

DOSSIER DE PROJET

FLORENTIN RHINO



TITRE PROFESSIONNEL DE DÉVELOPPEUR WEB ET WEB MOBILE
AFPA | FONTENY-LE-COMPTÉ

SOMMAIRE

Introduction	3
Compétences couvertes par les projets	3
Résumé	4
PARTIE FRONT-END	5
1 Généralités du projet	5
1.1 Les wireframes	5
2 L'environnement de travail	9
3 Le développement	9
3.1 Structure/Sémantique	9
3.2 Intégration	10
3.3 Le responsive	11
4 Rendu graphique	12
5 Améliorations	14
PARTIE BACK-END	15
6 La genèse d' « Hoplie – Réalisations »	15
6.1 Le cahier des charges	15
6.2 Les wireframes	15
7 Préparation de l'environnement de travail	16
7.1 Choix des technologies	16
7.2 L'espace de développement	16
8 Le développement	18
8.1 Création du projet Symfony	18
8.2 Le versioning avec Git et Gitlab	18
8.3 Base de données et entités	18
8.4 Accès aux données	21
8.4.1 Le CRUD	21
8.4.2 Formulaire	23
8.5 Validation et sécurité	24
8.6 Les « views »	25
8.6.1 Préparation à l'affichage	25

8.6.2	Affichage via Twig.....	27
9	Rendu graphique	28
10	Présentation du jeu d'essais.....	29
	VEILLE SECURITE	30
11	Les vulnérabilités de sécurité	30
11.1	Faille XSS.....	30
11.2	Injection SQL.....	30
12	Processus de la recherche	31
12.1	Extrait du site anglophone.....	32
12.2	Traduction	33
	Conclusion.....	34
	Remerciements.....	34

Introduction

Nous sommes sur le point de terminer notre formation de plus de 8 mois au sein de l'AFPA de Fontenay-Le-Comte. Formation effectuée dans le contexte spécial que nous savons, l'épidémie mondiale de coronavirus qui nous touche tous. Elle a donc été marquée par des périodes de télétravail ce qui a permis de travailler une compétence transverse qui est primordial dans le métier de développeur web et web mobile : l'autonomie.

La formation s'est déroulée en deux parties, la première était essentiellement axée sur la partie front-end d'une application et la deuxième sur la partie back-end. Ce qui nous a permis de balayer de manière non exhaustive le spectre du développement d'une solution en intégrant les aspects sécurité.

Ce dossier me permettra de vous présenter deux projets distincts, un pour la partie front-end et un autre pour la partie back-end. La raison est simple, c'est que lors de mon stage j'ai réalisé une application web interne à l'entreprise (le projet qui couvrira la partie back-end). C'est pourquoi nous n'avons pas établie de charte graphique, pas fait appel à un graphiste pour élaborer les maquettes.

Le projet qui couvrira la partie front-end a été effectué en groupe de 4 lors de notre formation.

Compétences couvertes par les projets

Voici la liste des compétences classée par activité que j'ai pu acquérir lors de mon projet, elles sont tirées du référentiel de certification du titre professionnel de développeur web et web mobile.

Activité type n°1 « Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité » :

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique

Activité type n°2 « Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité » :

- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Résumé

Dans ce dossier de projet, je vais vous présenter deux projets distincts car j'ai jugé que je n'avais pas un projet qui couvrirait la totalité des deux parties c'est-à-dire la partie front-end et la partie back-end.

Par conséquent, pour la partie front, je vous présenterai un projet qui a été réalisé en cours de formation, nous devons réaliser une solution web dynamique. J'ai donc pu mettre en application ce que j'avais appris durant les premiers mois de formation. En effet, le projet m'a permis de mettre en œuvre tout ce qui est post-développement, notamment, le maquetage, le cahier des charges, l'arborescence, les wireframes, les parcours utilisateurs, etc. Pour la partie technique, j'ai utilisé HTML, CSS, JAVASCRIPT, BOOTSTRAP, JQUERY.

Pour la partie back, nous allons nous baser sur un projet que j'ai réalisé durant mon stage. Parlons du contexte dans lequel j'ai effectué mon stage. Je l'ai réalisé au sein de la société Hoplie basée à Saint-Benoît (Vienne – 86), spécialisée dans le développement web. Mon tuteur de stage était Jérémy Moreau, un des chefs de projets de l'agence. Avec les circonstances que l'on connaît tous, j'étais 2 jours en entreprise et 3 jours en télétravail (sauf la partie où l'on était en confinement, j'étais en télétravail total) avec toujours en support les développeurs de l'équipe.

Venons-en au projet en lui-même, Jérémy m'a fait part d'un besoin que l'entreprise avait. Ce besoin était le suivant : avoir un support qui listerait toutes les réalisations de l'entreprise afin que la responsable commerciale, les chefs de projets et le PDG puissent l'utiliser lors de rendez-vous avec les clients.

Partant de ce besoin, j'ai donc commencé par élaborer le cahier des charges. J'ai ensuite réalisé des wireframes avec le logiciel Adobe XD. Étant un projet interne, nous avons considéré qu'il n'était pas nécessaire de faire intervenir un graphiste pour établir une charte graphique et des maquettes.

Passons au côté technique, j'ai choisi conjointement avec l'entreprise de l'établir avec le framework Symfony car il est très puissant, avec une grande communauté donc bien documentée et c'est aussi tout simplement la technologie utilisée à 90% par l'entreprise.

Je vais vous lister de manière non-exhaustive ce que j'ai pu utiliser en termes de langages, frameworks, procédés, bundles, etc. J'ai utilisé PHP, SYMFONY, JAVASCRIPT pour gérer mes filtres en AJAX, TWIG, SASS, BOOTSTRAP, WEBPACK ENCORE afin de gérer mes assets et de les optimiser, LIPLMAGINE pour gérer mon système d'upload d'image.

PARTIE FRONT-END

1 Généralités du projet

K-vavin est un projet qui a été élaboré en cours de formation, projet dans lequel nous avons travaillé en groupe de 4. Nous avons réalisé toutes les étapes du projet c'est-à-dire du cahier des charges au développement en passant par les wireframes, l'arborescence et le zoning. Il nous a permis d'aiguiser notre aptitude à travailler en groupe.

Le but était d'élaborer une interface web dynamique avec un système de panier et/ou de favoris.

