David BOZON

Solène DEMARS

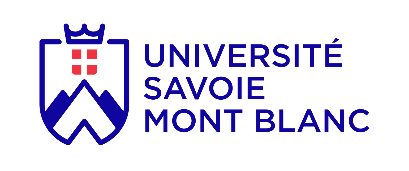
Gaétan DEPREZ

Émilie PARDOEN

Jérémy VALETTE

One Bee

Projet L3 informatique - Rapport écrit



# Sommaire

Table des matières

[Sommaire 2](#_Toc482779520)

[Présentation du projet 3](#_Toc482779521)

[Schéma fonctionnel 3](#_Toc482779522)

[Explication technique des choix 3](#_Toc482779523)

[Bootstrap 3](#_Toc482779524)

[Graphiques 3](#_Toc482779525)

[AJAX 6](#_Toc482779526)

[MySQL / SQlite 6](#_Toc482779527)

[Motion 7](#_Toc482779528)

[Connexion à la base de données 7](#_Toc482779529)

[Organigramme du code 7](#_Toc482779530)

# Présentation du projet

Dans le cadre du projet One Bee, notre équipe d’étudiants de L3 informatique est chargée de mettre en place une **interface web** pour faire le lien entre le système, installé dans la ruche, et l’utilisateur. On veut notamment pouvoir visualiser :

* les données des entrées-sorties, sous forme de graphiques
* le flux vidéo issu de la webcam, en direct

# Schéma fonctionnel

# Explication technique des choix

**Les différents fichiers qui composent le projet sont commentés** : il y a des explications tout au long des fichiers pour en simplifier la compréhension. En plus de ces commentaires, voici quelques explications.

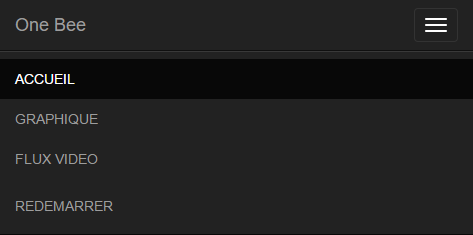
## Bootstrap

Afin de **faciliter la mise en page** d’un site web, on peut utiliser le framwework Bootstrap (<http://getbootstrap.com/>), qui nous fournit, entre autre, une feuille de style. Il ne faut pas oublier de récupérer les fichiers, et de les inclure dans les entêtes des fichiers correspondants aux pages du site.

Pour utiliser les règles de style fournies par Bootstrap, il faut rajouter les **classes** qui nous intéressent dans les balises. Par exemple, pour la barre de navigation, on a « <nav class="navbar navbar-inverse navbar-fixed-top"> », ainsi il n’y a plus besoin de rajouter d’autres règles à la main pour le menu.

Grâce au JavaScript fournit, quand la largeur de l’écran est trop petite, **l’apparence du menu** est modifiée : il faut appuyer sur le bouton en haut à droite pour afficher les boutons.

Menu d’origine :

