RAPPORT DE STAGE

Refonte d’un système d’informations

PhenodynDB : système de gestion d’expérimentations sur les plantes

RÉALISÉ PAR

Matthieu POIROT

SOUS LA DIRECTION DE

Vincent NÈGRE

Remerciements

Je tiens à remercier toutes les personnes qui m’ont aidé lors de la réalisation de ce stage ainsi que lors de la rédaction de ce rapport.

Tout d’abord, mes remerciements vont à Vincent NEGRE, mon encadrant, qui m’a fait confiance et a su être présent et m’apporter son aide durant les différentes étapes du projet.

Je tiens ensuite à remercier Nicolas BRICHET et Jonathan MINEAU, qui m’ont accordé une aide complémentaire à celle de mon encadrant en apportant leur expertise en cas de besoin.

Aussi je remercie Adel MEZIANE, Boris PARENT et Olivier TURC, les utilisateurs actuels du site pour le temps qu’ils ont consacré lors de réunion ou de tests utilisateurs à mon projet.

Finalement, je tiens à remercier toute l’équipe de l’unité LEPSE* de l’INRA* de Montpellier pour l’ambiance chaleureuse dans laquelle mon stage a pu s’effectuer, ainsi que toute l’équipe pédagogique de l’IUT informatique de Montpellier, qui m’ont appris les différentes pratiques en informatique.

Table des matières

Table des figures

Glossaire

UMR : Unité de Recherche Mixte.

INRA : Institut National de la Recherche Agronomique.

LEPSE : Laboratoire d'Ecophysiologie des Plantes sous Stress Environnementaux.

Agronomie : ensemble des sciences auxquelles il est fait appel dans la pratique et la compréhension de l'agriculture.

Système d'informations : un ensemble organisé de ressources qui permet de collecter, stocker, traiter et distribuer de l'information souvent grâce à un ordinateur. Dans mon cas, le système d'informations est constitué d'un base de données relationnelles et d'une interface web permettant de déposer, consulter ou exporter les données.

Phenotyping, ou phénotypage : le phénotypage est l'établissement de l'état des différents caractères observables chez un organisme vivant.

LIRMM : Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier.

CIRAD : Centre de coopération Internationale en Recherche Agronomique pour le Développement.

CNRS : Centre National de la Recherche Scientifique

Thermocouple : capteur permettant de mesurer la température grâce à l'effet Seebeck. Ils sont utilisés dans le laboratoire pour mesure la température des plantes.

R : Le langage R est un langage informatique dédié aux statistiques et à la science de données.

RDT : Rotative Displacement Transducers.Ce se sont des capteurs rotatifs à effet Hall qui le LEPSE a adapté pour suivre la croissance des plantes..

Framework : un framework est un ensemble d'outils organisés suivant une architecture et des designs patterns, constituant les fondations toutes prêtes d'une application web ou d'un logiciel.

IDE : Integrated Development Envrionment, ou en français environnement de développement intégré. L'objectif d'un environnement de développement est d'augmenter la productivité des programmeurs en automatisant une partie des activités et en simplifiant les opérations.

SGBD : un Système de Gestion de Base de Données est un logiciel système destiné à stocker et à partager des informations dans une base de données.

Design Pattern : un design pattern est un arrangement de module reconnu comme une bonne pratique pour résoudre un problème.

QueryString : partie de l'URL pouvant contenir des données.

Transaction : une transaction est un ensemble de requêtes qui sont exécutées en un seul bloc. Ainsi, si une des requêtes du bloc échoue, on peut décider d'annuler tout le bloc de requêtes (ou de quand même valider les requêtes qui ont réussi).

DOI : un DOI (Digital Object Identifier) est un identifiant pérenne qui permet l'identification unique d'un objet physique ou numérique et sa citation. Il fournit un lien stable à des ressources en ligne, comme les données de la recherche.

Introduction

L'agronomie* est un domaine de recherche majeur pour l'humanité depuis très longtemps, afin de répondre à différentes problématiques, comme la faim dans le monde ou le réchauffement climatique.

Ainsi, le LEPSE* est une UMR (INRA - Sup Agro Montpellier) qui étudie la réponse des plantes (vigne, maïs, *Arabidopsis Thaliana*,…) à différents stress environnementaux comme une température élevée, ou un manque d'eau.

Dans le cadre de ces expérimentations, des milliers de données recueillies sur les plantes sont insérées dans une base de données. Les scientifiques n'étant pas des administrateurs de base de données, il leur faut une interface afin de la consulter. Cette interface est matérialisée par le système d'information PhenodynDB (http://bioweb.supagro.inra.fr/phenodyn)développé par l'équipe informatique de l'unité de recherche.

Ce système d'informations* permet notamment d'insérer de nouvelles données, de les visualiser à l'aide de graphes ou encore de les exporter.

Depuis sa création en 2011, de nombreuses modifications ont été apportées à la base de données afin qu'elle réponde aux besoins des chercheurs et facilite le traitement des données. L'interface web n'a pas suivi ces changements, et ne répond donc plus aux besoins de ses utilisateurs. Ma tâche pendant ce stage a donc été la refonte de la partie web du système d'informations PhenodynDB, afin qu'il réponde de nouveau aux besoins des scientifiques.

Dans un premier temps, nous verrons plus en détail le fonctionnement de l'unité qui m'a accueilli. Viendra ensuite l'analyse des besoins des utilisateurs à travers le cahier des charges, puis la partie technique détaillant le processus de développement du site. Une partie fournissant un manuel d'utilisation suivra, pour terminer par la partie présentant la méthodologie utilisée lors de la réalisation de ce projet.

1. Présentation de la structure d'accueil

1.1. Création et évolution de l'entreprise

L'UMR LEPSE* a été créée en 1993 par l'association de deux entités : l'Institut National de Recherche Agronomique (INRA) de Montpellier et l'institut national d'études supérieures agronomiques de Montpellier (Montpellier SupAgro).

L'INRA est un institut fondé en 1946 afin de répondre à la problématique de la nourriture en France. En effet, au sortir de la guerre, le pays ne produisait pas assez pour nourrir tout le monde. Cela fut ensuite réglé, et la mission de l'INRA évolua en ce qu'elle est aujourd'hui : développer une agriculture répondant aux besoins de l'homme, tout en étant compétitive et respectueuse de l'environnement.

Montpellier SupAgro est un Établissement Public à Caractère Scientifique, Culturel et Professionnel (EPSCP) sous la tutelle du ministère de l'Agriculture et du ministère de l'Éducation Nationale, de l'Enseignement Supérieur et de la Recherche. Il est le résultat de la fusion de différents établissements liés à l'agronomie dont le plus vieux remonte jusqu'en 1842. Sa mission est de former des spécialistes en agronomie, mais aussi de développer la recherche en agronomie.

1.2. Activités

Le LEPSE tente de répondre à la problématique suivante : comment rendre l'agriculture plus économe en eau et résiliente face au changement climatique.

Pour cela, on y étudie la réponse des plantes à des situations de sécheresse ou de températures élevées, toutes en étudiant le génome de celles-ci. Ces données sont intégrées à des modèles permettant d'évaluer l'efficacité des modes de cultures, les performances des différentes espèces et variétés de plantes et d'en créer de nouvelles.

Les données sont produites sur 3 plateformes, regroupées dans l'infrastructure Montpellier Plant Phenotyping* Platforms (M3P) : Phenodyn, Phenopsis et PhenoArch (cf. **Annexe 1** : plaquette M3P).

La plateforme sur laquelle j'ai travaillé est Phenodyn. Elle comprend une serre de 180 m^2 agrémentée de 140 balances, de 420 capteurs de déplacement (qui permette de mesurer la croissance des feuilles), et de capteurs climatiques. Les données sont recueillies à intervalles réguliers et enregistrées dans une base de données.

1.3. Les partenaires

Le LEPSE a pour partenaire des entreprises et des laboratoires publics et privés, qui peuvent investir de l'argent ou du matériel. Certaines entreprises commandent par exemple des études sur des plantes qu'elles fournissent.

Le laboratoire dispose aussi de partenariats avec plusieurs autres centres de recherche, comme le LIRMM*, le CIRAD*, le CNRS*,…

L'unité participe aussi à différents projets qui ont une échelle nationale, européenne ou encore internationale. Par exemple, l'INRA coordonne le projet national Phenome, qui a reçu 24 millions d'euros d'investissement et a été lancé en 2013. Ce projet a pour ambition d'équiper la communauté scientifique française avec une infrastructure capable de mesurer, grâce à des méthodes précises et à haut débit, des caractères agronomiques de plantes soumises à divers scénarios de climats et d'itinéraires techniques associés au changement climatique global. C'est dans le cadre de ce projet qu'a été créée la plateforme PhenoArch.

1.4. L'organisation et le fonctionnement

Le LEPSE est composé d'agents permanents :

- des chercheurs,
- des enseignants chercheurs,
- des ingénieurs,
- des techniciens,
- des membres administratifs,
- un service informatique dont fait partie mon encadrant Vincent Nègre en tant qu'administrateur bases de données, système d'informations et maintenance informatique.

Mais il accueille aussi de nombreux membres non permanents, comme des étudiants en stage, des étudiants en thèse, du personnel scientifique, ou encore du personnel pour aider l'équipe technique.

Ces membres se répartissent sur 3 équipes :

- l'équipe ETAP : « Efficience de transpiration et adaptation des plantes aux climats secs » qui Cherche à identifier les leviers d'action génétiques et agronomiques qui permettent d'améliorer l'efficience d'utilisation de l'eau dans les systèmes viticoles soumis aux contraintes abiotiques ;

- l'équipe MAGE : « Modélisation et Analyse de l'interaction Genotype Environnement ». qui Cherche à identifier des mécanismes de tolérance à la sécheresse dans une large gamme de scénarios climatiques ;

- l’équipe SPIC : « Stress environnementaux et Processus Intégrés de la Croissance ». qui analyse l’effet des contraintes hydriques et thermiques sur le développement et la physiologie de la plante dans le but d’identifier des voies permettant un renforcement de la capacité d’adaptation.

Vous pouvez trouver un organigramme du LEPSE en **Annexe 2**.

2. Cahier des charges

2.1. Contexte

La plateforme appelée Phenodyn est composée :

- de capteurs de différents types :1. des capteurs climatiques permettant d'enregistrer des données micrométéorologiques (rayonnement, température de l'air, température des plantes, humidité, de l'air)
  2. des balances permettant de suivre le statut hydrique du sol et la transpiration des plantes,
  3. des capteurs d'élongation permettant de suivre la croissance des feuilles ;
- d'un logiciel LabView qui enregistre les données mesurées par les capteurs ;
- d'une base de données MySQL où sont enregistrées les données par ce logiciel LabView ;
- d'un site web développé entièrement en PHP, JavaScript, CSS et HTML par l'équipe informatique qui permet d'interagir avec cette base de données.

La base de données est représentée par un Modèle Conceptuel de Données en **Annexe 3**. On peut voir sur ce modèle les tables suivantes, en gras :

- **Manip** (pour manipulation) qui représente une période sur laquelle un scientifique déclare commencer une manipulation.
- Dans le cadre de cette manipulation, des **Pot**s sont déclarés.
- Ces mêmes pots contiennent des **Plante**s.
- Des **Capteurs** sont assignés aux plantes et aux pots dans les tables **periodeMesureBalance**, **périodeMesureCroissance**, et **périodeMesureThermocouple**. Par exemple, une balance est liée à un pot pour telle période, ou encore un thermocouple* est lié à telle plante pour telle période.
- Les données mesurées par les capteurs sont prises à intervalles réguliers (de 5 min à 15 min) et peuvent être trouvées dans les tables **MesureCroissance**, **MesureBalance**, **MesureMeteo**.
- Il y a d'autres tables que je ne détaillerai pas, car elles ne sont pas nécessaires à la compréhension globale de la structure de données. Par exemple un pot peut utiliser un substrat déclaré dans la table **Substrat**.

Cette structure de données permet ensuite aux chercheurs de les traiter à l'aide d'un script R* qu'ils ont créé.

Les données mesurées par les capteurs s'enregistrent automatiquement dans la base de données. Les autres informations relatives à l'expérimentation (pots et plantes étudiées, périodes de mesures) sont déclarées par les expérimentateurs à l'aide d'un site web, qui permet aussi la visualisation de différents graphiques tracés à partir de ces données (par exemple l'évolution du poids du pot numéro 1 de la manipulation qui a pour code « TRI1 » dans les dernières 24 h).

2.2. Problématique

1. La base de données a subi plusieurs évolutions et n'a donc pas toujours été telle que décrite plus haut, or le site web qui permet de déclarer les données n'a pas suivi ces évolutions.

Plusieurs tables ne disposent par d'interface utilisateur. Par exemple la table **periodeMesureBalance** a été créée afin de lier une balance (qui est un **Capteur**) à un **Pot** d'une certaine date à une autre, et le site ne permet pas d'alimenter cette table. La seule solution pour accéder à ces tables était d'interroger directement la base de données en SQL ce qui n'est pas une solution confortable pour des utilisateurs non informaticiens.

2. Le système d'informations était vieillissant (pas de PHP objet, pas de framework, anciennes versions de librairies JavaScript). Une refonte du code et l'utilisation d'un framework moderne étaient devenues nécessaires afin de renforcer sa stabilité, améliorer son ergonomie, faciliter sa maintenance et son évolution.

3. Certaines fonctionnalités manquantes devaient être développées (import de fichiers Excel, graphiques avancés).

Mon travail au cours de ce stage a donc été de réaliser un site web pouvant répondre à l'évolution de la base de données et d'implémenter les fonctionnalités déjà disponibles ainsi que des nouvelles.

2.3. Analyse des besoins fonctionnels

2.3.1. Analyse de l'existant

Le site possède donc certaines fonctionnalités que je vais détailler ici à l'aide d'un diagramme de cas d'utilisation, qui peut être trouvé en **Annexe 4**.

Il y a donc deux fonctionnalités principales :

- **Déclarer des données** :

La déclaration des données correspond à des insertions dans la base de données. La méthode d'assignation de capteurs à des pots ou des plantes est obsolète, comme expliqué dans la problématique. Pour les tables disposant d'une interface d'import des données la déclaration se fait par fichier CSV uniquement. Par ailleurs il n'y a pas d'interface pour déclarer simplement un élément, à part pour les Manips.