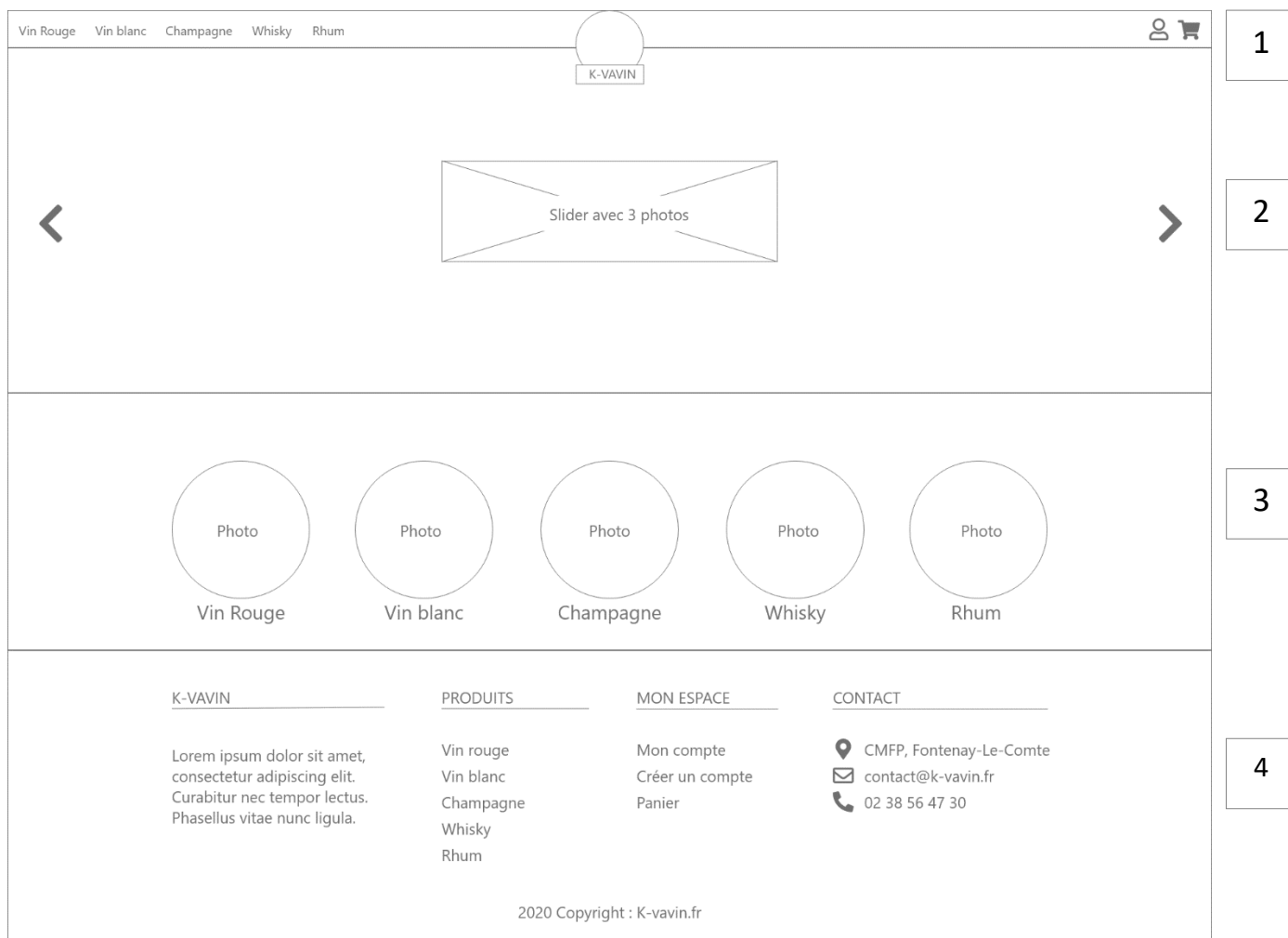
1.1 Les wireframes

Qu'est qu'un wireframe ? Un wireframe aussi appelé maquette fonctionnelle est un schéma utilisé lors de la conception d'une interface utilisateur pour définir les zones et composants qu'elle doit contenir. A partir du wireframe, le graphiste ou web designer peut réaliser les maquettes dites graphiques. Il permet de présenter au client un « prototype » de ce à quoi va ressembler la future interface mais il s'inscrit aussi dans une recherche d'ergonomie.

Toute cette étape de prototypage s'inscrit dans une démarche d'UX design. L'UX prend en compte le ressenti de l'utilisateur quand il navigue sur la solution numérique. En 3 mots, l'UX c'est de la compréhension, de l'optimisation et de la satisfaction.

Bien que l'on ait vu le logiciel Balsamiq pour élaborer les wireframes, je les ai effectués avec le logiciel Adobe XD. Afin de pouvoir présenter des wireframes de qualité, il faut en faire un pour au minimum les 3 formats les plus courants dans le web c'est-à-dire, le format « Desktop », tablette et mobile. Ceci permettra de voir le comportement de chaque élément sous différent format. Nous verrons ici, uniquement la présentation de la page d'accueil.

Pour commencer, je vais vous présenter le wireframe au format « Desktop » avec une largeur de 1920px :



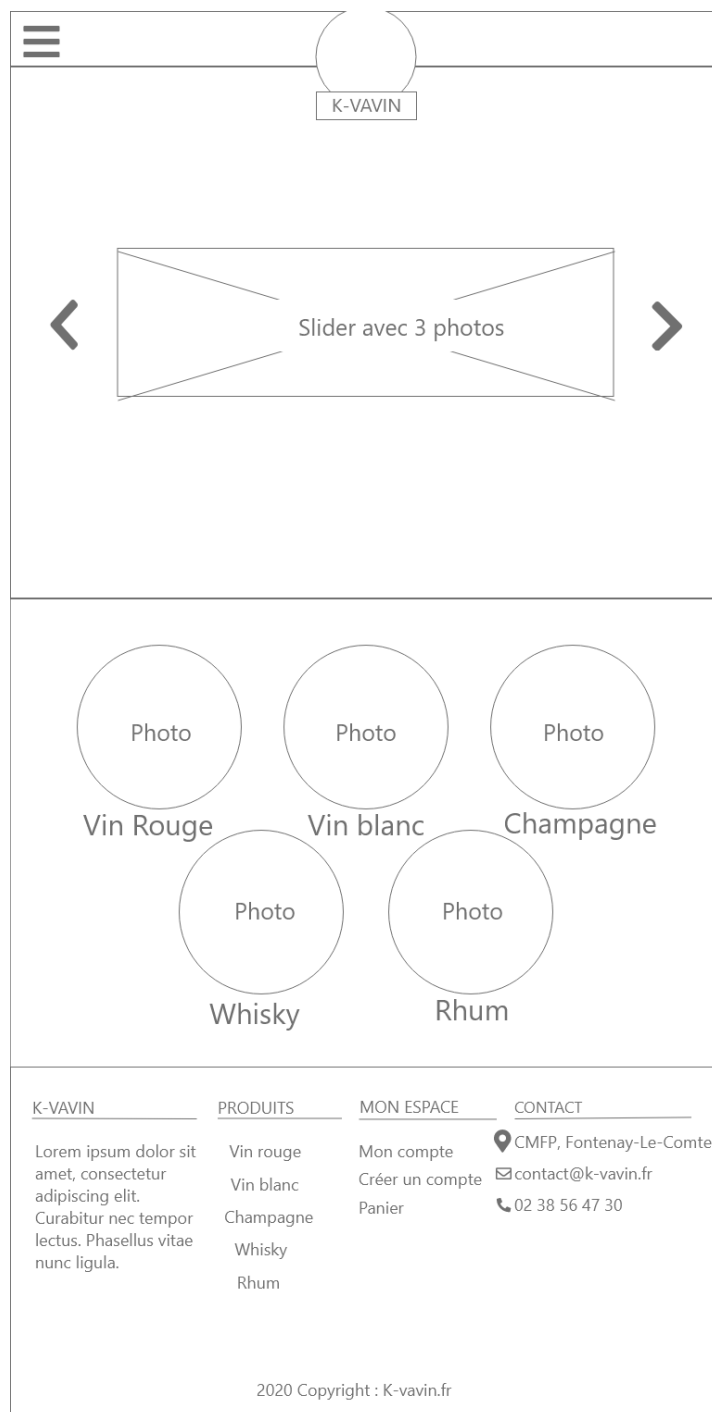
1 : « header » avec la navigation, le logo et les petites icônes pour l'accès au compte et au panier.

2 : « slider » avec 3 photos qui défilent et que l'on peut changer par les icônes « next » et « previous »

3 : la partie centrale avec des catégories différentes, chaque image sera cliquable et permettra la navigation vers la page dédiée.

4 : « footer » avec une petite description ainsi qu'une navigation qui permet d'aller sur toutes les pages du site ainsi qu'une zone de contact.

Dans un deuxième temps voici, le wireframe au format tablette avec une largeur de 768px :



Nous pouvons remarquer que les éléments ont soit changés d'aspect, soit de position.

