

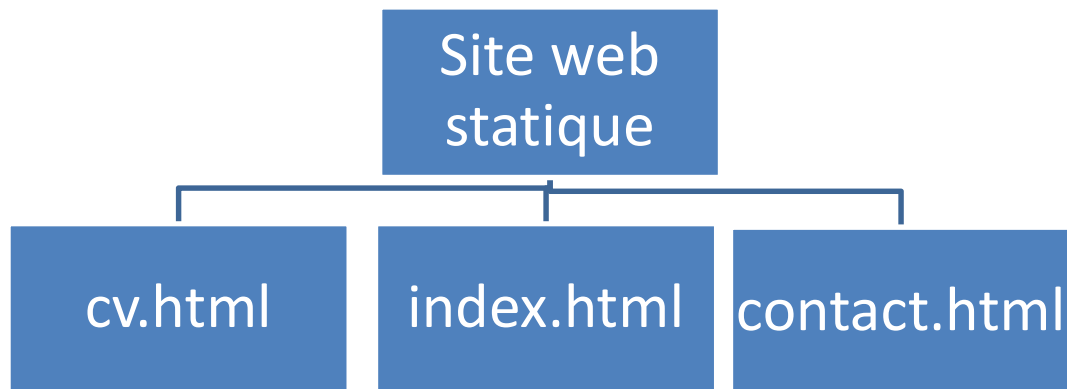
REFONTE DU SITE INTERNET ADRIEN-MARTIN.FR

En site dynamique avec le modèle MVC

Sommaire :

Architecture du site 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 5
3. Tests et problèmes rencontrés :	Page 9
4. Organisation du travail :	Page 11
EVALUATION DE LA MISSION :	Page 12
LEXIQUE :	Page 13
ANNEXES :	Page 14

Architecture du site au début de la mission :



L'architecture est basique. L'`index.html` sert d'index du site et de page de présentation. L'utilisateur peut aller sur la page `cv.html` et `contact.html` par le biais d'un menu intégré dans la page. La page contact ne sert que de lien vers des réseaux sociaux généraliste et professionnel. Les trois pages sont composées d'un langage de balise (HTML) et d'un langage de présentation (CSS) gérant la partie esthétique des pages.

But de la mission :

Préparer le site pour de futures améliorations visant à professionnaliser la gestion du contenu.

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 faire évoluer le site en rajoutant des modules comme par exemple un blog ou bien un forum dans un projet ultérieur et souhaite rendre dynamique son site.

Proposition d'une ou plusieurs solutions :

Il est évoqué deux solutions pour faire évoluer le site du client vers une version dynamique et avec une architecture préparant l'ajout de nouveaux modules plus simple :

- Refonte du site sur un **Framework** PHP de type « Symfony » supportant l'**architecture MVC** et étant très complet.

L'**avantage** est d'avoir une base très solide pour une évolution rapide du site.

L'**inconvénient** est qu'il faut des compétences solides dans l'utilisation du framework.

- Refonte du site sur une architecture MVC simple permettant de faire des modifications rapidement et d'ajouter des nouveaux modules de façon simple.

L'**avantage** est de structurer le code sur le principe d'un framework sans en subir toute la complexité et les nombreuses options non utiles pour le moment.

L'**inconvénient** est qu'il reste une solution provisoire ou plutôt intermédiaire permettant l'évolution et l'ajout de module mais ne remplaçant pas un framework complet.

