



# Dossier technique

Projet IRIST 2015 : METEOFOLK

---

## Sommaire

---

<b>1) Partie Commune</b>	<b>6</b>
1.1) Présentation	
1.1.1) Objectifs du projet	
1.1.2) Enjeux	
1.1.3) Architecture	
1.1.4) Le réseau SIGFOX	
1.1.5) Anémomètre/Girouette à ultrasons	
1.2) Contexte historique	
1.3) Spécification générale	
1.3.1) Diagrammes de cas d'utilisation	
1.4) Description de l'architecture	
1.4.1) Diagramme de classes	
1.4.2) Diagramme de déploiement	
1.5) Répartition du travail par étudiant	
<b>2) E1 :JAUME Benjamin - Acquisition/Traitement des données</b>	<b>17</b>
2.1) Présentation de la partie personnelle	
2.2) Mise en œuvre	
2.2.1) Technologies utilisées	
2.2.2) Mise en œuvre des techniques	
2.3) Conception détaillée	
2.3.1) Diagramme de séquence	
2.3.2) Diagramme de classes	
2.4) Implémentation	
2.4.1) Présentation	
2.4.2) Extraits de code	
2.5) Tests de validation	
2.5.1) Type de test	
2.6) Conformité aux missions	
2.7) Bilan personnel technique	
<b>3) E2 : NICOLAS Kévin - Acquisition/Traitement des données</b>	<b>33</b>
3.1) Description de la partie personnelle	

3.1.1) Travail à effectuer au sein du système

3.2) Mise en œuvre

3.2.1) Techniques / Technologies utilisées

3.3) Implémentation

3.3.1) Présentation

3.3.2) Les extraits de code

3.4) Bilan Personnel technique

#### 4) E3 : CAYUELA Julien - Collecte des données météorologiques

50

4.1) Description de la partie personnelle

4.2) Mise en œuvre

4.2.1) PHP

4.2.2) JSON

4.2.3) WAMPServer

4.2.4) Symfony2

4.3) Conception détaillée

4.3.1) Diagramme de cas d'utilisation (spécification)

4.3.2) Diagramme de classes d'entités

4.4) Implémentation

4.4.1) Création des entités

4.4.2) Enregistrement de l'application

4.4.3) Test : Réception callbacks

4.4.4) Réception et injection des données dans la Base de données

4.4.5) Agrégation des échantillons

4.5) Tests de validation

4.6) Bilan Personnel technique

#### 5) E4 : Baptiste DESCARD - Gestion du serveur METEOFOX

70

5.1) Description de la partie personnelles

5.2) Conception détaillée

5.3) Mise en œuvre

5.3.1) IHM

5.3.2) Symfony

5.4) FOSUserBundle

5.4.1) Gérer les balises

5.4.2) Mise en forme du site

5.5) Implémentation

Descriptif (sous système, ...)

Auteur(s) : E2

Page 3 / 130

<p>5.5.1) Tutoriel:</p> <p>5.6) Bilan personnel technique</p>	
<b>6) Tinet Ludovic - Historisation des balises</b>	<b>87</b>
6.1) Description partie personnelle	
6.1.1) Me situer	
6.1.2) Mes tâche	
6.2) Mis en œuvre	
6.2.1) Tutoriel Symfony2	
6.2.2) Fonctionnement Symfony	
6.3) Highcharts	
6.4) Bilan	
<b>7) RADULOV Lachezar : Affichage des balises en mode cartographique</b>	<b>92</b>
7.1) Description partie personnelle	
7.2) Mise en œuvre	
7.3) Conception détaillée	
7.3.1) Diagramme de séquence	
7.4) Implémentation	
7.4.1) Création d'une carte :	
7.4.2) Ajour de marqueur avec infobulles	
7.4.3) Modifier la taille de la carte ainsi, personnaliser le marqueur, centrer la carte	
7.4.4) Utiliser les entités sur la carte	
7.4.5) Afficher une liste des balises	
7.5) Bilan personnel technique	
<b>8) Étudiant 7</b>	<b>107</b>
← nom ?	
<b>9) Étudiant 8 : Audoy François</b>	<b>108</b>
9.1) Description de ma Partie	
9.1.1) Cas d'utilisation	
9.1.2) Principe général	
9.2) Mise en œuvre	
9.2.1) Technique /Technologies utilisées	
9.2.2) Mise en œuvre	
9.3) Conception détaillée	
9.3.1) Diagramme de classe	

**9.4) Implémentation**

- 9.4.1) Détails de l'application**
- 9.4.2) AccueilActivity**
- 9.4.3) GestOnglet**
- 9.4.4) MapsActivity**

**9.5) Test de Validation****9.6) Déploiement****9.7) Conclusion**

<b>10) Étudiant 9 : Rémi Amand</b>	<b>129</b>
<b>11) Conclusion</b>	<b>130</b>

---

Descriptif (sous système, ...) <b>???</b>
---

Auteur(s) : <b>E2</b> nom ?
-----------------------------

Page 5 / 130
--------------

## 1) Partie Commune

### 1.1) Présentation

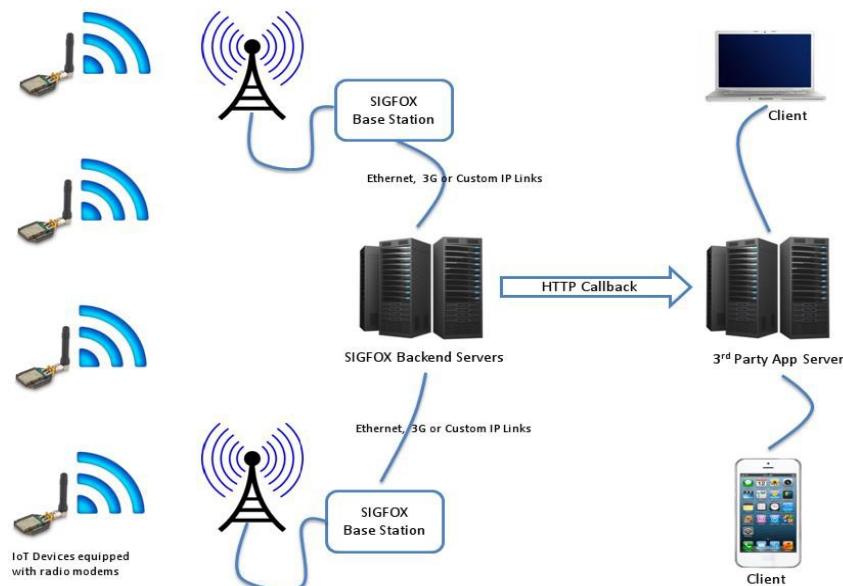
#### 1.1.1) Objectifs du projet

Nous proposons de développer un réseau de capteurs de vent / balises météo en France, basés sur la technologie SIGFOX, à destination des BTS SN.

Notre projet servira plus tard à :

- D'autres élèves du BTS SNIR (réforme du BTS IRIST) pour :
  - ✓ Réaliser des CCF (contrôles en cours de formation) pour l'épreuve **quelle épreuve ?**
  - ✓ En faire des éléments constitutifs comme supports de TP (travaux pratiques)
- Mettre à la disposition de tout le monde des informations météorologiques (température, vitesse du vent, direction du vent, pression atmosphérique) à l'aide d'un réseau de balise sans fil.

Le dispositif devra proposer une alternative plus économique et plus efficiente aux balises mise en place par la FFVL, les balises FFVL étant d'une technologie plus ancienne.



#### 1.1.2) Enjeux

Permettre aux étudiants qui poursuivront le BTS au lycée d'avoir un accès aux ressources de notre projet pour une exploitation pédagogique.

Développement d'un réseau de capteurs de vent / balises météo basés sur la technologie SIGFOX
Auteurs: AUDOY-BEYRAND-CAYUELA-DESCARD-JAUME-NICOLAS-RADULOV-TINET-AMAND

### 1.1.3) Architecture

Le système proposé serait composé des éléments suivants :

- La **balise METEOFOX** : elle a pour mission de mesurer les données météorologiques

(vitesse et direction du vent, température, pression atmosphérique) et de la transmettre périodiquement.

Elle comporte un anémomètre à ultrasons et un capteur barométrique.

Elle transmet périodiquement ses mesures via le réseau SIGFOX (c.f : 1.1.4) Réseau SIGFOX )

Elle est autonome (alimentation par panneau solaire) et peut être placée sur tout site en extérieur.

- Le **serveur METOFOX**: il comporte :

- un **service de collecte et d'archivage** : il a pour mission de recevoir et d'archiver les données transmises par les balises.
  - les données sont transmises par le réseau SIGFOX sous la forme de requêtes HTTP permettant de 'pousser' les données (HTTP callbacks).
  - les données sont archivées ('historiées') et consolidées afin de faciliter l'exploitation des données historiques sur de longues périodes.

les applications utilisatrices accèdent aux données du serveur de collecte via des webservices.

- un **service de publication Web** : sa mission est de permettre aux usagers la consultation des données météo via un navigateur web standard. On doit pouvoir consulter :
  - les données en temps réel
  - des statistiques sur une période donnée (min, max, moyenne, ...)
  - des recherches historiques

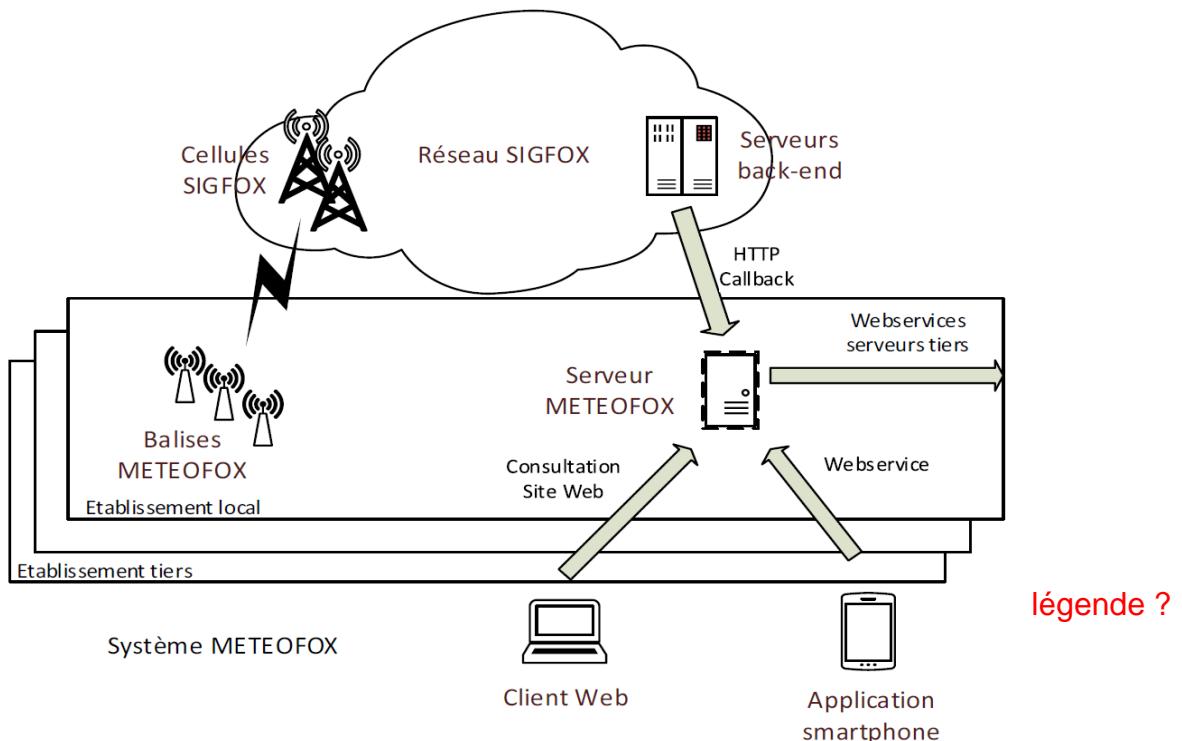
Un service Web M2M (machine to machine)

- il permet aux applications utilisatrices (application smartphone ou sites web tiers) d'avoir accès aux données météo.

L'interface de navigation Web devra permettre une navigation ergonomique parmi les diverses balises ou sources de données. Les balises étant géo localisées, une présentation cartographique semble pertinente. possible de consulter **l'ensemble des données des établissements participants au dispositif**.

- L'**application smartphone** : elle a pour mission de permettre à l'usager la consultation des données via un smartphone. La conception d'une application native doit permettre d'optimiser les flux réseau et d'améliorer la réactivité par rapport à la consultation du site Web. Elle doit aussi permettre à l'utilisateur de définir ses préférences.

Descriptif (sous système, ...)	???
Auteur(s) : <del>E2</del>	Page 7 / 130



#### 1.1.4) Le réseau SIGFOX



Le réseau SIGFOX est un réseau cellulaire dédié à l'Internet des objets. Il est issu d'une start-up toulousaine et connaît actuellement un développement extrêmement rapide (cf article ci-contre).

La transmission de données se fait par radio et utilise une modulation en bande ultra-étroite (*UNB = ultra-narrow band*).

Cette technologie permet une très longue portée (plus de 40Km) et la consommation d'une puce SIGFOX est 1000 fois plus faible quelle d'une puce GSM.

La portée des cellules SIGFOX permet ainsi de couvrir plus de 90% du territoire Français avec seulement 1000 antennes, et elle permet d'atteindre des objets enfouis à des endroits non accessibles par les ondes GSM.

Le coût d'abonnement est aussi très faible. Pour des flottes d'objets importantes, il devrait revenir à moins



d'un euro par an et par objet connecté. Pour des applications plus restreintes, le coût unitaire est plus élevé, mais reste beaucoup plus faible que celui d'un mobile GSM.

Mais ce réseau n'est pas destiné aux mêmes applications que le GSM. En effet, il ne permet que de très faibles débits de données. Ainsi, une puce SIGFOX peut émettre 150 messages par jour, chaque message peut contenir 12 octets maximum.

### 1.1.5) [Anémomètre/Girouette à ultrasons](#)

La plupart des anémomètres et girouettes traditionnels sont basés sur des éléments mécaniques tournants. Cela comporte nombre d'inconvénients (frottements, usure). Ces capteurs devant être placé en extérieur, ils sont soumis aux agressions climatiques, ce qui limite la longévité de ces capteurs, ou impose une maintenance régulière.

Les anémomètres à ultrason exploitent la mesure du temps de propagation du son entre 2 couples d'émetteurs et récepteurs d'ultrasons disposés orthogonalement.

Le vent modifiant le temps de propagation, il est ainsi possible de calculer la vitesse et l'orientation du vent sans aucune pièce mobile.

Du fait de leur grande résistance aux agressions climatiques, ces capteurs sont de plus en plus préférés aux anémomètres et girouettes rotatifs traditionnels.

Une étude de ce type de capteur a d'ailleurs été inclue dans le sujet d'informatique du BTS IRIST (Pont de Millau).



[légende ?](#)

## 1.2) [Contexte historique](#)

De nombreuses activités requièrent la mesure et l'historisation de données météorologiques. Les informations sur le vent en particulier sont utiles pour l'aviation, le vol libre (deltaplane, parapente, kite-surf), la voile notamment.

Ce besoin a conduit la Fédération Française de Vol Libre (FFVL) à créer un réseau de balises de mesure de vent permettant d'obtenir en temps-réel les mesures du vent sur les sites où sont implantées des balises de mesure et de télétransmission.

D'autres réseaux de balises existent : balises météo de météofrance, réseau viewsurf (équipé de webcams) etc...

**RÉSEAU FÉDÉRAL DE BALISES**

Sélectionnez une région :

**Accès direct**  
Saisissez directement une région, un département ou une balise :  GO

**Hors France**

- Guyane
- Nouvelle Calédonie
- Réunion

**Liens utiles**

- FFVL
- Ligue Rhône-Alpes
- Météo France
- Météo Monde

**News**

- Application Android
- Application iPhone

### légende ?

Les capteurs de vent et de données météorologiques doivent souvent être implantés sur des sites où l'on ne dispose pas d'alimentation électrique, et où l'accès à Internet ou à un réseau téléphonique filaire est rarement disponible.

Ainsi, les balises de la FFVL utilisent une transmission de données par GSM. Les données sont centralisées sur un site de collecte, et accessibles au public via une IHM Web ou des applications spécialisées sur smartphones.

Mais la transmission par GSM possède des inconvénients :

- il est nécessaire de disposer d'un abonnement téléphonique pour chaque balise (-> coût)
- la couverture GSM n'est pas complète, notamment dans les régions montagneuses.
- les modems GSM ont une consommation d'énergie non négligeable.

### 1.3) Spécification générale

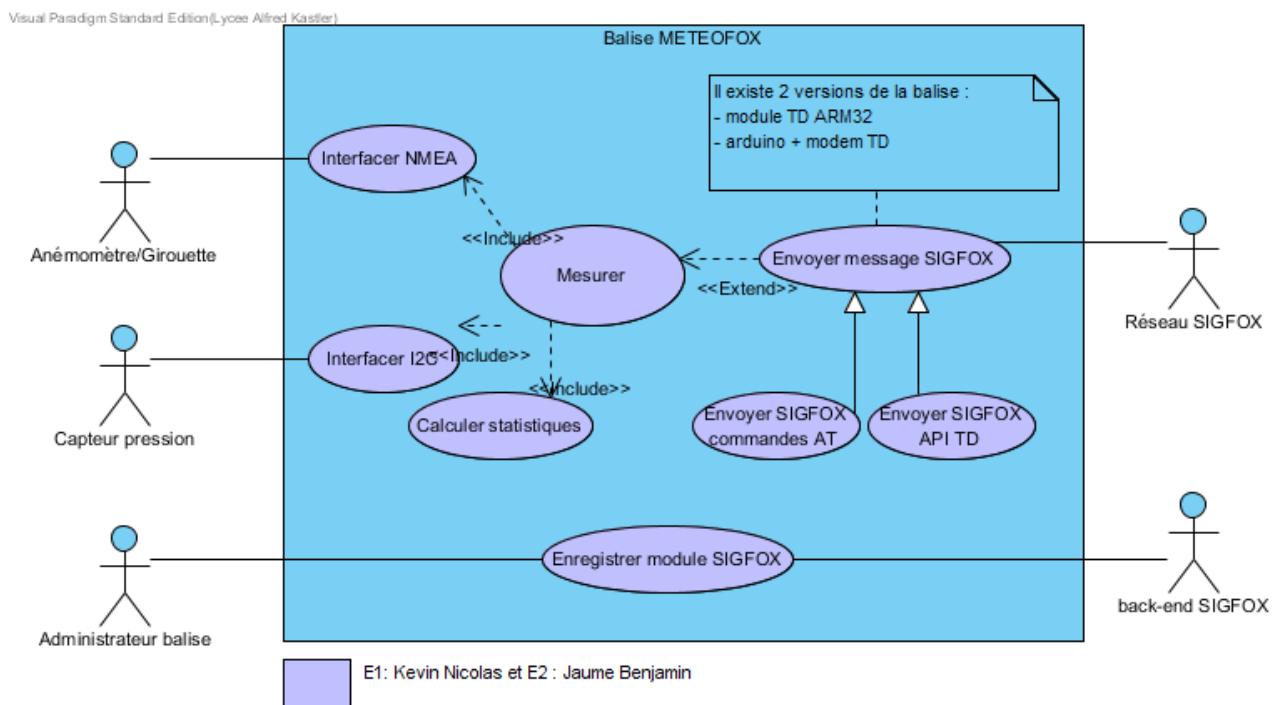
#### 1.3.1) Diagrammes de cas d'utilisation

##### 1) Cas d'utilisation : balise météorologique meteofox

La balise a deux versions :

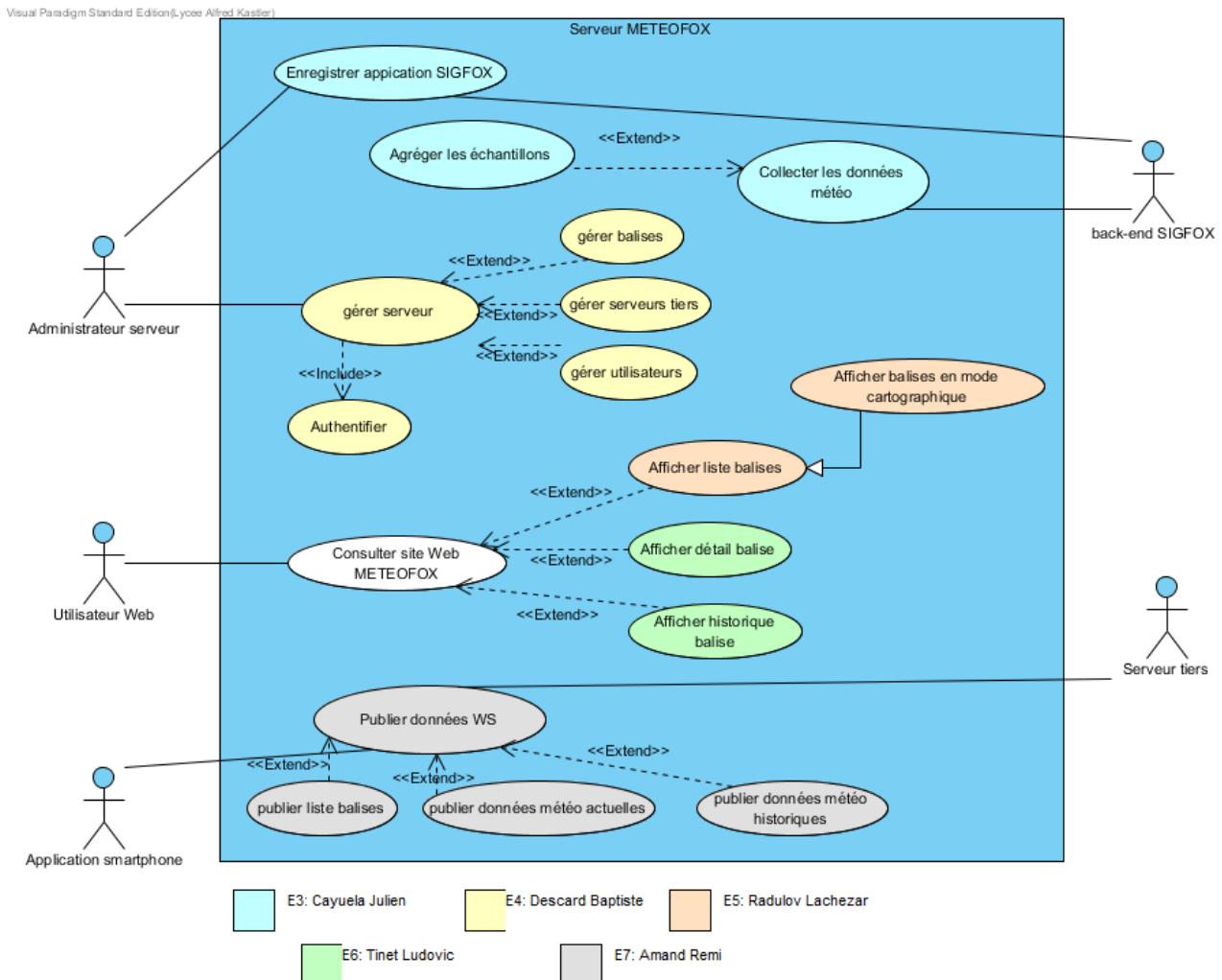
- une version à base d'Arduino. La transmission sur le réseau SIGFOX est effectuée par un modem SIGFOX. Les puces Telecom Design TD120x sont fournies avec un firmware standard qui implémente les fonctionnalités d'un modem utilisant un jeu de commandes AT sur liaison série.
- une version directement implantée sur une puce Telecom Design TD120x. Les puces TD comportent un microcontrôleur ARM32 et un kit de développement est disponible.

Ce kit de développement, basé sur Eclipse/GCC, comporte une API spécifique Telecom Design.



Cas d'utilisation	Description	Contraintes
Mesurer	Réaliser les mesures de : 1 vitesse et direction du vent 2 température 3 pression atmosphérique Les mesures sont effectuées périodiquement. Des statistiques sont calculées et envoyées périodiquement via SIGFOX.	
Interfacer I2C	Communiquer par I2C avec le capteur de pression.	Librairie Arduino disponible. Portage sur ARM32 possible a priori.
Interfacer NMEA	Analyser la trame NMEA en provenance de l'anémomètre-girovette à ultrasons.	
Calculer statistiques	Sur un période donnée, calculer : 4 min, max, moyenne	
Envoyer message SIGFOX	Envoyer les données météo vers le réseau SIGFOX	Taille de message limitée à 12 octets Fréquence env. 150 msg/jour maxi
Enregistrer module SIGFOX	Enregistrer le module SIGFOX sur le réseau SIGFOX	Créer un compte développeur Telecom Design. Procédure d'enregistrement des modules en ligne.

## 2) Cas d'utilisation : Serveur METEOFOX



Cas d'utilisation	Description	Contraintes
<b>Enregistrer application SIGFOX</b>	Enregistrer le serveur METEOFOX sur le back-end SIGFOX, et l'associer aux modules SIGFOX des balises, afin de recevoir les callbacks déclenchés par les messages SIGFOX	Créer un compte développeur Telecom Design. Procédure d'enregistrement des modules en ligne.
<b>Collecter les données météo</b>	Recevoir les messages envoyés par les balises METEOFOX et enregistrer les données dans la base de données	Webservice HTTP/JSON

Descriptif (sous système, ...)
--------------------------------

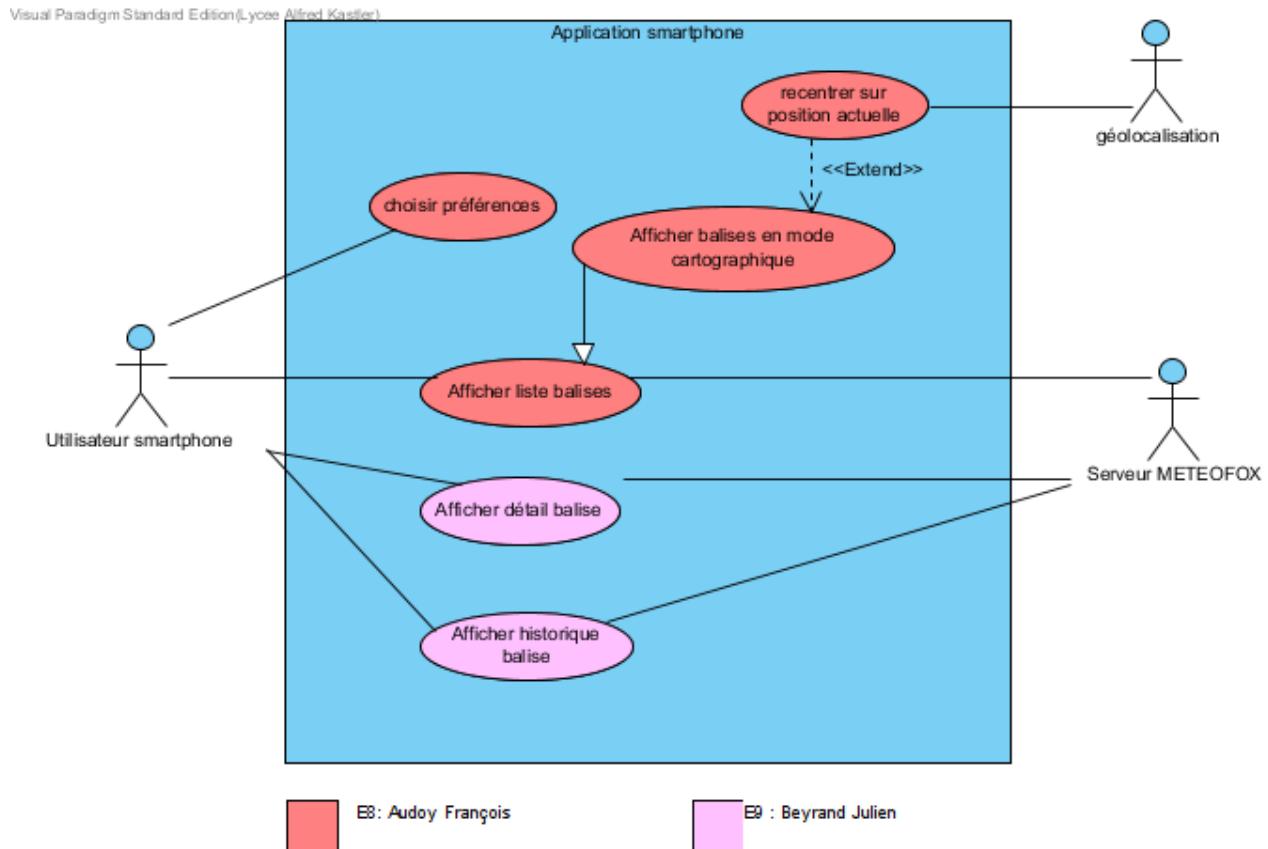
Auteur(s) : E2
----------------

Page 13 / 130
---------------

Agréger les échantillons	Afin de faciliter la conservation et l'exploitation des données sur de longues périodes, on agrège plusieurs échantillon pour former un échantillon synthétique sur la période correspondante.	<ul style="list-style-type: none"> <li>- calculer les valeurs statistiques (min, max, moyenne)</li> <li>- gérer correctement le cas des échantillons manquants</li> <li>- purger les échantillons anciens inutiles</li> </ul>
Gérer serveur	détaillé dans les 3 cas d'extension suivant	Interface Web – réservée à l'administrateur.
Gérer balises	Ajouter, éditer, supprimer des balises	
Gérer serveurs tiers	Ajouter, éditer, supprimer des serveurs tiers	
Gérer utilisateurs	Ajouter, éditer, supprimes des comptes utilisateurs	Bundle de gestion des utilisateurs de SYMFONY
Authentifier	Authentification l'utilisateur	login/mot de passe
Consulter site web METEOFOLY	Site web de consultation des données météo	<p>l'interface doit être adaptée à divers types de terminaux (PC, tablettes).</p> <p>Utilisation de la bibliothèque BOOTSTRAP</p>
Afficher liste et état des balises	Une liste des balises est affichée, avec leur état et un résumé des données météo.	Si des balises sont liées à des serveurs tiers, ces serveurs sont les sources des données via leur WS.
Afficher balises en mode cartographiques	Positionner les balises sur une carte géographique.	<ul style="list-style-type: none"> <li>- prévoir le centrage de la vue sur la position de l'utilisateur, ou sinon disponible une vue par défaut</li> </ul>
Afficher détail balise	Affiche toutes les données pertinentes pour l'utilisateur : <ul style="list-style-type: none"> <li>- localisation</li> <li>- données météo actuelles</li> <li>- résumé sur la période récente</li> </ul>	Présenter une rose des vents synthétisant la vitesse et la direction des vents sur la dernière heure écoulée.
Afficher historique balise	Afficher les données météo sous forme de graphes temporels	<p>Utilisation de la librairie HICARTS</p> <ul style="list-style-type: none"> <li>- permettre la navigation dans le temps (exploration)</li> </ul>

		<ul style="list-style-type: none"> <li>- des historiques)</li> <li>- tenir compte de l'agrégation des échantillons pour les longues périodes</li>   <li>- présenter une rose des vents pour la période étudiée.</li> </ul>
<b>Publier données WS</b>	Publier les données sous forme de webservices	-
publier liste balises	Fournit la liste des balises avec leurs caractéristiques	-
publier état et données balise	Fournit l'état d'une balise et ses données météo actuelles	<ul style="list-style-type: none"> <li>- données actuelles = dernier message reçu</li> <li>- signaler si données météo non disponibles</li> <li>- toutes les mesures doivent être horodatées</li> </ul>
publier données météo historiques	Fournit les données d'une balise pour une période donnée.	<p>Prévoir :</p> <ul style="list-style-type: none"> <li>- données détaillées (liste d'échantillons)</li> <li>- résumé (valeurs statistiques)</li> </ul>

### 3) Cas d'utilisation : Application Android

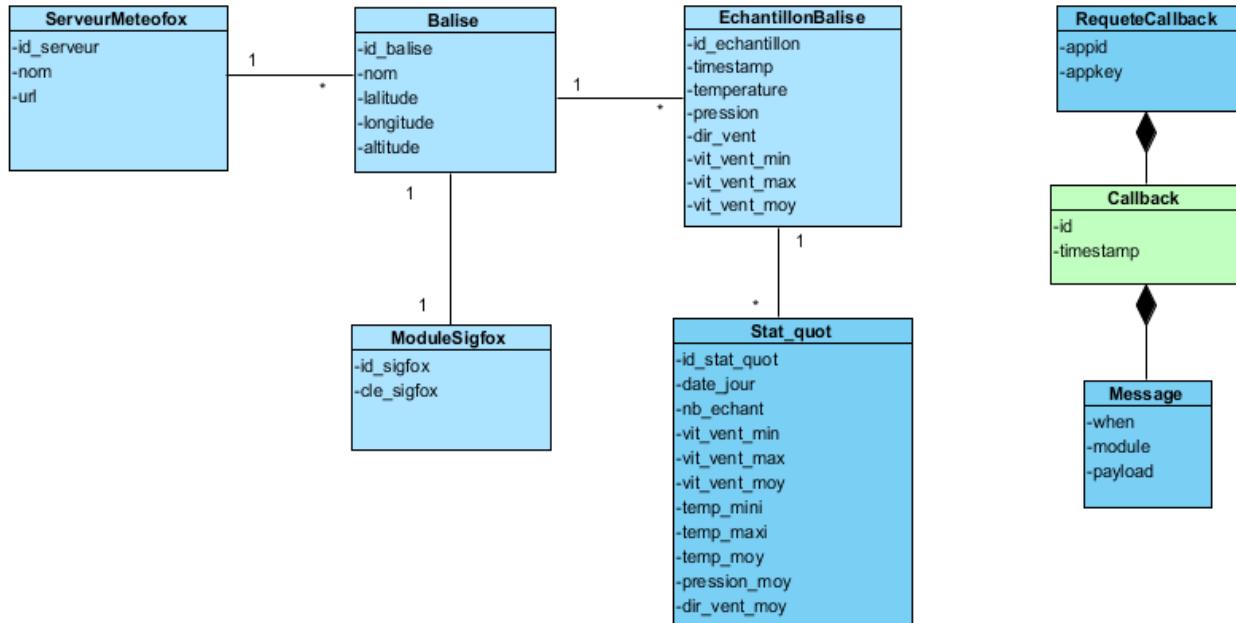


Cas d'utilisation	Description	Contraintes
Afficher liste et état des balises	Une liste des balises est affichée, avec leur état et un résumé des données météo	
Afficher balises en mode cartographiques	Positionner les balises sur une carte géographique.	
Recentrer sur position actuelle	Recentrer la carte sur la position actuelle	si géolocalisation disponible.
Choisir les préférences	Choisir les préférences de l'application : <ul style="list-style-type: none"> <li>- URL du serveur METEOFox</li> <li>- vue par défaut</li> <li>- ...</li> </ul>	
Afficher détail balise	Affiche toutes les données pertinentes pour l'utilisateur : <ul style="list-style-type: none"> <li>- localisation</li> </ul>	Présenter une rose des vents synthétisant la vitesse et la direction des vents sur la

	<ul style="list-style-type: none"><li>- données météo actuelles</li><li>- résumé sur la période récente</li></ul>	dernière heure écoulée.
Afficher historique balise	Afficher les données météo sous forme de graphes temporels	<p>Utilisation de la librairie <b>AChartEngine</b>.</p> <ul style="list-style-type: none"><li>- permettre la navigation dans le temps (exploration des historiques)</li></ul>

## 1.4) Description de l'architecture

### 1.4.1) Diagramme de classes

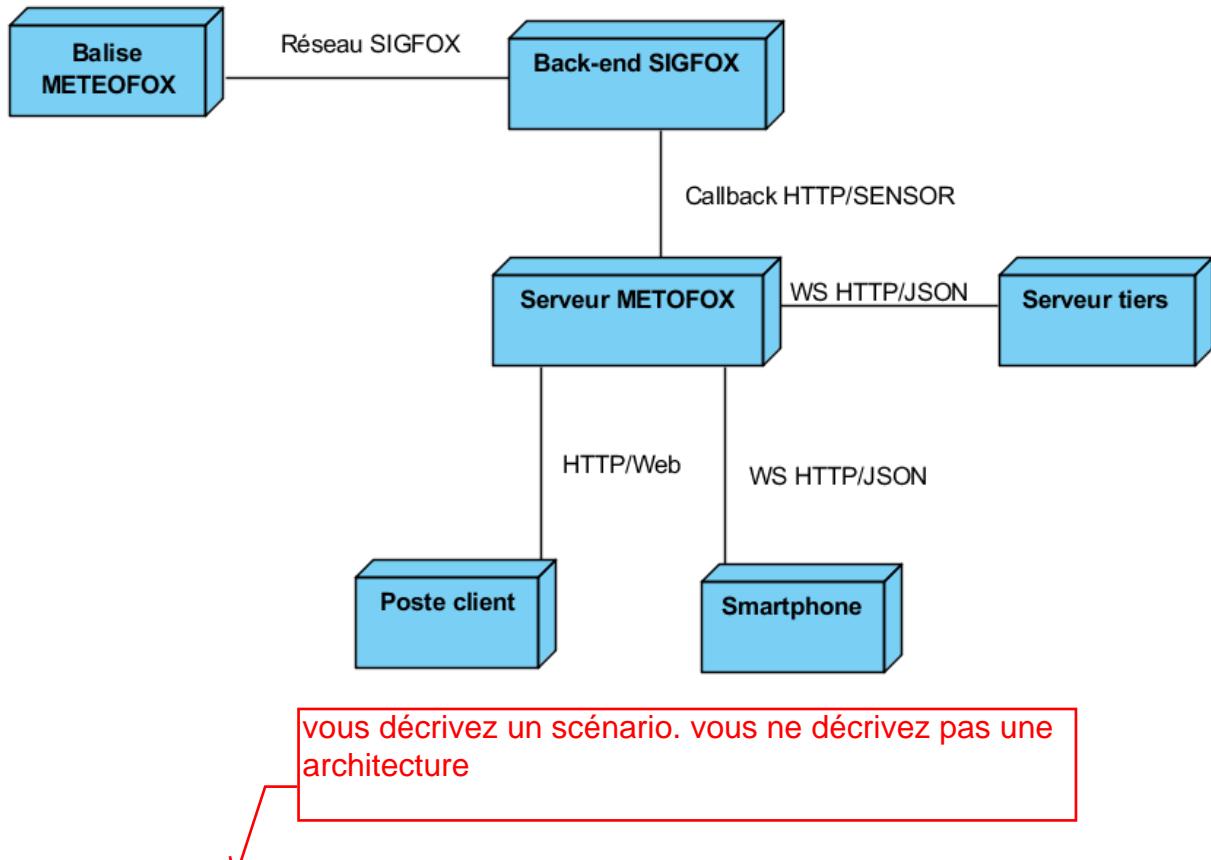


légende ?

