**Le dossier de projet respecte ce plan type :**

**- Liste des compétences du référentiel qui sont couvertes par le projet**

**- Résumé du projet en anglais d’une longueur d’environ 20 lignes soit 200 à 250 mots, ou environ 1200 caractères espaces non compris**

Today the centralization of information is very important, because it allows you to minimize the time of search and labor productivity.

In the schools, many times the teachers for their didactic courses need the centralized storage of videos that collaborate in the deepening of the content of a subject, besides that each person in the earth has his personal tastes and with them his favorite videos.

With the aim of solving this problem, BlogFavVideo has been developed, a web system conceived for the users to have the possibility of registering all their favorite videos from youtube within their personal private space by means of a web interface.

To do this the user must first register in the application with your personal data and then log in.

Within his personal space the user has the possibility to insert, modify, delete, consult the selection of all his favorite videos in different pages in order of insertion and modify his personal data.

BlogFavVideo also has a good security system where each user and only he has access to his private account.

This system was developed in Angular 8 as client-side web technology (Frontend), in Symfony 5 as server-side technology (Backend) and as GDBD Mysql.

**- Cahier des charges ou expression des besoins du projet**

**1- Gestion de projet (planning et suivi, environnement humain et technique, objectifs de qualité)**

**2 - Spécifications fonctionnelles du projet  
3 - Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité  
4 - Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité  
5 - Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données  
obtenues)  
6 - Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité. (**como yo hice para poner en practica la seguridad infomatica y porque es importante**)  
7 - Description d’une situation de travail ayant nécessité une recherche et effectuée par le candidat durant le projet (**que hiciste para tener la competance en lo que hisciste tegnologias, forma de conception del projecto etc **)**

1-

2-

3-

4- **Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité.**

**Authentification côté Api-Rest**

**Fonction LoginUser**

/\*\*

\*

\* Login utilisateur

\*

\* @param JwtAuth $jwtAuth

\* @param UserRepository $userRepository

\* @param LoginRepository $loginrepository

\* @param Request $request

\* @param ValidatorInterface $validator

\* @return $data

\*/

public function loginUser(JwtAuth $jwtAuth,

UserRepository $userRepository,

LoginRepository $loginrepository,

Request $request,

ValidatorInterface $validator)

{

}

Fonction qui a une responsabilité indispensable dans l'application puisqu'elle est chargée du contrôle de l'authentification des utilisateurs côté serveur.

**Note :** Une fois que **l'API-REST** a vérifié que l'authentification est correcte, un **token crypté** et un **token non crypté** seront envoyés à **l'interface client** pour être stockés dans le **LocalStorage** du navigateur pendant la navigation sur l'application et retirés du **LocalStorage** lors de la déconnexion.

Le client fera une pétition d'authentification à l’application, cette pétition sera envoyée en utilisant un service (protocole (http, post)) à **l’API-REST**.

Un responsable du traitement (**Controller**) recevra la pétition et vérifiera que le type de données est correct et qu'il existe en utilisant un service**(JWTAUTH)**. S'il existe, un **Token** sera créé avec les données de l'utilisateur enregistré. Ces données pour former le **Token**, seront recherchées dans leurs tables respectives en utilisant l'identifiant (**id**) qui correspond à chaque donnée dans la table USER (**idLogin**, **IdCustomer**, **IdRole**,...).

Une fois que le **Token** est formé selon la pétition faite à partir de l'interface client, le **Token** de vérification sera envoyé pour effectuer d'autres actions dans l’application.

**Token** **crypté :** s’utilise chaque fois que l'utilisateur veut effectuer une action dans l’application, ou il doit envoyer ce token de vérification pour avoir l'autorisation

**Token** **non crypté :** pour la manipulation de donnes et des pages dans l’application etc.

**Code**

**Controller api**

**Verificacion de petition :**

**//decode the json**

$params= json\_decode($json);// utilizando JsonResponse

**//Recuperation des donnes**

$email=(!empty($params->email)) ? $params->email : null ;

$password=(!empty($params->password)) ? $params->password :null;

$gettoken=(!empty($params->gettoken)) ? $params->gettoken :null;

if (!empty($email) && !empty($password)) {

$loginnew= new Login();

