Je m’appelle Laury, actuellement en reconversion professionnelle chez Elan formation. Je suis présente devant vous pour obtenir le titre de développeur web et web mobile.

Je vais vous présenter mon projet, les fonctionnalités mise en place et les technologies utilisées. Ma gestion de projet, la phase de conception, la maquette et les choix concernant l’interface utilisateur et son expérience. L’architecture logicielle et la sécurité.

1. Présentation du projet et benchmark

Mon projet fait suite de la découverte d’expositions éphémères de dinosaures et de discussion sur l’Histoire en général. C’est une plateforme de e-commerce permettant d’acheter des billets pour des expositions culturelles sur le thème général de la guerre. Elle est gérée par une association possédant un local de 200m2 pour les expositions.

J’ai consulté divers sites culturels que ce soit des musées, des centres d’interprétations ou tout simplement des lieux d’expositions.

2 Sites ont relevé mon attention :

* Musée Beyerler de Bale : J’ai apprécié la facilité d’achats des tickets qui respecte la loi de Fitts
* Hudson Film Festival qui a reçu une récompense en 2023 : J’ai apprécié son design et m’en suis inspirée en l’accordant avec le thème et le public visé.

2. Fonctionnalités et technos utilisées

Les fonctionnalités mises en place sont la gestion d’expositions dynamique, la commande, le panier, le paiement. J’ai également mis en place un back-office.

Pour mettre en place ces fonctionnalités, j’ai opté de travailler avec le framework Symfony, basé sur PHP, et ses composants comme Twig pour la vue. La communication avec ma base de données se fera via Doctrine, un ORM qui traduit les objets PHP en requêtes SQL

3. Gestion de Projet

Afin de gérer mon projet de manière flexible, j’ai utilisé Trello qui grâce à son système de cartes m’a permis de mettre en place des sprints allant de 2j (gestion des exposition) à 2 semaines (panier).

L’utilisation de github m’a également permis de m’organiser en me permettant de garder propre ma branche principale tout en explorant et testant des fonctionnalités sur des branches séparées.

4. MCL – MLD

Le modèle conceptuel de données m’a permis de mettre en évidence les différentes entités pour mon projet. Ici exhibition est l’entité centrale, elle forme une ternaire en ManytoMany avec les entités artist et room.

Et à droite, bien que l’entité Invoice soit crée au moment de la validation du paiement de la commande. Elle est indépendante car elle présente un état figé dans le temps de cette transaction. Cela respecte le droit commercial et sépare les responsabilités.

Les associations comme Show deviendront une table dans le modèle logique de données. La table Show comprendra alors les clés étrangères correspondantes aux clés primaires de ces 3 tables. Mettant ainsi en évidence leur lien.

5. Maquette et choix UX UI

MOCKUP

La maquette est la réalisation concrète suite à l’analyse des besoins et à la phase conception. Consciente que la majorité des consultations web s'effectuent désormais sur smartphone, j’ai opté pour une approche en mobile first.

Choix

UX

De plus, en m’inspirant de la loi de Jakob, j’ai veillé à structurer de façon à garder les habitudes de l’utilisateur tout en mettant en place un style sobre et épuré, le rendant accessible à un public adulte et sénior tout en conservant une esthétique moderne susceptible d'attirer également un public adolescent. Le panier est également visible à tout moment.

UI

Respect des contrastes, taille de police ajustables ?

SEO

Une structure sémantique et hiérarchique optimisée améliore l'indexation des pages par les moteurs de recherche.

En parallèle, l'utilisation d'attributs aria-label, à l'instar de l'élément title, rend le contenu accessible aux personnes utilisant des lecteurs d'écran.

Et la mise en place de slugs contribue à des URLs user friendly.

6. Architecture Logicielle

Symfony permet d’utiliser le design pattern mvc.

* Diapo 1

Ici nous sommes sur la **vue** du client, c’est la page d’exposition qui permet de choisir directement les tickets à commander.

Quand l’utilisateur souhaite ajouter un ticket, cela envoie une requete http au **controller** qui active la méthode addTicketToCart.

