

LE 08 JUIN 2023

RAPPORT DE STAGE

DÉVELOPPEMENT D'UN PLUGIN MOODLE



PERSONNE À CONTACTER

Jérémie
PILETTE

E-MAIL :

pilette@ifrass.fr

NOM DE L'ÉCOLE :

IFRASS

ADRESSE :

2bis Rue Emile Pelletier
31100 Toulouse

RÉDACTEUR :

Esteban BIRET-TOSCANO

A DESTINATION DE :

Mme. CANUT,
professeure
référente

M. PILETTE,
maitre de stage



Sommaire :

Introduction :	3
I- Présentation de l'organisme d'accueil :	4
II- Analyse des besoins et des moyens :	5
A) Analyse de la demande	5
B) Définition des contraintes et des objectifs	8
III- Réalisation technique :	10
A) Organisation & gestion de projet	10
B) Outils utilisés	11
C) Déroulement	12
D) Retour d'expérience & avancement	25
IV- Résultats :	26
V- Bilans :	30
A) Technique	30
B) Humain	30
C) Personnel	31
Conclusion :	32
Table des illustrations :	33
Lexique :	34
Bibliographie / Sitographie :	35
Table des annexes :	36
Annexes :	37
Remerciements :	39
Résumé :	40

Introduction :

Le monde de l'éducation est en constante évolution, et les outils numériques sont de plus en plus utilisés pour faciliter l'apprentissage et la gestion des travaux. Dans ce contexte, j'ai eu l'opportunité de réaliser mon stage de 2ème année de BUT Informatique en développant un plugin¹ Moodle 4 destiné à faciliter la gestion des rendus de travaux des étudiants pour les formateurs.

"Comment développer un plugin Moodle 4 efficace pour faciliter la gestion des rendus de travaux pour les professeurs, en prenant en compte les besoins et les contraintes de l'enseignement numérique, et en utilisant les outils de développement les plus adaptés ?". C'est ce à quoi nous répondrons dans ce rapport.

Ce plugin permettra donc aux formateurs et aux administrateurs de l'organisme de contrôler plus efficacement et facilement les travaux de leurs élèves, tout en offrant une meilleure expérience d'apprentissage. L'enjeu de ce projet était donc de répondre à un besoin croissant en matière de gestion des rendus de travaux et d'offrir une solution fiable, intuitive et efficace.

Nous commencerons par présenter l'organisme d'accueil, pour poursuivre sur l'analyse des besoins et des moyens nécessaires à la réalisation de ce projet. Nous décrirons ensuite les étapes de développement et les outils utilisés pour la réalisation technique de ce plugin. Enfin, nous présenterons les résultats obtenus et dresserons un bilan de cette expérience de stage.

¹ Plugin : logiciel conçu pour être greffé à un autre logiciel à travers une interface prévue à cet effet, et apporter à ce dernier de nouvelles fonctionnalités.

I- Présentation de l'organisme d'accueil :

L'organisme d'accueil est l'IFRASS (Institut de **F**ormation, **R**echerche, **A**nimation, **S**anitaire et **S**ocial), une école située à Toulouse qui propose plusieurs diplômes dans la santé, l'éducatif et le social (cf. Annexe 1), aussi bien en formation initiale, à distance ou continue qu'en alternance.

« L'IFRASS s'inscrit dans le courant de l'émancipation qui construit du pouvoir d'agir, pour et avec les autres, dans des institutions médico-sociales justes et solidaires. »

L'école est constituée de deux pôles et d'un département, nous agissons dans ce stage au niveau du département ressources, et plus précisément FOAD/Enseignements Numérique/MOOCs, dirigé par Jérémie PILETTE, notre tuteur professionnel.

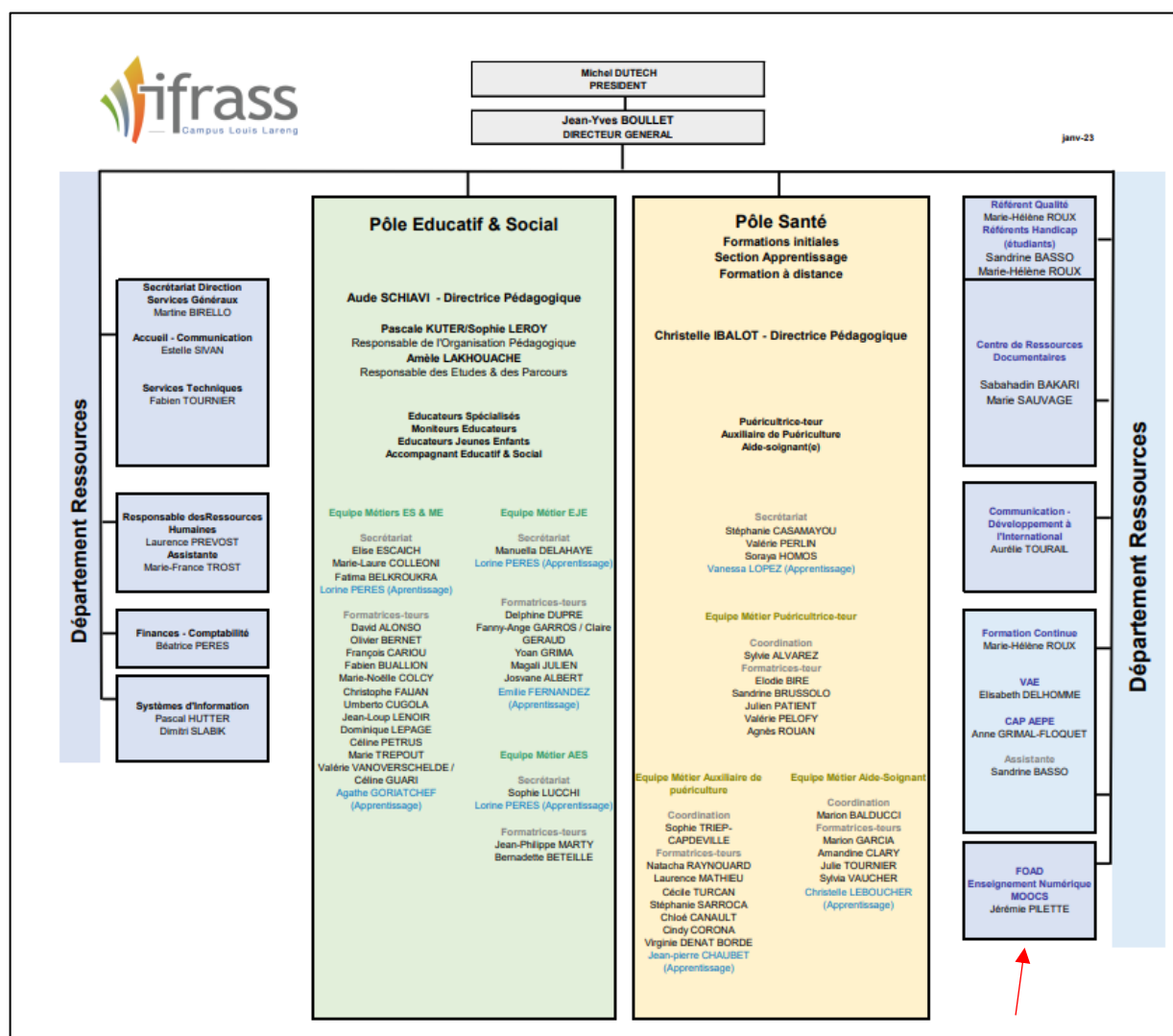


Figure 1 : Organigramme de l'IFRASS.

II- Analyse des besoins et des moyens :

A) Analyse de la demande :

Actuellement, pour suivre le travail des étudiants, nous devons aller dans un cours, puis cliquer sur un lien permettant d'afficher pour ce cours, les étudiants ayant effectué le travail ou non. Nous voyons bien que ceci est **laborieux**, car ces étapes doivent être répétées autant de fois qu'il y a de cours au sein d'une formation donnée. Notre mission était donc d'ajouter un point d'entrée à la racine de Moodle, pour avoir une vision globale du travail des étudiants dans **tous les cours à la fois**, sans avoir besoin de passer manuellement dans chaque cours.



Figure 2 : Item du menu pour accéder au plugin.

N'ayant pas d'existant, nous avons donc dû coder **l'entièreté** du plugin, que ce soit au niveau front-end (affichage des tableaux, formulaires et données) ou du back-end (gestion des données, traitement des informations, export). Néanmoins, la présence d'autres plugins sur le serveur ou même sur Internet nous ont permis de mieux comprendre la structure générale d'un plugin et de s'appuyer sur des exemples.

Une fois le plugin achevé, il pourra être utilisé par 2 types de personnes différentes, des **formateurs** et **administrateurs**.

