

# Rapport de stage

## **Station météo pour Raspberry**

ZIPPARI David

Concepteur développeur informatique - Stage en entreprise

Tuteur : G.GOUIN

# REMERCIEMENT

Je tiens tout particulièrement à remercier :

MR GUEVAR, mon formateur, qui a su faire preuve d'écoute et de diplomatie quand j'avais du mal à appréhender certains concepts abstraits de conception et de programmation.

Mon tuteur qui m'a guidé durant mon stage ainsi que les stagiaires qui m'ont soutenu durant ce dernier.

Ma compagne, qui a su faire preuve de compréhension durant cette année de formation.

# SOMMAIRE

INTRODUCTION .....	4
1 PRESENTATION .....	5
1.1 L'entreprise .....	5
1.1.1 Nom et raison sociale .....	5
1.1.2 Adresse du siège social .....	5
1.1.3 Historique .....	5
1.1.4 Activité principale .....	5
1.1.5 Contact.....	5
1.2 Le stagiaire .....	6
1.2.1 Profil et expérience .....	6
1.2.2 Projet réalisé durant la formation AFPA .....	6
1.2.3 Ambition personnelle .....	6
1.3 Le projet.....	7
1.3.1 Cadre du projet .....	7
1.3.2 Environnement de travail .....	7
1.3.4 Cahier des charges (validé par le client) .....	11
2 ETUDE PREALABLE .....	13
2.1 Analyse fonctionnel.....	13
2.1.1 Besoins fonctionnels .....	13
2.2.2 Besoins non fonctionnels.....	14
2.2.3 Cas d'utilisation : Diagramme de cas d'utilisation .....	15
2.2.3 Cas d'utilisation : Scenarii .....	16
3 CONCEPTION.....	18
3.1 Conception architecturale.....	18
3.1.1 MCD et MLD .....	18
3.1.2 Diagramme de classe .....	18
3.1.3 Schéma de deployment .....	23
3.2 Maquettage (application cliente).....	24
3.2.1 Vue d'ensemble .....	24
3.2.2 Gestion des enregistrements et des alertes.....	25
3.2.3 Affichage du graphique de données.....	26
4 DEVELOPPEMENT .....	27
4.1 Solutions techniques.....	27
4.1.1 Une application flexible 1 : Développement en couche.....	27
4.1.2 Une application flexible 2 : Le Pattern DAO.....	31
4.1.3 Une application évolutive : Classe abstraite et polymorphisme .....	32
4.1.4 Gérer l'enregistrement et les alertes : Pattern Observer.....	34
4.1.5 Stocker et manipuler les données : MySQL et JDBC.....	37
4.1.6 Afficher des graphiques: JSP, JSON, Ajax et jqPlot.....	40
4.2 Récapitulatif.....	43
5 CONCLUSION .....	44
GLOSSAIRE .....	45
SOURCES .....	47
ANNEXE .....	48

# INTRODUCTION

Je m'appelle David ZIPPARI, j'ai 34 ans et je suis en formation à l'AFPA. Dans le cadre de ma formation, j'ai effectué un stage en entreprise du 20 juin 2016 au 12 septembre 2016.

Le stage a été réalisé en télétravail, au sein de l'entreprise « La ferme des Loges », une exploitation agricole tenue par Mr Gouin qui fut mon tuteur.

Le projet de mon stage était de créer une application pour enregistrer le climat de serre agricole ; développé en Java, et hébergé par un Raspberry.

A travers ce rapport vous découvrirez les étapes qui m'ont permis de réaliser mon projet ; de l'étude du besoin à la solution technique apportée.

Bonne lecture à vous...

---

My name is David ZIPPARI, I'm 34 and I'm training at the AFPA. As part of my professional training, I completed a work placement from 20th June 2016 to 12th September 2016.

My work placement has been performed within the company "Farm lodges", a bio farm managed by Mr. Gouin who was my tutor.

My work placement project was a climate monitoring program for Greenhouses, developed in JAVA and supported by a Raspberry.

Through this report you will discover the realization of my project; since the study of the needs to the provided and applied technical solutions.

Good reading...

# 1 PRESENTATION

## 1.1 L'entreprise

### 1.1.1 Nom et raison sociale

L'entreprise s'appelle « La ferme des Loges ». C'est une S.A.R.L créée en 2015 et enregistrée sous le code NAF 0113Z.

### 1.1.2 Adresse du siège social

Située « aux Loges » (lieu-dit) sur la commune de MORTAIN-BOCAGE dans la Manche (50140).

### 1.1.3 Historique

L'entreprise « La ferme des Loges » a été créée par Monsieur Guillaume GOUIN le 01/11/2015.

Ancien informaticien, Mr Gouin a souhaité se reconvertir dans le domaine agricole, il a ainsi acheté quelques hectares de terrain afin d'y développer plusieurs activités agricoles en parallèles : maraichage, élevage d'ovins...

### 1.1.4 Activité principale

L'activité principale de l'exploitation est le maraichage BIO notamment la culture de rhubarbes et de cucurbitacées.

### 1.1.5 Contact

Mr GOUIN (mon tuteur) était mon unique contact lors de mon stage.

## 1.2 Le stagiaire

### 1.2.1 Profil et expérience

Passionné d'informatique depuis plus de 20 ans, j'ai fait mes débuts dans la programmation dans les années 2000, en suivant deux formations dont l'intitulés étaient : « concepteur de site E-commerce » et « multimédia et support numérique ».

A l'époque nous avons développé un site E-commerce en PHP avec une base de données MySQL. L'IHM était faite à base d'HTML, de CSS et de Flash.

J'ai travaillé quelques mois pour une personne qui crée des sites internet avant de trouver un emploi fixe qui n'était pas dans le domaine de l'informatique.

### 1.2.2 Projet réalisé durant la formation AFPA

Durant cette année de formation à travers la réalisation d'applications nous avons abordé plusieurs technologies et langages de programmation.

Nous avons étudié la technologie JSP avec les servlets, et JSF en concevant et développant un catalogue de pièces détachées ainsi qu'un client pour ce catalogue.

Le client était une plateforme E-commerce, avec la gestion de sessions (login, caddie), ainsi qu'une interface graphique complète.

### 1.2.3 Ambition personnelle

J'apprécie énormément le fait d'avoir pu travailler sur un Raspberry. Dans un futur proche je vais continuer à développer mon application qui me servira de vitrine, tout en cherchant un poste dans la conception et le développement.

Plus tard j'aimerais développer des applications dans le milieu professionnel (agricole, médical...), en créant pourquoi pas ma propre structure.

## 1.3 Le projet

### 1.3.1 Cadre du projet

Dans le cadre de son exploitation agricole, et de la gestion du climat de ses installations maraichères (serres et salle de croissance), Mr Gouin désire une application fonctionnant sur Raspberry, captant et stockant sur une base de données les différentes données climatiques, telles que la température, l'hygrométrie...

N'ayant jamais utilisé de Raspberry, la phase de recherche fut relativement longue. J'ai dû apprendre à configurer le Raspberry (configuration du réseau, installation de programmes...), et à brancher les capteurs en utilisant des composants électroniques spécifiques (résistances, puces électroniques...).

### 1.3.2 Environnement de travail

#### Le matériel : Raspberry et capteurs

Pour le projet, j'ai choisis et configuré moi-même les capteurs (température et humidité du sol ; le capteur d'humidité relative ayant été reçu 2 semaines avant la fin de la formation...)

#### Le Raspberry : Modèle PI 3 B.

Doté de 1Go de mémoire vive, il possède entre autre un port Ethernet (utilisé pour le projet) ainsi qu'un contrôleur wifi et Bluetooth.

Le Raspberry est doté entre autre d'un port GPIO, permettant de relier des capteurs, des relais et bien d'autres périphériques (camera, écran LCD...).

Le système d'exploitation du Raspberry est basé sur Debian (une version de Linux), et se nomme « Raspbian ».

#### Capteur de température : Ds18b20.

Pour les besoins du projet nous avons choisis un capteur protégé par une coque en métal et pourvu d'un câble déjà soudé au module

Ce capteur a une précision  $\pm 0.5^{\circ}\text{C}$ .<sup>^</sup>



*Photo du capteur Ds18b20*

Capteur d'humidité du sol : YL69 + SBT4447

Composé de deux électrodes que l'on enfouit dans le sol.  
Ce capteur a une précision de  $\pm 2\%$ .



*Capteur d'humidité du sol*

Capteur d'humidité relative (air) : Honeywell IH5030

Ce capteur a une précision de  $\pm 3\%$ .



*Capteur d'humidité relative*



## Le Raspberry et Java : Plateforme Java et JDK

Oracle décline l'environnement Java en plusieurs plateformes dont Java Me, dédiée aux périphériques mobiles ; qui permet un accès direct au port GPIO du Raspberry.

Néanmoins, Java Me étant une version « light » de Java, et le Raspberry étant plus proche d'un ordinateur que d'un smartphone, nous avons choisi de développer l'application grâce à Java Se.

Le système d'exploitation du Raspberry embarque le JDK de Java, permettant ainsi de compiler et de faire tourner des applications Java sur le Raspberry.

## Le Raspberry et Eclipse : le plugin « LaunchPi »

Le plugin LaunchPi pour Eclipse, permet de compiler et de lancer l'application directement sur le Raspberry.

Grâce à ce plug in, on évite les manipulations répétitives de devoir exporter l'application compilée sur le Raspberry, et de là le lancer depuis la console.

*Note : voir la notice fournie en annexe du rapport.*

## 1.3.3 Gestion de projet

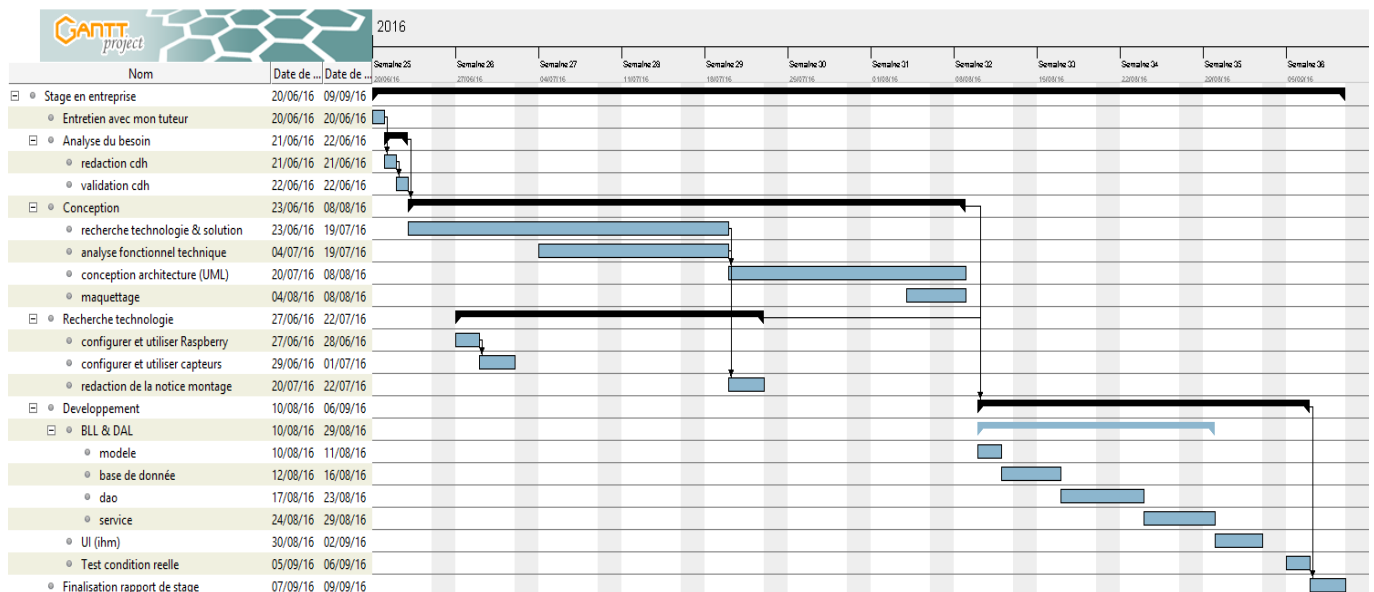
### Planification du projet

Voici un **diagramme de Gantt** détaillant point par point les étapes, depuis la conception jusqu'au déploiement et phase de test de l'application.