-> "classes du domaine"

aucune explication ....

### 1.4.2) Diagramme de déploiement



La balise va envoyer une trame sur le Back-end Sigfox via le Réseau Sigfox. Celui-ci va ensuite « pousser » les données vers notre Serveur Meteofox via une callback http. Les données météorologiques seront ensuite enregistré dans une base données. Celles-ci seront consultable soit depuis un site web via des requêtes http entre le serveur et le navigateur du client ou bien via une application smartphone Android via un webService http et JSON.

**JSON (JavaScript Object Notation)** est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

Les **services web** (en anglais *web services*) représentent un mécanisme de communication entre applications distantes à travers le réseau internet indépendant de tout langage de programmation.

Descriptif (sous système, ...) ???
Auteur(s) : E2 encore ce E2 !!

### 1.5) Répartition du travail par étudiant

Nom Prénom	Mission	Sous-système
E1 : JAUME Benjamin	Traitement des données des capteurs (avec un ARM32)	Balise météorologique
E2 : Nicolas Kevin	Traitement des données des capteurs (avec un Arduino)	
E3 : CAYUELA Julien	Injection et Organisation dans une Base de données des données reçues du module SIGFOX	
E4 : DESCARD Baptiste	Administration et gestion du site web	Serveur METEOFOX et Web
E5 : TINET Ludovic	Détails des balises en temps réel et historisation des données météo	Service de collecte des données
E6 : AMAND Rémi	Publication des données	
E7 : RADULOV Lachezar	Cartographie des balises	
E8 : AUDOY François	IHM application mobiles / gestion des balises	
E9 : BEYRAND Julien	Historisation des balises (application mobile)	Application Android

## 2) E1 :JAUME Benjamin - Acquisition/Traitement des données

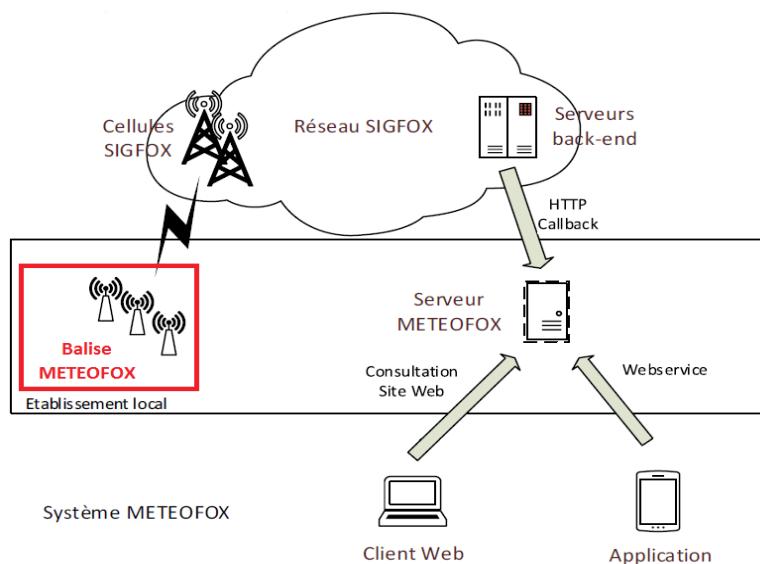
### 2.1) Présentation de la partie personnelle

Mon rôle dans le projet consiste à construire une balise METEOFOX. Cette balise comporte les éléments suivant :

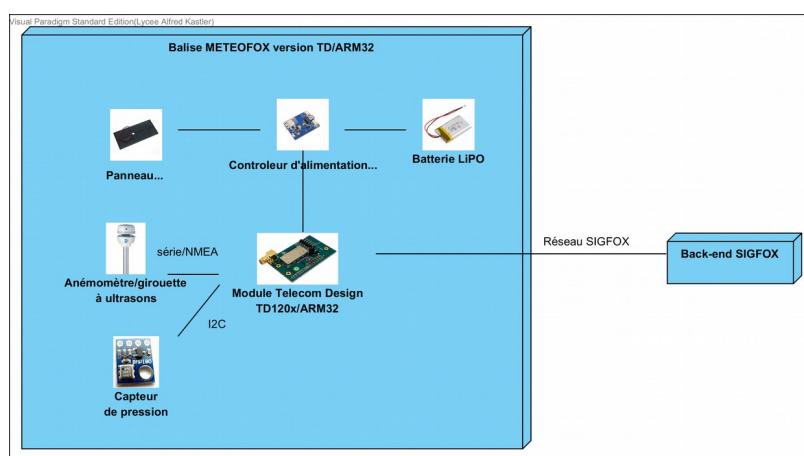
- Un capteur faisant anémomètre et girouette
- Un capteur de pression
- Un module de transmission sans fil : TD1204

Elle est à la base de notre projet, c'est grâce à la balise que l'on va récupérer les grandeurs physiques. Elle sera capable de transmettre les données météorologiques via le réseau Sigfox au serveur METEOFOX mis en place et géré par notre équipe.

#### Place de la balise dans le projet



#### Synoptique d'une balise METEOFOX



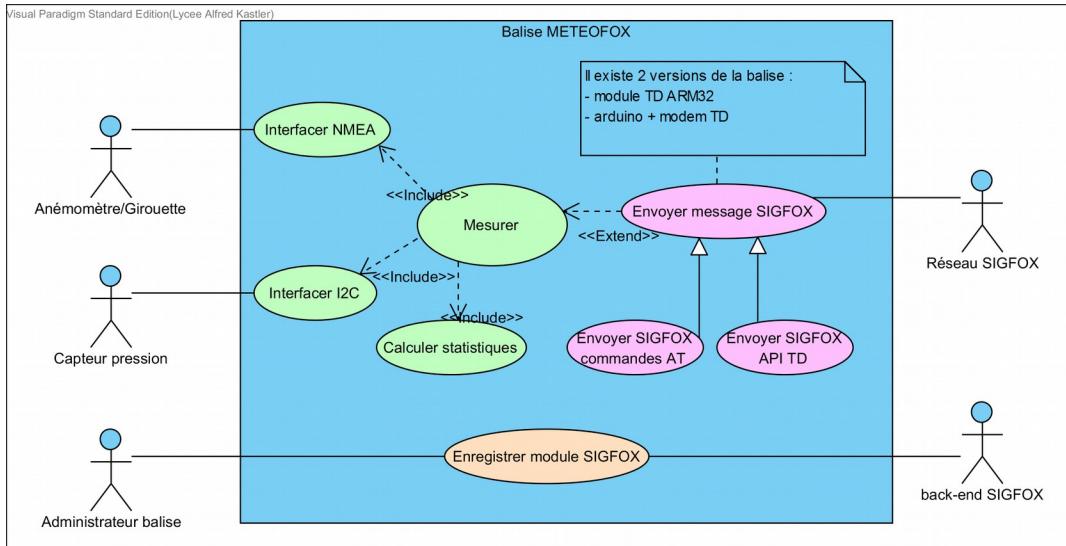
légende ?

Descriptif (sous système, ...)
--------------------------------

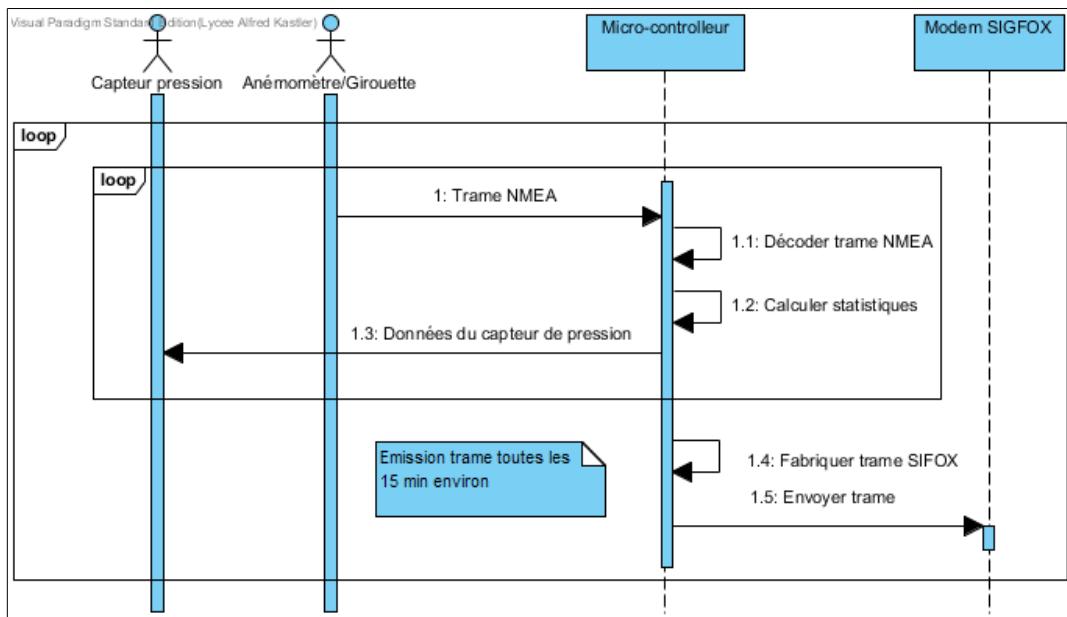
Auteur(s) : E2
----------------

Page 21 / 130
---------------

### Diagramme de cas d'utilisation de la balise METEOFOX



### Diagramme de séquence du cas d'utilisation « Mesurer »



Dans le diagramme de séquence général du sous – système « Balise », les étapes sont :

1. **Réception une trame NMEA** (environ toute les secondes lorsque la batterie du capteur est pleine, et environ toutes les 15 secondes lorsque la batterie n'est pas assez chargée)
2. **Décodage de la trame** en extrayant les données qui nous intéressent (voir plus bas)
3. **Calcul des statistiques**, qui va permettre de connaître les variations météorologiques au cours du temps
4. **Récupération des informations de pression** dans le capteur prévu à cet effet
5. Toutes les **15 minutes**, **fabrication d'une trame** contenant les informations météorologiques, puis on **envoie** la trame vers le serveur METEOFOX

## 2.2) Mise en œuvre

### 2.2.1) Technologies utilisées

#### 1) Anémomètre / girouette



Le capteur qui nous est proposé possède trois fonctions. C'est à la fois :

- un anémomètre
- une girouette
- un capteur de température

Il dispose de 4 émetteurs/récepteurs à ultrasons qui permettent d'évaluer le temps de propagation du son d'un émetteur/récepteur . Le vent va modifier la vitesse et la direction du faisceau, et permettre le calcul. Enfin, il transmet des trames de type NMEA par ondes radio.

#### Avantages par rapport aux capteurs mécaniques :

- Plus d'éléments tournants : empêche les frottements et l'usure
- Capteur placé en extérieur, donc augmentation de la longévité
- Évite une maintenance régulière

#### 2) Capteur de pression

Le capteur de pression se charge de mesurer la pression barométrique, indispensable pour compléter les informations météorologiques. Il est doté d'une interface bus I<sup>2</sup>C, et il est très peu coûteux.



### 3) Module de communication sans fil



C'est un TD1204 (composant fabriqué par la start – up toulousaine Telecom Design). Il utilise la technologie « Sigfox » pour transmettre ses messages sans fil. Il contient un processeur disposant d'une architecture ARM32.

### 4) Technologie Sigfox



Il s'agit d'un réseau cellulaire dédié à l'internet des objets. Il fonctionne sur la bande de fréquence 868 MHz utilisant UNB (« Ultra Narrow Band » : bande ultra étroite). Pour comparaison, le réseau GSM utilise les fréquences de 900 MHz jusqu'à 1900 MHz.

N.B : l'internet des Objets, c'est l'extension d'internet. Elle représente les échanges d'informations et de données provenant de dispositifs présent dans le monde réel vers le réseau internet.

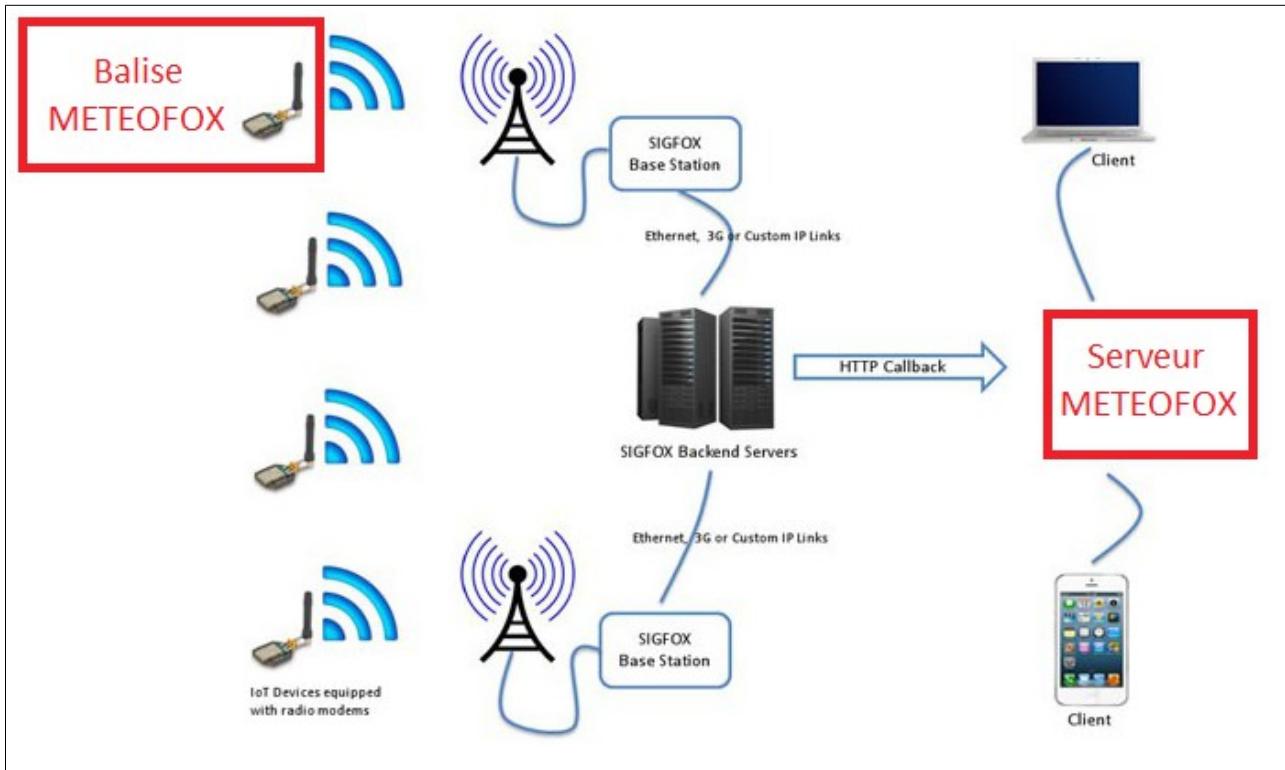
#### Avantages de cette technologie :

- 40 km de portée pour chaque antenne
- 1000 antennes en France, couvrant 90% du territoire
- La consommation des puces Sigfox est 1000 fois plus faible que celle des puces GSM
- Le coût d'abonnement est très faible : de 1€/par à 14€/an

#### Inconvénients :

- Limite de 150 messages / jour
- Limite de 12 octets / message

Fonctionnement du réseau  
Sigfox dans notre projet

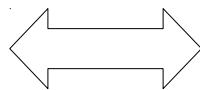


### 5) Sonde de programmation et de débogage

Pour programmer le module de transmission sans fil (TD1204), une sonde de programmation et de débogage est utilisée, elle va servir de passerelle entre l'ordinateur et le module.



EFM32 (sonde de débogage)



TD1204

C'est un microcontrôleur doté d'une architecture ARM32, qui consomme très peu d'énergie. La programmation se fera à l'aide de l'environnement de développement Eclipse en langage C pur.



## 2.2.2) Mise en œuvre des techniques

### 1) Montage

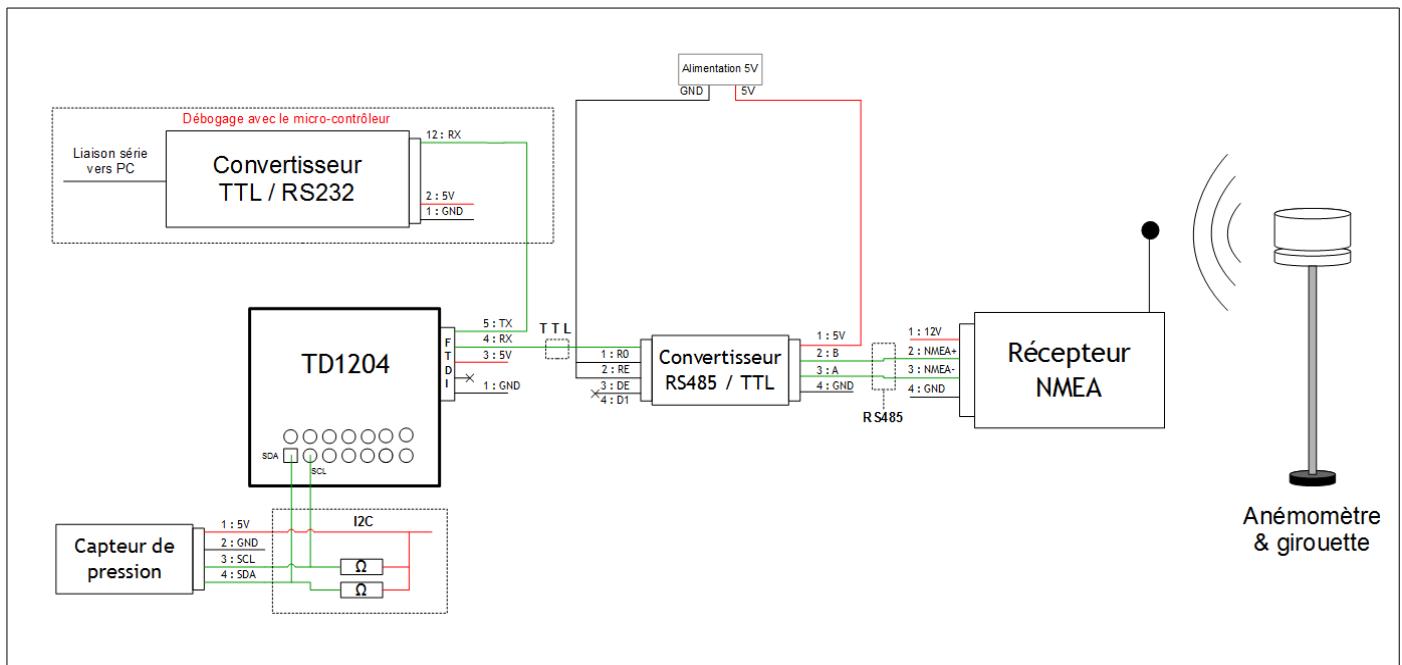


Schéma de câblage de la balise

Le montage ci dessus permet aux différents du système éléments de communiquer. Description des différents éléments :

1. Le capteur anémomètre/girouette envoie ses données par ondes radio sous forme de trames NMEA
2. Les ondes radio sont récupérées par le « Récepteur NMEA », et sont renvoyées sur une liaison série de type RS-485
3. Un convertisseur RS-485 vers TTL est mis en place pour permettre au TD1204 de lire des signaux TTL
4. Les données sont directement acheminées vers le TD1204 sur le port « RX » de sa connectique FTDI
5. Le capteur de pression est relié en I2C au TD1204.
6. Un convertisseur TTL vers RS-232 est mis en place pour permettre d'envoyer des caractères vers l'ordinateur afin de faire du débogage.

## 2) Format des trames NMEA

Par soucis de répartition des tâches, le protocole NMEA est expliqué plus en détails dans la partie personnelle de mon collègue NICOLAS Kévin.

Tableau d'analyse d'une trame NMEA de vent

\$ IIMWV , 136.0 , R , 004.80 , N , A * 05												
Start	« Tag »	Séparateur	Données								Checksum	Marqueur de fin de ligne
\$	Type de trame	,	Angle			Vitesse				*	0xXX	Retour Chariot + Retour à la ligne
\$	IIMWV	,	136.0	,	R	, 004.80	,	N	, A	*	05	[CR][LF]

Grâce au tableau, on sait :

- Qu'il s'agit d'une trame NMEA de vent (IIMWV)
- Que l'angle du vent est de 136.0 degrés
- Que la vitesse du vent est de 4.8 Nœuds (N)

Tableau d'analyse d'une trame NMEA de température

\$ WIXDR , C , 007 , C „ U , 4.1 , V „ * 63												
Start	« Tag »	Séparateur	Données								Checksum	Marqueur de fin de ligne
\$	Type de trame	,			Température						*	0xXX
\$	WIXDR	,	C	,	007	,	C	„	U	, 4.1	,	V „ * 63

Grâce au tableau, on sait :

- Qu'il s'agit d'une trame NMEA de température (WIXDR)
- Que la température est de 007 °Celsius

## 3) Format des trames envoyées sur le réseau Sigfox

Les mesures qui vont être envoyées au serveur METEOFOX sont :

- L'angle du vent
- La température
- La vitesse moyenne du vent
- Le vitesse minimale du vent
- La vitesse maximale du vent
- La pression

Descriptif (sous système, ...)	??
Auteur(s)	E2 ??

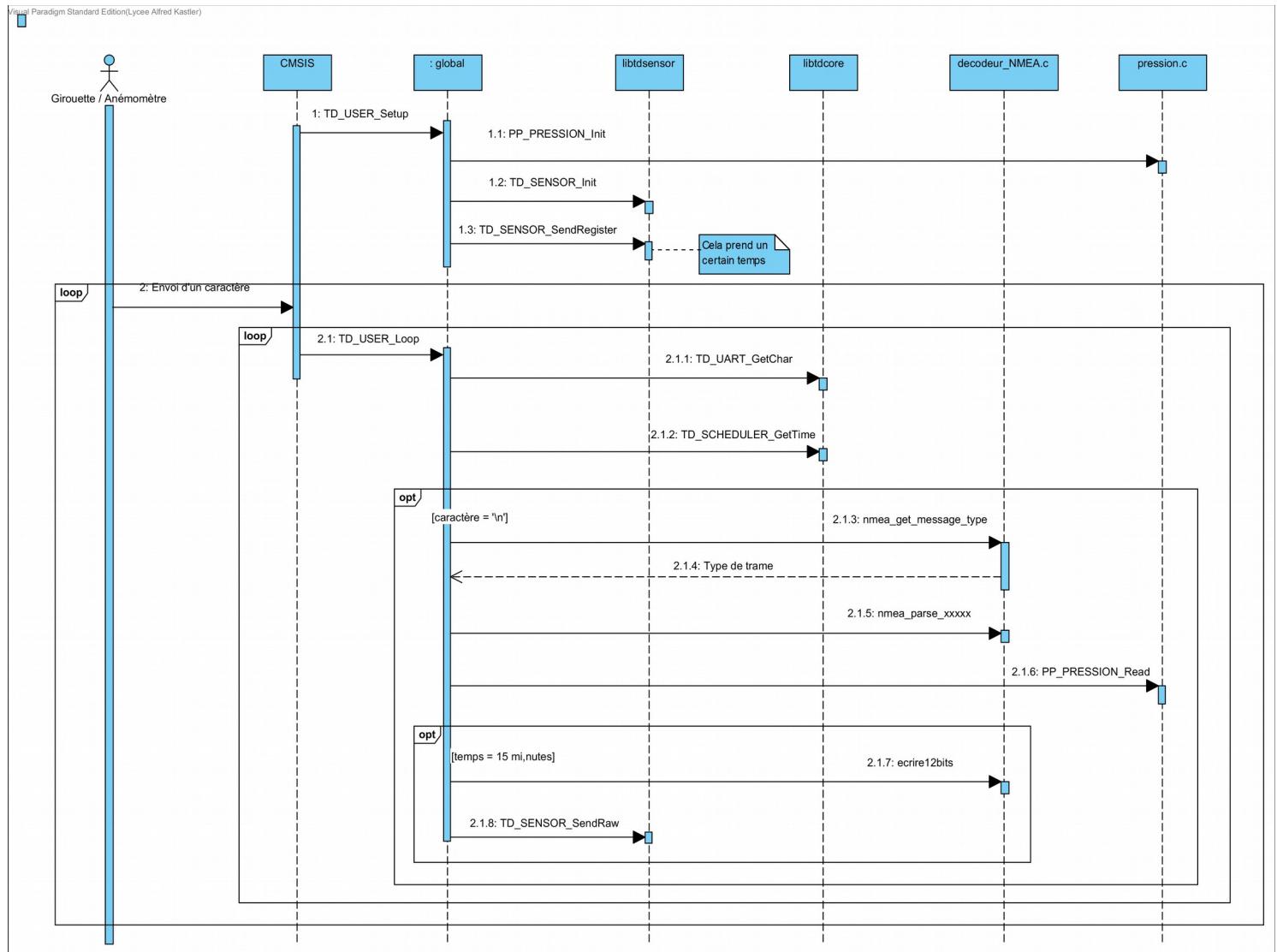
Grâce à la documentation technique, on sait que l'on ne peut envoyer que 10 octets par message à travers le réseau Sigfox. En réalité, 12 octets sont transmis par notre module, mais les deux octets perdus sont réservés aux serveurs SENSOR.



## 2.3) Conception détaillée

### 2.3.1) Diagramme de séquence

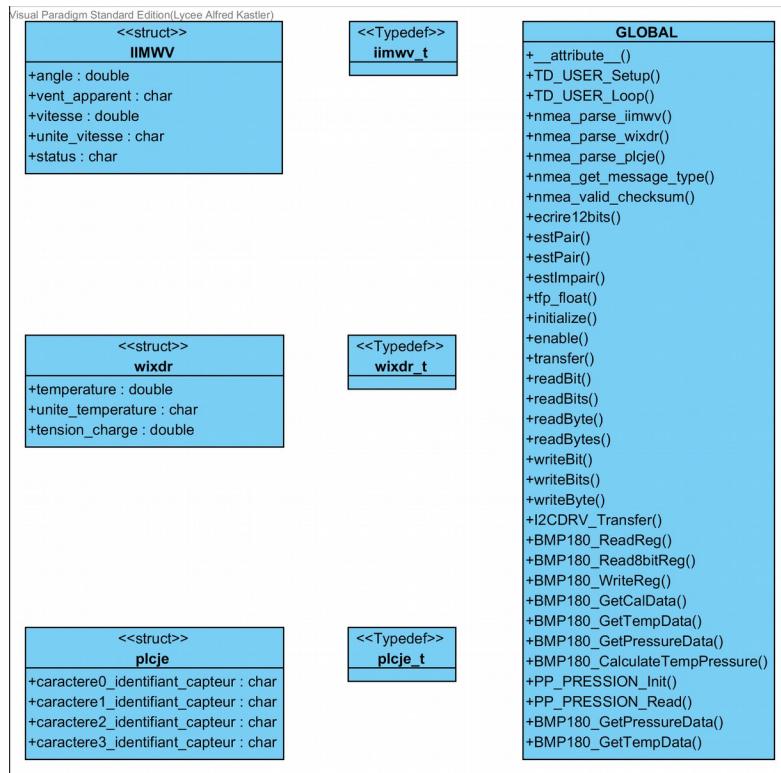
Diagramme de séquence détaillé  
du cas d'utilisation « Mesurer »



cela aurait mérité pas mal d'explications !!  
Vous n'exploitez pas du tout le diagramme.

### 2.3.2) Diagramme de classes

#### Diagramme de classe détaillé du sous système balise

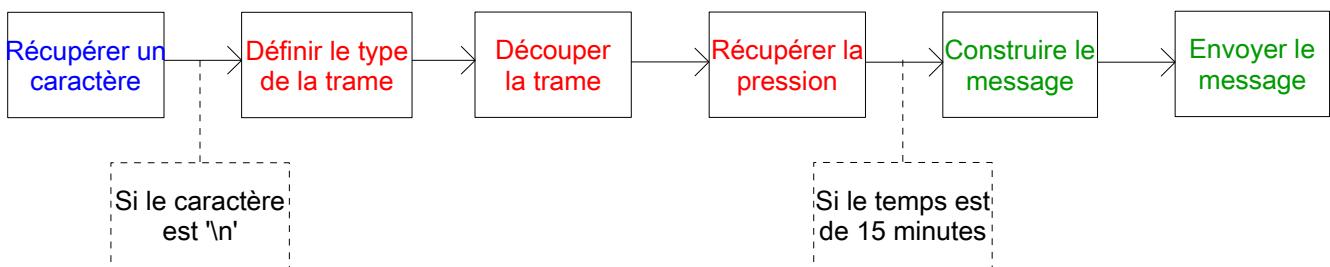


idem  
aucune explication

### 2.4) Implémentation

#### 2.4.1) Présentation

Le programme étant assez long, avec différentes fonctionnalités, il y aura une partie de code pour chaque fonction principale du programme.



## 2.4.2) Extraits de code

### 1) Récupération d'un caractère :

```
while ((caractere = TD_UART_GetChar()) >= 0) {
    [...]
}
```

Dans cette partie du code, on inspecte la liaison TTL, pour voir si un caractère passe (c'est la fonction « `TD_UART_GetChar()` » qui permet de le faire). Tant qu'il n'y a pas de caractère sur la liaison TTL, on continue à lire dessus.

### 2) Définir le type de la trame :

**déecter / analyser**

```
#define NMEA_IIMWN 0x01
#define NMEA_IIMWN_STR "$IIMWV"
#define NMEA_WIXDR 0x02
#define NMEA_WIXDR_STR "$WIXDR"
#define NMEA_PLCJE 0x03
#define NMEA_PLCJE_STR "$PLCJE"

[...]

uint8_t checksum_test = nmea_valid_checksum(message);

if (strstr(message, NMEA_IIMWN_STR) != NULL ) {
    return NMEA_IIMWN;
}

if (strstr(message, NMEA_WIXDR_STR) != NULL ) {
    return NMEA_WIXDR;
}

if (strstr(message, NMEA_PLCJE_STR) != NULL ) {
    return NMEA_PLCJE;
}

if (checksum_test != CHECK_OK) {
    return checksum_test;
}

return NMEA_UNKNOWN;
}
```

Dans cette partie du programme, on est dans la fonction appelée afin de définir le type de trame reçu. On utilise la fonction « `strstr` » (qui trouve la première occurrence dans une chaîne de caractère). Ainsi, si le type est trouvé, on retourne soit 0x01, 0x02, 0x03 pour définir le type de trame.

### 3) Découper la trame :

Il n'y aura comme exemple que le découpage d'une trame de température. Mais il en existe 3 au total : pour le vent, la température, et les informations du capteurs.