$loginnew->setEmail($params->email);

$password=$params->password ;

$pwd=Hash('sha256', $password);

$loginnew->setPassword($pwd);

$errors=$validator->validate($loginnew);

if (count($errors)) { //verifie le type de champs envoie depuis l'interface

$errorsString = (string) $errors;

return new Response($errorsString);

} else {

**Ce qui est envoyé a l’application client**

if ($gettoken) {

$signup=$jwtAuth->existsLogin($params->email, $pwd,

$params->gettoken);

} else {

$signup=$jwtAuth->existsLogin($params->email, $pwd);

}

return new JsonResponse($signup);

**Service (JwtAuth)**

/\*\*

\* Return un objet si le login existe

\* :;

\* @param [type] $email

\* @param [type] $password

\* @param [type] $gettoken

\* @return $data

\*/

public function existsLogin($email, $password , $gettoken=null){

// default answer

$data=[

'status'=>'error',

'code'=>'400',

'msg'=>'user not found',

];

// default answer

$login\_repo=$this->manager->getRepository(Login::class);

$login=$login\_repo->findOneBy([

'email'=>$email,

'password'=>$password

]);

// if the user exists generate the token

$signup=false;

if(is\_object($login)){

$signup=true;

}

if($signup){

$user\_repo=$this->manager->GetRepository(User::class);

$user=$user\_repo->findOneBy(['idLogin' =>$login->getId()

]);

$sex\_repo=$this->manager->GetRepository(Sex::class);

$sex=$sex\_repo->findOneBy(['id' =>$user->getIdSex()

]);

$nationality\_repo=$this->manager->GetRepository(Nationality::class);

$nationality=$nationality\_repo->findOneBy(['id' =>$user->getIdNationality()

]);

$marital\_status\_repo=$this->manager->GetRepository(MaritalStatus::class);

$marital\_status=$marital\_status\_repo->findOneBy(['id'=>$user->getIdMaritalStatus()

]);

$telephone\_repo=$this->manager->GetRepository(Telephone::class);

$telephone=$telephone\_repo->findOneBy(['idUser' => $user->getId()

]);

if($telephone){

$type\_telephone\_repo=$this->manager->GetRepository(TypeTelephone::class);

$type\_telephone=$type\_telephone\_repo->findOneBy(['id' =>$telephone->getIdTypeTelephone()

]);

$telep=$telephone->getTelephone();

$type\_telep=$type\_telephone->getTypetelephone();

}

else{

$telep=" ";

$type\_telep=" ";

}

$token=[

'sub'=>$user->getId(),

'name'=>$user->getName(),

'lastname'=>$user->getLastName(),

'birthday'=>$user->getBirthday(),

'login'=>$login->getId(),

'email'=>$login->getEmail(),

'password'=>$login->getPassword(),

'role'=>$user->getIdRole(),

'nationality'=>$nationality->getNationality(),

'maritalstatus'=>$marital\_status->getMaritalStatus(),

'sex'=>$sex->getSex(),

'telephone'=> $telep,

'type\_telephone'=> $type\_telep,

'createAt'=>$user->getCreateAt(),

'updateAt'=>$user->getUpdateAt(),

'iat'=>time(),

'exp'=>time() + (7 \* 24 \*60 \*60),

];

//code le token

$jwt=JWT::encode($token,$this->key,'HS256');

// check the flag gettoken with a condition

if(!empty($gettoken)){

$data=$jwt;

}else{

$decode=JWT::decode($jwt,$this->key,['HS256']);

$data=$decode;

}

}

// return the data

return $data;

}

**Authentification côté client**

**Fonction onSubmit()**

Fonction onSubmit chargé de recevoir un formulaire d'authentification et d'envoyer ensuite un e-mail et un mot de passe à un service Angular qui enverra ces données à un Api-Rest Symfony

onSubmit(form){

this.\_userservice.signup(this.user).subscribe(

response=>{

if(!response.status || response.status!="error"){

this.status="success";

this.identity=response;

this.\_userservice.signup(this.user,true).subscribe(

response=>{

if(!response.status || response.status!="error"){

this.token=response;

**//** **Enregistrer l'information dans LocalStoreage dans la section**

localStorage.setItem('token',this.token);

