Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ
DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance	Ferguenis
Nom d'usage	Ferguenis
Prénom	Bérenger
Adresse	14 impasse Maurice Ravel 22440 Ploufragan

Titre professionnel visé

Développeur Web / Web Mobile

MODALITE D'ACCES :

☒ Parcours de formation
☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen**.

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

http://travail-emploi.gouv.fr/titres-professionnels

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

Activité-type 1

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

Exemple n°1 ▸ ***Projet Clipboard***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pendant notre formation chez Studi, nous avons eu l'occasion de réaliser des évaluations d'entraînement pour mettre en pratique nos connaissances.

Pour le projet Clipboard, l'évaluation consistait à créer la maquette d'une application logicielle offrant des services de téléchargement pour enregistrer et retrouver des textes copiés ultérieurement. J'ai choisi d'utiliser Figma, un outil de conception d'interfaces utilisateur, pour réaliser la maquette de ce projet. Figma offre des fonctionnalités avancées qui ont facilité la création de la maquette et m'ont permis de visualiser et d'organiser les différents éléments de l'application de manière efficace.

La maquette devait être conçue sous la forme d'une seule page.

2. Précisez les moyens utilisés :

Dans le but d'optimiser l'avancement du projet et de permettre au client de suivre son évolution, je décide de mettre en place un tableau Trello. J'y crée des cartes représentant les différentes tâches à accomplir. Cela facilitera la gestion et la visualisation des étapes à réaliser.

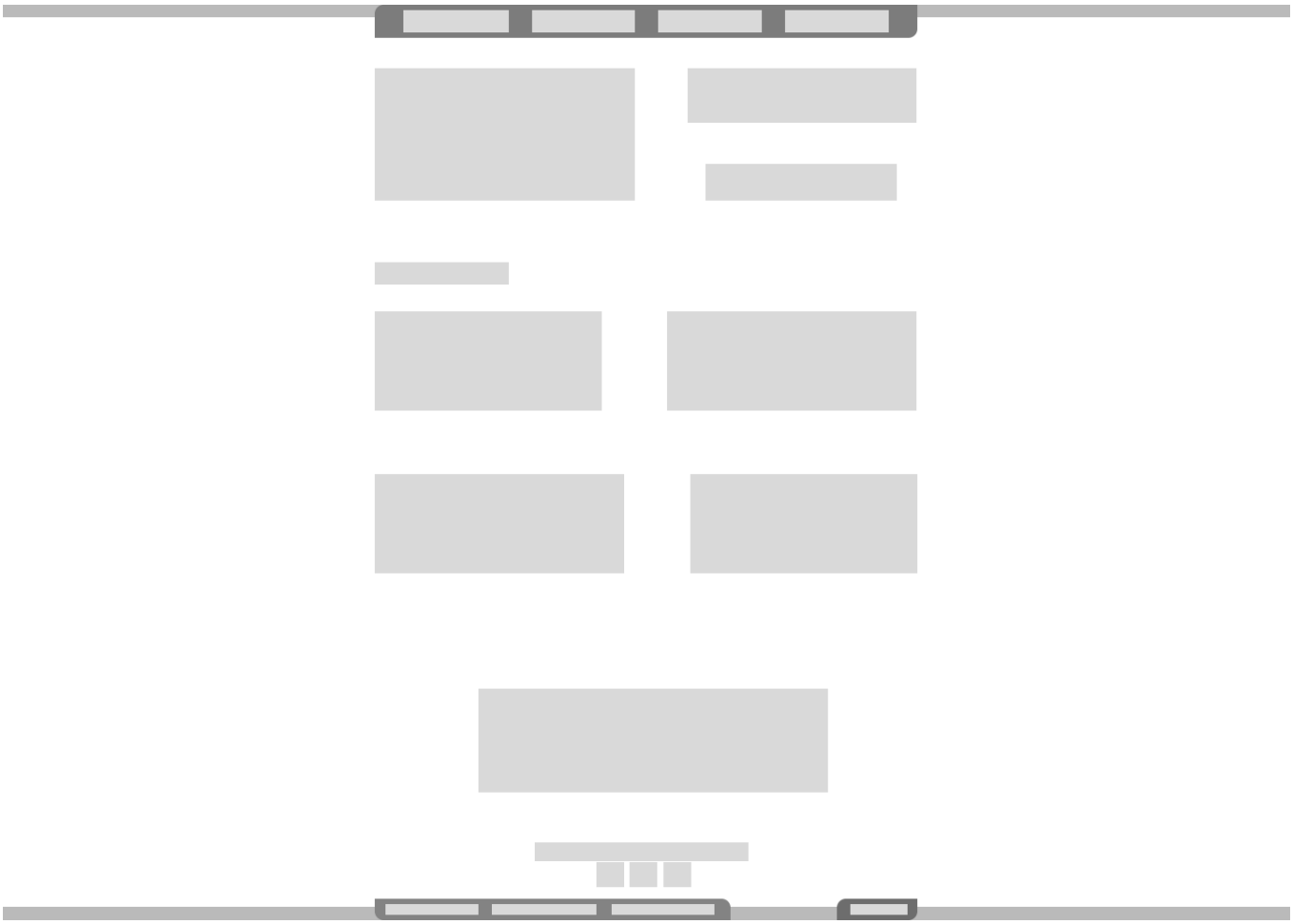
Afin de faciliter la gestion des images nécessaires pour ce projet, je prends la décision de créer un dossier dans Google Drive dédié à leur stockage. Cela permettra de centraliser et d'organiser les ressources visuelles nécessaires à l'avancement du projet.

En préparation du projet, je crée également un référentiel GitHub. Cela me permettra de versionner et de collaborer efficacement sur le code source du projet, tout en assurant une sauvegarde sécurisée du code.

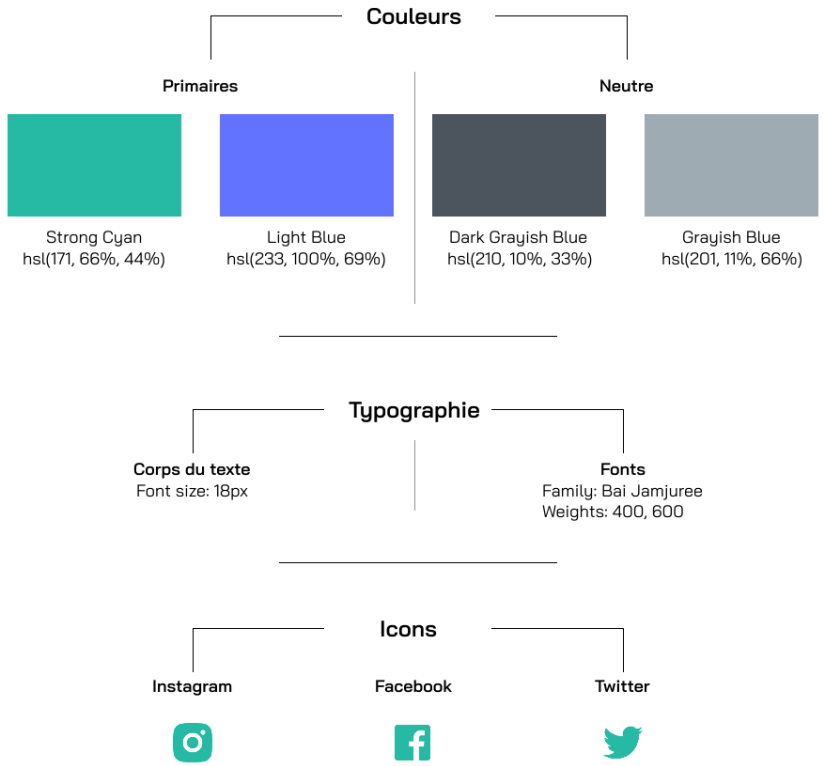
Dans l'objectif de m'inspirer et de prendre des références pour la conception de ma maquette, je visite plusieurs sites web qui proposent des fonctionnalités similaires. Cela m'a permis d'observer les bonnes pratiques, les tendances actuelles et d'obtenir des idées pour créer une maquette attrayante et conviviale.

Je débute en créant le wireframe sur Figma en utilisant des nuances de gris pour représenter les différents blocs. Ensuite, je commence à concevoir la disposition des éléments sur la page et à évaluer

sa faisabilité pour un langage HTML et CSS. Cela me permet de visualiser la structure de la page et de vérifier si elle est réalisable techniquement.



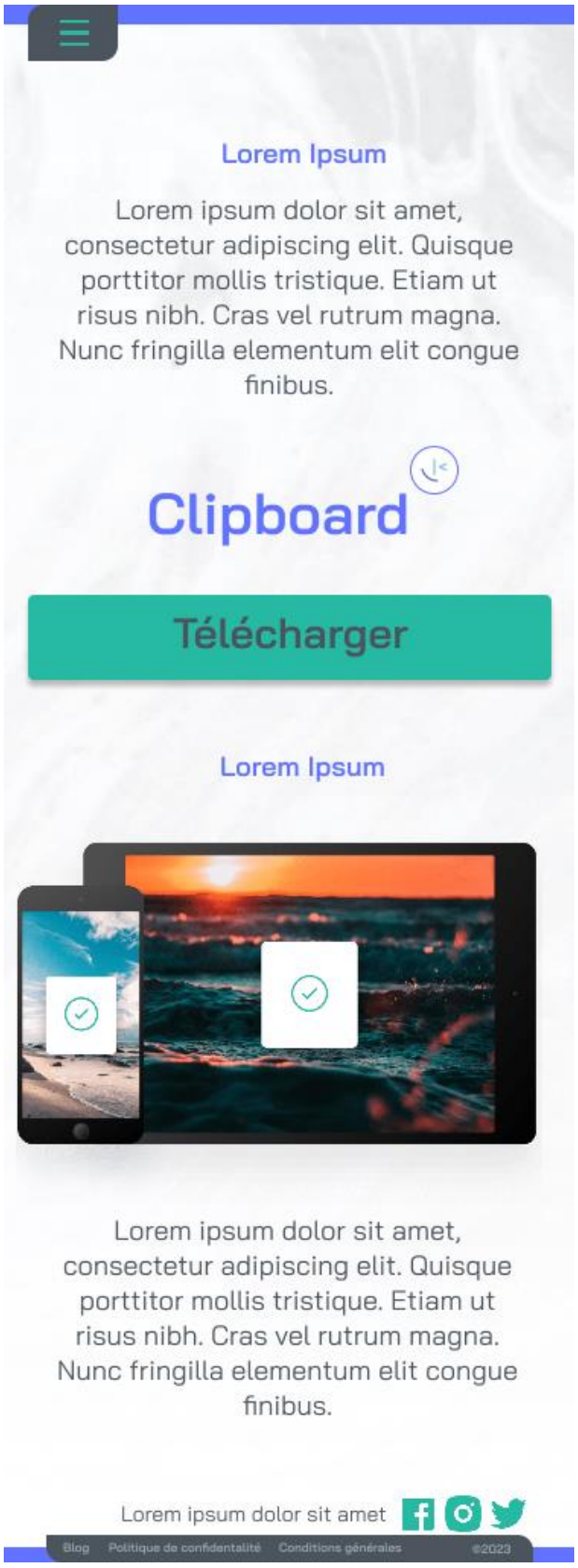
Ensuite, je passe à la création de la charte graphique. Une palette de couleurs a été fournie dans les consignes de l'évaluation d'entraînement. J'utilise cette palette de couleurs et je l'optimise en choisissant des polices d'écriture appropriées. Cela me permet d'établir une identité visuelle cohérente pour l'application.



Je décide d'associer les icônes des réseaux sociaux à la couleur dominante du projet. Cela permet de créer une harmonie visuelle en intégrant les icônes dans la palette de couleurs globale de l'application.

Après avoir finalisé le wireframe et défini la charte graphique, je procède à la création du mockup de l'interface. Je commence par concevoir le mockup pour le format desktop, en appliquant les couleurs, polices et éléments visuels spécifiés dans la charte graphique.
Une fois le mockup desktop validé, je passe à la création du mockup au format mobile, en adaptant la mise en page et les dimensions pour une expérience optimale sur les appareils mobiles.


Documentation
Caractéristiques
Version
Support
Forum
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque porttitor mollis tristique. Etiam ut risus nibh. Cras vel rutrum magna. Nunc fringilla elementum elit congue finibus.
Clipboard
Télécharger
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque porttitor mollis tristique. Etiam ut risus nibh. Cras vel rutrum magna. Nunc fringilla elementum elit congue finibus.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque porttitor mollis tristique. Etiam ut risus nibh. Cras vel rutrum magna. Nunc fringilla elementum elit congue finibus.
Google
Hewlett Packard Enterprise
IBM
Microsoft
VECTOR GRAPHICS
Lorem ipsum dolor sit amet
Blog
Politique de confidentalité
Conditions générales
© 2023


Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque porttitor mollis tristique. Etiam ut risus nibh. Cras vel rutrum magna. Nunc fringilla elementum elit congue finibus.
Clipboard
Télécharger
Lorem Ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque porttitor mollis tristique. Etiam ut risus nibh. Cras vel rutrum magna. Nunc fringilla elementum elit congue finibus.
Lorem ipsum dolor sit amet
Blog
Politique de confidentialité
Conditions générales
©2023

Dans le cadre du projet Clipboard, un fichier zip a été fourni dans l'énoncé pour ajouter les logos et les images nécessaires. Ce fichier zip contient les ressources graphiques telles que les logos des réseaux sociaux et les images spécifiques au projet. J'extrais ces fichiers du zip et les intégre dans le design de l'application selon les besoins et les spécifications du projet.

3. Avec qui avez-vous travaillé ?

J'ai réalisé ce projet en travaillant seul. Grâce à la plateforme Studi et aux cours sur la création de maquettes d'applications web responsive, j'ai pu réussir à accomplir ce projet. Les ressources fournies par Studi ainsi que les connaissances acquises lors des cours m'ont été d'une grande aide pour mener à bien ce projet et répondre aux exigences de réalisation d'une maquette d'application web responsive.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *01/05/2023* au : *02/05/2023*

5. Informations complémentaires *(facultatif)*

Activité-type 1

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

Exemple n°2 ▸ ***Projet association AFAV***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pendant ma formation chez Studi, j'ai eu l'occasion de réaliser des évaluations d'entraînement pour mettre en pratique mes connaissances.

Pour le projet AFAV (Association Française pour l'Archéologie du Verre), l'évaluation consistait à créer un site web en utilisant HTML, CSS et le framework Bootstrap pour une association de mon choix. J'ai décidé d'utiliser une association déjà existante et ayant déjà un site construit afin de proposer une autre version.

Pour ce projet j'ai utilisé le langage HTML, CSS, ainsi que la syntaxe SCSS du préprocesseur SASS. J'ai utilisé Figma pour le maquettage, npm(Node Package Manager) pour gérer les dépendances du projet, y compris le framework Bootstrap pour son système de classe et son responsive.

2. Précisez les moyens utilisés :

Pour débuter le projet, je commence par effectuer des recherches afin de m'inspirer et de trouver des références sur des sites d'associations. Ensuite, j'utilise l'outil Figma pour créer les maquettes du site. Je commence par définir la charte graphique, puis je crée un mockup pour représenter le design final. Je veille à créer des versions adaptées pour les écrans desktop et mobile.

Une fois les maquettes finalisées, je passe à la mise en place du projet en créant un dossier nommé « afav ». J'ouvre mon environnement de développement intégré (IDE), tel que Visual Studio Code, et j'ouvre un nouveau terminal. Je vérifie si nodejs est bien installé avec la commande :

node -v

À partir de là, j'installe les dépendances nécessaires en utilisant npm à l'aide de la commande :

npm install

Lors de l'installation, npm me demande de remplir plusieurs champs, tels que le nom du projet et sa description.
Dans la racine du projet, je retrouve deux fichiers, à savoir « package.json » et « package-lock.json », qui référencent toutes les dépendances du projet.

J'installe également les dépendances avec npm que je vais avoir besoin pour ce projet à savoir :

- npm i bootstrap
- npm i autoprefixer
- npm i get-google-fonts
- npm i postcss
- npm i postcss-cli
- npm i sass
- npm i remixicon

J'utilise l'autoprefixer pour automatiser la gestion des préfixes CSS nécessaires pour assurer une compatibilité optimale avec les différents navigateurs.

Quant au postcss et postcss-cli, ils me permettent de travailler avec des plugins CSS et d'effectuer des transformations sur mon code CSS, comme la compression, la minification ou encore l'ajout de variables.

J'utilise également Google Fonts pour accéder à une large sélection de polices d'écriture et pour les intégrer facilement dans mon projet.

J'utilise également SASS (Syntactically Awesome Style Sheets) dans mon projet. SASS est un préprocesseur CSS qui me permet d'écrire mon code CSS de manière plus efficace en utilisant des fonctionnalités telles que les variables, les mixins, les boucles, etc. Cela facilite la gestion du style et la réutilisation du code, en me permettant d'organiser et de structurer mes feuilles de style de manière plus modulaire. J'utilise généralement la syntaxe SCSS de SASS, qui est une extension de la syntaxe CSS traditionnelle, offrant une plus grande flexibilité et un meilleur contrôle sur mon style.

Enfin, j'utilise Remixicon pour l'insertion de logos et d'icônes dans mon site. Cela me permet d'avoir accès à une vaste collection d'icônes prêtes à être utilisées dans mes interfaces.

Dans mon fichier "package.json", je procède à une modification dans la section "scripts" afin de configurer SASS pour qu'il puisse compiler mon code SCSS en CSS et lui attribuer un chemin approprié.

Plus précisément, j'ajoute ou modifie le script "sass" de la manière suivante :

```
"scripts": {
  "sass": "sass sass/style.scss assets/css/style.css"
},
```

package.json

Cela indique à npm d'utiliser SASS pour compiler le fichier « style.scss » situé dans le dossier « sass » et de générer le fichier de sortie « style.css » dans le dossier « assets/css ». Ainsi, lorsque j'exécute ce script, SASS prendra en charge la compilation du SCSS en CSS, en respectant la structure des dossiers spécifiée.

Voici le contenu complet du fichier "package.json", qui englobe toutes les configurations et informations essentielles requises pour le projet :

package.json > ...

```json
{
  "name": "afav-bootstrap-sass",
  "version": "1.0.0",
  "description": "ecf entrainement STUDI 2022 - refonte site association AFAV avec Bootstrap et SASS",
  "main": "index.js",
  Debug
  "scripts": {
    "sass": "sass sass/style.scss assets/css/style.css"
  },
  "keywords": [
    "afav",
    "bootstrap-5",
    "bootstrap-sass",
    "sass"
  ],
  "author": "Berenger FERGUENIS",
  "license": "ISC",
  "dependencies": {
    "autoprefixer": "^10.4.13",
    "bootstrap": "^5.2.3",
    "get-google-fonts": "^1.2.2",
    "postcss": "^8.4.19",
    "postcss-cli": "^10.1.0",
    "remixicon": "^2.5.0"
  }
}
```

Pour effectuer la compilation du SCSS en CSS, j'utilise la commande suivante :

npm run sass

Je commence par créer un fichier initial que je nomme « index.html », dans lequel j'intègre la structure HTML de base.
Je crée un fichier « _custom.scss » à la racine du projet et j'inclus à l'intérieur de ce fichier les variables de Bootstrap récupérées depuis le fichier « node_modules » généré lors de l'installation de Bootstrap via npm.

