



CentraleSupélec

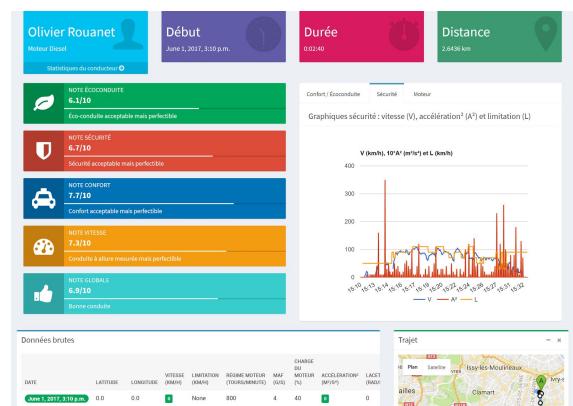


EasyDrive



SOLUTIONS D'ANALYSE DE CONDUITE INNOVANTES

Projet long 2016-2017



16 juin 2017

Léo Laugier – Arnaud d'Esquerre - Olivier Rouanet - Léon Cardineaud

Encadré par Jocelyn Fiorina

Table des matières

Introduction	5
Initialisation du projet : quel besoin, quels prospects, quels objectifs ?	7
Analyse du besoin	7
Les acteurs de la Nouvelle Économie : l'accent mis sur le confort	7
Les gestionnaires de véhicules : l'enjeu de l'écoconduite appliquée à une flotte de véhicules	7
Les assurances et les particuliers : la priorité orientée sur la sécurité routière	7
Notre solution	7
Planning prévisionnel (Septembre)	8
Récupération des données sur l'Arduino	8
Envoi des données avec LoRa	8
Serveur Django	8
Traitement des données	8
Interface utilisateurs	8
Déploiement et tests	8
Développement de EasyDrive	9
Hardware : notre solution EasySensor	9
Quels capteurs pour quelles informations ?	9
Le diagnostic embarqué, ou On-Board Diagnostics (abrégé en OBD)	9
Les données nécessaires	10
Le protocole OBD	10
Mode 1	10
Mode 2	10
Mode 3	10
Mode 4	11
Mode 5	11
Détail du fonctionnement du mode 1	11
Protocole de communication avec l'Arduino	12
Envoi d'un PID et réception d'un message	12
Conversion depuis hexadécimal en décimal	13
La centrale inertielle	14
Le GPS	14

Le calculateur Arduino	14
Assemblage des différents modules	15
Description du programme Arduino	16
Le module LoRa	19
Emission des données (format, Réseau LoRaWan)	19
Transmission sur le réseau Lora	21
Software : notre solution EasyApp	24
Réception des données	24
Sur le serveur d'Objenious	24
Sur le serveur du Rézo	25
Front-End: Application	29
Architecture de l'application	29
Utilisation de services externes : Google Charts, Google Maps API et admin LTE	33
Difficultés rencontrées	35
Back-End: Traitement des données et évaluation	36
Architecture de la base de donnée	36
Prétraitement	36
De la base donné à l'affichage : utilisation de Django	39
Confidentialité	43
Évaluation des paramètres de conduite	43
Paramètre accélération du véhicule a	43
Paramètre vitesse linéaire du véhicule v	44
Paramètre virage t	44
Paramètre régime moteur r	45
Notation d'un trajet	46
Note écoconduite en fonction des paramètres a et r	46
Note sécurité en fonction des paramètres a, v	46
Note confort des paramètres t, a	46
Note de respect des limitations de vitesse en fonction du paramètre v	46
Note globale en fonction des autres notes	46
Adaptation de la notation à chaque client	46
Activités annexes	48
Formation en ligne ouverte à tous (MOOCs)	48

Arduino	48
Développez votre site web avec le framework Django	48
Présentation de Django	48
Créez des applications web avec Django	48
Le fonctionnement de Django	48
Gestion d'un projet	49
Les bases de données et Django	49
Développement du pattern	50
Première page grâce aux vues	50
Les templates	50
Les modèles	50
L'administration	50
Les formulaires	51
Apprentissage Automatique	51
Étude théorique	51
Technique de l'Hyperplan Discriminant de Fisher	51
Le perceptron monocouche	53
Etude pratique : Initiation au Machine Learning avec Sckitlearn	55
Comment résoudre un problème de data science ?	55
Qu'est ce que le machine learning ?	56
Identifier les différents types de problème de machine learning	56
Application à notre projet	56
Perspective et ouverture	57
Conclusion	58
Remerciements	58
Aux enseignants-chercheurs de CentraleSupélec	58
À Objenious	58

Introduction

D'un côté l'Internet des Objets, l'extension d'Internet appliquée aux objets connectés, permet une analyse à distance de l'utilisation d'un objet à travers le réseau LoRaWAN développé par l'alliance LoRa. Les opportunités de l'intégration du monde physique dans un système informatisé ouvre des possibilités d'optimisation, d'efficacité et de bénéfice économique pour les professionnels. Les experts estiment que l'Internet des objets représentera 50 milliards d'objets connectés d'ici 2020¹.

D'un autre côté, les entreprises gestionnaires de flottes automobiles ont pour objectif de réduire leurs coûts et, de plus en plus, de réduire leur empreinte carbone le tout en assurant la sécurité des personnes et des biens.

L'équipe **EasyDrive** est convaincue que la transition d'un modèle classique de gestion d'un parc automobile vers une analyse automatisée de l'évaluation de la conduite et la synthèse des données récoltées s'effectuera *via* une exploitation intelligente de capteurs disponibles sur le marché comme Objenious et une interface simple et intuitive. L'enjeu est vaste : économique, marketing, et managérial; le *challenge*, plus que jamais, est d'actualité.

Nous avons profité d'une année académique pour développer notre projet technique en exploitant des connaissances acquises dans notre cursus d'ingénieur et en explorant des domaines techniques nouveaux.

Ce rapport présente le déroulement du projet en commençant par une synthèse précise des objectifs initiés en début d'année. Nous décrirons ensuite notre démarche scientifique et technique pour aboutir à la réalisation finale du projet. Enfin nous analyserons les axes d'ouverture et les perspectives induites par notre travail.

Le business des objets connectés

— Une croissance mondiale accélérée —



— L'analytique, locomotive de la création d'emplois —



¹ Maddyness - #IOT 18 chiffres à connaître pour comprendre le potentiel du marché des objets connectés - 2016

— 2018, l'ère du presque tout connecté —

420 millions

de voitures connectées
d'ici 2018. (6)



19 millions de vêtements
technologiques seront expédiés
en 2014. (7)



27% de toutes les données
seront générées par les objets connectés
en 2020. (2)



1,8 milliard de compteurs d'énergie

intelligents seront déployés dans le monde
en 2018. (8)



300 milliards

de dollars, c'est le marché prévu
pour les "bâtiments intelligents"
en 2022. (9)



847 millions

d'appareils connectés utilisés
par le secteur mondial de la santé
en 2023. (9)



24 milliards

de dollars de valeur
attendue dans le secteur de la chaîne d'approvisionnement
et de la production M2M en 2022. (9)



80%

des appareils connectés à Internet

présentent de potentielles failles de sécurité. (10)

26 milliards

d'unités connectées
en 2020 transformeront la demande
des centres de données dans le monde. (11)



Sources

(1) IDC - The Internet of Things Moves Beyond the Buzz - 2014

(2) IDC - The Digital Universe of Opportunity - 2014

(3) Cisco - The Internet of Things - Cisco Visualization

(4) SOGETI - Things: Internet of Business Opportunities - 2013

(5) ABI Research 2014

(6) IDate - 'Connected cars, already a reality'

(7) IDC - Worldwide Wearable Computing Device 2014-2018 Forecast and Analysis

(8) IDate - Smart Grid: the smart metering market and beyond

(9) Machina Research 2014

(10) HP - Internet of Things Research Study 2014

(11) Gartner press release 2014 - Gartner Says the Internet of Things Will Transform the Data Center

I. Initialisation du projet : quel besoin, quels prospects, quels objectifs ?

A. Analyse du besoin

Dans un premier temps, nous avons regrouper par thèmes nos prospects pour cibler la solution optimale à apporter à chacun d'eux.

1. Les acteurs de la Nouvelle Économie : l'accent mis sur le confort



Les entreprises de Voiture de Transport avec Chauffeur telle *Uber* et les entreprises de service de covoiturage comme *Blablacar* nécessitent une évaluation d'une conduite "confortable" et "sûre" qui passe actuellement par une notation subjective des passagers.

2. Les gestionnaires de véhicules : l'enjeu de l'écoconduite appliquée à une flotte de véhicules

Les entreprises de location de véhicule (*Hertz*), les services de livraison (*La Poste, UPS*) et les convoyeurs de fond (*Brinks*) ont besoin de surveiller la sécurité de leurs chauffeurs (et des biens transportés dans une moindre mesure). De plus, l'écoconduite doit être mesurée afin d'agir efficacement contre les conduites agressives peu soucieuses de l'environnement, dans le but de réduire les coûts et l'empreinte carbone de l'entreprise.



3. Les assurances et les particuliers : la priorité orientée sur la sécurité routière

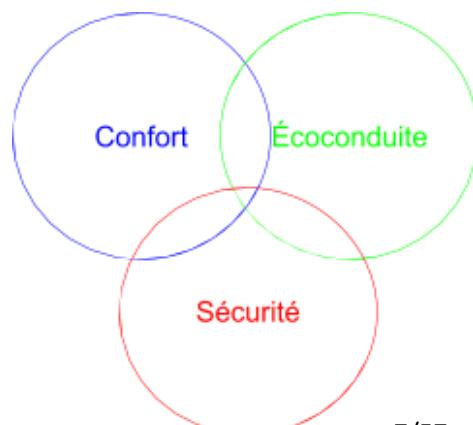


Évidemment, dans les prochaines années les assurances "connectées" vont prendre en compte la conduite des assurés pour proposer des tarifs qui récompenseront les comportements responsables sur la route. Les particuliers ont également besoin de chiffrer et comparer leur conduite afin de répondre aux questions que nous, automobilistes, nous posons tous : "Suis-je dangereux pour moi-même ?", "Suis-je dangereux pour les autres ?", "Ma conduite est-elle (trop) coûteuse ?"

B. Notre solution

Nous avons imaginé une solution pour répondre au besoin exprimé ci-dessus. Nous avons identifié trois axes majeurs qui serviront de cahier des charges pour évaluer une conduite sur un trajet puis sur l'ensemble des trajets d'un conducteur/véhicule.

Notre application est donc un outil d'aide à la décision et d'automatisation permettant d'optimiser la gestion de flotte de véhicules.



C. Planning prévisionnel (Septembre)

Dès le début du projet, nous avons conçu un planning prévisionnel sous forme de diagramme de Gantt (cf. Annexe 1).

Le découpage s'est profilé selon les activités suivantes :

1. Récupération des données sur l'Arduino
2. Envoi des données avec LoRa
3. Serveur Django
4. Traitement des données
5. Interface utilisateurs
6. Déploiement et tests

En parallèle, nous avons travaillé sur les formations associées aux différentes activités.

II. Développement de *EasyDrive*

A. Hardware : notre solution *EasySensor*

1. Quels capteurs pour quelles informations ?

a. *Le diagnostic embarqué, ou On-Board Diagnostics (abrégé en OBD)*

C'est l'objet central de notre projet, la prise OBD va nous permettre de récupérer des données intéressantes fournies par le véhicule. On peut donc se dispenser d'investir dans des capteurs difficiles à planter dans le véhicule en se servant directement de ceux conçus par le constructeur.

Avec l'arrivée de l'électronique sur le marché de l'automobile, tous les véhicules depuis les années 2000 sont équipés d'une prise de diagnostic dite prise OBD sur laquelle on peut connecter au moyen d'une interface, une valise ou un ordinateur. Ainsi on communique avec l'ordinateur de bord et l'on a accès à tous les réglages, les défauts enregistrés et les capteurs de la voiture. Elle est située le plus souvent sous le volant.



Figure 1 : Prise OBD femelle dans le véhicule

Lors de notre premier prototype, la prise OBD était reliée par un fil au boîtier et il a fallu faire des soudures supplémentaires pour la relier comme il fallait. Ce n'était pas très robuste et difficile d'implantation. Nous avons connu une panne avec ce système, voilà pourquoi nous nous sommes tournés vers une transmission *Bluetooth*.

Nous avons choisi une interface simple d'utilisation qui permet uniquement de lire les données du véhicule via *Bluetooth* comme ci-dessous.



Figure 2 : Prise OBD Bluetooth

Les données nécessaires

Grâce au module OBD nous allons lire les données suivantes : celles qui vont nous aider à noter la conduite selon plusieurs critères :

- La vitesse : nous allons la comparer à la limitation de vitesse
- Le régime moteur RPM : il reflète l'éco-conduite (économique ou non)
- Le débit massique d'air (MAF Mass Air Flow) : comme le mélange est stoechiométrique, 1 g d'essence pour 14 g d'air, c'est avec cela que l'on peut calculer la consommation.
- La charge du moteur en % : en multipliant par les RPM on obtient la puissance en pourcentage par rapport à la puissance maximale du moteur.

Le protocole OBD

Le protocole OBD est standardisé et nous avons développé nous-même la partie logicielle qui permet la communication avec la voiture.

Il existe plusieurs types de protocoles mais le plus courant est le ISO15765-4 (CAN-BUS). Nous envoyons à la voiture une commande comportant le mode et le PID voulu. Il existe neuf modes de communication :

Mode 1

Ce mode retourne les valeurs courantes de certains capteurs tels que :

- le régime moteur
- la vitesse du véhicule
- les températures du moteur (air, liquide de refroidissement)
- les informations sur les sondes à oxygène et la régulation du dosage air/carburant

Chaque capteur est caractérisé par un numéro appelé PID (Parameter Identifier) qui permet d'identifier le paramètre, c'est celui qui nous sera le plus utile dans notre projet.

Voici la liste des PID dans la page Wikipédia : https://en.wikipedia.org/wiki/OBD-II_PIDs

Mode 2

Ce mode retourne les données gelées (ou instantanées) d'un défaut. Lorsqu'un défaut est détecté par l'ECM, celui-ci enregistre les données des capteurs au moment précis de l'apparition du défaut.

Mode 3

Ce mode retourne les codes défauts enregistrés. Ces codes défauts ont été standardisés pour toutes les marques de véhicule et découpés en quatre catégories :

- P0xxx : pour les défauts standards liés au système de propulsion (moteur et transmission)
- C0xxx : pour les défauts standards liés au châssis
- B0xxx : pour les défauts standards liés à la carrosserie
- U0xxx : pour les défauts standards liés aux réseaux de communications

Mode 4

Permet d'effacer les défauts

Mode 5

Autodiagnostic

Les modes 6, 7, 8 et 9 sont très peu utilisés en Europe

Détail du fonctionnement du mode 1

Il faut d'abord initialiser la connexion en réglant le *baudrate* (vitesse de communication) et le type de protocole utilisé.

Nous envoyons *via* la carte Arduino une commande du type 01x00 : le 01 correspond au mode et le 00 au PID. Cette commande renvoie la liste des PID supportés par le véhicule sous forme d'une liste binaire (1 si le PID est supporté 0 sinon) le rang correspond au numéro du PID

Ex: 001110 (seul les PID 2,3 et 4 sont supportés)

Une fois que nous connaissons les capteurs présents dans le véhicule, nous envoyons le PID correspondant au capteur dont on veut avoir l'information.

Ex: 01x0C qui correspond au régime moteur, nous interrogeons le capteur numéro 12.

La communication est codée en hexadécimal. Ci-dessous se trouve deux réponses suite à l'envoi du PID 01x05 et 01x0F.

En ASCII :[b'0105', b'41 05 71 ']
En decodé:73°C

En ASCII :[b'010F', b'41 0F 47 ']
En decodé:31°C

Dans notre exemple, 41 correspond au numéro de l'unité de commande électronique (ECU) qui répond, 05 ou 0F au PID renvoyé et l'octet encodé en hexadécimal à l'information. Elle peut être codée sous plusieurs octets, les données sont : 0B AB.

[b'0123', b'41 23 0B AD ']

Nous avons fait beaucoup d'essais pour gérer les exceptions et les erreurs comme celles ci-dessous. Ces trois cas peuvent se présenter :

Pas de réponse de l'OBD

015C
[b'015C', b'SEARCHING...', b'NO DATA']

Réponse longue à venir

010B
[b'010B', b'SEARCHING...', b'41 0B 65 ']

Plusieurs ECU répondent à la requête

0143
[b'010143', b'SEARCHING...', b'0: 41 01 00 04 00 00 ', b'1: 00 80 00 00 11 00 00 ']

Protocole de communication avec l'Arduino

Ce paragraphe est consacré au détail de quelques fonctionnalités importantes de la bibliothèque OBD que nous avons entièrement développée pour Arduino.

Envoi d'un PID et réception d'un message

La lecture se fait caractère par caractère tant que le symbole '>' n'est pas lu ou que le temps n'est pas écoulé, seules les données après les '...' sont stockées dans le tableau de *char result*.

```
int OBD2::receive(char *result) {  
    int n=0;  
    unsigned long startTime = millis();  
    while (true) {  
        if (OBD_SERIAL.available()) {  
            char c = OBD_SERIAL.read();  
            if (c == '>') {  
                break;  
            }  
            else if (n < 256) {  
                if (c == '.' && n > 2 && result[n - 1] == '.' && result[n - 2] == '.') {  
                    // waiting signal  
                    n = 0;  
                }  
                else {  
                    result[n++] = c;  
                }  
            }  
        }  
        else {  
            if (millis()-startTime > 5000) {  
                Serial.print("TimeOut: "); Serial.println(result);  
                break;  
            }  
        }  
    }  
    result[n] = 0;  
    return n ;  
}
```

Conversion depuis hexadécimal en décimal

La fonction ci-dessous réalise le décodage de hexadécimal vers décimal.

```
byte hex2uint8(const char *p)
{
    byte c1 = *p;
    byte c2 = *(p + 1);

    if (c1 >= 'A' && c1 <= 'F')
        c1 -= 7;

    else if (c1 >='a' && c1 <= 'f')
        c1 -= 39;

    else if (c1 < '0' || c1 > '9')
        return 0;

    if (c2 >= 'A' && c2 <= 'F')
        c2 -= 7;

    else if (c2 >='a' && c2 <= 'f')
        c2 -= 39;

    else if (c2 < '0' || c2 > '9')
        return 0;

    return c1 << 4 | (c2 & 0xf);
}
```

Le détail des PID et de la fonction de normalisation des données est dans les bibliothèque OBD.cpp et OBD.h.

Grâce à cette prise on peut aussi transmettre les codes défauts appelés DTC détectés par le véhicule en direct. Ce serait une ouverture de notre projet actuel². Le morceau de code ci-dessous résume la manière de lire les codes défaut et il faudrait ensuite les traiter sur le serveur.

```
Serial.println("Recherche des défauts ...");

char DTC[100] = "DTC: ";
char dtc[50];

obd.read_cmd_brute("0101", dtc);
strcat(DTC, dtc);

Serial.println(DTC);
LORAWAN_SERIAL.print(DTC);
```

² cf. *supra* IV

b. La centrale inertie

Le gyroscope et l'accéléromètre sont en connexion I2C avec l'arduino, nous avons fait un programme pour initialiser l'accéléromètre de sorte qu'avec la gravité il puisse définir la position dans laquelle il est. Il mesure l'accélération dans le plan horizontal et le lacet autour de l'axe vertical. Nous pouvons effectuer des mesures toutes les 10 ms au maximum.