Le « header » n'a plus la navigation apparente, afin qu'elle se déplie il faut appuyer sur l'icône des 3 traits.

La partie centrale a une autre disposition, 3 catégories au-dessus de 2. Le « footer » garde la même disposition mais voit sa taille réduite.

Le dernier wireframe au format mobile et de largeur 375px :



Encore une fois, les éléments ont changé dans leur proportion et leur disposition. Nous voyons que les éléments de catégories sont tous l'un dessus l'autre. De même pour les éléments qui sont dans le « footer ».

Le format mobile est très important car en France le nombre de personnes équipés de mobile s'élève à plus de 64 millions d'utilisateurs et dans le monde il s'élève à plus de 5 milliards d'utilisateurs (rapport de 2018).

Non sans dire qu'il est devenu un élément à part entière de notre vie et que nous passons un temps incalculable sur nos mobiles. C'est pourquoi, il faut que les sites soient « responsive » afin de s'adapter à tout type d'écran.

Qu'est-ce qu'un site dit « responsive » ?

Un site web responsive (ou « réactif ») est un site web dont la conception vise, grâce à différents principes et techniques, à offrir une consultation confortable sur des écrans de tailles très différentes.

2 L'environnement de travail



Pour un projet de ce calibre, l'environnement de travail n'a pas besoin d'être très complexe. Il suffit d'avoir un éditeur de code, celui utilisé a été Visual Studio Code. Editeur conçu par Microsoft et open source, il est très complet avec toutes les extensions qu'il comporte.

Quant au choix des technologies, elles étaient imposées par le formateur, le projet a donc été développé avec HTML, CSS, le framework BOOTSTRAP, JAVASCRIPT et le framework JQUERY.



Le HTML est un langage de structuration, le CSS est un langage de mise en forme, bootstrap est un framework créé par Twitter et très populaire grâce à son système de quadrillage, javascript est un langage qui permet de dynamiser un site et JQUERY est un framework qui se veut réducteur de code comme son slogan l'indique « write less, do more. » que l'on peut traduire par « écrire moins, faire plus ».

Mais qu'est-ce qu'un framework ?

Un framework est un ensemble d'outils et de composants qui permet de simplifier, d'uniformiser le travail des développeurs. Il accroît aussi la productivité.

3 Le développement

3.1 Structure/Sémantique

Je me suis chargé durant le projet de créer la page « index.html », la page d'accueil du site. En m'appuyant sur mes wireframes, j'ai d'abords structuré ma page avec les balises essentielles au HTML :

- <!DOCTYPE html>
- <html></html>
- <head></head>
- <body></body>

Ensuite, j'ai dans le <head> effectué toutes les liaisons internes (css, js) et externes (cdn, bootstrap, jquery) dont j'avais besoin durant le projet. J'ai également mis la ligne de code qui permet d'initialiser le « responsive ».

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

3.2 Intégration

Dans la balise <main>, j'ai intégré mon carrousel que j'ai récupéré sur le framework de Bootstrap dans la catégorie « Components > Carousel ».

Nous verrons seulement les lignes de code qui permettent d'intégrer une image :

```
<div class="carousel-item active">
  
  <div class="carousel-caption d-none d-md-block">
    <h2>K-vavin, votre nouvelle cave en ligne</h2>
  </div>
</div>
```

Utilisation de la balise avec le « src » qui désigne le chemin où se trouve la photo dans le projet, « alt » qui permet d'afficher un petit texte si l'image n'est pas chargée et dans le cas présent « height » qui permet de gérer la taille directement dans le HTML.

Afin d'intégrer ce carrousel comme nous le souhaitons, j'ai ajouté quelques lignes de code dans le CSS.

```
.carousel {
  margin-bottom: 4rem;
}

.carousel-item {
  height: 32rem;
}
```

Notamment du « margin-bottom » pour rajouter de la marge en bas de manière à ce que la <div> du dessous soit assez espacée. Avec l'unité « rem », la valeur est relative au zoom que l'utilisateur utilise.

Un peu plus bas j'ai mis en place la partie principale « catégories ».

```
<!-- début catégories -->
<div class="container marketing">
  <div class="row justify-content-around h-100 align-items-center">
    <div class="col-lg-2 col-md-4 text-center">
      <a href="produit_vin.html">
      </a>
      <h2>Vin rouge</h2>
    </div><!-- /.col-lg-2 -->
```

Afin de centrer la <div> dans l'ensemble de la page j'ai utilisé la « class » de BOOTSTRAP « container ».

3.3 Le responsive

A partir de maintenant je vais prendre en compte le « responsive » en utilisant la classe « row », « justify-content-around », « align-items-center » qui est l'équivalent en CSS classique de :

```
.exemple_div {
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
}
```

Toutes ces classes permettent de rendre des éléments flexibles dans le sens horizontal et de les répartir de manière homogène dans son élément parent (avec les mêmes marges à droite et à gauche comme si c'était avec la classe « container ») tout en les centrant verticalement.

Les classes qui jouent un rôle majeur dans le « responsive » sont les classes de type « col-lg-2 ». Elle utilise une répartition désignée par le « grid » de BOOTSTRAP.

Voici l'explication de la grille de BOOTSTRAP avec ces points d'arrêts :

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Dans notre partie de code cela voudrait dire que l'élément prends 2 colonnes sur 12 dans une taille d'écran supérieure ou égale à 992px. Qu'à partir d'une taille d'écran supérieur ou égale à 768px l'élément prendra 4 colonnes sur 12.

Enfin pour terminer le bas de la page que l'on appelle le « footer » j'ai trouvé et cherché un « footer » en open source, que j'ai trouvé sur <https://mdbootstrap.com/>. Je l'ai donc récupéré et intégré au projet. Afin de le styliser et de le faire correspondre à tout le reste du site, j'ai créé quelques classes.

```

footer{
  background-color: #22211D;
  color: #C2A476;
  padding-top: 1px;
}

footer h6{
  border-bottom: 1px solid white;
}

footer a{
  color: #C2A476;
  text-decoration: none !important;
}

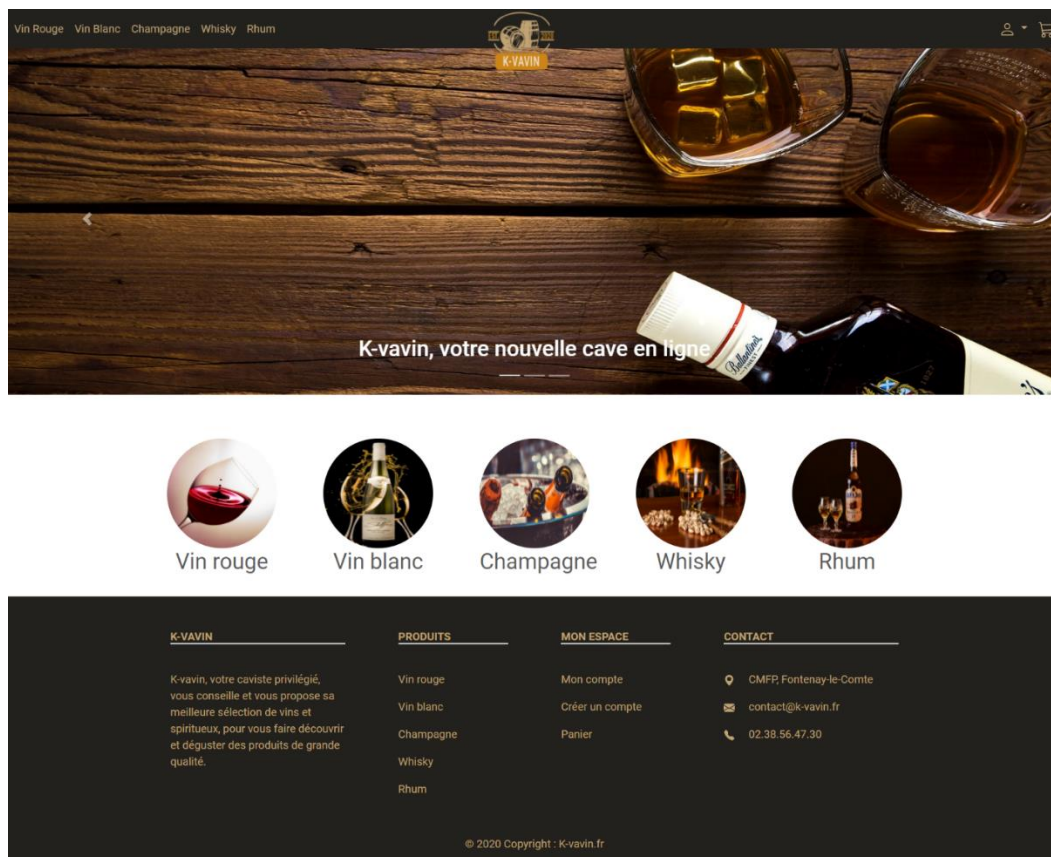
footer a:hover{
  color: #ce8517;
}