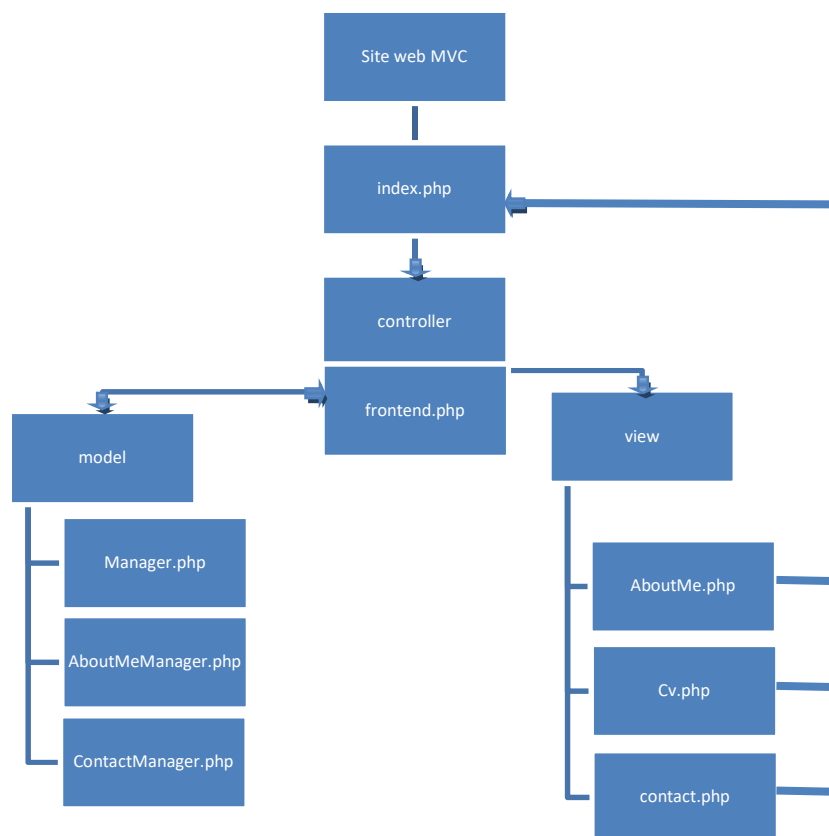
Après concertation avec le client, il est décidé que même si la version sur framework sera l'objet d'une mission future, le besoin actuel du client sur son site et les futurs besoins à venir ne sont pas aussi important pour justifier de mettre en place un important système **qualitatif**. **Le délai**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

La deuxième 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 sur le principe du modèle MVC.

Cela peut paraître plus complexe au premier abord et pourtant cela simplifie le développement. Fini la recherche d'erreurs dans un fichier faisant plus de mille lignes. Le code est séparé, découpé dans plusieurs fichiers dont le rôle est bien précis.

L'utilisateur ne verra sur son navigateur que les vues qui ne sont rien d'autres que les pages internet du site (présentation, cv et contact) et qui changeront en fonction de l'action demandée (accès à une page par exemple).

1. Ordonnancement des tâches

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

Le modèle MVC nous facilite le travail car il propose déjà un découpage que nous allons nous inspirer.

Nous allons découper le projet en non pas 3 tâches mais 4 tâches bien distinctes :

- une partie **modèle** où sera regroupé toute la partie requête SQL ;
- une partie **contrôleur** où sera regroupé le traitement PHP ;
- une partie **vues** où un travail de réorganisation des pages sera fait dans le but de le simplifier au maximum (HTML/CSS).
- Et une partie **index** qui sera l'élément essentiel du projet permettant diriger le site.



2. Partie développement des tâches

Partie Index :

Ce fichier va être la pièce centrale du site car chaque action transitera par ce document.

Vous trouverez le fichier complet en annexe 1.

Extrait du document :

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

if (isset($_GET['action'])) {
    if($_GET['action'] == 'aboutme') {
        aboutMe();
    }
}
```

Ce fichier est en langage PHP et va être une série de conditions if pour déterminer quelle fonction appeler. Le choix du « if » est volontairement choisi vu le nombre faible de conditions.

Nous commençons par préciser que nous souhaitons accéder au contrôleur par le biais de la fonction « require » (qui va préciser le chemin pour y accéder).

Chaque condition va chercher si une action est demandée.

Par exemple dans l'extrait on recherche une action qui s'appelle « aboutme ». Si c'est l'action demandée, alors on va appeler la fonction aboutMe() qui se trouve dans le contrôleur.

```
else {  
    aboutMe();  
}
```

On finira le fichier par un sinon (else) général qui sans action trouvée va appeler « par défaut » la fonction aboutMe().

Cette fin de code sera très pratique par exemple quand on arrive la première fois sur le site.

En effet, nous ne sommes pas censés avoir fait encore une action.

Partie contrôleur :

Ce fichier va être la partie traitement des actions du site. Chaque action trouvée par l'index va appeler une fonction se trouvant sur ce fichier.

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

Extrait du document :

