**Document Developpeur IHM**

**Sommaire :**

[1 Introduction 3](#_Toc127531813)

[2 Installer l’IHM 4](#_Toc127531814)

[3 Fonctionnement IHM (UML) 4](#_Toc127531815)

[4 ImportCourbe.php 5](#_Toc127531816)

[4.1 enTeteCSVIntoArray() 6](#_Toc127531817)

[4.2 Fgetcsv() 6](#_Toc127531818)

[4.3 TechAndEsti() 7](#_Toc127531819)

[4.4 CheckAlert() 7](#_Toc127531820)

[4.5 TakeDataFromEntete() 7](#_Toc127531821)

[4.6 addOrRemove() 7](#_Toc127531822)

[4.7 sonModulo() 7](#_Toc127531823)

[4.8 AddCourbeBDD() 8](#_Toc127531824)

[4.9 ArrayToString() 8](#_Toc127531825)

[4.10 Db\_connect() 8](#_Toc127531826)

[5 GenerCourbe.php 9](#_Toc127531827)

[5.1 afficheChoixCourbe() 10](#_Toc127531828)

[5.2 Database.php 11](#_Toc127531829)

[5.3 afficheFilteredTechnoCourbes() 11](#_Toc127531830)

[5.4 afficheEstimateurFiltre() 11](#_Toc127531831)

[5.5 afficheFilteredEstimateurCourbes() 11](#_Toc127531832)

[5.6 Affiche() 12](#_Toc127531833)

[5.7 affichageById() 12](#_Toc127531834)

[5.8 affichageByName() 12](#_Toc127531835)

[5.9 checkSelectedCourbesExist() 13](#_Toc127531836)

[5.10 afficheSelectedCourbesForCH1() 13](#_Toc127531837)

[5.11 afficheSelectedCourbesForCH2() 14](#_Toc127531838)

[5.12 Deselectionne() 14](#_Toc127531839)

[5.13 deleteCookie() 14](#_Toc127531840)

[5.14 XMLedit() 14](#_Toc127531841)

[5.15 singleShot.php 15](#_Toc127531842)

[5.16 alertCodeDefaut() 15](#_Toc127531843)

[6 Create\_Liste.php 16](#_Toc127531844)

[6.1 addCourbeTable() 16](#_Toc127531845)

[6.2 deleteLastCourbe() 17](#_Toc127531846)

[6.3 deleteCourbe() 17](#_Toc127531847)

[6.4 completionAuto() 18](#_Toc127531848)

[6.5 alertCreate() 18](#_Toc127531849)

[6.6 addListe() 19](#_Toc127531850)

[7 Edit\_Liste.php 19](#_Toc127531851)

[7.1 editCyclage.php 20](#_Toc127531852)

[7.2 stringToArray() 20](#_Toc127531853)

[7.3 delCyclage.php 21](#_Toc127531854)

[8 Select\_Liste.php 21](#_Toc127531855)

[8.1 getCookie() 22](#_Toc127531856)

[8.2 restartFromCookie() 22](#_Toc127531857)

[8.3 changeXML() 22](#_Toc127531858)

[8.4 arretDurgence() 23](#_Toc127531859)

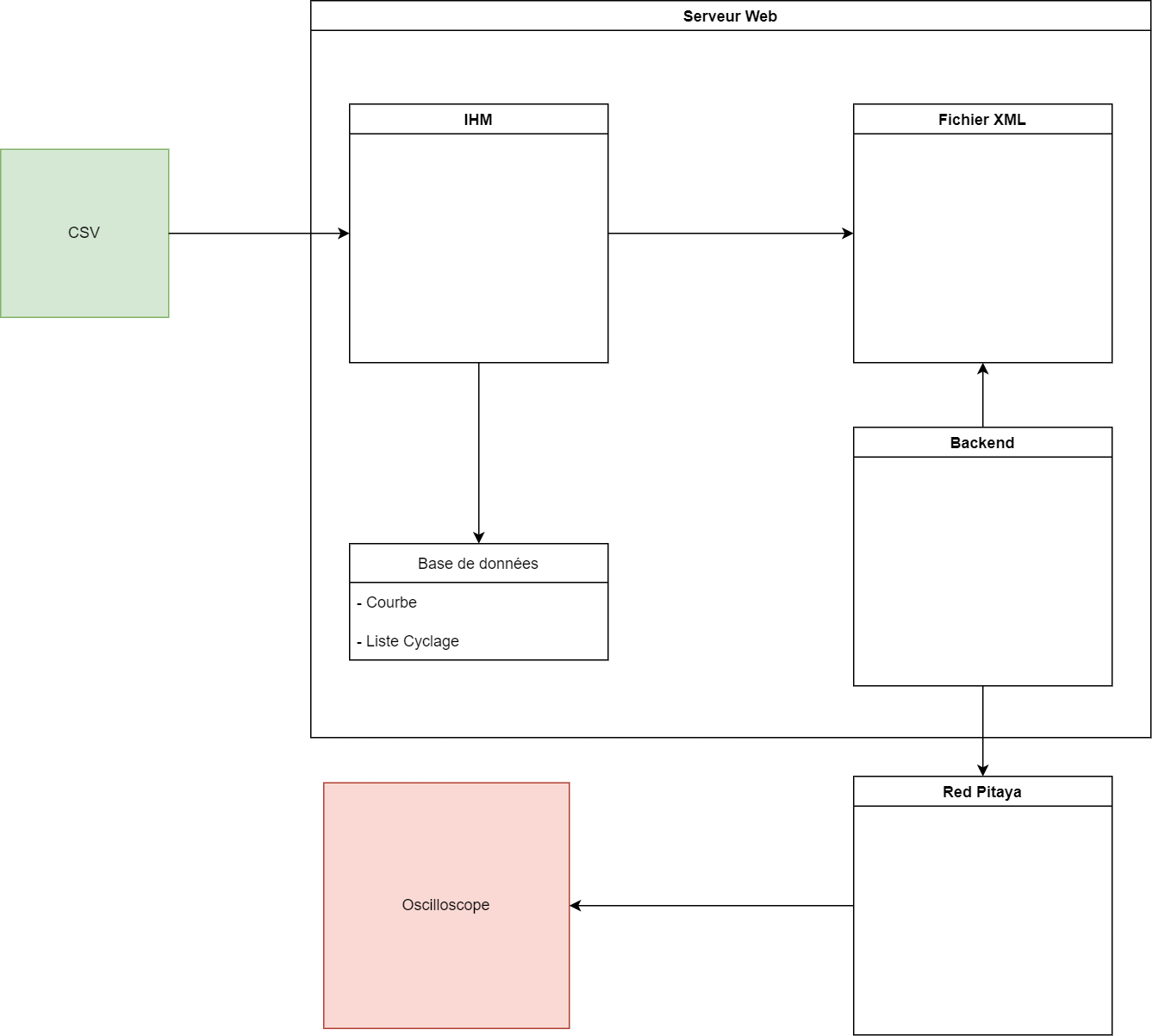
[8.5 alertCodeDefaut() 23](#_Toc127531860)

[9 Autres 23](#_Toc127531861)

# Introduction

Cette IHM permet de simuler des tirs d’un banc de test d’une centrale d’acquisition. Pour ce faire, on devra :  
   
- Importer des courbes venant de fichier CSV et les ajouter à une base de données.  
  
- Simuler des tirs unitaires en sélectionnant des courbes venant d’une base de données.   
  
- Simuler des tirs cyclage de par des listes cyclage créé par l’IHM.

La simulation des tirs sont possible grâce à un ‘RedPitaya’ qui de par des valeurs donné par un backend (qui celui-ci trouvera ses valeurs dans un fichier XML que l’IHM définira) pourra générer un signal.



