Projet professionnel

Développeur web et web mobile NIVEAU III

Pierre de Gail

KoLABSoN

SOMMAIRE

Présentation du projet

I. Présentation

Passionné de composition musicale et souvent isolé dans mon petit home studio, l'idée de créer une plate-forme collaborative gratuite de composition musicale, mettant en contact les compositeurs amateurs de tous horizons, a germé.

Permettre le partage de samples (échantillons), de techniques ou de présenter ses compositions grâce à une interface simple et intuitive.

Écouter, échanger, commenter, collaborer, **KoLABSoN** se présente comme un laboratoire collaboratif pour les compositeurs amateurs.

II. Objectifs

KoLABSoN offre ainsi la possibilité de :

- Mettre en ligne ses propres créations, comme des morceaux complets ou simplement des samples réutilisables par les autres utilisateurs,
- De télécharger des échantillons (samples) pour les intégrer à leurs propres compositions (tracks),
- D'écouter en ligne les morceaux téléversés,
- De rédiger des commentaires sur chaque fichier, sample ou morceau,
- Faciliter l'interaction entre utilisateur en intégrant un service de chat inter-utilisateurs permettant le dialogue en direct.

Avec ses fonctionnalités **KoLABSoN** offre une vitrine interactive pour les compositeurs amateurs désireux de sortir de l'ombre et de se confronter aux avis, conseils ou critiques des autres compositeurs, dans le but de favoriser les contacts et le développement des capacités de chacun.

Développement Front-end

I. Arborescence du site

KoLABSoN se compose, d'une page d'accueil comportant :

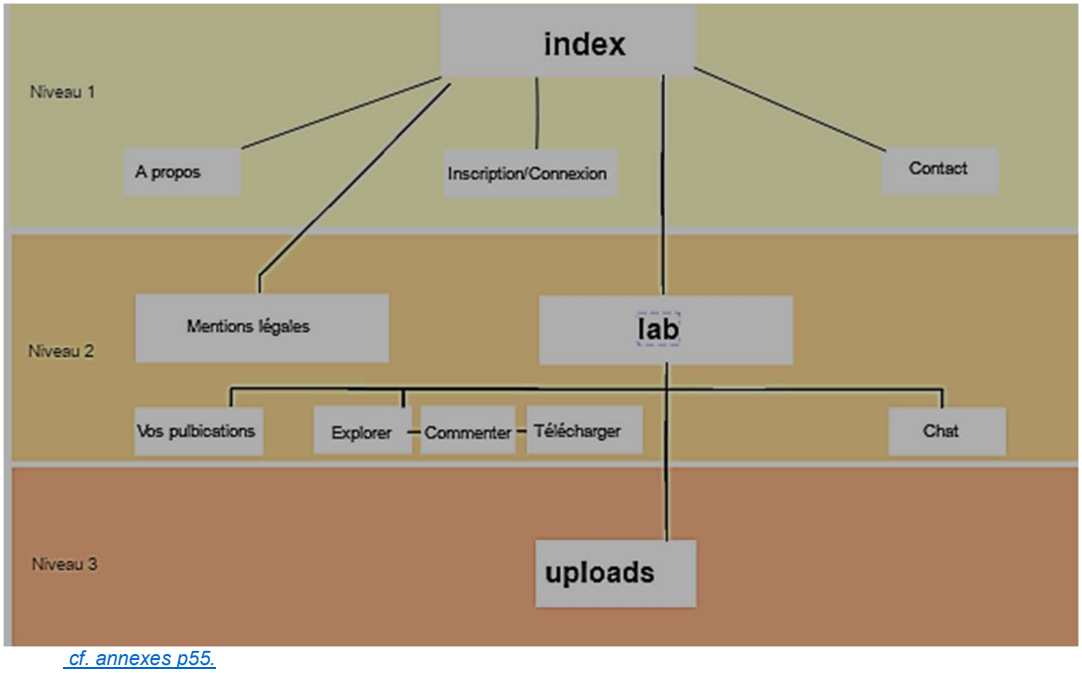
- Les informations sur le site dans la zone *« A PROPOS »*
- Les formulaires d'inscription et de connexion dans la zone *« INSCRIPTION/CONNEXION »*
- D'un formulaire de contact dans la zone *« CONTACT »*
- Les mentions légales et la politique de confidentialité accessibles depuis le footer.

Une fois l'utilisateur inscrit, il est redirigé vers **LE LAB** qui comporte :

- Une section « VOS PUBLICATIONS » dédiée à l'utilisateur et qui regroupe l'ensemble de ses fichiers télé versés sur le site.
- Une section pour le « CHAT » inter-utilisateurs.
- Une section « EXPLORER » ou paraîtrons l'ensemble des fichiers télé versés par les utilisateurs, découpée en deux parties, une pour les échantillons (samples) et l'autre pour les morceaux (tracks).
- Un **bouton** « UPLOADER VOS CREATIONS » qui amène l'utilisateur vers la page d'UPLOAD ou il pourra télé verser ses créations.
- Les **commentaires** se feront dans un formulaire présent sous chacun des éléments et s'afficheront en dessous.

Arrivé sur la page « UPLOAD » l'utilisateur dispose d'un formulaire permettant la télé versement de ses morceaux ou de ses échantillons.

Tous les formulaires présents sur KoLABSoN sont protégés des spam par des validations « CAPTCHA » intégrés.



cf. annexes p55.

II. Maquettage de l'application

A. *Charte graphique, ergonomique & responsable*

Dans un souci de respect des normes d'accessibilités j'ai opté pour un design épuré visant à garantir une navigation simple et intuitive.

La conception prend en compte certains concepts écologiques en rendant l'interface assez minimaliste pour limiter le trafic de données (images lourdes, vidéos, couleurs nombreuses, scripts, …), excepté sur la page d'accueil, de manière à garder un côté attractif et moderne.

En effet internet, en général, a un énorme impact écologique et KoLABSoN se veut de suivre le référentiel de recommandations, éditées par le GREEN IT, en limitant volontairement le nombre de données liées à l'esthétisme du site.

L'interface est totalement responsive de façon à être supportée par les périphériques mobiles et assurer une navigation optimale sur tous supports.

Pour l'ergonomie, les zone de saisie utilisateurs sont conçues de manière à faciliter leur usage (champs de saisie large et espacé, boutons larges et mis en évidence, affichage aéré, …) et la « *thumbnail navigation* » (navigation par le pouce)

Pour ne pas perturber les utilisateurs porteurs de déficiences visuelles, les couleurs se limiteront à un contraste noir et blanc. Le fond sera sur des tons blancs ou grisés et les polices de caractères noirs au format "*Helvetica Neue", sans-serif.*

KoLABSoN bénéficie également du certificat de la W3C.

Seuls les éléments essentiels seront mis en évidence par des encadrés vert (formulaires d'inscription) et rouge (formulaires de connexion).

(cf. Cahier des charges p.56)

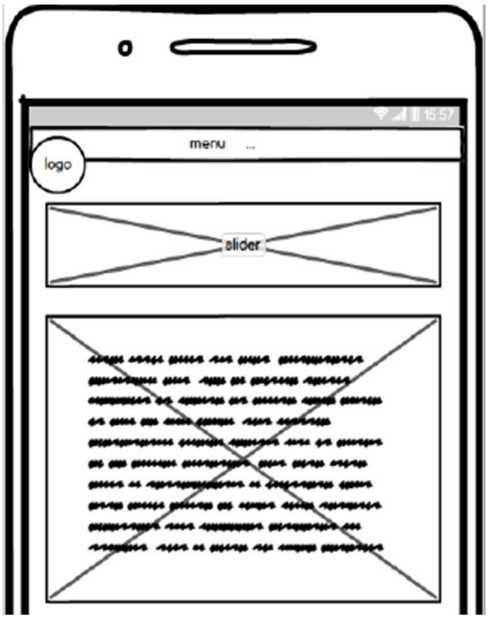
B. Wireframing

Maintenant que j'ai la vision du plan de site et de son approche graphique, je peux commencer le wireframe.

J'ai d'abord commencé par dessiner des esquisses sur papier pour me donner des idées de mise en page, compatibles avec la charte graphique (cf. page 58).

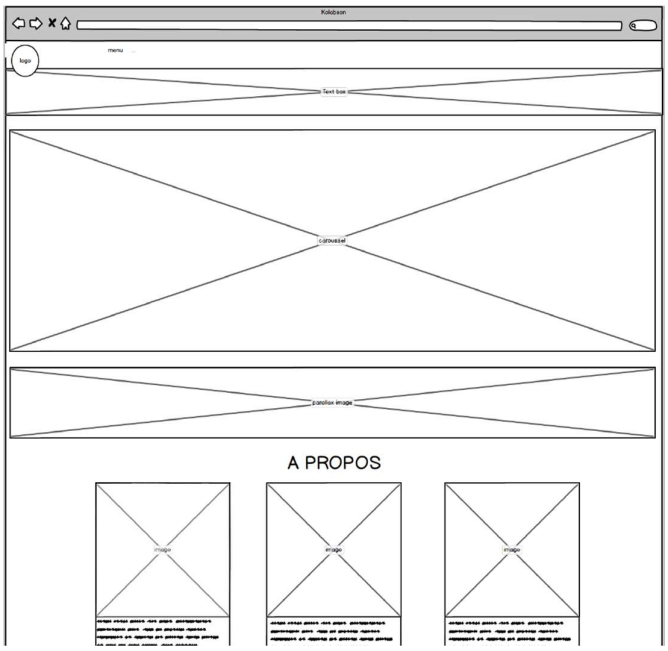
Afin de réaliser une interface responsive j'ai commencé à travailler sur la version mobile.

Pour favoriser l'ergonomie, je suis parti sur un affichage simple composé d'une succession de blocs. Chacune des différentes sections du site seront clairement séparées bar des bannières de façon claire et esthétique.



La page est découpée en trois colonnes pour les écrans plus larges et se réduit à une seule colonne sur la version mobile.

A l'aide de mockups (Pencil, Balsamiq) j'ai donc crée les wireframes des différents pages. (cf. annexe p.51).



C. Maquette graphique

Une fois que les wireframes ont été validés, j'ai conçu le logo ainsi que les icônes du site avec un logiciel de création graphique(Photoshop) et retouché les images à intégrer sur le site.

Puis j'ai créé des maquettes graphiques intégrants les images et couleurs du site en lien avec les souhaits de la charte graphique.

Icônes et logo de KoLABSoN

III. Implémentation de l'interface utilisateurs

A. Choix techniques

Le squelette de l'application est développé en ***HTML 5***, la forme en ***CSS 3*** et les effets, animations en ***JavaScript*** et ***JQuery***.

Pour rendre l'application responsive j'ai opté pour ***Bootstrap*** et les règles ***max*** et ***min-width*** en *CSS3* pour les éléments ne faisant pas appel à *Bootstrap.*

Pour un design plus attractif j'ai choisi d'intégrer des bannières d'images dotées d'un effet parallaxe (*Jquery*) qui délimiteront chaque section de la page pour une meilleure lisibilité et une animation textuelle pour accueillir les visiteurs.

Pour une approche éco-conceptuelle, le design se veut volontairement minimaliste de façon à réduire le transfert de données entre le serveur et le client. Cette démarche améliorera aussi la vitesse de chargement des pages et donc la navigation de l'utilisateur.

B. Implémentation

1) Implémentation web statique adaptable

J'ai donc implémenter 3 pages au format HTML composées d'un header et footer commun comportant respectivement le menu et les liens vers les Mentions légales et la politique de confidentialité de KoLABSoN respectant l'architecture classique d'une page HTML déclaré en UTF-8, le chargement des feuilles de styles et des scripts nécessaire au fonctionnement des composants *Bootstrap,* de l'effet parallaxe et de l'animation d'accueil ainsi que les formulaires.

Pour répondre au besoin adaptatif, j'ai donc utilisé les systèmes de colonnes pour diviser mes pages :

Un affichage sur 3 colonnes sur les grands écrans, qui se réduit à une seule sur les mobiles. (Cf. annexe p.51)

```
<div class="row form-group product-chooser">
    <div class="col-xs-12 col-sm-12 col-md-4 col-lg-4">
        <div class="product-chooser-item selected">
            <div id="icoApropos" class="img-rounded col-xs-4 col-sm-4 col-
md-12 col-lg-12">
```

En CSS j'ai utilisé les ***Media queries*** pour organiser les éléments selon les tailles d'écrans. La spécification CSS3 **Media Queries** définit les techniques pour l'application de feuilles de styles en fonction des périphériques de consultation utilisés pour du HTML. On nomme également cette pratique **Responsive Web Design**, pour dénoter qu'il s'agit d'adapter dynamiquement le design à l'aide de CSS.

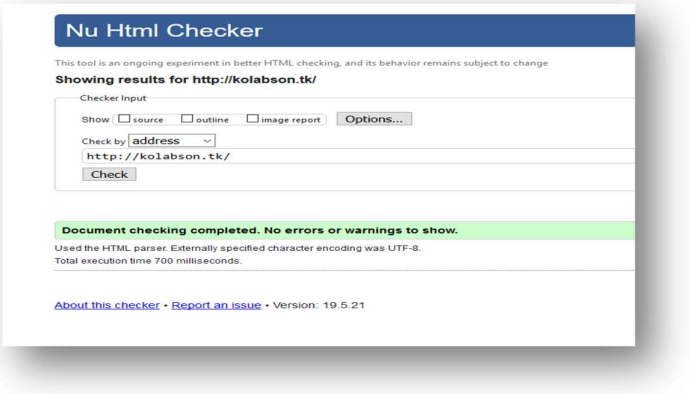
Dans l'exemple ci-dessous, on cible les écrans de largeur inférieure à 1250 pixels grâce à la règle ***max-width*** associée à la valeur ***1250px.*** On crée une media query avec la directive ***@media*** suivie du type d'écran et d'une ou plusieurs conditions (comme la largeur de l'écran). Le style ***CSS*** qui suit sera activé uniquement si les conditions sont remplies.

```
@media screen and (max-width: 1250px) {
    .container ul li {
        width: 40%;
        margin-left: 50px;
    }
}
```

On peut également modifier la largeur **viewport**(la largeur de la fenêtre du navigateur sur le mobile)**.** Pour s'adapter, les navigateurs mobiles affichent le site en « dézoomant », ce qui permet d'avoir un aperçu de l'ensemble de la page.

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

Par ailleurs, le code a reçu la validation de la W3C (*https://validator.w3.org/*).



a) La page d'accueil : index

L'index se compose :

- Du header,
- Une div pour l'animation d'accueil,
- Trois sections, séparées par trois bannières avec effet parallaxe, regroupant les informations,
- Ainsi que les formulaires d'inscription et de connexion et de contact,
- Et enfin du footer.