Figure 3 : Centrale inertie

c. Le GPS

La puce GPS est en connexion RX/TX classique avec la carte. Nous utilisons une bibliothèque Tiny GPS qui permet de récupérer les coordonnées géographiques avec une précision à moins de cinq mètres.



Figure 4 : Puce GPS

2. Le calculateur Arduino



Figure 5 : Carte Arduino

Tous les modules sont reliés à la carte Arduino qui les pilote et qui les alimente en énergie électrique. Pour communiquer avec le boîtier OBD il a fallu installer un module Bluetooth sur la carte et le configurer. Cette solution offre plus de flexibilité quant au placement de la carte dans la voiture.

En effet elle n'a plus besoin d'être près de la prise OBD de la voiture qui est souvent peu accessible.

La carte Arduino Méga a l'avantage de posséder quatre ports série et deux ports I2C ce qui nous permet de brancher tous les périphériques nécessaires.

a. Assemblage des différents modules

Nous avons dû assembler les différents modules et fabriquer un boîtier pour les mettre à l'intérieur.

La carte arduino est fixée au boîtier par des supports en caoutchouc et les différents modules sont vissés sur la partie supérieure du boîtier.

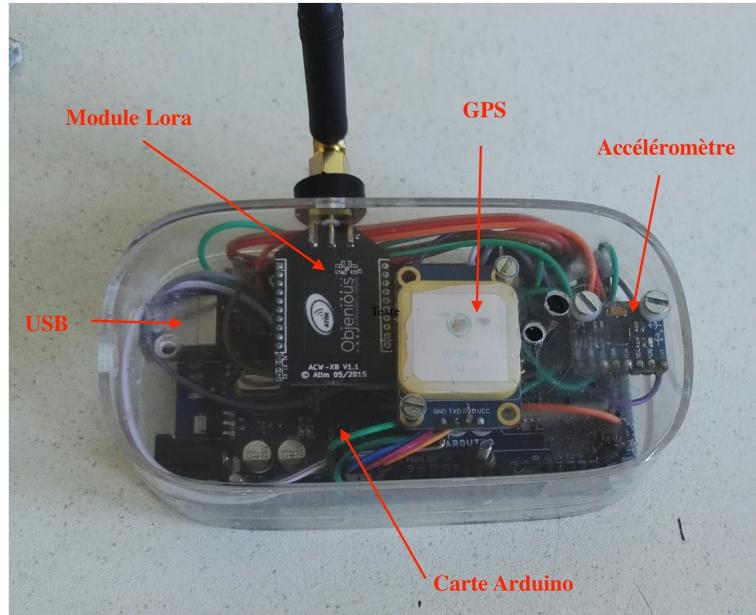


Figure 6.1 : EasySensor

Ci-dessous le schéma électrique de la connexion des composants entre eux à l'intérieur du boîtier.

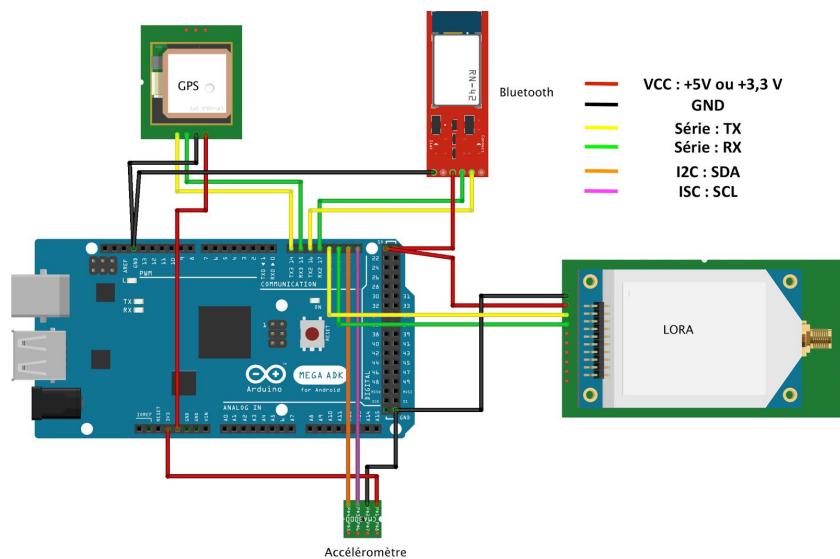


Figure 6.2 : Schéma électrique EasySensor

Il y a deux éléments de type *plug & play* à installer dans la voiture : la prise OBD qui est reliée par Bluetooth à l'*EasySensor* (boîtier que nous avons fabriqué contenant la carte et tous les autres capteurs) et l'*EasySensor* à l'allume-cigare.



Figure 7 : EasySensor embarqué

b. Description du programme Arduino

Le programme Arduino se divise en deux parties, un *Setup* et une *Loop*. Le *Setup* est exécuté à chaque démarrage de la carte ou après chaque appui sur le bouton *reset*. Il permet d'initialiser les différents modules et la communication avec la carte ainsi que de créer les variables qui resteront pendant tout le programme.

Dans l'extrait de code ci-dessous on voit le calibrage de l'accéléromètre et l'initialisation du module LoRa.

```
void setup() {
    Serial.begin(9600);
    Wire.begin(); //I2C bus
    // -----
    // Acceleromètre
    // -----
    accelgyro.initialize();
    // verify connection
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection réussie" :
    "MPU6050 connection échec");
    accelgyro.calibration();
    // -----
    // GPS Init and configuration
    // -----
    GPS_SERIAL.begin(9600); // connect gps sensor
    Serial.println("GPS OK");
```

```

// -----
// LoRaWAN module Init and configuration
// -----
//Init of the LoRaWAN module

if (myArm.Init(&LORAWAN_SERIAL) != ARM_ERR_NONE)
{
    Serial.println("Network Error");
}
else
{
    Serial.println("Connected to Objenious");
}

```

La boucle *loop* est exécutée en permanence, à chaque itération nous créons le tableau de *char* *data[39]* qui fait trente-neuf octets de long. Nous récoltons toutes les deux secondes l'accélération, le lacet, la vitesse, le régime moteur, la MAF et la charge moteur. Dans le code ci-dessous on ne voit que la prise de la donnée vitesse ; pour les autres mesures, c'est exactement le même schéma avec des PID différents.

```

void loop() {
    //création des variables et du tableau data
    unsigned char data[39];
    int valeur=0;
    int max_acc_xy;
    int max_lacet;
    int16_t ax, ay, az;
    int16_t gx, gy, gz;
    //Boucle OBD
    for (int i=0; i<5; ++i){
        max_acc_xy=0;
        max_lacet=0;
        Serial.print("i= ");Serial.println(i);
        previousMillis = millis();
        //Prise de l'accélération et du Lacet
        accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        max_acc_xy=max (module(ax,ay), max_acc_xy);
    }
}

```

```

    max_lacet=max (abs(gz), max_lacet);

    bool error = false;

    // Prise de la vitesse

    if (obd.read_pid(PID_SPEED, &valeur)==false){

        valeur=255;

        error=true;

    }

    data[8+i]=valeur; // Mise de la valeur dans la table data

    Serial.print("Speed: ");Serial.print(valeur);Serial.println(" km/h");

    delay(200); //Affichage et délai

```

Voici la demande des coordonnées GPS au module :

```

//Boucle GPS

int val = 0;

unsigned long startMillis = millis();

while ((val == 0) && ( millis() - startMillis < 1000)) {

    while (GPS_SERIAL.available()) { // check for gps data

        if (gps.encode(GPS_SERIAL.read())) { // encode gps data

            gps.get_position(&lat, &lon);

            val = 1;

        }

    }

}

```

Au bout de 5 cycles de 2 secondes, on interroge le GPS pour récupérer la position du véhicule c'est la partie de code intitulée Boucle GPS. Ensuite les données sont concaténées dans le tableau de char data avec un "/" pour signifier la fin de chaque message.

```

//Assemblage des données

data[0]=lat>>24; data[1]=(lat&0xfffffff)>>16;data[2]=(lat&0xffff)>>8;data[3]=(lat&0xff);

data[4]=lon>>24; data[5]=(lon&0xfffffff)>>16;data[6]=(lon&0xffff)>>8;data[7]=(lon&0xff);

data[38]='/';

```

Le message de 39 octets est envoyé via le module LoRa par l'instruction LRAWAN_SERIAL.write(data,39).

```

//Envoi des données

LORAWAN_SERIAL.write(data,39);

```

3. Le module LoRa

Nous utilisons le module LoRa pour envoyer les données sur notre serveur, il s'agit d'un émetteur basse consommation qui porte à plusieurs kilomètres. Les messages ont une taille limitée à 50 octets ce qui nous a obligés à réfléchir à comment minimiser les données envoyées. Un message est envoyé toutes les 10 secondes, la connexion avec la carte est aussi RX/TX.



Figure 8 : Module LoRa

4. Emission des données (format, Réseau LoRaWan)

Dans notre premier programme, nous envoyions des données volumineuses codées en ASCII, c'est-à-dire qu'il y avait un octet par caractère. Très vite nous avons compris que le message était divisé en plusieurs petits messages d'une cinquantaine d'octets qui arrivaient dans le désordre et au même moment avec un découpage assez aléatoire.

Le traitement des données était impossible et trop incertain. La problématique était donc de choisir un format adapté : quelles étaient les données utiles, à quelle fréquence avec quelle précision ? Et comment, pour une précision et une fréquence donnée, faut-il envoyer chaque valeur.

Nous avons opté pour le format suivant, Vitesse, Régime moteur, MAF, Charge, Accélération et Lacet sont collectés toutes les deux secondes. Toutes les **dix secondes**, nous relevons la position GPS et nous envoyons un message avec une paire de coordonnées et cinq valeurs de chaque donnée prise toutes les deux secondes. Les valeurs sont envoyées directement sous forme d'entier en binaire, éventuellement divisées par un coefficient afin de limiter l'amplitude.

Cet intervalle de dix secondes entre chaque message évite que deux messages ne s'intervertisquent, de plus un compteur et un *TimeStamp* identifient chaque message pour en conserver l'ordre. Ajoutons que si un message se perd, ce ne sont que dix secondes d'enregistrement qui sont perdues, ce qui est négligeable sur un trajet de plusieurs dizaines de minutes en général.

Pour arriver à réduire le nombre d'octets à envoyer, les données sont envoyées de la manière suivante. Sur certaines données, la précision requise est moindre par exemple il nous est inutile de connaître la vitesse de rotation du moteur (RPM) au tour près. Pour réduire la taille des messages nous avons décidé de diviser certaines grandeurs. De ce fait, elles tiennent sur un octet au lieu de deux ou trois.

4 octets	4 octets	5 octets	5 octets	1 octet				
Latitude	Long	Vitesse	RPM	Charge	MAF	ACC	Lacet	“/”
exacte	exacte	exacte	/32	exact	/4	/2 ⁷	/2 ⁷	exacte

Tableau 1 : Format d'émission des données

Le “/” mentionné plus haut marque la fin du message. Les facteurs de division sont choisis pour réduire les calculs, en effet comme nous travaillons en binaire, il suffit de faire un décalage vers

la gauche d'un bit pour diviser par deux, de deux bit pour diviser par quatre, etc...

Dans l'exemple ci-dessous le nombre "valeur" est divisé par trente-deux avant d'être mis dans le tableau *data*.

```
data[8+i]=valeur>>4
```

Les données sont cryptées et envoyées sur le réseau LoRa et nous les récupérons en hexadécimal.

Il faut aussi préciser qu'il est possible d'optimiser encore plus le format des données. Cela ne nous a pas été nécessaire, mais dans l'éventualité d'un prolongement du projet, il pourrait être nécessaire d'envoyer plus d'informations à travers le réseau LoRa. Pour cela, il faut envoyer chaque valeur sur un nombre d'octet non entier, ce qui implique une plus lourde charge pour l'encodage et le décodage, et une plus grande difficulté à débugger notre code, puisque les données en mémoire, que ce soit sur une carte Arduino où sur un processeur d'ordinateur "classique", ne peuvent être accédées que octet par octet. Il faudrait alors utiliser des opérations manipulant chaque valeur bit à bit pour les assembler à plusieurs sur un octet, puis décoder les octets vers des valeurs au format usuel.

Cette optimisation de l'espace disponible peut se faire aussi en envoyant, pour les grandeurs continues et à variation majorable comme la vitesse ou les RPM, un écart avec la première valeur plutôt que chaque valeur de manière indépendante. Il est aussi possible, sans perte d'information dommageable, de se contenter d'une précision légèrement moins importante, comme une vitesse à 4 km/h près, ou des RPM à 128 tours/minute près, on encore diviser par deux voir même par 4 la précision sur les valeurs de la charge et des RPM, et diviser par 8 voir 16 la précision des valeurs de l'accélération et du lacet. Une économie de 7 octets est alors possible.

Une autre possibilité envisagée dans un premier temps est la mise en place d'un système d'*acknowledgement* des messages, et de garder les anciens messages en mémoire dans l'Arduino afin de pouvoir les renvoyer en cas d'erreur de transmission. Le problème est que nous n'avons pas trouvé d'autre moyen de s'assurer de la réception des messages que l'utilisation de DOWNLINKS, c'est à dire de l'envoi de messages depuis le serveur Objenious vers la carte Arduino. Cependant, le protocole LoRaWan est pensé pour que ce type de message reste exceptionnel, pas pour envoyer un acknowledgement à chaque message reçu. Puisque le taux d'erreur reste acceptable pour notre application, nous avons donc fait le choix de nous passer de cette possibilité.

Transmission sur le réseau Lora



Dans un premier temps nous allons énoncer les caractéristiques du réseau LoRa.

C'est un protocole radio qui utilise la bande de fréquence 434 et 868 MHz en Europe et 915 MHz dans le reste du monde. LoRa ne s'appuie donc pas sur le protocole IP classique car il est trop consommateur d'énergie. En effet, les objets connectés sont prévus pour avoir une autonomie d'une dizaine d'années. Ils doivent émettre avec une consommation faible.

Le réseau est bidirectionnel mais nécessite un serveur LoRaWan.

Le débit est entre 0,3 et 50 kbit/s.

Le réseau LoRa s'est étendu au cours de l'année 2016 comme on peut le voir sur la carte ci-dessous.

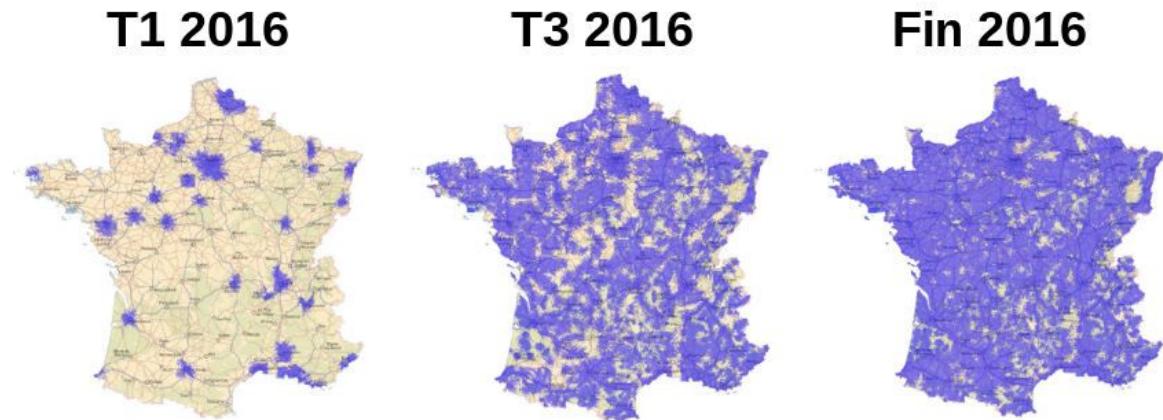


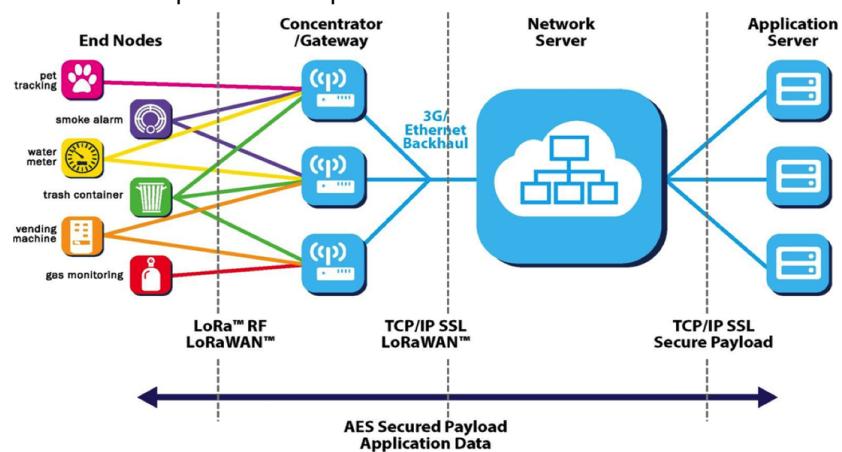
Figure 9 : Extension du réseau LoRa en 2016

La portée de chaque antenne est d'un rayon de 20 km en province et 1 km en zone urbaine.
Il y a trois classes de composants qui définissent le réseau LoRa :

- Les noeuds en fin de réseau formés par les capteurs par exemple
- Les passerelles
- Les serveurs

Les noeuds de la première classe communiquent avec les passerelles de la deuxième classe par le protocole radio. Ensuite ces passerelles sont reliées entre elles par Ethernet, Wifi ou 3G puis au serveur central vers lequel elles envoient toutes les données.

Voici deux schémas qui résume le protocole LoRa :



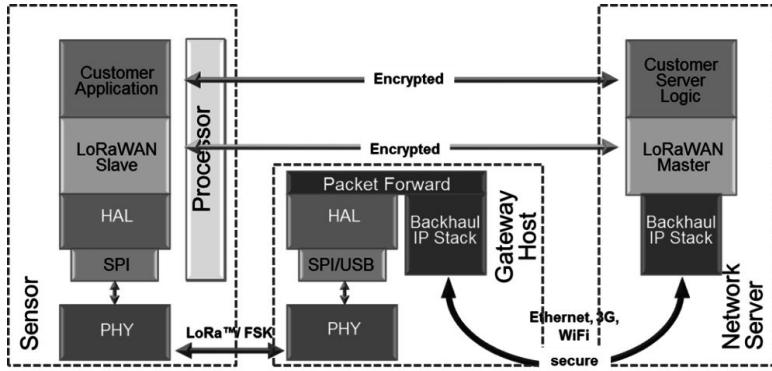


Figure 10 : À gauche, un objet connecté ; au milieu une antenne LoRa ; à droite le lien avec Internet

Les puces radios des capteurs peuvent émettre et recevoir des données mais ne peuvent faire ces deux fonctions en même temps. Les composants qui forment les passerelles du réseau LoRa peuvent gérer jusqu'à 5 000 noeuds au km².

Les composants pour la réalisation de nœuds sont facilement disponibles sous la forme de modules intégrés et certifiés R&TTE comme le Microchip RN2483. Côté logiciel, on trouve sur GitHub un projet LoRaMAC nœud et passerelle.