# Installer l’IHM

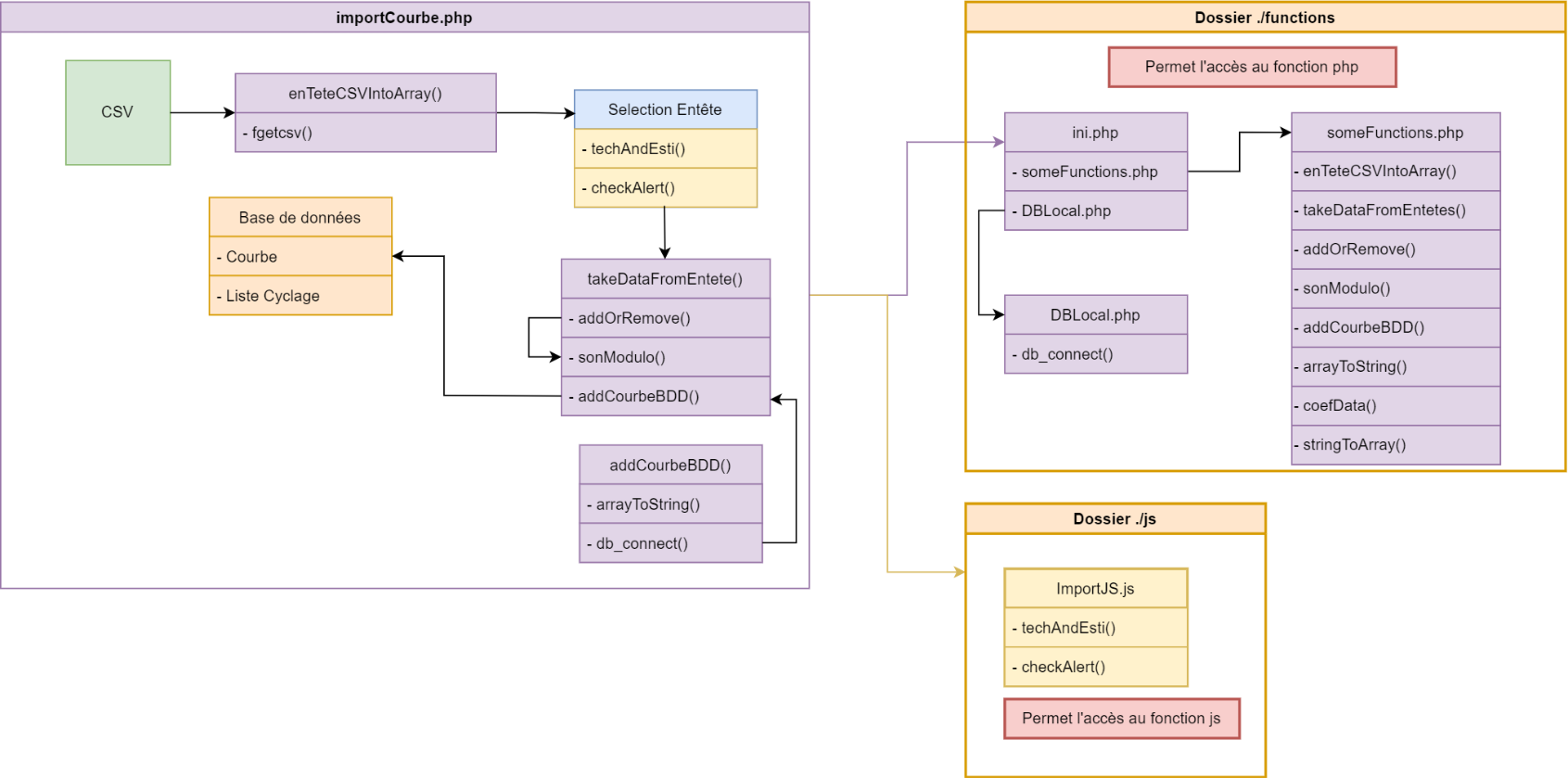
Pour pouvoir utiliser/installer l’IHM, il faut :   
  
- Un serveur ayant le service Apache et Mysql.  
- Démarrer les deux services.  
- Placer le dossier ‘IHM\_Banc’ dans le dossier répertoire d’Apache (par défaut : ‘/var/www/html’).  
- Dans le fichier ‘singleShot.php’ ce trouvant dans le dossier ‘requete\_JS’ il faut modifier la contenance des quotes « ‘ ’ » des lignes 13, 38 et 45 par le lien vers le fichier XML voulu.   
- Dans le fichier ‘checkCyclage.php’ et ‘xmlModifier.php’ ce trouvant dans le dossier ‘CYCLAGE/requete\_JS’ modifié :  
 - La contenance des quotes « ‘ ‘ » du fichier ‘checkCyclage.php’ à  
 ligne 7 par le lien vers le fichier XML voulu.  
 - La contenance des quotes « ‘ ‘ » du fichier ‘xmlModifier’ des lignes 7, 15, 21   
 et 36.  
  
- Importer le fichier ‘BasicDB.sql’ ce trouvant dans le dossier ‘Autres’, dans Mysql.

Une fois ses étapes de complétés, vous pouvez accéder à l’IHM depuis n’importe quelle navigateur.

# Fonctionnement IHM (UML)

L’UML étant un petit peu grande pour Word, l’image se trouve dans le dossier ‘Autres’ du dossier racine de l’IHM. (./IHM\_Banc/Autres)

# ImportCourbe.php



Le but de la page ‘importCourbe.php’ est d’importer des courbes données par fichiers CSV uniquement.

Lorsque la page est chargée pour la première fois, elle commence par inclure un fichier ‘ini.php’ à l'aide de la fonction PHP ‘require\_once’ permettant d’inclure 2 fichiers PHP, l’un pour se connecter à la base de données à l’aide de la fonction ‘db\_connect()’ et le second fichier permet d’inclure les fonctions PHP pour la page ‘importCourbe.php’.   
Ensuite un premier formulaire est affiché permettant à l'utilisateur d’uploader des fichiers CSV.  
Et pour à la toute fin de la page, on inclut le fichier JS ‘ImportJS.js’ pour utiliser les fonctions mis à dispositions dedans.

Lorsque le formulaire est soumis, le script vérifie si le bouton de soumission du formulaire a été cliqué. S'il n'a pas été cliqué, le formulaire est affiché à nouveau. Sinon le script parcourt chaque fichier téléchargé et utilise la fonction PHP ‘enTeteCSVIntoArray()’ pour stocker les données de l'en-tête dans un tableau. Il affiche ensuite un autre formulaire pour permettre à l'utilisateur de choisir les en-têtes souhaités.

## enTeteCSVIntoArray()

* La fonction PHP ‘enTeteCSVIntoArray()’ prend en paramètre un ‘handle’ de fichier CSV et deux tableaux vides. Le premier est ‘$allDataEntete’ (qui stockera toutes les datas de chaque entête pour faciliter les évènements d’après) et le deuxième ‘$enTete’ (qui stockera les entêtes uniquement).   
    
  La fonction lit le fichier CSV ligne par ligne et utilise la fonction PHP ‘fgetcsv()’ pour stocker chaque ligne dans le tableau ‘$data’. Pour chaque ligne, la fonction extrait les entêtes et les stocks dans le tableau ‘$enTete’.

Elle extrait également les valeurs numériques et les stocks dans un autre tableau temporaire ‘$arrayTmp’. Enfin, le tableau temporaire est ajouté au tableau ‘$allDataEntete’ qui contient toutes les données du fichier CSV.  
  
En faisant cela, notre tableau ‘$allDataEntete’ devient un tableau multi-dimensionnel. Le premier index représentera l’entête et le deuxième index représentera les datas de l’entête. Si l’on part du principe que chaque fichier CSV donné commence par une ligne d’entête, alors les données seront stockées selon l’ordre du tableau ‘$enTete’.

## Fgetcsv()

* Permet de récupérer les valeurs étant contenue dans un fichier CSV, pour notre cas, chaque valeur est séparée par une virgule, ainsi chaque index donné sera une valeur.

Ainsi nous avons un tableau ($allDataEntete) regroupant toutes les données numériques classées par fichier CSV donné et un autre tableau ($enTete) pour chaque entête existante classé dans l’ordre par fichier CSV donné.  
  
Une fois les données triées, le script vérifie ensuite si l'en-tête sélectionné est ‘times’. S'il ne l'est pas, il affiche les options pour choisir les technologies et les fréquences souhaitées pour l’entête.   
  