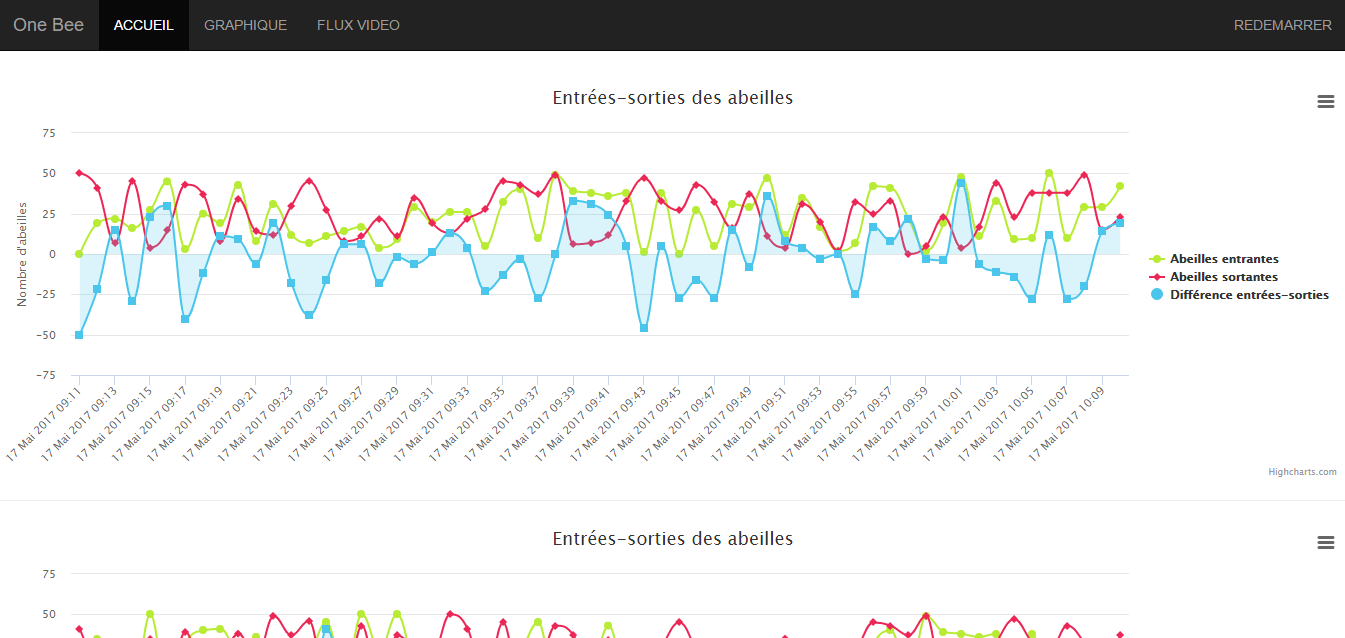


Menu avec un petit écran, développé :

C’est le bouton « <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar"> » qui le permet.

## Graphiques

Pour l’affichage des données, de nombreuses solutions existent déjà sous forme de librairies JavaScript. Dans notre cas, l’utilisateur doit pouvoir visualiser le nombre d’abeilles en fonction du temps, il peut choisir quelles séries afficher sur le graphique, changer la période de temps affichée ainsi que la granularité d’affichage (minutes, heures, etc.). Au passage de la souris, la valeur du point doit apparaître et l’utilisateur doit pouvoir exporter le fichier en tant qu’image. La **librairie Highcharts** (<https://www.highcharts.com/>) nous permet également de zoomer sur le graphique, et de l’exporter sous différents formats.



Plusieurs fichiers sont sollicités :

* graphe.php
  + la page affichée dans le navigateur
  + html, JavaScript / jQuery
* graphe.css
  + pour la mise en page de graphe.php
* graphiques.php
  + fait le lien avec la base de données, pour récupérer les données
  + appelle la fonction pour dessiner un graphique
* graphiques.js
  + c’est là qu’on trouve la fonction pour afficher les graphiques
  + les options du graphique sont à préciser ici

**Détails sur les fichiers :**

* graphe.php

L’entête et le menu sont les mêmes que pour les autres pages, il s’agit de html classique. On trouve des attributs spécifiques à Bootstrap, les explications sont dans la partie correspondante.

Ensuite, on trouve **3 graphiques** :

* + chart1 : affichage des données de la dernière heure, avec une unité de temps par minute, rechargement toutes les 60 secondes
  + chart2 et chart3 : même affichage que chart1 par défaut, mais l’utilisateur peut remplir un formulaire juste en dessous du graphique concerné pour modifier l’intervalle à afficher, et la granularité.

Sur Firefox, le champ pour les dates s’affiche comme un simple champ texte, d’où l’attribut « placeholder » pour afficher le format des dates à taper à la main.

Le script qui suit sera détaillé dans la partie **AJAX**. Il permet de charger les graphiques individuellement, sans recharger la page en entier.