B. Software : notre solution EasyApp

1. Réception des données

a. Sur le serveur d'Objenious

Objenious nous fournit, sur leur plateforme spot.objenious.com, une interface permettant d'accéder aux messages envoyés. Les données sont affichées en hexadécimal, et aucun traitement particulier n'est possible, il n'est donc pas possible de se contenter de cette application web pour notre traitement. De plus, toutes les méta-données qui y sont disponibles peuvent être récupérées sur une plateforme externe de manière automatisée. Pour cela, deux méthodes nous sont possibles : utiliser l'API REST qui se trouve à l'adresse api.objenious.com, ou utiliser une fonctionnalité de la plateforme d'Objenious qui transmet automatiquement chaque message au format JSON par une requête HTTP POST vers notre serveur.

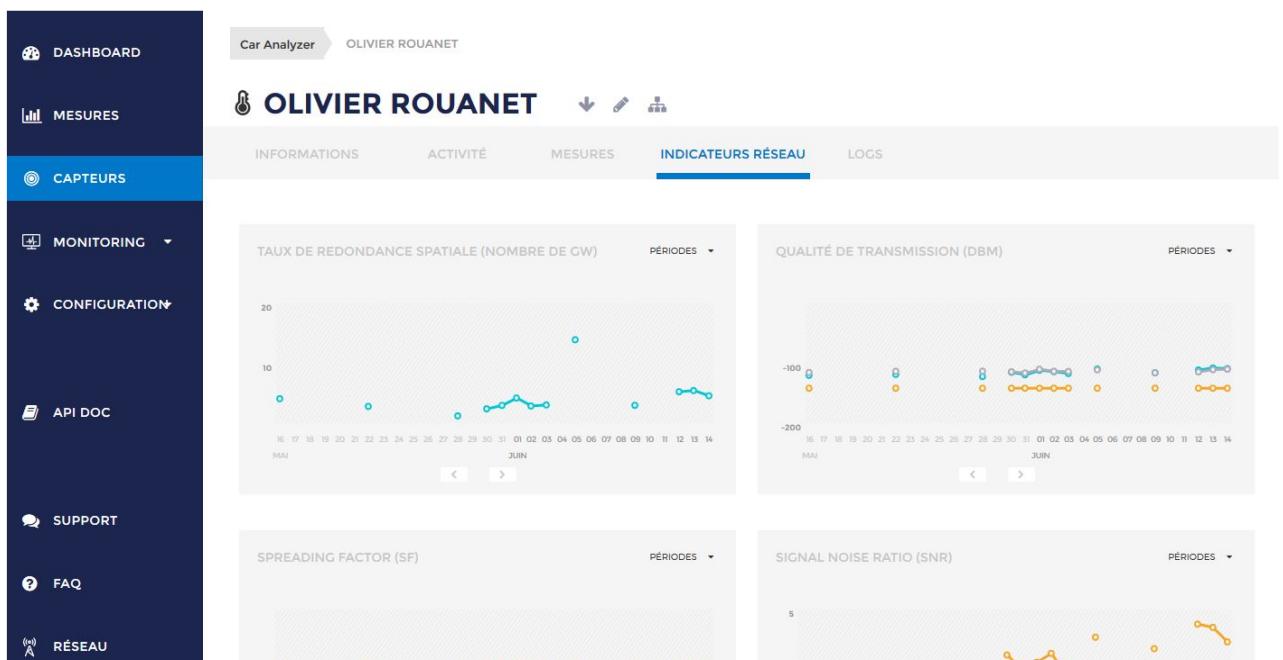


Figure 11 : Indications sur le Réseau Lora depuis spot.objenious.com



Figure 12 : Informations sur un capteur sur spot.objenious.com

TYPE MESSAGE	DATE MESSAGE	CONTENU	RÉSULTAT	COMPTEUR
UPLINK	14/06/2017 08:03:33	push-6221	✓	113
UPLINK	14/06/2017 08:02:45	push-6198	✓	109
UPLINK	14/06/2017 08:02:33	push-6194	✓	108
UPLINK	14/06/2017 08:02:22	push-6190	✓	107
UPLINK	14/06/2017 08:01:13	push-6182	✓	101
UPLINK	14/06/2017 08:01:01	push-6183	✓	100
UPLINK	14/06/2017 08:00:39	push-6182	✓	98

Figure 13 : Messages non décodés sur spot.objenious.com

SCÉNARIOS ALERTES

CRÉER UN SCÉNARIO

VOS SCÉNARIOS ET ALERTES

Vous pouvez activer, désactiver un scénario de la liste.

Afficher 25 éléments

NOM DU SCÉNARIO	TYPE D'ALERTE	GROUPE	DATE DE CRÉATION	ÉTAT	ACTIONS
MESSAGE FORWARDING	HTTP PUSH	CAR ANALYZER	08/02/2017 17:49:32	ON	

Premier Précédent 1 Suivant Dernier

Figure 14 : Scénario pour le transfert automatique des messages

b. Sur le serveur du Rézo

Une fois les données sur le serveur d’Objenious, nous avons besoin d’aller les chercher sur ce serveur et de les rapporter sur notre serveur à la résidence de CentraleSupélec. Pour cela, nous avons fait appel à la bibliothèque python Requests³. Tous les requêtes HTTP sont faites vers l’interface de programmation d’Objenious⁴.

La première étape pour mettre en place notre application web et notre base de donnée consiste à configurer le serveur utilisé. Celui-ci est une machine virtuelle, qui nous est prêtée par l’association étudiante Supélec Rézo, et est donc hébergé sur le campus de la résidence. Il s’agit d’un serveur tournant sous Linux, avec la distribution Debian 8. Pour y accéder, nous utilisons une connexion SSH, qui nous donne un accès sécurisé (chiffrement) à une console à distance. Depuis cette console, nous avons installé et configuré Apache2, un serveur HTTP très utilisé. Il faut noter que, puisque nous allons transmettre des données privées, il est nécessaire d’utiliser les protocoles SSL/TLS afin de chiffrer les connexions. On crée donc deux VirtualHost pour Apache2. Le premier écoute les requêtes en HTTP sur le port 80, et les redirige en HTTPS sur le port 443. Le VirtualHost a donc la configuration suivante :

```
<VirtualHost *:80>

    ServerName easydrive.rez-gif.supelec.fr
    ServerAlias easydrive.larez.fr easydrive

    ErrorLog ${APACHE_LOG_DIR}/error.log
```

³ <http://docs.python-requests.org/en/master/>

⁴ <https://api.objenious.com/v1>

```

CustomLog ${APACHE_LOG_DIR}/access.log combined

RewriteEngine on

RewriteCond %{SERVER_NAME} =easydrive.rez-gif.supelec.fr [OR]
RewriteCond %{SERVER_NAME} =easydrive.larez.fr [OR]
RewriteCond %{SERVER_NAME} =easydrive

RewriteRule ^ https:// %{SERVER_NAME}%{REQUEST_URI} [END,QSA,R=permanent]

</VirtualHost>

```

Par conséquent, le second écoute les requêtes HTTPS sur le port 443, il a donc la configuration suivante :

```

<VirtualHost *:443>

    ServerName easydrive.rez-gif.supelec.fr
    ServerAlias easydrive.larez.fr easydrive
    Alias /static/webDriver/local_static

    WSGIDaemonProcess easydrive
    python=/var/www/webDriver:/var/www/webDriver/lib/python3.4/site-packages

    WSGIProcessGroup easydrive
    WSGIScriptAlias / /var/www/webDriver/easydrive/wsgi.py process-group=easydrive

    <Directory /var/www/webDriver/local_static/>
        Require all granted
        Options -Indexes
    </Directory>

    <Directory /var/www/webDriver/easydrive/>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    SSLCertificateFile /etc/letsencrypt/live/easydrive.larez.fr/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/easydrive.larez.fr/privkey.pem
    Include /etc/letsencrypt/options-ssl-apache.conf

</VirtualHost>

```

Afin d'obtenir un certificat de sécurité signé, nécessaire à la mise en place correcte du chiffrement avec le protocole HTTPS, nous avons utilisé l'outil *Certbot*, fourni par l'Electronic Frontier Foundation, qui permet la génération et le renouvellement automatique d'un certificat, signé gratuitement par l'autorité de certification Let's Encrypt.

Ensuite, il a fallu mettre en place l'application Django. Il suffit pour cela de copier le dépôt

dans le dossier /var/www/webDriver, puis de lancer le script suivant pour collecter les fichiers statiques (CSS, JS et images), assurer les bonnes permissions de fichier, installer les bibliothèques nécessaires et s'assurer que la base de données est à jour et au bon format :

```
virtualenv --python=python3 /var/www/webDriver/.  
unzip webServer.zip -d /var/www/webDriver  
cd /var/www/webDriver  
source bin/activate  
pip install -Ur requirements.txt  
mkdir local_static  
cp -r objRequests/static/* local_static/  
python manage.py collectstatic --noinput  
chown -R www-data:www-data .  
python manage.py makemigrations  
python manage.py migrate  
python manage.py update  
deactivate  
systemctl restart apache2
```

Une fois tout cela mis en place, l'application est mise en production. Nous allons maintenant étudier quelques points qui méritent particulièrement de l'attention. Le code qui permet d'aller récupérer les mesures chez Objenious se trouve dans le fichier management/commands/update.py. En effet, celui-ci contient la classe Command qui hérite de la classe django BaseCommand qui permet de contrôler le lancement du script de récupération des données. Nous avons exploité cette fonctionnalité, particulièrement pratique dans la phase de développement du site web, puisque nous pouvions à tout moment supprimer toute notre base de données et la re-remplir par un simple lancement de la commande manage.py update.

Le fichier update.py se subdivise en trois parties : la récupération des ids des capteurs, le décodage des données et la suppression des données inexploitables, la création des instances dans la base de donnée.

Avant de faire toute requête à l'api Objenious, il faut bien sûr s'authentifier, il nous a donc fallu récupérer la clef fourni par Objenious, elle devra être spécifiée dans les *headers* des requêtes. Les réponses de l'api sont au format json⁵ :

⁵ <https://api.objenious.com/doc/doc.html>

```
{
    "id": 42,
    "link": "https://api.objenious.com/v1/devices/42",
    "label": "Example device",
    "group": {
        "id": 41,
        "link": "https://api.objenious.com/v1/groups/41"
    },
    "profile": {
        "id": 41,
        "link": "https://api.objenious.com/v1/profiles/41"
    },
    "properties": {
        "external_id": "45",
        "deveui": "0419b3000000128f",
        "appeui": "0419b300454f5231"
    },
    "status": "active"
}
```

Il nous faut donc extraire le nom et l'id des différents capteurs. Une fois que nous avons récupérer ces informations nous enregistrons les nouveaux capteurs dans la base de données grâce à la méthode `get_or_create()` de `Models.objects`. Ceci nous permet de nous assurer de ne pas créer deux fois le même capteur au cours d'un nouvel appel à `update.py`. Enfin nous passons à la récupération des messages pour chacun des capteurs.

```
url = 'https://api.objenious.com/v1/devices'
headers={'apikey':
'VB0w9QjJFj0UTzuZq1rhtnr3Uoxk376LKfHdh16BG2LZrnSTQwsrncaN+DwnweVG5Ul6MQ5h+z1ifqSS8J+
poA==  '}

r = requests.get(url,headers = headers)

devices = r.json()

y=0

for k in range(len(devices)):

    device_id = int(devices [k] ['id'])

    nom = devices [k] ['label']

device,d_created=Device.objects.get_or_create(device_id=device_id,nom = nom)

    if not (d_created):
```

```

        device.save()

        response
    requests.get(url+'/' +str(device_id)+'/messages', headers=headers, params=params)

        messages = response.json()['messages']
        messages = list(reversed(messages))

```

Pour la mise en production du site, plutôt que de relancer ce *script* à intervalle régulier en utilisant l'utilitaire *cron* fourni par UNIX, nous avons configuré l'application Objenious pour qu'elle transmette les messages vers l'URL easydrive.larez.fr/push. Nous pouvons donc traiter les messages quasiment en temps réel.

2. *Front-End: Application*

a. *Architecture de l'application*

Le site comporte principalement quatre pages : la page de connexion / accueil, la page listant les capteurs/conducteurs, la page listant les trajets, et la page listant les caractéristiques pour chaque trajet.

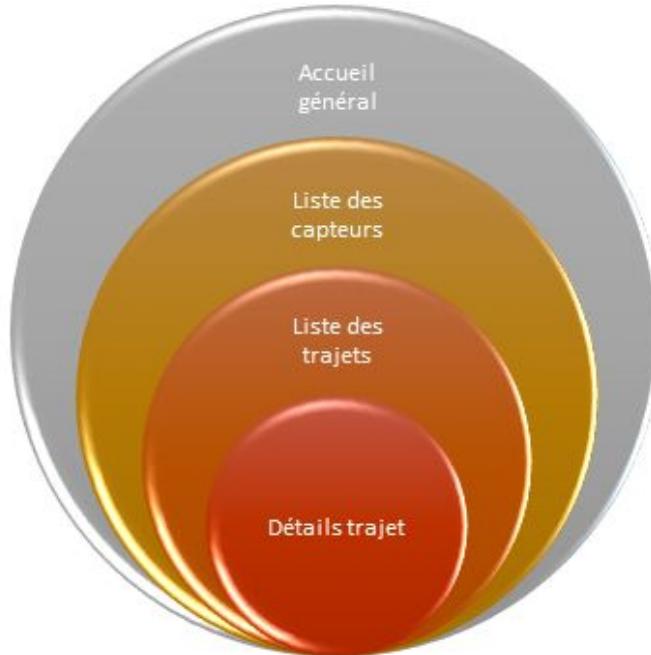


Figure 15 : Architecture synthétique du site

Voici un diagramme détaillé des informations disponibles sur chaque page du site et les liens disponibles entre chaque page.

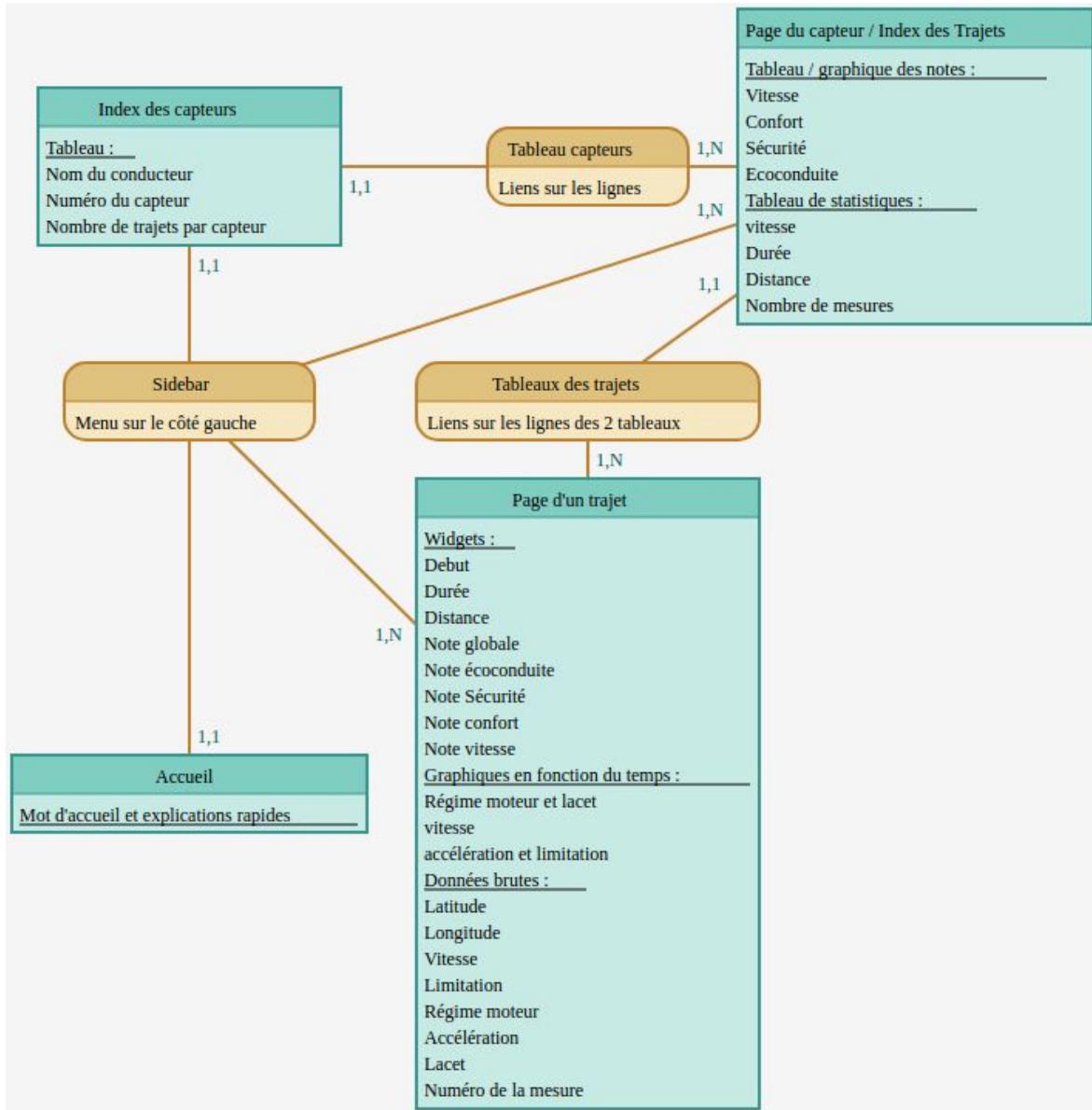


Figure 16 : Architecture détaillée du site

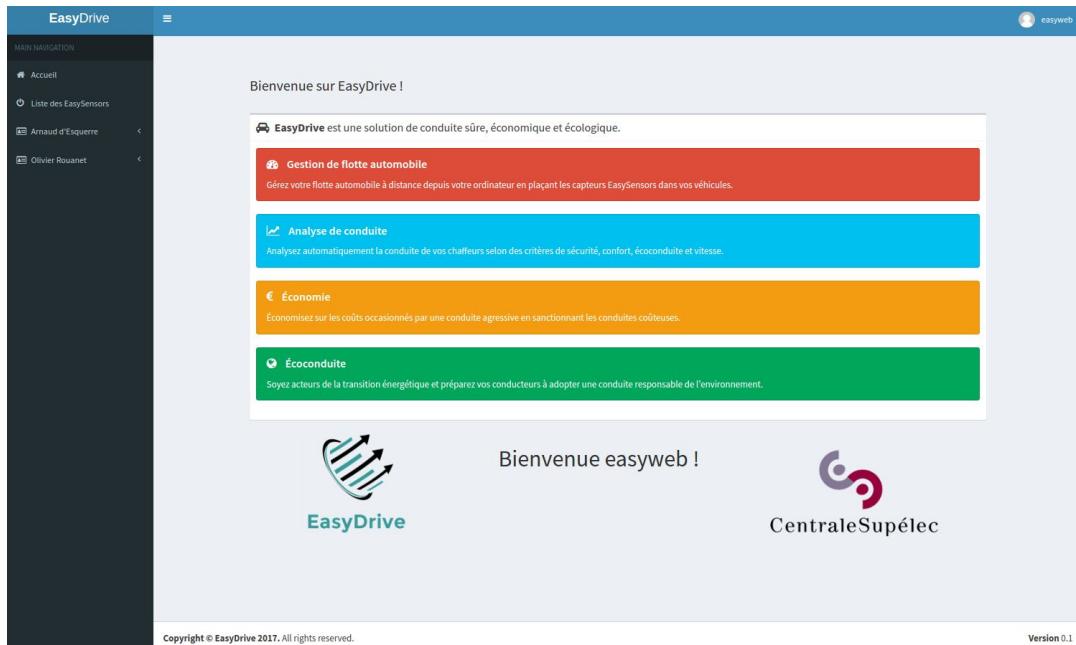


Figure 17 : Page Web de l'accueil du site

This screenshot shows a page for selecting a sensor. The layout is similar to Figure 17, with the 'EasyDrive' logo at the top and a sidebar with navigation links. The main content area is titled 'Sélection du capteur' and contains the message: 'Veuillez sélectionner un capteur pour afficher les caractéristiques et l'évaluation du chauffeur.' Below this is a table with three columns: 'Nom du conducteur', 'Numéro du capteur', and 'Nombre de trajets'. Two rows are listed: 'Arnaud d'Esquerre' (number 562949953422031, 0 trips) and 'Olivier Rouanet' (number 562949953422029, 20 trips). The footer includes 'Copyright © EasyDrive 2017. All rights reserved.' and 'Version 0.1'.

