

CREATION D'UN BACK-OFFICE

Pour le SITE INTERNET ADRIEN-MARTIN.FR

Sommaire :

Problématique au début de la mission :	Page 3
But de la mission:	Page 3
Analyse et de compréhension des besoins :	Page 3
Proposition d'une ou plusieurs solutions :	Page 3
Réalisation de la solution choisie :	Page 4
1. Ordonnancement des tâches :	Page 5
2. Partie développement des tâches :	Page 6
3. Tests et problèmes rencontrés :	Page 15
4. Organisation du travail :	Page 16
EVALUATION DE LA MISSION :	Page 17
LEXIQUE :	Page 18
ANNEXES :	Page 19

Problématique au début de la mission :

Le client nous a confié récemment l'ajout d'un blog sur son site mais ne trouve plus la gestion de son site facile par le biais de phpmyadmin. Il nous a contactés pour qu'on fasse un back-office, ou dit autrement une interface graphique de gestion pour son site, accessible directement depuis ce dernier par le biais d'une identification.

But de la mission :

Créer un module backoffice avec une interface simple d'utilisation dans le but de gérer le contenu de son site.

Analyse et de compréhension des besoins :

Pendant la phase d'analyse et de compréhension des besoins, le client fait ressortir plusieurs points importants notamment la volonté de modifier le contenu de son site simplement et de le faire directement sur le site par le biais d'une interface spéciale dont seulement lui en a l'accès (pour le moment).

Proposition d'une ou plusieurs solutions :

Il est évoqué deux solutions pour satisfaire le client:

- Création d'un bundle spécifique reprenant **la charte graphique** du site.
L'avantage : pas de refonte du site (gain de temps).
L'inconvénient : alourdi un peu plus le site.
- Refonte du site avec un framework PHP.
L'avantage : structure le code de façon professionnelle permettant l'ajout d'option et de module sans aucun problème.
L'inconvénient : l'apprentissage d'un framework complet nécessite du temps. L'utilisation d'un framework pour une structure de site relativement simple reste disproportionnée.

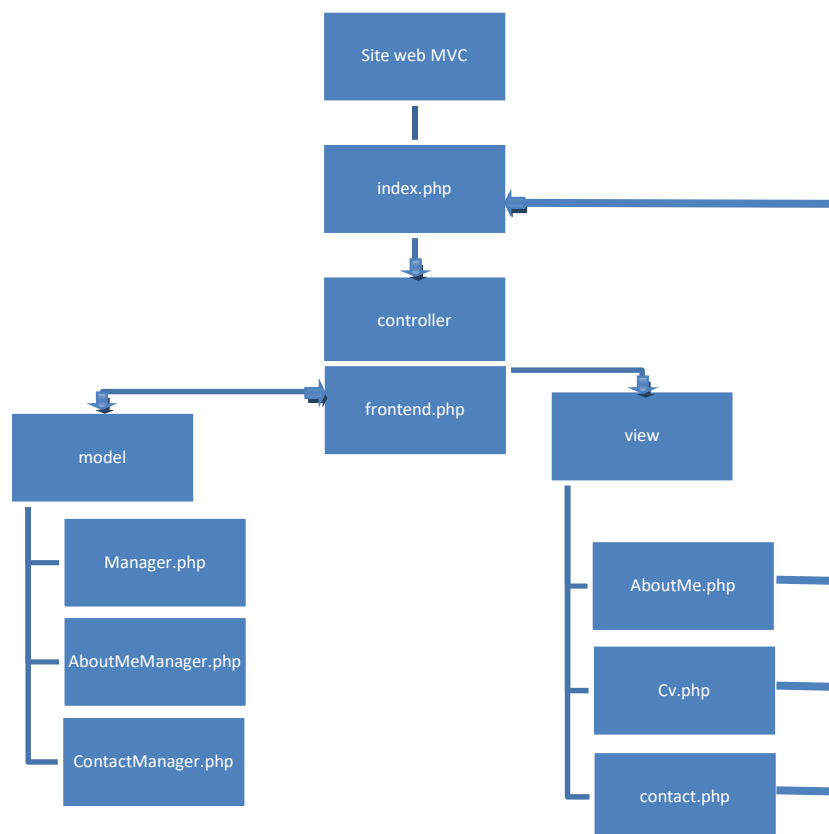
Après concertation avec le client, il est décidé que même si la version du framework sera l'objet d'une mission future, le besoin actuel du client sur son site et les futurs besoins à venir ne sont pas aussi importants ce qui ne justifie pas de mettre en place un important système aussi **qualitatif**. **Le délai** reste court et ne permet pas du temps de formation dans l'étude d'un framework et **le coût** demandé par le client reste faible pour proposer une solution complète pour le moment.

La première solution est donc retenue.

Réalisation de la solution choisie :