**//** **Dans le localStorage, nous ne pouvons pas sauvegarder un objet, il faut sauvegarder que du texte ou des chiffres, c’est pour cela qu’on doit convertir cet objet en json string**

localStorage.setItem('identity',JSON.stringify(this.identity));

**// Redirection vers la page d'accueil**

this.\_router.navigate(['/home']);

}else{

this.status="error";

}

},

error=>{

this.status='error';

}

)

}else{

this.status="error";

}

},

error=>{

this.status='error';

}

)

}

**Service (UserService)**

**Fonction signup**

signup(user, gettoken = null): Observable<any>{

if ( gettoken != null){

user.gettoken = 'true';

}

const json = JSON.stringify(user);

const params = 'json=' + json;

const header = new HttpHeaders().set('Content-Type', 'application/x-www-form-urlencoded');

return this.\_http.post(this.url + 'login/user', params, { headers : header});

}

**Sécurité des pages côté client**

Pour la sécurité du site, j’ai mis en place un **Service** appelé **IdentityGuard** qui se charge de rediriger vers la page d'accueil toute adresse url qui tente de s'identifier sans se connecter à l'application au préalable.

**app.routing.ts : c’est un fichier type script Angular où tous les réglages de toutes les routes sont effectués. Ces routes sont ceux qui permettent d'afficher les pages dans l’application.**

**Dans ce fichier type script c'est là que le service IdentityGuard est importé et assigné au component qui utilise une route x dont l'affichage nécessite une autorisation d'accès.**

**app.routing.ts**

import{IdentityGuard} from './services/identity.guard';

**//définir les itinéraires dans une constante avec le format json**

const appRoutes:Routes=[

{path:'setting', component:UserEditComponent, canActivate:[IdentityGuard]},

{path:'video-new', component:VideoNewComponent,canActivate:[IdentityGuard]},

{path:'video-edit/:id', component:VideoEditComponent,canActivate:[IdentityGuard]},

{path:'video-detail/:id', component:VideoDetailComponent,canActivate:[IdentityGuard]},

{path:'about', component:AboutComponent,canActivate:[IdentityGuard]},

{path:'contact', component:ContactComponent,canActivate:[IdentityGuard]},

];

**Service (identity.guard.ts)**

import{Injectable} from '@angular/core';

import{Router,ActivatedRoute,Params, CanActivate} from '@angular/router';

import{UserService} from '../services/user.services';

@Injectable()

export class IdentityGuard implements CanActivate {

constructor(

private \_route:Router,

private \_userService:UserService

){

}

canActivate() {

let identity=this.\_userService.getIdentity();

if(identity){

return true;

}else{

this.\_route.navigate(['/login']);

return false;

}

}}

**Service (UserService)**

/\*\*

\* Fonction qui permet de prendre l'identité du LocalStorage et de la passer à un

\* objet java script

\*/

getIdentity(){

**// Prendre identity du LocalStorage et la passer à un objet java script**

const identity = JSON.parse(localStorage.getItem( 'identity' ));

if (identity && identity !== 'undefine'){

this.identity = identity;

}else{

this.identity = null;

}

return this.identity;

}

**5- Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données  
obtenues).**

Dans chaque projet, chaque développeur web doit s'assurer du bon fonctionnement des fonctionnalités de l'application.

Une bonne pratique de développement web est l'utilisation d'outils qui permettent de tester le code que nous développons.

Ces outils permettent de rendre le résultat du code plus fiable et plus efficace.

Il existe trois types de tests différents :

- **Tests unitary**

Permet de tester une unité de code spécifique (méthodes). Lorsque cette unité est isolée du reste du code.

- **Tests functional**

Permet de tester une classe dans un contexte spécifique avec les autres composantes.

- **Tests End to End**

Permet de savoir si la demande donne le résultat escompté.

Permet de savoir que l'application dans son ensemble fonctionne.

Il est important de savoir, en fonction de la situation dans laquelle vous êtes confronté au projet, à quel moment vous ferez les tests.

C'est pourquoi, en fonction de son évolution et du moment où vous arrivez sur le projet, vous pouvez faire vos tests avant, pendant et après le développement du code.

- **Test verifier**

Le code est écrit et les tests sont effectués.