Le **heade**r comporte :

- *<!DOCTYPE html>,* la déclaration du type de fichier,

- La balise *<html> lang= 'fr'* qui englobe le contenu de la page et la langue utilisée,
- L'entête *<head>* donne des informations sur la page :
    - *<meta charset>* pour son encodage (pour les caractères spéciaux),
    - *<meta name= 'viewport'>* pour l'affichage de la page sur versions mobiles,
    - *<script></script>* fait appel aux scripts nécessaires à l'exécution de certains éléments de la page (cf. implémentation web dynamique),
    - *<link></link>* pour l'appel aux fichiers *Boostrap* et *CSS* employés pour la mise en forme de la page,
    - *<title></title>* pour le titre de la page.

```
<!doctype html>
<html lang="fr">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <script
src="https://www.google.com/recaptcha/api.js?render=6LdD2qUUAAAAADzDhVij6lx
ol-VtwKzdv5UKO7ty"></script>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
    <link rel="stylesheet" href="style.css">
    <title>KolabSon</title>
</head>
```

- *<body></body>*, le contenu visible de la page : le corps de page qui comprend :
    - Le **menu de navigation** :

      Son implémentation est commune à toutes les pages du site. Il intègre le logo et les liens vers les différentes sections de la page.

```
<nav>
    <div class="menu-icon">
        <img src="https://img.icons8.com/ios/20/000000/down-squared.png"
alt="icone menu" id="iconeMenu">
        <i class="fa fa-bars fa-2x"></i>
    </div>
    <div class="logo" id="logoHeader">
        <a href="index.php"><img src="images/logoMin.png" alt="logo"></a>
    </div>
    <div class="menu">
        <ul>
            <?php
            if (isset($_SESSION['connecte']) && $_SESSION['connecte'] ==
true) {

                echo '<li><a href="lab.php">LAB</a></li>';

            } else {
                echo 'probleme';
            }
            ?>
            <li><a href="index.php">ACCUEIL</a></li>
            <li><a href="#ancre">A PROPOS</a></li>
            <li><a href="#ancre2">INSCRIPTION / CONNEXION</a></li>
            <li><a href="#ancre3">CONTACT</a></li>
        </ul>
    </div>
</nav>
```

- La zone ou s'affichera **l'animation textuelle d'accueil** :

  Il s'agit d'un conteneur vide dédié à l'affichage du script 'animation.index.js'.

```
<div class="jumbotron jumbotron-fluid" id="textBox">
    <div class="container">
        <h1 class="heading" data-target-resolver></h1>
    </div>
</div>
```

Figure 0-1 Implémentation Html Animation textuelle

- Les **bannières séparatrices** identiques pour compartimenter la page

```
<div class="ha-bg-parallax text-center block-marginb-none" data-
type="background" data-speed="20" id="parallaxBlock1">
    <div class="ha-parallax-body">
        <div class="ha-heading-parallax">"Keep On Creating!" KoLABSoN
TEAM</div>
    </div>
</div>
```

Figure 0-2 Implémentation Html des séparateurs de sections

- La zone '**A PROPOS**' avec les informations générales du site : son but, ses fonctionnalités et un lien vers le formulaire de contact. Il est divisé en trois

colonnes pour une meilleure lisibilité et s'affiche sur une seule colonne sur mobile.

```
<h2>A PROPOS</h2>
<br>
<br>
<br>
<div class="row form-group product-chooser">
    <div class="col-xs-12 col-sm-12 col-md-4 col-lg-4">
        <div class="product-chooser-item selected">
            <div id="icoApropos" class="img-rounded col-xs-4 col-sm-4 col-
md-12 col-lg-12">
                <img src="images/icoLabMin.png" alt="icone du Lab">
                <img src="images/iconmonstr-user-20-72.png">
                <img src="images/icoChatMin.jpg">
                <img src="images/icoUpload.png">
            </div>
            <div class="col-xs-8 col-sm-8 col-md-12 col-lg-12">
                <span class="title">ECHANGE</span>
                <span class="description" id="pApropos1">Le but principal
de ce site est de favoriser les contacts entre les créateurs que nous
sommes! Echanger, commenter ou tout simplement faire découvrir vos
créations, KoLABSoN vous offre la possibilité de pouvoir vous mettre en
lumière sans pour autant quitter votre caverne !</span>
            </div>
            <div class="clear"></div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-4 col-lg-4">
        <div class="product-chooser-item selected">
            <img src="images/speaker3.jpg" class="img-rounded col-xs-4 col-
sm-4 col-md-12 col-lg-12"  alt="image d'illustration">
            <div class="col-xs-8 col-sm-8 col-md-12 col-lg-12">
                <span class="title">KOLABSON</span>
                <span class="description" id="pApropos">L'histoire commence
dans un modeste Home-Studio. Ma caverne. Me vient alors l'idée de créer une
plate-forme ou tout les hermites, créateurs de musique, pourraient se
retrouver pour pouvoir échanger leurs savoirs. Ainsi est né KoLABSoN.
(Kolaborativ Labs On)  </span>
            </div>
            <div class="clear"></div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-4 col-lg-4">
        <div class="product-chooser-item selected">
            <img src="images/headphones.gif" class="img-rounded col-xs-4
col-sm-4 col-md-12 col-lg-12" alt="image d'illustration">
            <div class="col-xs-8 col-sm-8 col-md-12 col-lg-12">
                <span class="title">PARTAGE</span>
                <span class="description" id="pApropos2">Sur KoLABSoN, vous
pouvez écouter, discuter, uploader vos créations (samples ou morceaux
complets) ou télécharger des samples pour les integrer à vos propres
créations! Par souci de protection de vos compositions seul les samples
peuvent être téléchargés. </span>
                <span class="description">Bienvenue dans KoLABSoN ! !   !
<a href="index.php#ancre3">Nous contacter</a> </span>
            </div>
            <div class="clear"></div>
        </div>
    </div>
</div>
```

Viennent ensuite les formulaires qui sont protégés par Google reCAPTCHA pour éviter les spam bots.

Ayant opté pour la version 3 de Google reCAPTCHA, seul un encadré dans le coin inférieur droit de la page est visible. Il indique aux utilisateurs l'utilisation de celui-ci sur la page et la politique de confidentialité ainsi que les conditions.

- Les **formulaires d'inscription et de connexion** :

  Ils sont, conformément au cahier des charges, mis en évidence par des couleurs.

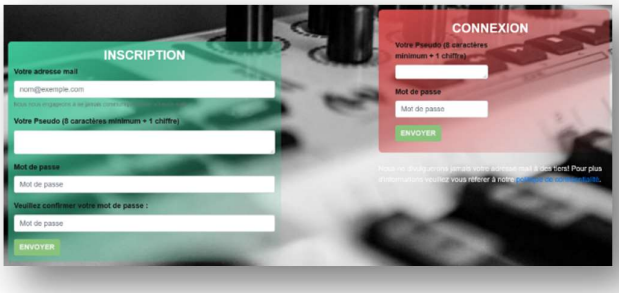
  Tous les champs bénéficient d'aide à la saisie.

```
<div class="container" id="connexionDroit">
    <div class="row" id="connexionForm">
        <h3 id="titreConnex">CONNEXION</h3>
        <!--<div class="col-xs-12 col-sm-12 col-md-6 col-lg-6">-->
        <form action="connexion.php" method="post">
            <div class="form-group">
                <label for="exampleFormControlTextarea1">Votre Pseudo (8
caractères minimum + 1 chiffre)</label>
                <textarea class="form-control"
id="exampleFormControlTextarea2" rows="1" name="pseudoCo"></textarea>
            </div>
            <div class="form-group">
                <label for="exampleInputPassword1">Mot de passe</label>
                <input type="password" class="form-control"
id="exampleInputPassword3" placeholder="Mot de passe" name="mdp">
            </div>
            <input type="hidden" name="tokenGoogle" class="token">
            <button type="submit" class="btn btn-primary"
id="btnEnvoyerCo">ENVOYER</button>
        </form>
        <!--</div>-->
    </div>
    <br>
    <div class="row">
        <span class="description" id="infoForm">Nous ne divulguerons jamais
votre adresse mail à des tiers!  Pour plus d'informations veuillez vous
référer à notre <a href="">politique de confidentialité</a>. </span>
    </div>
</div>
```

Formulaire de connexion

```
<div class="container" id="formulaire">
    <h3 id="titreInsc">INSCRIPTION</h3>
    <form action="inscription.php" method="post">
        <div class="form-group">
            <label for="exampleFormControlInput1">Votre adresse
mail</label>
            <input type="email" class="form-control"
id="exampleFormControlInput1" placeholder="nom@exemple.com" name="mail">
            <small id="emailHelp1" class="form-text text-muted">Nous nous
engageons à ne jamais communiquer votre adresse mail.</small>
        </div>
        <div class="form-group">
            <label for="exampleFormControlTextarea1">Votre Pseudo (8
caractères minimum + 1 chiffre)</label>
            <textarea class="form-control" id="exampleFormControlTextarea1"
rows="1" name="pseudo"></textarea>
        </div>
        <div class="form-group">
            <label for="exampleInputPassword1">Mot de passe</label>
            <input type="password" class="form-control"
id="exampleInputPassword1" placeholder="Mot de passe" name="mdp1">
        </div>
        <div class="form-group">
            <label for="exampleInputPassword1">Veuillez confirmer votre mot
de passe :</label>
            <input type="password" class="form-control"
id="exampleInputPassword2" placeholder="Mot de passe" name="mdp2">
        </div>
        <div class="g-recaptcha" data-sitekey="6LdD2qUUAAAAADzDhVij6lxol-
VtwKzdv5UKO7ty"> </div>

        <input type="hidden" class="token" name="tokenGoogle">
        <button type="submit" class="btn btn-primary"
id="btnEnvoye">ENVOYER</button>
    </form>
</div>
```

Formulaire d'inscription



- Le **formulaire de contact** de KoLABSoN

  Il permet de contacter l'administrateur du site sans avoir besoin d'être inscrit préalablement. L'utilisateur doit simplement renseigner son adresse mail et rédiger son message. Il bénéficie d'aide à la saisie. Le mail est expédié à l'adresse de contact de KoLABSoN : contact.kolabson@gmail.com. Cette fonctionnalité est développée en back end avec PHPMAILER (cf. p .40)

```
<div class="jumbotron jumbotron-fluid" id="blockForm">
   <div class="container">
       <form id="formContact" action="mail.php" method="post">
           <div class="form-group">
               <label for="exampleInputEmail1">Votre adresse mail:</label>
               <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp" placeholder="Enter email" name="expedMail">
               <small id="emailHelp" class="form-text text-muted">Nous nous engageons à ne
jamais communiquer votre adresse mail.</small>
           </div>
<br>
<br>
           <div class="form-group">
               <label for="exampleFormControlTextarea1">Example textarea</label>
               <textarea class="form-control" id="exampleFormControlTextarea3" name="textMail"
rows="4" placeholder="Votre message"></textarea>
           </div>
<br>
<br>
<!-- RECAPTCHA  -->
           <input type="hidden" name="tokenGoogle" class="token">
<br>
<br>
<br>
<br>
           <input type="submit" class="btn btn-primary" id="btnEnvoieMail">Submit</input>
       </form>
     </div>
 </div>
```

- Le '**footer**', le pied de page :

  Qui contient les liens vers l'accueil du site, les mentions légales et la politique de confidentialité.

  Il est commun à toutes les pages du site.

```
<!-- Footer -->
<section id="footer">
    <div class="container">
        <div class="row text-center text-xs-center text-sm-left text-md-
left">

            <div class="col-xs-12 col-sm-4 col-md-4">
                <h5>KoLABSoN</h5>
            </div>

            <div class="col-xs-12 col-sm-4 col-md-4">
                <h5>MENTIONS LEGALES</h5>
            </div>

            <div class="col-xs-12 col-sm-4 col-md-4">
                <h5>RGPD</h5>
            </div>
        </div>
        <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12 mt-2 mt-sm-2 text-
center text-white">
                <p><u><a
href="https://www.nationaltransaction.com/">National Transaction
Corporation</a></u> is a Registered MSP/ISO of Elavon, Inc. Georgia [a
wholly owned subsidiary of U.S. Bancorp, Minneapolis, MN]</p>
                <p class="h6"> All right Reversed.<a class="text-green ml-
2" href="https://www.sunlimetech.com" target="_blank">KoLABSoN</a></p>
            </div>
        </div>
    </div>
</section>
```

- Avant les balises fermantes </body> et </html> se trouve les appels aux scripts nécessaires à l'exécution de Bootstrap, des animations (parallaxe, animation textuelle, effet de scroll) et de la communication avec Google pour la protection reCaptcha.

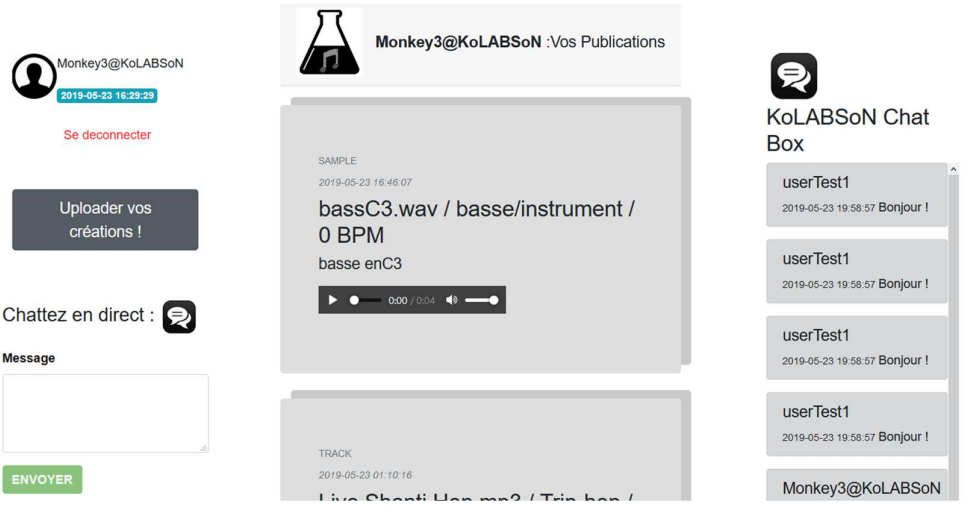
b) La page 'laboratoire' : LE LAB

Le LAB reprend la même structure Html pour le header, le menu et le footer. Le menu a été enrichi avec de nouveaux liens vers chacune des fonctionnalités du site.