On remarquera que le stage a été découpé en quatre tâches principales :

- Etude préalable
- Conception
- Développement

*Note : La phase de recherche des technologies ; fut relativement longue car j'ai dû choisir et configurer moi-même le matériel.*

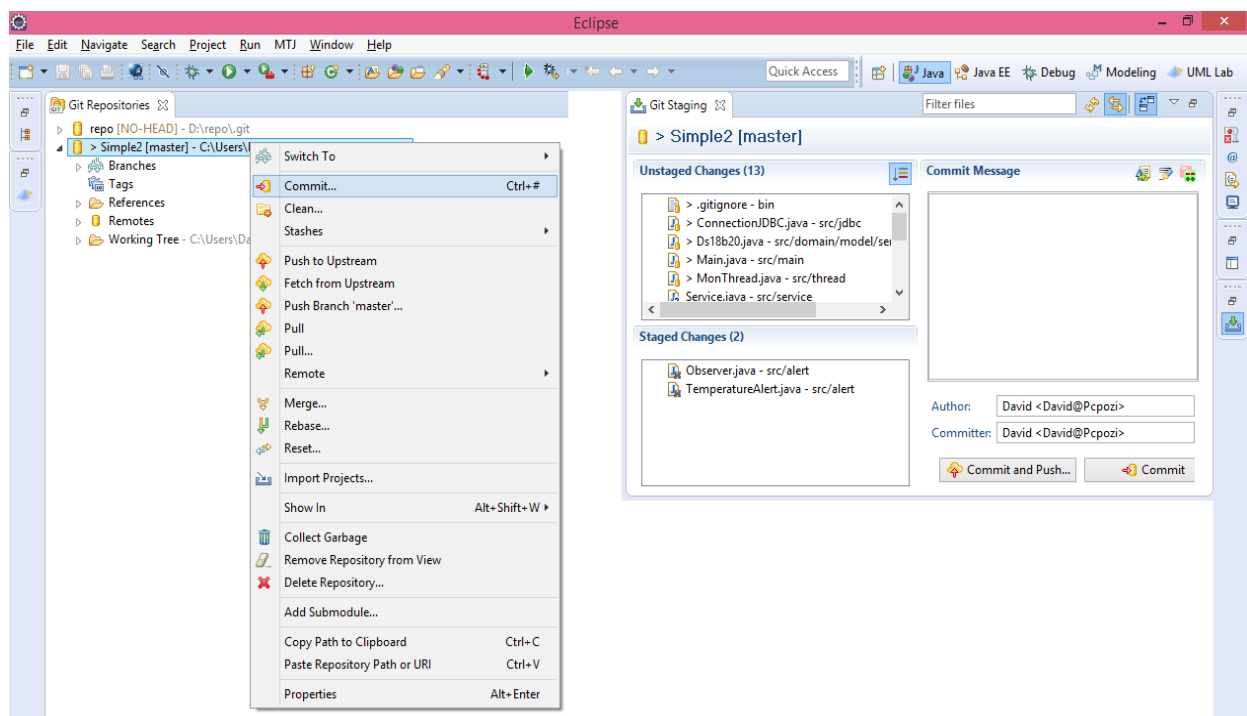


Le diagramme de Gantt

## Utilisation de GitHub

Afin de partager le code de l'application avec mon tuteur; j'ai installé le plugin d'Eclipse permettant de créer des dépôts (repository) et entre autre de mettre en ligne son code en gardant sa structure.

Lien du GitHub du projet : <https://github.com/PopoFR/Stage>



GitHub dans Eclipse

## 1.3.4 Cahier des charges (validé par le client)

En vue de gérer le micro climat nécessaire à l'amélioration de la productivité des serres et du local de croissance de son exploitation, Mr Gouin désire un logiciel permettant à un Raspberry d'enregistrer dans une base de données les différentes données climatique (température, hygrométrie...).

La solution logicielle devra permettre de:

- mesurer, et enregistrer la température, l'hygrométrie de l'air, l'humidité du sol.
- visualiser les données enregistrées.
- envoyer des alertes.

### Système existant

Mr Gouin ne dispose que du matériel (Raspberry) ; le projet est donc nouveau et ne s'appuiera sur rien d'existant.

### Spécifications fonctionnelles

Le stagiaire devra fournir une solution complète (matérielle et logicielle) afin d'enregistrer la température de l'air, l'hygrométrie de l'air ainsi que le taux d'humidité du sol mesurés par les capteurs du Raspberry...

La température s'exprimera en degrés Celsius.

L'hygrométrie de l'air s'exprimera en pourcentage.

L'humidité du sol s'exprimera en pourcentage.

L'application devra permettre d'enregistrer les données de plusieurs capteurs du même type.

Les données contiendront les informations suivantes :

- La valeur de la mesure.
- La provenance : le capteur qui a enregistré la donnée.
- La localisation : l'endroit où se trouve le capteur.
- La date : le jour, l'heure, la minute et la minute de la mesure enregistrée.

Un système d'alerte devra permettre à l'application d'envoyer une alerte (ici sous forme d'email), quand un des capteurs dépasse le seuil défini dans ses propriétés.

L'interface utilisateur devra permettre de visualiser les différentes données stockées sous forme de graphiques.

Les données qui devront être affichées dans le graphique :

- La localisation
- La date
- La valeur de la mesure

Le stagiaire fournira une documentation précise, des étapes nécessaires à l'utilisation de capteurs sur le Raspberry (branchement, configuration...).

## Contraintes et règles de gestions

Les délais étant courts, l'application principale est prioritaire.

L'application principale (enregistrant les données des capteurs) sera hébergée par le Raspberry, et l'application cliente sur un serveur d'application Tomcat hébergé par un professionnel.

Les capteurs seront filaires (le matériel sera choisis par le stagiaire).

La base de données sera une base MySQL.

Au vu du nombre conséquent d'enregistrements, chaque type de donnée mesurée aura sa propre table.

La base de données ne sera pas hébergée par le Raspberry mais sur un serveur.

L'application sera développée en Java (JDK 1.7).

Logback sera utilisé pour gérer les logs.

L'application sera développée grâce à l'ide nommé « Eclipse ».

## 2 ETUDE PREALABLE

### 2.1 Analyse fonctionnelle

#### 2.1.1 Besoins fonctionnels

##### Enregistrer les différentes constantes

L'application principale (hébergée par le Raspberry) devra être capable d'enregistrer :

- La température
- l'hygrométrie de l'air
- l'humidité du sol

La température sera exprimée en °Celsius et aura une précision d'un chiffre après la virgule.

L'hygrométrie de l'air et l'humidité du sol seront exprimés en pourcentage et seront des entiers.

##### Gestion d'alerte par email

L'application principale devra être capable d'envoyer un email à un intervalle régulier si l'un des capteurs dépasse la valeur minimale ou maximale qui lui est assignée. La valeur sera paramétrable pour chaque capteur, par l'utilisateur.

##### L'Affichage des mesures stockées

L'application cliente devra permettre d'afficher sous forme de graphique les mesures effectuées par les différents capteurs.

Les graphiques devront afficher les informations suivantes :

- Valeur de la mesure
- Date du jour, heure, minutes et secondes de la mesure
- Localisation du capteur ayant effectué la mesure

## 2.2.2 Besoins non fonctionnels

### Développer une application légère

Le Raspberry disposant de ressources limitées, nous prendrons cela en compte lors du développement de l'application.

### Développer grâce à l'IDE Eclipse

Mr Gouin a travaillé pour la société qui édite le logiciel Eclipse. De plus nous avons appris durant notre année de formation à développer sur Eclipse. C'est pour ces raisons que Mr Gouin désire que le logiciel soit développé grâce à cette application.

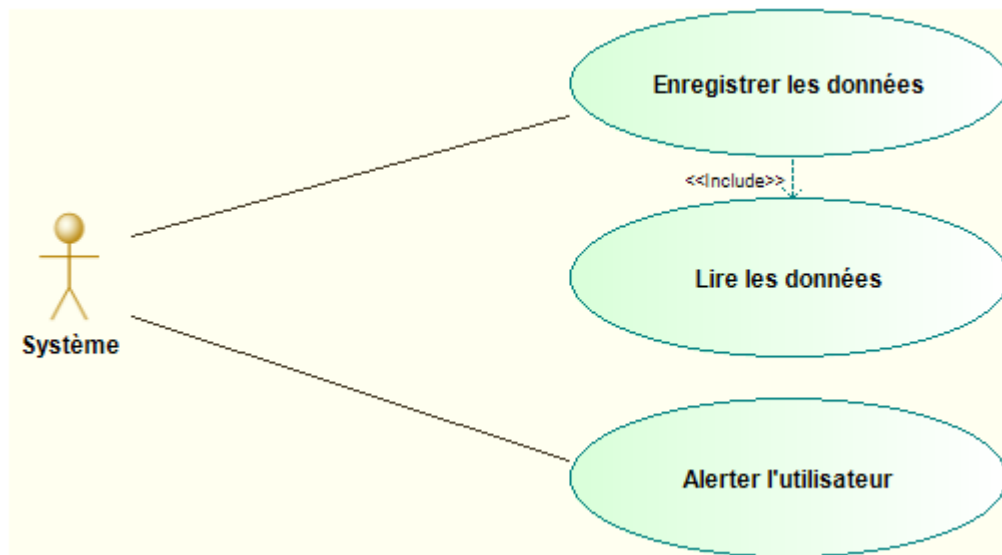
### Utiliser une base de données simple et légère

Afin de faciliter la maintenance, la base de données doit être relativement simple. De plus pour ne pas surcharger une table en particulier, il doit y avoir une table par constante mesurée.  
La base de données doit être gratuite.

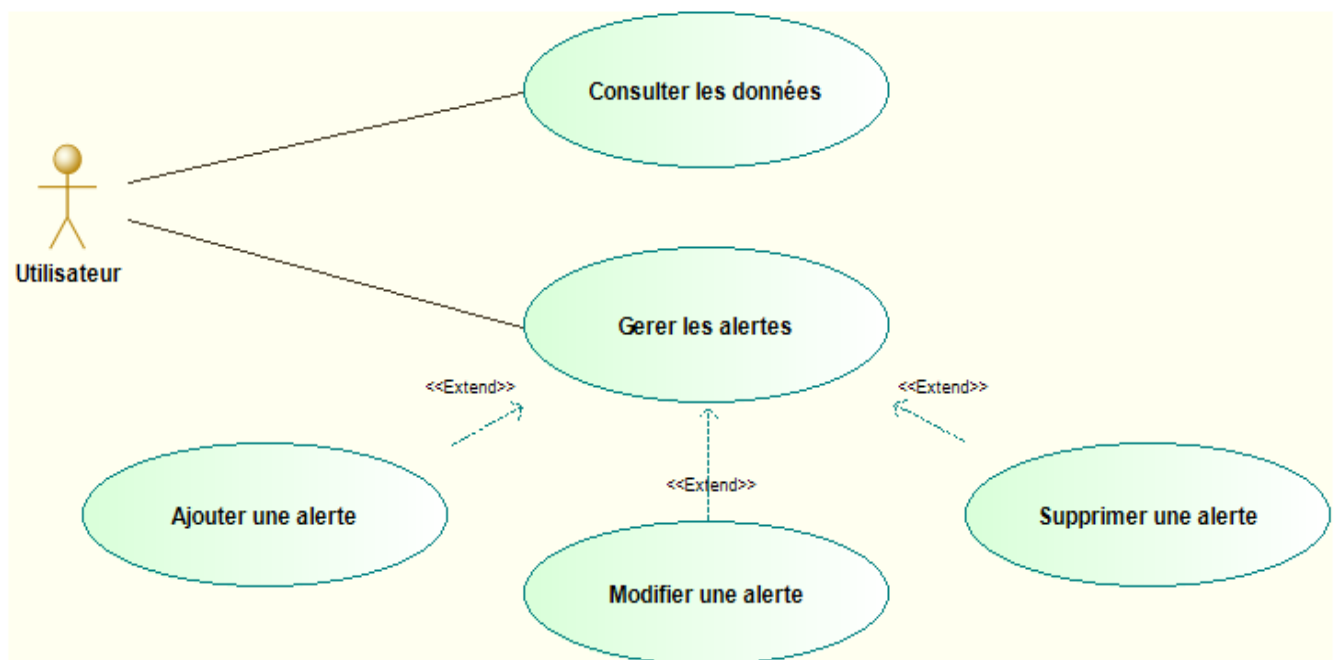
### Fournir une documentation

Mr Gouin désire une documentation la plus exhaustive possible. La documentation doit détailler le matériel nécessaire et les manipulations à effectuer, afin de pouvoir relier des capteurs au Raspberry et continuer à développer l'application.

## 2.2.3 Cas d'utilisation : Diagramme de cas d'utilisation



*Application principale*



*Application cliente*

## 2.2.3 Cas d'utilisation : Scenarii

### L'application principale

#### Cas n°1 : Lire les données

**Acteur** : Le système.

**Description** : Acquisitions et interprétations de données collectées par les capteurs.

**Précondition** : Le matériel est fonctionnel (capteurs branchés, Raspberry sous tension...).

**Démarrage** : Le thread de l'application déclenche périodiquement la lecture des données.

- Le système lit les données des capteurs.

#### Cas n°2 : Enregistrer les données.

**Acteur** : Le système.

**Description** : Lecture et enregistrement des données collectées par les capteurs.

**Précondition** : « Lire les données » (cas n°1), avoir une base de données opérationnelle.

**Démarrage** : La lecture des données déclenche l'enregistrement.

- Le système se connecte à la base de données et y enregistre les données.

#### Cas n°3 : Alerter l'utilisateur

**Acteur** : Le système.

**Description** : Alerter l'utilisateur qu'un capteur a mesuré une donnée dépassant le seuil toléré.

**Précondition** : « Lire les données » (cas n°1), avoir une base de données opérationnelle.

**Démarrage** : La lecture des données déclenche la gestion de l'alerte.

- Le système compare les données acquises durant la lecture (cas n°1), et les compares avec les seuils définis par l'utilisateur.
- le système envoie une alerte.