Ensuite, je modifie les valeurs des couleurs des variables **$primary**, **$secondary**, **$light** et **$dark** de Bootstrap pour les adapter à la palette de couleurs de la charte graphique.
Pour importer les styles CSS de Bootstrap, j'ajoute également le chemin du dossier « node_modules » dans le fichier SCSS, afin de pouvoir accéder aux fichiers CSS de Bootstrap nécessaires.

sass > _custom.scss > ...

```scss
//IMPORTE COLORS
$orange:    #EB9150;
$green:     #88C4B4;

//IMPORT DARK & LIGHT COLORS
$white:     #EEEEEE;
$black:     #080404;

//PRIMARY & SECONDARY COLORS
$primary:       $orange;
$secondary:     $green;
$light:         $white;
$dark:          $black;

//IMPORT FONT-SIZE
$font-size-root:              null;
$font-size-base:              1rem; // Assumes the browser default, typically `16px`
$font-size-sm:                $font-size-base * .875;
$font-size-lg:                $font-size-base * 2.25;


$navbar-brand-font-size:            $font-size-lg;

//IMPORT FONT-FAMILY
$font-family-base: 'Lato', sans-serif;

//IMPORT FONT-TITLE-SIZE
$h1-font-size:                $font-size-base * 2.5;
$h2-font-size:                $font-size-base * 2;
$h3-font-size:                $font-size-base * 1.75;
$h4-font-size:                $font-size-base * 1.5;
$h5-font-size:                $font-size-base * 1.25;


//IMPORT BOOTSTRAP
@import"../node_modules/bootstrap/scss/bootstrap.scss";

```

Pour importer les polices d'écriture de Google Fonts, j'utilise la balise <link> et j'ajoute le chemin du fichier CSS requis dans l'attribut « href ». Ainsi, je m'assure que les polices de caractères souhaitées sont chargées à partir des serveurs de Google Fonts. En spécifiant le chemin correct dans l'attribut « href », je garantie que les styles de police sont appliqués correctement à travers mon contenu.

Au sein des balises <header></header>, je récupère un modèle de barre de navigation (navbar) depuis le site Bootstrap. Ensuite, je procède à des modifications au niveau des balises <img> ainsi que de certaines classes Bootstrap afin de les harmoniser avec la maquette du site.

```
<!-- START NAVIGATION BAR -->
<nav class="navbar navbar-dark bg-dark">
  <div class="container">
    <a class="navbar-brand text-secondary header_nav-navbar-brand" href="#">afav
      <img src="./img/Logo_vase.svg" alt="Logo AFAV">
    </a>
    <button class="navbar-toggler" type="button" data-bs-toggle="offcanvas" data-bs-target="#offcanvasDarkNavbar" aria-controls="offcanvasDarkNavbar">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="offcanvas offcanvas-end text-bg-dark" tabindex="-1" id="offcanvasDarkNavbar" aria-labelledby="offcanvasDarkNavbarLabel">
      <div class="offcanvas-header">
        <h5 class="offcanvas-title text-secondary header_nav-navbar-brand" id="offcanvasDarkNavbarLabel"><a href="./index.html" class="text-secondary header_navbar-brand-link">afav</a></h5>
        <button type="button" class="btn-close btn-close-white" data-bs-dismiss="offcanvas" aria-label="Close"></button>
      </div>
      <div class="offcanvas-body">
        <ul class="navbar-nav justify-content-end flex-grow-1 pe-3 text-end">
          <li class="nav-item">
            <a class="nav-link active text-secondary" aria-current="page" href="./index.html">L'AFAV</a>
          </li>
          <li class="nav-item">
            <a class="nav-link active text-secondary" aria-current="page" href="./activite.html">Activité de l' AFAV</a>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle active text-secondary" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              Actualités
            </a>
            <ul class="dropdown-menu dropdown-menu-dark text-end">
              <li><a class="dropdown-item text-secondary" href="#">Actualités verrières</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Publications</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Archives des actualités</a></li>
              <li>
              </ul>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle active text-secondary" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              Rencontres AFAV
            </a>
            <ul class="dropdown-menu dropdown-menu-dark text-end">
              <li><a class="dropdown-item text-secondary" href="#">Archives des rencontres</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Poster des rencontres</a></li>
              <li>
              </ul>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle active text-secondary" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
              Publications
            </a>
            <ul class="dropdown-menu dropdown-menu-dark text-end">
              <li><a class="dropdown-item text-secondary" href="#">Bulletins AFAV</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Catalogues</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Actes et corpus</a></li>
              <li><a class="dropdown-item text-secondary" href="#">Publications en ligne</a></li>
              <li>
              </ul>
          </li>
        <li class="nav-item">
          <a class="nav-link active text-secondary" aria-current="page" href="#">Les gestes qui sauvent</a>
          <a class="nav-link active text-secondary" aria-current="page" href="#">Projet Veinar</a>
          <a class="nav-link active text-secondary" aria-current="page" href="#">Acteurs-Actrices de la recherche</a>
          <a class="nav-link active text-secondary" aria-current="page" href="#">Contacts</a>
          <a class="nav-link active text-secondary" aria-current="page" href="#">Liens utiles</a>
        </li>
      </ul>
      <form class="d-flex mt-3" role="search">
        <input class="form-control me-2" type="search" placeholder="Recherche..." aria-label="Search">
        <button class="btn btn-primary" type="submit">Recherche</button>
      </form>
    </div>
  </div>
</nav>
<!-- END NAVIGATION BAR -->
```

Pour respecter les bonnes pratiques de SASS, je crée un dossier nommé « sass » dans lequel j'inclus un sous-dossier « sections » contenant les fichiers SCSS correspondant aux différentes pages du site. Ensuite, à la racine de mon projet, je crée un fichier « style.scss » où j'importe les sections nécessaires à l'aide de l'instruction « @import ». De plus, j'utilise le fichier « _custom.scss » en l'important également avec l'instruction « @use », afin d'utiliser les modifications personnalisées que j'ai apportées aux variables de Bootstrap.

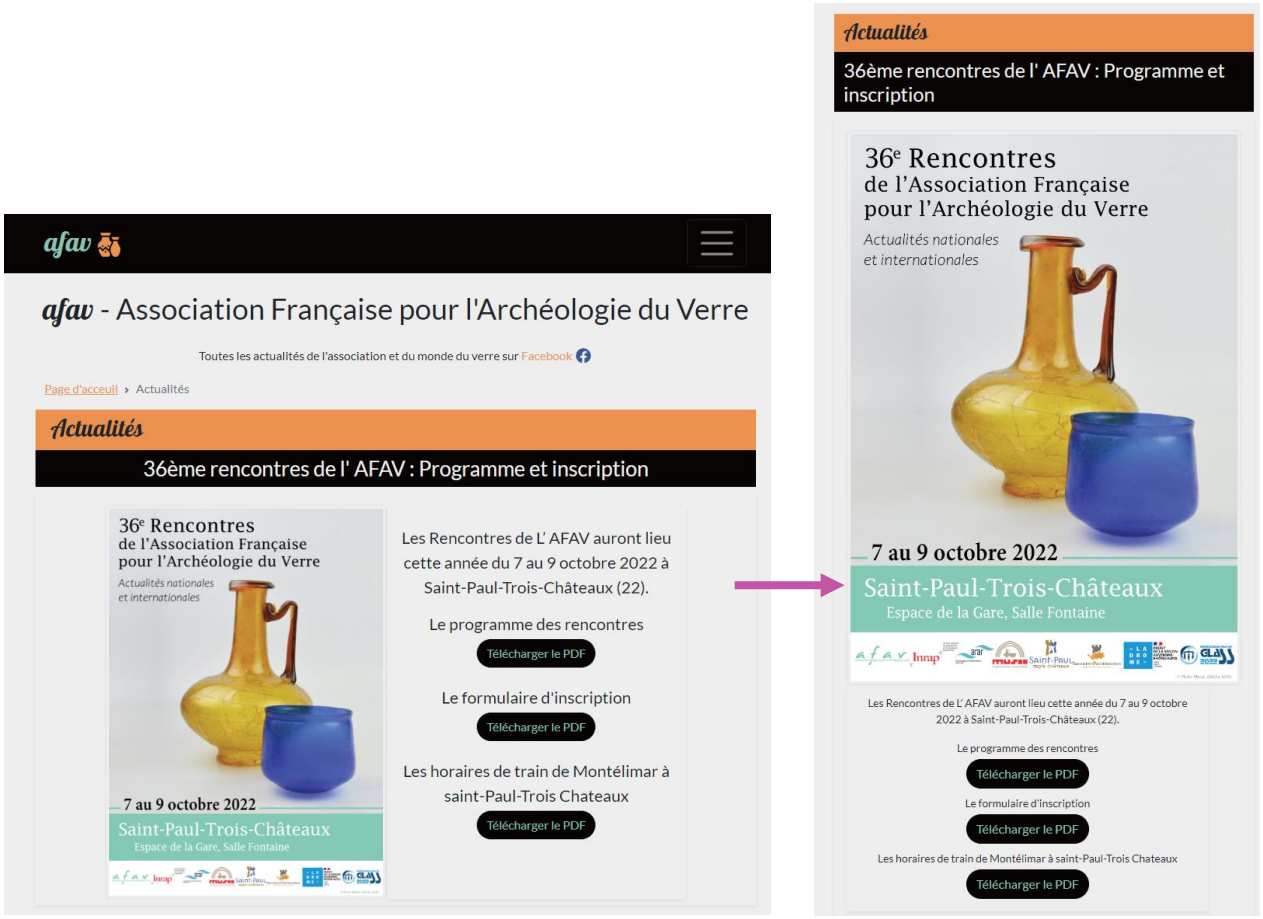
Dans la page d'accueil du site, je positionne un bouton permettant d'accéder à la section des dernières actualités de l’association.

Ensuite, je crée un nouveau fichier appelé « event.html » où j'utilise les classes Bootstrap « row » et « col » dans un conteneur pour placer une image et son texte côte à côte.

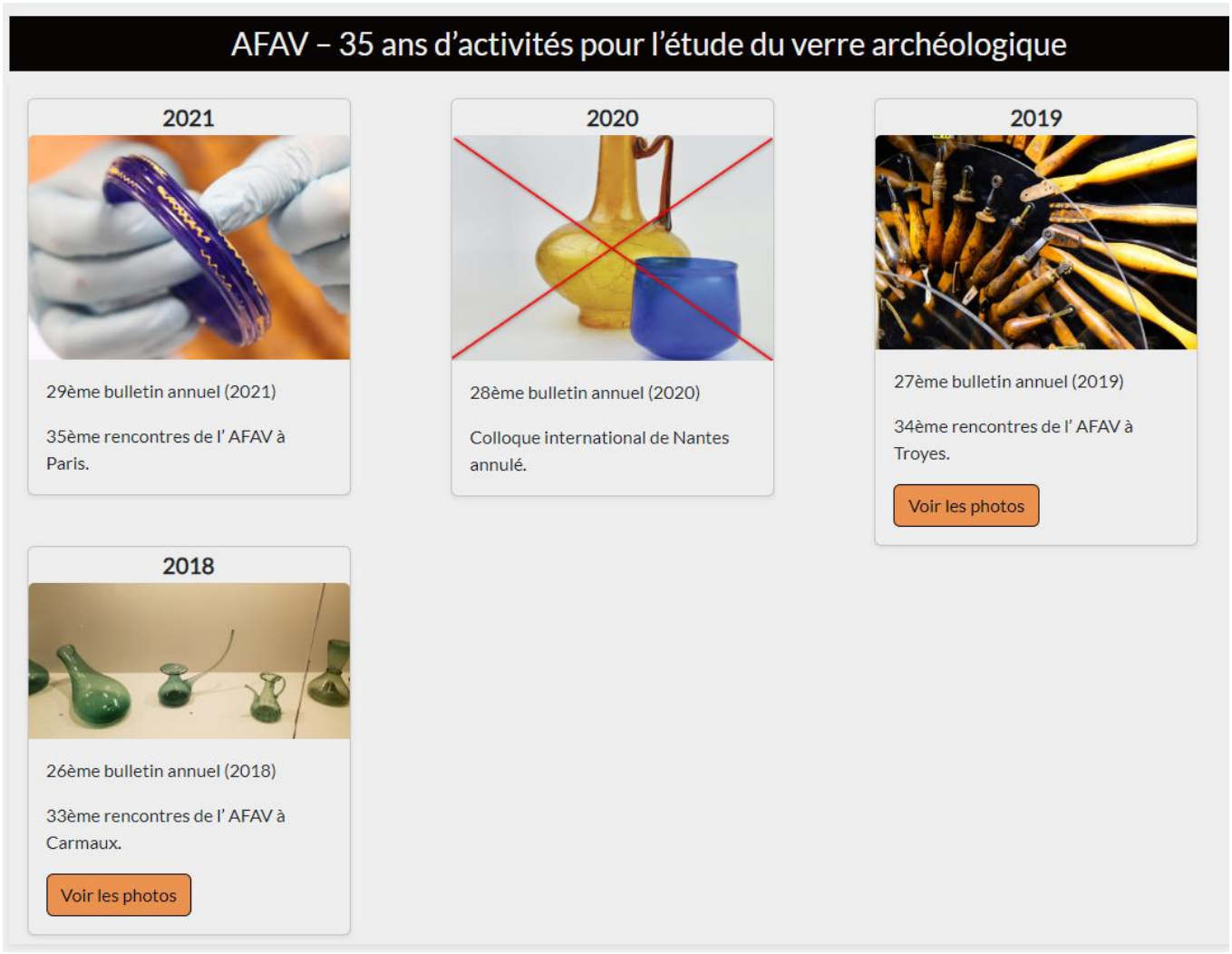
```
<div class="container">
  <div class="row">
    <div class="col-sm-1 col-md-1"></div>
    <div class="col-sm-12 col-md g-0">
      <a href="http://afaverre.fr/Afaverre/wp-content/uploads/2022/02/affiche-afav-20221-scaled.jpg" target="_blank">
        <img src="./img/affiche-afav-20221-scaled.jpg" class="img-fluid img-thumbnail" alt="Affiche 36ème rencontres de l'AFAV">
      </a>
    </div>
    <div class="col bg-white pt-1 shadow-sm pt-2 pb-2 mt-1 text-center">
      <p class="main_actualites-paragraphe">
        Les Rencontres de L' AFAV auront lieu cette année du 7 au 9 octobre 2022 à Saint-Paul-Trois-Châteaux (22).</p>
      <p class="main_actualites-paragraphe mb-1">Le programme des rencontres</p>
      <button type="button" class="btn btn-dark rounded-pill mb-2">
        <a href="http://afaverre.fr/Afaverre/wp-content/uploads/2022/08/2-Programme-36e-rencontres-AFAV.pdf" target="_blank"
        class="main_actualites-btn text-secondary">Télécharger le PDF</a></button>
      <p class="main_actualites-paragraphe mb-1">Le formulaire d'inscription</p>
      <button type="button" class="btn btn-dark rounded-pill mb-2">
        <a href="http://afaverre.fr/Afaverre/wp-content/uploads/2022/07/1-formulaire-Inscription_2022-1.pdf" target="_blank"
        class="main_actualites-btn text-secondary">Télécharger le PDF</a></button>
      <p class="main_actualites-paragraphe mb-1">Les horaires de train de Montélimar à saint-Paul-Trois Chateaux</p>
      <button type="button" class="btn btn-dark rounded-pill mb-2">
        <a href="http://afaverre.fr/Afaverre/wp-content/uploads/2022/07/3-DROME-Fiche-horaires-Ete-2021-L42-Mont%C3%A9limar-St-paul-Trois-Ch%C3%A2teaux
        target="_blank" class="main_actualites-btn text-secondary">Télécharger le PDF</a></button>
    </div>
    <div class="col-sm-1"></div>
  </div>
</div>
```

event.html

Lorsque l'écran de l'utilisateur est en mode mobile, le texte se déplace en dessous de l'image grâce à la classe Bootstrap « col-sm-12 ».



Pour les dernières activités de l'association, j'utilise les « cards » de Bootstrap afin de présenter les activités passées. À l'intérieur de certaines « cards », j'insère un bouton permettant d'accéder aux photos associées.



J'utilise des mixins dans mon code SASS pour faciliter la réutilisation de certaines fonctionnalités dans différentes sections. J'importe ces mixins en utilisant la directive « @use [chemin du fichier] as [alias]" et ensuite j'inclus le mixin spécifique dans ma classe CSS en utilisant « @include [alias].[nom du mixin] ». Cela me permet d'intégrer le mixin dans la classe CSS correspondante et de bénéficier de ses fonctionnalités.

```scss
//etude-archeo
@mixin sources {
    font-size: 0.75rem;
    margin-top: 0.25rem;
    font-style: italic;
}
```

```scss
@use'../custom';
@use'../components/mixins' as m;

.main_etude-archeo-photographer {
    @include m.sources;
}
```

J'utilise également des media queries lorsque je souhaite appliquer des styles CSS spécifiques en fonction de la taille de l'écran ou du dispositif utilisé par l'utilisateur. Les media queries me permettent de définir des règles CSS conditionnelles qui s'appliquent uniquement lorsque certaines conditions sont

remplies, telles que la largeur de l'écran, l'orientation ou d'autres caractéristiques du média. Cela me permet de rendre mon site web ou mon application réactif et de fournir une expérience utilisateur optimale sur différents appareils et résolutions d'écran.

```
@media (max-width: 576px) {
    .main_breadcrumb-queries {
        font-size: 0.688rem; //11px
    }
}
```

3. Avec qui avez-vous travaillé ?

J'ai réalisé ce projet d'évaluation d'entraînement en travaillant seul. Grâce à la plateforme Studi et aux différents cours et session de directe HTML, CSS, SASS et Bootstrap, j'ai pu réussir à accomplir ce projet. Les ressources fournies par Studi ainsi que les connaissances acquises lors des cours et sessions de directe m'ont été d'une grande aide pour mener à bien ce projet et répondre aux exigences pour créer un site web HTML, CSS et le framework Bootstrap d'une association.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *25/11/2022* au : *02/12/2022*

5. Informations complémentaires *(facultatif)*

Activité-type 1

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

Exemple n°3 ▸ *Projet - Mon Dictionnaire Dynamique*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Lors de notre formation, nous avons étudié le langage Javascript. Pour mettre en pratique nos connaissances, une formatrice expérimentée nous a guidés à travers plusieurs sessions en direct pour la création d'une application qui offre la possibilité d'obtenir la définition d'un mot en anglais, accompagnée d'une fonctionnalité de lecture audio intégrée appelé « Mon dictionnaire Dynamique ».

Ce projet tire parti de l'API « dictionaryapi » et utilise la méthode **fetch()**.

Pendant ces sessions, elle nous a accompagnés tout au long des étapes cruciales pour construire cette application selon les meilleures pratiques et les principes fondamentaux du langage Javascript.

2. Précisez les moyens utilisés :

Une fois la structure HTML et CSS mise en place, je crée un fichier nommé « script.js » à la racine du projet.