Figure 18 : Page Web de la liste des capteurs

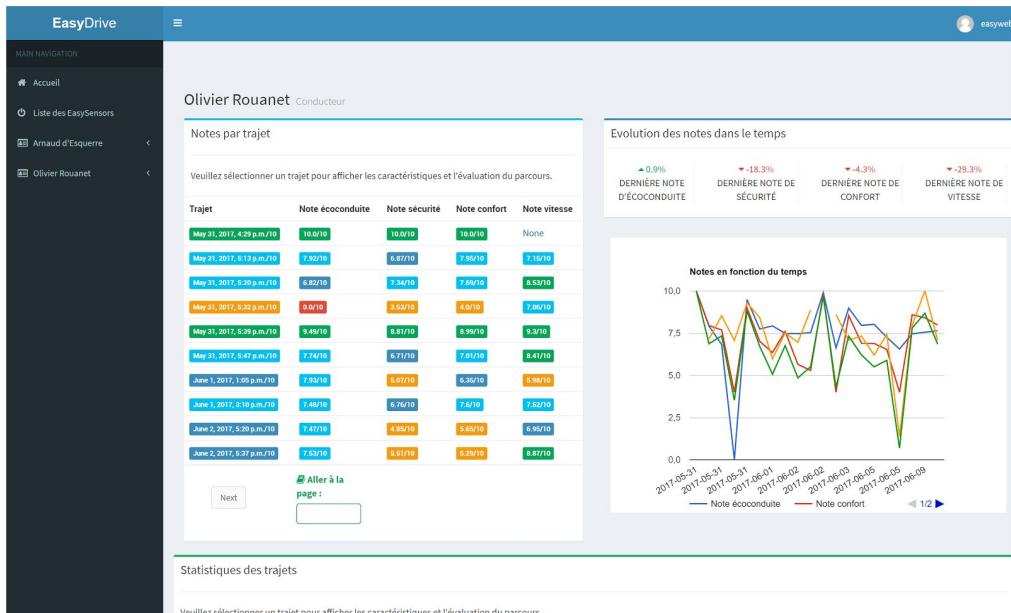


Figure 19 : Page Web de la liste des trajets pour un conducteur

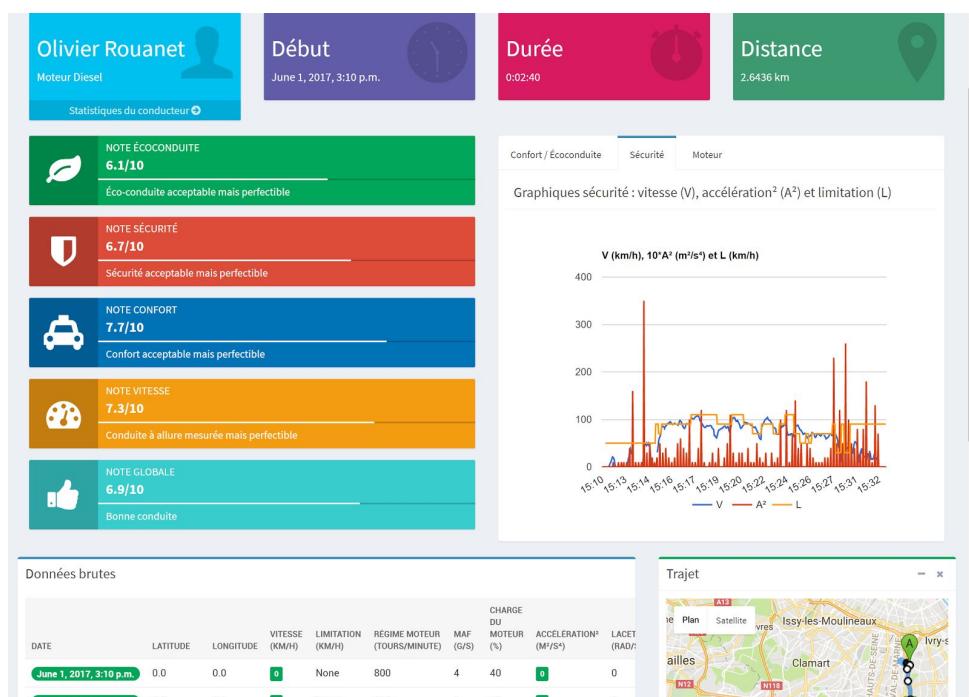


Figure 20 : Page Web de l'analyse des trajets

Sur une page trajet, l'affichage des résultats de l'analyse présente cinq sections :

- Une section récapitulant les caractéristiques clés d'un trajet
- Une section majeure présentant les évaluations de la conduite
- Une section affichant les graphiques des données importantes
- Une section qui montre les données brutes du trajet
- Une section qui intègre l'itinéraire géographique.

b. Utilisation de services externes : Google Charts⁶, Google Maps API⁷ et admin LTE

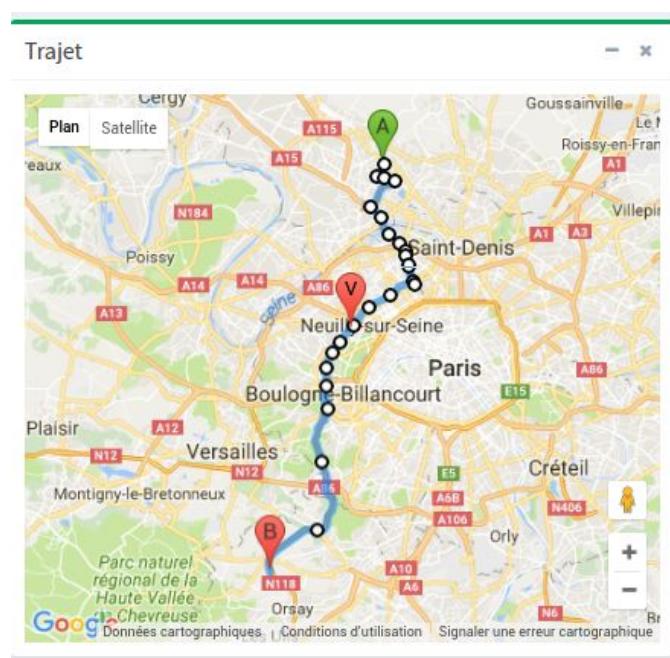
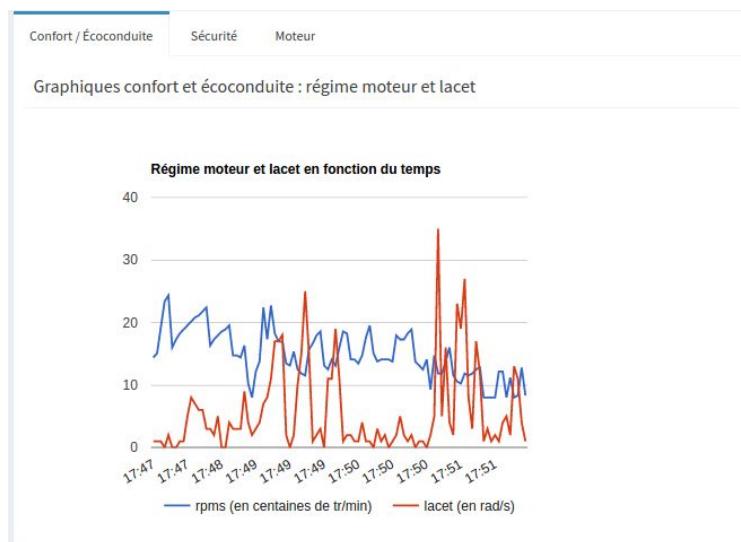
Nous nous sommes très vite rendus compte de l'intérêt de l'affichage des résultats de notre étude grâce à des supports visuels variés, tels des graphiques ou des cartes. Nous nous sommes donc orientés vers les services *Google Charts* pour les graphiques et *Google Maps* pour les cartes.

Concernant le premier, il existe des bibliothèques, se basant sur django tel que *graphos*, qui nous ont permis d'afficher des graphiques sans devoir manipuler le langage Javascript. La définition des options dans le fichier views.py et la présence d'une balise {{chart1.as_html}} suffisaient à afficher des graphes :

```
chart1 = LineChart(SimpleDataSource(data=data1),options={'title': 'V (km/h), A (m/s²) et L (km/h)', 'explorer': {
    'axis': 'horizontal',
    'maxZoomIn': 4.0},
    'legend': {'position': 'bottom'},
    'width': '600',
    'height': '400',
})
```

Néanmoins, pour ce qui est de la cartographie, nous avons dû utiliser Javascript. En effet, nous affichons pour chaque trajet une carte qui indique le chemin par lequel l'utilisateur est passé ainsi que le point qui correspond aux mesures présentes dans le tableau.

Pour cela, il a fallu extraire, parmi toutes les mesures du trajets, un maximum (fixé à vingt-trois par *Google Maps*) de mesures équiréparties dans le



⁶ <https://developers.google.com/chart/>

⁷ <https://developers.google.com/maps/?hl=fr>

temps. Celles-ci ne devaient pas comporter une latitude ou une longitude absente, puisque Google retournerait une erreur. Une des difficultés a été de transmettre depuis Django (qui est un *framework Python*⁸) les valeurs de la base de données à Javascript afin de pouvoir faire appel à l'Api de Google Maps. Nous avons pour cela dû passer par le format Json grâce à la commande `json.dumps()` puis utiliser l'envoie des données au *template* grâce à Django : `{{json_mesure|safe}}`

```
//Javascript
function initialize() {
    var mesure = {{json_mesures|safe}}
    directionsDisplay = new google.maps.DirectionsRenderer()
    var paris = new google.maps.LatLng(48.708808, 2.163690);

    var myOptions = {
        zoom: 6,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        center: paris
    }
    map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
    directionsDisplay.setMap(map);
    var uluru = {lat:mesure[0] , lng: mesure[1]};
    var marker = new google.maps.Marker({
        position: uluru,
        map: map,
        label : 'V'
    });

    calcRoute();
}
}
```

Le dernier service externe que nous avons utilisé est Admin LTE. Celui-ci s'appuie en partie sur [Bootstrap](#) (*framework CSS et HTML*) afin de construire efficacement une interface graphique agréable. Il permet par exemple de mettre en valeur rapidement des données dans un tableau à l'aide de balises *span* et de *class* :

```
<span class="badge bg-green">May 31, 2017, 5:47 p.m.</span>
```

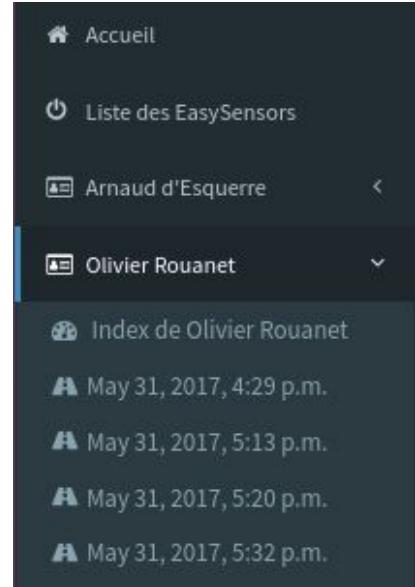
⁸ cf. *supra* III.A.2.

DATE	LATITUDE	LONGITUDE	VITESSE (KM/H)	LIMITATION (KM/H)	RÉGIME MOTEUR (TOURS/MINUTE)	MAF (G/S)	CHARGE DU MOTEUR (%)	ACCÉLÉRA (M ² /S ⁴)
May 31, 2017, 5:47 p.m.	48.709554	2.187007	23	50	1440	12	28	0.0
May 31, 2017, 5:47 p.m.	48.709554	2.187007	24	50	1504	28	59	0.0

Figure 21 : Exemple d'utilisation de Admin LTE pour l'affichage des données brutes

Admin LTE présentait aussi l'avantage d'être facilement intégrable à Django grâce à la bibliothèque [django-adminlte2](#). Nous avons ainsi pu faire des onglets sans avoir à programmer en Javascript ou JQuery. En effet, en spécifiant les bons paramètres dans le template main_sidebar.html nous avons obtenu le menu déroulant de la partie gauche du site.

Il restait néanmoins une certaine difficulté technique puisque ce *template* est appelé quelque soit la page du site. Le système modèle-vue-template⁹ n'est donc plus respecté puisque plusieurs vues correspondent à un *template*. Il a donc fallu utiliser les inclusion_tag de django qui permettent d'utiliser une vue quelque soit le *template*.



c. Difficultés rencontrées

Concernant l'interface graphique nous nous sommes très vite rendus compte qu'au regard de la quantité de mesures par trajet (de plusieurs centaines à plusieurs milliers), nous devions paginer nos tableaux. Si cela peut apparaître de prime abord immédiat, la mise en œuvre fut laborieuse. Heureusement, Django offre des outils adaptés (*Paginator*¹⁰). Cela reste cependant assez compliqué puisqu'il faut passer le numéro de la page en paramètre de la vue. De plus, il a fallu afficher des boutons qui permettent, non seulement de naviguer vers la page précédente ou à la page suivante, mais aussi d'aller à une page entrée par l'utilisateur. Pour cela nous avons utilisé des formulaires, comme il est courant de faire sur des pages internets.

Date	Longitude	Latitude	Vitesse (km/h)	Limitation (km/h)	Régime moteur (tours/minute)	MAF (g/s)	Charge du moteur (%)	Accéléra (m ² /s ⁴)
May 31, 2017, 5:49 p.m.	48.709693	2.171486	44	90	1312	16	0	0.0
May 31, 2017, 5:49 p.m.	48.709693	2.171486	41	90	1248	12	0	0.0
May 31, 2017, 5:50 p.m.	48.709693	2.171486	38	90	1408	16	0	13
May 31, 2017, 5:50 p.m.	48.709693	2.171486	28	90	1312	24	19	0.0
May 31, 2017, 5:50 p.m.	48.709693	2.171486	27	90	1600	8	10	0.0

Page 5 sur 10

Previous Next

Aller à la page :

⁹ cf. supra III.A.2.

¹⁰ <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-django/la-pagination-2>

3. Back-End: Traitement des données et évaluation

a. Architecture de la base de donnée

Avant de mentionner du traitement des données, détaillons la relation entre nos différents groupes de données afin de bien comprendre comment celles-ci vont être stockées. Les informations que nous allons manipuler se subdivisent en trois grandes catégories qui forment aussi trois des quatre tables de notre base de donnée : les mesures, les trajets, et les capteurs. En ajoutant, toutes les données que nous enregistrons, nous pouvons établir le schéma suivant de la base de donnée et donc de la relation entre tous les objets que nous manipulons. La table Dump sert uniquement à récupérer les messages qui n'auraient pas le format attendu.

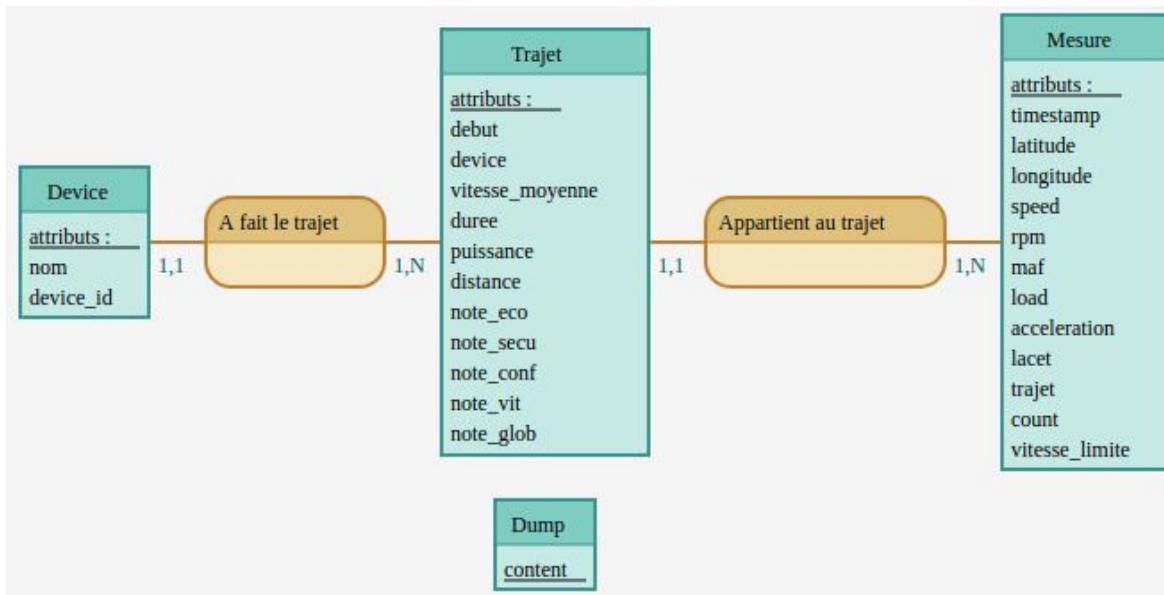


Figure 22 : Relations entre les données

b. Prétraitement

Le prétraitement a lui aussi lieu dans le fichier update.py. Il s'agit de transformer une réponse à une requête en format Json en des objets à l'intérieur de notre base de donnée. La réponse de la requête ressemble à ceci :

```

1+ {
2+   "messages": [
3+     {
4+       "id": "uplink-47-5879",
5+       "device": 562949953422029,
6+       "type": "uplink",
7+       "timestamp": "2017-06-09T16:44:06Z",
8+       "count": 47,
9+       "payload": [
10+         {
11+           "timestamp": "2017-06-09T16:44:06Z",
12+           "data": {
13+             "push": -6340
14+           }
15+         }
16+       ],
17+       "payload_encrypted": "01e96c7265f7dcf6b742b3c1bbd02ed9399016fac2f3fb60f8178779ff45d17f8f10f5463ce3ec",
18+       "payload_cleartext": "02e73c510020fdc9100c0a07001c1617151f00171d5c1c0101010003070a0b05000b0e0d12062f",
19+       "protocol_data": {
20+         "port": 5,
21+         "AppNonce": "c85da3",
22+         "DevNonce": "5879",
23+         "DevAddr": "0fe5elee",
24+         "NetID": "000007",
25+         "SF": 12,
26+         "appEui": "70b3d59ba00000004",
27+         "devEui": "70b3d59ba00004d2",
28+         "keymanagerId": "LNSInternalKMS",
29+         "best_gateway_id": "M14803",
30+         "gateways": 3,
31+         "noise": -118.9314718056,
32+         "signal": -118.9314718056,
33+         "rssI": -112
34+       },
35+       "lat": 48.692297153265,
36+       "lng": 2.1797374997537,
37+       "geolocation_type": "network",
38+       "geolocation_precision": 1500
39+     },
40+     {
41+       "id": "uplink-46-5879",
42+       "device": 562949953422029,