```

Nous pouvons voir que l'essentielle des modifications ont été fait sur les couleurs. Les couleurs au survol des liens via le « a:hover ».

4 Rendu graphique

La partie développement fini, il est temps pour nous de regarder le résultat en réel, d'abords en version « Desktop » puis tablette et enfin mobile :





Un large choix de produits



Vin rouge



Vin blanc



Champagne



Whisky



Rhum



Vin rouge



Vin blanc



Champagne



Whisky



Rhum

K-VAVIN	PRODUITS	MON ESPACE	CONTACT
K-vavin, votre caviste privilégié, vous conseille et vous propose sa meilleure sélection de vins et spiritueux, pour vous faire découvrir et déguster des produits de grande qualité.	Vin rouge	Mon compte	 CMFP, Fontenay-le-Comte
	Vin blanc	Créer un compte	
	Champagne	Panier	 contact@k-vavin.fr
	Whisky		 02.38.56.47.30
	Rhum		
© 2020 Copyright : K-vavin.fr			

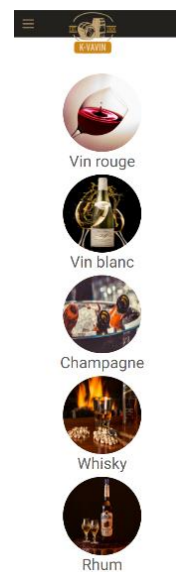
K-VAVIN
K-vavin, votre caviste privilégié, vous conseille et vous propose sa meilleure sélection de vins et spiritueux, pour vous faire découvrir et déguster des produits de grande qualité.
PRODUITS
Vin rouge
Vin blanc
Champagne
Whisky
Rhum
MON ESPACE
Mon compte
Créer un compte
Panier
CONTACT
📍 CMFP, Fontenay-le-Comte
✉ contact@k-vavin.fr
☎ 02.38.56.47.30
© 2020 Copyright : K-vavin.fr

5 Améliorations

Si le projet était à refaire, j'aurais fait disparaître le « slider » en format mobile. Je l'aurais fait avec un media query personnalisé :

```
@media (max-width: 575.98px) {  
  .carousel{  
    display: none;  
  }  
  .marketing{  
    margin-top: 5rem;  
  }  
}
```

Ceci veut dire que lorsque le support à une largeur maximum de 575.98px alors le « slider » disparaît et la <div> « marketing » a une marge supérieure de 5rem. Voici à droite une représentation de la modification.



PARTIE BACK-END

6 La genèse d' « Hoplie – Réalisations »

Ce projet découle directement d'un besoin de l'entreprise. L'idée était d'élaborer une solution qui permettrait de lister toutes les réalisations (selon des catégories) de la société Hoplie afin de pouvoir les présenter lors de rendez-vous client. Le tout avec une interface d'administration.

J'ai donc réalisé un benchmark, de manière à voir ce que d'autres sites proposaient en termes de « listing » de réalisations.

J'en ai retenu deux :

- Awwards : <https://www.awwwards.com/>
- ThemeForest : <https://themeforest.net/>

Pourquoi ces deux sites ? Tout simplement parce que leur système de « cards » est clair, les couleurs sont simples, les sites sont bien construits dans l'expérience utilisateur.

6.1 Le cahier des charges

Le chef de projet avait à cœur de me montrer toutes les étapes de conception d'une solution, c'est pourquoi nous avons décidé que j'allais établir le cahier des charges en respectant les attentes et les contraintes dont il m'a fait part.

Pour le réaliser je me suis appuyé sur des ressources propres à l'entreprise. Il m'a montré différents cahiers des charges afin de m'en inspirer. Sans vous dire qu'au cours de toute la durée du projet, le cahier des charges a évolué tout comme une demande client pourrait intervenir à n'importe quel moment de la production.

Pour plus de lisibilité dans le dossier, le cahier des charges complet est en ANNEXE 1.

6.2 Les wireframes

Une fois le cahier des charges établi et validé par le chef de projet, je me suis attaqué aux différents wireframes. Les wireframes ont été établis que pour la partie front-office (FO) car c'est la seule partie

qui est « responsive » (le chef de projet m’a expliqué que l’espace d’administration n’a pas besoin d’être « responsive car il sera exclusivement utilisé sur ordinateur).

Pour plus de lisibilité dans le dossier, les wireframes seront disponibles en ANNEXE 2.

7 Préparation de l’environnement de travail

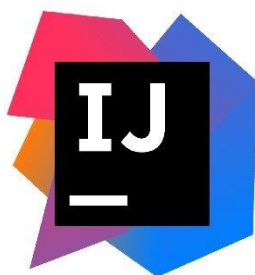
7.1 Choix des technologies

Pour la partie back-end, la technologie principale utilisée est PHP avec le framework SYMFONY. Pourquoi Symfony ? Deux réponses, premièrement parce que c’est un framework puissant, qui prend en compte l’aspect sécurité. De plus, il y a une multitude de bundle au besoin et une communauté très large ce qui fait de lui un framework très bien documenté. Deuxièmement et tout simplement parce que c’est un framework que l’on a vu durant notre formation.

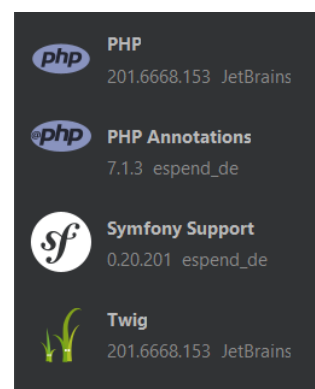
Pour la partie front-end, HTML l’incontournable. Pour la mise en forme, j’ai utilisé la technologie SASS car elle est plus poussée que le CSS dans le sens où l’on peut créer des fonctions complexes. La syntaxe est à mon sens plus lisible (le fait de pouvoir imbriquer un élément enfant dans son parent). De manière à être « plus efficace » j’ai utilisé le framework Bootstrap.

Afin de dynamiser le site, javascript a été utilisé notamment pour les requêtes AJAX.