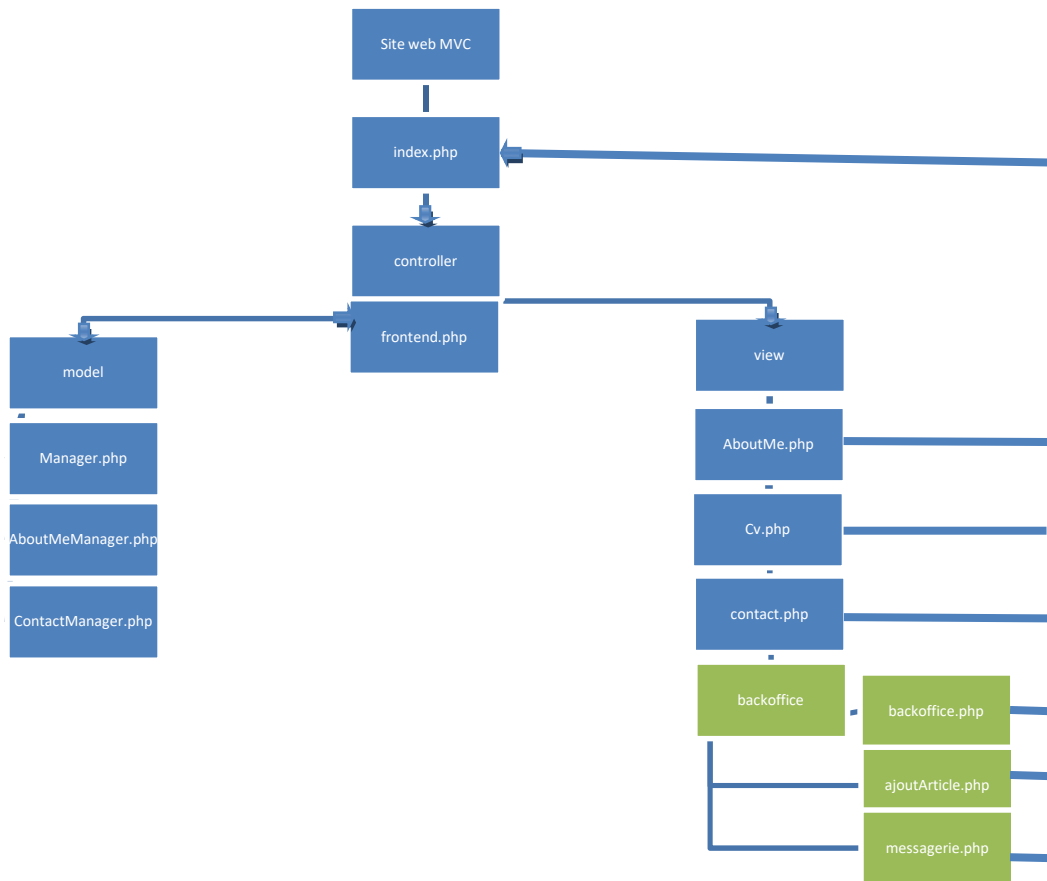
Avant de commencer il faut préciser ce qu'est le modèle MVC (Modèle, Vue, Contrôleur). C'est une architecture basé sur la séparation du code en trois parties. Le but étant de ne pas mélanger le code PHP, les requêtes SQL et le traitement JavaScript éventuel sur le même fichier.

Le modèle MVC donc, permet de bien séparer le code. Le principe est d'**utiliser l'index du site comme routeur** permettant de choisir le chemin à suivre vers le contrôleur en fonction de l'action de l'utilisateur. **Le contrôleur va procéder au traitement de l'action** en récupérant éventuellement des données sous forme de variables et va soit interagir directement vers la vue ou bien passer par un modèle permettant de le mettre en relation avec la base de données. Le modèle justement est une page ou un ensemble de page permettant de se connecter à la base de données et de faire un traitement par le biais de requêtes SQL (lecture, modification, suppression de données). Ce dernier renvoie les données au contrôleur qui va les afficher dans la vue qui n'est autre qu'une page internet. Il peut y avoir plusieurs pages qui servent de vue ou bien on peut se servir d'un **template**.



Architecture du site avec le modèle MVC.

Le but est d'ajouter le module back-office sur le site sans en modifier la structure d'origine :



1. Ordonnancement des tâches

Pour rendre la mission moins complexe, il faut découper le projet en différentes tâches plus simples.

Nous allons découper le projet en 4 tâches:

- une partie **connexion**;
- une partie **structure** générale du back-office;
- une partie **ajout** d'article;
- Et une partie gestion **messagerie**.



2. Partie développement des tâches

Partie Connexion :

Dans cette partie nous allons nous occuper de permettre au client de s'identifier sur son site et d'accéder à son back-office.

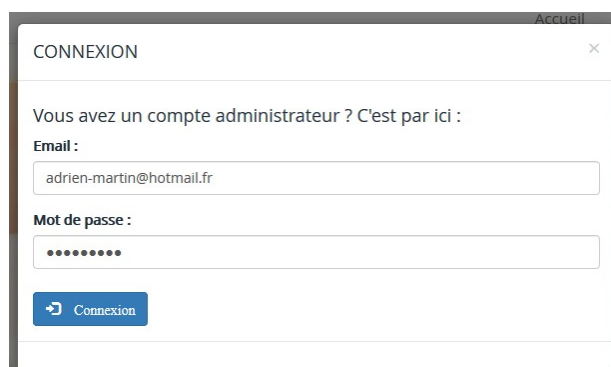
Plusieurs approches sont possibles :

- Créer « en dur » sur chaque page un mini-formulaire comportant deux zones de texte et un bouton connexion ;
- Ajouter un lien vers une page de connexion ;
- Ajouter un bouton affichant une mini-fenêtre de connexion.

La première idée est impensable de nos jours, vu la qualité des sites produits et grâce à l'amélioration de certains langages.

La deuxième idée paraît convenable mais reste trop simple. Ça manque de modernité.

La troisième solution est la version qui sera retenue car finalement très facile à faire et très utilisée sur les sites récents.

A screenshot of a modal window titled "CONNEXION" with a close button (X) in the top right corner. The window has a light gray border and a white background. Inside, the text "Vous avez un compte administrateur ? C'est par ici :" is displayed. Below this, there are two input fields: "Email :" with the value "adrien-martin@hotmail.fr" and "Mot de passe :" with masked characters ".....". At the bottom left of the modal is a blue button with a white right-pointing arrow and the text "Connexion". The modal is overlaid on a page with a dark header bar containing the word "Accueil" on the right.

Appelé « **Modal** », cette mini-fenêtre de connexion apparaît lorsque l'on clique sur le bouton connexion dans le menu du site.



Concernant le code, il y a un événement qui intervient sans même que l'on change de page ou qu'on la recharge. C'est grâce au langage JavaScript et plus précisément grâce à une de ses bibliothèques nommée JQUERY (dans ce code précisément). Grâce à ce langage, nous pouvons apporter de l'interactivité sur les pages sans même avoir à changer de page ou le rechargement de cette dernière.

Extrait du document JS:

```
$(function() {  
    // Acces back office par menu-connexion  
    $('#btn_connexion').on("click", function() {  
        var email = $("#email").val();  
        var password = $("#password").val();  
        $.ajax({  
            type : 'POST',  
            url : $('#form_connexion').attr('action'),  
            data : $('#form_connexion').serialize(),  
            dataType: 'html',  
            success: function(code_html, statut, data){  
                window.location.replace('index.php?action=connexion&email=' + email + '&password=' + password);  
            },  
            error : function(resultat, statut, erreur){  
            },  
            complete : function(resultat, statut){  
            }  
        });  
    });  
});
```

Ce fichier est en langage JavaScript et plus précisément dans la syntaxe JQUERY. Il est écrit que dès que le bouton connexion dans la modal est cliqué, on récupère les valeurs email et password du formulaire de connexion et on appelle l'action du formulaire pour le traiter.

