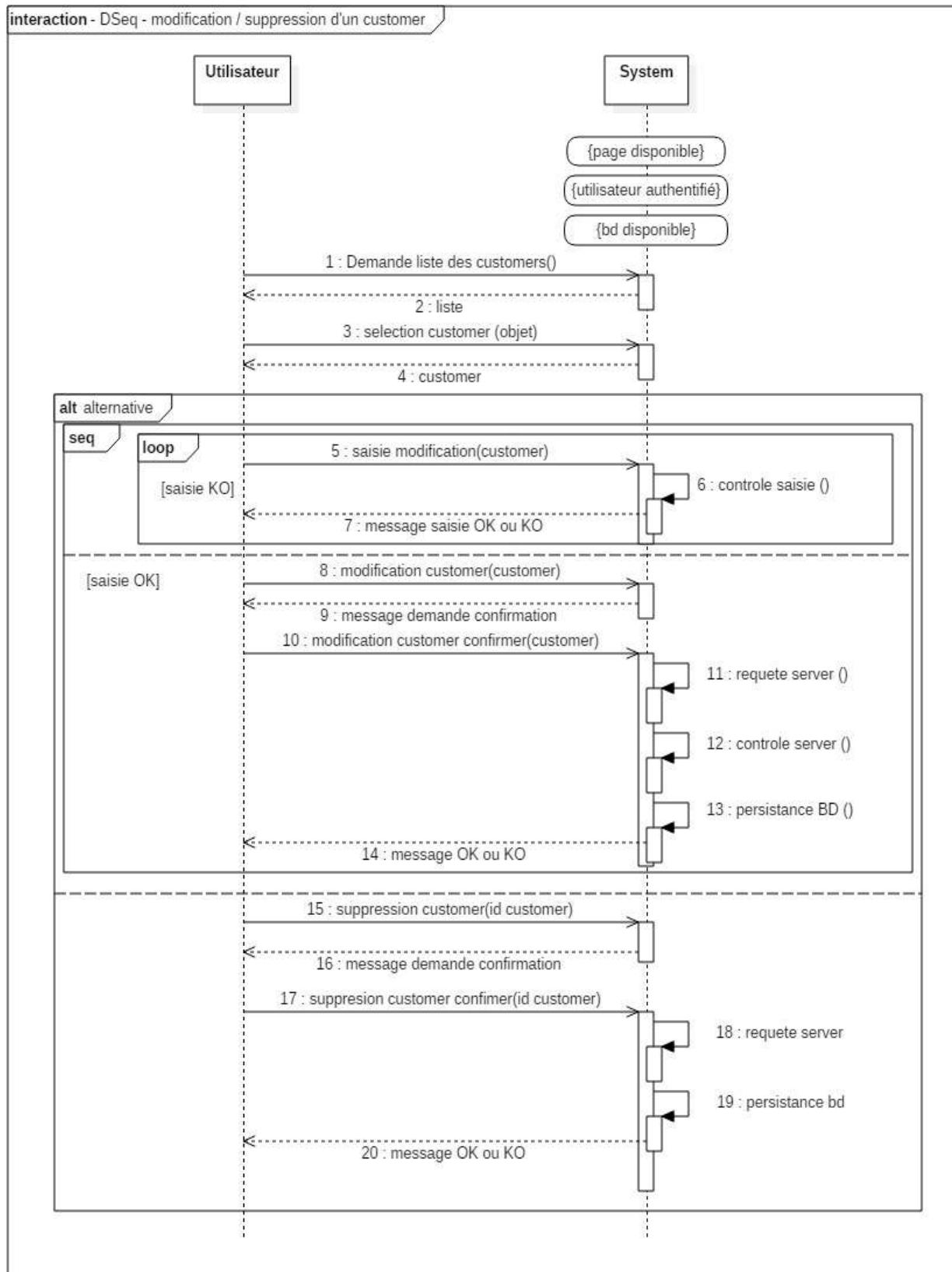
Il existe 5 types de message : le **CALL** (synchrone), le **SEND** (asynchrone), le **RETURN** (synchrone), le **CREATE** et enfin le **DESTROY**, ce qui permet de bien définir chaque message.

Les messages **CALL**, représentés par une flèche pleine et fermée, pointe vers une activation, représentée par un rectangle verticale et fin, positionnée sur la ligne de vie.

Les messages **SEND** sont eux représentés par une flèche pleine et ouverte, ils pointent généralement vers la ligne de vie d'un acteur.

Les messages **RETURN** sont tout simplement les réponses de messages **CALL**, représentés par une flèche en pointillés et ouverte.

Les messages **CREATE** et **DESTROY** sont représentés par une flèche pleine et fermée, la ligne de vie d'un **DESTROY** se terminant par une croix. Mots clés : objet, ligne de vie, message (synchrone, asynchrone, retour), paramètre de message, guard, alternative, séquences.



Mots clés : Objet, ligne de vie, message (synchrone, asynchrone, retour), paramètre de message, guard, alternative, séquences.

### 3.5 - LES DIAGRAMMES D'ACTIVITE

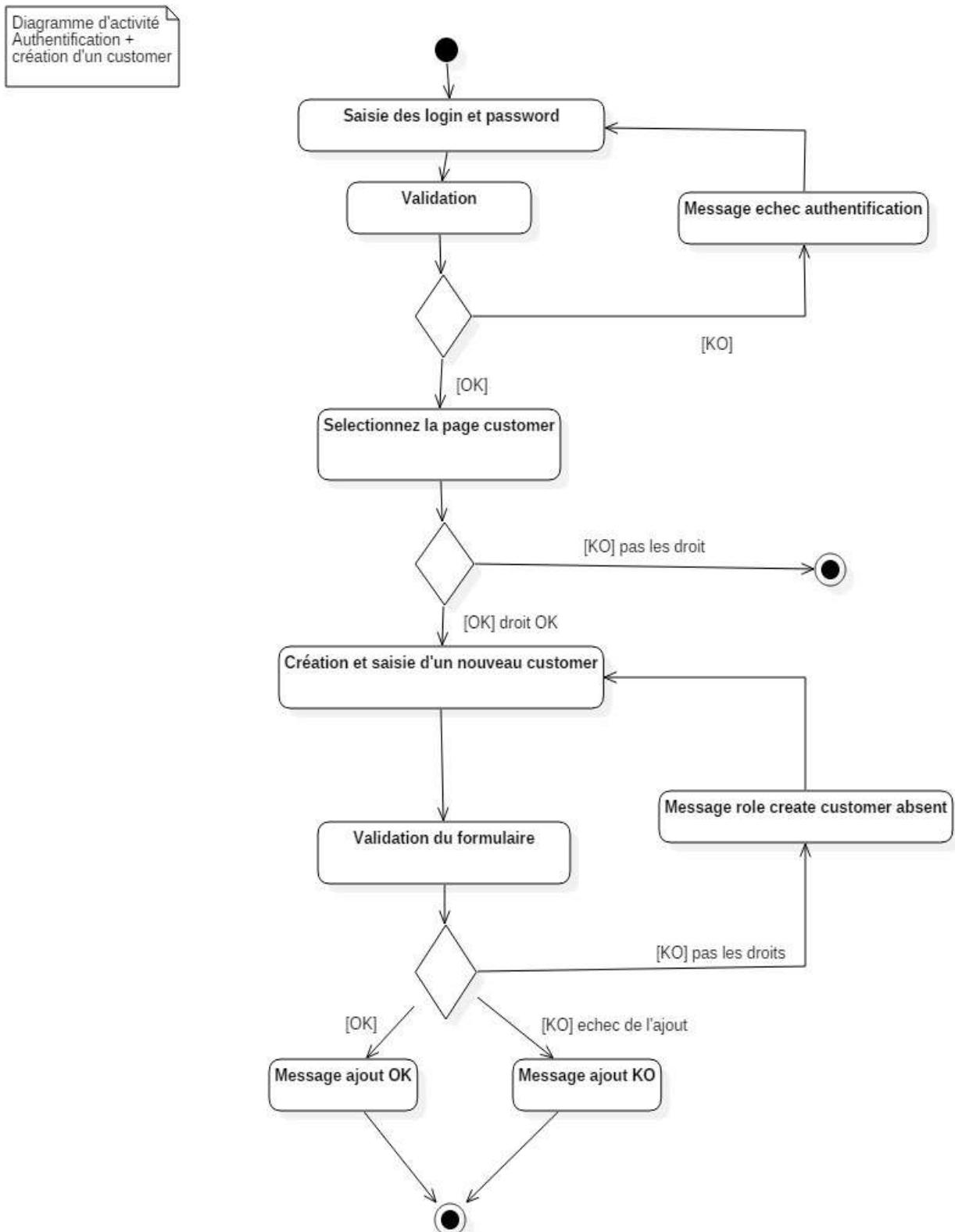
Le diagramme d'activité représente graphiquement le déroulé d'un cas d'utilisation du point de vue de l'utilisateur.

C'est l'observation de l'utilisateur, si c'est possible, effectuant une tâche qui permet de le construire.

Les composants essentiels du DAC sont les **actions** et les **transitions**. Il nous montre des **cas alternatifs** (un losange représente une condition) et donc des transitions avec des gardes (oui/non, ok/ko) en fonction des résultats des tests et des chemins parallèles. Il permet de faire la synthèse des différents scénarios du cas d'utilisation.

Dans le diagramme suivant, nous pouvons remarquer que l'utilisateur doit être connecté afin de poursuivre sa navigation. Ensuite, il doit disposer de droits afin d'effectuer certaines tâches. En l'occurrence ici un exemple de création de « customer » mais cela pourrait être le même schéma pour la création d'un « tutoriel d'appairage » ou d'un « devices ».

Le pseudo état final se finit soit par la lecture d'un message du système qui confirme l'ajout ou non du « customer », soit par le non droit d'accéder à la page des « customers ».



**Mots clés :** Pseudo état initial, pseudo état final, activité, transitions, garde.

## ***CHAPITRE 4 - CONCEPTION DE LA BASE DE DONNEES***

## 4.1 - INTRODUCTION

La base de données utilisé pour ce projet était une base de données NoSql (MongoDB) et était déjà existante, mais j'ai pu concevoir 3 collections (tables en SQL) pour ce projet.

L'entreprise a vocation à viser une volumétrie de données conséquentes (via le partenariat avec l'un des plus grands fabricants de box au monde). Le projet étant au stade du développement et donc dans le besoin de flexibilité, l'utilisation d'une base NoSQL apparait comme un compromis idéal.

Il est essentiel de garder à l'esprit que NoSQL apporte une réponse à des besoins bien spécifiques. Dit autrement, il est nécessaire d'avoir identifié au préalable la nécessité d'utiliser cette technologie dans vos services. Bien que le réflexe premier soit de penser aux performances sur des gros volumes de données ou des données faiblement structurées. Certaines utilisations peuvent se justifier, par exemple les bases orientées colonnes faciliteront l'évolution du "schéma" des données et donc vous orienteront vers une maintenance plus intelligente, plus agile et donc plus durable de vos applications.

Lors de mon analyse de la base de données existante, j'ai pu constater qu'elle avait tendance à s'orienté vers une structure relationnelle. Je tenterai de vous l'expliquer au cours des points suivant de ce chapitre.

## 4.2 - LA DEMARCHE UTILISEE

L'intérêt de la démarche utilisée, décrite ci-dessous, est d'obtenir un modèle de données normalisé au niveau de la 3ème forme normale de Boyce Codd.

La normalisation a pour objectif d'obtenir des données cohérentes par rapport au réel, d'éliminer les redondances dans la base de données ainsi que les anomalies de mise à jour.

Pour aboutir au « Modèle Conceptuel des Données » :

- Etablir le dictionnaire des données,
  - Faire la matrice des dépendances fonctionnelles,
  - Faire le graphe des dépendances fonctionnelles (GDF),
  - Réaliser le diagramme de classes.
- 
- Ou effectuer de la rétro-ingénierie d'une base existante.

Etant donné que la base est déjà existante, je réalisera la modélisation à partir du schéma de la base.

Pour l'élaboration du Modèle Physique des Données (MPD) :

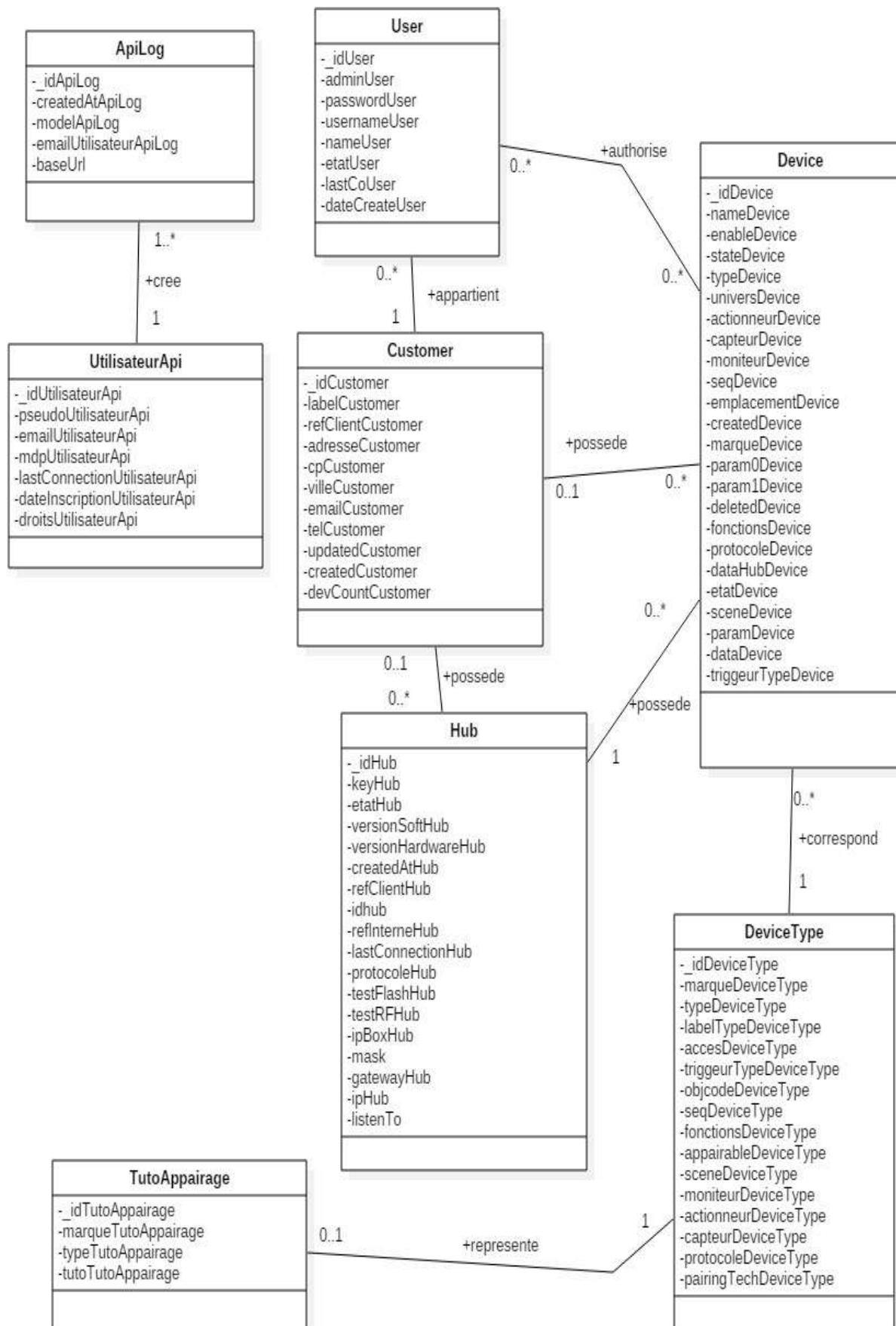
- Je l'ai réalisé à partir d'un outil nommé « dbSchema ».

Pour la création de la base de données :

- A l'aide de la commande mongorestore, j'ai réalisé une copie de la base existante pour mes tests et mes ajouts de collections en développement local.  
Puis par la suite, je me suis greffé au base existante selon les environnements (développement, homologation, production, soit 3 bases de données différentes).

#### **4.3 - LE DIAGRAMME DE CLASSES (UML) OU LE MCL (MERISE)**

Le diagramme de classes est un diagramme statique. Il permet de fournir une représentation abstraite des objets du système étudié. Je l'ai réalisé en effectuant de la rétro-ingénierie de la base de données existante. En analysant les liaisons entre les collections (tables en SQL), j'ai pu réaliser les associations entre les classes de ce diagramme.



**Mots clés :** nom de classe, attributs (encapsulation), opérations (méthode), associations, visibilités (publics, protégés, de niveau paquetage ou privés).

Liaisons :

- Un user appartient à un customer (foyer), un customer comporte zéro ou \*n user.
- Un user est autorisé par zéro ou \*n device, un device autorise zéro ou \*n user  
(Multiplicité \*n de chaque côté)
- Un customer possède zéro ou \*n device, un device appartient à zéro ou un customer.
- Un customer possède zéro ou \*n hub, un hub appartient à zéro ou un customer.
- Un device appartient à un hub, un hub possède zéro ou \*n device  
(Le device est créé lors de son appairage au hub)
- Un device correspond à un deviceType, un deviceType correspond à zéro ou \*n device.
- Un deviceType est représenté par zéro ou un tutoAppairage, un tutoAppairage représente un deviceType.

## 4.4 - LE MCD (MODELE CONCEPTUEL DE DONNEES)

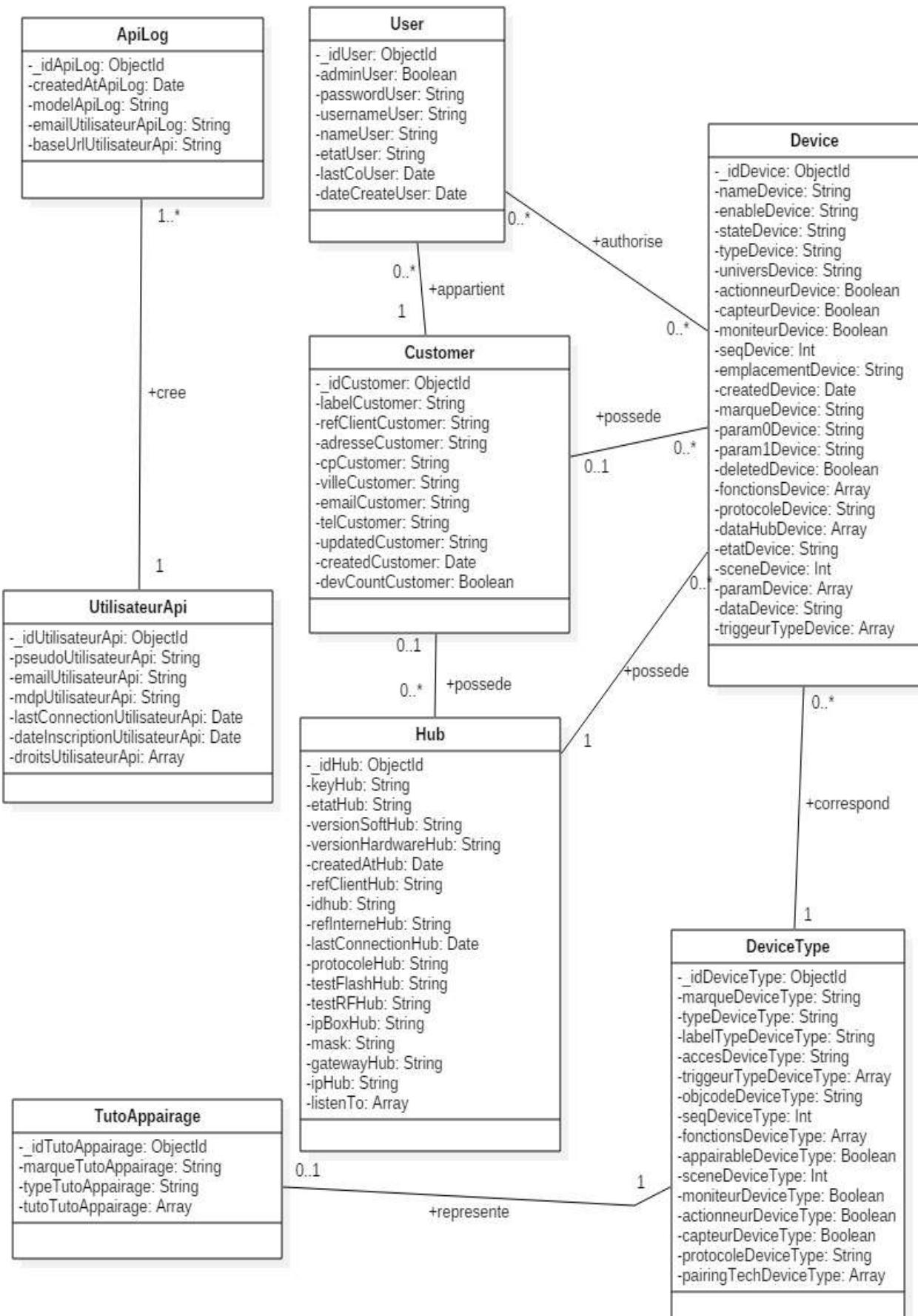
La modélisation est une étape fondamentale de la conception de la BD dans la mesure où, d'une part, on y détermine le contenu de la BD et, d'autre part, on y définit la nature des relations entre les concepts principaux.