Voici le diagramme des cas d'utilisation de ce que sera capable de faire notre plugin une fois terminé :

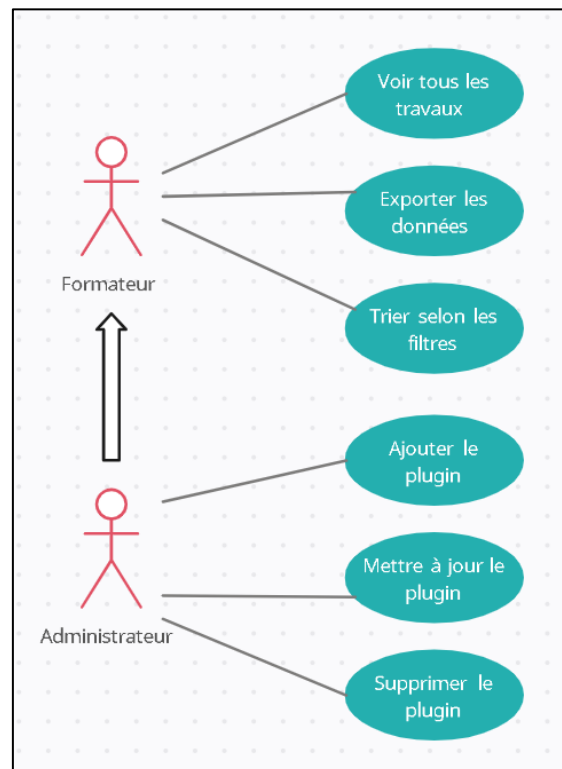


Figure 3 : Diagramme des cas d'utilisation du plugin.

Comme nous pouvons le voir sur ce diagramme des cas d'utilisation, un formateur peut effectuer 3 actions, et un administrateur **hérite** du formateur. Il peut donc faire toutes les actions du formateur, en plus de ses 3 actions (ajouter, mettre à jour et supprimer). Penchons-nous plus en **détail** sur ces 3 uses case :

- **Voir tous les travaux** : Il s'agit ici d'afficher, sous forme de section dynamique et de tableaux, les différentes données voulues,
- **Trier selon les filtres** : Il faut donc ajouter des filtres permettant de trier comme bon nous semble les résultats (par nom d'étudiant.e ou par promotions/formations),
- **Exporter les données** : Enfin, le plugin devra permettre à l'utilisateur, à l'aide d'un bouton, d'exporter les données voulues pour les manipuler ou encore les diffuser, dans un format Excel ou ods.

Cf. annexe 2 pour plus de détails sur la demande.

Notre travail sera donc de **développer ces 3 cas d'utilisation** (car ceux de l'administrateur sont déjà implémentés sur Moodle). Néanmoins, nous avons dû nous adapter en fonction des différentes contraintes posées par ce projet, afin de remplir nos objectifs.

B) Définition des contraintes et des objectifs :

La première contrainte était tout simplement la mise en place de l'environnement de travail. En effet, nous avons configuré un PC dans son entièreté, en le démarrant sous Linux. Ce PC n'avait pas d'OS² intégré, nous devions donc l'installer. Notre tuteur professionnel avait déjà choisi en amont la distribution Linux que nous allions utiliser, **Ubuntu 22.04**. La distribution était gravée sur un CD-ROM, nous avons donc dû l'insérer dans le PC et au démarrage, changer les options du démarrage au niveau du BIOS³. Après plusieurs tentatives, nous avons remarqué que ce PC ne détectait pas le démarrage sur un CD-ROM, nous avons donc dû mettre notre distribution sur une clé USB.

Le plugin devait s'insérer parfaitement dans le serveur existant de l'école, c'est-à-dire être placé au même endroit que tous les autres plugins, pour que Moodle le reconnaisse et puisse l'installer.

```
root@vicky:/var/www/html/moodle# ls
admin          composer.lock  grade          mnet           README.txt
analytics      config-dist.php group          mod            report
auth           config.php     Gruntfile.js  my             reportbuilder
availability    contentbank    h5p           notes          repository
backup         CONTRIBUTING.txt help_ajax.php npm-shrinkwrap.json  rss
badges         COPYING.txt    help.php      package.json   search
behat.yml.dist course         index.php     payment        security.txt
blocks         customfield    install       phpcs.xml.dist tag
blog           dataformat     install.php   phpunit.xml.dist  theme
brokenfile.php draftfile.php  INSTALL.txt   pix            tokenpluginfile
cache          editmode.php   iplookup     plagiarism      TRADEMARK.txt
calendar       enrol          lang          pluginfile.php  user
cohort         error          lib           portfolio       userpix
comment        favourites     local         privacy         version.php
competency     file.php       login         PULL_REQUEST_TEMPLATE.txt  webservice
completion     files          media         question
composer.json  filter         message      rating
```

Figure 4 : Arborescence du serveur Moodle.

Nous pouvons voir ici plusieurs dossiers (en bleu) et fichiers (en blanc) nécessaires au bon fonctionnement de Moodle et de ses différents plugins. En fonction de son type, un plugin peut voir sa structure de dossiers / fichiers différer des autres plugins, et il faut également savoir où le placer exactement dans l'arborescence du serveur de l'école. Nous verrons cela plus en détail dans la partie suivante. L'IFRASS avait Moodle en version 4.1.2, ce qui est important pour le développement d'un plugin, car celui-ci doit être compatible avec la version voulue. De plus, certaines fonctions ou objets varient selon les versions de Moodle, il fallait donc s'assurer de toujours produire du code cohérent avec Moodle 4.

² OS : C'est le système d'exploitation de la machine (MAC OS, Windows ou Linux pour les PC).

³ BIOS : C'est le programme que le microprocesseur d'un ordinateur personnel utilise pour démarrer celui-ci lors de la mise en marche.

La base de données était sous MariaDB. Nous avons dû installer un outil d'administration de base de données (HeidiSQL⁴), pour visualiser les différentes tables et les données. Enfin, pour les langages de programmation, PHP⁵ fut le langage principal, avec du SQL⁶, du CSS⁷ et du JavaScript⁸.

Nous avons donc pour objectif durant ce stage de développer un plugin conforme aux exigences de notre client, c'est-à-dire permettant de visualiser le suivi des travaux des étudiants en fonction de différents filtres, puis d'avoir la possibilité d'exporter ces données. Nous avons pu tout de même proposer des **suggestions d'amélioration** pour le plugin, et avons également une importante marge de manœuvre concernant la manière d'afficher les données. Nous allons désormais vous détailler notre **méthode de travail** et les outils utilisés durant ce projet.

⁴ HeidiSQL : outil d'administration de base de données possédant un éditeur SQL et un constructeur de requête.

⁵ PHP : PHP Hypertext Preprocessor, langage de programmation orienté objet.

⁶ SQL : Structured Query Language, langage de programmation servant à exploiter les bases de données relationnelles.

⁷ CSS : Cascading Style Sheet, langage informatique permettant de donner du style aux éléments de notre code.

⁸ JavaScript : langage de programmation de scripts.

III- Réalisation technique :

A) Organisation :

Au commencement de ce projet, nous avons établi un **Gantt⁹ prévisionnel**, afin d'anticiper les tâches à réaliser et d'être le plus organisé possible. Nous avons donc prévu d'installer le poste de travail la première semaine, et de commencer l'étude de la base de données en parallèle. La semaine suivante, nous avons prévu de commencer le développement du plugin, en lisant les documentations de Moodle 4, tout en rédigeant le rapport de stage et préparant la soutenance, pour être sûr de ne rien oublier.

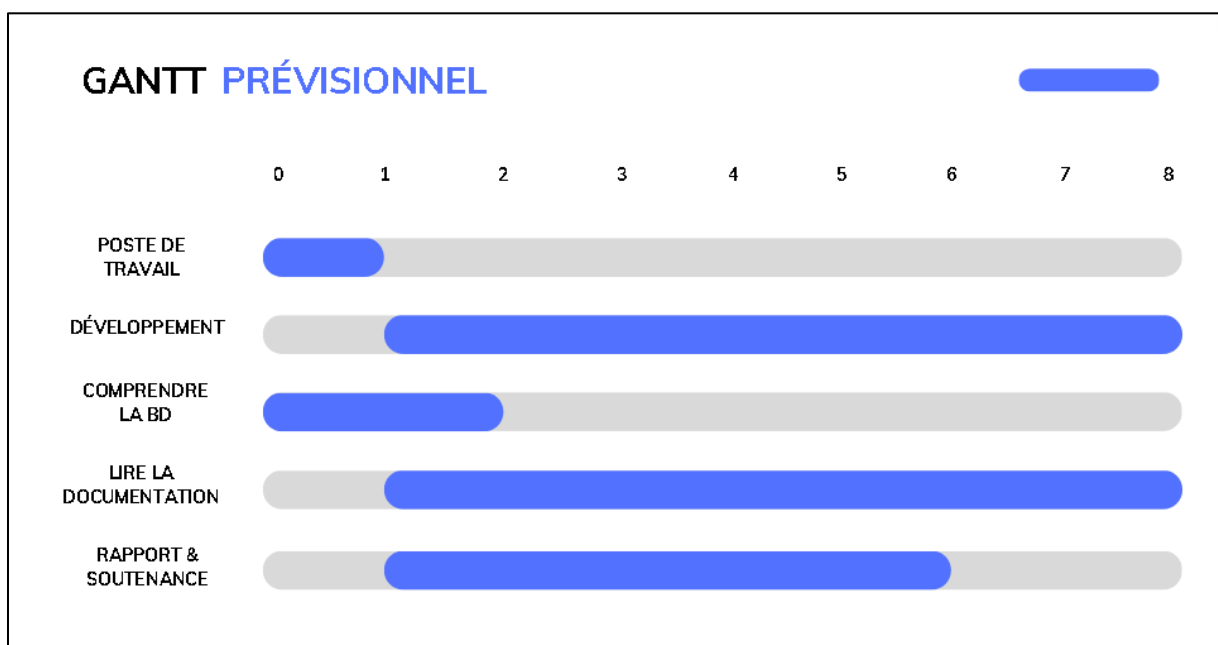


Figure 5 : Diagramme de Gantt prévisionnel du projet.

