



DOSSIER DE PROJET

Développement d'application web : Le salon virtuel

Access Code School (Lons-le-Saunier)

Auteur : Francis Pernot

Tuteur de stage : Damien Jeannot

Référent en entreprise : Thibault Drillien

Coach formateur ACS : Alain Merucci

Entreprise : Unsolite

Janvier – Mars 2022



UNSOLITE

ONLINE
F O R M A P R O

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à ces six mois de formations et deux mois de stage.

Tout d'abord, j'adresse mes remerciements à Damien Jeannot de m'avoir accueilli chez Unsolite ainsi qu'à Thibault Drillien pour avoir été disponible tout au long du stage et pour les nombreux conseils qu'il a pu m'apporter.

Pour ce qui est de la formation, je tiens à remercier le conseil régional de Bourgogne Franche-Comté qui fut d'une grande aide pour effectuer ma reconversion par le financement de cette formation.

Un grand merci à Alain Merucci notre Coach comme il aime que l'on ne l'appelle pas, pour son tempérament, sa bienveillance et bien sûr l'ensemble des dizaines de compétences qu'il m'a aidé à acquérir.

Enfin, je remercie les autres membres de la promotion pour leur humour, un certain nombre de lolcats, la motivation et l'aide qu'ils ont pu m'apporter ces huit derniers mois.

Liste des abréviations

Ajax : Asynchronous JavaScript XML

API : Application Programming Interface

CA : Crédit Agricole

CLI : Command Line Interface

CSRF : Cross Site Request Forgery

CSS : Cascade Style Sheet

DOM : Document Object Model

HTML : HyperText Markup Language

HTTP : Hypertext Transfer Protocol

HTTPS : Hypertext Transfer Protocol Secured

IDE : Integrated Development Environment

IP : Internet Protocol

JSON : JavaScript Object Notation

MVC : Model View Controller

PHP : PHP: Hypertext Preprocessor

SQL : Structured Query Language

URL : Uniform Resource Locator

WAMP : Windows Apache MySQL PHP

Table des matières

Remerciements	2
Liste des abréviations	3
Table des matières	4
Table des figures	6
Table des codes	6
I - Liste des compétences du référentiel qui sont couvertes par le projet	7
1. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	7
a) Réaliser une interface utilisateur web statique et adaptable	7
b) Développer une interface utilisateur web dynamique	7
2. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	8
a) Créer une base de données	8
b) Développer les composants d'accès aux données	8
c) Développer la partie back-end d'une application web ou web mobile	8
d) Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce	9
II - Résumé du projet	10
III - Cahier des charges, expression des besoins	11
IV - Spécifications techniques du projet	13
1) Outils utilisés lors du stage	13
2) Symfony	15
3) Stripe	15
V - Réalisations	16
1) Front-end	16
a) Création d'une extension Twig personnalisée	16
b) Création d'une animation de chargement de page	18
2) Back-end	20
a) Création de la base de données pour le système de paiement	20
b) Mettre en place le système de paiement	23
VI - Présentation du jeu d'essai	26

VII - Description de la veille effectuée sur les vulnérabilités de sécurité	30
1) Bundle de sécurité de Symfony	30
2) Les pare-feu (firewalls)	30
3) Les contrôles d'accès	31
VIII - Description d'une situation de travail ayant nécessité une recherche.....	32
IX - Extrait et traduction d'un site anglophone, utilisé dans le cadre du projet.....	33
Conclusion	35
Annexes	36

Table des figures

Figure 1 : Exemple de cahier des charges pour le ticket demandant de rendre paramétrable le choix de la page d'accueil du salon	12
Figure 2 : Interface de SourceTree	14
Figure 3 : Capture d'écran du tableau de bord de Stripe	15
Figure 4 : Liste des stands avant l'utilisation de l'extension Twig	16
Figure 5 : Liste des stands après l'utilisation de l'extension Twig	18
Figure 6: Capture d'écran de deux phases de l'animation de chargement	18
Figure 7 : Résultat de l'exécution des tests de la fonction testHallAccess()	29
Figure 8 : Résultat obtenu pour l'ajout d'une bulle d'aide dans un formulaire Symfony	32

Table des codes

Code 1 : Créer une extension Twig sous Symfony	17
Code 2 : Créer une animation en CSS	19
Code 3 : Afficher et cacher un élément en JavaScript à l'écoute d'un évènement	20
Code 4 : Exemple de fichier de migration autogénéré	22
Code 5 : Fonction permettant de créer et de récupérer une intention de paiement sous Stripe	24
Code 6 : Contrôleur du système de paiement	24
Code 7 : Intégration de l'API JavaScript de Stripe	25
Code 8 : Fonction permettant de vérifier l'autorisation d'accès d'un utilisateur à un hall	27
Code 9 : Test unitaire de la fonction userHaveAccessToHall	28
Code 10 : Jeux de données pour les tests unitaires	29
Code 11 : Configuration du fournisseur d'utilisateur pour le système de sécurité	30
Code 12 : Configuration d'un pare-feu pour le système de sécurité	31
Code 13 : Configuration des contrôles d'accès pour le système de sécurité	31

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

1. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

a) Réaliser une interface utilisateur web statique et adaptable

Lorsque l'on navigue sur le web, on utilise un navigateur, en allant sur un site, on entre une adresse dans l'URL, c'est l'adresse du site. Notre navigateur va utiliser les protocoles d'internet pour récupérer la page correspondant à l'adresse. La page est envoyée au navigateur (que l'on appelle client) par le server où elle est physiquement stockée. Cette page est en fait un fichier HTML que le navigateur va lire pour générer l'affichage.



HTML (HyperText Markup Language) est un langage de balisage qui permet de construire la structure d'une page web.

Lors de la lecture du fichier HTML, le navigateur crée le DOM (Document Object Model), c'est une interface de programmation avec une structure en arbre où chaque élément est un nœud. Cette interface permet à des scripts de modifier l'affichage même une fois que la page est chargée. Ces scripts sont écrits en JavaScript, un langage spécialement créé à cet effet.

Le fichier HTML peut faire appel à des fichiers CSS (Cascade Style Sheet) permettant de gérer le rendu des éléments affichés.

HTML et CSS permettent de réaliser des pages web statique et adaptable. Une page web statique est une page qui, à un instant donné s'affiche indistinctement, quelle que soit l'adresse IP qui l'appelle. Une page adaptable est une page qui se réorganise en fonction de la taille de la fenêtre qui l'affiche.

Il n'y a pas de pages web statiques dans l'application, car si l'utilisateur n'est pas connecté, il est automatiquement renvoyé à la page login.

b) Développer une interface utilisateur web dynamique

Une interface web dynamique est une page dont le contenu dépend de certains paramètres connus au moment du chargement de la page. C'est-à-dire qu'en appelant une même URL, on n'obtient pas forcément le même résultat. L'accès au salon étant restreint aux personnes inscrites, l'ensemble des pages du site sont dynamiques.

Une des missions que j'ai eu à effectuer lors de mon stage était justement de ne pas afficher certains stands à certains utilisateurs.

2. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

a) Créer une base de données

Une base de données est un ensemble de données structurées sous forme de tables. Chaque table possède un nom et un certain nombre d'attributs. Une bonne pratique est d'avoir pour chaque table un attribut Id (identifiant) qui est unique et non nul. Ainsi chaque enregistrement peut être distingué des autres en vérifiant un seul identifiant.

On se sert de ces identifiants pour relier les tables entre elles. On dit que l'on met en place des relations. Une relation consiste à placer un attribut dans une table dont la valeur correspond à l'Id de l'enregistrement d'une autre table. On appelle cette valeur clef étrangère.

Dans le cas où les enregistrements de deux tables auraient chacun besoin de plusieurs identifiants de l'autre table alors il est nécessaire de créer une nouvelle table qui peut ne contenir que deux paramètres les clefs étrangères des autres tables. On appelle ces tables intermédiaire des tables associatives.