```

Figure 23 : Réponse à une requête

Dans cette réponse figurent plusieurs types de messages : les “*downlinks*” et les “*uplinks*” entre autres. Nous traiterons les *uplinks* puisque ceux-ci correspondent aux messages transmis depuis l’*EasySensor*. Les valeurs que nous récupérerons sont count, timestamp et payload_cleartext. La partie qui nous intéresse le plus se trouve dans payload_cleartext car elle correspond aux données que nous avons envoyées grâce au module Arduino et au réseau LoRa. Ce message est en hexadécimal. Il va donc falloir le découper de manière à respecter le format de donnée (cf. 4. Emission des données) et le convertir en nombre flottant. De plus, nous vérifions au début si le message n’est pas trop grand et nous vérifions à la fin si nous avons bien reçu le caractère de fin “/” qui correspond à “2F” en hexadécimal. Voici la partie du code qui effectue cette fonction. Elle est assez fastidieuse. En effet, le message ne possédant pas de séparateur nous avons été obligés de trouver les bons indices à la main.

```
if m_type =='uplink':
    if len(messages [i]['payload_cleartext'])<=78:
        try :
            data = messages [i]['payload_cleartext']
            lat = int(data[:8],16)/1000000
            lng = int(data[8:16],16)/1000000
            speeds = [int(data [16:18], 16), int(data [18:20], 16), int(data [20:22], 16), int(data [22:24], 16), int(data [24:26], 16)]
            rpms = [32*int(data [26:28], 16), 32*int(data [28:30], 16),
32*int(data [30:32], 16), 32*int(data [32:34], 16), 32*int(data [34:36], 16)]
            loads =[int(data [36:38], 16), int(data [38:40], 16), int(data [40:42], 16),int(data [42:44], 16), int(data [44:46], 16)]
            mafs =[4*int(data [46:48], 16), 4*int(data [48:50], 16), 4*int(data [50:52], 16), 4*int(data [52:54], 16), 4*int(data [54:56], 16)]
            cf = 128*16000**2/(9.81**2)
            accs =[int(data [56:58], 16)/cf, int(data [58:60], 16)/cf,
cf*int(data [60:62], 16)/cf,int(data [62:64], 16)/cf,int(data [64:66], 16)/cf]
            lacets =[int(data [66:68], 16), int(data [68:70], 16), int(data [70:72], 16), int(data [72:74], 16), int(data [74:76], 16)]
            message_ok =data[76:78]
            if message_ok =='2f':
```

L’étape d’après consistait à identifier si la mesure appartenait à un nouveau trajet ou non. En effet, puisque nous traitons les données de manière séquentielle il s’agit de les affecter au bon trajet. Pour ceci, il faut donc créer les trajets au plus tôt. La présence pour chaque message d’un compteur rend le travail plus facile. Les messages comportant une valeur d’une compteur égale à un ne sont en pratique jamais perdus. En effet, avant d’envoyer des messages au réseau Lora le capteur met un certain temps à se connecter. Pendant cette phase, le capteur envoie des messages “join” au serveur Objenious. Dès que le capteur est connecté il envoie le message. Il ne peut donc pas y avoir de problèmes de connexion pour ce message.

```
count = messages [i] ['count']
nouveau_trajet = False
```

```

if count == 1 :
    nouveau_trajet = True

```

S'en suivent la transformation des données erronées en valeur inexistante dans la base de donnée (*None*), pour ne pas fausser les futurs calculs (de variance par exemple). Toutes les valeurs suivantes sont transformées en *None* :

- vitesse =255
- rpms = 224
- maf =252
- load = 255

Afin d'établir des notes en rapport avec les limitations de vitesse nous avons aussi fait appel à l'interface de programmation Nokia¹¹. Comme précédemment, nous spécifions des paramètres pour la requête et obtenons une réponse en format Json. Nous en extrayons la vitesse limite.

```

url_vit =
'http://route.st.nlp.nokia.com/routing/7.2/getlinkinfo.json?app_id=DemoAppId01082013GAL&app_
_code=AJKnXv84fjrb0KIHawS0Tg'

if lat !=0 and lng !=0 :
    params_vit ={'waypoint' : str(lat)+','+str(lng)}
    r_vit = requests.get(url = url_vit, params = params_vit)
    r_vit = r_vit.json()
    vitesse_limite =int(r_vit['response']['link'][0]['speedLimit']*3.6)

```

Enfin, s'il s'agit d'un nouveau trajet nous l'enregistrons en vérifiant qu'il n'a pas été sauvegardé auparavant. Au contraire, s'il s'agit d'une mesure n'étant pas la première d'un trajet, nous vérifions que cette mesure n'existe pas déjà, et si c'est le cas nous l'associons au trajet le plus récent. Dans ce deuxième cas nous disons que l'heure de la mesure correspond à l'heure de début du trajet plus le compteur Objenious multiplié par dix secondes plus le numéro de la mesure à l'intérieur de la salve de cinq mesures multiplié par deux secondes.

```

if nouveau_trajet :

    trj,created_trj= Trajet.objects.get_or_create(debut =
tmzone.make_aware(datetime.datetime.strptime(tmstamp[:-1],
'%Y-%m-%dT%H:%M:%S')+datetime.timedelta(hours=2),timezone = TIME_ZONE),device = device)

    if not (created_trj):
        trj.save()

    tsp = tmzone.make_aware(datetime.datetime.strptime(tmstamp[:-1],
'%Y-%m-%dT%H:%M:%S')+ datetime.timedelta (hours=2,seconds = j*2),timezone =TIME_ZONE)

    mesure,created = Mesure.objects.get_or_create(device=device,timestamp = tsp,latitude
= lat, longitude = lng,speed = speeds [j],rpm = rpms [j],load = loads [j],maf=mafs
[j],acceleration = accs [j], lacet = lacets [j],count = count,trajet = trj, vitesse_limite
= vitesse_limite)

```

¹¹ <http://route.st.nlp.nokia.com/>

```

if not (created):
    mesure.save()

else :
    trj2=Trajet.objects.get(debut = begin, device = device)
    tsp2 = begin + datetime.timedelta (seconds = (count-1)*10+j*2)

    if list(Mesure.objects.filter(device=device,latitude = lat, longitude = lng, speed =
speeds [j],rpm = rpms [j],load = loads [j],maf = mafs [j],acceleration = accs [j],lacet =
lacets [j],count = count,vitesse_limite = vitesse_limite)) ==[]:
        mesure, created = Mesure.objects.get_or_create(device=device,timestamp =
tsp2 ,latitude = lat,longitude = lng,speed = speeds [j],rpm = rpms [j],load = loads [j],maf =
= mafs [j], acceleration = accs [j],lacet = lacets [j],trajet = trj2, count = count,
vitesse_limite = vitesse_limite)

    if not (created):
        mesure.save()

```

c. De la base donné à l'affichage : utilisation de Django

L'intérêt du framework Django réside dans sa capacité à aller chercher des données dans la base de données, les manipuler dans des vues et les renvoyer vers les *templates*¹² selon le schéma suivant:

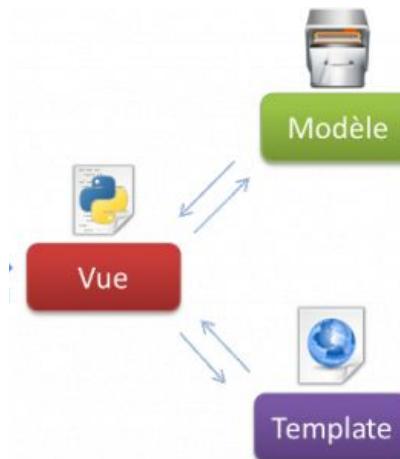


Figure 23 : Modèle permettant le traitement et l'affichage

Dans la vue trajet, nous commençons par chercher les données dans la base de données :

```

rpms = list(Mesure.objects.filter(trajet = trajet).values_list('rpm'))
rpms =[item[0] for item in rpms if item[0]]

limites = list(Mesure.objects.filter(trajet = trajet).values_list('vitesse_limite'))
lim = limites

limites =[item[0] for item in limites ]

```

¹² cf. supra III.A.2.

```

limites_no_none =[item[0] for item in lim if item[0] ]
accs = list(Mesure.objects.filter(trajet = trajet).values_list('acceleration'))
accs =[item[0] for item in accs ]
lacets = list(Mesure.objects.filter(trajet = trajet).values_list('lacet'))
lacets =[item[0] for item in lacets ]
speeds = list(Mesure.objects.filter(trajet = trajet).values_list('speed'))
speeds =[item[0] for item in speeds ]
mafs = list(Mesure.objects.filter(trajet = trajet).values_list('maf'))
mafs =[item[0] for item in mafs ]
loads = list(Mesure.objects.filter(trajet = trajet).values_list('load'))
loads =[item[0] for item in loads ]
##Graphique et info map
latitudes = list(Mesure.objects.filter(trajet = trajet).values_list('latitude'))
latitudes = [item[0] for item in latitudes if item[0]!=0]
longitudes = list(Mesure.objects.filter(trajet = trajet).values_list('longitude'))
longitudes = [item[0] for item in longitudes if item[0]!=0]
len_lat =len(latitudes)
dates = list(Mesure.objects.filter(trajet = trajet).values_list('timestamp'))
dates =[item[0]+datetime.timedelta(hours=2) for item in dates]

```

Une fois ces données récupérées, nous pouvons en commencer l'exploitation : par exemple nous créons un paramètre carburant qui indique le type de moteur du véhicule. Pour ce faire, nous regardons si des données MAF sont envoyées par l'OBD : si oui, nous sommes en présence d'un moteur Diesel.

```

try :
    if mafs[0] != None :
        carburant = "Diesel"
    else :
        carburant = "essence"
except :
    carburant = "essence"

```

Puis nous traitons les données brutes de conduite pour calculer les notes¹³ grâce aux fonctions Python définies plus tard.

```

p_acc=param_acc(accs)
p_vit=param_vit(speeds,limites_no_none)
p_vir=param_vir(lacets,accs)

```

¹³ cf. *supra* II.B.3.e

```

p_rpm=param_rpm(rpms)

note_eco = note_ecoconduite(p_acc,p_rpm)
note_secu = note_securite(p_vit,p_acc)
note_conf = note_confort(p_vir,p_acc)
note_vit = note_vitesse(p_vit)
note_glob = note_globale(note_eco,note_secu,note_conf,note_vit)

```

Une fois ces notes statistiques calculées¹⁴, nous traitons au cas par cas ce que nous afficherons dans le *template* trajet.html, c'est-à-dire le message personnalisé correspondant à la note attribuée et la barre de progression proportionnelle à la note obtenue. Ces traitements sont réalisés dans l'optique de rendre notre *EasyApp* aussi *user-friendly* que possible. Notez que le logo associé à la note globale est un paramètre qui lui aussi dépend de la note obtenue. Voici un exemple pour la préparation de l'affichage de la note globale :

```

if note_glob != None:

    trajet.note_glob = round(note_glob,2)
    trajet.save()

    if note_glob < 2 :
        message_glob = "Conduite inacceptable"
        logo_glob = "fa fa-warning"

    elif note_glob>=2 and note_glob<4 :
        logo_glob = "fa fa-thumbs-down"
        message_glob = "Mauvaise conduite"

    elif note_glob>=4 and note_glob<6 :
        message_glob = "Conduite moyenne"
        logo_glob = "fa fa-meh-o"

    elif note_glob >=6 and note_glob<8 :
        message_glob = "Bonne conduite"
        logo_glob = "fa fa-thumbs-up"

    else :
        message_glob = "Excellente conduite"
        logo_glob = "fa fa-star"

    progress_bar_glob = "width:"+str(note_glob*10)+"%"

else :
    progress_bar_glob = "width: 0%"
    message_glob = "Note non-disponible"

```

¹⁴ cf. *supra* II.B.3.e

Comment les données brutes et les notes sont-elles affichées dans la page trajetx.html ? Le template s'en charge avec un code HTML utilisant les variables Python issues de la vue trajet. Grâce à la dépendance des messages et de la bar de progression mentionnées plus tôt et consignée dans des variables spécifiques, nous pouvons afficher le rendu souhaité. Par exemple pour l'affichage de la note globale, nous codons (grâce à la bibliothèque AdminLTE¹⁵) :

```
<div class="info-box bg-teal">
    <span class="info-box-icon"><i class="{{logo_glob}}" aria-hidden="true"> </i> </span>
    <div class="info-box-content">
        <span class="info-box-text">Note globale</span>
        <span class="info-box-number"> {{note_glob|floatformat}} / 10 </span>
        <!-- The progress section is optional -->
        <div class="progress">
            <div class="progress-bar" style="{{progress_bar_glob}}></div>
        </div>
        <span class="progress-description">
            {{message_glob}}
        </span>
    </div><!-- /.info-box-content -->
</div><!-- /.info-box -->
```

Ceci permet obtenir le rendu suivant, selon la note globale :



Figure 24 : Exemple de rendus différents à l'affichage selon la note globale

¹⁵ cf. supra II.B.2.b.

d. Confidentialité

Grâce au décorateur `login_required`, si l'utilisateur est connecté, il exécute la vue normalement mais pas s'il est déconnecté. Le code de la vue peut légitimement considérer l'utilisateur comme connecté.

En cas de déploiement auprès de plusieurs clients, notre système de *login* permet d'isoler les données des capteurs de chaque client. De plus, si notre application était dédiée au particuliers, nous pourrions orienter chaque client vers ses données propres sans qu'il ne puisse avoir accès aux données des autres particuliers en protégeant la vue d'un utilisateur.

e. Évaluation des paramètres de conduite

Nous traitons les données de façon à évaluer quatre paramètres de la conduite. On note σ_X l'écart-type de X.

- Paramètre accélération du véhicule a

$Var(accélération^2)$ mesure les fortes accélérations et freinages brusques. On note ce paramètre sur 10 grâce à une échelle linéaire déterminée empiriquement : $a = 10 - \frac{Var(accélération^2)}{2.5}$ si $Var(accélération^2) \leq 25 \text{ m}^4/\text{s}^8$ et $a = 0$ si $Var(accélération^2) > 25 \text{ m}^4/\text{s}^8$ de façon à attribuer une note de 0/10 pour des accélérations telles que $\sigma_{|accélération|} > 2,3 \text{ m/s}^2 (0.2 * g)$. Nous avons fait des tests sur des trajets agressifs où $\sigma_{|accélération|} > 2,3 \text{ m/s}^2$.

Ce sont les accélérations ou forces que l'être humain ressent (ou qui peut jouer sur un objet), et non une vitesse constante sans accélération, qui sont les changements ou dérivées des vitesses, et dérivées seconde des distances, voire aussi les variations d'accélérations ou jerks.

Pour un transport usuel (ascenseur, avion, voiture, train, ...), les accélérations horizontales ou verticales sont étudiées pour être réduites pour le confort des passagers, souvent à environ 0,1 g et ne dépassant guère 0,4 g.

Voici le code Python de la fonction correspondante :

```
def param_acc(accs):
    try :
        var = variance(accs)
        if var<=25 :
            param_acc = 10-variance(accs)/2.5
        else :
            param_acc = 0
    except :
        param_acc = None
    return(param_acc)
```

- Paramètre vitesse linéaire du véhicule v

Le principe est de pénaliser les dépassements de limitation de vitesse selon le rapport $\frac{\eta}{n}$ où $\eta = \sum \frac{\text{Excès de vitesse}}{\text{Vitesse limite}}$ et n est le nombre de données de la vitesse pour un trajet, de façon à prendre en compte les excès de vitesses par rapport à la limitation de vitesse et l'occurrence de ces excès sur un trajet. Nous avons fixé à 0/10 la note à un conducteur qui roulerait à 70 km/h sur un trajet limité à 50 km/h pendant 50% du temps du trajet : $v = 10 * (1 - \frac{\eta}{0.2*n})$.

Voici le code Python de la fonction correspondante :

```
def param_vit(speeds, limites_no_none):  
    ratio_depassemement_total = 0  
    pas_de_limite=0  
    for j in range(len(speeds)):  
        try :  
            if speeds[j]>limites_no_none[j]:  
                ratio_depassemement_total+=(speeds[j]-limites_no_none[j])/limites_no_none[j]  
        except :  
            pas_de_limite+=1  
  
    if pas_de_limite != len(speeds):  
        param_vit = ratio_depassemement_total/(len(speeds)-pas_de_limite)  
        param_vit = (10*(1-param_vit/0.2))  
    else :  
        param_vit = None  
    return (param_vit)
```

- Paramètre virage t

Nous nous intéressons à $lacet * accélération^2$ car sur rond-point l'accélération joue le rôle déterminant tandis que sur un parking ou en agglomération c'est surtout la vitesse de lacet qui devient importante lorsque le virage est très serré et rapide.

Les seuils de détection de virage et de virage dangereux ont été déterminés sur plusieurs tests de virages différents, à allures, agressivités et accélérations différentes. Au regard des données de ces virages, nous avons conclus que si $lacet * accélération^2 > 10^\circ * m^2 * s^{-5}$, nous détectons un virage puis si $lacet * accélération^2 > 700^\circ * m^2 * s^{-5}$ alors nous détectons un virage dangereux. Nous évaluons le paramètre virage via $t = (1 - \frac{\text{Nombre de virages dangereux}}{\text{Nombre de virages}}) * 10$.

Voici le code Python de la fonction correspondante :

```
def param_vir(lacets, accs):  
    virages = 0
```

```

virages_dangereux = 0

try :
    for i in range(len(lacets)):
        if lacets[i] and accs[i]:
            if lacets[i] > 10 :
                virages +=1
            if lacets [i] * accs [i] > 700 :
                virages_dangereux +=1
    if virages !=0 :
        param_vir = (1-virages_dangereux/virages)*10
    else :
        param_vir = 10
except :
    param_vir = None
return(param_vir)

```

- Paramètre régime moteur r

$Var(régime\ moteur)$ caractérise des phases de sur-régime et de sous-régime par rapport à la moyenne du régime moteur. Nous effectuons l'évaluation suivante :
 $r = \frac{10}{\sigma_{min}^2 - \sigma_{max}^2} * (Var(régime\ moteur) - \sigma_{max}^2)$ si $\sigma_{min} < \sigma_{rpm} < \sigma_{max}$, $r = 10$ si $\sigma_{rpm} \leq \sigma_{min}$ et $r = 0$ si $\sigma_{rpm} \geq \sigma_{max}$ avec $\sigma_{max} = 800\ tr/min$ et $\sigma_{min} = 300\ tr/min$ de façon à accorder une note de 10/10 si $\sigma_{rpm} \leq \sigma_{min}$ et de 0/10 si $\sigma_{rpm} \geq \sigma_{max}$.

Voici le code Python de la fonction correspondante :

```

def param_rpm(rpms):
    seuil_min = 300**2
    seuil_max = 800**2
    try :
        var = variance(rpms)
        if var<=seuil_min:
            param_rpm=10
        elif var>=seuil_max:
            param_rpm = 0
        else:
            param_rpm = 10/(seuil_min-seuil_max) * (var - seuil_max)
    except :
        param_rpm = None
    return(param_rpm)

```

Toutefois, cette notation est imparfaite, car en ville ou dans un trafic dense, les variations moteurs sont plus fortes que sur une route à trafic fluide car les démarriages/arrêts sont plus fréquents. N'ayant pas accès au trafic extérieur où à la signalisation routière, nous n'avons pas pu corriger ce biais. Par ailleurs, n'ayant pas accès au rapport de transmission enclenché, nous ne pouvons sanctionner un conducteur à fort régime moteur à tort ("en troisième à 70 km/h") par rapport à une conduite sur autoroute "en cinquième" et donc à fort régime moteur justifié.

f. Notation d'un trajet

- Note écoconduite en fonction des paramètres a et r

La note écoconduite correspond au paramètre a pondéré de 30% et du paramètre r pondéré de 70% car l'accent a été mis sur la maîtrise du régime moteur dans la note d'écoconduite. Le paramètre a reflétant les accélérations et freinages brusques, il nous semblait intéressant de l'inclure dans la note écoconduite.