* graphe.css

Le fichier s’occupe de quelques détails de **mise en page** qui ne sont pas pris en charge par Bootstrap. Il est principalement utilisé pour rendre plus esthétique le formulaire sous les graphiques.

* graphiques.php

C’est ici qu’on **récupère les données de la base de données**, entre les dates précisées par l’utilisateur, ou les valeurs par défaut (la dernière heure), avec une unité de temps (granularité) également précisée (valeur par défaut : minute).

Pour des détails sur la **connexion** à la base de données, voir la partie correspondante.

On commence par stocker les **valeurs par défaut**, dans notre fuseau horaire. Ces valeurs sont modifiées si le formulaire associé au graphique est rempli. Les dates sont ensuite converties en chaînes de caractères.

On passe ensuite à la fonction getDonnees(), dans laquelle on fait les **requêtes à la base de données**. On ne veut récupérer que les données comprises dans l’intervalle de temps voulu.

**Les données vont être stockées dans des tableaux**, pour pouvoir être affichées dans le graphique, chaque « case » correspondra à un point dans le graphique. Avec une granularité par minute, il y aura autant de « cases » que de données dans la plage de données sélectionnée, puisqu’on a un enregistrement par minute. Sinon, il va falloir additionner des données ensembles (exemple avec une granularité par heure, toutes les données, stockées par minute, d’une même heure seront additionnées pour obtenir le total d’abeilles dans l’heure).

Les variables de type Datetime en php nous permettent de **manipuler des dates** grâce à des fonctions déjà définies. Le nombre de jours variable selon le mois est déjà géré par exemple.

Quand **on récupère les données de notre requête** une par une, dans le cas où la granularité n’est pas par minute, on commence par récupérer la date du premier enregistrement. Quand la date courante ($currDate) est égale à la date suivante ($nextDate), ou au-delà, on a besoin d’une **nouvelle « case »** dans nos tableaux. C’est le cas pour la toute première donnée récupérée : $index commence à -1, or un tableau php commence à l’index 0, on incrémente $index avant d’ajouter les données du premier enregistrement à la première « case », d’index (-1)+1 = 0.

**On ne veut pas que les dates s’affichent dans le format par défaut** pour php. On change l’ordre des éléments pour qu’il corresponde à l’ordre français, et les deux tableaux $search et $replace nous permet d’écrire en toutes lettres les mois.

Pour **la courbe de la différence** entre les entrées et sorties, les données ne sont pas stockées dans la base de données : on utilise les données qui sont maintenant stockées dans des tableaux pour créer un autre tableau.

Une fois que toutes les données sont disponibles sous forme de tableaux, on va pouvoir dessiner les graphiques. Mais il s’agit d’une fonction JavaScript, qui a besoin de **tableaux JavaScript**. Grâce à la fonction implode(), on transforme les tableaux php en chaîne de caractères, dans le bon format. On rassemble tous les tableaux dans un seul tableau à passer en paramètre pour la fonction JavaScript drawChart(). Le premier tableau correspond aux dates qui seront affichés sur l’axe x, les suivants sont les données pour les différentes courbes.

Si on **rajoute des nouvelles données** dans la base de données, les entrées et sorties des abeilles avec du pollen par exemple, il va falloir afficher ces données dans le graphique. Pour cela, il suffit de reproduire ce qui est déjà fait pour les autres données. Dans le code, en commentaire, on peut trouver des lignes correspondant à ce genre d’ajout.

* graphiques.js

Ce fichier ne contient que la fonction pour **dessiner un graphique**, dans une balise <div></div>. On précise en paramètres quel est le nom de cette balise, le tableau des textes à afficher sur la graduation de l’axe x (a priori les dates), et enfin un tableau de tableau de données. Les différentes options de la librairie utilisée, **Highcharts**, sont assez explicites. Pour les modifier, on peut trouver ce qu’il est possible de faire dans la documentation de la librairie (<https://www.highcharts.com/docs>), ou dans les nombreux exemples disponibles (<https://www.highcharts.com/demo>).