## L'application cliente

### Cas n°1 : Consulter les données

**Acteur :** L'utilisateur

**Description :** Lecture des données collectées par le Raspberry.

**Précondition :** Avoir « Enregistrer les données » (Raspberry - cas n°2).

**Démarrage :** Depuis la vue d'ensemble, l'utilisateur sélectionne un lieu et clique sur « Historique ».

- Le système affiche le graphique des données.

### Cas n°2 : Ajouter une alerte

**Acteur :** L'utilisateur

**Description :** Ajouter une alerte à un capteur.

**Précondition :** Avoir un capteur relié au Raspberry.

**Démarrage :** Depuis la vue d'un lieu, l'utilisateur sélectionne une mesure et coche la case « alerte ».

- Le système affiche les champs nécessaires à la création de l'alerte (seuil mini/maxi, délai entre deux alertes...)
- L'utilisateur renseigne les informations et clique sur « ajouter ».
- Le système crée une nouvelle alerte et affiche un message de succès.

### Cas n°3 : Modifier une alerte

**Acteur :** L'utilisateur

**Description :** Modifier une alerte d'un capteur.

**Précondition :** Avoir un capteur relié au Raspberry et une alerte activée.

- L'utilisateur modifie les informations de l'alerte et clique sur « ajouter ».
- Le système enregistre les modifications et affiche un message de succès.

### Cas n°4 : Supprimer une alerte

**Acteur :** L'utilisateur

**Description :** Supprimer une alerte d'un capteur.

**Précondition :** Avoir un capteur relié au Raspberry et une alerte attribuée.

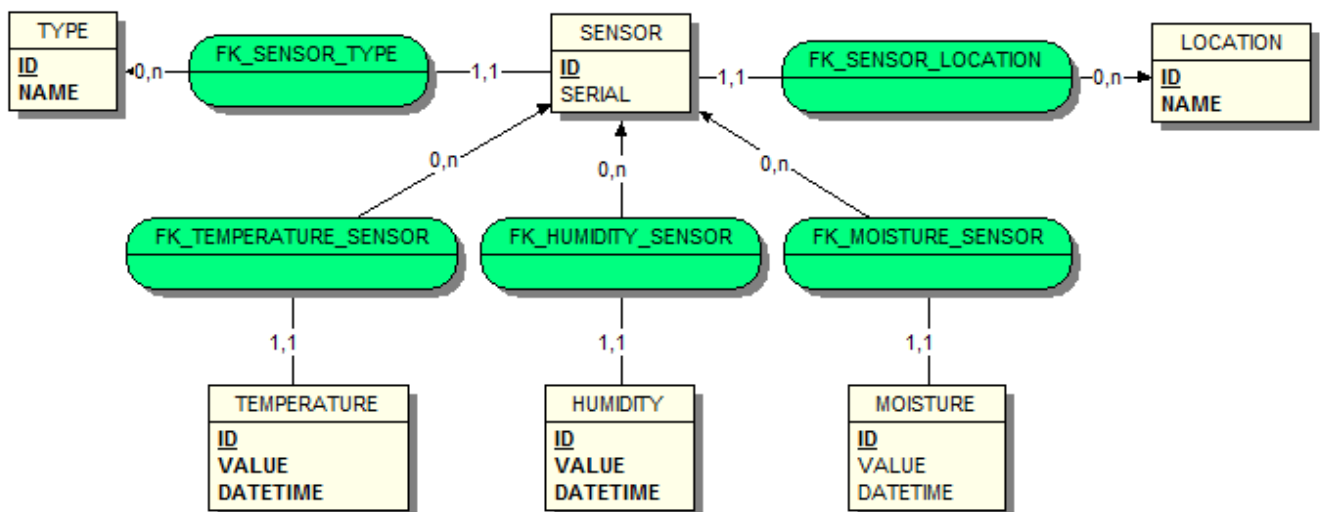
- L'utilisateur sélectionne une alerte et décoche la case.
- Le système enregistre la suppression et affiche un message de succès.