La partie mise une interface de paiement présente la création de tables et la mise en place d'une relation dite ManyToOne.

b) Développer les composants d'accès aux données

Pour accéder aux données stockées dans la base de données, on utilise des requêtes SQL. Avec Symfony, l'accès aux données se fait via des objets appelés *repository* qui évitent dans la majorité d'avoir à écrire les requêtes à la main.



SQL est un langage informatique servant à effectuer des appels (requêtes) à des bases de données relationnelles. Le langage permet notamment de rechercher, ajouter, modifier et supprimer des données.

c) Développer la partie back-end d'une application web ou web mobile

La majorité de mon stage a été de développer des fonctionnalités dites « back » notamment dans l'admiration du site qui représente la majeure partie du code. Vous retrouvez dans la partie III – Cahier des charges, une partie des tâches back-end que j'ai pu réaliser.

d) Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

Enfin, j'ai mis en place un système de paiement en ligne à l'aide du bundle et de l'API (interface de programmation d'application) de Stripe, une société qui met à disposition des développeurs des outils simples pour ajouter différents moyens de paiement à leur site web. La partie V-2-b traite de cette intégration.

II - Résumé du projet

Mon stage s'est déroulé dans la start-up Unsolite installée au Village by CA, un incubateur d'entreprise basé à Besançon. Unsolite est une société de service numérique qui effectue des missions en tant que prestataire pour d'autres entreprises d'une part et qui développe des produits en interne d'autre part. Le Salon Virtuel est l'un de ces projets et celui sur lequel j'ai effectué mon stage. Il se présente comme une application web qui a pour vocation de remplacer ou de compléter des événements physiques comme des portes ouvertes d'universités ou tous types de salons. Ce projet a commencé à être développé en 2020 lors des premiers confinements, je suis donc arrivé sur une application déjà opérationnelle sur laquelle je devais ajouter de nouvelles fonctionnalités en tenant compte de l'ensemble des développements qui avaient déjà été réalisés.

Le salon virtuel est fait pour convenir à n'importe quel client dans des domaines très divers, ce qui nécessite d'en faire un outil extrêmement paramétrable. À la manière des salons classiques, chaque salon est composé de plusieurs halls, eux-mêmes composés de multiples stands dans lesquels on peut rencontrer plusieurs exposants. L'application propose également de participer à des forums et conférences. Enfin, elle met une messagerie de type « chat » à disposition des participants ainsi qu'un agenda. Chaque utilisateur étant invité à se créer un compte à l'arrivée sur le site pour participer au salon.

L'ensemble est développé sous Symfony, un framework PHP, implémentant un modèle MVC (Model View Controller).

III - Cahier des charges, expression des besoins

L'objectif de mon stage était de résoudre un certain nombre de problèmes où de bugs et d'implémenter de nouvelles fonctionnalités demandées par le client. Je n'ai donc pas eu de cahier des charges pour l'ensemble du stage, mais une liste de tickets qui pouvait aller de la simple remise en page d'éléments à l'ajout d'un paramètre au salon permettant de restreindre l'accès de certains groupes d'utilisateurs à certains halls ou stands en passant par l'ajout d'un loader lors de la navigation de page en page.

Voici une liste non exhaustive des missions que j'ai pu réaliser durant mon stage, toutes ne seront pas détaillées ce rapport :

- Ajout d'une animation (loader) avant et après le chargement d'une page de l'administration.
- Ajout d'un paramètre permettant aux organisateurs de restreindre l'accès aux stands et aux halls à certains groupes d'utilisateurs.
- Création de tests unitaires.
- Ajouter une double vérification de l'email et du mot de passe dans le formulaire d'inscription.
- Création d'une extension Twig pour clarifier le code et résoudre un problème d'affichage lorsque les images à afficher étaient introuvables.
- Ajouter une confirmation de l'email et du mot de passe avec les modules Symfony.
- Permettre aux conseillers de choisir les statistiques à afficher en fonction du stand qu'ils choisissent s'ils sont conseillers sur plusieurs stands.
- Mettre en place un système qui compte le nombre d'utilisateurs différents qui assistent à une conférence.
- Permettre l'accès à un salon sans authentification.
- Rendre paramétrable le choix de la page d'accueil du salon.
- Rendre l'accès à un salon payant.
- Ajouter la possibilité d'effectuer des paiements en ligne.

Voici le type de spécification que je pouvais avoir pour une mission, ici pour rendre paramétrable le choix de la page d'accueil du salon :

Paramétrage page par défaut d'ouverture du salon #222

Actuellement une fois connecté au salon le visiteur est automatiquement redirigé sur la page de présentation. Il faudrait pouvoir paramétrer la page sur laquelle le salon doit s'ouvrir par défaut.

Tâches techniques :

1. Ajouter dans les propriétés du salon un nouveau champ : "Page d'ouverture par défaut". (modification de l'entité, migration, formulaire)
2. Ce nouveau champ sera alimenté par une liste déroulante proposant les options suivantes :
 - Présentation
 - Halls
 - Stands
 - Conférences
 - Forum
3. Dans la classe LoginFormAuthenticator, modifier la méthode onAuthenticationSuccess au niveau de la redirection chargeant l'index de la présentation afin de charger dans une case cette nouvelle propriété de salon en fonction de l'option choisie et ainsi rediriger vers la bonne page (la page par défaut, si aucun paramétrage n'existe restera la page de présentation).

Figure 1 : Exemple de cahier des charges pour le ticket demandant de rendre paramétrable le choix de la page d'accueil du salon

IV - Spécifications techniques du projet

1) Outils utilisés lors du stage

Étant arrivé sur un projet avancé, je n'ai pas eu à choisir de technologie de développement. L'ensemble des fonctionnalités sont développées sous Symfony et des évolutions sont régulièrement effectuées pour maintenir le code à jour et profiter des dernières évolutions du framework. Actuellement, le projet fonctionne sous Symfony 6.0.4 et PHP 8.1.3.

Pour le développement, j'ai choisi d'utiliser Laragon comme WAMP (Windows Apache MySQL PHP) que j'avais déjà utilisé plutôt que Wamp Server que le reste de l'équipe utilisait. Un WAMP est une plateforme permettant d'exécuter des scripts PHP en local.



Un WAMP (Windows Apache MySQL PHP) est une plateforme permettant d'exécuter des scripts PHP localement.

Concernant les méthodes de travaux, nous avons utilisé Git pour versionner le code et GitHub pour le partager. En pratique, le projet est construit sur deux branches, la branche master où est rassemblé le code utilisé en production, et la branche develop où est rassemblé le code en cours de développement. Pour chaque ticket qui m'était assigné, je créais une nouvelle branche sur laquelle travailler et une fois le travail effectué, je prévenais le chef de projet que les fonctionnalités étaient prêtes avec une « pull request » pour qu'il puisse réviser et fusionner (merge) le code sur la branche develop.



Git est un logiciel de versionning, il permet de conserver un historique des modifications de chaque fichier d'un projet. Git est basé sur un modèle distribué : le code source est hébergé sur un serveur distant (dépôt) et chaque développeur télécharge l'intégralité du code source sur sa propre machine.



GitHub est une plateforme permettant d'héberger les dépôts Git et de partager leurs accès avec d'autres utilisateurs.

Pour gérer facilement le versionning, j'ai utilisé SourceTree, un logiciel qui permet de simplifier les interactions avec le dépôt Git.