```
<?php  
require_once('model/AboutMeManager.php');  
require_once('model/ContactManager.php');  
  
function aboutMe()  
{  
    $aboutMeManager = new AboutMeManager();  
    $aboutMe = $aboutMeManager->getAboutMe();  
  
    require('view/frontend/aboutMe.php');  
}
```

Ce fichier est en langage PHP et va être une série de fonctions très spécifiques.

Nous commençons par préciser que nous souhaitons accéder au fichier modèle par le biais de la fonction require_once (qui va préciser le chemin pour y accéder et le charger qu'une seule fois).

Dans l'extrait, nous retrouvons la fonction aboutMe() qui va créer un objet rassemblant toutes les données du résultat de la requête dans le but de les exploiter dans la vue.

Une fois les données récupérées, un « require » demande de charger la page aboutMe.php qui est la vue (la partie visible du site) de la page de présentation.

Partie Modèle :

Ce fichier va être la partie requête SQL du site. Chaque requête va faire une action précise (lecture de données, modification ou suppression).

Vous trouverez les modèles complets en annexe 3.

Extrait du document :

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

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

        $abouts = $db->query('SELECT * FROM about');

        return $abouts;
    }
}
```


Comme précisé plus haut, un modèle n'apparaîtra pas dans l'annexe, le fichier Manager.php. Ce document n'est autre que la partie connexion à la base de données.

Dans cet extrait on voit le contenu du modèle de AboutMeManager.php qui est une définition **d'une classe objet avec un getter** qui va par le biais d'une requête SQL va aller chercher les données concernées dans la fonction getAboutMe().

Pour être concret, on va rechercher le contenu des paragraphes de texte de la page présentation dans une table spécifique. Le client ayant des notions de phpmyadmin, il modifiera dans un premier temps son contenu par le biais de cette interface relativement simple.

Vous trouverez des captures d'écran de l'interface phpmyadmin en annexe 4.

Structure de la table about

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	id 	int(11)		Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	title	varchar(255)	utf8_general_ci	Non	Aucun(e)		
<input type="checkbox"/>	3	content	text	utf8_general_ci	Non	Aucun(e)		

Bien que sommaire, cela reste une excellente préparation à l'ajout d'un blog par exemple.

Partie Vue :

Nous arrivons à la dernière partie du projet qui est l’affichage des pages. Fidèle à la séparation du code que nous avons mené, La vue sera évacuée du langage de présentation CSS pour laisser place au langage de balise HTML et ponctuellement de balise PHP pour l’affichage des données.

La partie CSS sera placée dans un dossier PUBLIC où l’on retrouvera les images du site, le CSS des pages ainsi que les polices et éventuellement les fichiers JavaScript.

Vous trouverez les vues en annexe 5.

Extrait du document :

```
<?php
while ($aboutme = $aboutMe->fetch())
{
?>
    <div class="row">
        <div class="panel panel-default">
            <div class="panel-heading">
                <?= $aboutme['title'] ?>
            </div>
            <div class="panel-body">
                <?= $aboutme['content'] ?>
            </div>
        </div>
    </div>
<?php
$i++;
}
$aboutMe->closeCursor();
$pdo=null;
?>
```

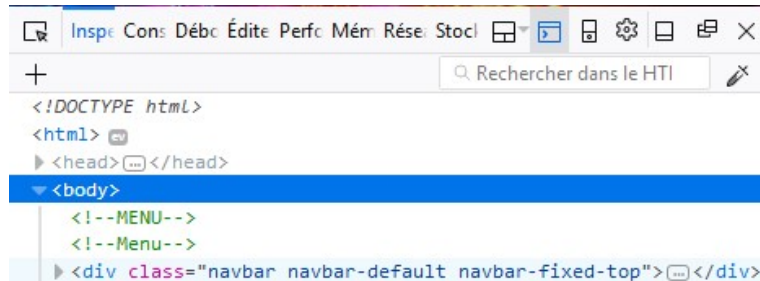
Dans cet extrait nous retrouvons uniquement la partie qui nous intéressent, soit la partie affichage des données.

Nous procédons à une boucle « while » qui va par le biais **d’un curseur** afficher ligne par ligne le résultat soit précisément le titre et le contenu du paragraphe de chaque partie de la page de présentation.