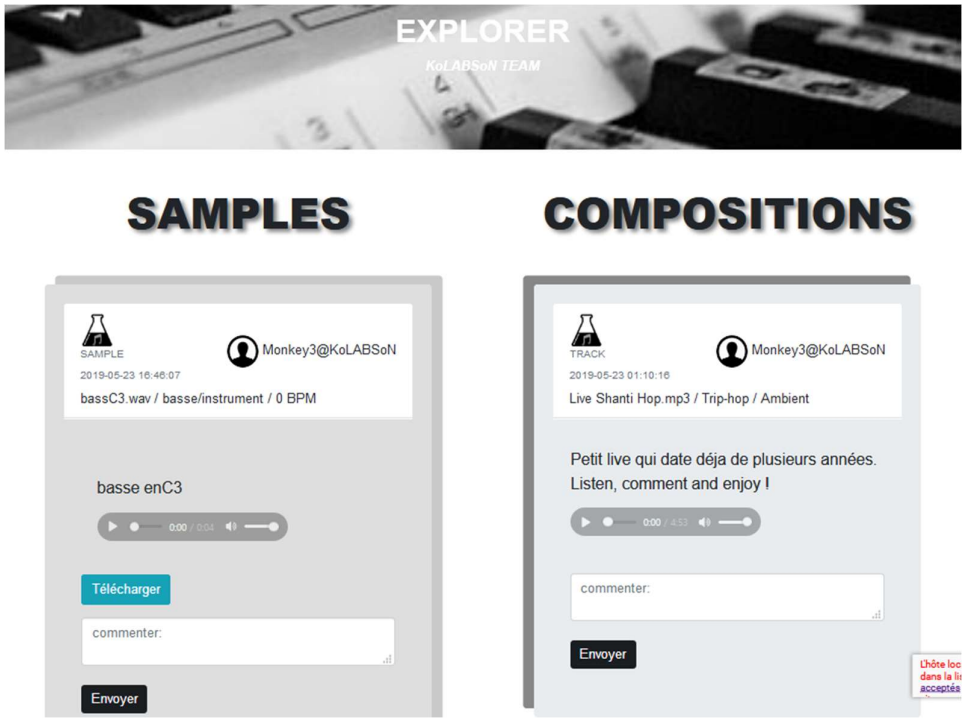
Il est composé de deux sections :

- L'espace utilisateur ou s'affiche l'ensemble de ses publications, le chat inter-utilisateurs et un lien vers la page dédiée à l'envoie de fichiers.

```
<div class="container" id="profilBox">
    <div class="row">
        <div class="span4 well">
            <div class="row">
                <div class="span1"><img src="images/iconmonstr-user-20-
72.png"
                                        alt="icone utilisateur"></div>
                <div class="span3">
                    <p><strong><?php echo $_SESSION['pseudo'];
?></strong></p>
                    <span class=" badge badge-info"><?php echo
$_SESSION['dateInscription'];?></span>
                </div>
            </div>
            <br>
            <button type="button" class="dropdown-item header" data-
toggle="modal" data-target="#modalDeconnexion" ><a id="lienDeconnexion"
href="deconnexion.php" >Se deconnecter</a></button>
        </div>
    </div>
</div>
<div class="container">
    <a href="uploads.php" class="btn btn-secondary btn-lg active"
role="button"
       aria-pressed="true" >Uploader vos créations !</a>
</div>
```

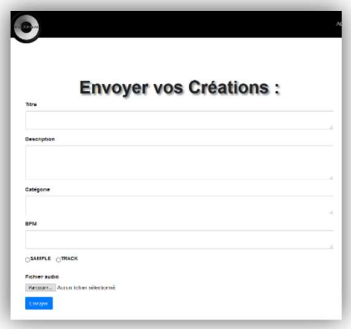


- La section ‘EXPLORER’ ou s’affiche toutes les publications postées sur le site. Elle est divisée en deux parties : une pour les échantillons (samples) et l’autre pour les morceaux (tracks). Les commentaires se rédigent et s’affichent sous chaque publication. Sous chaque sample il y a un bouton pour télécharger le fichier.



c) *La page de téléversement de fichiers : Uploads*

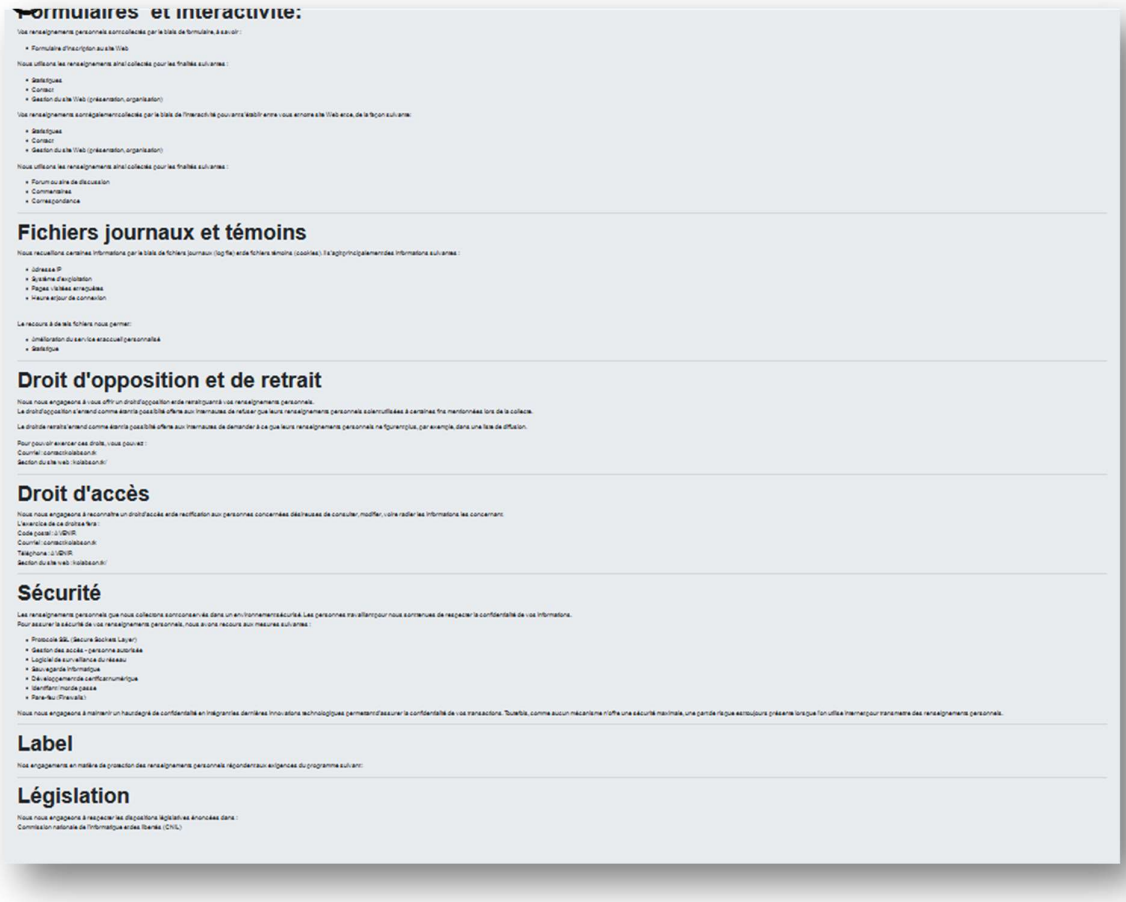
C'est une page minimaliste est dédiée au téléversement et intègre le header, le footer et le formulaire. Ici sont collectés, le titre, une description, la catégorie, les BPM (Battements Par Minute) et s'il s'agit d'un échantillon ou d'un morceau.



d) *Respect des nouvelles lois RGPD*

Conformément aux nouvelles lois éditées par la CNIL (Commission Nationale de l'Informatique et des Libertés), l'utilisateur est informé par un encart visible de l'utilisation de cookies sur le site et les mentions légales et politiques de confidentialités sont consultables depuis les liens en pied de page.

Les cookies utilisés le sont par le Google reCAPTCHA et il est prévu, dans une prochaine version, d'utiliser d'avantage les cookies.



2) Implémentation web statique avec JavaScript

1) L'animation textuelle d'accueil

Cette animation est développée en HTML, CSS et Babel.js. Babel.js est un compilateur JavaScript gratuit et à code source ouvert.

J'ai utilisé le code trouvé sur codepen.io (Futuristic Resolving/Typing Text Effect) et je l'ai adapté pour KoLABSoN.

Ce script appel une fonction nommée *« resolve »* qui prend en paramètres *« options »* et *« callback »*.

```
resolve: function resolve(options, callback) {
    // The string to resolve
    const resolveString = options.resolveString ||
options.element.getAttribute('data-target-resolver');
    const combinedOptions = Object.assign({}, options, {resolveString:
resolveString});
```

Ensuite elle génère aléatoirement des caractères avec les fonctions *« randomCharacters »* et *« getRandomInteger »*

```
function getRandomInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
};

function randomCharacter(characters) {
    return characters[getRandomInteger(0, characters.length - 1)];
};
```

Une fois que la fonction à finie une chaine (string), elle fait un appel à la fonction « callback » et reboucle sur la chaine suivante.

```
// Callback function when resolve completes
function callback() {
    setTimeout(() => {
        counter ++;
        if (counter >= strings.length) {
            document.getElementById('textBox').style.visibility='hidden';//
Cache l'animation une fois le compteur arrivé à la fin
            $textBoxIntro = document.getElementById('textBox');
//Emplacement ou afficher l'animation
        }
        else {
            let nextOptions = Object.assign({}, options, {resolveString:
strings[counter]});
            resolver.resolve(nextOptions, callback);
        }
    }, 1000);
}
```

J'ai donc modifié les chaines de caractères de façon à afficher le message souhaité, et sélectionné l'emplacement dans le code Html avec cette ligne de code :

```
$textBoxIntro = document.getElementById('textBox'); //Emplacement ou
afficher l'animation
```

J'ai également ajouté une ligne pour cacher l'animation une fois la boucle finie.

```
document.getElementById('textBox').style.visibility='hidden';// Cache
l'animation une fois le compteur arrivé à la fin
```

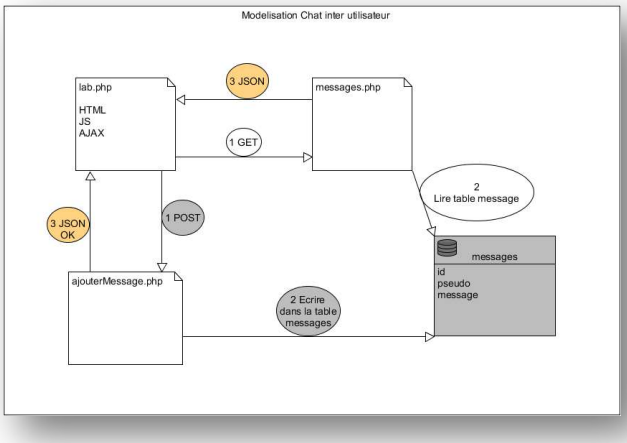
2) Le Chat inter utilisateurs : La Chat Box

Le chat inter utilisateurs est implémenté en JavaScript, AJAX et JSON. **AJAX** est l'acronyme d'*Asynchronous JavaScript and XML*, ce qui, transcrit en français, signifie « JavaScript et XML asynchrones ». **JSON** (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

```
<form>
    <div class="form-group">
        <input type="hidden" class="form-control"
id="formGroupExampleInputPseudo"
               value="<?php echo $_SESSION['pseudo']; ?>" name="pseudo">
    </div>
    <div class="form-group">
        <label for="formGroupExampleInput2">Message</label>
        <textarea class="form-control" id="exampleFormControlTextarea1"
rows="3"
                  name="message"></textarea>
    </div>
    <input type="hidden" name="tokenGoogle" class="token">
    <input type="submit" class="btn btn-primary" id="btnEnvoye"></input>
    <div class="g-recaptcha" data-
sitekey="6LeloaAUAAAAAGwsCQ2CIc7IvFYV8vGYA5-A7L60"> </div>
</form>
```

Le chat s'affiche sur la page du LAB. Les messages sont rédigés depuis le formulaire dédié et sont envoyé sur la page de traitement *ajouterMessage.php.*

Les messages sont ensuite récupérés dans la base de données et affichés sur la page avec le traitement effectué dans la page *message.php (cf : back end p :)*



A l'envoi du formulaire, on effectue une requête HTTP (« POST ») au serveur pour lui envoyer les données saisies. Les messages sont affichés à l'aide de requêtes HTTP asynchrones et ne mettent ainsi à jour seulement cette partie de la page.

```
//Méthode POST
    var reqPost = new XMLHttpRequest(); //Déclaration de la requête HTTP
    reqPost.open('POST','ajoutermessage.php'); // Cible de la requête
    reqPost.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
    reqPost.send('pseudo=' + champPseudo.value + '&message=' +
champMessage.value );
    e.preventDefault();
}
```

L'objet « *XMLHttpRequest* » permet de créer des requêtes HTTP en JavaScript. Sa méthode « open » permet de configurer la requête en prenant en paramètres le type de requête (GET ou POST généralement), l'URL cible ainsi qu'un booléen (qui indique si la requête est asynchrone ou non). Une requête **synchrone** bloque le programme appelant jusqu'à ce que la réponse soit disponible et la page web semblera bloquée et ne répondra plus aux actions de l'utilisateur.

J'ai utilisé le gestionnaire d'évènements pour effectuer une requête **asynchrone** : l'évènement de type *« load »* indique la fin du traitement de la requête par le serveur.

```
//Ajout du gestionnaire d'évenement
var formulaire = document.getElementsByTagName("form");
formulaire[0].addEventListener("submit", envoyerFormulaire);
```

Figure 0-3 Gestionnaire d'évènement

Pour gérer les erreurs j'ai utilisé une condition (*if { } else { }*). On distingue deux cas d'erreur :

- Le cas où la requête n'a pas réussi à atteindre le serveur (nom de serveur incorrect, erreur de réseau…) et déclenche l'apparition d'un message dans la console.

```
req.addEventListener("error", function () {
        console.error("Erreur réseau avec l'URL :" + url);
    });
    req.send(null);
}
```

- Le cas où la requête a atteint le serveur mais son traitement a échoué (ressource non trouvée, problème de serveur…). C'est le code de retour HTTP qui indique son résultat (« status »). Un code supérieur ou égal à 200 et strictement inférieur à 400 signale la réussite de la requête.

```
req.addEventListener("load", function () {
    if (req.status >= 200 && req.status < 400) {          // Appelle la
fonction callback en lui passant la réponse de la requête
        callback(req.responseText);
    } else {
        console.error(req.status + " " + req.statusText + " " + url);
    }
});
```

La gestion du format JSON par JavaScript se fait par la fonction « JSON.parse ». Elle permet de transformer une chaine de caractères en un objet JavaScript et donc de gérer les tableaux d'objets JSON.

J'ai utilisé la fonction générique « *ajaxGet* » dans ma fonction nommé « *afficherReponse* » :

```
function afficher(reponse)
{
    console.log(reponse);
    var messages = JSON.parse(reponse);
    var conteneur = document.getElementById("messages");      //declaration
de la div parent
    conteneur.innerHTML = "";
```

Ensuite j'ai créé une boucle pour parcourir le tableau renvoyé par la réponse JSON et afficher les données souhaitées à leurs emplacements.

```
    for (var i = 0; i < messages.length; i++) {
        var newDivElt = document.createElement("div");        //Déclaration
nouvel élement div
        newDivElt.classList.add("alert");
        newDivElt.classList.add("alert-dark");
        newDivElt.setAttribute("role", "alert");
        conteneur.appendChild(newDivElt);
//Ajout du nouvel enfant
        var pseudoElt = document.createElement("h5");         //Déclaration
du pseudo
        pseudoElt.textContent = "" + messages[i].pseudo;
        var dateElt = document.createElement("p.dateMessage");
        dateElt.textContent = "" + messages[i].dateMessage + "\n" ;
        var messageElt = document.createElement("p.messagePseudo");
//Déclaration du message
        messageElt.textContent = "" + messages[i].message;
        //Ajout des élements
        newDivElt.appendChild(pseudoElt);
        newDivElt.appendChild(dateElt);
        newDivElt.appendChild(messageElt);
    }
}
```

Figure 0-4 Boucle Javascript pour afficher la réponse JSON

J'accède aux DOM (Document Object Model) via « *document*. » et je crée une « div » avec *« document.createElement('div') »*, j'ajoute une classe pour la mise en forme en récuperant les classes *« alert »* de Bootstrap, je leur ajoute des attributs avec *« " »*et je leur créé de nouveaux enfants avec *« newDivElt.appendChild ».*

Ensuite je déclare l'élement *« pseudo »* au sein d'un titre de niveau 5 (h5) avec *« var pseudoElt = document.createElement(''h5") ».*

Dans le fichier *message.php* les informations sont transmises en JSON :

```
header('Content-Type: application/json');
```

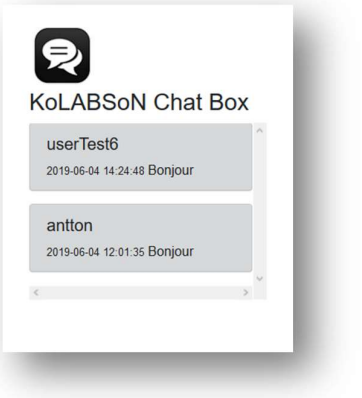


Figure 0-5 Chat Box

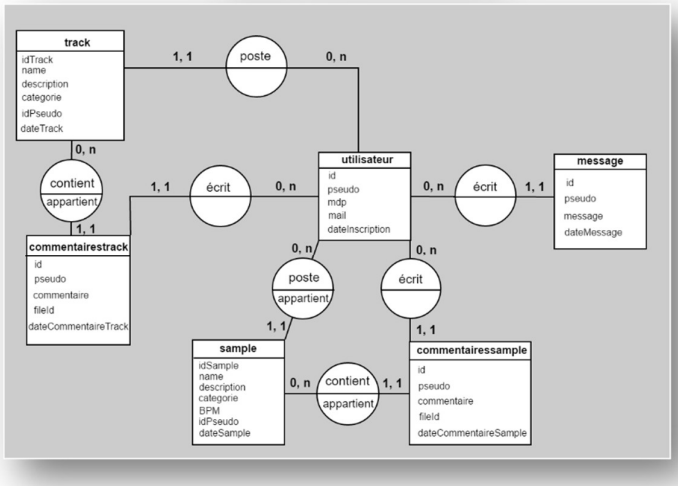
(Le traitement en back end de la Chat Box sera abordé p. 27)

Développement Back-end

I. Base de données

Pour élaborer la base de données de KoLABSoN, j'ai commencé par lister toutes les données nécessaires ainsi que leurs types. Une fois le dictionnaire de données terminé, j'ai conçu le MCD (Modèle Conceptuel de Données) selon la méthode MERISE (Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprises), ou chaque entité correspond à une table, chaque propriété devient une colonne de la table, chaque occurrence de l'entité devient une ligne de la table et l'identifiant de l'entité est la clé primaire.