Les éléments de base du modèle Entité-Relation ou Entité Association sont les suivants :

- Les Entités : objet pouvant être identifié distinctement
- Les attributs : caractéristiques ou propriétés des entités
- Type de relation : cardinalités ou le nombre de participation d'une entité à une relation

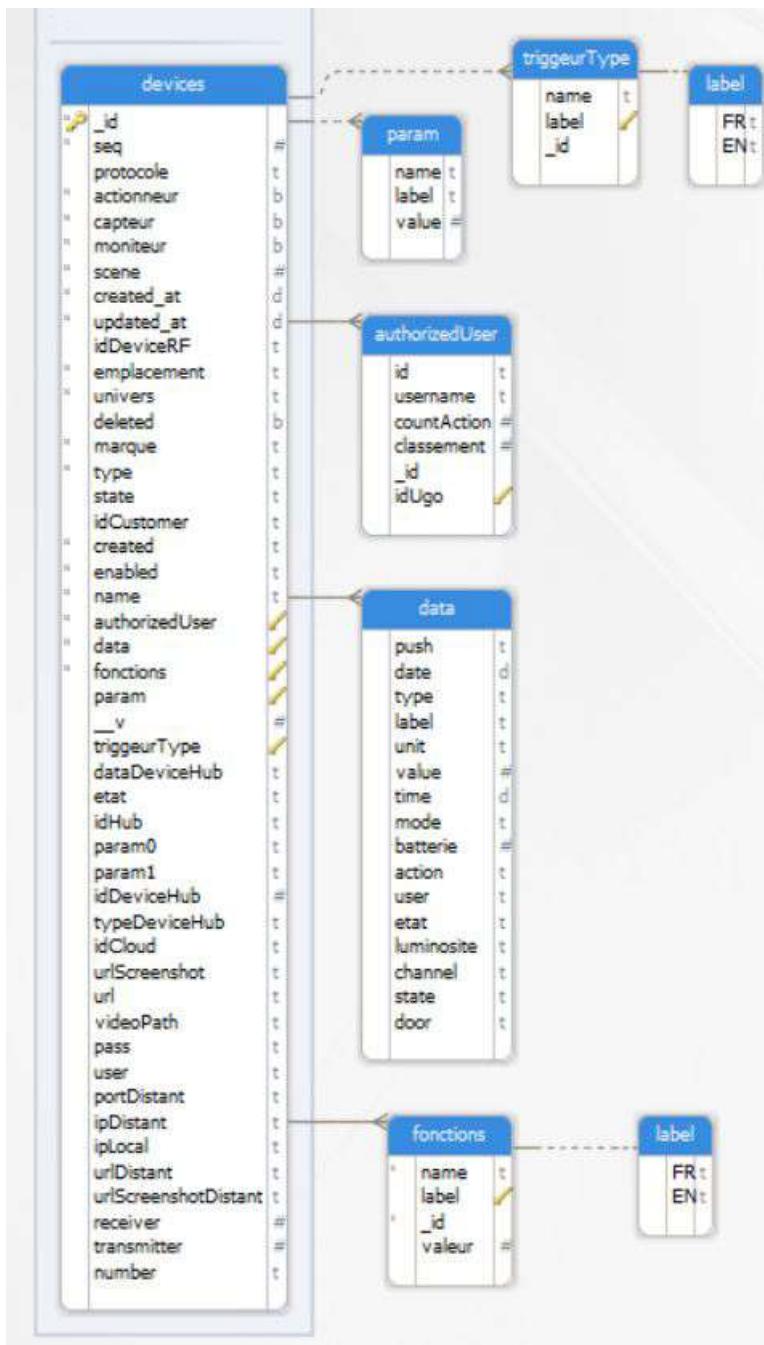
Nous réalisons le recensement des éléments cités ci-dessus à partir d'une analyse de la base de données existante, et les présentons sous forme d'un schéma conceptuel réalisé avec StarUML.

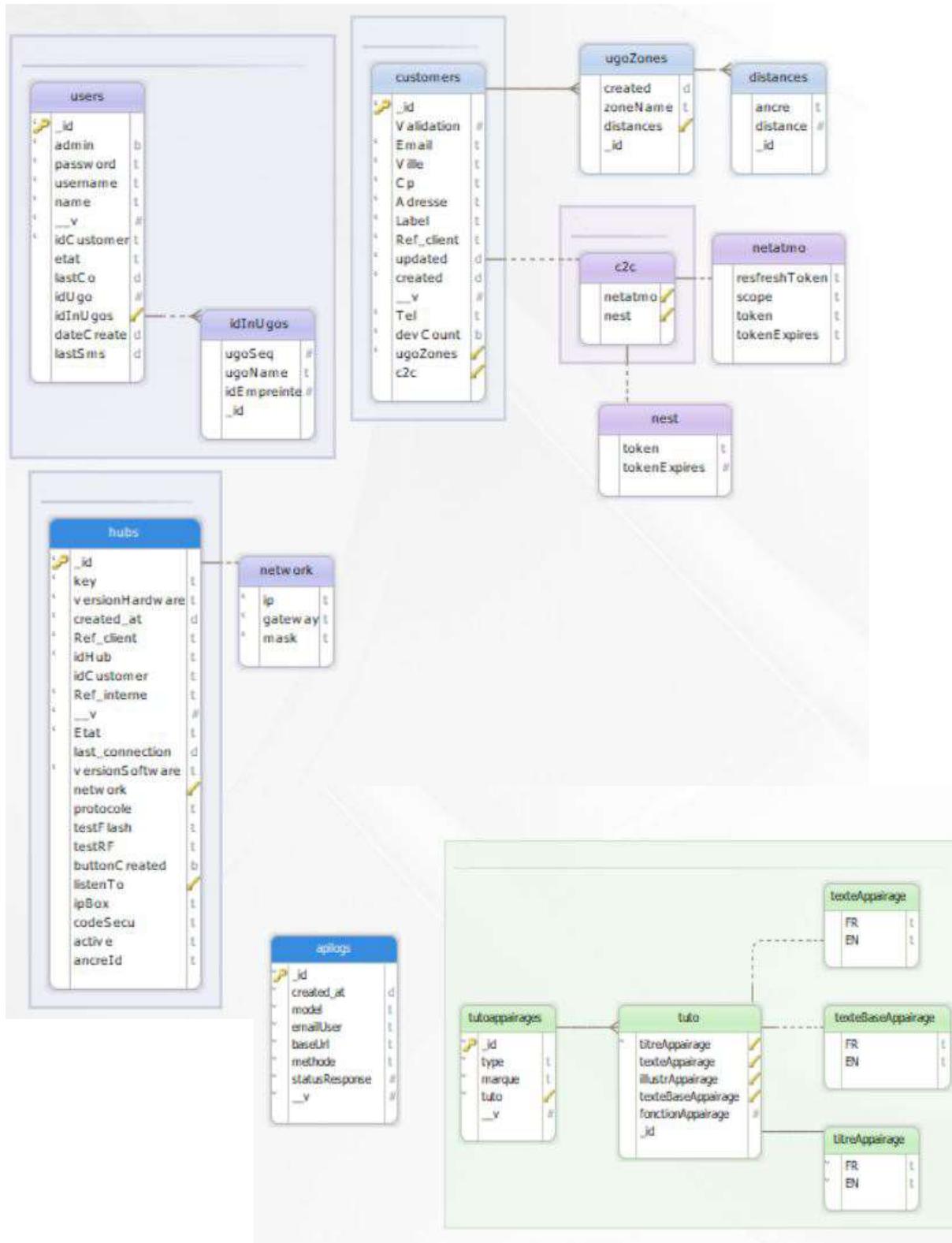
- La Relation [user - device] est une relation de type n...m : un user peut être autorisé par n devices, et un device autorise m user.
- La Relation [user - customer] est une relation de type 0...n : un user appartient à un customer, et un customer peut posséder n user.
- La Relation [device - customer] est une relation de type 0...n : un device appartient à un customer, et un customer peut posséder n devices.
- La Relation [hub - customer] est une relation de type 0...n : un hub appartient à un customer, et un customer peut posséder n hubs.
- La Relation [device - hub] est une relation de type 1...n : un device appartient à un hub, et un hub peut posséder plusieurs device.
- La Relation [device - deviceType] est une relation de type 1...n : un device correspond à un deviceType, et un deviceType peut correspondre n device.
- La Relation [deviceType - tutoApparige] est une relation de type 0...1 : un deviceType peut être représenté par un tutoApparige, et un tutoApparige représente un deviceType.
- La Relation [utilisateurApi - apiLog] est une relation de type 0...1 : un utilisateurApi peut créer un apiLog, et un apiLog est créé par un utilisateurApi.

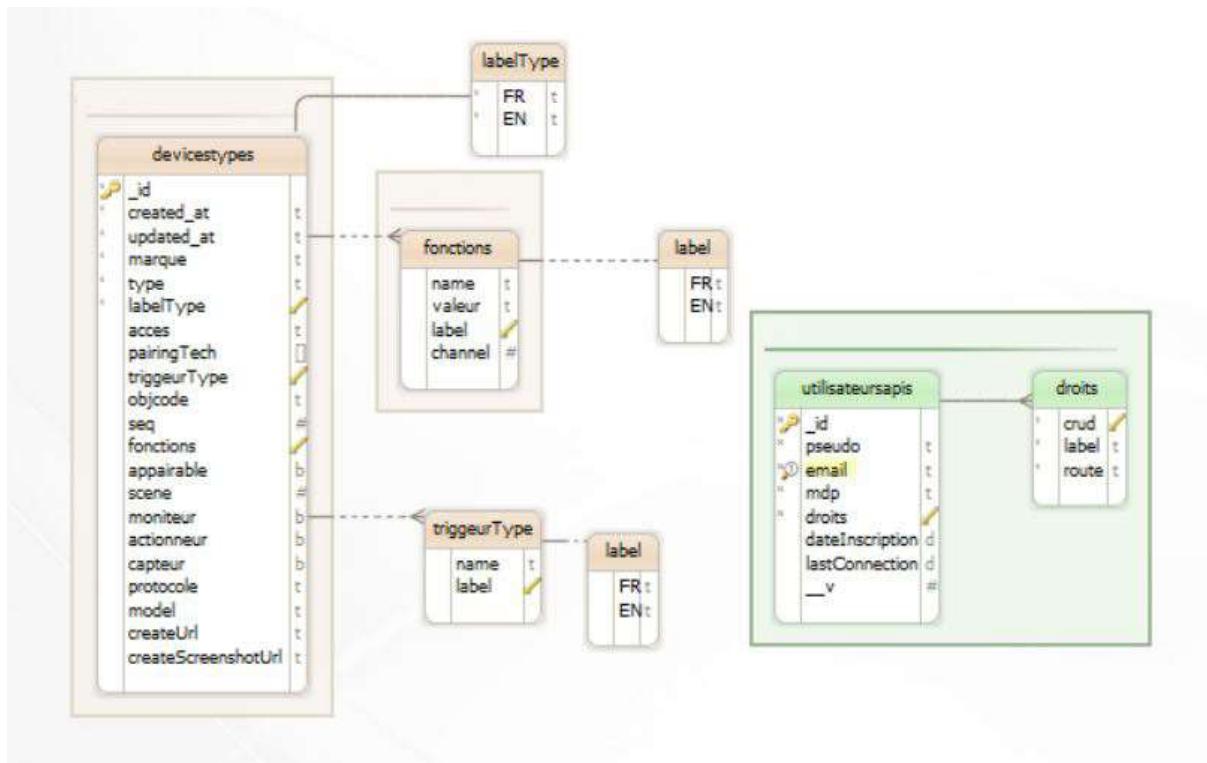


## 4.5 - LE MPD (SCHEMA DE LA BD)

A ce stade nous pouvons générer le modèle physique de données à partir du modèle précédent et à l'aide de la base de donnée existante.







Les différentes liaisons représentées dans les schémas du dessus sont la représentation des tableaux ou des objets dans les collections.

## 4.6 - MONGODB : LE LDD (LANGUAGE DE DEFINITION DES DONNEES)

Pour créer cette base de données, il suffit de dire à MongoDB que l'on souhaite l'utiliser et comme elle n'existe pas, il va la créer tout seul :

- `use nomBaseDeDonnees`

Pour créer une collection (table en SQL), il y a 2 manières :

- `db.nomCollection.insert({champ1:"valeur1", champN:"valeurN"})`  
La création de la collection est implicite, elle se fait à l'insertion du premier document.
- `db.createCollection('nomCollection')`

Attention : Si une collection ne porte pas la même orthographe, MongoDB crée une nouvelle collection.

Les documents de MongoDB correspondent aux enregistrements (rows) d'une table SQL. Ils sont au format BSON (Binary Serialized object Notation) qui est un héritier de JSON de JavaScript.  
C'est un ensemble de paires clé => valeur. Les valeurs pouvant être de différents type :

### Difference entre JSON et BSON

<u>DocumentDB</u>	<u>MongoDB</u>
<b>JSON</b>	<b>BSON = JSON +</b>
▪ Number	▪ ObjectId
▪ String	▪ DateTime
▪ Boolean	▪ Byte Arrays
▪ Array	▪ GeoSpatial
▪ Object	
▪ null	

Remarque : Le seul champ commun et obligatoire à tous les documents de toutes les collections est le champ de clé principale (" \_id").

Résumé des commandes LDD (langage de définition des données) :

Objectif	Commande
<b>Liste des BD d'un serveur</b>	show dbs
<b>Création d'une BD</b>	use nouveauNomDeBD db.createCollection("nouvelleCollection")
<b>Suppression d'une BD</b>	db.dropDatabase()
<b>Liste des collections d'une BD</b>	use NomDeLaBD show collections
<b>Création d'une collection (conteneur)</b>	db.createCollection("nouvelleCollection")
<b>Suppression d'une collection</b>	db.NomDeCollection.drop()

## 4.7 - MONGODB : LE LMD (LANGAGE DE MANIPULATION DES DONNEES)

Le CRUD: Create - Read - Update - Delete.

Pour créer un document, on utilise la commande `insert()` :

- `db.nomCollection.insert({champ1:"valeur1", champN:"valeurN"})`

Pour lister tous les documents d'une collection, on utilise la méthode `find()` :

- `db.nomCollection.find()`

Le listage avec des critères (select + where en SQL) :

- `db.nomCollection.find({champ : valeur})`

Le choix de projection lors d'un listage (« select nomTable from table » en SQL) :

- `db.nomCollection.find({}, {champ : boolean, champ2 : boolean,...})`

Pour modifier un document, on utilise les méthodes `update()`, `save()` ou `findAndModify()` :

- `db.nomCollection.update({where}, {$set : {modification}})`  
Sans le `$set` la modification effectue un « remplace » de tous les champs du (ou des) documents par celui (ou ceux) de la modification.  
Avec le `$set`, on ne change pas la structure du document et on modifie que le ou les champs nommés dans la modification.
- `db.nomCollection.findAndModify(query: {champ : valeur,...}, update :{champ : valeur,...})`  
Cette méthode effectue un `find()` avec la restriction « `query` » qui retourne zéro, un ou plusieurs documents. Ils sont ensuite modifiés au sens SQL. C'est-à-dire qu'on modifie que le ou les champs déclaré dans le `update` sans modifier la structure du document.

- `db.nomCollection.save(document)`

Cette méthode effectue un update si la valeur du champ « `_id` » est présent en base de données, sinon elle effectue un insert.

Pour supprimer un document, on utilise la méthode `remove()` :

- `db.nomCollection.remove({champ : valeur, ...})`

Si plusieurs résultats sont trouvés, à l'image d'un `findAndModify()`, ils sont tous supprimés.

Résumé des commandes LMD (langage de manipulation des données) :

Objectif	Commande
<b>Création d'une collection (conteneur)</b>	<code>db.createCollection("nouvelleCollection")</code>
<b>Suppression d'une collection</b>	<code>db.NomDeCollection.drop()</code>
<b>Création d'un document (contenu)</b>	<code>db.NomDeCollection.insert( { clé: valeur, clé: valeur ...} )</code>
<b>Liste des documents d'une collection avec leur contenu</b>	<code>db.NomDeCollection.find()</code>
<b>Projection : Contenu d'un document (quelques attributs)</b>	<code>db.nomDeLaCollection.find( {}, {clé: booléen, ...} )</code>
<b>Restriction : Contenu d'un document (quelques documents)</b>	<code>db.NomDeCollection.find({clé:valeur})</code>
<b>Restriction et Projection : Contenu d'un document (quelques documents, quelques attributs)</b>	<code>db.NomDeCollection.find({clé:valeur}, {clé1: booléen, [clé2: booléen, ], ...} )</code>
<b>Suppression d'un document</b>	<code>db.nomDeLaCollection.remove( { prédicat } )</code>
<b>Modification d'un document</b>	<code>db.nomDeLaCollection.update( { where }, { modification } )</code> cf aussi <code>findAndModify()</code> ...