7.2 L’espace de développement



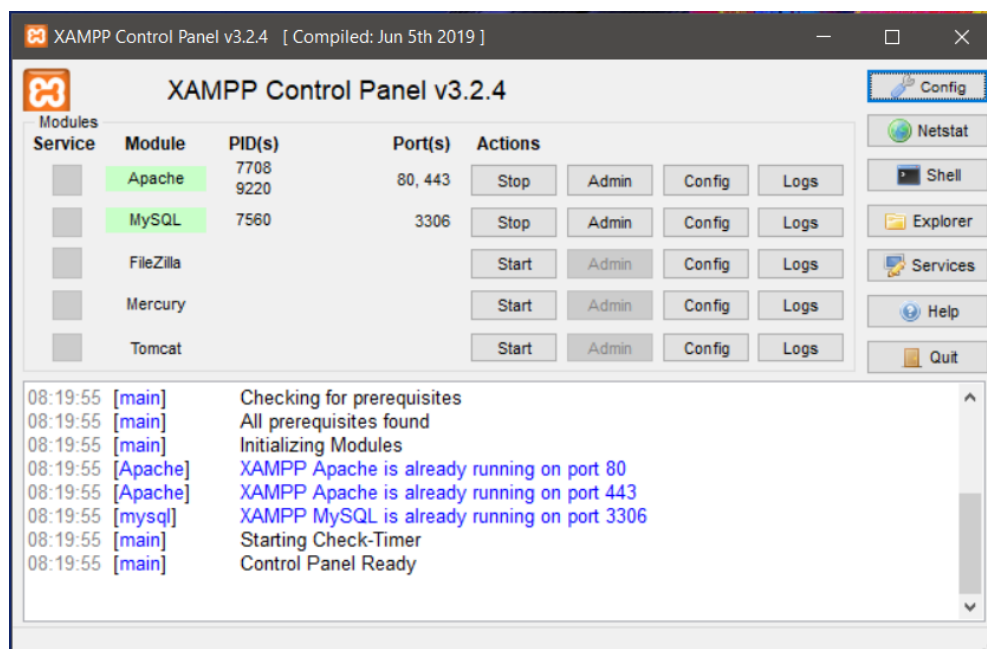
Pour commencer le développement, j’ai installé l’IDE IntelliJ IDEA de la société JetBrains avec toutes les extensions dont j’aurais besoin pour développer en PHP et SYMFONY. Notamment celle pour gérer la syntaxe du moteur de template Twig. L’extension PHP Annotations et Symfony support sont très pratiques car ils permettent d’autocompléter parfois et très souvent de proposer des méthodes ou autres qui ont été appelés au préalable.



Afin que le langage puisse être interprété, il faut utiliser un serveur externe ou interne. Voici un schéma qui représente ce à quoi sert à un serveur.



Ma solution avant de le déployer dans l'avenir sur le serveur d'Hoplie a été d'utiliser un serveur interne, j'ai donc utilisé le client Xaamp. Voici une image du panel de contrôle avec les services utilisés en cours. Notamment MySQL afin de pouvoir utiliser le langage SQL et communiquer avec la base de données (BDD) et le serveur Apache qui simule un serveur web distant.



8 Le développement

8.1 Création du projet Symfony

Après avoir installé SYMFONY via <https://symfony.com/>, ajouter toutes les variables d'environnements et les services dont j'aurais besoin c'est-à-dire composer, Node.js, yarn j'ai pu créer un projet SYMFONY basé sur un « template fonctionnel » avec la commande :

```
PS C:\> composer create-project symfony/website-skeleton Hoplie|
```

8.2 Le versioning avec Git et Gitlab



git

Pour ce projet j'ai utilisé un outil de versioning qui se nomme Git et Gitlab lui est un dépôt distant au même titre que Github.



Il a fallu initialiser le projet avec Git, pour cela je vais vous montrer quelques lignes de commandes que j'ai pu utiliser (durant tout le projet).

```
PS C:\Hoplie> git add -A|
```

```
PS C:\Hoplie> git push origin master|
```

```
PS C:\Hoplie> git status|
```

```
PS C:\Hoplie> git commit -m"Exemple de commit"|
```

J'avais comme fonctionnement, qu'à la fin de chaque « features » importantes, j'envoyais tout sur le dépôt distant.

8.3 Base de données et entités

Maintenant que tout le projet est initialisé et versionné, je vais créer la base de données (BDD) avec phpmyadmin.

Bases de données

Création d'une base de données :

Créer

Maintenant il faut lier la BDD avec le projet que l'on vient de créer, pour cela nous nous rendrons dans le fichier «.env» et y faire :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/hoplie_realisations?serverVersion=mariadb-10.5.5"
```

Maintenant que nous avons une BDD et la liaison qui est fait, il faut créer les entités qui vont correspondre au projet. Pour cela j'ai utilisé l'invite de commande. Nous prendrons l'exemple de la table « Realisation ».

```
PS C:\Hoplie> php bin/console make:entity Realisation

created: src/Entity/Realisation.php
created: src/Repository/RealisationRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> nom

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/Realisation.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> |
```

Voici ce que j'ai réalisé comme commande, ensuite SYMFONY nous propose d'entrer le nom d'une nouvelle propriété, puis nous demande le type de la propriété. Par défaut elle est de type « string », si l'on tape « ? » alors il nous proposera tous les types possibles. Puisque c'est un type « string », il me demande la taille, par défaut j'ai laissé 255. Après il nous demande, est-ce que cette propriété peut être « NULL » dans la BDD. Ici, non car nous voulons que chaque « Réalisation » ait un nom obligatoirement. Ensuite il nous demande si l'on veut entrer d'autres propriétés.

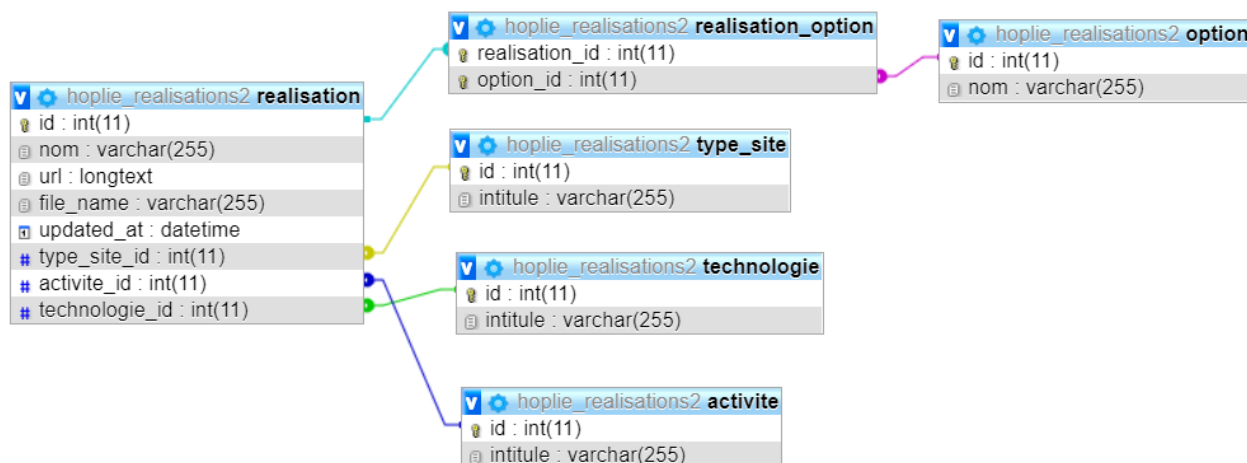
Ensuite, j'ai créé toutes les propriétés adéquate et les autres entités dont j'avais besoin. Seulement j'avais besoin de données dans la table « Realisation » mais qui était présente dans la table « Technologie » pour cela il a fallu les mettre en relation ManyToOne en partant de la table « Realisation ». Voici une liste qui illustre toutes les relations possibles entre tables :

What type of relationship is this?

Type	Description
ManyToOne 49m	Each Realisation relates to (has) one Technologie . Each Technologie can relate to (can have) many Realisation objects
OneToMany 49m objects.	Each Realisation can relate to (can have) many Technologie Each Technologie relates to (has) one Realisation
ManyToMany 49m objects.	Each Realisation can relate to (can have) many Technologie Each Technologie can also relate to (can also have) many Realisation objects
OneToOne 9m.	Each Realisation relates to (has) exactly one Technologie Each Technologie also relates to (has) exactly one Realisation .

Ensuite, j'ai effectué les migrations, qui permettent d'avoir un suivi sur les changements de la BDD et si besoin de revenir à une version précédente.

Voici la BDD finale :



8.4 Accès aux données

Maintenant que nous avons notre BDD et nos entités opérationnelles, il est tout aussi important de pouvoir « interagir » avec elle. Pour cela, je vais procéder à la mise en place de ce que l'on appelle le CRUD (pour CREATE, READ, UPDATE, DELETE).

8.4.1 Le CRUD

Pour le réaliser j'aurais pu utiliser la commande :