- **Tracer des graphiques** :

L'interface de gestion des graphes permet de sélectionner le capteur dont on veut afficher les données récoltées. Pour se faire, on peut voir en **Annexe 5** un menu, qui permet d'accéder à deux systèmes d'entrées différentes :

- On choisit le type de capteur - météo, balance ou RDT* (sachant que « météo » contient quatre genres de capteurs qui pourront être sélectionnés ensuite) - puis on peut trier les capteurs en fonction du site où ils se trouvent (voir **Annexe 6**).
- On choisit l'élément dont on veut extraire les données (pot ou plante), puis on peut filtrer les pots ou plantes par Manip (voir **Annexe 7**). Le dernier menu fonctionne de la même façon, mais propose des plantes en fonction de leur génotype. Le système va ensuite vérifier quel capteur est assigné à quelle plante ou pot pour la date définie et va chercher les données correspondantes dans la base de données et les afficher dans le navigateur web.

2.3.2. Nouvelles fonctionnalités

Suite à différentes réunions, nous avons pu avec les utilisateurs de la plateforme établir les nouvelles fonctionnalités qu'ils souhaiteraient retrouver sur le nouveau site :

- la possibilité de déclarer une par une certaines données (import unitaire),
- la possibilité de déclarer un jeu de données (import en masse) à l'aide d'un fichier CSV ou XLSX (type de fichier du logiciel Excel de Microsoft),
- la possibilité de déclarer les données des nouvelles tables,
- un système d'affichage des erreurs rencontrées s'il y en a lors de la déclaration de données (par exemple une contrainte de clé étrangère non respectée) avec la mise en place de règles métiers à respecter,
- un module permettant de tracer les graphes avec une ergonomie revisitée (une interface unifiée ; la possibilité d'afficher plusieurs graphes l'un au-dessous de l'autre).

2.4. Analyse des besoins non fonctionnels

2.4.1. Spécifications techniques

Le développement s'effectuera, à l'aide des outils suivants :

- **Creately**, qui permet de tracer des diagrammes UML facilement en ligne,
- **Balsamiq**, une application web permettant de faire des maquettes de site web,
- **les langages de programmation PHP, JavaScript, HMTL et CSS**,

- **Yii2 Framework**, un framework* PHP sur lequel je reviendrai plus en détail dans la partie technique,
- **le framework HTML, CSS et JS Bootstrap**, pour faciliter la partie design,
- **JetBrains PHPStorm**, un IDE* PHP qui m'a permis de travailler directement sur le serveur de tests à l'aide du protocole SCP,
-
- **le SGBD* MySQL,**
- **PHPMyAdmin**, afin de naviguer dans la base de données simplement pour vérifier certaines valeurs et la structure des tables.

2.4.2. Contraintes ergonomiques

Le site devra pouvoir être utilisé par n'importe quel chercheur. Il faudra donc une interface claire, pas trop chargée. De plus, des chercheurs étrangers peuvent être amenés à utiliser l'outil, une traduction en anglais est importante. Des maquettes ont été réalisées pour faire des démonstrations aux utilisateurs.

3. Rapport technique

3.1. Conception

L'utilisation d'un framework fait que la partie conception s'oriente surtout sur une explication de la structure de ce framework. C'est donc ce en quoi consistent les parties suivantes.

3.1.1. Présentation du framework Yii2

Yii2 est un framework PHP orienté objet suivant l'architecture MVC (Model – View – Controller) créée par une équipe de développeurs toujours actifs, afin d'incorporer au framework les dernières technologies. Utiliser Yii2 signifie avoir accès à beaucoup d'outils utiles et d'extensions permettant la mise en place native de certaines stratégies.

Yii2, sorti officiellement en 2013 est le descendant de Yii, sorti en 2008.

3.1.2. La structure et la création d'un projet Yii2

L'installation de Yii2 se fait en téléchargeant une archive qui contient déjà un site fonctionnel, une sorte de template à modifier à sa guise (cf. **Annexe 8**).

On peut distinguer sur la **Figure 1** les différents dossiers et leurs usages :

```
basic/                  application base path
    composer.json       used by Composer, describes package information
    config/             contains application and other configurations
        console.php     the console application configuration
        web.php         the Web application configuration
    commands/           contains console command classes
    controllers/        contains controller classes
    models/             contains model classes
    runtime/            contains files generated by Yii during runtime, such as logs and cache files
    vendor/             contains the installed Composer packages, including the Yii framework itself
    views/              contains view files
    web/                application Web root, contains Web accessible files
        assets/         contains published asset files (javascript and css) by Yii
        index.php       the entry (or bootstrap) script for the application
    yii                 the Yii console command execution script
```

Figure 1 : Yii2 application structure

Le dossier **config/** contient donc les différentes options de configurations. Tous les éléments du dossier construisent un tableau $config qui contient les différents éléments nécessaires au site. C'est par exemple là qu'on renseigne les identifiants de connexion à la base de données, voir **Figure 2**.

Figure 2 : fichier de configuration : identifiants de connexion à la base de données

Les dossiers **controllers/**, **models/** et **views/** sont liés à l'architecture MVC (plus de détails dans la partie suivante).

Voici comment le framework gère une requête à l'aide d'un cas précis : supposons que l'on veuille accéder à la vue de la Manip qui a pour code identifiant TH2

1. Un utilisateur fait une requête (en cliquant sur un bouton par exemple), qui se traduit dans l'URL par :
   « lps-XXX.supagro.inra.fr/phenodynV2/web/index.php?**r=manip/view&id=TH1** » où r est la requête à résoudre. Cette requête est traitée par un script qu'appelle le début de l'URL, web/index.php.
2. Ce script charge la configuration de l'application et crée une instance de l'application (Yii::$app).
3. L'application résout la requête, ce qui lui permet de savoir que manip est le controller qu'il faut instancier, et view l'action de se controller. La requête se présente toujours de la forme r=controller/action.
4. Le système créé donc une instance du controller.
5. Une instance de l'action est créée, et le système vérifie si la méthode dispose de tout ce dont elle a besoin, ainsi que d'autres filtres (comme les droits utilisateurs s'il y en a). Ici, elle a besoin de l'id d'une manip.
6. Si les filtres ne passent pas, un message d'erreur est directement envoyé et traité par le composant traitant les réponses à envoyer à l'utilisateur.
7. Sinon le code de l'action est exécuté.
8. L'action va accéder à la base de données pour aller chercher le tuple correspondant à la manip demandée.
9. L'action fournit le modèle à la vue appelée par l'action.
10. La vue ainsi créée en utilisant le model va être envoyé au composant qui traite les réponses à envoyer à l'utilisateur.
11. Ce composant va finalement envoyer le résultat au navigateur de l'utilisateur.

Ce cas précis est la traduction et la mise en situation d'un diagramme d'activité fourni par Yii2 (cf. **Annexe 9**).

3.1.3. L'architecture MVC implémentée par Yii2

La couche Model

La couche Model du concept MVC représente la partie de l'application qui exécute la logique métier. Cela signifie qu'elle est responsable de gérer les données, et la logique qui lie ces données (validation, lecture, enregistrement…).

Les modèles dans Yii2 sont représentés par des classes comme models/Capteur.php.

Ces modèles héritent de la classe ActiveRecords, qui implémente le design pattern* ActiveRecord.

Les classes ActiveRecord représentent une table de la base de données. Cette représentation est effective grâce aux éléments suivants :

- Des attributs : représentent les données métier et auxquels on peut accéder comme des objets du model ($CapteurModel->nomCapteur). On peut leur donner une valeur selon **ce que l'utilisateur sais**it à l'aide d'un formulaire, ou alors **en chargeant un tuple de la base de données**.
  On peut pour chaque model, même héritant de ActiveRecord, rajouter des attributs de à la classe.
- Des règles de validation : règles que les données chargées dans le modèle doivent respecter (voir **Figure 3**). On appelle $model->validate(), qui vérifie la valeur de chaque attribut et qui si elle échoue écrit dans le tableau $model->_errors. On peut trouver les valeurs possibles pour ces règles dans la partie « Core Validator » de la documentation de Yii2.

```php
/**
 * @return array validation rules for model attributes
 */
public function rules()
{
    return [
        [['nomCapteur', 'site', 'type'], 'required'],
        [['nomCapteur'],'unique'],
        [['site', 'type'], 'string'],
        [['nomCapteur'], 'string', 'max' => 16],
    ];
}
```

Figure 3 : règles de validation d'un capteur

Dans la **Figure 3**, on peut voir par exemple que l'élément nomCapteur doit être unique. Cela se traduit par une requête vérifiant si la valeur rentrée en attribut $modelCapteur->nomCapteur lors de l'appel à $model->validate() n'existe pas déjà.

- Des labels d'attributs : gère comment l'attribut va s'appeler dans les vues qui utilisent les modèles, voir **Figure 4**.

```php
public function attributeLabels()
{
    return [
        'codeManip' => 'Code manip',
        'dateDebut' => 'Date de début',
        'dateFin' => 'Date de fin',
        'description' => 'Description',
        'finalite' => 'Finalité',
        'responsable' => 'Responsable',
        'contact' => 'Email de contact',
        'protocole' => 'Protocole',
        'planExperience' => 'Plan expérience',
    ];
```

Figure 4 : labels d'attributs

- Les méthodes CRUD : CRUD signifie Create, Read, Update, et Delete. Ce sont les actions de bases sur des objets devant être sauvés dans la base de données. Create signifie créer un nouveau tuple, Read signifie lire un tuple, Update signifie mettre à jour un tuple et Delete signifie supprimer un tuple.

La couche Controller

La couche Controller gère les requêtes des utilisateurs. Elle est responsable de retourner une réponse avec l'aide mutuelle des couches Model et View.

Comme détaillé plus tôt, les controllers contiennent des actions, qui peuvent être appelées grâce à des URL.

Il y a 5 actions de base, que je vais détailler dans le cadre du CRUD d'un capteur.

- Action Index :

On accède à cette action à l'aide de l'URL suivant : « lps-XXX.supagro.inra.fr/phenodynV2/web/index.php?r=capteur/index ».

La partie r=manip/index peut être généralisée en r=controller/action. Ainsi r=site/about va appeler actionAbout() dans SiteController.php

```php
/**
 * Lists all Capteur models.
 * @return mixed
 */
public function actionIndex()
{
    $searchModel = new CapteurSearch();
    $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

    return $this->render( view: 'index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}
```

Figure 5 : méthode actionIndex() de CapteurController.php

Cette action va instancier deux classes nécessaires pour la vue (qui sont utilisées par des widgets, que nous allons voir plus en détail par la suite) puis appeler la méthode render ().

La méthode render() définit le rendu à afficher. Le rendu inclut le code HTML statique (le layout du site, incluant le header et le footer, se trouvant dans views/layouts/main.php) et la sortie résultant du code PHP. Le premier argument est le nom de la vue à appeler (sans l'extension PHP). Le second est un tableau contenant les données à utiliser dans la vue.

- Action Create et Update :

Dans la **Figure 6** on peut voir en détail l'action Create pour un capteur. Les commentaires expliquent le fonctionnement de celle-ci. Le fonctionnement d'Update est similaire.

```php
/**
 * Creates a new Capteur model.
 * If creation is successful, the browser will be redirected to the 'view' page.
 * @return mixed
 */
public function actionCreate()
{
    $model = new Capteur();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        /*
         * Cette méthode doit donc être accédées à l'aide d'un formulaire
         * $model->load(Yii::$app->request->post()) va vérifier
         * les éléments en POST de la requête qui a appelée cette action et les charger dans le model.
         * Par exemple si la requête contient nomCapteur=tc_01 alors $model->nomCapteur va maintenant valoir tc_01
         * $model->save() va vérifier les règles de validation
         * si tout passe, les nouvelles valeurs vont être enregistrées dans la base de données */
        return $this->redirect(['view', 'nomCapteur' => $model->nomCapteur, 'site' => $model->site, 'type' => $model->type]);
    } else {
        return $this->render( view: 'create', [
            'model' => $model,
        ]);
    }
}
```

Figure 6 : méthode actionCreate() de CapteurController.php

- Action View :

L'action View correspond au Read du CRUD. Comme on peut le voir dans la **Figure 7**, elle prend en paramètre un nom de capteur, site et type (la clé primaire de la table dans la base de données).

Ces paramètres sont récupérés dans d'URL :

« http://lps-XXX.supagro.inra.fr/phenodynV2/web/index.php?r=capteur/view&nomCapteur=50y1h_cA&site=cA&type=m »

Cette URL affiche la vue du capteur qui a comme nom 50y1h_cA, comme site cA et comme type m (pour météo).

Elle va ensuite utiliser render comme vu précédemment pour afficher la vue « view.php » et afficher les détails de cette manip (**Figure 7**).

```php
/**
 * Displays a single Capteur model.
 * @param string $nomCapteur
 * @param string $site
 * @param string $type
 * @return mixed
 */
public function actionView($nomCapteur, $site, $type)
{
    return $this->render( view: 'view', [
        'model' => $this->findModel($nomCapteur, $site, $type),
    ]);
}
```

Figure 7 : méthode actionView() de CapteurController.php

- Action Delete :

L'action Delete correspond au Delete du CRUD. Elle prend aussi en paramètre un nom de capteur, un site et un type (clé primaire de la table).

La couche Vue

La Vue retourne une présentation des données venant du modèle, elle est responsable de l'utilisation des informations dont elle dispose pour produire une interface de présentation pour l'application.

Par exemple, de la même manière que la couche Model retourne un ensemble de données, la Vue utilise ces données pour fournir une page HTML les contenant.

Comme on peut le voir dans la **Figure 8**, le dossier Views contient un dossier par Model.

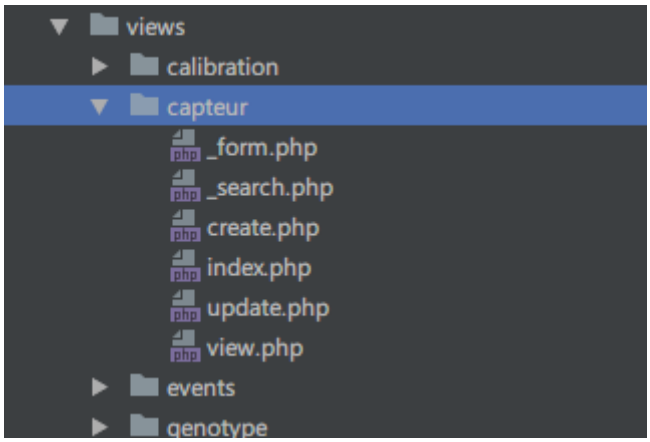


Figure 8 : dossier « views/capteur »

- « index.php » correspond à la page dans laquelle se trouve la liste de tous les éléments présents dans la table.
- « create.php » et « update.php » correspondent aux vues contenant un formulaire où créer ou mettre à jour un élément dans la table comme dans notre cas un capteur.
- « view.php » permet de visualiser un élément de la table.
- « _form.PHP » et « _search.PHP » sont des fichiers utiles à d'autres pages : le premier contient le formulaire affiché dans les vues de création et de mise à jour, tandis que le second contient un outil permettant de rechercher des éléments dans la base affiché dans l'index (cf. **Annexe 10**).