⁹ Diagramme de GANTT : Graphique permettant de visualiser dans le temps les diverses tâches composant un projet

B) Outils utilisés :

Comme dit précédemment dans la partie II- **Analyse des besoins et des moyens, B) Définition des contraintes et des objectifs**, les technologies et langages à utiliser durant ce projet nous ont été imposés (principalement par Moodle en lui-même, les plugins doivent respecter de nombreuses contraintes) :

- Pour récupérer des fichiers du serveur Debian de l'école, et à l'inverse en transférer (pour tester le plugin, il faut transférer les différents dossiers et fichiers sur le serveur), nous avons utilisé le **terminal Ubuntu** avec ses différentes commandes.

```
esteban@esteban-HP-EliteBook-8570p:~/Documents$ scp report_students_achievements_moodle40_2023042000.zip esteban@192.168.1.50:/home/esteban
```

Figure 6 : Commande pour transférer le plugin sur le serveur.

- Pour le développement, nous avons codé sur **Visual Studio Code**. C'est un IDE¹⁰ que nous avons beaucoup utilisé durant les différents semestres à l'IUT et pour nos projets personnels, qui est facile de prise en main et ergonomique. Nous pouvons choisir en quel langage nous souhaitons développer, et Visual Studio Code s'occupe de créer un code couleur associé. Il est donc très agréable de travailler dessus, nous pouvons nous-mêmes personnaliser notre thème et ainsi être le plus à l'aise possible.

```
<?php session_start();
if ($_SESSION['access'] != 'OK') {
    header('location: formulaireConnexion.php');
}
?>
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Panier</title>
    <link rel="stylesheet" href="./include/panier.css">
</head>
<?php include("../include/header.php"); ?>
```

Figure 7 : Visuel de lignes de codes sur Visual Studio Code.

Comme langages de programmation, nous avons utilisé SQL, CSS, JavaScript et PHP. Ce dernier fut le langage principal, nous permettant de **structurer** le code et de produire l'affichage. Nous avons utilisé un peu de CSS pour **styler** les différents éléments de notre plugin, SQL nous a permis de faire les **requêtes d'interrogation** de la base de données. Enfin,

¹⁰ IDE : Environnement de développement intégré. C'est le logiciel sur lequel nous développons.

nous avons utilisé JavaScript pour rendre plus agréable l'utilisation du plugin, et mettre à jour les données affichées en **temps réel**.

Après avoir les différents outils utilisés, je vais vous détailler le déroulement du projet.

C) Déroulement :

La première chose à faire, après avoir configuré le poste de travail, était **d'installer** Visual Studio Code. Nous avons donc dû télécharger le paquet .deb sur le site officiel, puis l'installer à l'aide de la commande suivante : **sudo dpkg -i [nom_paquet.deb]**. Ensuite, il fallait installer HeidiSQL, pour **visualiser la base de données**. Néanmoins, c'est un logiciel uniquement disponible sur Windows, il nous a donc fallu **contourner le problème** en installant Wine¹¹. Ceci fait, nous avons terminé de configurer le poste de travail et d'installer tous les outils, nous pouvions commencer à travailler.

Après avoir lu et compris les documentations en lignes concernant l'API¹² de Moodle, nous avons dû choisir le **type** du plugin à développer. En effet, il existe plusieurs types de plugins, répondant chacun à des besoins spécifiques (*liste non-exhaustive*) :

- ❖ Les plugins de **thème** : « Les plugins de thème permettent de personnaliser l'apparence de Moodle. Les thèmes sont particulièrement utiles pour les organisations qui souhaitent harmoniser l'interface de leur plateforme avec leur marque ou leur charte graphique ».
- ❖ Les plugins **d'activités** : « Les plugins d'activités ajoutent de nouvelles fonctionnalités aux activités existantes dans Moodle, comme les forums, les devoirs, les quiz, etc. Les plugins d'activités peuvent être utiles si vous avez besoin de fonctionnalités spécifiques pour des activités particulières ».
- ❖ Les plugins de **blocs** : « Les plugins de blocs permettent d'ajouter des blocs personnalisés à votre page d'accueil Moodle, qui peuvent afficher des informations telles que des liens vers d'autres cours, des calendriers, des flux RSS, etc. Les plugins de blocs sont particulièrement utiles pour personnaliser l'interface utilisateur de votre plateforme Moodle ».
- ❖ Les plugins de **rapports** : « Les plugins de rapports fournissent des informations détaillées sur l'utilisation de Moodle, telles que les résultats des quiz, les statistiques

¹¹ Wine : logiciel libre permettant à des logiciels conçus seulement pour Windows de fonctionner dans d'autres environnements comme Linux ou MacOS.

¹² API : Application Programming Interface ou « interface de programmation d'application », est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

des forums, etc. Les plugins de rapports peuvent être utiles pour évaluer l'efficacité de Moodle et identifier les domaines à améliorer ».

Nous pouvons remarquer que le plugin le plus approprié est ici le plugin de type **rapport**. En effet, grâce à celui-ci, nous allons pouvoir récupérer les informations des travaux réalisés des étudiants, pour les afficher sous forme de sections dynamiques et de tableaux.

Ensuite, il fallait créer un dossier pour le plugin, en suivant une structure particulière. Pour que Moodle **reconnaisse** un plugin, il faut créer certains fichiers, communs à tous les types de plugins :

- ❖ `version.php` : un fichier obligatoire qui définit les informations de base du plugin, telles que la version, les prérequis, le nom et l'identifiant unique du composant. Ce fichier est inclus par Moodle pour obtenir des informations sur le plugin.
- ❖ `lang/en/pluginname.php` : le dossier `lang` contient tous les fichiers de langues, c'est-à-dire les traductions de notre plugin dans les langues souhaitées. Nous devons au minimum mettre un fichier pour l'anglais, contenu dans le dossier '**lang/en**'.
- ❖ `index.php` : un fichier qui empêche l'accès direct au dossier du plugin depuis le navigateur web. Ce fichier est inclus pour des raisons de sécurité.
- ❖ `db/` : ce dossier contient les fichiers de définition de la base de données pour le plugin. Ces fichiers sont utilisés pour créer et maintenir les tables de base de données associées au plugin, et définir les droits d'accès.
- ❖ `classes/` : ce dossier contient les fichiers de classe du plugin, qui contiennent le code PHP pour les fonctionnalités principales du plugin. Les fichiers de classe sont généralement organisés en sous-dossiers selon leur fonctionnalité, comme "`lib`" pour les bibliothèques de fonctions communes, "`locallib`" pour les fonctions spécifiques au plugin, etc.

Après avoir créé le dossier contenant le plugin, nommé **report_students_achievements**, et ajouté les dossiers et fichiers nécessaires, nous l'avons transféré sur le serveur de Moodle de l'école, pour vérifier qu'il reconnaissait bien le plugin et ne générât pas d'erreurs. Comme nous pouvons le voir ci-dessous, le plugin a bien été reconnu.

Students Achievements report_students_achievements	4.1.2 2022112802.09	Désinstaller	Additionnel
---	------------------------	--------------	-------------

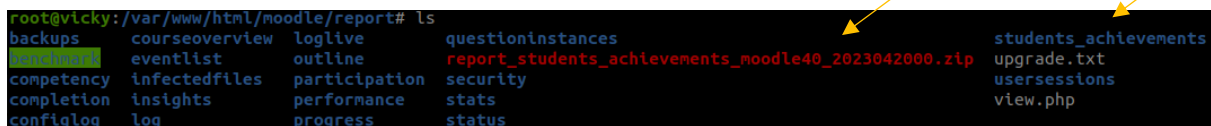
Figure 8 : Plugin bien reconnu par Moodle.

À noter que pour ajouter un plugin sur Moodle, il faut créer un fichier zip du dossier contenant le plugin, et l'ajouter dans le dossier de Moodle correspondant au **type** du plugin, dans l'arborescence suivante :

/var/www/html/moodle/report/

(cf. figure 3)

Ensuite, il faut se positionner à l'endroit où le plugin zippé s'est déposé ('cd' pour se déplacer sur Linux), puis le dézipper (unzip).

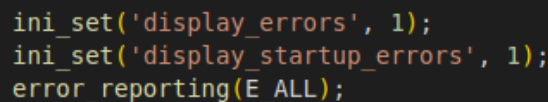


```
root@vicky:/var/www/html/moodle/report# ls
backups      courseoverview  loglive      questioninstances  students_achievements
benchmark    eventlist       outline      report_students_achievements_moodle40_2023042000.zip  upgrade.txt
competency   infectedfiles   participation security            usersessions
completion   insights       performance  stats              view.php
configlog    log            progress     status
```

Figure 9 : Le plugin placé au bon endroit dans l'arborescence du serveur.

(On peut voir en rouge le dossier du plugin zippé, puis en haut à droite le dossier du plugin).