```
PS C:\Hoplie> php bin/console make:crud
```

Cependant, j'avais la contrainte de devoir le faire seul sans commande (demande du Lead développeur de la société afin de voir si j'étais capable de le faire).

J'ai donc créé « RealisationController.php » qui a le chemin suivant « App\Controller\Admin ».

Création de la class « RealisationController » :

```
class RealisationController extends AbstractController
```

Ensuite j'ai créé la fonction qui va permettre de créer une nouvelle réalisation :

```
.../**
 * @Route("/admin/realisation/new", name="admin_realisation_new")
 */
.../
...public function new(Request $request)
...{
...    $realisation = new Realisation();
...    $form = $this->createForm( type: RealisationType::class, $realisation);
...    $form->handleRequest($request);
...
...    if ($form->isSubmitted() && $form->isValid()) {
...        $this->em->persist($realisation);
...        $this->em->flush();
...        $this->addFlash( type: 'success', message: 'La création a été réalisée avec succès');
...        return $this->redirectToRoute( route: 'admin_realisation_index');
...    }
...
...    return $this->render( view: 'Admin/Realisation/new.html.twig', [
...        'realisation' => $realisation,
...        'form' => $form->createView()]);
...}
```

J'ai fait une injection de dépendance afin de récupérer la futur requête. Je fais appel à mon formulaire « RealisationType » afin de récupérer les informations dont je vais avoir besoin. Ensuite, si le formulaire est envoyé et valide alors je fais persister ma nouvelle réalisation pour ensuite l'écrire sur la BDD avec « flush ». Si tout s'est bien passé alors j'affiche le message « La création a été réalisée

avec succès » et je me redirige vers la route « admin_realisation_index » soit l'index des réalisations dans l'administration.

Tant que tout ceci n'est pas effectué on retourne la vue avec le formulaire pour créer une réalisation.

Voici la fonction qui permet de supprimer une réalisation :

```

.../**
 * @Route("admin/realisation/{id}/delete", name="admin_realisation_delete", methods="DELETE")
 */
public function delete(Realisation $realisation, Request $request)
{
    if ($this->isCsrfTokenValid( 'id: 'delete'. $realisation->getId(),
        $request->get( key: '_token')) ) {
        $this->em->remove($realisation);
        $this->em->flush();
        $this->addFlash( type: 'success', message: 'La suppression a été réalisée avec succès');
    }

    return $this->redirectToRoute( route: 'admin_realisation_index');
}

```

J'injecte l'entité « Realisation » dans ma fonction et j'effectue une condition qui consiste à faire si le « token delete » est valide alors je récupère l'id de la réalisation en question pour ensuite la supprimer avec la méthode « remove » pour enfin l'écrire dans la BDD avec « flush » et donc la supprimer. Ensuite, j'affiche un message de confirmation. Enfin une fois que s'est fini je redirige l'utilisateur vers la route « admin_realisation_index ».

Voici la fonction qui permet de modifier une réalisation :

```

.../**
 * @Route("/admin/realisation/{id}/edit", name="admin_realisation_edit")
 */
public function edit(Realisation $realisation, Request $request)
{
    $form = $this->createForm( type: RealisationType::class, $realisation);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->em->flush();
        $this->addFlash( type: 'success', message: 'La modification a été réalisée avec succès');
        return $this->redirectToRoute( route: 'admin_realisation_index');
    }

    return $this->render( view: 'admin/Realisation/edit.html.twig', [
        'realisation' => $realisation,
        'form' => $form->createView()]);
}

```

Cette fonction est similaire à la fonction « new », les seules différences sont le fait qu'elle n'ont pas le même message et que les vues ne sont pas les mêmes.

8.4.2 Formulaire

Le formulaire dans lequel les données sont rentrés a été créé avec ces commandes :

```
PS C:\Hoplie1> php bin/console make:form
```

Une fois la commande effectuée et quelques modifications afin que les labels soient comme je le souhaitais voici le formulaire :

```
class RealisationType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            >add( child: 'nom', type: TextType::class, [
                'required' => true,
            ])
            >add( child: 'typeSite', type: EntityType::class, [
                'required' => false,
                'class' => TypeSite::class,
            ])
            >add( child: 'activite', type: EntityType::class, [
                'label' => 'Secteur d\'activité',
                'required' => false,
                'class' => Activite::class
            ])
            >add( child: 'technologie', type: EntityType::class, [
                'required' => false,
                'class' => Technologie::class
            ])
            >add( child: 'url')
            >add( child: 'imageFile', type: FileType::class, [
                'required' => false,
                'label' => false
            ])
            >add( child: 'options', type: EntityType::class, [
                'class' => Option::class,
                'choice_label' => 'nom',
                'multiple' => true
            ])
        ;
    }
}
```

Dans les formulaires il est important de bien typer chaque « input ». C'est pourquoi nous voyons plusieurs types différents comme « TextType », « EntityType » ou même « FileType » qui respectivement corresponde à un type texte, un champ appartenant à une autre entité, type fichier.

Nous pouvons aussi définir plusieurs options différentes afin de changer le rendu visuel ou même des options comme « required » qui permet de rendre le remplissage d'un champ obligatoire, ce sans quoi le formulaire ne sera pas envoyé.

8.5 Validation et sécurité

Avant de pouvoir enregistrer les données, il est important de les valider afin de s'assurer que l'on récupérera les bonnes données. Pour cela j'ai utilisé l'outil de validation de SYMFONY qui est « Assert ». Comme exemple nous prendrons les validations du « nom » et de « url » d'une réalisation :

```

/**
 * @Assert\Regex("/[a-z]+\ \"\ -^[0-9]/",
 *          message="Le nom n'est pas valide")
 * @ORM\Column(type="string", length=255)
 */
private $nom;

/**
 * @Assert\Url(
 *          protocols={"http", "https"},
 *          message="L'url '{value}' n'est pas valide"
 * )
 * @ORM\Column(type="text", nullable=true)
 */
private $url;

```

Afin de valider le nom d'une réalisation j'ai créé une expression régulière aussi appelée regex, celle-ci :

- **[a-z]** : autorise toutes les lettres de a à z sans se soucier de la casse
- **+** : le tout répété au moins une fois
- **\ \"** : les apostrophes et les tirets si jamais une réalisation est constituée d'une abréviation ou d'un jeu de mot
- **^[0-9]** : exclus tous les chiffres compris entre 0 et 9
- Un message qui permet d'alerter l'utilisateur que le nom n'est pas bon

Pour l'url, symfony possède déjà une vérification de base des URL, j'ai ajouté le fait que seulement les protocoles « http » et « https » seront acceptés et un message qui indique que les valeurs ne sont pas bonnes.

8.6 Les « views »

8.6.1 Préparation à l’affichage

Maintenant que nous avons tous les éléments pour faire fonctionner toutes nos méthodes entre elles, il faut que l’on puisse les afficher et interagir avec elles. Ceci se fait par le biais des « views » ou « vues » en français. Pour cela nous prendrons les vues de l’administration car ce sont elles qui ont le plus de méthodes utilisables.

Avant de pouvoir afficher quoi que ce soit, il faut revenir à notre « Contrôleur » pour lui dire que l’on veut afficher ce que l’on veut par le biais d’une variable ou de plusieurs variables.