Dans ce fichier, je définis une fonction appelée **watchSubmit()**. À l'intérieur de cette fonction, je récupère un élément de formulaire spécifié par l'ID « form » en utilisant la méthode **querySelector()**. Ensuite, j'ajoute un écouteur d'événement pour la soumission du formulaire, où j'empêche le comportement par défaut d'actualisation de la page avec **event.preventDefault()**. Ensuite, je crée un objet **FormData()** à partir du formulaire, puis j'extrais la valeur du champ de recherche avec le nom « search ».

```
1   /* Étape #1 : Récupération d'un mot */
2   const watchSubmit = () => {
3       const form = document.querySelector("#form")
4       form.addEventListener("submit", (event) => {
5           event.preventDefault()
6           const data = new FormData(form)
7           const wordToSearch = data.get("search")
8           apiCall(wordToSearch)
9       })
10  }
```

Pour la deuxième étape, j'écris la fonction **apiCall()**. Cette fonction envoie un mot à une API en utilisant la méthode **fetch()**. Je récupère les données renvoyées par l'API, puis j'appelle la fonction **extractData()**

pour extraire les informations nécessaires. Ensuite, j'appelle la fonction **renderToHTML()** pour afficher les informations dans le HTML. Si une erreur se produit ou si le mot n'existe pas, j'affiche un message d'alerte et j'affiche l'erreur dans la console. J'appelle la fonction **apiCall()** en passant la valeur du champ de recherche en tant qu'argument.

```
12  /* Étape #2 : Envoyer un mot à l'API */
13  const apiCall = (word) => {
14      fetch(`https://api.dictionaryapi.dev/api/v2/entries/en/${word}`)
15          .then((response) => response.json())
16          .then((data) => {
17              /*  Étape #3 : Récupérer la donnée */
18              const informationsNeeded = extractData(data[0])
19              renderToHTML(informationsNeeded)
20          })
21          .catch((error) => {
22              alert('Le mot demandé n\'existe pas')
23              console.error(error)
24          })
25  }
```

La fonction **extractData()** est responsable d'extraire les informations nécessaires à partir des données passées en paramètre. Je récupère le mot, l'écriture phonétique, la prononciation (audio) et les définitions. Ensuite, je renvoie un objet contenant ces informations.

```
27  const extractData = (data) => {
28      // 1 - Mot
29      const word = data.word
30      // 2 - Écriture phonétique
31      const phonetic = findProp(data.phonetics, "text")
32      // 3 - Prononciation (audio)
33      const pronoun = findProp(data.phonetics, "audio")
34      // 4 - Définition(s)
35      const meanings = data.meanings
36
37      return {
38          word: word,
39          phonetic: phonetic,
40          pronoun: pronoun,
41          meanings: meanings
42      }
43  }
```

J'utilise également la fonction **findProp()** qui est utilisée par **extractData()** pour rechercher une propriété spécifique dans un tableau d'objets. J'itère à travers le tableau et je renvoie la valeur de la propriété si elle est trouvée.

```
44  const findProp = (array, name) => {
45      // Elle parcours un tableau d'objets
46      for (let i = 0; i < array.length; i++) {
47          // Et cherche dans ce tableau, si l'objet en cours contient une certaines propriété
48          const currentObject = array[i]
49          const hasProp = currentObject.hasOwnProperty(name)
50          // Alors elle renvoit cette propriété
51          if (hasProp) return currentObject[name]
52      }
53  }
```

La fonction **renderToHTML()** de l'étape 4 est responsable de l'affichage des informations du mot sur la page. J'utilise la propriété **textContent** pour modifier les textes affichés dans les éléments HTML. Je crée des éléments HTML dynamiques pour afficher les différentes significations du mot. J'affiche le titre, l'écriture phonétique et les significations dans le HTML.

```
55  /* ÉTAPE 4 : Afficher les informations de mon mot sur ma page */
56  const renderToHTML = (data) => {
57      const card = document.querySelector('.js-card')
58      card.classList.remove('card--hidden')
59
60      // Manipulation de textes avec la propriété textContent
61      const title = document.querySelector(".js-card-title")
62      title.textContent = data.word
63      const phonetic = document.querySelector(".js-card-phonetic")
64      phonetic.textContent = data.phonetic
```

J'ajoute également la gestion de l'audio associé au mot. Lorsque le bouton est cliqué, j'utilise l'objet « Audio » pour jouer l'audio correspondant. J'ajoute des classes au bouton pour refléter son état. Lorsque l'audio se termine, je mets à jour les classes du bouton.

```
66      // Création d'élements HTML dynamiques
67      const list = document.querySelector('.js-card-list')
68      list.innerHTML = ""
69      for(let i = 0; i < data.meanings.length; i++) {
70          const meaning = data.meanings[i]
71          const partOfSpeech = meaning.partOfSpeech
72          const definition = meaning.definitions[0].definition
```

```
74      // La création d'élements
75          const li = document.createElement('li')
76          li.classList.add('card__meaning')
77          const pPartOfSpeech = document.createElement('p')
78          pPartOfSpeech.textContent = partOfSpeech
79          pPartOfSpeech.classList.add('card__part-of-speech')
80          const pDefinition = document.createElement('p')
81          pDefinition.textContent = definition
82          pDefinition.classList.add('card__definition')
83
84          li.appendChild(pPartOfSpeech)
85          li.appendChild(pDefinition)
86          list.appendChild(li)
87      }
```

```
89      // Ajout de l'audio en JS
90      const button = document.querySelector('.js-card-button')
91      const audio = new Audio(data.pronoun)
92      button.addEventListener('click', () => {
93          button.classList.remove("card__player--off")
94          button.classList.add("card__player--on")
95          audio.play()
96      })
97      audio.addEventListener('ended', () => {
98          button.classList.remove("card__player--on")
99          button.classList.add("card__player--off")
100     })
101 }
```

Enfin, la dernière ligne **watchSubmit** lance le programme en appelant la fonction **watchSubmit()**.

```
104 // LANCEMENT DU PROGRAMME
105 watchSubmit()
```

3. Avec qui avez-vous travaillé ?

J'ai travaillé seul, mais j'ai pu compter sur le soutien de la formatrice tout au long des sessions en direct pour m'aider à mener à bien ce projet.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *22/02/2023* au : *24/02/2023*

5. Informations complémentaires *(facultatif)*

Activité-type 2

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

Exemple n°1 ▸ ***Créer et administrer une base de données - Projet Cinéma***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pendant notre formation chez Studi, nous avons eu l'opportunité de mettre en pratique nos connaissances à travers des évaluations d'entraînement. L'un de ces projets, intitulé « Cinema », consistait à créer et administrer une base de données pour un logiciel de réservation de places de cinéma.

Pour ce projet, j'ai choisi d'utiliser le langage SQL. Mes tâches comprenaient la conception physique des données, l'implémentation des instructions de création et de modification de la base de données et de ses tables en SQL, la rédaction des instructions pour mettre en place les contraintes, la création d'un script pour l'alimentation factice de données dans la base, la maîtrise des techniques de gestion de la sécurité d'une base de données, ainsi que l'utilisation d'un utilitaire de sauvegarde et de restauration.

2. Précisez les moyens utilisés :

Pour démarrer le projet d'évaluation d'entraînement - Créer et administrer une base de données, je commence par créer un dossier dédié au projet sur mon ordinateur. Ensuite, j'ouvre mon terminal à partir de ce dossier spécifique.

Une fois dans le terminal, j'initialise un dépôt Git en utilisant la commande appropriée. Cela me permettra de garder une trace des modifications apportées au code et de faciliter la collaboration si nécessaire.

Ensuite, je me place dans le dossier du projet en utilisant le terminal et j'ouvre mon environnement de développement intégré (IDE), dans ce cas précis, Visual Studio Code.

Une fois dans l'IDE, je crée un fichier que je nomme « cinema.sql ». Ce fichier sera utilisé pour écrire les scripts SQL relatifs à la création et à l'alimentation de la base de données.

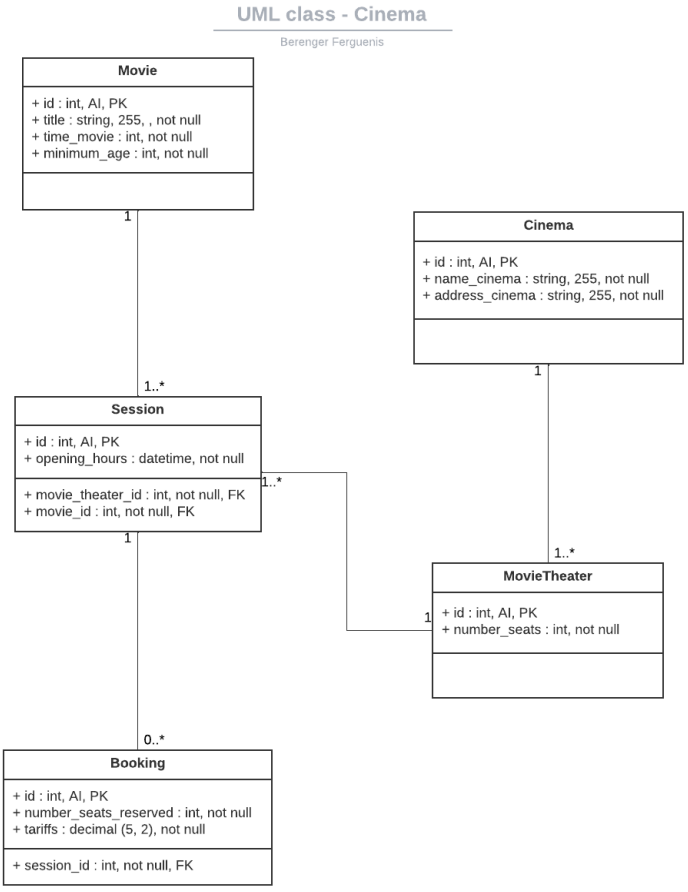
En parallèle, je crée également un fichier "readme.md" qui servira à documenter le projet, en fournissant des instructions, des explications et toute autre information pertinente.

Ces étapes préliminaires permettent de mettre en place l'environnement de travail et les fichiers nécessaires pour commencer à concevoir et à administrer la base de données du projet.

Pour héberger mon code source sur GitHub, je me rends sur le site web de GitHub et crée un référentiel public. Ensuite, j'établis une connexion entre mon dépôt local et mon référentiel distant sur GitHub.

J'utilise ensuite la ligne de commande dans le terminal pour créer mon premier commit et le pousser vers mon référentiel distant.

Afin d'élaborer le diagramme de classe et ses relations entre les différentes tables, j'utilise le site web de Lucidchart. Je prends en compte les consignes fournies pour créer l'UML en respectant les spécifications demandées.



Étant déjà en possession d'un compte MySQL enregistré sur Alwaysdata, j'accède à celui-ci depuis mon terminal en utilisant la commande suivante :

mysql -u [mon nom d'utilisateur] -p[mon mot de passe MySQL] -h [nom du serveur] ;

Cela me permet d'établir une connexion à mon serveur MySQL et d'interagir avec ma base de données à travers le terminal en ligne de commande.

Par la suite, pour vérifier mes privilèges, j'utilise la commande « SHOW GRANTS ». Cependant, après l'exécution de cette commande, je remarque que je ne suis pas doté du privilège « CREATE » qui me permettrait de créer une base de données directement à partir de la ligne de commande.

```
mysql> SHOW GRANTS;
+-----------------------------------------------------------------------------------------------------------------+
| Grants for berenger@%                                                                                           |
+-----------------------------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `berenger`@`%` IDENTIFIED BY PASSWORD '*7509950EB30B8D726D894E5E9D066D2562DF319F' |
| GRANT ALL PRIVILEGES ON `berenger_cuisinea_studi_live`.* TO `berenger`@`%` WITH GRANT OPTION                    |
+-----------------------------------------------------------------------------------------------------------------+
```

Par conséquent, je me rends sur la plateforme web d'Alwaysdata pour créer la base de données.
Je vérifie depuis le terminal si la base de données est bien présente.

```
mysql> show databases;
+-----------------------------+
| Database                    |
+-----------------------------+
| berenger_cinema_db          |
| berenger_cuisinea_studi_live |
+-----------------------------+
```

Je crée un nouveau dossier nommé "schemas" à la racine du projet. Ensuite, je crée le fichier "cinema.sql" à l'intérieur de ce dossier. Dans ce fichier, je crée les tables pour chaque entité en respectant le plan de l'UML pour le nom des propriétés, les types et les contraintes.

Lorsque j'utilise l' IDE (Environnement de Développement Intégré) pour écrire des requêtes SQL, j'ai la possibilité de bénéficier d'un surlignage syntaxique, ce qui facilite la lecture et la compréhension du code. De plus, l'IDE peut également détecter les erreurs syntaxiques potentielles dans mes requêtes, ce qui me permet de les corriger avant de les exécuter.

```sql
/* Création des tables */
CREATE TABLE Cinema
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    nameCinema VARCHAR(255) NOT NULL,
    addressCinema VARCHAR(255) NOT NULL
);
```

Pour exemple, dans la table « Cinema ». Cette table a trois colonnes : "id", « nameCinema » et « addressCinema ». La colonne « id » est de type entier (INT) et elle est définie comme clé primaire (PRIMARY KEY) avec la fonctionnalité d'auto-incrémentation (AUTO_INCREMENT), ce qui signifie qu'elle sera automatiquement augmentée à chaque nouvel enregistrement. Les colonnes « nameCinema » et

« addressCinema » sont de type chaîne de caractères (VARCHAR) avec une longueur maximale de 255 caractères. Ces deux colonnes ne peuvent pas être vides (NOT NULL), ce qui signifie qu'il est obligatoire de spécifier une valeur pour chacune d'entre elles lors de l'insertion de données dans la table.

Dans le code suivant j'établis une contrainte de clé étrangère (FOREIGN KEY) sur la colonne « cinema_id ». Cette contrainte établit une relation entre la colonne « cinema_id » de la table « MovieTheater » et la colonne « id » de la table « Cinema ». Cela signifie que la valeur de « cinema_id » dans la table « MovieTheater » doit correspondre à une valeur existante dans la colonne « id » de la table « Cinema ». Cette relation garantit l'intégrité référentielle entre les deux tables.

```
CREATE TABLE MovieTheater
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  numberSeats INT NOT NULL,
  cinema_id INT NOT NULL,
  FOREIGN KEY (cinema_id) REFERENCES Cinema(id)
);
```

Je procède à l'insertion de données fictives dans plusieurs tables. Pour la table « Cinema », j'insère trois enregistrements avec différents noms et adresses de cinéma. Pour la table « MovieTheater », j'ajoute des données relatives au nombre de sièges et à l'identifiant du cinéma associé. Dans la table « Movie », j'insère des informations sur trois films, tels que le titre, la durée et l'âge minimum requis. Pour la table « Session », je spécifie les horaires d'ouverture, l'identifiant de la salle de cinéma et l'identifiant du film. Enfin, dans la table « Booking », j'ajoute des enregistrements représentant les réservations de places, avec le nombre de sièges réservés, les tarifs appliqués et l'identifiant de la séance correspondante.

```sql
/* Insertions de données factices */

/* Pour la table Cinema */
INSERT INTO Cinema (nameCinema, addressCinema) VALUES ('Cinema A', '123 Rue du Cinéma');
INSERT INTO Cinema (nameCinema, addressCinema) VALUES ('Cinema B', '456 Avenue des Films');
INSERT INTO Cinema (nameCinema, addressCinema) VALUES ('Cinema C', '789 Boulevard du Spectacle');

/* Pour la table MovieTheater */
INSERT INTO MovieTheater (numberSeats, cinema_id) VALUES (100, 1);
INSERT INTO MovieTheater (numberSeats, cinema_id) VALUES (80, 2);
INSERT INTO MovieTheater (numberSeats, cinema_id) VALUES (120, 3);

/* Pour la table Movie */
INSERT INTO Movie (title, timeMovie, minimumAge) VALUES ('Movie 1', 120, 12);
INSERT INTO Movie (title, timeMovie, minimumAge) VALUES ('Movie 2', 90, 12);
INSERT INTO Movie (title, timeMovie, minimumAge) VALUES ('Movie 3', 105, 16);

/* Pour la table Session */
INSERT INTO Session (openingHours, movie_theater_id, movie_id) VALUES ('2023-04-27 10:00:00', 1, 1);
INSERT INTO Session (openingHours, movie_theater_id, movie_id) VALUES ('2023-04-27 15:30:00', 2, 2);
INSERT INTO Session (openingHours, movie_theater_id, movie_id) VALUES ('2023-04-27 20:00:00', 3, 3);

/* Pour la table Booking */
INSERT INTO Booking (numberSeatsReserved, tariffs, session_id) VALUES (2, 15.99, 1);
INSERT INTO Booking (numberSeatsReserved, tariffs, session_id) VALUES (3, 12.50, 2);
INSERT INTO Booking (numberSeatsReserved, tariffs, session_id) VALUES (4, 9.99, 3);
```

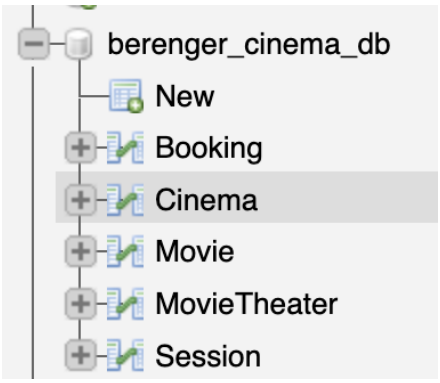
Ensuite, je procède à la sauvegarde de ma base de données en utilisant la commande « mysqldump ». J'indique mon nom d'utilisateur et mon mot de passe. Je spécifie également le nom de ma base de données. puis, j'utilise le symbole « > » pour rediriger la sortie de la commande vers un fichier nommé « backup.sql ». Cela signifie que toutes les informations de la base de données seront enregistrées dans ce fichier en tant que sauvegarde.

backup.sql

```
1  /* Sauvegarde de la base de données */
2  mysqldump -u berenger -p[redacted] Cinema_db > backup.sql
```

Par mesure de sécurité, j'ajoute le fichier « backup.sql » au fichier « .gitignore ». Cela permet de l'exclure de la gestion de version (versioning) et d'éviter de transmettre mes informations liées à mon compte MySQL.

Pour ajouter les tables et les données dans l'interface PHPMyAdmin, je procède à l'importation du fichier « cinema.sql ». Une fois l'importation terminée, j'observe que toutes les tables, les propriétés et les données présentes dans le fichier sont bien affichées dans l'interface. Cela me permet de vérifier que les tables et les données sont correctement ajoutées à ma base de données.



	id	nameCinema	addressCinema
Edit Copy Delete	1	Cinema A	123 Rue du Cinéma
Edit Copy Delete	2	Cinema B	456 Avenue des Films
Edit Copy Delete	3	Cinema C	789 Boulevard du Spectacle

	id	title	timeMovie	minimumAge
Edit Copy Delete	1	Movie 1	120	12
Edit Copy Delete	2	Movie 2	90	12
Edit Copy Delete	3	Movie 3	105	16

Je suis également en mesure d'effectuer des vérifications à l'aide de commandes MySQL, telles que :

mysql> SHOW DATABASES ;
mysql> connect [nom de ma base de données];
mysql> SHOW TABLES;

```
mysql> SHOW TABLES;
+-------------------------------+
| Tables_in_berenger_cinema_db |
+-------------------------------+
| Booking                       |
| Cinema                        |
| Movie                         |
| MovieTheater                  |
| Session                       |
+-------------------------------+
```

Je peux également consulter chaque table pour voir les données.

*mysql> SELECT * FROM [table];*

```
mysql> SELECT * FROM Movie;
+----+---------+-----------+------------+
| id | title   | timeMovie | minimumAge |
+----+---------+-----------+------------+
|  1 | Movie 1 |       120 |         12 |
|  2 | Movie 2 |        90 |         12 |
|  3 | Movie 3 |       105 |         16 |
+----+---------+-----------+------------+
```

3. Avec qui avez-vous travaillé ?

J'ai réalisé ce projet en travaillant seul. Grâce à la plateforme Studi et aux cours back-end, créer et administrer une base de données, j'ai pu réussir à accomplir ce projet. Les ressources fournies par Studi ainsi que les connaissances acquises lors des cours m'ont été d'une grande aide pour mener à bien ce projet et répondre aux exigences de la création et l'administration d'un base de données.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *27/04/2023* au : *28/04/2023*

5. Informations complémentaires *(facultatif)*

Activité-type 2

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

Exemple n° 2 ▸ ***Projet - Cuisinea - PHP***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Lors de notre formation, nous avons étudié le langage PHP.