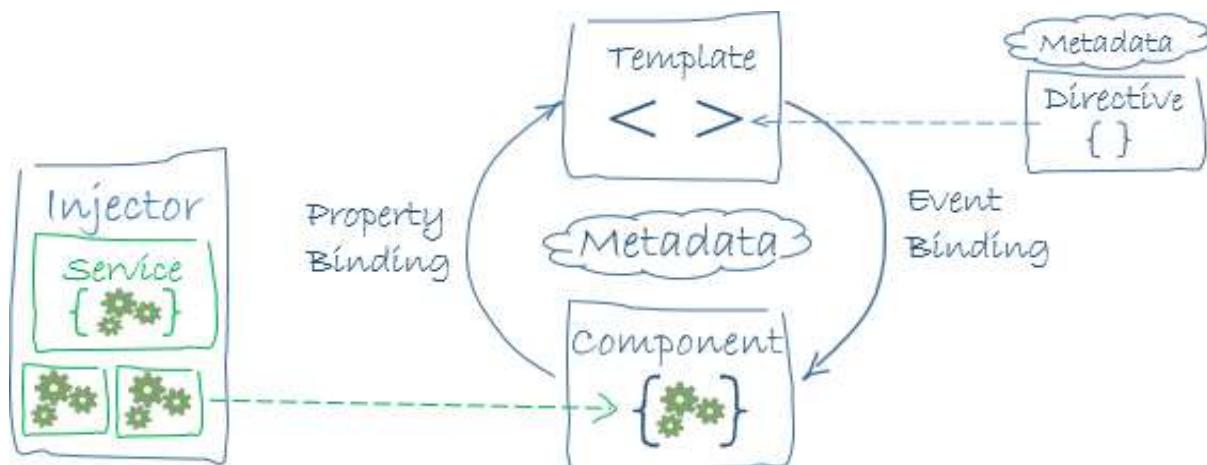
## ***CHAPITRE 5 - CONCEPTION DE L'APPLICATION***

## 5.1 - TECHNOLOGIES UTILISEES

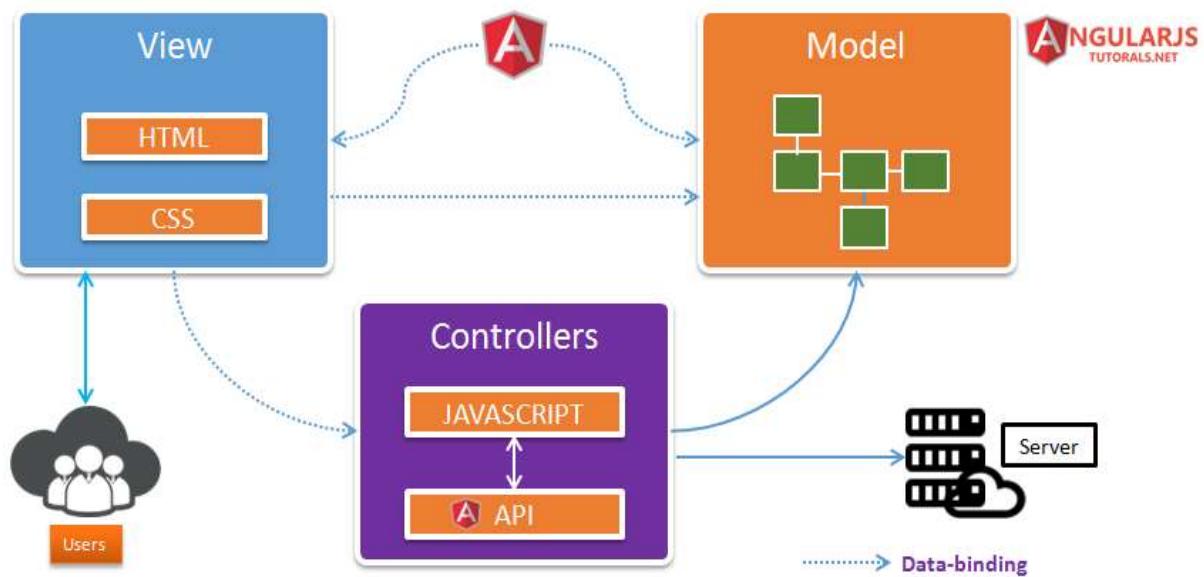


Le Framework Angular 4 a pour but d'aider le développeur à concevoir et organiser son code JavaScript comme une vraie application dédiée à l'UI, en implémentant des patterns dérivés du MVC (Modèle - Vue - Contrôleur) et en fournissant des boîtes outils riches pour la gestion des appels asynchrones, la mise à jour de l'interface et la gestion des données métiers.

Il se compose de quatre couches dominantes qui sont la vue, les contrôleurs (une vue a un contrôleur), des services et des directives.



Et j'ai développé l'application sur un design pattern propre Modèle-Vue-Vue-Modèle ou plus exactement DataServices Vue Controller. La force du Framework réside dans le data-binding bi directionnel entre la vue et le modèle, ce qui facilite grandement le développement.



Le data-binding bidirectionnel représenté par le A de Angular 4 sur le schéma permet une relation automatique entre la vue et le modèle de la vue. C'est-à-dire que si je modifie la valeur d'un input dans la vue, elle sera automatiquement changée dans le modèle de la vue. Si l'utilisateur émet un évènement via un click d'un bouton pour sauvegarder les données modifiées de la vue, le contrôleur qui reçoit l'évènement utilisera simplement le modèle de la vue.

Exemple :

➤ Le modèle et ses modèles-vues

```
16  ****
17  * ATTRIBUTS
18  ****
19  searchFish = '';      // barre de recherche mise à chaîne vide
20  orderbySomeFish = "created"; // ordre de tri sur created par défaut
21  orderType: boolean = false; // false par défaut pour un tri asc
22  //listes des entêtes du foyers pour le tableau
23  customersEntetes = [
24    { label: "ID", orderby: "_id", orderType: false },
25    { label: "Label", orderby: "Label", orderType: false },
26    { label: "Référence Client", orderby: "Ref_client", orderType: false },
27    { label: "Adresse", orderby: "Adresse", orderType: false },
28    { label: "CP", orderby: "Cp", orderType: false },
29    { label: "Ville", orderby: "Ville", orderType: false },
30    { label: "Téléphone", orderby: "Tel", orderType: false },
31    { label: "Email", orderby: "Email", orderType: false },
32    { label: "Date de Crédit", orderby: "created", orderType: false }
33];
34 // liste des foyers instanciée
35 customersListe: any = [];
36 // foyer sélectionné instancié à null
37 customerSelected = null;
38 usersListe = null;
39 // titre
40 titre = 'Liste des foyers (Customers)';
41 // variable message pour afficher les erreurs
42 message: string = "";
43 // variable pour le contrôle du formulaire
44 rForm: FormGroup;
```

Les entêtes du tableau sont à ce jour entrées à la main dans une variable reliée bidirectionnellement avec la vue.

La liste des customers est initialisée à un tableau vide. Et lors de l'initialisation de la page, une méthode « getAllCustomers() » est appelée afin d'affecter via le contrôleur ce modèle.

Le customer sélectionné est instancié à « null » pour ne pas afficher le formulaire de modification/création.

- La vue avec des liaisons, une direction (modèle=> vue) pour l'affichage à l'aide de « {{ma variable du modèle}} »

```

24   <table class="table table-striped table-hover table-sm table-responsive">
25     <!-- les entêtes du tableau -->
26     <thead>
27       <tr>
28         <!-- boucle sur la variable qui contient les entêtes -->
29         <th *ngFor="let entete of customersEntetes"
30           (click)="orderbySomeFish = entete.orderby; entete.orderType = !entete.orderType;
31           orderType = !entete.orderType">
32           <!-- au clique de l'entête on affecte la variable qui ordonne le tableau -->
33           {{entete.label}}
34           <!-- affichage selon l'entête et selon si le tri est asc ou desc -->
35           <span [hidden]="entete.orderType || entete.orderby !== orderbySomeFish">
36             <i class="fa fa-caret-up" aria-hidden="true"></i>
37           </span>
38           <span [hidden]="!entete.orderType || entete.orderby !== orderbySomeFish">
39             <i class="fa fa-caret-down" aria-hidden="true"></i>
40           </span>
41         </th>
42         <th>
43           Voir users
44         </th>
45       </tr>
46     </thead>
47     <!-- body du tableau -->
48     <tbody>
49       <!-- boucle sur la variable qui contient la liste des customers
50           avec un event "click" sur chaque customer-->
51       <tr *ngFor="let customer of customersListe | filter : searchFish">
52         <td (click)="onSelect(customer)">{{customer._id}}</td>
53         <td (click)="onSelect(customer)">{{customer.Label}}</td>
54         <td (click)="onSelect(customer)">{{customer.Ref_client}}</td>
55         <td (click)="onSelect(customer)">{{customer.Adresse}}</td>
56         <td (click)="onSelect(customer)">{{customer.Cp}}</td>
57         <td (click)="onSelect(customer)">{{customer.Ville}}</td>
58         <td (click)="onSelect(customer)">{{customer.Tel}}</td>
59         <td (click)="onSelect(customer)">{{customer.Email}}</td>
60         <td (click)="onSelect(customer)">{{customer.created}}</td>
61         <td>
62           <button class="btn btn-info pull-center" type="button"
63             (click)="SeeUsers(customer)">Users</button>
64         </td>
65       </tr>
66     </tbody>
67   </table>

```

\*ngFor permet de bouclé sur les tableaux d'objet dans l'exemple.  
La méthode onSelect(customer) utilise this.customer, c'est-à-dire celui de ce tour de boucle.

## ➤ Le contrôleur et ses méthodes

```
83  /**
84  * méthodes
85  */
86  Complexity is 6 It's time to do something...
87  █ getAllCustomer() {
88  // utilisation de la méthode getAll de l'ApiService
89  Complexity is 4 Everything is cool
90  this.apiService.getAll("customer").then( █ (response: any) => {
91  console.log(response);
92  // en cas de réponse avec un success à true
93  if (response.success) {
94  // on récupère la liste
95  this.customersListe = response.data;
96  } else {
97  // sinon on est déconnecté
98  console.log(response.message);
99  this.authService.logout();
100 this.router.navigate(['/login']);
101 }
102 .catch((err) => {
103 // en cas d'erreur système on est déconnecté
104 this.authService.logout();
105 this.router.navigate(['/login']);
106 console.log(err + " : Une erreur système");
107 })
108
109
110 onSelect(customer) {
111 // le foyer clické devient le customerSelected
112 this.customerSelected = customer;
113 this.message = "";
114 }
```

Ici la méthode « `getAllCustomer()` » fait appel à un service qui a des méthodes qui effectuent des requêtes sur l'API Rest afin de récupérer des données. En cas de succès de la requête, on affecte au modèle la liste des `customer` qui automatiquement grâce au `DataServices` de `AngularJS`.

## ➤ La Relation bidirectionnelle entre la vue et le modèle

```

86 <div [formGroup]="rForm">
87   <div class="col-md-6 col-md-offset-3">
88     <div class="form-group">
89       <label class="control-label labelFormulaire">_id</label>
90       <input type="text" [(ngModel)]="customerSelected._id" class="form-control" formControlName="_id">
91     </div>
92     <div class="form-group">
93       <label class="control-label labelFormulaire">Label</label>
94       <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Label'].valid">*Champs requis et doit compo...
95       <input type="text" [(ngModel)]="customerSelected.Label" class="form-control" formControlName="Label">
96     </div>
97     <div class="form-group">
98       <label class="control-label labelFormulaire">Ref_client</label>
99       <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Ref_client'].valid">*Champs requis et doit ...
100      <input type="text" [(ngModel)]="customerSelected.Ref_client" class="form-control" formControlName="Ref_client">
101    </div>
102    <div class="form-group">
103      <label class="control-label labelFormulaire">Adresse</label>
104      <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Adresse'].valid">*Champs requis</label>
105      <input type="text" [(ngModel)]="customerSelected.Adresse" class="form-control" formControlName="Adresse">
106    </div>
107    <div class="form-group">
108      <label class="control-label labelFormulaire">Cp</label>
109      <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Cp'].valid">*Champs requis</label>
110      <input type="text" [(ngModel)]="customerSelected.Cp" class="form-control" formControlName="Cp">
111    </div>
112    <div class="form-group">
113      <label class="control-label labelFormulaire">Ville</label>
114      <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Ville'].valid">*Champs requis</label>
115      <input type="text" [(ngModel)]="customerSelected.Ville" class="form-control" formControlName="Ville">
116    </div>
117    <div class="form-group">
118      <label class="control-label labelFormulaire">Tel</label>
119      <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Tel'].valid">*Champs requis</label>
120      <input type="text" [(ngModel)]="customerSelected.Tel" class="form-control" formControlName="Tel">
121    </div>
122    <div class="form-group">
123      <label class="control-label labelFormulaire">Email</label>

```

Ici lorsque la variable du modèle « customerSelected » est non « null », on affiche le formulaire.

C'est-à-dire lors d'une création ou d'une modification d'un customer via les méthodes `onSelect(customer)` et `onCreate()`.

Les inputs sont reliés au modèle bi directionnellement grâce à Angular 4 et le « NgModel ».

Si la saisie de l'input est modifiée, elle est automatiquement modifiée dans le modèle et inversement comme lors du onSelect(customer).

Lors de l'appel de la méthode « save() », celle-ci utilise simplement le modèle « customerSelected ».



Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge.

Node.js utilise le moteur d'exécution ultrarapide V8 de Google Chrome. Ce moteur V8 avait fait beaucoup parler de lui à la sortie de Google Chrome, car c'est un outil open source créé par Google qui analyse et exécute du code JavaScript très rapidement.

Comme JavaScript est un langage conçu autour de la notion d'évènement, Node.js a pu mettre en place une architecture de code entièrement non bloquante.

Imaginez un programme dont le rôle est de télécharger un fichier puis de l'afficher. Voici comment on écrirait le code dans un **modèle bloquant** :

```
Télécharger un fichier  
  
Afficher le fichier  
  
Faire autre chose
```

Les actions sont effectuées dans l'ordre. Il faut lire les lignes de haut en bas :

1. Le programme va télécharger un fichier sur Internet
2. Le programme affiche le fichier à l'utilisateur
3. Puis ensuite le programme peut faire d'autres choses (effectuer d'autres actions)

Maintenant, on peut écrire le même code sur un **modèle non bloquant** :

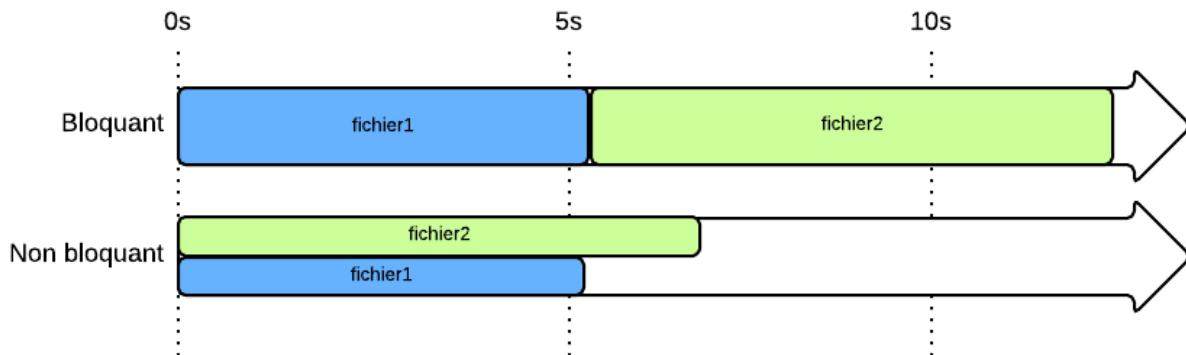
```
Télécharger un fichier  
  
Dès que c'est terminé, afficher le fichier  
  
Faire autre chose
```

Le programme n'exécute plus les lignes dans l'ordre où elles sont écrites. Il fait ceci :

1. Le programme lance le téléchargement d'un fichier sur Internet

2. Le programme fait d'autres choses (le programme suit son cours)
3. Dès que le téléchargement est terminé, le programme effectue les actions qu'on lui avait demandées : il affiche le fichier

Du coup, le téléchargement des 2 fichiers au total va beaucoup plus vite puisque le programme fait les 2 à la fois :



Node.js contient une bibliothèque de serveur HTTP intégrée, ce qui rend possible de faire tourner un serveur web sans avoir besoin d'un logiciel externe comme Apache ou lighttpd, et permettant de mieux contrôler la façon dont le serveur web fonctionne.

```
var http = require('http');

//création d'un objet serveur:
http.createServer(function (req, res) {
  res.write('Hello World!'); //écriture de la réponse envoyée au client
  res.end(); //fin de la réponse
}).listen(8080); //le serveur est écouté sur le port 8080
```

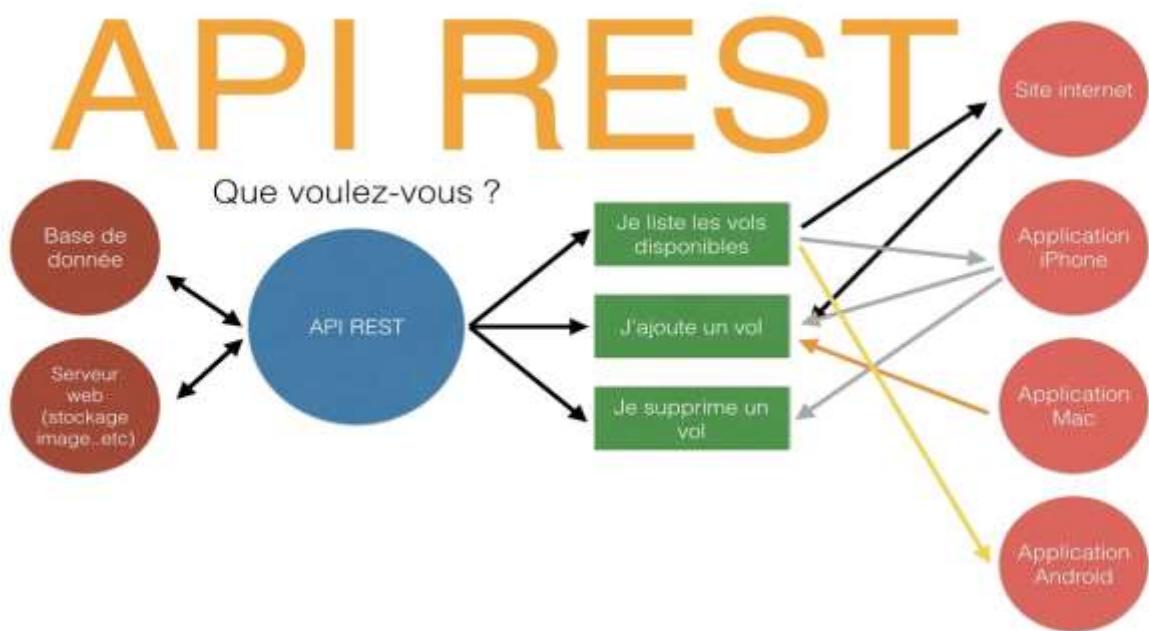
Concrètement, Node.js est un environnement d'assez bas niveau permettant d'exécuter du JavaScript non plus dans le navigateur web mais sur le serveur.

Node.js est de plus en plus populaire comme plateforme serveur, elle est utilisée par Groupon, SAP, LinkedIn, Microsoft, Yahoo!, Walmart, Rakuten, Sage et PayPal.

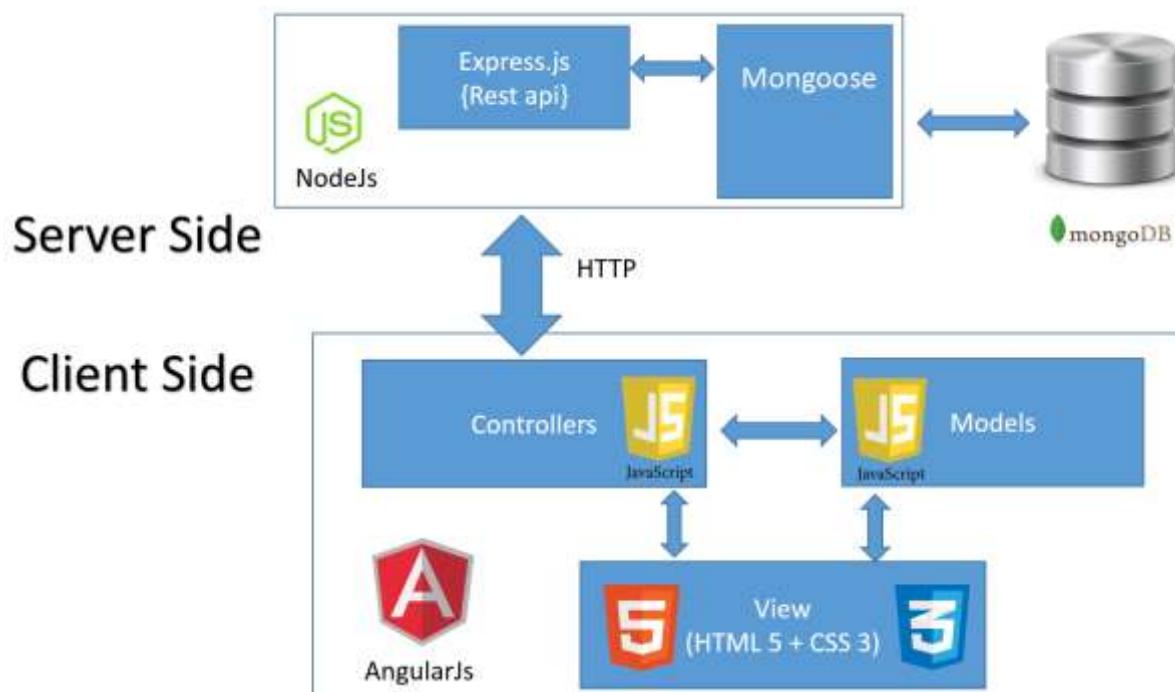
## 5.2 - ARCHITECTURE DU PROJET

C'est avec la partie conception de la base de données, ainsi qu'avec les technologies utilisées qui je peux élaborer l'architecture de mon projet.