# 3 CONCEPTION

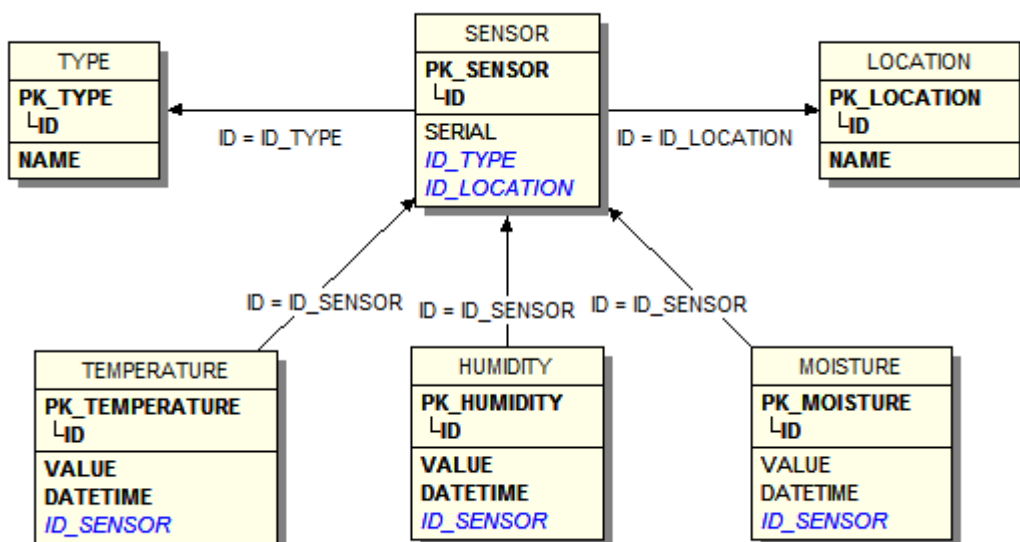
## 3.1 Conception architecturale

### 3.1.1 MCD et MLD

Application principale



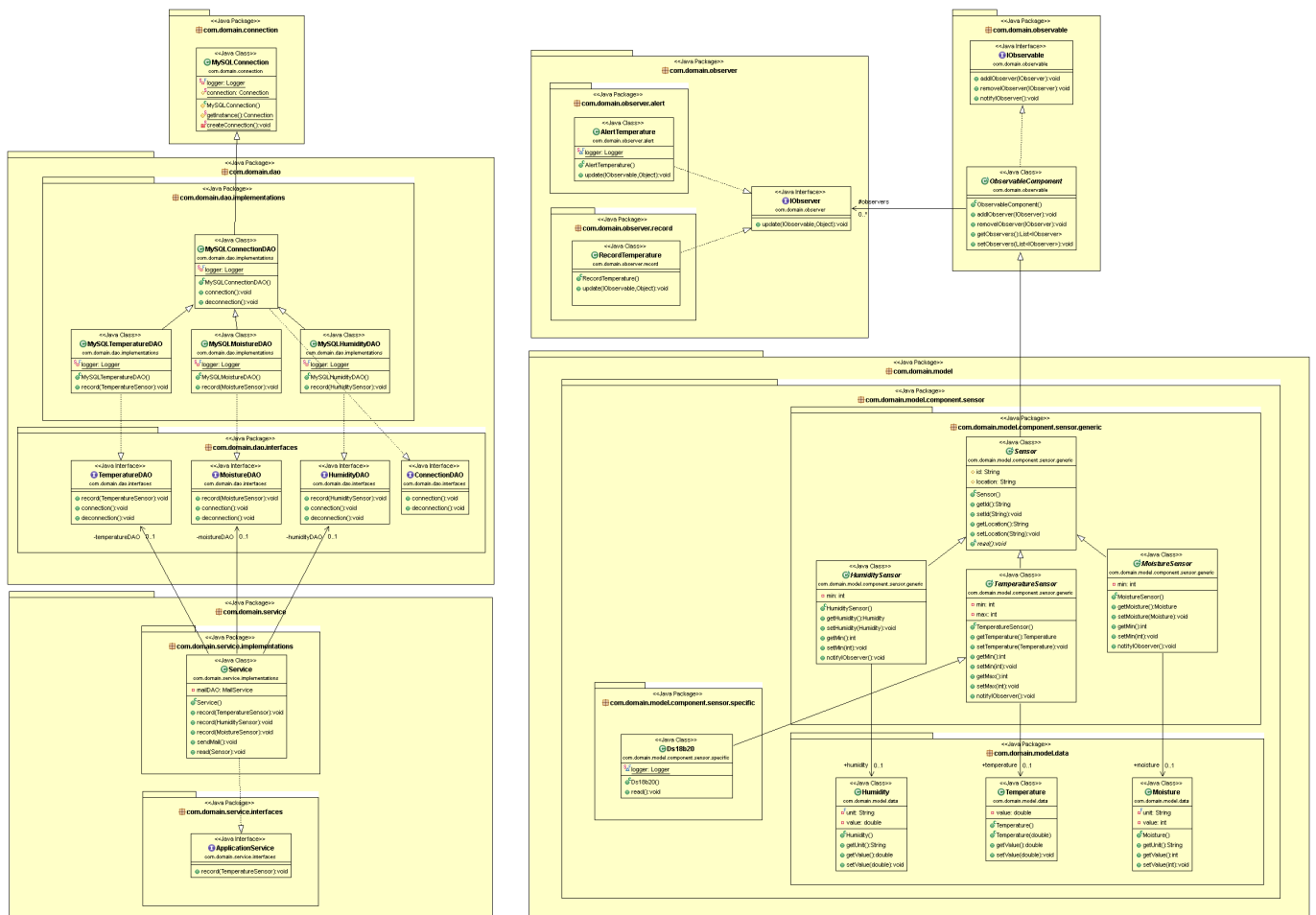
*Modèle conceptuel de l'application*



*Modèle logique de données*

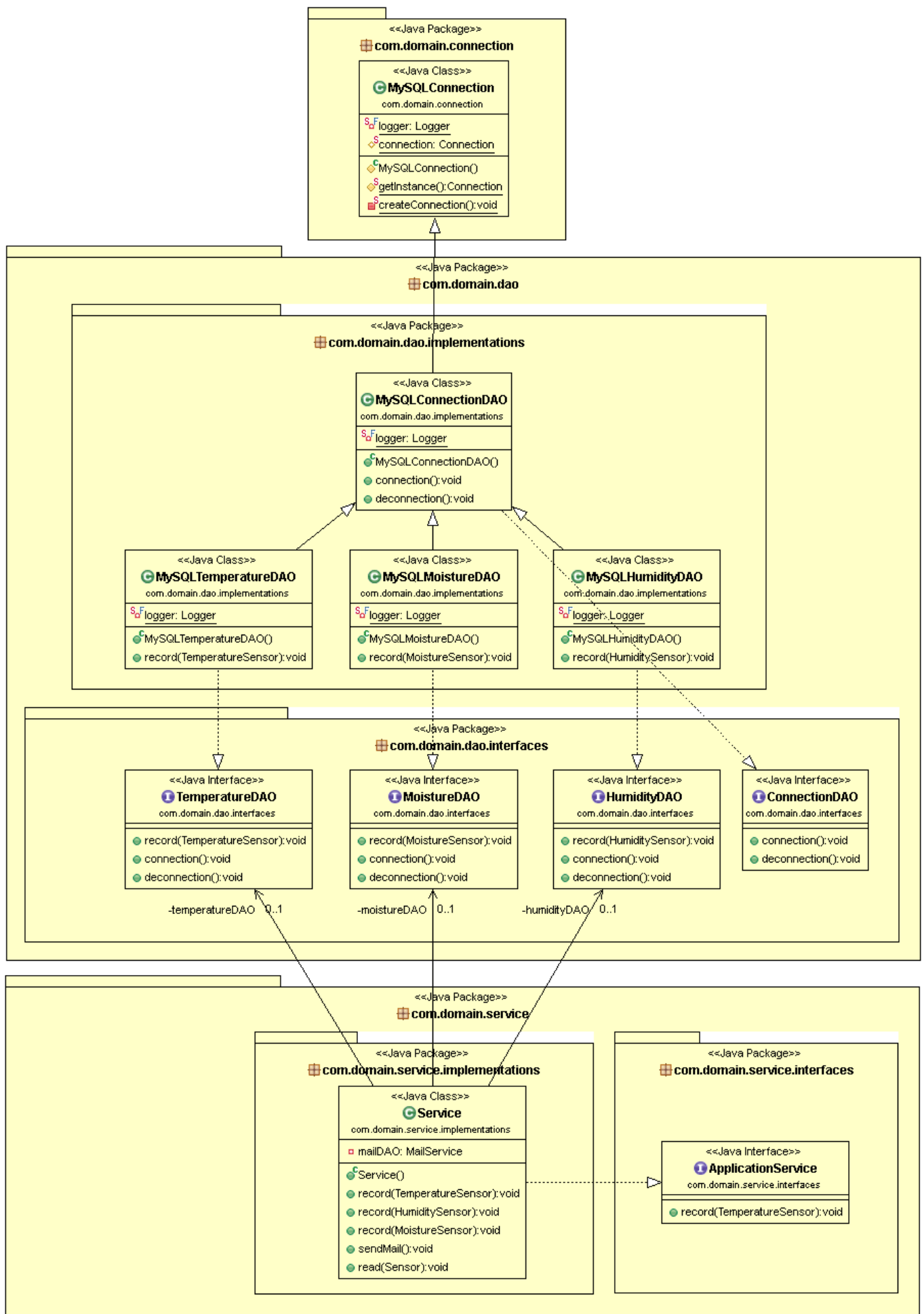
## 3.1.2 Diagramme de classe

### Application principale



Vue d'ensemble de l'application



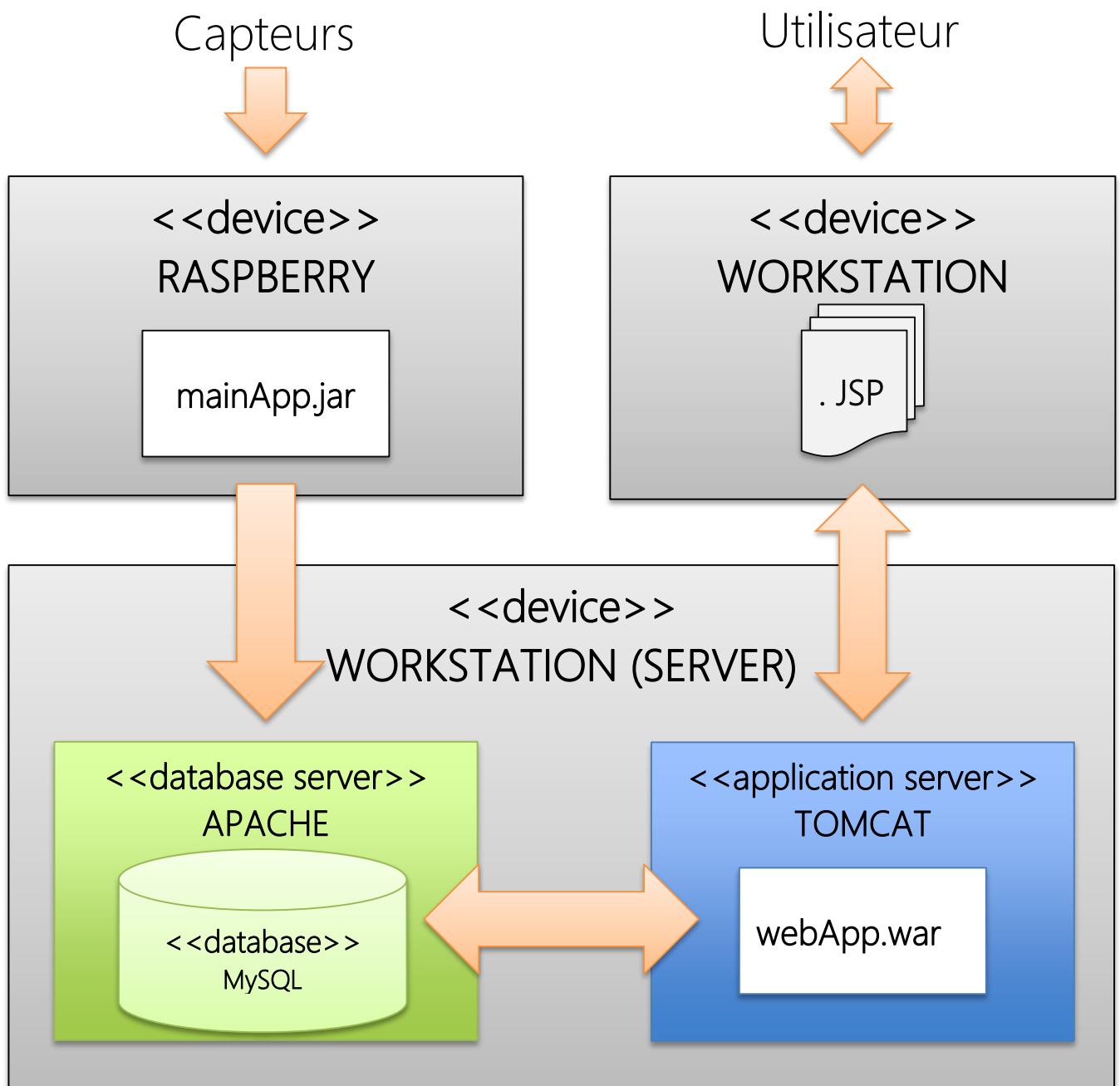


La partie service et DAO

- > Application\_Raspberry [Application\_Raspberry master ↑5]
  - > src
    - > com.domain
      - connection
        - MySQLConnection.java
      - dao
        - implementations
          - MySQLConnectionDAO.java
          - MySQLHumidityDAO.java
          - MySQLMoistureDAO.java
          - MySQLTemperatureDAO.java
        - interfaces
          - ConnectionDAO.java
          - HumidityDAO.java
          - MoistureDAO.java
          - TemperatureDAO.java
      - model
        - component.sensor
          - generic
            - HumiditySensor.java
            - MoistureSensor.java
            - Sensor.java
            - TemperatureSensor.java
          - specific
        - data
          - Humidity.java
          - Moisture.java
          - Temperature.java
      - > observable
        - IObservable.java
        - > ObservableComponent.java
      - > observer
        - > alert
        - record
        - IObserver.java
      - > service
        - > implementations
          - > Service.java
        - interfaces
          - ApplicationService.java
      - thread
        - TThread.java

*Structure en package de l'application principale*

### 3.1.3 Schéma de déploiement



## 3.2 Maquettage (application cliente)

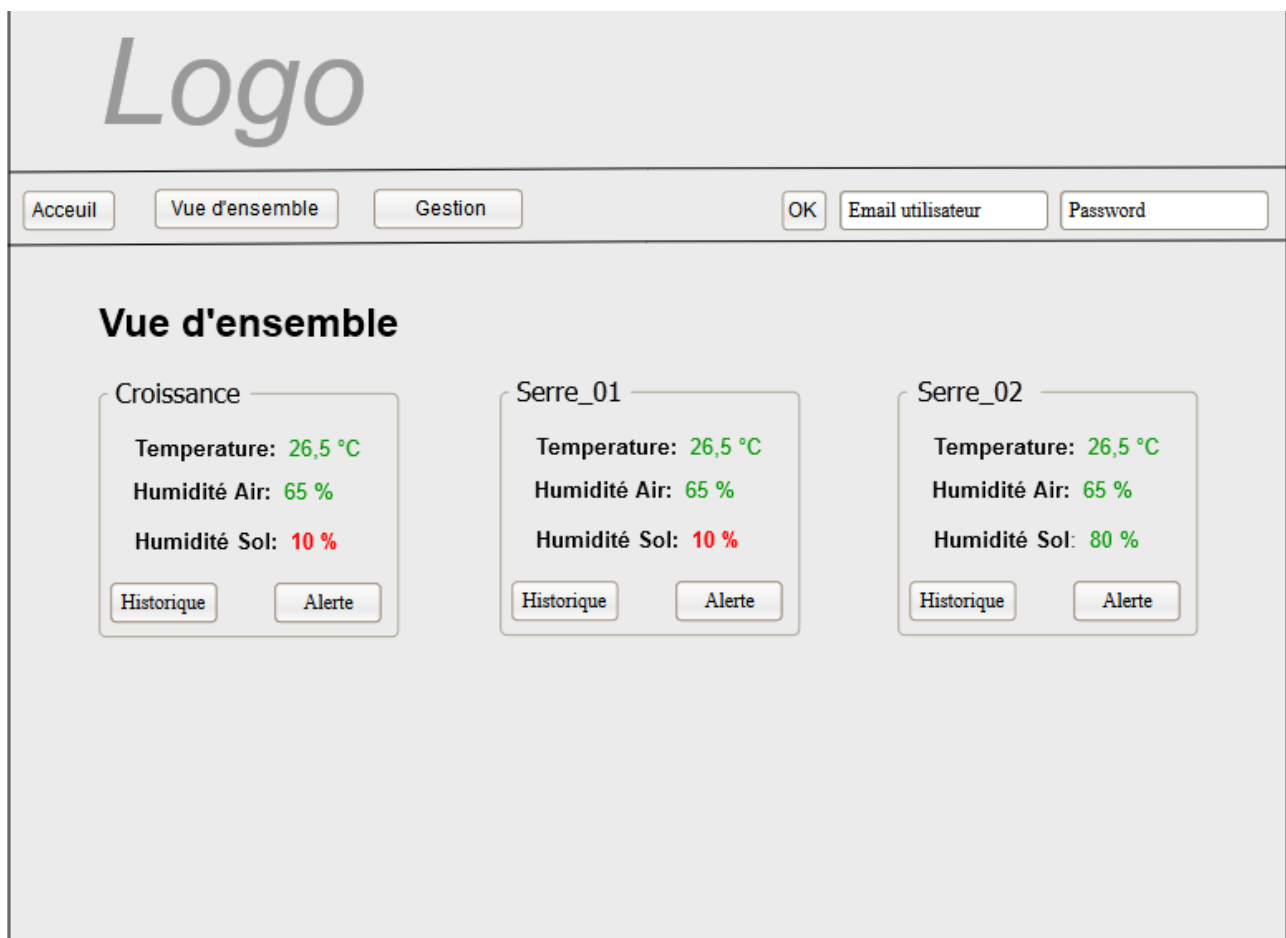
L'application principale étant prioritaire, à l'heure actuelle l'interface homme-machine se contente d'afficher les températures stockées dans la base (aucune navigation, gestion...).

Néanmoins, voici quelques maquettes réalisées pendant la phase de conception afin de cerner les besoins et contraintes liés à l'affichage.

### 3.2.1 Vue d'ensemble

La « vue d'ensemble » permet de voir l'ensemble des mesures en temps réelles.

Les boutons « Historique » et « Alerte » qui permettent d'afficher les graphiques et de gérer les alertes.



*Vue d'ensemble regroupée par lieu*



## 3.2.2 Gestion des enregistrements et des alertes

La vue « Enregistrement et alerte » permet d'activer ou désactiver l'enregistrement d'un capteur ; de modifier une alerte.

Une « checkbox » permet d'activer/désactiver l'enregistrement d'un capteur.

Une « checkbox » permet d'activer/désactiver l'alerte d'un capteur.

Un ensemble de « spinner » permet de gérer les alertes en indiquant le seuil de déclenchement des alertes (min/max), la fréquence en heures des alertes.

Un bouton « modifier » permet d'enregistrer les modifications.

Une liste déroulante permet de choisir un autre lieu sans avoir à repasser par la vue d'ensemble.

The screenshot shows a web interface for managing sensor recordings and alerts. At the top, there is a header with a large 'Logo' placeholder. Below the header is a navigation bar with buttons for 'Accueil', 'Vue d'ensemble', 'Gestion', and a login section with 'OK', 'Email utilisateur', and 'Password' fields. The main content area is titled 'Croissance: Enregistrement et alerte' and features a dropdown menu currently set to 'Serre\_0'. Below this, there are three columns, each representing a different sensor: 'Temperature: 26,5 °C', 'Humidité Air: 60 %', and 'Humidité Sol: 23 %'. Each column has an 'Enregistrement' checkbox (all checked) and an 'Alerte' checkbox (checked for Temperature and Humidité Sol, unchecked for Humidité Air). Under each 'Alerte' checkbox are three spinner controls for 'Min:', 'Max:', and 'Delai:'. The 'Delai' spinner is set to 3 in all three columns. Each column also has a 'modifier' button at the bottom.

