Certification DWWM

**I- Maquetter une Application :**

Avant de maquetter une application il faut savoir ce que veut le client, il faut donc établir le cahier des charges.

**1-Etablir le cahier des charges :**

Présentation d’ensemble :

Ici le site est destiné pour une personne en statut d’auto-entrepreneur dont l’entreprise a pour but de diffuser du contenu sur internet.  
L’objectif du site est de montrer le contenu proposé et de pouvoir contacter le client via le site.  
La cible du site sont les personnes jeune (17-25 ans) qui s’intéresse aux contenu internet (vidéo YouTube, Stream etc)   
Pour ce projet nous n’avons pas de objectifs quantitatifs précis.  
Le périmètre du projet sont les pays francophones car les contenus sont en langue française.

Description graphique et ergonomique :

Pour la charte graphique nous sommes partis sur les couleurs violet et bleus et bleus foncé pour la partie admin.  
Au niveau du design le logo nous a été fourni (voir sur la diapo) Le choix de la police a été libre. J’ai donc choisi la police Chilanka-Regular pour les barres de navigation, pied de page et certains titres et la police Princetown pour un titre précis et la police Calibri pour le corps du site.  
Enfin je n’ai pas eu de demande particulière pour la disposition des éléments.

Description fonctionnelle et technique :

Pour l’arborescence du site je n’ai eu aucune exigence, j’ai donc choisi l’architecture MVC (Modèle Vue Controlleur).  
Aussi pour les fonctions on détermine les rôles et ce que ceux-ci peuvent faire. Ici il n’y aura qu’un seul rôle qui est le rôle admin, il va permettre de gérer les messages reçus dans le formulaire de contact  
L’authentification se fera grâce à un email et un mot de passe.  
Il faut savoir comment seront ajouté les informations relatives au contenu. Là le contenu (Youtube) sera ajouté grâce notamment à des API.  
Il faudra prendre en compte les contraintes techniques. Pour ce site je n’ai pas eu de contraintes. J’ai donc choisi le PHP et Symfony pour faire le site. Cela permet d’avoir plus d’expérience sur ce Framework.

Prestations attendues :

. Développement  
. Design(sans création des logos et images)  
.Intégration

**2-Maquettage**

Une fois que le cahier des charges est créé, on pourra maquetter le site. Cela permet d’avoir une ébauche. C’est une version simplifiée où l’on pourra voir l’emplacement des blocs de la page. On réalisera une maquette pour la version Mobile, Tablette et PC. Les couleurs sont optionnelles. Ici j’ai utilisé Power Mock uP pour effectuer ces maquettes.

**II-Réaliser une interface utilisateur web statique et adaptable**

**1-Interface web statique**

Après avoir maquetter le projet on effectue une version statique de ce dernier, en utilisant html et CSS pour le style.

Ici on peut voir la structure de la page HTML avec les balises <HTML>,<head>(pour notamment définir le Charset avec <meta>, lier une feuille de style avec la balise <link> ) et la balise <body> qui contient le contenu du site.

Sur la deuxième image on peut voir une partie de la feuille de style que j’ai créée pour notamment intégrer des polices particulières (@font-face), ajouter des styles sur des balises, et créer des classes.

**2-Interface adaptable**

Avec l’expansion des smartphones de plus en plus de monde consulte les sites sur leur mobile, c’est pour cela que lorsqu’on crée un site adaptable on utilise la technique du MOBILE FIRST. On commence par concevoir la partie mobile du site et on développe sur des médias plus grands.

Pour avoir un site adaptable on utilise des Media Queries. Ce sont des styles définis pour des cas particuliers. Par exemple : réduire la police d’écriture pour les écrans ou taille de fenêtre inférieur à 500px.

Aussi pour nous faciliter la tâche des frameworks CSS ont été créés (Ex : Bootstrap)  
Ils sont basés sur un système de « quadrillage » avec des lignes et des colonnes. La longueur est découpée en 12 colonnes. Dans mon exemple la colonne a la classe col-sm-12, donc quand ma longueur est inférieure 540px la colonne à une longueur de 100% (12/12 \*100) et elle a une classe col-lg-6 donc quand ma longueur est supérieure à 992 px ma colonne a une longueur de 50%

**III-Développer une interface web dynamique :**

**1-Intégration des éléments de sécurité :**

Faille XSS=Cross-Site Scripting. C’est l’insertion de script via les formulaires.J’ai utilisé la fonction htmlspecialchars. Celle-ci permet de remplacer les caractères inférieur et supérieur par leur code et donc éviter d’avoir des blocs « <script>…</script> » C’est pour cela que l’on vérifie ce que l’utilisateur rentre avant de pouvoir l’utiliser.