En premier lieux, que l'Api Rest coté server pourra être appelé par le client Angular qui j'aurai développé mais également par d'autre client (qui pourront être aussi des servers). Elle aura une architecture similaire au schéma suivant :



Et ensuite par assemblage l'architecture du projet prendra l'allure d'un diagramme de déploiement présenté dans quelques chapitres.  
Ce schéma annoncera également le diagramme de séquence détaillé du chapitre suivant.



## 5.3 - LES DIAGRAMMES DE SEQUENCE DETAILLÉS

En UML, le diagramme de séquence (DSEQ) permet de décrire et de formaliser les interactions entre les acteurs et le système ou les classes, par l'intermédiaire de messages, d'un point de vue chronologique et spatial. Le temps est représenté à la verticale et l'espace à l'horizontal.

Dans la phase de conception de l'application, ce diagramme est détaillé et les messages correspondent aux messages des langages informatiques.

Le diagramme de séquence détaillé représente, selon les design pattern (patron de conception) ECB (Entity - Control - Boundary) et MVC (Modèle-Vue-Contrôleur) que je vais utiliser pour le développement de l'application :

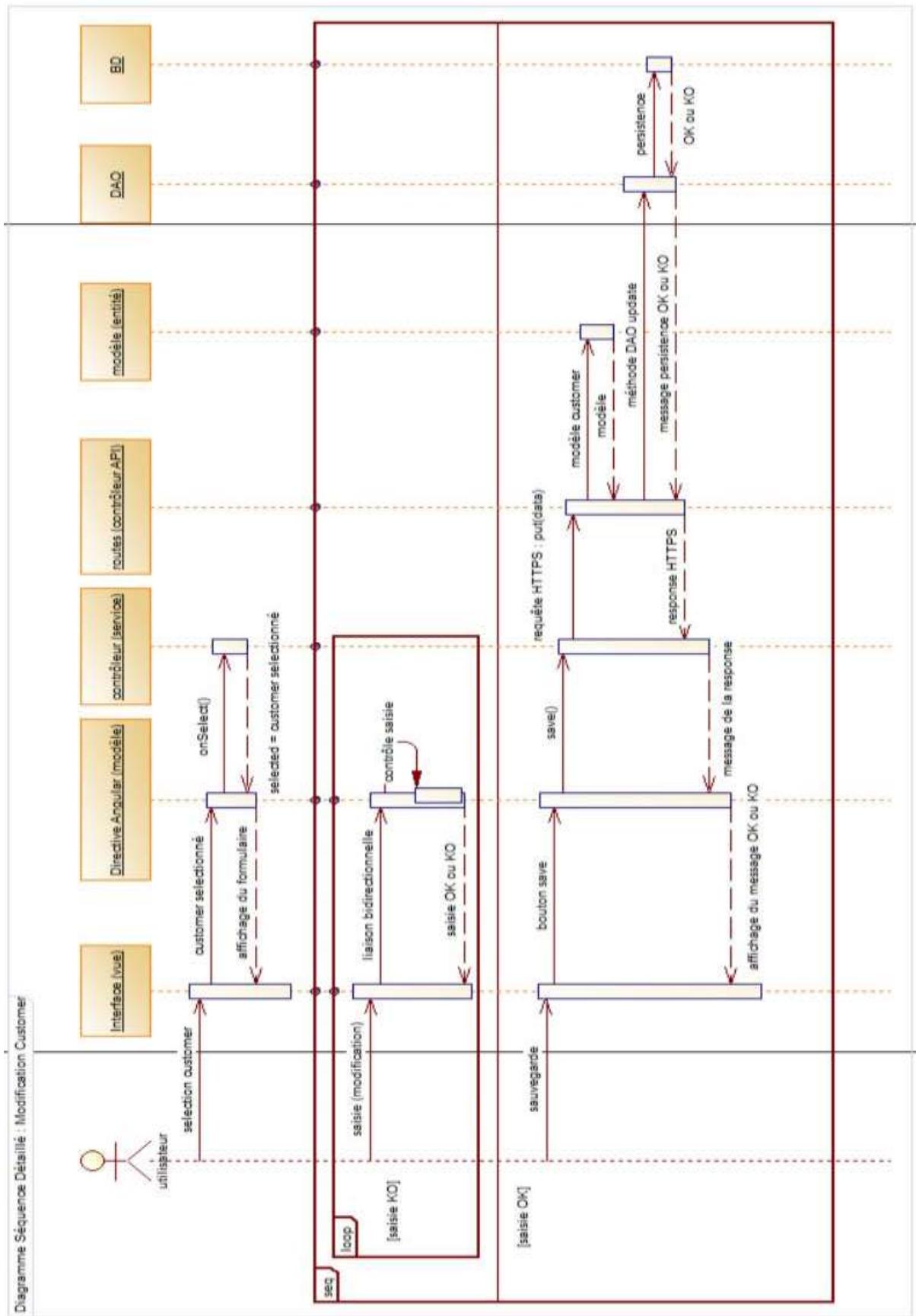
- Les utilisateurs qui interagissent avec le système au moyen des Boundaries (utilisant un design pattern MVC : html pour la vue, TypeScript pour le modèle et le contrôleur),
- Les Boundaries qui envoient des instructions aux contrôleurs (Route de l'API),
- Les Contrôleurs qui font l'intermédiaire entre les Boundaries et les entités et qui orchestrent l'exécution des commandes,
- Les modèles (Entities et les DAOs) qui représentent l'accès aux données du système.

Dans l'application :

Boundaries : Angular

Controller : Routes de l'API

Entities : Modèles et DAO de l'API.



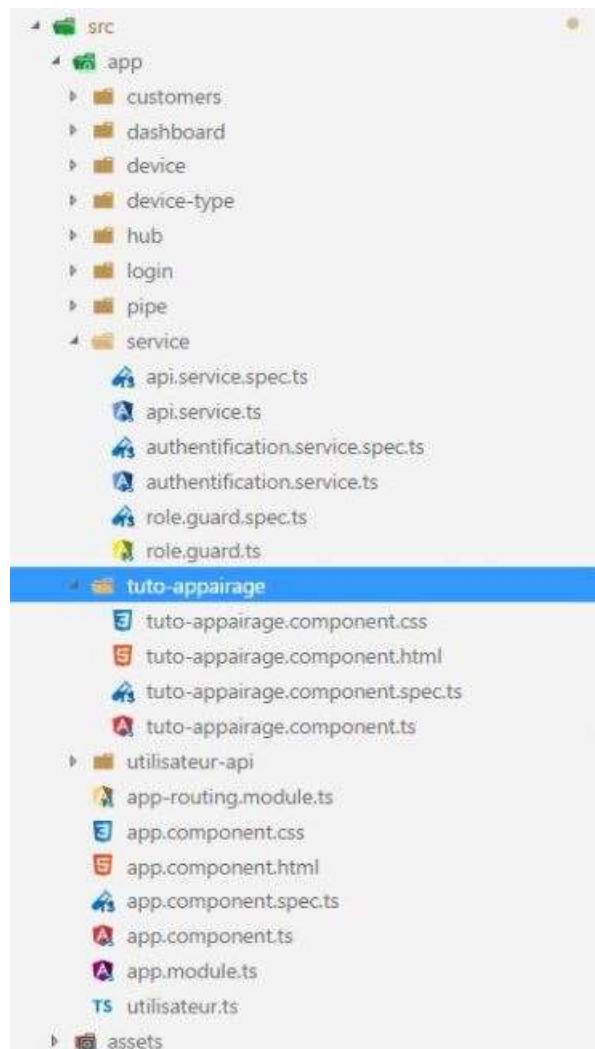
## **CHAPITRE 6 - DEVELOPPEMENT PARTIE FRONT**

## 6.1 - DEVELOPPEMENT

Au cours des différents codes qui vont défiler, je vais garder une logique séquentielle (celle du diagramme de séquences détaillées) :

- Interface utilisateur (Chrome),
- Codes semi-statique (HTML),
- Codes dynamique des écrans (TypeScript),
- Codes DAO (Data-Access-Object) des services.

On y détaillera premier lieu la structure du projet avec le component principal. Ensuite le component login puis le component des tutoriaux pour changer de celui des customers.



Un projet Angular est composé de plusieurs « component » (groupe de fichiers étant composé principalement d'un template HTML, d'une feuille de style CSS propre à ce component et d'un fichier script TypeScript).

Ces components se veulent indépendants les uns des autres pour obtenir une structure modulable.

Un fichier « app.module.ts » recense justement les différents component, provider (services) et dépendances de Angular type HttpModule, FormModule ou AppRoutingModule.

Le component « AppComponent » servira de structure parent aux autre components et contiendra la barre de navigation.

### 6.1.1 - AppComponent.html

```

1  <!-- affichage route principale du site.-->
2
3  <!-- navbar -->
4  <!-- affichage de la navbar si l'utilisateur est authentifié via la méthode isAuthenticated() -->
5  <div *ngIf="isAuthenticated()">
6      <nav class="navbar navbar-default">
7          <div class="container-fluid">
8              <!-- Brand and toggle get grouped for better mobile display -->
9              <div class="navbar-header">
10                  <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1"
11                      aria-expanded="false">
12                      <span class="sr-only">Toggle navigation</span>
13                      <span class="icon-bar"></span>
14                      <span class="icon-bar"></span>
15                      <span class="icon-bar"></span>
16                  </button>
17                  <a class="navbar-brand" href="#">
18                      
19                  </a>
20                  <!-- affichage du pseudo de l'utilisateur connecté -->
21                  <p class="navbar-brand">Bonjour {{user.pseudo}}</p>
22          </div>
23
24          <!-- Collect the nav links, forms, and other content for toggling -->
25          <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
26              <ul class="nav navbar-nav">
27                  <!-- boucle sur la liste des routes -->
28                  <li *ngFor="let route of routesListe">
29                      <a [routerLink]="route.route" routerLinkActive="active" [routerLinkActiveOptions]="{{exact:true}}">{{route.titre}}

```

### 6.1.2 - AppComponent.ts

La balise <router-outlet> contiendra les templates HTML des components appelés par les [routerLink] de la barre de navigation ou ceux de la page Dashboard.

La liste des routes est construite en fonction des droits de l'utilisateur connecté.

```
25
26 //méthode
27 // methode de angular à l'ouverture du component ( ici l'application)
    Complexity is 3 Everything is cool!
28 █ ngOnInit() {
29     // appelle de la méthode du service authentification isAuthenticated
30     // qui renvoie un boolean si un user est présent dans le localStorage
31     if (this.auth.isAuthenticated()) {
32         // on récupère l'utilisateur pour afficher son preudo
33         this.user = this.auth.getUser();
34         // on récupère la liste des routes permises à cet utilisateur
35         this.routesListe = this.apiService.getRoutesListe();
36         this.authenticated = true;
37         // on est redirigé vers la route home du site
38         this.router.navigate(['/dashboard']);
39     } else {
40         //sinon on cache la barre de navigation avec l'attribut boolean authenticated
41         this.authenticated = false;
42         // on initialise la liste des routes à seulement la route home
43         this.routesListe = this.apiService.getRoutesListe();
44         // et on est redirigé vers la route du login
45         this.router.navigate(['/login']);
46     }
47 }
```

### 6.1.3 - Api.service.ts

Méthode appelé lors du `ngOnInit()` du `AppComponent`. Elle retourne une liste de routes.

```
61  ■ getRoutesListe() {
62    let droits = this.auth.getRole();
63    let admin = false;
64    if (droits !== null) {
65      droits.forEach(droit => {
66        if (droit.route == "admin") {
67          admin = true;
68        }
69      });
70      if (admin) {
71        // this.listeDroitsApi reste celle affichée plus haut
72        this.routesListePermis = this.routesListe;
73      } else {
74        //on récupère la liste des roles que l'user a.
75        this.listeDroitsApi = this.auth.getRole();
76        if (this.listeDroitsApi !== null) {
77          //tableau mis à zéro
78          this.routesListePermis = [];
79          // pour chaque route de la liste de toutes les routes
Complexity is 3 Everything is cool!
80          this.routesListe.forEach( ■ route => {
81            //je le compare à la route de chaque droit
82            droits.forEach(droit => {
83              if (droit.route == route.route) {
84                // s'il y a coïncidence on incorpore la route à la liste de celles permises
85                this.routesListePermis.push(route);
86              }
87            });
88          });
89        }
90        //on affecte selon les droits les routes disponible pour l'application
91      }
92    } else {
93      //sinon si l'utilisateur n'a pas de droit la liste des routes permises sera :
94      this.routesListePermis = [{ titre: "Menu principal", route: "dashboard", image: "" }];
95    }
96    return this.routesListePermis;
97  }
```

```

13 * ATTRIBUTS
14 ****
15 | private adresseAPI = 'http://localhost:8280/api';//dev LOCAL
16 |
17
18 //liste des routes de toutes l'application
19 private routesListe = [
20     { titre: "Les foyers", route: "customer", image: "foyer.jpg" },
21     { titre: "Les hubs", route: "hub", image: "hub.jpg" },
22     { titre: "Tous les devices", route: "device", image: "devices.jpg" },
23     { titre: "Les Types de devices", route: "deviceType", image: "type-devices.jpg" },
24     { titre: "Les tutos d'appairage", route: "tutoAppairage", image: "tutorial.png" },
25     { titre: "Les Users de l'API", route: "utilisateurApi", image: "users.jpg" }
26 ];
27 private routesListePermis = [{ titre: "Menu principal", route: "dashboard", image: "" }];
28 //liste de tous les droits de l'api, potentiellement à migrer vers l'Api elle même et à récupérer en tant que admin
29 private listeDroitsApi: any = [
30     {{ label: "Administrateur", route: "admin", crud: [] }},
31     {{ label: "Alerteurs", route: "alerteurs", crud: ["create", "read", "update", "delete"] }},
32     {{ label: "Alerts", route: "alerts", crud: ["create", "read", "update", "delete"] }},
33     {{ label: "Code", route: "code", crud: ["create", "read", "update", "delete"] }},
34     {{ label: "Count", route: "count", crud: ["create", "read", "update", "delete"] }},
35     {{ label: "Customers (foyers)", route: "customer", crud: ["create", "read", "update", "delete"] }},
36     {{ label: "Devices", route: "device", crud: ["create", "read", "update", "delete"] }},
37     {{ label: "Type de devices", route: "deviceType", crud: ["create", "read", "update", "delete"] }},
38     {{ label: "Hubs", route: "hub", crud: ["create", "read", "update", "delete"] }},
39     {{ label: "Interaction des Logs", route: "interactionLog", crud: ["create", "read", "update", "delete"] }},
40     {{ label: "Logs", route: "log", crud: ["create", "read", "update", "delete"] }},
41     {{ label: "Monitoring", route: "monitoring", crud: ["create", "read", "update", "delete"] }},
42     {{ label: "Protocol", route: "protocol", crud: ["create", "read", "update", "delete"] }},
43     {{ label: "Scene", route: "scene", crud: ["create", "read", "update", "delete"] }},
44     {{ label: "Type de triggers", route: "triggerType", crud: ["create", "read", "update", "delete"] }},
45     {{ label: "Tuto d'appairage", route: "tutoAppairage", crud: ["create", "read", "update", "delete"] }},
46     {{ label: "Univers", route: "univers", crud: ["create", "read", "update", "delete"] }},
47     {{ label: "Client user", route: "user", crud: ["create", "read", "update", "delete"] }},
48     {{ label: "Utilisateur API", route: "utilisateurApi", crud: ["create", "read", "update", "delete"] }}
49 ];

```

#### 6.1.4 - Interface login



### 6.1.5 - Login.html

```
1 <!-- présentation du site -->
2 <div style="text-align:center">
3   <h1>
4     | Welcome to {{title}}!
5   </h1>
6   
7 </div>
8 <br>
9 <br>
10 <!-- formulaire de login -->
11 <div class="row">
12   <div class="col-md-4 col-md-offset-4">
13     <form (ngSubmit)=login()
14       <div class="form-group">
15         <label for="email">Login</label>
16         <input id="email" name="email" type="email" [(ngModel)]="credentials.email" class="form-control" required>
17       </div>
18       <div class="form-group">
19         <label for="password">Password</label>
20         <input id="password" name="mdp" type="password" [(ngModel)]="credentials.mdp" class="form-control" required>
21       </div>
22       <div class="form-group">
23         <button class="btn btn-primary" type="submit" [disabled]="credentials.email == '' || credentials.mdp == ''">Sign in</button>
24       </div>
25     </form>
26     <div *ngIf="message!==''">
27       <p class="alert alert-danger">
28         | {{message}}
29       </p>
30     </div>
31   </div>
32 </div>
33 <!-- fin formulaire -->
```

Le bouton 'Sign-in' est désactivé tant que les champs ne sont pas remplis, ça effectue un pré-contrôle des saisies.

Lors du 'submit' du formulaire d'authentification, la méthode login() est lancée.