*La page de gestion des « observers »*

### 3.2.3 Affichage du graphique de données

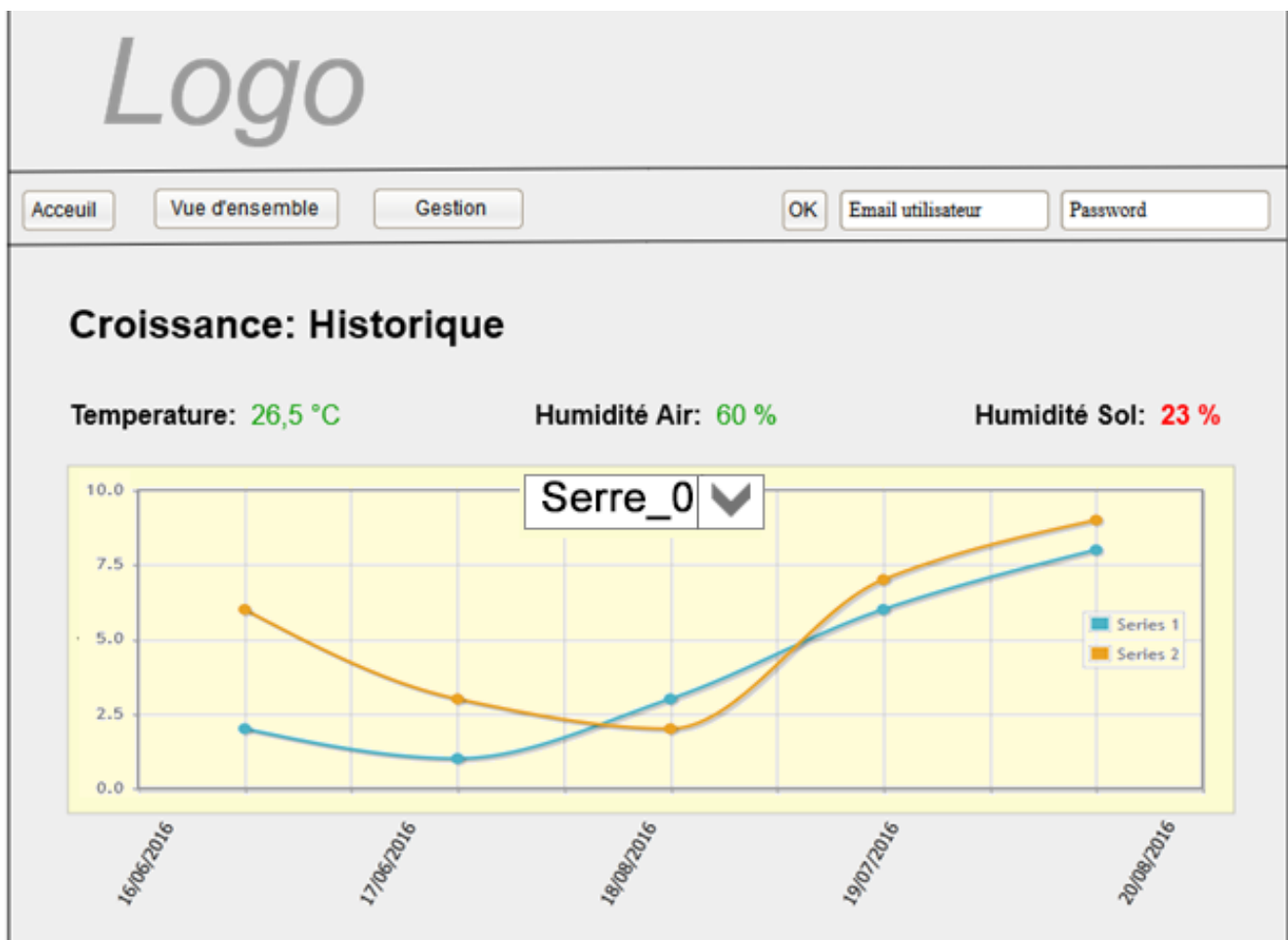
La vue « historique » permet à l'utilisateur de visionner, sous forme de graphique, les données d'un même lieu, enregistrées par les capteurs.

Axe-X (horizontal) : La date de la mesure au format « dd/MM/yyyy hh:mm:ss ».

Axes-Y (vertical) : La valeur de la mesure.

Chaque courbe est colorée et labélisée de façon à différencier le type de mesure.

Une liste déroulante permettant de choisir un autre lieu avoir à repasser par la vue d'ensemble.



*Vue du graphique des différentes mesures d'un même lieu*

# 4 DEVELOPPEMENT

## 4.1 Solutions techniques

### 4.1.1 Une application flexible 1 : Développement en couche

**Besoin :** Développer une application flexible et facilement maintenable. Structurer l'application de façon à améliorer la compréhension/maintenance de cette dernière.

**Solution apportée :** Pour l'application principale, j'ai développé une couche d'accès aux données, une couche de service et une couche métier.  
Pour l'application cliente, j'ai développé une couche de présentation, une couche métier, une couche d'accès aux données.

#### Rappel

Le développement en couche permet de regrouper les composants partageant les rôles au sein de l'application (accès aux données, sécurité, interaction avec l'utilisateur...).

Une couche ne communiquant qu'avec les couches adjacentes, le couplage entre les différentes parties de l'application s'en trouve considérablement réduit.

Prenons l'exemple de la couche d'accès à la base de données ; elle nous permet d'abstraire la façon dont on accède aux données stockées dans la base de données.

Ce qui peut être pratique en cas de changement de type de base de données, car nous n'aurons alors que les éléments de la couche DAO à retoucher ; les traitements du reste de l'application n'étant pas impactée.

## Les différentes couches

### La couche de présentation

Elle permet l'interaction entre l'utilisateur et l'application.

### La couche métier

Elle est composée des objets métier et de leurs traitements.

### La couche de service

En s'appuyant sur les DAO, elle fournit et expose les fonctionnalités de l'application. Elle implémente ainsi les besoins clients.

### La couche DAO

Elle est composée des DAO, c'est-à-dire des objets spécifiques nous permettant d'accéder aux données des objets métier.

Elle permet de :

- centraliser et simplifier les accès aux données,
- d'abstraire l'accès aux données du support de stockage,

## Le pattern MVC

La couche de présentation de l'application cliente, repose sur le patron MVC qui propose de diviser cette dernière en trois sous-couches distinctes : Modèle – Vue – Contrôleur.

Le but de ce MVC est de déclencher le traitement demandé par l'utilisateur et intercepté par le contrôleur, et de renvoyer les données des modèles à la vue.

Il permet donc l'interaction entre l'utilisateur et l'application.

**Couche Modèle :** elle est composée des objets métiers et des méthodes d'accès à leurs données.

**Couche Vue :** elle permet l'interaction entre l'utilisateur et le système.

**Couche Contrôleur :** elle permet d'intercepter les actions de l'utilisateur et d'effectuer les traitements appropriés.

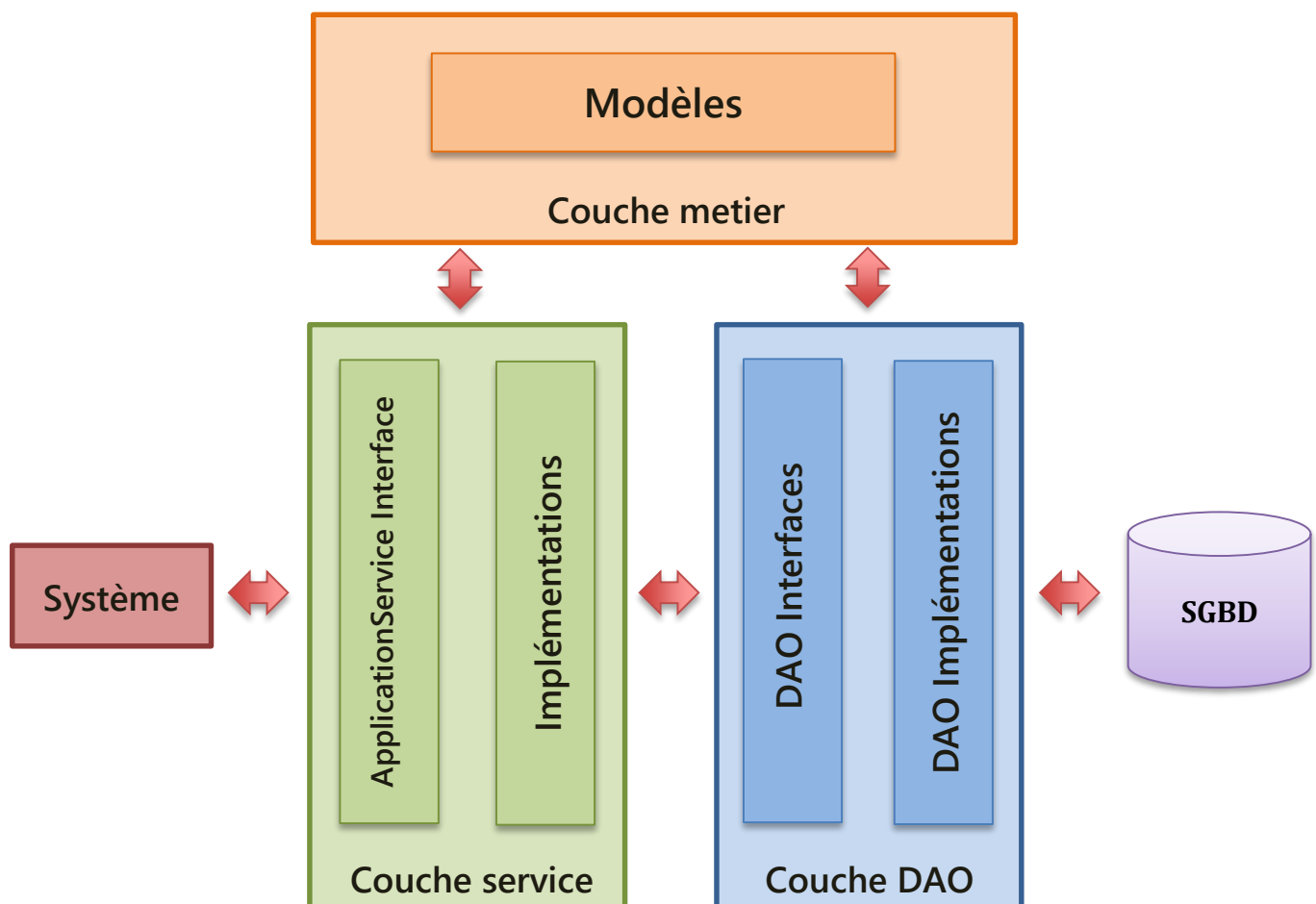
## Application principale

L'application principale contient les couches suivantes :

**La couche métier** : Elle est composée des classes métier et des classes contenant les méthodes spécifiques aux capteurs (ex : DS18B20).

**La couche de service** : Composée de l'interface ApplicationService et de son implémentation Service. Elle regroupe et expose toutes fonctionnalités de l'application (lire un capteur, enregistrer une donnée, envoyer une alerte...).

**La couche DAO** : Elle est composée des différentes DAO et de leur implémentation permettant de stocker les données dans la base.  
Ex : pour enregistrer une température on utilise MySQLTemperatureDAO qui est la façon spécifique à MySQL, d'accéder à l'objet température de la base.



*Schéma du développement en couche de l'application principale*

## Application cliente

L'application cliente est composée des couches suivantes :

### La couche de présentation (composé d'un MVC)

Elle contient les couches suivantes :

#### La couche Modèle

Elle correspond à nos composants (Sensor...), et aux données (Temperature, Humidity...).

#### La couche Vue

Correspond aux pages JSP (ex : demo.jsp)

#### La couche Contrôleur

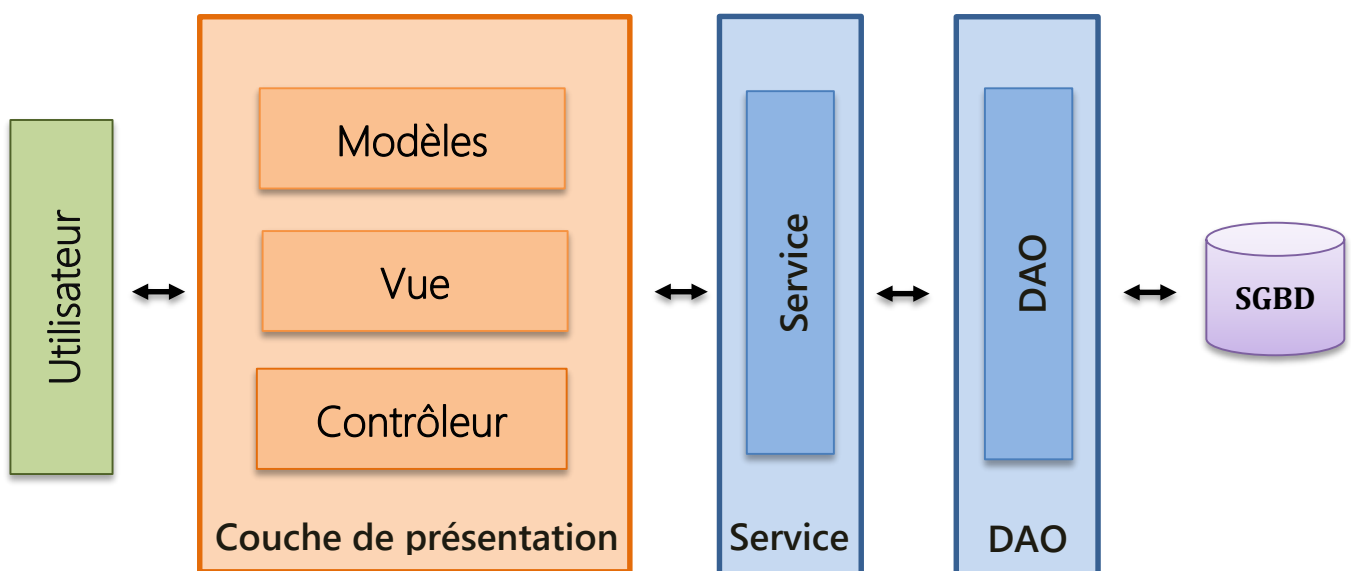
(Je n'ai pas eu le temps de développer cette partie)

### La couche service

(Je n'ai pas eu le temps de développer cette partie)

### La couche DAO

(Je n'ai pas eu le temps de développer cette partie)



*Schéma du développement en couche de l'application cliente*

## 4.1.2 Une application flexible 2 : Le Pattern DAO

**Besoin :** Développer une application flexible où objets métiers ne seront pas dépendant du support de stockage (base de données).