Afin de mettre en pratique nos connaissances, un formateur expérimenté nous a guidés à travers plusieurs sessions en direct pour la création d'un site de recettes de cuisine appelé « Cuisinea ».

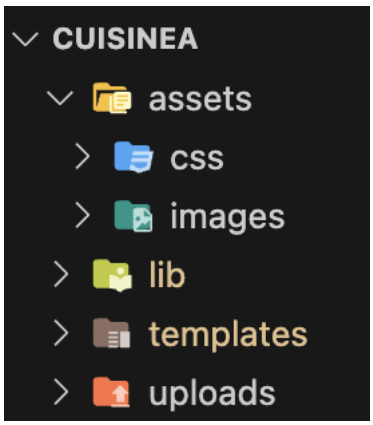
Pendant ces sessions, il nous a accompagné tout au long des étapes cruciales pour construire une application selon les meilleures pratiques et les principes fondamentaux du langage PHP.

2. Précisez les moyens utilisés :

Je commence par initialiser le nouveau projet en créant un fichier « index.php » dans un nouveau répertoire situé dans le dossier « MAMP/htdocs ». J'ajoute la structure HTML de base dans ce fichier. Ensuite, je récupère le CDN de la bibliothèque Bootstrap via jsDelivr et l'ajoute aux balises <link> de la section <head> du fichier.

Le formateur nous fournit un fichier ZIP contenant toutes les images, ressources Bootstrap et fichiers CSS nécessaires. Pour la partie front-end du site, j'utilise des exemples fournis par Bootstrap. Je modifie simplement les différents éléments du fichier « index.php » pour qu'ils correspondent au thème de l'application.

Afin d'améliorer l'organisation du projet, je réparti les fichiers PHP dans différents dossiers.



Chaque fichier est réutilisé avec la fonction **require_once()**.

Après cela, je procède à la mise en place de la base de données. Le formateur nous fournit un fichier SQL pré-rempli contenant des tables et des données. Pour accomplir cette tâche, j'établi une connexion avec les serveurs MySQL et le serveur web, puis j'utilise PHPMyAdmin pour importer le script SQL.

Base de données - PHPMyAdmin

J'établi la connexion avec ma base de données à l'aide de la classe **PDO** (PHP Data Objects). Le code ci-dessous permet de créer une connexion à la base de données MySQL et fournit un point d'accès pour interagir avec la base de données dans le reste du code à travers des requêtes SQL.
Une fois la connexion établie, l'objet **PDO** est assigné à la variable **$pdo**.

```
lib > pdo.php > ...
1  <?php
2  $pdo = new PDO(
3      'mysql:dbname=studi_live_cuisinea;host=localhost;charset=utf8mb4;unix_socket=/Applications/MAMP/tmp/mysql/mysql.sock', 'root', 'root'
4  );
5  ?>
6
```

Pour sécuriser les requêtes et prévenir les attaques par injection SQL lors de l'inscription et de la connexion des utilisateurs. J'utilise la méthode **bindParam()** dans plusieurs fonctions.

```
lib > user.php > PHP Intelephense > addUser
1  <?php
2
3  function addUser(PDO $pdo, string $first_name, string $last_name, string $email, string $password) {
4      $sql = "INSERT INTO `users` (`first_name`, `last_name`, `email`, `password`, `role`) VALUES (:first_name, :last_name, :email, :password, :role);";
5      $query = $pdo->prepare($sql);
6
7      $password = password_hash($password, PASSWORD_DEFAULT);
8
9      $role = 'subscriber';
10     $query->bindParam(':first_name', $first_name, PDO::PARAM_STR);
11     $query->bindParam(':last_name', $last_name, PDO::PARAM_STR);
12     $query->bindParam(':email', $email, PDO::PARAM_STR);
13     $query->bindParam(':password', $password, PDO::PARAM_STR);
14     $query->bindParam(':role', $role, PDO::PARAM_STR);
15
16     return $query->execute();
17 }
18
19 function verifyUserLoginPassword(PDO $pdo, string $email, string $password) {
20     $query = $pdo->prepare("SELECT * FROM users WHERE email = :email");
21     $query->bindParam(':email', $email, PDO::PARAM_STR);
22     $query->execute();
23     $user = $query->fetch();
24
25     if ($user && password_verify($password, $user['password'])) {
26         return $user;
27     } else {
28         return false;
29     }
30 }
```

Dans le code ci-dessus, pour un utilisateur ayant le rôle « subscriber », la méthode **bindParam()** sur la première ligne, lie la valeur de la variable **$first_name** au paramètre **:first_name** dans la requête préparée. Lorsque la requête est exécutée avec **execute()**, la valeur de **$first_name** est utilisée pour exécuter la requête SQL de manière sécurisée.

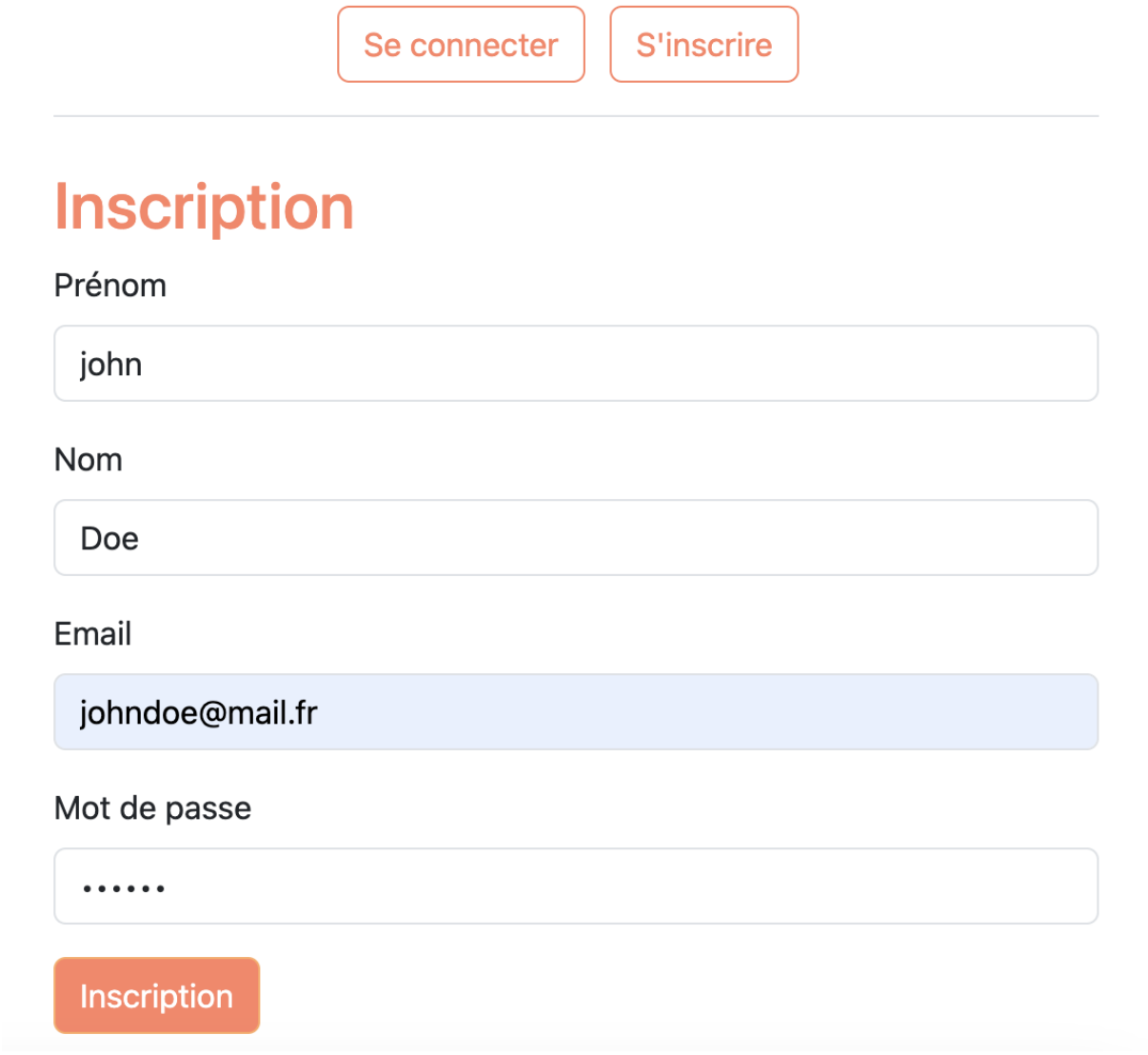
Le mot de passe est haché en utilisant la fonction **password_hash()** de PHP, qui génère une version sécurisée du mot de passe.

Lors du formulaire d'inscription, je crée des tableaux vides pour gérer les éventuelles erreurs lors de la soumission du formulaire. Ensuite, je vérifie si le formulaire a bien été soumis en passant les données saisies dans le formulaire **$_POST()** et l'objet de connexion à la base de données **$pdo**.

```php
<?php
  require_once('templates/header.php');
  require_once('lib/user.php');

  $errors = [];
  $messages = [];

  if (isset($_POST['addUser'])) {
    $res = addUser($pdo, $_POST['first_name'], $_POST['last_name'], $_POST['email'], $_POST['password']);
    if ($res) {
        $messages[] = 'Merci pour votre inscription';
    } else {
        $errors[] = 'Une erreur s\'est produite lors de votre inscription';
    }
  }
?>

<h1>Inscription</h1>
<?php foreach ($messages as $message) { ?>
    <div class="alert alert-success">
        <?=$message; ?>
    </div>
<?php } ?>
<?php foreach ($errors as $error) { ?>
    <div class="alert alert-danger">
        <?=$error; ?>
    </div>
<?php } ?>
<form method="POST" enctype="multipart/form-data">
    <div class="mb-3">
        <label for="first_name" class="form-label">Prénom</label>
        <input type="first_name" name="first_name" id="first_name" class="form-control">
    </div>
    <div class="mb-3">
        <label for="last_name" class="form-label">Nom</label>
        <input type="last_name" name="last_name" id="last_name" class="form-control">
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" name="email" id="email" class="form-control">
    </div>
    <div class="mb-3">
        <label for="password" class="form-label">Mot de passe</label>
        <input type="password" name="password" id="password" class="form-control">
    </div>
    <input type="submit" value="Inscription" name="addUser" class="btn btn-primary">
</form>

<?php
require_once('templates/footer.php');
?>
```

inscription.php

Je maintiens la même approche pour les fichiers « login.php », puis je procède à des tests d'inscription et vérifie si toutes les données sont correctement enregistrées dans la base de données.



Inscription et données PHPMyAdmin

Après avoir vérifié que le mot de passe est correct, je démarre les sessions en utilisant la fonction **session_start()**. Cela permet de stocker l'utilisateur en session et d'accéder à ces informations dans d'autres pages afin de les utiliser ultérieurement.

```php
<?php
session_start();
```

lib/session.php

Ensuite, je passe à la représentation des recettes sur la page d'accueil du site sous forme de cartes Bootstrap. J'élabore des fonctions qui récupèrent les recettes depuis la base de données de manière sécurisée en utilisant la fonction **bindParam()** pour garantir la sécurité des données.

```php
function getRecipeById(PDO $pdo, int $id) {
    $query = $pdo->prepare("SELECT * FROM recipes WHERE id = :id");
    $query->bindParam(':id', $id, PDO::PARAM_INT);
    $query->execute();
    return $query->fetch();
}
```

lib/recipe.php

```php
function getRecipes(PDO $pdo, int $limit = null) {
    $sql = 'SELECT * FROM recipes ORDER BY id DESC';

    if ($limit) {
        $sql .= ' LIMIT :limit';
    }

    $query = $pdo->prepare($sql);

    if ($limit) {
        $query->bindParam(':limit', $limit, PDO::PARAM_INT);
    }

    $query->execute();
    return $query->fetchAll();
}
```

lib/recipe.php

La fonction **getRecipeById()** récupère les informations d'une recette à partir de la base de données en utilisant un identifiant spécifique, en veillant à sécuriser la requête SQL grâce à la méthode **bindParam()**.
La fonction **getRecipes()** récupère toutes les recettes à partir de la base de données utilisant un affichage aléatoire par la requête SQL **ORDER BY** id **DESC**, avec une limite qui sera spécifiée dans une constante ultérieure, en utilisant également la méthode **bindParam()** pour sécuriser la requête SQL. Le **fetchAll()** sert à récupérer plusieurs éléments.

Pour afficher dynamiquement les recettes, je stocke la fonction **getRecipes()** dans une variable **$recipes** ensuite, j'utilise une boucle **foreach()** pour parcourir les résultats en incluant le fichier « recipe_partial.php » qui contient les cartes Bootstrap pour la présentation des recettes.
La fonction accepte deux arguments, **$pdo** pour se connecter à la base de données et **_HOME_RECIPES_LIMIT_** , une constante qui est définie dans un fichier à part nommé config.php pour la limite du nombre de recettes affichées sur la page.

```php
<?php
  require_once('templates/header.php');
  require_once('lib/recipe.php');

  $recipes = getRecipes($pdo, _HOME_RECIPES_LIMIT_);

?>

    <div class="row flex-lg-row-reverse align-items-center g-5 py-5">
      <div class="col-10 col-sm-8 col-lg-6">
        <img src="assets/images/logo-cuisinea.jpg" class="d-block mx-lg-auto img-fluid" alt="Logo Cuisinea" width="350" loading="lazy">
      </div>
      <div class="col-lg-6">
        <h1 class="display-5 fw-bold lh-1 mb-3">Cuisinea - Recettes de cuisine</h1>
        <p class="lead">Découvrez des recettes savoureuses, créatives et faciles à réaliser.
          Que vous soyez un amateur passionné ou un chef en herbe,
          notre site est conçu pour satisfaire vos papilles et
          vous faire voyager à travers les saveurs.
          Que vous soyez à la recherche de recettes traditionnelles revisitées ou de créations
          originales, vous trouverez votre bonheur parmi nos pages.
          Préparez-vous à épater vos convives et à créer des moments mémorables autour
          de la table. Rejoignez-nous dans cette aventure gastronomique
          et laissez Cuisinea être votre guide vers de délicieuses expériences culinaires !</p>
        <div class="d-grid gap-2 d-md-flex justify-content-md-start">
          <a href="recettes.php" class="btn btn-primary">Voir nos recettes</a>
        </div>
      </div>
    </div>
    <div class="row">
      <?php foreach ($recipes as $key => $recipe) {
        include('templates/recipe_partial.php');
      } ?>
    </div>

<?php
require_once('templates/footer.php');
?>
```

index.php

```php
define('_HOME_RECIPES_LIMIT_', 6);
```

lib/config.php

La fonction **getRecipeImage()** du code ci-dessous utilise le chemin d'accès à l'image associée à une recette. Elle prend en argument la variable **$image** qui représente le nom de l'image associée à la recette.
La fonction utilise une condition pour vérifier si la valeur de **$image** est « null ».

Si c'est le cas, cela indique qu'aucune image n'est associée à la recette. Dans cette situation, la fonction renvoie le chemin d'accès à une image par défaut qui est définie par les constantes **_ASSETS_IMG_PATH_** et **_RECIPES_IMG_PATH_** dans le fichier config.php. Ces constantes spécifient le chemin d'accès au dossier « assets/images » où l'image par défaut est stockée.

D'autre part, si la valeur de **$image** n'est pas « null », cela signifie qu'une image spécifique est associée à la recette. Dans ce cas, la fonction renvoie le chemin d'accès complet de l'image spécifique de la recette en utilisant les mêmes constantes **_ASSETS_IMG_PATH_** et **_RECIPES_IMG_PATH_** pour construire le chemin approprié.

```php
function getRecipeImage(string|null $image) {
    if ($image === null) {
        return _ASSETS_IMG_PATH_.'recipe_default.jpg';
    } else {
        return _RECIPES_IMG_PATH_.$image;
    }
}
```

lib/recipe.php

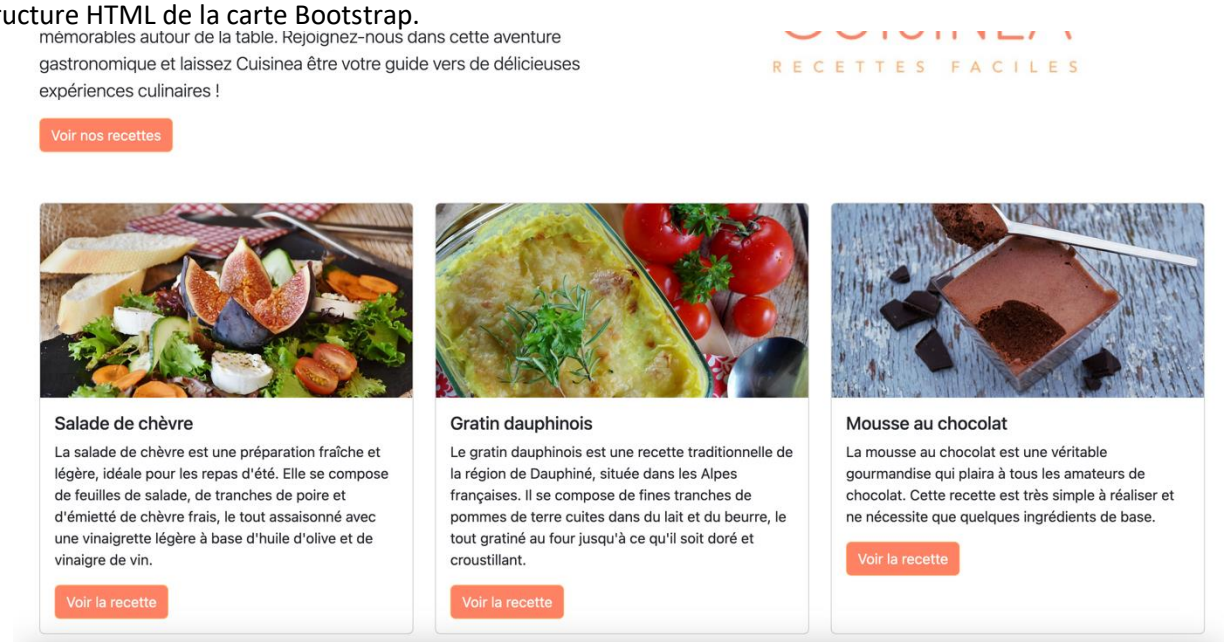
```php
define('_RECIPES_IMG_PATH_', 'uploads/recipes/');
define('_ASSETS_IMG_PATH_', 'assets/images/');
```

lib/config.php

```php
<div class="col-md-4 my-2 d-flex">
    <div class="card">
        <img src="<?=getRecipeImage($recipe['image']); ?>" class="card-img-top" alt="<?= $recipe['title']; ?>">
        <div class="card-body">
            <h5 class="card-title"><?= $recipe['title']; ?></h5>
            <p class="card-text"><?= $recipe['description']; ?></p>
            <a href="recette.php?id=<?=$recipe['id']; ?>" class="btn btn-primary">Voir la recette</a>
        </div>
    </div>
</div>
```

templates/recipe_partial.php

la clé **$key** de la boucle **foreach()** est utilisée pour itérer sur les éléments de recette et afficher leur contenu dans la structure HTML de la carte Bootstrap.



Il est possible de consulter les détails d'une recette à l'aide du bouton « Voir la recette » directement depuis la cards.

