Plateforme E-LEARNING

Knowledge Learning

KNOWLEDGE

Réalisation d'un site pour le compte de la société Knowledge afin de proposer différents cours sur une plateforme en ligne.

Projet réalisé dans le cadre de la validation du niveau 3 du titre Professionnel Développeur Web et Web Mobil.

SOMMAIRE

1 - Présentation du projet

2 - La base de donnée

3 - Extraits de code

4 - Sécurité

5 - Validation du site

1 - Présentation du projet:

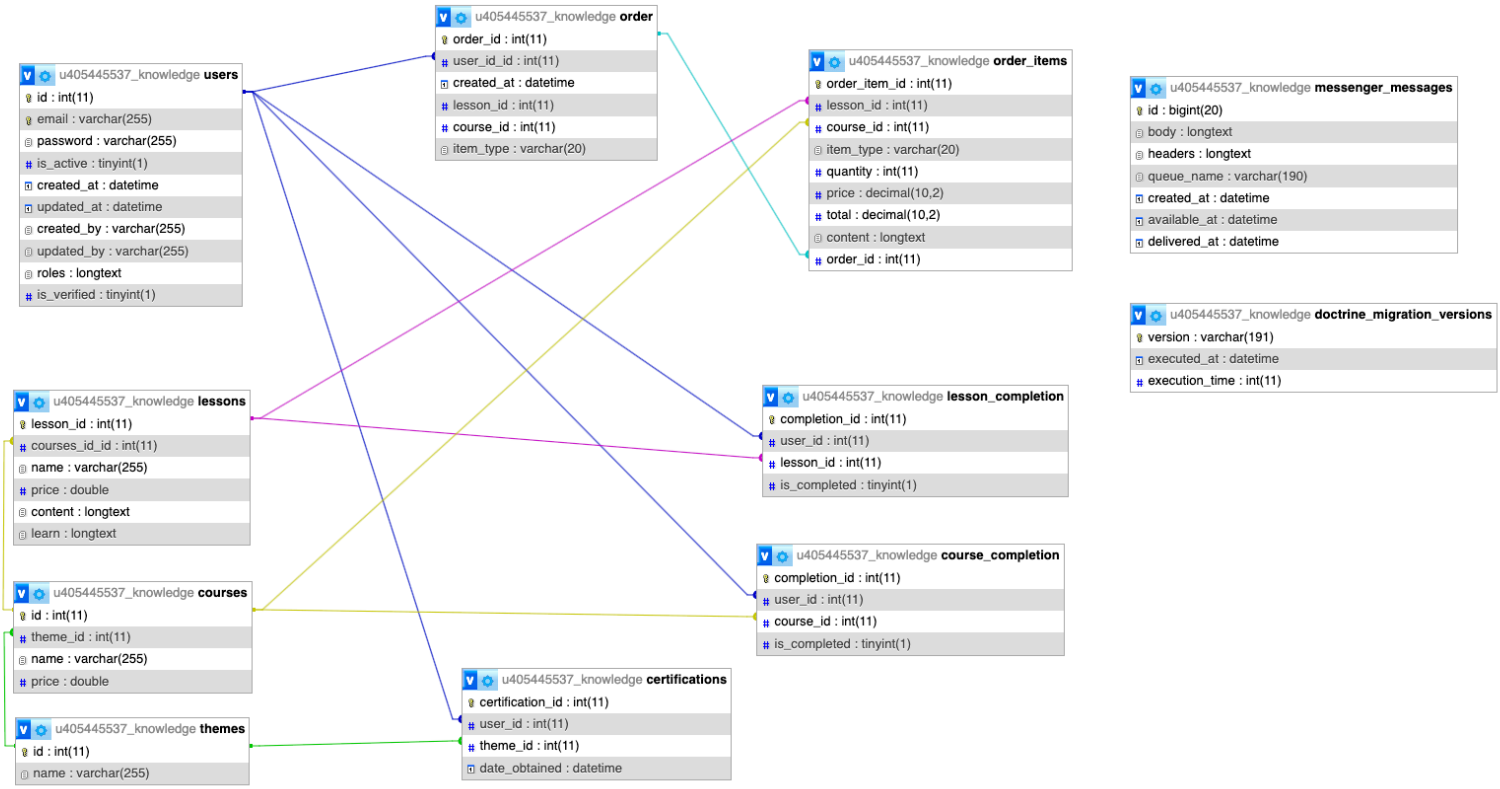
L'entreprise Knowledge, connue pour ses supports de formation vendus en librairie, m'a contacté pour la création de leur site de e-learning afin de proposer à la vente leurs différents cours et d'en permettre leurs suivis.