Ici on peut voir que dans mon Entité Contact j’ai appliqué dans mes setters, la fonction htmlspecialchars ainsi que la fonction trim qui permet de supprimer les espaces avant et après (cela permet de gagner de la place). Grâce à ça quand une personne va remplir le formulaire de contact, elle ne pourra pas insérer de script.

Aussi Javascript et notamment JQuery peuvent être utilisé pour afficher de manière dynamique.

**2-Affichage dynamique :**

On peut rendre notre site dynamique avec l’affichage d’éléments. Ici j’ai utilisé implicitement JQuery pour ma barre de navigation. En effet avec Bootstrap on peut créer une navbar qui s’affiche pleinement en grand écran et qui va afficher un bouton qui lorsque l’on clique sur celui-ci une liste des éléments de la barre s’affiche.

**3-API :**

Sur mon projet j’ai rendu le site dynamique en travaillant sur l’API Google. Tout d’abord j’ai inscrit mon application sur le site développeur de Google. Puis j’ai activé l’API YouTube. Je récupère la clé et j’ai utilisé cURL pour récupérer le contenu. Je me suis servi du cache de symfony pour mettre le résultat en cache.

CURLOPT\_HEADER sert à transmettre le header.

CURLOPT\_RETURNTRANSFER sert à retourner le transfert en tant que chaîne de caractère de la valeur retournée par curl\_exec() au lieu d’afficher directement.

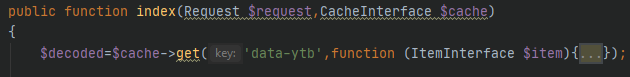
CURLOPT\_URL sert à définir l’url pour récupérer nos informations.

CURLOPT\_FOLLOWLOCATION à 1 sert à suivre tous les en-têtes « Location : » que le serveur envoie.

CURLOPT\_VERBOSE sert à afficher les informations de façon verbeuse quand il est à faux/0.

CURLOPT\_SSL\_VERIFYPEER à false il sert à arrêter cURL de vérifier le certificat du pair.

Un problème s’est posé à moi dans la gestion de l’API. En effet le nombre de requête est limité et à chaque rafraichissement de la page de nouvelles requêtes sont faites. Il faut donc palier à ça et pour cela j’ai utilisé j’ai utilisé le système de cache de Symfony (grâce à un tutoriel) : Avec celui-ci je peux mettre des données et j’ai donc mis la réponse en cache. Donc si ‘data-ytb’ existe on va récupérer les données directement



Le flux YouTube est un flux assez dynamique il faut donc que le cache soit effacé régulièrement. Ici j’ai choisi de nettoyer tous les jours. Pour faire cela, j’ai utilisé un objet ItemInterface qui permet de gérer le cache avec notamment la méthode expire. Ici le temps est exprimé en seconde.



**4-Réupération d’un flux Instagram via un Plugin :**

J’ai utilisé un Plugin Javascript pour récupérer le flux Instagram d’un profil Public. On appelle le fichier Js dont on a besoin. On crée un objet InstagramFeed. L’avantage est qu’il ne nécessite pas d’Access Token

On peut choisir le nombre d’items que l’on veut afficher, le nombre d’items par ligne

**IV-Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce :**

**1-Gestion de contenu:**

Dans mon projet, la gestion de contenu se fait par la gestion des message reçus via le formulaire de contact sur la page Admin. En effet j’ai créé une page pour se connecter en tant qu’administrateur et avoir accès aux messages reçus. J’ai aussi créé un formulaire pour répondre à ces messages. On peut gérer le contenu en mettant les messages à la corbeille, les restaurer, en marquant des messages comme lu ou non lu etc

Description des Entités :  
Ici j’ai deux entités : User pour pouvoir me connecter en tant qu’admin. La classe User contient l’Id, l’email, et le rôle. Je n’ai pas créé de formulaire d’inscription car le ou les admins du site seront ajoutés manuellement en base de données. Et la class Contact qui contient le nom complet, le mail, le sujet, le contenu, un booléen si le message est lu et un booléen s’il est effacé.