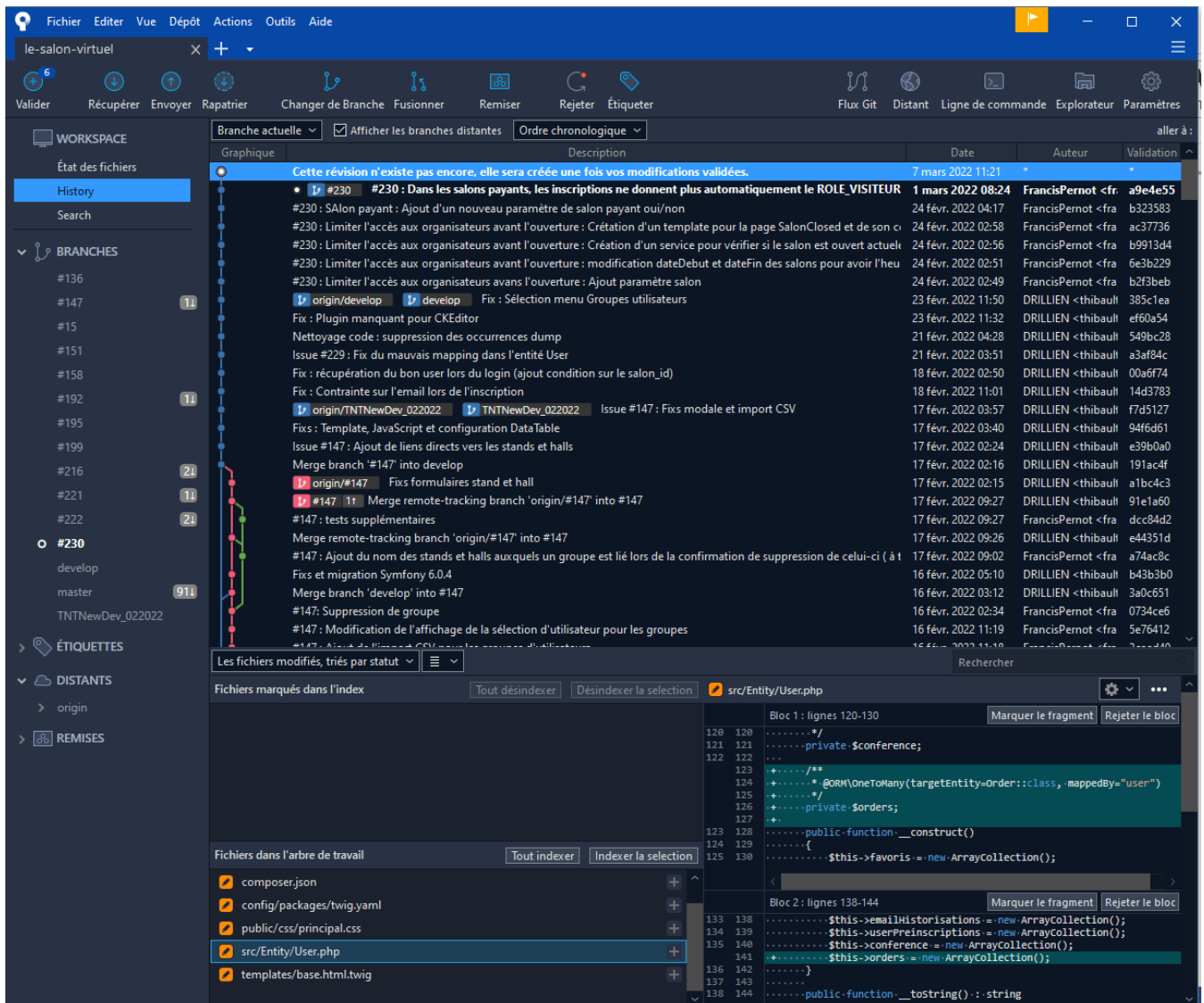


Figure 2 : Interface de SourceTree

Sur cette interface, on retrouve les principales composantes de Git :

- Dans le bandeau de gauche : les différentes branches du projet, on peut y voir pour chaque branche si la branche locale est à jour par rapport à la branche distante.
- Dans le bandeau du bas : SourceTree affiche tous les fichiers qui ont été modifiés et à droite les modifications ligne par ligne dans le fichier sélectionné.
- Dans le bandeau du haut, on retrouve les fonctions de bases de Git (push, pull, merge, checkout...).

Pour ce qui est du développement, j'utilise Visual Studio Code.



Un IDE (Integrated Development Environment) est un logiciel permettant d'être plus efficace lors de l'écriture de code. Ils intègrent de nombreux outils permettant d'automatiser certaines opérations.

Symfony est un Framework PHP basé sur un Modèle Vue Contrôleur (MVC) qui propose de nombreuses bibliothèques permettant de faciliter et ainsi de gagner en productivité lors du développement d'application web. Les différents avantages de Symfony vont être développés dans ce rapport puisque l'intégralité de mon stage a été réalisée sur cet outil.

J'ai eu à mettre en place un système de paiement pour l'application comme détaillé en V-2-b. Après avoir étudié un certain nombre de systèmes facilitant l'intégration de systèmes de paiement (PayPal, Lydia...), j'ai décidé d'utiliser Stripe pour sa relative simplicité d'implémentation.

Stripe est une société américaine de traitement de paiement en ligne qui propose des API Front et Back à destination des entreprises pour ajouter à leur site de multiples méthodes de paiement qui suivent les dernières recommandations de sécurité. Stripe est l'un des leaders du marché, il propose une documentation détaillée et une interface de test très détaillée permettant d'effectuer tous les tests fonctionnels nécessaires avant de passer le projet en phase de production. La figure 3 présente le tableau de bord de Stripe sur lequel on peut voir les paiements effectués par les utilisateurs de l'application.



V - Réalisations

1) Front-end

a) Création d'une extension Twig personnalisée

L'une des premières missions que j'ai eu à réaliser durant mon stage fut de faire disparaître le petit pictogramme qui remplace une image qui n'a pas pu être chargée comme le montre la figure 4.



Département Chimie

Le département Chimie forme les étudiants aux métiers de la chimie. Les futurs diplômés acquièrent toutes les connaissances et les compétences pour travailler dans tous les domaines où la chimie est présente : industries chimiques, pharmaceutiques, traitements de surface, aéronautique, automobile, énergie, agroalimentaire, environnement...



Département Génie Civil et Construction Durable

Le département GC-CD propose des formations à bac+3 : - un Bachelor Universitaire de Technologie (B.U.T.) en 3 ans, accessible postbac, avec 3 parcours ouverts à l'alternance à partir de la 2e année, - une licence professionnelle CTPEB, accessible à bac+2, ouverte à l'alternance.



Département Génie industriel et maintenance

Situé sur le Pôle universitaire de Vesoul, le département Génie industriel et maintenance propose deux diplômes : - un Bachelor universitaire de technologie (B.U.T.) GIM, parcours Management, méthodes, maintenance innovante (3MI) - une licence professionnelle Maintenance et technologie : systèmes pluritechniques, parcours Maintenance et énergétique

Figure 4 : Liste des stands avant l'utilisation de l'extension Twig

Pour ce faire, il suffit de ne pas faire appel à la balise image lorsque celle-ci n'existe pas. Le code était initialement le suivant :

```

```

On remarque que l'on utilise le bundle Vich Uploader qui est un bundle Symfony qui facilite l'accès aux fichiers.

Lorsque l'on affiche le chemin que renvoi `vich_uploader_asset(stand, 'imageStandFile')`, on obtient par exemple les résultats suivants :

- uploads/Salon/107/Stand/5ff85e9b0631d313663937.jpg dans le cas d'une image qui existe.
- uploads/Salon/107/Stand/5ffd7a5edebfb713808793.jpg dans le cas d'une image qui n'existe pas.

Il est impossible avec cette information de savoir si l'image existe ou non. On ne peut donc pas écrire simplement :

```
{% if vich_uploader_asset(stand, 'imageStandFile') %}
    
{% endif %}
```

On sait que PHP possède une fonction `file_exists()` dont on a besoin, mais Twig ne possède pas d'équivalent, j'ai donc décidé de créer une extension Twig pouvant être utilisée facilement comme ceci :


```
{% if twig_file_exists(vich_uploader_asset(stand, 'imageStandFile')) %}  
      
{% endif %}
```

Où `twig_file_exists` est une fonction de notre extension personnalisée.