Rappel de la composition d'une trame de température :

**\$ WIXDR , 007 , C , , U , 4.1 , V , , \* 63**

```
void nmea_parse_wixdr(char *phrase_nmea, wixdr_t *message) {
    char *phrase = phrase_nmea;

    phrase = strchr(phrase, ',') + 1; // passer le 'C' de départ

    phrase = strchr(phrase, ',') + 1;
    message->temperature = atof(phrase); // récupération de la température

    phrase = strchr(phrase, ',') + 1;
    switch (phrase[0]) {
        case 'C':
            message->unit_temperature = 'C'; // récupération l'unité de température
            break;
        case ',':
            message->unit_temperature = '\0';
            break;
    }

    phrase = strchr(phrase, ',') + 1;
    phrase = strchr(phrase, ',') + 1;

    phrase = strchr(phrase, ',') + 1;
    message->tension_charge = atof(phrase); // récupération de la tension de
                                                // charge du module
}
```

Principe de découpage de la trame : comme toutes les informations sont séparées par des virgules, on balaye alors la trame afin de les chercher. On met ensuite les informations (de température, etc) dans les variables d'une structure afin de les récupérer dans notre programme principal. **-> conversion ASCII -> float avec atof()**

### 4) Récupération de la pression

```
#define ALTITUDE 26.3345 ???(je suis choqué !)

PP_PRESSION_Read(&temp, &pression_Pa);
pression_Pa = pression_Pa - (ALTITUDE * 843);
pression_hPa = pression_Pa / 100;
```

Pour récupérer la pression, une fonction toute prête existe déjà. On sait ensuite, avec un peu de recherche, qu'elle renvoie une pression en « Pascal » (Pa). Pour la convertir en « hecto Pascal » (hPa), il faut donc diviser cette valeur par 100. La pression dépend également de l'altitude à laquelle

on se trouve, il faut donc que ce paramètre entre dans l'équation, sachant la pression augmente de 1 hPa tous les 8.43 mètres (donc de 10000 Pascal tous les 843 mètres)

### 5) Construire le message

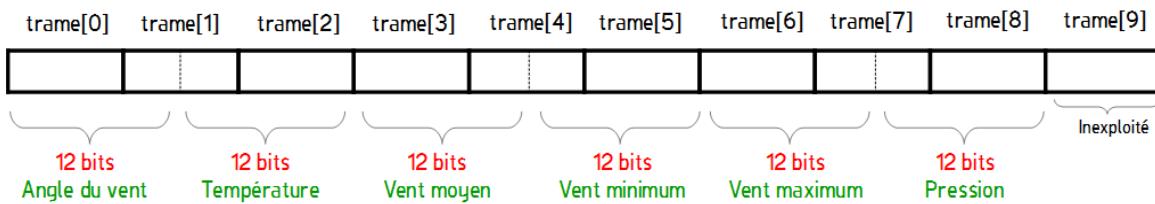
```
écrire12bits(trame, (int) (angle *  
écrire12bits(trame, (int) (températ  
écrire12bits(trame, (int) (vent_moy  
écrire12bits(trame, (int) (vent_max  
écrire12bits(trame, (int) (vent_min  
écrire12bits(trame, (int) (pression  
  
// Assemblage de la trame envoyée au serveur METEOFOX  
  
char message[10] = { trame[0], trame[1], trame[2], trame[3],  
trame[4], trame[5], trame[6], trame[7], trame[8],  
trame[9] };
```

Attention : la pression mesurée est telle que délivrée par le capteur (non corrigée avec l'altitude).

Votre discussion fait probablement référence à la pression ramenée au niveau de la mer, laquelle exige effectivement un calcul selon l'altitude.

Ce n'est visiblement pas bien compris.

Pour que toutes les mesures (vent, température, etc) puissent rentrer dans la « payload », la fonction « `écrire12bits` » va les mettre à la suite dans un tableau de taille 10 octets non signés : les 12 premiers bits de chaque valeurs seront exploités (c'est dans les 12 premiers bits de poids faible que se trouvent les données).



Grâce à ce principe, toutes les info en une seule fois.

cela fait des mois que je vous demande un document mentionnant les unités des différentes mesures. Résultat : on ne sait toujours pas ce que l'on envoie comme unités de mesures.

### 6) Envoi du message

```
GPIO_PinOutToggle(LED_PORT, LED_BIT); // Allumage de la led  
  
if (TD_SENSOR_SendRaw(message, 10)) {  
    tfp_printf(" --> MESSAGE ENVOYE AVEC SUCCES \r\n\r\n");  
    GPIO_PinOutToggle(LED_PORT, LED_BIT); // Extinction de la led  
} else {  
    tfp_printf(" --> PROBLEME LORS DE L'ENVOI DU MESSAGE\r\n\r\n");  
    GPIO_PinOutToggle(LED_PORT, LED_BIT); // Extinction de la led  
}
```

Pour envoyer le message précédent « assemblé », on utilise la fonction « `TD_SENSOR_SendRaw` », qui renvoi un code d'erreur si une erreur c'est produite. Un témoin lumineux nous permet de constater que le message est en train d'être envoyé.

on aurait aimé voir des explications sur le codage de la fonction `écrire12bits()` !

## 2.5) Tests de validation

### 2.5.1) Type de test

Grâce au convertisseur TTL vers RS-232, on peut envoyer des caractères sur la liaison série directement vers l'ordinateur, ce qui permet de faire du débogage. Ainsi, on peut observer le débogage du système avec le logiciel Putty (émulateur de terminale connecté sur le port série du PC) :

```
----- BOOT -----  
  
- Initialisation du module : OK  
- Enregistrement du module : OK  
  
-----  
  
# TIMER : 12 secondes  
# Temps d'attente : 900 secondes  
  
--> Angle : 0 degrès  
--> Vitesse moyen : 0 noeud(s) (0 km/h)  
--> Vitesse minimum : 0 noeud(s) (0 km/h)  
--> Vitesse maximum : 0 noeud(s) (0 km/h)  
--> Température : 23 degrès Celsius  
--> Pression : 106954 Pa (1069 hPa)  
  
-----  
  
# TIMER : 43 secondes  
# Temps d'attente : 900 secondes  
  
--> Angle : 315 degrès  
--> Vitesse moyen : 0 noeud(s) (0 km/h)  
--> Vitesse minimum : 0 noeud(s) (0 km/h)  
--> Vitesse maximum : 0 noeud(s) (0 km/h)  
--> Température : 23 degrès Celsius  
--> Pression : 106819 Pa (1068 hPa)  
  
-----
```

Copie d'écran montrant les caractères envoyés sur la liaison série par le TD1204 à des fins de débogage

<b>⌚ History</b>				
Emission Date	Reception Date	Index	Msg	[Payload]
2015-05-28 10:38:13	2015-05-28 10:38:14	0	raw	<ul style="list-style-type: none"> <li>Frame Type : srv_frm_raw</li> <li>Raw : 050d1f00000000bfeb00</li> <li>ASCII Raw : ???????é?</li> </ul>
2015-05-28 10:23:10	2015-05-28 10:23:11	0	raw	<ul style="list-style-type: none"> <li>Frame Type : srv_frm_raw</li> <li>Raw : 000d1f00000000afeb00</li> <li>ASCII Raw : ???????é?</li> </ul>
2015-05-28 10:08:23	2015-05-28 10:08:23	0	raw	<ul style="list-style-type: none"> <li>Frame Type : srv_frm_raw</li> <li>Raw : 050d1f00000000aeee00</li> <li>ASCII Raw : ???????@é?</li> </ul>

Capture d'écran du « dashboard »  
(tableau de bord) du TD1204

Sur cette copie d'écran, on peut observer que des messages sont transmis toutes les 15 minutes. Les messages sont de types « raw » (brut).

**Analyse d'un message 'raw'**

Trame :		050 019 00A 00D 000 3F5 00
		Interprétation
Valeurs en hexadécimal	Valeurs en décimal	Interprétation
050	80	Angle du vent : <b>80</b> de degrés
019	25	Température : <b>25</b> degrés Celsius
00A	10	Vitesse moyenne du vent : <b>10</b> de nœuds (19 km/h)
00D	13	Vitesse maximale du vent : <b>13</b> de nœuds (25 km/h)
000	0	Vitesse minimale du vent : <b>0</b> de nœuds (XX km/h)
3F5	1013	Pression atmosphérique : <b>1013</b> hPa
00	0	/

## 2.6) Conformité aux missions

Les différentes missions ont toutes étaient réalisées, j'ai donc accomplie toutes les tâches qui m'ont été attribué :

Mesurer, interfacer I2C, interfacer NMEA, calculer les statistiques, envoyer message Sigfox

Descriptif (sous système, ...)	
Auteur(s) : E2	Page 35 / 130

## 2.7) Bilan personnel technique

Grâce à ce projet technique, j'ai pu approfondir mes connaissances en C pur, découvrir un nouveau type de transmission (Sigfox) que je compte utiliser plus tard à des fins personnelles.

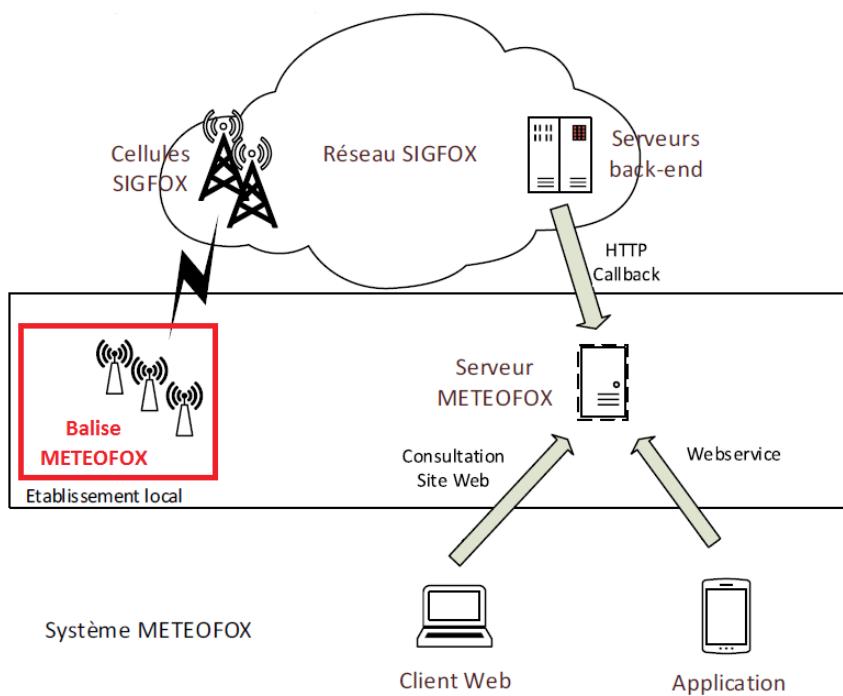
### 3) E2 : NICOLAS Kévin - Acquisition/Traitement des données

#### 3.1) Description de la partie personnelle

##### 3.1.1) Travail à effectuer au sein du système

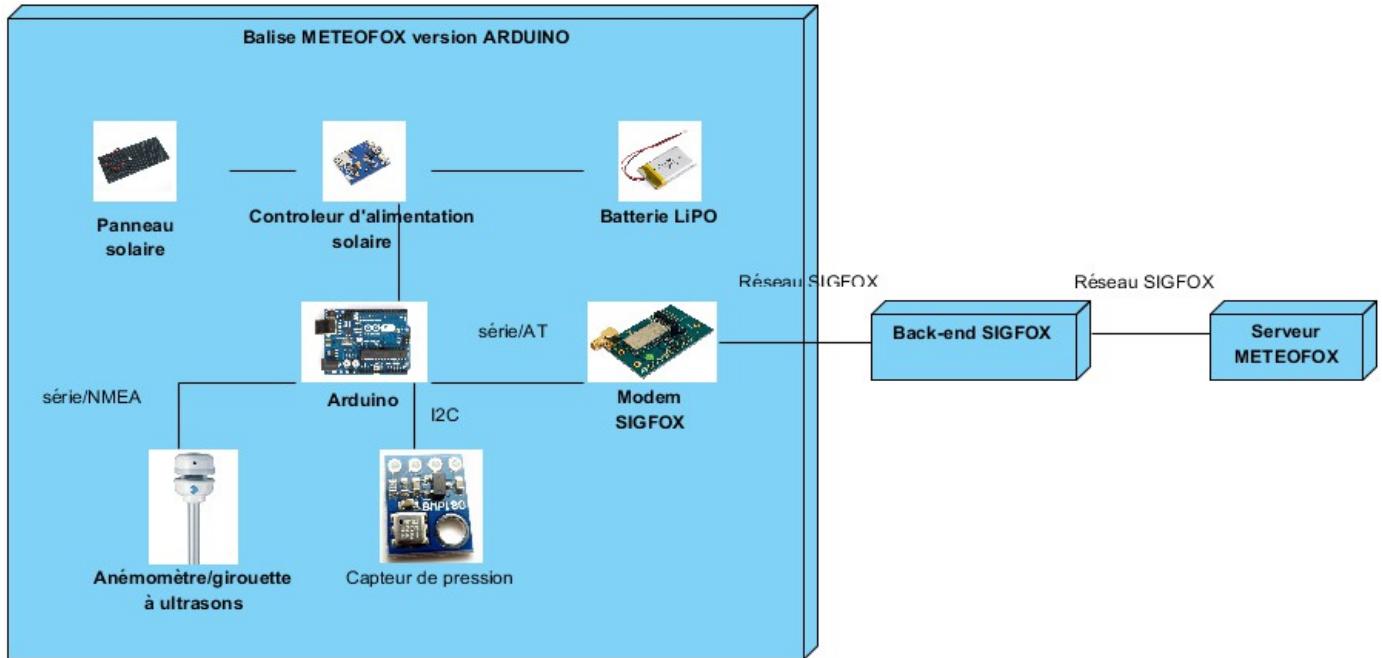
Je suis chargé de la mise en œuvre d'un capteur de pression (BMP180), d'une anémomètre (CV7SF) et d'un module Telecom Design (TD1208) qui est un module de communication par radio et qui permet d'envoyer les trames via le réseau SIGFOX.

Tout ceci est relié à un microcontrôleur de type Arduino (le Arduino UNO) qui est, lui, chargé de collecter les données et de créer la trame qui sera envoyée au serveur web du projet.



*Représentation de ma partie dans le système*

Descriptif (sous système, ...)	??
Auteur(s) : E2	toujours ce mystérieux E2 !!

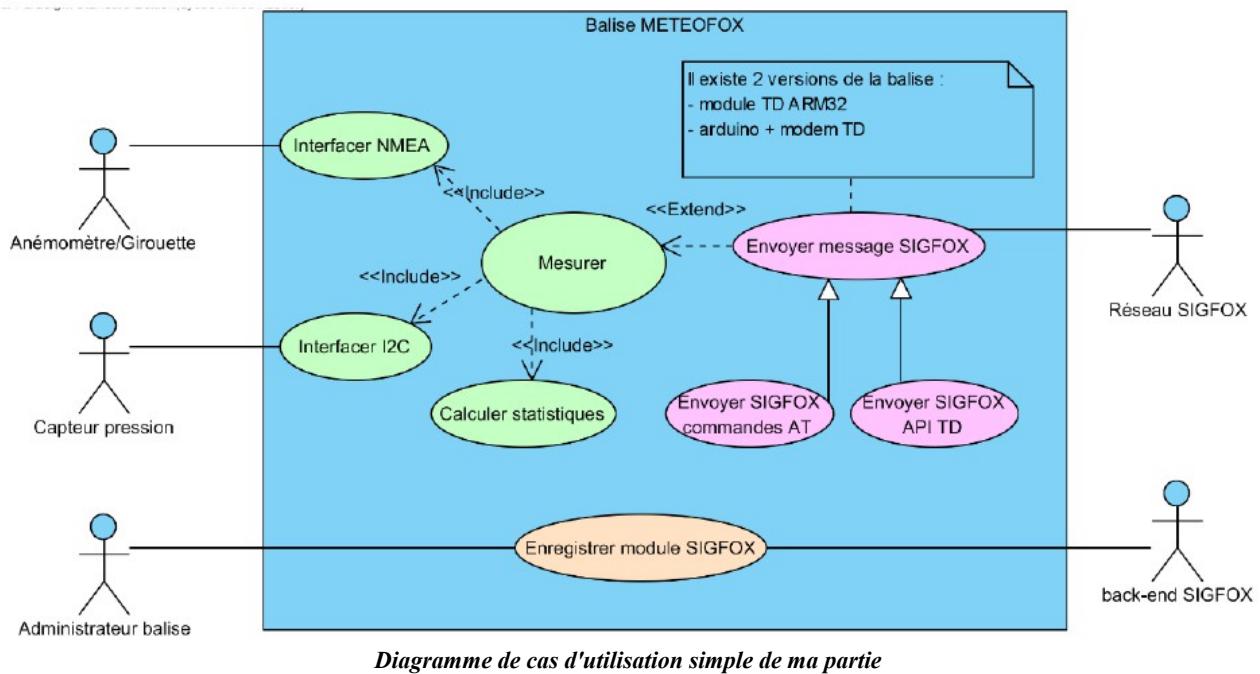
*Diagramme de déploiement de ma partie*

On peut voir sur ce diagramme la composition de la balise METEOFOX, on a une vue globale du système sur lequel j'interviens.

On peut voir, au centre, le microcontrôleur Arduino qui est le cerveau du système, il récupère les données de vents de l'anémomètre et la température via la liaison série.

On a ensuite le capteur de pression qui envoie, sur demande de l'Arduino, les données de pression et de température mais on ne garde que la pression, la température étant déjà récupérée par l'anémomètre.

On fini par envoyer ces données au serveur METEOFOX grâce au modem SIGFOX  
(les données passent par les back-end entre le modem et le serveur)



Sur ce diagramme on peut voir les interactions qui ont lieu entre la balise et son environnement.

En vert la partie des mesures, en rose la partie communication et en beige il s'agit de l'enregistrement du module sur les serveurs SIGFOX.

### 3.2) Mise en œuvre

#### 3.2.1) Techniques / Technologies utilisées

##### 3.2.1.1) *Le microcontrôleur : Arduino UNO*



***Un arduino Uno***

Pour répondre aux besoins du cahier des charges, un arduino Uno est donc utilisé.

Il s'agit d'un microcontrôleur de type AVR le ATMEGA 328 à 8 bits.

Il est programmé en C/C++ et pour ma part avec la suite Visual Studio et le plugin Visual Micro permettant d'ajouter le support de l'arduino par visual studio.

Facile d'accès et possédant une communauté active, travailler avec ce type de module est un plaisir car on trouve facilement de l'aide si l'on recherche un minimum sur internet et il y a beaucoup de projets déjà réalisés avec cette technologie.

- Pourquoi le modèle UNO ?

Ce modèle a été choisi en fonction des besoins du projet, il est peu gourmand en énergie (7 - 12V) et possède de nombreuses entrées/sorties et est capable de fournir l'alimentation requise aux capteurs (il génère du 3,3V, du 5V et est de plus munis de multiples GND).

De plus il comporte une entrée VIN pour l'alimenter avec une source de notre choix.

Parmi ces entrées/sorties nous pouvons noter les plus importantes qui sont :

- Les broches pour le BUS I2C du capteur de pression : A4 pour le SDA et A5 pour le SCL
- Les broches pour les liaisons série avec le module SIGFOX et l'anémomètre 0(RX) et 1(TX), il nous faut deux paires donc la deuxième se fera de manière logiciel avec la fonction **SoftwareSerial**

### 3.2.1.2 ) L'anémomètre/girouette à ultrasons



*L'anémomètre avec son boîtier*

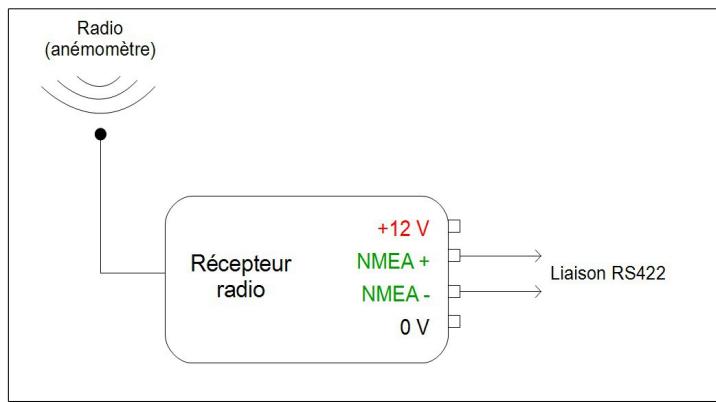
oui, mais cela n'a pas été mis en oeuvre



L'anémomètre est autonome en énergie, il est alimenté par son panneau solaire ce qui est indispensable pour notre projet car il peut être placé à des endroits difficiles d'accès de plus c'est un appareil à ultrasons contrairement aux anémomètre-girouette classiques qui sont mécaniques et donc souffre des contraintes de forces. Ici nous avons donc une plus grande durabilité.

Pour ce qui est du fonctionnement, l'anémomètre transmet des données qui nous renseigne sur la vitesse, la direction et la température du vent via des ondes radio.

Ces données sont récupérées via un petit boîtier qui est fourni (voir photo ci-dessus).



*Schéma du récepteur anémomètre*

Ce récepteur transmet les données reçues via une liaison série RS422, qui, d'après la documentation doit être utilisé avec un format de transmission TTL.

Nous l'avons donc, dans notre montage, intégré avec à un convertisseur TTL (Transistor / Transistor Logic) pour l'acheminement des données jusqu'à l'arduino Uno sur une broche de réception série RX. Pour ce qui est des trames transmises, il s'agit de trames NMEA, la data est transmise en ASCII.

### 3.2.1.3 ) Le protocole NMEA

NMEA (National Marine & Electronics Association) est une Association à but non lucratif fondée par des professionnels de l'industrie électronique maritime, conjointement avec des fabricants, des distributeurs, des revendeurs et des institutions d'enseignements. NMEA est à l'origine de nombreux standards et en particulier du Standard NMEA-1083 qui nous intéresse.

Toutes les données sont transmises sous forme de caractères ASCII.

Les appareils NMEA peuvent être maîtres ou esclaves et utilisent une liaison série asynchrone pour dialoguer avec les paramètres suivants :

- 4800 bauds/s
- 8 bits de data ( bit 7 = 0 )
- 1 bit stop au moins
- Pas de parité

Les phrases NMEA transmises par le capteur sont composées d'un :

- délimiteur « **start** » (\$)
- préambule (le **type de la trame** : vent, température)
- champ de **données** (séparés par des virgules)
- caractère « \* »
- CRC32 « **Checksum** » (deux chiffres hexadécimaux)
- marqueur de **fin de ligne**

Tableau d'analyse d'une trame NMEA de vent

<b>\$ IIMWV , 136.0 , R , 004.80 , N , A * 05</b>												
Start	« Tag »	Séparateur	Données								Checksum	Marqueur de fin de ligne
\$	Type de trame	,	Angle		Vitesse				*	0xXX	Retour Chariot + Retour à la ligne	
\$	IIMWV	,	136.0	,	R	,	004.80	,	N	,A*	05	[CR][LF]

Grâce au tableau, on sait :

1. Qu'il s'agit d'une trame NMEA de vent (IIMWV)
2. Que l'angle du vent est de **136.0 degrés**
3. Que la vitesse du vent est de **4.8 Nœuds** (N)

### Tableau d'analyse d'une trame NMEA de température

		<b>S WIXDR , C , 007 , C , U , 4.1 , V , * 63</b>													
Start	« Tag »	Séparateur	Données										Checksum	Marqueur de fin de ligne	
\$	Type de trame	,		Température								*	0xXX	Retour Chariot + Retour à la ligne	
\$	WIXDR	,	C	007	C	,	U	,	4.	1	V	,	*	63	[CR][LF]

Grâce au tableau, on sait :

1. Qu'il s'agit d'une trame NMEA de température (WIXDR)
2. Que la température est de 007 °Celsius

#### 3.2.1.4 ) Le réseau SIGFOX



SIGFOX est un réseau cellulaire dédié à l'internet des objets autrement dit le Machine To Machine (M2M). Il fonctionne sur la bande de fréquence 868 MHz utilisant UNB (« Ultra Narrow Band » : bande ultra étroite). Bande

#### Avantages :

- 40 km de portée pour chaque antenne
- 90 % de couverture en France avec seulement 1000 antennes
- Consommation des puces SIGFOX 1000 fois plus faible qu'une carte GSM classique
- Le coût d'un abonnement est très faible de 1€ à 14€ par année et par appareil.

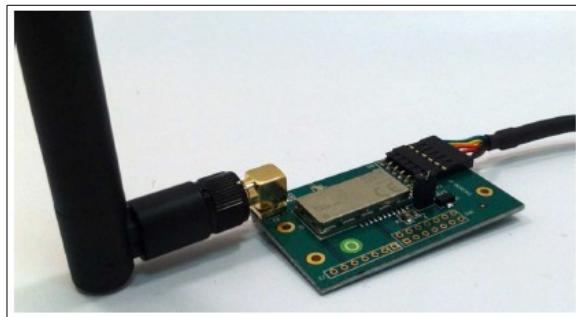
#### Inconvénients :

- Limite de 150 messages / jour
- Limite de 12 octets / message

Avec SIGFOX on découvre l'internet des objets qui est une évolution d'internet. Elle représente les échanges d'informations et de données provenant de dispositifs présent dans le monde réel vers le réseau internet.

L'une des grande révolution apporté par SIGFOX, c'est son accessibilité à moindre coûts, n'importe qui peut essayer de développer un mini projet avec ce réseau.

### 3.2.1.5 ) Le modem SIGFOX de chez Telecom Design (TD 1208)



*Le module TD 1208 avec son antenne*

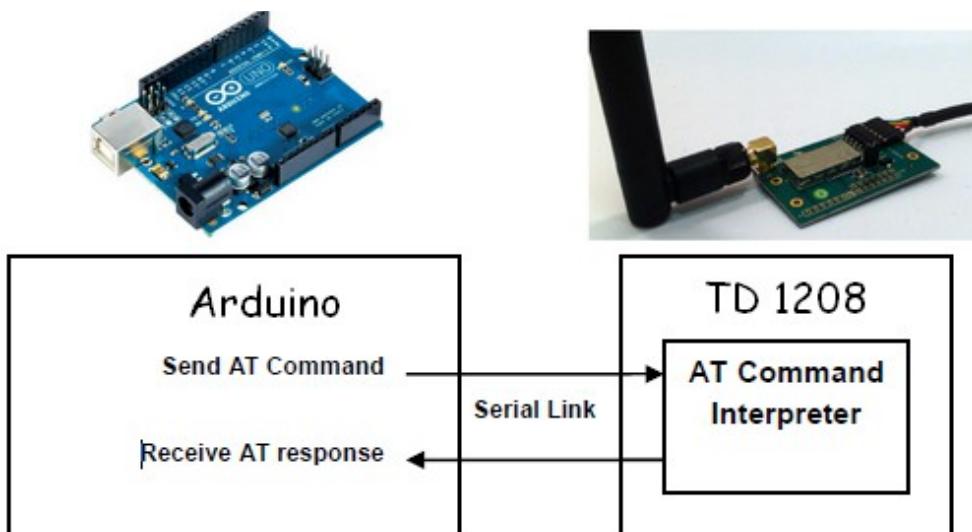
Utilisable via une liaison série à 9600 bauds (bit/s), ce modem fonctionne avec un jeu de commandes AT et communique sur une fréquence ISM 868 Mhz (le réseau SIGFOX).

Les bandes de fréquence ISM (Industriel, Scientifique et Médical) sont des fréquences utilisables pour diverses utilisations domestiques et les domaines cités précédemment (sauf communication) sans avoir à demander l'autorisation aux autorités.

Gestionnaire de liaison série intégré et paramètres par défaut (paramètres d'usine, modifiables) suivant :

- niveau électrique LVTTL
- 9600 bauds/s
- 8 bits de données
- 1 bit stop
- pas de parité

On peut représenter le fonctionnement de ce modem avec le schéma suivant :



L'Arduino envoie des commandes AT au modem, qui va les interpréter et exécuter les actions adéquates. Finalement le modem TD 1208 va retourner une réponse à l'Arduino.

### 3.2.1.6 ) Les commandes Hayes (ou AT)

AT est un langage de commande développé à l'origine pour le modem Hayes Smartmodem 300 et qui s'est ensuite retrouvé dans les modem produits.

Chaque commande commence par 'AT' suivi de la requête que l'on veut effectuer.

Ce sont des commandes que l'on peut directement envoyer au modem, sous la forme d'une ligne de texte en ASCII terminée par \r (code ASCII 13).

Le modem renvoie alors une réponse sous la même forme terminant par \r et \n (ASCII 13 et 10).

*Voici une liste d'exemples de commandes utilisable par le modem TD1208 :*

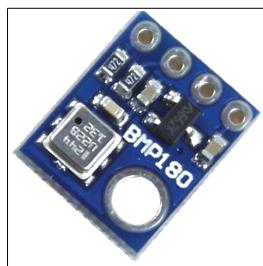
Requête	Commande	Réponse	Interprétation
Aide "Help" → ?	AT?	« Liste de commande » OK	Affiche la liste des commandes disponibles
Affichage configuration → &V	AT&V	Telecom Design TD1208 Hardware Version... ..... OK	Affiche la configuration du modem
Sauvegarder Configuration → W	AT&W	OK	Sauvegarde la configuration dans la mémoire
Envoyer SMS → \$ssms	AT\$ssms=proj et	OK	Envoie le message « projet » à un numéro configuré précédemment
Envoyer un message SIGFOX → \$SS (12 octets max)	AT\$SS=0D 0A 0B	OK	Envoie le message « 0D 0A 0B » sur le réseau SIGFOX  Très important pour envoyer les données météo avec le modem

```
at$ssms=projet
OK
at$ssms=BTS
OK
```

L'image ci-dessus est un screenshot d'un test de dialogue avec la commande permettant d'envoyer un sms.

On a en première ligne la commande voulue et en deuxième ligne la réponse du modem : OK

### 3.2.1.7) Le capteur de pression : BMP 180



*Le BMP180 en question*

On peut dialoguer avec ce capteur via un BUS I2C sur les broches SDA et SCL, c'est à dire SDA pour serial data et SCL pour serial clock.

Ce capteur est de type esclave et se contente d'exécuter les ordres du maître, l'Arduino UNO.

Avec un arduino, il se trouve qu'il est facilement utilisable grâce aux librairies et exemples du web.  
En sortie on récupère les températures en °C et la pression en hPa.

### 3.2.1.8) Environnement de développement intégré (EDI) : Visual Studio

Pour développer en C/C++ j'utilise Visual Studio avec le plugin Visual Micro qui ajoute le support de l'Arduino. L'avantage de cette IDE, face à celui d'Arduino, c'est la possibilité de mettre des points d'arrêts et d'avoir une gestion simplifiée de l'arborescence de notre programme avec le listing des différentes librairies.

 A screenshot of the Arduino IDE interface. The title bar says "meteofolyst | Arduino 1.6.1". The main window shows C++ code for a sketch named "decodeur\_nmea.h". The code includes headers for stdio.h, stdlib.h, inttypes.h, string.h, math.h, SoftwareSerial.h, decodeur\_nmea/decodeur\_nmea.h, SoftwareSerial.h, Wire.h, and SFE\_BMP180.h. It defines pins for receiving (anemoRX) and transmitting (anemoTX). It initializes a SoftwareSerial object for the BMP180 at address 0x77 on pins A4 (SDA) and A5 (SCL). The code then declares variables for character reception, a buffer for NMEA phrases, and an index for the navigation table. At the bottom, it includes the SFE\_BMP180 library. The status bar at the bottom right indicates "Arduino Uno on COM4".

```

meteofolyst | Arduino 1.6.1
Fichier Édition Croquis Outils Aide
meteofolyst decodeur_nmea.h
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <string.h>
#include <math.h>
#include <SoftwareSerial.h>
#include <decodeur_nmea/decodeur_nmea.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include "SFE_BMP180.h"

#define anemoRX 2 // Broche 2 Arduino en reception
#define anemoTX 3 // Broche 3 Arduino en transmission

//BMP180_ADDR 0x77 // Définition de l'adresse i2c du module de pression pin
// A4 = SDA pin A5 = SCL
SoftwareSerial SerialAnemometre(anemoRX, anemoTX); // RX,TX

char carac_dispo = 0;
char carac_recu = 0;
int caractere = 0;
char phrase[100]; // Stockage reception NMEA
int i_nmea = 0; // indice de navigation tableau NMEA phrase

imwv_t reception_phrase_imwv;
wixdr_t reception_phrase_wixdr;
plcje_t reception_phrase_plcje;

SFE_BMP180 bmp180;

```

#### Illustration de l'IDE Arduino

```
meteofoxirist - Microsoft Visual Studio
FICHIER EDITION AFFICHAGE PROJET GÉNÉRER DÉBOUGER ÉQUIPE OUTILS TEST ARCHITECTURE ANALYSER Connexion
FENÊTRE ?
COM4 | Débogueur Windows local | Debug | Win32 | Notifications
Arduino 1.6 | Arduino Uno | ? |
Explorateur de serveurs Boîte à outils
meteofoxirist.ino | SFE_BMP180.h | decodeur_nmea.h | meteofoxirist | (Portée globale) | loop()
void loop()
{
    char etat;
    double T, P;
    //-----Partie Capteur de pression-----
    etat = bmp180.startTemperature(); // Ordonne au BMP180 de démarrer la mesure
    if (etat != 0)
    {
        delay(etat);

        etat = bmp180.getTemperature(T); // Recupere la température mesurée de
        if (etat != 0)
        {
            Serial.print("Temperature : ");
            Serial.print(T,2);
            Serial.print(" deg C, ");
        }

        etat = bmp180.startPressure(3);
        if (etat != 0)
        {
            delay(etat);

            etat = bmp180.getPressure(P, T);
            if (etat != 0)
        }
    }
}
100 %
Sortie
Afficher la sortie à partir de :
Liste d'erreurs | Sortie
Prêt
Ln 57 Col 21 Car 15 INS
Explorateur de solutions
Rechercher dans l'Explorateur de sol
Solution 'meteofoxirist' (1 projet)
meteofoxirist
_core
_libraries
decodeur_nmea
SoftwareSerial
SoftwareSerial.cpp
SoftwareSerial.h
Dépendances externes
Fichiers de ressources
Fichiers d'en-tête
.meteofoxirist.vsarduino.h
decodeur_nmea.h
SFE_BMP180.h
Fichiers sources
meteofoxirist.ino
Anal... Expl... Tea... Affic... Affic...
Propriétés
meteofoxirist Propriétés du projet
Dépendances du p
C++

```

Illustration de l'IDE visual studio

### 3.3) Implémentation

#### 3.3.1) Présentation

Dans cette partie figurent les différents morceaux de codes réalisés pour le fonctionnement de la balise. Le programme n'étant pas terminé lors de la rédaction du dossier, il n'y a pas les parties concernant la création de la trame et son envoi.

Pour l'envoie des messages, on sait qu'il y a une limite à la taille de la trame dû au réseau SIGFOX.

Le **message** ne **doit pas excéder 10 octets**, en réalité on a 12 octets de disponibles mais 2 octets sont réservés au serveur sensor qui enregistre les modem TD.