- Note sécurité en fonction des paramètres a , v

La note sécurité correspond au paramètre a pondéré de 50% et au paramètre v pondéré de 50% pour mettre à pied d'égalité les accélérations et freinages brusques avec les dépassements de limitation de vitesse dans la note sécurité

- Note confort des paramètres t , a

La note confort correspond au paramètre t pondéré de 40% et du paramètre a pondéré de 60%. Le confort est caractérisé plutôt par les accélérations et freinages brusques mais aussi par l'agressivité des virages.

- Note de respect des limitations de vitesse en fonction du paramètre v

La note de respect des limitations de vitesse correspond directement au paramètre v .

- Note globale en fonction des autres notes

La note globale est une note qui fait la moyenne arithmétique des autres notes.

g. Adaptation de la notation à chaque client

Selon le client à qui nous vendrions l'application, nous pouvons insister sur certains critères plutôt que d'autres.



U B E R

Les entreprises de Voiture de Transport avec Chauffeur telle *Uber* ou les entreprises de service de covoiturage comme *Blablacar* s'appuient aujourd'hui sur une évaluation subjective du confort de conduite et sur la sécurité. Nous pouvons pondérer de 40% chacune des notes "sécurité" et "confort" dans le calcul de la note globale afin de fournir une évaluation plus objective du confort. Concernant les données brutes, nous afficherions alors les données relatives à la vitesse, l'accélération et le lacet

Les entreprises de location de véhicule (*Hertz*) et les entreprises d'assurances automobiles s'intéressent à la conduite de ses clients en terme de sécurité, afin d'adapter le prix de l'assurance. Dès lors l'accent serait mis sur la sécurité et l'évaluation du respect des limitations de vitesse avec une pondération de 50% de ces deux notes pour chaque trajet. Nous afficherions alors les données relatives à la vitesse, l'accélération et le lacet.

Hertz



Les services de livraison (*La Poste*, *UPS*), les convoyeurs de fond (*Brinks*) et les particuliers cherchant à réduire leur coût de conduite mettent l'accent sur l'écoconduite. Par conséquent, nous pondérerions la note globale par 80% d'écoconduite. Nous afficherions alors les données relatives à l'accélération, le régime moteur et la MAF en particulier.

III. Activités annexes

A. Formation en ligne ouverte à tous (*MOOCs*)

1. Arduino
2. Développez votre site web avec le framework Django¹⁶
 - a. *Présentation de Django*
 - **Créez des applications web avec Django**

Un *framework* (cadre de travail en français) est un ensemble d'outils qui simplifie le travail d'un développeur. Django est un *framework web* pour le langage Python très populaire, très utilisé par les entreprises dans le monde : Mozilla, Instagram ou encore la NASA l'ont adopté !



- **Le fonctionnement de Django**

Django respecte l'architecture MVT, directement inspirée du très populaire modèle MVC. Django gère de façon autonome la réception des requêtes et l'envoi des réponses au client (partie contrôleur). Un projet est divisé en plusieurs applications, ayant chacune un ensemble de vues, de modèles et de schémas d'URL. Si elles sont bien conçues, ces applications sont réutilisables dans d'autres projets, puisque chaque application est indépendante.

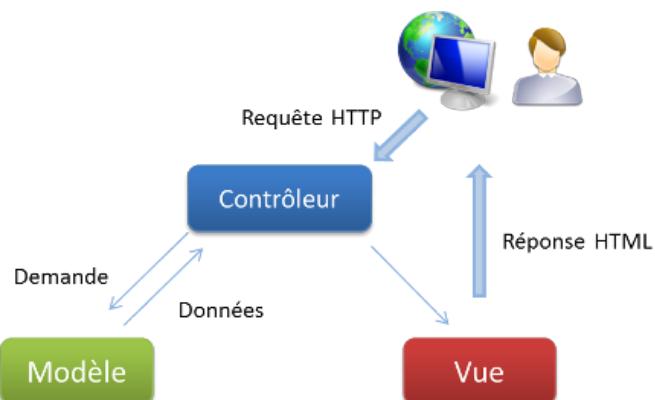


Figure 25 : Schéma de l'architecture Modèle-Vue-Contrôleur étudiée en Première Année dans le cadre du cours Génie Logiciel

¹⁶ OpenClassroom

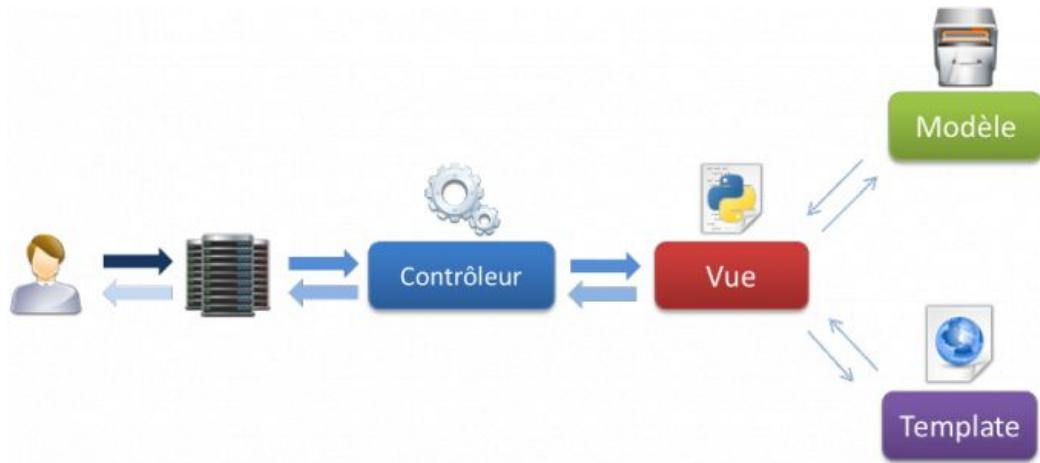


Figure 26 : Schéma de l'architecture Modèle-Vue-Template

- Gestion d'un projet

L'administration de projet s'effectue *via* le script `manage.py`. Tout particulièrement, la création d'un projet se fait grâce à la commande `django-admin.py startproject mon_projet`. À la création du projet, Django déploie un ensemble de fichiers, facilitant à la fois la structuration du projet et sa configuration. Il est nécessaire de modifier le `settings.py`, afin de configurer le projet selon nos besoins.

- Les bases de données et Django

Une base de données permet de stocker les données de façon organisée et de les récupérer en envoyant des requêtes au système de gestion de base de données (SGBD). De manière générale, nous communiquons la plupart du temps avec les bases de données via le langage **SQL**. Il existe plusieurs systèmes de gestion de bases de données, ayant chacun ses particularités. Pour faire face à ces différences, Django intègre une couche d'abstraction, afin de communiquer de façon uniforme et plus intuitive avec tous les systèmes : il s'agit de l'**ORM** (*object-relational mapping*). Une ligne dans une table peut être liée à une autre ligne d'une autre table *via* le principe de clés étrangères : nous gardons l'identifiant de la ligne de la seconde table dans une colonne de la ligne de la première table.



Figure 27 : Schéma de l'architecture de l'ORM Django

b. Développement du pattern

- Première page grâce aux vues

Le minimum requis pour obtenir une page web avec Django est une vue, associée à une URL. Une vue est une fonction placée dans le fichier views.py d'une application. Cette fonction doit toujours renvoyer un objet HttpResponseRedirect. Pour être accessible, une vue doit être liée à une ou plusieurs URL dans les fichiers urls.py du projet. Les URL sont désignées par des expressions régulières, permettant la gestion d'arguments qui peuvent être passés à la vue pour rendre l'affichage différent selon l'URL visitée.

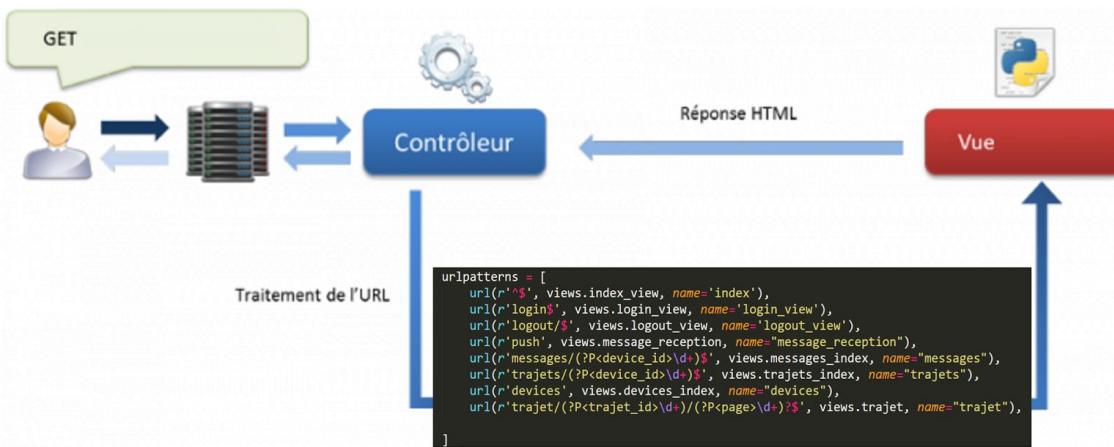


Figure 28 : Schéma d'exécution d'une requête (sans templates et sans modèles)

- Les templates

En pratique, et pour respecter l'architecture dictée par le framework Django, toute vue doit retourner un objet HttpResponseRedirect construit via un template. Pour respecter cette règle, il existe des fonctions nous facilitant le travail, comme *render*. Elle permet de construire la réponse HTML en fonction d'un fichier template et de variables. Les templates permettent également de faire plusieurs traitements, comme afficher une variable, la transformer, faire des conditions... Attention cependant, ces traitements ont pour unique but d'afficher les données, pas de les modifier. Il est possible de factoriser des blocs HTML (comme le début et la fin d'une page) via l'utilisation des tags `{% block %}` et `{% extends %}`. Afin de faciliter le développement, Django possède un tag `{% url %}` permettant la construction d'URL en lui fournissant la vue à appeler et ses éventuels paramètres. L'ajout de fichiers statiques dans notre template (images, CSS, JavaScript) peut se faire via l'utilisation du tag `{% static %}`.

- Les modèles

Un modèle représente une table dans la base de données et ses attributs correspondent aux champs de la table. Tout modèle Django hérite de la classe mère *Model* incluse dans django.db.models. Chaque attribut du modèle est typé et décrit le contenu du champ, en fonction de la classe utilisée : CharField, DateTextField, IntegerField... Il est possible d'afficher les attributs d'un objet dans un template de la même façon qu'en Python via des appels. Il est également possible d'itérer sur une liste d'objets.

- **L'administration**

L'administration est un outil optionnel : il est possible de ne pas l'utiliser. Une fois activée, de très nombreuses options sont automatisées, sans qu'il y ait besoin d'ajouter une seule ligne de code ! Ce module requiert l'usage de l'authentification, et la création d'un super-utilisateur afin d'en restreindre l'accès aux personnes de confiance. De base, l'administration permet la gestion complète des utilisateurs, de groupes et des droits de chacun, de façon très fine. L'administration d'un modèle créé dans une de nos applications est possible en l'enregistrant dans le module d'administration, via `admin.site.register(MonModele)` dans le fichier `admin.py` de l'application. Il est également possible de personnaliser cette interface pour chaque module, en précisant ce qu'il faut afficher dans les tableaux de listes, ce qui peut être édité, etc.

- **Les formulaires**

Un formulaire est décrit par une classe, héritant de `Django.forms.Form`, où chaque attribut est un champ du formulaire défini par le type des données attendues. Chaque classe de `Django.forms` permet d'affiner les données attendues : taille maximale du contenu du champ, champ obligatoire ou optionnel, valeur par défaut... Il est possible de récupérer un objet `Form` après la validation du formulaire et de vérifier si les données envoyées sont valides, via `form.is_valid()`.

B. Apprentissage Automatique

L'apprentissage automatique n'a pas été utilisé dans le cadre de notre projet. Cependant, dans l'idée d'un développement à grande échelle, il nous semble indispensable de profiter des avantages que peuvent apporter les algorithmes de classification supervisés pour évaluer la conduite à partir d'un grand nombre de paramètres. Ainsi, même si notre faible nombre de jeux de données est pour l'instant un facteur bloquant pour l'application de ces techniques d'apprentissage automatique, nous nous sommes intéressés à différents algorithmes théoriques, mais aussi à certaines implémentations qui pourraient être mises en place si notre base de donnée venait à s'étendre suffisamment.

1. Étude théorique

On se place dans un espace vectoriel dans lequel chaque coordonnée correspond à une valeur mesurée, et dans lequel chaque trajet nous donne donc un point. On cherche à trouver un moyen pour l'algorithme de différencier différentes classes, chaque classe correspondant à une note pour le trajet. Dans un cadre théorique, on s'intéresse d'abord à la discrimination de deux classes, la méthode pouvant être généralisée ensuite par des comparaisons successives.

a. Technique de l'Hyperplan Discriminant de Fisher

Pour ce premier algorithme, on cherche à déterminer un hyperplan qui sépare les deux classes de manière optimale. C'est ce qu'on appelle un algorithme de classification linéaire. Pour cela, on effectue la projection de tous les points sur une droite mettant en évidence la séparation des deux classes. Trouver l'hyperplan optimal revient alors à trouver un point sur cette droite. On commence donc par chercher une droite F passant par l'origine telle que la projection orthogonale des points sur F sépare au mieux les deux classes. Une fois déterminée, il faut trouver le point de cette droite par lequel passe l'hyperplan orthogonal à F discriminant de l'espace d'origine.

On introduit les notations suivantes :

Soient μ_1, m_1 et μ_2, m_2 les moyennes et cardinaux respectifs des points d'apprentissage des

classes ω_1 et ω_2 , et f un vecteur directeur de F .

Les moyennes des projections de points d'apprentissage sur F :

$$\begin{aligned} - \bar{\mu}_1 &= \frac{1}{m_1} \sum_{x \in \omega_1} f^T x \\ - \bar{\mu}_2 &= \frac{1}{m_2} \sum_{x \in \omega_2} f^T x \end{aligned}$$

La dispersion de ces projections :

$$\begin{aligned} - \bar{s}_1 &= \sum_{x \in \omega_1} (f^T x - \bar{\mu}_1)^2 \\ - \bar{s}_2 &= \sum_{x \in \omega_2} (f^T x - \bar{\mu}_2)^2 \end{aligned}$$

Le critère de Fisher correspond à choisir la droite F telle que la valeur

$$- J(F) = \frac{(\bar{\mu}_1 - \bar{\mu}_2)^2}{\bar{s}_1^2 + \bar{s}_2^2}$$

soit maximale, ce qui est le cas lorsque les projections des vecteurs d'une même classe sont aussi peu dispersés que possible, alors que la distance entre les moyennes des projections des vecteurs de chaque classe est aussi grande que possible. Afin de déterminer f , on montre que J s'exprime aussi comme étant :

$$- J(F) = \frac{f^T S_J f}{f^T S_I f}$$

où

$$\begin{aligned} - S_I &= \sum_{x \in \omega_1} (x - \mu_1)(x - \mu_1)^T + \sum_{x \in \omega_2} (x - \mu_2)(x - \mu_2)^T \\ - S_J &= (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \end{aligned}$$

On peut alors démontrer que ce critère est maximal pour :

$$- \hat{f} = S_I^{-1}(\mu_1 - \mu_2)$$

L'équation de l'hyperplan discriminant de Fisher est alors :

$$- \hat{f}x - f_0 = 0$$

Le problème est ainsi réduit à une unique dimension, il reste donc uniquement à déterminer f_0 , on est libre du choix de la méthode. Nous allons le faire ici par une version simplifiée de l'Analyse Discriminante Linéaire, en supposant que la répartition des données d'apprentissage de chaque classe suit une loi correspondant à un vecteur gaussien. On choisit alors f_0 tel qu'en ce point, la probabilité d'appartenir à chacune des classes soit la même. Cela correspond à :

- $p(f_0 \in \omega_1) = p(f_0 \in \omega_2)$
- Soit : $\frac{1}{2\pi s_1} e^{-\frac{(f_0 - \mu_1)^2}{2s_1^2}} = \frac{1}{2\pi s_2} e^{-\frac{(f_0 - \mu_2)^2}{2s_2^2}}$

En supposant de plus que les classes ont même variance, on obtient :

- $f_0 = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)}$

On a ainsi obtenu, au prix de certaines hypothèses importantes, notre classificateur linéaire, qui associe chaque vecteur x à une classe en fonction du signe de :

- $\hat{f}x - f_0$

b. Le perceptron monocouche

Le perceptron est le type de réseau de neurones le plus simple. Sera ici exposé uniquement le cas simplifié du perceptron monocouche.

On rappelle que l'on cherche à trouver un hyperplan $a^T x + a_0 > 0$ si $x \in \omega_1$, et $a^T x + a_0 < 0$ sinon.

Posons $X = (1, x_1, \dots, x_n)^T = (1, x)^T$ et $A = (a_0, a_1, \dots, a_n)^T = (a_0, a)^T$.

Alors $a^T x + a_0 = A^T X$.

Supposons que A représente un hyperplan séparateur de ω_1 et ω_2 . Alors le réseau suivant aura pour sortie 1 si x est dans ω_1 , 0 si x est dans ω_2 .

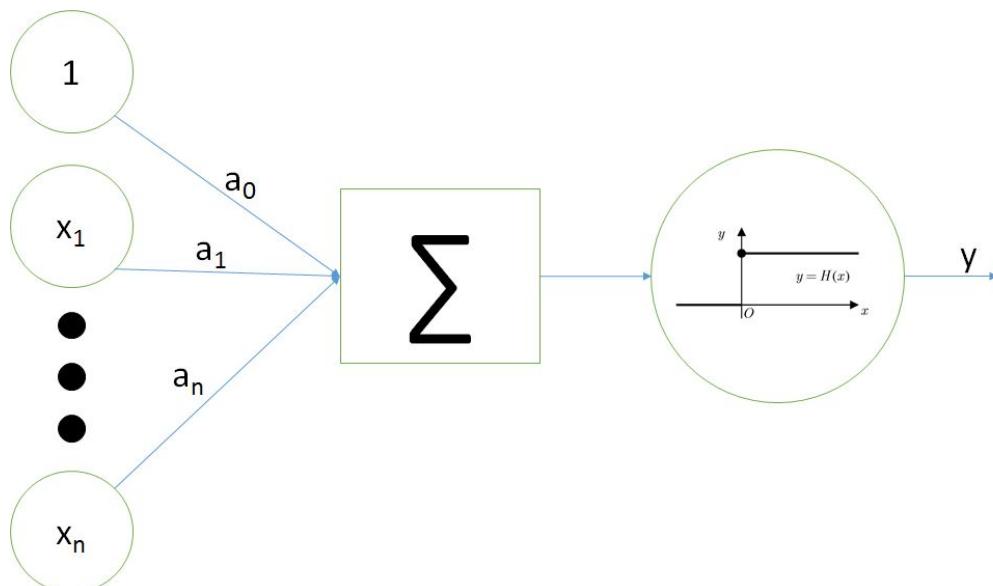


Figure 29 : Schéma du perceptron monocouche

Afin de déterminer les poids de A , on utilise cela un algorithme de correction d'erreur qui

existe sous deux formes : stochastique et non stochastique.