Les options sont affichées que si la checkbox des entêtes est coché de par la fonction JS ‘techAndEsti()’, si oui les inputs sont présentés sous forme de boutons radio (pour les technologies) et de listes déroulantes (pour les estimateurs).   
  
Lors de la soumission du formulaire, la fonction JS ‘checkAlert()’ est exécuté en premier pour vérifier les sélections. Après avoir sélectionné les entêtes voulus, la technologie, l’estimateur et la fréquence, la fonction PHP ‘takeDataFromEntete()’ sera exécuté par nombre de fichier et d’entête donné.

## TechAndEsti()

* Cette fonction JS affiche les options d’une courbe selon l’entête coché, si la checkbox est coché, alors on affiche les options, sinon on les cache.

## CheckAlert()

* Cette fonction JS permet de vérifier que les inputs requis pour l’ajout de la courbe à la base de données soit bien rempli, si des inputs important à l’import sont vides, une alerte JS sera déclenché et notifiera les inputs manquants.

## TakeDataFromEntete()

* Cette fonction PHP extrait les données spécifiques à une entête donnée. Elle prend en paramètre l'ID de l'entête et le tableau ‘$allDataEntete’ qui contient toutes les données du fichier CSV. Elle crée un tableau ‘$valueEnteteArray’ et y stocke les valeurs associées à l'entête donnée. Ce tableau est ensuite envoyé à la fonction PHP ‘addOrRemove()’ pour être modifié et obtenir une courbe à 16384 points.

## addOrRemove()

* Cette fonction PHP modifie le tableau ‘$valueEnteteArray’ en fonction de sa taille.   
    
  Si le nombre d'éléments est supérieur à 16384, la fonction utilise un modulo pour ne conserver que les valeurs qui ne sont pas des multiples de ce modulo.   
  Si le nombre d'éléments est inférieur à 16384, la fonction ajoute des zéros à la fin du tableau pour atteindre une taille de 16384.

## sonModulo()

* Cette fonction PHP calcule le modulo nécessaire pour la fonction PHP ‘addOrRemove()’. Elle prend en paramètre le nombre de points (donc le nombre d'éléments dans le tableau ‘$valueEnteteArray’) et trouve le premier nombre supérieur à ce nombre qui est un multiple de ce nombre.

Une fois les datas venant des entêtes de récupéré, la fonction PHP ‘addCourbeBDD()’ est exécuté, elle permet l’uploade en base de données de la courbe.

## AddCourbeBDD()

* Cette fonction PHP permet d’uploade une courbe en base de données. Cependant, nos datas par entête son sous forme de tableau, nos valeurs seront stockées dans une propriété de la base de données qui ne prends que des strings, ainsi on doit transformer notre tableau en string, la fonction PHP ‘arrayToString()’ s’occupera de ce travail.

## ArrayToString()

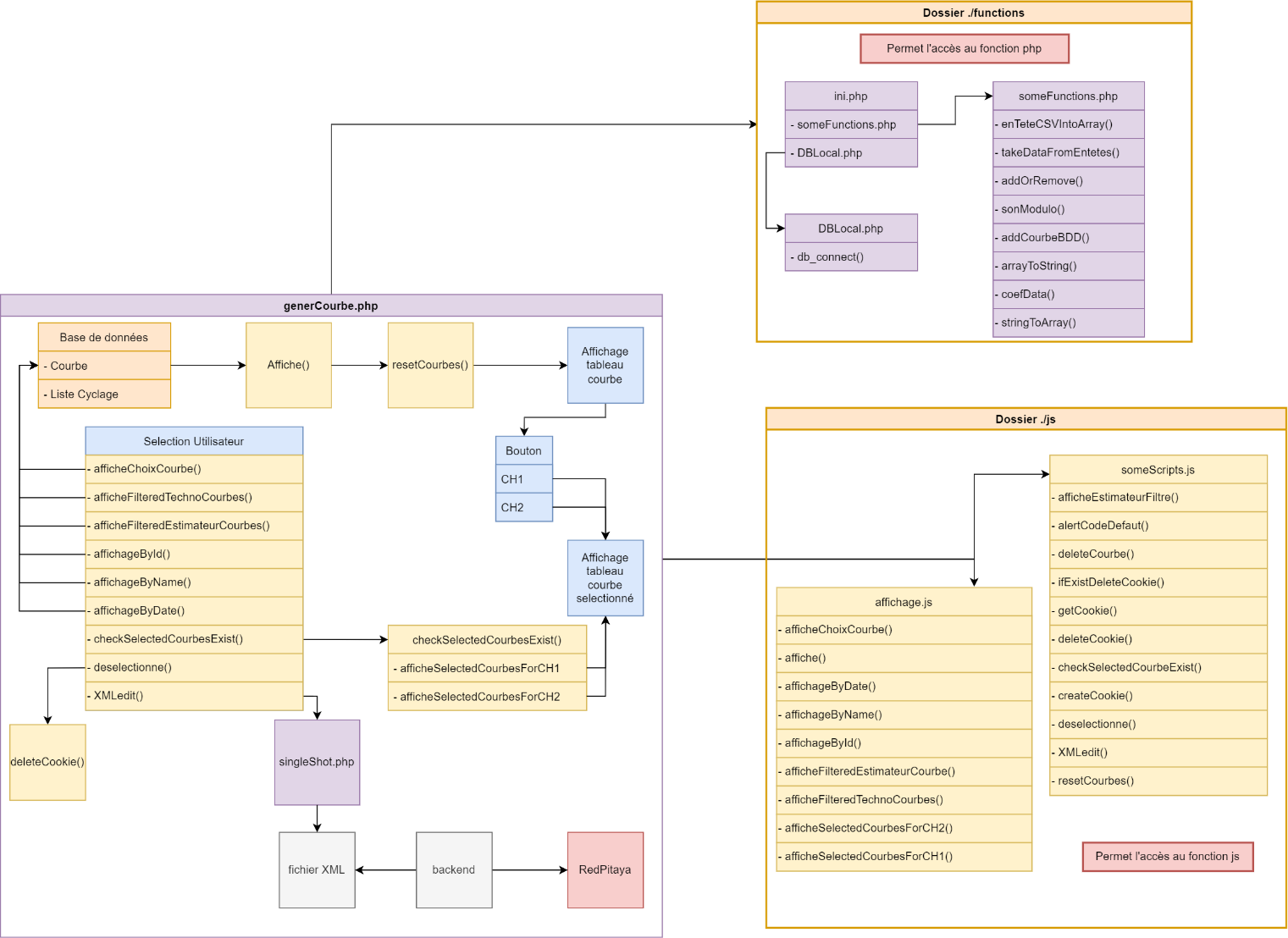
* Cette fonction PHP permet de transformer un tableau en string. La boucle ‘foreach’ permet de faire tourner une boucle selon le nombre de valeur dans le tableau. La condition ‘IF’ permet de créer une variable lors du premier tour. Cette variable sera concaténée avec la valeur donnée par la boucle ‘foreach’ suivit d’une virgule. Une fois la boucle ‘foreach’ terminé, la fonction PHP ‘rtrim()’ permet de retirer le dernier caractère d’un string (Pour notre cas elle sert à supprimer la virgule en trop). Pour finir, on retourne notre string prête à être envoyé en base de données.

Une fois nos valeurs transformés en string, on se connecte à la base de données par la fonction PHP ‘db\_connect()’, on crée notre requête SQL, et on l’envoie à la base de données.

## Db\_connect()

* Cette fonction permet de connecter une variable à la base de données.

# GenerCourbe.php



Le but de la page ‘generCourbe.php’ est de pouvoir sélectionner des courbes existantes en base de données et de pouvoir simuler un tir en modifiant les valeurs des balises venant d’un fichier XML. Ainsi un code backend viendra lire si la balise correspondant à un tir est égal à ‘true’ et lancera une génération de courbe à l’aide d’un Red Pitaya.  
  
