

Dossier de projet

Gestion de site d'hôtellerie



GUILLERM

Nolwenn

DWWM 2021-2022

SOMMAIRE :

Présentation du projet.....	3
Compétences acquises.....	3
Technologies utilisées :.....	4
– Les langages.....	4
– Les logiciels.....	5
Le cahier des charges.....	6
La base de données.....	7
Traduction.....	9
Le back office.....	10
– Présentation.....	10
– L'installation.....	12
– La connexion.....	14
– Les produits.....	16
– La médiathèque.....	23
– Les statistiques.....	25
– Les réservations.....	30
Conclusion.....	34

Présentation

Mon chef de projet, Jérôme LESAINT, ayant réalisé un site d'hôtellerie sous wordpress, son projet était de faire un back-office qui pourrait être utilisé pour n'importe quel site d'hôtellerie. Nous étions 2 stagiaires sur ce projet.

Le back-office de wordpress étant trop fourni avec des informations peu utiles et peu intuitives, nous devons refaire cette interface sous Symfony. Tout en gardant la base de données wordpress, car, en effet, le front office est réalisé avec les thèmes de ce dernier. Il ne fallait pas perturber le fonctionnement ni du front-office ni du back-office de wordpress. La complexité de ce projet était alors de comprendre la base de données, de trouver les informations qu'il nous fallait, et pouvoir les utiliser sans modifier ou supprimer quoi que ce soit dans la base de données. On pouvait seulement rajouter des éléments.

L'application a pour but d'administrer un site d'hôtellerie, de pouvoir ajouter, modifier ou supprimer des produits (des chambres, des services), ainsi que des réservations. Elle dispose aussi d'une médiathèque pour ses produits. Les réservations se font à l'aide de fullcalendar. Il y a aussi une partie statistique, avec un suivi des visites, des réservations, , des commandes, un comparatif entre les réservations et les commandes, et un bilan financier.

Compétences acquises dans ce projet

- Compréhension et gestion d'une base de données complexes.
- Maquetter une application.
- Réaliser une interface utilisateur web statique et adaptable.
- Développer une interface utilisateur web dynamique.
- Réaliser une interface utilisateur avec une solution de gestion de contenu.
- Développer les composants d'accès aux données.
- Développer la partie backend d'une application web et web mobile.

Les technologies

Les langages utilisés :

Frontend



HTML, HyperText Markup Language, est un langage qui permet de mettre en forme du contenu. Les balises permettent de mettre en forme le text et de placer des éléments interactifs, tel des liens, des images ou bien encore des animations.



CSS, Cascading Style Sheets, est un langage qui permet de gérer la présentation d'une page web. Les styles permettent de définir des règles appliquées à un ou plusieurs documents HTML. Ces règles portent sur le positionnement des éléments, l'alignement, les polices de caractères, les couleurs, etc.

JavaScript



Javascript, est un langage de scripts principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web. Il est la 3ème couche des technologies standards du web, les deux premières étant les langages HTML et CSS.

Backend



PHP, Hypertext Preprocessor, est un langage de scripts généraliste et open source. Il est spécialement conçu pour le développement d'applications web. Il peut être facilement intégré au HTML. C'est un langage orienté objet.



Twig est un langage de templating différent de php destiné aux intégrateurs et développeurs web. Il est utilisé par défaut par le framework Symfony.

Framework



Symfony est un puissant framework PHP utilisé pour développer des applications web ou des sites web complexes, voire haut de gamme. Symfony est un ensemble de composants (<<bibliothèque>>) qui facilitent le développement web en réduisant le temps et l'effort requis pour créer des composants génériques.



Bootstrap

Bootstrap est un framework open source de développement web orienté interface graphique. Utilisant les langages HTML, CSS et JS, il est pensé pour développer des sites avec un design responsive.



jQuery est une bibliothèque Javascript, open source, qui permet aux développeurs d'ajouter des fonctionnalités supplémentaires aux sites web.

Base de données



MariaDB, est un système de gestion de base de données. Ce système est un fork de MySQL (My Structured Query Language), ce qui signifie que c'est un nouveau logiciel créé à partir du code source de MySQL.

Logiciels utilisés :

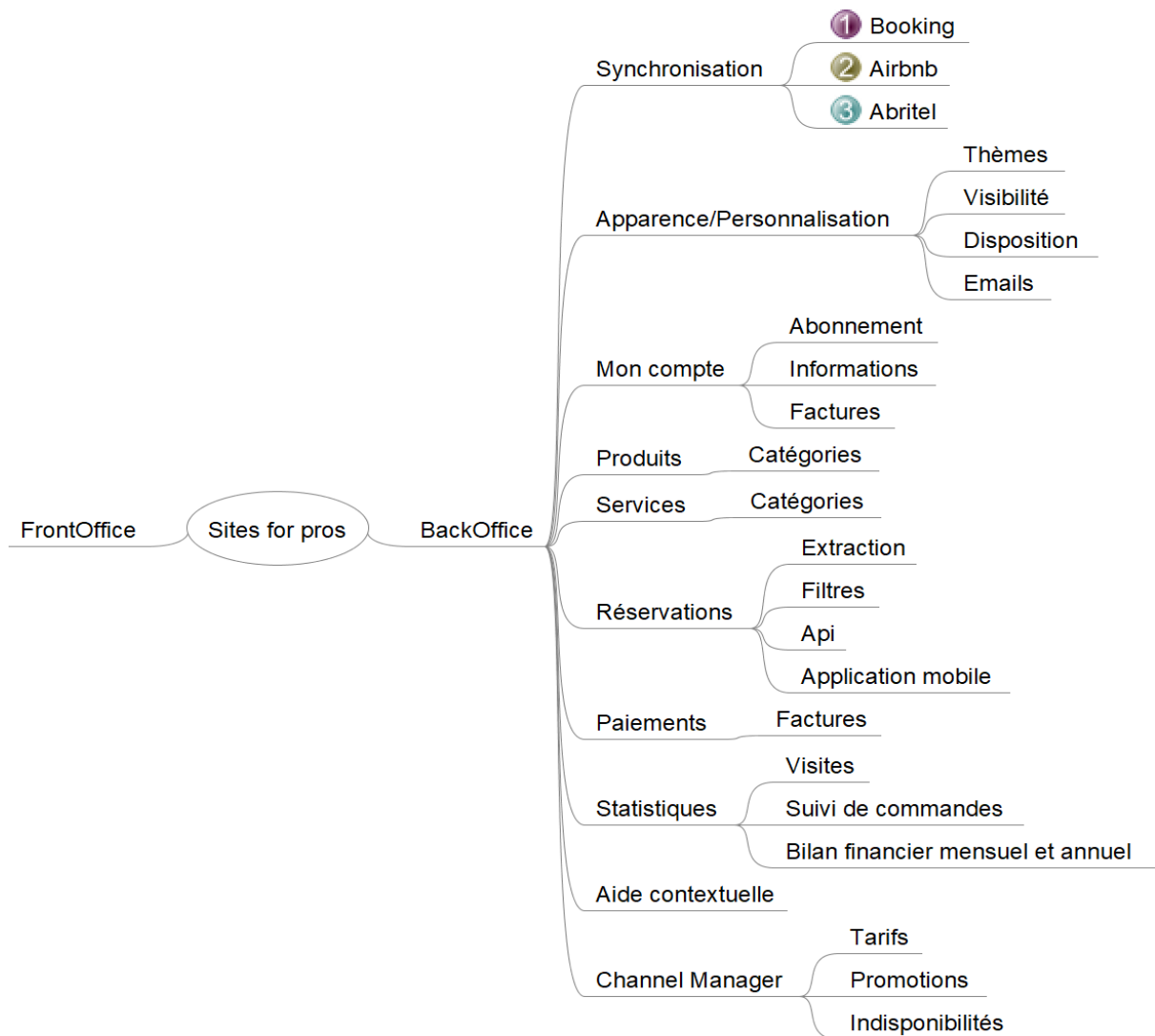


phpMyAdmin est une application web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB. Elle est réalisée principalement en PHP.



XAMPP, X Apache MySQL Perl Php, est un ensemble de logiciels permettant de mettre en place un serveur web local, un serveur FTP et un serveur de messagerie électronique.

Le cahier des charges



Voici le schéma présenté en début de projet. C'est donc un projet beaucoup plus long que 11 semaines.

Le but de cette refonte de back-office est de pouvoir proposer une interface plus intuitive, plus rapide et moins encombrée que le back-office de wordpress. Nous devons commencer par les produits avec un CRUD et une gestion multilingue. Ensuite Flavien BERVILLER s'est attardé sur la partie réservation en mettant en place un calendrier. Ce dernier devait être administré sans rechargement de page. Un CRUD des réservations devait aussi être mise en place depuis ce calendrier, ainsi qu'une synchronisation avec Booking.

Pour ma part à la suite des produits, Jérôme m'a demandé de mettre en place une médiathèque. J'ai aussi mis en place la partie des statistiques, avec plusieurs graphiques dynamiques. Pour finir, j'ai configuré la connexion.