C'est utile dans le cas où vous récupérez le code source qui vient de quelqu'un d'autre

- **Test driven development**

Le test est écrit avant le code, ce qui permet de ne pas se rendre à chaque fois à l'application pour voir si le résultat est bon ou non.

- **Test first**

Le test est écrit en même temps que le code.

Il permet de décomposer un problème sur place afin de pouvoir écrire le code étape par étape.

Il est très utile lorsqu'on travaille seul.

Dans le cas du développement de BlogFavVideo, l'outil de test est implicite dans le framework Symfony et s'appelle PHPUNIT.

Cet outil est ajouté au projet à l'aide de la ligne de commande php bin/phpunit.

**6 - Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité. (**como yo hice para poner en practica la seguridad informatica y porque es importante**)**

La sécurité informatique est généralement résumée en cinq objectifs principaux **:**

* Intégrité : garantir que les données sont ce qu'elles sont censées être.
* Confidentialité : garantir que seules les personnes autorisées ont accès aux ressources échangées.
* Disponibilité : assurer le bon fonctionnement des systèmes d'information.
* Éviter le rejet : s'assurer que vous ne pouvez pas refuser une opération qui a été effectuée.
* Authentification : garantir que les entités qui communiquent sont bien celles qu'elles disent être.

Ces cinq points principaux contre la vulnérabilité et l'amélioration de la sécurité de l'application ont été mis en pratique dans le développement de l'application comme suit.

* **Empêcher un utilisateur d'accéder à l'application en utilisant une url du site sans se connecter au préalable.**

Il arrive que des utilisateurs qui ont accédé à un site spécifique et qui ont conservé l'une des url du site puis, depuis leur domicile ou d'autres ordinateurs, aient essayé d'accéder à cette page en utilisant cette url sans s'être préalablement connectés.

Une bonne pratique de sécurité, en tant que développeur web, consiste à effectuer une redirection vers la page d'accueil du site au cas où l'utilisateur ne serait pas connecté ou enregistré et essaierait se connecter directement au site à l'aide d'une url en la tapant dans le navigateur.

Pour la réalisation de cette importante pratique de sécurité, j’ai mis en place le service identity.guard.ts qui vérifie si l'utilisateur est connecté ou non et au cas où il ne l'est pas, il effectue une redirection vers la page d'accueil.

Dans le fichier type script app.routing.ts se trouvent les déclarations de tous les routes. Chaque route protégé est associé à un component et à la fonction canActivate du service identity.guard.ts. Cette fonction canActivate se charge de rediriger vers la page d'accueil au cas où l'utilisateur ne serait pas connecté.

**Exemple**

**Service (identity.guard.ts)**

import{Injectable} from '@angular/core';

import{Router,ActivatedRoute,Params, CanActivate} from '@angular/router';

import{UserService} from '../services/user.services';

@Injectable()

export class IdentityGuard implements CanActivate {

constructor(

private \_route:Router,

private \_userService:UserService

){ }

canActivate() {

let identity=this.\_userService.getIdentity();

if(identity){

return true;

}else{

this.\_route.navigate(['/login']);

return false;

}

}

}

**app.routing.ts**

import{IdentityGuard} from './services/identity.guard';

**//définir les itinéraires dans une constante avec le format json**

const appRoutes:Routes=[

{path:'setting', component:UserEditComponent, canActivate:[IdentityGuard]},

{path:'video-new', component:VideoNewComponent,canActivate:[IdentityGuard]},

{path:'video-edit/:id', component:VideoEditComponent,canActivate:[IdentityGuard]},

{path:'video-detail/:id', component:VideoDetailComponent,canActivate:[IdentityGuard]},

{path:'about', component:AboutComponent,canActivate:[IdentityGuard]},

{path:'contact', component:ContactComponent,canActivate:[IdentityGuard]},

];

* **Vérifier chaque champ d'insertion et de modification pour éviter les données non désirées.**

Lorsque vous travaillez avec des bases de données, chaque table a des champs et ces champs sont d'un type de données spécifique (varchar, int etc. etc).

Dans ce projet, la vérification de chaque champ avant d'être inséré et modifié a été prise en compte pour assurer l'enregistrement et la modification corrects des données dans la base de données.