Cette page utilise des méthodes similaires à celles du fichier « index.php ». Elle utilise également une boucle **foreach()** pour afficher les ingrédients et les instructions récupérés depuis la base de données. La fonction **explode(PHP_EOL, $string)** est utilisée pour diviser la chaîne **$string** en utilisant la séquence de fin de ligne spécifique au système d'exploitation. Cela permet d'obtenir un tableau contenant les lignes individuelles de la chaîne d'origine en tant qu'éléments distincts. Ainsi, chaque ligne de texte est traitée séparément et affichée de manière appropriée dans le contenu de la page.

```php
function linesToArray(string $string) {
    return explode(PHP_EOL, $string);
}
```

lib/tools.php

```php
<?php
require_once('templates/header.php');
require_once('lib/tools.php');
require_once('lib/recipe.php');

$id = (int)$_GET['id'];
$recipe = getRecipeById($pdo, $id);

if ($recipe) {
    $ingredients = linesToArray($recipe['ingredients']);
    $instructions = linesToArray($recipe['instructions']);

?>

<div class="row flex-lg-row-reverse align-items-center g-5 py-5">
    <div class="col-10 col-sm-8 col-lg-6">
        <img src="<?=getRecipeImage($recipe['image']); ?>" class="d-block mx-lg-auto img-fluid" alt="<?=$recipe['title']; ?>" width="700" height="500" loading="lazy"
    </div>
    <div class="col-lg-6">
        <h1 class="display-5 fw-bold lh-1 mb-3"><?=$recipe['title']; ?></h1>
        <p class="lead"><?=$recipe['description']; ?></p>
    </div>
</div>

<div class="row flex-lg-row-reverse align-items-center g-5 py-5">
    <h2>Ingrédients</h2>
    <ul class="list-group">
        <?php foreach ($ingredients as $key => $ingredient) { ?>
            <li class="list-group-item"><?=$ingredient; ?></li>
        <?php } ?>
    </ul>
</div>

<div class="row flex-lg-row-reverse align-items-center g-5 py-5">
    <h2>Instructions</h2>
    <ol class="list-group list-group-numbered">
        <?php foreach ($instructions as $key => $instruction) { ?>
            <li class="list-group-item"><?=$instruction; ?></li>
        <?php } ?>
    </ol>
</div>

<?php } else { ?>
    <div class="row text-center">
        <h1>Recette introuvable</h1>
    </div>
<?php } ?>


<?php
require_once('templates/footer.php');
?>
```

recette.php

Salade de chèvre

La salade de chèvre est une préparation fraîche et légère, idéale pour les repas d'été. Elle se compose de feuilles de salade, de tranches de poire et d'émietté de chèvre frais, le tout assaisonné avec une vinaigrette légère à base d'huile d'olive et de vinaigre de vin.

Ingrédients

- 1 boule de chèvre frais
- 1 botte de salade (laitue, roquette, mâche, etc.)
- 1 poignée de noix (noisettes, amandes, noix de cajou, etc.)
- 1 poire
- Quelques feuilles de menthe
- 1 cuillère à soupe de vinaigrette (huile d'olive, vinaigre de vin, moutarde, sel et poivre)

Instructions

1. Commencez par laver et essorer votre salade. Découpez-la en fines lamelles et répartissez-la dans les assiettes.
2. Coupez la poire en fines tranches et répartissez-les sur la salade.
3. Émiettez le chèvre et répartissez-le sur la salade.
4. Parsemez la salade de noix concassées et de feuilles de menthe ciselées.
5. Préparez la vinaigrette en mélangeant une cuillère à soupe d'huile d'olive, une cuillère à soupe de vinaigre de vin, une pincée de moutarde, du sel et du poivre. Versez-la sur la salade et mélangez bien.
6. Servez la salade de chèvre fraîche, accompagnée d'un pain croustillant.

La page qui affiche les recettes utilise le même système de présentation avec les cartes Bootstrap et une boucle **foreach()** pour afficher toutes les recettes.

```php
<?php
  require_once('templates/header.php');
  require_once('lib/recipe.php');

  $recipes = getRecipes($pdo);

?>

    <div class="row flex-lg-row-reverse align-items-center g-5 py-5">
        <h1>Liste des recettes</h1>
    </div>

    <div class="row">
      <?php foreach ($recipes as $key => $recipe) {
        include('templates/recipe_partial.php');
      } ?>
    </div>

<?php
require_once('templates/footer.php');
?>
```

recettes.php

Dans la barre de navigation, je souhaite mettre en évidence le nom de la page active en utilisant une classe Bootstrap. Je stocke dans un tableau associatif les éléments de la barre de menus qui est utilisé dans une boucle **foreach()**. Ensuite, j'ajoute une condition pour vérifier si la page actuelle correspond strictement à la valeur de la clé **$key** d'un élément sélectionné dans la barre de menus. Si c'était le cas, la classe Bootstrap "active" est activée.

```php
$mainMenu = [
  'index.php' => 'Accueil',
  'recettes.php' => 'Nos recettes',
  'ajout_modification_recette.php' => 'Ajout/modif recette'
];
```

lib/config.php

En utilisant **basename($_SERVER['SCRIPT_NAME'])**, j'obtiens donc le nom du fichier de la page actuelle, sans le chemin complet.

```php
$currentPage = basename($_SERVER['SCRIPT_NAME']);
```

templates/header.php

```php
<ul class="nav col-12 col-md-auto mb-2 justify-content-center mb-md-0 nav nav-pills">
  <?php foreach ($mainMenu as $key => $value) { ?>
    <li class="nav-item"><a href="<?=$key; ?>" class="nav-link
    <?php if ($currentPage === $key) { echo 'active'; } ?>"><?=$value ;?></a></li>
  <?php } ?>
</ul>
```

templates/header.php

Afin de permettre à un utilisateur connecté d'effectuer les opérations de création, lecture, mise à jour et suppression (CRUD) sur les recettes, je mets en place un formulaire HTML. Ce formulaire utilise la méthode **POST**, ce qui permet de soumettre les données saisies vers le serveur de la base de données une fois le formulaire envoyé.
Pour permettre à l'utilisateur de remplir les champs d'un formulaire, je crée une variable **$recipe** qui est un tableau associatif initialisé avec des chaînes vides. Cela garantit que les champs du formulaire sont affichés vides, prêts à être remplis par l'utilisateur.
En utilisant **isset($_POST)** comme condition, le code vérifie si le formulaire a été soumis avec des données via la méthode **POST**, et si c'est le cas, il appelle une fonction d'insertion de recette.
Par la suite, des conditions sont utilisées pour vérifier si des fichiers sont téléchargés. Des validations sont effectuées sur ces fichiers, telles que la vérification de la taille et du type de fichier, ainsi que la prévention des doublons de noms d'image. Ensuite, la recette est enregistrée dans la base de données en utilisant les données fournies par l'utilisateur. Si l'opération réussit, un message de succès est affiché. En revanche, si des erreurs se produisent, elles sont enregistrées et affichées à l'utilisateur. Cela permet de gérer les différents scénarios liés à l'enregistrement des recettes et de fournir des messages appropriés à l'utilisateur en fonction du résultat.
Pour permettre à l'utilisateur de placer ses recettes dans des catégories, je créé une fonction appelée **getCategories()**. Cette fonction utilise les données de la table « categories » de la base de données en utilisant la variable **$pdo** qui représente l'objet de connexion à la base de données. Ensuite, j'utilise une boucle **foreach()** pour itérer à travers les catégories récupérées. Cela me permet d'afficher les différentes catégories à l'utilisateur.

```php
function getCategories(PDO $pdo) {
    $sql = 'SELECT * FROM categories';
    $query = $pdo->prepare($sql);

    $query->execute();
    return $query->fetchAll();
}
```

lib/category.php

```php
<?php
require_once('templates/header.php');

if(!isset($_SESSION['user'])) {
    header('location: login.php');
}

require_once('lib/tools.php');
require_once('lib/recipe.php');
require_once('lib/category.php');

$errors = [];
$messages = [];
$recipe = [
    'title' => '',
    'description' => '',
    'ingredients' => '',
    'instructions' => '',
    'category_id' => '',
];
```

```php
$categories = getCategories($pdo);

if (isset($_POST['saveRecipe'])) {
    $fileName = null;
    // Si un fichier a été envoyé
    if(isset($_FILES['file']['tmp_name']) && $_FILES['file']['tmp_name'] != '') {
        // la méthode getimagessize va retourner false si le fichier n'est pas une image
        $checkImage = getimagesize($_FILES['file']['tmp_name']);
        if ($checkImage !== false) {
            // Si c'est une image on traite
            $fileName = uniqid().'-'.slugify($_FILES['file']['name']);
            move_uploaded_file($_FILES['file']['tmp_name'], _RECIPES_IMG_PATH_.$fileName);
        } else {
            // Sinon on affiche un message d'erreur
            $errors[] = 'Le fichier doit être une image';
        }
    }

    if (!$errors) {
        $res = saveRecipe($pdo, $_POST['category'], $_POST['title'], $_POST['description'], $_POST['ingredients'], $_POST['instructions'], $fileName);

        if ($res) {
            $messages[] = 'La recette a bien été sauvegardée';
        } else {
            $errors[] = 'La recette n\'a pas été sauvegardée';
        }
    }
    $recipe = [
        'title' => $_POST['title'],
        'description' => $_POST['description'],
        'ingredients' => $_POST['ingredients'],
        'instructions' => $_POST['instructions'],
        'category_id' => $_POST['category'],
    ];

}

?>

<h1>Ajouter une recette</h1>

<?php foreach ($messages as $message) { ?>
    <div class="alert alert-success">
        <?=$message; ?>
    </div>
<?php } ?>

<?php foreach ($errors as $error) { ?>
    <div class="alert alert-danger">
        <?=$error; ?>
    </div>
<?php } ?>

<form method="POST" enctype="multipart/form-data">
    <div class="mb-3">
        <label for="title" class="form-label">Titre</label>
        <input type="text" name="title" id="title" class="form-control" value="<?=$recipe['title'] ;?>">
    </div>
    <div class="mb-3">
        <label for="description" class="form-label">Description</label>
        <textarea name="description" id="description" cols="30" rows="5" class="form-control"><?=$recipe['description'] ;?></textarea>
    </div>
    <div class="mb-3">
        <label for="ingredients" class="form-label">Ingredients</label>
        <textarea name="ingredients" id="ingredients" cols="30" rows="5" class="form-control"><?=$recipe['ingredients'] ;?></textarea>
    </div>
    <div class="mb-3">
        <label for="instructions" class="form-label">Instructions</label>
        <textarea name="instructions" id="instructions" cols="30" rows="5" class="form-control"><?=$recipe['instructions'] ;?></textarea>
    </div>
    <div class="mb-3">
        <label for="category" class="form-label">Catégorie</label>
        <select name="category" id="category" class="form-select">

            <?php foreach ($categories as $category) { ?>
                <option value="<?=$category['id']; ?>" <?php if ($recipe['category_id'] == $category['id']) { echo 'selected="selected"'; } ?>><?=$category['name'];?></option>
            <?php } ?>
        </select>
    </div>
    <div class="mb-3">
        <label for="file" class="form-label">Image</label>
        <input type="file" name="file" id="file">
    </div>
    <input type="submit" value="Enregistrer" name="saveRecipe" class="btn btn-primary">
</form>

<?php
require_once('templates/footer.php');
?>
```

ajout_modification_recette.php

3. Avec qui avez-vous travaillé ?

J'ai travaillé seul, mais j'ai pu compter sur le soutien du formateur tout au long des sessions en direct pour m'aider à mener à bien ce projet PHP.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *10/01/2023* au : *14/02/2023*

5. Informations complémentaires *(facultatif)*

Activité-type 2

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

Exemple n° 3 ▸ ***Projet - Bookeo (PHP - POO)***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Au cours de notre formation, nous avons exploré le langage PHP et la programmation orientée objet (POO). Dans le but de mettre en pratique nos compétences acquises, nous avons bénéficié de l'encadrement d'un formateur expérimenté lors de plusieurs sessions en direct dédiées à la création d'un mini-site de gestion de bandes dessinées, mangas et livres intitulé "Bookeo".

Tout au long de ces sessions, le formateur nous a guidé à travers les étapes essentielles de la construction d'une application en suivant les meilleures pratiques et les principes fondamentaux du langage PHP et de la POO.

2. Précisez les moyens utilisés :

Pour suivre un pattern, une recommandation optimisée, je vais me baser sur le MVC (Modèle-Vue-Contrôleur) afin de séparer les responsabilités des différentes parties de l'application en trois composantes distinctes :
Modèles :
Représente les données et règles métiers associés. Contient les méthodes qui permettent de manipuler ces données, fonctions de validation, transformation et persistance.
Vue :
L'interface utilisateur de l'application, la manière dont les données sont présentées à l'utilisateur final. Responsable de l'affichage des données, mise en forme et organisation.
Contrôleur :
Agit comme l'intermédiaire entre le modèle et la vue. Gère les interactions utilisateurs (requêtes), récupère les données du modèle et les transmet à la vue.
La méthode **route()** dans le fichier « Controller.php » est chargée d'analyser le contenu de l'URL et, en fonction de celui-ci, de transmettre le contrôle à d'autres contrôleurs à l'aide d'une structure **switch()**

App > Controlleur > Controller.php > PHP Intelephense > Controller > route

```php
<?php

namespace App\Controller;

Class Controller
{
    public function route(): void
    {
        if (isset($_GET['controller'])){
            switch ($_GET['controller']){
                case 'page':
                    //charger controlleur page
                    break;
                case 'book':
                    //charger controlleur book
                    break;
                default:
                    //erreur
                    break;
            }
        }else{
            //charger la page d'accueil
        }
    }
}
```

La méthode **spl_autoload_register()** permet de charger automatiquement les classes lorsqu'elles sont utilisées, sans nécessiter de « require » explicite pour chaque fichier. En instanciant une nouvelle instance du contrôleur et en appelant la méthode **route()**, j'évite de répéter le chargement des classes à chaque page. Cela simplifie le processus de gestion des dépendances et améliore l'efficacité du chargement des classes.

index.php > ...

```php
<?php
spl_autoload_register();

use App\Controller\Controller;

$controller = new Controller();
$controller->route();

?>
```

Dans ma méthode **render()**, je commence par construire le chemin du fichier en combinant « _ROOTPATH_ » que j'ai défini dans le fichier « index.php », avec le chemin du fichier template spécifié par le paramètre « path ». Ensuite, je vérifie si le fichier existe en utilisant la fonction **file_exists()**. Si le fichier n'existe pas, je peux générer une erreur appropriée ou prendre toute autre action nécessaire. Sinon, si le fichier existe, j'inclus son contenu en utilisant « require_once » pour l'afficher à l'écran.

```php
define('_ROOTPATH_', __DIR__);
```

index.php

```php
<?php require_once _ROOTPATH_.'/templates/header.php'; ?>

<h1>À propos</h1>

<?php require_once _ROOTPATH_.'/templates/footer.php'; ?>
```

templates/page/about.php

```php
    protected function render(string $path, array $params = []): void
    {
        $filePath = _ROOTPATH_. '/templates/'.$path.'.php';
        if(!file_exists($filePath)){
            //Générer erreur
        }else {
            require_once $filePath;
        }
    }
}
```

App/Controller/Controller.php

Dans cette partie du code, pour le système de gestions d'exceptions, j'ajoute une gestion d'exception pour gérer les cas où le fichier template n'est pas trouvé.
J'essaie d'exécuter le code suivant : si le fichier spécifié par le chemin **$filePath** n'existe pas, je génère une exception avec un message indiquant que le fichier n'a pas été trouvé. Sinon, si le fichier existe, j'inclus son contenu en utilisant « require_once » pour l'afficher à l'écran. Si une exception est levée, j'attrape cette exception et j'affiche le message d'erreur correspondant à l'écran.

```php
protected function render(string $path, array $params = []): void
{
    $filePath = _ROOTPATH_. '/templates/'.$path.'.php';

    try{
        if(!file_exists($filePath)){
            //Générer erreur
            throw new \Exception("Fichier non trouvé : ".$filePath);
        }else {
            require_once $filePath;
        }
    }catch(\Exception $e){
        echo $e->getMessage();
    }
}
```

App/Controller/Controller.php

Dans la méthode « about », je définis un tableau associatif **$params** avec des valeurs spécifiques. Ensuite, j'appelle la méthode **render()** en lui passant le chemin du template « page/about » et le tableau **$params** afin que les valeurs soient extraites et rendues disponibles dans le fichier de template.
Enfin, dans cette partie du code, j'ajoute la méthode **extract()** qui est utilisée pour extraire les éléments d'un tableau associatif dans des variables individuelles

```php
protected function about()
{
    $params = [
        'test' => 'abc',
        'test2' => 'abc2'
    ];

    $this->render('page/about', $params);
}
```

App/Controller/PageController.php

```php
protected function render(string $path, array $params = []): void
{
    $filePath = _ROOTPATH_.'/templates/'.$path.'.php';

    try{
        if(!file_exists($filePath)){
            //Générer erreur
            throw new \Exception("Fichier non trouvé : ".$filePath);
        }else {
            extract($params);
            require_once $filePath;
        }
    }catch(\Exception $e){
        echo $e->getMessage();
    }
}
```

App/Controller/Controller.php

Pour continuer dans la gestion des erreurs, j'inclus aussi la méthode « try-catch » à la fonction **route()** du Controller dans le fichier « Controller.php ».
Je crée un fichier « default.php » dans le dossier « templates/errors », j'inclus le header et le footer, puis j'ajoute le message d'erreur.