Le programme final doit donc :

- Acquérir les trames NMEA
- Identifier les trames NMEA
- Découper les trames NMEA
- Récupérer la pression
- Construire le message SIGFOX
- Envoyer le message via SIGFOX

#### 3.3.2) Les extraits de code

##### 3.3.2.1 ) Récupération des trames NMEA

```
caractere = SerialAnemometre.read();
            phrase[i_nmea] = caractere; // Rempli le tableau de chaque caractère
reçus
            i_nmea++; // incrémentation de l'indice du tableau
```

Le programme récupère les caractères lu sur la liaison série (1ere ligne) dans une variable caractère.

Ensuite les cases d'un tableau « phrase » sont remplis une par une par ces caractères grâce à l'incrémentation qui suit.

on ne voit pas jusqu'à quand/quel critère on ajoute des caractères dans phrase[]

### 3.3.2.2 ) Identifier le type de la trame

```
#define CHECK_OK 0x00
#define NMEA_IIMWV 0x01
#define NMEA_IIMWV_STR "$IIMWV"
#define NMEA_WIXDR 0x02
#define NMEA_WIXDR_STR "$WIXDR"
#define NMEA_PLCJE 0x03
#define NMEA_PLCJE_STR "$PLCJE"

.....



uint8_t nmea_get_message_type(const char *message)
{
    uint8_t checktest = 0;
    if ((checktest = nmea_valid_checksum(message)) != CHECK_OK)
    {
        return checktest;
    }

    if (strstr(message, NMEA_IIMWV_STR) != NULL)
    {
        return NMEA_IIMWV;
    }

    if (strstr(message, NMEA_WIXDR_STR) != NULL)
    {
        return NMEA_WIXDR;
    }

    if (strstr(message, NMEA_PLCJE_STR) != NULL)
    {
        return NMEA_PLCJE;
    }

    return NMEA_UNKNOWN;
}
```

C'est la partie du programme qui permet d'identifier le type de la trame réceptionnée grâce à la fonction « strstr ». Cette fonction va en réalité « scanner » la chaîne de caractère pour trouver une correspondance, on fait alors une condition qui, en fonction de la chaîne retrouvée, retournera la résultat 0x01 pour la trame « IIMWV », 0x02 pour la trame « WIXDR » et 0x03 pour la trame « PLCJE »

### 3.3.2.3 ) Découpage de la trame

```
struct IIMWV {
    // trame type: "$IIMWV,136.0,R,004.80,N,A *05"

    double angle;          // Angle du vent de 000.0° à 359.0°
    char vent_apparent;    // Vent apparent
    double vitesse;        // vitesse du vent
    char unite_vitesse;    // unité de la vitesse du vent ( N = Noeuds,
M = m/s, K = km/h )
    char status; // Status de anemometre (CV7SF) A = Correct V = Alarme
};
```

Pour le découpage des trames, on a crée des structures qui contiennent les informations.

Ici on a l'exemple d'une structure IIMWV, donc celle qui contient les informations du vent qui est alimentée par le code qui suit.

```
void nmea_parse_iimwv(char *nmea, iimwv_t *message) // préambule NMEA de vent
{
    char *p = nmea;
    p = strchr(p, ',') + 1;
    message->angle = atof(p); // On récupère l'angle du vent

    p = strchr(p, ',') + 1;
    message->vent_apparent = 'R'; // On récupère les données de vent apparent

    p = strchr(p, ',') + 1;
    message->vitesse = atof(p); // On récupère la vitesse

    p = strchr(p, ',') + 1;
    if (*p == 'N')
        message->unite_vitesse = 'N'; // Noeuds
    if (*p == 'M')
        message->unite_vitesse = 'M'; // M/s
    if (*p == 'K')
        message->unite_vitesse = 'K'; // Km/h

    p = strchr(p, ',') + 1;
    if (*p == 'A')
        message->status = 'A'; // Status de CV7SF Correct
    else
        message->status = 'V'; // Status de CV7SF en alarme
}
```

Le principe utilisé pour le découpage d'une trame est simple, les trames NMEA étant composées de termes séparés par des virgules, si on décide de récupérer chaque information de la trame, on doit alors rechercher ces virgules pour récupérer ce qui la succède (d'où le +1 qui suit chaque appel de strchr) on affecte alors à une structure les données récupérées pour les réutiliser après.

### 3.3.2.4 ) Récupération de la pression

```

etat = bmp180.startPressure(3);
    if (etat != 0) // Vérifie si la mesure est possible
    {
        delay(etat); // Attend le temps de la mesure

        etat = bmp180.getPressure(P, T);
        if (etat != 0)
        {
            Serial.print("Pression absolue: ");
            Serial.print(P, 2); // Affiche la pression, en mbar ,
                                // sachant que 1 mbar = 1 hPa
            Serial.println(" hPa, ");
            Serial.println("");
        }
    }
}

```

Pour utiliser le capteur de pression une librairie est fournie sur le web avec une multitude de fonctions prêtes à être utilisées.

On va donc utiliser la fonction **startPressure** qui lance la requête de mesure et retourne le nombre de milliseconde à attendre dans « **etat** » en cas de réussite, dans le cas contraire c'est un 0 qui est retourné.

On fait donc un test de « **etat** » et si il est différent de 0, cela veut dire que la mesure est en cours et va durée le temps de la valeur de la variable « **etat** » d'où le délai qui suit pour être sur que la mesure ai eu le temps de se dérouler.

On veut ensuite récupérer la mesure avec **getPressure** puis l'afficher sur la liaison série.

Si il y a réussite (« **etat** » vaudra 1) et donc on peut afficher la mesure si il y a un échec (« **etat** » vaudra 0) et il ne se passera rien.

Pour la futur transmission via SIGFOX la mesure sera récupérée dans une variable et ajoutée à la construction de la trame à envoyer.

### 3.3.2.5 ) Construire le messages

PARTIE PAS ENCORE REALISEE DANS PROGRAMME

### 3.3.2.6 ) Envoyer le message

PARTIE PAS ENCORE REALISEE DANS PROGRAMME

### 3.4) Bilan Personnel technique

Ce projet est intéressant, j'ai pu découvrir l'internet des objets et le réseau SIGFOX qui, selon moi, permet, et permettra de réaliser beaucoup de projets, il y a pas mal de potentiel.

J'ai eu quelques difficultés dû à la partie matérielle (convertisseurs défectueux et erreur de montage) ce qui m'a fait perdre du temps.

Je n'ai pas encore fini la programmation de la balise, il me reste la partie construction et envoie du message SIGFOX et quelques améliorations/résolutions de problèmes sur la récupération des trames NMEA.

## 4) E3 : CAYUELA Julien - Collecte des données météorologiques

### 4.1) Description de la partie personnelle

Ma partie va consister à :

- récupérer les données envoyées par les modules via les serveurs Sigfox et à les enregistrer dans une base de données.

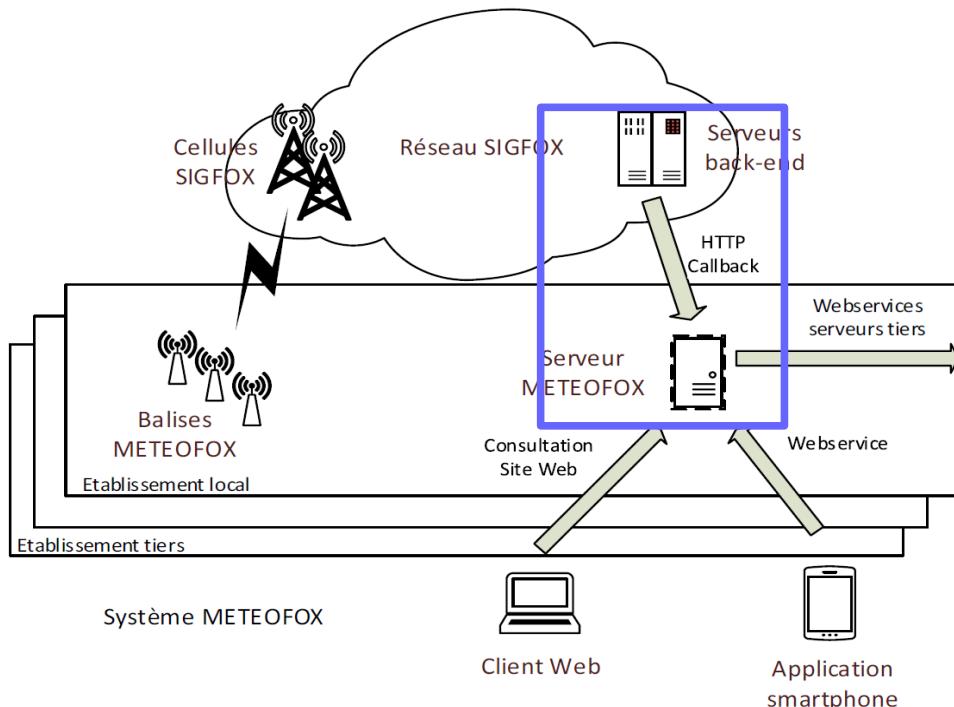


Figure 1 : synoptique du système METEOFOX

- Enregistrer notre application chez Telecom Design.
- Agréger plusieurs échantillons en un échantillon synthétique afin de pourvoir exploiter les mesures sur de longues périodes.

Lorsqu'un module envoie un échantillon de données météorologiques (toutes les 15 mins \*) au serveur Sigfox, celui-ci effectue un « callback » vers notre serveur METEOFOX avec une requête http, c'est-à-dire qu'il va appeler notre serveur, pour lui fournir l'échantillon.

*\*Rappelons qu'une puce SIGFOX peut émettre 150 messages par jour, et que chaque message peut contenir 12 octets maximum. Cela m'a permis de déterminer la périodicité minimale d'envoi des données : toutes les 10 min. Nous avons fait le choix d'élargir cette période à 15 mins afin de conserver une marge.*

#### 4.2) Mise en œuvre

##### 4.2.1) PHP

Afin de réaliser mon projet, j'ai utilisé le langage 'PHP'



PHP est un langage interprété, exécuté du côté serveur et non du côté client

*Figure 2 : Logo PHP*

(un script écrit en javascript s'exécute sur votre ordinateur...). La syntaxe du langage provient de celle du langage C, du Perl et de Java. Ses principaux atouts sont :

- Une grande communauté de développeurs partageant des centaines de milliers d'exemples de script PHP ;
- La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL) ;
- La simplicité d'écriture de scripts ;
- La possibilité d'inclure le script PHP au sein d'une page HTML;
- La simplicité d'interfaçage avec des bases de données



##### 4.2.2) JSON

*Figure 3 : Logo JSON*

**JSON (JavaScript Object Notation)** est un format de données textuelles dérivé de la notation des objets du langage Javascript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

un exemple aurait été intéressant

##### 4.2.3) WAMP Server

Afin de pouvoir faire fonctionner symfony localement sur notre poste de travail, j'ai installé WampServer.



*Figure 4 : Logo WAMP*

WampServer est une plate-forme de développement Web sous Windows pour des applications Web dynamiques à l'aide du serveur Apache2, du langage de scripts PHP et d'une base de données MySQL. Il possède également PHPMyAdmin pour gérer plus facilement nos bases de données.



*Figure 5 : Logo phpMyAdmin*



*Figure 6 : Logo Apache*



*Figure 7 : Logo MySQL*

#### 4.2.4) Symfony2

*Afin de mener à bien ce projet, j'ai utilisé le Framework symfony*

*Lancé en 2005, Symfony est aujourd'hui un framework stable connu et reconnu à l'international. Symfony dispose aussi une communauté active de développeurs, intégrateurs, utilisateurs et d'autres contributeurs qui participent à l'enrichissement continu de l'outil.*

#### **Qui est derrière tout ça ?**

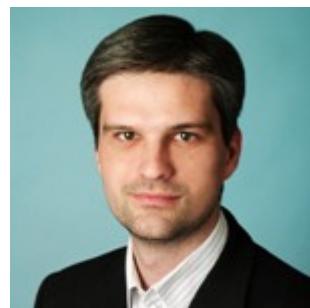
*Derrière Symfony se cache une entreprise : Sensio. Sensio est une agence web créée il y a 12 ans. Oui, ce sont bien des français qui sont derrière Symfony, plus particulièrement Fabien Potencier et toute son équipe. Imaginé au départ pour ses propres besoins, le framework Symfony est aujourd'hui encore l'outil utilisé au quotidien par ses propres équipes. Symfony est un outil qui arrive à répondre aux exigences du monde professionnel.*

copié/collé d'openclassroom



**Symfony**

*Figure 8 : Logo Symfony*



*Figure 9:Fabien Potencier  
Co-fondateur de symfony*

Symfony2 est un framework codé en PHP qui va nous permettre de développer une immensité d'applications web de la plus simple à la plus complexe imaginable.

Car en effet, Symfony définit un squelette de base à respecter, mais dans lequel on peut mettre tout le code que l'on souhaite.

*L'avantage d'un framework, c'est aussi de pouvoir travailler dans le respect des bonnes pratiques et de réutiliser des méthodologies de conception standardisées.*

*On peut particulièrement citer le design pattern MVC, acronyme de l'expression "**Modèle - Vue - Contrôleur**". Un design pattern est traduit par "patron de conception", c'est donc un modèle qui a fait ses preuves et qui s'avère indispensable à connaître dans le domaine de la programmation.*

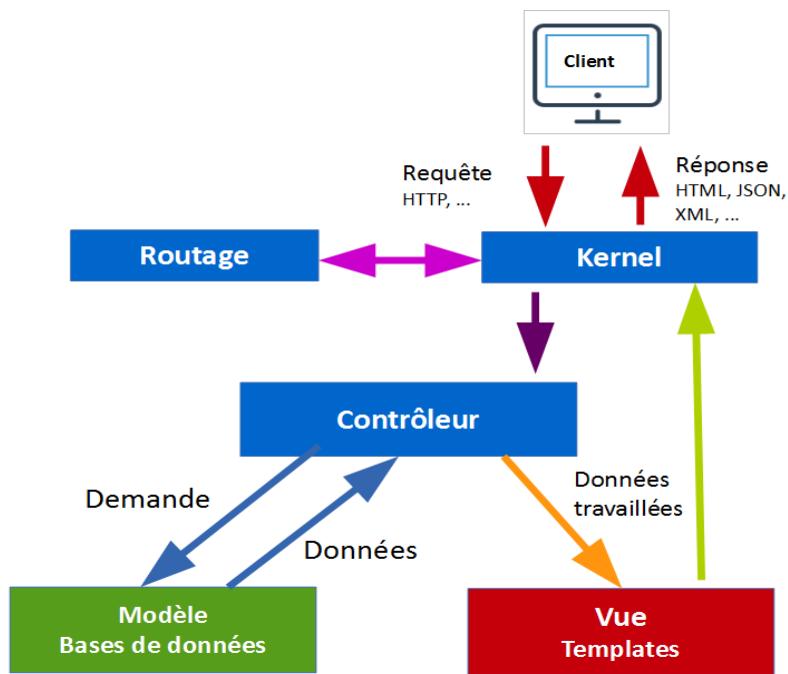


Figure 10 : Modèle de conception de symfony

Découpons un peu tout ça :

- **Kernel** : C'est le noyau de Symfony qui contient les fichiers de configuration et de création de bundles. Il fait appel au routeur pour acheminer les requêtes des clients.
- **Routeur** : Il détermine quel contrôleur exécuter en fonction de l'URL appelée. Cela permet de configurer son application pour avoir de très belles URL, ce qui est important pour le référencement et même pour le confort des visiteurs.
- **Contrôleur** : Le contrôleur est le centre de notre *design pattern*. Il reçoit la requête HTTP acheminé par le kernel, l'interprète et coordonne le tout. Il se charge de demander au modèle les données puis effectue plus ou moins de traitements dessus afin d'envoyer à la vue les données à afficher et de retourner une réponse à l'émetteur de la requête.
- **Modèle** : Le modèle est la partie du code qui se charge d'interagir avec la base de données.
- **Vue** : La vue est la partie du code qui se charge uniquement d'afficher les données qui lui ont été fournies. Nous utilisons un moteur de template twig.

Comme c'était la première fois que j'utilisais Symfony, j'ai donc suivi un tutoriel qui m'a fait travailler les principes de bases de ce nouveau Framework, avec notamment Doctrine 2.

Voici ce que j'ai effectué principalement :

- Génération d'un bundle avec la commande suivante :

```
php app/console generate:bundle --namespace=Toto/LalaBundle
```

Un Bundle est, pour faire simple, un ensemble de fichiers et répertoires permettant d'implémenter une ou plusieurs fonctionnalités.

- Etude du routeur.
- Génération d'un contrôleur / méthode / vue  
Ce contrôleur contient des méthodes.
- Étude des entités (ORM *doctrine*)

L'objectif d'un ORM (pour *Object-Relation Mapper*, soit en français « lieu objet-relation ») est de se charger de l'enregistrement des données en faisant oublier que nous avons une base de données. Il s'occupe de tout ! Nous n'allons plus écrire de requêtes, ni créer de tables via phpMyAdmin.

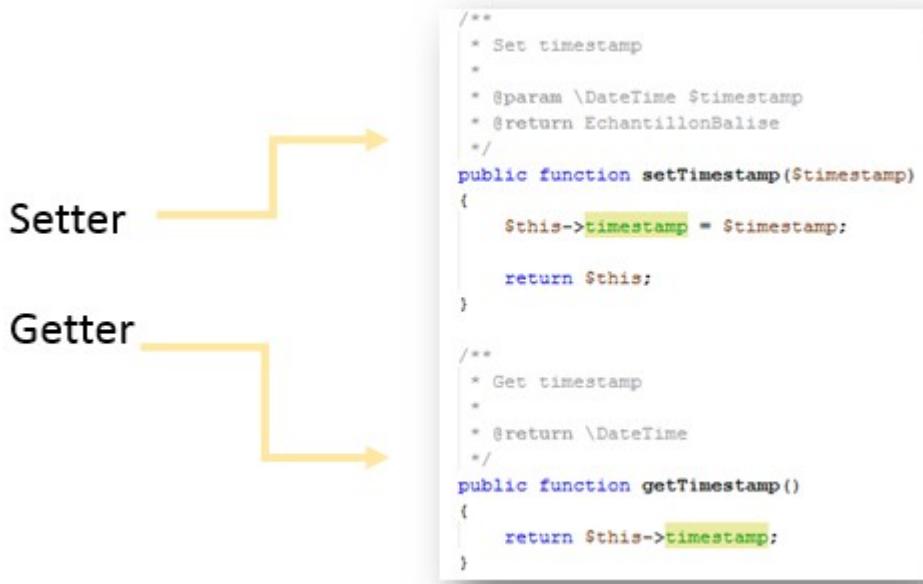
Dans notre code PHP, nous allons faire appel à *Doctrine2*, l'ORM par défaut de Symfony2, pour faire tout cela.

Une entité, ce que l'ORM va manipuler et enregistrer dans la base de données, ce n'est vraiment rien d'autre qu'un simple objet.



*La méthode « get » permet la récupération des données à la différence de « set » qui permet de mettre en mémoire l'objet avant qu'il soit 'persisté' (enregistré dans la base de donnée).*

Un exemple ci-dessous :



#### Relations entre entités :

➤ Types de relation :

Il y a plusieurs façons de lier des entités entre elles.

- ◆ One-To-One

La relation *One-To-One*, ou **1-1**, est assez classique. Elle correspond, comme son nom l'indique, à une relation unique entre deux objets.

- ◆ Many-To-One

La relation *Many-To-One*, ou **n-1**, est assez classique également. Elle correspond comme son nom l'indique, à une relation qui permet à une entité A d'avoir plusieurs relations avec plusieurs entités B.

- ◆ Many-To-Many

il n'y a pas de relation plus 'classique' qu'une autre !

cette notion de 'propriétaire' est propre à l'ORM.  
Elle n'existe pas dans le modèle entité/relation strict.

Dans une relation entre deux entités, il y a toujours une entité dite **propriétaire**, et une dite **inverse**. L'entité propriétaire est celle qui contient la référence à l'autre entité.

➤ Sens de relation :

- ◆ Unidirectionnelle



- ◆ Bidirectionnelle



### 4.3) Conception détaillée

#### 4.3.1) Diagramme de cas d'utilisation (spécification)

il me semblait pourtant vous avoir dit que les C.U. ne font pas partie de la conception détaillée

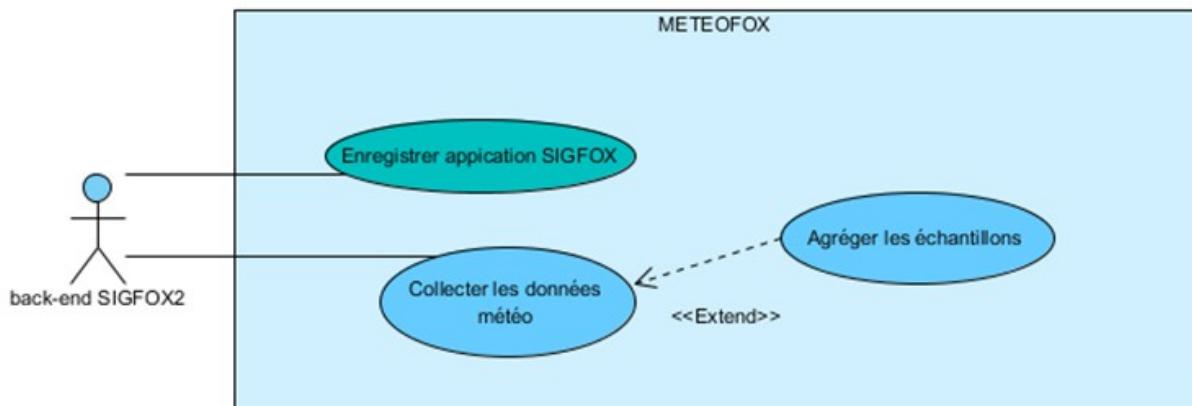


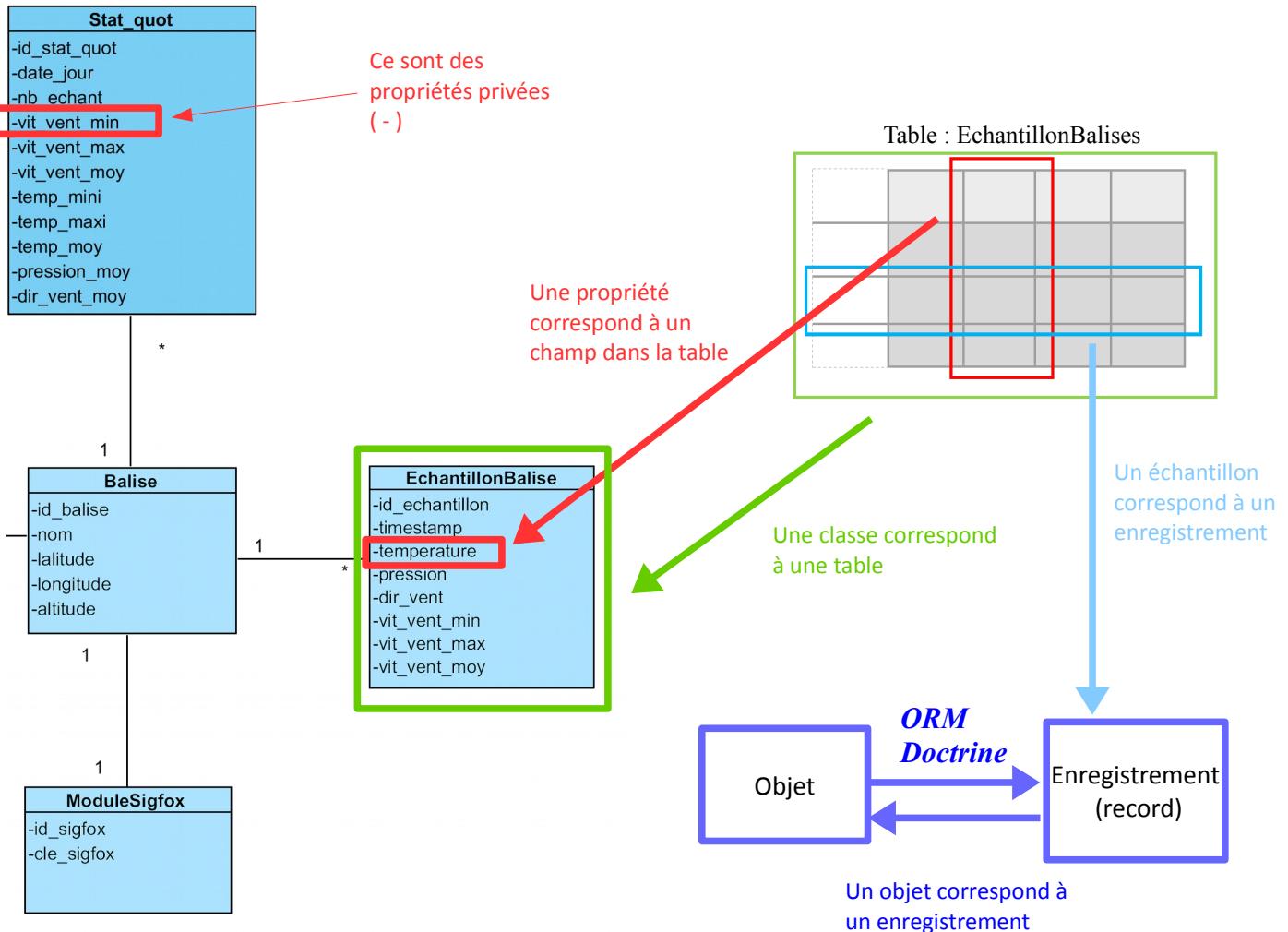
Figure 11 : Diagramme de cas d'utilisations

Cas d'utilisation	Description	Contraintes
<b>Enregistrer application SIGFOX</b>	Enregistrer le serveur METOFOX sur le back-end SIGFOX, et l'associer aux modules SIGFOX des balises, afin de recevoir les callbacks déclenchés par les messages SIGFOX	Créer un compte développeur Telecom Design.
<b>Collecter les données météo</b>	Recevoir les messages envoyés par les balises METOFOX et enregistrer les données dans la base de données	Webservice HTTP/JSON
<b>Agréger les échantillons</b>	Afin de faciliter la conservation et l'exploitation des données sur de longues périodes, on agrège plusieurs échantillons pour former un échantillon synthétique sur la période correspondante.	- calculer les valeurs statistiques (min, max, moyenne)

### 4.3.2) Diagramme de classes d'entités

Au démarrage de notre projet, nous avons dû définir les données persistantes.

Lien entre le diagramme de classe et la base de données :



A partir de ce diagramme j'ai pu créer mes entités.

Chaque balise :

- est liée à un module.
- peut avoir plusieurs échantillons ( un échantillon ne peut appartenir qu'à une seule balise)
- peut avoir plusieurs statistiques ( une statistique ne peut appartenir qu'à une seule balise)

→ Des statistiques sont effectuées sur une période de 24h (donc ~365 statistiques/an/balise ).

#### 4.4) Implémentation

##### 4.4.1) Création des entités

J'ai débuté ce projet par la création des entités associées donc au diagramme de classes.

Avant toute chose, j'ai configuré l'accès à la base de données dans Symfony2. Pour cela, il m'a suffit d'ouvrir le fichier app/config/parameters.yml et de mettre les bonnes valeurs aux lignes commençant par database\_ : serveur, nom de la base, nom d'utilisateur et mot de passe.

Il faut créer la base de données. Pour cela, il faut exécuter la commande :

```
C:\wamp\www\meteofox>php app/console doctrine:database:create
```

On peut ensuite créer nos entités avec la commande suivante :

```
C:\wamp\www\meteofox>php app/console generate:doctrine:entity
```

J'ai donc créé 4 entités :

- Balise - EchantillonBalise - ModuleSigfox - Stat\_quot

Une fois que ses entités sont créées, il faut les synchroniser avec la base de données pour y créer la ou les tables associées.

```
C:\wamp\www\meteofox>php app/console doctrine:schema:update --dump-sql
```

Cette dernière commande est vraiment performante. Elle va comparer l'état actuel de la base de données avec ce qu'elle devrait être en tenant compte de toutes nos entités. Puis, elle affiche les requêtes SQL à exécuter pour passer de l'état actuel au nouvel état.

Pour l'instant, rien n'a été fait en base de données, Doctrine nous a seulement affiché la ou les requêtes qu'il s'apprête à exécuter. Mais maintenant, il est temps de passer aux choses sérieuses, et d'exécuter concrètement cette requête avec la commande suivante :

```
C:\wamp\www\Symfony>php app/console doctrine:schema:update --force
```

Il n'y a plus qu'à vérifier le résultat dans phpmyadmin :

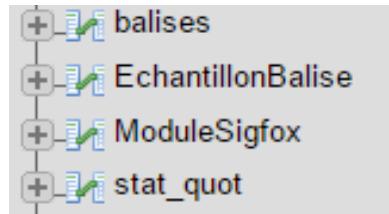


Figure 12 : Tables présentes dans la Base de données

J'ai ensuite effectué :

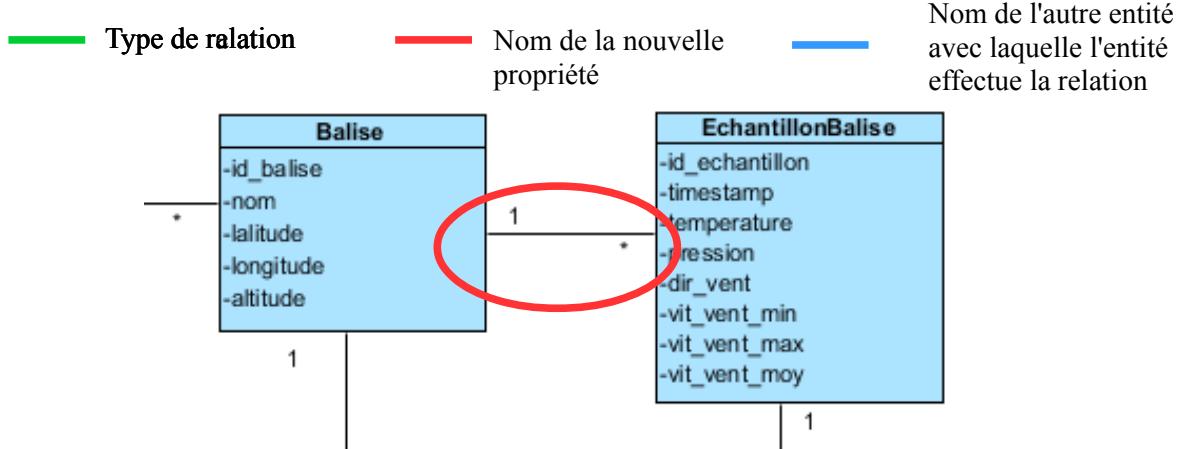
- une relation **Many-To-One Bidirectionnelle** entre l'entité Balise et l'entité EchantillonBalise

```
class EchantillonBalise
{
    //association many-to-one bidirectionnelle entre l'entité "balise" et "EchantillonBalise"
    /**
     * @ORM\ManyToOne(targetEntity="Balise", inversedBy="echantillons", fetch="EXTRA_LAZY")
     * L'annotation suivante évite de sérialiser la balise liée :
     * @Exclude
     */
    private $balise;
}
```

Entité propriétaire : EchantillonBalise

```
class Balise
{
    //association many-to-one bidirectionnelle entre l'entité "balise" et "EchantillonBalise"
    // l'option fetch="EXTRA_LAZY" permet de ne charger les échantillons qu'à la demande
    /**
     * @ORM\OneToOne(targetEntity="EchantillonBalise", mappedBy="balise", fetch="EXTRA_LAZY")
     * @ORM\OrderBy({"timestamp" = "ASC"})
     * L'annotation suivante évite de sérialiser la collection d'échantillons liée :
     * @Exclude
     */
    private $echantillons;
    // ...
}
```

Entité inverse : Balise



- une relation **One-To-One Bidirectionnelle** entre l'entité Balise et l'entité ModuleSigfox

```
//association one-to-one bidirectionnelle avec l'entité ModuleSigfox
/**
 * @ORM\OneToOne(targetEntity="ModuleSigfox", inversedBy="balise")
 */
private $module;
```

Entité propriétaire : Balise

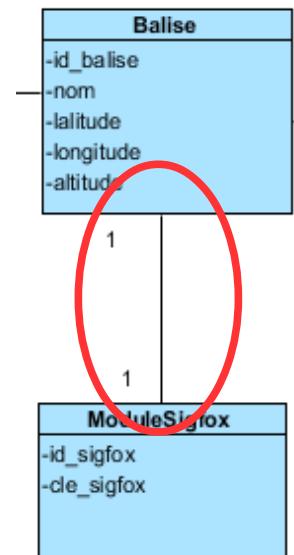
```
class ModuleSigfox
{
    //association one-to-one bidirectionnelle entre l'entité "balise" et "ModuleSigfox"
    /**
     * @ORM\OneToOne(targetEntity="Balise", mappedBy="module")
     */
    private $balise;
```

Entité inverse : ModuleSigfox

 Type de relation

 Nom de la nouvelle propriété

 Nom de l'autre entité avec laquelle l'entité effectue la relation



#### 4.4.2) Enregistrement de l'application

Afin de pouvoir recevoir les callback envoyées par les serveurs SIGFOX, j'ai dû enregistrer notre application sur le site des développeurs de TelecomDesign (TelecomDesign étant l'entreprise qui fabrique les modules) à l'adresse suivante :

<https://developers.insgroup.fr/dashboards/developer.html>



Figure 13 : Logo TELECOMDesign

**Application : meteofox.lyceekastler.fr**

Details	Modules
Name	meteofox.lyceekastler.fr *
Callback Url	http://meteofox.lyceekastler.fr/meteofox/web/app_dev.php
Description	Envoi des trames SIGFOX vers notre serveur.
Created	Jan 22, 2015 1:27:36 PM
Key	e9c3cc61902e5973fe482f17cca6d70b
Secret	39dc0994d0e1ffcec8de64800aca455b

**Save** **Cancel**

Figure 14 : Formulaire de création d'une nouvelle application

J'y ai précisé l'URL de callback, c'est-à-dire l'url que les serveurs SIGFOX vont « appeler ». J'y ai précisé la route vers la méthode « addechantillon » de mon contrôleur :

[http://meteofox.lyceekastler.fr/meteofox/web/app\\_dev.php/addechantillon](http://meteofox.lyceekastler.fr/meteofox/web/app_dev.php/addechantillon)

#### 4.4.3) Test : Réception callbacks

J'ai codé en php un petit script de test qui m'a permis d'attester de la bonne réception des callbacks.

Le site de TelecomDesign dispose d'un simulateur de callback dont vous pouvez voir un aperçu ci-dessous :

The screenshot shows a web-based application for testing IoT callbacks. At the top, there are tabs: Applications, Registered Modules, IOT Callbacks Simulator (which is selected), UDM Frame Decoder, and My Account. Below the tabs, there are two input fields: 'Callback url' containing 'http://meteofox.lyceekastler.fr/meteofox' and 'Frequency' set to '5 sec'. There are three buttons: 'Start / Stop' (blue), '0' (disabled), and 'Clear Messages'. Below these controls is a section titled 'Callback Messages' with a table. The table has columns: Status, Date, Url, and Callback. One row is shown, highlighted with a blue border: 'OK' status, date '02-05-2015 14:36:14', URL 'http://meteofox.lyceekastler.fr/meteofox/web/app\_dev.php/addechantillon', and a JSON callback message: {"id": "48801", "date": "2015-05-02T14:36:14Z"}. A blue arrow points from the text 'Ici, c'est l'historique des "callback" envoyés' up towards this table row.