Modèle Conceptuel de Données



Sur phpMyAdmin j'ai créé les tables à l'aide des commandes SQL :

```
CREATE TABLE utilisateurs (id INT, pseudo VARCHAR, mdp VARCHAR, mail VARCHAR,
dateInscription DATE);

CREATE TABLE messages (id INT, pseudo VARCHAR, message VARCHAR, dateMessage DATE);

CREATE TABLE samples (idSample INT, name VARCHAR, description VARCHAR, catégorie
VARCHAR, BPM INT, idPseudo INT, dateSample DATE);

CREATE TABLE tracks (idTrack INT, name VARCHAR, description VARCHAR, categorie
VARCHAR, idPseudo INT, dateTrack DATE);

CREATE TABLE commentairesSample (id INT, pseudo VARCHAR, commentaire VARCHAR,
dateCommentaireSamples DATE) ;

CREATE TABLE commentairesTracks (id INT, pseudo VARCHAR, commentaire VARCHAR,
dateCommentaireTrack DATE);
```

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	**id**	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	**pseudo**	varchar(25)	utf8_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
3	**commentaire**	varchar(250)	utf8_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
4	**fileID**	int(11)			Non	*Aucun(e)*			Modifier Supprimer Plus
5	**dateCommentaireSample**	datetime			Non	*Aucun(e)*			Modifier Supprimer Plus

Figure 0-1PhpMyAdmin - table commentairesSample

J'ai défini les id en clés primaires (Primary Key), les idPseudo et fileID en clés étrangères (Foreign Key).

```
ALTER TABLE utilisateurs ADD PRIMARY KEY(id);

ALTER TABLE utilisateurs ADD FOREIGN KEY(fileId);
```

On peut désormais communiquer avec la base en utilisant les commandes SQL.

II. Implémentation web-dynamique : le chat inter-utilisateurs

Le traitement des messages saisis par l'utilisateur est réalisé dans la page *ajoutermessages.php*. Nous avons vu (page : 21) le fonctionnement de la requête AJAX et de sa réponse en JSON.

Nous allons maintenant voir le traitement en back end des données.

Une fois que l'utilisateur a saisi son message et cliqué sur le bouton *envoyer*, les données sont envoyées par une requête POST à la page de traitement *ajoutermessage.php* qui va les enregistrer dans la base de données.

D'abord on vérifie qu'il existe bien une valeur dans la variable *$_POST :*

```
if (isset($_POST['pseudo']) && isset($_POST['message']) )//Si les champs pseudo et message existent
```

Si la condition est remplie on se connecte à la base de données avec le *« include 'connectBDD.php' ; »* qui fait appel au code contenu dans le fichier *'connectBDD.php' :*

```
try {
    $bdd = new PDO('mysql:host=localhost;dbname=kolabson;charset=utf8', '****', '******');
    $bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $bdd->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
} catch (Exception $e) {
    die('Erreur : ' . $e->getMessage());
}
```

J'utilise la méthode PDO pour l'accès à la base de données et je renseigne le nom de l'hôte (*localhost)*, la base (*kolabson),* l'encodage (*utf-8)* ainsi que l'identifiant et le

mot de passe de connexion. Le premier paramètre (*mysql :)* est le Data Source Name, le type de base de données.

Les lignes contenant les « *setAttribute »* sont activées uniquement dans l'environnement de développement et servent à afficher les erreurs.

Ensuite il faut tester la présence d'erreurs avec les lignes suivantes :

```
catch (Exception $e) {
   die('Erreur : ' . $e->getMessage());
}
```

Php essaie d'exécuter les instructions contenues dans le bloc « *try »* et s'il y a une erreur il rentre dans le bloc « *catch* » et arrête l'exécution de la page en affichant un message d'erreur.

Une fois la connexion à la base validée et activée, on passe à **l'écriture des données** dans la base, à l'aide d'une requête préparée de façon à ce qu'elle puisse contenir une ou plusieurs variables :

```
$req = $bdd->prepare('INSERT INTO messages (id, pseudo, message, dateMessage) VALUES(NULL, :pseudo, :message, :dateMessage)');
```

On insère dans la table *messages* les paramètres *id, pseudo, message* et *dateMessage* prenant respectivement comme valeur les données contenues dans la variable *$_POST (VALUES)*

La requête est alors exécutée avec les paramètres, sous forme de tableau (*array*), contenus dans *:pseudo, :message* et *:dateMessage* . Ils font référence aux noms des champs du formulaire (*name="pseudo"* et *name="message*").

Pour récupérer ces valeurs j'utilise *$_POST['pseudo']* et *$_POST['message'].*

Pour éviter les failles de sécurité, j'utilise la fonction *« htmlspecialchars »* qui convertit les caractères spéciaux en entités HTML et rend donc inactive l'exécution d'éventuels scripts insérés dans les champs du formulaire.

```
$req->execute(array(
      'pseudo' => htmlspecialchars ($_POST['pseudo']),
      'message' => htmlspecialchars($_POST['message']),
      'dateMessage' => date("Y-m-d H:i:s")));
}
```

En cas de champs vides un message d'erreur est affiché :

```
else
{
   echo "Veuillez recommencer SVP";
}
```

Voyons maintenant le traitement nécessaire à la **récupération** des éléments stockés dans la base et à leur affichage sur la page. Ce traitement à lieu dans le fichier *message.php.*

Il faut, dans un premier temps, se connecter à la base de données avec :
« *include 'connectBdd.php'* »

Dans un deuxième temps, on organise la récupération et l'affichage des messages.

A l'aide d'une requête simple je sélectionne les données que je souhaite afficher, soit le pseudo, le message et la date à laquelle le message a été posté.

Ces données sont contenues dans la variable « *$resultatRequete* ».

```
$resultaRequete = $bdd->query('SELECT pseudo, message, dateMessage FROM messages ORDER BY dateMessage DESC ');
```

J'indique ici que je sélectionne le pseudo, le message et la date ordonnés par date décroissante.

« *$resultatRequete* » contient maintenant la requête MySQL, qui est, dans l'état, inexploitable car les informations qu'elle contient ne sont pas organisées.

Pour récupérer une entrée, on prend la requête MySQL et on exécute « *fetch()* » (*'*va chercher' en anglais) et qui renvoie la première ligne du tableau (array). Ici j'utilise

« *fetchAll() »* qui va renvoyer toutes les lignes du tableau contenu dans *$resultatRequete* et les stocker dans *$allreponse.*

```
$allreponse = $resultaRequete->fetchAll();
```

Maintenant je vais organiser ce tableau de manière à obtenir des données structurées :

Paramètre : valeur

```
$data = array();
    foreach ($allreponse as $reponse) {
        $reponseTab = array('pseudo' => $reponse['pseudo'], 'message' => $reponse['message'], 'dateMessage' => $reponse['dateMessage']);
        array_push($data, $reponseTab);
    }
```

Je vais créer un tableau vide (*$data*) que je vais remplir ligne par ligne à l'aide du « *foreach()* » et de « *array_push ».*

« foreach() » parcours toutes les lignes du tableau *$allreponse* et assigne à chaque itération la valeur de *$reponse.* Ce nouveau tableau est stocké dans la variable *$reponseTab.*

« array push » est une fonction qui va 'empiler' les variables contenues dans $data et $reponse.

Nous obtenons donc un tableau équivalent à ceci :

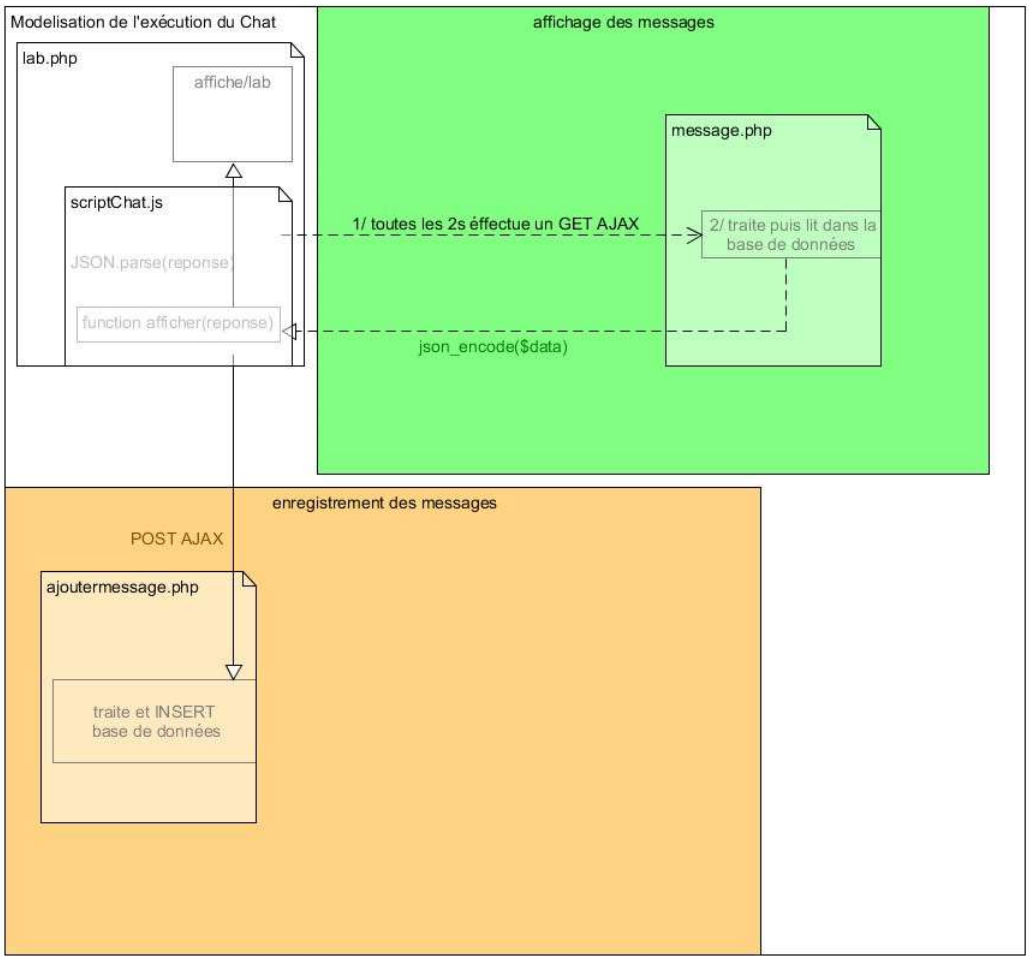
```
$data = array (
'pseudo' => $reponse['pseudo'],
'message' => $reponse['message'],
'dateMessage => $reponse['dateMessage'] ) ;
```

Maintenant j'utilise la fonction *header()* qui modifie l'entête de la requête HTTP qui sera envoyée au client, le navigateur et va préciser à ce dernier qu'il s'agit de données JSON.

```
header('Content-Type: application/json');
```

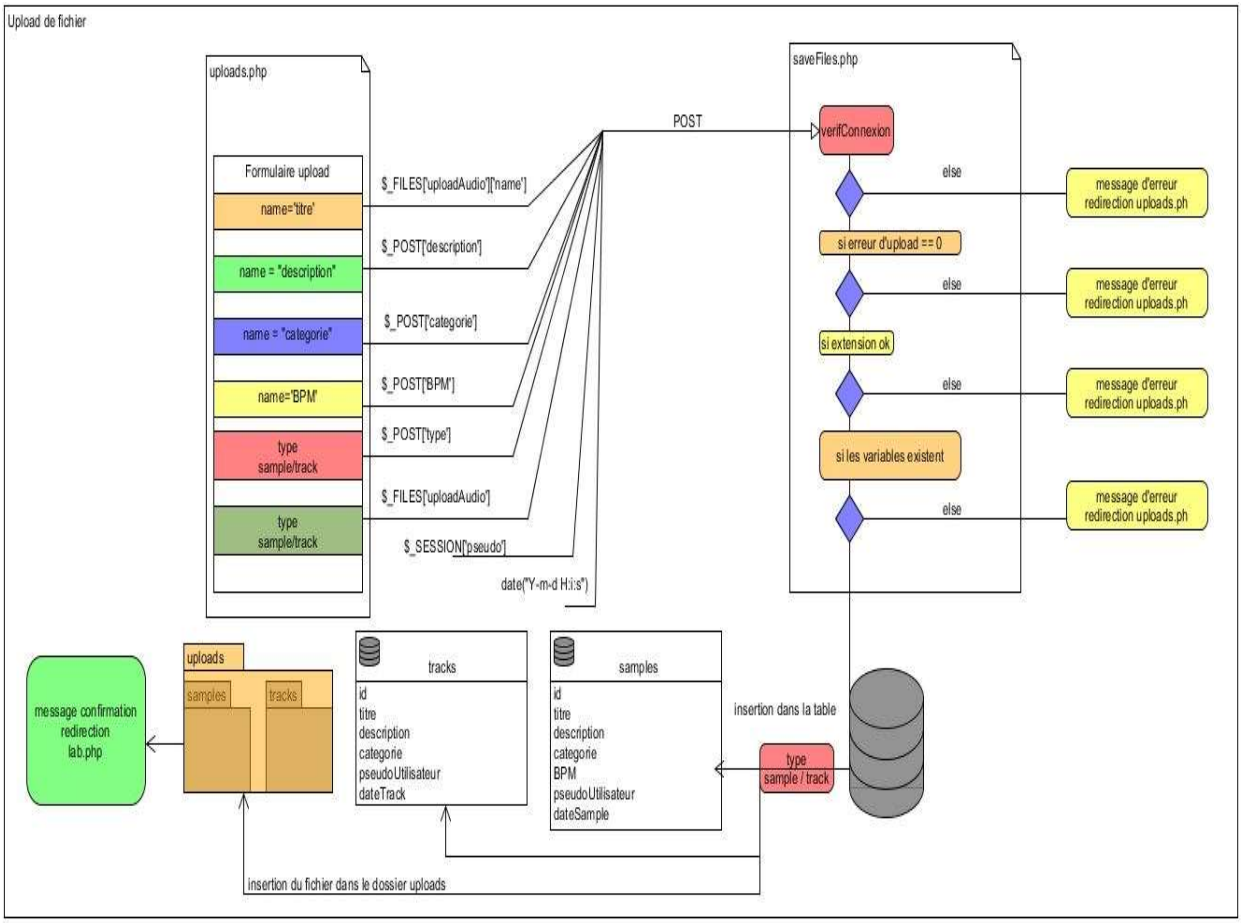
Enfin on affiche la réponse JSON sur la page avec :

```
echo json_encode($data);
```