```

/**
 * @Route("/admin", name="admin_realisation_index")
 * @return Response
 */
public function index(PaginatorInterface $paginator, Request $request): Response
{
    $data = new SearchData();
    $form = $this->createForm( type: SearchDataType::class, $data);
    $form->handleRequest($request);
    $realisations = $paginator->paginate($this->realisationRepository->findBySearch($data),
        $request->query->getInt( key: 'page', default: 1),
        limit: 50);
    if($request->get( key: 'ajax')) {
        return new JsonResponse([
            'content' => $this->renderView( view: 'Admin/Realisation/_content.html.twig', [
                'realisations' => $realisations
            ]),
            'pagination' => $this->renderView( view: 'Admin/Realisation/_pagination.html.twig', [
                'realisations' => $realisations
            ])
        ]);
    }

    return $this->render( view: 'Admin/Realisation/index.html.twig', [
        'realisations' => $realisations,
        'form' => $form->createView()
    ]);
}

```

Cette méthode est un peu particulière car je ne l’ai pas affichée de manière traditionnelle, j’ai ajouté une condition qui définit que si dans la requête nous avons le terme « ajax » alors l’affichage se fera en AJAX (méthode qui permet de pouvoir afficher une page sans la recharger entièrement, elle permet de dynamiser une page et ainsi gagner en rapidité). Sinon si la condition n’est pas respectée elle rend la vue « index.html.twig » qui a pour chemin « Admin\Realisation » soit la façon classique d’afficher une vue.

Pour cela, j'ai créé une variables « \$realisations » qui va récupérer toutes les réalisations en fonction de la recherche qui a été faite « findBySearch(\$data) ». Les résultats sont donc stockés dans la variable. En toute fin de la méthode comme dit plus haut, je demande à rendre une vue qui va afficher ce qui est stocké dans la variable, c'est pour cela que je place la variable dans un tableau associatif afin de pouvoir l'afficher dans ma vue « 'realisations' => \$realisations ».

Mais qu'est ce qui est stocké dans la variable ? Du moins, comment est-ce qu'on cherche ce que comporte la variable ?

Pour cela, il faut s'intéresser au repository, dans notre cas au « RealisationRepository », c'est ici que l'on fait appel à la BDD, que l'on fait en quelque sorte nos requête SQL. Voici ce que comporte le repository en question :

```
public function findBySearch(SearchData $searchData): array
{
    $query = $this->createQueryBuilder( alias: 'r')
    $query->select( select: 'r','s','a','t')
    $query->leftJoin( join: 'r.typeSite', alias: 's')
    $query->leftJoin( join: 'r.activite', alias: 'a')
    $query->leftJoin( join: 'r.technologie', alias: 't')
    // $query->leftJoin('r.options','o')
    $query->orderBy( sort: 'r.nom');

    if(!empty($searchData->s)) {
        $query = $query
        $query->andWhere('r.nom LIKE :s')
        $query->setParameter( key: 's', value: "%{$searchData->s}%");
    }

    if(!empty($searchData->typeSites)) {
        $query=$query
        $query->andWhere('s.id IN (:typeSite)')
        $query->setParameter( key: 'typeSite', $searchData->typeSites);
    }

    if(!empty($searchData->activites)) {
        $query=$query
        $query->andWhere('a.id IN (:activite)')
        $query->setParameter( key: 'activite', $searchData->activites);
    }

    if(!empty($searchData->technologies)) {
        $query=$query
        $query->andWhere('t.id IN (:technologie)')
        $query->setParameter( key: 'technologie', $searchData->technologies);
    }

    return $query->getQuery()->getResult();
}
```

Ici nous voyons tout simplement une requête qui affiche tous les résultats avec des jointures sur plusieurs tables et plusieurs conditions avec les « andWhere ». Chaque « andWhere » possède un paramètre selon ce que l'on veut rechercher dans chacune des tables.

8.6.2 Affichage via Twig

Une fois que toutes les « Controller », « Repository » et méthodes sont prêts alors nous pouvons passer à l'affichage à proprement parlé avec notre vue Twig. Notre vue se composera dans notre exemple de deux fichiers Twig (nous en verrons qu'un seul), un qui elle la page en elle-même et un autre qui est le contenu de la page, j'ai choisi ce système afin de faire fonctionner ma méthode AJAX.