*Figure 15 : Simulateur de callbacks*

Ici, c'est l'historique des 'callback' envoyés

Le script qui suit va réceptionner les trames envoyées par les serveurs SIGFOX, les imprimer dans le fichier d'erreur du serveur ;

```
// Reception de toute la trame envoyées des serveurs SIGFOX
$requete = file_get_contents('php://input');

// impression de la requête dans le fichier d'erreur du serveur
error_log("Contenu de la requête : " . $requete);

// N° de l'appli
$appkey = $_POST['appkey'];
error_log('$appkey : ' . print_r($appkey,true) );
```

[Fri May 08 20:10:25.821410 2015] [:error] [pid 19059] [client 95.142.174.228:43344] Contenu de la requête
:
callback=%7B%22id%22%3A%2259651%22%2C%22timestamp%22%3A%22May+8%2C+2015+6%3A08%3A35+PM%22%2C%22msg%22%3A%7B%
22when%22%3A1431108515599%2C%22received%22%3A1431108515599%2C%22l1%22%3A%2226.96%22%2C%22rss1%22%3A%22-60.1
9%22%2C%22station%22%3A%222698%22%2C%22lat%22%3A%2247.50%22%2C%22long%22%3A%22-10.16%22%2C%22avg1v1%22%3A%224
0.37%22%2C%22crxrt%22%3A%7B%22id%22%3A%2262758%22%2C%22module%22%3A%2226FR%22%2C%22serial%22%3A%2226FR%2F15%2

Il va aussi imprimer le numéro de l'application afin de vérifier que le numéro qu'il nous envoie correspond à celui que nous avons reçu.

App Key  
61936a046eb1c7238ba03c706bd1ceba

[Fri May 08 20:10:25.821488 2015] [:error] [pid 19059] [client 95.142.174.228:43344] \$appkey :

#### 4.4.4) Réception et injection des données dans la Base de données

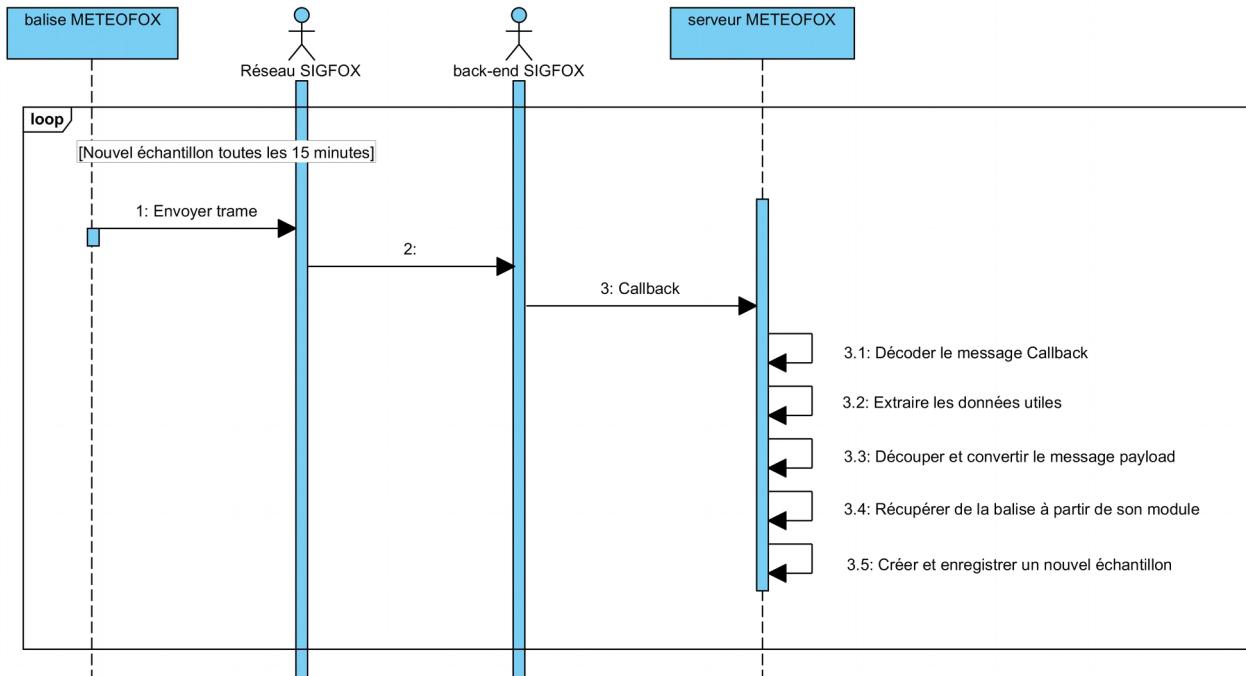


Figure 16 : Diagramme de séquence : Réception et enregistrement d'un échantillon

Par la suite je vais développer le scénario de ce diagramme de séquence par des parties de code pertinentes.

Lorsque la balise va envoyer une trame sur les serveurs Sigfox via le réseau Sigfox. Eux même vont rediriger cette trame via callback (requête http) vers notre serveur Meteofox.

##### 3.1 : Décoder message callback

Cette trame, codée en format JSON, est décodée avec la fonction json\_decode()

```
// l'objet callback est décodé de JSON
$callback = json_decode($_POST['callback']);
```

##### 3.2 : Extraire les données utiles

On obtient une suite de données arborées au format JSON, où on va y extraire les données utiles :

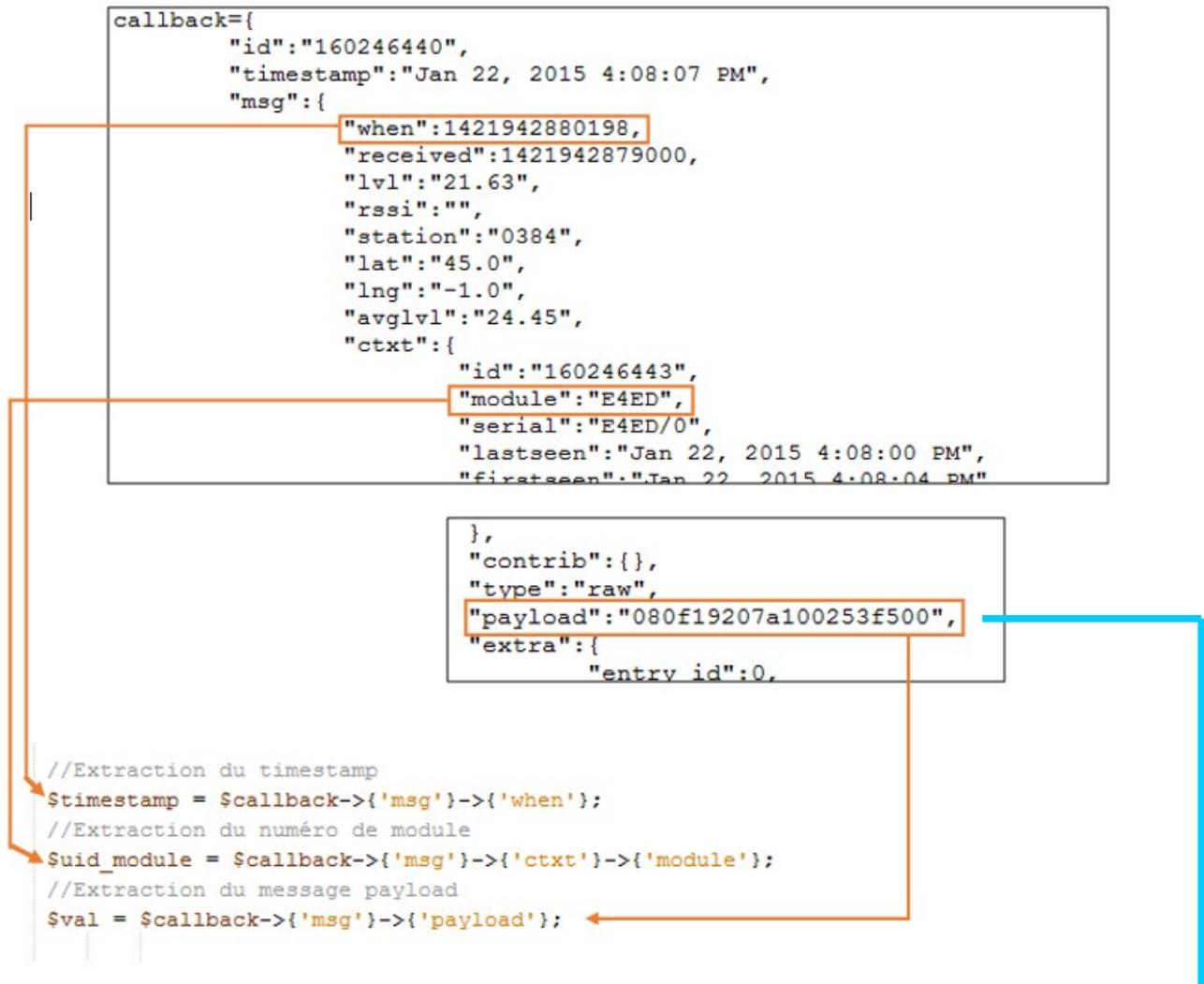
- **when** (timestamp) : Le moment où a été envoyé l'échantillon météorologique

*Le timestamp (horodateur en français) est le nombre de secondes écoulées depuis le 1<sup>er</sup> Janvier 1970 à 00h*

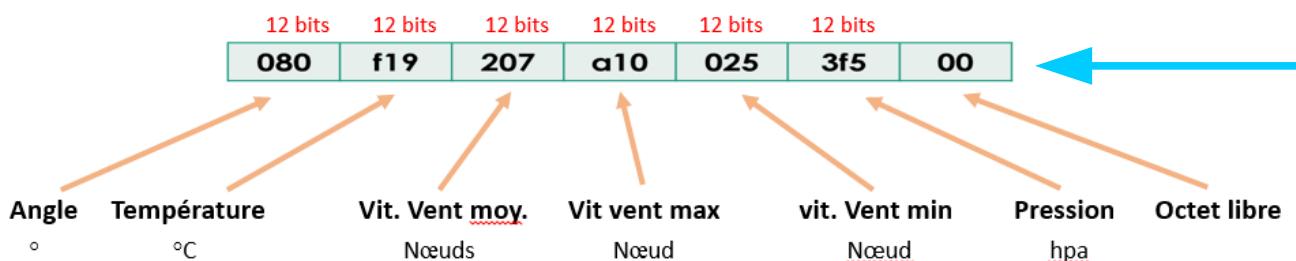
- **module** : Le numéro du module qui a envoyé l'échantillon météorologique

- **payload** : Message de 10 octets qui contient les données météorologiques

**NON : \$callback n'est plus au format JSON, puisque JSON a été décodé !!**  
**C'est un tableau associatif PHP, qui peut contenir des tableaux associatifs imbriqués (d'où le côté 'arboré').**



Grâce à la documentation technique de Sigfox, on sait que l'on ne peut envoyer que 12 octets à travers le réseau Sigfox. En réalité, 10 octets sont transmis par notre module car deux octets sont réservés aux serveurs SENSOR.



Trois quartets de bits (12 bits) correspondent à une mesure .

vous n'expliquez pas clairement que hex12decsigned convertit des blocs de 3 chiffres hexa

### 3.2 : Découper et convertir le message payload

J'ai donc codé une fonction qui , tout d'abord, convertit des caractère hexadécimaux en un entier signé, et enfin corrige le signe :

```
function hex12decsigned($h) {
    $val = hexdec($h);
    if ($val > 2047) {
        $val = $val-4096;
    } else {
        return $val;
    }
}
return $val;
```

que fait hexdec() ?

Ensuite, j'ai fait une boucle qui parcours et découpe une chaîne en trois quartets de bits. Chaque bloc de trois quartets est convertit avec la fonction hex12decsigned().

A chaque fois que la boucle fait un tour un nom de variable est associée une mesure

```
for($i = 0 ; $i < 20; $i=$i+3){

    $hex_val = substr($val, $i,3);
    $dec_val = hex12decsigned($hex_val);

    ${'val'.'$i'} = $dec_val;
}
```

### 3.2 : Récupérer la balise à partir de son module

Comme on souhaite associer l'id de balise à son échantillon, j'ai dû trouver un moyen de retrouver l'id de la balise à partir de son module. Cela a été possible grâce aux relations entre entités.

```
// Récupération de la balise dont on connaît l'uid
// obtient l'entité module
$em = $this->getDoctrine()->getEntityManager();
$module = $em->getRepository("MeteofoxBundle:ModuleSigfox")->findOneByIdSigfox($uid_module);
// obtient la balise liée au module
$balise = $module->getBalise();
```

### 3.5 : Créer et enregistrer un nouvel échantillon

A partir de cela, on peut créer un nouvel échantillon,

```
// Crédation de l'échantillon
$echantillon = new EchantillonBalise();
$echantillon->setBalise($balise);           // on lie la balise à l'échantillon

$echantillon->setTimestamp($timestamp);
$echantillon->setTemperature($val3);
$echantillon->setPression($val15);
$echantillon->setVitVentMax($val9);
$echantillon->setVitVentMin($val12);
$echantillon->setVitVentMoy($val16);
$echantillon->setDirVent($val0);
```

ensuite le *persist* (l'enregistrer) et le *flush* (l'envoyer) dans la base de données.

```
$em = $this->getDoctrine()->getManager();
// Étape 1 : On « persiste » l'entité
$em->persist($echantillon);
// Étape 2 : On « flush » tout ce qui a été persisté avant
$em->flush();
```

#### 4.4.5) Agrégation des échantillons

Afin de faciliter la conservation et l'exploitation des données sur de longues périodes, je vais devoir agréger plusieurs échantillons pour former un échantillon synthétique sur la période de 24h.  
J'ai donc créer une nouvelle entité :

Stat_quot
-id_stat_quot
-date_jour
-nb_echant
-vit_vent_min
-vit_vent_max
-vit_vent_moy
-temp_mini
-temp_maxi
-temp_moy
-pression_moy
-dir_vent_moy

Malheureusement, je n'ai pas eu le temps de développer cette partie au niveau du codage...

#### 4.5) Tests de validation

Objectif du test : Vérifier que les données reçues soient bien envoyées dans la base de données.

Résultat : Les données sont bien réceptionnées et enregistrées dans la base de donnée, mais ces données ne sont pas justes.

vous pouvez être plus clair ?



	id	balise_id	timestamp	temperature	pression	dir_vent	vit_vent_min	vit_vent_max	vit_vent_moy
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier	2147483647	19	2147483647	1817	616	128	9	0	0
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier  Effacer	3	2147483647	702	-231	616	80	9	0	0
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier  Effacer	20	2147483647	702	746	176	n	n	n	n

#### 4.6) Bilan Personnel technique

J'ai bien enregistré notre application sur le site de TelecomDesign.

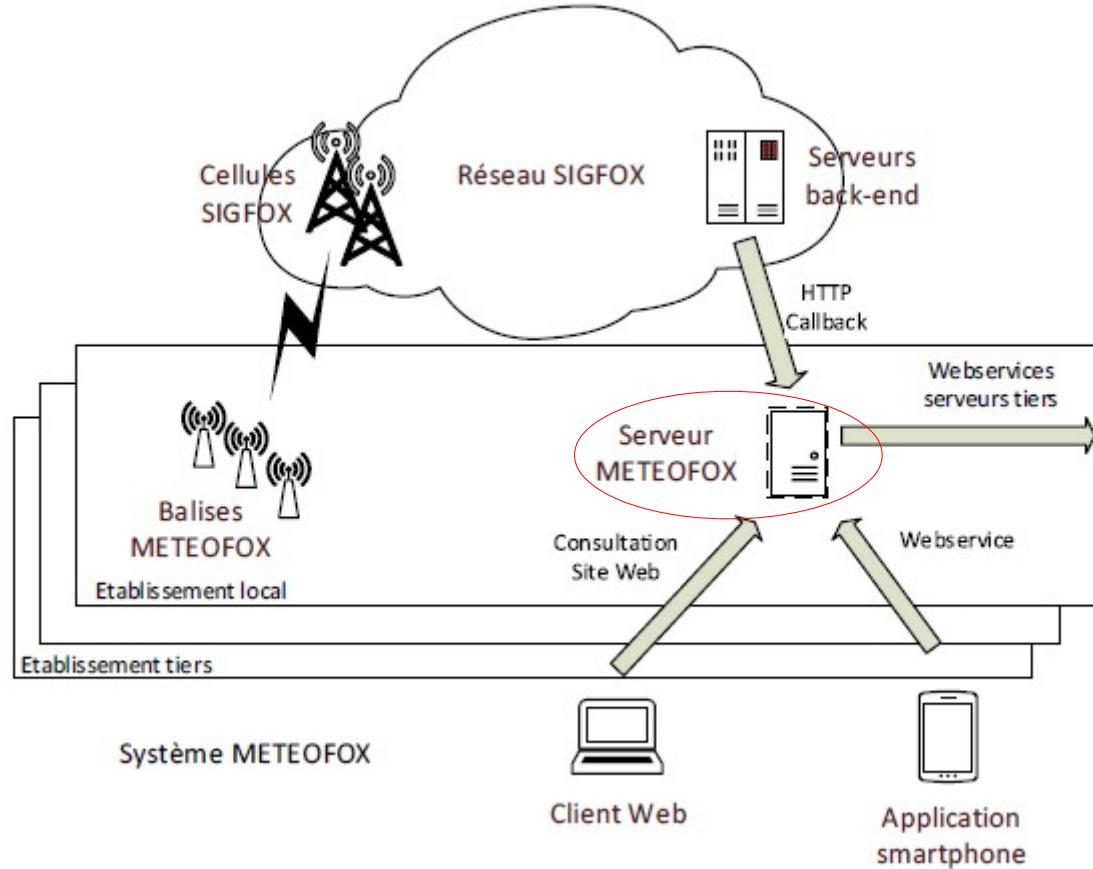
Mon programme est capable de recevoir les données envoyées par la balise, de les traiter et de les injecter dans la base de données.

Il reste maintenant à effectuer l'agrégation des échantillons en un échantillon synthétique (Partie que je n'ai pas eu le temps de développer au moment de la création de dossier).

## 5) E4 : Baptiste DESCARD - Gestion du serveur METEOFOX

### 5.1) Description de la partie personnelles

Dans le projet METEOFOX, je me situe au niveau du Serveur METEOFOX comme on peut le voir dans le schéma ci-dessous.



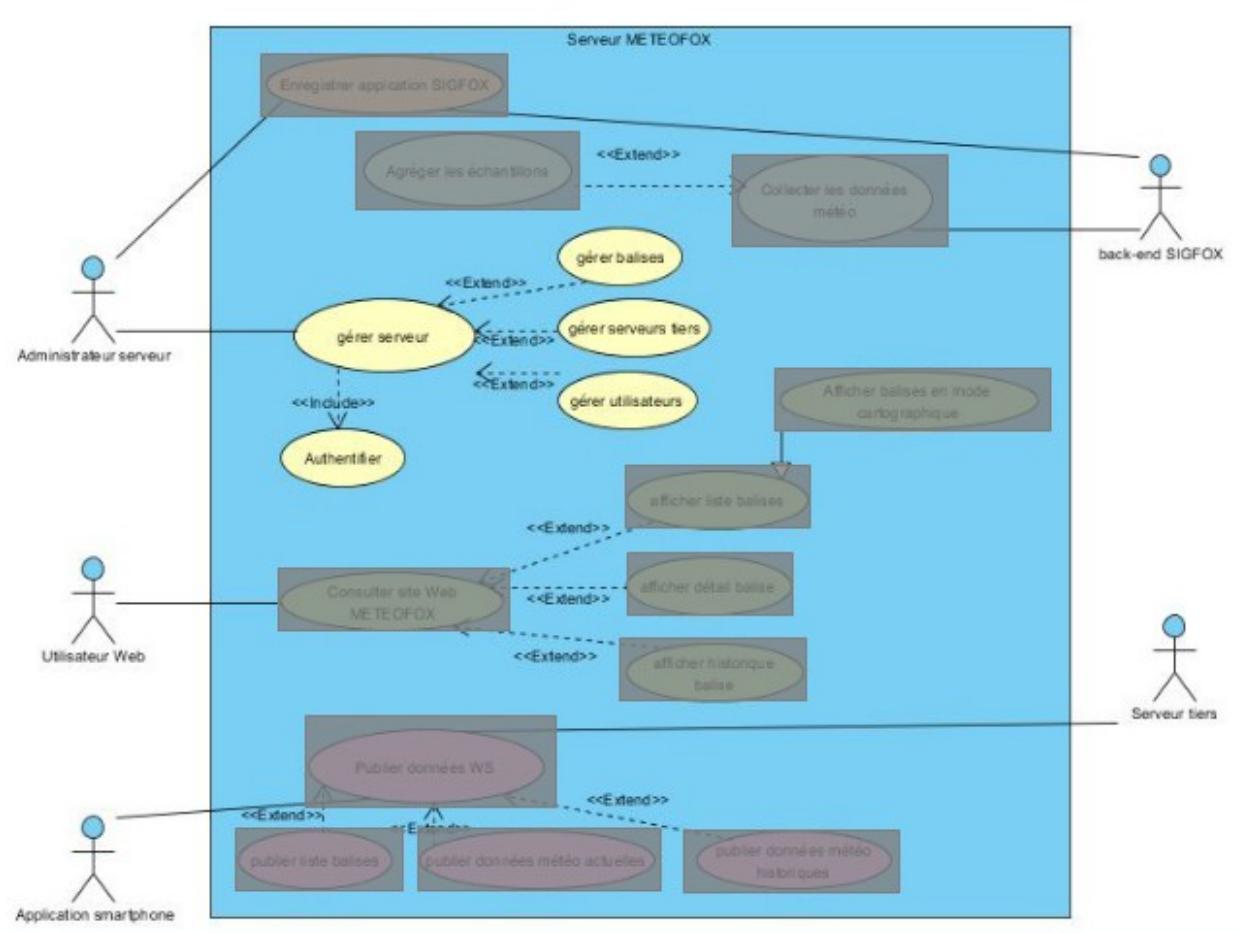
Différentes tâches m'ont été confiées :

<b>Gérer serveur</b>	détaillé dans les 3 cas d'extension suivant	Interface Web – réservée à l'administrateur.
Gérer balises	Ajouter, éditer, supprimer des balises	
Gérer serveurs tiers	Ajouter, éditer, supprimer des serveurs tiers	
Gérer utilisateurs	Ajouter, éditer, supprimer des comptes utilisateurs	Bundle de gestion des utilisateurs de SYMFONY
Authentifier	Authentification l'utilisateur	login/mot de passe

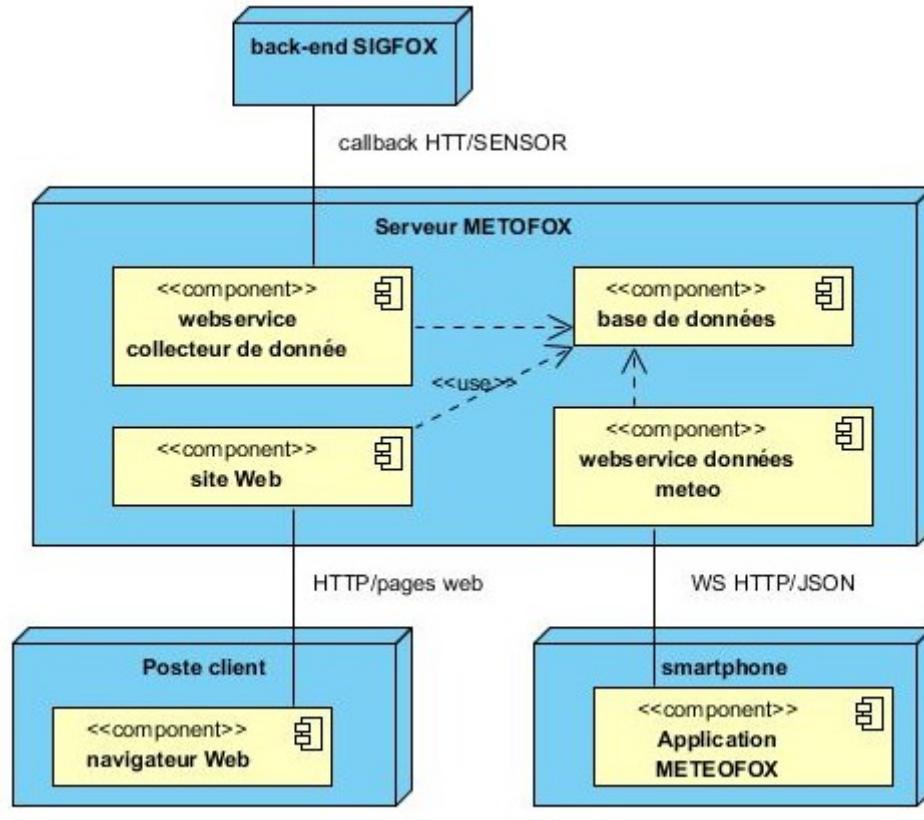
## 5.2) Conception détaillée

On va à présent avec l'aide de diagramme voir plus en détail la partie que je développe.

Dans un premier temps on peut voir avec le diagramme de cas d'utilisation du serveur web ci-dessous les différentes tâches à réaliser (non caché) que nous avons déjà évoquées plus tôt.



A présent, on trouve le diagramme de déploiement ci-dessous :



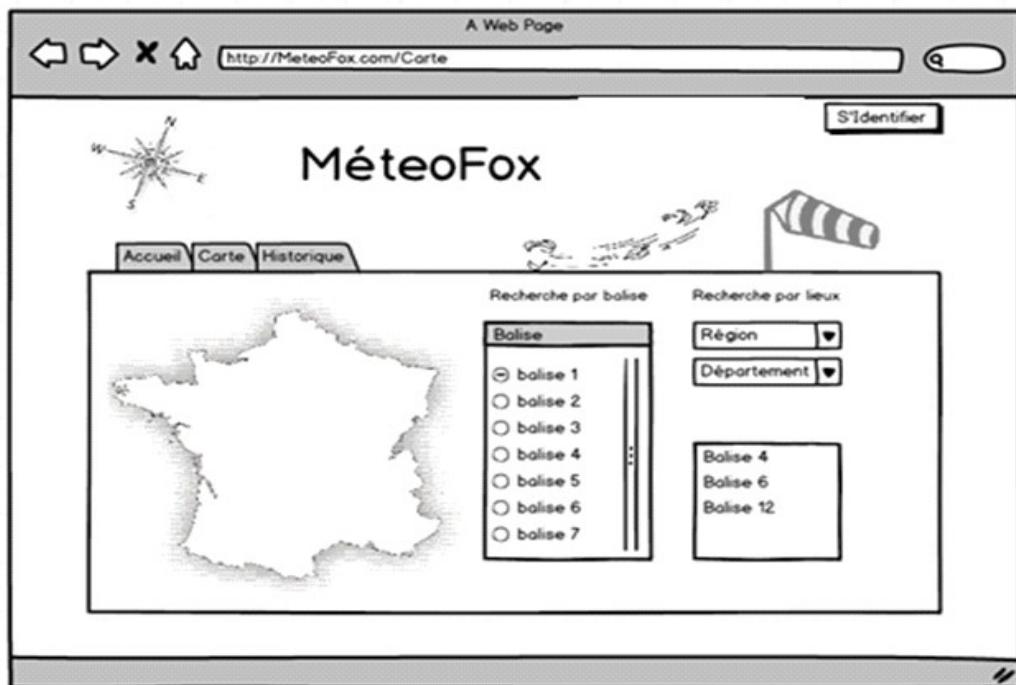
Ce diagramme nous montre le fonctionnement du serveur Météofox dans son ensemble. Pour ce qui me concerne, on voit que le site Web utilise les données de la base pour permettre aux utilisateurs d'avoir en temps réel les données des balises.

### 5.3) Mise en œuvre

#### 5.3.1) IHM

Afin d'avoir une idée de la conception du site avant de débuter sa réalisation, nous avons utilisé le logiciel Balasmiq Mockups pour dessiner son interface graphique. Nous avons pris en compte le cahier des charges et sommes parvenus à une interface qui prend en compte tous les critères. Nous retrouvons donc dans cette interface un bouton pour l'authentification, des onglets menant aux pages de la cartographie et de l'historisation où l'on retrouve nos balises.

On retrouve ci-dessous un exemple de cette ihm avec la page de la cartographie :

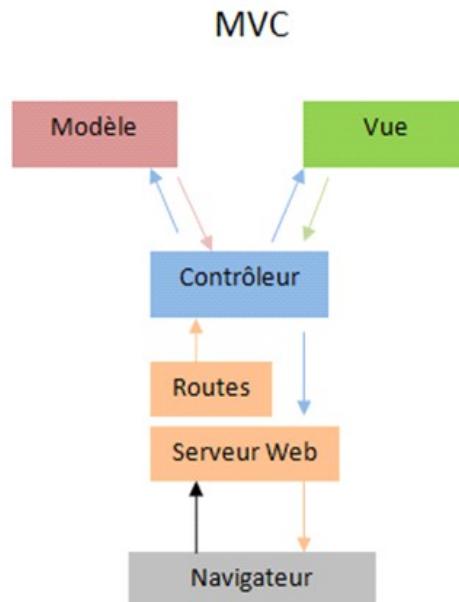


#### 5.3.2) Symfony

Pour réaliser ce serveur Web, on utilise le Framework Symfony. Ne connaissant pas ce Framework, nous avons suivis avec les personnes intéressées un tutoriel nous permettant une prise en main rapide et efficace. Ce Framework utilise le langage de programmation PHP 5.2.4, de plus, il facilite et accélère le développement de sites et d'applications Internet. Nous utilisons l'environnement de développement NetBeans pour ouvrir et modifier le code. Symfony fonctionne sous le modèle MVC.

Le modèle MVC fonctionne sous 3 couches:

- Modèle: Son rôle est de gérer les données ainsi que leurs contenus.
- Vue : Permet d'afficher les pages.
- Contrôleur : Génère la réponse à la requête HTTP demandée par notre visiteur.



Durant ce tutoriel, nous avons visé des thèmes précis:

### Les bundles:

On génère un bundle avec la commande: `php app/console generate:bundle`

Une fois le bundle généré, la portion de code ci-dessous se rajoute au projet créé:

```

irist_gest_stg1:
    resource: "@IRISTGestStg1Bundle/Controller/" // Chemin de la resource
    type:     annotation                         // Format
    prefix:   /                                  // Prefix
  
```

### Les contrôleurs et leurs méthodes:

Pour ce qui est du contrôleur, la commande pour le générer est:

`php app/console generate:controller`

Pendant la génération du contrôleur, on ajoute nos méthodes, et une fois le contrôleur créé, on retrouve ses méthodes:

```
class EntrepriseController extends Controller
{
    /**
     * @Route("/index")
     * @Template()
     */
    public function indexAction()
    {
        return array(
            // ...
        );
    }

    /**
     * @Route("/del")
     * @Template()
     */
    public function delAction()
    {
        return array(
            // ...
        );
    }

    /**
     * @Route("/add")
     * @Template()
     */
    public function addAction()
    {
        return array(
            // ...
        );
    }
}
```

On peut voir ci-dessous un exemple après avoir modifié la méthode index:

## Bundle: IRISTGestStg

- nom contrôleur: EntrepriseController
- nom méthode (action): indexAction()

"net"

## Paramètre:C2I-net

### Les Entités:

Les entités se génèrent avec la commande: [php app/console generate:doctrine:entity](#)

On crée ensuite la base de données, puis on génère et synchronise des tables dans cette base associées aux entités.

Avec l'aide de phpMyAdmin, on peut vérifier la génération de notre entité et ses modifications:

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	id	int(11)			Non	Aucune	AUTO_INCREMENT	<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Primaire</a> <a href="#">Unique</a> <a href="#">Ind</a>
2	nom	varchar(255) utf8_unicode_ci			Non	Aucune		<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Primaire</a> <a href="#">Unique</a> <a href="#">Ind</a>
3	ville	varchar(255) utf8_unicode_ci			Non	Aucune		<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Primaire</a> <a href="#">Unique</a> <a href="#">Ind</a>
4	cp	bigint(20)			Non	Aucune		<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Primaire</a> <a href="#">Unique</a> <a href="#">Ind</a>

Les Formulaires: On crée un formulaire à partir d'un objet du contrôleur où on y ajoute du code et on retrouve le formulaire voulu par exemple:

IRISTGestStg1Bundle:Entre... x +

localhost/symfony/web/app\_dev.php/addForm

Formulaire d'annonce

Nom  
Ville  
Cp  
[Ajouter](#)

[Accueil](#)

## 5.4) FOSUserBundle

Le FOSUserBundle permet l'authentification et l'autorisation des comptes utilisateurs. Il utilise des pare-feu ainsi que des formulaires de connexion. Pour installer le FOSUserBundle, il faut modifier certain fichier:

- security.yml
- config.yml
- routing.yml
- entité: UserBundle
- appKernel.php

Puis on lance la commande suivante pour installer le bundle :

```
php composer.phar update friendsofsymfony/user-bundle
```

Après création, quand un utilisateur anonyme se connecte, une figure lui indique qu'il est bien authentifié:



Grâce aux vues login et layout du FosUser, que j'utilise via un sur-chargement (avec la même héritage de dossier), je peux me connecter avec un compte utilisateur. Grâce à cette authentification, une fois reconnue et les droits d'utilisateur vérifiés, l'internaute pourra ajouter une balise grâce au formulaire prévu qui se rajoutera à sa liste de page. On retrouve ci-dessous une première vue de cette page de login :

A screenshot of a login form. At the top, there are links for 'Accueil', 'Liste complète des balises', 'Afficher la carte', and 'Ajouter une balise'. The main form area has fields for 'Identifiant' (username) and 'Mot de passe' (password). There is also a 'Se souvenir de moi' (remember me) checkbox and a 'Connexion' (login) button.

#### 5.4.1) Gérer les balises

Afin d'ajouter une balise, j'ai créé un formulaire que j'ai inséré dans une nouvelle page. Ce formulaire est codé en html et il permet de donner à sa balise, un nom ainsi que des coordonnées. On pourra ainsi retrouver par la suite cette balise sur la carte et dans la liste complète.

où est ce formulaire ??

#### 5.4.2) Mise en forme du site

Afin de retrouver la même mise en forme sur tout le site, j'utilise la vue base.html.twig. Dans cette vue, j'appelle le css et c'est ici que je place mon menu pour naviguer sur les différentes pages. Dans cette base, on retrouve aussi une partie du code du layout, qui nous permet de pouvoir nous identifier sur toute les pages.

la notion de layout n'est pas présentée

### 5.5) Implémentation

Pour revenir sur notre partie réalisation, on va à présent se plonger plus en détail dans le code de Symfony que l'on a utilisé.