III. Implémentation fonctionnalité Upload et Download

A. Upload



Le formulaire de chargement de fichiers se situe sur la page *uploads.php* et le traitement s'exécute dans la page *saveFiles.php.*

On se connecte à la base de données avec *include ('connectBDD.php')*. Ensuite vérifions que l'utilisateur est bien connecté :(*$_SESSION['connecte'] === true).*

Puis je vérifie la présence du fichier et l'absence d'erreurs et l'extension du fichier :

```
if (isset($_FILES['uploadAudio']) && ($_FILES['uploadAudio']['error'] == 0))    //Si le fichier audio s'est bien envoyé, error==0
  {
     $infoNomFichierAudio = pathinfo($_FILES['uploadAudio']['name']);
     echo $_FILES['uploadAudio']['name'];
     $extensionFile = $infoNomFichierAudio['extension'];
        if ($_POST['type'] == 'sample') {
              $extensionsFileAutorisees = array("WAV");//Si l'extension du fichier correspond bien aux extensions autorisées
        if (in_array(strtoupper($extensionFile), $extensionsFileAutorisees) == true && in_array(strtoupper($extensionFile), $extensionFileAutorisees) == true) {
```

A présent testons le type de fichier (sample ou track) et que les variables *$_POST['titre'], $_POST['description'], $_POST['categorie'], $_POST['BPM']* et *$_SESSION['pseudo']* ne sont pas vides :

```
if (isset($_POST['titre']) && isset($_POST['description']) && isset($_POST['categorie']) && isset($_POST['BPM']) && isset($_SESSION['pseudo'])) {
```

Et bien sûr la présence du fichier audio à charger :

```
if (isset($_FILES['uploadAudio']) && ($_FILES['uploadAudio']['error'] == 0)) {
```

Les vérifications faites et validées, je fais une requête préparée pour écrire dans la base :

```
$req = $bdd->prepare('INSERT INTO samples(idSample, name, description, categorie, BPM, pseudoUtilisateur, dateSample) VALUES(NULL, :titre, :description, :categorie, :BPM, :pseudoUtilisateur, :dateSample)');
```

Enfin la requête s'exécute sous forme de tableau et écrit les données dans la base :

```
$req->execute(array(
                    'titre' => htmlspecialchars($_FILES['uploadAudio']['name']),
                    'description' => htmlspecialchars($_POST['description']),
                    'categorie' => htmlspecialchars($_POST['categorie']),
                    'BPM' => htmlspecialchars($_POST['BPM']),
                    'pseudoUtilisateur' => ($_SESSION['pseudo']), //On récupère le pseudo depuis le serveur pour qu'il ne soit pas modifiable.
                    'dateSample' => date("Y-m-d H:i:s")
                ));
```

Et enfin l'enregistrement du fichier et de son identifiant :

```
$_ID = $bdd->lastInsertId(NULL);//Retourne l'identifiant autogénéré qui a pu être éxecuté correctement sur cette session

                move_uploaded_file($_FILES['uploadAudio']['tmp_name'], 'uploads/samples/' . $_FILES['uploadAudio']['name']);
```

B. Download

Le téléchargement de fichier se fait à partir du LAB. Seuls les samples sont téléchargeables.

Le bouton « Télécharger » est un lien vers la page *download.php* en prenant en paramètre l'id du sample :


```
<a href="download.php?idSample=<?php echo $ligneSample['idSample']; ?>"
```

Dans la page de traitement *download.php* on vérifie que l'utilisateur est connecté. Sinon il est renvoyé vers la page d'accueil :

```
if (!isset($_SESSION['connecte']) || $_SESSION['connecte'] != true) {
    header('Location: index.php#ancre2');
    exit(0);
}
```

On se connecte ensuite à la base de données :

```
try {
   $bdd = new PDO('mysql:host=localhost;dbname=kolabsonv9;charset=utf8', '****', '**********');
     } catch (Exception $e) {
   die('Erreur : ' . $e->getMessage());
     }
```

Avec une requête préparée on sélectionne le fichier à télécharger dans la base de données :

```
$req = $bdd->prepare('SELECT name FROM samples WHERE idSample=?');
```

La requête s'exécute ensuite en passant le tableau de paramètres de *$_GET['idSample']*

```
$req->execute(array(
   $_GET['idSample']
));
```

Je créé une variable *$sampleATelecharger* qui contient la première ligne de la réponse ($req->*fetch()*). Puis je créé une nouvelle variable *$file* qui prend en valeur le nom du fichier : *$sampleATelecharger['name'].*

J'indique ensuite le chemin du fichier : *$filepath = "uploads/samples/" . $file;*

S'en suit une vérification pour savoir si le fichier existe bien. Si oui, on procède alors au téléchargement, sinon on affiche un message à l'utilisateur.

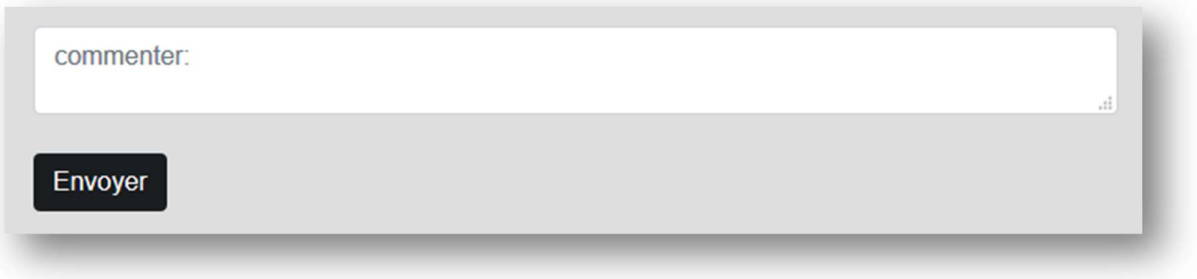
```
if(file_exists($filepath)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($filepath).'"');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . filesize($filepath));
    flush(); // Flush system output buffer
    readfile($filepath);
    exit;
```

(Source : https://www.php.net/manual/fr/function.readfile.php)

IV. Implémentation de la fonctionnalité 'Commentaires'

A. *Envoi de commentaires*

L'envoi de commentaires s'effectue, depuis la zone prévue à cet effet, en dessous de chaque publication dans la partie *EXPLORER.* L'utilisateur rédige son commentaire et l'envoie en cliquant sur le bouton *ENVOYER.* Le traitement s'exécute sue les pages *commentairesSample.php* et *commentairesTrack.php* suivant le type de fichier.



Ici, je vais présenter uniquement le processus de commentaires des samples. Le processus est le même pour les tracks. Nous sommes donc sur la page de traitement *commentairesSample.php.*

En premier lieu, je vérifie que l'utilisateur est connecté avec *« include 'verifConnexion.php' »* puis je vérifie que les variables *$_SESSION['pseudo']*, *$_POST['commentaire']* et *$_GET['id']* existent. Si oui, on passe à l'étape suivante. Si non l'utilisateur reçoit un message d'erreur. Une fois la vérification effectuée et validée j'établis la connexion avec la base de données.

A l'aide d'une requête « *prepare »* j'insère dans la table *commentairessample* l'id, le pseudo, le commentaire, l'identifiant du fichier concerné (*fileID*) et la date du commentaire. Je récupère le pseudo du rédacteur avec la variable de session *$_SESSION['pseudo']* et le commentaire avec la variable *$_POST['commentaire'].* La

variable *fileID* est transmise dans l'url avec *$_GET('id']* et la date avec la fonction PHP : date() *(date("Y-m-d H:i:s").*

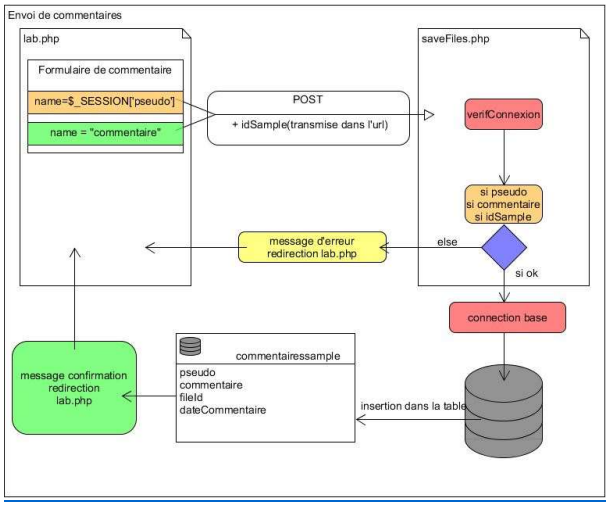
```
// Insertion du message à l'aide d'une requête préparée
    $req = $bdd->prepare('INSERT INTO commentairessample (id, pseudo, commentaire, fileID,
dateCommentaireSample) VALUES(NULL, :pseudo, :commentaire, :fileID,
:dateCommentaireSample)');
    $req->execute(array(
        'pseudo' => htmlspecialchars ($_SESSION['pseudo']),
        'commentaire' => htmlspecialchars($_POST['commentaire']),
        'fileID' =>  htmlspecialchars($_GET['id']),
        'dateCommentaireSample' => date("Y-m-d H:i:s")
    ));
```

Vous noterez que, comme pour le Chat, les entrées sont protegées par « *htmlspecialchars* ».

Une fois le traitement terminé et le commentaire écrit dans la base, on affiche un message à destination de l'utilisateur.

```
echo "<div class=\"alert alert-success\" role=\"alert\">
                Votre commentaire a été envoyé !
            </div>";
    echo "<p><a href='lab.php'>retour vers le site</a></p>";
```

Voici un schéma qui illustre ce processus :



B. Affichage des commentaires

L'affichage des commentaires se fait dans l'espace réservé sous chaque publication. Reprenons le cas d'un commentaire sur un sample.

Le traitement s'effectue directement dans la page du LAB (lab.php).

```
<!--AFFICHAGE DES COMMENTAIRES SAMPLES-->
        <?php
        $req = $bdd->prepare('SELECT * FROM commentairessample WHERE fileID=?');
        $req->execute(array(
           $ligneSample['idSample']
        ));
        $commentairesSample = $req->fetchAll();
        foreach ($commentairesSample as $commentaireSample){
        ?>
        <div class="container" id="divCommentaires">
           <div class='text-muted h7 mb-2'><i class='fa fa-clock-o'></i><?php echo
$commentaireSample['dateCommentaireSample'] ;?></div>
              <p><?php echo $commentaireSample['pseudo']; ?></p>
              <p><?php echo $commentaireSample['commentaire']; ?></p>
        </div>
```

Comme pour l'insertion dans la base j'utilise une requête « *prepare* » (préparée) pour sélectionner les éléments souhaités (dans ce cas, la totalité des commentaires alloués à un sample donné). L'id du sample concerné est transmise dans l'url par un GET (*WHERE fileID=?*) .

```
$req = $bdd->prepare('SELECT * FROM commentairessample WHERE fileID=?');
```

La requête s'exécute et renvoie un tableau contenant les valeurs de la requête. Avec le « *fetchAll()* » je sélectionne l'ensemble du tableau et le stock dans la variable *« $commentairesSample ».*

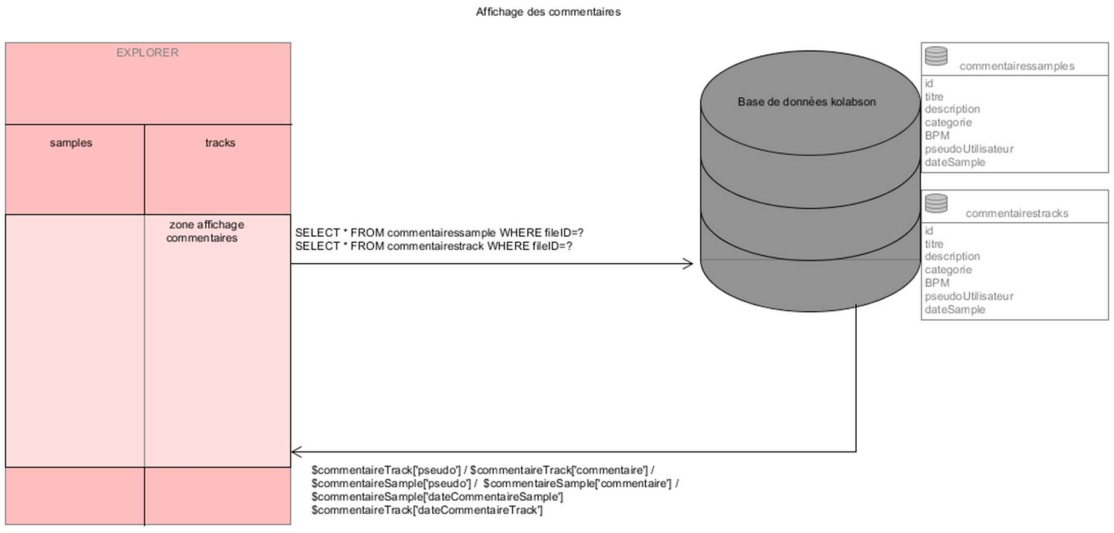
Le *« foreach ($commentairesSample as $commentaireSample) »* parcours le tableau. « *$commentairesSample* » stock le tableau des commentaires et « *$commentaireSample* » les lignes de ce tableau.

Ainsi, pour récupérer le pseudo de l'utilisateur qui a posté le commentaire, je peux désormais utiliser *$commentaireSample['pseudo']* : ligne [valeur de la colonne 'pseudo'].

```
$commentairesSample = $req->fetchAll();
 foreach ($commentairesSample as $commentaireSample){
 ?>
        <div class="container" id="divCommentaires">
            <div class='text-muted h7 mb-2'><i class='fa fa-clock-o'></i>
           <?php echo $commentaireSample['dateCommentaireSample'] ;?>
            </div>
             <p><?php echo $commentaireSample['pseudo']; ?></p>
             <p><?php echo $commentaireSample['commentaire']; ?></p>
         </div>
```

Enfin je construis la *« divCommentaires »* qui contient les valeurs parcoures par le *foreach* : *$commentaireSample[‘dateCommentaireSample’],*

$commentaireSample[‘pseudo’] et *$commentaireSample[‘commentaire’].*



V. Le formulaire de contact PHP MAILER

PHP Mailer est une classe PHP fournissant une collection de fonctions, qui permet de construire et d'envoyer des emails en utilisant une authentification sur le serveur SMTP, cette méthode sécurisée évite de voir les noms de serveurs sur les entêtes des emails reçus à travers le formulaire de contact par exemple, ou que reçoivent les utilisateurs, puisque les emails sont envoyés après une authentification réussie sur le serveur mail.

PHP MAILER est un projet *Git Hub* (https://github.com/PHPMailer/PHPMailer).