On commence par appeler 3 fichiers, l’un et le fichier ‘styles.css’ qui est rangé dans le dossier ‘css’ et les 2 derniers sont ‘someScripts.js’ et ‘affichage.js’ qui sont rangés dans le dossier ‘js’.  
Le fichier ‘style.css’ permet d’avoir un affichage de toute la page sans scroll sur les filtres du tableau contenant les courbes existante.   
Les filtres sont séparés en 3 colonnes (c’est pour cela qu’il y a 3 balises HTML ‘<div>’ ayant comme id ‘colonne1/2/3’).   
  
Le fichier ‘affichage.js’ s’occupera des requêtes PHP pour l’affichage des courbes dans les tableaux avec ou sans filtre.  
  
Le fichier ‘someScripts.js’ s’occupera de garder les fonctions qui serviront d’évènement JS ‘onClick’ dans les balises HTML.

La première balise HTML ‘<div>’ (donc id=‘colonne1’) permet l’affichage des filtres technologies et estimateurs (l’estimateur est caché par un script JS, voir plus bas). Les filtres qui permettent de ranger le tableau par technologie sont des boutons radio, lorsque l’on clique sur ses boutons, une fonction JS dans le fichier ‘affichage.js’ est exécuté.

## afficheChoixCourbe()

* Cette fonction JS affichera sans aucun filtre toutes les courbes venant de la base de données dans un tableau HTML, c’est pour cela qu’elle est exécuté au premier chargement de la page.  
    
  Pour ce faire on utilise l'objet ‘fetch’ pour effectuer une requête PHP à l'adresse 'requete\_JS/dataBase.php' et obtenir les données des courbes.

Il commence par définir la variable ‘posTable’ à 1 pour indiquer la position des en-têtes de colonnes pour le tri. Il appelle ensuite la fonction JS ‘resetCourbes()’ (qui est défini dans le fichier ‘someScripts.js’) qui va vider le tableau (utile si un filtre a été exécuté auparavant).

Ensuite, le code crée un tableau HTML ‘tableauCourbe’ en référençant l'élément HTML avec l'ID 'courbeBDD'. Il crée également une première ligne de tableau ‘tr\_ligne’ et définit son contenu HTML en utilisant ‘innerHTML’. Cette première ligne comprend les en-têtes de colonnes pour le tableau.

Lorsque les données sont reçues avec la réponse ‘response’, elles sont converties en objet JavaScript en utilisant ‘.json()’ et stockées dans la variable courbes.

Enfin, le code utilise une boucle ‘FOR’ pour parcourir le tableau de données courbes et ajouter une ligne pour chaque courbe dans le tableau ‘tableauCourbe’. Chaque ligne comprend des cellules qui contiennent les informations sur la courbe, telles que son ID, son nom, sa description, sa technologie, son estimateur, la date d'ajout, et un bouton ‘Supprimer’. Les boutons ‘CH1’ et ‘CH2’ permettent d'afficher la courbe correspondante dans le tableau de sélection en appelant les fonctions JS ‘afficheSelectedCourbesForCH1()’ et ‘afficheSelectedCourbesForCH2()’ (dans le fichier JS ‘affichage.js’), avec l'ID de la courbe comme argument.

En cas d'erreur lors de la récupération des données, une erreur est affichée dans la console.

## Database.php

* Ce fichier PHP permet d’exécuter une requête SQL selon deux conditions ‘IF’, la première vérifie si l’utilisateur a sélectionné une technologie et un estimateur, si la condition est vrai, elle exécutera une requête SQL qui répondra par toute les courbes selon la technologie et l’estimateur voulu.  
  Sinon la deuxième condition vérifie si une technologie est sélectionné, si la condition est vrai, alors elle répondra avec toute les courbes ayant la technologie voulu.  
  Si les deux premières conditions sont fausses, le fichier PHP exécutera une requête SQL qui récupèrera toute les courbes de la base de données

## afficheFilteredTechnoCourbes()

* Cette fonction JS permet d’afficher dans le tableau, toute les courbes ayant la technologie sélectionné par l’utilisateur.  
  Pour ce faire, elle définit 2 variables, l’un ayant la technologie voulu et l’autre est le chemin vers le bon fichier PHP qui exécutera la requête SQL correspondant au filtre. Ses 2 valeurs sont donné à la fonction JS ‘affiche()’ qui s’occupera d’afficher dans le tableau.

## afficheEstimateurFiltre()

* Cette fonction permet de changer la visibilité du sélecteur pour l’estimateur en fonction de la technologie choisi.

On vérifie si le bouton radio précédent a été sélectionné (oldBool == true) et si sa valeur n'est pas "Aucun" (oldVal != "Aucun"). Si ces deux conditions sont remplies, il cache le sélecteur associé en changeant son style CSS en ‘display: none’.

On définit une variable ‘radioButton’ qui correspond au bouton radio qui a déclenché l'événement.

Si le bouton radio est coché (radioButton.checked == true) et que sa valeur n'est pas "Aucun" (radioButton.value != "Aucun"), le code affiche le sélecteur associé en changeant son style CSS en ‘display: inline-block’. Il définit également les variables ‘oldBool’ et ‘oldVal’ pour se rappeler du bouton radio sélectionné et la valeur associée pour la manipulation vu au début de ses paragraphes. Enfin, il appelle la fonction JS ‘afficheFilteredTechnoCourbes()’ avec la valeur du bouton radio en argument.Haut du formulaireBas du formulaire

## afficheFilteredEstimateurCourbes()

* Cette fonction JS permet l’initialisation des variables pour le filtre ‘estimateur’, il initialise dans la variable ‘ESTIMATEUR’ l’estimateur et dans la variable ‘TECHNO’, la technologie de l’estimateur. Lors de la requête PHP (fetch), ses variables sont envoyées comme argument à la fonction JS ‘affiche()’.

## Affiche()

* Cette fonction fera exactement le même travail que la fonction ‘afficheChoixCourbe()’ cependant il appellera en ‘fetch’ un fichier PHP en fonction de la valeur de la variable ‘path’ qui contient le chemin vers un fichier PHP.  
  Le fichier appelé exécutera une requête SQL fabriqué avec les valeurs des filtres sélectionné. Ainsi elle répondra par des courbes selon le filtre choisi.

La deuxième balise HTML ‘<div>’ (donc id=‘colonne2’) permet l’affichage des filtres ‘ID’, par nom (croissant ou décroissant) et la date d’ajout (récent ou ancien).  
  
Le filtre par ‘ID’ est une balise ‘input’ de type ‘number’ (ce qui permet d’envoyer que des valeurs numérique seulement), cet input est écouté par un script JS qui a chaque changement de valeur, exécute une fonction JS nommé ‘affichageById()’ et affichera une courbe dans le tableau selon l’ID donné.

## affichageById()

* Dans cette fonction JS, la valeur de l’évènement entré comme argument sera testé dans une conditions ‘IF’ pour vérifier que la valeur soit bien numérique, si oui alors on prépare nos deux variables qui auront l’ID entré et le chemin d’accès pour la requête PHP (qui lui fera une requête SQL pour récupérer une courbe par son ID), on appelle la fonction JS ‘affiche()’ avec nos deux variables.  
  Sinon, si la valeur de l’évènement n’est pas un nombre ou si elle est vide, alors on appelle la fonction JS ‘afficheChoixCourbe()’, ce qui affichera toute les courbes.

Le filtre pour trier les courbes par nom croissant / décroissant est un sélecteur qui a deux options, ‘Croissant’ et ‘Décroissant’. Les deux options ont une valeur qui sera utilisé pour créer la requêtes SQL (ASC / DESC).  
Lorsque l’utilisateur sélectionne une option, un évènement JS ‘onClick’ qui appelle une fonction JS nommé ‘affichageByName()’ et affichera les courbes dans un ordre croissant ou décroissant (selon choix utilisateur).

## affichageByName()

* Cette fonction permet un affichage par ordre croissant/décroissant.  
  Quand l’utilisateur clique sur une option donnée, la valeur de cette option sera donnée comme argument dans cette fonction-ci. Ensuite on prépare la variable qui donnera le chemin vers le fichier ‘requete\_JS/byName.php’, ainsi ce fichier PHP exécutera une requête SQL dans la base de données. On prépare nos 2 variables qui seront envoyé à la fonction ‘affiche()’ et affichera les courbes triés dans le tableau.