On remarquera au préalable que quelque soit le A trouvé, on peut le diviser par a_0 pour obtenir A' tel que $a'_0 = 1$, on peut donc fixer $a_0 = 1$. Le cas $a_0 = 0$ est alors approximé par un choix de valeurs très élevée pour les autres vecteurs.

L'algorithme stochastique est le suivant :

```

Données:  $t_{max}$ 
Résultat:  $a$ 
Choisir  $a_{(0)}$  et  $\alpha$  positif quelconques;
 $t \leftarrow 0;$ 
tant que  $t \leq t_{max}$  faire
    tirer au hasard une donnée d'apprentissage  $x$ ;
    si  $x$  est bien classé alors
        |  $a_{t+1} \leftarrow a_t$ 
    sinon
        | si  $x \in \omega_1$  alors
            | |  $a_{t+1} \leftarrow a_t + \alpha x$ 
        | sinon
            | |  $a_{t+1} \leftarrow a_t - \alpha x$ 
        | fin
    fin
     $t \leftarrow t + 1;$ 
fin
```

Algorithm 2: Algorithme du perceptron monocouche stochastique

Cet algorithme prend aléatoirement t_{max} éléments (avec remise) dans les données d'apprentissage, et corrige son résultat actuel.

En effet, si x est mal classé, et que $x \in \omega_1$, alors $a_t^T x$ est trop petit, on ajoute donc αx à a_t , on alors alors $a_{t+1}^T x = a_t^T x + \alpha |x|^2 \geq a_t^T x$, le résultat s'oriente donc vers le bon sens.

Il existe de plus un théorème, le théorème de Novikoff qui prouve la convergence de cet algorithme dans le cas séparable en un nombre fini d'étapes si α est bien choisi.

L'algorithme non stochastique est le suivant :

```

Données:  $t_{max}$ 
Résultat:  $a$ 
Choisir  $a_{(0)}$  et  $\alpha$  positif quelconques;
 $t \leftarrow 0$ ;
tant que  $t \leq t_{max}$  faire
   $modif \leftarrow 0$ ;
  pour chaque donnée d'apprentissage  $x$  faire
    si  $x$  est mal classé alors
      si  $x \in \omega_1$  alors
         $modif \leftarrow modif + \alpha x$ 
      fin
    sinon
       $modif \leftarrow modif - \alpha x$ 
    fin
  fin
   $t \leftarrow t + 1$ ;
   $a_{t+1} \leftarrow a_t + modif$ ;
fin

```

Algorithm 3: Algorithme du perceptron monocouche non stochastique

2. Etude pratique : Initiation au Machine Learning avec Scikitlearn¹⁷

Avant d'appliquer des modèles complexes à nos données, nous avons essayé de comprendre, dans quels contextes ils sont utilisables, ce à quoi ils correspondent et comment les choisir en fonction du problème que l'on rencontre.

a. Comment résoudre un problème de data science ?

Le cycle final de travail du *data scientist* que nous vennons d'esquisser se présente de la manière suivante :

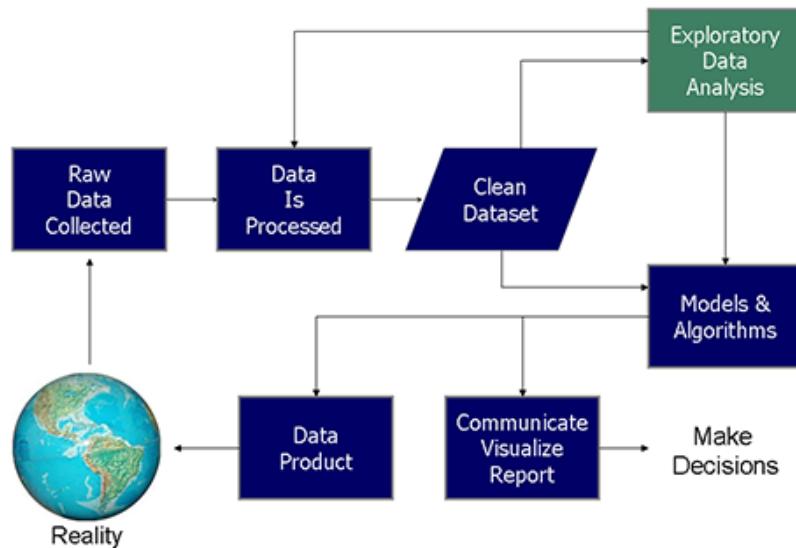


Figure 30 : Schéma du cycle final de travail du data scientist

Il comprend notamment : la récupération des données utiles à l'étude, le nettoyage des données pour les rendre exploitables, une longue phase d'exploration des données afin de comprendre en profondeur l'articulation des données, la modélisation des données, l'évaluation et

¹⁷ Openclassrooms

l'interprétation des résultats, la conclusion de l'étude : prise de décision ou déploiement en production du modèle.

b. Qu'est ce que le machine learning ?

Une fois que l'on a compris le contexte de travail d'un *data scientist* il est important de comprendre à quoi correspond son outil principal : l'apprentissage automatique.

Le *machine learning* est l'apprentissage d'un modèle statistique par la machine, grâce à des données d'entraînement. Un problème de machine learning comporte plusieurs éléments spécifiques : des données, une tâche à accomplir, un algorithme d'apprentissage, une mesure des performances.

c. Identifier les différents types de problème de machine learning

Les problèmes de *machine learning* peuvent se différencier selon deux critères :

- Les données dont vous disposez sont-elles annotées ou non ? Si c'est le cas, vous avez affaire à un problème d'**apprentissage supervisé**. Sinon vous serez obligé d'utiliser un algorithme d'**apprentissage non supervisé**.
- Quel est le type de résultat que vous souhaitez prédire ? S'il s'agit d'un nombre (par exemple le coût par clic d'une publicité) c'est un problème de **régression**. S'il s'agit plutôt d'une valeur discrète, d'une catégorie (par exemple le type d'animal présent sur une photo) alors c'est un problème de **classification**.

d. Application à notre projet

Malheureusement, l'application de techniques d'apprentissage automatique à notre projet n'est pas encore possible. En effet, une quantité d'au moins un millier de trajets serait nécessaire afin d'avoir des résultats satisfaisants. De plus, il s'agirait d'un problème de classification supervisée, il faudrait donc être capable de labelliser les trajets, c'est à dire d'attribuer une note de référence à chacun des mille trajets afin que notre modèle d'apprentissage puisse déduire la note d'un nouveau trajet. Nous nous sommes donc rendus compte de l'impossibilité d'appliquer ces techniques dans le cadre de ce projet sur une année.

IV. Perspective et ouverture

Notre projet long s'achève en remplissant nos objectifs imposés dès le départ. Nous voyons dores et déjà quelles seraient les étapes suivantes dans l'optique de création de startup.

D'abord, nous pouvons intégrer une fonction de suivi des véhicules en temps réel. Un tel système combine l'utilisation de la géolocalisation automatique des véhicules individuels avec une collecte des données de la flotte pour une image complète des emplacements des véhicules et de leur état. Les informations sur les véhicules pourraient être visualisées sur une carte électronique dans *l'EasyApp*. Les informations d'état des véhicules regrouperaient niveau de carburant, état et impacts physiques afin de détecter en temps réel pannes et accidents, qui seraient traités avec un système d'alertes.



Ce système d'alerte en temps réel est une deuxième perspective de notre projet : le gestionnaire de flotte automobile (ou propriétaire d'un véhicule en particulier) pourrait recevoir des alertes sur son adresse courriel ou sur son téléphone mobile directement notamment pour être tenu au courant en temps réel de pannes, accidents ou effractions. De même, un système de géorepérage permettrait de surveiller à distance la position et le déplacement d'un véhicule et de prendre des mesures si la position ou le déplacement s'écartait de certaines valeurs fixées d'avance (utilité pour les convoyeurs de fond, les sociétés de livraison, système antivol). Les véhicules modernes et de plus en plus de véhicules connectés vont être amenés à stocker des données relatives à l'entretien du véhicule : grâce à un système automatisé d'alertes, notre application pourrait informer les usagers ou les gestionnaires de flotte de la nécessité d'entretien d'un composant particulier du véhicule.

En terme économique, l'entreprise d'assurance connectée pourrait exploiter notre service très simplement en ajustant un barème de prix d'assurance en fonction du classement de ces conducteurs, basé sur leur note sécurité. Par ailleurs, grâce à la consommation de carburant, l'automobiliste pourrait chiffrer en fonction du temps le coût réel de sa conduite. Enfin, en optimisant les trajets et en faisant de la prédition, les entreprises de livraison réduiraient leurs coûts.

Dans la perspective où notre application serait déployée sur une échelle importante et si un nombre très important de voitures étaient équipées de notre *EasySensor*, ceux-ci pourraient communiquer entre eux de façon à obtenir par exemple la vitesse ou la distance relative entre deux véhicules ce qui nous intéresserait pour évaluer une conduite agressive.



Conclusion

Nous sommes tous les quatre extrêmement fiers de ce que nous avons accompli : en un an, nous avons su maîtriser des technologies que nous ignorions jusqu'alors. Nous avons réussi à créer sur deux fronts d'une part une centrale *EasySensor* de type **Plug and Play** et d'autre part nous avons conçu entièrement un site Web fonctionnel, l'*EasyApp*, capable d'analyser les données issues du capteur. Le travail accompli représente des milliers de lignes de code en Arduino, Python, JavaScript, jQuery, HTML et évidemment rien de tout cela n'aurait été possible sans passion pour les domaines étudiés et pour le projet global. Ainsi, nous avons exprimé un fort intérêt pour ce projet pluridisciplinaire axé sur des technologies d'avenir (*Internet of Things*, traitement de données). Évidemment l'apprentissage technique est très valorisant, complémentaire de la formation d'excellence généraliste d'Ingénieur Supélec. Cette année nous a permis également d'améliorer nos *softs skills* : notamment en gestion de projet (*plannings respectés*), *management* et communication (réunions régulières de suivi de l'avancement du projet et de répartition des tâches).

Les difficultés rencontrées ne nous ont pas empêché de progresser dans la réalisation du projet. Au contraire, nous avons su chercher des solutions là où des problèmes nous bloquaient. Par exemple, nous avons surpasser les problèmes de réception non chronologique des données, l'évaluation subjective de conduite, la gestion lourde des messages incomplets ou erronées à cause de la transmission *via* le réseau LoRa. Enfin nous avons appris à nous confronter à des langages de programmation et des méthodes de développement nouveaux.

Remerciements

Aux enseignants-chercheurs de CentraleSupélec

- Jocelyn Fiorina
- Jacques Antoine
- Anthony Kolar
- Raul De Lacerda

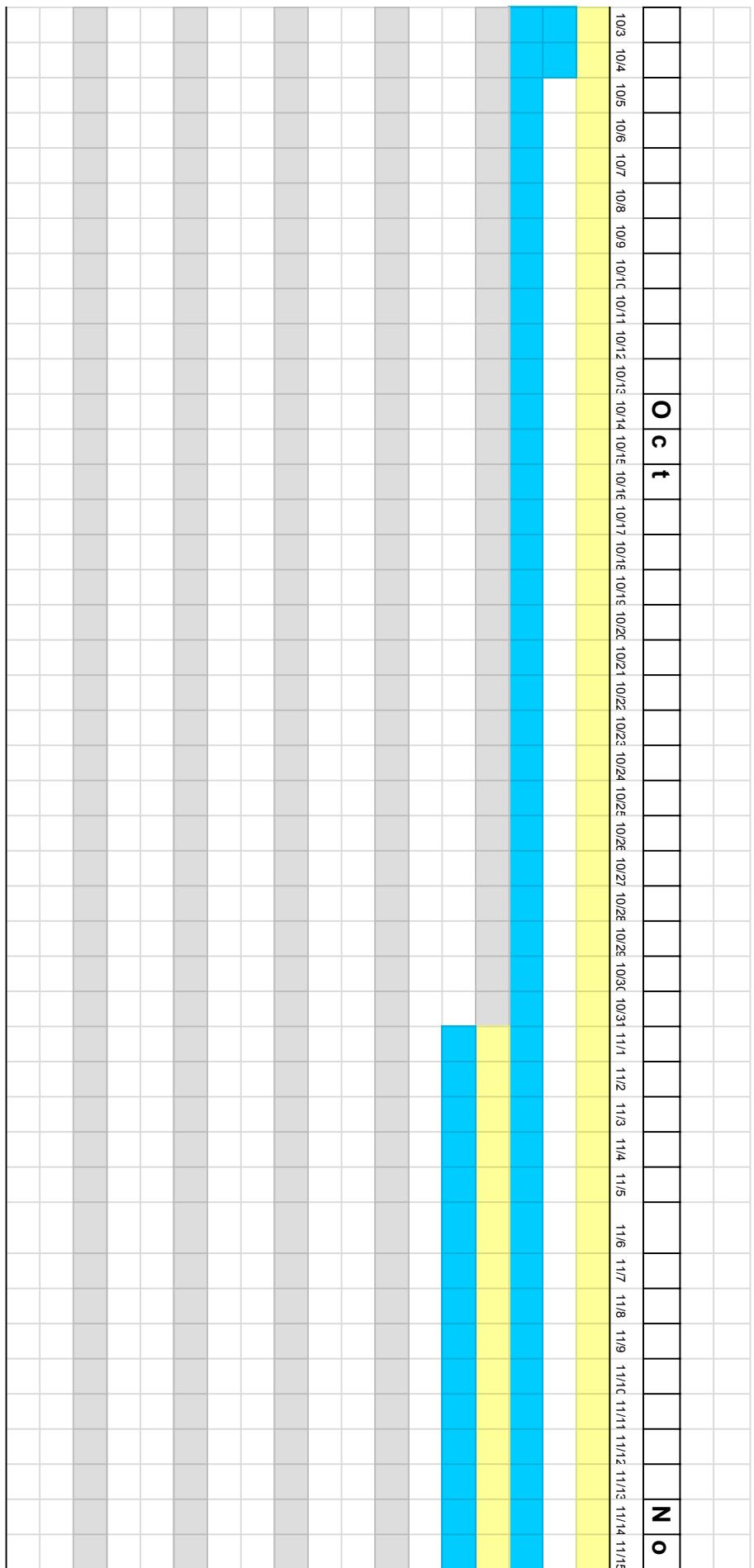
À Objenious

- Fanny Brun
- Charlie Remy

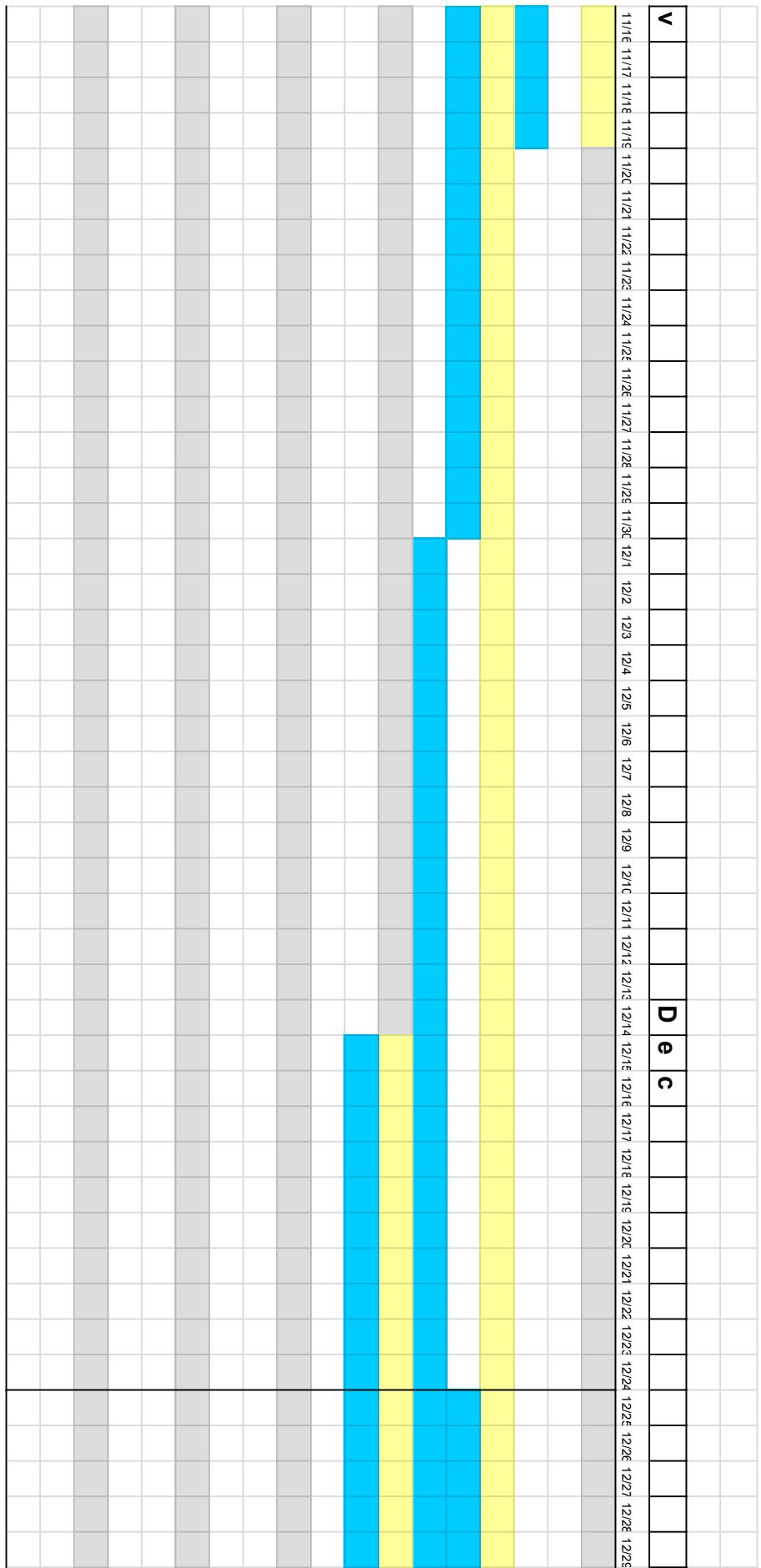
Annexe 1 : Diagramme de Gantt du projet - Sheet1

Easydrive							Start Date: September 15, 2016												
CentraleSupélec																			
Tache	Début	Fin	Durée	Avancement	e	p	Date: 9/15 9/16 9/17 9/18 9/19 9/20 9/21 9/22 9/23 9/24 9/25 9/26 9/27 9/28 9/29 9/30 10/1 10/2												
1.0 Récupération des données sur l'Arduino	2016-09-15	2016-11-19	66	100.00%															
1.1 Vitesse	2016-09-15	2016-10-04	20	100.00%															
1.3 Coordonnées GPS	2016-10-01	2016-11-19	50	100.00%															
2.0 Envoi des données avec Lora	2016-11-01	2017-02-28	120	100.00%															
2.1 Premiers tests	2016-11-01	2016-11-30	30	100.00%															
2.2 Choix du format	2016-12-01	2017-02-28	90	100.00%															
3.0 Serveur Django	2016-12-15	2017-03-21	97	100.00%															
3.1 Reception des données	2016-12-15	2017-02-12	60	100.00%															
3.2 Automatisation du traitement	2017-02-20	2017-03-21	30	100.00%															
4.0 Traitement des données	2017-01-05	2017-03-25	80	100.00%															
4.1 Vitesse	2017-01-05	2017-02-03	30	100.00%															
4.2 Coordonnées GPS (virages) et limitations	2017-02-24	2017-03-25	30	100.00%															
5.0 Interface utilisateurs	2017-03-15	2017-05-23	70	100.00%															
5.1 Choix des données à afficher	2017-03-15	2017-03-29	15	100.00%															
5.2 Implémentation en Django	2017-03-25	2017-05-23	60	100.00%															
6.0 Déploiement	2017-05-15	2017-06-03	20	100.00%															
6.1 Tests et vérification du cahier des charges	2017-05-15	2017-06-03	20	100.00%															
6.2 Préparation de la soutenance finale	2017-05-20	2017-06-02	14	100.00%															