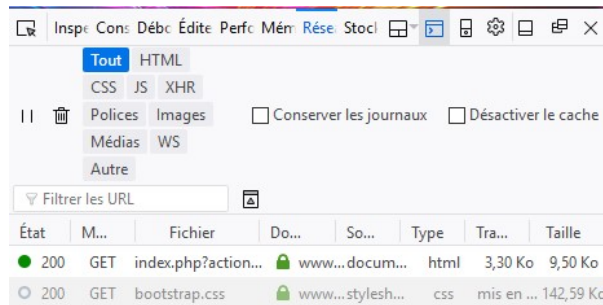
Le projet est maintenant terminé. Avant de passer le site en **production**, c'est-à-dire de le mettre en ligne, **des tests sont effectués** pour contrôler la bonne tenue du site aux diverses actions.

3. Tests et problèmes rencontrés :

- Affichage du site => page blanche.
Premier test d'affichage non concluant. La page est vide. Pour rechercher l'erreur, j'ai utilisé « **l'outil de développement** » de mon navigateur pour en savoir plus :



L'onglet « Réseau » permet de connaître l'état de la page.



L'erreur avait le numéro 404. C'est-à-dire que la page n'existe pas où est introuvable. Après contrôle de l'index.php, il y avait effectivement une simple erreur de syntaxe sur le nom de la fonction à appeler.

Après modification, la page de présentation s'est affichée... en partie.

- Affichage des données de la page de présentation => page en partie vide.
Toute la partie concernant le contenu des paragraphes (titre et contenu) ne s'est pas affiché.

Pour rappel, ce contenu n'est plus inscrit en dur sur la page mais bien en base de données.

Par déduction, l'erreur doit certainement se trouver sur la page contrôleur (traitement des données) ou sur la page modèle (accès à la base de données).

Après l'examen du contrôleur, rien à corriger.

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

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

Pas plus d'erreurs sur le modèle également.

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

    $abouts = $db->query('SELECT * FROM about');

    return $abouts;
}
```

Si la page s'affiche en partie, ce ne peut pas être une erreur dans l'index sinon la page ne s'afficherait pas du tout. Il ne reste plus qu'à regarder du côté de l'affichage de la vue.

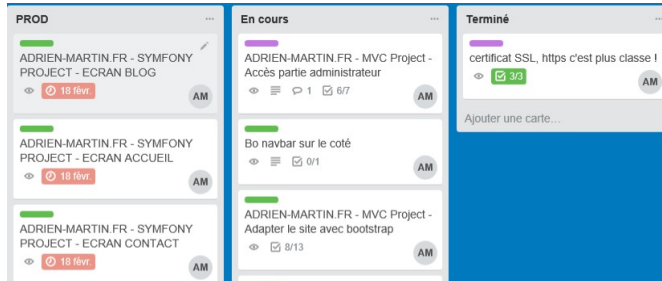
Après examen attentif de ce dernier, ce n'est finalement qu'une erreur d'écriture qui en est à l'origine. Effectivement dans le contrôleur nous avons la variable \$aboutMe qui stocke les informations pour les afficher dans la vue. Mais dans notre cas, dans la vue, \$aboutme s'est mystérieusement transformé en \$about se qui explique que les données ne s'affichaient pas.

Une fois la correction effectuée, Le site fonctionne correctement et pas d'autres erreurs sont à signaler.

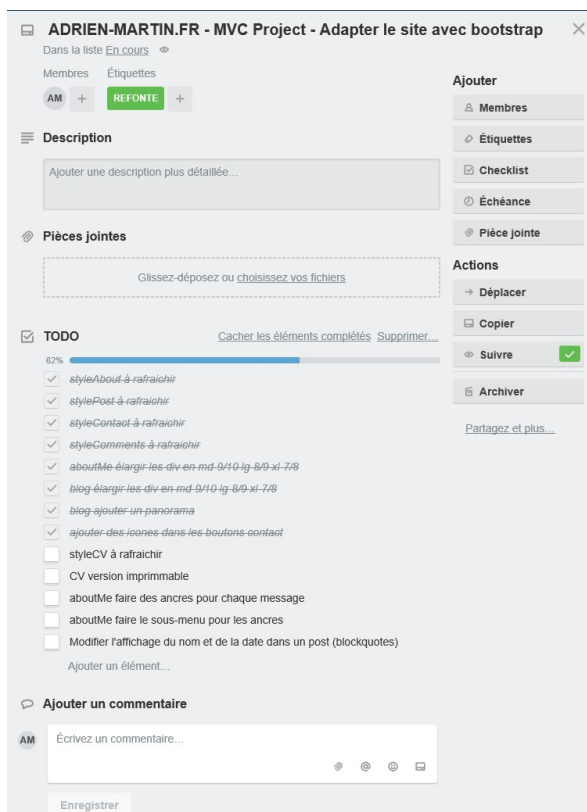
4. Organisation du travail :

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 une carte qui sera une tâche en particulier. Pour rappel dans notre mission, nous avons 4 tâches (index, contrôleur, modèle et vue). 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 Zelifit.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 fonctions des données reçu du contrôleur.

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 destiné 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'] == 'cv') {
        cv();
    }

    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 !';
        }
    }
}
else {
    aboutMe();
}
```