Le filtre pour trier par date d’ajout fait exactement le même procéder que pour le tri par nom, à l’exception qu’au lieu d’appeler le fichier ‘requete\_JS/byName.php’, on appelle ‘requete\_JS/byDate.php’ et la requête SQL récupèrera les courbes par leur date.  
  
La troisième balise HTML <div> (donc id=’colonne3’) permet l’affichage d’un autre tableau qui affichera les courbes sélectionnés par les boutons ‘CH1’ et ‘CH2’. Un script JS est exécuté et appelle une fonction JS nommé ‘checkSelectedCourbesExist()’.

## checkSelectedCourbesExist()

* Cette fonction JS permet de savoir si des courbes sont stockés dans un cookie, ainsi si l’utilisateur quitte la page ou la recharge, ses courbes sélectionnés ne quitte pas le tableau.  
  Si une courbe est stockée dans le cookie, alors elle s’affiche dans le tableau en fonction de son Canal (ch1 appellera la fonction pour ch1, pareil pour ch2).  
  Si une courbe en CH1 existe, alors on appelle la fonction JS ‘afficheSelectedCourbesForCH1()’.  
  Même procéder pour CH2.

## **afficheSelectedCourbesForCH1()**

* Cette fonction JS permet d’afficher la courbe en CH1 si le cookie ch1 est existant OU si le bouton ‘CH1’ est cliqué.

Il commence par initialiser plusieurs variables qui représentent des éléments HTML pour afficher les informations de la courbe.

La méthode ‘fetch’ est utilisée pour effectuer la requête PHP à "requete\_JS/byId.php" avec les options définies. Si la réponse est réussie, elle est convertie en JSON.

Une boucle "for" est utilisée pour parcourir les courbes récupérées et remplir les éléments HTML correspondants avec les informations de chaque courbe.

La fonction JS "ifExistDeleteCookie()" vérifie si un cookie "ch1" existe et le supprime si oui. Ensuite la fonction JS "createCookie()" est utilisée pour créer un nouveau cookie "ch1" avec l'ID de la courbe sélectionnée.

Enfin, la propriété "disabled" de l'élément HTML avec l'ID "tirButton" est définie sur "false", ce qui permet d'activer le bouton et de permettre de simuler un tir.

En cas d'erreur lors de la récupération des données, un message d'erreur est affiché dans la console.

## **afficheSelectedCourbesForCH2()**

* Même procéder que la fonction ‘afficheSelectedCourbesForCH1()’ mais pour ‘CH2’.

Pour finir, un formulaire qui sera utilisé pour le tableau contenant les courbes sélectionnés. Elle contient un input de type ‘bouton’ qui lors qu’il est cliqué, active l’évènement JS ‘onClick’ et permet d’exécuter la fonction ‘deselectionne()’.

## Deselectionne()

* Cette fonction efface les informations des deux canaux. Il utilise l'objet document pour récupérer les éléments HTML associés à chaque champ du canal. Les champs incluent ‘id’, ‘nom’, ‘description’, ‘technologie’, ‘estimateur’, et ‘timestamp’. Pour chaque canal, le code vide le contenu de chacun de ces champs en utilisant la propriété ‘innerHTML’.

Enfin, on désactive le bouton ‘tirButton’ en mettant sa propriété ‘disabled’ à ‘true’, et supprime les deux cookies en appelant la fonction ‘deleteCookie()’ avec les arguments "ch1" et "ch2".

## deleteCookie()

* Cette fonction permet de supprimer le cookie par le nom donné comme argument.

Ensuite un input de type ‘number’ pour récupérer une valeur numérique qui concernera le bruit du tire (La valeur est limité entre 0 et 20). Une checkbox qui récupérera un booléen pour savoir s’il y a un auto-amorçage. Et pour finir un bouton qui lorsqu’il est cliqué, exécutera un évènement JS ‘onClick’ qui lui exécutera une fonction JS nommé ‘XMLedit()’ qui permet de changer les valeurs d’un fichier XML et ainsi de simuler le tir.

## XMLedit()

* Cette fonction JS permet de faire une requête PHP pour transmettre les données dans le fichier ‘singleShot.php’ et vérifier de part un formulaire si une courbe CH1 est définie, sinon on retourne une réponse vide. Il utilise la méthode ‘fetch()’ pour envoyer les données au serveur et la méthode ‘.then()’ pour gérer la réponse du serveur.

Les variables ‘ch1’, ‘amor’, ‘bruit’, et ‘ch2’ sont des références aux éléments HTML avec les ID respectifs. La valeur d'un champ peut être modifiée si elle est vide ou si elle doit être convertie avant d'être envoyée au serveur.

L'objet ‘tire’ est créé pour stocker les données du formulaire, et les options sont définies pour indiquer que le type de contenu envoyé sera un objet JSON.

La méthode fetch() envoie les données de tire au fichier "requete\_JS/singleShot.php" et permettre de modifier les balises d’un fichier XML pour simuler le tir. Si la réponse du serveur est reçue avec succès, la fonction JS ‘alertCodeDefaut()’ est appelée pour afficher une alerte JS avec le code défaut envoyé par le serveur.

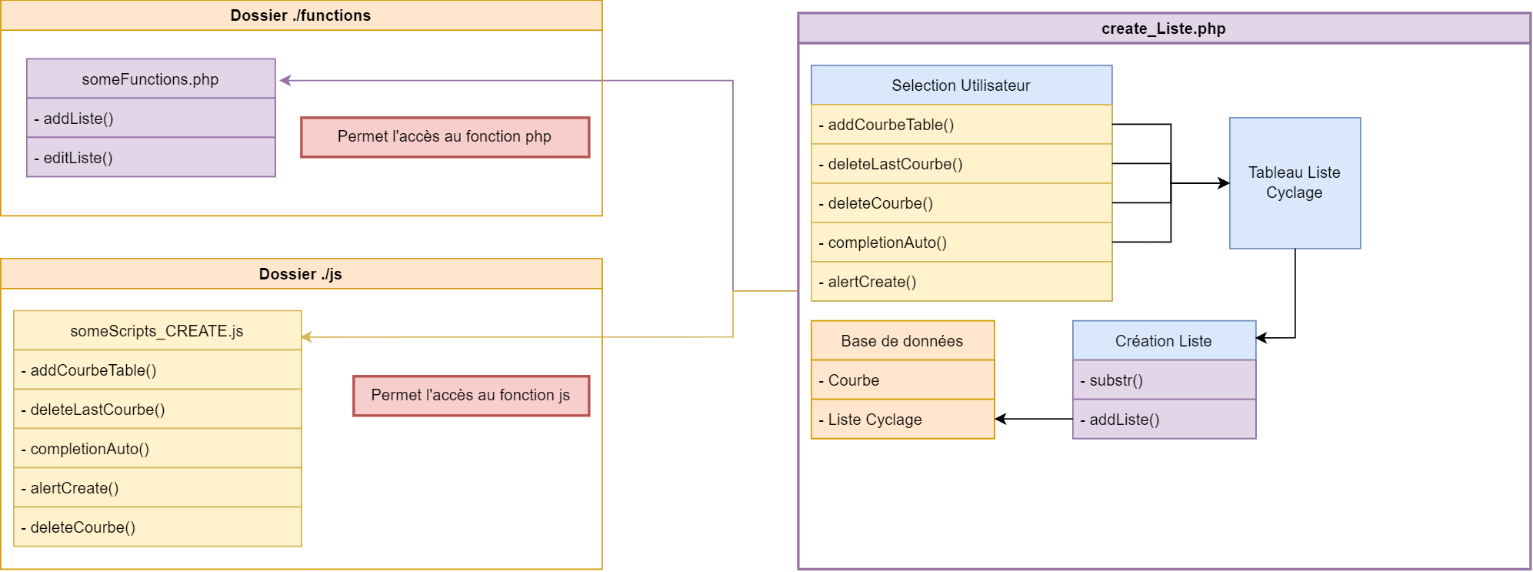
## singleShot.php

- Ce fichier permet la modification du fichier XML pour un tir unitaire.  
 Pour commencer on récupère les valeurs envoyées et on test avec une condition ‘IF’ si une courbe en canal 1 est existante, si la condition est fausse, aucune modification ne sera faite dans le fichier XML. Si la condition est vrai alors on test par une autre condition ‘IF’ si la valeur du bruit n’est pas égal à 0, si la condition est fausse, aucune modification ne sera effectué dans la valeur. Si la condition est vrai, alors on divise la valeur par 100 (pour obtenir une décimal à deux 0).  
  