L'objectif est de permettre au visiteur d'acheter diverses formations, directement par cours complet ou juste la ou les leçon(s) qui l'intéressent.

Le visiteur, une fois inscrit sur la plateforme a accès à un magasin qui lui permet d'acheter un cours complet ou une leçon, une fois acheté il peut suivre les enseignements et enregistrer sa progression une fois qu'il a fini.

La conception du projet a nécessité la création d'une base de données qui permet de gérer le contenu proposé pour le client, et d'enregistrer les achats et la progression des visiteurs.

2 - La base de données



J'ai choisi de concevoir la base de données en utilisant la méthode Merise, car elle permet de garder chaque entité bien définie et de structurer efficacement les relations entre elles. En me basant sur cette méthode, j'ai pu organiser les données de manière claire et logique, en m'assurant que chaque élément reste distinct, tout en intégrant uniquement les informations nécessaires à chaque fonctionnalité de l'application. Cette approche m'a permis de représenter précisément les dépendances entre les entités, ce qui facilite la gestion des données et l'évolution du projet sur le long terme.

De plus, la méthode Merise offre une vision globale et claire du modèle de données avant même de commencer à coder, ce qui est un avantage précieux dans la phase de conception. Par exemple, j'ai pu identifier rapidement quelles entités avaient besoin de communiquer entre elles et comment les relier, ce qui a simplifié la création des tables.

Merise m'a permis de réduire les erreurs de conception.

3 - Extraits de code

Le header

```twig
<div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav ms-auto">
        {% if app.user %}
            <li class="nav-item">
                <a class="nav-link" href="{{ path('home') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('shop_index') }}">Cours</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('completion_list') }}">Avancement</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('cart_view') }}">Panier</a>
            </li>
            {% if is_granted('ROLE_ADMIN') %}
                <li class="nav-item">
                    <a class="nav-link" href="{{ path('admin_dashboard') }}">Back-office</a>
                </li>
            {% endif %}
            <li class="nav-item">
                <a class="nav-link" href="{{ path('logout') }}">Se déconnecter</a>
            </li>
        {% else %}
            <li class="nav-item">
                <a class="nav-link" href="{{ path('home') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('register') }}">S'inscrire</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('login') }}">Se connecter</a>
            </li>
        {% endif %}
    </ul>
</div>
```

Dans ce projet, j'ai choisi de faire une barre de navigation dynamique grâce à l'utilisation de Symfony et Twig.

Selon que l'utilisateur soit connecté ou non, le menu affiche des liens différents.

Les administrateur voient également un lien vers le back-office pour la gestion du contenu du site.

Si l'utilisateur n'est pas connecté, seuls les liens pour s'inscrire ou se connecter sont visibles.

Ce système de navigation garantie une fluidité et une sécurité d'accès adaptée aux différents types d'utilisateur.

La boutique

```php
#[Route('/shop', name: 'shop_index')]
public function index(CoursesRepository $coursesRepo, LessonsRepository $lessonsRepo, EntityManagerInterface $entityManager): Response {

    if (!$this->isGranted('ROLE_CLIENT') && !$this->isGranted('ROLE_ADMIN')) {
        throw $this->createAccessDeniedException('Vous n\'avez pas les droits nécessaires pour accéder à cette page.');
    }

    // Retrieve logged in user
    $user = $this->getUser();

    // Collect all courses and lessons
    $courses = $coursesRepo->findAll();
    $lessons = $lessonsRepo->findAll();

    // Retrieve user commands
    $orderRepo = $entityManager->getRepository(Order::class);
    $purchasedItems = $orderRepo->findBy(['user_id' => $user]);

    // Extract IDs from purchased courses and lessons
    $purchasedCourses = [];
    $purchasedLessons = [];
    foreach ($purchasedItems as $item) {
        if ($item->getItemType() === 'course' && $item->getCourseId()) {
            $purchasedCourses[] = $item->getCourseId();
        } elseif ($item->getItemType() === 'lesson' && $item->getLessonId()) {
            $purchasedLessons[] = $item->getLessonId();
        }
    }

    return $this->render('shop/shop.html.twig', [
        'courses' => $courses,
        'lessons' => $lessons,
        'purchasedCourses' => $purchasedCourses,
        'purchasedLessons' => $purchasedLessons,
        'userRoles' => $this->getUser()->getRoles(),
    ]);
}
```

```
<h5 class="card-title">{{ course.name }}</h5>
<p><strong>Prix du cursus : {{ course.price }}€</strong></p>

<a href="{{
    'ROLE_ADMIN' in userRoles or (course.id in purchasedCourses)
    ? path('follow_course', { 'id': course.id })
    : path('shop_add_to_cart', { 'type': 'course', 'id': course.id })
}}"
class="btn {{
    'ROLE_ADMIN' in userRoles or (course.id in purchasedCourses)
    ? 'btn-follow'
    : 'btn-primary'
}}">
    {{ 'ROLE_ADMIN' in userRoles or (course.id in purchasedCourses)
        ? 'Suivre le Cursus'
        : 'Acheter le Cursus'
    }}
</a>
```

La page de la boutique (sous la route /shop) permet à l'utilisateur de consulter l'ensemble des cours et leçons proposés.