Annexe 2 – fichier contrôleur :

```
<?php
require_once('model/AboutMeManager.php');
require_once('model/ContactManager.php');

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

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

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");
    }
}
```

Annexe 3 - fichier modèle :

AboutmeManager.php :

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

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

        $abouts = $db->query('SELECT * FROM about');

        return $abouts;
    }
}
```

ContactManager.php

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

class ContactManager extends Manager
{
    public function addMail($name, $firstname, $email, $message)
    {
        $db = $this->dbConnect();
        $sendMail = $db->prepare('INSERT INTO email (name, firstname, email, message, email_date) values(?, ?, ?, ?, NOW())');
        $sendMail->execute(array($name, $firstname, $email, $message));

        return $sendMail;
    }
}
```

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 – partie phpmyadmin :

Structure de la table about

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	id	int(11)		Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	title	varchar(255)	utf8_general_ci	Non	Aucun(e)		
<input type="checkbox"/>	3	content	text	utf8_general_ci	Non	Aucun(e)		

Interface de modification des paragraphes existants :

`SELECT * FROM `about``

☐ Tout afficher | Nombre de lignes : 25

+ Options

id	title	content	← T →
1	BREAKING NEWS	Bonjour à tous, je recherche une entreprise qui po...	Supprimer Copier Éditer <input type="checkbox"/>
3	Mon histoire	Comme dit dans l'introduction, j'ai bien 32 ans. J...	Supprimer Copier Éditer <input type="checkbox"/>
4	Mon présent	Même si j'aime le métier que j'occupe actuellement...	Supprimer Copier Éditer <input type="checkbox"/>
5	Mon avenir	Demain se prépare aujourd'hui. Je prépare déjà l'a...	Supprimer Copier Éditer <input type="checkbox"/>

☐ Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

☐ Tout afficher | Nombre de lignes : 25

Opérations sur les résultats de la requête

Imprimer Copier dans le presse-papiers Exporter Afficher le graphique Créer une vue

Interface d'ajout de paragraphe de présentation :

PARCOURIR STRUCTURE SQL RECHERCHER **INSÉRER** EXPORTER II

Colonne	Type	Fonction	Null	Valeur
id	int(11)	<input type="text"/>	<input type="checkbox"/>	<input type="text"/>
title	varchar(255)	<input type="text"/>	<input type="checkbox"/>	<input type="text"/>
content	text	<input type="text"/>	<input type="checkbox"/>	<input type="text"/>

Annexe 5 – fichier vue :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="public/bootstrap/css/bootstrap.css" />
    <link rel="stylesheet" href="public/css/styleAbout.css" />
    <title>About Me</title>
  </head>
  <body>
<!-- MENU -->
    <?php include 'menu.php'; ?>
<!-- CONTENT -->
    <div class="container-fluid">
      <div class="row">
        <div class="col-12 col-md-2 col-lg-2 col-xl-3 bd-sidebar"></div>
        <div class="col-12 col-md-8 col-lg-8 col-xl-6 bd-content" role="main">
          <?php
            while ($aboutme = $aboutMe->fetch())
            {
              ?>
              <div class="row">
                <div class="panel panel-default">
                  <div class="panel-heading">
                    <?= $aboutme['title'] ?>
                  </div>
                  <div class="panel-body">
                    <?= $aboutme['content'] ?>
                  </div>
                </div>
              </div>
              <?php
                $i++;
              }
              $aboutMe->closeCursor();
              $pdo=null;
              ?>
            </div>
            <div class="col-12 col-md-2 col-lg-2 col-xl-3 bd-toc"></div>
          </div>
        </body>
      </html>
```