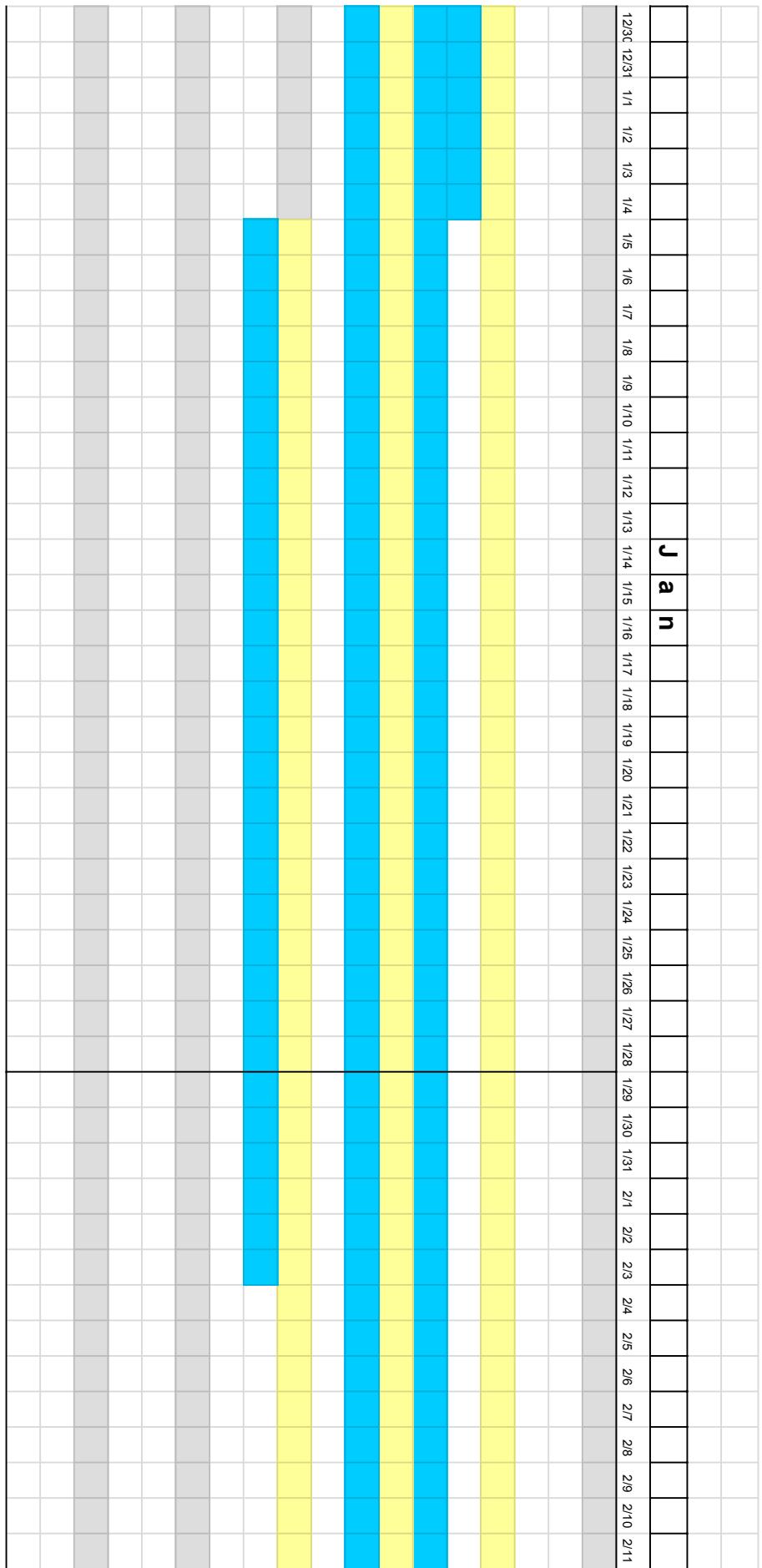
Annexe 1 : Diagramme de Gantt du projet - Sheet1



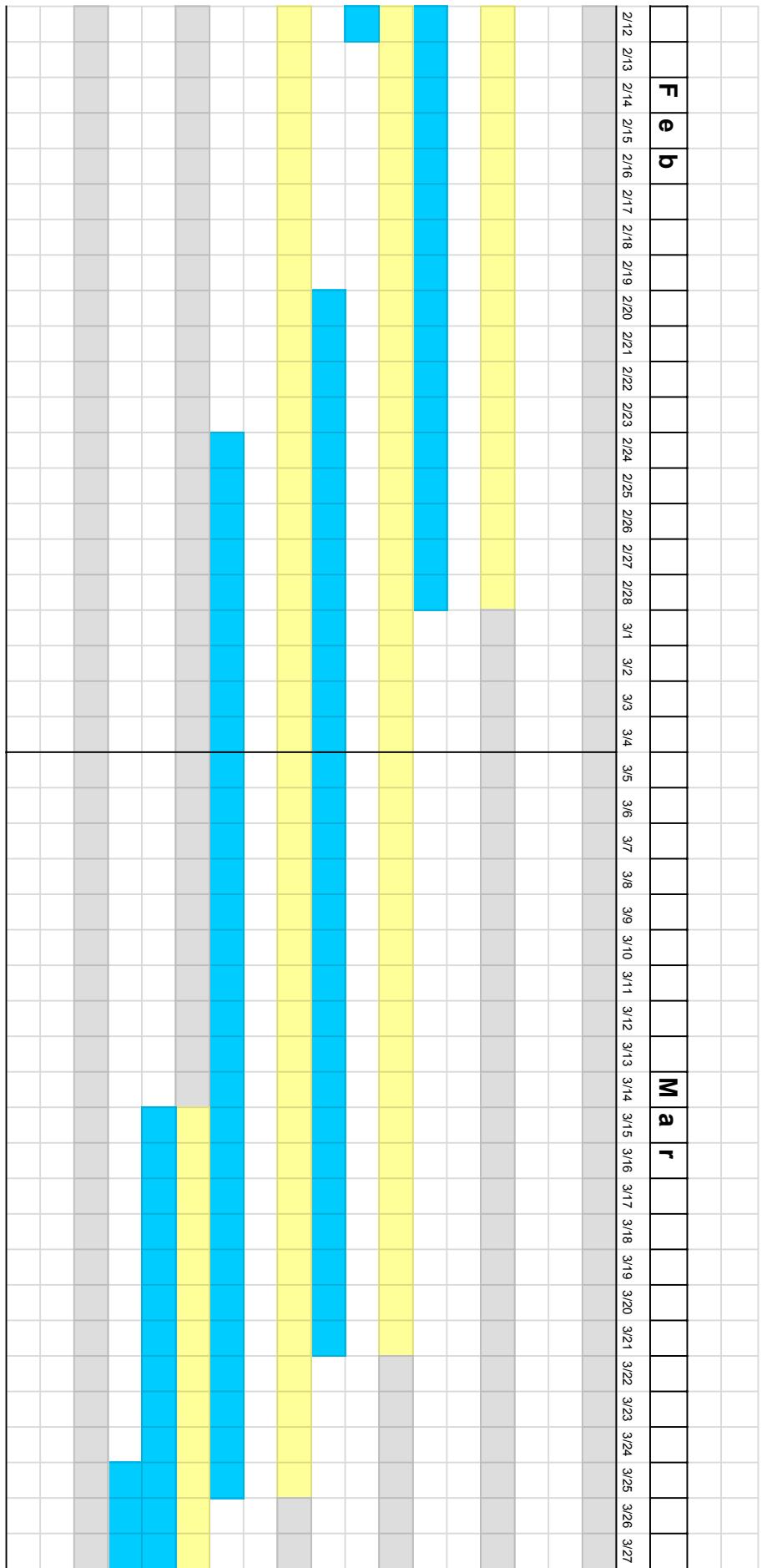
Annexe 1 : Diagramme de Gantt du projet - Sheet1



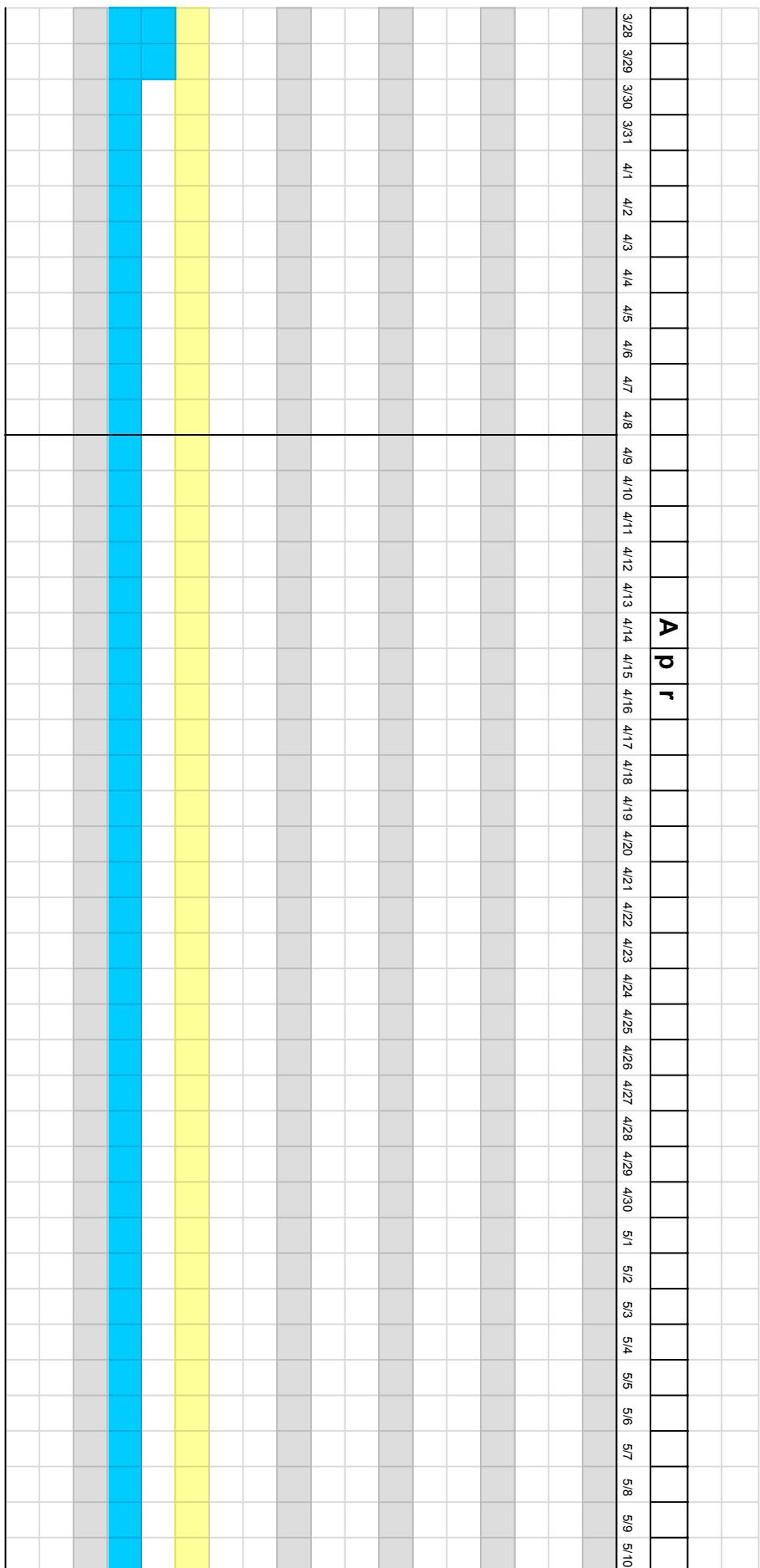
Annexe 1 : Diagramme de Gantt du projet - Sheet1



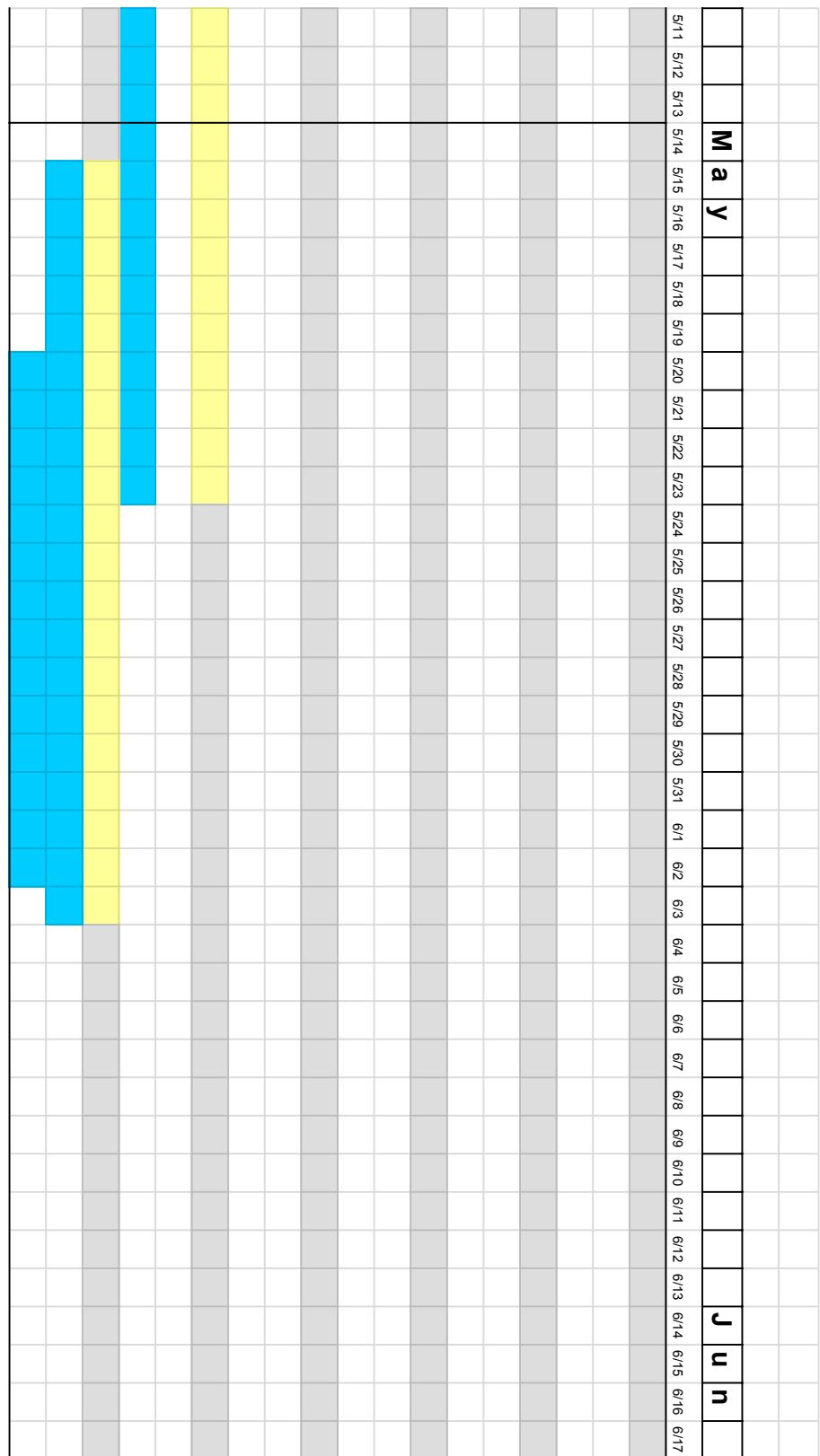
Annexe 1 : Diagramme de Gantt du projet - Sheet1



Annexe 1 : Diagramme de Gantt du projet - Sheet1



Annexe 1 : Diagramme de Gantt du projet - Sheet1



Pour EasySensor, nous avons développé une bibliothèque, OBD2, comme couche d'abstraction des communications avec l'interface OBD2. Elle est réalisée en C++, et définit plusieurs constantes pour les codes de commande, ainsi que deux fonctions de conversion et une classe OBD2, qui est l'abstraction de la connexion et correspond à l'interface à utiliser dans notre programme principal.

Le programme principal, quand à lui, contient les fonctions setup et loop, pour le fonctionnement de la carte Arduino (setup est appelée à l'allumage de la carte, puis loop est appelée en boucle tant que la carte reste allumée), ainsi que la fonction module, qui calcule le carré du module d'un couple d'entier, utilisé pour l'accélération.

La bibliothèque OBD2 contient donc :

Toutes les constantes correspondant à chaque PID.

Les fonctions de conversion, dont les signatures sont :

```
uint8_t hex2uint8(const char *p)  
uint16_t hex2uint16(const char *p)
```

Ces fonctions convertissent donc une ou deux paire de caractères hexadécimaux respectivement en entier non signé sur 8 ou 16 bits.

La classe OBD2, donc les méthodes accessibles sont :

```
OBD2(long baudrate = 38400);
```

Constructeur de la classe

```
bool init();
```

Méthode initialisant la connexion avec la prise OBD2

```
bool send_cmd(const char* cmd);
```

Méthode lançant la commande reçue en paramètre, sous la forme d'une chaîne de caractère

```
void send_pid(int pid);
```

Méthode lançant la commande reçue en paramètre, sous la forme d'un entier correspondant au PID. On peut utiliser les constantes définies par la bibliothèque

```
int receive(char *result);
```

Méthode qui lit la dernière réponse de la prise, l'écrit dans le tableau result, et renvoie le nombre de caractères écrits.

```
int receive_all(char *result);
```

Méthode qui récupère toutes les données envoyées par la prise, l'écrit dans le tableau result, et renvoie le nombre de caractères écrits. Permet de vider le buffer d'entrée.

`int normalize_data(int pid, char *data);`

Décode la réponse de la prise OBD2 et renvoie résultat sous la forme d'un entier

`void read_cmd_brute(const char* cmd, char *result);`

Lance une commande sous la forme d'une chaîne de caractères, et récupère le résultat.

`bool read_pid(int pid, int *result);`

Lance une commande sous la forme d'un entier correspondant au PID demandé, récupère le résultat et le décode.

`void recover();`

Réinitialise la prise OBD2, en cas d'erreur.

`void read_DTC(char *result);`

Récupère d'éventuels code d'erreur de la prise.

`void clear_DTC();`

Vide les codes d'erreur de la prise.

`void create_pid_map();`

Récupère la liste des PID supportés par l'ordinateur de bord, et l'enregistre dans l'attribut m_pid_map, codé bit à bit.

`bool valid_pid(int pid);`

Vérifie que le PID est supporté par l'ordinateur de bord. Il est nécessaire que la méthode create_pid_map ait été appelée préalablement.

Annexe 3 : arborescence de l'EasyApp