```
{% for realisation in realisations %}
<tr>
  <td>{{ realisation.nom }}</td>
  <td>
    {% if realisation.url is defined %}
    <li><a href="{{ realisation.url }}" target="_blank"
    class="lien">{{ realisation.url }}</a></li>
    {% endif %}
  </td>
  <td>
    {% if realisation.activite is defined %}
    <li class="text-center">{{ realisation.activite }}</li>
    {% endif %}
  </td>
  <td>
    {% if realisation.typesite is defined %}
    <li class="text-center">{{ realisation.typesite }}</li>
    {% endif %}
  </td>
  <td>
    {% if realisation.technologie is defined %}
    <li class="text-center">{{ realisation.technologie }}</li>
    {% endif %}
  </td>
  <td>
    <div class="d-flex justify-content-around">
      <a href="{{ path('admin_realisation_edit', {id: realisation.id}) }}"
      class="btn btn-primary"><i class="fas fa-edit"></i></a>
      <form method="post"
      action="{{ path('admin_realisation_delete', {id: realisation.id}) }}">
        <input type="hidden" name="_method" value="DELETE">
        <input type="hidden" name="_token" value="{{ csrf_token('delete' ~ realisation.id) }}">
        <button class="btn btn-danger" onclick="return confirm('Êtes-vous sûr de vouloir supprimer ?')"><i class="fas fa-trash-alt"></i></button>
      </form>
    </div>
  </td>
</tr>
{% endfor %}
```









Afin de parcourir toutes les données selon la recherche ou les filtres, j'ai fait une boucle « for » et afficher les résultats sous condition que la variable soit définie.


Il y a également des boutons qui permette d'interagir avec les données, le bouton « supprimer » qui utilise la méthode au préalable développer, « delete » et le bouton modifier qui lui utilise la méthode « edit », plus haut dans la vue il y a le bouton « créer un réalisation » qui utilise la méthode « new ». Nous pouvons constater que tout ce que nous avons fait avant est donc lié. C'est ce que l'on appelle l'architecture « MVC » pour « Manager, View et Controller ».

9 Rendu graphique

Voici le rendu graphique dans l'interface d'administration et du front.

[Créer une réalisation](#)

Nom	URL	Secteur d'activité	Type de site	Technologie		
Entre mer et forêt	https://www.entre-mer-et-foret.com/	Tourisme	Site vitrine	Symfony 4.4		
Kinedo	https://www.kinedo.com/		Site vitrine			
Kinemagic	https://www.kinemagic.fr/		Site vitrine			
Recogest	https://www.recogest.fr/	Finance		Symfony 3		




Entre mer et forêt

Site vitrine


Tourisme

Symfony 4.4




Kinedo

Site vitrine



Kinemagic

Site vitrine



Recogest

Finance

Symfony 3

Voici les filtres qui sont les mêmes sont le front et le back :

Type de Sites

☐ Site vitrine

Secteur d'activités

☐ Finance

☐ Tourisme

Technologies

☐ Symfony 4.4

☐ Symfony 3

[Filtrer](#)

10 Présentation du jeu d'essais



État initial



Si les identifiants sont bons alors nous allons directement sur la page d'accueil.



État si le mot de passe n'est pas bon



État si l'email n'a pas été trouvé dans la BDD.

VEILLE SECURITE

11 Les vulnérabilités de sécurité

Pour les vulnérabilités, je me suis uniquement dirigé vers les failles les plus courantes liées au langage PHP, a fortiori il en existe plein d'autres.

Nous parlerons de deux failles majeures qui sont les failles XSS et les injections SQL.

11.1 Faille XSS

XSS veut dire « cross-site scripting ». Le principe est d'injecter des données dans un site web par des paramètres d'URL, des messages sur les forums, etc. Si ces données ne sont pas vérifiées alors il existe une faille. On pourrait donc injecter du code malveillant en langage de script (ex : javascript, java, etc).

Les risques :

- Redirection de l'utilisateur
- Vol d'informations (sessions, cookies, données personnelles)
- Actions sur le site, à l'insu d'une personne (envoi de messages, suppressions de données, etc)
- Rendre une page illisible avec par exemple une boucle infinie

Protection :

- Traiter systématiquement le code HTML
- Utiliser la fonction htmlspecialchars()
- Traiter les données au niveau du serveur

11.2 Injection SQL

C'est une méthode d'exploitation de faille de sécurité d'une application qui interagit avec une base de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système ce qui peut compromettre la sécurité.

Les risques :

- Extraction de données caractère par caractère (méthode blind based et time based)
- Extraction de données champ pas champ (méthode error based)

- Extraction de données d'un ensemble de données (méthode union based)
- La plus dangereuse de toutes et qui permet d'exécuter n'importe quelle requête SQL (méthode stacked queries)

Protection :

- Utilisation d'une requête préparée
- Vérifier et valider les données (expression régulière)
- Echapper les caractères spéciaux

SITUATION DE TRAVAIL

12 Processus de la recherche

Lors du développement du projet, j'ai dû beaucoup me documenter afin d'apprendre de nouvelles techniques ou pour approfondir celle que dont j'avais déjà connaissance.

Dans le monde de l'informatique en particulier dans le développement web, la quasi-totalité des documentations sont en anglais. Nous pouvons en trouver en français mais l'anglais est de rigueur surtout pour les technologies et frameworks populaires. C'est pourquoi il faut un niveau d'anglais minimum, je dirais que j'ai un niveau d'anglais technique c'est-à-dire que je comprends les documentations et tout ce qui a trait à l'informatique. Cependant, je ne pourrais pas tenir une conversation complète sans la moindre difficulté avec quelqu'un qui parle anglais.

Revenons-en au projet, ma recherche s'est portée sur la gestion des rôles.

En tout premier lieu j'ai effectué une recherche sur le moteur de recherche de Google, j'ai tapé les mots suivants :

« **Symfony documentation** »

Avant de commencer le projet, je savais que SYMFONY avait une documentation officielle alors j'ai décidé de m'appuyer dessus tout au long de mon projet.

J'ai cliqué sur le premier lien, lien de la documentation officielle :

« <https://symfony.com/doc/current/index.html> »

J'ai cliqué sur « **Guides** » puis choisis « **Security** »

Ensuite j'ai cliqué sur la partie qui m'intéressait dans le sommaire soit « **4) Denying Access, Roles and other Authorization** »

12.1 Extrait du site anglophone

C'est ainsi que je me suis retrouvé sur cette page « <https://symfony.com/doc/current/security.html> » que je vais vous traduire sans l'utilisation d'un logiciel ou application de traduction :

4) Denying Access, Roles and other Authorization ¶

Users can now log in to your app using your login form. Great! Now, you need to learn how to deny access and work with the User object. This is called **authorization**, and its job is to decide if a user can access some resource (a URL, a model object, a method call, ...).

The process of authorization has two different sides:

1. The user receives a specific set of roles when logging in (e.g. `ROLE_ADMIN`).
2. You add code so that a resource (e.g. URL, controller) requires a specific "attribute" (most commonly a role like `ROLE_ADMIN`) in order to be accessed.

Roles ¶

When a user logs in, Symfony calls the `getRoles()` method on your `User` object to determine which roles this user has. In the `User` class that we generated earlier, the roles are an array that's stored in the database, and every user is *always* given at least one role: `ROLE_USER` :

```
// src/Entity/User.php

// ...
class User
{
    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    // ...
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least has ROLE_USER
        $roles[] = 'ROLE_USER';

        return array_unique($roles);
    }
}
```

This is a nice default, but you can do *whatever* you want to determine which roles a user should have. Here are a few guidelines:

- Every role **must start with** `ROLE_` (otherwise, things won't work as expected)
- Other than the above rule, a role is just a string and you can invent what you need (e.g. `ROLE_PRODUCT_ADMIN`).

You'll use these roles next to grant access to specific sections of your site. You can also use a [role hierarchy](#) where having some roles automatically give you other roles.

12.2 Traduction

4) Refuser l'accès, les rôles et autres autorisations

Les utilisateurs peuvent désormais se connecter à votre application via notre formulaire de connexion. Génial ! Vous devez maintenant apprendre à refuser l'accès et à travailler avec l'objet User. C'est ce que l'on appelle l'autorisation et son travail savoir si un utilisateur peut accéder à une ressource (une URL, un modèle, un appel de méthode,...).

Le processus d'autorisation a deux aspects différents :

- L'utilisateur reçoit un rôle spécifique à la connexion (par exemple : `ROLE_ADMIN`)
- Vous ajoutez du code pour qu'une ressource (par exemple, une URL, un contrôleur) nécessite un « attribut » spécifique (le plus souvent un rôle comme `ROLE_ADMIN`) pour être accessible.

Rôles

Lorsqu'un utilisateur se connecte, Symfony appelle la méthode `getRoles()` sur votre objet User pour déterminer les rôles de cet utilisateur. Dans la classe User que nous avons générée précédemment, les rôles sont un tableau stocké dans la base de données, et chaque utilisateur se voit toujours attribuer au moins un rôle, `ROLE_USER` :

Code source

C'est une bonne valeur par défaut mais vous pouvez faire tout ce que vous voulez pour déterminer les rôles qu'un utilisateur devrait avoir. Voici quelques conseils :

- Chaque rôle doit commencer par `ROLE_` (sinon les choses ne fonctionneront pas comme prévu)
- En dehors de la règle ci-dessus, un rôle n'est qu'une chaîne et vous pouvez inventer ce dont vous avez besoin (par exemple `ROLE_PRODUCT_ADMIN`).

Vous utiliserez ces rôles pour accorder l'accès à des sections spécifiques de votre site. Vous pouvez également utiliser une hiérarchie de rôles dans laquelle certains rôles vous donnent automatiquement d'autres rôles.

Conclusion

J'aimerais conclure en précisant que le projet que j'ai présenté pour la partie back-end n'est que le « MVP » pour « Minimum Viable Product » et que j'ai des modifications ainsi que des améliorations à apporter en termes de fonctionnalités.

Techniquement ce projet m'a réellement permis de progresser car j'ai dû apprendre de moi-même, faire des recherches, établir une veille. Pour autant j'ai conscience que je touche simplement du doigt ce qu'est le monde du développement web. Ce métier est en perpétuelle évolution et c'est ce qui le rend passionnant.

Remerciements

Je tiens tout d'abords à remercier tous mes formateurs qui même si le contexte n'a pas toujours été facile, ont su m'apporter les bases fondamentales du développement web.

J'aimerais aussi remercier l'entreprise Hoplie qui m'a accueilli durant mon stage et qui continuera de m'accompagner durant une alternance afin d'obtenir le titre professionnel de niveau 6, Concepteur et Développeur d'Applications au sein de l'ENI école.

Pour finir, je vous remercie pour votre lecture.