Nous suivons donc l'action qui va nous mener au fichier index qui est pour rappel le routeur du site. Nous retrouvons un test de vérification qui va vérifier que les champs ne sont pas vide et si c'est le cas va appeler la fonction « addConnexion » et en envoyant les valeurs pour contrôle.

Extrait du document Index:

```
elseif ($_GET['action'] == 'connexion') {  
    if(!empty($_GET['email']) && !empty($_GET['password'])) {  
        addConnexion($_GET['email'], $_GET['password']);  
    }else {  
        echo 'Erreur : Login et/ou password ne sont pas remplis !';  
        header ("Refresh: 3;URL=index.php?action=contact");  
    }  
}
```

[Vous trouverez le fichier index complet en annexe 1.](#)

Poursuivons l'action vers le contrôleur qui va appeler la fonction addConnexion. Cette dernière va hasher le mot de passe et faire un appel à la base de données par le biais du modèle ConnexionManager.

Précision, le mot de passe est également hashé dans la base de données et donc le contrôle se fera sur le même mot. Le hashage est une manière de crypter le mot de passe pour ne pas permettre à des intrus de le trouver.

Extrait du document contrôleur:

```
function addConnexion($email, $password){
    $pass_hache = password_hash($password, PASSWORD_DEFAULT);
    $cnx = new ConnexionManager();
    $resultat = $cnx->getConnexion($email, $pass_hache);

    if (!$resultat) {
        echo 'Impossible de vous connecter!';
        echo 'Veuillez vérifier vos identifiants !';
        echo 'Vous allez être redirigé vers la page connexion dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=connexion");
    }
    else {
        session_start();
        $_SESSION['email'] = $email;
        header ("Refresh: 0;URL=index.php?action=session");
    }
}
```

[Vous trouverez le fichier complet en annexe 2.](#)

Nous continuons le cheminement vers le modèle ConnexionManager.php.

Extrait du document modèle:

```
public function getConnexion($email, $password){
    $db = $this->dbConnect();

    $connexion = $db->prepare('SELECT * FROM users WHERE email = ? AND password = ?');
    $connexion->execute(array($email, $password));

    $data = $connexion->fetch();

    return $data;
}
```

[Vous trouverez le fichier complet en annexe 3.](#)

La fonction fait une requête à la base de données pour vérifier si l'identifiant et le mot de passe saisis trouve une correspondance dans la table où sont stockés ces derniers puis renvoie le résultat au contrôleur.

Le contrôleur récupère le résultat de la requête et donne accès au backoffice s'il y a correspondance sinon affiche un message d'erreur renvoyant sur la page contact.

Partie structure :

Cette partie relativement simple concerne la structure du bundle back-office et son interface.

Le dossier où seront toutes les pages du back-office sera dans le dossier des vues mais séparé des autres pages du site dans un sous-dossier pour éviter la confusion.

Concernant l'accès, il se fait en suivant de la première partie qui s'était arrêté au niveau du retour au contrôleur.

Extrait du document :

```
else {  
    session_start();  
    $_SESSION['email'] = $email;  
    header ("Refresh: 0;URL=index.php?action=session");  
}
```

Ce dernier renvoi à l'index du site pour appeler une fonction qui va se charger dans un premier temps de créer une session puis d'accéder au back-office.

Extrait du document :

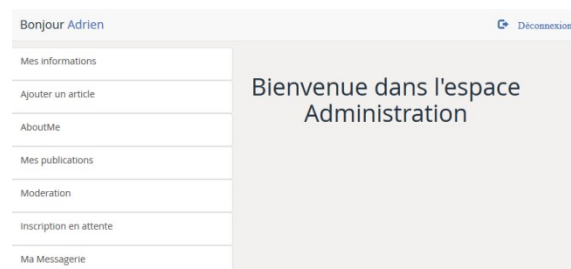
```
elseif ($_GET['action'] == 'session') {  
    session();  
}
```

```
function session() {  
    $ses = new ConnexionManager();  
    $resultat = $ses->getFirstname();  
  
    $recup = $resultat->fetch();  
    session_start();  
    $_SESSION['prenom'] = $recup[2];  
    header ("Refresh: 0;URL=index.php?action=backoffice");  
}
```

```
elseif ($_GET['action'] == 'backoffice') {  
    backoffice();  
}
```

```
function backoffice() {  
    require('view/frontend/backoffice/backoffice.php');  
}
```

Interface du back office :



****You will find a file « identifiant » in the dossier E4/fiche2 to be able to access.****

Partie Ajout :

Dans la partie précédente, nous avons créé l'interface générale du back-office qui sert de **noyau**. Maintenant nous allons ajouter de nouvelles fonctions. Cette partie va concerner la partie ajout d'article.

Le but étant pour notre client d'ajouter un article par le biais du back-office et plus comme dans le passé en passant par l'interface web de phpmyadmin. Le menu est prédéfini mais est comme « une coquille vide ». Pour le moment on ne peut rien faire car l'ajout des fonctions se fera au fur et à mesure. Nous allons commencer par activer le lien qui, sur l'action d'un simple clic, va aller à l'index du site.

Extrait du document :

```
elseif ($_GET['action'] == 'boPageAjout') {  
    boPageAjout();  
}
```

On appelle la fonction et on va sur le contrôleur.

Extrait du document :

```
function boPageAjout() {  
    require('view/frontend/backoffice/ajoutArticle.php');  
}
```

Le contrôleur affiche la page concernée.

The screenshot shows a web application interface. On the left is a sidebar menu with the following items: 'Bonjour Adrien', 'Mes informations', 'Ajouter un article', 'AboutMe', 'Mes publications', 'Moderation', 'Inscription en attente', and 'Ma Messagerie'. The 'Ajouter un article' item is highlighted. The main content area is titled 'Ajouter un Article :'. It contains a form with two fields: 'Titre *:' with a text input field containing 'Titre de l'article', and 'Article *:' with a rich text editor. The rich text editor has a toolbar with icons for bold, italic, underline, strikethrough, font color, background color, bulleted list, numbered list, link, unlink, and a help icon. Below the form, there is a note: '* : Tous les champs sont obligatoires.' and a blue 'Publier' button.

Nous avons fait la moitié du parcours. Maintenant il nous reste à enregistrer les données au bon endroit. Dans la partie ajouter un article, nous retrouvons un formulaire qui va enregistrer en mémoire le titre de l'article et son contenu.