Dans un premier temps, nous allons voir en ce que nous avons appris dans le tutoriel, puis nous développerons sur la réalisation du projet.

#### 5.5.1) Tutoriel:

Une fois que le contrôleur a été généré, nous avons modifié le code de ses méthodes.

- ❖ Prenons l'exemple de la méthode listAction, pour arriver à afficher:



#### Bundle: IRISTGestStg

- nom contrôleur: EntrepriseController
- nom méthode (action): listAction()

#### Liste Entreprises

- C2I | 1 | Bordeaux
- MK2 | 2 | Talence
- CxTechnology | 3 | Paris
- 6PO | 4 | tataouine

- Il a fallu ajouter un paramètre à la méthode:

```

public function listAction()
{
    $listeEntreprise = array(
array(
'nom' => 'C2I',
'id' => 1,
'Ville' => 'Bordeaux'),
array(
'nom' => 'MK2',
'id' => 2,
'Ville' => 'Talence'),
array(
'nom' => 'CxTechnology',
'id' => 3,
'Ville' => 'Paris'),
array(
'nom' => '6PO',
'id' => 4,
'Ville' => 'tataouine')
);
    return array('listeEntreprise' =>$listeEntreprise);
}

```

- Puis modifier la vue associée:

```

{# extends "::base.html.twig" #-}

{# block title #}IRISTGestStg1Bundle:Entreprise:list{# endblock #}

{# block body #}
<h1>Bundle: IRISTGestStg</h1>
<ul>
    <li>nom contrôleur: EntrepriseController</li>
    <li>nom méthode (action): listAction()</li>
</ul>

<h2> Liste Entreprises</h2>
<ul>
{# for Ent in listeEntreprise #}
    <li>{{ Ent.nom }} | {{ Ent.id }} | {{ Ent.Ville}}</li>
{# endfor #}
</ul>
{# endblock #}

```

- ❖ Prenons maintenant l'exemple de la méthode listAll qui affiche les entreprises que l'on a ajouté à la base de données grâce à la méthode add.

1. Pour ce faire nous avons modifié les paramètres de la méthode add avec le code:

```
 /**
 * @Route("/add/{nomEntreprise}/{villeEntreprise}/{cpEntreprise}")
 * @Template()
 */
public function addAction($nomEntreprise, $villeEntreprise, $cpEntreprise)
{
    $entEntreprise = new Entreprise;
    $entEntreprise->setNom ($nomEntreprise);
    $entEntreprise->setVille ($villeEntreprise);
    $entEntreprise->setCp ($cpEntreprise);

    // On récupère l'EntityEntreprise
    $em = $this->getDoctrine()->getManager();

    // Etape 1 : On "persiste" l'entité
    $em->persist($entEntreprise);

    //Etape 2 : On "flush" tout ce qui a été persisté avant
    $em->flush();

    return array('nomEntreprise'=>$nomEntreprise);
}
```

2. Modifier la vue de cette méthode add :

```
{% extends "::base.html.twig" %}

{% block title %}IRISTGestStgBundle:Entreprise:add{% endblock %}

{% block body %}
<h1>Bundle: IRISTGestStg</h1>
<ul>
    <li>nom contrôleur: EnterpriseController</li>
    <li>nom méthode (action): addAction()</li>
</ul>
{{ dump (nomEntreprise) }}
<p><h2>Paramètre:C2I-{(nomEntreprise)}</h2></p>
{% endblock %}
```

3. Ce qui affiche:

localhost/symfony/web/app\_dev.php/add/Projet/Talence/33

## Bundle: IRISTGestStg

- nom contrôleur: EnterpriseController
- nom méthode (action): addAction()

"Projet"

Paramètre:C2I-Projet

4. Une fois la méthode add achevée, modifions les paramètres de la méthode listAll :

```
 /**
 * @Route("/listAll")
 * @Template()
 */
public function listAllAction()
{
    $repository = $this
        ->getDoctrine()
        ->getManager()
        ->getRepository('IRISTGestStg1Bundle:Entreprise')

    $listeEntreprise = $repository->findAll();

    return array('listeEntreprise'=>$listeEntreprise);
}
```

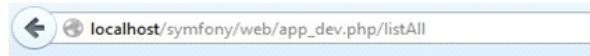
5. Ajoutons le code à sa vue :

```
{% extends "::base.html.twig" %}

{% block title %}IRISTGestStg1Bundle:Entreprise:listAll{% endblock %}

{% block body %}
<h1>Bundle : IRISTGestStg</h1>
<ul>
<li> nom contrôleur : EntrepriseController </li>
<li> nom méthode (action) : listAllAction() </li>
</ul>
<h2> Liste Entreprises : </h2>
<ul>
    {% for Ent in listeEntreprise %}
        <li>Numéro : {{Ent.id}}</li>
        <li>Entreprise :{{Ent.nom}}</li>
        <li>Ville : {{Ent.Ville}}</li>
        <li>Code Postale : {{Ent.cp}}</li>
        <p></p>
    {%endfor%}
</ul>
{% endblock %}
```

6. Une fois toutes ces étapes faites, la méthode listAll nous affiche les entreprises ajoutées à la base de données via la méthode add, ce qui nous donne:



## Bundle : IRISTGestStg

- nom contrôleur : EntrepriseController
- nom méthode (action) : listAllAction()

### Liste Entreprises :

- Numéro : 20
- Entreprise :descard
- Ville : bx
- Code Postale : 33
- Numéro : 24
- Entreprise :Descard
- Ville : Bergerac
- Code Postale : 24100
- Numéro : 28
- Entreprise :Projet
- Ville : Talence
- Code Postale : 33

dommage que tout cela concerne un exercice/tutoriel sans lien direct avec le projet meteofox

## **2-Projet:**

### **Gérer les utilisateurs:**

Pour permettre aux utilisateurs avec un compte de s'authentifier, j'ai utilisé la vue login déjà présente dans le FosUserBundle. L'utilisateur n'a plus qu'à renseigner son identifiant et son mot de passe. Les utilisateurs avec un compte sont des membres du projet, qui ont tous des rôles administrateur qui leurs permet de pouvoir ajouter ou modifier les balises. Ci-dessous ce trouve le code de la vue login du FosUserBundle avec qu'elle que modification de visuel.

```
{% if error %} ??  
    <div>{{ error.messageKey|trans(error.messageData, 'security') }}</div>  
{% endif %}  
  
<form action="{{ path("fos_user_security_check") }}" method="post">  
    <input type="hidden" name="_csrf_token" value="{{ csrf_token() }} />  
  
    <table align="center">  
        <tr>  
            <td><label for="username">{{ 'Identifiant'|trans }}</label></td>  
            <td><input type="text" id="username" name="_username" value="{{ last_username }}" required="required" /></td>  
        </tr>  
  
        <tr>  
            <td><label for="password">{{ 'Mot de passe'|trans }}</label></td>  
            <td><input type="password" id="password" name="_password" required="required" /></td>  
        </tr>  
  
        <tr>  
            <td><input type="checkbox" id="remember_me" name="_remember_me" value="on" /></td>  
            <td><label for="remember_me">{{ 'Se souvenir de moi'|trans }}</label></td>  
        </tr>  
  
        <tr>  
            <td><input type="submit" id="_submit" name="_submit" value="{{ 'Connexion'|trans }}"/></td>  
        </tr>  
  
    </table>  
</form>
```

Dans le code, on voit que le formulaire nous demande un identifiant et un mot de passe, mais de plus il nous permet de garder en mémoire ces informations.

### Gérer les balises:

Gérer les balises comprend en ajouter. Afin de pouvoir ajouter une balise, j'ai crée un formulaire dont un extrait apparaît ci-dessous:

```

/**
 * @Route("/ajouter")
 * @Template()
 */
public function ajouterAction() {

    $balise = new Balise();
    // On crée le FormBuilder grâce au service form factory
    // On ajoute les champs de l'entité que l'on veut à notre formulaire
    $form = $this->createFormBuilder($balise)
        ->add('nom', 'text')
        ->add('latitude', 'text')
        ->add('longitude', 'text')
        ->add('altitude', 'text')
        ->add('Envoyer', 'submit')
        ->getForm();

    $request = $this->get('request');

}

/**
 * @Route("/ajouter")
 * @Template()
 */
public function ajouterAction() {

    $balise = new Balise();
    // On crée le FormBuilder grâce au service form factory
    // On ajoute les champs de l'entité que l'on veut à notre formulaire
    $form = $this->createFormBuilder($balise)
        ->add('nom', 'text')
        ->add('latitude', 'text')
        ->add('longitude', 'text')
        ->add('altitude', 'text')
        ->add('Envoyer', 'submit')
        ->getForm();

    $request = $this->get('request');
}


```

pourquoi le code est-il dupliqué ?

On remarque que pour créer une nouvelle balise on doit lui administrer un nom ainsi que des coordonnées comme énoncé plus haut.

#### Questions :

- Y a-t-il un lien avec une vue ?
- Comment la balise est-elle enregistrée après avoir validé le formulaire ?

### Mise en forme du site:

Dans cette mise en forme j'ai créé un menu avec un système d'onglet pour naviguer entre les pages. De plus, j'ai utiliser `{%if is_granted('ROLE_ADMIN')%}` pour le lien qui conduit à la page qui permet d'ajouter une balise. Cette fonction permet d'ajouter au menu « Ajouter balise » quand l'utilisateur est authentifié.

On retrouve une partie du code de la base.html.twig ci-dessous :

```
<body>
```

```
<div style="position: absolute; top: 25px; right: 70px">
    {% if is_granted("IS_AUTHENTICATED_REMEMBERED") %}
        {{ '|trans({'%username%': app.user.username}, 'FOSUserBundle') }}>
        <a class="btn btn-info" href="{{ path('fos_user_security_logout') }}>
            {{ 'Déconnexion'|trans({}, 'FOSUserBundle') }}
        </a>
    {% else %}
        <a class="btn btn-info" href="{{ path('fos_user_security_login') }}>
            {{ 'S authentifier'|trans({}, 'FOSUserBundle') }}</a>
    {% endif %}
</div>
<table align="center">
    <tr>
        <td><a class="btn btn-info" type="button" href="{{path('meteofox_gestion_accueil')}}> Accueil</a></td>
        <td><a class="btn btn-info" type="submit" href="{{path('meteofox_carte_listall')}}> Liste complète des
            balises</a></td>
        <td><a class="btn btn-info" type="button" href="{{path('meteofox_carte_affcarte')}}> Afficher la
            carte</a></td>
        <td> {%if is_granted('ROLE_ADMIN')%}<a class="btn btn-info" type="button"
            href="{{path('meteofox_gestion_ajouter')}}> Ajouter une balise</a>{% endif %}</td>
    </tr>
</table>
    {% block body %}
    {% endblock %}
</body>
```

il y aurait beaucoup de closes qui mériteraient des explications.

- à quoi servent les directives `{%block xxx%}` par exemple ?
- et autres directives `{% %}`

## 5.6) Bilan personnel technique

Pour conclure cette partie personnelle, on va reprendre toute les taches qui ont été réalisées, et celles qui restent à faire.

Jusqu'à présent, j'ai pris connaissance du projet dans son ensemble et pas uniquement de la partie que j'aurai à développer, j'ai suivi un tutoriel de Symfony, le Framework que j'utilise pour réaliser le serveur METEOFOLX. Avec l'aide de plusieurs de mes partenaires, nous avons réalisé l'ihm du site qui reprend les critères du cahier des charges. J'ai ensuite utiliser la vue « login et layout » du FOSUserBundle pour gérer les utilisateurs, puis j'ai créé un formulaire pour ajouter des balises.

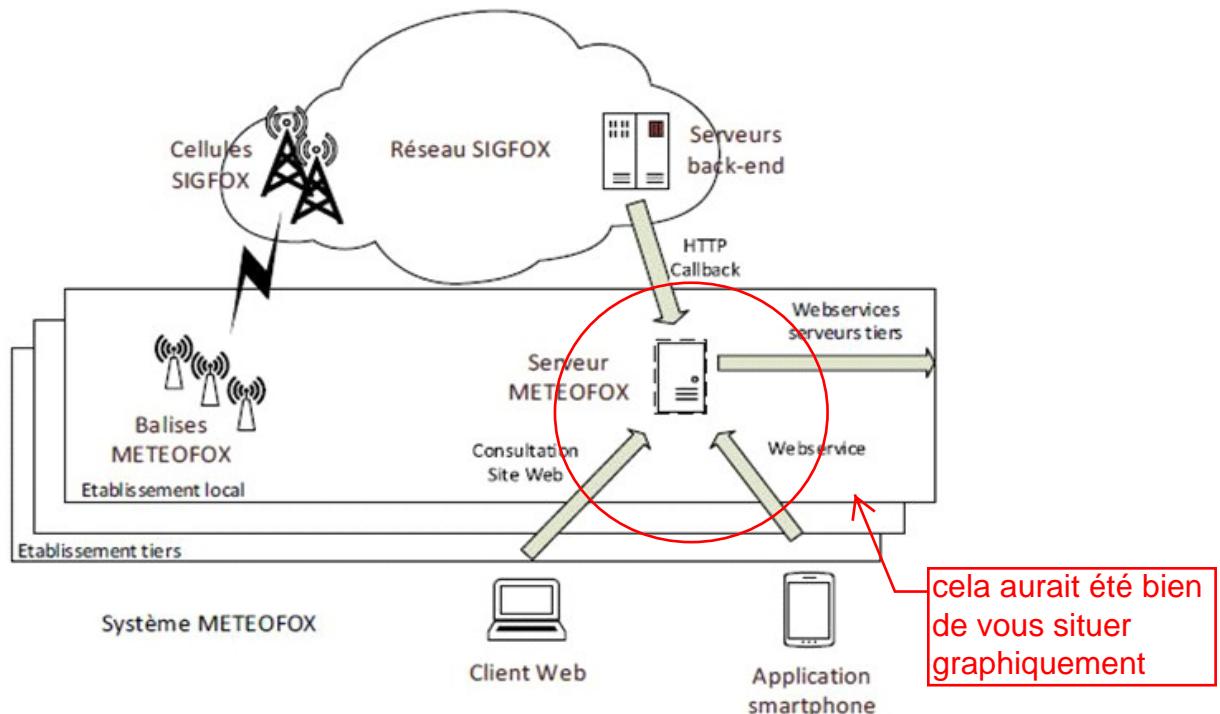
Il me reste à présent à créer un formulaire qui permet de supprimer les balises, puis à rendre l'aspect visuel du serveur plus agréable à l'aide du CSS.

## 6) Tinet Ludovic - Historisation des balises

### 6.1) Description partie personnelle

#### 6.1.1) Me situer

Ci-dessous un schéma représentatif du projet METEOFOX. A l'intérieur de ce schéma, je me situe donc au niveau du serveur METEOFOX et je suis celui qui doit faire le lien entre le serveur METEOFOX et l'affichage de la page web du client.



#### 6.1.2) Mes tâche

##### Afficher historique des balises :

→ Donnée météo sous forme de graphe

##### Afficher les détails des balises :

→ Localisation  
→ Données météo actuelles

## 6.2) Mis en œuvre

### 6.2.1) Tutoriel Symfony2

Pour réaliser mon projet, je vais devoir utiliser Symfony, et pour cela on a du suivre un tutoriel pour apprendre les bases et ensuite commencer mon projet.

Mais qu'est ce que symfony ?

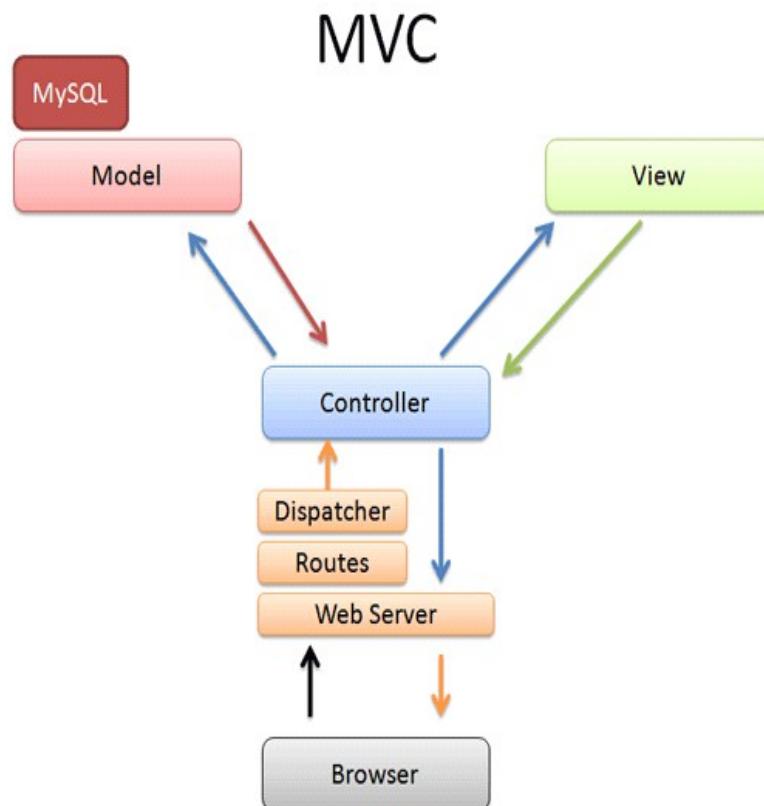
*Symfony est un ensemble de composants PHP, un cadre d'application Web mais pas seulement c'est aussi une Framework fonctionnant sous le modèle MVC.*

Le modèle MVC est un modèle qui fonctionne sous 3 couches:

→ Modèle: Son rôle est de gérer vos données ainsi que leurs contenus.

→ Vue: Permet d'afficher les pages.

→ Contrôleur: Génère la réponse à la requête HTTP demandée par notre visiteur.



### 6.2.2) [Fonctionnement Symfony](#)

#### Les bundles:

Pour commencer, il faut générer un bundle à l'aide de la commande :

```
php app/console generate:bundle
```

Une fois le bundle généré, la portion de code ci-dessous se rajoute au projet créé:

```
irist_gest_stg1:
    resource: "@IRISTGestStg1Bundle/Controller/" // Chemin de la resource
    type: annotation // Format
    prefix: / // Prefix
```

#### Les contrôleurs:

Pour ce qui est du contrôleur, la commande pour le générer est:

```
php app/console generate:controller
```

Pendant la génération du contrôleur, on ajoute nos méthodes, et une fois le contrôleur créé, on retrouve ses méthodes:

```
class EntrepriseController extends Controller
{
    /**
     * @Route("/index")
     * @Template()
     */
    public function indexAction()
    {
        return array(
            // ...
        );
    }

    /**
     * @Route("/del")
     * @Template()
     */
    public function delAction()
    {
        return array(
            // ...
        );
    }

    /**
     * @Route("/add")
     * @Template()
     */
    public function addAction()
    {
        return array(
            // ...
        );
    }
}
```

### Les Entités:

Les entités ce génèrent avec la commande: **php app/console generate:doctrine:entity**.

On crée ensuite la base de données, puis on génère et synchronise des tables dans cette base associées aux entités.

Avec l'aide de phpMyAdmin, on peut vérifier la génération de notre entité et ses modifications:

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	<b>id</b>	int(11)			Non	Aucune	<b>AUTO_INCREMENT</b>	<b>Modifier</b> <b>Supprimer</b> <b>Primaire</b> <b>Unique</b> <b>Ind</b>
2	<b>nom</b>	varchar(255)	utf8_unicode_ci		Non	Aucune		<b>Modifier</b> <b>Supprimer</b> <b>Primaire</b> <b>Unique</b> <b>Ind</b>
3	<b>ville</b>	varchar(255)	utf8_unicode_ci		Non	Aucune		<b>Modifier</b> <b>Supprimer</b> <b>Primaire</b> <b>Unique</b> <b>Ind</b>
4	<b>cp</b>	bigint(20)			Non	Aucune		<b>Modifier</b> <b>Supprimer</b> <b>Primaire</b> <b>Unique</b> <b>Ind</b>

### [6.3\) Highcharts](#)

n'importe quel ?

Highcharts est une librairie javascript qui sert à créer n'importe quel graphiques. J'utilise donc cette librairie pour faire l'historisation des balises météo que je dois mettre sous formes de graphe.

Dans un premier temps, il faut rentrer les données météo dans une base de données.

Après avoir rentrer les données il faut se connecter à la base de données pour pouvoir afficher de graphe grâce à ce code dans le contrôleur:

```
$repository = $this
    ->getDoctrine()
    ->getManager()
    ->getRepository('IRISTGestStgBundle:Historique');

$listeHisto = $repository->findAll();
```

Ensuite pour créer le graphique, il faut ajouter le code dans le contrôleur suivant la forme et le nombre de données qu'on veux rentrer, dans notre cas 3 (vitesse max, min, moy).

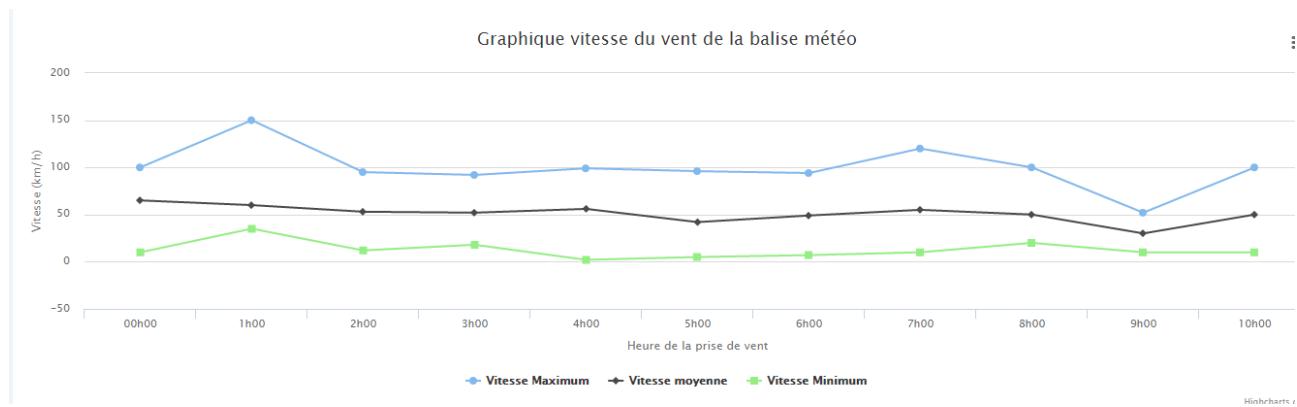
que contient  
'\$listeHisto' ?

Et après appelé la librairie highcharts pour créer notre vue ce qui serviras à voir notre graphique.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js" type="text/javascript"></script>
<script src="http://code.highcharts.com/highcharts.js"></script>
<script src="http://code.highcharts.com/modules/exporting.js"></script>
<script type="text/javascript">
    {{ chart(linechart) }}
</script>

<div id="linechart" style="min-width: 310px; height: 400px; margin: 0 auto"></div>
```

Au final, on obtiens ceci:



#### 6.4) Bilan

En conclusion de ma partie, travailler sur symfony à étais un outil compliqué à comprendre, mais une fois certaines choses de comprises, le projet a pu avancer .

Pour ma part créer un graphique à l'aide d'highcharts, étais complexe et grâce a plusieurs tutoriels trouver j'ai pu réaliser ce que je voulais faire.

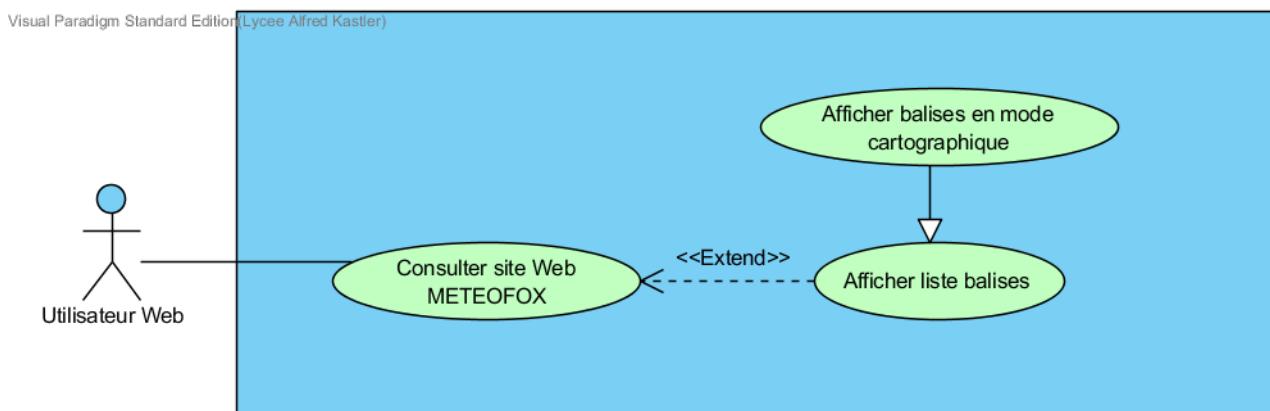
## 7) RADULOV Lachezar : Affichage des balises en mode cartographique

### 7.1) Description partie personnelle

Description de la partie personnelle du projet :

**Cas d'utilisation :** Dans les cas d'utilisation du projet ma partie consiste à afficher balises en mode cartographiques

Diagramme cas utilisation :



Ce diagramme montre mon rôle dans le projet. Ici un utilisateur ( web ) doit être en mesure de consulter sur notre site web METEOFOX une liste de balises, où ses balises sont placées à différentes endroits. Ils doivent être également visibles sur une carte avec leurs coordonnées exactes.

**Description :** Dans cette partie du projet je suis chargé de cartographier des balises :

- dans un premier temps je dois afficher une carte de France
- afficher toutes les balises qui sont disponibles
- si un administrateur ajoute une balise dans notre base de données elle doit automatiquement apparaître sur la carte
- ou bien si un administrateur supprime une balise elle ne doit plus s'afficher sur la carte
- personnaliser la carte : modifier les marqueurs, la carte doit être bien visible sur notre page

**Contraintes :** Les contraintes que je dois respecter sont :

- Prévoir le centrage de la vue sur la position de l'utilisateur, ou sinon disponible une vue par défaut

## 7.2) Mise en oeuvre

Technologies utilisées dans le projet :

### **Wamp Serveur :**

WampServer est une plate-forme de développement Web sous Windows pour des applications Web dynamiques à l'aide du serveur Apache2, du langage de scripts PHP et d'une base de données MySQL. Il possède également PHPMyAdmin pour gérer plus facilement vos bases de données

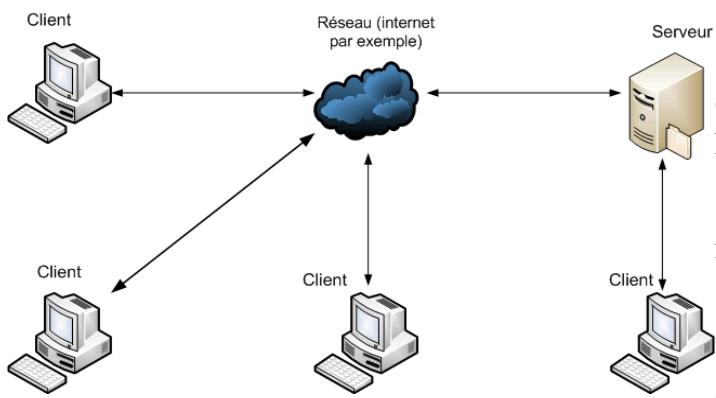


On utilise la version WampServeur 2.5, qui nous permet de développer notre projet localement.

### **TortoiseSVN :**



Nous sommes plusieurs sur le même projet et il nous arrive dans certains cas travailler sur les mêmes fichiers (exemple : des Contrôleurs, pages HTML, Twig..) c'est pour cela qu'on utilise un Subversion (ou plutôt un SVN) qui est un système de gestion de versions des fichiers. Ce logiciel nous permet non seulement de récupérer la partie sur laquelle travaille chaque étudiant mais aussi de revenir à une version antérieure de notre code au cas où. Ce logiciel nous permet de voir aussi les différences entre les versions des fichiers.



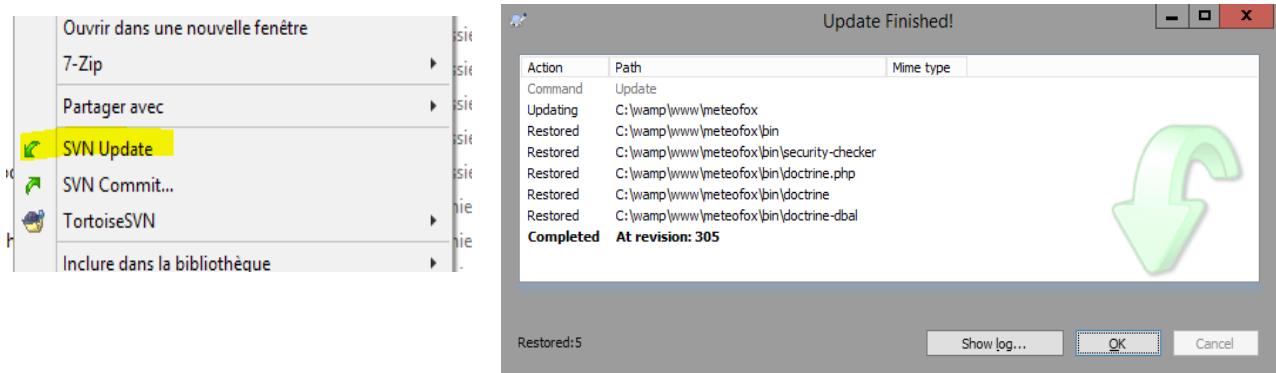
Il est possible pour plusieurs personnes d'utiliser le système en même temps, ce qui peut nous donner le schéma contre :

Ceci nous permet de travailler depuis notre domicile comme dans les labos.

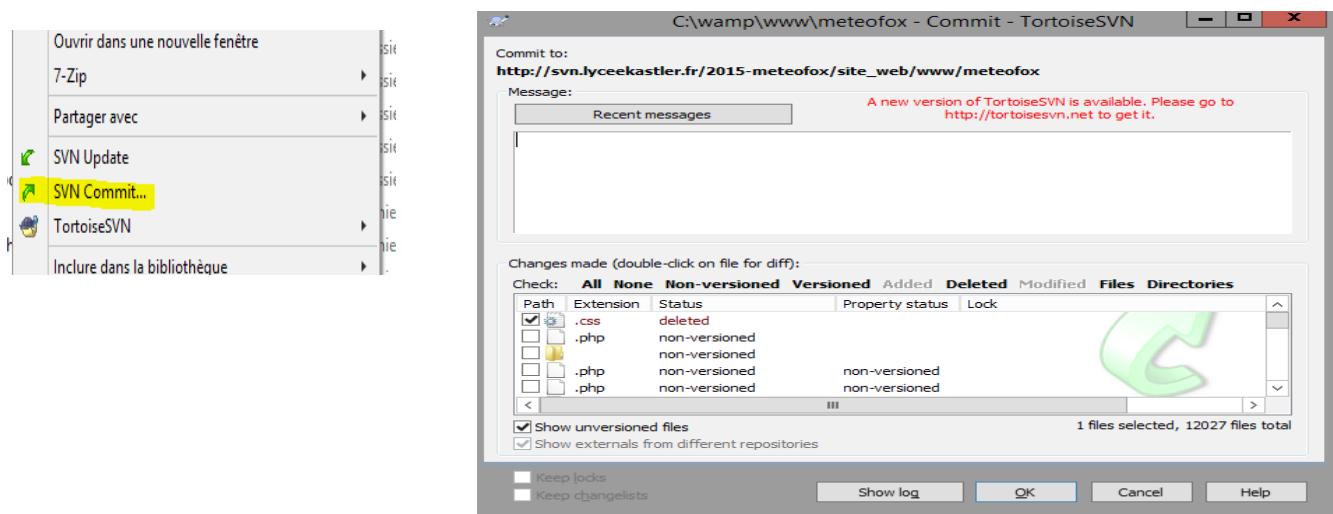
Le lien où sont stockés tous nos fichiers, se situe sur un serveur dans notre lycée qui est :

[http://svn.lyceekastler.fr/2015-meteofox/site\\_web/www/meteofox](http://svn.lyceekastler.fr/2015-meteofox/site_web/www/meteofox)

SVN Update : Cette option nous permet de mettre à jour nos fichiers qui sont stockés dans le serveur. Une fois l'option exécuter une fenêtre s'affiche et nos montre les fichiers récupérés.



SVN Commit : Cette option nous permet d'envoyer des fichiers sur le serveur. Il est possible de « commit » un seule fichier ou un dossier entier. Une fenêtre nous montre l'ensemble des fichiers qu'on envoi. Pour que les autres étudiant récupèrent notre travail suffit de faire un « SVN Update »



**Le frameWork Symfony :** Notre projet à était développé avec le frameWork Symfony :

En quelque mots :

Symfony est un framework PHP développé par une équipe française dont le chef d'orchestre est Fabien Potencier. Son but est l'accélération du développement et de la maintenance d'applications web, en se basant sur le modèle maintenant classique : Model-View-Controller.

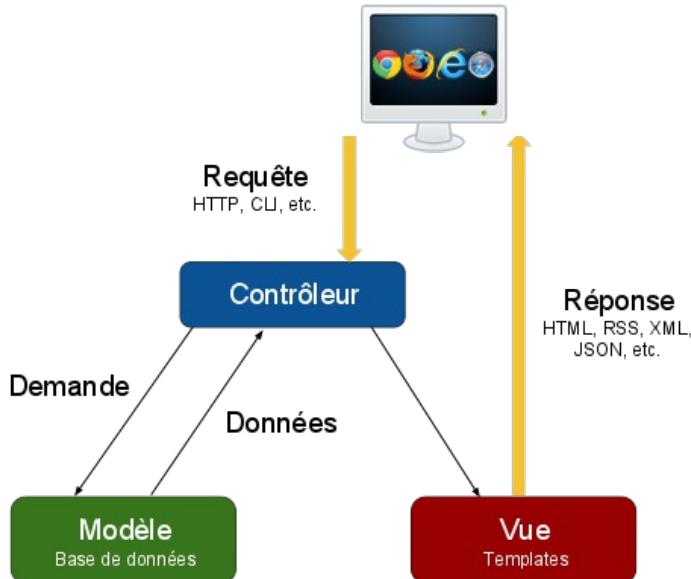
Pourquoi utiliser un framwork ?

L'avantage d'un framework, c'est aussi de pouvoir travailler dans le respect des bonnes pratiques et de réutiliser des méthodologies de conception standardisées.

On peut particulièrement citer le design pattern MVC, acronyme de l'expression "**Modèle - Vue - Contrôleur**". Un design pattern est traduit par "patron de conception", c'est donc un modèle qui a fait ses preuves et qui s'avère indispensable à connaître dans le domaine de la programmation.

Découpons un peu tout ça :

- **Modèle** : Le modèle est la partie du code qui se charge d'intégrer avec la base de données.
- **Vue** : La vue est la partie du code qui se charge uniquement d'afficher les données qui lui ont été fournies. LA vue ne traite pas les données, sauf dans le but d'en embellir l'affichage.
- **Contrôleur** : Le contrôleur est le centre de notre design pattern. Il reçoit la requête HTTP, l'interprète et coordonne le tout. Il se charge de demander au modèle les données puis effectue plus ou moins de traitements dessus afin d'envoyer à la vue les données à afficher et de retourner une réponse à l'émetteur de la requête.



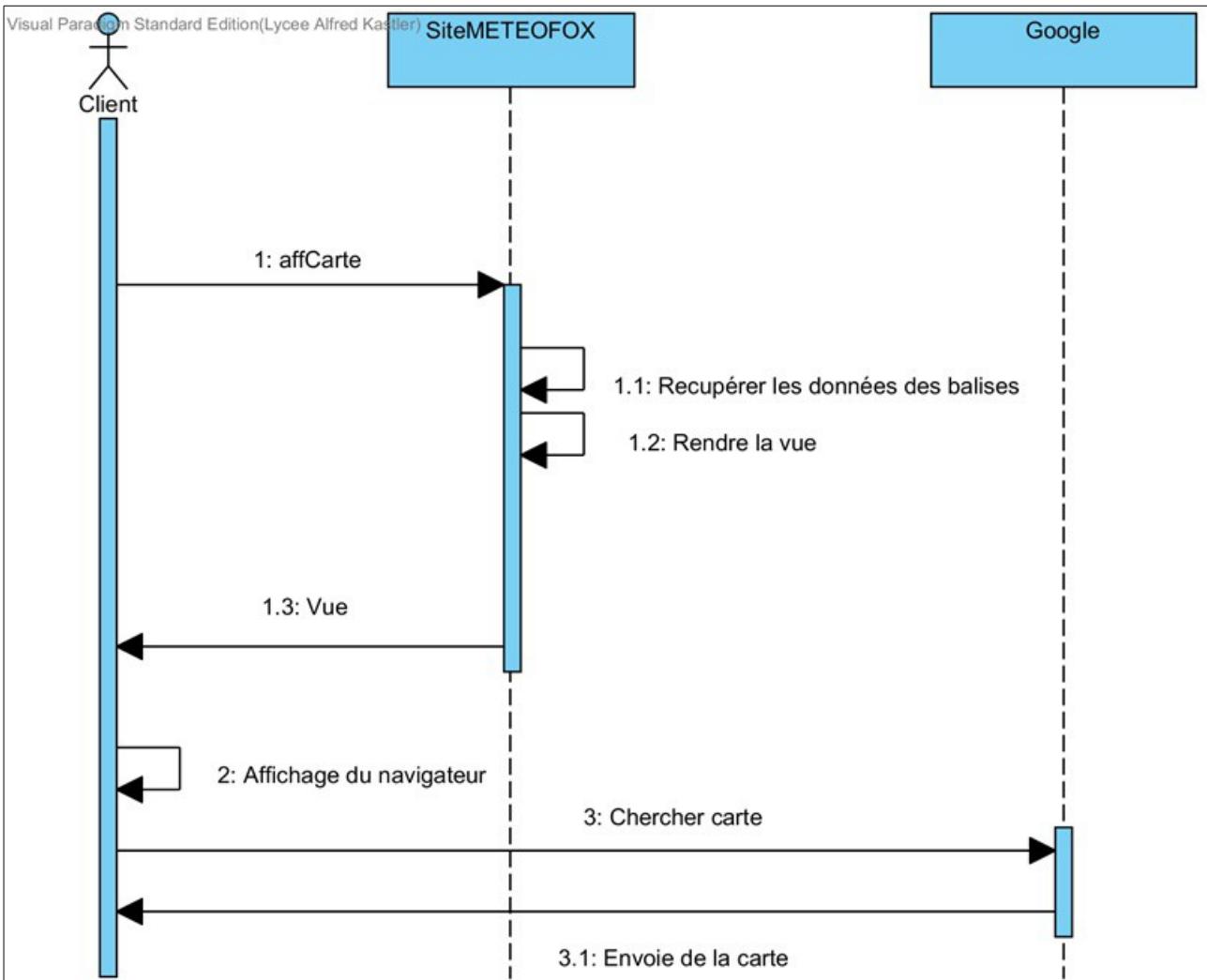
*Schématisation du design pattern MVC*

On remarque bien que chaque phase du travail est bien séparée, ce qui permet une organisation plus claire du code.

**Chaque chose à sa place.**

### 7.3) Conception détaillée

#### 7.3.1) Diagramme de séquence



Dans ce diagramme je montre comment le client accède à la page qui affiche la carte :

- Dans un premier temps on accède à notre site Meteofox pour récupérer les données des balise
- La vue s'affiche sur la page web mais la carte n'est pas encore afficher
- On va chercher notre carte chez Google ( on remarque que la vue met quelques secondes pour s'afficher c'est parce que la carte charge )
- En fin, on re renvoi une vue complète ai client, il pourra voir la carte et les balises placées sur elle.

#### 7.4) Implémentation

Pour la suite du projet on aura besoin d'utiliser un Bundle :

**Définir un Bundle** : c'est un ensemble de fichiers et répertoires permettant d'implémenter une ou des fonctionnalités.

Nous allons pas créer notre propre Bundle mais nous allons utiliser le Bundle suivant :

IvoryGoogleMapBundle qui va nous permettre de :

- créer une carte
- il va nous permettre de manipuler toutes les options de carte
- entrée sur les coordonnées, configuré avec un zoom
- créer des marqueur..

#### *Mise en œuvre du bundle IvoryGoogleMapBundle*

Avant de commencer avec le code nous devons faire quelques manipulations avant :

Les parties que l'on va intégrer sont obligatoires pour le bon fonctionnement du Bundle

Ajout dans la section « Require » du fichier **composer.json** des deux lignes suivantes :