Enfin, il nous suffit de rafraîchir la page d'accueil de Moodle, pour voir l'ajout du plugin et les différentes erreurs s'il y en a. En ajoutant ces lignes de codes à chacun de nos fichiers PHP, nous pouvons voir les erreurs directement affichées sur le navigateur :



```
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

Figure 10 : Lignes de code permettant de voir les erreurs des fichiers PHP.

La dernière chose à faire était donc de coder le corps du plugin. Nous allons vous expliquer ici le développement d'une fonctionnalité, à savoir le formulaire permettant de choisir une cohorte (promotion), et d'afficher sous forme de tableau tous les étudiants de celle-ci.

```

ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL); //pour afficher les erreurs dans le navigateur

define('NO_OUTPUT_BUFFERING', true);

require(__DIR__ . '/../../config.php');
require_once($CFG->dirroot . '/cohort/lib.php');
require_once('classes/form/cohort_students_form.php');
require_once('classes/form/autocomplete_students_form.php');
require_once('classes/form/cohorts_of_student_form.php');

require_login();

//require_capability('report/students_achievements:view', context_system::instance());*/

//Set context and url
$PAGE->set_context(context_system::instance());

//include css file
$PAGE->requires->css($CFG->dirroot . '/report/students_achievements/styles.css');

//set url of the page
$url = new moodle_url('/report/students_achievements/index.php');
$PAGE->set_url($url);

//js
echo '<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>';
echo '<script src="https://192.168.1.50/moodle/report/students_achievements/script_details.js"></script>';

//Title of the page
$PAGE->set_title(get_string('title', 'report_students_achievements'));

//Add moodle stuff
echo $OUTPUT->header();

```

Figure 11 : Contenu du fichier index.php (1/4).

Dans le fichier index.php, qui s'exécute lorsque nous cliquons sur le plugin, on ajoute d'abord tous les **prérequis** (liens vers les fichiers de configuration, vers les classes contenant les formulaires, vers les fichiers de l'api Moodle, etc.). On ajoute ensuite à notre page son contexte, sa feuille de style CSS, son URL et ses fichiers javascript, qui vont nous permettre d'afficher **dynamiquement** les formulaires et résultats. Enfin, on met le nom du header, puis on affiche le contenu supérieur de Moodle (barre de navigation, logo...).

```

//heading
echo '<div class="title">';
|   echo $OUTPUT->heading(get_string('heading', 'report_students_achievements'));
echo '</div>';

//2 forms + 1 hidden
$form = new cohort_students_form();
$form2 = new autocomplete_students_form();
$form3 = new cohorts_of_student_form();

```

Figure 12 : Contenu du fichier index.php (2/4).

Ensuite, on instancie les 3 **formulaires**, mais nous ne verrons que le premier dans notre exemple. Ces formulaires sont définis dans les fichiers prévus à cet effet, qui se trouvent dans le dossier **'/classes/form/'**, à la racine du plugin.

```
echo '<div class="form-container">
    <div class="form-column">;
        $form->display();
    echo '</div>

    <div class="form-column">;
        $form2->display();
    echo '</div>
    </div>';

echo '<div id="number-student"></div>'; //text that indicates the number of results found

echo '<div id="hidden-form">';
    $form3->display(); //hidden form, cohorts of the student
echo '</div>';

//HTML table
$table = new html_table();
$table->id = 'result-table';

echo html_writer::table($table);
```

Figure 13 : Contenu du fichier index.php (3/4).

On affiche les formulaires à l'écran, contenus dans des balises html (pour pouvoir gérer leur position sur la page avec CSS). On instancie notre tableau, un objet de type **html_table**, fourni par l'api Moodle. On ajoute un identifiant à ce dernier, afin de pouvoir le sélectionner pour effectuer des actions dessus, comme supprimer les données par exemple. Enfin, on affiche ce tableau, par le biais de la méthode **html_writer**.

```
echo '<div id = "container"></div>';

//echo $OUTPUT->single_button('javascript:void(0)', '-->', 'post', array('class' => 'button_next'));

echo '<div id = "completion-student"></div>';
//right blocks and footer
echo $OUTPUT->blocks('side-post');
echo $OUTPUT->footer();
```

Figure 14 : Contenu du fichier index.php (4/4).

Les dernières lignes de cet index.php créent 2 div, pour afficher de futures informations. On affiche enfin les blocs de contenu de Moodle (blocs d'informations sur le côté), puis le footer.

Voici le résultat :

Figure 15 : Résultat de la page d'index.

On voit bien ici le titre de la page, les 2 formulaires (le 3^{ème} est caché au départ), le header et le footer.

Nous allons maintenant voir le fichier qui contient la définition du formulaire, **/classes/form/cohort_students_form.php**.

```
<?php

defined('MOODLE_INTERNAL') || die();
// moodleform is defined in formslib.php
require_once("$CFG->libdir/formslib.php");
require_once('/var/www/html/moodle/config.php');
$PAGE->requires->js('/report/students_achievements/script.js');

class cohort_students_form extends moodleform {
```

Figure 16 : Contenu du fichier du 1er formulaire (1/3).

Comme pour l'index.php, il nous faut tout d'abord définir les prérequis. On ajoute ici la classe de Moodle '**formslib.php**', qui contient tous les formulaires, le fichier de configuration et enfin le fichier javascript, qui va s'occuper d'afficher dynamiquement les étudiant de chaque cohorte. On voit que notre classe étend la classe fournie par Moodle, **moodleform**, qui fournit des méthodes pour agir sur des formulaires.

```

public function definition() {
    global $CFG, $DB;
    // A reference to the form is stored in $this->form.
    // A common convention is to store it in a variable, such as '$mform'.
    $mform = $this->_form; // Don't forget the underscore!

    // Récupérer toutes les cohortes de la base de données
    $cohort = cohort_get_all_cohorts(0, -1);

    // Créer un tableau d'options pour la liste déroulante des cohortes
    $allCohorts = array();
    $allCohorts[] = ''; //first value

    foreach ($cohort['cohort'] as $cohort) {
        $allCohorts[$cohort->id] = $cohort->name;
    }
}

```

Figure 17 : Contenu du fichier du 1er formulaire (2/3).

On stocke dans un premier temps la référence du formulaire actuel dans une variable. Dans une variable appelée 'cohortes', on stocke **toutes les cohortes** présentes dans la base de données. On crée ensuite un tableau, qui contiendra toutes les options de notre formulaire, et on ajoute comme première valeur une chaîne de caractères vide (pour ne pas avoir la première cohorte de sélectionnée par défaut). Ensuite, à l'aide d'un **foreach**, on parcourt toutes les cohortes, et plus particulièrement le champ '**cohortes**', qui contient des informations sur chacune des cohortes (id, nom entier, nom court, description, date de création, etc.). Le tableau **\$allCohorts** contiendra donc tous les noms des cohortes, ainsi que leurs id respectifs. La structure de ce tableau permettra de voir affiché dans la liste déroulante le nom des cohortes, et de récupérer l'id de la cohorte sélectionnée lorsque l'on clique dessus.

```

$options = array(
    'placeholder' => get_string('select_cohort', 'report_students_achievements'),
);

// Ajouter la liste déroulante des cohortes
$mform->addElement('select', 'cohort', get_string('cohort', 'report_students_achievements'), $allCohorts, $options);
$mform->setType('cohort', PARAM_INT);
$mform->addElement('html', '<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>');
}
}

```

Figure 18 : Contenu du fichier du 1er formulaire (3/3).

On crée ensuite un tableau de paramètres (optionnel), qui permettra d'ajouter certaines options dans notre liste déroulante (le **placeholder** ne marche pas sur une liste déroulante, je l'ai laissé pour montrer un exemple). Enfin, on ajoute à notre formulaire le champ souhaité, ici une liste déroulante (**select**), qui a pour identifiant '**cohort**', comme nom '**cohort**' (récupéré dans le **dossier de langues** via la méthode 'get_string'), et qui contient les valeurs des cohortes que nous avons stockées dans le tableau **\$allCohorts**. On précise que le type de valeur de ce champ sera un entier (l'identifiant des cohortes), puis on ajoute un fichier

javascript, qui permettra via jQuery¹³ et Ajax¹⁴ d'afficher en temps réel les résultats en cliquant sur une cohorte, d'où l'absence de boutons '**Valider**' et '**Annuler**' dans le formulaire (la page ne se recharge pas de cette manière).

Voici le résultat :

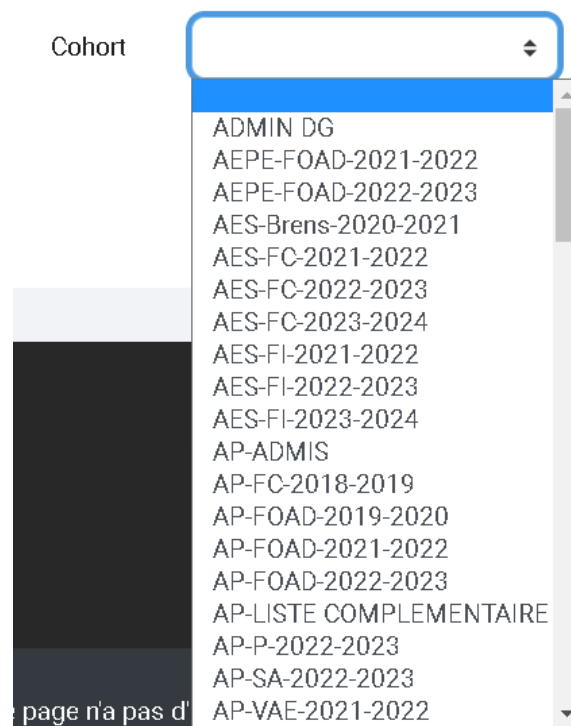


Figure 19 : Résultat du premier formulaire.

¹³ jQuery : jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.

¹⁴ Ajax : Ajax est une méthode utilisant différentes technologies, permettant de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète.

Nous allons désormais nous pencher sur le fichier javascript qui s'occupe de détecter quand l'utilisateur clique sur une cohorte.