Sur l'image au dessus, nous voyons que le champ de saisie du contenu de l'article est un peu différent. C'est tout à fait normal, nous avons fait appel à l'éditeur de texte Summernote. Summernote en quelques mots, c'est quoi ? C'est un éditeur WYSIWYG (What You See Is What You Get) traduit par "ce que vous voyez c'est que vous faites". Le client pourra simplement s'occuper de la mise en page et l'éditeur l'interprétera.

Vous trouverez le code de la page ajoutArticle.php en annexe 4.

Que ce passe t-il quand on clique sur le bouton « publier » ? On va passer par un contrôle en JavaScript qui va enregistrer les données et appeler l'action que l'index doit faire.

Extrait du document :

```
// ajout d'un post
$('#btn_addPost').on("click", function() {
    var title = $('#title').val();
    var content = $('#textarea[name="content"]').html($('#summernote').code());
    $.ajax({
        type: 'POST',
        url: $('#formAddPost').attr('action'),
        data: $('#formAddPost').serialize(),
        dataType: 'html',
        success: function(code_html, statut, data){
            window.location.replace('index.php?action=addPost&title=' + title + '&content=' + content + '&code=' + code);
        },
        error: function(resultat, statut, erreur){
        },
        complete: function(resultat, statut){
        }
    });
});
```

Extrait du document :

```
elseif ($_GET['action'] == 'addPost') {
    if(!empty($_POST['title']) && !empty($_POST['content'])) {
        addPost($_POST['title'], $_POST['content']);
    }
}
```

Petit contrôle pour savoir si les données sont bien envoyées et appel de la fonction se trouvant dans le contrôleur.

Extrait du document :

```
function addPost($title, $content) {
    session_start();
    $prenom = $_SESSION['prenom'];

    $ajout = new PostManager();
    $ajout_article = $ajout->addPost($title, $content, $prenom);
    if (!$ajout_article) {
        echo 'Impossible d\'ajouter l\'article pour le moment!';
    }
}
```

```

    echo 'Vous allez être redirigé dans 3 secondes.<br>';
    header ("Refresh: 3;URL=index.php?action=boPageAjout");
}
else {
    echo 'Réussite de l\'ajout de l\'article!<br>';
    echo 'Vous allez être redirigé dans 3 secondes.<br>';
    header ("Refresh: 3;URL=index.php?action=backoffice");
}
}
}

```

La fonction reçoit en entrée deux variables. Elle récupère ensuite le nom de l’auteur qui pour le moment n’est exclusivement que notre client et appelle le modèle PostManager.php.

Extrait du document :

```

public function addPost($title, $content, $prenom)
{
    $db = $this->dbConnect();

    $req = $db->prepare('INSERT INTO posts (title, content, creation_date, author) values(?, ?, NOW(), ?)');
    $req->execute(array($title, $content, $prenom));

    return $req;
}

```

Vous trouverez le modèle complet en annexe 5.

Une requête d’insertion va ajouter dans la table réservée, un nouvel article.
Il ne reste plus qu’à aller sur la page « blog » du site pour voir notre dernier article saisie.

Back-office version 2

Bonjour à tous,

le back-office du site est enfin de retour avec bien sur la refonte de l'ancienne version en modèle MVC, la refonte de l'affichage avec BOOTSTRAP et enfin l'ajout de nouvelles fonctionnalités.

Dans le but de préparer l'accès progressif de tout le monde dans cet espace réservé, j'ai en tête les niveaux de privilège permettant en fonction de son grade d'avoir plus ou moins de fonctionnalités. Le lancement de ces nouveaux accès fera l'objet d'un article très prochainement.

Pour le moment, le back-office donne accès à :

- Les informations personnelles avec la possibilité de les modifier,
- La possibilité d'ajouter un article (et/ou un commentaire très prochainement),
- L'ensemble des publications de l'utilisateur avec possibilité de modifier et supprimer,
- Un onglet modération pour contrôler facilement les derniers ajouts (modérateur uniquement),
- Toutes les inscriptions en attente avec la possibilité d'ajouter dans la base ou de supprimer (modérateur uniquement),
- Tout les messages envoyés par le biais du formulaire contact (modérateur uniquement).

De nouvelles fonctionnalités font leurs apparitions grâce à JQUERY:

- des **collapse** : Quand l'utilisateur fait un clic sur un bouton par exemple permet d'afficher une nouvelle zone. Dans le cas du back-office, il y aura des collapse quand vous voulez modifier (affichant le post en mode édition) et quand vous voulez supprimer avec l'apparition d'un nouveau bouton suppression précédé de "Confirmer la suppression ?".
- des éditeur **Summernote WYSIWYG** : WYSIWYG = What You See Is What You Get traduit par "ce que vous voyez c'est que vous faites". Concrètement, l'ajout de texte jusqu'à présent était assez limité dans des input et de rare textarea. La mise en page était très simple. Maintenant nous embrassons le monde moderne avec cet éditeur permettant comme sur n'importe quel site de blog ou de forum d'écrire normalement comme dans un éditeur de texte classique. L'éditeur interprète notre saisie et l'enregistre tel quel dans la base. Vous le retrouverez dans Ajouter un article et bien sur dans mes publications quand vous souhaitez modifier.

Voilà c'est tout pour cette semaine, j'essaie de vous expliquer très simplement mon travail et de vous en faire profiter. N'hésitez à me laisser un commentaire ou me contacter, je me ferai plaisir de vous répondre.

Crée le 11/02/2018 à 11h23min43s
par Adrien

Partie Messagerie :

Nous arrivons à la dernière partie du projet qui est la partie messagerie. Nous allons retrouver dans cette page tous les messages que les utilisateurs ont laissé sur notre site dans la page contact précisément. Le travail sera un peu différent de la partie précédente car il va consister à faire apparaître directement les données dans le but de s'en informer bien sur mais aussi de les supprimer.

En cliquant sur le lien de messagerie, nous allons tout droit à l'index dans le but de faire apparaître la page.

Extrait du document :

```
elseif ($_GET['action'] == 'messagerie') {  
    messagerie();  
}
```

La fonction messagerie() est appelé dans le contrôleur.