La documentation étant très détaillée, *PHP MAILER* est relativement facile à intégrer et à paramétrer.

```php
<?php
// Import PHPMailer classes into the global namespace
// These must be at the top of your script, not inside a function
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

// Load Composer's autoloader
require 'vendor/autoload.php';

// Instantiation and passing `true` enables exceptions
$mail = new PHPMailer(true);

try {
    //Server settings
    $mail->SMTPDebug = 2;                                 // Enable verbose debug output
    $mail->isSMTP();                                      // Set mailer to use SMTP
    $mail->Host       = 'smtp1.example.com;smtp2.example.com';  // Specify main and backup SMTP servers
    $mail->SMTPAuth   = true;                             // Enable SMTP authentication
    $mail->Username   = 'user@example.com';               // SMTP username
    $mail->Password   = 'secret';                         // SMTP password
    $mail->SMTPSecure = 'tls';                            // Enable TLS encryption, `ssl` also accepted
    $mail->Port       = 587;                              // TCP port to connect to

    //Recipients
    $mail->setFrom('from@example.com', 'Mailer');
    $mail->addAddress('joe@example.net', 'Joe User');     // Add a recipient
    $mail->addAddress('ellen@example.com');               // Name is optional
    $mail->addReplyTo('info@example.com', 'Information');
    $mail->addCC('cc@example.com');
    $mail->addBCC('bcc@example.com');

    // Attachments
    $mail->addAttachment('/var/tmp/file.tar.gz');         // Add attachments
    $mail->addAttachment('/tmp/image.jpg', 'new.jpg');    // Optional name

    // Content
    $mail->isHTML(true);                                  // Set email format to HTML
    $mail->Subject = 'Here is the subject';
    $mail->Body    = 'This is the HTML message body <b>in bold!</b>';
    $mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

    $mail->send();
    echo 'Message has been sent';
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

VI. Jeu d'essai

Le jeu d'essai consiste à créer une série de tests automatisés des entrées utilisateurs et contrôler les sorties attendues. Ces tests portent sur la sécurité du site.

Pour ce faire, j'ai utilisé *PHP Storm* et *phpunit* qui permettent de faire des tests unitaires.

Les formulaires de la page d'accueil étant protégés par le *google reCAPTCHA,* j'ai réalisé ces tests en local pour ne pas être bloqué par les *token* du reCAPTCHA (un genre de clé de session attribuée par Google pour valider le fait que l'utilisateur ne soit pas un robot.) qui bloque l'exécution des tests.

La 'bonne pratique' est d'avoir un code couvert à au moins 80%.

Dans l'exemple ci-dessous je test le formulaire d'inscription et plus particulièrement le cas où un utilisateur le rempli correctement.

La sortie attendue est donc que l'inscription soit validée et qu'il soit redirigé vers le formulaire de connexion.

Phpunit est implémenté en Programmation Orientée Objet.

Je créé une fonction nommée *testInscriptionOk* et je simule une entrée avec une adresse mail (au format valide), un pseudo et de mots de passes respectant le format imposé et identiques.

Ensuite je simule l'exécution de la page de traitement (*inscription.php*) avec *include.*

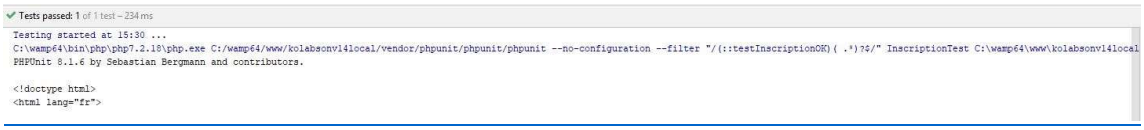
ob_start() démarre la temporisation de sortie. Tant qu'elle est enclenchée, aucune donnée, hormis les en-têtes, n'est envoyée au navigateur, mais temporairement mise en tampon.

ob_get_clean() lit le contenu courant du tampon de sortie puis l'efface.

Xdebug_get_headers renvoie tous les en-têtes () de PHP, ou tout autre en-tête défini en interne dans PHP (par exemple via setcookie ()), sous forme de tableau.
Enfin on exécute une assertion en comparant la chaine de caractères de sortie, ici une redirection, qui renvoie un résultat.

Ce jeu d'essai sera développé dans la version ultérieure dans l'optique d'atteindre les 80% de couverture préconisés.

```
use PHPUnit\Framework\TestCase;
// Pour que ces test puissent être executés, il faut que php soit lancé avec xdebug activé
final class InscriptionTest extends TestCase
{
   /**
    * @test
    * @runInSeparateProcess
    **/
   public function testInscriptionOK() {
      // On a saisi un email, et deux mots de passes identiques, l'inscription devrait bien se passer
      $_POST = array('mail'=>'monemail@gmail.com', 'pseudo'=> 'utilisateurTestInscription1',
'mdp1'=>'supermotdepasse123.', 'mdp2'=>'supermotdepasse123.');
      // On simule l'execution de notre page de traitement inscription.php
      include '../inscription.php';

      ob_start();
      ob_get_clean();

      $enTeteApresExecution = xdebug_get_headers();