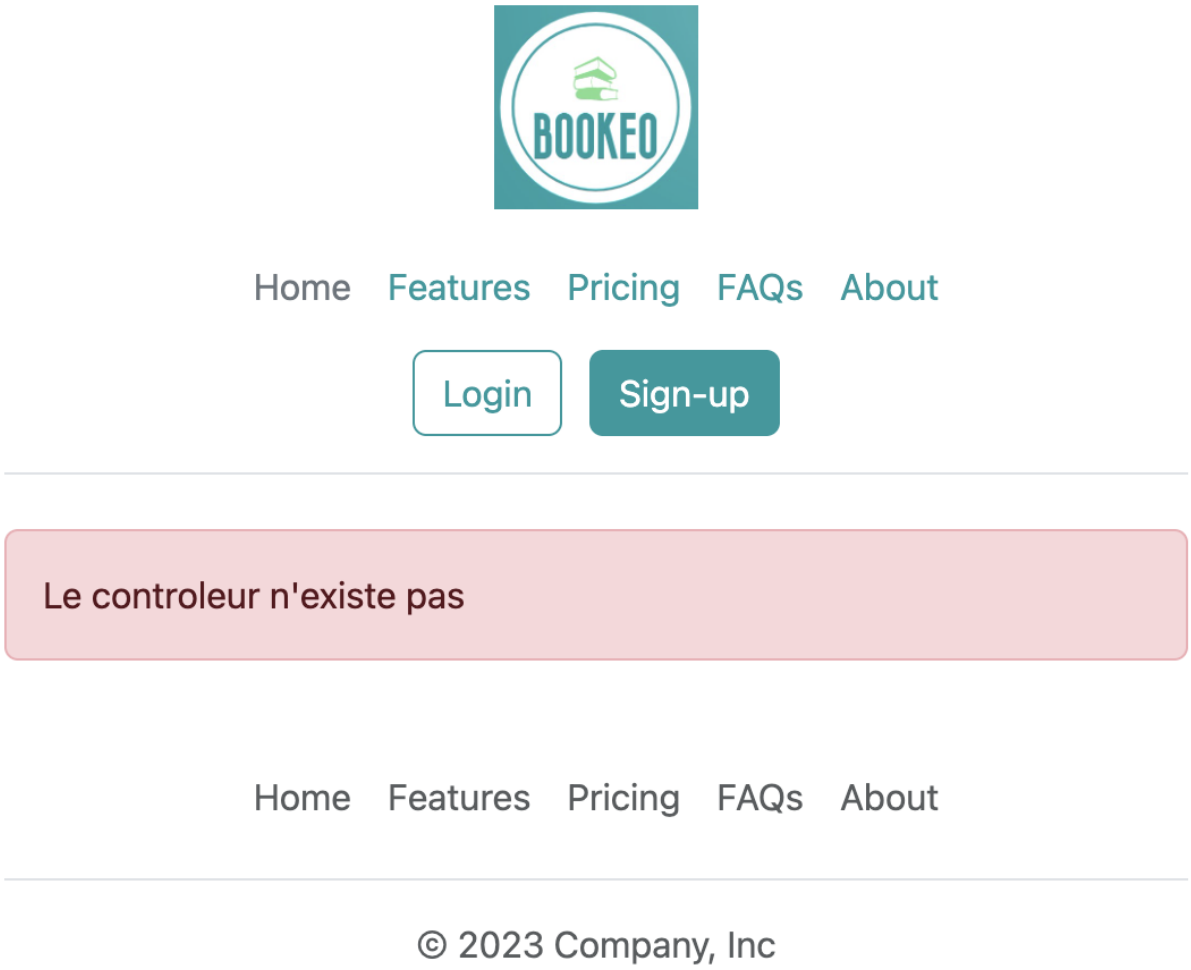
templates > errors > default.php > ...

```php
<?php require_once _ROOTPATH_.'/templates/header.php'; ?>

<?php if($error) { ?>
    <div class="alert alert-danger">
        <?=$error; ?>
    </div>
<?php } ?>


<?php require_once _ROOTPATH_.'/templates/footer.php'; ?>
```

Dans le bloc **catch()**, je renvoie la méthode **render()** en utilisant la clé « error » avec la valeur correspondant au message de l'exception contenu dans le fichier « default » de la méthode **switch**.

```php
}catch (\Exception $e) {
    $this->render('errors/default', [
        'error' => $e->getMessage()
    ]);
}
```

App/Controller/Controller.php



http://localhost:3000/index.php?controller=pageXXX&action=about

Voici ce qui se produit en cas d'erreur d'URL pour une page XXX.

Afin de garantir une optimisation maximale, j'ajoute des blocs **catch()** incluant la méthode **render()** dans chaque contrôleur qui contiennent des instructions « try-catch » pour capturer les exceptions en cas d'erreur.

Dans le fichier « Book.php » du dossier « App/Entity », je crée la classe Book avec les propriétés protégées. En utilisant l'extension « PHP 8 Getter & Setter » de VSCode, je génère les getters et setters correspondants pour la classe Book.

Je crée le fichier « BookController.php » et j'inclus, comme pour le « PageController.php », la fonction **route()** avec une structure **switch** et une gestion des erreurs utilisant des blocs « try-catch. »
Dans le dossier « Repository », je crée le fichier « BookRepository.php » où je définis la fonction **findOneById()**. À l'intérieur de cette fonction, je simule un appel à la base de données en utilisant un tableau associatif. Ensuite, je crée une nouvelle instance de la classe **Book**, j'affecte les valeurs des clés du tableau aux propriétés correspondantes de l'objet **Book**, puis je retourne l'objet **Book**.

App > Repository > BookRepository.php > PHP Intelephense > BookRepository

```php
<?php

namespace App\Repository;

use App\Entity\book;

class BookRepository
{
    public function findOneById(int $id)
    {
        // Appel BDD
        $book = ['id' => 1,  'title' => 'Titre test', 'description' => 'Description test'];

        $bookEntity = new Book();
        $bookEntity->setId($book['id']);
        $bookEntity->setTitle($book['title']);
        $bookEntity->setDescription($book['description']);

        return $bookEntity;
    }
}
```

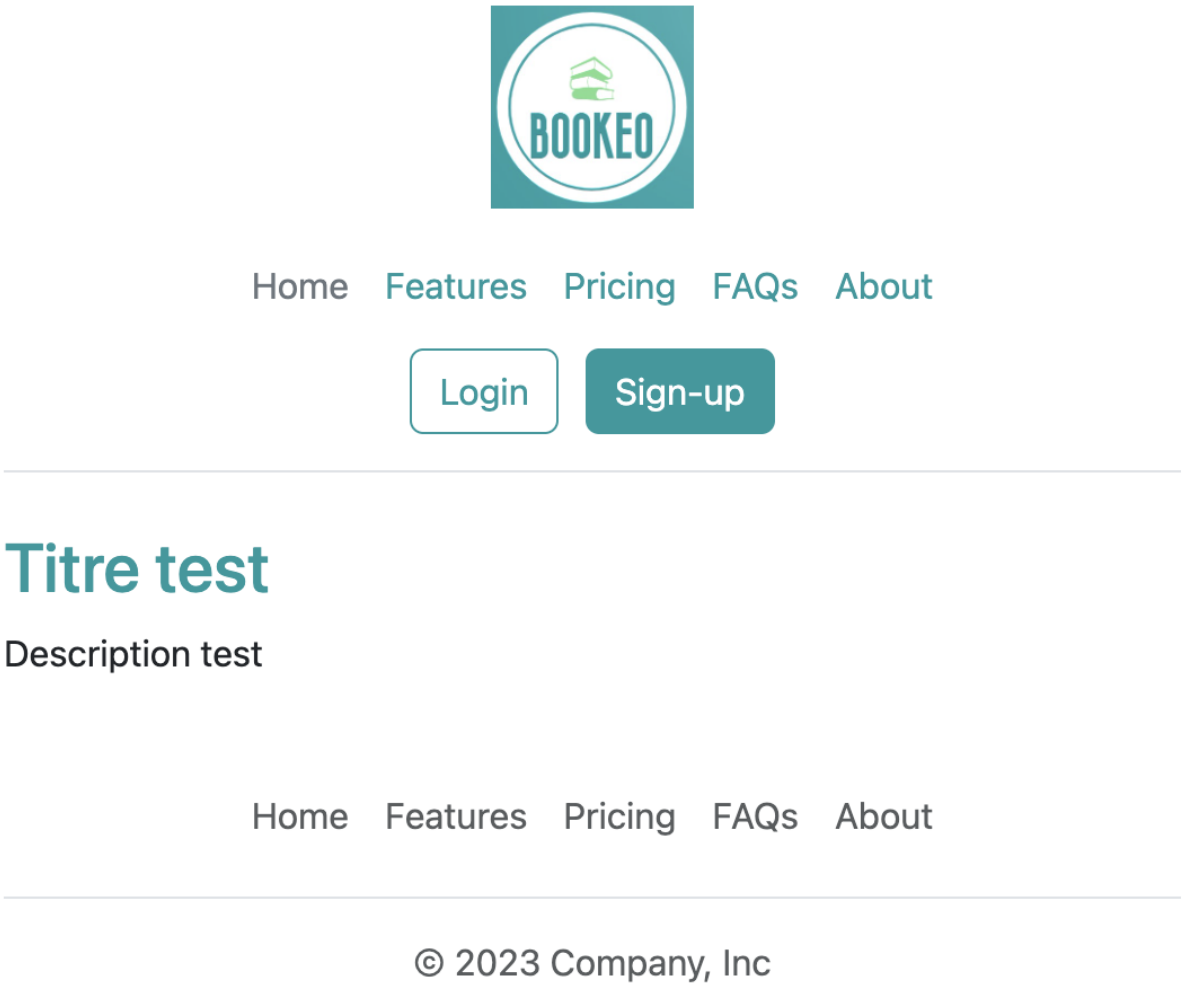
Dans le bloc try de la fonction **show()** du « BookController.php », je vérifie si l'identifiant (id) est défini dans la requête **GET**. Si c'est le cas, je convertis sa valeur en entier et j'appelle la fonction **findOneById($id)** du repository pour charger le livre correspondant. Ensuite, j'utilise la fonction **render()** pour afficher la vue « book/show » en transmettant le livre en paramètre.

```php
protected function show()
{
    try {
        if (isset($_GET['id'])){
            $id = (int)$_GET['id'];
            // Charger le livre par un appel au repository
            $bookRepository = new BookRepository();
            $book = $bookRepository->findOneById($id);

            $this->render('book/show', [
                'book' => $book
            ]);
        }else{
            throw new \Exception("L'id est manquant en paramètre");
        }
    }catch (\Exception $e){
        $this->render('errors/default', [
            'error' => $e->getMessage()
        ]);

    }
}
```

Si l'identifiant n'est pas présent dans la requête **GET**, je lance une exception avec le message « L'id est manquant en paramètre ».
En cas d'erreur capturée (dans le bloc catch) dans la fonction **show()**, j'affiche la vue « errors/default » en transmettant le message d'erreur dans la variable « error ».

templates > book > show.php > ...

```php
<?php require_once _ROOTPATH_.'/templates/header.php';
/* @var $book \App\Entity\book*/
?>

<h1><?=$book->getTitle(); ?></h1>
<p><?=$book->getDescription(); ?></p>

<?php require_once _ROOTPATH_.'/templates/footer.php'; ?>
```



http://localhost:3000/index.php?controller=book&action=show&id=1

Depuis le dossier « App/Database/Mysql.php », je crée une classe **Mysql** avec des propriétés privées pour les informations de connexion à la base de données.

```php
class Mysql
{
    private $db_name;
    private $db_user;
    private $db_password;
    private $db_port;
    private $db_host;
    private $pdo;
    private static $_instance = null;

    private function __construct()
    {
        $conf = require_once _ROOTPATH_.'/config.php';
        if (isset($conf['db_name'])){
            $this->db_name = $conf['db_name'];
        }
        if (isset($conf['db_user'])){
            $this->db_user = $conf['db_user'];
        }
        if (isset($conf['db_password'])){
            $this->db_password = $conf['db_password'];
        }
        if (isset($conf['db_port'])){
            $this->db_port = $conf['db_port'];
        }
        if (isset($conf['db_host'])){
            $this->db_host = $conf['db_host'];
        }
    }
```

App/Database/Mysql.php

Dans le constructeur de la classe, je récupère les informations de configuration à partir d'un fichier appelé « config.php » et j' attribue les valeurs correspondantes aux propriétés de la classe, comme **$db_name**, **$db_user**, **$db_password**, **$db_port** et **$db_host**.

config.php

```php
<?php

return [
    'db_name' => 'studi_bookeo',
    'db_user' =>'root',
    'db_password' =>'root',
    'db_port' =>'3306',
    'db_host' =>'localhost'
];

```

Je crée la base de données dans PHPMyAdmin et j'importe le fichier SQL préalablement rempli par le formateur. Cela me permet d'obtenir les tables dans la base de données.
La méthode statique **getInstance()** retourne une instance de la classe **Mysql**. Si la propriété statique **$_instance** est « null », cela signifie qu'aucune instance n'a été créée auparavant. Dans ce cas, une nouvelle instance de la classe **Mysql** est créée et assignée à la propriété **$_instance**. Ensuite, l'instance est retournée. Si une instance existe déjà, la même instance est renvoyée sans en créer une nouvelle.

```php
public static function getInstance():self
{
    if (is_null(self::$_instance)){
        self::$_instance = new Mysql();
    }
    return self::$_instance;
}
```

App/Database/Mysql.php

Dans la méthode **getPDO()** de la classe, je vérifie si la propriété **$pdo** est nulle. Si c'est le cas, j'instancie un nouvel objet de la classe **PDO** en utilisant les informations de connexion fournies dans les propriétés de la classe. La connexion est établie avec la base de données MySQL en utilisant le nom de la base de données, l'hôte, le port, l'utilisateur et le mot de passe.
Ensuite, je retourne l'objet **PDO** nouvellement créé ou l'objet **PDO** existant si la connexion a déjà été établie précédemment.

```php
public function getPDO():\PDO
{
    if (is_null($this->pdo)){
        $this->pdo = new \PDO('mysql:dbname=' . $this->db_name . ';charset=utf8;host=' . $this->db_host.':'.$this->db_port, $this->db_user, $this->db_password);
    }
    return $this->pdo;
}
```

App/Database/Mysql.php

Depuis le « BookRepository.php », dans la méthode **findOneById()** de la classe **BookRepository**, j'obtiens une instance de la classe **Mysql** en appelant la méthode statique **getInstance()** de cette classe. Ensuite, j'obtiens une instance de la classe **PDO** en appelant la méthode **getPDO()** sur l'instance de **Mysql**.

Ensuite, je prépare une requête SQL pour sélectionner une ligne de la table « book » où l'ID correspond à celui passé en paramètre. J'utilise un paramètre nommé « :id » pour éviter les injections SQL. J'exécute la requête et je récupère les résultats dans la variable **$book** sous forme de tableau associatif.

Ensuite, j'instancie un objet de la classe **book**. Je parcours le tableau **$book** et j'utilise la méthode « set » pour appeler dynamiquement les setters de la classe **book** en utilisant les noms des clés du tableau après les avoir convertis en PascalCase à l'aide de la classe **StringTools**.
Enfin, je retourne l'objet **book** ainsi créé.

```php
public function findOneById(int $id)
{
    // Appel BDD
    $mysql = Mysql::getInstance();
    $pdo = $mysql->getPDO();

    $query = $pdo->prepare('SELECT * FROM book WHERE id = :id');
    $query->bindValue(':id', $id, $pdo::PARAM_INT);
    $query->execute();
    $book = $query->fetch($pdo::FETCH_ASSOC);


    /*
    $book = ['id' => 1,  'title' => 'Titre test', 'description' => 'Description test'];
    $bookEntity = new Book();
    $bookEntity->setId($book['id']);
    $bookEntity->setTitle($book['title']);
    $bookEntity->setDescription($book['description']);
    */

    $bookEntity = new book();

    foreach ($book as $key => $value){
        $bookEntity->{'set'.StringTools::toPascalCase($key)}($value);
    }

    return $bookEntity;
}
```

App/Repository/BookRepository.php

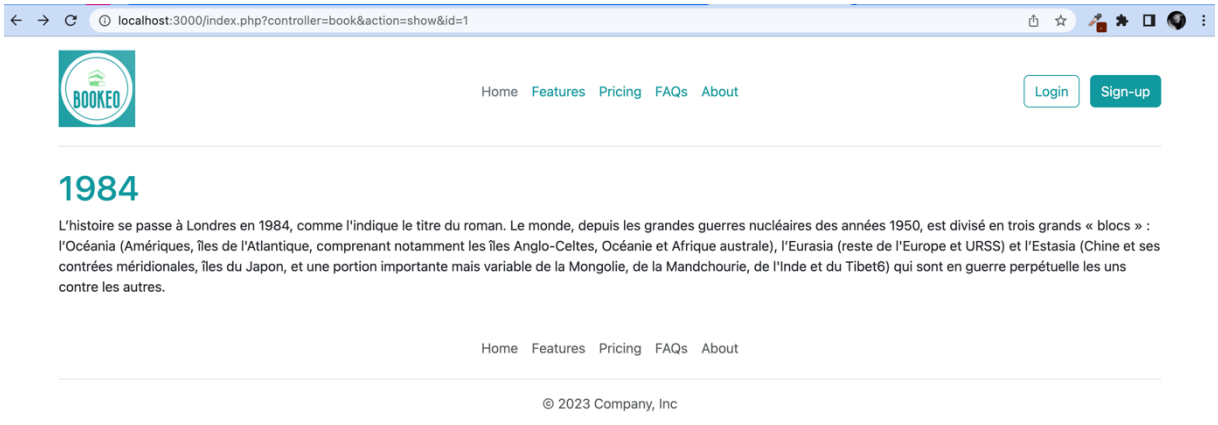
Dans un dossier à part nommé « Tools », je crée le fichier « StringTools.php ». La classe **StringTools** contient deux méthodes statiques : **toCamelCase()** et **toPascalCase()**. La méthode **toCamelCase()** convertit une chaîne en notation camelCase, tandis que la méthode **toPascalCase()** convertit une chaîne en notation PascalCase. Ces méthodes sont utiles pour manipuler et convertir des chaînes de caractères selon différentes conventions de casse.

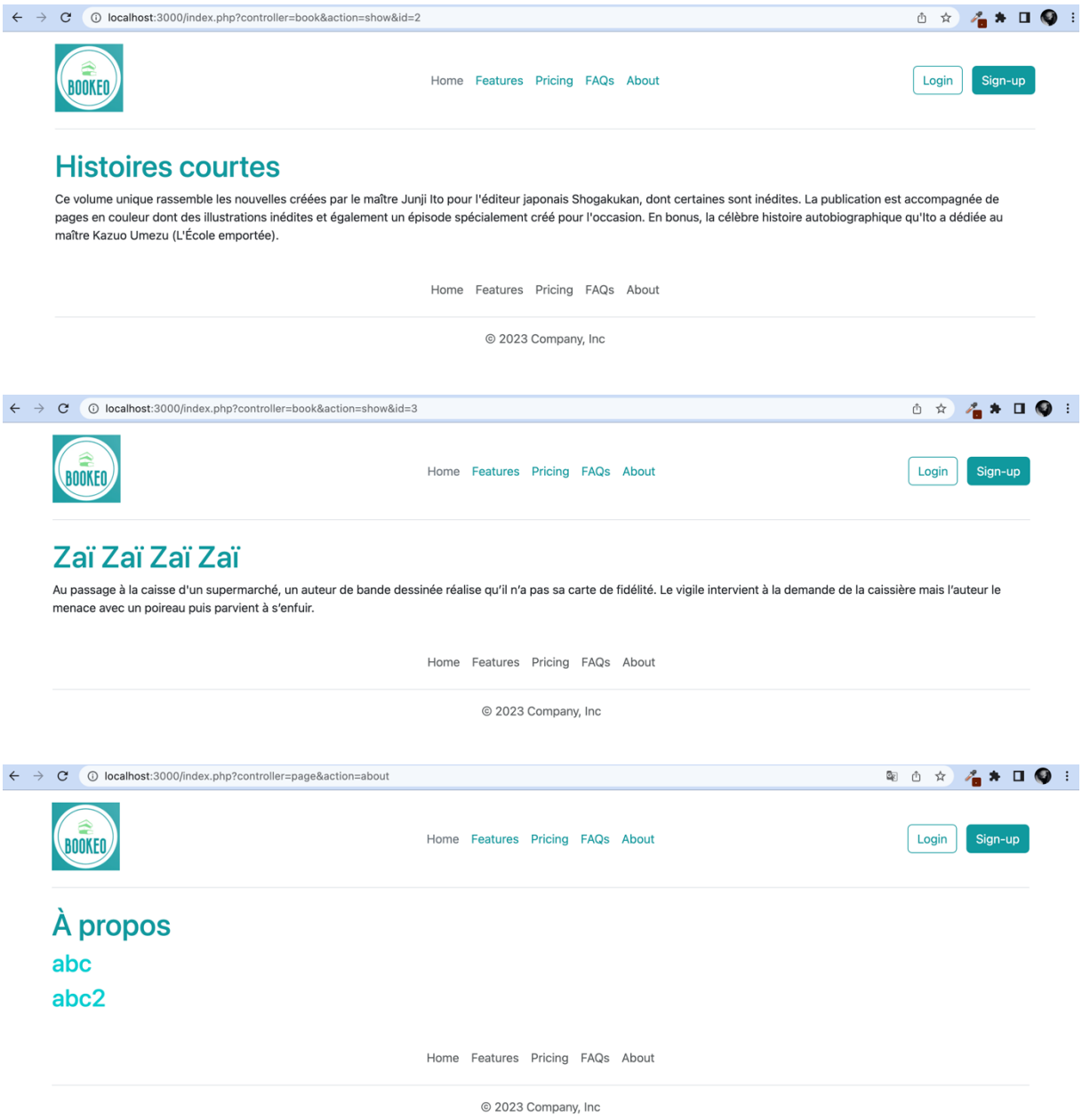
```php
<?php

namespace App\Tools;

class StringTools
{

    public static function toCamelCase($value, $pascalCase = false){
        $value = ucwords(str_replace(array('-', '_'), ' ', $value));
        $value = str_replace(' ', '', $value);
        if ($pascalCase === false) {
            return lcfirst($value);
        } else {
            return $value;
        }
    }
    public static function toPascalCase($value){

        return self::toCamelCase($value, true);
    }

}
```

App/Tools/StringTools.php

Pour terminer, dans l'entité "Book", j'ajoute les propriétés associées à ma base de données et j'injecte les getters et setters restants.