```
"egeloen/google-map-bundle": "~2.1",
"willdurand/geocoder": "*"
```

Ajout dans la section \$bundles = array du fichier « app/kernel.php » de :

```
new Ivory\GoogleMapBundle\IvoryGoogleMapBundle(),
```

Et pour finir on ajoute tout les includes spécifique à Ivory\GoogleMap.

```
use Ivory\GoogleMap\Controls\ControlPosition;
use Ivory\GoogleMap\Controls\MapTypeControl;
```

Avant de commencer le projet en lui-même nous allons faire quelques étapes de tests sur une page.

### 7.4.1) Création d'une carte :

`$map = $this->get('ivory_google_map.map');`

On a ici une variable qui s'appelle « map » qui reçoit les fonctionnalités de notre Bundle

Voici le code dans le contrôleur :

```
 /**
 * @Route ("/index")
 * @Template()
 */
public function indexAction() {

    $map = $this->get('ivory_google_map.map');
    $map->setAutoZoom(false);
//centrer la carte
    $map->setCenter(44.252341, 11.939396, true);
// définir le zoom sur la carte
    $map->setMapOption('zoom', 1);
    $mapTypeControl = new MapTypeControl();
    $map->setMapTypeControl($mapTypeControl);
    $map->setMapTypeControl(
        array(MapTypeId::ROADMAP, MapTypeId::SATELLITE),
        ControlPosition::TOP_RIGHT, MapTypeControlStyle::DEFAULT_);
    return array('map' => $map);
}
```

- Tout d'abord on défini la route : c'est le liens avec le quelle on pourra accéder à notre page.

- propriété dans la variable 'map' on définir et a changer en cas de besoin ( le zoom automatique, a quelle endroit la carte sera centrer..)

-La méthode SetCenter (), permet de centrer la carte.

on retrouve deux valeurs qui

correspondent a la longitude et a la latitude du point central de la carte. Pour définir une position voulu il suffit d'aller sur le site 'maps.google.fr' , de retrouver la position et de remplacer les données.

Extrait du code de la vue /index.html.twig et la vue qui correspond au codes

Ici on retrouve la variable 'map' qu'on a utiliser dans le contrôleur.

```
1 {{ google_map_css(map) }}
2 {{ google_map_container(map) }}
3 {{ google_map_js(map) }}
```



On retrouve la carte sur le liens qu'on a défini dans le contrôleur. On a retrouvé une carte, que par la suite on devra center et définir un zoom cohérent.

La carte a été bien évidement récupérer par Google, d'où on remarque leur logo sur la carte, puis on a deux options pour visualiser notre carte : soit en mode « satellite » ou bien comme dans ce cas en mode « map »

### 7.4.2) Ajour de marqueur avec infobulles

A la suite du contrôleur dans le quelle on a crée notre carte on ajoute le code qui va nous permettre de placer un marqueur avec une infobulle :

```
$marker = new Marker();
$marker->setPrefixJavascriptVariable('marker_image_');
$marker->setPosition(44.252341, 11.939396, true);
$marker->setAnimation(Animation::DROP);
$marker->setOptions(array(
    'clickable' => true,
    'flat'      => true,
    'tooltip'   => true,
    'text'      => "test"));
$infoWindow = new InfoWindow();
$infoWindow->setPrefixJavascriptVariable('info_window_');
$infoWindow->setPosition(0, 0, true);
$infoWindow->setPixelOffset(1.1, 2.1, 'px', 'pt');
$infoWindow->setContent('<p>'.z.</p>');
$infoWindow->setOpen(true);
$infoWindow->setOpenEvent(MouseEvent::CLICK);
$infoWindow->setAutoClose(true);
$infoWindow->setOption('disableAutoPan', true);
$infoWindow->setOption('zIndex', 10);
$infoWindow->setOpenEvent('mouseover');
$marker->setInfoWindow($infoWindow);
$map->addMarker($marker);
```

On crée une variable qui s'appelle \$marker et on crée un nouveau marqueur. On peut modifier plusieurs paramètres afin de personnaliser le marqueur comme :

- à quoi va ressembler le marqueur ici on a une image standard qui vient de chez Google, mais on peut très bien la modifier
- avec setPosition(), on définit l'emplacement exacte de notre marqueur, avec sa latitude et longitude.
- Avec le setContent() on peut mettre un texte qui devrait apparaître dans une infobulle quand on clique sur

le marqueur

Voici le résultat :



On a notre carte qu'on a créer précédemment puis on peut voir qu'un marqueur vient d'apparaître sur la carte. Bien évidemment pour faire le test on a choisi une position du marqueur au hasard et un texte simple.

Après cette série de tests nous allons nous intéresser à notre vraie page.

e

### 7.4.3)      Modifier la taille de la carte ainsi, personnaliser le marqueur, centrer la carte

```
// taille de la carte
$map->setStylesheetOptions(array(
    'width' => '700px',
    'height' => '700px',
));
```

Cette partie du code va nous permettre de définir la taille de la carte sur la page web. J'ai choisi de faire un carte carré, de 700pixels de largeur et 700pixels de hauteur.

La carte sera centrée par défaut sur une position, que j'ai défini. Quand on recharge la page web on aura une vue sur la France.

```
// center la carte,
$map->setCenter(46.799966, 1.758466, true);
// Ajuster le zoom sur la carte
$map->setMapOption('zoom', 6);
```

```
$marker->setIcon('https://cdn1.iconfinder.com/data/icons/Map-Markers-Icons-Demo-PNG/48/Map-Marker-Marker-Outside-Azure.png')
```

Pour personnaliser l'icône de notre marqueur il suffit d'utiliser la fonction setIcon() dans laquelle j'intègre une image en format .png.

### 7.4.4)      Utiliser les entités sur la carte

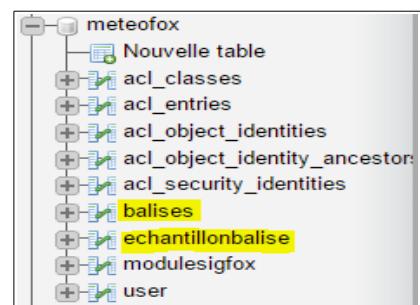
- **Définir la notion d'entités :**

Les Entity dans Symfony2 sont des classes métiers qui décrivent chaque objet de notre application, nous définirons ce qu'elles contiennent

- 

Dans la base de données j'utilise exactement deux tables :

- Dans la table « balise » on retrouve la liste des balise
- Dans la table « echantillonbalise » on retrouve les données prélevées par les balise météorologiques.



Voici la table « balise » :

			<b>id</b>	<b>nom</b>	<b>latitude</b>	<b>longitude</b>	<b>altitude</b>	
<input type="checkbox"/>	 <a href="#">Modifier</a>	 <a href="#">Copier</a>	 <a href="#">Effacer</a>	1	Dune du Pyla	44.554906760063446	-1.2393951416015625	40
<input checked="" type="checkbox"/>	 <a href="#">Modifier</a>	 <a href="#">Copier</a>	 <a href="#">Effacer</a>	3	Lycée Alfred Kastler	44.802073	-0.59946300000000143	60
<input type="checkbox"/>	 <a href="#">Modifier</a>	 <a href="#">Copier</a>	 <a href="#">Effacer</a>	4	Lycée Gaston Crampe (Aire s/Adour)	43.691263	-0.27644799999999594	150
<input type="checkbox"/>	 <a href="#">Modifier</a>	 <a href="#">Copier</a>	 <a href="#">Effacer</a>	5	Lycée Jean Monnet (Aurillac)	44.935548	2.45170099999999573	650

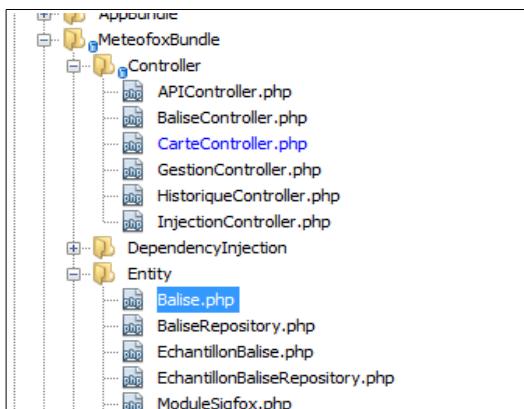
Dans cette table contient cinq champs:

**id** : un numéro unique que correspond a une seule balise, quand on veut ajouter une balise, l'id va s'incrémenter automatiquement.

**nom** : On donne un nom de chaque balise, dans ce cas le nom de nos balises correspond à l'endroit où elles sont situées.

**latitude/longitude** : c'est les coordonnées exacte des chaque balise.

**altitude** : c'est à quelle altitude se situe la balise, elle peut être donnée par le capteur de pression.



Dans le répertoire Entity, nous avons crée l'entité « Balise » qui interagit avec notre base de données.

Pour la suite du projet nous allons nous occuper des entités et non pas avec la base de données. C'est le but des entité.

Donc nous allons utiliser la base de données dans le seul but de vérifier nos données

Dans l'entité balise on retrouve la classe « Balise » qui interagit avec la table « balise » dans notre base de données « meteofox »

```
 /**
 * Set nom
 *
 * @param string $nom
 * @return Balise
 */
public function setNom($nom)
{
    $this->nom = $nom;
    return $this;
}

 /**
 * Get nom
 *
 * @return string
 */
public function getNom()
{
    return $this->nom;
}
```

```
 /**
 * Balise
 *
 * @ORM\Table(name="balises")
 * @ORM\Entity(repositoryClass="MeteofoxBundle\Entity\BaliseRepository")
 * @ExclusionPolicy("none")
 */
class Balise
{
```

Extrait de l'entité balise :

Dans cet classe on retrouve des fonctions qu'on pourra utiliser

par la suite :

par exemple les :

`Set()` qui nous permet d'envoyer des données vers notre base

de données

`Get()` qui nous permet de récupérer des données depuis la

base, nous allons

utiliser ici pour récupérer les nom

des balise ainsi que l'altitude, la longitude..

Voici quelles données qui proviennent de la table « echantillonbalise » :

<input type="checkbox"/>		Modifier		Copier		Effacer	1	1	1428340179	20	1016
<input type="checkbox"/>		Modifier		Copier		Effacer	2	1	1428341079	20	1016
<input type="checkbox"/>		Modifier		Copier		Effacer	3	1	1428341979	19	1015
<input type="checkbox"/>		Modifier		Copier		Effacer	4	1	1428342000	15	1013

Cette table contient les champs suivants :

**id** : un numéro unique qui correspond a un échantillon météorologique d'une balise.

**balise\_id** : grâce a ce champs on retrouve de quelle baise provient l'échantillon.

**timestamp** : c'est a quelle moment l'échantillon a été envoyer, le « timestamp »

(ou l'horodatage ) est le nombre de seconde écoulé depuis le 01/01/1970 à 00:00.

**temperature** : la balise est équiper d'un capteur de pression qui peut mesurer la température en degrés Celsius.

**pression** : la pression est donner en hecto-Pascal.

**dir\_vent** : un anémomètre nous donne la direction du vent en degrés a partir du Nord .

**vit\_vent\_min** :c'est la plus petite valeur que l'anémomètre nous envoi sur la plage de 15min

**vit\_vent\_max** : c'est la plus grande valeur que l'anémomètre nous envoi sur la plage de 15min.

**vit\_vent\_moy** : c'est la vitesse du vent moyen, pour cela on prélevé toutes les valeurs et on les additionnent puis on divise par le nombre de valeurs.

On utilise le méthode suivante pour accéder a la table « Balise » dans notre base de données « Meteofox ».

```
$repository1 = $this
    ->getDoctrine()
    ->getManager()
    ->getRepository('MeteofoxBundle:Balise');
$listebalise1 = $repository1->findAll();
```

La méthode findAll() retourne toutes les entités contenue dans la base de données. Le format du retour est un simple Array, que nous allons parcourir avec une boucle « foreach » pour utiliser les objets qu'il contient.

```
foreach ($echantillon as $echantillonbalise) {
```

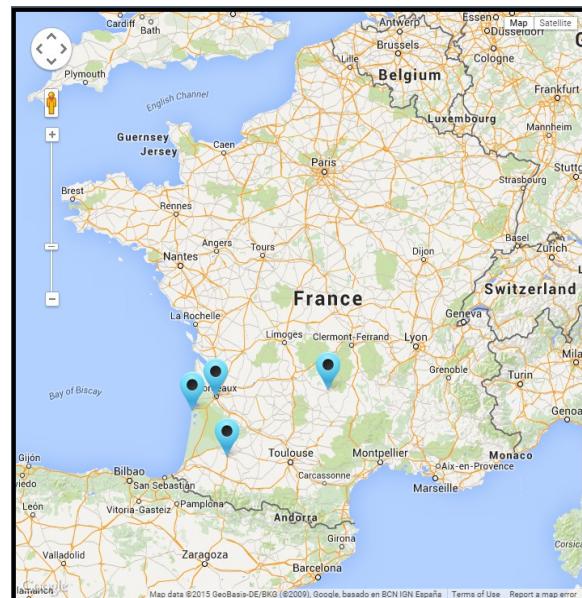
Les données qu'on utilise depuis la base de données sont ; la latitude et la longitude des balises.

```
$marker->setPosition($balise->getLatitude(), $balise->getLongitude(), true);
```

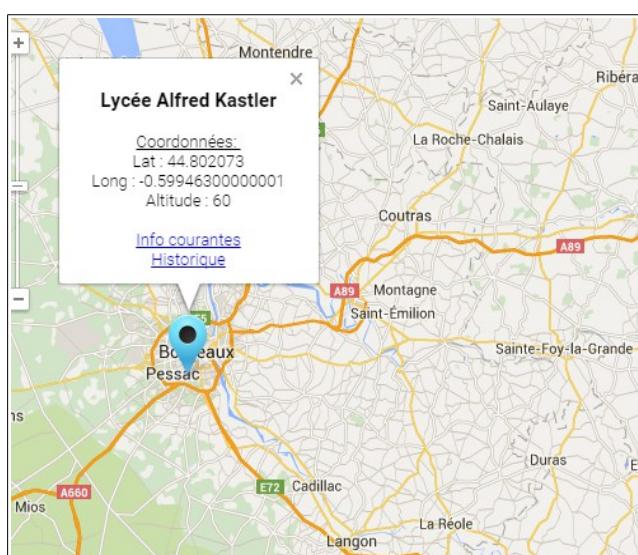
Pour cela on accède à l'entité « balise » et on utilise les méthodes getLatitude() et getLongitude() grâce aux quelles on pourra récupérer les données souhaiter directement depuis la base.

Résultat :

- La carte est bien centrer sur la France
- Les marqueur sont personnaliser
- On voit apparaître toutes les balises qui sont dans notre base de données



Quand on clique sur une balise on retrouve les informations suivantes :



- Le nom de la balise
- Les coordonnées de la balise ( latitude et longitude )
- L'altitude à la quelle se situe la balise
- Un lien vers une page où on pourra voir les informations météorologiques actuelles
- Un lien vers une page où on peut retrouver l'historique des données méthodologies sur de longue période

Collecte des données météorologiques
--------------------------------------

```
$infoWindow->setContent(
    '</br><a href="/meteofox/web/app_dev.php/detail/' . $balise->getId() . '"><br>' .
    'Info courantes' .
    '<a href="/meteofox/web/app_dev.php/historique/' . $balise->getId() . '"><br>' .
    'Historique' . '<br>');

```

Voici le code qui nous permet de nous rediriger vers les pages : Info courantes ou bien Historique  
Les liens qui correspondent vers ces pages sont les suivants :



On récupère l'identifiant de la balise, dans ce cas on aura les informations et l'historique de la balise numéro « 3 » qui correspond à la balise située au lycée A.Kaslter.

#### 7.4.5) Afficher une liste des balises

Dans une deuxième partie je dois afficher une liste des balises.

```
/**
 * @Route ("/listall")
 * @Template()
 */
public function listallAction() {
    $repository = $this
        ->getDoctrine()
        ->getManager()
        ->getRepository('MeteofoxBundle:Balise');
    $listebalises = $repository->findAll();

    return array('listebalise' => $listebalises);
}
```

Toujours dans le contrôleur de la carte je créer une nouvelle fonction qui me permettra d'afficher une liste des balises présente dans la table « Balise » dans notre base de données.

On définit une nouvelle route, elle va nous permettre d'aller sur une page où dans un tableau on retrouvera toutes les balises.

On utilise la même méthode qu'on a utiliser précédemment, on accède à la base et on parcourt toutes les données.

Dans la page « /listall.html.twig » on retrouve le code suivant : ici on reconnaît le langage HTML 5 et quelques bouts du langage TWIG

Grâce au langage twig on va créer une boucle « for » qui parcourt le tableau « listebalise » où sont stockées toutes les balises. Puis on rentre les données dans un tableau

```
<table>
    <tr align="center"><strong>
        <th class="id"> ID balise</th>
        <th class="id"> Nom </th>
        <th class="id"> Latitude </th>
        <th class="id"> Longitude</th>
        <th class="id"> altitude </th>
    </tr> </strong>
    {% for Entre in listebalise%}
    <tr align="center" >
        <td class="id">{{Entre.id}}</td>
        <td> {{Entre.nom}}</td>
        <td> {{Entre.latitude}}</td>
        <td> {{Entre.longitude}}</td>
        <td> {{Entre.altitude}}</td>
    </tr>
    {% endfor%}

```

On obtient le résultat suivant :

ID balise	Nom	Latitude	Longitude	altitude
1	Dune du Pyla	44.554906760063	-1.2393951416016	40
3	Lycée Alfred Kastler	44.802073	-0.59946300000001	60
4	Lycée Gaston Crampe (Aire s/Adour)	43.691263	-0.27644799999996	150
5	Lycée Jean Monnet (Aurillac)	44.935548	2.451701	650

Ici on a bien l'Identifiant et le nom de chaque balise, ainsi que les coordonnées et l'altitude.

## 7.5) Bilan personnel technique

Ce projet m'a permis d'approfondir mes connaissances informatiques, j'ai découvert de nouveaux outils, tel que le frameWork Symfony, le langage Twig, etc. Cela m'a permis de créer et utiliser des relations entre entités, gérer des contrôleurs.. de plus, j'ai réutilisé des langages comme le HTML et le PHP que j'ai découvert en première année. Je suis désormais capable de créer carte et de la manipuler de différentes façons.

**8) Étudiant 7****nom ?**

## 9) Étudiant 8 : Audoy François

### 9.1) Description de ma Partie

Ma partie personnelle consiste à concevoir une application Android.

Pour cette application j'ai dû préparer l'IHM pour répondre au cahier des charges, à savoir :

- Afficher la liste et l'état des balises
- Afficher les balises en mode cartographique
- Recentrer sur la position actuelle
- Choisir les préférences.

#### 9.1.1) Cas d'utilisation

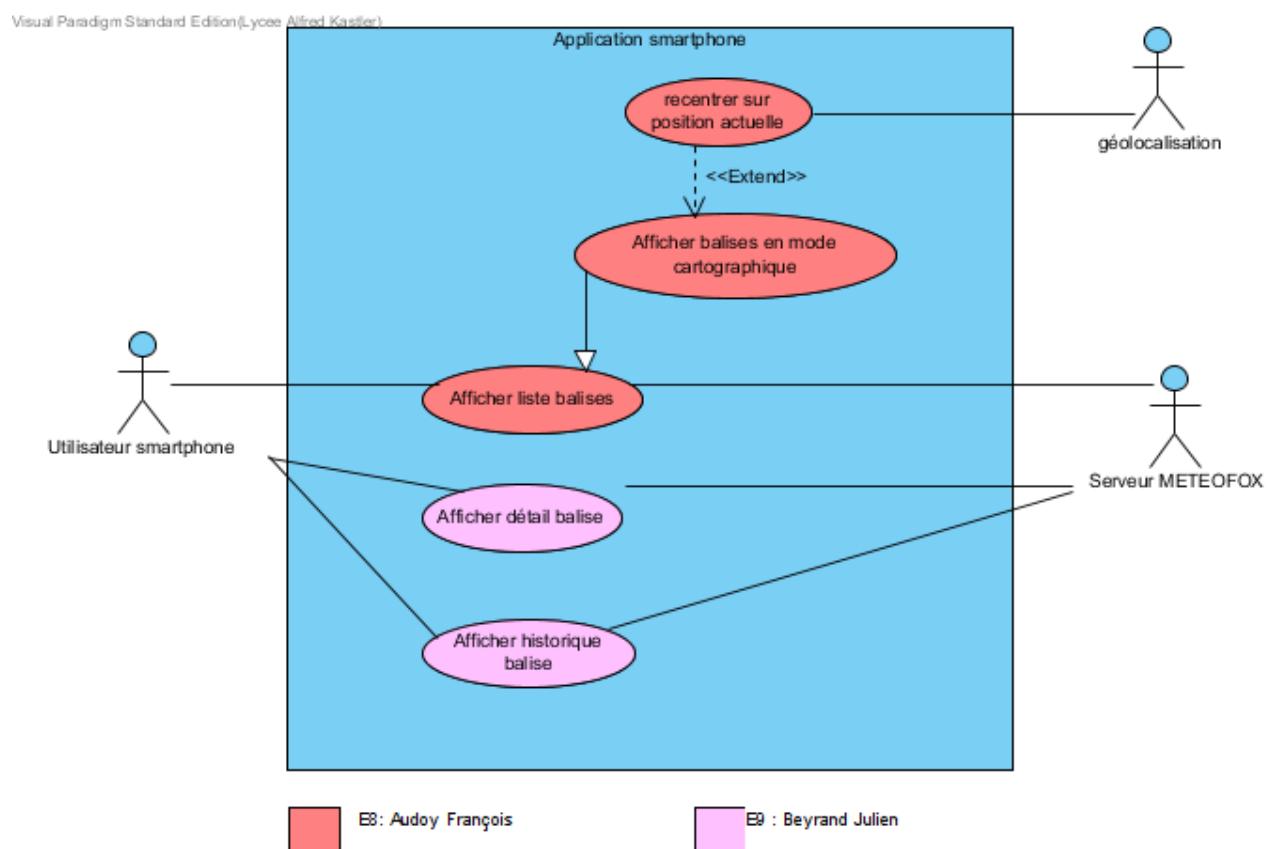


Illustration 1: Cas d'utilisation de l'application smartphone

Cas d'utilisation	Description	Contraintes
Afficher liste et état des balises	Une liste des balises est affichée, avec leur état et un résumé des données météo	
Afficher balises en mode cartographique	Positionner les balises sur une carte géographique.	Si la position est disponible
Recenter sur position actuelle	Recenter la carte sur la position actuelle	Si géolocalisation disponible.
Choisir les préférences	Choisir les préférences de l'application : <ul style="list-style-type: none"> <li>- URL du serveur METEOFOLY</li> <li>- vue par défaut</li> <li>- ...</li> </ul>	
Afficher détail balise	Affiche toutes les données pertinentes pour l'utilisateur : <ul style="list-style-type: none"> <li>- localisation</li> <li>- données météo actuelles</li> <li>- résumé sur la période récente</li> </ul>	Présenter une rose des vents synthétisant la vitesse et la direction des vents sur la dernière heure écoulée.
Afficher historique balise	Afficher les données météo sous forme de graphes temporels.	Utilisation de la librairie <b>AChartEngine</b> . <ul style="list-style-type: none"> <li>- permettre la navigation dans le temps (exploration des historiques)</li> </ul>

### 9.1.2) Principe général

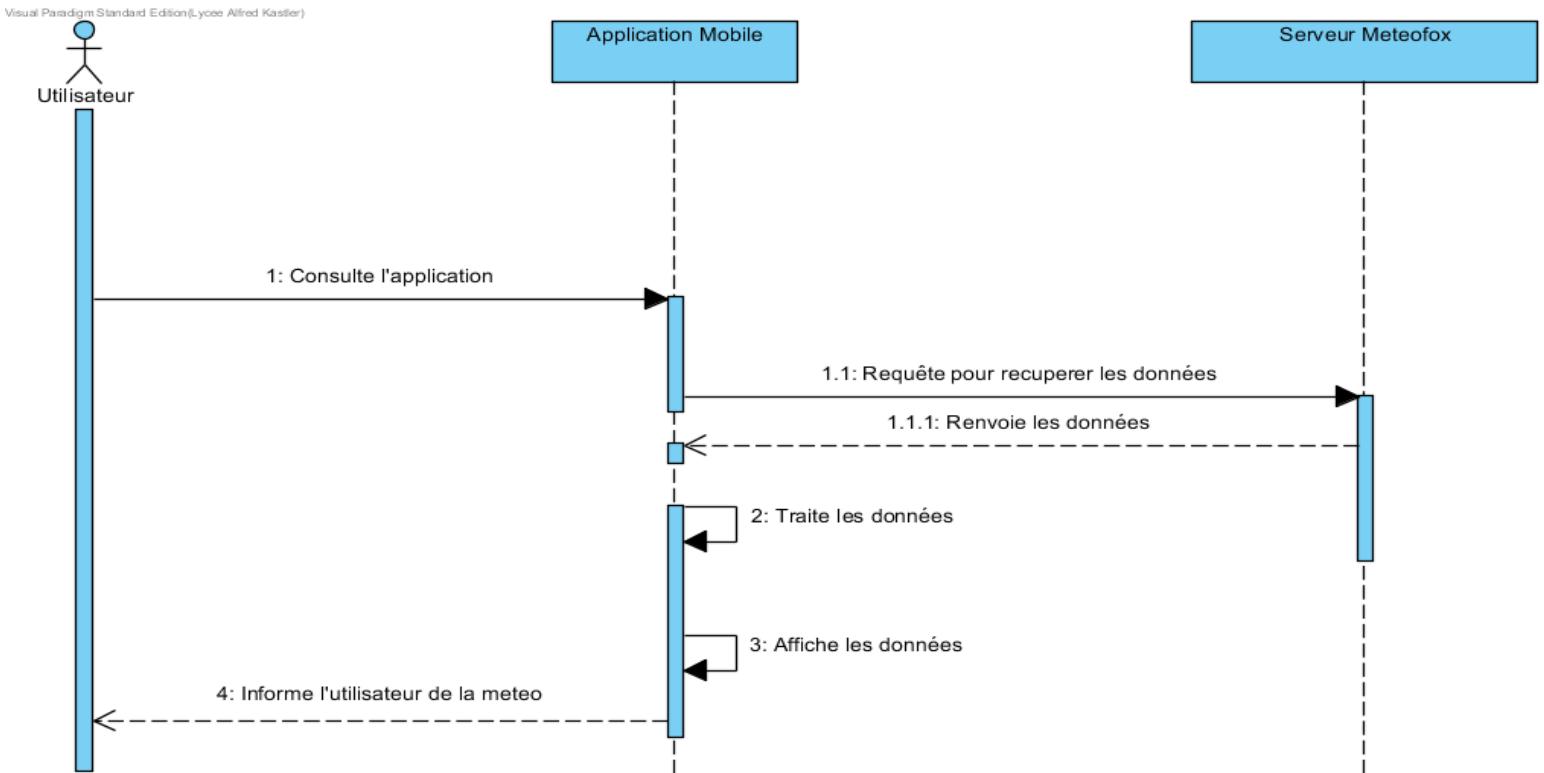


Illustration 2: Diagramme de séquence: principe de fonctionnement de l'application smartphone

## 9.2) Mise en œuvre

### 9.2.1) Technique / Technologies utilisées

Les techniques et technologies utilisées sont :

- Des tablettes sous Android
- Le langage Java pour Android
- Android studio

Android est un système d'exploitation permettant de faire fonctionner des smartphones et des tablettes.

### 9.2.2) Mise en œuvre

Android studio est l'IDE officiel de Google, permettant de développer facilement des applications Android .

Pour prendre en main ce logiciel, j'ai suivi le tutoriel proposé par le site officiel d'Android Studio.

### 9.3) Conception détaillée

#### 9.3.1) Diagramme de classe

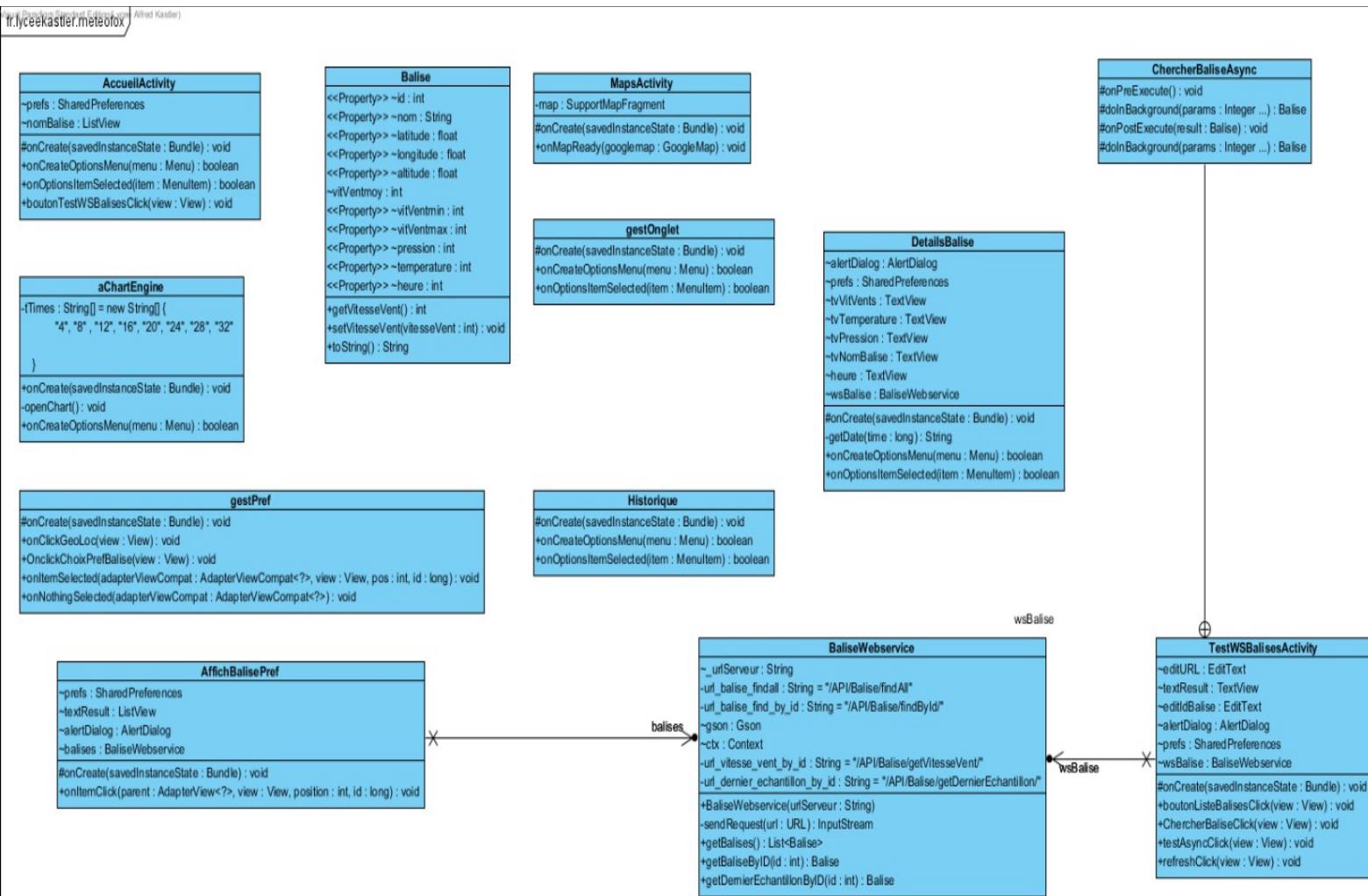


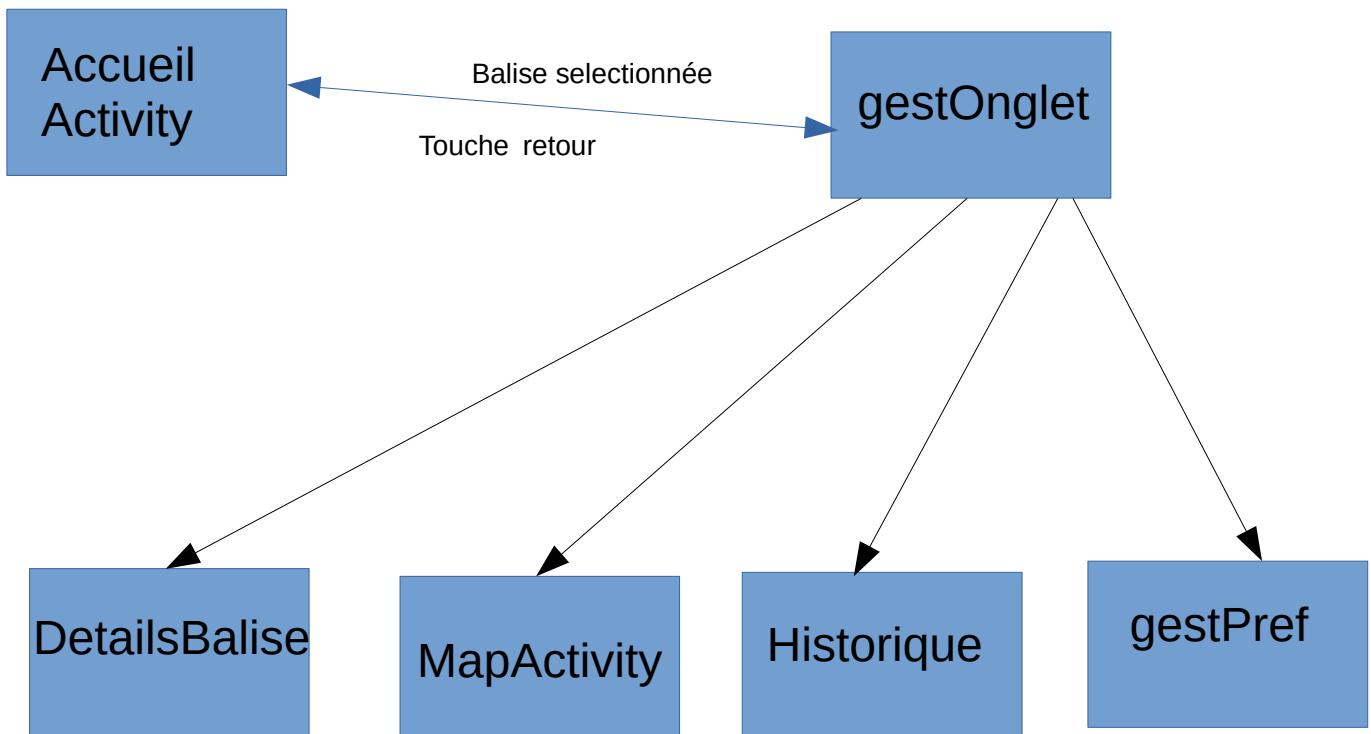
Illustration 3: Diagramme de classe de l'application Android

On s'y perd un peu, car sur le même diagramme se mélagent :

- des classes d'activités
  - des classes de modèle (Balise)
  - des classe utilitaires (ChercherBaliseAsync, aChartEngine)
- il faudrait y mettre un peu d'ordre et être capable d'expliquer le rôle de ces différentes classes

## 9.4) Implémentation

### 9.4.1) Détails de l'application



Dessin 1: Schéma du fonctionnement de l'application

Lors de l'ouverture de l'application, la vue l'AccueilActivity s'affiche montrant une liste de balises sélectionnables. Pour passer à la suite, il est nécessaire de sélectionner une balise. Une fois la sélection faite, l'activité gestOnglet se lance. Cela permet d'avoir des onglets qui simplifient la navigation dans l'application, pour passer d'activité en activité.

Les activités me concernant sont :

- AccueilActivity
- gestOnglet
- MapActivity
- gestPref

Pour commencer, la classe Balise est la classe qui permet de définir ce qu'est une balise, et ainsi de pouvoir récupérer les informations du webservice.

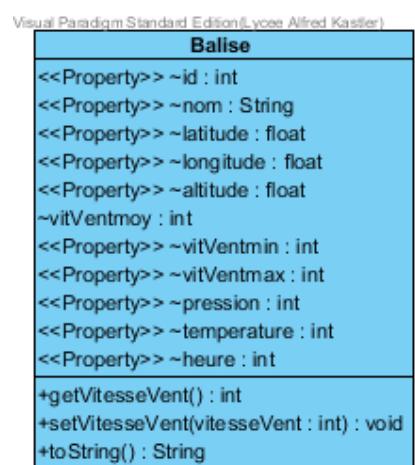
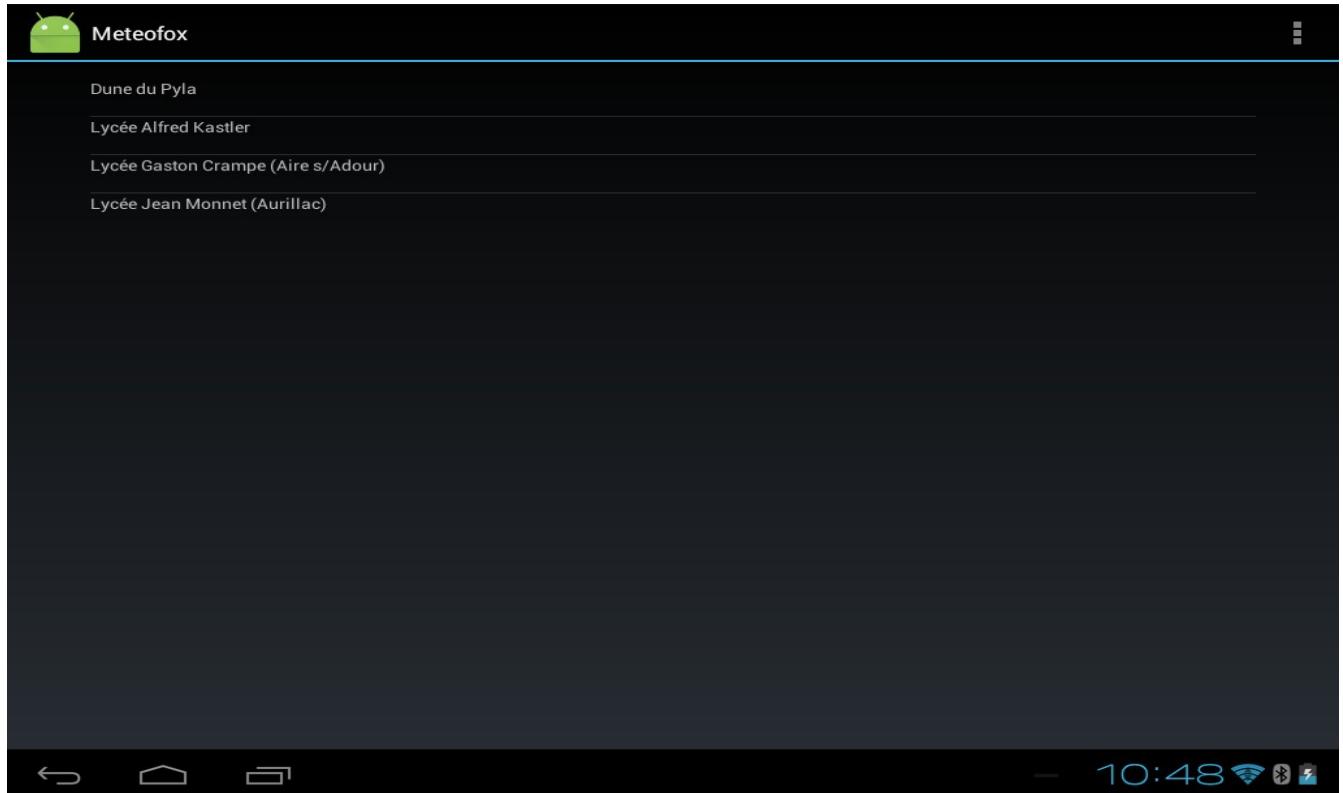


Illustration 4: Classe balise

Descriptif (sous système, ...)	
Auteur(s) : E8	Page 114 / 130

#### 9.4.2) AccueilActivity



Dessin 2: Vue de l'accueilactivity

L'activité AccueilActivity affiche la liste des balises et permet d'en sélectionner une pour afficher plus d'informations.

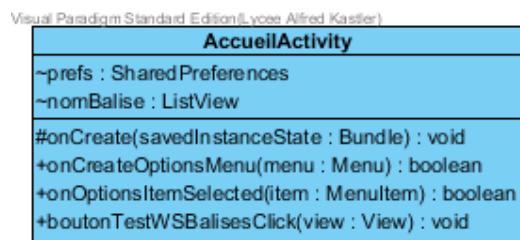


Illustration 5: Classe AccueilActivity