3.1.4. Créer ses propres classes

L'outil GII est un outil graphique fourni par Yii2, auquel on accède comme une page web du site, mais avec dans l'URL le QueryString* r=gii, qui permet de générer la classe model, le controller avec les actions de base CRUD, pour Create, Read, Update, Delete et les vues génériques (_form.PHP, _search.PHP, index.php, view.php, update.php, create.php) à partir d'une connexion à la base de données. Pour cela, il va observer la structure de la table pour laquelle on demande la génération du CRUD et créer un model avec des règles de validations qui respectent les contraintes d'intégrité de la base de données.

Les contraintes de clés étrangères sont respectées aussi, chaque fois que l'on veut valider, si un attribut doit respecter une contrainte de clé étrangère, un appel à la base de données est effectué pour vérifier que l'élément existe bel et bien.

Pour créer des éléments non liés à des tables, il suffit de créer les fichiers « models/XxxModel.php », « views/xxx/index.php », « controllers/XxxController.php ».

La vue peut contenir ce que vous voulez, mais les autres fichiers doivent ressembler aux autres models ou controllers, et doivent hériter des fichiers pour le premier \yii\db\ActiveRecord et de app\controllers\controller pour le second. Il faut ensuite créer l'action index dans XxxController.php comme vue précédemment pour qu'elle retourne la vue « index.php ».

3.1.5. Les extensions, les widgets et leur gestion

Yii2 propose différents éléments permettant de faciliter la vie des développeurs et les widgets en font partie. Les widgets sont des blocs de code qui s'utilisent dans les vues. La vue index basique de Yii2 utilise par exemple un widget appelé GridView, qui lorsqu'il est fourni d'une classe model permet d'afficher les tuples de la base de données et de les trier.

Les widgets peuvent être fournis dans des extensions. Ce sont des paquets redistribuables contenant des fonctions toutes prêtes à être utilisées.

Ces éléments sont gérés par un outil : Composer. C'est un outil de gestion des dépendances. Il permet de déclarer les librairies dont le projet dépend et de le gérer (installation/mise à jour). Pour cela, il va chercher dans les banques d'extensions de git ou autres VCS (Version Control System) et les télécharges. Dans ces paquets sont définies des règles comme dans la **Figure 9**, qui permettent de garder tous les paquets compatibles entre eux.
Les extensions installées sont stockées dans le dossier « vendor/ » du projet.

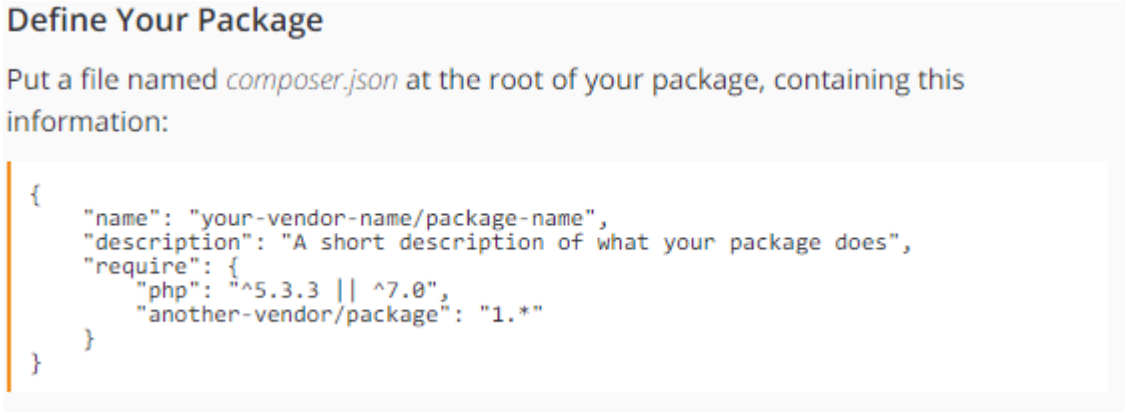


Figure 9 : Création d'un paquet Composer

3.1.6. Les classes d'aide

Il existe des classes définies dans le framework destinées à faciliter la programmation. On peut citer par exemple la classe « yii\helpers\Html », que l'on import de la façon suivante, avant la définition de la classe : « use yii\helpers\Html ». Elle fournit des méthodes comme celles vues dans l'**Annexe 10**. Ces méthodes liées à la classe activeForm que l'on importe de la même façon permettent de générer des formulaires dynamiques selon le schéma de la base de données.

On peut aussi utiliser cette classe d'aide de façon statique, par exemple :

```
<?php
    echo Html::dropDownList("list", ["poire", "pomme"]);
?>
```

Cela donnera un élément HTML « select » dont l'attribut name sera « list » et dont les options seront celles fournies dans le tableau en deuxième argument.

Yii2 possède de nombreuses classes d'aide comme celle-ci.

3.2. Résultat

3.2.1. Les outils utilisés

Dans un souci de simplicité, je vais lister ici les outils utilisés, puis j'y ferais référence dans les parties suivantes :

- Windows 10 (développement sur l'IDE PHPStorm, envoi sur le serveur de test (Linux Debian) par SCP),
- un serveur de test web avec Apache2
- une base de données de test avec MySQL,
- les langages HTML, CSS et PHP et JavaScript (utilisation d'Ajax),
- le framework PHP Yii2,
- la librairie phpoffice/PHPExcel,
- la librairie kartik-v/yii2-widgets et en particulier le widget select2,
- l'extension yii2 Jui, qui fournit le widget DatePicker qui permet d'afficher un calendrier sur lequel on peut cliquer pour choisir une date
- les librairies JavaScript JQuery et HighCharts.

3.2.2. Le CRUD des tables de Phenodyn

Un des éléments majeurs du projet a été d'établir un CRUD correct pour les différentes tables. Il a d'abord fallu générer le CRUD initial de chaque table, avec l'outil gii décrit précédemment. Il suffit de spécifier la table dans la base de données pour générer un modèle, de spécifier le modèle pour générer le controller, et de spécifier le modèle et le controller pour générer les vues basiques.

Grâce aux formulaires web, les chercheurs pourront ajouter des éléments dans la base de données, par exemple de nouveaux capteurs ou de nouvelles plantes. Les pages importantes sont donc celles de création et de mise à jour.

La méthode rules du modèle est l'élément le plus important du projet. Comme vu plus haut dans la partie sur la couche Model implémentée par Yii2, elle permet d'appliquer des règles de validation sur les attributs du model. Ces règles doivent être vérifiées à l'enregistrement des données ou la mise à jour de celles-ci.
On peut distinguer plusieurs règles importantes :

- La règle « required » qui permet de respecter la règle « NOT NULL » des tables de la base de données, sur la clé primaire par exemple.
  À l'enregistrement l'attribut ayant la règle « required » devra contenir quelque chose, sinon une erreur surviendra.

- La règle « unique ». Elle permet de respecter les contraintes d'unicité, comme celle sur la clé primaire. Par exemple, les codes identifiants les manips doivent être uniques.
- La règle « exist » qui permet de respecter les contraintes de clé étrangère. À la création d'une plante, il faut renseigner un code identifiant un pot auquel elle est liée. Cette règle va effectuer la requête vérifiant que ce code existe dans la base de données.
- Les règles plus basiques, comme « number » qui vérifie qu'un attribut est un nombre, ou email ; qui vérifie qu'un attribut à la structure d'une adresse email.
- Les règles personnalisées, qui appellent une méthode définie par le programmeur. Cette méthode exécute des vérifications sur la valeur d'un ou plusieurs attributs et renvoie « true » si tout s'est bien passé, et false sinon.
  Par exemple, lors de l'insertion dans periodeMesureBalance, un validateur vérifie à l'aide d'une requête SQL s'il n'y a pas déjà une assignation pour ce capteur lors de la période donnée. S'il y en a une, il affiche un message d'erreur (plus de détail en **Annexe 11**).

Chaque modèle a eu droit à ses règles de validations qui ont dû être vérifiées par les chercheurs. En effet, il ne faut pas quelque chose de trop permissif pour éviter les erreurs, mais il ne faut pas non plus des règles trop complexes qui les limitent dans leur travail.

3.2.3. Déclaration multiple par fichier CSV ou XLSX

La fonctionnalité la plus utilisée est complémentaire au CRUD, c'est la déclaration de données par fichier XLSX ou CSV. En effet, les chercheurs déclarent souvent plusieurs éléments d'un seul coup, comme 200 plantes. Le système existant proposait un système de déclaration par import de fichier CSV. Il a donc fallu réimplémenter cette fonctionnalité. Pour cela a été utilisé la librairie PHP PHPExcel, installée grâce à Composer de la façon détaillée précédemment.

J'ai fait le choix de créer les fichiers ImportController.php et views/import/index.php afin de traiter le problème. Un modèle n'est pas nécessaire, car le controller va utiliser les models des autres classes, celles dont on veut déclarer les données.

La fonctionnalité se déroule en différentes étapes :

Un formulaire dans la vue index permet de choisir le type de fichier que l'on veut importer (cf. Figure 10) et affiche les règles à suivre ainsi que les entêtes à utiliser.

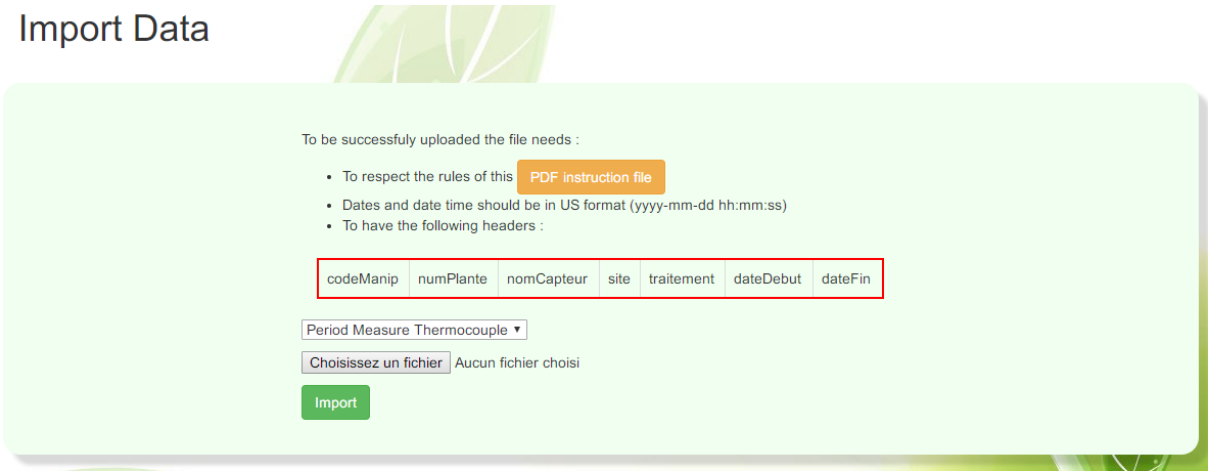


Figure 10 : vue index de views/import

Le formulaire a pour action la méthode actionImport() de « ImportController.php ». Pour plus de détails, le formulaire est donné en **Annexe 12**.

Ce fichier va ensuite être uploadé dans le serveur dans le dossier « web/uploads/import/ », à l'aide de la méthode de la **Figure 11**.

```php
private function saveImportedFile($file,$typeImport,$ext){
    $pathToFile = __DIR__ . DIRECTORY_SEPARATOR . '..' . DIRECTORY_SEPARATOR . 'web' .
        DIRECTORY_SEPARATOR . 'uploads' . DIRECTORY_SEPARATOR . 'import' . DIRECTORY_SEPARATOR .
        $typeImport .' - ' .date( format: 'Y-m-d h:i:s').'.'. $ext;
    move_uploaded_file($file, $pathToFile);
    return $pathToFile;
}
```

Figure 11 : méthode d'upload de fichier

Selon le type d'import choisi, une méthode différente est appelée à l'aide d'un « switch ». On va traiter le cas de la déclaration d'un capteur.

On sélectionne le type capteur, on uploade un fichier construit comme le schéma d'en-tête affiché, et à l'aide du « switch », la méthode actionImportCapteur va être appelée avec en argument le chemin du fichier qui a été uploadé.

C'est cette méthode qui va utiliser la librairie PHPExcel. On va créer une instance du parser qui va lire le fichier (celui-ci est différent selon si le fichier est de type CSV ou XLSX).

Figure 12 : méthode actionImportCapteur()

```php
function actionImportCapteur($pathToFile)
{
    $errors = [];
    $ligneErreur = [];
    $successful = true;
    $objReader = PHPExcel_IOFactory::createReader(PHPExcel_IOFactory::identify($pathToFile)); //identify donne le type, permet de fonctionner pour CSV ou XLSX

    if ((strpos(get_class($objReader), needle: 'CSV') !== false)) {
        $objReader->setDelimiter($this->getFileDelimiter($pathToFile)); //si CSV, getFileDelimiter va observer le fichier et donner son délimiteur de données
    }

    $objPHPExcel = $objReader->load($pathToFile);
    //la ligne suivante est nécessaire pour obtenir la feuille, même s'il n'y en a qu'une
    $sheet = $objPHPExcel->getSheet( pIndex: 0);
    $highestRow = $sheet->getHighestRow(); //renvoie le numéro de ligne max
    $highestColumn = $sheet->getHighestColumn(); //renvoie la lettre de colonne max

    $transaction = Yii::$app->db->beginTransaction();

    for ($row = 2; $row <= ((int)$highestRow); $row++) {
        //la ligne suivante créé un tableau qui correspond à une ligne et qui change à chaque tour de boucle
        $rowData = $sheet->rangeToArray( pRange: 'A' . $row . ':' . $highestColumn . $row, nullValue: NULL, calculateFormulas: TRUE, formatData: FALSE);

        $modelCapteur->nomCapteur = (string)$rowData[0][0];
        $modelCapteur->site = (string)$rowData[0][1];
        $modelCapteur->type = (string)$rowData[0][2];


        if (!$modelCapteur->save() || !$modelCapteur->validate()) {
            $errors[] = $modelCapteur->getErrors();
            $ligneErreur[] = $row;
            $successful = false;
        }
    }
    if ($successful) {
        $transaction->commit();
    } else {
        $transaction->rollback();
    }
    Yii::$app->session->set('successful',$successful);
    Yii::$app->session->set('errors',$errors);
    Yii::$app->session->set('ligneErreur',$ligneErreur);

    return $this->redirect([
        'import/index',
    ]);
}
```

Comme on peut le voir dans la **Figure 12**, une boucle va se charger de parcourir chaque ligne du fichier en créant un tableau contenant une ligne après l'autre à chaque tour de boucle. Cette façon de faire permet d'économiser beaucoup de temps par rapport à l'utilisation d'un tableau contenant toutes les lignes.