Extrait du document :

```
function messagerie() {  
    $messagerie = new MessagerieManager();  
    $messages = $messagerie->getMessages();  
  
    require('view/frontend/backoffice/messagerie.php');  
}
```

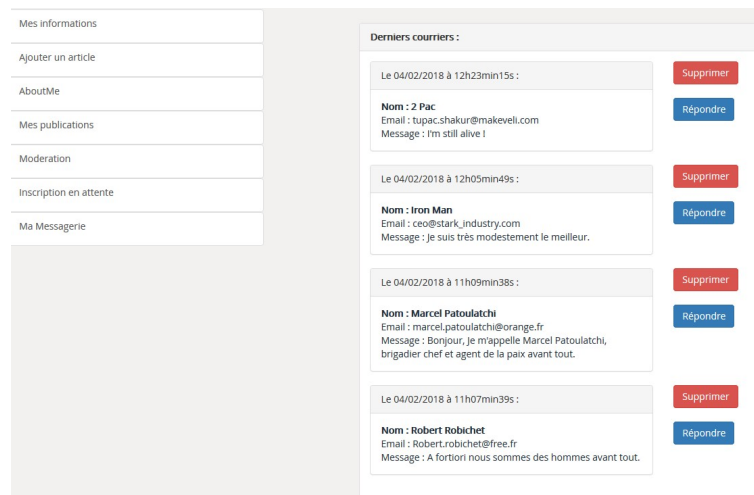
Ici, nous voyons que l'on instancie un objet de classe messagerie qui va avoir comme données tout le contenu de la table email.

Extrait du document :

```
public function getMessages()  
{  
    $db = $this->dbConnect();  
  
    $mail = $db->query('SELECT id_messagerie, name, firstname, email, message, DATE_FORMAT(email_date,  
\'%d/%m/%Y à %Hh%imin%ss\') AS email_date_fr FROM email ORDER BY email_date DESC');  
  
    return $mail;  
}
```

[Vous trouverez le modèle MessagerieManager.php en annexe 6.](#)

Ce dernier va faire une requête de lecture de la table en changeant le format de la date pour apparaître en français et renvoi les données dans le contrôleur qui va se charger d'appeler la vue et afficher toute les données qu'il a reçu.



Nous arrivons au but de cette dernière partie mais il reste une dernière chose à faire, la possibilité de supprimer le mail. En cliquant sur le bouton supprimer, une étape de transition apparaît demandant la confirmation de suppression.



En cliquant sur annuler, cette partie disparaît. C'est pour information un bouton collapse qui affiche ou cache des données sur simple clic sur le bouton.

En cliquant sur supprimer, on va récupérer l'identifiant du message en question et l'envoyer dans l'index par le biais de l'action du formulaire.

Extrait du document :

```
elseif ($_GET['action'] == 'delete_mail') {
    delete_mail($_POST['id_messagerie']);
}
```

L'appel de la fonction `delete_mail` avec l'identifiant du message va dans le contrôleur.

Extrait du document :

```
function delete_mail($id_messagerie) {
    $delete = new MessagerieManager();
    $delete_mail = $delete->delete_mail($id_messagerie);
    if (!$delete_mail) {
        echo 'Impossible de supprimer le mail pour le moment!';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=messagerie");
    }
    else {
        echo 'Suppression réussie!<br>';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=messagerie");
    }
}
```

La fonction fait appel au modèle qui va faire une requête de suppression si la recherche trouve bien une entrée avec le même identifiant

Extrait du document :

```
public function delete_mail($id_messagerie)
{
    $db = $this->dbConnect();

    $deleteMail = $db->prepare('DELETE FROM email WHERE id_messagerie = ?');
    $deleteMail->execute(array($id_messagerie));

    return $sendMail;
}
```

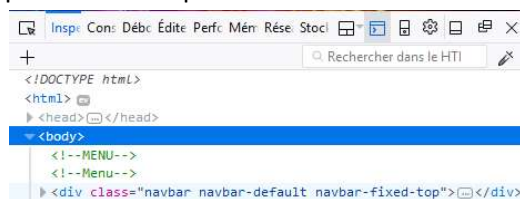
Que la requête aboutisse ou pas, un message d'erreur ou de succès apparaît et une redirection vers la page de messagerie se fait automatiquement pour voir le résultat.

Le projet est maintenant terminé. Avant de passer le back-office en **production**, c'est-à-dire de le mettre en ligne, **des tests sont effectués** pour contrôler la bonne tenue des fonctions du back-office.

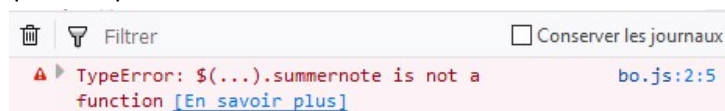
3. Tests et problèmes rencontrés :

- Non affichage de l'éditeur de texte summernote.

Pour rechercher l'erreur, j'ai utilisé « **l'outil de développement** » de mon navigateur pour en savoir plus :



L'erreur apparaît directement, c'est un problème de compatibilité du summernote qui n'est pas reconnu.



J'ai solutionné le problème en ajoutant un lien CSS dans la partie haute du document qui a apporté tous les éléments pour pouvoir utiliser l'éditeur.

```
<link href="//cdnjs.cloudflare.com/ajax/libs/summernote/0.8.9/summernote.css" rel="stylesheet">
```

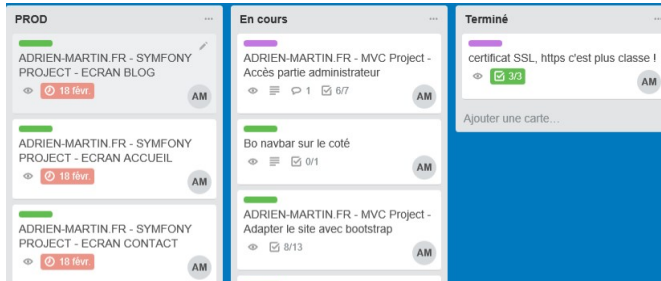
- Suppression d'un message n'étant pas celui demandé
Après contrôle du fichier de messagerie du back-office, il a été révélé que l'id de chaque message n'était pas placé dans le formulaire de suppression.

Une fois la correction effectuée, La suppression du bon message fonctionne correctement et pas d'autres erreurs sont à signaler.