Du côté client, (interface web), j’ai utilisé les patterns, propriété html qui offre la possibilité, en utilisant des regex, pour s'assurer que les données insérées sont du bon type.

Si certains champs ne contiennent pas le type de données correct, un message d'avertissement rouge est lancé.

Pour doubler la sécurité, le bouton qui envoie les informations du formulaire ne sera activé que lorsque le formulaire aura les bons types de données.

Comme le projet est divisé en trois parties et voici l'importance aussi de **MVC**, une fois que le formulaire (objet java script) est passé au format JSON pour être envoyé à L'Api, Symfony utilise des annotations telles que "assert" qui se charge de faire les validations pour chaque champ de l'entité.

Avec cet outil on a la garantie que l'information passe par deux filtres avant d'être enregistrée, le filtre de l'interface et le filtre de l’api.

**Exemple**

**Interface (Vue)**

 <div class="form-group">

    <label for="email">Email</label>

    <input type="email" name="email" #email="ngModel" [(ngModel)]="user.email"  required pattern="[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$" class="form- control" id="email" placeholder="email">

     <small \*ngIf="!email.valid && email.touched" class="invalid-feedback d-block"> adresse mail pas valide</small>

 </div>

<input type="submit" class="btn btn-primary float-right" value="Login" [disabled]="loginForm.invalid">

**Api (Contrôleur)**

**Entity Login**

use Symfony\Component\Validator\Constraints as Assert;

 /\*\*

     \* @ORM\Column(type="string", length=255 )

     \* @Assert\Email(message = "Le mail'{{ value }}' n'est pas un mail valide. .")

     \* @Assert\NotBlank(message = "Le mail ne peut pas être vide")

     \*/

    private $email;

**Controller User**

 $loginnew= new Login();

 $loginnew->setEmail($params->email);

 $password=$params->password ;

 $pwd=Hash('sha256', $password);

 $loginnew->setPassword($pwd);

 $errors=$validator->validate($loginnew);

 if (count($errors)) { //verifie le type de champs envoie depuis l'interface

     $errorsString = (string) $errors;

     return new Response($errorsString);

    }

**Note :** Cette vérification peut également être effectuée en construisant un **form** de type l’entité où les champs sont déclarés avec son type.

Ensuite, dans le contrôleur se fait l’appel de cet form et la creation avec les données qui arrivent de la request. Si le formulaire est valide l'action est exécutée.

**Commande php symfony pour cree le form** :

* php bin/console make:form

**Creacion et control du form :**

* $form=$this->createForm(NomFromType ::class,NomEntity) ;
* $form->handleRequest(request) ;
* If($form->isSubmitted() && $form->isValid())

**Mise en place d'un token crypté pour autoriser l'accès au site**

Le processus d'authentification qui est lie l’application, est un processus sécurisé où chaque utilisateur qui demande l'accès à l’application en utilisant son email et son mot de passe y aura accès si et seulement son email et son mot de passe sont corrects.

Comme mesure de sécurité et de privacité, l'application, après avoir vérifié l’email et le mot de passe, donne à l'utilisateur une clé unique (chaîne de caractères chiffrée) avec son identifiant, sa date de création, sa date d'expiration et quelques données supplémentaires de l'utilisateur. Cette clé unique apple **token ‘autorisation**, garantit à chaque utilisateur un accès privé à son espace personnel au sein de l'application, car chaque fois qu'une demande d'accès est faite, une clé unique et nouvelle est générée.

Chaque fois que l'utilisateur se déconnecte de l'application, cette clé unique est détruite et générée à nouveau à chaque connexion, ce qui garantit que c'est le bon utilisateur qui se trouve dans son espace personnel et non un autre.