L'enregistrement dans la base de données se fait grâce à la méthode « save() » et « validate() », qui ont été décrite dans la partie 3.1.2.

Ces interactions avec la base de données sont englobées dans une transaction*. Cela permet de ne garder aucune modification si une erreur survient. La liste des erreurs est envoyée à l'utilisateur qui devra corriger son fichier et l'uploader à "nouveau.

3.2.4. La gestion des erreurs de la déclaration multiple

La gestion des erreurs est importante pour chacune des parties du site. Heureusement, les formulaires Create et Upload ont leur propre gestion d'erreur, côté client (Ajax qui entoure en rouge le champ s'il contient quelque chose de faux) et côté serveur (qui redirige vers la page en créant une alerte bootstrap).

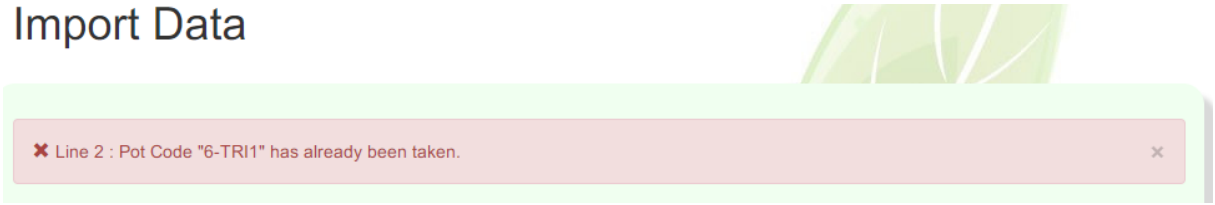


Figure 13 : Message d'erreur lors de la déclaration multiple

Cette méthode d'affichage d'alertes bootstrap a donc été reprise pour l'import multiple, voir **Figure 13**. Comme on peut le voir en **Figure 12**, les erreurs sont passées dans la session PHP. Dans la vue index, on vérifie si l'utilisateur possède ces éléments dans la session qui lui correspond. Si c'est le cas, selon sa valeur, on affiche une alerte bootstrap contenant l'erreur, et on efface de la session les erreurs, pour qu'elles ne se réaffichent pas.

Ici il y a 2 types d'erreurs : une erreur lors du téléchargement (upload) ou une erreur de validation. Dans les deux cas, la méthode s'arrête et redirige l'utilisateur vers la page actuelle, mais cette fois comme l'utilisateur possède des erreurs dans sa session des alertes seront générées.

Un fichier texte contenant les erreurs de validation est proposé en téléchargement afin de pouvoir reprendre la correction du fichier de déclaration ultérieurement

3.2.5. Les graphiques

La visualisation des données par graphiques est une des fonctionnalités déjà présentes dans le système d'informations existant, il faut donc la ré implémenter. Pour cela j'ai utilisé la même librairie JavaScript : HighCharts.

Afin de faciliter la création des graphes, des menus dynamiques sont mis en place. Il y a deux types de menus comme on peut le voir en **Annexe 13**

- un menu pour les chercheurs, qui permet de trier les données par plante ou pot (une balance est liée à un pot, un capteur de croissance à une plante,…),
- un menu pour les techniciens qui ne font pas de recherche par plante ou par pot mais directement par nom de capteurs. Les capteurs pourront être triés en fonction de leur emplacement dans le dispositif..

Ces menus déroulants sont remplis dynamiquement grâce à des requêtes de type SELECT dans la base de données. Pour les codes identifiant les manips, il n'y a pas de contrainte, c'est donc un simple « SELECT » SQL. Par contre, une fois que l'utilisateur a choisi une manip en particulier, la liste des pots ou des plantes est réduite aux pots ou aux plantes liées à cette manip (ajout d'une clause WHERE).

Cette fonctionnalité est mise en place par des requêtes Ajax qui s'exécutent à la sélection d'une manip, à l'aide d'une méthode exécutée par un EventListener JavaScript. Une requête Ajax va appeler une méthode de GrapheController.php qui en se voyant fournit la manip sélectionnée, va fournir une liste de pots ou de plantes. De la même façon une fois un pot ou une plante sélectionnée, à l'aide d'une autre requête Ajax la liste des capteurs est mise à jour comme dans la **Figure 14**.

```php
/**
 * Doit être appelé avec : un code manip ($_POST['codeManipBalance']) et des codePot dans un string ($_POST['codePotBalance']=''01-TRI1','02-TRI1'',
 * @return string un tableau d'objets [{nomCapteur:"tc01_s1"},{nomCapteur:"tc02_s1"}]
 */

public function actionGetSensorByPot()
{
    if ((isset($_POST['codeManipBalance']) && (isset($_POST['codePotBalance']))
        && ($_POST['codeManipBalance'] == "") && $_POST['codeManipBalance'] == ""))
    {
        $codeManip = $_POST['codeManipBalance'];
        $codePot = $_POST['codePotBalance'];

        $query = 'SELECT DISTINCT nomCapteur FROM periodeMesureBalance WHERE codeManip=\'' . $codeManip . '\' AND codePot IN (' . $codePot . ')';

        $command = Yii::$app->db->createCommand($query);
        $data = $command->queryAll();

        if (!($data === null || $data === "")) {
            return Json::encode(array_values($data));
        } else {
            return Json::encode( value: "");
        }
    }
    else return "Error while trying to fetch sensors";
}
```

Figure 14 : Méthode actionGetSensorByPot() de GrapheController.php

Une fois qu'un capteur, une date de début et une date de fin sont choisis, une autre requête Ajax va s'effectuer, fournissant la liste des capteurs sélectionnés à une autre méthode du controller de graphe. Une requête SQL va s'exécuter autant de fois qu'un capteur est sélectionné pour récupérer les données.

Le résultat de la requête est formaté comme on peut le voir en **Annexe 14** afin de respecter le format de données qu'attend la librairie HighCharts tel qu'on peut le voir dans la **Figure 15**.

Ces données sont passées à la fonction s'exécutant en cas de succès de la requête Ajax, qui va faire appel à une fonction s'occupant de tracer le graphe.

Figure 15 : affichage de la variable "options" à fournir au constructeur de graphe

La checkbox que l'on peut voir en **Annexe 13** permet d'afficher les graphes un par un, ou de les ajouter les uns sur les autres.

3.2.6. L'internationalisation

Le LEPSE est un laboratoire qui accueille souvent des étrangers dans son enceinte. Un choix des langues a donc été créé, Anglais et Français. La traduction se fait à l'aide de deux choses :

- lLa méthode Yii::t;
- les fichiers de traduction.

Les fichiers de traduction se trouvent dans « translations/ » avec un dossier par langue à l'intérieur de celui-ci. On peut créer des fichiers dans ces dossiers, qui sont des tableaux associatifs associant un texte à un autre, comme on peut le voir en **Figure 16**.

```
'Add one plant' => 'Ajouter une plante',
'Are you sure you want to delete this item' => 'Etes-vous sûr de vouloir supprimer cet élément ?',
```

Figure 16 : extrait du fichier translations/fr/app.php

La méthode Yii::t(app,message) va traduire le message en fonction du langage défini et de ces fichiers de traduction. Chaque élément de texte du site est constitué de cette méthode.

Le nom de ces fichiers correspond à la catégorie renseignée dans la méthode Yii :: t. Pour ce projet aucune catégorie n'a été créée, et toutes les traductions anglais-français sont dans le fichier « en_US/app.php » et toutes les traductions français-anglais sont dans le fichier « fr/app.php ».

Deux liens en haut de la page notés « anglais » et « français » appellent une méthode de SiteController.php (controller de base dans un projet Yii2, permet par exemple d'accéder à la page « login » ou « contact »). Cette méthode vérifie sur lequel de ces deux liens on a cliqué et défini dans la session un élément « langage » contenant celui sélectionné.

Il a ensuite fallu trouver comment charger chaque page avec la bonne langue. Cela s'est fait à l'aide de la méthode Init() des controllers qui s'exécute automatiquement avant chaque action.

Tous les controllers vont hériter d'un autre controller implémentant la méthode Init() définissant le langage pour chaque action à l'aide de la ligne de code suivante :

```
Yii::$app->language = 'fr';
```

La chaine de caractère 'fr' est remplacée par la valeur de `$session['language']` qui correspond à un code identifiant un pays. Si la session ne contient pas de langue, alors l'anglais est choisi.

3.2.7. Le design

Tout le design du site a été réalisé en utilisant le framework bootstrap. Pour positionner les éléments, j'ai utilisé le principe de grille bootstrap. Selon ce principe, la largeur de l'écran est divisée en 12, et à l'aide de classe tels que col-*-5 on peut par exemple définir une colonne prenant 5/12 de l'écran.

3.2.8. Les perspectives de développement

Le site, ayant été développé sur un framework, suit une logique de conception qui permet de reprendre le projet et de continuer facilement en connaissant le framework. Certaines fonctionnalitées n'ont encore pas pu être programmées à savoir :

- la gestion d'utilisateurs possédant des droits d'accès différents,

- la mise en place de services web permettant de communiquer avec d'autres système d'informations ou avec des d'autres programmes (script R permettant l'analyse des données par exemple),

- la mise en place de DOI* permettant d'identifier de manière unique et pérenne les données.,

- l'optimisation des performances (voir http://www.w3ii.com/fr/yii2/guide-tutorial-performance-tuning.html) car certaines tables sont très volumineuses (plus de 150 millions d'enregistrements pour certaines).

Le site, ayant été développé sur un framework, suit une logique de conception qui permet de reprendre le projet et de continuer facilement en connaissant ce framework.

4. Manuel d'utilisation

4.1. Consultation des données

Pour consulter les données, l'utilisateur peut utiliser le menu en haut du site, comme dans la figure suivante.

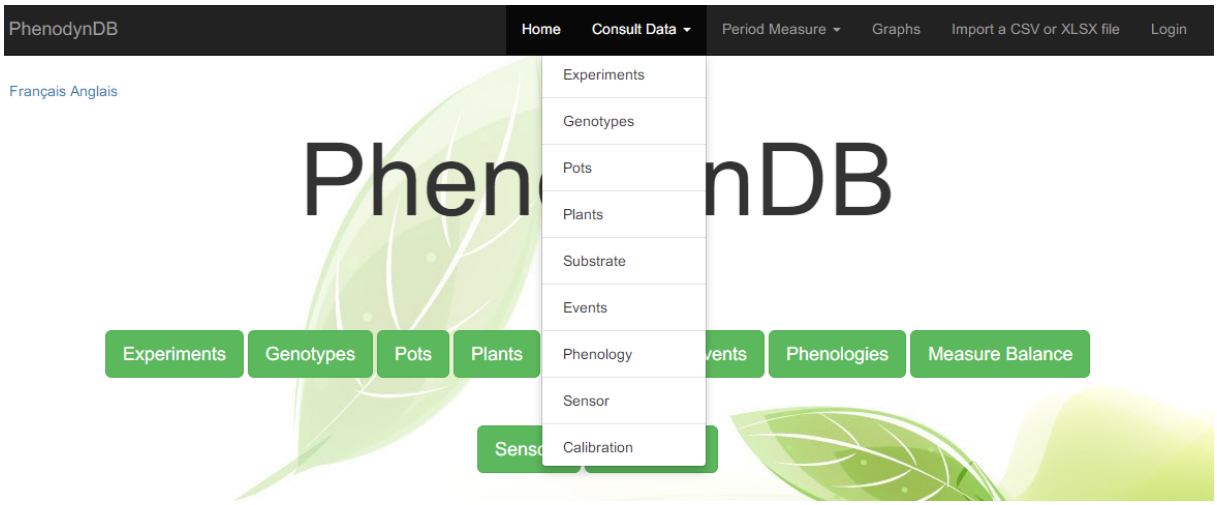


Figure 17 : Menu de PhenodynV2

Cela l' amènera vers une page contenant une liste des données. Chaque ligne de la liste est un enregistrement (« tuple ») dansd la base de données.

4.2. La déclaration des données

4.2.1. La déclaration unique

Une fois sur la page listant tous les enregistrements d'une table, l'utilisateur peut choisir de déclarer un nouvel élément, comme une plante ou un pot, voir **Figure 18**. Cela conduira à un formulaire (voir **Figure 19**) permettant de choisir les éléments à insérer dans la base de données.