Pour créer une extension Twig dans Symfony, il faut créer une classe qui étende `AbstractExtension` et créer une fonction appelée `getFunction` (si on veut créer un filtre, on l'appellera `getFilters`). Cette fonction doit renvoyer un objet `TwigFunction` qui prend en paramètre :

- Le nom de la fonction tel qu'il faudra l'appeler dans Twig.
- Un tableau contenant l'objet lui-même ainsi que la fonction qui devra être exécutée lors de l'appel.

Dans notre cas la fonction est très simple puisqu'on réutilise simplement la fonction `file_exists` de PHP en ajoutant un petit test pour avoir un booléen `false` lorsque la source est `null`.

```
PHP :  
class FileExistsExtension extends AbstractExtension  
{  
    public function getFunctions()  
    {  
        return [  
            new TwigFunction('twig_file_exists', [$this,  
'twig_file_exists']),  
        ];  
    }  
  
    public function twig_file_exists($src): bool  
    {  
        if($src){  
            return file_exists($src);  
        }  
        else{  
            return false;  
        }  
    }  
}
```

Code 1 : Créer une extension Twig sous Symfony

Finalement on obtient bien le résultat souhaité et la fonction `twig_file_exists()` sera utilisée pour l'affichage de chaque image.

Département Chimie

Le département Chimie forme les étudiants aux métiers de la chimie. Les futurs diplômés acquièrent toutes les connaissances et les compétences pour travailler dans tous les domaines où la chimie est présente : industries chimiques, pharmaceutiques, traitements de surface, aéronautique, automobile, énergie, agroalimentaire, environnement...



Département Génie Civil et Construction Durable

Le département GC-CD propose des formations à bac+3 : - un Bachelor Universitaire de Technologie (B.U.T.) en 3 ans, accessible postbac, avec 3 parcours ouverts à l'alternance à partir de la 2e année, - une licence professionnelle CTPEB, accessible à bac+2, ouverte à l'alternance.

Département Génie industriel et maintenance

Situé sur le Pôle universitaire de Vesoul, le département Génie industriel et maintenance propose deux diplômes : - un Bachelor universitaire de technologie (B.U.T.) GIM, parcours Management, méthodes, maintenance innovante (3MI) - une licence professionnelle Maintenance et technologie : systèmes pluritechniques, parcours Maintenance et énergétique

Figure 5 : Liste des stands après l'utilisation de l'extension Twig

b) Création d'une animation de chargement de page

Dans l'administration, le chargement de certaines pages peut parfois être long, et ce même entre le clic sur un lien et le chargement de la page suivante. Pour que l'expérience utilisateur soit meilleure, il a été décidé d'afficher une animation dès qu'un lien est actionné. L'animation doit donc se déclencher deux fois : au moment du clic, puis sur la page suivante dès le chargement de celle-ci, et finalement disparaître. L'animation que l'on va réaliser est présentée en figure 6.



Figure 6: Capture d'écran de deux phases de l'animation de chargement

```
CSS :
.loader {
  height: 100%;
  width: 100%;
  z-index: 10000;
  background-color: rgba(0, 0, 0, 0.2);
  overflow: hidden;
  position: fixed;
  display: flex;
  top: 0px;
  left: 0px;
  align-items: center;
  justify-content: center;
}
.loader__element {
  border-radius: 100%;
  border: 10px solid #d9230f;
  margin: calc(15px);
}
.loader__element:nth-child(1) {
  animation: preloader .6s ease-in-out alternate infinite;
}
.loader__element:nth-child(2) {
  animation: preloader .6s ease-in-out alternate .2s infinite;
}
.loader__element:nth-child(3) {
  animation: preloader .6s ease-in-out alternate .4s infinite;
}
@keyframes preloader {
  100% { transform: scale(2); }
}
```

Code 2 : Créer une animation en CSS

Pour faire une animation en CSS, on utilise la règle *@keyframes* qui permet de définir la séquence de l'animation. Ici, notre élément va doubler de taille entre la 1^{ère} et la dernière frame. Pour ce qui est des autres informations de l'animation, on utilise la propriété *animation* sur l'élément que l'on veut animer en précisant le nom de l'animation, la durée, la vitesse, la direction, le délai et la répétition.

Grâce à ce code, on va obtenir l'animation souhaitée, maintenant, il faut l'afficher et la faire disparaître au bon moment. Pour ce faire nous allons utiliser du JavaScript :

```
JavaScript :
let loader = document.querySelector(".loader");
let navLinks = document.querySelectorAll(".nav-item.loading, .content-
wrapper.loading");
for(navLink of navLinks){
    navLink.addEventListener("click", function(){
        loader.style.display = "flex";
    })
};

document.addEventListener('readystatechange', event => {
    if (event.target.readyState === "complete") {
        loader.style.display = "none";
    }
});
```

Code 3 : Afficher et cacher un élément en JavaScript à l'écoute d'un évènement

Le principe est de placer un écouteur sur tous éléments qui doivent déclencher l'animation et de faire apparaître le loader au moment de la détection d'un clic. Au chargement d'une page, le loader est initialement affiché et il sera masqué dès que l'état du document passe sur « *complete* ».

2) Back-end

a) Création de la base de données pour le système de paiement

La dernière mission que j'ai eu à réaliser lors de mon stage est la mise en place d'un système de paiement pour le salon. Il existe plusieurs plateformes qui permettent d'intégrer un tel système. J'ai choisi d'utiliser Stripe pour sa facilité d'implémentation.

Avant de créer le système de paiement, on va avoir besoin d'un certain nombre d'éléments supplémentaire dans la base de données. Un achat c'est un utilisateur qui achète un produit, on a déjà la table *user*, il nous faut une table *product* et une table *order* pour pouvoir garder une trace des commandes effectuées.

Pour cela, on utilise la CLI (Command Line Interface) de Symfony qui va nous permettre de gagner beaucoup de temps. On veut créer une entité *Order* qui possèdera deux clefs étrangères vers les tables *User* et *Product* ainsi que le prix, la date et la référence de l'achat. Pour créer une table, on utilise la commande *make:entity*.

```
PS C:\Users\Acs\Documents\Cours\Projet\Unsolite\le-salon-virtuel> php bin/console make:entity
Class name of the entity to create or update (e.g. DeliciousPuppy):
> order
order

created: src/Entity/Order.php
created: src/Repository/OrderRepository.php

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

Cette commande nous demande alors le nom de la table puis deux fichiers sont créés : l'entité *Order* et son repository.

Ensuite, la CLI me propose d'ajouter des champs, je vais donc ajouter un champ *user* qui sera de type relation (on peut afficher tous les types de champs avec un point d'interrogation). La commande *relation* permet d'avoir une aide pour choisir la relation. Dans notre cas, une commande ne peut être associée qu'à un utilisateur et un utilisateur peut être associé à plusieurs commandes ce qui correspond à la phrase *Each Order relates to (has) one User. Each User can relate to (can have) many Order objects*. La relation est donc ManyToOne. Ensuite, j'indique que la propriété ne peut pas être nulle (il ne peut pas y avoir de commande sans utilisateur). Enfin, la CLI me propose de créer des méthodes dans l'entité *User* pour récupérer facilement les commandes effectuées par un utilisateur, c'est extrêmement pratique donc je mets oui. On peut ensuite ajouter d'autres propriétés de la même manière. L'annexe 1 permet de visualiser les étapes importantes de la séquence.

À ce stade, Symfony créé l'entité *Order* et mis à jour l'entité *User* mais je n'ai encore rien sur ma base de données. Il faut faire appel à la commande *make:migration* qui va créer des fichiers de migration. Un exemple de fichier de migration est présenté en dans le code 4, on voit qu'il contient les requêtes SQL pour la création des tables.



Les fichiers de migrations sont une des forces de Symfony. Ce sont des fichiers qui enregistrent toutes les modifications effectuées sur la base de données. Il suffit de les exécuter pour avoir la base de données. Ainsi il suffit de partager ces fichiers pour travailler sur une base de données identique à celle de ses collaborateurs. (Dans la forme, mais pas dans le contenu)