**Solution apportée :** Pour appliquer le patron de conception DAO, j'ai créé des interfaces regroupant les accès à la base de données (ex : méthode d'enregistrement de température...).

J'ai créé des classes qui implémentent ces interfaces et qui regroupent les traitements spécifiques à la technologie MySQL/JDBC.

### Rappel

Patron de conception DAO (Data Access Object) de séparer la couche d'accès aux données de la couche métier. Il permet donc de séparer l'accès au données, spécifique à une technologie comme MySQL par exemple, des objets métier et de leur traitements.

### Exemple d'utilisation : Enregistrement de la température

La classe MySQLTemperatureDAO contient la méthode « record() » spécifique à MySQL et qui permet d'enregistrer la température.

Cette classe implémente l'interface TemperatureDAO.

Quand je veux enregistrer une température, j'appelle la méthode record(), de la classe de Service, qui elle-même utilise la méthode record() de MySQLTemperatureDAO.

De cette façon, si Mr Gouin décide de changer de base de données, pour continuer à enregistrer la température il faudra :

- Ajouter une implémentation de TemperatureDAO spécifique au nouveau support des données.
- Modifier le constructeur de la classe de service afin d'instancier cette nouvelle implémentation.

On n'a donc pas à retoucher au reste des services.

### 4.1.3 Une application évolutive : Classe abstraite et polymorphisme

**Besoin :** Pouvoir changer les composants (capteurs) sans avoir trop de code à modifier.

**Solution apportée :** J'ai créé une classe abstraite (Sensor), qui regroupe les attributs communs aux différents types de capteurs (TemperatureSensor...) et possédant une méthode abstraite « read( ) ».

Cette méthode est implémentée par les classes spécialisées des capteurs.

(Ex : Ds18b20 la classe spécialisé pour un capteur de température du modèle Ds18b20).

De cette façon, si on change de capteur, on a juste à créer une classe qui implémente la méthode read( ) de Sensor, sans avoir à retoucher à tous les traitements utilisant la méthode read( ).

#### Rappel

Les classes abstraites, non instanciables, permettent d'ajouter un niveau d'abstraction à une même famille d'objets.

Le polymorphisme est le fait de faire hériter à une classe, les attributs d'une classe abstraite tout en redéfinissant ses méthodes.

Cela nous permet de pouvoir avoir plusieurs implémentations pour la même méthode.

Ainsi quand on lance le traitement de la classe abstraite, cette dernière se charge d'appeler la méthode appropriée.

#### Exemple d'utilisation dans l'application

« Ds18b20 » est la classe spécialisée pour un capteur de température du modèle Ds18b20.

La classe abstraite Sensor (étendue par TemperatureSensor), contient une méthode abstraite read( ) dont le corps est défini dans la classe Ds18b20.

Ainsi quand on read( ) un Sensor qui est une instance de Ds18b20, on read( ) la méthode de la classe Ds18b20.



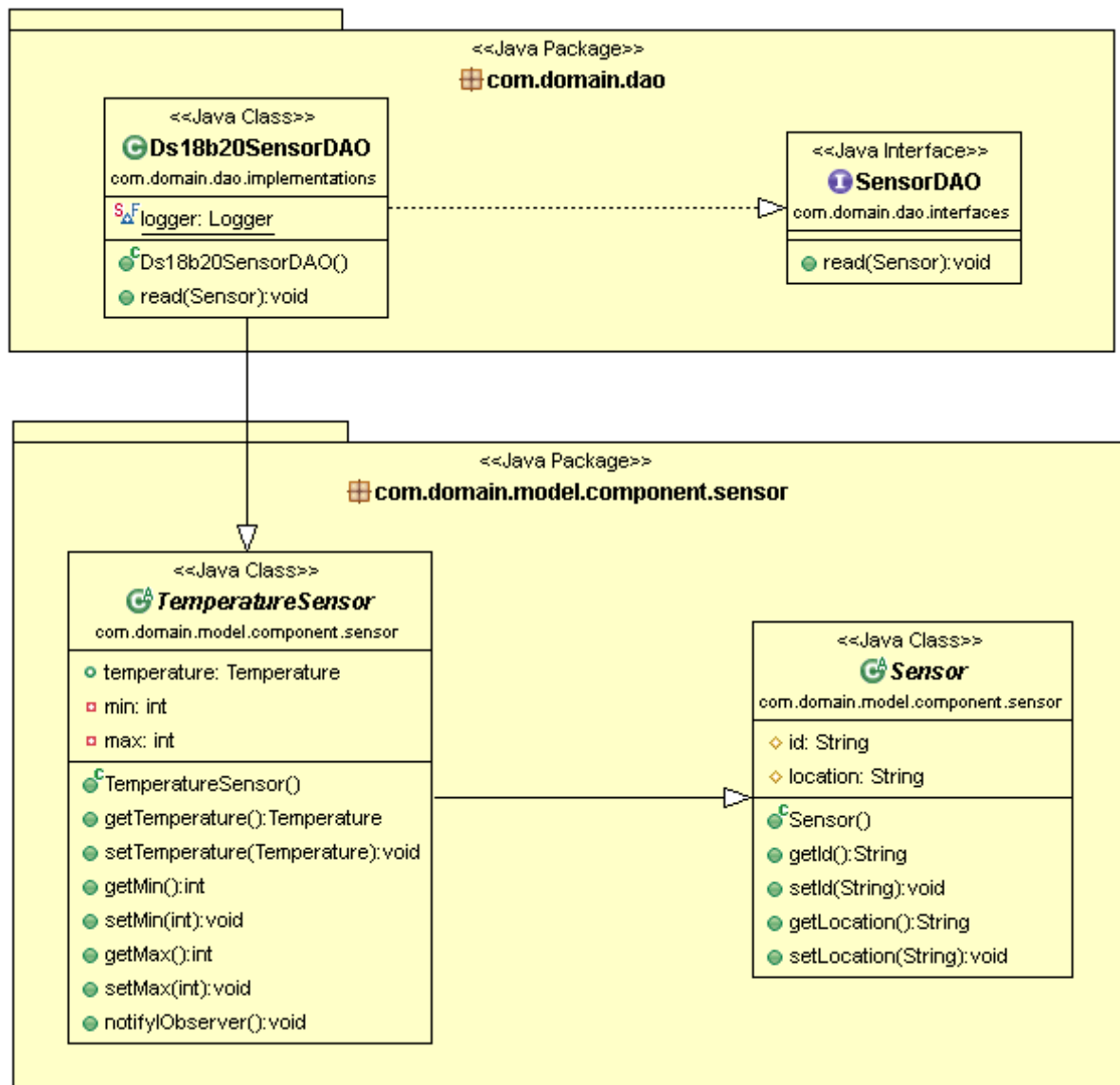


Diagramme de classe d'un capteur de température

## 4.1.4 Gérer l'enregistrement et les alertes :Pattern Observer

**Besoin :** Enregistrer les données et envoyer des alertes quand un capteur effectue une mesure.

**Solution apportée :** Mise en place du patron de conception observer.

Création d'objets « Observer » et d'un objet « Observable ».

Les capteurs qui sont des composants observables, étendent la classe ObservableComponent.

Quand un capteur effectue une mesure, il déclenche les traitements des classes Observers dépendantes (Alert pour une alerte, Record pour un enregistrement de données...).

### Rappel

Le « patron observer », est un patron de conception (design pattern) permettant à des objets donnés dit observables (observable) de notifier (notify) tout changement d'état à des objets dépendants dits observateurs (observers).

Ces notifications permettent aux observers de déclencher des traitements.

Les dépendances entre observables et observateurs sont gérées dynamiquement. On peut ainsi abonner ou désabonner un observateur d'un objet observable.

L'Api de base de Java offre une solution au patron de conception observable, en fournissant la classe « Observable » et l'interface « Observer ».

### Utilisation du patron « observer »

Dans l'application principale, les capteurs jouent le rôle d'élément observable (ObservableComponent).

Quand un capteur modifie sa mesure (setTemperature(...)) il envoie une notification (notifyObserver()) à ses observateurs (RecordTemperature, AlertTemperature). Ces derniers déclenchent les traitements (update()) qu'ils leur sont propres ; en l'occurrence l'enregistrement de la mesure dans la

base de données (record()), et la comparaison de la mesure avec le seuil d'alerte (checkValue()) avec l'envoi d'une alerte en cas de dépassement (sendAlert()).