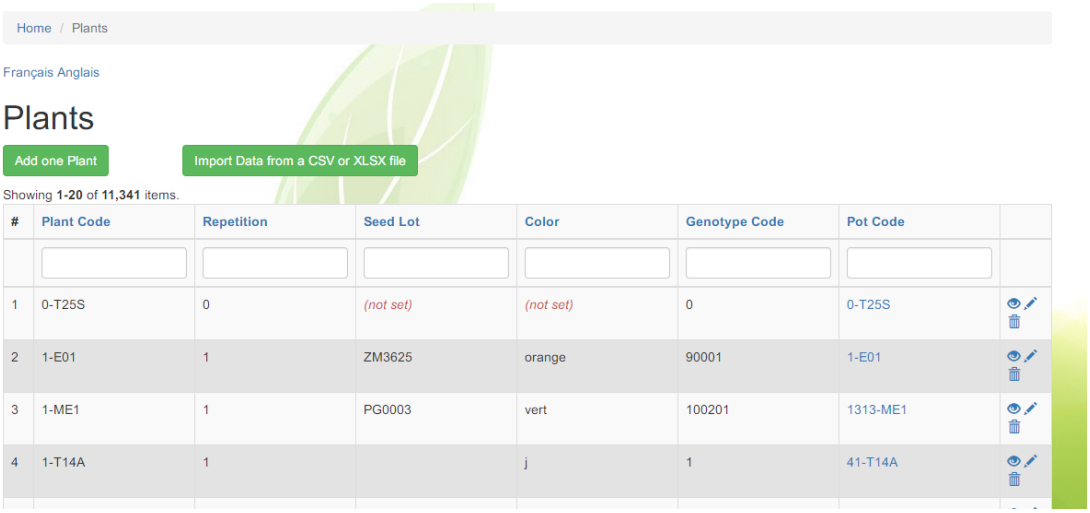


Figure 18 : déclaration de données (déclaration uniqueou déclaration multiple)

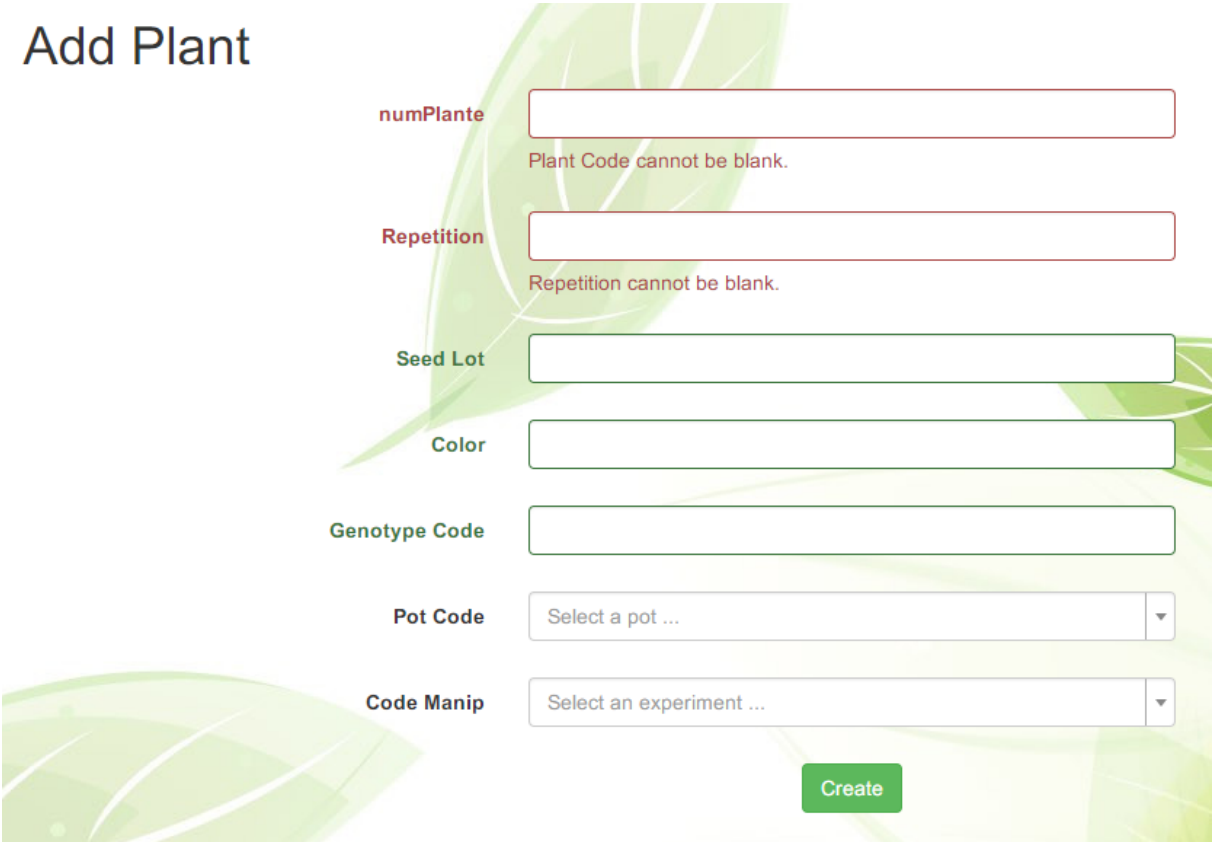


Figure 19 : formulaire de déclaration unique

4.2.2. La déclaration multiple

Les utilisateurs choisissent majoritairement cette fonctionnalité. Elle permet de télécharger dans le site un fichier CSV ou XLSX pour déclarer plusieurs éléments d'un coup. Pour cela comme on peut le voir en **Figure n 19**, il faut choisir un type d'élément que l'on veut importer. Il suffit ensuite de cliquer sur « choisissez un fichier », de sélectionner le fichier CSV ou XLSX voulu, puis de cliquer sur le bouton vert « Import » (voir **Figure n°20**). Ce

fichier devra suivre les règles affichées sur la page, notamment l'ordre des headers. De plus, il faudra suivre les règles du fichier PDF mis en lien (bouton orange). S'il y a des erreurs dans le fichier l'enregistrement ne se fait pas et l'utilisateur doit les corriger avant de resoumettre le fichier.

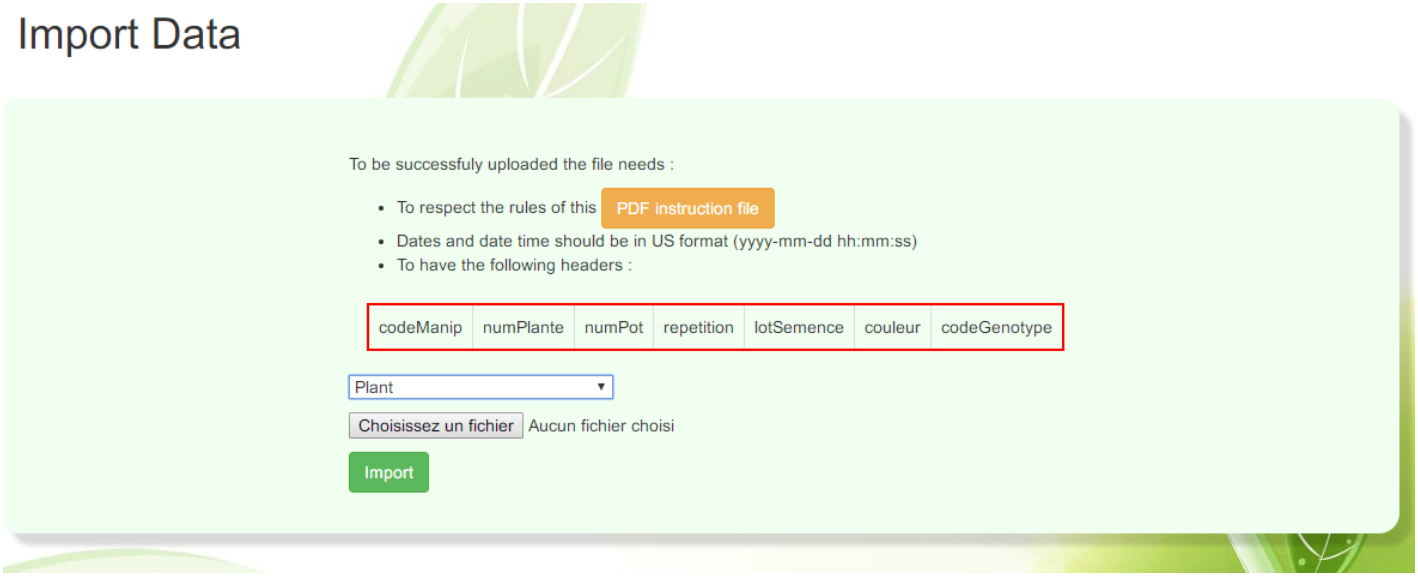


Figure 20 : import multiple avec l'élément plante sélectionné

4.3. Tracer un graphe

Pour obtenir un graphe, il faut sélectionner un capteur. Les autres menus permettent de trier la liste de ces capteurs. Je vais donc détailler le cas d'un graphe de l'évolution du poids du pot par rapport au temps.

Des balances sont liées à des pots, et pour sélectionner la bonne balance il faut soit connaître son nom de capteur, soit le trouver en remplissant les différents menus.

Choisir un code identifiant une manip va permettre d'afficher la liste des pots liés à celle-ci. On peut choisir plusieurs éléments en les sélectionnant en restant appuyé sur la touche CTRL. Choisir un pot va afficher son capteur correspondant. Il suffit ensuite de fournir une date de début et une date de fin pour afficher le graphe des données pour ce capteur de la date de début à la date de fin.

Cliquer à l'intérieur d'un menu et écrire quelque chose va déplacer la sélection sur un élément de la liste qui contient au début de son nom ce qui a été écrit.

5. Gestion de projet

5.1. Démarche personnelle

Le but de ce stage était de créer un projet en autonomie. En ce sens, mon encadrant Vincent NÈGRE m'a donné ses conseils afin d'établir les différentes étapes, mais j'ai choisi de coder un maximum sans le solliciter. Des points réguliers (à minima hebdomadaire) était cependant fait avec lui.

Le développement de ce projet a été réalisé dans les locaux du LEPSE, ce qui signifie que les utilisateurs travaillaient juste à côté de mon poste de travail. Le cycle de développement n'était donc pas un cycle en cascade, avec une seule livraison finale, mais un cycle itératif, avec des retours utilisateurs dès qu'une fonctionnalité était terminée. Certains points devaient nécessairement être réalisés dans l'ordre, par exemple la création de la structure du projet et la création du CRUD des tables de la base de données devaient être faites tout au début. Une fois ces fonctionnalités établies, les utilisateurs ont pu choisir de rajouter certaines fonctionnalités, comme l'ajout de document lié à une manip, qui m'a été donné à réaliser avant de continuer ce qui était prévu au début.

L'organisation de réunion (réservation de salle à l'aide d'outils internes au LEPSE) après la réalisation de chaque grosse fonctionnalité a permis de présenter les nouveautés, et surtout de définir avec précisions les besoins utilisateurs. Étant un système conçu pour les utilisateurs, ce sont à eux de cerner les fonctionnalités qui leur serviraient. Il a fallu faire la part des choses entre ce que l'on appelle en méthode agile la valeur client, c'est-à-dire à quel point la fonctionnalité est importante pour le client, et l'effort, qui est la difficulté et surtout le temps que l'on va devoir accorder à une fonctionnalité. A titre d'exemple nous avons décidé de ne pas implementer une gestion complexe des droits utilisateurs car cela nécessitait trop de temps et n'apparait pas comme une priorité pour les utilisateurs.

Les principes des méthodes agiles ont donc inspiré la démarche suivie lors du développement de ce projet.

5.2. Planification

Des outils découverts à l'IUT lors des différents projets ont permis de bien le projet, par exemple Gantt et Trello.

Trello est un site web permettant de créer des genres de Post its et de les organiser par colonne. Cela m'a servi de liste de choses à faire. J'y ai créé différentes colonnes, comme « fonctionnalités » ou « Tâches » (tâches comme contacter l'utilisateur X) et y créait des « Post its » (voir **Figure 21**).

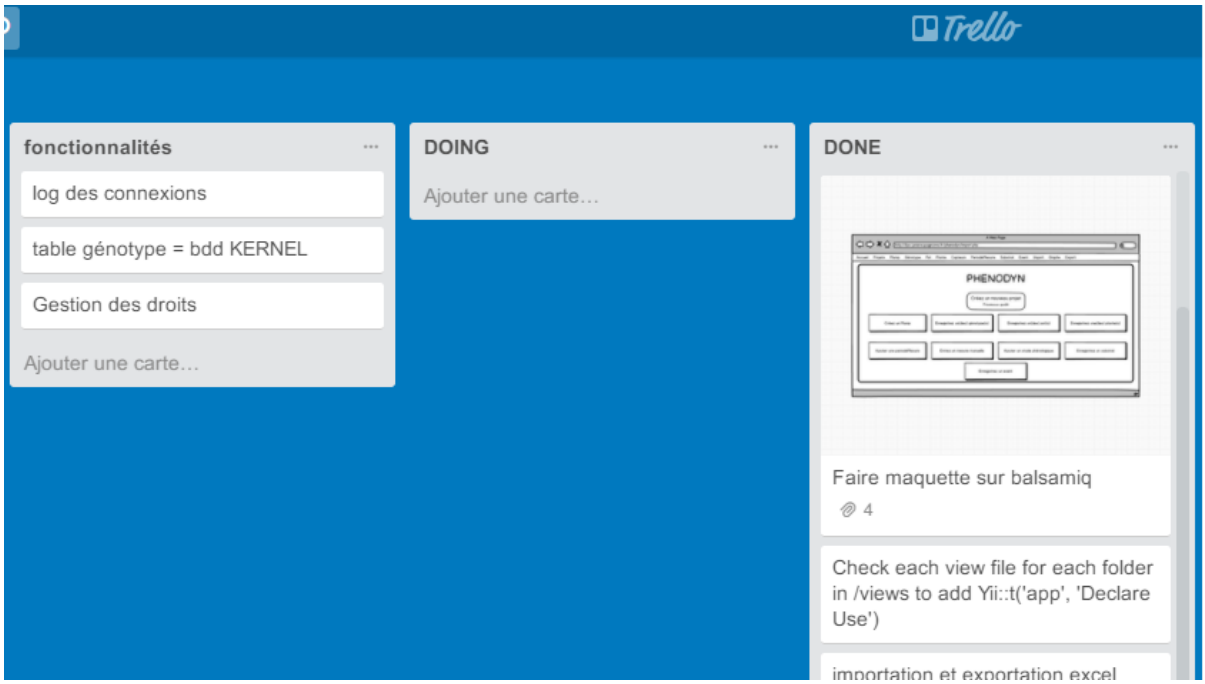


Figure 21 : l'application web de gestion de projet Trello

Un carnet de bord sur Google Drive m'a permis de noter différents commentaires lorsque je travaillais, afin de pouvoir reprendre une fonctionnalité ultérieurement.

Les diagrammes de Gant m'ont permis de planifier le projet. Un diagramme de Gantt prévisionnel a été établi au début du projet, voir **Annexe 15**.

Ce diagramme était une prévision assez floue, car ne connaissant pas les technologies, j'avais du mal à définir le temps que je mettrai à les prendre en main. On peut voir les différentes fonctionnalités prévues qui ont en fait été établies lors de l'analyse de l'existant.

Ce diagramme prévisionnel ne se trouve pas finalement si loin de la vérité, comme on peut le voir dans le diagramme de Gantt final à l'**Annexe 16**. La tâche « Prise en main de Yii2 » est étirée sur tout le long du projet, car l'apprentissage du framework a en fait pris du temps sur toutes les autres tâches, car pour chaque fonctionnalité il fallait se renseigner sur la meilleure pratique à utiliser par rapport au framework.

Les dates pour chaque fonctionnalité se sont trouvées être optimistes. Par exemple il a fallu passer plus de temps que prévu sur la création de l'outil permettant de tracer des graphes. La fonctionnalité de gestion des droits a été abandonnée pour l'instant, peut-être sera-t-elle programmée après la soutenance de ce projet. En effet la faire rapidement en la bâclant ne constituait pas une solution.