### 6.1.6 - Login.ts

```

32 //méthode login qui fait appel à la méthode du service authentification
33 Complexity is 6 It's time to do something...
34 login() {
35   console.log(this.credentials)
36   //appel de la méthode
37   this.authService.login(this.credentials)
38   // .then() et .catch() s'exécutent une fois la promesse de la méthode du service fini (resolve et reject)
39   Complexity is 4 Everything is cool!
40   .then( response: any ) => {
41     //si la réponse comporte un success true
42     if (response.success) {
43       console.log(response);
44       //appel de la méthode du component parent qui nous dirige ou non vers le dashboard si l'user est dans le storage
45       this.appComponent.ngOnInit();
46     } else {
47       console.log(response);
48       // si le success est faux on affiche un message
49       this.message = response.message;
50     }
51   })
52   .catch((err) => {
53     console.log(err);
54     this.message = err;
55   })
56 }

```

Angular permet de ne pas avoir besoin de manipuler le DOM pour récupérer des valeurs du code HTML. Exemple :

- var email = document.getElementById("idDeLaBaliseDuDOM").value();

La relation bi-directionnelle entre l'input et l'attribut permet de récupérer rapide et simplement la valeur de cet input.

```

12 //attribut
13 // titre
14 title = 'Otodo App Dev';
15 // message lors de l'authentification
16 message = "";
17 //objet comprenant l'email et le mdp, en dev je laisse l'email pour des tests rapides
18 credentials = {
19   email: "s@m.fr",
20   mdp: ""
21 };

```

### 6.1.7 - Authentication.service.ts

```
22  /**
23  *
24  *
25  * @param {any} credentials
26  * @returns
27  * @memberof AuthenticationService
28 */
29  Complexity is 7 It's time to do something...
30  ■ login(credentials) {
31    var url = this.adresse + '/sign/login';
32    // Promesse retournée à LoginPage
33    return new Promise(
34      Complexity is 5 Everything is cool!
35      ■ (resolve, reject) => {
36        //Appel asynchrone au backend
37        this.http.post(url, credentials).subscribe(
38          //callback de la requête http (succès)
39          Complexity is 3 Everything is cool!
40          ■ (response) => {
41            //transformation de la réponse texte en objet json
42            var data = response.json();
43            //hydratation de UserProvider en fonction des données de la requête
44            if ('success' in data && data.token !== null) {
45              console.log("ici je veux storage")
46              this.authenticated = data.success;
47              localStorage.setItem('currentToken', data.token);
48              localStorage.setItem('currentUser', JSON.stringify(data.utilisateur));
49            }
50            //Envoi des données à la page qui a initié la promesse
51            resolve(data);
52          },
53          //callback d'erreur http
54          (error) => {
55            console.log(error);
56            //reject du message d'erreur contenu dans le _body de l'erreur
57            reject(JSON.parse(error._body).message);
58          }
59        );
56      );
57    );
58  );
59 }
```

### 6.1.8 - Interface Dashboard (Home)



The screenshot shows a web browser window for the otodo Dashboard. The URL is https://api.todo.io/dashboard. The page title is "otodo". The main heading is "Veuillez selectionnez ce que vous voulez faire." Below it are five cards with icons and labels:

-  Les foyers
-  Les hubs
-  Tous les devices
-  Les Types de devices
-  Les tutoriels d'appareils
-  Les Users de l'API

### 6.1.9 - Dashboard.html

```
1  <h1>
2  | ENVIRONEMENT DEV
3  </h1>
4  <h1>
5  | {{titre}}
6  </h1>
7  <br>
8  <br>
9
10 <div class="row">
11   <div class="col-xs-2 text-center" *ngFor="let route of routesListe">
12     <a routerLink="/{{route.route}}>" class="thumbnail">
13       
14     </a>
15     <div class="caption">
16       <h5>
17         <strong>{{route.titre}}</strong>
18       </h5>
19     </div>
20   </div>
21 </div>
```

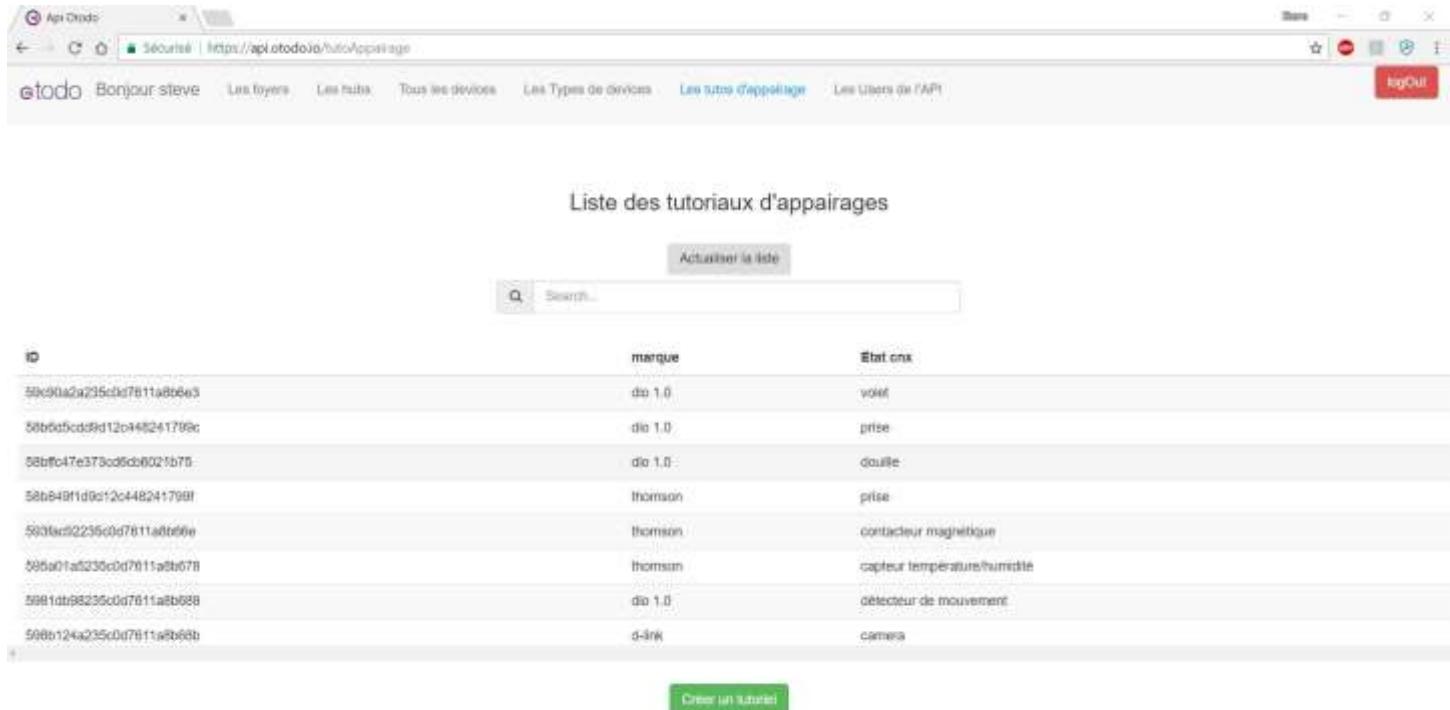
La liste des routes est construite de manière identiques à la liste des routes de la barre de navigation.

### 6.1.10 - Dashboard.ts

```
1 //imports
2 import { AuthentificationService } from './service/authentification.service';
3 import { Component, OnInit, NgZone } from '@angular/core';
4 import { Router, ActivatedRoute } from '@angular/router';
5 import { LoginComponent } from '../login/login.component';
6 import { ApiService } from '../service/api.service';
7
8 @Component({
9   selector: 'app-dashboard',
10  templateUrl: './dashboard.component.html',
11  styleUrls: ['./dashboard.component.css']
12 })
13
14 // class du component
15 export class DashboardComponent implements OnInit {
16   // attribut
17   titre: String = "Veuillez selectionnez ce que vous voulez faire."
18   //liste des routes
19   routesListe = [];
20
21   //constructeur
22   constructor(private router: Router, private route: ActivatedRoute, private apiService: ApiService,
23               private auth: AuthentificationService, private zone: NgZone) { }
24
25   //verification à chaque action dans le dashboard de la présence d'un user dans le localStorage
26   ngDoCheck() {
27     console.log("ngDoCheck dashbord");
28     if (!this.auth.isAuthenticated()) {
29       console.log('go to logout');
30       this.auth.logout();
31     }
32   }
33
34   // à l'ouverture de ce component, on récupère la liste des routes via le service Api
35   ngOnInit() {
36     console.log("init dashbord");
37     this.routesListe = this.apiService.getRoutesListe();
38   }
39
40 }///fin class
```

### 6.1.11 - Interface Tutoriaux

#### Liste des tutoriaux.

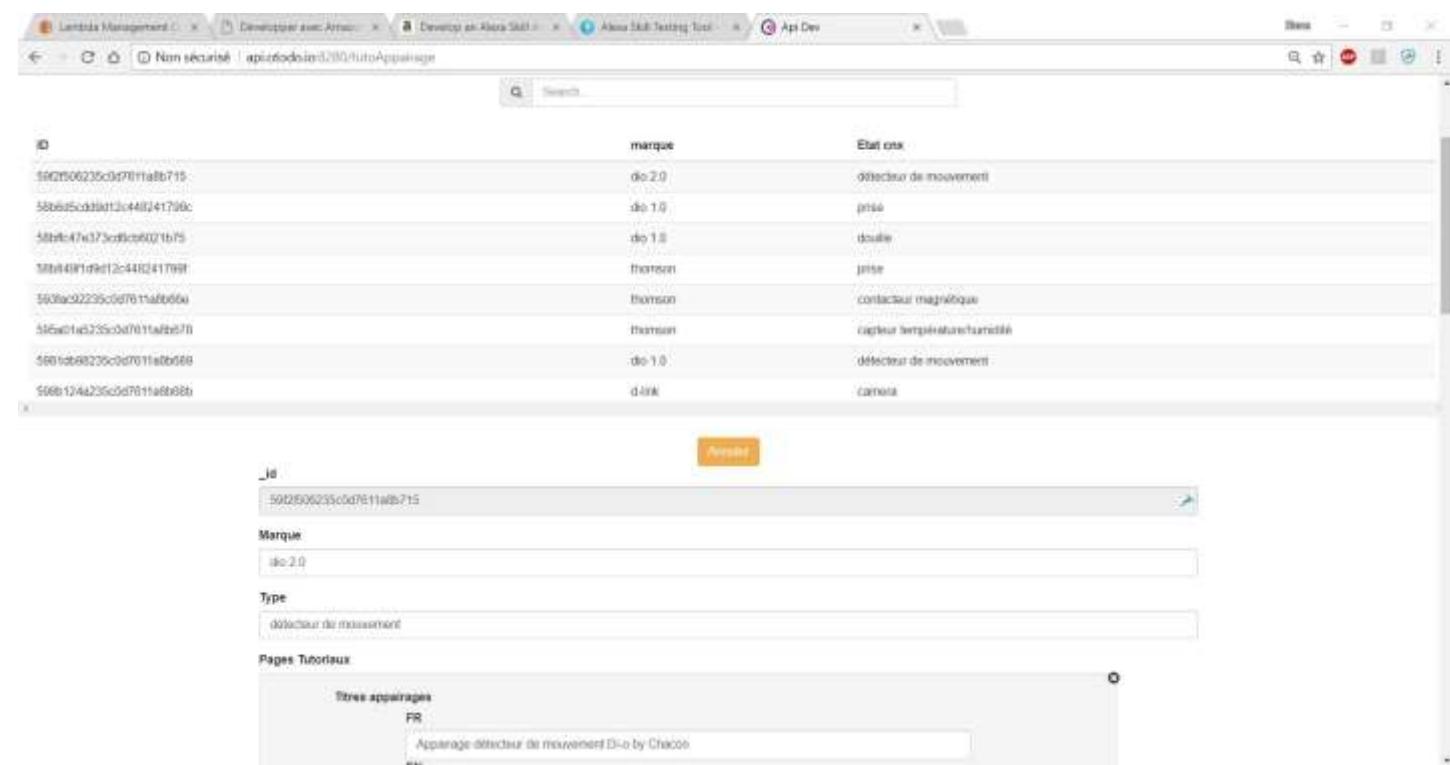


The screenshot shows a web application interface for managing paired devices. The title bar includes the URL <https://api.todo.io/tuto/appairage>. The main content area is titled "Liste des tutoriaux d'appairages". Below the title is a search bar with a placeholder "Search...". A button labeled "Actualiser la liste" is visible. The table has columns for "ID", "marque", and "Etat on". The data rows are:

ID	marque	Etat on
58c90a2a295cd7811a8b66e3	do 1.0	volet
58b6dfcd8d12b448241798c	do 1.0	prise
58bf047e373cd5db6021b70	do 1.0	douille
58ba4911d0c12c448241798f	thomson	prise
583a01af2235c0d7811a8b66e	thomson	contacteur magnétique
585a01af2235c0d7811a8b678	thomson	capteur température/humidité
5881db98235c0d7811a8b688	do 1.0	détecteur de mouvement
588b124a235c0d7811a8b68b	d-link	camera

A green button labeled "Créer un tutoriel" is located at the bottom right of the table area.

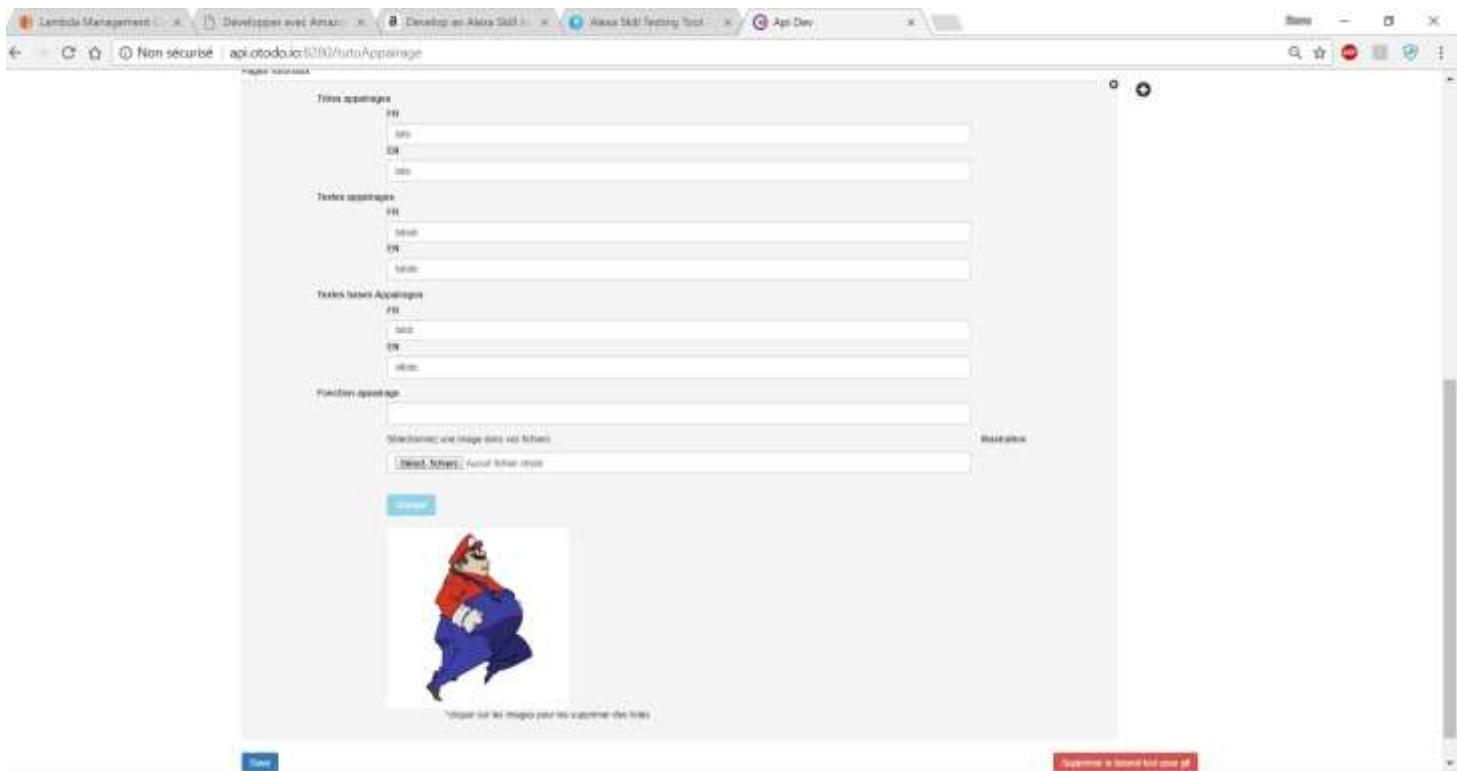
#### Lorsqu'un tutoriel est sélectionné.



The screenshot shows a detailed view of a selected device for pairing. The top navigation bar includes the URL <https://api.todo.io/tuto/appairage>. The main content area is titled "Ajouter un tutoriel". The table from the previous screenshot is present. Below the table, a form is displayed with the following fields:

- ID:** 582506235c0d7811a8b715
- Marque:** do 2.0
- Type:** détecteur de mouvement
- Pages Tutoriaux:**
  - Sites appairages:** FR
  - FR:** Appareil détecteur de mouvement Di-o by Chacof

# Rapport de Stage CDI Steve MORGEAT chez todo



Lors de la création d'un tutoriel :

A screenshot of the otodo application showing a tutorial creation form. The form includes fields for "Marque" (Brand), "Type" (Type), and "Pages Tutoriaux" (Tutorial pages). Under "Pages Tutoriaux", there are sections for "Titres apparaissants" (Title appearing), "Textes apparaissants" (Text appearing), "Textes bases Apparaissent" (Text bases appearing), and "Fonction apparaissant" (Function appearing). An "Illustration" section contains a placeholder for an image and a "Select\_fichiers" button. A note at the bottom says "\*cliquer sur les images pour les supprimer des listes". At the bottom left is a "Save" button.

## 6.1.12 - Tuto.html

### La vue.

```

1  <!-- titre -->
2  <h3 class="text-center">{{titre}}</h3>
3
4  <br>
5  <div class="row">
6    <!-- bouton pour rafraîchir le tableau -->
7    <div class="col-md-12 text-center">
8      <button class="btn btn-light pull-center" type="button" (click)="onRefresh()">Actualiser la liste</button>
9    </div>
10   <br>
11   <br>
12  <div class="col-md-4 col-md-offset-4 text-center">
13    <div class="form-group">
14      <div class="input-group">
15        <div class="input-group-addon">
16          <i class="fa fa-search"></i>
17        </div>
18        <input type="text" class="form-control" placeholder="Search..." [(ngModel)]="search">
19      </div>
20    </div>
21  </div>
22  <!-- tableau scrollable qui affiche le contenu de la collection en question -->
23  <div class="col-sm-12 scrollable">
24    <table class="table table-striped table-hover table-sm table-responsive">
25      <!-- les entêtes du tableau -->
26      <thead>
27        <tr>
28          <!-- boucle sur la variable qui contient les entêtes -->
29          <th *ngFor="let entete of entetes" (click)="orderbyColonne = entete.orderby; entete.orderType = !entete.orderType; orderbyType = !entete.orderType">
30            <!-- au clique de l'entête on affecte la variable qui ordonne le tableau -->
31            {{entete.label}}
32            <!-- affichage de l'icone selon l'entête et selon si le tri est asc ou desc (up ou down) -->
33            <span [hidden]="entete.orderType || entete.orderby !== orderbyColonne">
34              <i class="fa fa-caret-up" aria-hidden="true"></i>
35            </span>
36            <span [hidden]!="entete.orderType || entete.orderby !== orderbyColonne">
37              <i class="fa fa-caret-down" aria-hidden="true"></i>
38            </span>
39          </th>
40        </tr>
41      </thead>
42      <!-- body du tableau -->
43      <tbody>
44        <!-- boucle sur la variable qui contient la liste des tutos avec un event "click" sur chaque tuto-->
45        <tr *ngFor="let tuto of liste | filter : search" (click)="onSelect(tuto)">
46          <td>{{tuto._id}}</td>
47          <td>{{tuto.marque}}</td>
48          <td>{{tuto.type}}</td>
49        </tr>
50      </tbody>
51    </table>
52  </div>
53 </div>
54
55  <!-- affichage du message -->
56  <h4>{{message}}</h4>