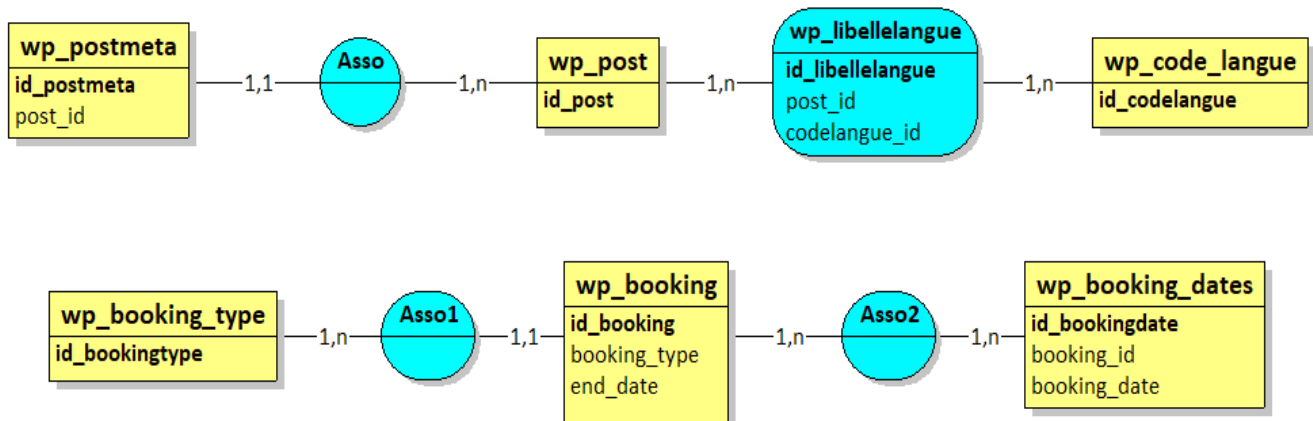
La base de données

La base de données a été la partie la plus compliquée du projet. Il a fallu beaucoup de temps pour pouvoir l'appréhender, la comprendre et réussir à travailler avec. C'est une base de données wordpress, qui comprend une multitude de tables. Nous n'en avons utilisé moins de la moitié pour réaliser le projet. J'en ai aussi rajouté 2.

Voici les tables que nous avons utilisées :

- Pour les produits :
 - wp_post
 - wp_postmeta
- et les deux tables que j'ai créées :
 - wp_libellelangue
 - wp_code_langue
- Pour les statistiques :
 - wp_booking
 - wp_booking_dates
 - wp_booking_types
 - wp_statistics_visit
 - wp_wc_order_stats
- Pour les réservations :
 - wp_booking
 - wp_booking_dates
 - wp_booking_types
- Pour la connexion :
 - wp_user

Voici le schéma des tables qui ont des relations entre elles :



Ce qu'il a fallu comprendre c'est où se trouvaient les données dont on avait besoin et où les insérer. Par exemple pour ajouter un produit, il fallait insérer 4 lignes dans la table wp_post : une ligne pour le produit en français, une ligne pour le produit en anglais, et une ligne pour l'image de chaque produit. Il fallait aussi insérer une ligne dans la table wp_postmeta pour rentrer des informations complémentaires notamment les capacités d'une chambre. Il a fallu comprendre la procédure et ne pas la changer car le front-office du thème était fait par rapport à cette base de données. Pour gérer la gestion multilingue, j'ai facilité cette procédure. J'ai créé la table wp_code_langue avec 1 pour le code langue français et 2 pour le code langue anglais. J'ai aussi créé la table wp_libellelangue pour y insérer les informations des produits.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id	bigint(20)			Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	2 id_post	bigint(20)			Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	3 id_codeLangue	bigint(20)			Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	4 product_title	varchar(255)	utf8_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	5 description_product	longtext	utf8_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	6 legend_img	longtext	utf8_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	7 description_img	longtext	utf8_unicode_ci		Oui	NULL			 Modifier  Supprimer  Plus
<input type="checkbox"/>	8 id_chemin_img	bigint(20)			Oui	NULL			 Modifier  Supprimer  Plus

Ainsi, la procédure est devenue la suivante : pour l'ajout d'un produit deux lignes sont insérés dans la table wp_post : une pour le produit et une pour l'image. Une autre ligne est inséré dans la table wp_postmeta pour des informations complémentaires. Et deux sont insérés dans la table wp_libellelangue pour la gestion multilingue : une ligne en français et une en anglais.

Traduction :

<https://symfony.com/why-use-a-framework>

Version anglaise

Why should I use a framework?

A framework is not absolutely necessary: it is "just" one of the tools that is available to help you develop better and faster!

Better, because a framework provides you with the certainty that you are developing an application that is in full compliance with the business rules, that is structured, and that is both maintainable and upgradable.

Faster, because it allows developers to save time by re-using generic modules in order to focus on other areas. Without, however, ever being tied to the framework itself.

Investing in the task, not in the technology

This is the basic principle of a framework: Not having to reinvent the wheel. And doing away with foreboding, low value added tasks (for example, the development of generic components) in order to fully focus on the business rules.

Version française

Pourquoi utiliser un framework ?

Un framework n'est pas absolument une nécessité : c'est "juste" un des outils disponible pour vous aider à développer mieux et plus vite !

Mieux, parce qu'un framework vous donne la certitude que vous développez une application parfaitement conforme aux règles du métier, qui est structurée, et qui est à la fois maintenable et évolutive.

Plus vite, car il permet aux développeurs de gagner du temps en réutilisant des modules génériques afin de se concentrer sur d'autres domaines. Sans toutefois jamais être lié au framework lui-même.

Investir dans la tâche, pas dans la technologie

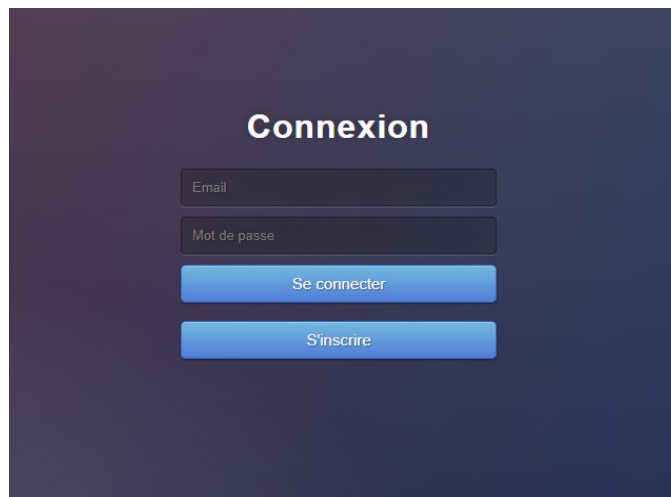
C'est le principe de base d'un framework : ne pas avoir à réinventer la roue. Et se débarrasser des tâches prémonitoires à faible valeur ajoutée (par exemple, le développement de composants génériques) afin de se concentrer pleinement sur les règles métier.

Le back office

Présentation

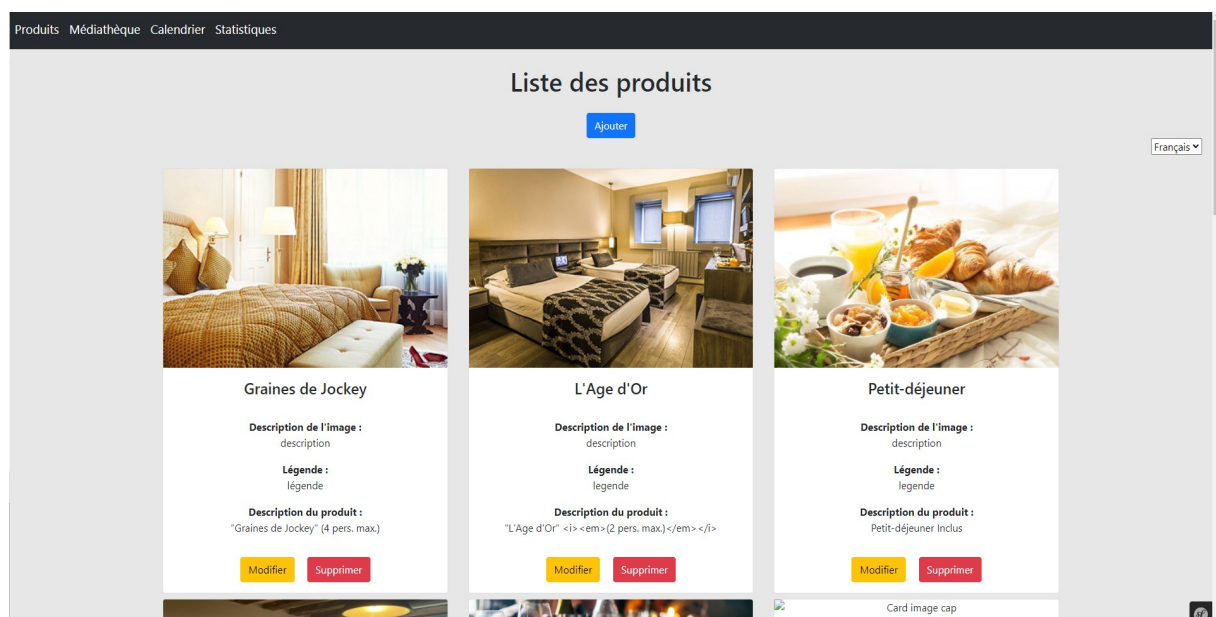
Au terme de ce stage l'application permet de faire un certain nombre de choses déjà. Cependant elle est loin d'être terminée et elle se poursuivra avec d'autres stagiaires.