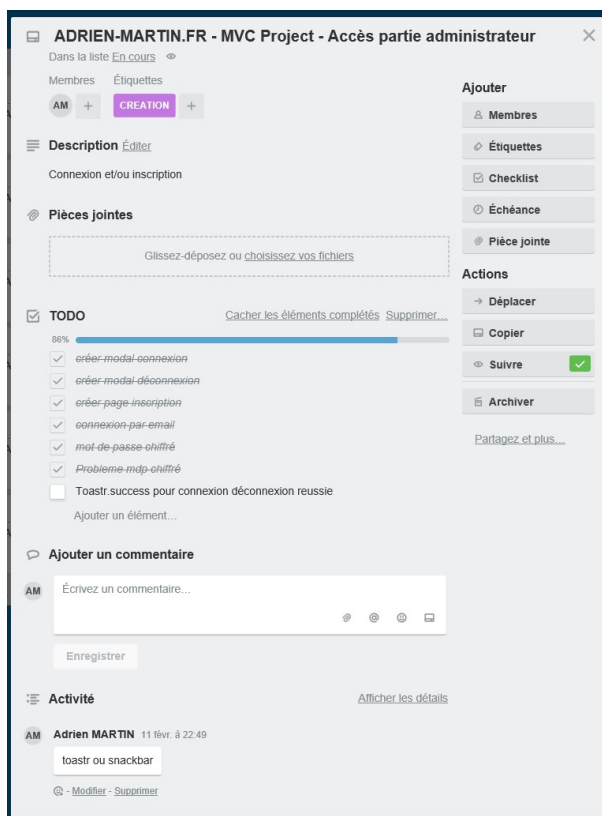
4. Organisation du travail :

Pour bien structurer le travail, j'ai utilisé le logiciel **TRELLO** qui est un outil de gestion de projet en ligne. Cet outil reprend l'idée du tableau accroché au mur avec des post-it de tâches réparties dans des colonnes (à faire, en cours ou terminé) mais au format numérique avec un système de cartes très pratique.

Extrait du tableau TRELLO concernant le site adrien-martin.fr :



Nous retrouvons bien ici les colonnes PROD (à faire), en cours et terminé. A l'intérieur, nous créons autant de carte qu'il y aura de tâche à faire. Pour rappel dans notre mission, nous avons 4 tâches (connexion, Structure, Ajout et Messagerie). A l'intérieur de chacune de ces cartes, nous trouvons le détail de la tâche avec les divers points à traiter et l'avancement de la tâche.



Il y a beaucoup de possibilités avec cet outil et dans le but de faire un travail collaboratif ou plutôt d'avoir un outil simple de **retour d'expérience utilisateur**, il est convenu avec le client de créer une carte « report de bug » qui sera accessible par le client pour qu'il notifie tous les problèmes rencontrés à l'utilisation et qui n'aurait pas été vu dans la phase de test.

Evaluation de la mission :

Je suis monté en compétence dans cette mission qui est fortement inspirée de mon stage de deuxième année de BTS.

L'objectif de mon stage était la refonte du back-office du site Zelift.com. Après entretien de stage concluant, j'ai eu un mois pour faire évoluer mes compétences. J'ai forcé mon apprentissage sur le framework Symfony pour pouvoir l'utiliser correctement et bien sur travailler mes connaissances en PHP et Javascript. En passant mon site personnel en architecture MVC, ce modèle n'a plus de secret pour moi.

Premier jour de stage, j'ai fait connaissance de Putty, mais aussi de m'exercer à la ligne de commande pour utiliser un logiciel de gestion de version (GIT). Je ne compte plus les commit, push et checkout que j'ai fait ! J'ai travaillé sur un IDE très sympathique (PHPStorm) qui est payant. Le framework Symfony est vraiment génial et je regrette de ne pas avoir eu assez de temps pour présenter une mission sur ça. J'ai appris à utiliser TWIG qui est un moteur de template. J'ai appris à utiliser MongoDB Qui est un système de gestion de base de données orientée documents et faisant partie de la mouvance NoSQL. Je suis arrivé en stage avec des connaissances de base en JavaScript et j'ai du travailler dur pour utiliser JQUERY qui même si cela simplifie le code, il faut prendre le temps de le comprendre. J'ai même fait un peu d'AJAX.

En dehors de tout ce que j'ai appris sur la programmation, ce stage m'a permis de découvrir le monde des start-up et de mettre en application une bonne partie de la méthode AGILE. Une SCRUM (réunion de 10minutes max) tous les matins pour faire le point des tâches de la veille et énumérer celles du jour. Utilisation de TRELLO pour l'organisation du travail. Mise à jour de ce dernier à chaque fin de tâche (durant en moyenne deux heures). Fin de journée, établir un « burn down chart » qui est une courbe montrant l'avancement du projet.

Ce que j'ai lu plus apprécié en plus de tous les points énumérés juste avant, a été de travailler en équipe sur un même projet. Sur mes deux années à étudier seul chez moi avec mes cours par correspondance du CNED, j'ai pu découvrir ce qu'est le travail collaboratif.

Pour résumer cette année, elle a été une année charnière me permettant de confirmer mon souhait de poursuivre dans cette voie. J'ai beaucoup appris mais cela reste très peu comparé à tout ce qu'il y a à savoir en développement informatique. La formation ne s'arrêtera jamais et je serai tout le temps entrain d'évoluer mes compétences dans un univers en perpétuel changement. J'ai progressé en développement Web et je l'espère, dès l'année prochaine, pouvoir progresser et monter en compétences sur la programmation logicielle.

Lexique :

HTML (HyperText Markup Language): est le langage de balisage conçu pour représenter les pages web.

CSS (Cascading Style Sheets) : langage informatique qui décrit la présentation des documents.

TEMPLATE (modèles ou patrons en français): Dans notre mission, cela concerne une vue générique. Une sorte de page générique comprenant la structure visuelle du site mais vide de contenu qui sera remplie en fonction des données reçues du contrôleur.

Charte graphique : Le but de la charte graphique est de conserver une cohérence graphique dans les réalisations graphiques d'une même organisation.

JavaScript : est un langage de scripts employé dans les pages web interactives.

JQUERY : est une bibliothèque JavaScript créée pour faciliter l'écriture de scripts côté client.

PHP (Hypertext Preprocessor): langage de programmation principalement utilisé pour produire des pages Web dynamiques.

PHPMYADMIN : Application Web de gestion pour les SGBD (Systèmes de Gestion de Base de Données) MySQL.