 On ouvre le fichier XML par la fonction ‘load()’ et on modifie les balises XML :  
 - ‘id\_ch1’ qui recevra l’ID de la courbe du canal 1  
 - ‘id\_ch2’ qui recevra l’ID de la courbe du canal 2  
 - ‘bruit’ qui recevra la valeur du bruit   
 - ‘autoamorcage’ recevra la valeur de l’auto-amorçage  
 - ‘defaut’ recevra la valeur ‘0’  
 - ‘enable’ recevra la valeur ‘true’

## **alertCodeDefaut()**

* Cette fonction permet d’afficher un message d’alerte si le code défaut n’est pas égal à 0 ou 1. Si la condition ‘IF’ est vrai, alors un switch récupère la valeur du code, et affichera le numéro du code et sa définition en message d’alerte.

# Create\_Liste.php



Cette page permet l’ajout en base de données d’une liste de cyclage.  
Elle commence par un formulaire contenant des inputs de type ‘number’ pour récupérer l’ID de la courbe pour le canal 1 et 2 (à noter que la courbe pour le canal 2 n’est pas obligatoire pour un tir de cyclage, ainsi lors de l’ajout en base de données, une valeur par défaut ‘none’ sera donné si la case est vide), le bruit (limité entre 0 et 20), une checkbox pour l’auto-amorçage (si la checkbox est égal à true, la valeur de 1 sera donné dans le tableau), un bouton ‘Ajouter Courbe’ (qui lorsqu’elle est cliqué, un évènement JS ‘onClick’ exécutera une fonction JS nommé ‘addCourbeTable()’) et un bouton ‘Supprimer dernière courbe’ (qui lorsqu’elle est cliqué, un événement JS ‘onClick’ exécutera une fonction JS nommé ‘deleteLastCourbe()’.).

## addCourbeTable()

* Cette fonction permet d’ajouter une courbe dans un tableau avec les valeurs donné par un formulaire.  
  Pour commencer, on récupère la valeur de ‘taille’ qui permettra de tester de par une condition ‘IF’ si notre liste a le nombre voulu par l’utilisateur, si on a moins de courbe dans la liste que de courbe maximum voulu par l’utilisateur, alors la fonction ajoutera la courbe précédemment défini avec le formulaire dans la liste. Sinon elle affiche un message d’erreur nous empêchant d’ajouter la courbe.  
    
  Pour ajouter la courbe dans la liste, on récupère toutes les valeurs données par le formulaire de par l’objet ‘document’, on vérifie si l’auto-amorçage est égal à true pour lui remplacer sa valeur par 1 si vrai, sinon par 0 si faux.

Ainsi, on peut insérée une nouvelle ligne dans le tableau avec les informations suivantes :

* La première cellule contient le numéro(/ position dans le tableau) de la courbe (qui servira pour une fonction JS de supprimer la courbe).
* La deuxième cellule contient un ‘input’ pour CH1 avec la valeur précédemment définis.
* La troisième cellule même procédé que CH1 mais pour CH2.
* La quatrième cellule contient un ‘input’ pour le bruit.
* La cinquième cellule contient un ‘input’ pour l'auto-amorcage.
* La sixième cellule contient un bouton permettant de supprimer la courbe. Lorsqu’il est cliqué, exécute un évènement JS ‘onClick’ qui exécute une fonction JS nommé ‘deleteCourbe()’.

Après l’ajout d’une ligne, on active les boutons ‘Compléter Automatiquement’ et ‘Créer liste’ en modifiant la valeur de l’objet ‘disabled’ en false et on incrémente la variable ‘nbTire’ qui sert à connaitre le nombre de courbe dans le tableau HTML.

## deleteLastCourbe()

* Cette fonction permet de supprimer la dernière courbe du tableau HTML.  
  Pour cela la fonction vérifie par une condition ‘IF’ si le nombre de courbe donné n’est pas égal à 0, si la condition est fausse c’est qu’il y a aucune courbe dans le tableau. Sinon on supprime la dernière courbe grâce à la valeur donnée par ‘nbTire’. Cette valeur sera forcément égale à la dernière position du tableau HTML.

Une fois la courbe supprimée, on décrémente la variable ‘nbTire’ et on test de par une condition ‘IF’ si la variable ‘nbTire’ est égal à 0, si la condition est vrai alors on désactive les boutons ‘Compléter Automatiquement’ et ‘Créer liste’. Si la condition est fausse, alors la fonction s’arrête sans changer les boutons.

## deleteCourbe()

* Cette fonction permet de supprimer une courbe dans une position spécifique du tableau donné par l’argument de cette fonction.  
  Pour commencer on vérifie que la position donnée n’est pas égale à 0 (si la condition est fausse, c’est que la position de la ligne concerne les entêtes et non les courbes), si la condition est vrai, alors on récupère l’objet du tableau pour y supprimer la ligne voulu.

On déclare une nouvelle variable nommée ‘nbTireMax’ ayant la valeur de ‘nbTire’ (qui sera utile pour la boucle qui redonnera la bonne position des courbes dans le tableau HTML une fois une courbe de supprimé), on décrémente la variable ‘nbTire’ et on repositionne les courbes qui suive celle que l’on a supprimé.

Ainsi on a supprimé une courbe dans un tableau HTML dans une position spécifique, on a redonné une position à toute les courbes qui suive la courbe que l’on a supprimé, ce qui permettra de resupprimer une autre courbe si l’utilisateur le souhaite.

Après ce formulaire, nous avons un input ‘number’ qui définira la taille du tableau HTML et un bouton ‘Compléter Automatiquement’ qui est disponible uniquement si une courbe est déjà mise dans le tableau HTML. Si ce bouton est cliqué, un événement JS ‘onClick’ exécute une fonction JS nommé ‘completionAuto()’.

## completionAuto()

* Cette fonction permet de compléter un tableau HTML avec les courbes déjà présentes dans celui-ci. Pour ce faire on vas stocker dans un tableau multidimensionnelle les courbes et les valeurs de chaque courbe. Ainsi notre première index concernera une courbe et notre deuxième index concernera les valeurs de la courbe. Donc si je fais ‘arrayDarray[0][1]’, j’appelle la valeur de ‘CH2’ de la première courbe.  
    
  Notre fonction va récupérer la valeur de la taille donné par l’utilisateur, et complètera le tableau HTML jusqu’à que celui-ci est le même nombre de courbe que la taille donné.

Pour finir, un autre formulaire contenant 4 inputs : ‘text’ pour le nom du cyclage, ‘number’ pour le temps cyclage et 2 boutons ‘submit’ qui permettrons d’envoyer le formulaire pour uploade la liste cyclage en base de données. Lorsqu’un bouton ‘submit’ est cliqué, un évènement JS ‘onClick’ exécutera une fonction JS nommé ‘alertCreate()’ qui vérifiera si le nombre de courbe dans notre tableau est égal à la taille donné par l’utilisateur.

## alertCreate()

* Cette fonction permet de vérifier si le nombre de courbe présent dans le tableau HTML corresponds à la taille donné par l’utilisateur, si non alors le formulaire n’est pas envoyé et un message d’alerte et envoyé.