```
console.log('Le fichier JavaScript est exécuté.');
```

```
$(document).ready(function() {
    // Écouter les modifications du champ select
    $('#id_cohort').on('change', function() {
        console.log('Click promo');

        // Récupérer la valeur sélectionnée
        var cohortId = $(this).val();
        console.log(cohortId);

        // Effectuer une requête AJAX
        $.ajax({
            url: 'ajax_query.php',
            type: 'POST',
            data: { cohortId: cohortId }, // Envoyer l'ID de la cohorte sélectionnée en tant que paramètre
        });
    });
});
```

Figure 20 : Contenu du fichier javascript du 1er formulaire (1/3).

Dans un premier temps, on affiche un message en console pour vérifier que le fichier javascript a bien été détecté. Ensuite, on sélectionne, à l'aide de l'identifiant, le champ select qui contient toutes les cohortes. On détecte lorsque l'utilisateur **clique** dessus, on affiche en console l'identifiant de la cohorte cliquée pour vérifier que tout est cohérent, puis on le stocke dans une variable '**cohortId**'. On effectue ensuite une requête Ajax, qui va nous permettre d'afficher en **temps réel** les résultats, sans avoir besoin de recharger la page. On définit le nom du fichier qui s'occupera d'effectuer la requête SQL pour récupérer le.s **étudiant.e.s** de cette cohorte, on définit la méthode de requête à utiliser (ici **POST**), puis on envoie à ce fichier PHP les données, c'est-à-dire dans notre cas l'identifiant de la cohorte.

Voyons le contenu de ce fichier :

```
<?php

require_once(dirname(__FILE__) . '/../../config.php');
require_once($CFG->libdir . '/tablelib.php');

// Récupérer l'ID de la cohorte depuis le paramètre POST
$cohortId = $_POST['cohortId'];

$sqlusers = "SELECT DISTINCT u.*
FROM {user} u
INNER JOIN {role_assignments} ra ON (ra.userid = u.id)
INNER JOIN {cohort_members} cm ON (cm.userid = u.id)
WHERE ra.roleid = :roleid
AND cm.cohortid = :cohortid
ORDER BY u.lastname ASC;";

$params = array('roleid' => 5,
                'cohortid' => $cohortId); //array of parameters

$students= $DB->get_records_sql($sqlusers, $params);

$count_students = 0;

$result = array();
```

Figure 21 : Contenu du fichier PHP qui récupère les données et les transmet (1/2).

Comme habituellement, on ajoute les fichiers nécessaires en début de code. Via la variable globale `$_POST`, on récupère ensuite les données transmises précédemment par le fichier javascript, l'identifiant de la cohorte choisie, que l'on stocke dans une variable nommée '`$cohortId`'. On effectue par la suite une simple requête SQL pour récupérer les étudiants appartenant à cette cohorte. On ajoute des paramètres à la requête, sur l'id de la cohorte, et sur le rôle de l'utilisateur (le chiffre 5 correspond au rôle '**Étudiant**'). La méthode `get_records_sql()` fournie par Moodle nous permet d'exécuter la requête, et de stocker ses résultats dans une variable, ici nommée '`$students`'. On définit une variable contenant le nombre d'étudiants, puis le tableau final, qui sera retourné.

```
foreach ($students as $student) {
    $count_students++;
}

$result[0][0] = $count_students; //number of results

// Boucle pour afficher les données dans le tableau
foreach ($students as $student) {
    $row = array(
        $student->firstname,
        $student->lastname,
        html_writer::tag('span', get_string('viewdetails', 'report_students_achievements'), array('class' => 'student-details', 'studentId' => $student->id, 'cohortId' => $cohortId))
    );
    $result[] = $row;
}

// Envoyer les résultats en tant que réponse JSON
header('Content-Type: application/json');
echo json_encode($result);
```

Figure 22 : Contenu du fichier PHP qui récupère les données et les transmet (2/2).

Ensuite, on parcourt tous les étudiants retournés par la requête, afin de mettre à jour la variable qui contient le nombre d'étudiants. On ajoute ce nombre dans la première case du tableau multidimensionnel du résultat. Ensuite, on re parcourt tous les étudiants retournés par la requête, pour stocker dans un tableau pour chaque étudiant, son prénom, son nom et un élément html ``, qui contiendra du texte cliquable, qui permettra d'afficher la **progression** de cet étudiant. On ajoute ce tableau au tableau du résultat, puis on retourne le tableau multidimensionnel en l'encodant en JSON¹⁵.

¹⁵ JSON : JavaScript Object Notation (JSON) est un format de données textuel dérivé de la notation des objets du langage JavaScript.

Revenons à notre fichier javascript :

```
success: function(response) {[
    // Traiter la réponse et mettre à jour le tableau HTML existant
    console.log(response);

    var numberStudent = response[0][0];
    var divResult = document.getElementById('number-student');

    var completion = document.getElementById('completion-student');
    completion.innerHTML = '';

    if (numberStudent != 0) {
        divResult.innerHTML = numberStudent + ' students';
    }

    else {
        divResult.innerHTML = 'No result';
    }
}
```

Figure 23 : Contenu du fichier javascript du 1er formulaire (2/3).

La variable '**response**' contient donc la valeur retournée par le script PHP précédent. On affiche en console le tableau, pour vérifier que tout s'est bien passé, puis on stocke le nombre d'étudiants dans une variable '**numberStudent**'. Via la méthode 'getElementById', on sélectionne la <div> définie dans l'index.php, qui contiendra donc une information textuelle sur le nombre de résultats trouvés (nombre d'étudiants de la cohorte choisie par l'utilisateur). On sélectionne également la <div> qui contient les résultats, pour tout effacer (quand on choisit pour la 1^{ère} fois une cohorte, rien ne se passe, mais quand on change de cohorte, on efface ainsi les résultats de la cohorte précédente). La méthode **innerHTML** permet donc de remplacer le contenu de notre div par ce que l'on veut, ici une chaîne de caractère vide, on efface donc juste l'ancien contenu. On effectue un test sur le nombre d'étudiants, si ce nombre est différent de 0, on affiche le nombre d'étudiants trouvés, suivi de la chaîne de caractères 'students'. Sinon, on affiche 'No result'.

```
var table = document.getElementById('result-table');
table.innerHTML = '';

// Itérer sur chaque étudiant dans le tableau
for (var i = 1; i < response.length; i++) {
    var student = response[i];

    // Extraire les informations de l'étudiant
    var info1 = student[0];
    var info2 = student[1];
    var info3 = student[2];

    // Créer une nouvelle ligne dans le tableau
    var newRow = table.insertRow();

    // Ajouter les cellules avec les informations de l'étudiant
    newRow.insertCell().innerHTML = info1;
    newRow.insertCell().innerHTML = info2;
    newRow.insertCell().innerHTML = info3;
}