En arrivant sur l'application, une page de connexion est présente.



The screenshot shows a login interface with a dark blue background. At the top, the word 'Connexion' is centered in white. Below it are two input fields: 'Email' and 'Mot de passe'. Under the 'Mot de passe' field is a small eye icon for toggling visibility. At the bottom are two blue buttons: 'Se connecter' and 'S'inscrire'.

Un formulaire d'inscription permet aussi de s'inscrire. Une fois l'inscription faite on revient sur la page de connexion. Le compte administrateur de wordpress déjà présent dans la base de données est actuellement verrouillé, j'y reviendrai dans la partie connexion. Mais avec un compte créé depuis l'application, on peut tout à fait se connecter et on arrive sur la page d'accueil qui est l'affichage des produits.



The screenshot displays the 'Liste des produits' page. At the top, there's a navigation bar with links: 'Produits', 'Médiathèque', 'Calendrier', and 'Statistiques'. Below the title 'Liste des produits' is a blue 'Ajouter' button and a language dropdown set to 'Français'. The main content area shows three product cards. Each card has an image at the top, followed by the product name, a 'Description de l'image' field, a 'Légende' field, and a 'Description du produit' field. At the bottom of each card are 'Modifier' and 'Supprimer' buttons. The products shown are 'Graines de Jockey', 'L'Age d'Or', and 'Petit-déjeuner'. The 'Description du produit' for 'L'Age d'Or' includes HTML tags like <i> and .

Les produits sont affichés avec les cards de bootstrap : j'ai affiché l'image du produit et les informations présentes dans la table wpLibellesLangues. On peut ajouter un produit, le modifier ou le supprimer. On peut afficher le produit en français ou en anglais.

La seconde page est une médiathèque. On y ajoute les images qu'on souhaite pour les produits.

Ensuite vient le calendrier, qui est la page de gestion des réservations. L'affichage du calendrier a été faite avec fullcalendar. On y voit les réservations programmées, on peut en ajouter, les modifier ou les supprimer. On voit aussi les réservations faites sur booking.

En dernier vient la page des statistiques. 5 graphiques programmés avec highcharts sont présents :

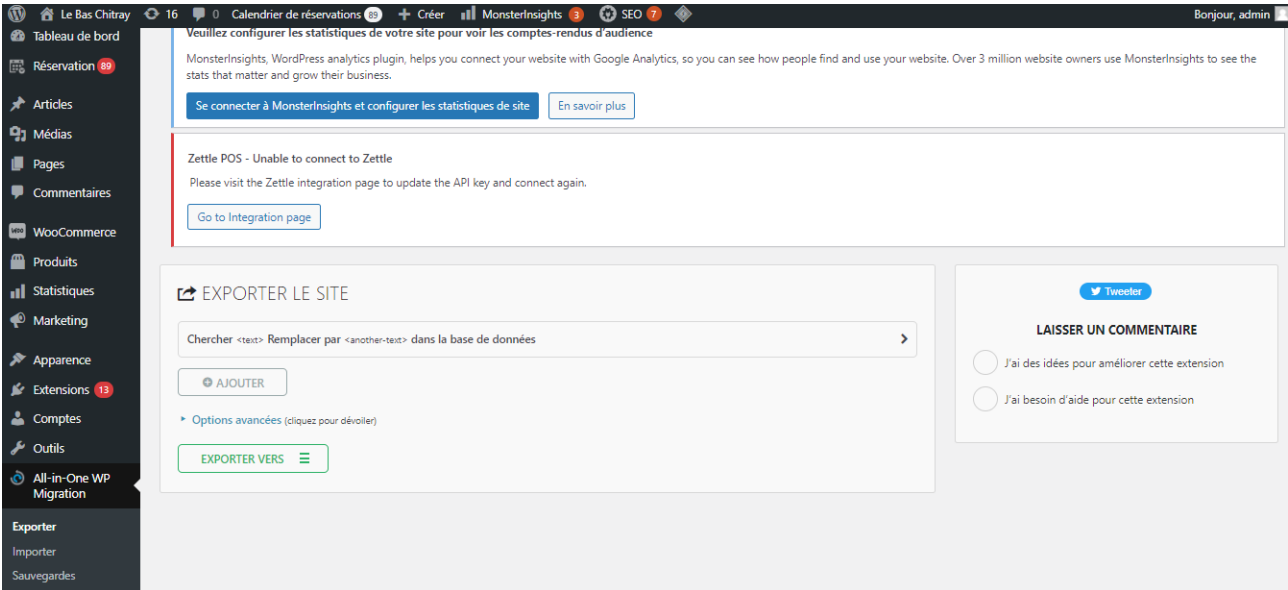
- Le nombre de visites.
- Le nombre de réservations.
- Le nombre de commandes.
- Un comparatif entre le nombre de réservations et de commandes.
- Un bilan financier.

Tous ses graphiques peuvent être affichés sur 1 mois, 3 mois, 6 mois ou 1 an.

L'installation

Installation de la base de données :

Mon chef de projet a commen   par me donner le wordpress d  j   fonctionnel de son site lebaschitray.fr. Il m'a fallu l'installer et t  l  charger l'extension All-in-One Wp Migration pour pouvoir exporter la base de donn  es. Une fois export  , j'ai pu l'installer sur mon pc et pouvoir l'administrer via phpMyAdmin de xampp.



Pour développer cette application, j'ai créé un dossier à côté de celui de wordpress. L'application a donc été développée en parallèle du dossier du back-office fonctionnel de wordpress, mais avec sa base de données. Ensuite dans mon dossier, j'ai commencé à installer Symfony.

Installation de Symfony :

Pour commencer j'ai installé composer. Il est à télécharger et à installer sur : <https://getcomposer.org>.

Ensuite j'ai installé la version 5.3.10 de Symfony. J'ai utilisé la commande "composer create-project symfony/skeleton". Ensuite j'ai installé un à un les bundles nécessaires dans le projet :

- Pour ajouter doctrine (gère les bases de données) : composer require symfony/orm-pack.
- Pour ajouter la prise en charge des annotations : composer require annotations.
- Pour ajouter l'outil maker afin de générer du code en ligne de commande : composer require symfony/maker-bundle --dev.

- Pour ajouter l'outil de gestion des formulaires : `composer require symfony/form`.
- Pour ajouter l'outil de gestion de templates : `composer require twig`.
- Pour ajouter un outil supplémentaire à la console : `composer require --dev symfony/var-dumper`.
- Pour ajouter le module de compatibilité Apache (serveur web) : `composer require symfony/apache-pack`.
- Pour ajouter la console développeur en environnement de développement : `composer require --dev symfony/profiler-pack`.
- Pour ajouter tous les outils de sécurité : `composer require security`.

Dernière étape : je vais dans le fichier `.env` qui se trouve à la racine du projet et je rajoute la ligne suivante qui permet la connexion avec la base de données :

```
# DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root:@127.0.0.1:3306/wordpress?serverVersion=mariadb-10.4.21"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&cl
### doctrine/doctrine-bundle ###
```

La connexion

La connexion a été mis en place en utilisant les outils d'authenticator de symfony.

Pour commencer, j'ai installé le SecurityBundle : "composer require symfony/security-bundle". Cela crée un fichier de configuration, le security.yaml.

```
siteHotel-app > config > packages > ! security.yaml
1  security:
2      enable_authenticator_manager: true
3      # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
4      password_hashers:
5          Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
6      # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
7      providers:
8          app_user_provider:
9              entity:
10                 class: App\Entity\WpUsers
11                 property: userEmail
12      firewalls:
13          dev:
14              pattern: ^/(_(profiler|wdt)|css|images|js)/
15              security: false
16          main:
17              lazy: true
18              provider: app_user_provider
19              custom_authenticator: App\Security\UsersAuthenticator
20              logout:
21                  path: app_logout
22                  # where to redirect after logout
23                  target: app_login
24
25              # activate different ways to authenticate
26              # https://symfony.com/doc/current/security.html#the-firewall
27
28              # https://symfony.com/doc/current/security/impersonating_user.html
29              # switch_user: true
30
```

J'ai laissé le chiffrement en auto, donc en bcrypt qui est à l'heure actuelle le meilleur hash disponible.

Ensuite, il faut repartir dans les lignes de commande avec le "make:auth". Plusieurs questions sont posées, et ensuite Symfony génère plusieurs fichiers : le UsersAuthenticatorController.php ; le UsersAuthenticator.php ; un fichier twig de login ; et une mise à jour des fichiers security.yaml et WpUsers.php.