Une fois notre formulaire d’envoyé, notre page sera rechargé et une condition ‘IF’ est testé pour savoir si notre bouton ‘submit’ est existant. Si la condition est vrai alors de par une boucle ‘FOR’ qui tournera par le nombre de courbe en CH1 existante (qui est obligatoire à avoir pour faire un tir) va transformer nos tableaux en string pour que notre base de donnée puisse récupérer les valeurs.

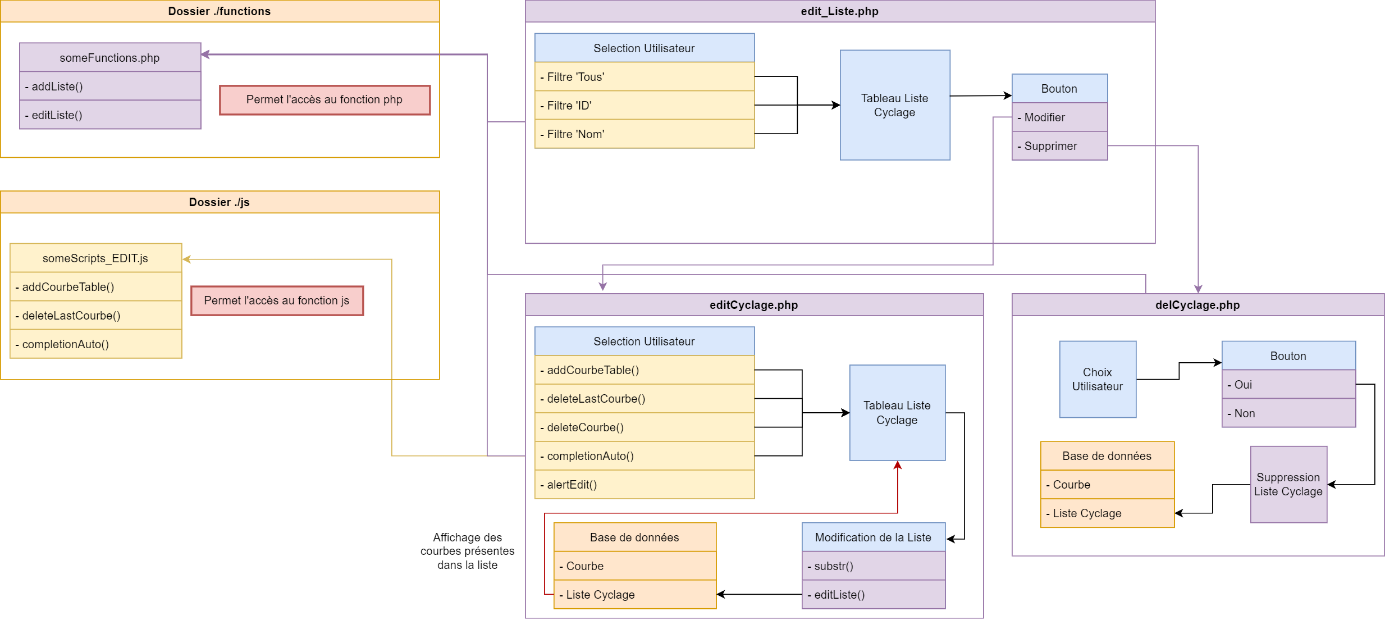
Pour transformer nos tableaux en string, on concatène la valeur donné par le formulaire à une nouvelle variable et elle sera suivi d’une virgule. Cependant si la valeur de CH2 venant du tableau est vide, une valeur par défaut lui sera donné (‘none’). Si la valeur pour le bruit et l’auto-amorçage est vide, une valeur par défaut lui sera aussi attribué (‘0’)

Une fois notre boucle de fini, on supprime le dernier caractère de nos variables (car c’est une virgule inutile) et on exécute la fonction PHP ‘addListe()’

## addListe()

* Cette fonction PHP permet d’uploader une liste cyclage dans une base de données. Les variables donné en argument dans cette fonction contienne les valeurs permettant à l’uploade.  
  Ensuite la fonction crée la requête SQL et l’exécute.

# Edit\_Liste.php



Cette page permet d’accéder à la modification d’une liste de cyclage.

On commence par un formulaire avec un bouton ‘Tous’, qui rechargera la page sans aucun filtre. Ensuite un autre formulaire avec deux champs de saisie : "ID Liste" et "Nom Liste". Ils permettront d’afficher le tableau selon le nom / id donné par l’utilisateur une fois le bouton ‘envoyer’ de cliqué.

Le script PHP commence avec une condition ‘IF’ qui vérifie si le bouton ‘submitIdListe’ a été cliqué. Si la condition est vrai, une série de condition ‘IF’ qui servent à fabriquer la requête SQL, les conditions seront vrai selon le filtre de sélectionné. Si aucun filtre n'a été sélectionné, le script récupérera toutes les informations de la table "LISTE\_CYCLAGE".

Une fois la requête SQL d’envoyé et les listes cyclage de récupéré, on utilise une boucle "foreach" pour afficher les informations dans un tableau.

Pour chaque ligne du tableau, un bouton "Modifier" est fourni qui permet d’accéder à une nouvelle page pour modifier la liste cyclage. De plus, un bouton "Supprimer" est fourni pour supprimer la liste cyclage voulu.

## editCyclage.php

* Cette page est accessible en cliquant sur le bouton modifier dans la page ‘edit\_Liste.php’.  
  Elle permet de modifier une liste cyclage, en changeant les courbes du CH1 et CH2, de modifier le bruit et l’auto-amorçage. Elle permet aussi de changer la taille de la liste.

Cette page agis de la même manière que ‘create\_Liste.php’ à la différence que le tableau aura déjà des courbes d’établis selon la liste cyclage choisi.

On prérempli le tableau grâce au valeur que l’on récupère par l’ID donné par le formulaire de ‘edit\_Liste.php’, dans ce formulaire, il y a une valeur qui représente l’ID de la liste cyclage. Cette ID permettra d’exécuter une requête SQL et de récupérer les valeurs en strings des ID courbes par canaux, bruits et auto-amorçages.

Ensuite chaque variable récupérée sera soumise à une fonction PHP nommé ‘stringToArray()’ qui permet de transformer notre variable string en un tableau.

Une fois que la fonction est fini, on affiche les valeurs dans le tableau HTML.

## stringToArray()

* Cette fonction permet de transformer une string en un tableau.

On commence par déclarer 2 variables, l’une sera un tableau qui sera retourné à la fin de la fonction et l’autre qui servira à stocker les valeurs. On a une boucle qui tournera selon le nombre de caractère dans la string donnée comme argument dans cette fonction. On vérifie dans une condition ‘IF’ si le charactère n’est pas égal à une virgule, si la condition est vrai, alors on la concatène avec la variable string déjà créé avant.

Dès que le charactère testé est une virgule, on push la string contenant une valeur dans notre tableau précédemment créé et on reset notre string.

À la fin de la boucle, on retourne le tableau.