Lorsque l'utilisateur accède à cette page, la fonction « index » vérifie en premier lieux s'il dispose des droits nécessaires pour visualiser le contenu (si l'utilisateur a le rôle ROLE_CLIENT ou ROLE_ADMIN).

Si l'utilisateur n'est pas connecté ou n'a pas les droits requis, l'accès à la page lui est interdit.

Je récupère les données via les repositories CoursesRepository et LessonsRepository, ce qui permet d'afficher l'intégralité des cours et leçons dans la boutique.

Ensuite, le contrôleur vérifie les commandes déjà validées par l'utilisateur en interrogeant la base de données pour retrouver les achats effectués. Les IDs des éléments achetés sont extraits et transmis à la vue.

Dans la vue, l'utilisateur voit une liste des cours et leçons avec des boutons d'action adaptés.

Si l'utilisateur a déjà acheté l'élément, celui-ci voit le bouton « acheter » se transformer en « suivre » qui lui permet d'accéder à l'élément en question.

Pour les administrateurs, la vue est directement adaptée pour leur permettre d'accéder à l'affichage de tous les cours et leçons.

L'avancement

```php
#[Route('/completions', name: 'completion_list')]
public function completion_list(LessonCompletionRepository $lessonCompletionRepository, CourseCompletionRepository $courseCompletionRepository): Response
{
    if (!$this->isGranted('IS_AUTHENTICATED_FULLY')) {
        throw $this->createAccessDeniedException('Vous devez être connecté pour accéder à cette page.');
    }

    $courseCompletions = [];
    $lessonCompletions = [];
    if ($this->isGranted('ROLE_ADMIN')) {
        // If user is admin, load data for all users
        $courseCompletions = $courseCompletionRepository->findAll();
        $lessonCompletions = $lessonCompletionRepository->findAll();

        return $this->render('admin/completion_users.html.twig', [
            'courseCompletions' => $courseCompletions,
            'lessonCompletions' => $lessonCompletions,
        ]);
    } elseif ($this->isGranted('ROLE_CLIENT')) {
        // If the user is a client user, load their own data
        $user = $this->getUser();
        $courseCompletions = $courseCompletionRepository->findBy(['user' => $user]);
        $lessonCompletions = $lessonCompletionRepository->findBy(['user' => $user]);
    }

    return $this->render('completion/completion.html.twig', [
        'courseCompletions' => $courseCompletions,
        'lessonCompletions' => $lessonCompletions,
    ]);
}
```

La route « /completions » permet à l'utilisateur de consulter son avancement dans les cours et leçons. Suivant l'utilisateur connecté, le contrôleur vérifie son rôle: un client peut uniquement voir ses propres avancements, tandis qu'un administrateur peut accéder aux informations de tous les utilisateurs.

Le contrôleur interroge les repositories LessonCompletionRepository et CourseCompletionRepository pour récupérer les données d'avancement.

Ces informations sont ensuite envoyées à la vue. L'administrateur voit toutes les données via la vue ***completion_users.html.twig***, tandis que le client consulte ses propres progrès via la vue ***completion.html.twig***.

Ce contrôleur permet à lui seul de gérer l'affichage de 2 templates différents suivant le rôle de l'utilisateur (ROLE_CLIENT et ROLE_ADMIN).

4 - Sécurité

```php
if (!$this->isGranted('ROLE_CLIENT') && !$this->isGranted('ROLE_ADMIN')) {
    throw $this->createAccessDeniedException('Vous n\'avez pas les droits nécessaires pour accéder à cette page.');
}
```

Dans chaque contrôleur nécessitant un utilisateur connecté, j'ai ajouté cette condition qui vérifie que l'utilisateur connecté possède l'un des rôles requis, soit ROLE_CLIENT, soit ROLE_ADMIN. Si ce n'est pas le cas, l'accès à la page est refusé.

```php
$user = $this->getUser();
if (!$user) {
    return $this->redirectToRoute('login');
}
```

Pour compléter cette sécurité d'accès, j'ai ajouté une redirection si aucun utilisateur n'est connecté vers la page de connexion.