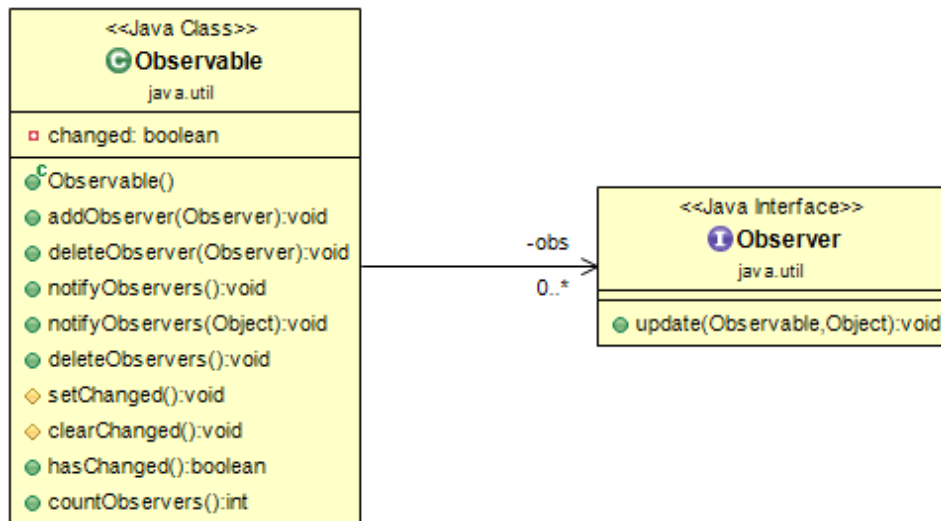


Diagramme de classe du patron observer fournit par l'API Java

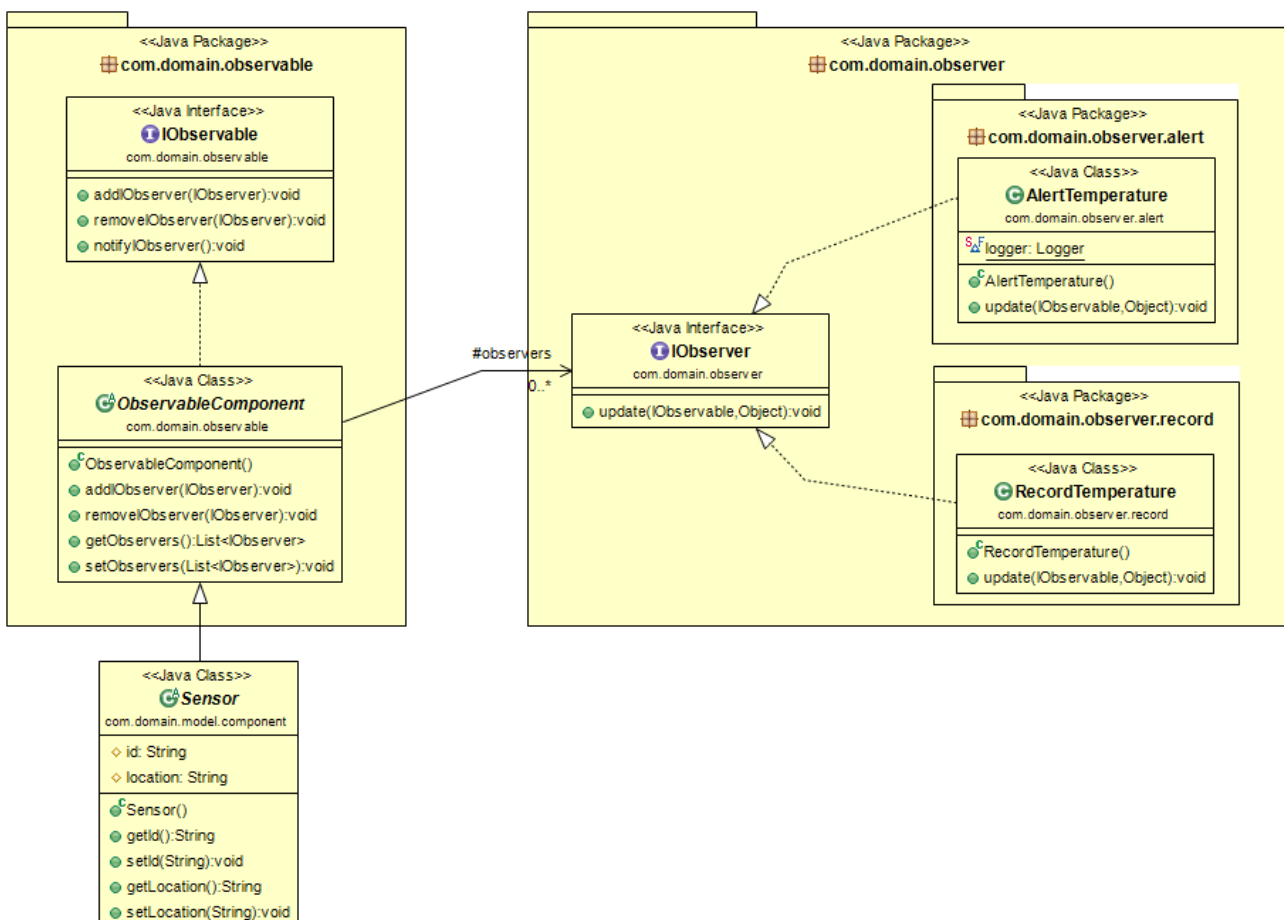
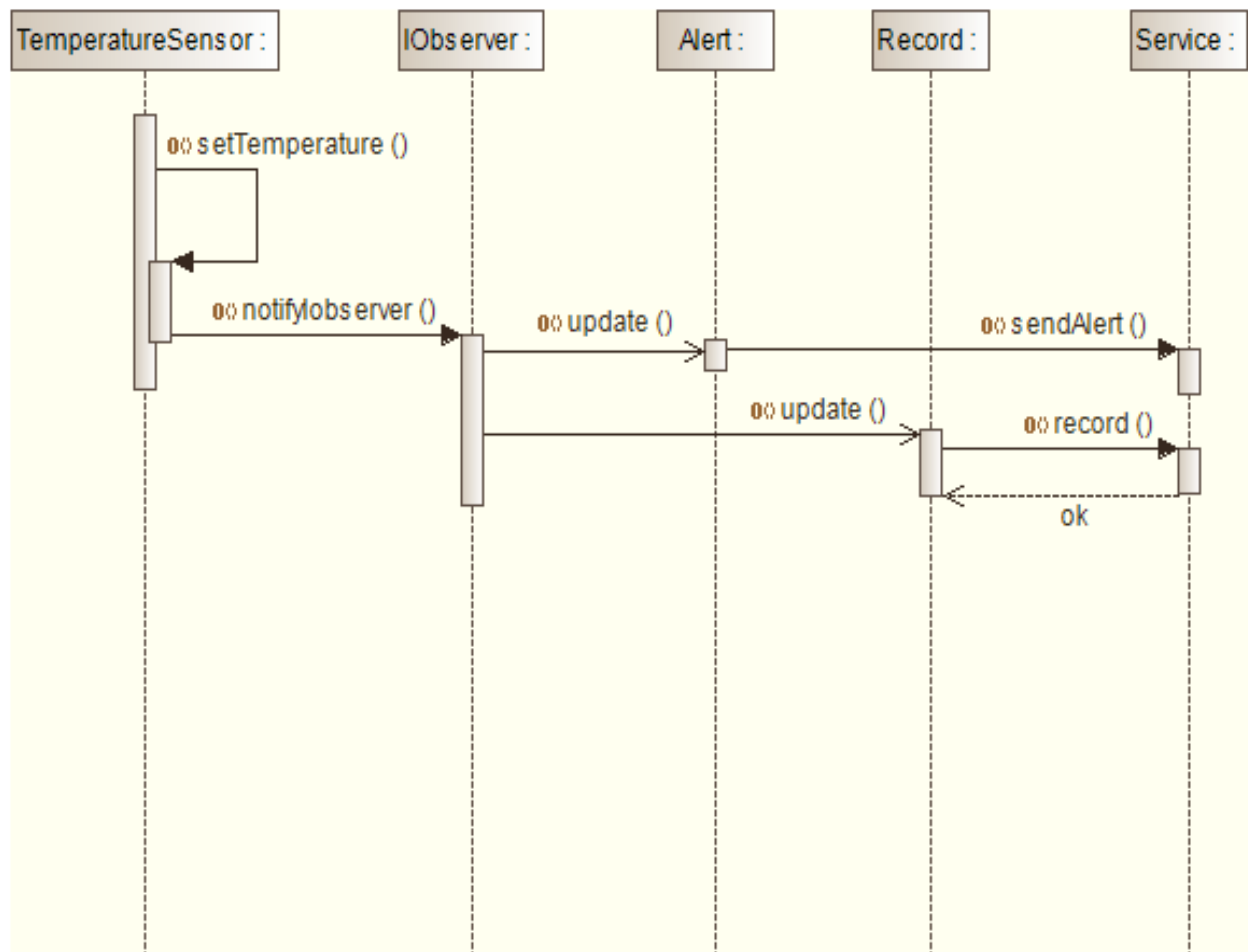


Diagramme de classe du patron observer utilisé (application principale)



*Diagramme de séquence du patron observer (application principale)*

## 4.1.5 Stocker et manipuler les données : MySQL et JDBC

**Besoin :** Stocker et lire les données issues d'une base de données.

**Solution apportée :** J'ai créé une base de données MySQL. Afin de pouvoir accéder aux données et les manipuler, j'ai utilisé l'API Java JDBC.

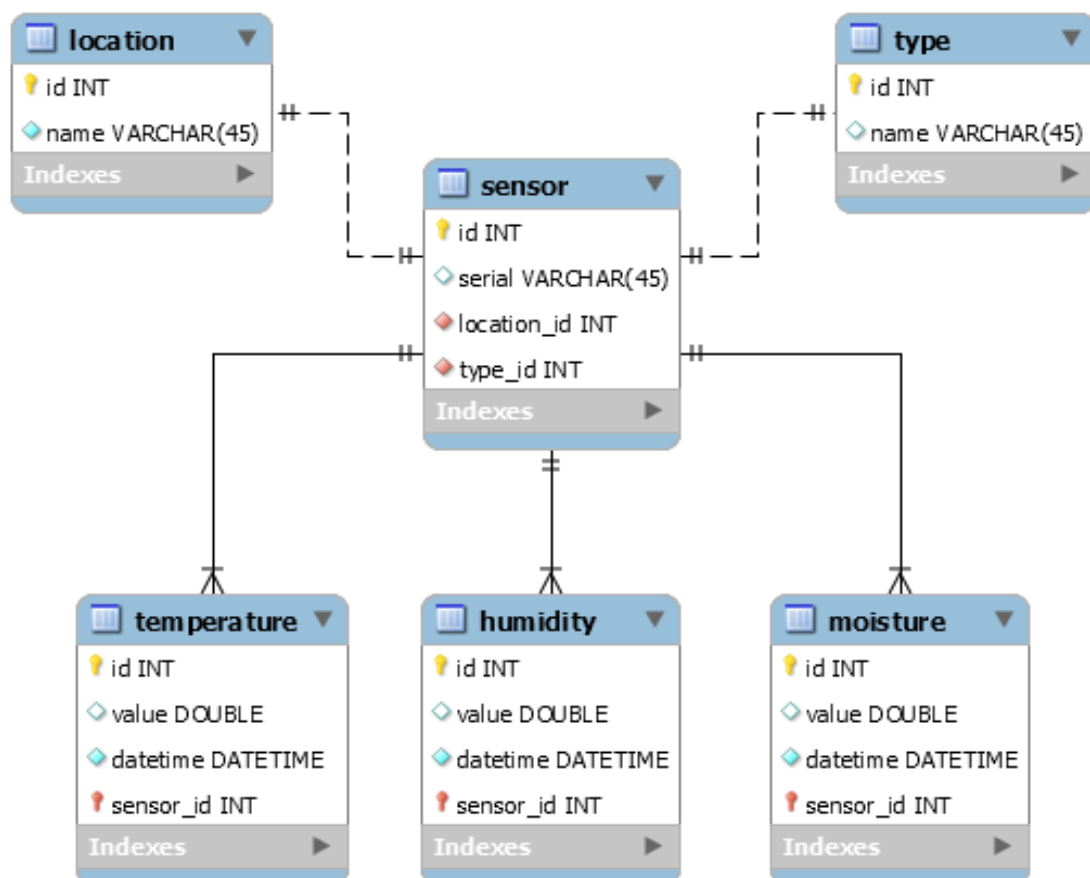
### Rappel

MySQL est une base de données gratuite (GPL) distribué par Oracle.

JDBC est une API Java permettant à l'application l'accès et la manipulation de sources de données. Elle met à disposition les classes et méthodes nécessaires pour accéder au contenu d'une base de données MySQL.

Pour pouvoir l'utiliser, il faut charger un pilote propre à la technologie de la source de données, ici MySQL avec pour pilote Connector/J.

### La base de données MySQL



*Schéma relationnel de la base MySQL*

Dans la base de données, nous retrouvons les champs suivants :

**location** : l'endroit où est implanté le capteur.

- id : identifiant généré automatiquement (clé primaire, unique).
- name : le nom de la localisation, ex : nurserie (unique).

**type** : le type du capteur

- id : identifiant généré automatiquement (clé primaire, unique).
- name : le nom du type, ex : capteur de température (unique).

**sensor** : le capteur

- id : identifiant généré automatiquement (clé primaire, unique).
- serial : le numéro de série du capteur (unique).
- location\_id : l'id de la location du capteur (clé étrangère).
- type\_id: l'id du type du capteur (clé étrangère).

**température** : la température

- id : identifiant généré automatiquement (clé primaire, unique).
- value : la valeur de la mesure.
- datetime : la date au format yyyy-MM-dd hh :mm :ss
- sensor\_id: l'id du capteur qui a effectué la mesure (clé étrangère).

**humidity** : humidité relative de l'air

- id : identifiant généré automatiquement (clé primaire, unique).
- value : la valeur de la mesure.
- datetime : la date au format yyyy-MM-dd hh :mm :ss
- sensor\_id: l'id du capteur qui a effectué la mesure (clé étrangère).

**moisture** : l'humidité du sol

- id : identifiant généré automatiquement (clé primaire, unique).
- value : la valeur de la mesure.
- datetime : la date au format yyyy-MM-dd hh :mm :ss
- sensor\_id: l'id du capteur qui a effectué la mesure (clé étrangère).

***Note** : On remarque la redondance de certains champs...*

*Mon tuteur m'a conseillé de créer une table par type de mesure, afin de ne pas surcharger une table en particulier.*

## La librairie JDBC


```
public void record(TemperatureSensor sensor){
    Date date = new java.util.Date();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    String sql = "INSERT INTO temperature (value, sensor_id, datetime)"
        + " VALUES (?, (SELECT id from sensor where serial=?),?)";

    try(PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
        preparedStatement.setDouble(1, sensor.getTemperature().getValue());
        preparedStatement.setString(2, sensor.getId());
        preparedStatement.setString(3, sdf.format(date));
        preparedStatement.executeUpdate();
        logger.debug("Querie SUCCESS");
    } catch (SQLException e) {
        logger.error(MessageFormat.format("Querie FAILED ! [stackTrace: {0}]", e));
    }
}
```

*Exemple d'utilisation de JDBC : enregistrement de la température*

*Note : JqPlot demande un format de date spécifique : yyyy-MM-dd hh :mm :ss. Afin de télécharger l'application, le formatage de la date aurait très bien pu être effectué coté client grâce à jQuery.*

<input type="checkbox"/>	▼ id	value	datetime	sensor_id 
<input type="checkbox"/>	113	24.812	2016-09-09 14:42:37	2
<input type="checkbox"/>	112	23.75	2016-09-09 14:42:36	1
<input type="checkbox"/>	111	24.75	2016-09-09 14:42:25	2
<input type="checkbox"/>	110	23.75	2016-09-09 14:42:24	1
<input type="checkbox"/>	109	24.812	2016-09-09 14:42:13	2
<input type="checkbox"/>	108	23.75	2016-09-09 14:42:12	1
<input type="checkbox"/>	107	24.812	2016-09-09 14:42:01	2
<input type="checkbox"/>	106	23.75	2016-09-09 14:42:00	1
<input type="checkbox"/>	105	24.812	2016-09-09 14:41:50	2
<input type="checkbox"/>	104	23.75	2016-09-09 14:41:49	1
<input type="checkbox"/>	103	24.812	2016-09-09 14:41:38	2
<input type="checkbox"/>	102	23.75	2016-09-09 14:41:37	1
<input type="checkbox"/>	101	24.812	2016-09-09 14:41:26	2
<input type="checkbox"/>	100	23.75	2016-09-09 14:41:25	1

*Exemple d'enregistrement (température)*

## 4.1.6 Afficher des graphiques: JSP, JSON, Ajax et jqPlot

**Besoin :** Afficher les données stockées dans la base sous forme de graphique.

**Solution apportée :** Développement un « client léger » grâce à la technologie Java JSP.

Utilisation d'un plugin jQuery (JavaScript) permettant de générer des graphiques.

Utilisation D'Ajax et de JSON pour transmettre des données structurées de façon à ce que jQuery puisse les interpréter.

### Rappel

JSP est une technologie Java permettant de générer des pages web dynamiques.

Les pages ainsi créées au format .jsp sont en réalité des pages HTML contenant du code Java ; permettant ainsi de faire le lien entre la vue et les données.

JqPlot est un plugin jQuery basé sur le langage de programmation JavaScript et spécialisé dans la génération de graphiques de données.

Ajax est une technologie permettant de mettre à jour dynamiquement le contenu d'une page web.