Bibliographie/Sitographie

INRA de Montpellier :

LEPSE : https://www6.montpellier.inra.fr/lepse/

Wikipédia :

Creately (modélisation UML) : https://creately.com

Guide official Yii2 :

Stack Overflow :

Yii2.0 Cookbook :

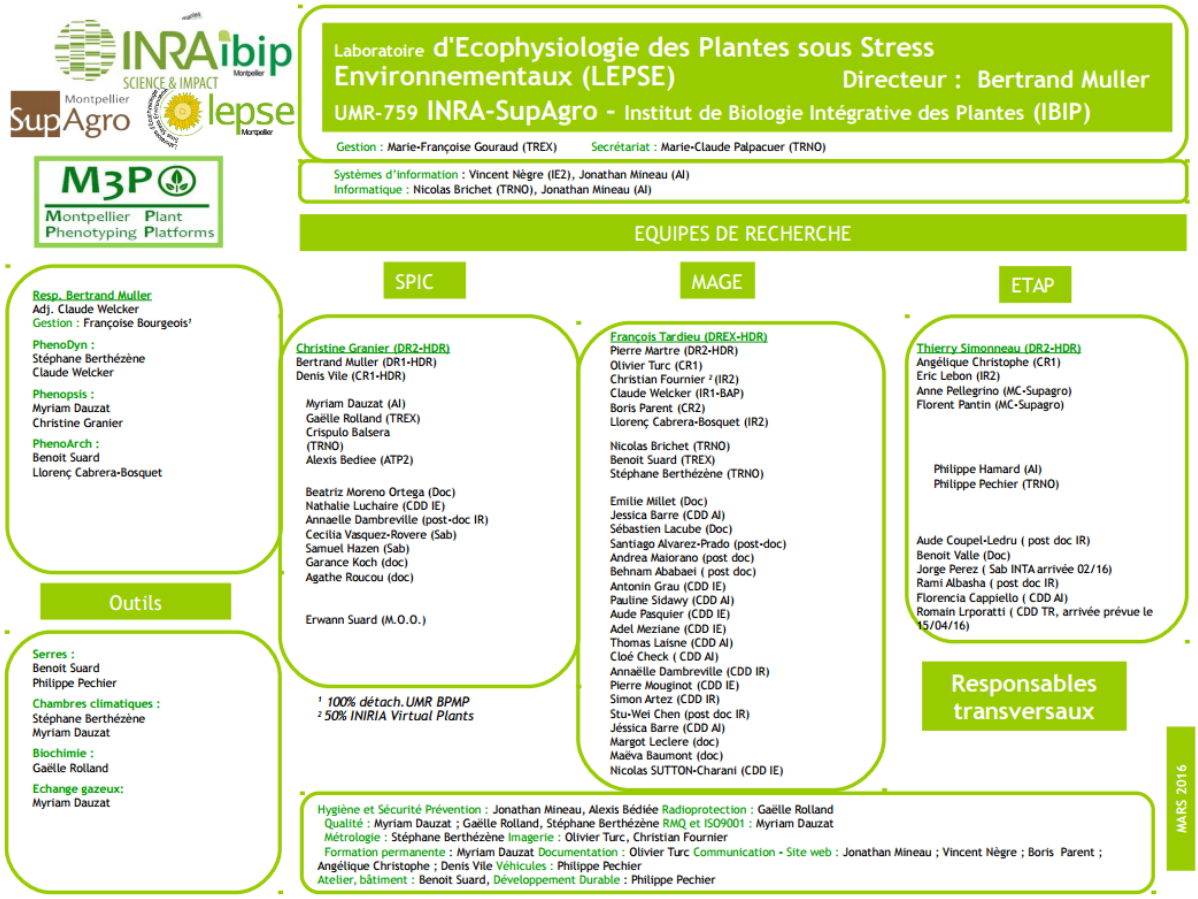
Guide Bootstrap :

Issues Yii2 GitHub :