 /\*\*

     \* Return un objet si le login existe

     \*

     \* @param [type] $email

     \* @param [type] $password

     \* @param [type] $gettoken

     \* @return $data

     \*/

    public function existsLogin($email, $password , $gettoken=null){

         // default answer

         $data=[

          'status'=>'error',

          'code'=>'400',

          'msg'=>'user not found',

        ];

      // default answer

          $login\_repo=$this->manager->getRepository(Login::class);

          $login=$login\_repo->findOneBy([

              'email'=>$email,

              'password'=>$password

          ]);

      //  if the user exists generate the token

          $signup=false;

          if(is\_object($login)){

              $signup=true;

          }

          if($signup){

              $user\_repo=$this->manager->GetRepository(User::class);

              $user=$user\_repo->findOneBy(['idLogin' =>$login->getId()

              ]);

              $sex\_repo=$this->manager->GetRepository(Sex::class);

              $sex=$sex\_repo->findOneBy(['id' =>$user->getIdSex()

              ]);

              $nationality\_repo=$this->manager->GetRepository(Nationality::class);

              $nationality=$nationality\_repo->findOneBy(['id' =>$user->getIdNationality()

              ]);

              $marital\_status\_repo=$this->manager->GetRepository(MaritalStatus::class);

              $marital\_status=$marital\_status\_repo->findOneBy(['id' =>$user->getIdMaritalStatus()

              ]);

              $telephone\_repo=$this->manager->GetRepository(Telephone::class);

              $telephone=$telephone\_repo->findOneBy(['idUser' => $user->getId()

              ]);

           $token=[

                      'sub'=>$user->getId(),

                      'name'=>$user->getName(),

                      'lastname'=>$user->getLastName(),

                      'birthday'=>$user->getBirthday(),

                      'login'=>$login->getId(),

                      'email'=>$login->getEmail(),

                      'password'=>$login->getPassword(),

                      'role'=>$user->getIdRole(),

                    /\*'nationality'=>$user->getIdNationality(),

                      'maritalstatus'=>$user->getIdMaritalStatus(),

                      'sex'=>$user->getIdSex(),\*/

                      'nationality'=>$nationality->getNationality(),

                      'maritalstatus'=>$marital\_status->getMaritalStatus(),

                      'sex'=>$sex->getSex(),

                      'telephone'=> $telep,

                      'type\_telephone'=> $type\_telep,

                      'createAt'=>$user->getCreateAt(),

                      'updateAt'=>$user->getUpdateAt(),

                      'iat'=>time(),

                      'exp'=>time() + (7 \* 24 \*60 \*60),

               ];

      //code le token

          $jwt=JWT::encode($token,$this->key,'HS256');

      //  check the flag gettoken with a condition

          if(!empty($gettoken)){

              $data=$jwt;

          }else{

              $decode=JWT::decode($jwt,$this->key,['HS256']);

              $data=$decode;

          }

      }

      //  return the data

        return $data;

  }

**7 - Description d’une situation de travail ayant nécessité une recherche et effectuée par le candidat durant le projet (**que hiciste para tener la competance en lo que hisciste tegnologias, forma de conception del projecto etc **)**

Il existe actuellement un large éventail de technologies pour le développement d'applications web, comme par exemple du côté client, Vue, React, Angular etc qui sont des technologies qui facilitent le développement web des programmeurs d'une manière organisée permettant une meilleure efficacité et productivité.

Côté serveur, on peut citer Ruby, Laravel, Symfony entre autres.

Il existe également différents types de gestionnaires de bases de données tels que, Sql Server, Mysql, Oracle, etc.

Il y avait que faire un choix dans l'immense gamme de technologies pour le développement de mon application.

La base de ce choix de technologies a commencé avec l'entreprise où j'ai entamé mon contrat de professionnalisation où est développé un produit web appelé Dematdoc qui traite de la dématérialisation des documents. Ce produit est développé en Angular Js comme technologie côté client, en Symfony3 comme technologie côté serveur et Mysql comme gestionnaire de base de données.

Ces technologies ne sont pas les plus utilisées actuellement, mais la structure du projet est conforme à la philosophie du MVC.

Ayant à l'esprit qu'il existe des versions plus récentes de ces technologies, comme par exemple Angular 8 et Symfony 5, j'ai fait le choix de ces technologies car, en plus de me permettre de continuer avec le sponsor de développement existant dans l'entreprise, elles m'ont donné l'opportunité d'évoluer en termes de connaissance des nouvelles technologies qui sont largement utilisées dans le marché actuel du développement web.