```

```
54      <!-- affichage du message --&gt;
55      &lt;h4&gt;{{message}}&lt;/h4&gt;
56
57
58      <!-- si aucun customer est sélectionné on affiche le bouton de création --&gt;
59      &lt;div *ngIf="selected == null" class="col-md-12 text-center"&gt;
60          &lt;button class="btn btn-success pull-center" type="button" (click)="onCreate()"&gt;Créer un tutoriel&lt;/button&gt;
61      &lt;/div&gt;
62
63      <!-- si un customer est sélectionné ou créé on affiche : --&gt;
64      &lt;div *ngIf="selected"&gt;
65          &lt;div class="row"&gt;
66              &lt;div class="col-md-12 text-center"&gt;
67                  &lt;button class="btn btn-warning pull-center" type="button" (click)="selected = null"&gt;Annuler&lt;/button&gt;
68              &lt;/div&gt;
69
70              &lt;br&gt;
71              &lt;!-- div pour un affichage 2vides-8pleins-2vides sur une grille bootstrap de 12 --&gt;
72              &lt;div class="col-md-8 col-md-offset-2"&gt;
73                  &lt;div class="form-group"&gt;
74                      &lt;label class="control-label labelFormulaire"&gt;_id&lt;/label&gt;
75                      &lt;!-- ngModel relie cet input à l'attribut selected et plus précisement au champs _id --&gt;
76                      &lt;input type="text" [(ngModel)]="selected._id" class="form-control" disabled&gt;
77                  &lt;/div&gt;
78
79                  &lt;div class="form-group"&gt;
80                      &lt;label class="control-label labelFormulaire"&gt;Marque&lt;/label&gt;
81                      &lt;!-- ngModel permet une relation bidirectionnelle : l'input affiche le contenu de l'attribut et l'input peut modifier instantanément la valeur de l'attribut --&gt;
82                      &lt;input type="text" [(ngModel)]="selected.marque" class="form-control"&gt;
83                  &lt;/div&gt;
84
85                  &lt;div class="form-group"&gt;
86                      &lt;label class="control-label labelFormulaire"&gt;Type&lt;/label&gt;
87                      &lt;input type="text" [(ngModel)]="selected.type" class="form-control"&gt;
88                  &lt;/div&gt;
89
90                  &lt;div class="form-group"&gt;
91                      &lt;label class="control-label labelFormulaire"&gt;Pages Tutoriaux&lt;/label&gt;</pre>
```

```
85 | <div class="form-group">
86 |   <label class="control-label labelFormulaire">Pages Tutoriaux</label>
87 |   <div class="row">
88 |     <!-- boucle sur le tableau des tutos, récupération de l'index des tutos pour faciliter la suppression dans le tableau -->
89 |     <div class="col-xs-11" *ngFor="let pageTuto of selected.tuto; let indexTuto= index">
90 |       <!-- div parent pour faciliter les affichages via le css -->
91 |       <div class="well parentIconeRemove">
92 |         <!-- icone de suppression, au click on passe l'index du tuto en question pour le supprimer dans le tableau des tutos -->
93 |         <i class="iconRemove remove glyphicon glyphicon-remove-sign glyphicon-white" (click)="supprTutoPage(indexTuto)"></i>
94 |         <!-- creation d'une grille dans une grille pour créer une marge -->
95 |         <div class="row">
96 |           <div class="col-xs-10 col-xs-offset-1">
97 |             <div class="row">
98 |               <label class="control-label labelFormulaire">Titres appairages</label>
99 |               <div class="col-xs-10 col-xs-offset-1">
100 |                 <div class="form-group">
101 |                   <label class="control-label labelFormulaire">FR</label>
102 |                   <input type="text" [(ngModel)]="pageTuto.titreAppairage.FR" class="form-control">
103 |                   <label class="control-label labelFormulaire">EN</label>
104 |                   <input type="text" [(ngModel)]="pageTuto.titreAppairage.EN" class="form-control">
105 |                 </div>
106 |               </div>
107 |             </div>
108 |             <div class="row">
109 |               <label class="control-label labelFormulaire">Textes appairages</label>
110 |               <div class="col-xs-10 col-xs-offset-1">
111 |                 <div class="form-group">
112 |                   <label class="control-label labelFormulaire">FR</label>
113 |                   <input type="text" [(ngModel)]="pageTuto.texteAppairage.FR" class="form-control">
114 |                   <label class="control-label labelFormulaire">EN</label>
115 |                   <input type="text" [(ngModel)]="pageTuto.texteAppairage.EN" class="form-control">
116 |                 </div>
117 |               </div>
118 |             </div>
119 |             <div class="row">
120 |               <label class="control-label labelFormulaire">Textes bases Appairages</label>
121 |               <div class="col-xs-10 col-xs-offset-1">
122 |                 <div class="form-group">
123 |                   <label class="control-label labelFormulaire">FR</label>
124 |                   <input type="text" [(ngModel)]="pageTuto.texteBaseAppairage.FR" class="form-control">
125 |                   <label class="control-label labelFormulaire">EN</label>
126 |                   <input type="text" [(ngModel)]="pageTuto.texteBaseAppairage.EN" class="form-control">
127 |                 </div>
128 |               </div>
129 |             </div>
130 |             <div class="row">
131 |               <label class="control-label labelFormulaire">Fonction appairage</label>
132 |               <div class="col-xs-10 col-xs-offset-1">
133 |                 <div class="form-group">
134 |                   <input type="text" [(ngModel)]="pageTuto.fonctionAppairage" class="form-control">
135 |                 </div>
136 |               </div>
137 |             </div>
```

```
138 |         <div class="row">
139 |             <label class="control-label labelFormulaire">Illustration</label>
140 |             <div class="col-xs-10 col-xs-offset-1">
141 |                 <div class="form-group">
142 |                     <p>Sélectionnez une image dans vos fichiers. </p>
143 |                     <!-- input type filé acceptant que les image jpeg et png et gif, choix multiple autorisé, quand l'input change on ajoute les images dans la liste temporaire d'image -->
144 |                     <input type="file" accept="image/jpeg,image/png,image/gif" multiple class="form-control" (change)="imageFileOnChange($event)">
145 |                     <!-- affichage de la liste temporaire des image à ajouter -->
146 |                     <label *ngIf="inputImageFileTemp.length != 0">Image à ajouter : </label>
147 |                     
149 |                 </div>
150 |                 <br>
151 |                 <!-- ajout à la liste d'illustration de la page de tuto (correspondant au tuto indexé "indexTuto") les images de la liste temporaire-->
152 |                 <button class="btn btn-info pull-center" type="button" (click)="encodeImage(indexTuto)" [disabled]="inputImageFileTemp.length == 0">Charger</button>
153 |                 <br>
154 |                 <br>
155 |                 <!-- vérification de la présence d'image dans la liste d'illustration -->
156 |                 <div *ngIf="pageTuto.illustrAppairage != null">
157 |                     <div class="row">
158 |                         <!-- boucle sur le tableau (liste) des illustration de cette page de tuto -->
159 |                         <div *ngFor="let imageBase64 of pageTuto.illustrAppairage; let j = index">
160 |                             
161 |                         </div>
162 |                     </div>
163 |                     </div>
164 |                     <div class="col-xs-10 col-xs-offset-1">
165 |                         <p>cliquer sur les images pour les supprimer des listes</p>
166 |                     </div>
167 |                     </div>
168 |                 </div>
169 |                 </div>
170 |             </div>
171 |             <i class="fa fa-plus-circle fa-2x col-xs-1" aria-hidden="true" (click)="ajoutTuto()"/>
172 |         </div>
173 |         <div class="row">
174 |             <div class="col-md-3 text-left">
175 |                 <button class="btn btn-primary pull-center" type="button" (click)="onSave()">Save</button>
176 |             </div>
177 |             <!-- si l'id du customer sélectionné est différent de undefined, on affiche le bouton de suppression -->
178 |             <div *ngIf="selected._id != undefined" class="col-md-9 text-right">
179 |                 <button class="btn btn-danger pull-center" type="button" (click)="onDelete(selected._id)">Supprimer le tutoriel {{selected.marque}} {{selected.type}}</button>
180 |             </div>
181 |         </div>
182 |         <br>
183 |         <br>
184 |     </div>
185 | 
```

### 6.1.13 - Tuto.ts

#### Les attributs modèles de la classe

```
17 export class TutoAppairageComponent implements OnInit {
18   ****
19   * ATTRIBUTS
20   ****
21   search = ''; // barre de recherche mise à chaîne vide
22   orderbyColonne = "marque"; // ordre de tri sur marque par défaut
23   orderType: boolean = false; // false par défaut pour un tri asc
24   // listes des entêtes des tutos pour le tableau
25   entetes = [
26     { label: "ID", orderby: "_id", orderType: false },
27     { label: "marque", orderby: "marque", orderType: false },
28     { label: "Etat cnx", orderby: "type", orderType: false },
29   ];
30   // liste des tuto instanciée
31   liste: any = [];
32   // tuto sélectionné instancié à null
33   selected = null;
34   // titre
35   titre = "Liste des tutoriaux d'appairages";
36   // variable message pour afficher les erreurs
37   message: string = "";
38   // variable images qui contient l'image du tuto avant d'être modifié
39   imageFile;
40   // tableau des image file avant encodage
41   inputImageFileTemp = [];
42
43   ****
44   * CONSTRUTOR
45   ****
46   constructor(private apiService: ApiService, private authService: AuthenticationService,
47     private router: Router, private fb: FormBuilder, private sanitizer: DomSanitizer) { }
```

## Les méthodes contrôleur de la classe

```
48  ****
49  * Méthodes
50  ****
51 /**
52 * méthode du lifeCycle de angular qui est lancée à la création de ce component
53 */
54 ngOnInit() {
55     console.log("On init tutoAppairage");
56     //utilisation de la méthode getAll
57     this.getAll();
58 }
59 /**
60 * méthodes
61 */
62 Complexity is 6 It's time to do something...
63 █ getAll() {
64     // utilisation de la méthode getAll de l' ApiService
65     Complexity is 4 Everything is cool!
66     this.apiService.getAll("tutoAppairage").then( █ (response: any) => {
67         console.log(response);
68         // en cas de reponse avec un success à true
69         if (response.success) {
70             // on récupère la liste
71             this.liste = response.data;
72         } else {
73             // sinon on est déconnecté
74             console.log(response.message);
75             this.authService.logout();
76             this.router.navigate(['/login']);
77         }
78     })
79     .catch((err) => {
80         // en cas d'erreur système on est déconnecté
81         this.authService.logout();
82         this.router.navigate(['/login']);
83         console.log(err + " : Une erreur système");
84     })
85 }
```

```
88 onSelect(tuto) {
89     // le tutoriel clické dans la liste des tutos devient le selected
90     this.selected = tuto;
91     this.message = "";
92 }
93
94 onRefresh() {
95     //mise à zero de la zone de recherche
96     this.search = "";
97     //appel de la méthode getAll pour rafraîchir la le tableau
98     this.getAll();
99     this.message = "";
100 }
101
102 onCreate() {
103     //à la création d'un tuto, on affecte à la variable selected cet objet
104     this.selected = {
105         'marque': "",
106         'type': "",
107         'tuto': []
108     };
109     this.message = "";
110 }
111
112 ajoutTuto() {
113     //ajout d'un objet dans le tableau des tuto
114     this.selected.tuto.push({
115         titreAppairage: { FR: "", EN: "" },
116         texteAppairage: { FR: "", EN: "" },
117         illustrAppairage: [],
118         texteBaseAppairage: { FR: "", EN: "" },
119         fonctionAppairage: 0
120     });
121 }
122
123 supprTutoPage(pos) {
124     if (window.confirm('Etes-vous sûre de supprimer la page de ce tutoriel?')) {
125         // suppression de l'objet dans l'array tuto grâce à son index
126         this.selected.tuto.splice(pos, 1);
127     }
128 }
```

## Gestion des images des pages de tutos.

```
130    supprImageInInputImageFileTemp(posInArrayIllustrer) {
131        if (window.confirm('Etes-vous sûre de supprimer la photo de cet liste?')) {
132            // suppression de l'objet dans l'array illustrAppairage grace à son index
133            this.inputImageFileTemp.splice(posInArrayIllustrer, 1);
134        }
135    }
136
137    supprImageSelectedIllustrer(posInArrayIllustrer, posInArrayTuto) {
138        if (window.confirm('Etes-vous sûre de supprimer la photo de ce tutoriel?')) {
139            // suppression de l'objet dans l'array illustrAppairage grace à son index
140            this.selected.tuto[posInArrayTuto].illustrAppairage.splice(posInArrayIllustrer, 1);
141        }
142    }
143
144    Complexity is 3 Everything is cool!
145    ■ imageFileOnChange(event) {
146        //recupération des fichiers de l'input file
147        var files = event.target.files;
148        //s'il y a des fichiers
149        if (files) {
150            //pour chaque fichier
151            for (let i = 0; i < files.length; i++) {
152                // on instancie un lecteur de fichiers
153                var reader = new FileReader();
154                let file = files[i];
155                // fonction evenement lors du chargement du ou des fichier
156                reader.onload = this._handleReaderLoaded.bind(this);
157                //lecture du fichier
158                reader.readAsBinaryString(file);
159            }
160        }
161
162        _handleReaderLoaded(readerEvt) {
163            // transformation en base64 des fichier images
164            var binaryString = readerEvt.target.result;
165            //stockage des image dans la liste temporaire
166            this.inputImageFileTemp.push(btoa(binaryString));
167        }
168
169        Complexity is 3 Everything is cool!
170        ■ encodeImage(index) {
171            if (window.confirm('Etes-vous sûre de charger la ou les photos dans ce tutoriel?')) {
172                for (let i = 0; i < this.inputImageFileTemp.length; i++) {
173                    let image = this.inputImageFileTemp[i];
174                    // les images sont en base64
175                    this.selected.tuto[index].illustrAppairage.push(image);
176                }
177                this.inputImageFileTemp = [];
178            }
179        }
```

## Gestion de la sauvegarde (insert ou update).

```
171  ■ onSave() {
172    console.log(this.selected._id);
173    //alerte de confirmation
174    if (window.confirm('Etes-vous sûre d\'enregistrer cet item ?')) {
175      // si _id est undefined, il s'agit donc d'une création de tuto
176      if (this.selected._id == undefined) {
177        // on fait appel à la méthode insertOne de l'ApiService
178        Complexity is 4 Everything is cool!
179        this.apiService.insertOne("tutoAppairage", this.selected).then( ■ (response: any) => {
180          console.log(response);
181          // en cas de success
182          if (response.success) {
183            console.log("data : " + response);
184            this.message = response.message;
185            console.log("l'objet a été inséré");
186            this.selected = null;
187            this.getAll();
188            // sinon
189          } else {
190            console.log(response.message)
191            this.message = response.message;
192          }
193        })
194        //en cas d'erreur
195        .catch((err) => {
196          this.message = err;
197          console.log(err + " : Une erreur système");
198        })
199      // si _id est pas undefined alors c'est un update du tuto
200    } else {
201      // on fait appel à la méthode updateOne de l'ApiService
202      Complexity is 4 Everything is cool!
203      this.apiService.updateOne("tutoAppairage", this.selected).then( ■ (response: any) => {
204        console.log(response);
205        // en cas de success on recharge la liste des customers et on met le tuto selected à null
206        if (response.success) {
207          this.message = response.message;
208          console.log("l'objet avec _id '" + this.selected._id + "' a été modifié");
209          this.selected = null;
210          this.getAll();
211          //sinon en cas de non success
212        } else {
213          console.log(response.message)
214          this.message = response.message;
215        }
216      })
217      //en cas d'erreur
218      .catch((err) => {
219        this.message = err;
220        console.log(err + " : Une erreur système");
221      })
222    }
223  }
```

## Et la suppression de tutoriel.

```
224  ┃ ┃ ┃ onDelete(psId) {  
225  ┃ ┃ ┃ //alerte de confirmation  
226  ┃ ┃ ┃ if (window.confirm('Etes-vous sûre de supprimer cet item ?')) {  
227  ┃ ┃ ┃ // on fait appel à la méthode deleteOne de l'ApiService  
228  ┃ ┃ ┃ Complexity is 4 Everything is cool!  
229  ┃ ┃ ┃ this.apiService.deleteOne("tutoAppairage", psId).then( ┃ (response: any) => {  
230  ┃ ┃ ┃   console.log(response);  
231  ┃ ┃ ┃   // en cas de success on recharge la liste des customers et on met le tuto selected à null  
232  ┃ ┃ ┃   if (response.success) {  
233  ┃ ┃ ┃     console.log("data : " + response);  
234  ┃ ┃ ┃     this.message = response.message;  
235  ┃ ┃ ┃     this.selected = null;  
236  ┃ ┃ ┃     this.getAll();  
237  ┃ ┃ ┃     console.log("l'objet avec _id '" + psId + "' a été supprimé");  
238  ┃ ┃ ┃   //sinon en cas de non success  
239  ┃ ┃ ┃ } else {  
240  ┃ ┃ ┃     console.log(response.message)  
241  ┃ ┃ ┃     this.message = response.message;  
242  ┃ }  
243  ┃ }  
244  ┃ //en cas d'erreur  
245  ┃ .catch((err) => {  
246  ┃   this.message = err;  
247  ┃   console.log(err + " : Une erreur système");  
248  ┃ })  
249  }  
250 }
```

### 6.1.14 - Api.service.ts

Méthode qui crée le header pour les requêtes http.

```

106  private buildHeaders() {
107    var token = this.auth.getToken();
108    let headers = new Headers({ 'Content-Type': 'application/json; charset=utf-8' });
109    headers.set('Accept', 'text/plain');
110    headers.set('x-access-token', token);
111    return headers;
112  }

```

Méthodes du CRUD (génériques) reliant les requêtes http du client (Angular ici) vers l'Api Rest côté server :

- Le SelectAll.

```

114  getAll(psRouteCollection) {
115    var url = this.adresseAPI + "/" + psRouteCollection + "/";
116    return new Promise(
117      Complexity is 4 Everything is cool!
118      (resolve, reject) => {
119        //utilisation de la méthode pour contruire le headers et l'option part la suite
120        let headers = this.buildHeaders();
121        // structure du parameters object - mais pas utilisé avec le front angular
122        // let params: URLSearchParams = new URLSearchParams();
123        // let params = JSON.stringify({ searchParams: { conditions:{Ville: "Paris", Cp : 75015}},filterResult : "_id , Label , Ville"});
124        // console.log(params);
125        let params = "";
126        let options = new RequestOptions({ headers: headers, params: { params: params } });
127        console.log(options);
128        //Appel asynchrone au backend
129        this.http.get(url, options).subscribe(
130          //callback de la requête http (succès)
131          (response) => {
132            //transformation de la réponse texte en objet json
133            var data = response.json();
134            //si le
135            if (data) {
136              //Envoi des données à la page qui a initié la promesse
137              resolve(data);
138            }
139          },
140          //callback d'erreur http
141          (error) => {
142            console.log("callback error " + error);
143            reject(JSON.parse(error._body).message);
144          }
145        );
146      }
147    );
148  }

```

- L'insert.

```
149 |■ insertOne(psRouteCollection, poBody) {  
150 |    var url = this.adresseAPI + "/" + psRouteCollection + "/";  
151 |    return new Promise(  
152 |        Complexity is 4 Everything is cool!  
153 |        ■ (resolve, reject) => {  
154 |            //utilisation de la methode pour construire le headers et l'option part la suite ainsi que le body  
155 |            let headers = this.buildHeaders();  
156 |            let options = new RequestOptions({ headers: headers });  
157 |            let body = { "data": poBody };  
158 |            //Appel asynchrone au backend  
159 |            this.http.post(url, body, options).subscribe(  
160 |                //callback de la requête http (succès)  
161 |                (response) => {  
162 |                    //transformation de la réponse texte en objet json  
163 |                    var data = response.json();  
164 |                    //si la réponse contient un objet json  
165 |                    if (data) {  
166 |                        //Envoi des données à la page qui a initié la promesse  
167 |                        resolve(data);  
168 |                    }  
169 |                },  
170 |                //callback d'erreur  
171 |                (error) => {  
172 |                    console.log("callback error : " + error);  
173 |                    //on rejette le message contenu dans le json reçu de l'api  
174 |                    reject(JSON.parse(error._body).message);  
175 |                }  
176 |            );  
177 |        }  
178 |    }  
179 |}
```

- L'update.

```
181  ■ updateOne(psRouteCollection, pObject) {
182    var url = this.adresseAPI + "/" + psRouteCollection + "/" + pObject._id;
183    return new Promise(
184      Complexity is 4 Everything is cool
185      ■ (resolve, reject) => {
186        //utilisation de la méthode pour construire les headers et l'option part la suite ainsi que le body
187        let headers = this.buildHeaders();
188        let options = new RequestOptions({ headers: headers });
189        let body = { "data": pObject };
190        //Appel asynchrone au backend
191        this.http.put(url, body, options).subscribe(
192          //callback de la requête http (succès)
193          (response) => {
194            //transformation de la réponse texte en objet json
195            var data = response.json();
196            //si le
197            if (data) {
198              //Envoi des données à la page qui a initié la promesse
199              resolve(data);
200            }
201          },
202          //callback d'erreur http
203          (error) => {
204            console.log("callback error " + error);
205            //on rejette le message contenu dans le json reçu de l'api
206            reject(JSON.parse(error._body).message);
207          }
208        );
209      );
210    }
}
```

- Le delete.

```
244  ┌─▶ deleteOne(psRouteCollection, psId) {  
245    var url = this.adresseAPI + "/" + psRouteCollection + "/" + psId;  
246    return new Promise(  
247      Complexity is 4 Everything is cool!  
248      ┌─▶ (resolve, reject) => {  
249        //utilisation de la méthode pour construire les headers et l'option part la suite  
250        let headers = this.buildHeaders();  
251        let options = new RequestOptions({ headers: headers });  
252        //Appel asynchrone au backend  
253        this.http.delete(url, options).subscribe(  
254          //callback de la requête http (succès)  
255          (response) => {  
256            //transformation de la réponse texte en objet json  
257            var data = response.json();  
258            //si le  
259            if (data) {  
260              //Envoi des données à la page qui a initié la promesse  
261              resolve(data);  
262            }  
263          },  
264          //callback d'erreur http  
265          (error) => {  
266            console.log(error);  
267            //on rejette le message contenu dans le json reçu de l'api  
268            reject(JSON.parse(error._body).message);  
269          }  
270        );  
271      }  
272    }  
273  };
```

## 6.2 - LES CONTROLS DE SAISIES

Angular propose un module « FormBuilder » qui construit les formulaires et propose des validations de saisies type require (champ requis), minLength (longueur de saisie minimum), pattern (pouvant être une expression régulière).