Pour tester cette connexion, je crée un formulaire d'inscription avec la commande : "make:registration-form". Là aussi, Symfony nous pose plusieurs questions, et nous génère plusieurs fichiers : RegistrationController.php, RegistrationFormType.php, une page twig d'inscription, et une mise à jour du WpUsers.php.

J'ai donc tous mes fichiers utiles à la connexion. J'ai aussi rajouté dans le WpUsers.php les éléments permettant la connexion que voici :

```

public function getUsername(): ?string
{
    return $this->userEmail;
}

/**
 * The public representation of the user (e.g. a username, an email address, etc.)
 *
 * @see UserInterface
 */
public function getUserIdentifier(): string
{
    return (string) $this->userEmail;
}

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

    return array_unique($roles);
}

public function setRoles(array $roles): self
{
    $this->roles = $roles;

    return $this;
}

public function getSalt(): ?string
{
    return null;
}

public function eraseCredentials()
{
    // If you store any temporary, sensitive data on the user, clear it here
    // $this->plainPassword = null;
}

```

L'application permet donc de s'inscrire et de se connecter. Cependant, un souci réside dans la connexion avec le compte administrateur de wordpress déjà présent dans la base de données. En effet, ce dernier est verrouillé et par manque de temps je n'ai pas réussi à le déverrouiller pour pouvoir l'utiliser avec la connexion Symfony.

Les produits

L'affichage

```
#[Route('/accueil', name: 'accueil')]
public function index(): Response
{
    $repository = $this -> getDoctrine() -> getRepository(WpPosts::class);
    $produits = $repository -> findAll();

    $repositoryImages = $this -> getDoctrine() -> getRepository(WpPosts::class);
    $images = $repositoryImages -> findAll();

    $repositoryLanguages = $this -> getDoctrine() -> getRepository(WpLibelleslangues::class);
    $languages = $repositoryLanguages -> findAll();

    return $this->render('accueil.html.twig', [
        'produits' => $produits,
        'languages' => $languages,
        'images' => $images,
    ]);
}
```

Ci-dessus, ma fonction index qui gère l'affichage de mes produits. Elle rend 2 entités :

- WpPosts où sont stockés les chambres et services.
- WpLibellesLangues où sont stockées les informations concernant les produits.

Avant les informations des produits étaient éparpillées dans diverses tables. J'ai regroupé les informations essentielles dans une seule table : la WpLibellesLangues. On y trouve l'id du produit, d'id du code langue : dans la table WpLangue : 1 pour le français et 2 pour l'anglais. On y trouve aussi le titre du produit, sa description, la légende de l'image, sa description et enfin l'id du chemin de celle-ci qui se trouve dans la table WpPosts (cf page 8).

Toujours dans ma fonction index, j'appelle deux fois mon entité WpPosts : une fois pour afficher les produits, la 2ème pour me permettre d'afficher les images correspondantes.

Ensuite dans mon fichier accueil.html.twig, pour afficher les produits, je parcours la table WpPosts 2 fois : la 1ère pour parcourir tous les produits, la 2ème pour parcourir les images. L'id du produit doit correspondre à l'idparent de l'image. Ensuite intervient aussi la gestion de la langue. Si l'id de la langue correspond à 1 qui est le code langue pour le français alors j'insère les informations de ma table WpLibellesLangues dans cette <div>. J'ai une 2ème <div> avec les mêmes informations mais en anglais, cette <div> est mise en hidden. Avec le select je peux donc passer du français à l'anglais en jouant sur l'hidden des <div>.

```
<script>
var language;
$('#selectLanguage').on('change',function(){
    language=this.value;
    console.log(language);
    if( language == 'en'){
        $('.english').attr('hidden', false);
        $('.french').attr('hidden', true);
    }else {
        $('.english').attr('hidden', true);
        $('.french').attr('hidden', false);
    }
});
</script>
```

Côté visuel, j'ai choisi d'utiliser les cards de bootstrap.

CRUD

Ajout d'un produit :

Pour ajouter un produit, j'ai créé un formulaire avec la commande "composer require symfony/form". Dans ce formulaire on y trouve le titre du produit qui sera rentré dans la table WpPosts, ainsi qu'un formulaire de collection de l'entité WpLibellesLangues. Ce formulaire comprend le titre du produit, sa description, ainsi que la légende de l'image et sa description. Il est utilisé pour les données en français. Pour les données en anglais, j'ai utilisé des inputs HTML.

```
class CrudType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('postTitle', TextType::class, ["attr" => ['placeholder' => "Titre en français"],]);

        $builder
            ->add('wpLibelleslangues', CollectionType::class, [
                'entry_type' => LangueType::class,
                'entry_options' => ['label' => false],
            ]);

        $builder
            ->add('save', SubmitType::class, [
                'label' => 'Enregistrer',
                'attr' => ['class' => 'col-3 btn btn-dark'],
            ]);
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => WpPosts::class,
        ]);
    }
}
```

Voici le schéma des tables utilisées pour comprendre la collection :



Entre WpPost et WpCodeLangue, nous avons une relation many to many :

- Pour un produit je peux avoir plusieurs langues, et pour une langue j'ai plusieurs produits. L'association entre les deux tables s'affiche dans la table WpLibellesLangues.

Entre WpPost et WpLibellesLangues, nous avons une relation many to one :

- Pour un produit je peux avoir plusieurs libellés de langue, mais pour un libellé de langue j'ai qu'un produit.

Entre WpLibellesLangues et WpCodeLangue, nous avons une relation one to many :

- Pour un libellé de langue j'ai un code langue. Mais pour un code langue je peux avoir plusieurs libellés de langue.

Comment cela se traduit-il dans le code ?

Reprenons l'explication du formulaire du CrudType. Je remplis le titre du produit qui est stocké dans la table WpPost. Le reste des champs est mis dans la public function __construct, car les valeurs sont les mêmes. Par exemple pour le postType ça sera toujours "product" car le formulaire est fait pour créer un produit. Pour les dates on remplit la date et l'horaire auquel le formulaire a été soumis grâce à new \DateTime(). Ci-dessous le code de la fonction :

```
public function __construct()
{
    $this->postAuthor = "1";
    $this->postDate = new \DateTime();
    $this->postDateGmt = new \DateTime();
    $this->postContent = "";
    $this->postExcerpt = "";
    $this->commentStatus = "open";
    $this->postStatus = "publish";
    $this->pingStatus = "closed";
    $this->postPassword = "";
    $this->toPing = "";
    $this->pinged = "";
    $this->postModified = new \DateTime();
    $this->postModifiedGmt = new \DateTime();
    $this->postContentFiltered = "";
    $this->postParent = "0";
    $this->menuOrder = "0";
    $this->postType = "product";
    $this->postMimeType = "";
    $this->commentCount = "0";

    $this->wpLangue = new ArrayCollection();
    $this->wpLibelleslangues = new ArrayCollection();
}
```

Ensuite dans ce formulaire du CrudType j'ai inclus le formulaire LangueType qui est donc une collection de la table WpLibellesLangues :

- Pour faire cela, dans la table WpPosts j'ai une variable "*\$wpLibelleslangues*" qui mappe vers la table WpLibellesLangues et avec les options de cascade "*cascade={\"persist\", \"remove\"}*". Dans la fonction du construct, cette variable est déclarée en "*new ArrayCollection()*". Enfin on retrouve aussi les fonctions "*addWpLibelleslangues*" et "*removeWpLibelleslangues*".

```

public function addWpLibelleslangues(WpLibelleslangues $wpLibelleslangue): self
{
    if (!$this->wpLibelleslangues->contains($wpLibelleslangue)) {
        $this->wpLibelleslangues[] = $wpLibelleslangue;
        $wpLibelleslangue->setWpPosts($this);
    }
    return $this;
}

public function removeWpLibelleslangues(WpLibelleslangues $wpLibelleslangue): self
{
    if ($this->wpLibelleslangues->removeElement($wpLibelleslangue)) {
        if ($wpLibelleslangue->getWpPosts() === $this) {
            $wpLibelleslangue->setWpPosts(null);
        }
    }

    return $this;
}

```

- Dans la table WpLibellesLangues on a la variable "\$wpPosts" pour faire le lien. La jointure est faite grâce aux id. Dans cette table pas de fonction particulière simplement les traditionnels "getWpPosts" et "setWpPosts".

```

/**
 * @var \WpPosts
 *
 * @ORM\ManyToOne(targetEntity="WpPosts")
 * @ORM\JoinColumns({
 *     @ORM\JoinColumn(name="id_post", referencedColumnName="ID")
 * })
 */
protected $wpPosts;

```

Concernant la page twig :

La particularité était que dans le CrudType j'ai une collection, mais je dois avoir deux lignes dans la table WpLibellesLangues. En effet, il faut une ligne pour les données en français et une ligne en anglais. Pour résoudre ce souci j'ai dû mettre des inputs HTML pour les champs en anglais et les récupérer avec la variable \$request dans mon controller. Pour les champs en français les données sont affichées grâce au formulaire du CRUDDTYPE.