-diapo 2 :

Le **controller** interroge le model pour recupèrer le ticket souhaité via le repository. Puis appelle la méthode addCart du CartService qui permet d’ajouter le ticket à la session du panier avec toutes les informations nécessaires via un tableau multi dimensionnel.

Une fois que l’utilisateur a ajouté tous les tickets souhaités au panier il peut alors accèder à son panier soit par la page d’exposition soit par la navbar.

* Diapo 3

stripeCheckout fait appel au **model** pour vérifier le stock des tickets. Pour cela, foreach boucle sur les quantités du panier. Si la quantité demandée est négative par rapport au stock maximal alors l’information est stockée dans un tableau associatif d’erreurs. A l’issue, une notification est affichée à l’utilisateur avec le nom de l’exposition et le nombre de tickets restants.

Si la boucle foreach n’enregistre aucune erreur alors les informations de la commande sont préparées pour envoi à Stripe. Pour ce faire, une boucle foreach parcourt le panier à l’aide du repository et rempli le tableau de données (exposition, ticket, prix en centimes, quantité, devise).

Ensuite je configure de la clé API de Stripe pour l’authentification ainsi que les urls de sorties.

La création d’une session de paiement avec les informations relatives à la transaction (email, type de paiement, le tableau du panier, les urls de sorties.

Puis on sort du **model** MVC pour aller sur Stripe

* Diapo 4

Si le paiement est validé Stripe redirige vers l’url de succès et donc la méthode stripeSuccess qui en passant de nouveau par le **Model** instancie une nouvelle commande. Elle prépare les informations à enregistrer avec les informations de l’utilisateur, le contenu du panier grace à la boucle. Puis le **model** gère la préparation et la sauvegarde des données en base de données. Il fait de même pour l’entité facture.

7. Sécurité

**Faille xss :**

Mise en place de contraintes lors de la saisie du formulaire.

Twig échappe automatiquement les données {{ maVariable }} en sortie.

(natif : filtres de validation (filter\_var) en entrée et htmlspecialchars en sortie)

**Faille csrf**: **Cross-Site Request Forgery**

Symfony utilise un token unique inclus à chaque formulaire (form\_start)

**Attaque par force brute et par dictionnaire :**

Prise de l’empreinte numérique du mot de passe et ajout d’une regex

Empreinte :

Nom de l’algorithme : Ici, il s’agit de bcrypt.

Le coût : Représente la complexité du calcul (2^10).

Le processus de salage (*salt*) : ajoute une valeur aléatoire et unique avant le hachage du mot de passe.

La dernière partie : représente le mot de passe transformé de manière irréversible. Il est mélangé a des lettres, chiffres et caractères spéciaux.

Détail de la regex :

Le mot de passe doit contenir au moins :

(A-Z) : Une lettre majuscule.

(\d) : Un chiffre.

([\W\_]) : Un caractère spécial.

{12,} : Douze caractères minimum.

**Injection sql**

Symfony utilise Doctrine pour la bdd et prépare les requêtes par le biais de requête paramétrées.

Ce code Doctrine QueryBuilder construit une requête pour sélectionner toutes les expositions (e) à partir de la table associée à l'entité Exhibition. Il filtre les résultats pour ne retenir que les expositions dont la date (dateExhibit) est postérieure à la date et l'heure actuelles, et les ordonne par date d'exposition croissante.

En SQL, cela se traduirait par une sélection de toutes les colonnes de la table exhibitions où la colonne date\_exhibit est supérieure à la date actuelle, triée par ordre croissant de date\_exhibit.

Le setParameter() en DQL permet de lier de manière sécurisée la valeur de la date et l'heure actuelles au paramètre nommé utilisé dans la clause WHERE.

8. Axes d’améliorations

Afin de respecter en totalité le rgpd sans faille humaine, il serait intéressant de mettre en place un planificateur pour anonymiser les artistes le souhaitant après leur dernière participation à une exposition. Le composant Messenger de Symfony conviendrait avec la mise en place d’un dossier Command pour l’automatisation.

Afin de pouvoir déployer cet e-commerce, il faudrait pouvoir mettre en place des tests unitaires par le biais de PHPUnit et l’intégrer lors d’une pipeline CI telle que gitlab CI.