MySQL : Système de gestion de bases de données relationnelles (SGBDR).

Framework : Ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.

Architecture MVC (Modèle-vue-contrôleur): Architecture logicielle destinée aux interfaces graphiques pour les applications web.

Classe objet : structure interne des données, Dans ce cas, on parle d'instances de classe.

Getter : Accesseurs permettant de récupérer la valeur de données.

Curseur (ou pointeur):variable contenant une adresse mémoire.

L'outil de développement : Dans notre cas, c'est un outil permettant d'inspecter les pages Web.

Retour d'expérience utilisateur (REX): Ce sont les retours (FEEDBACK) des utilisateurs concernant l'utilisation ou le report de bug d'une application.

Annexe 1 – fichier Index :

```
<?php
require('controller/frontend.php');

if (isset($_GET['action'])) {
    if($_GET['action'] == 'aboutme') {
        aboutMe();
    }
    elseif ($_GET['action'] == 'listPosts') {
        listPosts();
    }
    elseif ($_GET['action'] == 'post') {
        if(isset($_GET['id']) && $_GET['id'] > 0) {
            post();
        }
        else {
            echo 'Erreur : aucun identifiant de billet envoyé';
        }
    }
    elseif ($_GET['action'] == 'cv') {
        cv();
    }
    elseif ($_GET['action'] == 'addComment') {
        if(isset($_GET['id']) && $_GET['id'] > 0) {
            if(!empty($_POST['author']) && !empty($_POST['comment'])) {
                addComment($_GET['id'], $_POST['author'], $_POST['comment']);
            }
            else {
                echo 'Erreur : tous les champs ne sont pas remplis !';
            }
        }
        else {
            echo 'Erreur : aucun identifiant de billet envoyé pour ajouter un commentaire';
        }
    }
    elseif ($_GET['action'] == 'contact') {
        contact();
    }
    elseif ($_GET['action'] == 'addContact') {
        if(!empty($_POST['nom']) && !empty($_POST['prenom']) && !empty($_POST['email']) && !empty($_POST['message'])) {
            addContact($_POST['nom'], $_POST['prenom'], $_POST['email'], $_POST['message']);
        }
        else {
            echo 'Erreur : tous les champs ne sont pas remplis !';
        }
    }
    elseif ($_GET['action'] == 'connexion') {
        if(!empty($_GET['email']) && !empty($_GET['password'])) {
            addConnexion($_GET['email'], $_GET['password']);
        }
        else {
            echo 'Erreur : Login et/ou password ne sont pas remplis !';
            header ("Refresh: 3;URL=index.php?action=contact");
        }
    }
    elseif ($_GET['action'] == 'backoffice') {
        backoffice();
    }
    elseif ($_GET['action'] == 'messagerie') {
        messagerie();
    }
    elseif ($_GET['action'] == 'delete_mail') {
        delete_mail($_POST['id_messagerie']);
    }
    elseif ($_GET['action'] == 'session') {
        session();
    }
    elseif ($_GET['action'] == 'boPageAjout') {
        boPageAjout();
    }
    elseif ($_GET['action'] == 'addPost') {
        if(!empty($_POST['title']) && !empty($_POST['content'])) {
            addPost($_POST['title'], $_POST['content']);
        }
    }
}
else {
    listPosts();
}
```

Annexe 2 – fichier contrôleur :

```
<?php
require_once('model/PostManager.php');
require_once('model/CommentManager.php');
require_once('model/AboutMeManager.php');
require_once('model/ContactManager.php');
require_once('model/ConnexionManager.php');
require_once('model/MessagerieManager.php');
require_once('model/RegisterManager.php');

function aboutMe()
{
    $aboutMeManager = new AboutMeManager();
    $aboutMe = $aboutMeManager->getAboutMe();

    require('view/frontend/aboutMe.php');
}

function listPosts() {
    $postManager = new PostManager();
    $posts = $postManager->getPosts();

    require('view/frontend/listPostsView.php');
}

function post() {
    $postManager = new PostManager();
    $commentManager = new CommentManager();

    $post = $postManager->getPost($_GET['id']);
    $comments = $commentManager->getComments($_GET['id']);

    require('view/frontend/postView.php');
}

function addComment($postId, $author, $comment) {
    $commentManager = new CommentManager();

    $affectedLines = $commentManager->postComment($postId, $author, $comment);

    if($affectedLines === false) {
        echo 'Impossible d\'ajouter le commentaire !';
    }
    else {
        header('Location: index.php?action=post&id=' . $postId);
    }
}

function cv() {
    require('view/frontend/cv.php');
}

function contact() {
    require('view/frontend/contact.php');
}

function addContact($name, $firstname, $email, $message) {
    $mailManager = new ContactManager();
    $addMail = $mailManager->addMail($name, $firstname, $email, $message);

    if($addMail === false) {
        echo 'Impossible d\'ajouter votre message !';
        echo 'Vous allez être redirigé vers la page contact dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=contact");
    }
    else {
        echo 'Message envoyé !<br>';
        echo 'Vous allez être redirigé vers l\'accueil dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php");
    }
}

function addConnexion($email, $password){
    $pass_hache = password_hash($password, PASSWORD_DEFAULT);
    $cnx = new ConnexionManager();
    $resultat = $cnx->getConnexion($email, $pass_hache);

    if (!$resultat) {
```

```

        echo 'Impossible de vous connecter!';
        echo 'Veuillez vérifier vos identifiants !';
        echo 'Vous allez être redirigé vers la page connexion dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=connexion");
    }
    else {
        session_start();
        $_SESSION['email'] = $email;
        header ("Refresh: 0;URL=index.php?action=session");
    }
}

function session() {
    $ses = new ConnexionManager();
    $resultat = $ses->getFirstname();

    $recup = $resultat->fetch();
    session_start();
    $_SESSION['prenom'] = $recup[2];
    header ("Refresh: 0;URL=index.php?action=backoffice");
}

function backoffice() {
    require('view/frontend/backoffice/backoffice.php');
}

function messagerie() {
    $messagerie = new MessagerieManager();
    $messages = $messagerie->getMessages();

    require('view/frontend/backoffice/messagerie.php');
}

function deconnexion() {
    session_destroy();
    echo 'Vous êtes déconnecté !<br>';
    echo 'Vous allez être redirigé vers l\'accueil dans 3 secondes.<br>';
    header ("Refresh: 3;URL=index.php");

    //suppression des cookies de connexion automatique
    //setcookie('login', "");
    //setcookie('pass_hache', "");
}

function delete_mail($id_messagerie) {
    $delete = new MessagerieManager();
    $delete_mail = $delete->delete_mail($id_messagerie);
    if (!$delete_mail) {
        echo 'Impossible de supprimer le mail pour le moment!';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=messagerie");
    }
    else {
        echo 'Suppression réussie!<br>';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=messagerie");
    }
}

function boPageAjout() {
    require('view/frontend/backoffice/ajoutArticle.php');
}

function addPost($title, $content) {
    session_start();
    $prenom = $_SESSION['prenom'];

    $ajout = new PostManager();
    $ajout_article = $ajout->addPost($title, $content, $prenom);
    if (!$ajout_article) {
        echo 'Impossible d\'ajouter l\'article pour le moment!';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=boPageAjout");
    }
    else {
        echo 'Réussite de l\'ajout de l\'article!<br>';
        echo 'Vous allez être redirigé dans 3 secondes.<br>';
        header ("Refresh: 3;URL=index.php?action=backoffice");
    }
}
}