```

migration/Version20220301082051.php
/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220301082051 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        $this->addSql('CREATE TABLE `order` (id INT AUTO_INCREMENT NOT
        NULL, user_id INT DEFAULT NULL, product_id INT DEFAULT NULL, reference
        VARCHAR(255) NOT NULL, [...]);
        $this->addSql('CREATE TABLE product (id INT AUTO_INCREMENT NOT
        NULL, reference VARCHAR(255) NOT NULL, name VARCHAR(255) NOT NULL,
        description VARCHAR(255) NOT NULL, [...]);
        $this->addSql('ALTER TABLE `order` ADD CONSTRAINT
        FK_F5299398A76ED395 FOREIGN KEY (user_id) REFERENCES user (id)');
        $this->addSql('ALTER TABLE `order` ADD CONSTRAINT
        FK_F52993984584665A FOREIGN KEY (product_id) REFERENCES product (id)');
    }
}

```

Code 4 : Exemple de fichier de migration autogenerated

Nous avons donc à ce stade :

- Une table Product représentant les “produits” qui vont être vendus.

le_salon_virtuel product	
id	int(11)
reference	varchar(255)
name	varchar(255)
description	varchar(255)
description_short	varchar(255)
price	varchar(255)
image	varchar(255)

- Une table *Order* représentant les commandes effectuées. La table *Order* est reliée en ManyToOne à la table User ainsi qu'à la table Product dans le but de savoir qui à acheter quel produit. On y ajoute un certain nombre d'informations proposé par Stripe ainsi que la date et le prix payé. Le prix est précisé dans les deux tables, car dans le cas du changement de prix d'un produit (variation, réduction...) il peut être utile de savoir combien le produit a réellement été vendu.

le_salon_virtuel order	
id	int(11)
user_id	int(11)
product_id	int(11)
reference	varchar(255)
stripe_token	varchar(255)
brand_stripe	varchar(255)
last4_stripe	varchar(255)
id_charge_stripe	varchar(255)
status_stripe	varchar(255)
created_at	datetime
updated_at	datetime
price	double

b) Mettre en place le système de paiement

Maintenant, nous pouvons passer au système de paiement en tant que tel.

Pour installer le package Stripe dans notre projet, on utilise composer avec la commande suivante :

Package : `composer require stripe/stripe-php`

Sur la plateforme en ligne de Stripe, dans l'onglet développeur, on a accès aux clefs publiques et privées. Je les stocks directement dans le fichier d'environnement .env pour y avoir accès facilement. Il y a 4 clefs : 2 jeux de pair privé/public, l'un pour faire des tests l'autre pour utiliser en production (live dans

```
.env
STRIPE_PUBLIC_KEY_TEST=pk_test_51KYPUGLwWIuqLC5NVqMVMlm7ON2IviiAVRy1LMeOk1
VDirNEpi0CoDB4Bn6GTAb1DLdJyUTSTj71juzA47YDxljl00YoSTGZCJ
STRIPE_SECRET_KEY_TEST=sk_test_51KYPUGLwWIuqLC5NvIliYrCSnSsQ6HJ3AfujJN6hYO
oRAlyPBjxcQtmjb7iXyDqTEq1yFkx68QGAqmBtsuenmXil00vbZ8pb4d

STRIPE_PUBLIC_KEY_LIVE=clef_public_prod
STRIPE_SECRET_KEY_LIVE=clef_prive_prod
```

Stripe).

Ensuite, nous allons ajouter les clefs publiques dans le fichier de configuration de Twig pour y avoir accès facilement depuis les templates.

```
#config/packages/twig.yaml
twig:
  globals:
    app_environment: '%env(APP_ENV)%'
    stripe_public_key_test: '%env(STRIPE_PUBLIC_KEY_TEST)%'
    stripe_public_key_live: '%env(STRIPE_PUBLIC_KEY_LIVE)%'
```

Pour le paiement, nous allons utiliser l'API Front que propose Stripe puis l'API Back pour initialiser le paiement et récupérer certaines informations sur le paiement.

En pratique, l'utilisateur va (1) regarder la liste des produits, (2) choisir le produit qu'il souhaite acheter ce qui va l'envoyer vers un formulaire de paiement, (3) remplir le formulaire et l'envoyer et enfin (4) voir la confirmation de son achat.

Dans cette partie nous n'allons traiter que les parties (3) et (4).

Nous commençons donc par créer un contrôleur avec une `@Route` qui prend un paramètre en entrée qui est l'id du produit sélectionné. Ce contrôleur va afficher le formulaire de paiement. À ce stade, le paiement ne semble pas avoir commencé, en réalité les normes de sécurité demandent d'avoir déjà créé à ce moment-là un canal de connexion entre le site et le système de paiement. En pratique, on crée une intention de paiement qui se matérialise par la génération d'une clef. Pour générer cette clef, nous allons créer une fonction `getPaymentIntent` dans un fichier de Service. Cette fonction utilise l'API de Stripe auquel on donne trois informations : le prix de l'achat en centimes, la devise et la méthode de paiement et qui renvoie un objet contenant l'intention de paiement.

```
Service/StripeService.php
public function getPaymentIntent(Product $product)
{
    \Stripe\Stripe::setApiKey($this->privateKey);
    return \Stripe\PaymentIntent::create([
        'amount' => $product->getPrice() *100, #(en centimes)
        'currency' => 'eur',
        'payment_method_types' => ['card']
    ]);
}
```

Code 5 : Fonction permettant de créer et de récupérer une intention de paiement sous Stripe

```
src/Controller/StripeService.php
/**
 * @Route("/paiement/{id}", "paiement")
 * @param Product $product
 */
public function payment(Product $product): Response
{
    $intent = $this->stripeService->getPaymentIntent($product);
    $intent = $intent['client_secret'] ?? null;
    return $this->render('payment/index.html.twig', [
        'user' => $this->getUser(),
        'intentSecret' => $intent,
        'product' => $product
    ]);
    return $this->render('payment/index.html.twig');
}
```

Code 6 : Contrôleur du système de paiement

Maintenant que nous avons le contrôleur, nous allons créer l'interface et effectuer le paiement. Une interface de paiement est proposée par Stripe, elle requière simplement un formulaire composé de deux div avec les ids *card-elements* et *card-errors*. Ainsi qu'un script JavaScript qui est stocké sur leur server et que l'on appelle comme présenté dans le code ci dessous.