Pour acquérir les compétences nécessaires dans les technologies sélectionnées, j'ai été soutenu par des sources Internet telles que :

* [https://symfony.com/](https://symfony.com/.-)
* [https://angular.io/](https://angular.io/%20)
* <https://www.youtube.com/>
* [Wikipédia, l'encyclopédie libre](https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal)
* <https://www.udemy.com/>
* <https://www.w3schools.com/>

En plus de consulter les enseignants (Jean Bernard, Rodolphe, Eric) de la formation sur chaque étape de l'avancement du projet.

Pour établir une bonne conception du projet une fois les technologies de développement web sélectionnées.

Je me suis appuyé sur la méthode agile **SCRUM** qui m'a permis de mieux organiser, structurer et faire avancer le projet.

J’ai mis en place un cahier de charges où j’ai expliqué les principales tâches et leurs fonctionnalités

Utilisez l'outil en ligne **Trello**, pour l'organisation des tâches et le temps d'exécution de chacune d'entre elles.

Maquette (**Méthode Wirframe**) le projet sous forme de fiches pour avoir une visualisation des différentes interfaces et des relations entre elles.

Pour l'analyse détaillée et rigoureuse de la base de données, j'ai créé un diagramme des relations entre les entités qui m'a permis de visualiser les relations entre toutes les tables de la base de données et d’amener chaque table à la troisième forme normale et d'éliminer ainsi la répétition des données.

En général, j'ai structuré le projet en trois niveaux, après d’avoir fait une analyse de ce qui serait la meilleure façon de développement adaptée au projet et au temps présent.

De nos jours, les applications multicouches sont développées lorsqu'il y a plusieurs équipes et que chacune d'entre elles est responsable du développement de son produit de manière indépendante pour la livraison finale.

Cette façon de travailler est très efficace et permet une progression colossale des projets.

C'est pourquoi j'ai divisé mon projet en trois couches.

* **Couche interface**

C'est une application web développée en Angular 8 qui est responsable de la construction des interfaces web du projet et qui récupère toutes les informations nécessaires à l'utilisateur qui fait des requêtes à la deuxième couche (Contrôleur) qui est l'intermédiaire de la troisième couche (Modèle) où se trouve la base de données.

* **Couche métier (Api-Rest)**

Il s'agit d'un Api-rest développé dans Symfony 5 dont la principale fonctionnalité est de contrôler le flux d'informations (données) du projet.

Et pour cela, nous avons utilisé les principaux outils fournis par le cadre Symfony :

Gestionnaire, Dépôt, ORM, Itinéraires, Demande, etc.

* **Couche modèle**

C'est la couche où sont stockées toutes les informations du projet et qui a été développée avec le gestionnaire de base de données MySql.

Le projet a été conçu et développé du point de vue d'un concepteur d'application, suivant une structure de conception de projet qui a permis de développer d'un point de vue spécifique et global les fonctionnalités qui ont rendu possible la livraison d'un produit final de qualité.

La description de la veille sur les vulnérabilités de sécurité est liée à l’une des compétences « Développer la partie front-end d’une interface utilisateur  
web », « Développer la partie back-end d’une interface utilisateur web », « Développer des composants d’accès aux données », « Développer des  
composants dans le langage d’une base de données », « Développer des composants métier » ou « Développer une application mobile ». Cette veille  
est effectuée dans le cadre du projet en entreprise.  
La description de la situation de travail, ayant nécessité une recherche basée sur un ou des sites francophones ou anglophones, concerne un  
problème technique ou une nouvelle fonctionnalité à mettre en œuvre, dans le cadre du projet en entreprise. Elle est liée à l’une des compétences du  
titre.  
Dans le cas de la recherche de solution, basée sur un ou des sites anglophones, à un problème technique ou une nouvelle fonctionnalité à mettre en  
œuvre, le candidat décrit le besoin d’information, et indique comment il a effectué la recherche : les mots clés de recherche utilisés et la liste des sites  
retournés. Il précise les critères de sélection du (ou des) site(s). Il indique la solution trouvée et si elle a pu être mise en œuvre.  
Dans le cas de la veille sur les vulnérabilités, le candidat indique comment il a effectué la veille : les sites et les mots clés utilisés. Il indique les  
vulnérabilités trouvées et éventuellement les failles potentielles corrigées.  
La longueur du dossier de projet hors annexes est de 40 à 60 pages, soit environ 75000 caractères espaces non compris