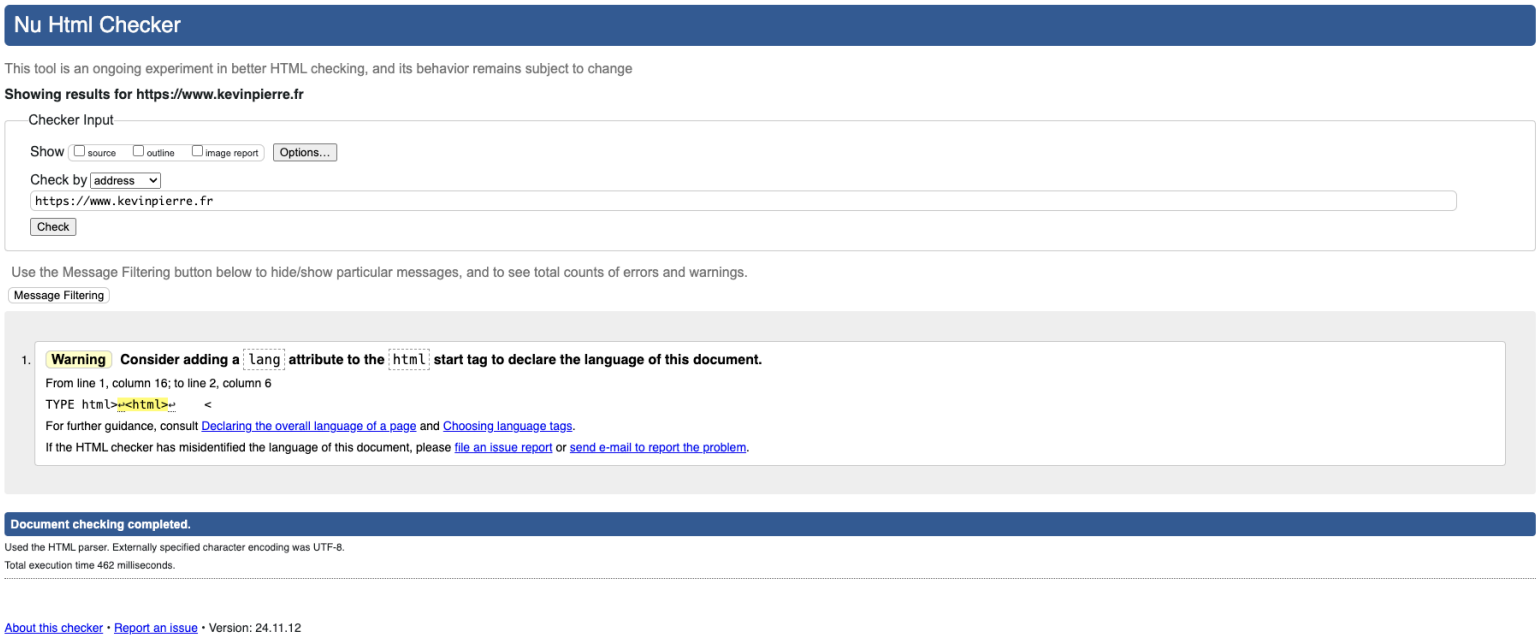
```php
#[IsGranted('ROLE_ADMIN')]
class AdminController extends AbstractController
{
    private $entityManager;

    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    #[Route('/admin', name: 'admin_dashboard')]
    public function index(CoursesRepository $coursesRepo, LessonsRepository $lessonsRepo): Response
    {
```

Pour tout le contrôleur qui nécessite une restriction ROLE_ADMIN, j'ai choisi une sécurité à haut niveau en utilisant l'annotation #[IsGranted('ROLE_ADMIN)]

5 - Validation du site



J'ai passé l'URL de mon site une fois hébergé chez Hostiguer au validateur du site https://www.w3.org/ qui me ressort un warning pour la déclaration de la langue utilisée sur le site, pourtant, comme le voici dans l'image ci-dessous (une capture d'écran du code directement chez mon hébergeur) la langue utilisée est bien présente.

base.html.twig

public_html > templates > base.html.twig

```
1  <!DOCTYPE html>
2  <html lang="fr">
3      <head>
4          <meta charset="UTF-8">
5          <meta name="description" content="Knowledge Learning - Apprenez facilement avec nos
6          <meta name="author" content="Kévin PIERRE">
7          <meta name="robots" content="index, follow">
8          <title>{% block title %}Knowledge{% endblock %}</title>
9          <link rel="icon" href="{{ asset('assets/images/favicon.png') }}" type="image/png">
10         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.
11         {% block stylesheets %}
12             <link rel="stylesheet" href="{{ asset('assets/styles/base.css') }}">
13         {% endblock %}
14
15         {% block javascripts %}
16             {% block importmap %}{{ importmap('app') }}{% endblock %}
17         {% endblock %}
18     </head>
```