```
Templates/payment/index.html.twig
form action="{{ path('subscription_paiement', {'id': product.id }) }}"
method="post" id="payment-form">
  <div class="form-row">
    <div id="card-elements"></div>
    <script src="https://js.stripe.com/v3/"></script>
    <div id="card-errors" role="alert"></div>
  </div>
  <button class="btn btn-primary mt-4">
    Acheter {{ product.price }} €
  </button>
</form>
```

Code 7 : Intégration de l'API JavaScript de Stripe

Une fois l'affichage réalisé, on utilise l'API de Stripe pour créer un objet Stripe auquel on donne notre clef publique. Depuis cet objet, on crée les éléments d'interface qui permettent à l'utilisateur d'entrer ses données de paiements.

Le paiement doit être effectué lorsque l'utilisateur soumet le formulaire. Je crée donc un écouteur de type *submit* sur le formulaire puis on appelle la méthode *confirmCardPayment* de l'objet Stripe. On passe l'intention de paiement en paramètre ainsi que d'autres informations comme spécifiées dans la documentation. Si tout se passe bien, le paiement est effectué et on renvoie l'utilisateur vers une page de confirmation. Nous en avons terminé pour ce qui est du paiement en lui-même, mais il peut être intéressant de sauvegarder certaines informations de paiement.

Dans le service que nous avons créé, ajoutons une méthode *paiementInfo* pour récupérer les informations de paiement qui nous intéressent. Elles sont accessibles via l'API back de Stripe de la même manière qu'en JavaScript, en fournissant notre clef privée à l'API et en appelant la méthode *retrieve* qui prend en paramètre la clef du paiement effectué. De cette manière, on obtient un tableau contenant de multiples informations que l'on peut enregistrer dans notre table *Order*. Vous pouvez consulter un exemple de tableau en Annexe 2.

VI - Présentation du jeu d'essai

Une des fonctionnalités qui manquait au salon virtuel était de limiter l'accès à certains halls et/ou stands à certains utilisateurs. Il m'a été demandé de réaliser cette fonctionnalité et ensuite de créer des tests unitaires pour vérifier le bon fonctionnement du code.

Pour implémenter la fonctionnalité, j'ai donné la possibilité aux administrateurs de créer des groupes d'utilisateurs et d'assigner ces groupes à des halls ou à des stands. Ensuite, lors de l'affichage de la liste des stands on vérifie si l'utilisateur de la session actuel est dans l'un des groupes ayant accès au stand, si oui on affiche le stand, sinon le stand reste caché. La même sécurité a été utilisée à l'arrivée sur un stand au cas où l'utilisateur connaisse l'URL pour y accéder directement.

La difficulté est qu'un salon peut être paramétré de 4 manières différentes : aucune restriction d'accès, restriction d'accès sur les halls, restriction d'accès sur les stands et restriction d'accès sur les halls et sur les stands. De plus, il faut savoir que chaque stand est inclus dans un hall. Enfin, dernière contrainte, on estime que si un stand n'a aucun groupe auquel l'accès est restreint alors tous les utilisateurs y ont accès.

Ainsi, un certain nombre de questions se pose, prenons un exemple avec un Hall H1 restreint au Groupe G1, dans lequel il y a un Stand S1 restreint au groupe G2. (On se place dans le cas où le salon possède des restrictions d'accès sur les halls et sur les stands).

On note U l'utilisateur, voici quelque cas simple :

- Si U appartient à G1 et que G2 est vide alors on accepte l'utilisateur
- Si U appartient à G1 et à G2 alors on accepte l'utilisateur
- Si U n'appartient ni à G1 ni à G2 alors on accepte l'utilisateur

Voici quelques exemples particuliers qui demandent réflexion :

- Si U appartient à G2, mais pas à G1 (et G1 non vide)

Ces tests étant utilisés dans de nombreux fichiers différents, j'ai créé un service `UserRestriction` ce qui permettra de rendre facilement accessibles les méthodes de tests. J'ai créé deux méthodes `userHaveAccessToHall` et `userHaveAccessToStand` qui renvoient un booléen : `true` si l'utilisateur a accès au hall et `false` sinon. L'une de ces fonctions est détaillée si dessous :

```

service/UserRestriction.php
class UserRestriction extends AbstractController
{
    public function userHaveAccessToHall($userEmail, $hallGroupes,
    $salonParamRestrictionAcces): bool {
        if($salonParamRestrictionAcces ==
        SalonParametreRepository::$VALEUR_RESTRICTION_ACCES_HALLS_STANDS ||
        $salonParamRestrictionAcces ==
        SalonParametreRepository::$VALEUR_RESTRICTION_ACCES_HALLS){
            $emailsHall = [];
            foreach($hallGroupes as $groupeHall){
                -> $emailsHall = array_merge($emailsHall, $groupeHall-
                >getEmailsAsArray());
            }
            if($emailsHall != [] && !in_array($userEmail, $emailsHall) ){
                return false;
            }
        }
        return true;
    }
}

```

Code 8 : Fonction permettant de vérifier l'autorisation d'accès d'un utilisateur à un hall

Étant donné le grand nombre de cas possibles et que les manipulations pour réaliser les tests sont longues, j'ai mis en place des tests unitaires qui permettent de créer rapidement des jeux de données à tester. Le principe d'un test unitaire est d'appeler une « fonction » en spécifiant le résultat attendu pour telle ou telle donnée. Si la fonction renvoie le résultat attendu alors le test est passé, sinon il est rejeté.



Les tests unitaires sont des procédures permettant de vérifier le bon fonctionnement d'une partie précise du code. Dans les gros projets et les projets en équipe, c'est une bonne pratique de mettre en place des tests unitaires pour chaque fonctionnalité. Il est possible d'activer ces procédures automatiquement avant d'effectuer un merge sur Git pour s'assurer que toutes les fonctionnalités fonctionnent encore après les modifications.

Pour créer des tests avec Symfony, j'ai utilisé la librairie PHPUnit. Elle met à disposition plusieurs méthodes dédiées aux tests unitaires, notamment *assertSame()* qui permet d'effectuer un test qui prend en paramètre le résultat attendu et la fonction à tester avec ces paramètres.

L'avantage de PHPUnit est que l'on peut rapidement créer un jeu de données pour tester des variables. Par exemple, *\$userEmail* est facile à remplacer, il suffit de donner une chaîne de caractère.

Un problème survient quand on utilise des objets. On ne peut pas ou on ne veut pas avoir à créer un jeu de données avec des objets complexe. Par exemple, *\$hallGroupes* est un tableau contenant des objets de type Groupe auxquels on va appeler la méthode *getEmailsAsArray()* pour avoir la liste des emails des

personnes du groupe pour vérifier si l'utilisateur en fait partie. Ici on voudrait simplement avoir à passer un tableau d'emails et pas des objets Groupe. Il va donc falloir créer des doublures (mock) d'objet. Pour cela, PHPUnit met à disposition la fonction *createMock()* qui crée une doublure vide de l'objet passé en paramètre. Vu que l'objet est vide, l'appel à *\$groupeHall->getEmailsAsArray()* renverra systématiquement *null*. Il faut donc avant d'exécuter le test, prévenir le test que notre objet va recevoir un appel à la méthode *getEmailsAsArray()* qui doit renvoyer notre tableau d'email. Pour ce faire, on fait appel à une suite de méthodes sur notre mock : *expects()* pour indiquer combien de fois la méthode va être appelée puis *method()* pour préciser le nom de la méthode et enfin *willReturn()* pour spécifier ce qui va être retourné.

```
src/service/UserRestrictionTest.php
class UserRestrictionTest extends TestCase
{
    /**
     * @dataProvider dataGroupes
     */
    public function testHallAccess($emailUser, $dataGroupes, $paramAcces,
    $expected){

        $groupes = new ArrayCollection();
        foreach($dataGroupes as $dataGroupe){
            $groupeI = $this->createMock(Groupe::class);
            $groupeI->expects($this->any())
                ->method('getEmailsAsArray')
                ->willReturn($dataGroupe);
            $groupes->add($groupeI);
        }
        $userRestriction = new UserRestriction();
        $this->assertSame($expected, $userRestriction-
        >userHaveAccessToHall($emailUser, $groupes, $paramAcces));
    }
}
```

Code 9 : Test unitaire de la fonction userHaveAccessToHall

Pour effectuer plusieurs tests simultanément on utilise l'annotation *@dataProvider dataGroupes* où *dataGroupe* est une fonction qui renvoie un tableau de valeurs où chaque valeur correspond à un jeu de test. Le code ci-dessous présente deux jeux de données, on y retrouve bien l'email de l'utilisateur, la liste d'email ayant accès au hall, le paramètre de salon et le résultat attendu.