Ajouter votre produit :

En francais :

Image

Photo :
Aucune photo sélectionnée

Légende :

Description :

Produit

Titre (en français) :

Titre :

Description :

Capacité :

En anglais :

Picture

Legend :

Description :

Product

Legend :

Description :

Dans ce formulaire on peut aussi ajouter une photo. Pour cela, un lien renvoi sur la page de la médiathèque. Je récupère ensuite le nom du fichier grâce à la méthode GET. Et lors de la validation du formulaire, dans mon controller je récupère le nom du fichier grâce à la variable `$request`, comme pour mes données en anglais.

Au niveau du controller :

Dans ma fonction `created`, je commence par créer les objets dont j'ai besoin. Dans ma variable `$wpPost` (qui est un objet de `WpPosts`), j'ajoute bien les données en français de ma collection `WpLibellesLangues` grâce à la fonction `"addWpLibelleslangues"`. Pour les données en anglais je les récupère quand le form est envoyé.

```

if($form->isSubmitted() && $form->isValid()){
    $em = $this -> getDoctrine() -> getManager();
    $em -> persist($wpPost);
    $em -> flush($wpPost);
    $em -> refresh($wpPost); //Je refresh pour que je puisse récupérer le bon ID

    $codeLangueEn = $em -> getRepository(WpLangue::class) -> findOneBy(['codeLangue' => 'en']);
    $idCodeLangueEn = $codeLangueEn -> getId();
    $codeLangueFr = $em -> getRepository(WpLangue::class) -> findOneBy(['codeLangue' => 'fr']);
    $idCodeLangueFr = $codeLangueFr -> getId();

    $lastProduit = $em -> getRepository(WpPosts::class) -> findOneBy([], ['id' => 'desc']);
    $lastId = $lastProduit -> getId();

    //Partie code langue fr
    $wpLangue -> setIdPost($lastId);
    $wpLangue -> setIdCodeLangue($idCodeLangueFr);
    $em -> persist($wpLangue);

    //Je récupère les input mis manuellement pour mettre la ligne en anglais dans la table WpLibelleslangues
    $legendImgEn = $request -> get('legendImgEn');
    $descriptionImgEn = $request -> get('descriptionImgEn');
    $productTitleEn = $request -> get('productTitleEn');
    $descriptionProductEn = $request -> get('descriptionProductEn');
    $wpLangueEn -> setProductTitle($productTitleEn);
    $wpLangueEn -> setLegendImg($legendImgEn);
    $wpLangueEn -> setDescriptionImg($descriptionImgEn);
    $wpLangueEn -> setDescriptionProduct($descriptionProductEn);
    $em -> persist($wpLangueEn);
    $em -> flush($wpLangueEn);
}

```

Ensuite, s'il y a une image d'insérée, je la récupère aussi avec \$request, et j'insère manuellement les données dans la base de données de la même façon que pour les données en anglais. Les données pour l'image vont dans une 2ème ligne de la table WpPost avec pour postParent l'id inséré lors de l'envoi de \$wpPost. La capacité des chambres est envoyée dans la table wpPostMeta. Une alerte flash s'affiche en cas de succès.

Modification :

Concernant la modification, la difficulté à été de trouver comment séparer la collection français et anglais. En effet lors de la récupération de la collection les deux lignes de la table WpLibellesLangues étaient présentes. Il fallait donc les différencier pour pouvoir les éditer.

Au niveau du twig cela n'a pas été très compliqué. Une fois mon formulaire CrudType affiché, pour séparer ces deux lignes, j'ai fait deux foreach, 1 avec le code langue français (1) et un 2ème avec le code langue de l'anglais (2).

```

{% for value in form.wpLibelleslangues %}
{% if value.vars.value.idCodeLangue == 1 %}
    <b>{{ form_label(value.legendImg) }}</b><br>
    {{ form_widget(value.legendImg) }}<br><br>

    <b>{{ form_label(value.descriptionImg) }}</b><br>
    {{ form_widget(value.descriptionImg) }}<br><br>
{% endif %}
{% endfor %}
</div>

```

Pour valider les modifications cela a été un peu plus complexe. Il a fallu faire un foreach lors de la soumission du formulaire. En effet, dans la variable *\$collection*, se trouvaient les deux lignes de la table WpLibellesLangues. Il fallait récupérer ces données et parcourir ce tableau pour y insérer les nouvelles données.

```
if($form -> isSubmitted() && $form -> isValid()){
    $em = $this -> getDoctrine() -> getManager();
    $em -> persist($wpPost);
    $em -> flush($wpPost);

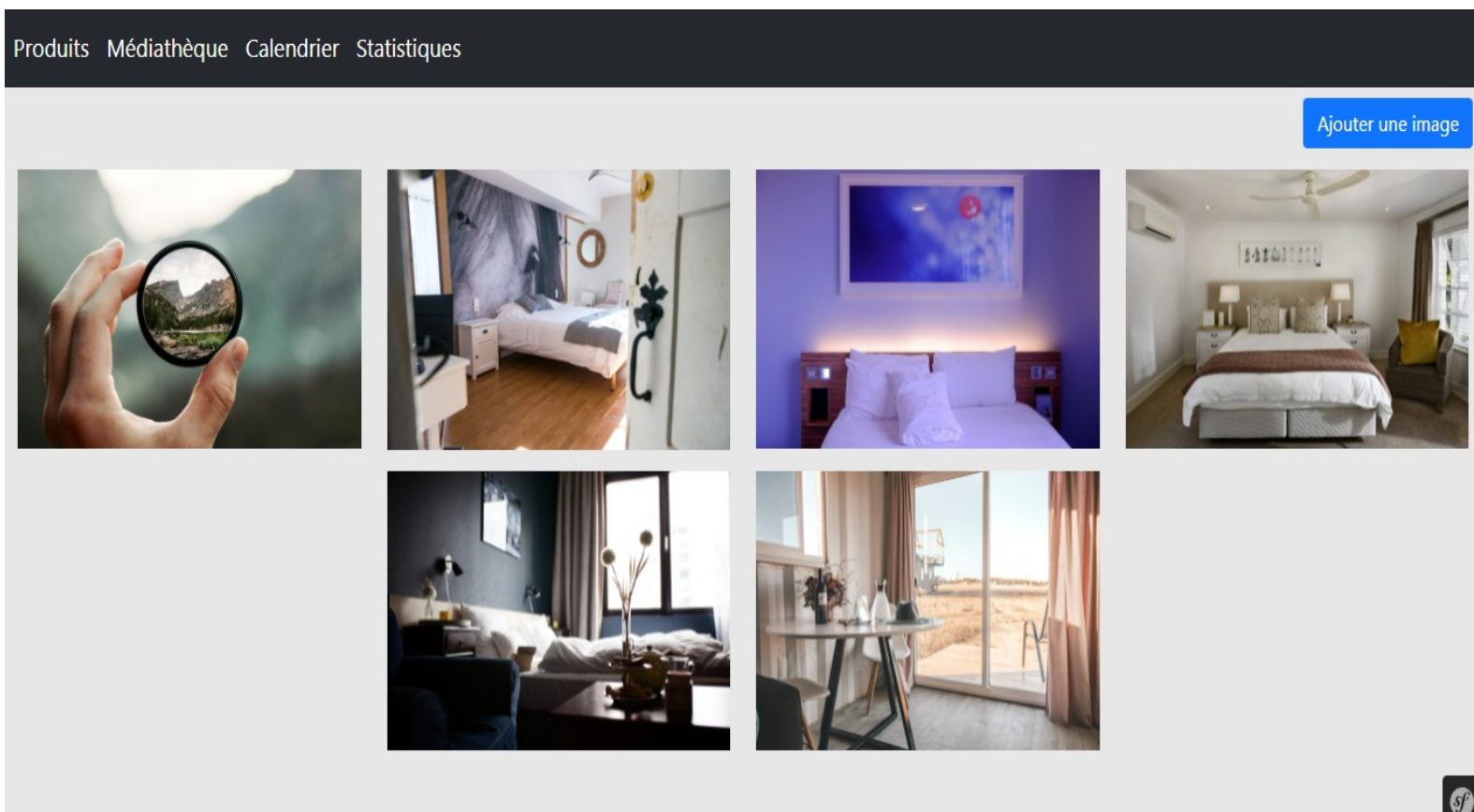
    //Pour modifier dans chaque collection au bon endroit il faut parcourir les collections, trouver le
    bon code langue et persister à ce moment là
    $collection = $wpPost -> getWpLibelleslangues();
    foreach($collection as $value){
        if($value -> getIdCodelangue() == 1){
            $em -> persist($value);
            $em -> flush($value);
        } else {
            $em -> persist($value);
            $em -> flush($value);
        }
    }
}
```

Suppression :

La fonction delete supprime bien toutes les données précédemment rentrées, grâce à la méthode remove :

- Le produit dans la table WpPosts
- Les deux lignes français et anglais dans la table WpLibellesLangues
- La capacité dans la table WpPostMeta
- L'image qui se trouve dans la médiathèque.