},

error: function(xhr, status, error) {
    // Gérer les erreurs d'AJAX
    console.log(xhr);
    console.log(status);
    console.log(error);
}
});
});
});
```

Figure 24 : Contenu du fichier javascript du 1er formulaire (3/3).

On sélectionne l'identifiant de notre **tableau** (objet de type **html_table**), pour ensuite vider son contenu. À l'aide d'une boucle **for**, on parcourt **tous les éléments** du tableau, pour afficher le prénom de l'étudiant (**info1**), le nom (**info2**) et le texte cliquable qui permettra d'afficher les informations de complétion des activités de l'étudiant (**info3**). Si javascript a détecté une erreur dans la valeur de retour '**response**', il sautera ces instructions, et ira dans la partie **error**, et l'affichage en console de ces 3 paramètres 'xhr', 'status' et 'error' nous permet de comprendre l'erreur en question.

Voici le résultat final pour cette fonctionnalité :

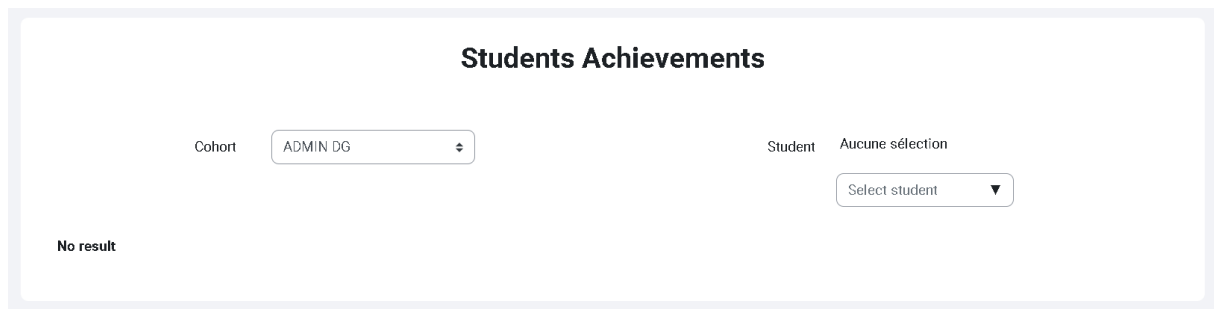


Figure 25 : Résultat de la fonctionnalité (1/2).

Quand il n'y a aucun étudiant dans la cohorte sélectionnée, on affiche le message 'No result'.

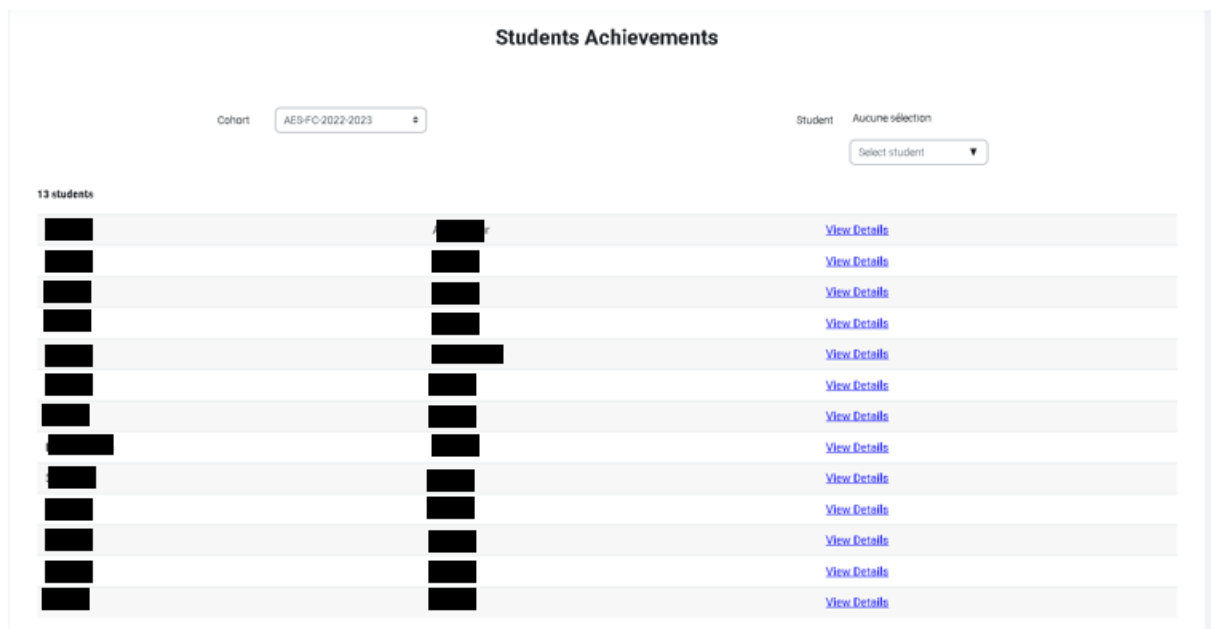


Figure 26 : Résultat de la fonctionnalité (2/2).

Sinon, on affiche le nombre d'étudiants appartenant à la cohorte en question en haut à gauche, puis on affiche le tableau, contenant le prénom et le nom de l'étudiant, et un lien permettant d'afficher pour chaque étudiant, son avancement dans les activités.

Les informations des étudiants sont cachées pour des raisons de confidentialité.

D) Retour d'expérience & avancement :

Ayant eu les 2 semaines de vacances de Pâques, ce rapport est rendu au bout de **6 semaines de travail**, le plugin n'est donc pas entièrement fini, et certaines choses sont susceptibles de changer, particulièrement la structure du code. Nous devons durant les 2 dernières semaines de ce stage coder la fonctionnalité d'export, et **refactorer** le code (le rendre plus lisible, exporter des parties de code dans des fonctions, l'optimiser et finir de le commenter).

Voici le GANTT réel, à la suite de 6 semaines de travail :

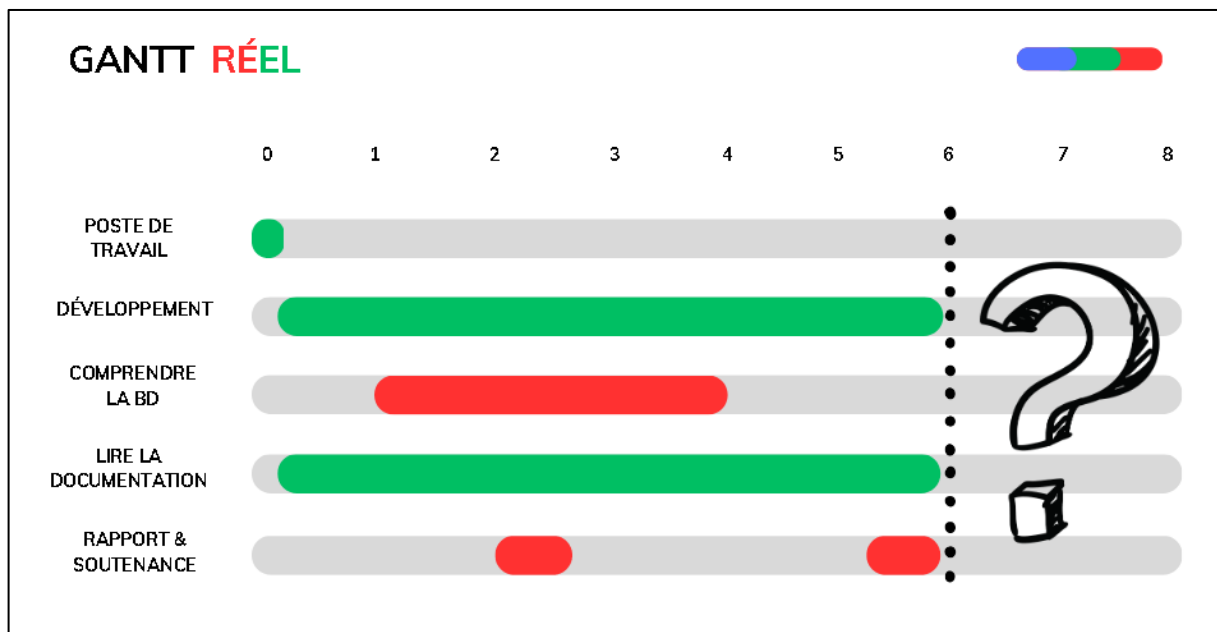


Figure 27 : Diagramme de GANTT réel du projet.

Nous pouvons voir qu'il y a eu certains **écarts** par rapport au GANTT prévisionnel. En effet, nous avons fini de configurer le poste de travail la 1^{ère} journée, cela nous a donc permis de commencer à lire la documentation et à coder en parallèle bien plus tôt que prévu ! Nous avons commencé à se pencher sur la base de données un peu plus tard, car le début du stage a été consacré à lire beaucoup de documentations, comprendre la structure d'un plugin et l'architecture de Moodle. Enfin, nous avons commencé à rédiger le rapport de stage en milieu de 3^{ème} semaine, pour le poursuivre et le finir la semaine du rendu (8 juin).

Nous venons de voir le déroulement du projet, mais quand est-il des **résultats obtenus** ?

IV- Résultats :

Je vais vous exposer ici plusieurs captures d'écrans représentant les différents visuels du plugin, mais avant ça, voici ce qu'il faut savoir pour mieux comprendre les affichages :

- Chaque cohorte est inscrite à des blocs, qui peuvent contenir des modules, qui eux peuvent contenir des séquences, qui elles peuvent contenir enfin des activités

(Blocs > Modules > Séquences > Activités).

- Un.e étudiant.e peut se trouver dans **une** ou **plusieurs** cohortes, et même dans **aucune** cohorte. S'il n'a pas de cohortes, alors il est inscrit **individuellement** à chaque cours, et donc l'affichage se fait directement sous la forme *Séquences > Activités*.
- Tous les étudiant.es au sein d'une même cohorte ont donc les mêmes blocs, modules, séquences et activités, il faut donc à l'aide de requêtes afficher les bonnes informations de **complétion** pour chaque étudiant distinct.
- Il y a des **restrictions** d'accès sur certaines activités, ce qui veut dire que deux étudiants d'une même cohorte peuvent ne pas voir les mêmes activités sur Moodle (champs de restriction sur le profil, sur le groupe, etc.). Il faut donc prendre en compte ces restrictions, afin de ne pas afficher les informations de complétions d'activités à un.e étudiant.e qui n'y a pas accès.
- Enfin, on affichera pour chaque activité, son intitulé, son type (devoir, quiz, atelier, etc.), son état de complétion et enfin sa date d'ouverture.

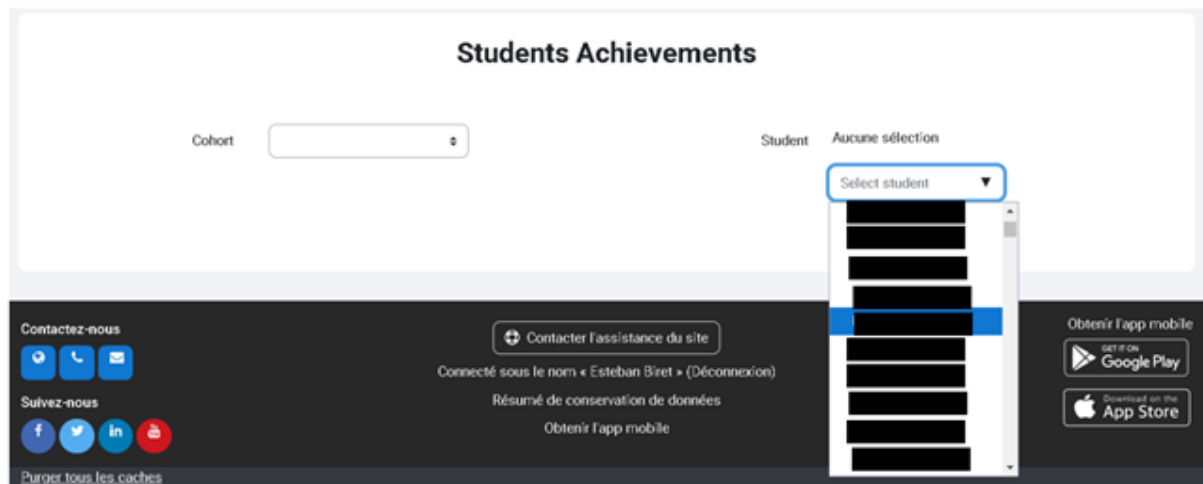


Figure 28 : Sélection d'étudiant par le nom et le prénom.

Students Achievements

All cohorts

All students

Select student ▼

SWITCH - II/ Business Plan

SWITCH - III/ Communication

SWITCH - I/ State of the art

Nom de l'activité	Type	Statut de complétion	Date d'ouverture
Ressources	folder	✓	-
Ressources	folder	✓	-
Ressources Complémentaires	folder	✓	-
Stakeholders Resources	folder	✓	-
[FR] - [COURS] - Les flux	folder	✓	-

Figure 29 : Étudiant n'ayant pas de cohorte (affichage séquences > activités).

Students Achievements

All cohorts

All students

Select student ▼

► Bloc 1 - Accompagnement et soins de l'enfant dans les activités de sa vie quotidienne et de sa vie sociale

► Bloc 2 - Evaluation de l'état clinique et mise en œuvre de soins adaptés en collaboration

► Bloc 3 - Information et accompagnement des personnes et de leur entourage, des professionnels et des apprenants

► Bloc 4 - Entretien de l'environnement immédiat de la personne et des matériels liés aux activités en tenant compte du lieu et des situations d'intervention

► Bloc 5 - Travail en équipe pluri-professionnelle et traitement des informations liées aux activités de soins, à la qualité/gestion des risques

Figure 31 : Blocs d'une cohorte.

▼ Bloc 1 - Accompagnement et soins de l'enfant dans les activités de sa vie quotidienne et de sa vie sociale

▼ Module 1 - Accompagnement de l'enfant dans les activités de sa vie quotidienne et de sa vie sociale

Raisonnement clinique en EAJE

▼ Interculturalité

Nom de l'activité	Type	Statut de complétion	Date d'ouverture
test devoir	assign	✖	-

► Parentalité

Soin professionnel

Le portage

▼ Soins d'hygiène et de confort

Nom de l'activité	Type	Statut de complétion	Date d'ouverture
Activité - "Pour croquer la vie à pleines dents"	assign	✔	-

Sommeil

Diététique infantile: alimentation diversifiée

Figure 30 : Listes déroulantes pour voir les activités.

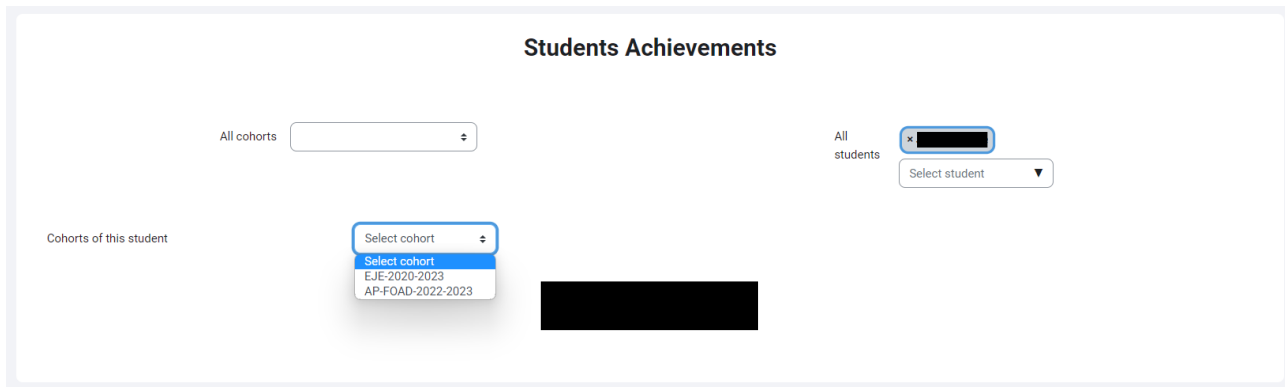


Figure 32 : Étudiant choisi par son nom, ayant plusieurs cohortes.