**Les courbes du graphique** sont précisées dans la partie « series ». Il suffit de reproduire les blocs déjà rédigés, et de les adaptés, si on a besoin de nouvelles courbes. Bien sûr, il ne faut pas oublier de fournir des nouveaux tableaux à la fonction, en modifiant le fichier graphiques.php.

## AJAX

Il est dommage de recharger toute la page « Graphique » quand l’utilisateur veut changer les données du graphe. C’est pourquoi nous avons fait le choix d’utiliser ajax pour **ne réafficher que la zone du graphique,** avec les nouvelles données. Nous avons pu le mettre en place grâce à la librairie JavaScript **jQuery**, et sa fonction load(), qui nous permet de remplacer le contenu d’une balise <div></div> dans la page. Puisque la page n’est pas entièrement rechargée, le formulaire en dessous du graphique n’est pas remis à zéro, ce qui rend l’expérience utilisateur beaucoup plus agréable.

Les fonctions sont situées dans le fichier **graphe.php**, entre les balises <script></script>, sous les graphiques.

Au premier chargement de la page, on charge tous les graphiques. Toutes les 60 secondes, le graphique 1 est rechargé, le reste de la page reste inchangé. Quand l’utilisateur met à jour le graphique 2, ou 3, seul le graphique 2, ou 3, est rechargé.

Dans **loadGraphique()**, on demande à charger, dans la <div></div> associée au graphique, le fichier graphiques.php, et on envoie, en méthode POST, le numéro du graphique et le contenu du formulaire. Pour mieux comprendre le fonctionnement de load(), il existe de nombreux tutoriels en ligne (<https://openclassrooms.com/courses/un-site-web-dynamique-avec-jquery/la-methode-load-1>).

En ce qui concerne le compteur qui est affiché en dessous du flux vidéo nous avons utilisé un script qui permet d’afficher une page complète à l’intérieur d’une div et de réactualiser uniquement cette page sans tous rafraichir. Nous avons donc codé le compteur dans un autre fichier php afin de pouvoir l’inclure dans une div de la page flux.

## MySQL / SQlite

Nous sommes parti sur une base de données MySQL car il y a une interface de gestion intégré à wamp, qui est le serveur apache que l’on utilise, et nous utilisons un champs date particulier qui n’existe pas en SQlite.

## Motion

## Connexion à la base de données

# Organigramme du code

Conclusion : liste fait / reste à faire / erreurs que nous éviterions si nous devions recommencer

fait :

* site web pouvant accueillir les différentes fonctionnalités
  + en page d’accueil, message de bienvenue et logos
* base de données, et un script pour la remplir de données aléatoires
  + champs créés pour rajouter des données pour « abeilles avec pollen », etc.
* graphiques qui récupèrent les données d’une bdd
  + choix des séries à afficher (entrées, sorties, différence)
  + choix de l’intervalle de temps à afficher
  + choix de la granularité des données à afficher
  + formulaire et bouton pour valider les changements
  + affichage des données au passage de la souris
  + export en jpeg/png/pdf/svg
  + zoom sur une zone du graphique, sur l’axe x
  + affichage des dates en x
  + mise à jour du graphique automatiquement toutes les 60 secondes
  + format de la date affiché dans le champ de texte pour Firefox
* gestion du flux vidéo
  + affichage de la vidéo issue de la webcam, en direct
  + affichage du compteur d’abeilles, à partir d’une certaine date
  + format de la date affiché dans le champ de texte pour Firefox
* raspberry

à faire :

* mettre en lien l’interface avec la bdd remplie par le script de l’autre équipe, et la webcam de la ruche

erreurs :

* gestion de projet un peu approximative

**+ tuto installation**

**+ quelqu’un qui reprend le projet, comment il ferait ?**

**+ bien commenter le code, bien expliquer, avec des phrases françaises**