La médiathèque



La médiathèque est l'album photo de l'application. On y upload les images que l'on veut et on peut les attribuer aux produits.

Pour télécharger une image dans la médiathèque, on clique sur le bouton ajouter qui nous affiche une modal avec l'input de type file. Ensuite j'ai utilisé du bootstrap pour gérer l'affichage des images, et un script en javascript pour gérer l'animation du bouton attribuer. Bouton qui renvoie au formulaire d'ajout d'un produit avec le nom de la photo déjà attribué. Quand on clique sur l'image il y a simplement le nom du fichier, mais une possibilité d'évolution est de savoir si l'image est déjà attribuée à un produit.

```

<script>
window.addEventListener('load', function() {
  var a = document.getElementById("ahref{{ value.id }}");
  var img = document.getElementById("image{{ value.id }}");
  var p = document.getElementById("btnAttribuer{{ value.id }}");

  img.addEventListener('mouseenter', function() {
    p.style.display = 'flex';
  });

  a.addEventListener('mouseenter', function() {
    p.style.display = 'flex';
  });

  img.addEventListener('mouseleave', function() {
    p.style.display = 'none';
  });

  a.addEventListener('mouseleave', function() {
    p.style.display = 'none';
  });
});
</script>

```

Particularité au niveau des liens de page, car on peut arriver sur la page depuis le formulaire d'ajout d'un produit ou de la modification du produit. J'ai donc mis en place un si la page correspond à "edit" alors en cliquant sur le bouton le lien renvoie bien sur le formulaire de modification. Sinon il part sur le formulaire d'ajout.

```

<div class="col-3 mb-3">
{% if page == "edit" %}
<a href="{{ path('admin_edit', {id:idEdit }) }}"?img={{ value.nameFile }}" id="ahref{{ value.id }}">
{% else %}
<a href="{{ path('admin_add') }}"?img={{ value.nameFile }}" id="ahref{{ value.id }}">
{% endif %}
<button type="submit" class="btn btn-outline-dark displayBtnImg" id="btnAttribuer{{ value.id }}">Attribuer</button></a>
<div class="animation"></div>

```

Les statistiques

Que ce que Highcharts ?

Highcharts est une bibliothèque JavaScript qui nous permet de créer des graphiques interactifs de nature dynamique ou statique. Cette librairie possède des caractéristiques qui font d'elle un outil indispensable pour la création de graphiques. Highcharts est compatible avec tous les navigateurs et est responsive. Elle possède également une documentation complète. C'est la raison pour laquelle j'ai choisi cet outil pour réaliser les différentes statistiques.

Pour commencer à utiliser cet outil, il faut commencer par télécharger le fichier highcharts.js. On a aussi le choix entre différents thèmes pour le rendu visuel. J'ai choisi le thème brand-dark.js.

En utilisant cette librairie, on remarque qu'on peut utiliser soit HighCharts ou HighStocks. La différence entre les deux se trouve dans les options du graphique. HighStocks permet d'utiliser des graphiques est un peu plus sophistiqué avec des options de navigation comme une petite série de navigateurs, des plages de dates prédéfinies, un sélecteur de date, un défilement et un panoramique. C'est cette technologie que j'ai eu besoin d'utiliser pour avoir une gestion du temps sur 1 mois, 3 mois, 6 mois et une année avec son sélecteur de date.

Statistique des visites

Dans mon statistique controller, ma première fonction stats sert juste à me rendre la vue sur mon twig.

Dans ma 2ème fonction, *statsJsonVisites*, je récupère les données de la table WpStatisticsVisit. Ensuite je parcours le tableau rendu pour récupérer le nombre de visites pour une date donnée. Date que je mets au format Y-m-d. J'insère mes données dans un tableau, et je mets la date en valeur et en clé pour pouvoir trier le tableau avec ksort. Une fois trié, je supprime les dates en clé

```
#[Route('/stats_visites', name: 'stats_visites')]
public function statsJsonVisites(): Response
{
    //GRAPHIQUE N°1
    $repositoryStatsVisit = $this -> getDoctrine() -> getRepository(WpStatisticsVisit::class);
    $wpStatisticsVisit = $repositoryStatsVisit -> findAll();

    $tabVisites = [];

    foreach($wpStatisticsVisit as $value){
        $dateLastVisite = $value -> getLastVisit();
        $date = date_format($dateLastVisite, 'Y-m-d');
        $nombreVisite = $value -> getVisit();

        $tabVisites[$date] = array(); //Je met les dates en clef pour pouvoir le trier la tab
        $tabVisites[$date][0] = $date;
        $tabVisites[$date][1] = $nombreVisite;
    }

    ksort($tabVisites);
    $tabVisites = array_values($tabVisites); //pour enlever les dates en clef car mtn elles so

    return new JsonResponse($tabVisites);
}
```

avec `array_values`, pour pouvoir utiliser mon tableau avec highcharts. Une fois le tableau envoyé en json, je peux l'exploiter dans le script.

Dans le fichier `stats.html.twig`, côté HTML, une simple balise `figure`, avec une `div` avec un `id` à l'intérieur suffit. Le reste du code est configuré dans un script.

Je récupère mes données à l'aide d'un `$.getJSON`. Je crée un nouveau tableau, *seriesVisites*, pour pouvoir récupérer mon nombre de visites par date au format `Date.Parse`, format qui renvoie le nombre de millisecondes écoulées depuis le 1er janvier 1970. J'ai en effet besoin de ce format-là pour l'utiliser avec la `rangeSelector` de mon graphique. Une fois mes données récupérées dans mon nouveau tableau, je commence à configurer `Highcharts.setOptions`. Cette partie-là est valable pour tous les graphiques de la page, c'est une configuration globale. J'utilise `lang` pour traduire les parties textes du graphique. La documentation de highcharts est bien faite, on s'y retrouve assez facilement.

```
$.getJSON("http://localhost/hotel/gite/public/stats_visites", function (dataJson1) {

    var seriesVisites = new Array();

    for (var i=0;i<dataJson1.length;i++) {
        console.log(dataJson1[i][0]);
        seriesVisites[i] = new Array();
        seriesVisites[i][0] = Date.parse(dataJson1[i][0]);
        seriesVisites[i][1] = dataJson1[i][1];
    }

    Highcharts.setOptions({
        lang: {
            months: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
                'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],
            hortMonths: ['Janv', 'Févr', 'Mars', 'Avr', 'Mai', 'Juin', 'Juill', 'Août', 'Sept', 'Oct', 'Nov', 'Déc'],
            shortMonths: ['Janv', 'Févr', 'Mars', 'Avr', 'Mai', 'Juin', 'Juill', 'Août', 'Sept', 'Oct', 'Nov', 'Déc'],
            weekdays: ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche'],
            shortWeekdays: ['Lundi', 'Mardi', 'Mercre', 'Jeudi', 'Vendr', 'Sam', 'Dim'],
            downloadPNG: "Télécharger au format PNG",
            downloadJPEG: "Télécharger au format JPEG",
            downloadPDF: "Télécharger au format PDF",
            downloadSVG: "Télécharger au format SVG",
            downloadCSV: "Télécharger au format CSV",
            downloadXLS: "Télécharger au format XLS",
            viewFullscreen: "Mettre en plein écran",
            exitFullscreen: "Quitter le plein écran",
            viewData: "Montrer les données",
            hideData: "Cacher les données",
            printChart: "Imprimer le graphique",
        },
    });

    Highcharts.stockChart('container', {
```

Ensuite vient la configuration de ma 1ère <div> avec Highcharts.stockChart. Ici pour 5 graphiques j'ai bien 5 configurations de Highcharts.stockChart différentes. Tout est déjà pré-codé, je ne modifie que quelques options. La 1ère sont les boutons de mon rangeSelector. On peut mettre tous les x mois que nous souhaitons. J'ai aussi modifié évidemment le titre. Pour l'axe x qui est celle des dates, j'ai modifié le format pour l'afficher avec le mois et l'année. Dans les series, c'est ici dans data qu'on insère nos données : mon tableau *seriesVisites*. J'ai choisi un type area pour la forme du graphique et j'ai mis un dégradé de couleurs dans le fillColors pour une meilleure visibilité.

Une fois un graphique de configuré, les autres scripts sont configurés pareillement.