Notre plugin a eu plusieurs **impacts** vis-à-vis de l'école. Dans un premier temps, il a permis aux formateurs de l'IFRASS de gagner du temps lors du suivi des travaux des étudiants en distanciel, et de pouvoir manipuler et diffuser ces données grâce à l'export. Ce projet pourra être repris aisément par d'autres développeurs, afin de le faire évoluer et de le rendre encore plus pertinent, grâce à la lisibilité du code et à ses commentaires.

Après avoir détaillé les résultats de notre projet, nous allons nous concentrer sur les différents **bilans**.

V- Bilans :

Nous allons désormais vous présenter les différents bilans de ce projet. Le bilan **technique** servira à évaluer le travail accompli et faire un point sur les découvertes, l'**humain** conclura sur le relationnel avec le maître de stage, et enfin le bilan **personnel** exposera les apports du stage, en faisant le lien entre les compétences acquises à l'IUT et celles développées durant ce stage.

A) Technique :

Il est difficile d'estimer le pourcentage du travail effectué à la suite de ce stage, car il nous reste encore **2 semaines** de développement. Néanmoins, nous avons codé le système de choix par cohortes & étudiant, la récupération et l'affichage des données pertinentes. Il nous reste actuellement à développer le système d'export des données, et à restructurer le code.

Ce stage m'a donc fait découvrir un nouvel aspect du développement, celui des outils pédagogiques. J'ai pu découvrir la communauté de développement de Moodle, et participer à la vie du forum. J'ai progressé en Javascript, SQL et PHP, mais également en administration système.

B) Humain :

Durant le stage, nous nous sommesentraînés avec mon tuteur professionnel, M. Jérémie PILETTE. En effet, il m'a expliqué la structure du serveur de l'IFRASS, celle de la base de données et le fonctionnement du Moodle de l'école. Il se formait également au développement de plugin Moodle en même temps que moi, et en développait un pour la gestion de groupe. Nous avons donc pu nous prodiguer des conseils mutuellement, et ainsi être plus efficace et travailler en équipe.

C) Personnel :

Ce projet m'a permis de développer de multiples compétences et de découvrir la vie professionnelle. Voici une liste détaillant pour chaque compétence les apprentissages liés :

- ❖ **Réaliser un développement d'application** : Développer le plugin en utilisant divers langages de programmation.
- ❖ **Optimiser des applications informatiques** : Produire un plugin intuitif, qui ne ralentit pas la vitesse de chargement de la page, créer du code et des requêtes optimisés.
- ❖ **Administrer des systèmes informatiques communicants complexes** : Installer le poste de travail (Ubuntu), comprendre la structure du serveur (Debian) et à quels endroits y intervenir.
- ❖ **Gérer des données de l'information** : Analyser la base de données existante, produire des requêtes SQL pour récupérer les informations.
- ❖ **Conduire un projet** : Savoir s'organiser le plus efficacement possible, prendre des décisions.
- ❖ **Travailler dans une équipe informatique** : Être ponctuel et cordial, s'intégrer dans la vie professionnelle en entreprise/école, et échanger avec les différents membres de l'IFRASS.

Ce stage m'a donc permis de m'améliorer sur toutes les compétences développées à l'IUT. Les cours de **communication** de ces trois semestres m'ont permis de savoir comment bien se comporter en milieu professionnel, et les bonnes pratiques à adopter. Pour ce qui est du développement principalement en PHP et de l'optimisation, je me suis servi de tout ce que j'avais vu en cours de **développement web** et en **qualité de développement**. En matière de **base de données**, j'ai pu compter sur tous les cours des deux années à l'IUT. Enfin, les cours de **système** m'ont guidé dans l'appréhension du poste de travail.

Conclusion :

Durant ces 8 semaines de stage, nous avons pu réaliser qu'une communication très fréquente avec le client et l'organisation du travail étaient d'importants facteurs de réussite d'un projet informatique. Nous avons donc développé ce plugin en n'ayant aucun existant, et à 2 semaines de la fin du stage, il ne reste plus qu'à coder la fonctionnalité d'export et refactorer le code. Ce stage m'a donc permis de découvrir la vie professionnelle, et de renforcer mes compétences en développement et en algorithmie. J'ai découvert le monde du développement Moodle, et il m'a plutôt plu. Grâce à ce stage, je suis confiant pour mon second stage en entreprise l'année prochaine.

Si c'était à refaire, je m'attarderais plus sur la compréhension de l'API Moodle et les conventions de codage des plugins, afin de ne pas produire du code incohérent et ainsi gagner du temps, même si cela fait partie du processus de découverte d'un nouvel environnement.

Table des illustrations :

Figure 1 : Organigramme de l'IFRASS.....	4
Figure 2 : Item du menu pour accéder au plugin.....	5
Figure 3 : Diagramme des cas d'utilisation du plugin.	6
Figure 4 : Arborescence du serveur Moodle.....	8
Figure 5 : Diagramme de Gantt prévisionnel du projet.	10
Figure 6 : Commande pour transférer le plugin sur le serveur.....	11
Figure 7 : Visuel de lignes de codes sure Visual Studio Code.	11
Figure 8 : Plugin bien reconnu par Moodle.....	13
Figure 9 : Le plugin placé au bon endroit dans l'arborescence du serveur.	14
Figure 10 : Lignes de code permettant de voir les erreurs des fichiers PHP.	14
Figure 11 : Contenu du fichier index.php (1/4).....	15
Figure 12 : Contenu du fichier index.php (2/4).....	15
Figure 13 : Contenu du fichier index.php (3/4).....	16
Figure 14 : Contenu du fichier index.php (4/4).....	16
Figure 15 : Résultat de la page d'index.	17
Figure 16 : Contenu du fichier du 1er formulaire (1/3).	17
Figure 17 : Contenu du fichier du 1er formulaire (2/3).	18
Figure 18 : Contenu du fichier du 1er formulaire (3/3).	18
Figure 19 : Résultat du premier formulaire.....	19
Figure 20 : Contenu du fichier javascript du 1er formulaire (1/3).....	20
Figure 21 : Contenu du fichier PHP qui récupère les données et les transmet (1/2).....	20
Figure 22 : Contenu du fichier PHP qui récupère les données et les transmet (2/2).....	21
Figure 23 : Contenu du fichier javascript du 1er formulaire (2/3).....	22
Figure 24 : Contenu du fichier javascript du 1er formulaire (3/3).....	23
Figure 25 : Résultat de la fonctionnalité (1/2).	24
Figure 26 : Résultat de la fonctionnalité (2/2).	24
Figure 27 : Diagramme de GANTT réel du projet.....	25
Figure 28 : Sélection d'étudiant par le nom et le prénom.	27