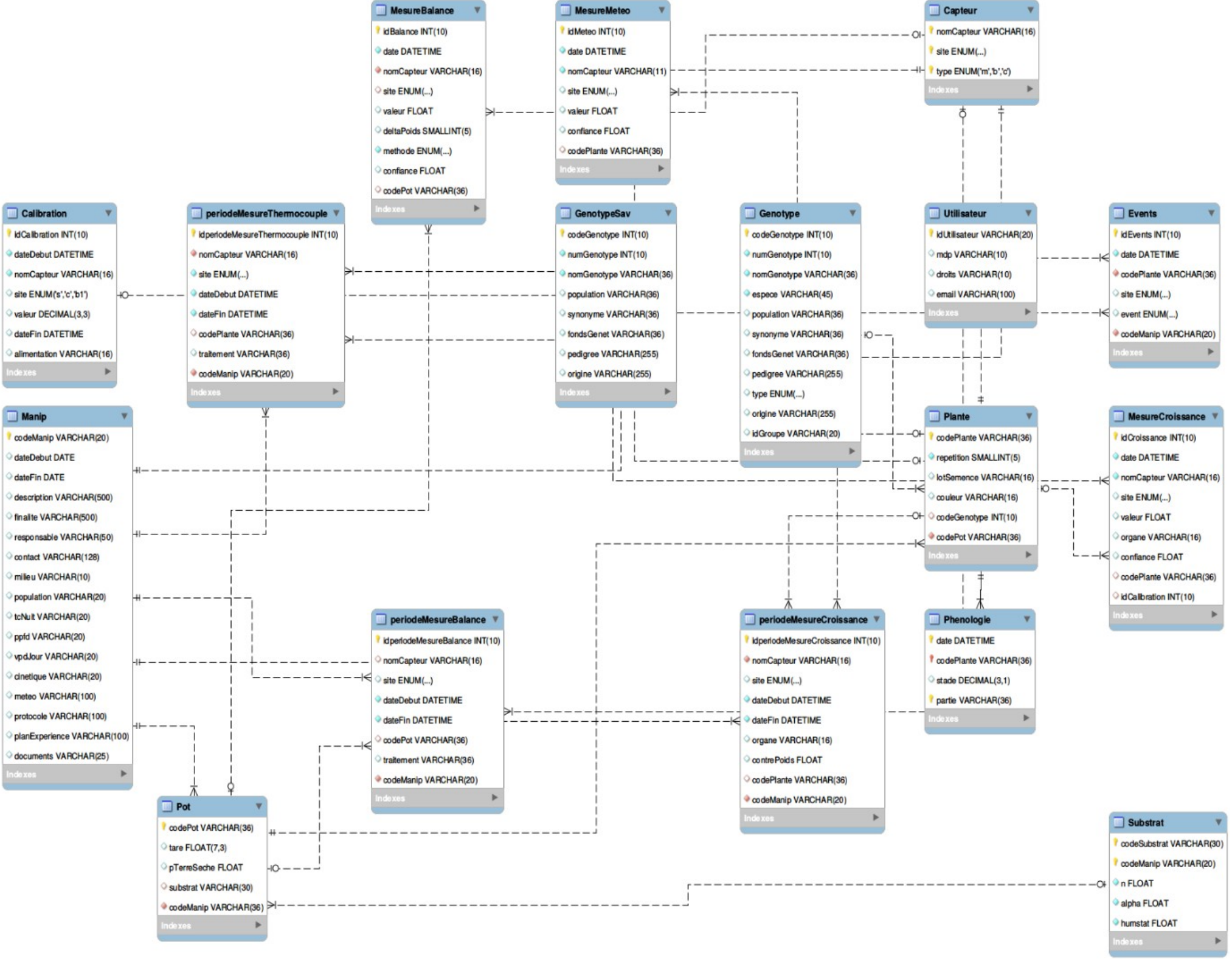
Table des annexes

Annexes

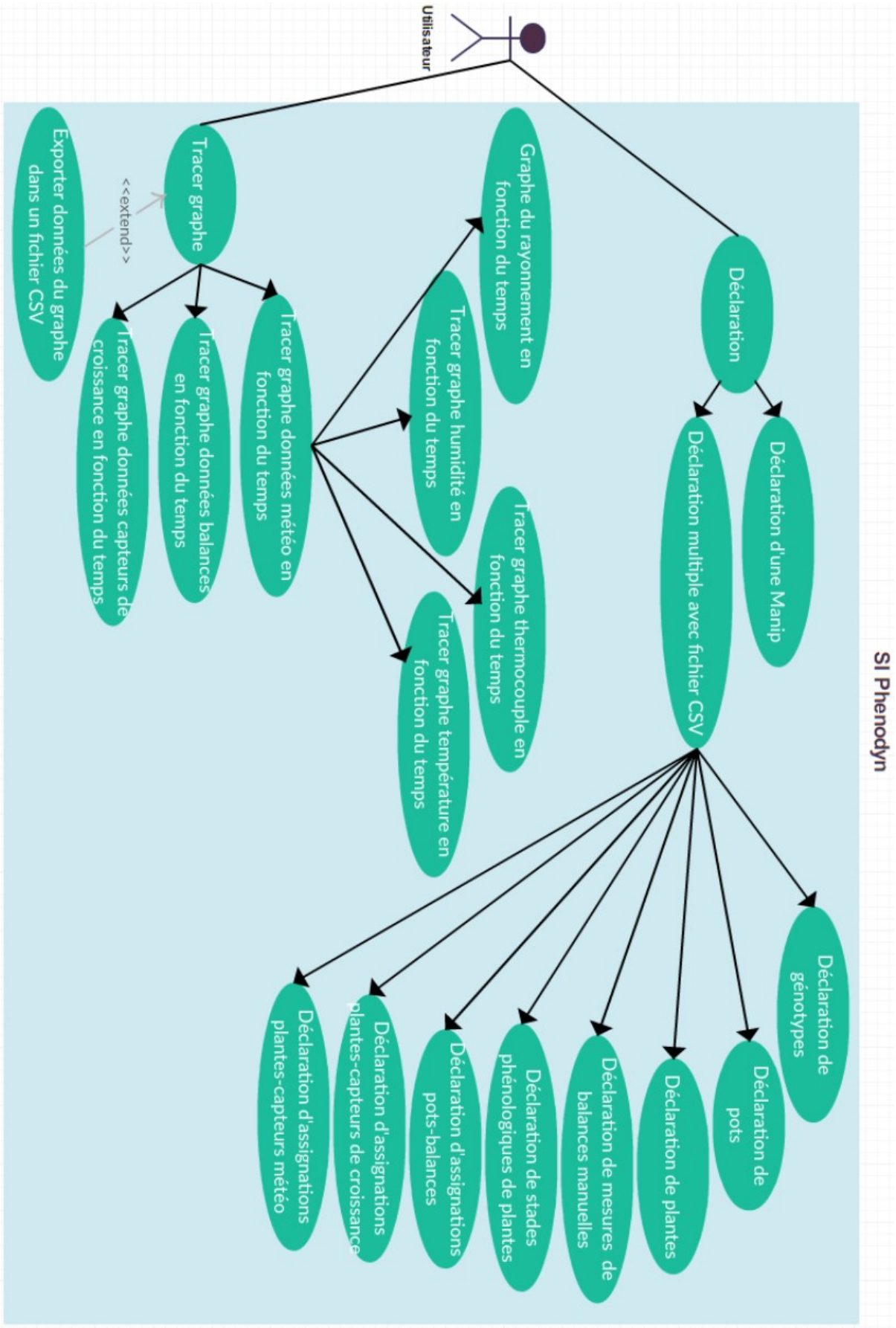
Annexe 1 : plaquette M3P



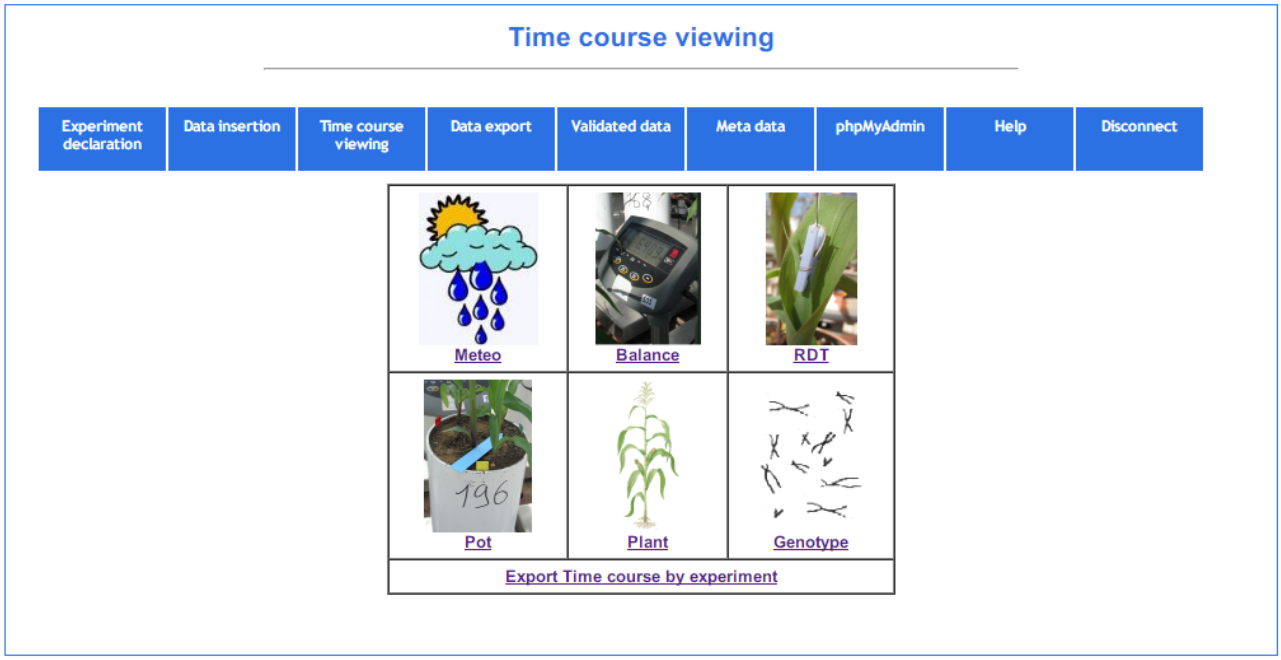
Annexe 2 : organigramme du LEPSE



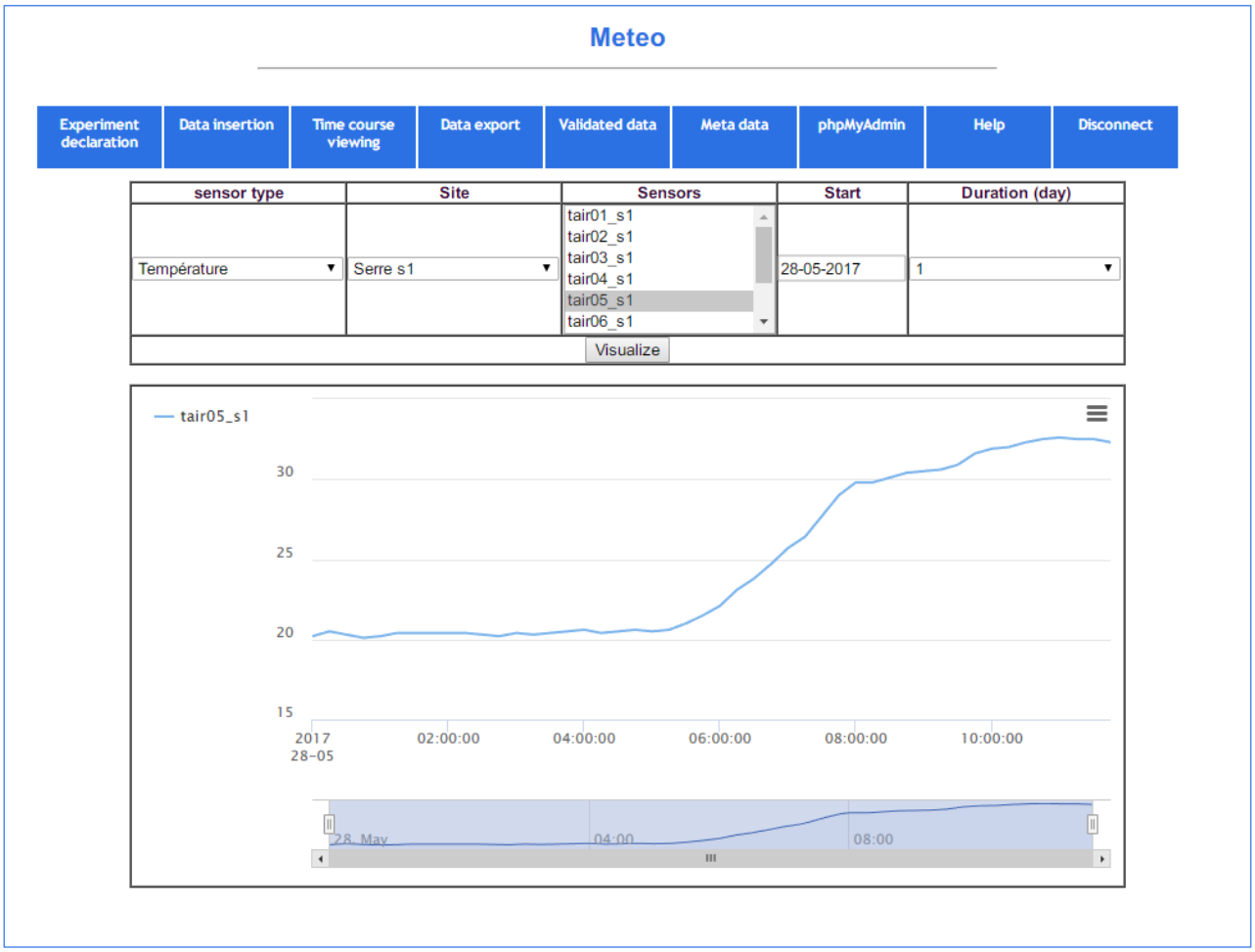
Annexe 3 : Model Conceptuel de Données PhenodynDB



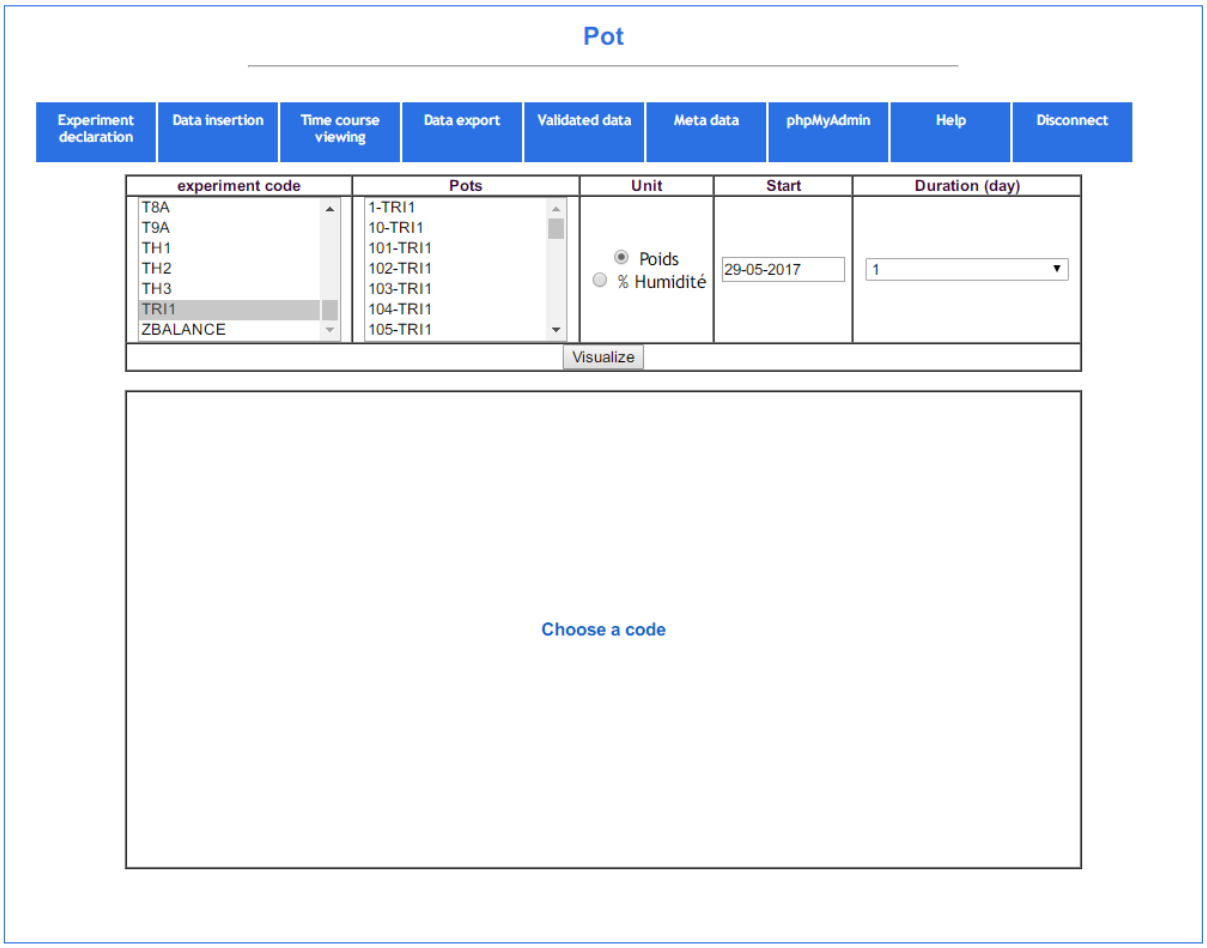
Annexe 4 : diagramme de cas d'utilisation du SI PhenodynDB



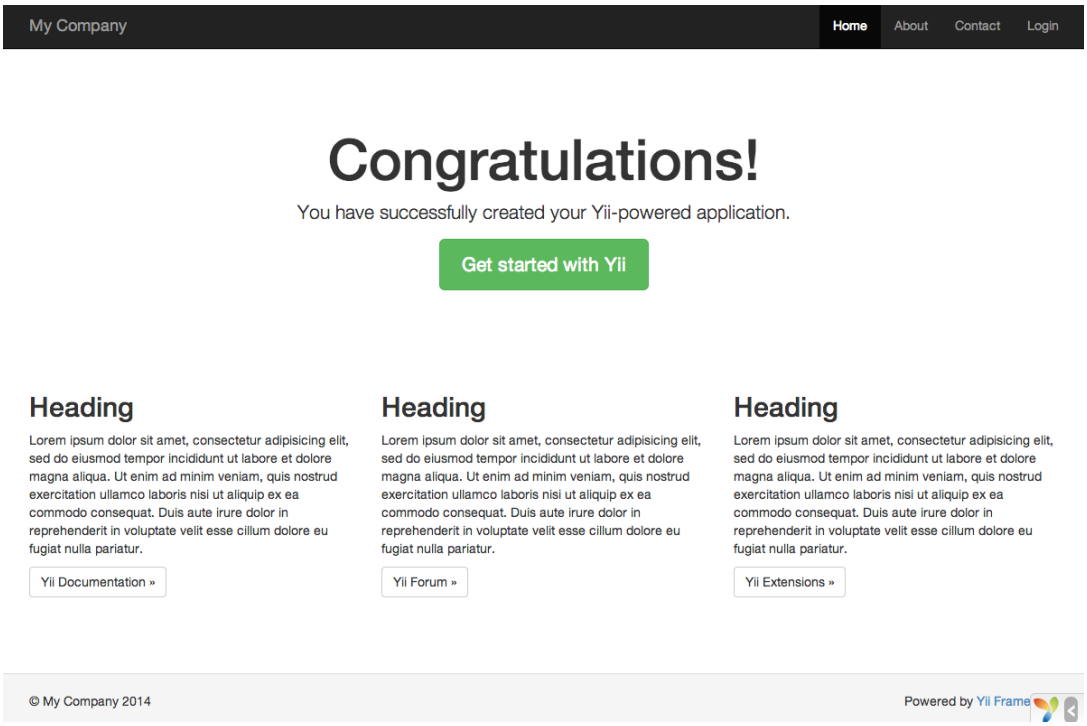
Annexe 5 : menu des graphes de PhenodynD existant



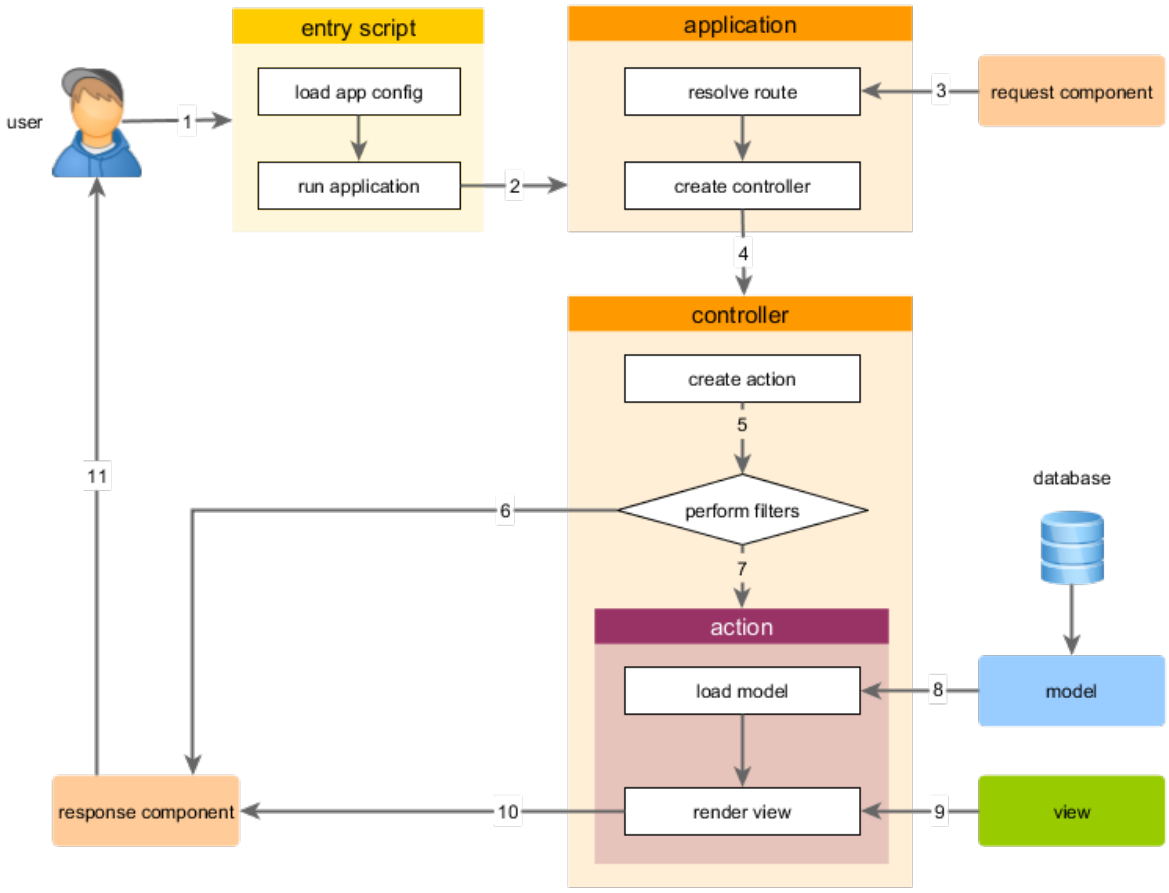
Annexe 6 : graphes, entrée par type de capteur (ici météo)