Une fois le code écrit, voici le résultat qui s'affiche dans un navigateur avec différentes URL.





3. Avec qui avez-vous travaillé ?

J'ai travaillé seul, mais j'ai pu compter sur le soutien du formateur tout au long des sessions en direct pour m'aider à mener à bien ce projet PHP et POO.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *15/02/2023* au : *28/02/2023*

5. Informations complémentaires *(facultatif)*

Activité-type 2

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

Exemple n°4 ▸ ***Projet inter-spécialités - Yourbook - Symfony (Backoffice)***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

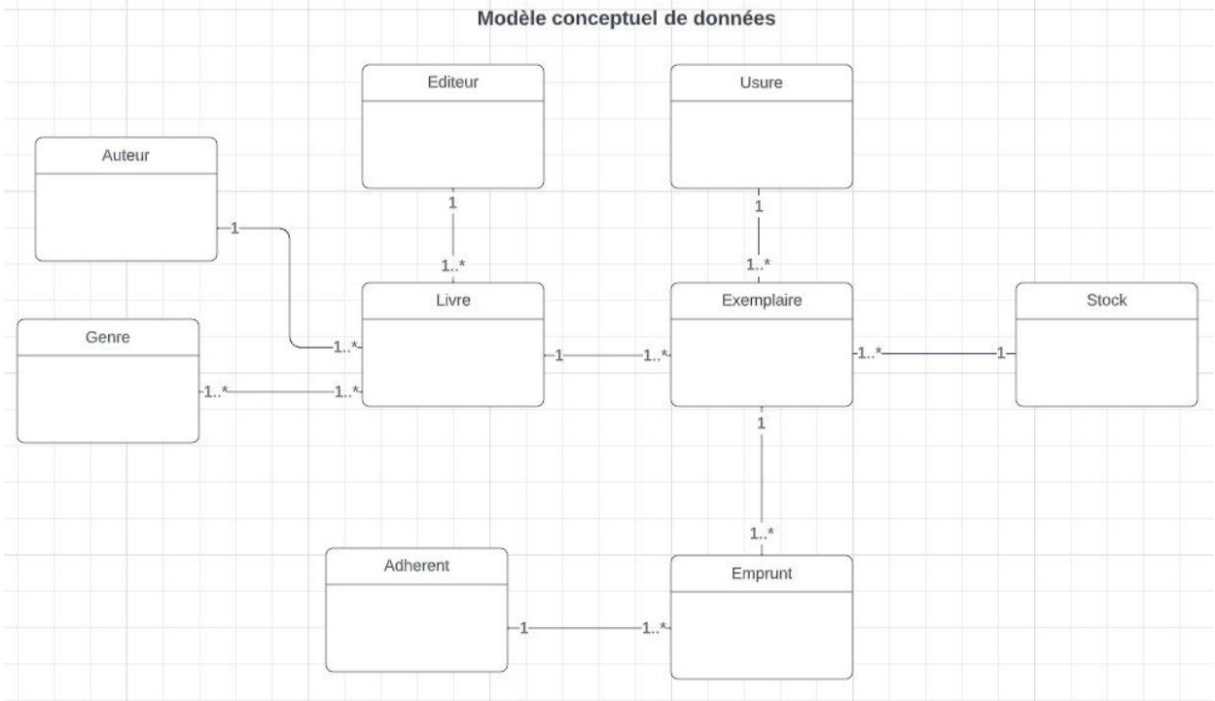
Durant notre formation, nous avons étudié en profondeur le langage PHP ainsi que le framework Symfony.

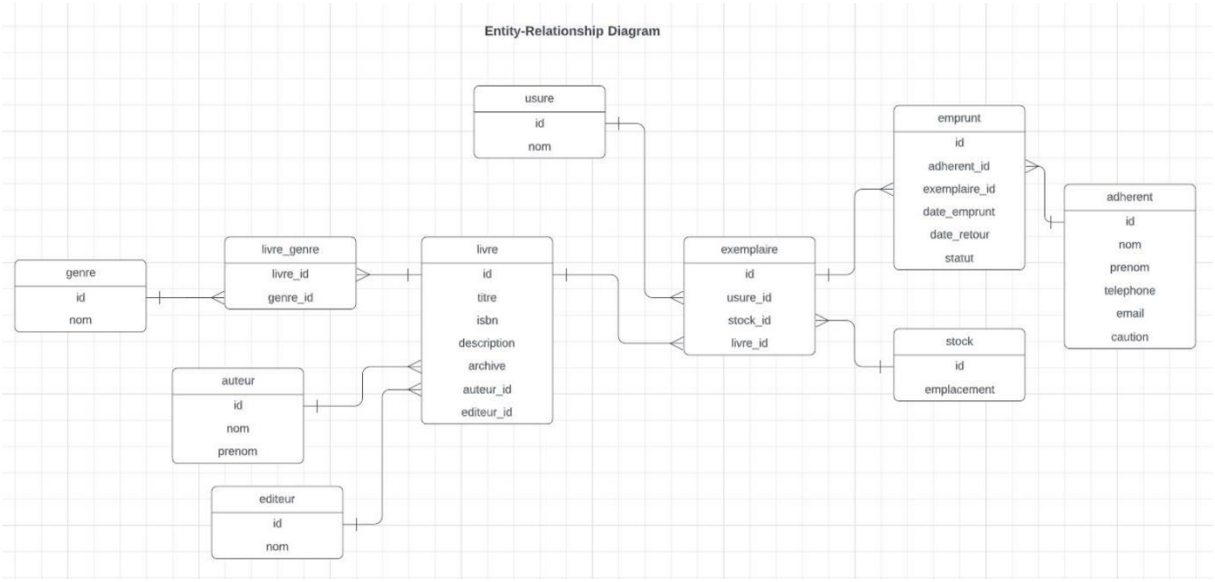
Pour mettre en pratique nos connaissances acquises, Studi a mis en place un projet collaboratif rassemblant diverses spécialités. Dans le cadre de ce projet intitulé « YourBook », notre formateur, spécialisé dans le développement du backoffice, nous a guidé dans l'utilisation de la bibliothèque EasyAdmin fournie par Symfony.

Nous avons bénéficié de séances d'encadrement en direct avec ce formateur expérimenté afin de bien appréhender la mise en place de ce projet.

2. Précisez les moyens utilisés :

Une fois que le Modèle Conceptuel de Données (MCD) et l'Entity-Relationship Diagram sont réalisés.





j'initialise un nouveau projet Symfony en utilisant la commande suivante :

symfony new symfony_yourbook --webapp

Je lance le serveur Symfony afin d'avoir une première vue dans mon navigateur web avec la commande :

symfony serve -d

Afin d'éviter de créer manuellement la structure de base d'un contrôleur, j'utilise la commande suivante pour générer mon premier contrôleur :

symfony console make:controller

Je crée les entités en me basant sur le MCD et l'Entity-Relationship Diagram avec la commande suivante :

symfony console make:entity

Avant d'effectuer les migrations, je procède de la manière suivante :
Tout d'abord, je crée la base de données sur la plateforme alwaysdata. Ensuite, dans Symfony, je duplique le fichier « .env » pour créer un fichier nommé « .env.local ». Ce fichier « .env.local » est exclu de la gestion de version pour des raisons de sécurité.
Dans le fichier « .env.local », j'effectue des modifications spécifiques, notamment dans la variable « DATABASE_URL », afin de sécuriser l'URL de ma base de données.

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"
DATABASE_URL="mysql://berenger-[redacted]@mysql-berenger.alwaysdata.net/berenger_symfony_yourbook?serverVersion=10.6.11-MariaDB - MariaDB Server"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=15&charset=utf8"
###< doctrine/doctrine-bundle ###
```

.env.local

Ensuite, je crée les migrations et j'utilise Doctrine pour les intégrer à ma base de données avec les commandes suivantes :

symfony console make :migration

et

symfony console doctrine :migrations :migrate

Après avoir effectué les migrations. Je consulte les instructions fournies par la documentation de Symfony pour utiliser la bibliothèque EasyAdmin et mettre en place les opérations CRUD (Create, Read, Update, Delete).
J'utilise la documentation pour créer facilement des interfaces d'administration pour les entités dans le projet Symfony « YourBook ».
J'installe le bundle avec la commande suivante :

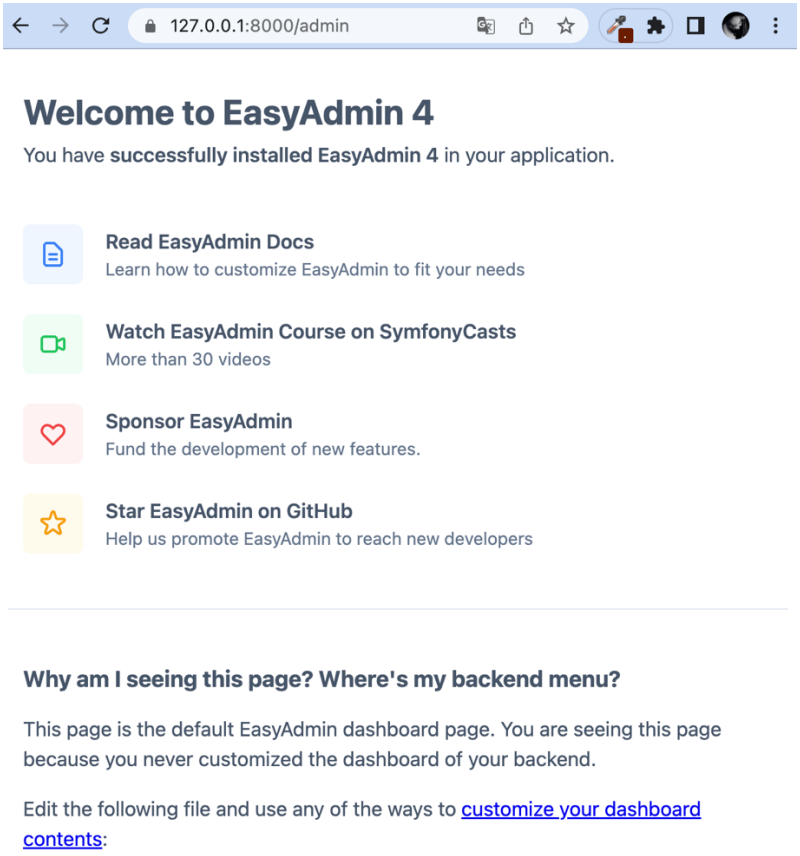
composer require easycorp/easyadmin-bundle

L'installation d'un bundle avec Composer présente l'avantage de pouvoir installer non seulement le bundle demandé, mais également toutes ses dépendances nécessaires.

Je crée le dashboard qui va faire office de page d'accueil EasyAdmin avec la commande

php bin/console make:admin:dashboard

Lorsque j'exécute cette commande, elle crée automatiquement un dossier « admin » dans le répertoire « src » de mon projet Symfony et configure une route « /admin ». Ensuite, je peux vérifier si tout fonctionne correctement en accédant à cette route « /admin » depuis mon navigateur et consulter le dashboard par défaut.

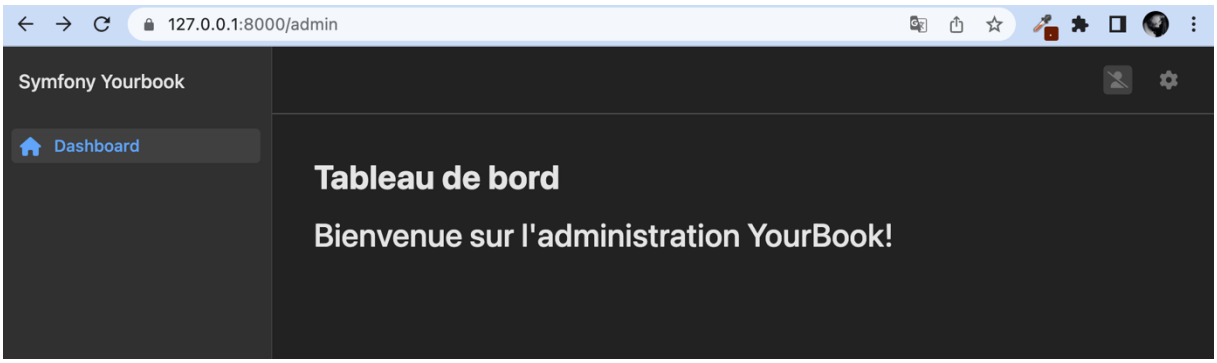


Je redirige la route en utilisant la méthode **render()** vers un fichier « dashboard.html.twig » que je crée dans le dossier « templates/admin ». Pour intégrer le layout fourni par EasyAdmin, je consulte la documentation et j'insère le « content Page Template » dans mon fichier Twig.

En utilisant l'instruction « Extends » dans mon fichier Twig, j'ai la possibilité de créer différents menus.

templates > admin > dashboard.html.twig

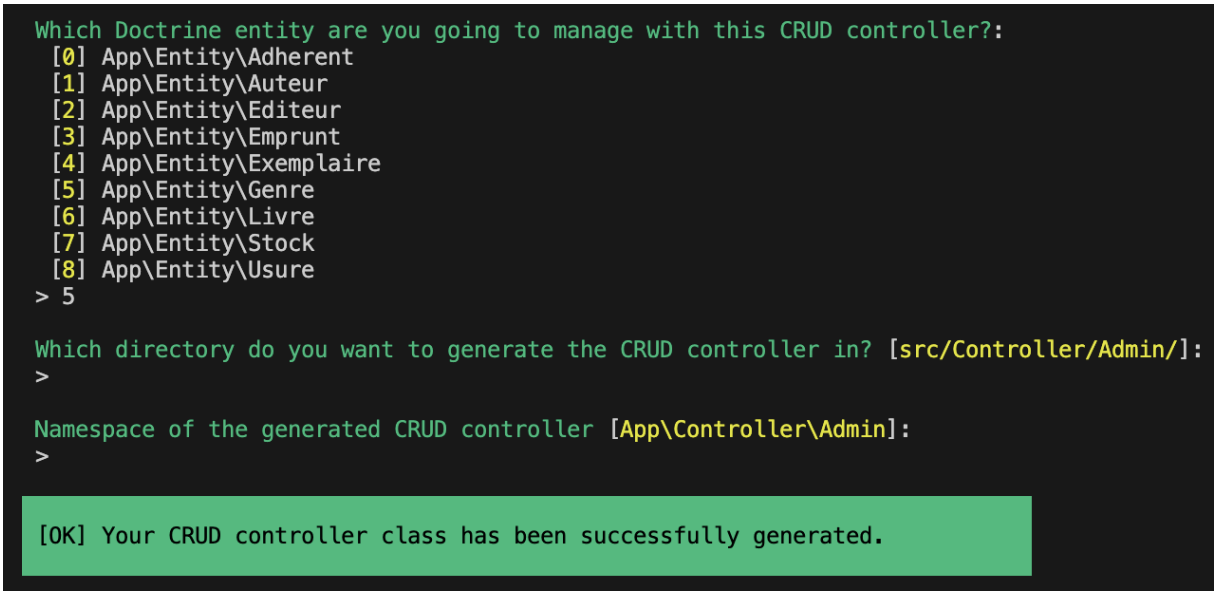
```
1  {% extends "@EasyAdmin/page/content.html.twig" %}
2
3  {% block content_title %}Tableau de bord{% endblock %}
4
5  {% block main %}
6  <h2>Bienvenue sur l'administration YourBook!</h2>
7  {% endblock %}
```



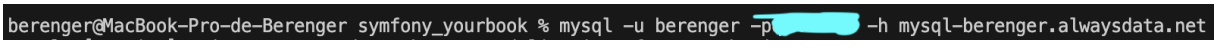
Lorsque j'exécute la commande suivante dans mon terminal :

php bin/console make:admin:crud

Je suis guidé à travers différentes étapes pour créer le premier CRUD. Chaque fois que j'exécute cette commande, elle me présente les différentes étapes à suivre pour générer un CRUD.



Pour vérifier si le CRUD est fonctionnel dans le dashboard, j'insère des données en utilisant des commandes SQL depuis mon terminal. Cela me permet de voir si les données sont correctement gérées et affichées dans le CRUD du dashboard.



```
mysql> show databases;
+--------------------------------+
| Database                       |
+--------------------------------+
| berenger_cinema_db             |
| berenger_cuisinea_studi_live   |
| berenger_ecf-quai-antique      |
| berenger_moto_clubb            |
| berenger_symfony_yourbook      |
| information_schema             |
+--------------------------------+
6 rows in set (0,04 sec)
```

```
mysql> show tables;
+-----------------------------------+
| Tables_in_berenger_symfony_yourbook |
+-----------------------------------+
| adherent                          |
| auteur                            |
| doctrine_migration_versions       |
| editeur                           |
| emprunt                           |
| exemplaire                        |
| genre                             |
| livre                             |
| livre_genre                       |
| messenger_messages                |
| stock                             |
| usure                             |
+-----------------------------------+
12 rows in set (0,03 sec)
```

```
mysql> INSERT INTO `genre` (`id`, `nom`) VALUES (NULL, 'policier');
Query OK, 1 row affected (0,14 sec)
```

```
mysql> SELECT * FROM genre;
+----+-----------+
| id | nom       |
+----+-----------+
|  3 | policier  |
+----+-----------+
1 row in set (0,03 sec)
```

Dans le fichier « DashboardController.php », j'utilise la méthode **configureMenuItems()** pour configurer le menu. La fonction **yield** permet de retourner plusieurs éléments à la fois.

```php
public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa-solid fa-house-user');
    yield MenuItem::linkToCrud('Genre', 'fas fa-list', Genre::class);
}
```

En utilisant la classe « AbstractCrudController », celle-ci analyse tous les champs présents dans mon entité et est capable de déduire automatiquement leur type, que ce soit une chaîne de caractères, un entier, etc.