      //var_dump($enTeteApresExecution); // A COMMENTER
      $this->assertStringContainsString("Location: index.php?message=ok#ancre2",
$enTeteApresExecution[0]);    }
```



Sources : https://stackoverflow.com/questions/2964834/php-check-if-url-redirects

Supports de développement et outils de veille

Pour implémenter ce projet je me suis documenté sur de nombreux sites comme :

Stack Overflow: https://stackoverflow.com/

Code Pen: https://codepen.io/qkevinto/pen/WQVNWO

PHP MANUAL: https://www.php.net/

Bootstrap: https://getbootstrap.com/docs/4.3/getting-started/introduction/

Git Hub : https://github.com/

OpenClassrooms: https://openclassrooms.com

Grafikart: https://www.grafikart.fr/

Alsacréations: https://www.alsacreations.com

World Wide Web Consortium (W3C): https://www.w3.org

Pour ma veille technologique j'utilise des sites comme :

Netvibes : https://www.netvibes.com

Korben : https://korben.info

GitHub : https://github.com

BleepingComputer : https://www.bleepingcomputer.com

Evolutions envisagées

Cette version de KoLABSoN est la première. Je compte continuer à la développer à l'issue de la formation. Les améliorations prévues sont :

- Mise en place d'un système de messages privés, personnalisation du profil utilisateur (photo ou image de profil).
- Création de catégories pour les samples et morceaux.
- Améliorations de l'interface (CSS), ajout de textes,
- Évolution du code PHP vers la Programmation Orientée Objet.
- Ajout d'un formulaire en cas d'identifiants perdus.
- Ajout de contenus audio (Création musicales).
- Installation de Google Analytics.

A l'issue de cette formation je compte continuer mon apprentissage en :

- Continuant à me former sur la programmation orientée objet en PHP.
- Apprenant à utiliser le framework ***Symfony.***
- Apprenant le langage C de manière à pourvoir développer des interfaces logiciels de création musicale.

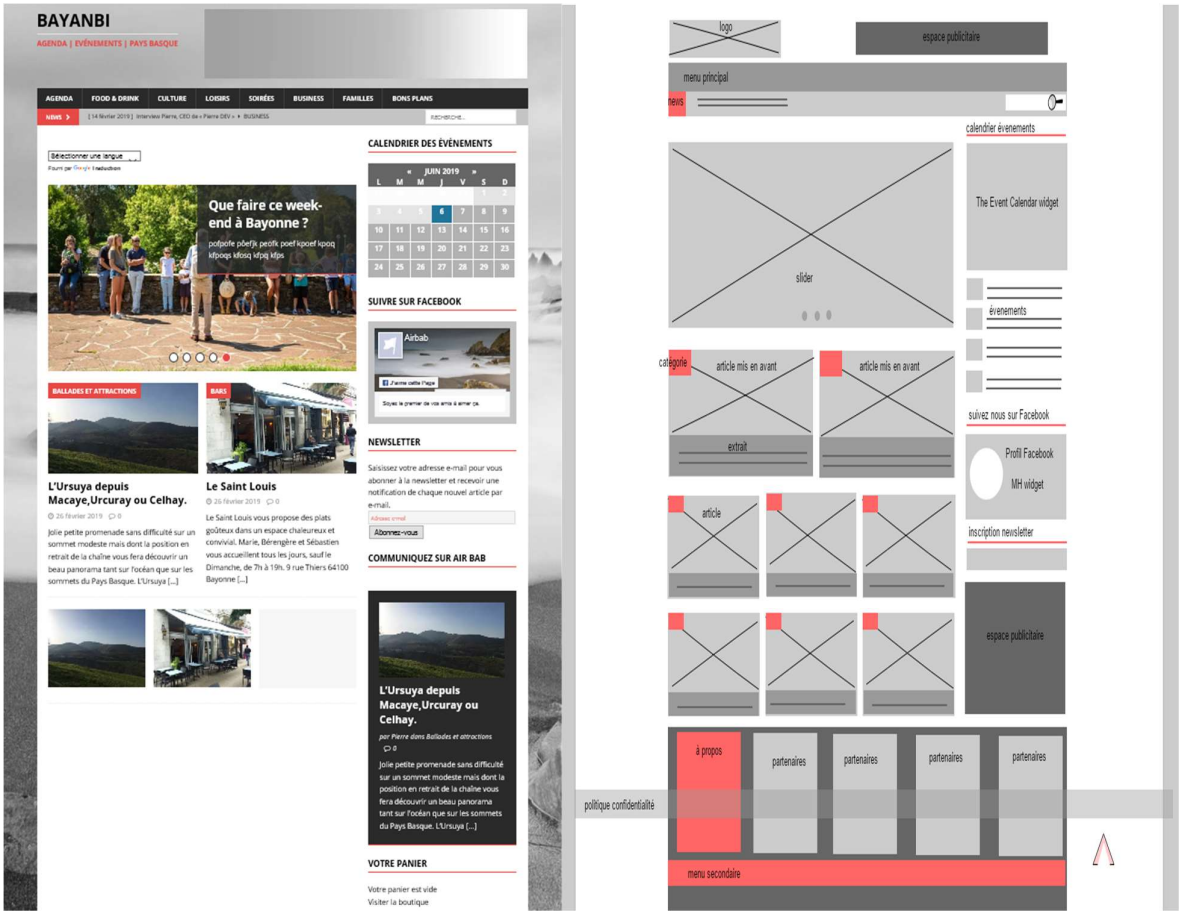
J'aimerais continuer dans le développement web de manière à solidifier mes acquis et en parallèle me former sur des langages permettant de développer des interfaces logiciels. Mon objectif premier reste cependant de trouver un emploi dans le développement Web et Web mobile.

Stages professionnels

J'ai effectué mon premier stage au sein d'une agence immobilière (***Côté Particuliers***). Le gérant a pour projet de créer un site « d'info sorties » sur la région élargie du BAB (50 kms).

Mon travail a consisté à développer la partie Front end du site sur WordPress. Création du menu, des catégories, mise en place des widgets… . J'ai intégré plusieurs plugins comme *MailChimp* ou The *EventCalendar* préalablement choisis par mon tuteur et je les ai paramétrés en suivant les documentations des éditeurs.

J'ai également effectué des modifications css dans le thème choisi. C'est un projet toujours actif que je compte poursuivre à l'issue de ma formation.



Mon second stage a eu lieu au sein de l'entreprise ***PIC Digital*** dirigée par Caroline Phillips.

Pendant ce stage j'ai fait de la maintenance de sites sur WordPress ou Presta shop, comme par exemple des résolutions de problèmes d'affichage ou de fonctionnement de plugins ou de la traduction en français de thèmes WordPress anglophones.

J'ai également implémenté des sites en suivant les maquettes fournies par le designer, organisé des rendez-vous clientèle afin de finaliser la mise en page de leur site et résoudre quelques bugs ainsi que des démonstrations.

J'ai également intégré divers plugins comme des rappels de panier, des modules de paiement ou encore résoudre des tickets sur des sites déployés.

Annexe 1 : Cas d'utilisations

1. Inscription

- L'utilisateur atterrit sur la page d'accueil.
- Accès au formulaire via le menu du header ou en scrollant jusqu'à la div.
- Il complète le formulaire avec son pseudonyme, son adresse mail, son mot de passe et la confirmation du mot de passe.

  -Si un ou plusieurs champs ne sont pas validés, il reçoit une alerte pour recommencer.

  - Si tous les champs sont rempli et validés, il est inscrit et reçoit une alerte
  l''invitant à se connecter.

2. Connexion

- L'utilisateur entre son pseudonyme et son mot de passe :

  - Si l'association pseudonyme / mot de passe n'est pas validée, l'utilisateur reçoit une alerte l'invitant à recommencer.

  - Si l'association pseudonyme / mot de passe est validée, l'utilisateur est redirigé vers le LAB.

3. Contact

- L'utilisateur entre son adresse mail, son message :

  - Si les champs ne sont pas validés, l'utilisateur est invité à recommencer.
  - Si les champs sont validés, le message est envoyé. L'utilisateur reçoit une alerte de confirmation.

4. Le LAB

Une fois que l'utilisateur est arrivé dans le LAB, il peut écouter, charger et télécharger des fichiers audio (samples uniquement).

Le LAB dispose d'une section DÉCOUVRIR où est regroupé l'ensemble des créations chargées par les différents utilisateurs.

L'utilisateur peut également chatter avec d'autres utilisateurs afin de favoriser la collaboration s'ils le souhaitent.

1. Chargement de fichiers audio:

Le chargement de fichiers se fait en cliquant sur l'onglet UPLOADS dans le menu de navigation. L'utilisateur est alors redirigé vers la page uploads.php.

- L'utilisateur doit renseigner les champs titre, description, le type de sample charger (rythme, instrument, vocal…..) ainsi que la vitesse à laquelle il a été élaboré (BPM).

- Il doit également renseigner de quelle nature est le fichier, Sample ou Track (échantillon ou morceau) en cochant la case correspondante.

- Pour finir, l'utilisateur doit valider le captcha et cliquer sur le bouton 'Envoyer'.

- Si les champs ne sont pas validés ou que le fichier chargé n'est pas valide:
  L'utilisateur reçoit une alerte sur le type de fichier autorisés et leur limite de taille.
  Il est invité à recommencer.

- Si les champs sont validés et que le fichier chargé a été vérifié et validé :

Le fichier est envoyé dans la base de données.
L'utilisateur est renvoyé dans le LAB et reçoit une alerte l'informant du chargement de
son fichier.

2. Téléchargement de fichiers audio:

Dans la section EXPLORER du LAB se trouve, sous chaque fichier disponible, un bouton 'Télécharger' permettant d'enregistrer le fichier et de l'utiliser dans ses propres
créations (Samples uniquement).

3. L'écoute de fichiers audio :

Dans la section EXPLORER du LAB se trouve, sous chaque fichier disponible, un lecteur minimal permettant d'écouter l'élément choisi.

4. Le Chat:

- Le Chat de KoLABSoN est accessible depuis le LAB.
- L'utilisateur doit simplement entrer son message et cliquer sur le bouton ENVOYER dans le formulaire situé sur la gauche de l'écran.

- Si les champs ne sont pas validés:

  L'utilisateur reçoit une alerte et est invité à recommencer.

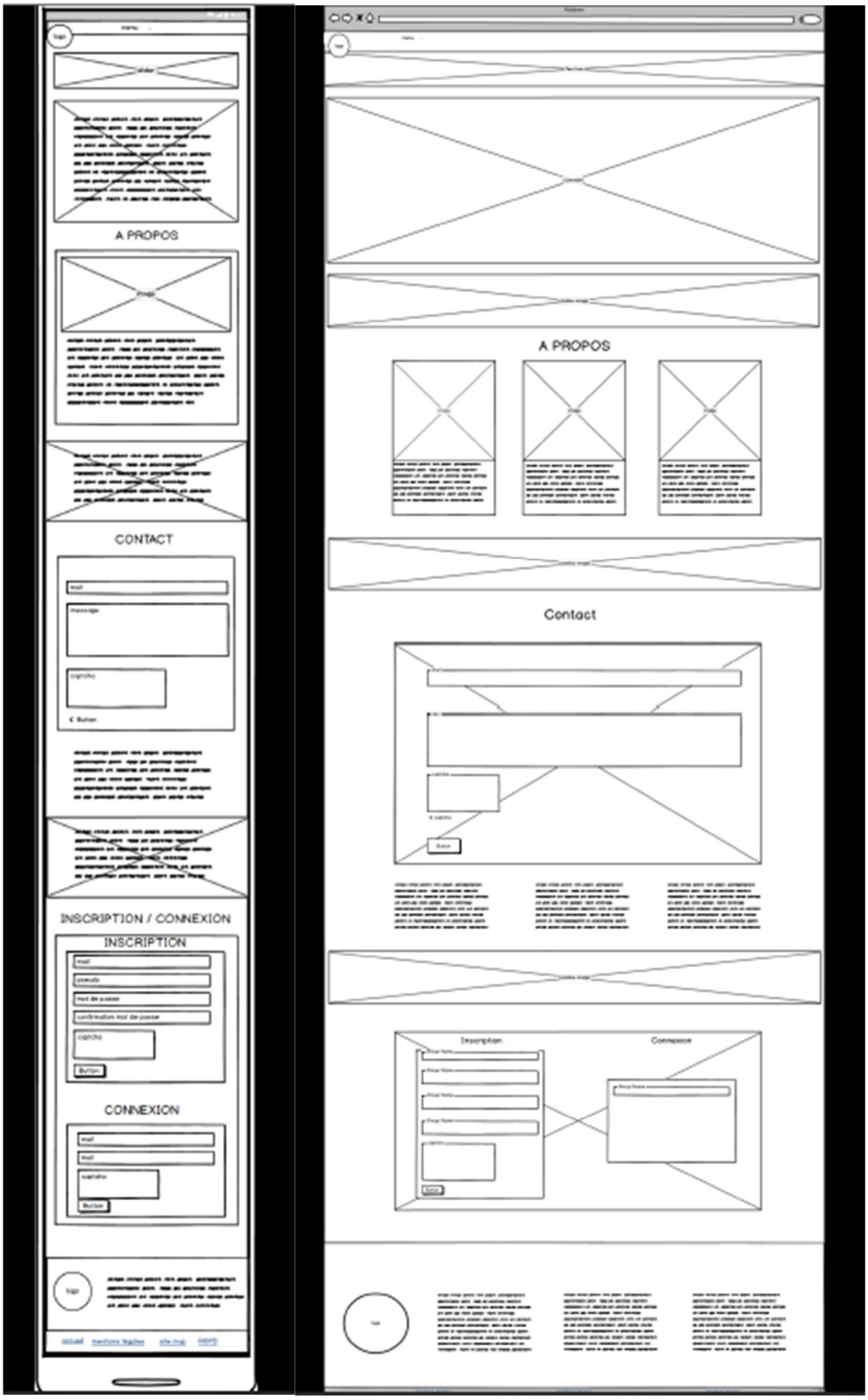
- Si les champs sont vérifiés et validés :

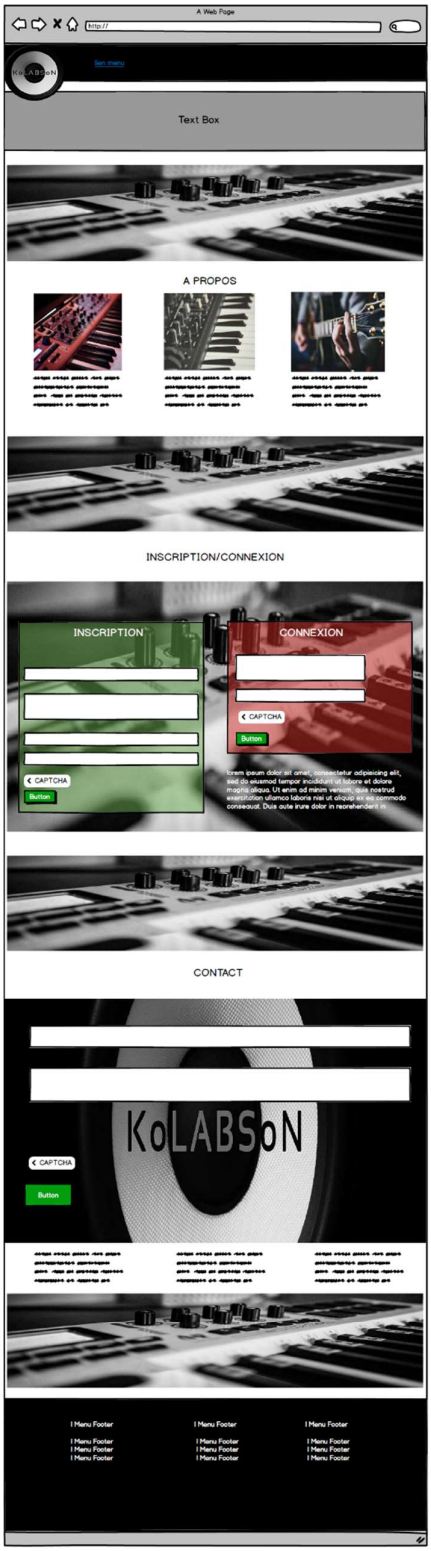
  L'utilisateur verra alors son message s'afficher dans la partie dédiée à droite de l'écran.

5. Les commentaires :

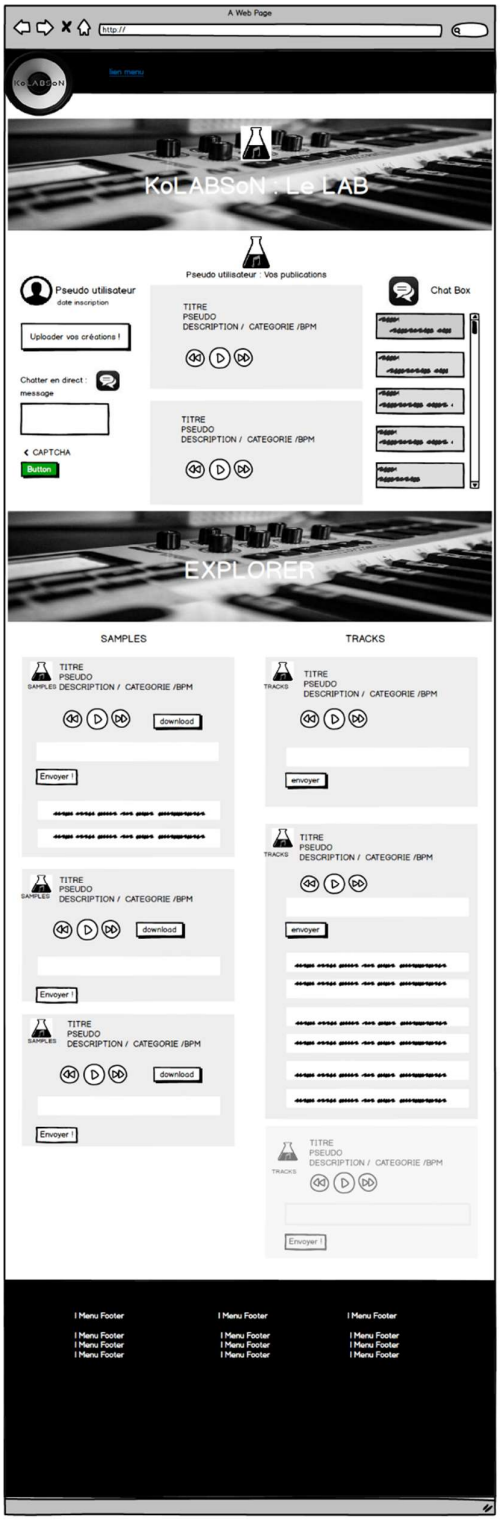
Les commentaires se font depuis le formulaire placé sous chaque publication.
L'utilisateur entre son commentaire et clique sur le bouton Envoyer.
Le commentaire laissé s'affichera à la suite des autres, en précisant par qui et quand il a été posté.
En cas d'erreur l'utilisateur est averti par un message et est invité à recommencer.

Annexe 2 : Maquettes




A Web Page
http://
Text Box
A PROPOS
INSCRIPTION/CONNEXION
INSCRIPTION
CONNEXION
CAPTCHA
Button
CONTACT
KoLABSoN
Menu Footer

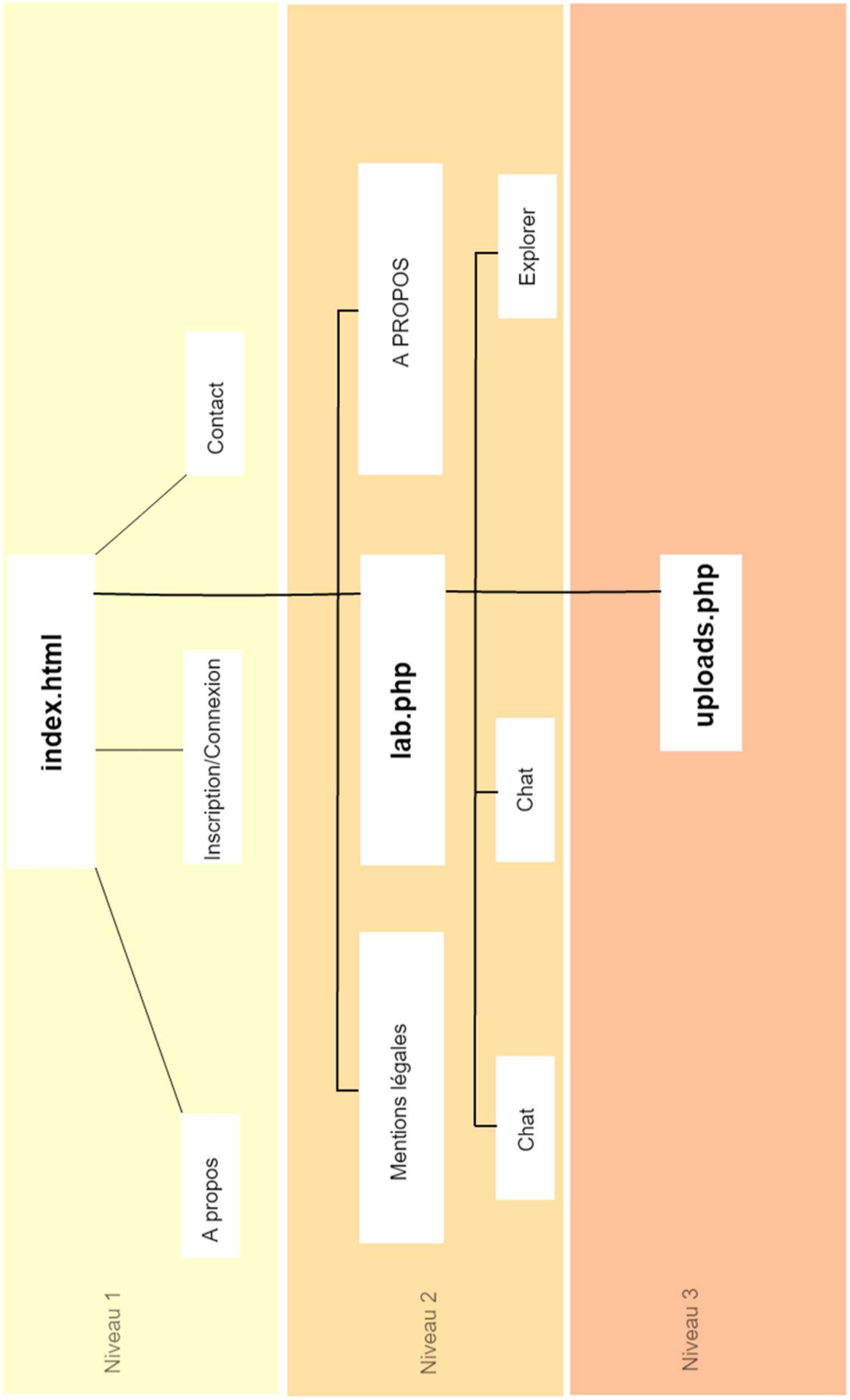

A PROPOS
INSCRIPTION/CONNEXION
INSCRIPTION
CONNEXION
CAPTCHA
Button
CONTACT
Menu Footer


A Web Page
http://
lien menu
KoLABSoN : Le LAB
Pseudo utilisateur
date inscription
Uploader vos créations !
Chatter en direct :
message
CAPTCHA
Button
Pseudo utilisateur : Vos publications
TITRE
PSEUDO
DESCRIPTION / CATEGORIE /BPM
Chat Box
EXPLORER
SAMPLES
TRACKS
download
Envoyer !
envoyer
Menu Footer


KoLABSoN : Le LAB
Pseudo utilisateur
date inscription
Uploader vos créations !
Chatter en direct :
message
CAPTCHA
Button
Pseudo utilisateur : Vos publications
TITRE
PSEUDO
DESCRIPTION / CATEGORIE /BPM
Chat box
EXPLORER
Envoyer !
Menu Footer


A Web Page
http://
lien menu
LAB - UPLOADS
Pseudo utilisateur
date inscription
Checkbox
Checkbox
CAPTCHA
Button
Menu Footer
Envoyer vos créations

Annexe 3 : Arborescence



Annexe 4 : Cahier des charges

CAHIER DES CHARGES

Sommaire

Présentation du projet

A. Présentation de KoLABSoN:

KoLABSoN est une contraction de ' Collaboratif ' , 'Laboratoire' et ‘son’.

Comme son nom l'indique, il s'agit d'un laboratoire de créations musicales ou sonores.

Les objectifs du site :

KoLABSoN est une plate-forme ou les compositeurs amateurs peuvent 'exposer' leurs créations, commenter, échanger et partager des échantillons.
Détaillez le ou les objectifs de votre site web. Préciser s’il s’agit :

Cible adressée par le site :

KoLABSoN s’adresse à tous les compositeurs amateurs désireux de mettre en lumière leurs créations ou simplement aux utilisateurs curieux d'écouter de nouvelles compositions.

Périmètre du projet :

Pour une meilleure visibilité KoLABSoN aura une version anglaise.

Le site sera adapté pour les supports mobiles

Description de l’existant :

Il s'agit d'un projet qui va se construire « from scratch », à partir de zéro.

Description graphique et ergonomique

Charte graphique :

Dans un souci de respect du référentiel de la W3C (www.w3.org) en matière d'accessibilité, le code couleur sera simple et accessible, noir et blanc avec des tons grisés, qui rappelleront celles du logo et qui ne perturberons pas les personnes présentant des déficiences visuelles.

Les éléments principaux seront mis en valeur par des couleurs vives, vert et rouge. Vert pour la section d'inscription et rouge pour la connexion.

La police de caractère sera « Helvetica Neue sans-serif » supportée par tous les navigateurs.

Pour une pratique plus écologique, KoLABSoN met tout en œuvre pour se conformer au référentiel édité par la GreenIt (www.greenit.fr / « Ecoconception web : les 115 bonnes pratiques »), notamment en livrant un design simple et épuré de manière à éviter le chargement de fichiers volumineux superflus (images, sliders, …)

Design:

Cependant la page d'accueil sera pourvue d'effets et d'animations ayant pour but de séduire le potentiel futur utilisateur.

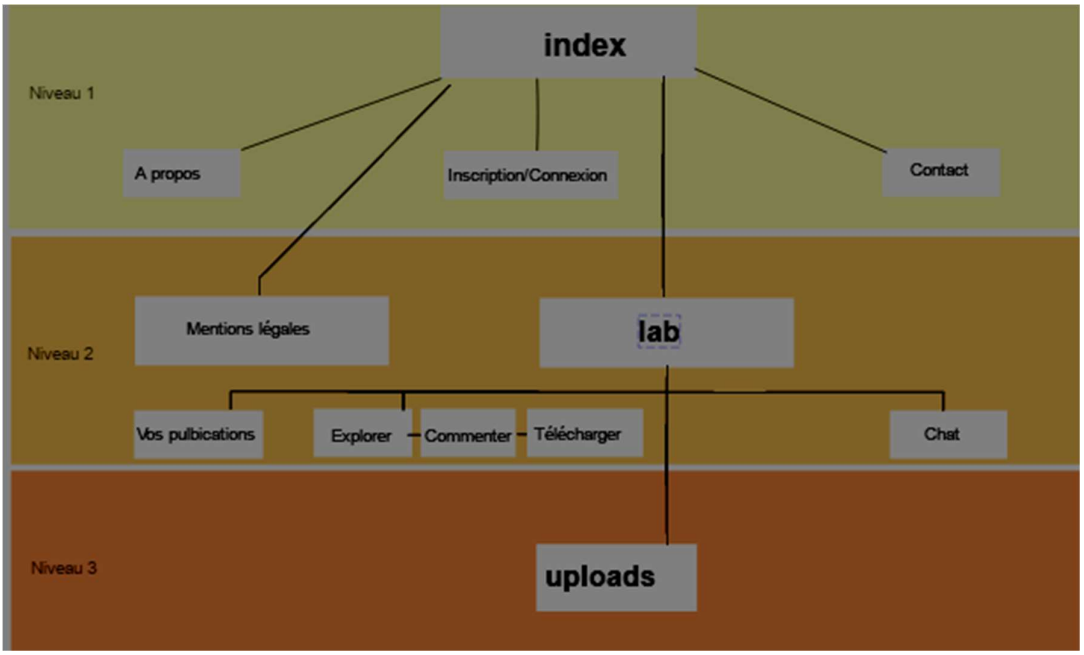
D'un côté esthétique un effet parallaxe est souhaité pour 'styliser' la navigation ainsi qu'une animation textuelle discrète à l'accueil de l'utilisateur.

Le design doit être simple et intuitif et sera donc construit, de manière logique, en une succession de blocs.

Les éléments clés seront mis en évidence pour une interaction intuitive.

Description fonctionnelle et technique

Arborescence du site :



Description fonctionnelle :

Le LAB ainsi que l'UPLOAD seront accessible uniquement par inscription et connexion.

Chaque utilisateur aura un pseudo et un mot de passe.

L'affichage devra être personnalisé pour chaque utilisateur.

Informations relatives aux contenus :

KoLABSoN hébergera des fichiers audio au format .wav et .mp3

KoLABSoN se veut de respecter les standards SEO en matière de code HTML notamment des balises titres et des méta-informations.

Contraintes techniques :

Une implémentation en PHP est privilégiée de manières à offrir une application web dynamique.

Le code HTML sera validé par la W3C et les règles de sécurité mises en œuvre.

Le logo et les icônes seront conçus sur Photoshop.

Cette version sera développée en locale puis mise en ligne sur un serveur test, mis à disposition temporairement par l'organisme de formation.

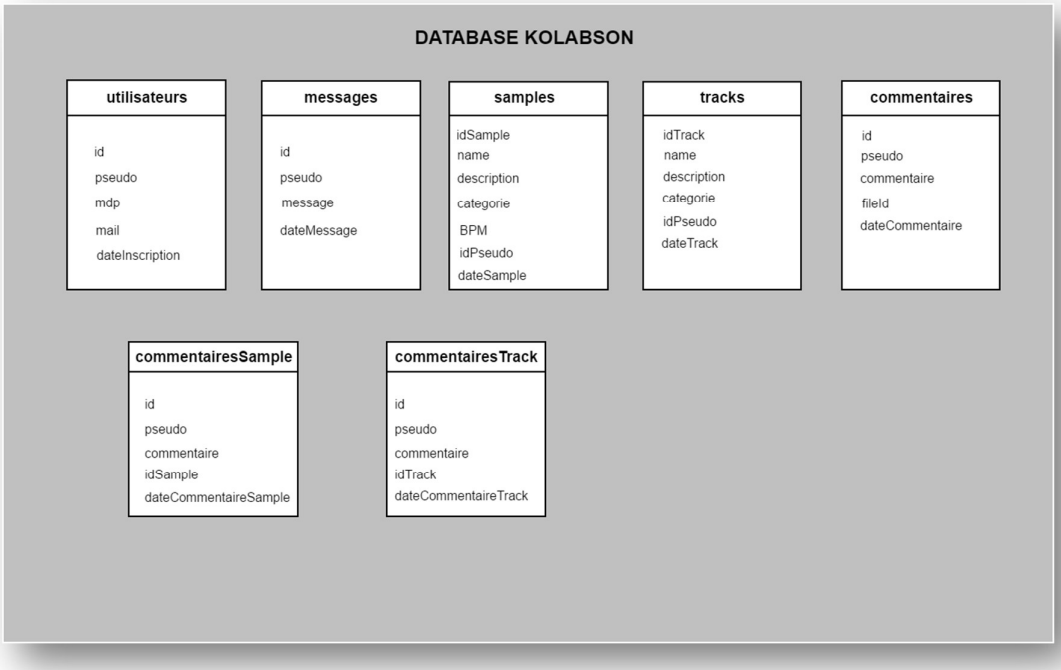
Prestations attendues

Cette version de KoLABSoN doit contenir tous les éléments principaux nécessaires à son fonctionnement:

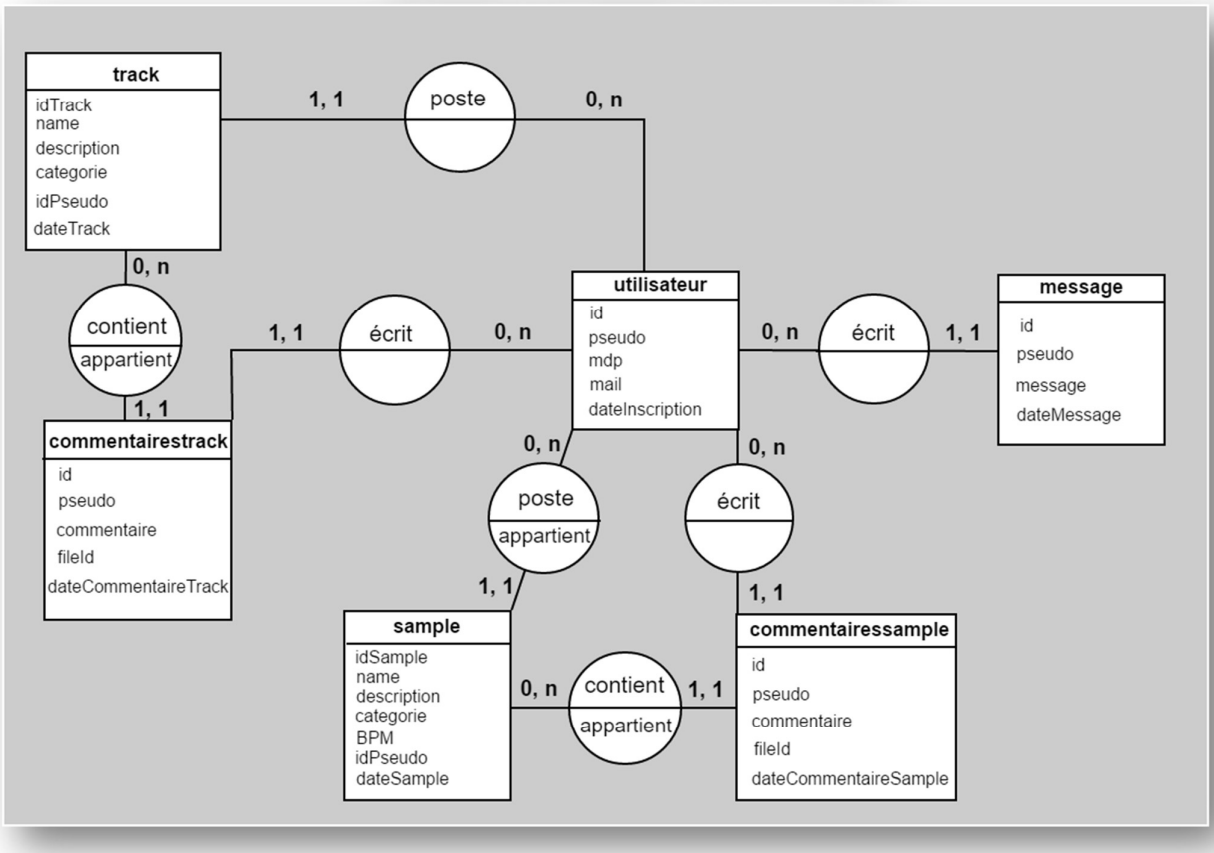
Les formulaires de contact, inscription et connexion,
Les fonctionnalités d'écoute, de chargement et de téléchargement de fichiers audio,
Le chat inter-utilisateurs,
Le système de commentaires,
Les systèmes de protection contre les spams et intrusions,
Le respect des lois sur le RGPD,
Le logo et les icônes du site.

Des améliorations ultérieures sont cependant prévues (*cf. projet professionnel*)

Annexe 5 : Base de données et MCD



Modèle Conceptuel de Données



Annexe 6 : Traduction de la documentation de PHPMAILER

Extrait:

Installation & loading

PHPMailer is available on Packagist (using semantic versioning), and installation via Composer is the recommended way to install PHPMailer. Just add this line to your `composer.json` file:

```
"phpmailer/phpmailer": "~6.0"
```

or run

```
composer require phpmailer/phpmailer
```

Note that the `vendor` folder and the `vendor/autoload.php` script are generated by Composer; they are not part of PHPMailer.

If you want to use the Gmail XOAUTH2 authentication class, you will also need to add a dependency on the `league/oauth2-client` package in your `composer.json`.

Alternatively, if you're not using Composer, copy the contents of the PHPMailer folder into one of the `include_path` directories specified in your PHP configuration and load each class file manually:

```
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

require 'path/to/PHPMailer/src/Exception.php';
require 'path/to/PHPMailer/src/PHPMailer.php';
require 'path/to/PHPMailer/src/SMTP.php';
```

If you're not using the `SMTP` class explicitly (you're probably not), you don't need a `use` line for the SMTP class.

If you don't speak git or just want a tarball, click the 'zip' button on the right of the project page in GitHub, though note that docs and examples are not included in the tarball.

PHPMAILER documentation:

https://github.com/PHPMailer/PHPMailer/blob/master/README.md

Annexe 6 : Traduction de la documentation de PHPMAILER

Traduction:

PHPMAILER est disponible sous forme de pack (en utilisant la version sémantique), et l'installation via *Composer* est la manière recommandée. Ajoutez simplement ces lignes à votre fichier *composer.json* :

```
"phpmailer/phpmailer": "~6.0"
```

Ou exécutez

```
composer require phpmailer/phpmailer
```

Notez que le dossier `vendor` et le script `vendor/autoload.php` sont générés par Composer ; Ils ne font pas parti de PHPMailer.

Si vous voulez utiliser la classe d'authentification de Gmail XOAUTH2, vous devrez aussi ajouter une dépendance avec le pack *league/oath2-client* dans votre fichier *composer.json.*

Si par une autre alternative, vous n'utilisez pas Composer, copiez le contenu du dossier de PHPMailer à l'intérieur d'un des répertoires spécifiés dans votre configuration PHP et chargez chaque fichier de classe manuellement.

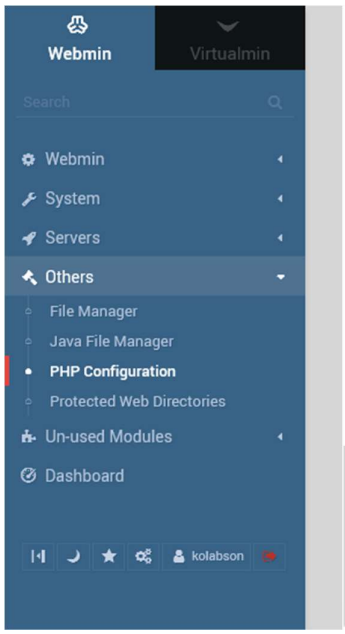
```
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

require 'path/to/PHPMailer/src/Exception.php';
require 'path/to/PHPMailer/src/PHPMailer.php';
require 'path/to/PHPMailer/src/SMTP.php';
```

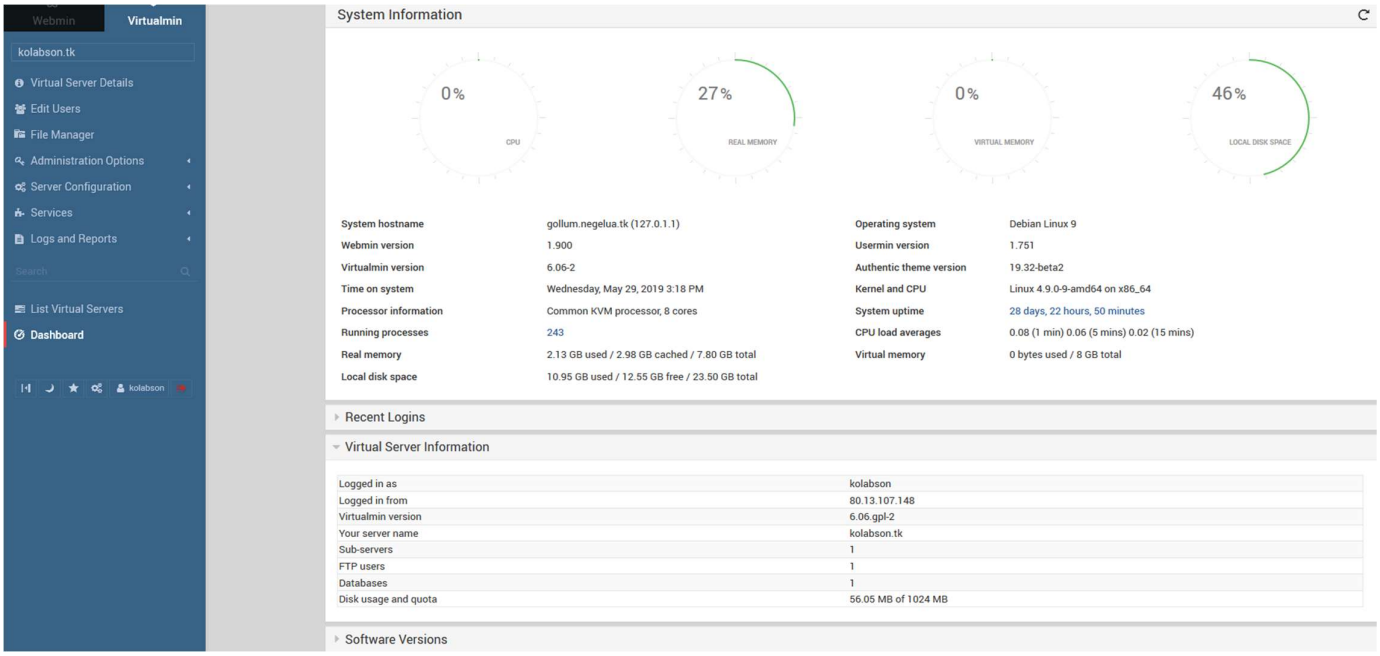
Si vous n'utilisez pas la classe SMTP de façon explicite (vous ne le faites probablement pas), vous n'avez pas besoin de la ligne `use` pour la classe SMTP.

Si vous n'utilisez pas Git ou que vous voulez juste une archive, cliquez sur le bouton 'zip' à droite de la page dans le projet sur Git Hub. Notez bien que ces documents et ces exemples ne sont pas inclus dans l'archive.

Annexe 7: Exemples de paramétrages du serveur

Paramétrage de la taille maximale autorisée, des fichiers uploader, par le serveur via l'interface administrateur.






```
; http://php.net/upload-max-f
upload_max_filesize = 250M
```

```
; is disabled through enable_post_data_
; http://php.net/post-max-size
post_max_size = 250M

; Automatically add files before PHP do
```