Avec ces validations, nous pouvons facilement interagir avec le css.

Exemple où on cache le label des recommandations de saisies si les saisies sont valides :

```
88 <div [formGroup]="rForm">
89   <div class="col-md-6 col-md-offset-3">
90     <div class="form-group">
91       <label class="control-label labelFormulaire">_id</label>
92       <input type="text" [(ngModel)]="customerSelected._id" class="form-control" formControlName="_id">
93     </div>
94     <div class="form-group">
95       <label class="control-label labelFormulaire">Label</label>
96       <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Label'].valid">*Champs requis et doit comporter entre 3 et 20 caractères</label>
97       <input type="text" [(ngModel)]="customerSelected.Label" class="form-control" formControlName="Label">
98     </div>
99     <div class="form-group">
100       <label class="control-label labelFormulaire">Ref_client</label>
101       <label class="alert-danger labelFormulaire" [hidden]="rForm.controls['Ref_client'].valid">*Champs requis et doit comporter entre 3 et 20 caractères</label>
102       <input type="text" [(ngModel)]="customerSelected.Ref_client" class="form-control" formControlName="Ref_client">
103     </div>
```

Exemple où l'on désactive le bouton de sauvegarde si le formulaire n'est pas complètement valide :

```
142   <div class="row">
143     <div class="col-md-3 text-left">
144       <button class="btn btn-primary pull-center" type="button" (click)="onSave()" [disabled]="!rForm.valid">Save</button>
145   </div>
```

La création du formulaire permettant la liaison entre les HTML et le script :

## ***CHAPITRE 7 - DEVELOPPEMENT PARTIE BACK***

## 7.1 - DEVELOPPEMENT

Au cours des différents codes qui vont défiler, je vais également garder une logique séquentielle (celle du diagramme de séquences détaillées) :

- Codes du contrôleur,
- Codes de l'entité (modèle mongoose),
- Codes DAO (Data-Access-Object).

On y détaillera premier lieu la structure de l'api avec le script principal. Ensuite la gestion du login et des tutos.



J'ai organisé le projet de manière à distinguer des couches :

- Le server.js (permettant le routage),
- Les contrôleurs,
- Les modèles (entités),
- Les modules (DAO et méthodes).

### 7.1.1 - Création du server

```
1 console.log("*****");
2 console.log("Welcome to Otodo ApiRest v1.0.0");
3 console.log("*****");
4 /**
5  * IMPORT
6  */
7 //Import du module express
8 const express = require('express');
9 //import conf
10 const conf = require('./config');
11 //Création du serveur express
12 const app = express();
13 // constante du port utilisé par le server
14 const PORT = process.env.PORT || 8280;
15 //Import d'un module pour gérer les données transmises en POST
16 const bodyParser = require('body-parser');
17 //Import d'un module de gestion des cookies
18 const cookieParser = require('cookie-parser');
19 //Import du module FileSystem
20 const fs = require('fs');
21 //Importation du module de connexion
22 const dbConnection = require('./module/db-connection');
23 //import des méthodes globale de l'api
24 const methodesApi = require('./module/MethodesApi')
25 //Import du module jwt
26 const jsonWebToken = require('jsonwebtoken');
27 // import des routes et conf
28 const alerteursRoutes = require('./routesCTRL/alerteurs-routes');
29 const alertsRoutes = require('./routesCTRL/alerts-routes');
30 const codeRoutes = require('./routesCTRL/code-routes');
31 const countRoutes = require('./routesCTRL/count-routes');
32 const customerRoutes = require('./routesCTRL/customer-routes');
33 const deviceRoutes = require('./routesCTRL/device-routes');
34 const deviceTypeRoutes = require('./routesCTRL/deviceType-routes');
35 const enregistrementUserRoutes = require('./routesCTRL/enregistrementUser-routes');
36 const evenementRoutes = require('./routesCTRL/evenement-routes');
37 const hubRoutes = require('./routesCTRL/hub-routes');
38 const interactionLogRoutes = require('./routesCTRL/interactionLog-routes');
39 const logRoutes = require('./routesCTRL/log-routes');
40 const monitoringRoutes = require('./routesCTRL/monitoring-routes');
41 const protocolRoutes = require('./routesCTRL/protocol-routes');
42 const sceneRoutes = require('./routesCTRL/scene-routes');
43 const signRoutes = require('./routesCTRL/sign-routes');
44 const triggerTypeRoutes = require('./routesCTRL/triggerType-routes');
45 const tutoAppairageRoutes = require('./routesCTRL/tutoAppairage-routes');
46 const universRoutes = require('./routesCTRL/univers-routes');
47 const userRoutes = require('./routesCTRL/user-routes');
48 const utilisateurApiRoutes = require('./routesCTRL/utilisateurApi-routes');
49
50 // appel du module cors
51 const cors = require('cors');
```

```
70 //*****
71 * MIDDLEWARE
72 *****/
73 //Gestion des CORS et du cross-domain
74 app.use(cors({ origin: '*', optionsSuccessStatus: 200 }));
75 app.use(function (req, res, next) {
76   res.header("Access-Control-Allow-Origin", "*");
77   res.header('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT, OPTIONS');
78   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, Authorization, x-access-token");
79   res.header("Access-Control-Expose-Headers", "x-access-token");
80   next();
81 });
82 // autorise le cross-domaine sur toutes les routes
83 app.all('/', function(req, res, next) {
84   res.header("Access-Control-Allow-Origin", "*");
85   res.header("Access-Control-Allow-Headers", "X-Requested-With");
86   next();
87 });
--
```

```
90 //*****
91 * ROUTES
92 *****/
93 //First Routes qui renvoie vers l'index de angular4 (front independant)
94 app.get('/', function(req, res) {
95   res.sendFile('../dist/index.html');
96 });
97 //Routes du login
98 app.use('/sign', signRoutes);
99 //Routes sécurisé par une fonction middleware requireRole
100 app.use('/api/alerteurs', methodesApi.requireRole(["alerteurs"]), alerteursRoutes);
101 app.use('/api/alerts', methodesApi.requireRole(["alerts"]), alertsRoutes);
102 app.use('/api/code', methodesApi.requireRole(["code"]), codeRoutes);
103 app.use('/api/count', methodesApi.requireRole(["count"]), countRoutes);
104 app.use('/api/customer', methodesApi.requireRole(["customer"]), customerRoutes);
105 app.use('/api/device', methodesApi.requireRole(["device"]), deviceRoutes);
106 app.use('/api/deviceType', methodesApi.requireRole(["deviceType"]), deviceTypeRoutes);
107 app.use('/api/evenement', methodesApi.requireRole(["evenement"]), evenementRoutes);
108 app.use('/api/hub', methodesApi.requireRole(["hub"]), hubRoutes);
109 app.use('/api/enregistrementUser', methodesApi.requireRole(["enregistrementUser"]), enregistrementUserRoutes);
110 app.use('/api/interractionLog', methodesApi.requireRole(["interractionLog"]), interractionLogRoutes);
111 app.use('/api/log', methodesApi.requireRole(["log"]), logRoutes);
112 app.use('/api/monitoring', methodesApi.requireRole(["monitoring"]), monitoringRoutes);
113 app.use('/api/protocol', methodesApi.requireRole(["protocol"]), protocolRoutes);
114 app.use('/api/scene', methodesApi.requireRole(["scene"]), sceneRoutes);
115 app.use('/api/triggerType', methodesApi.requireRole(["triggerType"]), triggerTypeRoutes);
116 app.use('/api/tutoAppairage', methodesApi.requireRole(["tutoAppairage"]), tutoAppairageRoutes);
117 app.use('/api/univers', methodesApi.requireRole(["univers"]), universRoutes);
118 app.use('/api/user', methodesApi.requireRole(["user"]), userRoutes);
119 app.use('/api/utilisateurApi', methodesApi.requireRole(["utilisateurApi"]), utilisateurApiRoutes);
--
```

```
121 //*****
122 * LANCEMENT DU SERVER
123 *****/
124 //API sur ecoute du FBI
125 app.listen(PORT);
```

### 7.1.2 - Route login (contrôleur)

```

35 //Traitement du login utilisateur
Complexity is 9 It's time to do something...
36 router.post('/login', (request, response) => {
37     console.log("///////////---UTILISATION API---/////////");
38     console.log("*====> méthode Login : *");
39     //Récupération des données postées
40     var email = request.body.email;
41     var mdp = request.body.mdp;
42     console.log(request.body);
43     // on poste les params dans une variable
44     var params = [email, mdp];
45     //Test de l'existence d' l'utilisateur via une fonction qui retourne une promesse
46     checkLogin(params).then(
47         //Callback de succès
        Complexity is 6 It's time to do something...
48         (data) => {
49             //Cas d'un utilisateur absent de la base de données
50             if (data == null) {
51                 console.log({ success: false, message: 'Utilisateur non trouvé', token: null });
52                 console.log("*<==== fin méthode Login : *");
53                 response.status(404).json({ success: false, message: 'Utilisateur non trouvé', token: null });
54                 //return response.status(404).send();
55             } else {
56                 //Récupération des données de l'utilisateur
57                 var utilisateur = data;
58                 //appel de la méthode de moongoose pour la mise a jour du champs lastConnection
Complexity is 3 Everything is cool!
59                 utilisateursApiModel.findByIdAndUpdate(utilisateur._id, utilisateur, { new: true }, function (err, data) {
60                     if (data) {
61                         //Récupération des données de l'utilisateur
62                         var utilisateur = data;
63                         //Création du token
64                         var token = jsonWebToken.sign(utilisateur, secret, { expiresIn: "5h" });
65                         console.log({ success: true, message: 'Vous êtes authentifiés', token: "un token long", utilisateur: utilisateur });
66                         console.log("*<==== fin méthode Login : *");
67                         //Envoi de la réponse de succès
68                         response.status(200).json({ success: true, message: 'Vous êtes authentifiés', token: token, utilisateur: utilisateur });
69                     } else {
70                         console.log({ success: false, message: 'Erreur lors de l\'authentication', token: null });
71                         console.log("*<==== fin méthode Login : *");
72                         response.status(401).json({ success: false, message: 'Erreur lors de l\'authentication', token: null });
73                     }
74                 });
75             }
76         },
77         //callback d'échec
78         (error) => {
79             console.log("err checkLogin = " + error);
80             response.json({ success: false, message: 'Erreur' });
81         }
82     ).catch(function (error) {
83         console.log("catch err checkLogin = " + error)
84     });
85 });
-->
```

```
17 //methode qui verifie le login
Complexity is 6 It's time to do something...
18 var checkLogin = (params) => {
19     return new Promise(
        Complexity is 4 Everything is cool!
20         (resolve, reject) => {
21             //Récupération des données postées
22             var email = params[0];
23             var mdp = params[1];
24             var mdpHashe = require('crypto').createHash('sha256').update(mdp).digest('hex');
Complexity is 3 Everything is cool!
25             utilisateursApiModel.findOne({ email: email, mdp: mdpHashe }, (err, rows) => {
26                 if (err) {
27                     reject(err);
28                 } else {
29                     resolve(rows);
30                 }
31             });
32         });
33     );
};
```

### 7.1.3 - UtilisateursApi.js (entités)

```
1 //Imports
2 const conf = require('../config');
3 var mongoose = require('mongoose');
4 mongoose.Promise = global.Promise;
5 var Schema = mongoose.Schema;
6 const crypto = require('crypto');
7 const sha256 = crypto.createHash('sha256');
8 /**
9 * Mongoose Schema UtilisateursApi
10 * @module models/utilisateursApi
11 */
12 /**
13 * Mongoose Shema UtilisateursApi
14 * @inner
15 * @type {Schema}
16 * @property {String} pseudo - pseudo
17 * @property {String} email - email
18 * @property {String} mdp - mot de passe
19 * @property {Array} droits - droits de l'utilisateur pour accéder au routes
20 * @property {Date} dateInscription - date de creation de
21 * @property {Date} lastConnection - dernière co
22 */
23 const UtilisateursApiSchema = new Schema({
24     pseudo: String,
25     email: { type: String, required: true, unique: true },
26     mdp: {
27         type: String, required: true, set: value => {
28             return require('crypto').createHash('sha256').update(value).digest('hex');
29         }
30     },
31     droits: [],
32     dateInscription: { type: Date, default: Date.now },
33     lastConnection : { type: Date, default: Date.now }
34 });
35 // à l'appel de la méthode save() cette méthode lance la fonction middleware juste avant
Complexity is 3 Everything is cool!
36 UtilisateursApiSchema.pre('save', function(next) {
37     var currentDate = new Date();
38     // if date doesn't exist, add to that field
39     if(!this.dateInscription) {
40         this.dateInscription = currentDate;
41     }
42     if(!this.pseudo) {
43         this.pseudo = this.email;
44     }
45     this.lastConnection = currentDate;
46     next();
47 });
48 //Instanciation du modèle
49 const UtilisateursApiModel = mongoose.model("utilisateursApi", UtilisateursApiSchema);
50
51 //exportation du modèle
52 module.exports = UtilisateursApiModel;
```

### 7.1.4 - Route tutoAppairage (contrôleur)

```

1 //imports
2 const express = require('express');
3 const router = express.Router();
4 //import du modèle
5 const model = require('../model/tutoAppairage');
6 //import des modules
7 const dao = require('../module/DaoApi');
8 const methodesApi = require('../module/MethodesApi');
```

- Le select all.

```

14 // appel d'un middleware requireRoleCRUD contrôlant l'accès à la route en fonction du CRUD
Complexity is 4 Everything is cool!
15 router.get('/', methodesApi.requireRoleCRUD("tutoAppairage", "read"), (request, response) => {
16     //gestion des paramètres dans la requête
17     var params = "";
18     if (request.query.params) {
19         params = JSON.parse(request.query.params);
20         console.log(params);
21     }
22     //appel de la méthode selectAll du dao de l'api, méthode then exécuté à la fin de la promesse pouvant faire appel
23     //jusqu'à 2 callback( une success, une error)
24     dao.selectAll(model, params).then(
25         //callback de success
26         (data) => {
27             //réponse à la requête http
28             response.status(data.status).json(data.json);
29         },
30         //callback d'erreur
31         (err) => {
32             //réponse à la requête http
33             response.status(500).json({ success: false, message: 'Erreur système lors de la recherche de datas', data: null });
34         }
35     );
36 })
```

- Le select one by id

```

38 // appel d'un middleware requireRoleCRUD contrôlant l'accès à la route en fonction du CRUD
Complexity is 3 Everything is cool!
39 .get('/:id', methodesApi.requireRoleCRUD("tutoAppairage", "read"), (request, response) => {
40     //appel de la méthode selectOneById du dao de l'api, méthode then exécuté à la fin de la promesse pouvant faire appel
41     // jusqu'à 2 callback( une success, une error)
42     dao.selectOneById(model, request.params.id).then(
43         //callback de success
44         (data) => {
45             //réponse à la requête http
46             response.status(data.status).json(data.json);
47         },
48         //callback d'erreur
49         (err) => {
50             //réponse à la requête http
51             response.status(500).json({ success: false, message: 'Erreur système lors de la recherche de datas', data: null });
52         }
53     );
54 })
```

- L'insert

```

56 // appel d'un middleware requireRoleCRUD contrôlant l'accès à la route en fonction du CRUD
57 Complexity is 3 Everything is cool!
58 .post('/', methodesApi.requireRoleCRUD("tutoAppairage", "create"), (request, response) => {
59     //création d'un nouveau modèle
60     var newModel = new model(request.body.data);
61     //appel de la méthode insert du dao de l'api, méthode then exécuté à la fin de la promesse pouvant faire appel
62     //jusqu'à 2 callback( une success, une error)
63     dao.insert(newModel).then(
64         //callback de success
65         (data) => {
66             //réponse à la requête http
67             response.status(data.status).json(data.json);
68         },
69         //callback d'erreur
70         (err) => {
71             //réponse à la requête http
72             response.status(500).json({ success: false, message: 'Erreur système lors de l\'insertion de datas', data: null });
73         }
74     );
}

```

- L'update

```

76 // appel d'un middleware requireRoleCRUD contrôlant l'accès à la route en fonction du CRUD
77 Complexity is 3 Everything is cool!
78 .put('/:id', methodesApi.requireRoleCRUD("tutoAppairage", "update"), (request, response) => {
79     //récupération des données modifiées
80     var updateData = request.body.data;
81     //appel de la méthode update du dao de l'api, méthode then exécuté à la fin de la promesse pouvant faire appel
82     //jusqu'à 2 callback( une success, une error)
83     dao.update(model, request.params.id, updateData).then(
84         //callback de success
85         (data) => {
86             //réponse à la requête http
87             response.status(data.status).json(data.json);
88         },
89         //callback d'erreur
90         (err) => {
91             //réponse à la requête http
92             response.status(500).json({ success: false, message: 'Erreur système lors de la modification de données', data: null });
93         }
94     );
}

```

- Le delete

```

96 // appel d'un middleware requireRoleCRUD contrôlant l'accès à la route en fonction du CRUD
97 Complexity is 3 Everything is cool!
98 .delete('/:id', methodesApi.requireRoleCRUD("tutoAppairage", "delete"), (request, response) => {
99     //appel de la méthode delete du dao de l'api, méthode then exécuté à la fin de la promesse pouvant faire appel
100    //jusqu'à 2 callback( une success, une error)
101    dao.delete(model, request.params.id).then(
102        //callback de success
103        (data) => {
104            //réponse à la requête http
105            response.status(data.status).json(data.json);
106        },
107        //callback de success
108        (err) => {
109            //réponse à la requête http
110            response.status(500).json({ success: false, message: 'Erreur système lors de la suppression des données', data: null });
111        }
112    );
}

```

### 7.1.5 - DaoApi.js (DAO)

```
4  /**
5   * Objet DAO qui contient des méthodes génériques du CRUD+ (select, selectOne, insert, update, delete || create, read, update, delete + read one)
6  */
7 var dao = {
```