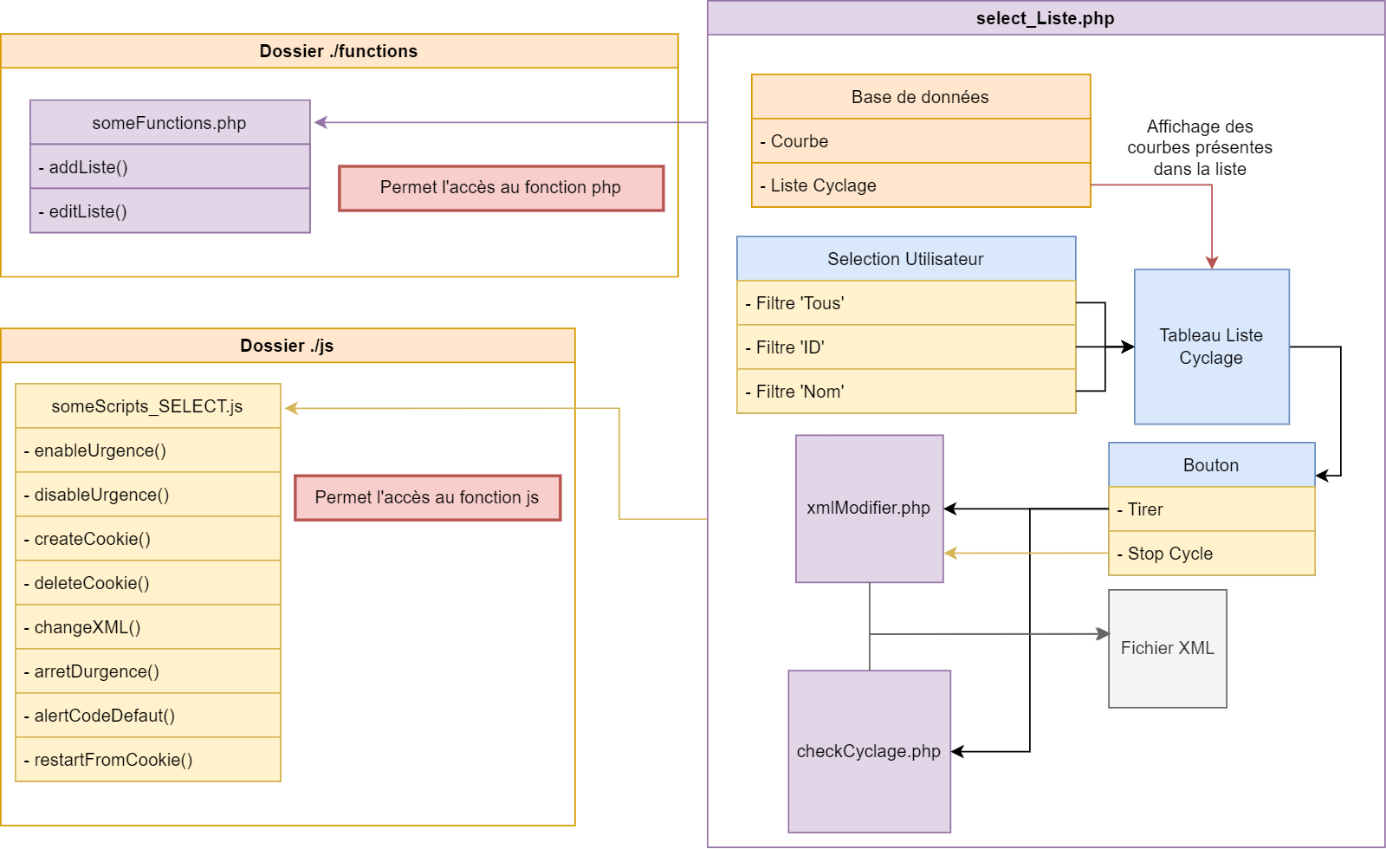
Une fois que l’on a récupéré toutes nos valeurs et quelles sont stockés dans un tableau, on l’affiche dans le tableau HTML. Le reste de la page effectue les mêmes procéder que la page ‘create\_Liste.php’.

## delCyclage.php

* Cette page est accessible quand on clique sur le bouton ‘supprimer’ dans le tableau HTML de la page ‘edit\_Liste.php’. Elle permet de demander à l’utilisateur une confirmation avant la suppression d’une courbe. Lorsque l’on clique sur le bouton ‘supprimer’ on envoie par la méthode ‘POST’ l’ID de la courbe. Dans la confirmation si l’utilisateur a cliqué sur le bouton ‘Oui’ alors on supprime la courbe ayant comme valeur dans ID celle que l’on a donnée par méthode ‘POST’ et on redirige l’utilisateur dans la page ‘edit\_Liste.php’.

Si l’utilisateur a cliqué sur le bouton ‘Non’ on redirige l’utilisateur dans la page ‘edit\_Liste.php’.

# Select\_Liste.php



Cette page permet de simuler un cyclage. On y modifiera des valeurs dans des balises XML comme sur la page ‘singleShot.php’ pour y simuler un tir.

La page commence par un filtre qui fonctionne de la même manière que la page ‘edit\_Liste.php’. Ensuite une fonction JS nommé ‘getCookie()’ récupèrera un cookie nommé ‘isOn’ (ce cookie est créé lorsque l’on clique sur le bouton ‘TIR’). Si ce cookie est existant, le script exécute une fonction nommé ‘restartFromCookie()’.

## getCookie()

* Permet de récupérer un cookie grâce au nom donné comme argument.

## restartFromCookie()

* Cette fonction permet de réafficher la page de manière sécurisé (cette a dire en ayant les boutons permettant de modifier le fichier XML de désactiver) car cela veut dire qu’un cyclage est en cour.

Par défaut, la page affiche dans un tableau HTML, toute les listes cyclage présent en base de données. Chaque ligne a pour dernière colonne un bouton ‘Tirer’ qui quand cliqué, exécute un évènement JS ‘onClick’ qui exécute une fonction JS ‘changeXML()’.

## changeXML()

* Cette fonction JS permet d’appeler deux fichiers PHP, l’un pour modifier un fichier XML et l’autre pour vérifier si le cyclage est terminé ou non.

Pour commencer, on appelle deux fonctions JS, ‘enableSecurity()’ et ‘createCookie()’.

La première fonction permet d’afficher la page PHP ‘select\_Liste.php’ de façon sécurisé. Cette à dire d’enlever la possibilité de faire un nouveau tir cyclage, de chercher une autre liste cyclage et d’activer le bouton ‘Stop cycles’ qui si cliqué, exécutera un évènement JS ‘onClick’ qui exécutera la fonction ‘stopCycle()’.

La deuxième fonction fait le même travail que dans la page PHP ‘generCourbe.php’.

Ensuite on définit une variable ID qui aura pour valeur l’ID de la liste cyclage et on appelle par ‘fetch()’ le fichier PHP ‘requete\_JS/xmlModifier.php’ qui permettra de changer les valeurs des balises XML et ainsi de lancer le cyclage.

Une fois les valeurs des balises XML de changé, on appelle par un autre ‘fetch()’ le fichier PHP ‘requete\_JS/checkCyclage.php’ qui permettra de par une boucle ‘while’ de savoir si notre cyclage est fini ou non en vérifiant la valeur de la balise XML ‘enable’, si la valeur est false, c’est qu’aucun cyclage est en cour, donc que le cyclage est fini.  
Par contre si la valeur est true, c’est que le cyclage est toujours en cour.

Une fois le cyclage fini, la réponse du ‘fetch’ est défini par la fonction ‘.then()’, la valeur de la réponse ira dans la variable courbes, le script exécutera trois fonction après avoir reçu la réponse.

La première est ‘disableSecurity()’, elle permet de réinitialiser l’affichage de la page PHP ‘select\_Liste.php’ comme par défaut.

La deuxième est ‘deleteCookie()’, elle permet de supprimer le cookie qui donnait l’information si un cycle était en cour ou non.

Et la troisième est ‘alertCodeDefaut()’, qui a pour argument la valeur du code défaut, si le code défaut n’est pas égal à 0 ou à 1, on affiche une alerte avec le code défaut reçu et sa définition.

## stopCycle()

* Cette fonction permet d’arrêter un cyclage en cour. Elle appelle deux fonctions JS, la première est ‘disableSecurity()’ qui permet de désactiver la possibilité de cliqué sur le bouton ‘stop cycle’ et la deuxième est ‘deleteCookie’ qui permet de supprimer le cookie créé lorsque l’utilisateur clique sur le bouton ‘Tirer’.

Ensuite notre fonction va donner la valeur true dans une variable nommée ‘check’, ainsi lors que le ‘fetch’ exécutera la requête PHP dans le fichier ‘requete\_JS/xmlModifier.php’, on passera dans une condition ‘IF’ qui modifiera la valeur de la balise XML ‘stop’ à true, le backend lira cette balise et arrêtera le cyclage.

## alertCodeDefaut()

* Cette fonction JS permet de par une condition ‘IF’ et un switch, d’afficher un code défaut avec sa définition depuis une alerte JS.

La condition ‘IF’ permet de tester la valeur donné en argument dans cette fonction, si la valeur est égal à 0 ou à 1, rien ne sera affiché.  
Si inversement, on affiche une alerte du code défaut, avec sa définition.

# Autres