```
src/service/UserRestrictionTest.php
public function dataGroupes(){
    $groupesHall = [
        ["inHall@orange.fr", "chevre@orange.fr"],
        ["autre@orange.fr", "renard@orange.fr"],
    ];
    return [
        ["inHall@orange.fr",
            $groupesHall,
            SalonParametreRepository::$VALEUR_RESTRICTION_ACCES_HALLS_STANDS,
            true
        ],
        ["notIn@orange.fr",
            $groupesHall,
            SalonParametreRepository::$VALEUR_RESTRICTION_ACCES_HALLS_STANDS,
            false
        ],
    ],
]
```

Code 10 : Jeux de données pour les tests unitaires

Enfin pour exécuter les tests, on exécute `vendor/bin/phpunit --filter testHallAccess` dans l'invite de commande. On obtient un résultat comme celui-là quand tous les tests sont réussis :

```
Testing
..... 11 / 11 (100%)
Time: 00:00.126, Memory: 12.00 MB
OK (11 tests, 11 assertions)
```

Figure 7 : Résultat de l'exécution des tests de la fonction `testHallAccess()`

VII - Description de la veille effectuée sur les vulnérabilités de sécurité

Symfony fournit de nombreux outils pour sécuriser les applications. Certains outils de sécurité liés à HTTP, tels que les cookies de session sécurisée ou encore la génération de jeton CSRF. Dans cette partie, nous allons nous intéresser au bundle Security.

1) Bundle de sécurité de Symfony

Pour installer le bundle de sécurité on exécute la commande :

```
composer require Symfony/security-bundle
```

L'importation du bundle Security va créer un SecurityController, un LoginFormAuthenticator et un template /security/login.html.twig.

Ce qui est important à comprendre c'est que ce module est lié à l'entité User qui représente l'utilisateur en termes d'authentification. Pour créer cette entité, on utilise la commande :

```
Symfony console make:user
```

Cette commande ajoute également la configuration d'un fournisseur d'utilisateurs que l'on peut modifier dans le fichier de configuration. Ici par exemple, on spécifie que ce serait l'email de l'utilisateur qui serait utilisé comme identifiant de l'utilisateur plutôt que son id comme par défaut.

```
config/packages/security.yaml
security:
  providers:
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email
```

Code 11 : Configuration du fournisseur d'utilisateur pour le système de sécurité

2) Les pare-feu (firewalls)

Le pare-feu est le cœur du système d'authentification : le pare-feu définit quelles parties de l'application sont sécurisées et comment les utilisateurs peuvent s'authentifier. On utilise un provider pour déterminer les pages qui nécessitent que l'utilisateur soit authentifié, s'il n'y a pas de provider, toutes les pages sont prises en compte.

On utilise login_path pour spécifier le nom de la Route vers laquelle rediriger les visiteurs non authentifiés lorsqu'ils tentent d'accéder à une page sécurisée. On peut également mettre en place une authentification personnalisée qui doit être une classe qui implémente

AuthenticationEntryPointInterface. Le code ci-dessous montre un exemple de « pare-feu » avec une authentification personnalisée.

```
config/packages/security.yaml
security:
  firewalls:
    main:
      form_login:
        login_path: login
        custom_authenticator: App\Security\CustomAuthenticator
        entry_point: form_login
      logout:
        path: app_logout
```

Code 12 : Configuration d'un pare-feu pour le système de sécurité

3) Les contrôles d'accès

Dans le cas du salon, nous n'avons pas uniquement besoin que l'utilisateur soit authentifié, il faut également aussi distinguer les administrateurs, les organisateurs, les exposants des visiteurs. Pour ce faire, le bundle de sécurité propose d'utiliser les contrôles d'accès (Access Control).

Le principe d'utilisation et l'implémentation des contrôles d'accès est simple ce qui rend cet outil redoutablement efficace. Le principe est d'affecter des rôles à nos utilisateurs ainsi qu'à nos pages et uniquement les utilisateurs ayant un rôle correspondant à une page pourront y accéder.

En pratique, il suffit d'ajouter à l'entité User un tableau qui sera aussi en base de données dans lequel on liste tous les rôles de l'utilisateur. Ensuite dans le fichier de configuration de sécurité on utilise l'écriture `access_control` comme exposé ci-dessous :

```
config/packages/security.yaml
security:
  access_control:
    - { path: ^/administration, roles: [ROLE_ADMIN, ROLE_ORGANISATEUR,
    ROLE_EXPOSANT] }
    - { path: ^/login, roles: [PUBLIC_ACCESS] }
```

Code 13 : Configuration des contrôles d'accès pour le système de sécurité

Ici, on spécifie que toutes les pages commençant par `"/administration"` sont restreintes aux utilisateurs ayant pour rôle au moins l'un des suivants : `ROLE_ADMIN`, `ROLE_ORGANISATEUR`, `ROLE_EXPOSANT`.

On utilise le rôle `PUBLIC_ACCESS` pour spécifier une page qui est accessible à tout le monde.

VIII - Description d'une situation de travail ayant nécessité une recherche

Symfony fournit des outils pour créer très rapidement des formulaires, ils sont regroupés dans l'interface FormBuilderInterface. On peut spécifier en quelques lignes de code les champs à créer, le type de champs, les labels, les attributs... Ensuite, on peut afficher l'ensemble du formulaire avec un appel simple à la fonction Twig `form()`.

Dans un formulaire d'administration du salon, il était demandé par le client d'ajouter un champ d'aide pour spécifier la taille des images à sélectionner comme présenter en Figure 8 :

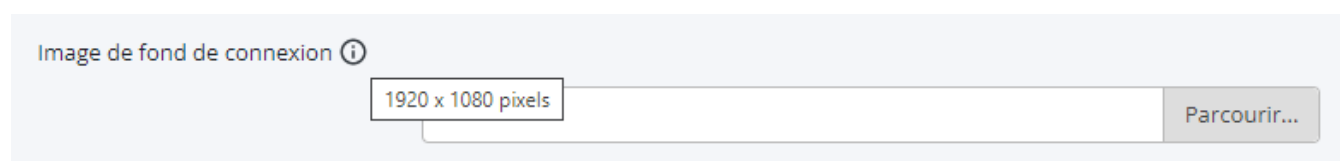
The image shows a snippet of a web form. It has a label "Image de fond de connexion" followed by an information icon (a small 'i' inside a circle). Below the label is a text input field containing the value "1920 x 1080 pixels". To the right of the input field is a button labeled "Parcourir...".

Figure 8 : Résultat obtenu pour l'ajout d'une bulle d'aide dans un formulaire Symfony

Le formulaire étant généré automatiquement par la fonction `form()`, je ne savais pas comment ajouter un élément au milieu. J'ai donc fait la recherche « add html element to symfony form » et j'ai eu comme premier résultat la page "form customisation" de la documentation de Symfony :

https://symfony.com/doc/current/form/form_customization.html.

Cette documentation m'a permis de comprendre que le plus simple était d'ajouter lors de la création du formulaire un attribut `title` à mon élément et de gérer l'affichage du petit « i » en CSS plutôt que de réécrire ligne après ligne tout le formulaire.

IX - Extrait et traduction d'un site anglophone, utilisé dans le cadre du projet

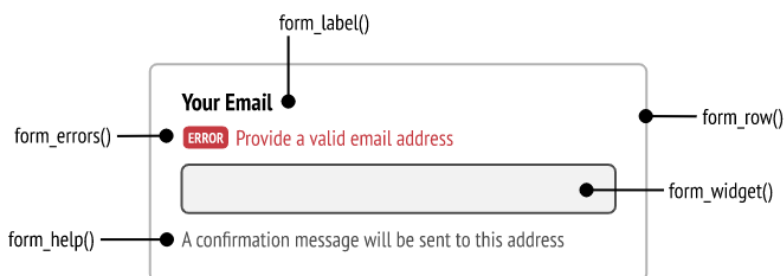
How to Customize Form Rendering

Symfony gives you several ways to customize how a form is rendered. In this article you'll learn how to make single customizations to one or more fields of your forms. If you need to customize all your forms in the same way, create instead a form theme or use any of the built-in themes, such as the Bootstrap theme for Symfony forms.

Form Rendering Functions

A single call to the `form()` Twig function is enough to render an entire form, including all its fields and error messages.

The next step is to use the `form_start()`, `form_end()`, `form_errors()` and `form_row()` Twig functions to render the different form parts so you can customize them adding HTML elements and attributes:



The `form_row()` function outputs the entire field contents, including the label, help message, HTML elements and error messages. All this can be further customized using other Twig functions, as illustrated in the following diagram:

The `form_label()`, `form_widget()`, `form_help()` and `form_errors()` Twig functions give you total control over how each form field is rendered, so you can fully customize them.

Comment personnaliser le rendu des formulaires

Symfony propose de multiples manières pour personnaliser l'affichage d'un formulaire. Dans cet article, vous allez apprendre comment personnaliser un ou plusieurs champs de vos formulaires. Si vous avez besoin de personnaliser tous vos formulaires de la même manière, il est préférable de créer un thème

de formulaire ou d'utiliser l'un des thèmes préintégrés, comme le thème Bootstrap pour les formulaires Symfony.

Fonctions de rendu de formulaire

Un appel unique à la fonction Twig `form()` est suffisant pour afficher l'intégralité d'un formulaire, incluant tous ces champs ainsi que les messages d'erreurs.

L'étape suivante est d'utiliser les fonctions Twig `form_start()`, `form_end()`, `form_errors()` et `form_row()` pour afficher les différentes parties du formulaire que vous pouvez personnaliser en ajoutant des éléments HTML et des attributs.

La fonction `form_row()` renvoi l'entièreté du contenu d'un champ, incluant le label, le message d'aide, les éléments HTML et les messages d'erreurs. Tout cela peut être personnalisé en utilisant d'autres fonctions Twig comme illustré dans le diagramme suivant :

Les fonctions Twig `form_label()`, `form_widget()`, `form_help()` et `form_errors()` donnent un contrôle total sur la façon dont chaque champ du formulaire est rendu, pour que vous puissiez totalement les customiser.

Conclusion

Ce stage, d'une durée de deux mois m'a permis de mettre en place dans un contexte professionnel les compétences et méthodes de travail acquises lors des six mois qui l'ont précédé. J'ai pu constater que je me débrouillais plutôt bien malgré quelques difficultés initiales pour prendre en main de Framework et le projet.

J'ai également pris beaucoup de plaisir durant ce stage ce qui renforce ma motivation pour devenir développeur web. J'envisage maintenant de trouver un emploi pour parfaire mes compétences que ce soit technique ou non, dans le but à terme de me mettre à mon compte.

Annexes

Annexe 1 : Étapes de la création de propriétés avec l'interface de commande de Symfony.

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

> user

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

> relation

relation

What class should this entity be related to?:

> User

User

What type of relationship is this?

Type	Description
ManyToOne	Each Order relates to (has) one User. Each User can relate to (can have) many Order objects
OneToMany	Each Order can relate to (can have) many User Each User relates to (has) one Order
ManyToMany	Each Order can relate to (can have) many User Each User can also relate to (can also have) many Order objects
OneToOne	Each Order relates to (has) exactly one User Each User also relates to (has) exactly one Order.

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:

> ManyToOne

ManyToOne

Is the Order.user property allowed to be null (nullable)? (yes/no) [yes]:

> no

Do you want to add a new property to User so that you can access/update Order objects from it - e.g. \$user->getOrders()? (yes/no) [yes]:

> yes

A new property will also be added to the User class so that you can access the related Order objects from it.

New field name inside User [orders]:

>

updated: src/Entity/Order.php

updated: src/Entity/User.php

Annexe 2 : Tableau de données JSON renvoyé par Stripe après un paiement.