- Le select avec ou sans conditions et avec ou sans projection.

```
9  /**
10  * Méthode selectAll
11  *
12  * @param {any} pmModel
13  * @param {any} params
14  * @returns
15  */
16 Complexity is 12 You must be kidding
17 ■selectAll(pmModel, params) {
18   console.log("====> méthode selectAll, route: " + pmModel.modelName + ", if params = " + params);
19
20   // on retourne une promesse qui à pour argument 2 méthodes resolve() et reject() en cas de promesse tenue ou pas
21   // Complexity is 10 it's time to do something...
22   return new Promise( ■ (resolve, reject) => {
23     //préparation au log
24     log.model = pmModel.modelName;
25     log.methode = "selectAll";
26     // méthode find() de mongoose sur un model mongoose qui execute une fonction callback gérant erreur et les datas
27     var search = pmModel.find();
28     // variable de filtrages des champs à retourner instancié à chaîne vide
29     filterResult = "";
30     // si les paramètres ne sont pas vides
31     if (params != "") {
32       // si dans les params on a des paramètres de recherche, on :
33       if (params.searchParams) {
34         // Les paramètres de recherche
35         var searchParams = params.searchParams;
36         // si dans les paramètres de recherche on a des conditions :
37         if (searchParams.conditions) {
38           var tCondition = searchParams.conditions;
39           // on affecte à la variable search un model.find(avec les conditions)
40           search = pmModel.find(tCondition);
41         }
42       }
43       // si dans les params on a des filtres pour le résultat de la recherche, on :
44       if (params.filterResult) {
45         filterResult = params.filterResult;
46       }
47     }
48   })
49 }
```

```
46 //utilisation de la méthode find() avec ou sans condition, avec ou sans select des champs à retourner
47 Complexity is 5 Everything is cool!
48 search.select(filterResult).exec( ■ function (err, data) {
49     // si erreur on renvoie via la méthode reject des datas au format json
50     if (err) {
51         console.log("status 500, err : " + err);
52         log.statusResponse = 500;
53         new apiLogModel(log).save();
54         reject({ status: 500, json: { success: false, message: 'Erreur système lors de la recherche de datas', data: null } });
55     } else {
56         if (data == null) {// s'il n'y a pas de data trouvé on envoie via la méthode resolve des datas à nul
57             console.log("status 404, data = null");
58             log.statusResponse = 404;
59             new apiLogModel(log).save();
60             resolve({ status: 404, json: { success: false, message: 'les datas non trouvé', data: null } });
61         } else {// si les datas ont été trouvées, on envoie via la méthode resolve
62             console.log("status 200, route: " + pmModel.modelName);
63             log.statusResponse = 200;
64             new apiLogModel(log).save();
65             resolve({ status: 200, json: { success: true, message: "Les datas", data: data } });
66         }
67     }
68 });
69 },
```

- Le select one by id

```
78 ■ selectOneById(pmModel, psId) {
79
80     // on retourne une promesse qui à pour argument 2 méthodes resolve() et reject() en cas de promesse tenue ou pas.
81     Complexity is 6 It's time to do something...
82     return new Promise( ■ (resolve, reject) => {
83         //préparation au log
84         console.log("===== méthode 'selectOneById', route: " + pmModel.modelName + ", psId: " + psId);
85         log.modelUsed = pmModel.modelName;
86         log.methode = "selectOneById";
87         // méthode findById(qui à pour arguments l'id du document) de mongoose sur un model mongoose qui execute une fonction callback gérant erreur et les datas
88         Complexity is 5 Everything is cool!
89         pmModel.findById(psId).exec( ■ function (err, data) {
90             // si erreur on renvoie via la méthode reject des datas au format json
91             if (err) {
92                 console.log("status 500, err : " + err);
93                 log.statusResponse = 500;
94                 new apiLogModel(log).save();
95                 reject({ status: 500, json: { success: false, message: 'Erreur lors de la recherche des datas par id', data: null } });
96             } else {
97                 if (data == null) {// s'il n'y a pas de data trouvé on envoie via la méthode resolve des datas à nul
98                     console.log("status 404, data = null");
99                     log.statusResponse = 404;
100                     new apiLogModel(log).save();
101                     resolve({ status: 404, json: { success: false, message: "les datas n'ont pas été trouvées par id", data: null } });
102                 } else {
103                     console.log("status 200, data = some datas par id");
104                     log.statusResponse = 200;
105                     new apiLogModel(log).save();
106                     resolve({ status: 200, json: { success: true, message: "Les datas par id", data: data } });
107                 }
108             });
109         },
```

- L'insert.

```
111 /**
112  * Méthode insert
113  *
114  * @param {any} pmNewModel
115  * @returns
116 */
117 Complexity is 8 It's time to do something...
118 insert(pmNewModel) {
119     // on retourne une promesse qui à pour argument 2 méthodes resolve() et reject() en cas de promesse tenue ou pas.
120 Complexity is 6 It's time to do something...
121     return new Promise( (resolve, reject) => {
122         //préparation au log
123         console.log("====> méthode insert , route: " + pmNewModel.modelName);
124         log.modelUse = pmNewModel.modelName;
125         log.methode = "insert";
126         // méthode save() de mongoose sur un model mongoose qui execute une fonction callback gérant erreur et les datas
127 Complexity is 5 Everything is cool!
128         pmNewModel.save( function (err, data) {
129             // si erreur on renvoie via la méthode reject des datas au format json
130             if (err) {
131                 console.log("status 500, err : " + err);
132                 log.statusResponse = 500;
133                 new apiLogModel(log).save();
134                 reject({ status: 500, json: { success: false, message: 'Erreur lors de l\'inscription des datas', data: null } });
135             } else {
136                 if (data == null) {// s'il n'y a pas de data trouvé on envoie via la méthode resolve des datas à nul
137                     console.log("status 404, data = null");
138                     log.statusResponse = 500;
139                     new apiLogModel(log).save();
140                     resolve({ status: 404, json: { success: false, message: 'Erreur de l\'insertion des données', data: null } });
141                 } else {// si les datas ont été trouvées, on envoie via la méthode resolve
142                     console.log("status 200, data = some datas enregistrées");
143                     log.statusResponse = 200;
144                     new apiLogModel(log).save();
145                     resolve({ status: 200, json: { success: true, message: "Les datas sont enregistrées", data: data } });
146                 }
147             }
148         });
149     });
150 },
```

- L'update

```
159  █ update(pmModel, psId, dataUpdate) {
160
161    // on retourne une promesse qui à pour argument 2 méthodes resolve() et reject() en cas de promesse tenue ou pas
Complexity is 6 It's time to do something...
162    return new Promise( █ (resolve, reject) => {
163      //préparation au log
164      console.log("***** méthode update, route: " + pmModel.modelName + ", psId: " + psId + ", dataUpdate: " + dataUpdate);
165      log.modelUse = pmModel.modelName;
166      log.methode = "update";
167      // méthode findByIdAndUpdate(qui a pour argument l'id, les data modifiée, une option à true pour renvoyé les nouvelles données)
168      // de mongoose sur un model qui execute une fonction callback gérant erreur et les datas
Complexity is 5 Everything is cool
169      pmModel.findByIdAndUpdate(psId, dataUpdate, { new: true }, █ function (err, data) {
170        // si erreur on renvoie via la méthode reject des datas au format json
171        if (err) {
172          console.log("status 500, err : " + err);
173          log.statusResponse = 500;
174          new apiLogModel(log).save();
175          reject({ status: 500, json: { success: false, message: 'Erreur lors de la recherche de datas à modifier', data: null } });
176        } else {
177          if (data == null) {// s'il n'y a pas de data trouvé on envoie via la méthode resolve des datas à nul
178            console.log("status 404, data = null");
179            log.statusResponse = 404;
180            new apiLogModel(log).save();
181            resolve({ status: 404, json: { success: false, message: 'les datas à modifier non trouvé', data: null } });
182          } else {// si les datas ont été trouvées, on envoie via la méthode resolve
183            console.log("status 200, data = some datas modifiées");
184            log.statusResponse = 200;
185            new apiLogModel(log).save();
186            resolve({ status: 200, json: { success: true, message: "Les datas sont modifiées", data: data } });
187          }
188        }
189      });
190    });
191  },
```

- Le delete.

```

193 /**
194 * méthode delete
195 *
196 * @param {any} pmModel
197 * @param {any} psId
198 * @returns
199 */
200 Complexity is 8 It's time to do something-
201 delete(pmModel, psId) {
202
203     // on retourne une promesse qui à pour argument 2 méthodes resolve() et reject() en cas de promesse tenue ou pas
204     //Complexity is 6 It's time to do something-
205     return new Promise( [ (resolve, reject) => {
206         //préparation au log
207         console.log("===== méthode delete, route: " + pmModel.modelName + ", psId: " + psId);
208         log.modelUse = pmModel.modelName;
209         log.methode = "delete";
210         // méthode findByIdAndRemove(qui à pour arguments l'id du document) de mongoose sur un model mongoose qui execute une fonction callback gérant erreur et les datas
211         //Complexity is 5 Everything is cool
212         pmModel.findByIdAndRemove(psId, [ function (err, data) {
213             // si erreur on renvoie via la méthode reject des datas au format json
214             if (err) {
215                 console.log("status 500, err : " + err);
216                 log.statusResponse = 500;
217                 new apilogModel(log).save();
218                 reject({ status: 500, json: { success: false, message: 'Erreur lors de la suppression des datas', data: null } });
219             } else {
220                 if (data == null) {// si il n'y a pas de data trouvé on envoie via la méthode resolve des datas à nul
221                     console.log("status 404, data = null");
222                     log.statusResponse = 404;
223                     new apilogModel(log).save();
224                     resolve({ status: 404, json: { success: false, message: 'les datas à supprimer non trouvé', data: null } });
225                 } else {// si les datas ont été trouvées, on envoie via la méthode resolve
226                     console.log("status 200, data = some datas supprimés");
227                     log.statusResponse = 200;
228                     new apilogModel(log).save();
229                     resolve({ status: 200, json: { success: true, message: "Les datas sont supprimés", data: null } });
230                 }
231             }
232         })
233     }]);
234 }

```

## **CHAPITRE 8 - DEPLOIEMENT**

## 8.1 - LE DIAGRAMME DE DÉPLOIEMENT

Le diagramme de déploiement est réalisé en fin de parcours. Il correspond à la description de l'environnement d'exécution du système et de la façon dont les composants y sont installés.

Un DPL représente l'architecture physique des matériels (nœuds) qui composent un système ainsi que la répartition des composants et des artefacts (fichiers exécutables et data) sur les matériels et les liens entre les différents nœuds (le plus souvent des liaisons réseaux).

Les nœuds sont représentés par des cubes. Chaque nœud ou processeur assure un ensemble de traitements représentés par des composants.

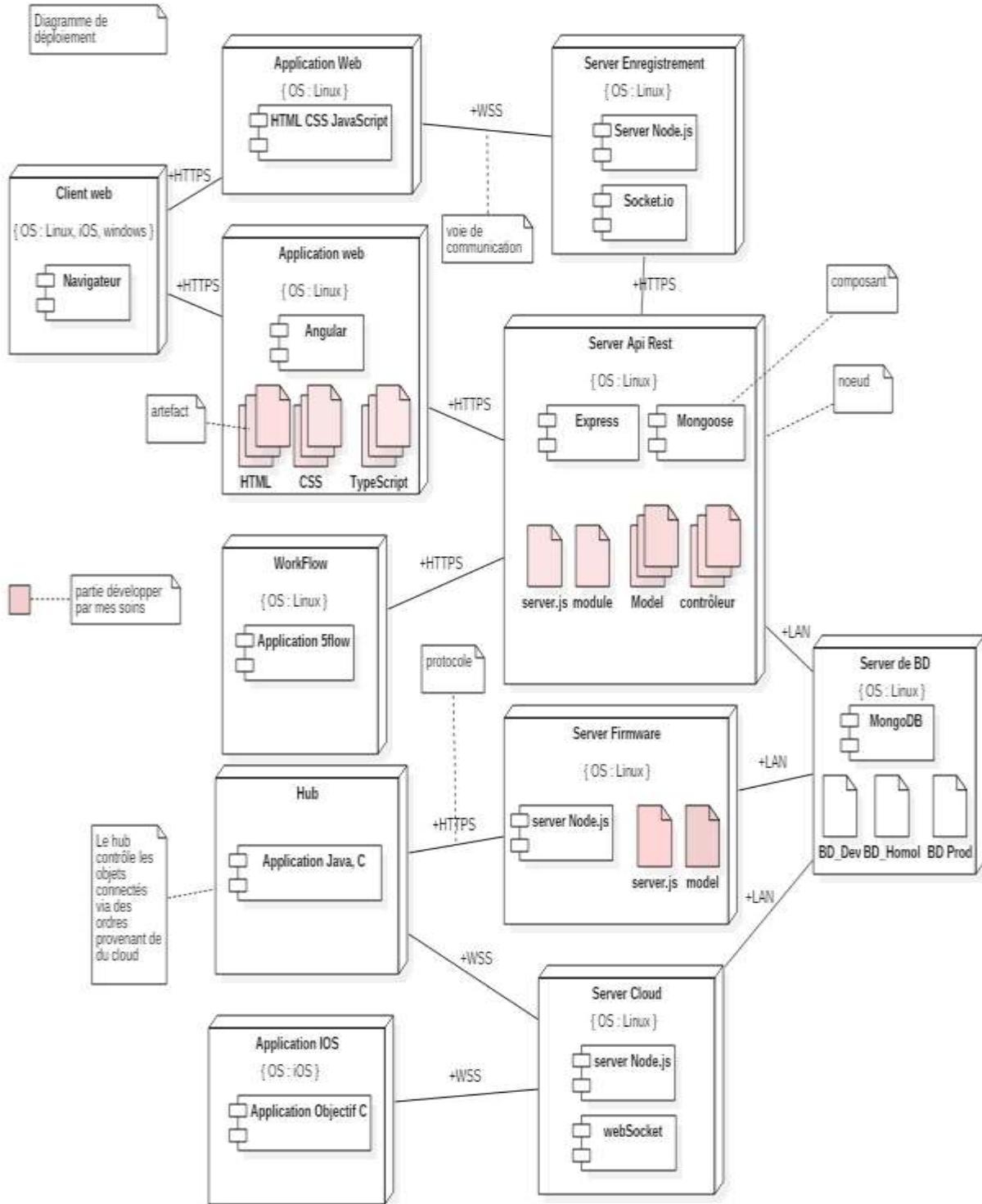
La nature de l'équipement peut être précisée au moyen d'un stéréotype. Les liens sont représentés par des arcs. Ils peuvent être surmontés d'un stéréotype pour préciser le protocole utilisé. Des multiplicités peuvent préciser les associations entre nœuds.

Les nœuds qui correspondent à des processeurs (pas des dispositifs) portent le nom des exécutables qu'ils hébergent.

Chez Otodo, il y a 3 grands nœuds (le hub, l'application mobile iOS, et le server cloud).

Ensuite vient se greffer mon projet qui aujourd'hui sert de passerelle à 2 clients externes à mon projet :

1. Un server d'enregistrement de nouveau utilisateur de l'app mobile via un procès dédié avec lecture de QR Code sur de l'emballage, passant par la vérification de l'existence du hub par son adresse mac (via l'api) et finissant par l'insertion de l'user en base (via l'api).
2. Un client qui s'occupe de la gestion d'incident et qui récupère la liste des hubs.



## 8.2 - LE DEPLOIEMENT

Le déploiement se fait sur 3 environnement distinct : dev, homologation, et production.

J'ai donc dans un premier temps à un moment du projet dupliqué la version initial (dev).

Au fur et à mesure de l'avancer du code et des tests, à l'image d'un outil de versioning (GIT, que j'ai utilisé), je transférerais les parties de code en environnement d'homologation, puis après des tests d'autres personnes je transférerais ces parties en environnement de production.

J'ai utilisé le logiciel FTP FileZilla pour le transfert du projet vers l'hébergeur.

La gestion du serveur s'est fait à l'aide de PM2 (process- manager). Outil simple et qui permet de gérer ses serveurs aussi bien chez un hébergeur qu'en local.

Il peut lancer un serveur, l'arrêter, le supprimer, voir ses logs, le redémarrer.

J'avais déjà eu l'occasion de l'utiliser grâce au tutoriel OpenClassrooms sur Node.js

L'application web de Angular est compilée via la commande :

- `ng build`

Un dossier « dist » est créé et comprend un groupe de fichiers dont le fichier `index.html`.

## ***CHAPITRE 9 - LA SECURITE***

## 9.1 - LA SECURITE DE L'APPLICATION

Le framework Angular empêche les failles XSS et aussi les injections de code.

La couche « API REST » est une seconde protection sur les injections de code. Diffusion des requêtes en HTTPS.

J'ai utilisé un nom de domaine déjà sécurisé pour le déploiement du server. Enfin la partie logique de l'API REST, qui est développée avec une identification avec token et une gestion des droits par route, permet un accès limité aux données.

## ***CHAPITRE 10 - CONCLUSION***

## 10.1 - RETOUR SUR L'EXPERIENCE ACQUISE

Ce stage m'a réellement été très bénéfique. Il m'a permis de revoir un grand nombre de choses que j'avais apprises durant la formation : UML, HTML/CSS, JavaScript (et son framework Angular), Node.js, MongoDB, la programmation orientée objet, le design pattern MVC, ainsi que le développement en couches (ECB).

J'y ai appris le développement d'une API REST et consolidé mes connaissances du protocole http.

D'un point de vue personnel, il m'a aussi montré à quel point il fallait être rigoureux, surtout au niveau de l'analyse et de la conception où j'ai tendance à m'éparpiller.

Ce stage m'a aussi rassuré sur mes capacités à exercer ce métier et va me permettre une recherche d'activité avec plus d'assurance. Il m'a aussi donné envie d'approfondir mes connaissances dans le domaine des objets connectés.

En conclusion ce stage est tombé du ciel car il réunit des langages modernes et que j'ai découvert plus en profondeurs et dans un domaine qui était celui qui fut mon fer de lance dans mon projet de reconversion via ma lettre de motivation faite au Fongecif (la domotique).

Cependant il ne s'agit là que d'un début car je suis conscient du travail qu'il reste à faire pour devenir un bon concepteur développeur full-stack.

## **CHAPITRE 11 - ANNEXES**

## 11.1 - CORRESPONDANCES PROJET/REAC

A vous de mettre en gras dans la colonne de gauche ce que vous avez réalisé dans votre projet.

Activité	Compétence	Correspondance
<b>Développer des composants d'interface</b>		
	<b>Maquetter une application</b>	UML (DCU, Maquettes, DSEQ, DNAV, DAC)
	<b>Développer une interface utilisateur</b>	Interface
	<b>Développer des composants d'accès aux données</b>	Langage serveur (Node.js)
	<b>Développer des pages web en lien avec une base de données</b>	HTML, CSS, JavaScript pour les contrôles des données entrantes, Langage serveur (Node.js)
<b>Développer la persistance des données</b>		
	<b>Concevoir une base de données (Schéma entité-Association, modèle physique normalisé)</b>	UML (DCL) et Merise (MCD, MLD, MPD)
	<b>Mettre en place une base de données (intégrité des données, ...)</b>	MongoDB avec un restoremongo en local
	<b>Développer des composants dans le langage d'une base de données</b>	Développement à l'aide de mongoose de composants
	<b>Utiliser l'anglais dans son activité professionnelle en informatique (Ecrit et parlé)</b>	Abstract
<b>Développer une application n-tiers</b>		

	<i>Concevoir une application</i>	<i>UML</i>
	<i>Collaborer à la gestion d'un projet informatique</i>	<i>Gestion de projet avec git</i>
	<i>Développer des composants métier</i>	<i>Classes « Modèle »</i>
	<i>Construire une application organisée en couches</i>	<i>ECB (Diagrammes de séquence détaillés) + MVC</i>
	<i>Développer une application de mobilité numérique</i>	
	<i>Préparer et exécuter les plans de tests d'une application</i>	
	<i>Préparer et exécuter le déploiement d'une application</i>	<i>Diagramme de déploiement (UML)</i> <i>Déploiement.</i> <i>Raccord à la BD.</i> <i>Installation des codes HTML, CSS et JavaScript de Angular</i> <i>Installation des codes Node.js.</i>

## 11.2 - GLOSSAIRE/LEXIQUE OU/ET LISTE DE MOTS-CLES ET SIGLES

Mot clé	Description
CRUD	<p>Concerne le LMD (<i>Langage de manipulation de données</i>)</p> <p>C : <i>create</i> R : <i>read</i> U : <i>update</i> D : <i>delete</i></p>
HTTP	<p>Protocole de communication entre 2 logiciels (un client et un serveur)</p>
API	<p><i>Application Programming Interface</i></p>
API Rest	<p><i>Application Programming Interface - Representational State Transfer</i></p>