Figure 29 : Étudiant n'ayant pas de cohorte (affichage séquences > activités).....	27
Figure 30 : Listes déroulantes pour voir les activités.	28
Figure 31 : Blocs d'une cohorte.....	28
Figure 32 : Étudiant choisi par son nom, ayant plusieurs cohortes.	29

Lexique :

Plugin : Logiciel conçu pour être greffé à un autre logiciel à travers une interface prévue à cet effet, et apporter à ce dernier de nouvelles fonctionnalités.

OS : C'est le système d'exploitation de la machine (MAC OS, Windows ou Linux pour les PC).

BIOS : C'est le programme que le microprocesseur d'un ordinateur personnel utilise pour démarrer celui-ci lors de la mise en marche.

HeidiSQL : Outil d'administration de base de données possédant un éditeur SQL et un constructeur de requête.

PHP : PHP HyperText Preprocessor, langage de programmation orienté objet.

SQL : Structured Query Langage, langage de programmation servant à exploiter les bases de données relationnelles.

CSS : Cascading Style Sheet, langage informatique permettant de donner du style aux éléments de notre code.

Javascript : Langage de programmation de scripts.

Diagramme de GANTT : Graphique permettant de visualiser dans le temps les diverses tâches composant un projet.

IDE : Environnement de développement intégré. C'est le logiciel sur lequel nous développons.

Wine : Logiciel libre permettant à des logiciels conçus seulement pour Windows de fonctionner dans d'autres environnements, comme Linux ou MacOS.

API : Application Programming Interface ou « interface de programmation d'application », est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

jQuery : jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.

Ajax : Ajax est une méthode utilisant différentes technologies, permettant de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète.

JSON : JavaScript Object Notation (JSON) est un format de données textuel dérivé de la notation des objets du langage JavaScript.

Bibliographie / Sitographie :

<https://moodle.org/> : Pour les questionnements liés au développement du plugin.

<https://moodledev.io/docs/apis> : La documentation officielle de l'API Moodle 4.

<https://www.canva.com/> : Pour les visuels de la 1^{ère} page de couverture et de la 4^{ème} de couverture.

<https://www.ifrass.fr/> : Site web de l'IFRASS.

Table des annexes :

Annexe 1 : Diplômes en FI.....	37
Annexe 2 : Besoin de l'organisme.	38

Annexes :

SANTÉ
Diplôme d'Etat de Puéricultrice
Diplôme d'Etat d'Auxiliaire de Puériculture
Diplôme d'Etat d'Aide-soignant
CAP Accompagnant Educatif Petite Enfance
ÉDUCATIF ET SOCIAL
Diplôme d'Etat d'Educateur de Jeunes Enfants
Diplôme d'Etat d'Educateur Spécialisé
Diplôme d'Etat de Moniteur Educateur
Diplôme d'Etat d'Accompagnant Educatif et Social

Annexe 1 : Diplômes en FI.



Demande de fonctionnalité Moodle

Contexte : Moodle est utilisé au sein de l'IFRASS par les élèves/étudiants, inscrits à une formation, pour accéder aux divers compte-tenus de cours déposés par les formateurs.

Chaque formation a une organisation différente de l'accès aux ressources dans moodle.

Pour la formation des Auxiliaires de puériculture, Moodle est utilisé pour deux modalités d'enseignements :

- Enseignement en présentiel : dépôt des ressources par les formateurs avant ou après les cours avec paramétrage d'accessibilité.
- Enseignement en distanciel : ressource + Activités disponible sur moodle et accessible en fonction de planning de cours, l'accès est chronologique tout au long de l'année.

Les élèves sont dans l'obligation du suivi des cours. En présentiel, les élèves doivent émarger des feuilles de présence. En distanciel, les élèves ont plus de flexibilité dans l'organisation de leur travail quotidien, le contrôle du suivi des cours de fait par la réalisation des activités liées aux ressources.

En tant qu'organisme de formation, l'IFRASS doit pouvoir fournir à la région qui finance la formation la preuve que chaque élève a effectué ses heures de formations en fonction de son cursus.

Pour la promotion en présentiel, les feuilles d'émargement font foi. Pour la promotion en distanciel c'est l'achèvement des activités.

Besoin : A ce jour pour pouvoir contrôler l'achèvement des activités pour la promotion AP en distanciel, il faut accéder à chaque séquence de cours et relever à la main chaque achèvement pour chaque élève et le consigner dans un outil de suivi externe.

Le besoin est d'inclure dans moodle une fonctionnalité qui permettrait aux formateurs de suivre globalement l'achèvement des activités rattachée à une formation choisie pour une promotion donnée. Ces informations doivent pouvoir être extraite de Moodle pour être diffusées si besoin.

Les informations nécessaires sont : nom et prénom de l'étudiant, nom de l'activité, nom de la séquence de cours, bloc, module, date d'ouverture de l'activité, date d'achèvement de l'activité, pourcentage d'achèvement.

Cette fonctionnalité doit être accessible qu'au formateur et administrateur.

Annexe 2 : Besoin de l'organisme.

Remerciements :

J'aimerais dans un premier temps remercier **mon maître de stage**, Jérémie PILETTE, qui a pu m'aider en m'expliquant le fonctionnement du serveur et de Moodle 4. Il a également été très agréable et à l'écoute.

Je remercie également ma professeure **référente du stage**, Mme. CANUT, pour son amabilité et son aide, et ma professeure de **communication**, Mme. LEGRAND, pour ses conseils en communication orale et écrite, sans quoi ce rapport n'aurait pas la forme qu'il a aujourd'hui.

Enfin, j'aimerais remercier toute l'équipe de **l'IFRASS** pour leur accueil et leur bienveillance.

Résumé :

Durant 8 semaines, nous avons travaillé sur le développement intégral d'un plugin Moodle 4. Notre client, l'IFRASS, nous a contacté pour ce projet, afin de simplifier et d'améliorer le suivi de travaux des étudiants en distanciel. Nous nous sommes donc organisés afin de rendre un travail de qualité en temps et en heure. Nous avons utilisé les langages PHP, JavaScript, CSS et SQL. Notre base de données était en MariaDB, que nous avons visualisée sur HeidiSQL, et nous avons hébergé le plugin sur le serveur Debian existant de l'école. Ce projet a été une réussite, et a eu plusieurs impacts vis-à-vis de l'IFRASS.

Rapport de stage

Développement d'un plug-in Moodle

ETUDIANTS
MOODLE
SQL **PHP** IFRASS
UBUNTU
JAVASCRIPT

esteban.biret@gmail.com

Le 08 juin 2023



A destination de :

MME. CANUT

M. PILETTE

Rédigé par :

ESTEBAN BIRET-TOSCANO