```

Annexe 3 - fichier modèle :

ConnexionManager.php :

```
<?php
require_once("model/Manager.php");

class ConnexionManager extends Manager
{
    public function getConnexion($email, $password){
        $db = $this->dbConnect();

        $connexion = $db->prepare('SELECT * FROM users WHERE email = ? AND password = ?');
        $connexion->execute(array($email, $password));

        $data = $connexion->fetch();

        return $data;
    }

    public function getFirstname() {
        session_start();
        $email = $_SESSION['email'];

        $db = $this->dbConnect();
        $prenom = $db->prepare('SELECT * FROM users WHERE email = ?');
        $prenom->execute(array($email));

        return $prenom;
    }
}
```

Manager.php

```
<?php
class Manager
{
    protected function dbConnect()
    {
        $DBhost = "#####.mysql.db";
        $DBowner = ".....";
        $DBpw = ".....";
        $DBName = $DBowner;
        $DBPort = "....";

        $pdo_options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES UTF8');
        $DBconnect = "mysql:dbname=".$DBName.";host=".$DBhost.";port=".$DBPort;
        $pdo = new PDO($DBconnect, $DBowner, $DBpw, $pdo_options);

        return $pdo;
    }
}
```

Annexe 4 – ajoutArticle.php :

```
<?php
session_start();
?>

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="public/css/styleAbout.css" />
    <link rel="stylesheet" href="public/bootstrap/css/bootstrap.css" />
    <link rel="stylesheet" href="public/css/stylePosts.css" />
    <title>Back-office - Ajouter un Article</title>
  </head>
  <body>
    <?php include 'menu2.php'; ?>
    <div class="container-fluid">
      <div class="row">
        <div class="col-5 col-md-4 col-lg-3 col-xl-1 bd-sidebar">
          <?php include 'navBar.php'; ?>
        </div>
        <div class="col-6 col-md-7 col-lg-7 col-xl-7 bd-content" role="main">
          <div class="row">
            <div class="col-md-2"></div>
            <div class="col-md-10">
              <div class="panel panel-default">
                <div class="panel-heading">
                  <strong>Ajouter un Article :</strong>
                </div>
                <div class="panel-body">
                  <form id="formAddPost" action="index.php?action=addPost" method="post">
                    <div class="form-group">
                      <label for="title">Titre * : </label>
                      <input type="text" id="title" name="title" required placeholder="Titre de l'article" class="form-control">
                    </div>
                    <div id="error_titre" class="alert alert-danger" style="display: none;">
                      Veuillez saisir un titre.
                    </div>
                    <div class="form-group">
                      <label for="content">Article * : </label>
                      <textarea id="content" name="content" required placeholder="Corps de l'article" class="form-control"
rows="5"></textarea>
                    </div>
                    <div id="error_message" class="alert alert-danger" style="display: none;">
                      Veuillez saisir un article.
                    </div>
                    * : Tous les champs sont obligatoires.
                    <div><br>
                      <button type="submit" id="btn_addPost" name="btn_addPost" class="btn btn-primary">Publier</button>
                    </div>
                  </form>
                </div>
              </div>
            </div>
          </div>
          <div class="col-12 col-md-2 col-lg-2 col-xl-2 bd-toc"></div>
        </div>
      </div>
    </div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="public/bootstrap/js/bootstrap.js"></script>
    <script src="public/js/bo.js"></script>
  </body>
</html>
```

Annexe 5 – PostManager.php :

```
<?php
require_once("model/Manager.php");

class PostManager extends Manager
{
    public function getPosts()
    {
        $db = $this->dbConnect();

        $req = $db->query('SELECT id, title, content, DATE_FORMAT(creation_date, \'%d/%m/%Y à %Hh%imin%ss\') AS
creation_date_fr, author FROM posts ORDER BY creation_date DESC');

        return $req;
    }

    public function getPost($postId)
    {
        $db = $this->dbConnect();

        $req = $db->prepare('SELECT id, title, content, DATE_FORMAT(creation_date, \'%d/%m/%Y à %Hh%imin%ss\')
AS creation_date_fr, author FROM posts WHERE id = ?');
        $req->execute(array($postId));
        $post = $req->fetch();

        return $post;
    }

    public function addPost($title, $content, $prenom)
    {
        $db = $this->dbConnect();

        $req = $db->prepare('INSERT INTO posts (title, content, creation_date, author) values(?, ?, NOW(), ?)');
        $req->execute(array($title, $content, $prenom));

        return $req;
    }
}
```


Annexe 6 - MessagerieManager.php :

```
<?php
require_once("Manager.php");

class MessagerieManager extends Manager
{
    public function getMessages()
    {
        $db = $this->dbConnect();

        $mail = $db->query('SELECT id_messagerie, name, firstname, email, message, DATE_FORMAT(email_date,
\'%d/%m/%Y à %Hh%imin%ss\') AS email_date_fr FROM email ORDER BY email_date DESC');

        return $mail;
    }

    public function delete_mail($id_messagerie)
    {
        $db = $this->dbConnect();

        $deleteMail = $db->prepare('DELETE FROM email WHERE id_messagerie = ?');
        $deleteMail->execute(array($id_messagerie));

        return $sendMail;
    }
}
```