Statistiques des réservations

Revenons donc au controller. La fonction statsJsonReservation sert pour afficher les réservations. Le code est idem que pour la fonction ajax du reservationController. Une fois mes dates récupérées, c'est là que le code change. Je parcours les dates obtenues et je les insère dans mon tableau nbreReservation. Pour éviter des doublons de date, j'ai mis les dates en clé en plus d'être en valeur. J'utilise array_key_exists pour dire que si la date est différente de la clé alors je mets dans mon tableau la date et je mets 1 à mon compteur du nombre de réservation. Si par contre la date existe déjà en clé alors je rajoute seulement +1 au compteur. J'ai donc mon nombre de réservation par date. Je trie ensuite le tableau en ordre décroissant, et pour pouvoir utiliser mes données dans le script, j'enlève les dates en clé avec array_values.

```
title: {
  text: "Suivi des visites"
},
legend: {
  enabled:true
},
xAxis: {
  labels: {
    format: '{value:%b %Y}'
  }
},
series: [{
  name: "Nombre de visites",
  data: seriesVisites,
  type: 'area',
  threshold: null,
  tooltip: {
    xDateFormat: '%e %B %Y',
  },
  fillColor: {
    linearGradient: {
      x1: 0,
      y1: 0,
      x2: 0,
      y2: 1
    },
  },
}
```

```
$nbrReservation = [];

foreach($bookingReservation as $value){
  if (!array_key_exists($value,$nbrReservation)) {
    $nbrReservation[$value] = array(); //Je met les dates en clef pour éviter les doublons
    $nbrReservation[$value][0] = $value; //Je remet les dates ici pour pouvoir faire le graphique
    $nbrReservation[$value][1] = 1;
  } else {
    $nbrReservation[$value][1]++;
  }
}

ksort($nbrReservation);
$nbrReservation = array_values($nbrReservation);

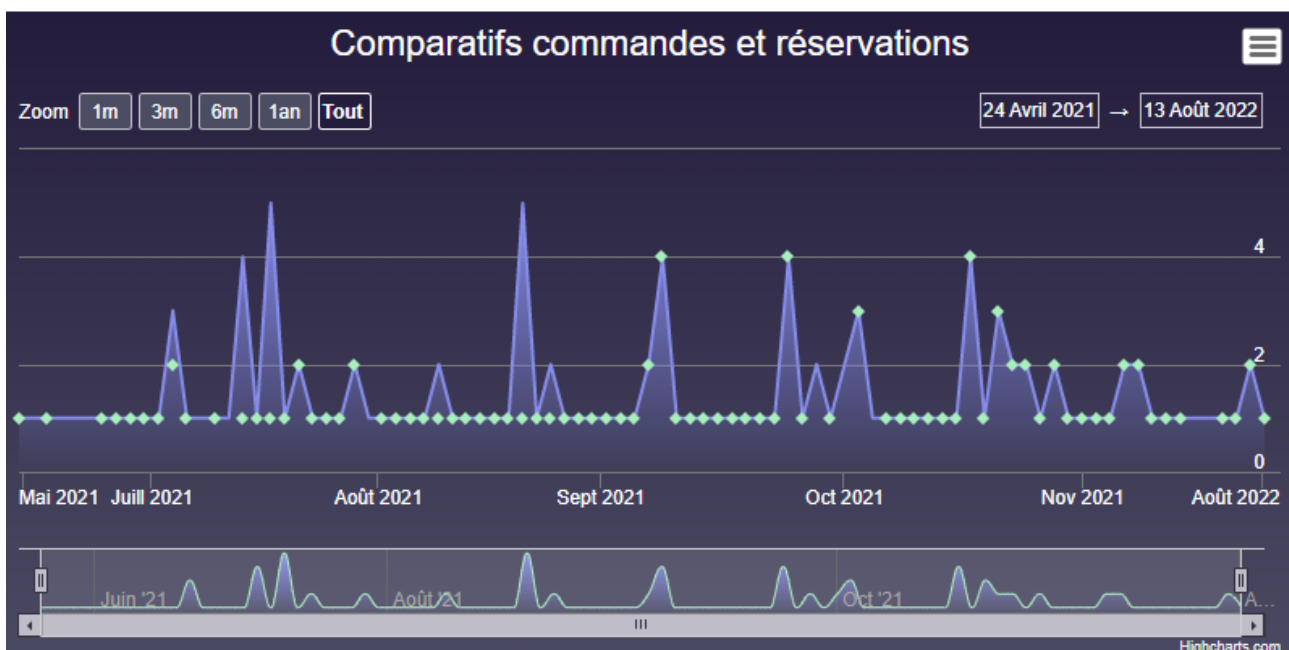
return new JsonResponse($nbrReservation);
```

Statistiques des commandes

Dans mon StatsJsonCommande le code est presque exactement le même que pour ma fonction des réservations. La seule différence se situe dans l'insertion des données dans mon tab2 qui insère les dates de la table WpBookingDates. Dans les réservations je prends toutes les dates, alors qu'ici je ne prends que les dates qui ont été approuvées, grâce au getApproved qui retourne vrai. Le reste de la fonction est parfaitement identique.

Statistique de comparaison des réservations et des commandes

Ici pas de fonction dans le controller, j'ai repris les Json des fonctions StatsJsonCommande et statsJsonReservation. Dans mon script, j'ai fait 2 get.Json , et deux tableaux qui récupèrent ses données. Au niveau de la configuration du highstock, j'ai rajouté dans les series, un data supplémentaire. Pour plus de visibilité, j'ai fait deux types différents pour les datas. Pour la représentation des réservations j'ai mis un type area, et pour les commandes j'ai mis un type scatter.



Statistique bilan financier

Dernier graphique celui du bilan financier. Pour celui-ci j'utilise la table WpWcOrderSats. Je parcours ce tableau et je rentre dans un autre tableau les dates avec les sommes reçus. J'utilise la même technique avec array_key_exists pour éviter des doublons de date. Si la date est déjà présente dans le tableau, les montants se cumulent grâce à array_sum. Ensuite je trie mon tableau avec ksort et j'enlève les clefs avec array_values. Je peux alors envoyer mon JSON.

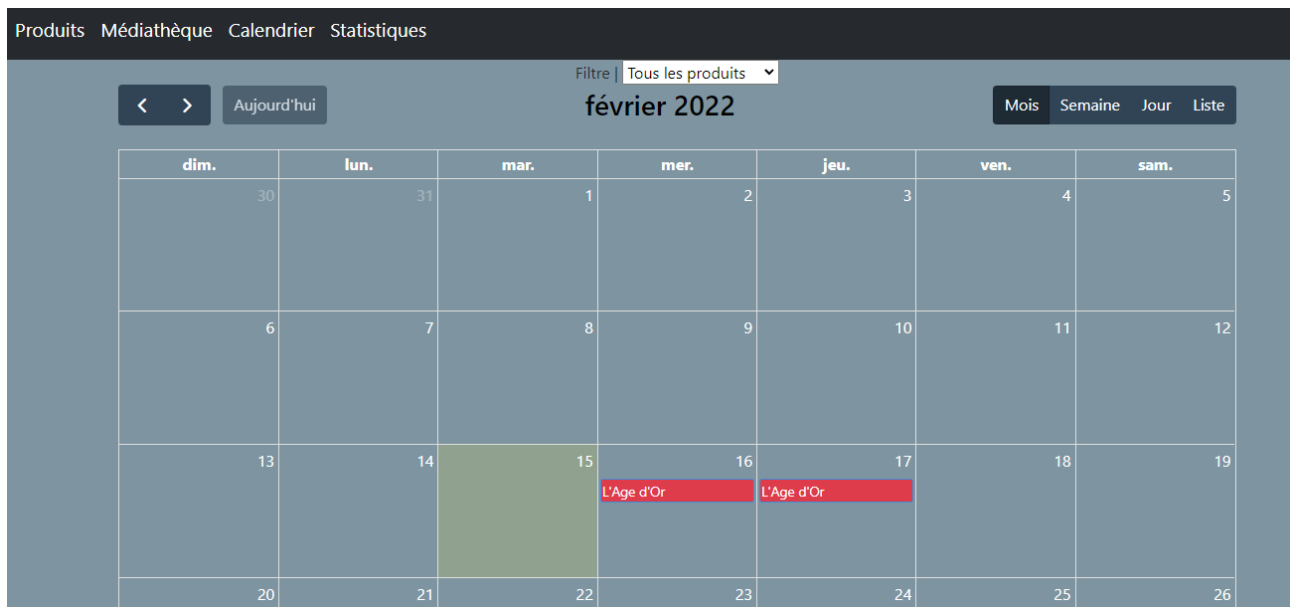
```
foreach($tabDates as $value){
    if (!array_key_exists(key($value),$tab)) {
        $tab[key($value)][0] = key($value);
        $tab[key($value)][1] = array_sum($value);
    } else {
        $tab[key($value)][1] += array_sum($value);
    }
}

ksort($tab);
$tab = array_values($tab);

return new JsonResponse($tab);
```


Les réservations

Dans cette partie je vous présente la partie des réservations qui a été faite par Flavien BERVILLER. Voici un aperçu de ce que rend le calendrier des réservations :



dim.	lun.	mar.	mer.	jeu.	ven.	sam.
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16 L'Âge d'Or	17 L'Âge d'Or	18	19
20	21	22	23	24	25	26

Affichage :

L'affichage du calendrier est mis en place par fullcalendar.