```php
<?php

namespace App\Controller\Admin;

use App\Entity\Genre;
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;

class GenreCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Genre::class;
    }
```

src/Controller/admin/GenreCrudController.php

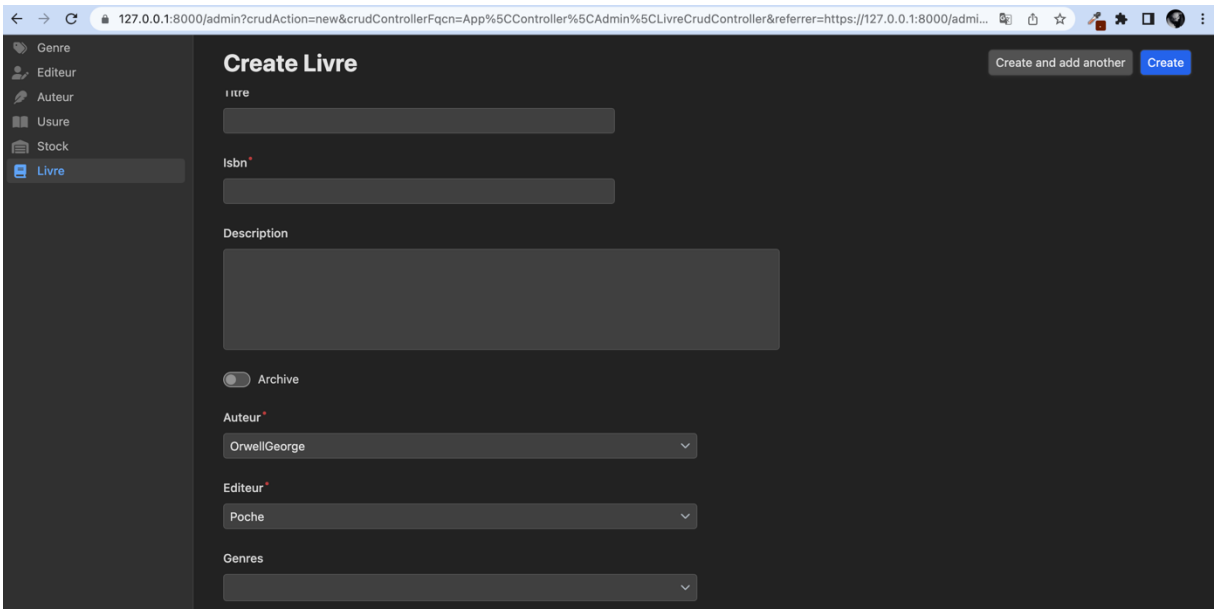
Dans la méthode **configureFields()**, lorsque j'ai des entités avec des relations vers d'autres entités, j'appelle d'abord la méthode parent pour configurer les champs existants. Ensuite, j'ajoute les champs d'association manquants en utilisant le type d'association approprié. De plus, j'implémente une méthode magique **__toString()** dans chaque entité associée pour spécifier comment ces entités doivent être affichées dans le CRUD. Cette approche me permet de gérer efficacement les relations entre les entités et de contrôler leur affichage dans l'interface EasyAdmin.

```php
class LivreCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Livre::class;
    }


    public function configureFields(string $pageName): iterable
    {
        yield from parent::configureFields($pageName);
        yield AssociationField::new('auteur');
        yield AssociationField::new('editeur');
        yield AssociationField::new('genres');
    }

}
```

src/Controller/admin/LivreCrudController.php

```php
public function __toString()
{
    return $this->getNom(). '' .$this->getPrenom();
}
```

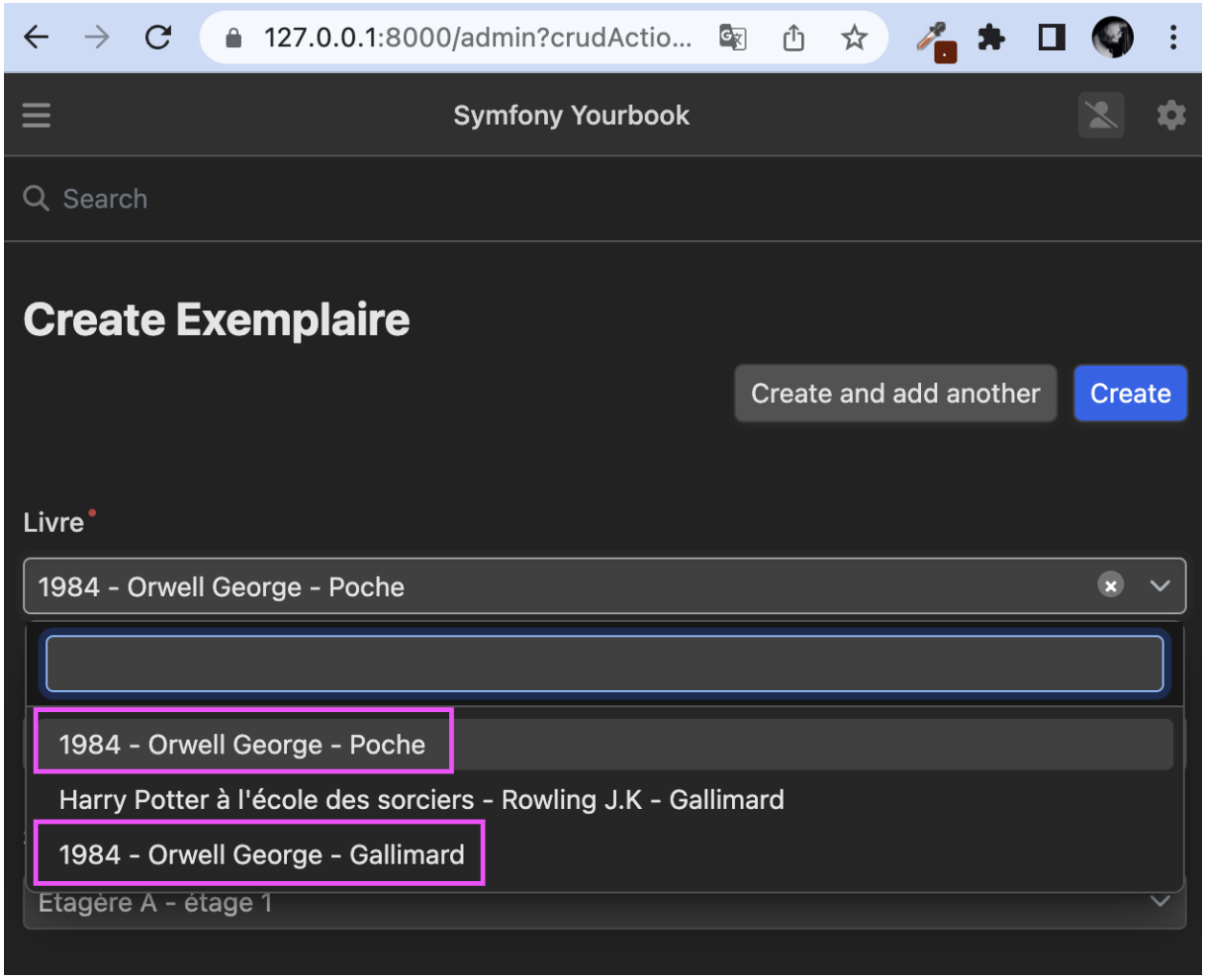
src/Entity/Auteur.php



Afin de mieux distinguer les livres des auteurs dans le CRUD « Exemplaire », j'ajoute des champs d'association dans le fichier « ExemplaireCrudController.php ». Pour faciliter l'affichage des informations des livres, j'ai également implémenté la méthode magique **__toString()** dans le fichier « Livre.php » de l'entité « Livre ». Cette méthode effectue une concaténation du titre du livre, du nom de son auteur et de l'éditeur, permettant ainsi une représentation claire et concise des livres qui aurait un même titre dans l'interface.

```php
public function __toString()
{
    return $this->getTitre(). ' - ' .$this->getAuteur(). ' - ' .$this->getEditeur();
}
```

src/Entity/Livre.php



Dans le fichier « Emprunt.php », j'ajoute une méthode **getDatePrevisionnelle()** pour obtenir une date prévisionnelle. Cette méthode n'est pas incluse dans l'ORM. Son fonctionnement consiste à récupérer la date d'emprunt, puis à la convertir en objet « DateTime » en utilisant la méthode **modify()**. Cette conversion est nécessaire car je ne peux pas modifier directement une date dans un objet **DateTimeInterface**. J'initialise la date d'emprunt à « null » et j'ajoute une condition pour effectuer les calculs nécessaires.

```php
public function getDatePrevisionnelle(): ?\DateTimeInterface
{
    $datePrevisionnelle = null;
    if ($this->getDateEmprunt()){
        $datePrevisionnelle = DateTime::createFromInterface($this->getDateEmprunt());
        $datePrevisionnelle->modify('+20days');
    }
    return $datePrevisionnelle;
}
```

src/Entity/Emprunt.php

Pour sécuriser l'interface d'administration, j'ajoute un « login » à l'aide de la commande suivante :

symfony console make:user Admin

Symfony génère automatiquement une entité Admin pour moi. Je procède ensuite aux migrations. Pour me connecter en tant qu'administrateur, j'ajoute une adresse e-mail et un mot de passe pour un administrateur. Avant cela, j'exécute une commande qui me permet de chiffrer (hasher) le mot de passe.

symfony console security:hash-password

Ensuite, Je génère l'authentification de l'utilisateur avec la commande. :

symfony console make:auth

Quand j'essaie d'accéder à la route « /admin » depuis mon navigateur web, je suis automatiquement redirigé vers la route « /login ». Je dois m'authentifier pour pouvoir accéder au tableau de bord de l'interface d'administration.

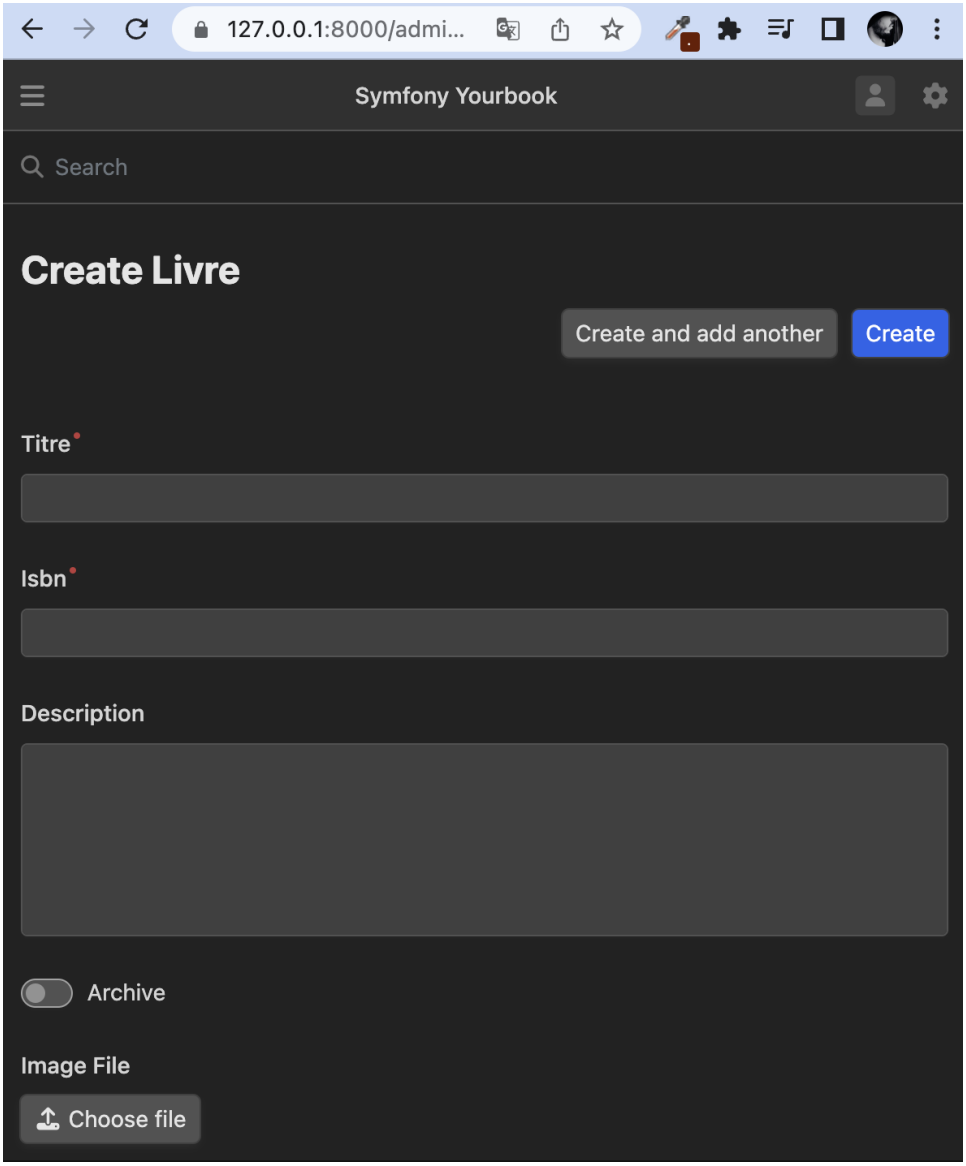
J'ajoute le bundle VichUploader en utilisant la commande suivante pour faciliter le téléchargement des images. :

composer require vich/uploader-bundle

Après avoir consulté la documentation de VichUploader, j'ajoute les getters et setters requis à l'entité « Livre.php ». Ensuite, je procède à l'exécution des migrations. Dans le fichier « LivreCrudController.php », j''ajoute un champ **TextField** pour le titre du livre, suivi d'un autre champ **TextField** pour l'ISBN. Pour la description du livre, j'utilise un champ **TextareaField**. Ensuite, j'ajoute un champ **BooleanField** pour indiquer si le livre est archivé. Pour gérer le téléchargement d'images, j'utilise un champ **TextareaField** avec **VichImageType**, masqué sur la page d'index. Enfin, j'ajoute un champ **ImageField** nommé « imageName » pour afficher l'image associée au livre, masqué sur le formulaire.

```php
public function configureFields(string $pageName): iterable
{
    //yield from parent::configureFields($pageName);

    yield TextField::new('titre');
    yield TextField::new('isbn');
    yield TextareaField::new('description');

    yield BooleanField::new('archive');

    yield TextareaField::new('imageFile')->setFormType(VichImageType::class)->hideOnIndex();

    yield ImageField::new('imageName')->setBasePath('/images/livres')->hideOnForm();

    yield AssociationField::new('auteur');
    yield AssociationField::new('editeur');
    yield AssociationField::new('genres');
}
```

src/Controller/admin/LivreCrudController.php



Une fois que l'image est téléchargée, elle apparaît bien dans l'interface d'administration dans le navigateur web.

https://127.0.0.1:8000/admin?crudAction=index&crudControllerFqcn=App%5CController%5CAdmin%5CLivreCrudController

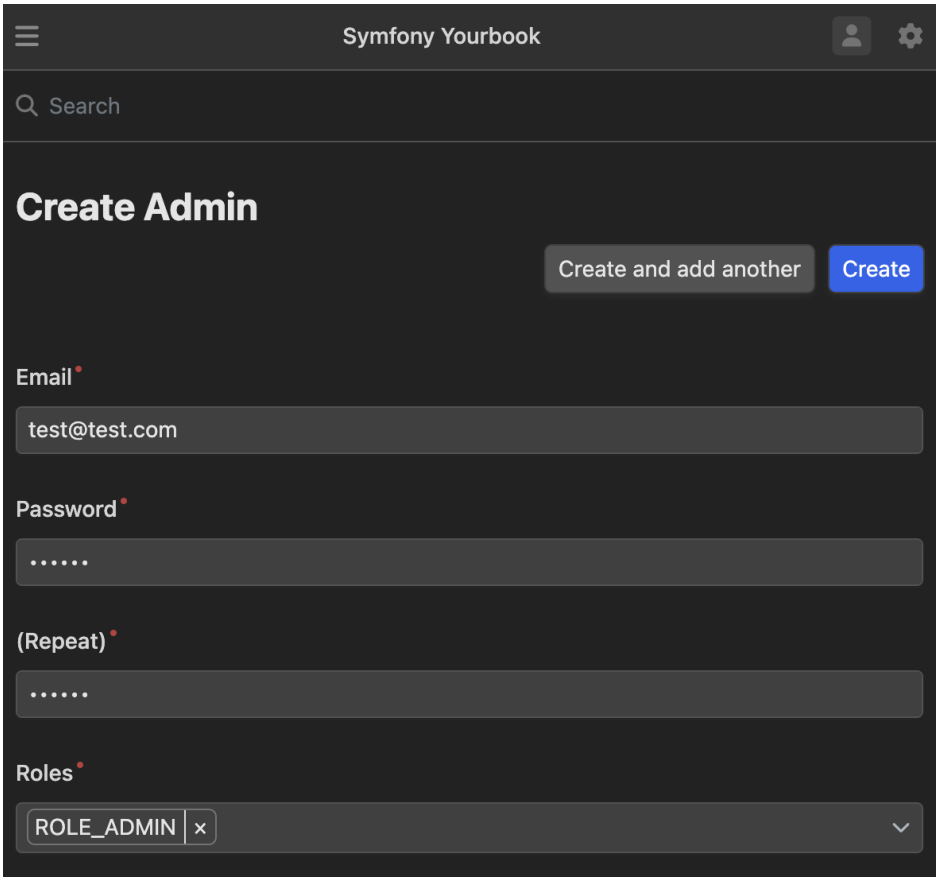
Pour administrer et ajouter des administrateurs avec des rôles attribués et des mots de passe sécurisés (hachés), j'utilise le code fourni dans la documentation Symfony. Je copie et colle le code donné pour l'ajout d'un « Hashing Password » dans le « AdminCrudController.php ».

Je rajoute le code suivant qui crée un champ de choix multiple appelé « roles ». Les choix possibles sont définis dans le tableau **$roles** avec les valeurs 'ROLE_ADMIN' et 'ROLE_USER'.

J'utilise la classe « ChoiceField » pour le champ « roles » et je le configure avec les choix provenant du tableau **$roles**. J'ajoute également l'option **renderAsBadges()** pour afficher les choix sous forme de badges.

```php
$fields[] = $password;

$roles = ['ROLE_ADMIN', 'ROLE_USER'];

$fields[] = ChoiceField::new('roles')
    ->setChoices(array_combine($roles, $roles))
    ->allowMultipleChoices()
    ->renderAsBadges();
```

src/Controller/admin/AdminCrudController.php



Une fois l'administrateur créé, je consulte la base de données pour vérifier si le mot de passe est bien crypté (hash) et que le rôle est bien attribué.

	id	email	roles (DC2Type:json)	password
Edit Copy Delete	1	admin@test.com	["ROLE_ADMIN"]	$2y$13$WRVm1DSjf.4v6X3bQicmz.cQNJqNW1Umy8aYBEm7ePA...
Edit Copy Delete	2	test@test.com	["ROLE_ADMIN"]	$2y$13$4EWrk7r33X9/uzOG9oAnvOKGCHMFCypHFx0i.ivXrE9...

https://phpmyadmin.alwaysdata.com/phpmyadmin/index.php?route=/sql&server=1&db=berenger_symfony_yourbook&table=admin&pos=0

Pour permettre aux utilisateurs d'ajouter un livre depuis la partie Front Office de l'application, j'utilise la commande suivante pour générer un CRUD :

symfony console make:crud

Cette commande me permet de créer facilement les opérations CRUD (Create, Read, Update, Delete) pour l'entité « Livre ». Ainsi, les utilisateurs peuvent ajouter, consulter, mettre à jour et supprimer des livres à partir de l'interface Front Office de mon application.

3. Avec qui avez-vous travaillé ?

J'ai travaillé seul, mais j'ai pu compter sur le soutien du formateur tout au long des sessions en direct pour m'aider à mener à bien ce projet Symfony inter-spécialités sur le back office.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Studi*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *04/04/2023* au : *04/05/2023*

5. Informations complémentaires *(facultatif)*

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
BAC professionnelle Commerce	Lycée privé Jeanne La Lorraine, Le Raincy	16/06/2008
BEP Métier de la comptabilité	Lycée privé Jeanne La Lorraine, Le Raincy	16/06/2006

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] FERGUENIS Bérenger,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à Ploufragan le 14/06/2023

pour faire valoir ce que de droit.

Signature :

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
https://www.php.net/
https://symfony.com/doc/current/index.html
https://dictionaryapi.dev/
https://www.npmjs.com/
https://stackoverflow.com/

ANNEXES

(Si le RC le prévoit)