```
// Obtention de la liste des balises par le webservice
List<Balise> tablBalises = balises.getBalises();
//Creation d'un adaptateur de données stocker sous forme de liste generique

ArrayAdapter<String> strBalise = new ArrayAdapter<String>(nomBalise.getContext(),
R.layout.montexte);

for(Iterator<Balise> i = tablBalises.iterator(); i.hasNext();) {

    Balise balise = i.next();
    // On affiche la balise courante
    strBalise.add(balise.getNom() + balise.getId());
}
nomBalise.setAdapter(strBalise);
```

Tableau 1: extrait de code de la classe AccueilActivity

tablBalises récupère les informations des balises par le webservice.

strBalise reçoit le nom et l'ID des balises.

nomBalise récupère strBalise pour pouvoir afficher les informations qu'il contient.

#### 9.4.3) GestOnglet



Dessin 3: Onglets

Cette activité permet de gérer les onglets qui simplifient la navigation dans l'application.



Illustration 6: Classe gestOnglet

```
TabHost th1 = getTabHost();  
  
//ajout d'onglet  
th1.addTab(th1.newTabSpec("Balise").setIndicator("Balise").setContent(new Intent(this ,DetailsBalise.class)));  
th1.addTab(th1.newTabSpec("Map").setIndicator("Map").setContent(new Intent(this ,MapsActivity.class)));  
th1.addTab(th1.newTabSpec("Preference").setIndicator("Preference").setContent(new Intent(this ,gestPref.class))));  
th1.addTab(th1.newTabSpec("Historique").setIndicator("Historique").setContent(new Intent(this ,aChartEngine.class))));  
//Onglets par défaut  
th1.setCurrentTab(0);
```

Tableau 2: Gestion d'onglet

Le composant TabHost est le conteneur qui permet la création et la gestion d'onglets.

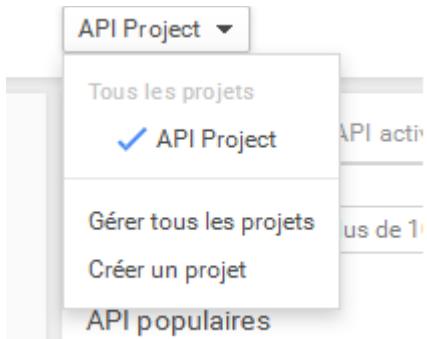
La fonction addTab associée au TabHost permet de créer un onglet, de lui donner un nom et de définir l'activité qui lui est associée.

#### 9.4.4) MapsActivity

Pour l'utilisation d'une GoogleMap, il est nécessaire de récupérer une clé API.

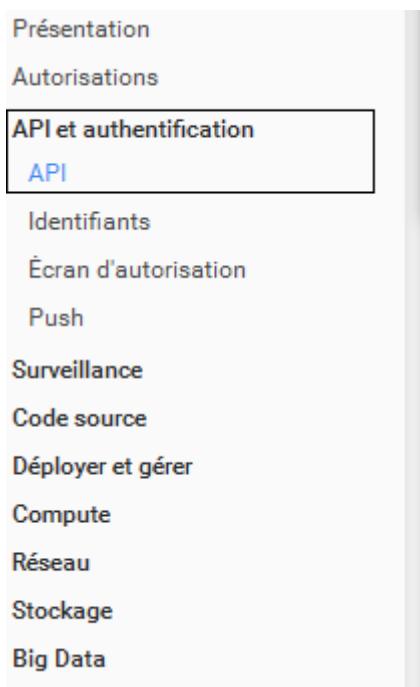
Pour la récupérer, il faut :

- un compte Google
- aller sur la console API de Google (<https://code.google.com/apis/console> )
- s'authentifier avec son compte Google
- créer un nouveau projet



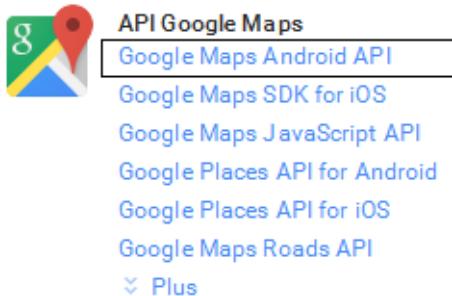
Dessin 4: création d'un nouveau projet

- cliquer sur « API et authentification » et sélectionner « API »



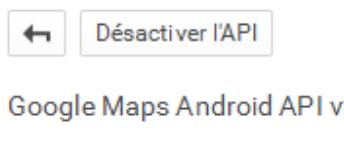
Dessin 5: API et authentification

- cliquer sur « Google Maps Android API »



Dessin 6: Google Maps Android API

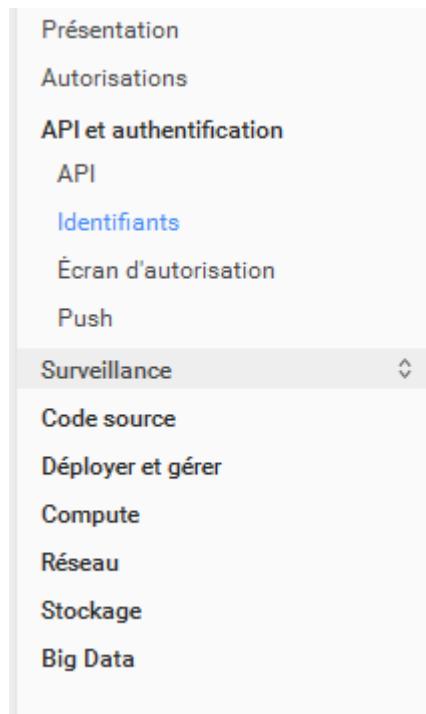
- activer le service Google Maps v2



Dessin 7: Activation du service Google Maps v2

Descriptif (sous système, ...)
Auteur(s) : E8

- cliquer sur « Identifiants »



Dessin 8: Identifiants

Ainsi, on peut copier la clé API.

The screenshot displays a table with the following information:

Clé pour les applications Android	
Clé de l'API	[REDACTED]
Applications Android	7E:C8:F0:06:4D:49:95:B2:57:05:5F:27:B1:F8:0C:D6:32:1D:F5:D3;fr.lyceekastler.meteofox
Date d'activation	12 mars 2015 06:56:00
Activée par	francois.audoy@gmail.com ( <b>vous</b> )

Buttons at the bottom:

- Modifier les applications Android autorisées
- Regénérer la clé
- Supprimer

Dessin 9: Récupération de la clé API

Descriptif (sous système, ...)	
Auteur(s) : E8	Page 120 / 130

Une fois cette clé récupérée, la coller dans une balise meta-data dans le manifeste.

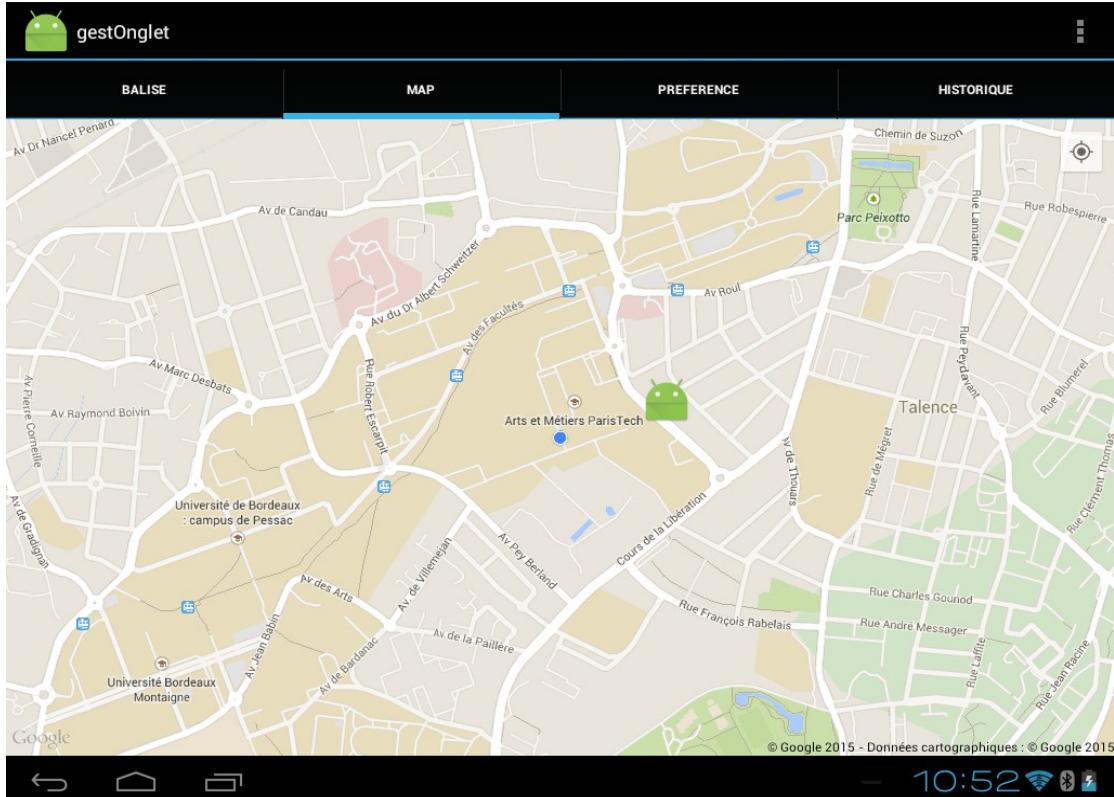
```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="clé API" />
```

Tableau 3: Ajout de la clé API

Une fois la clé API rentrée dans le manifeste de l'application, il suffira de créer une MapsActivity

```
Visual Paradigm Standard Edition (Lycée Alfred Kastler)
MapsActivity
-map : SupportMapFragment
#onCreate(savedInstanceState : Bundle) : void
+onMapReady(googleMap : GoogleMap) : void
```

Illustration 7: MapsActivity



Dessin 10: GoogleMap vue centrée

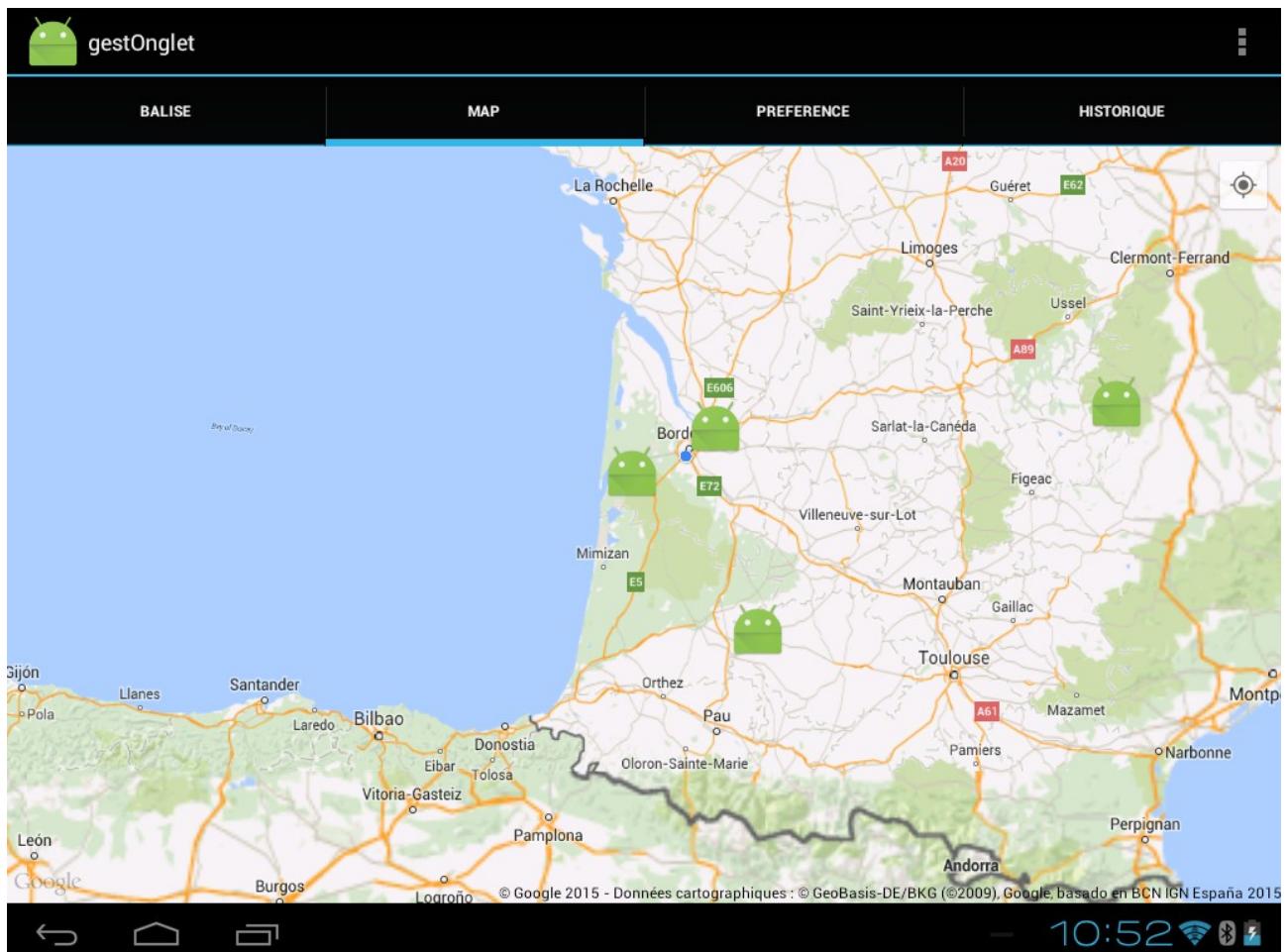
Descriptif (sous système, ...)	
Auteur(s) : E8	Page 121 / 130

Les extraits de codes suivants se situent dans la fonction onMapReady (Quand la carte est prête) :

```
//recentrer la carte sur l'utilisateur
googlemap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(44.801938,
-0.603685), 16));
```

Tableau 4: Positionnement de la camera sur la position du lycée

La fonction moveCamera permet de positionner la GoogleMap au moment de son apparition sur les longitude et latitude précisées (pour ce cas, la caméra est positionnée sur le Lycée Kastler).



Dessin 11: Vue d'ensemble de la position des balises

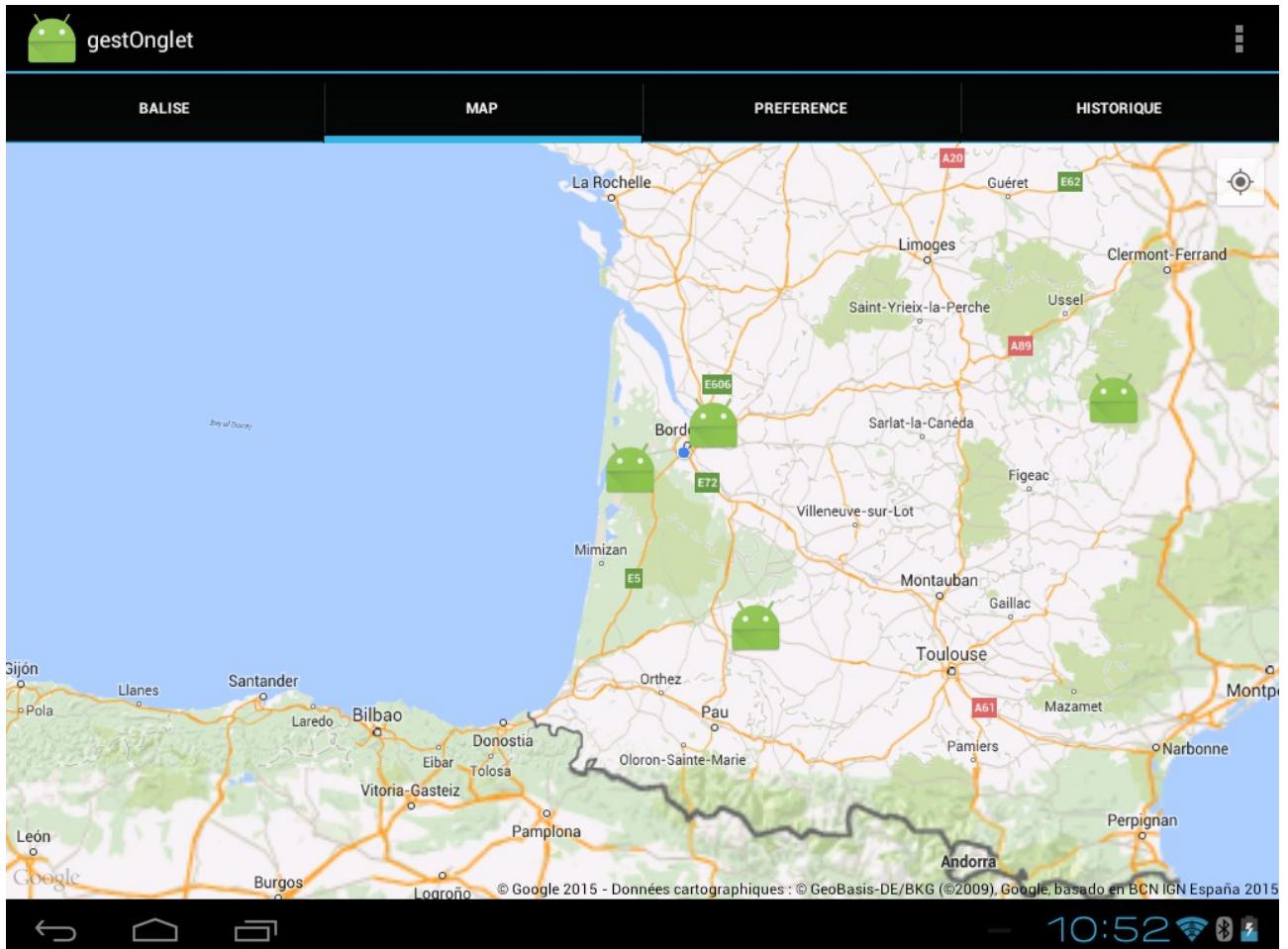
```
googlemap.addMarker(new
MarkerOptions().icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_launcher)).anc
hor(0.0f, 1.0f)
    .position(new LatLng(balise.getLatitude(), balise.getLongitude())));
//lycee kastler
```

Tableau 5: Ajout de toute les balises sur la carte

Ce code permet de créer des marqueurs à la position (longitude et latitude) inscrite dans la base de données.

## 9.5) Test de Validation

Dans ce test, je vais mettre en avant l'utilisation de la longitude et latitude des balises contenues dans la base de données du serveur meteofox.



Dessin 12: Résultat obtenu après le test

Sur le débogueur, on voit que l'on récupère le début de l'url du serveur meteofox contenu dans les « préférences » (c'est une fonction qui permet d'enregistrer des informations dans la mémoire du smartphone ).

Descriptif (sous système, ...)
Auteur(s) : E8

```

▼ └─ url = {java.lang.String@830020813984} "http://meteofox.lyceekastler.fr/meteofox/web/app\_dev.php"
  ► └─ value = {char[56]@830020813848}
    └─ hashCode = 0
    └─ offset = 0
    └─ count = 56
  
```

Dessin 13: Début de l'URL

On a du mal à comprend à quoi sert précisément cette URL.  
Quel est le lien avec ce qui a été présenté avant ?

Ici, on voit le début de l'adresse url donnée par la variable url et les fins d'adresses contenues dans des variables statiques précisées dans la classe BaliseWebservice.

```

▼ └─ wsBalise = {fr.lyceekastler.meteofox.BaliseWebservice@830022689336}
  ► └─ _urlServeur = {java.lang.String@830020813984} "http://meteofox.lyceekastler.fr/meteofox/web/app\_dev.php"
    └─ ctx = null
  ► └─ gson = {com.google.gson.Gson@830022689376} "{serializeNulls:false,factories:[Factory[typeHierarchy=com.google.gso
  ► └─ url_balise_find_by_id = {java.lang.String@830020815248} "/API/Balise/findByld/""
  ► └─ url_balise_findall = {java.lang.String@830020815152} "/API/Balise/findAll/""
  ► └─ url_dernier_echantillon_by_id = {java.lang.String@830020815456} "/API/Balise/getDernierEchantillon/""
  ► └─ url_vitesse_vent_by_id = {java.lang.String@830020815344} "/API/Balise/getVitesseVent/""
  
```

Dessin 14: Début + fin de l'adresse url

On peut voir la réception des informations des balises contenues dans la base de données.

```

▼ └─ listeBalises = {java.util.ArrayList@830021519336} size = 4
  └─ 0 = {fr.lyceekastler.meteofox.Balise@830021050648} "fr.lyceekastler.meteofox.Balise@411c2118"
  └─ 1 = {fr.lyceekastler.meteofox.Balise@830022404912} "fr.lyceekastler.meteofox.Balise@4130cb30"
  └─ 2 = {fr.lyceekastler.meteofox.Balise@830021894296} "fr.lyceekastler.meteofox.Balise@41290098"
  └─ 3 = {fr.lyceekastler.meteofox.Balise@830022311280} "fr.lyceekastler.meteofox.Balise@412f5d70"
  
```

Dessin 15: Reception des information des balises

par quel mécanisme ces informations ont-elles été obtenues ?

Dans votre dossier, il manque un scénario ou un diagramme de séquence qui montrerait les échanges concrets qui ont lieu entre les objets ou sous-systèmes (il y a un diagramme mais il est très général, on ne fait pas le lien avec le code)

```
▼ 0 = {fr.lyceekastler.meteofox.Balise@830021050648} "fr.lyceekastler.meteofox.Balise@411c2118"
  ▼ nom = {java.lang.String@830022252400} "Dune du Pyla"
    ► value = {char[16]@830022243784}
      hashCode = 0
      offset = 0
      count = 12
      heure = 0
      id = 1
      latitude = 44.554905
      longitude = -1.2393951
      altitude = 40.0
      pression = 0
      temperature = 0
      vitVentmax = 0
      vitVentmin = 0
      vitVentmoy = 0
```

Dessin 16: Information reçues pour la balise Dune du Pyla

## 9.6) Déploiement

Pour déployer l'application, il est nécessaire de posséder un compte Google et de l'associer à la console développeur de Google pour un coût de 25€. Par la suite, il faut :

1. Renseigner son nom (il sera visible dans le Playstore)
2. Préciser les différents moyens pour être contacté
3. Se connecter au tableau de bord Google Play
4. Aller dans la section « Publier »
5. Indiquer le titre de l'application et préciser la langue
6. Importer le fichier APK
7. Donner des informations sur l'application (titre, description, 2 captures d'écran au minimum, l'icône, etc)
8. Définir si l'application sera payante ou non
9. Si l'application sera payante, indiquer le compte marchand.

### 9.7) Conclusion

Le projet n'a pas été réalisé dans sa totalité, les parties fonctionnelles sont :

- L'IHM
- L'affichage des informations d'une balise
- L'affichage des balises en mode cartographique
- L'affichage de la liste de balises

Les parties non fonctionnelles sont :

- Recentrage de la carte sur la position de l'utilisateur
- L'affichage de l'historique des balises
- Le choix des préférences de l'application
- L'affichage de l'état des balises.

**10) Étudiant 9 : Rémi Amand**

---

Descriptif (sous système, ...)
--------------------------------

Auteur(s) : E8
----------------

Page 128 / 130
----------------

## 11) Conclusion

---

Descriptif (sous système, ...)
--------------------------------

Auteur(s) : E8
----------------

Page 129 / 130
----------------

Descriptif (sous système, ...)

Auteur(s) : E8

Page 130 / 130