Au niveau du controller, un index est présent pour le render sur la page twig :

```
#[Route('/calendrier', name: 'calendrier')]
public function index(Request $request): Response
{
    // on récupère les chambres pour les envoyer à la variable dans le
    // calendrier twig
    $repositoryRooms = $this->getDoctrine()->getRepository(
        WpBookingTypes::class);
    $rooms = $repositoryRooms->findAll();
    return $this->render('reservation/calendrier.html.twig', [
        'rooms' => $rooms,
    ]);
}
```

Ensuite, dans la fonction ajax, il a fallu appeler 3 tables : la wpBooking, la wpBookingDates et la wpBookingTypes. Ensuite il a fallu faire 3 foreach, 1 foreach pour chacune des entités appelées dans lesquelles toutes les informations étaient appelées avec un findAll(). Dans chacune des boucles, les informations nécessaires on était mis dans un tableau avec array_push :

- pour l'entité wpBooking les informations rentrées dans le tableau sont les suivantes : l'id, la date du début de la réservation, la date de fin et le type de chambre (qui fait référence à l'id de la table wpBookingTypes).
- pour l'entité wpBookingDates : la date et l'id qui fait référence à l'id de la

table wpBooking.

- pour l'entité wpBookingTypes : l'id , le nom de la chambre et sa couleur (pour afficher des couleurs différentes dans le calendrier en fonction de la réservation de la chambre).

Puis il a fallu parcourir les 3 tableaux et y émettant différentes conditions pour pouvoir afficher les réservations sur le calendrier :

```
foreach ($tab1 as $v) {  
    $dId = $v;  
    foreach ($tab2 as $d) {  
        $dDate = $d;  
        foreach ($tab3 as $a) {  
            $dRooms = $a;  
            if ($dId["debut"] == $dDate["bDBD"]) {  
                if ($dId["type"] == $dRooms["bTRI"]) {  
                    if ($dId["id"] == $dDate["bDBI"]) {  
                        if ($dRooms["bTRN"] == "Par défaut") {  
                        } elseif ($dId["fin"] == NULL) {  
                            array_push($bookingReservation, array(  
                                'id' => $dId["id"],  
                                'title' => $dRooms["bTRN"],  
                                'start' => date_format($dId["debut"],  
                                    'Y-m-d H:i:s'),  
                                'end' => date_format($dId["debut"],  
                                    'Y-m-d H:i:s'),  
                                'backgroundColor' => $dRooms["bTRC"],  
                                'room' => $dRooms["bTRI"],  
                                'allDay' => true,  
                            ));  
                        } else {  
                            array_push($bookingReservation, array(  
                                'id' => $dId["id"],  
                                'title' => $dRooms["bTRN"],  
                                'start' => date_format($dId["debut"],  
                                    'Y-m-d H:i:s'),  
                                'end' => date_format($dId["fin"], 'Y-m-d  
                                    H:i:s'),  
                                'backgroundColor' => $dRooms["bTRC"],  
                                'room' => $dRooms["bTRN"],  
                                'allDay' => true,  
                            ));  
                        }  
                    }  
                }  
            }  
        }  
    }  
}  
  
return $this->json($bookingReservation);
```

Au niveau du twig, le code source est généré par fullcalendar, il reste à le personnaliser. Dans le script du calendrier.html.twig, on retrouve la variable calendar qui comprend le code du calendrier. Les données du json sont mises dans l'option events.

CRUD :

Concernant les réservations toutes sont gérées en Ajax pour ne pas avoir de rechargement de page.

Ajout

Quand on clique sur une case du calendrier, une modal s'affiche avec les informations nécessaires pour l'ajout. Dans le script, l'ajout se passe au niveau du select. Quand le bouton save est cliqué, les données sont récupérées pour y être inséré dans un tableau. Une requête est initialisée avec la méthode put. Cette requête est envoyée dans une url qui sera récupérée dans le controller. Enfin, ses données sont converties en Json grâce à la méthode JSON.stringify.

Dans le controller, un json_decode permet de pouvoir manipuler ses données. Puis les informations sont récupérées une à une pour les modifier dans la base de données grâce au setter.

```
#[Route('/calendar_add', name: 'calendar_add', methods: ['PUT'])]
public function created_reservation(Request $request)
{
    $dataF = json_decode($request->getContent());
    $wpBd = new WpBookingdates();
    $wpB = new WpBooking();
    $em = $this->getDoctrine()->getManager();
    // récupération des données de la request envoyé par JS
    $fS = $dataF->date;
    $fE = $dataF->end;
    $fR = $dataF->room;
    $idR = $dataF->idroom;
    $fFn = $dataF->firstName;
    $fLn = $dataF->lastName;
    $fM = $dataF->mail;
    $fP = $dataF->phone;
    $fRe = $dataF->remark;
    $fA = $dataF->adult;
    $fC = $dataF->child;
    $fCa = $dataF->childAge;
    $hash = md5(uniqid());
    $dTfs = new DateTime($fS);
    $dTfe = new DateTime($fE);
    $form = "text^name^" . $fFn . "~text^secondname^" . $fLn . "~email^email^" . $fM . "~text^phone^" . $fP .
    "~textarea^details^~select-one^adultes^" . $fA . "~select-one^enfants^" . $fC . "~text^chambre^" . $idR . "^" . $fR .
    "~text^agedesenfants^" . $fCa . "~text^prix^125 €";
    $wpB->setSortDate($dTfs);
    $wpB->setEndDate($dTfe);
    $wpB->setBookingType($idR);
    $wpB->setForm($form);
    $wpB->setRemark($fRe);
    $wpB->setHash($hash);
    $em->persist($wpB);
    $em->flush($wpB);
    $em->refresh($wpB);
    $l = $em->getRepository(WpBooking::class)->findOneBy([], ['bookingId' => 'desc']);
    $lastId = $l->getBookingId();
    $dWpB = $l->getSortDate();
    $wpBd->setBookingDate($dWpB);
    $wpBd->setBookingId($lastId);
    $em->persist($wpBd);
    $em->flush($wpBd);

    $this->addFlash('sucess', 'La réservation a bien été créé !');

    return new Response('La réservation a bien été créé !');
```

Modification & suppression

Quand on clique sur une réservation du calendrier, une modal apparaît. On peut soit la modifier, soit la supprimer. Les deux options se traitent dans l'eventClick du script.

Concernant la modification, les étapes sont les mêmes que pour l'ajout d'une réservation.

Concernant la suppression, pas besoin de récupérer les informations une à une, un lien est simplement envoyé. Lien récupéré dans la fonction delete du controller. Grâce à l'id inséré dans l'url, on supprime les données de la réservation avec la méthode remove.

Mobilité du calendrier

Le calendrier permet une gestion dynamique des réservations.

La fonction eventChange permet d'incrémenter les modifications dans la base de données lorsque l'on déplace les réservations sur le calendrier et que l'on prolonge cette dernière.

```
eventChange: function(info){
    // url de la fonction pour l'edit en mode ajax sans recharge de la page
    let url = '/hotel/gite/public/calendar_edit_'+(info.event.id);
    // données a envoyer des event suivant la position de l'event en question
    let s = info.event.start;
    let start = s.toLocaleDateString();
    let e = info.event.end;
    let end = e.toLocaleDateString();
    let data = {
        "title":info.event.title,
        "start": start,
        "end": end ,
    }
    console.log(e);
    // initialisation du XMLHttpRequest
    let xhr = new XMLHttpRequest;
    //console.log(JSON.stringify(data));
    // on envoie la request suivant la method PUT GET POST ... en fonction de
    // l'url mit plus haut
    xhr.open("PUT",url);
    // on envoie la request qui sera reprimt par le code php de la fonction
    // qui utilise l'url
    // La méthode JSON.stringify() convertit une valeur JavaScript en chaîne
    // JSON.
    xhr.send(JSON.stringify(data));
},
```

Conclusion

Qu'est ce que ce stage m'a apporté ?

Ce stage m'a apporté une confiance dans le développement d'une application backend que je n'avais pas. En effet, j'étais beaucoup plus à l'aise sur les problématiques du frontend, dorénavant je me sens autant à l'aise à devoir résoudre des problématiques du backend que du frontend.

J'ai rencontré quelques difficultés durant le stage. La 1ère à été le télétravail, cela est propre à chacun mais pour ma part cela a été pénalisant. En effet, ne pas être sur un lieu de travail n'a pas été propice à la concentration. Hors dans ce métier la réflexion est essentielle. Aussi, les échanges avec Flavien n'étaient pas aussi fluides à distance qu'elles ne l'auraient été en présentesielles. Je retiens donc que le télétravail n'est, à l'heure actuelle, pas fait pour moi.

La 2ème difficulté que j'ai rencontrée est : l'autonomie. Jérôme a voulu nous laisser très autonome, pour que nous puissions développer une application fonctionnelle le plus possible par nous-même. Nous avons passé beaucoup de temps à rechercher des solutions sur internet. Cela m'a aidé à beaucoup progresser, à réfléchir et à trouver des solutions par moi-même. Car l'autonomie est importante dans le développement. Cependant, je me suis rendu compte que nous n'avions pas toujours les connaissances techniques nécessaires à la réalisation d'un projet complet et complexe, ce qui me donne envie de m'améliorer constamment.

Le point essentiel que je retiens est ma progression sur le langage PHP et l'utilisation du framework Symfony.

Le projet se poursuivra

Le projet se poursuivra par d'autres stagiaires qui continueront à développer les différentes parties du cahier des charges.