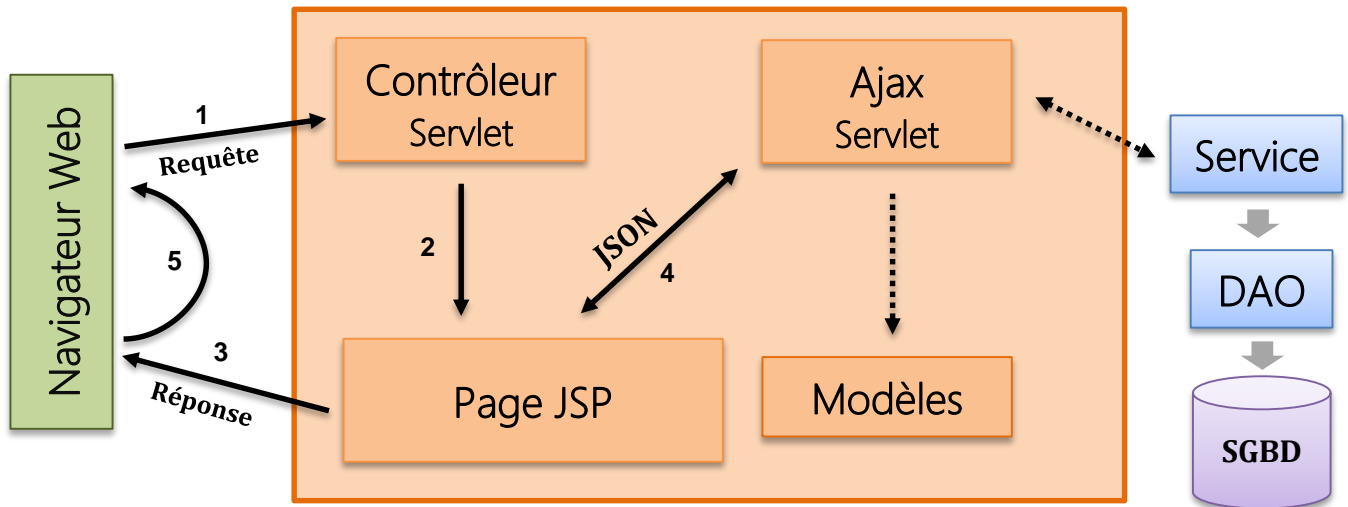
JSON permet de représenter de l'information structurée comme le XML par exemple.

### Java Server Pages

L'application cliente est développée en JSP.

J'ai développé une servlet permettant de générer un JSON contenant des objets métiers issus de la base de données.





*Schéma de fonctionnement de l'application cliente*

1. L'utilisateur : Choisit une action à réaliser en envoyant requête HTML au contrôleur.  
Ex : afficher le graphique de données d'un lieu choisit
2. Le contrôleur : Intercepte l'action de l'utilisateur, et le redirige vers la page JSP correspondante.
3. L'utilisateur récupère la page JSP par le biais d'une réponse HTTP, et affiche cette dernière en interprétant le code HTML dans son navigateur internet.
4. La page JSP : contient entre ses balises HTML <HEAD>, un script jQuery qui récupère le JSON généré par la servlet Ajax.

***Note :** La servlet Ajax utilise les modèles pour formater les données récupérées dans la base grâce aux services et aux DAO.*

5. Le navigateur, interprète le JSON et jqPlot génère alors le graphique.

## JSON

Afin de pouvoir transmettre le JSON à la vue, j'ai utilisé la librairie JAVA Gson de Google; notamment la fonction « toJson( ) » permettant de mettre au format JSON un tableau de données (ArrayList).

Exemple d'un fichier JSON généré : [{"value":23.312,"datetime":"Aug 23, 2016 3:16:34 AM"}, {"value":23.25,"datetime":"Aug 23, 2016 3:16:46 AM"}]

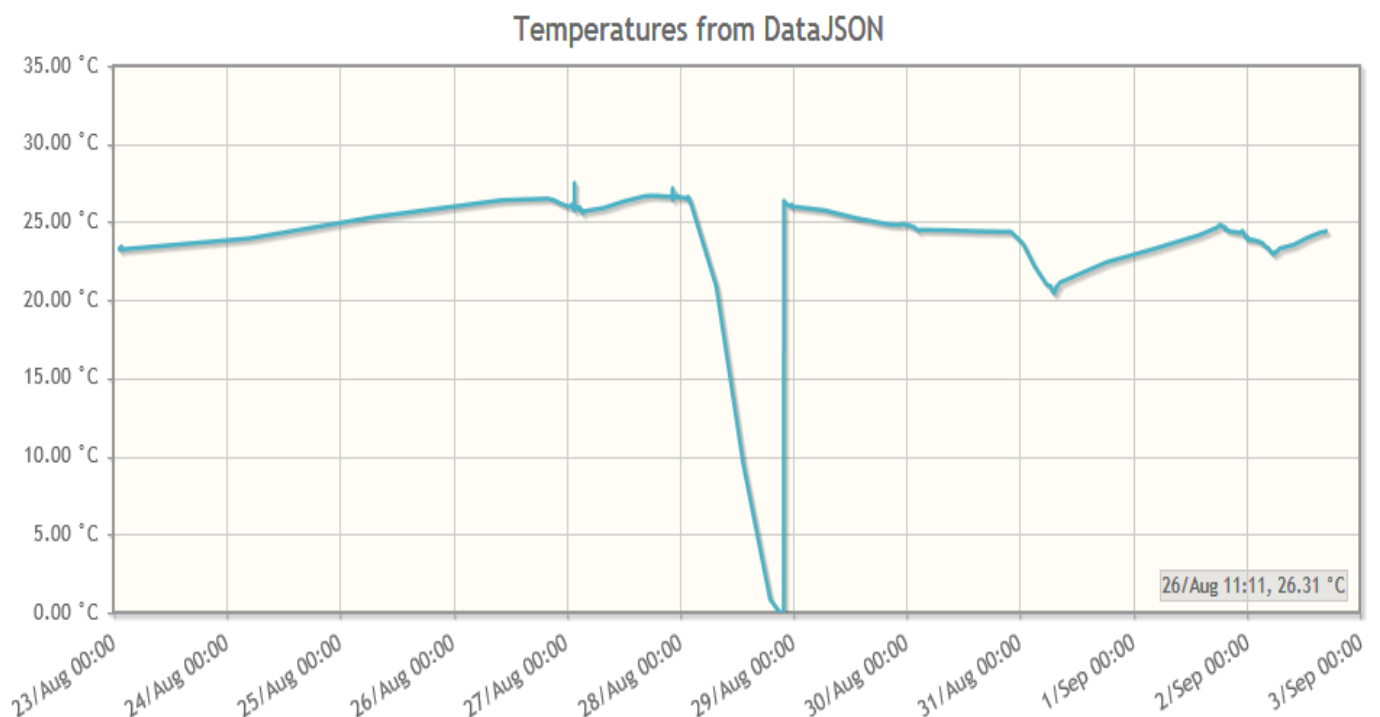
## AJAX

J'ai utilisé la librairie jQuery et sa méthode « `ajax( )` » permettant de faciliter l'utilisation de la technologie Ajax.

J'ai pour cela renseigné les paramètres URL et `DataType`, en renseignant le chemin de la servlet nommée `DataJSON` et en indiquant le format de données JSON.

## JqPlot

Voir le code source en annexe, notamment le fichier « `graph.js` », se trouvant dans le répertoire `WebContent/js` de l'application cliente. Vous trouverez le code source du script qu'il a permis de générer le graphique ci-dessous.



*Exemple d'un graphique de température*

## 4.2 Récapitulatif

### 4.2.1 Application principale

L'application est codée en Java (se) et est hébergée par le Raspberry.

L'application est autonome grâce à un thread secondaire ( $\neq$  Main), qui toutes les six minutes, déclenche la lecture des capteurs.

La classe spécifique au capteur (ex : Ds18b20) lit et interprète le fichier se trouvant sur le Raspberry qui contient les données du capteur.

Quand la lecture du capteur est effectuée, ce dernier envoie une notification aux observers « RecordTemperature » et « AlertTemperature », les classes d'enregistrement et d'alerte.

Le module d'enregistrement se charge alors du stockage de la mesure, grâce aux services s'appuyant sur des DAO.

Le module d'alerte quant à lui, compare la mesure, aux seuils définis (ex : température max) et envoie une alerte à l'utilisateur en cas de dépassement. (L'alerte pour le moment n'est qu'un log... l'implémentation du service de mailing n'étant pas terminé).

### 4.2.2 Application cliente

L'application cliente est une application web JSP fonctionnant sur un serveur Tomcat, hébergée par un professionnel.

L'utilisateur demande l'affichage des données, en envoyant une requête à la servlet contrôleur.

La servlet contrôleur, génère la page JSP correspondante et contenant un script JavaScript (jQuery+jqPlot), puis redirige l'utilisateur vers cette dernière.

Grace à Ajax JQuery récupère le fichier JSON généré par la servlet « DataJSON ».

Le script jqPlot interprète alors le JSON et génère le graphique approprié.

# 5 CONCLUSION

A travers ce stage, j'ai pu revoir et approfondir les différentes parties importantes de ma formation qui sont : le développement en couche, la gestion de base de données, la création d'interface homme-machine.

La gestion de projet fut un point sensible de ce stage ; car j'étais seul à travailler en télétravail sur ce projet.

La recherche et la conception, grâce à l'UML notamment, ont eu une importance majeure dans l'aboutissement de ce projet ; car je voulais absolument développer une application flexible et évolutive, afin de pouvoir par la suite continuer son développement.

Pour finir, je pense que cette expérience m'a permis de me décomplexer et de m'affirmer en tant que concepteur/développeur, en me prouvant que j'étais capable d'analyser précisément un besoin et d'y répondre en concevant et développant les solutions techniques adaptées.

# GLOSSAIRE

**1-Wire** : Bus qui permet de connecter (en série, parallèle ou en étoile) des composants avec seulement deux fils.

**API**: Interface de **P**rogrammation **A**pplicative, regroupant des méthodes et des routines sous forme de librairie, facilitant de ce fait le travail du programmeur.

**Bus** : Dispositif de transmission de données partagé entre plusieurs composants d'un système numérique.

**GPIO**: **G**eneral **P**urpose **I**nput/**O**utput sont des ports d'entrée/sortie présent sur le Raspberry. Ils permettent d'y connecter divers périphériques.

**I2C**: **I**nter-**I**ntegrated **C**ircuit est un bus informatique qui comme 1-Wire, permet de connecter des composants avec deux fils.

**JAR** : **J**ava **A**rchive est un fichier ZIP contenant l'ensemble des classes constituant le programme. Il sert à lancer le programme à l'image d'un EXE(cutable).

**Java ME** : **J**ava **P**latform, **M**icro **E**dition est une plateforme JAVA dédiée aux applications embarquées (téléphone mobile, domotique...).

**Java SE**: **J**ava **P**latform, **S**tandard **E**dition est la plateforme standard dédiée aux applications pour poste de travail (Workstation).

**JavaScript**: Langage de programmation de scripts principalement employé dans les pages web interactives, en permettant des traitements coté client.

**JDBC** : **J**ava **D**ata**B**ase **C**onnectivity est une API Java fournie par ORACLE et permettant d'accéder à des sources de données (Ex : base de données MySQL).

**JDK** : Ensemble bibliothèques et d'outils, permettant de compiler des programmes JAVA.

**jqPlot**: Plugin jQuery spécialisé dans la création de graphique de données.

**jQuery** : Api JavaScript.

**JSON** : JavaScript Object Notation est un format d'échange de données, permettant de représenter l'information de façon structurée à l'image du XML.

**JSP** : Java Server Pages est une technologie JAVA permettant de créer des pages web dynamique, mêlant HTML et JAVA.

**Patron de conception (design pattern)** : Proposition de solution éprouvée (méthodologie) répondant à un problème spécifique et récurrent.

**Singleton** : Patron de conception permettant de restreindre l'instanciation d'une classe (Ex : connexion à la base de données).

**UML** : Langage de Modélisation Unifié est un langage de modélisation graphique fournissant une méthodologie afin de mieux visualiser la conception d'un système.

# SOURCES

Voici quelques-uns des nombreux livres et sites où j'ai pu trouver de l'information concernant la réalisation du projet.

## Bibliographie :

« Programmer en Java » des éditions Eyrolles  
« UML 2 par la pratique » des éditions Eyrolles

## Webographie :

### Raspberry

<http://www.framboise314.fr>  
<https://www.raspberrypi.org/forums/>

### Programmation

<https://docs.oracle.com>  
[www.jmdoudoux.fr/java/](http://www.jmdoudoux.fr/java/)  
<https://openclassrooms.com/>  
<http://java.developpez.com/>  
<http://balusc.omnifaces.org>  
<http://stackoverflow.com/>  
<http://design-patterns.fr/>

# ANNEXE

Joint à ce rapport, vous trouverez au format PDF:

La documentation réalisée pour Mr Gouin, qui retrace les étapes permettant de configurer et d'utiliser le Raspberry et ses capteurs.

.