```
Stripe\PaymentIntent JSON: {
  "id": "pi_3KblrZLwWIuqLC5N00EXLF62",
  "object": "payment_intent",
  "amount": 3500,
  "amount_capturable": 0,
  "amount_received": 3500,
  "application": null,
  "application_fee_amount": null,
  "automatic_payment_methods": null,
  "canceled_at": null,
  "cancellation_reason": null,
  "capture_method": "automatic",
  "charges": {
    "object": "list",
    "data": [
      {
        "id": "ch_3KblrZLwWIuqLC5N04kBMUiT",
        "object": "charge",
        "amount": 3500,
        "amount_captured": 3500,
        "amount_refunded": 0,
        "application": null,
        "application_fee": null,
        "application_fee_amount": null,
        "balance_transaction": "txn_3KblrZLwWIuqLC5N0Fhia42u",
        "billing_details": {
          "address": {
            "city": null,
            "country": null,
            "line1": null,
            "line2": null,
            "postal_code": "42424",
            "state": null
          },
          "email": null,
          "name": "test@test.com",
          "phone": null
        },
        "calculated_statement_descriptor": "Stripe",
        "captured": true,
        "created": 1646918234,
        "currency": "eur",
        "customer": null,
        "description": null,
        "destination": null,
        "dispute": null,
        "disputed": false,
        "failure_code": null,
        "failure_message": null,
        "fraud_details": [],
        "invoice": null,
        "livemode": false,
        "metadata": [],
        "on_behalf_of": null,
        "order": null,
        [...]
      ]
    }
  }
}
```

```

    "outcome": {
      "network_status": "approved_by_network",
      "reason": null,
      "risk_level": "normal",
      "risk_score": 16,
      "seller_message": "Payment complete.",
      "type": "authorized"
    },
    "paid": true,
    "payment_intent": "pi_3KblRZLwWIuqLC5N00EXLF62",
    "payment_method": "pm_1KblriLwWIuqLC5NQJlVQXLv",
    "payment_method_details": {
      "card": {
        "brand": "visa",
        "checks": {
          "address_line1_check": null,
          "address_postal_code_check": "pass",
          "cvc_check": "pass"
        },
        "country": "US",
        "exp_month": 4,
        "exp_year": 2024,
        "fingerprint": "fcRUe9E7Q1ByRMnx",
        "funding": "credit",
        "installments": null,
        "last4": "4242",
        "network": "visa",
        "three_d_secure": null,
        "wallet": null
      },
      "type": "card"
    },
    "receipt_email": null,
    "receipt_number": null,
    "receipt_url":
"https://pay.stripe.com/receipts/acct_1KYPUGLwWIuqLC5N/ch_3KblRZLwWIuqLC5N04kBMUiT/rcpt_LIMahroP7G0e0nIvzb0Gchh4hfb6R19",
    "refunded": false,
    "refunds": {
      "object": "list",
      "data": [],
      "has_more": false,
      "total_count": 0,
      "url":
"\v1\charges/ch_3KblRZLwWIuqLC5N04kBMUiT/refunds"
    },
    "review": null,
    "shipping": null,
    "source": null,
    "source_transfer": null,
    "statement_descriptor": null,
    "statement_descriptor_suffix": null,
    "status": "succeeded",
    "transfer_data": null,
    "transfer_group": null
  },
  [...]

```

```

        "has_more": false,
        "total_count": 1,
        "url":
"\v1\v1\charges?payment_intent=pi_3KblrZLwWIuqLC5N00EXLF62"
    },
    "client_secret":
"pi_3KblrZLwWIuqLC5N00EXLF62_secret_MgzF4wHHnAY2zWvTDq6FOk3nm",
    "confirmation_method": "automatic",
    "created": 1646918225,
    "currency": "eur",
    "customer": null,
    "description": null,
    "invoice": null,
    "last_payment_error": null,
    "livemode": false,
    "metadata": [],
    "next_action": null,
    "on_behalf_of": null,
    "payment_method": "pm_1KblrLwWIuqLC5NQJlVQXLv",
    "payment_method_options": {
        "card": {
            "installments": null,
            "network": null,
            "request_three_d_secure": "automatic"
        }
    },
    "payment_method_types": [
        "card"
    ],
    "processing": null,
    "receipt_email": null,
    "review": null,
    "setup_future_usage": null,
    "shipping": null,
    "source": null,
    "statement_descriptor": null,
    "statement_descriptor_suffix": null,
    "status": "succeeded",
    "transfer_data": null,
    "transfer_group": null
}

```