Annexe 7 : graphes, entrée par manip/pot



Annexe 8 : affichage initial d'un projet Yii2

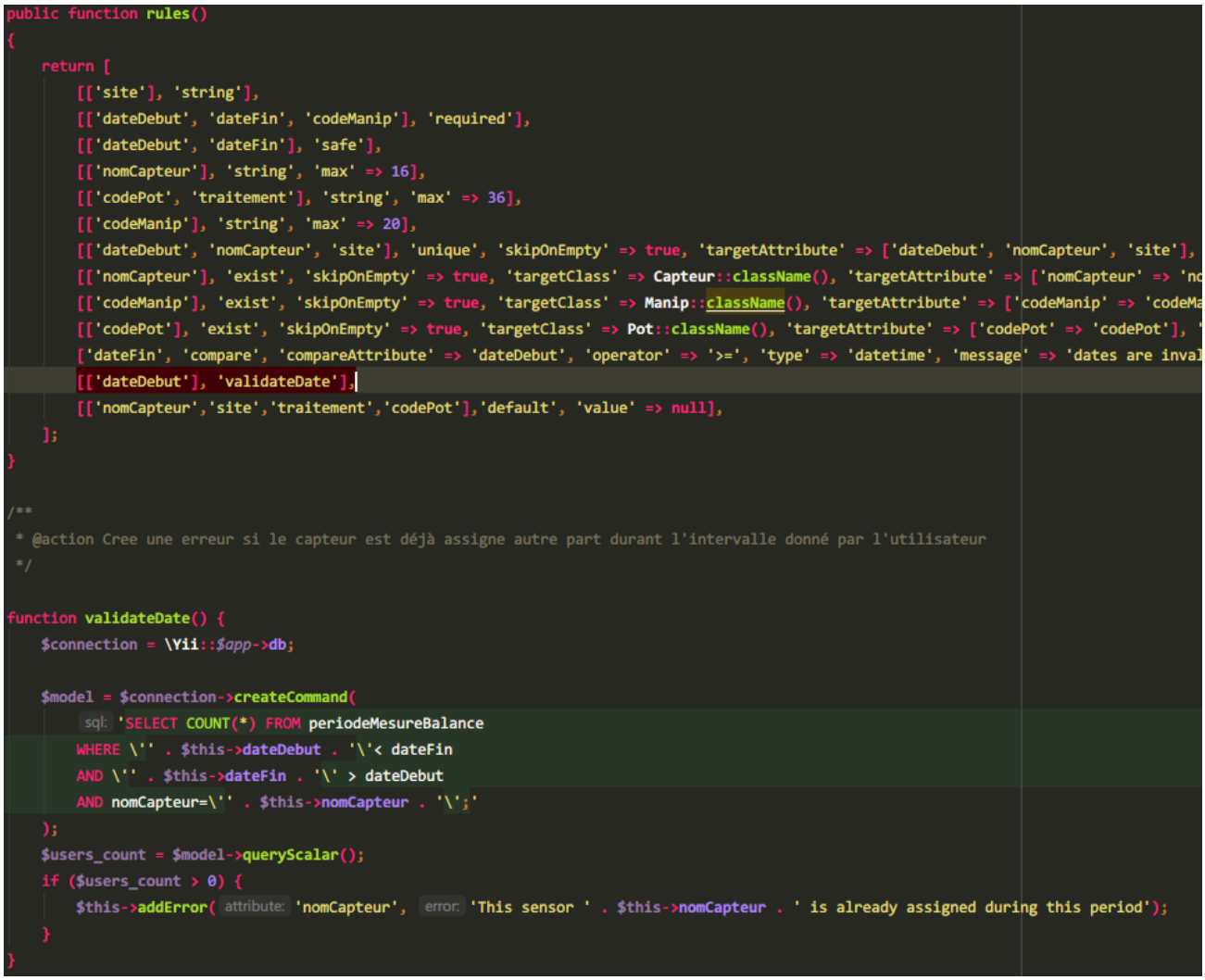


Annexe 9 : diagramme d'activité d'une requête soumise au framework

```php
<div class="capteur-form">

    <div class="col-md-10 col-md-offset-1">

        <?php $form = ActiveForm::begin(['layout' => 'horizontal']); ?>

        <?= $form->field($model, attribute: 'nomCapteur')->textInput(['maxlength' => true]) ?>

        <?= $form->field($model, attribute: 'site')->dropDownList(
                [
                    'b' => 'b', 's1' => 's1', 's0' => 's0',
                    'p' => 'p', 'cA' => 'cA', 'cE' => 'cE', 'b1' => 'b1', 'b2' => 'b2', 'b3' => 'b3', 'b4' => 'b4',
                    'b5' => 'b5', 'b6' => 'b6',
                ],
                ['prompt' => '']
        ) ?>

        <?= $form->field($model, attribute: 'type')->dropDownList(['m' => 'm', 'b' => 'b', 'c' => 'c',], ['prompt' => '']) ?>

        <div class="text-center">
            <?= Html::submitButton( content: $model->isNewRecord ? 'Create' : 'Update', ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
        </div>

        <?php ActiveForm::end(); ?>

    </div>

</div>
```

Annexe 10 : vue capteur/_form.php

On peut voir ici le formulaire qui va permettre d'enregistrer dans la base de nouveaux capteurs. « $model » est fourni à cette vue par le controller qui appelle cette vue. Il existe différents types de champs, comme on peut le voir (« textInput », « dropDownList »). Les différents champs possibles peuvent être retrouvés dans la documentation de Yii2.



```php
public function rules()
{
    return [
        [['site'], 'string'],
        [['dateDebut', 'dateFin', 'codeManip'], 'required'],
        [['dateDebut', 'dateFin'], 'safe'],
        [['nomCapteur'], 'string', 'max' => 16],
        [['codePot', 'traitement'], 'string', 'max' => 36],
        [['codeManip'], 'string', 'max' => 20],
        [['dateDebut', 'nomCapteur', 'site'], 'unique', 'skipOnEmpty' => true, 'targetAttribute' => ['dateDebut', 'nomCapteur', 'site'],
        [['nomCapteur'], 'exist', 'skipOnEmpty' => true, 'targetClass' => Capteur::className(), 'targetAttribute' => ['nomCapteur' => 'no
        [['codeManip'], 'exist', 'skipOnEmpty' => true, 'targetClass' => Manip::className(), 'targetAttribute' => ['codeManip' => 'codeMa
        [['codePot'], 'exist', 'skipOnEmpty' => true, 'targetClass' => Pot::className(), 'targetAttribute' => ['codePot' => 'codePot'], '
        ['dateFin', 'compare', 'compareAttribute' => 'dateDebut', 'operator' => '>=', 'type' => 'datetime', 'message' => 'dates are inval
        [['dateDebut'], 'validateDate'],
        [['nomCapteur','site','traitement','codePot'],'default', 'value' => null],
    ];
}

/**
 * @action Cree une erreur si le capteur est déjà assigne autre part durant l'intervalle donné par l'utilisateur
 */

function validateDate() {
    $connection = \Yii::$app->db;

    $model = $connection->createCommand(
        sql: 'SELECT COUNT(*) FROM periodeMesureBalance
        WHERE \'' . $this->dateDebut . '\'< dateFin
        AND \'' . $this->dateFin . '\' > dateDebut
        AND nomCapteur=\'' . $this->nomCapteur . '\';'
    );
    $users_count = $model->queryScalar();
    if ($users_count > 0) {
        $this->addError( attribute: 'nomCapteur', error: 'This sensor ' . $this->nomCapteur . ' is already assigned during this period');
    }
}
```

Annexe 11 : règle de validation validateDate pour le modèle de la table periodeMesureBalance

```php
    <!-- ['import/import'] correspond à l'élément r dans l'url : plus généralement [controller/action] -->
    <?= Html::beginForm(['import/import'], method: 'post', ['enctype' => 'multipart/form-data']) ?>

    <?php

    //si l'accès a cette page a été fait grâce à un bouton, dans la vue d'une certaine table,
    // cette table sera séléctionnée directement dans le DropDown
    //grâce à la variable typeImport passée dans l'url ($_GET), qui sera fournie en deuxième argument du dropdown

    $typeImport = (isset($_GET['typeImport'])) ? $_GET['typeImport'] : null;

    echo Html::dropDownList(
        name: 'typeImport',
        $typeImport,
        [
            'genotype' => Yii::t( category: 'app', message: 'Genotype'),
            'pot' => Yii::t( category: 'app', message: 'Pot'),
            'plante' => Yii::t( category: 'app', message: 'Plant'),
            'periodeMesureCroissance' => Yii::t( category: 'app', message: 'Period Measure Growth'),
            'periodeMesureThermocouple' => Yii::t( category: 'app', message: 'Period Measure Thermocouple'),
            'periodeMesureBalance' => Yii::t( category: 'app', message: 'Period Measure Balance'),
            'phenologie' => Yii::t( category: 'app', message: 'Phenology'),
            'events' => Yii::t( category: 'app', message: 'Events'),
            'mesureBalance' => Yii::t( category: 'app', message: 'Measure Balance'),
            'calibration' => Yii::t( category: 'app', message: 'Calibration'),
            'capteur' => Yii::t( category: 'app', message: 'Sensor'),
            'substrat' => Yii::t( category: 'app', message: 'Substrate')
        ],
        ['id' => 'dropDown']
    );
    ?>

</div>
<div class="row petitEspacementHaut">
    <!-- La ligne suivante est celle affichant le champ input pour uploader un fichier -->
    <?= Html::fileInput( name: 'importedFile') ?>
</div>
<div class="row petitEspacementHaut">
    <?= Html::submitButton( content: 'Import', ['class' => 'btn btn-success']) ?>

    <?= Html::endForm() ?>
```

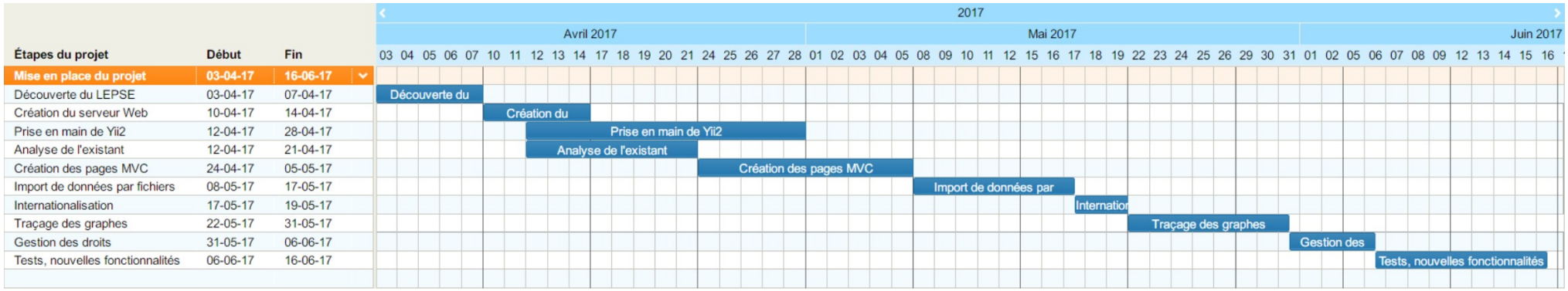
Annexe 12 : formulaire d'import pour le fichier de déclaration de données utilisant la classe HTML pour générer des éléments comme des « select » ou des « textInput »

Annexe 13 : menu graphe balance pour les chercheurs

```php
public function actionBalanceSeries(){
    if ((!isset($_POST['nomCapteurs'])) || (!isset($_POST['dateDebut'])) || (!isset($_POST['dateFin']))) {
        return Json::encode( value: "");
    } else {
        $dateDebut = $_POST['dateDebut'];
        $dateFin = $_POST['dateFin'];
        $nomCapteurs = json_decode(stripslashes($_POST['nomCapteurs']));
        $queries = []; // va contenir le résultat de chaque requête
        $series = []; // va contenir les series de données

        foreach ($nomCapteurs as $nomCapteur) {
            $nomCapteur = '\'' . $nomCapteur . '\'';
            $query = '
                    SELECT valeur,date
                    FROM MesureBalance WHERE date BETWEEN \'' . $dateDebut . '\' AND \'' . $dateFin . '\' AND nomCapteur IN (' . $nomCapteur . ') ORDER BY date ASC';
            $command = Yii::$app->db->createCommand($query);
            $queries[] = $command->queryAll();
        }

        //creation du tableau series :[{"name"="nomCapteur","data"=[[x,y],[x,y]]},{"name"="nomCapteur","data"=[[x,y],[x,y]]}]
        foreach ($queries as $queryNo => $query) {
            foreach ($query as $row) {
                $timestamp = strtotime($row['date']) * 1000; //HighCharts attends des timestampUnix*1000
                $valeur = (floatval($row['valeur']) != 'null') ? floatval($row['valeur']) : null;
                $series[$queryNo]['data'][] = [$timestamp, $valeur];
            }
            $series[$queryNo]['name'] = $nomCapteurs[$queryNo];
        }

        if (!($series === null || $series === "")) {
            return Json::encode($series, options: JSON_NUMERIC_CHECK);
        } else {
            return Json::encode( value: "");
        }
    }
}
```

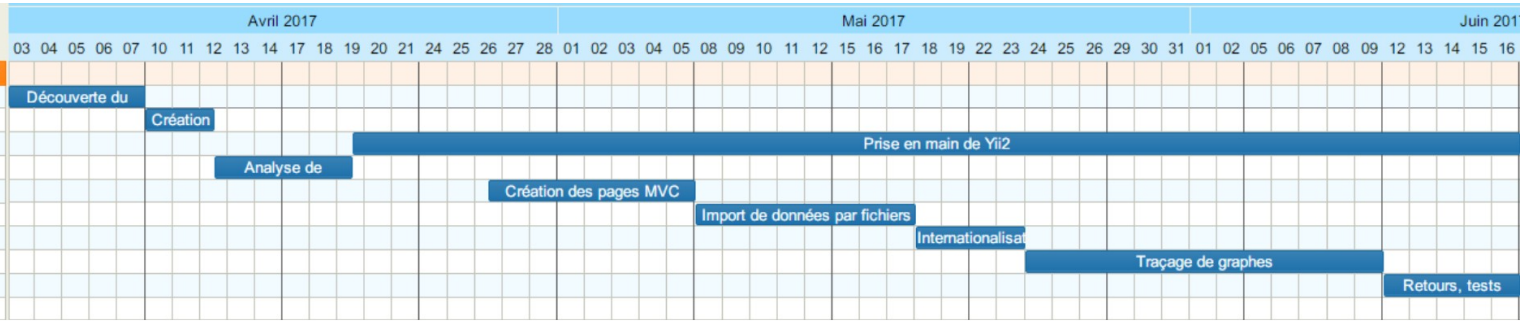
Annexe 14 : méthode actionBalanceSeries, qui récupère les données nécessaires au traçage d'un graphe et les formate selon les attentes de HighChartsJS

Annexe 15 : diagramme de Gant Prévisionnel



Annexe 16 : diagramme de Gantt final

Étapes du projet	Début	Fin
Mise en place du projet	**03-04-17**	**16-06-17**
Découverte du LEPSE	03-04-17	07-04-17
Création du serveur Web	10-04-17	12-04-17
Prise en main de Yii2	19-04-17	16-06-17
Analyse de l'existant et cahier des charges	12-04-17	19-04-17
Création des pages MVC	26-04-17	05-05-17
Import de données par fichiers	08-05-17	17-05-17
Internationalisation	18-05-17	23-05-17
Traçage de graphes	24-05-17	09-06-17
Retours, tests, nouvelles fonctionnalités	12-06-17	16-06-17



Conclusion