Aussi pour gérer le contenu j’ai utilisé un plugin qui permet d’envoyer des mails, c’est SwiftMailer. Ainsi les messages envoyés et reçus sont stockés en base de données et sont envoyés par mail.

**V-Créer une base de données :**

Dans mon projet je n’ai pas utilisé la méthode MERISE (Méthode d’Etude et de Réalisation Informatique par les Système d’Entreprise) car ma base de données est simple. Elle se compose de deux tables User et Contact. Elles ne sont pas liées entre elles. Ma table User sert à se connecter à la partie administration pour gérer les messages du formulaire de contact et ma table Contact contient les messages reçus et envoyés via ce formulaire.

Je créé tout d’abord la base de donnée avec la commande « doctrine :database :create ». Puis on créé notre entité user avec « make :user » Et la table contact avec la commande « make :entity »

Après on effectue la migration avec « make : migration »

Enfin on fait les changements en base de données avec la commande « doctrine : migration :migrate ».

**VI-Développer les composants d’accès aux données :**

J’ai utilisé Symfony dans mon projet, il y a donc des outils pour gérer les données. En effet on utilise les objets et des librairies. Ici j’ai utilisé Doctrine ORM pour faire mes requêtes sur ma base de données :

Pour la sélection on peut observer que j’utilise la méthode déjà implanté findBy et je mets en paramètres un tableau de critère. En requête SQL c’est l’équivalent de:

SELECT \* FROM Contact WHERE isRead =0 and isDeleted=0

Pour mettre à jour, on récupère ce qu’on veut modifier sous forme d’objet, on modifie la valeur que l’on veut avec le setter approprié.

Pour insérer des données on applique la méthode persist à notre nouvel objet contact puis en appliquant la méthode flush on ajoute en base de données.

Pour supprimer des données d’une base de données on récupère ces données, puis on utilise la méthode remove qui se trouve dans le Manager qui lui-même se retrouve dans la Doctrine.

Enfin on peut créer des requêtes personalisées avec la méthode cerateQueryBuilder.  
Dans notre exemple cela permet d’ajouter plusieurs conditions   
Aussi cette méthode permet de faire des jointure avec les méthodes innerJoin(), leftJoin(), rightJoin().  
Enfin dans les méthodes déjà créé telles que findBy on selectionne tout les champs alors que l’on peut choisir ce que l’on veut dans la requête personnalisée. Cela peut être très utile quand on possède des grosses bases de données.

**VII-Développer la partie back-end d’une application web ou web mobile :**

Authentification :

La première étape de la partie back-end est l’accès à la partie administration. Après avoir créer la classe User j’ai crée la page login grâce à la commande « make :auth » Dans le fichier security.yml j’ai ajouté l’access\_control admin :

Donc les pages qui ont /admin sont accessibles seulement pour le rôle admin.

Création des pages avec gestion des données :

On créé les pages grâce à la commande « make :controller » cela nous donne un fichier twig pour le côté vue et un fichier controller qui permet de récupérer les éléments que l’on a besoin.

Ici on peut voir sur cette fonction que le formulaire de recherche a été créé (le SearchType a été créé grâce à la commande « make :form ») Si le formulaire a été soumis et valide on récupère ce qui a été cherché grâce à la méthode get et on lui applique la méthode search que j’ai créé dans le ContactRepository.

Pagination :

Ensuite pour avoir un affichage efficace j’ai choisi de faire une pagination. J’ai utilisé le package « Paginator Bundle » j’ai choisi les messages à afficher et le nombre de messages par page (ici 12) et je renvoie les informations dont j’ai besoin sur la vue correspondante (Si le formulaire est valide et soumis à la page result\_search et sinon à la page admin\_contact)

Dans le fichier de config j’ai réglé le nombre de page affichées et le style de la pagination (lien et template)

Autres fonctions :

Sur ce controller j’ai créé d’autre fonction comme une fonction pour marquer comme lu, non lu, pour mettre à la corbeille.

**Conclusion :**

Durant mon stage j’ai pu aborder beaucoup de points différents, j’ai pu aussi bien travailler le front que le back (même si ma base de données n’était pas complexe) J’ai pu développer mes connaissances sur Symfony. J’ai appris comment mettre des données en cache sur Symfony. J’ai créé un site de A à Z, j’ai pu donc participer à tous les processus de création du site (du Maquettage à la création de l’application) Mon site a donné satisfaction à la personne responsable de mon stage.