Je m’appelle Laury, actuellement en reconversion professionnelle. Je suis présente devant vous pour vous présenter mon projet. Pour cela, je vais vous présenter les différentes fonctionnalités et technologies. La gestion de projet mise en place la modélisation de données, la maquette et les choix effectués, l’architecture logicielle via le mvc puis la fonctionnalité phare et nous continuerons sur la sécurité que j’ai mise en place et nous finirons sur les axes d’améliorations.

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

L’utilisateur peut consulter toutes les pages publiques comme celles des expositions, page à propos ou encore la politique de confidentialité. Il peut ajouter des tickets au panier, se créer un compte ou se connecter en respectant le principe de minimisation des données personnelles et bien sûr contacter nos services.

L’utilisateur connecté peut effectuer une commande, exercer son droit à modification ou encore son droit à l’oubli en modifiant son profil. Accéder à son historique de commande et télécharger ses factures.

J’ai également mis en place un back office afin que les administrateurs puissent effectuer une gestion des utilisateurs, expositions des factures clients dans le respect du droit commercial et celle des stocks.

Root actuellement gère le rôle des administrateurs.

Pour mettre en place ces fonctionnalités, j’ai opté pour le framework Symfony. J’ai utilisé le système de gestion de base de données HeidiSql avec le langage sql. La communication entre les les deux se fait par Doctrine l orm de symfony.

3. Gestion de Projet

Afin de gérer mon projet de manière flexible, j’ai opté pour une méthodologie Agile pour cela 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 expositions) à 2 semaines (panier).

J’ai utilisé la méthode kanban avec les différentes colonnes : A faire – En cours – Terminé

J’ai également utilisé la méthode Moscow pour définir les ordres de priorité Must have pour représenter le minimum viable product. Le should have en orange pour ce qui devrait avoir, le could have en vert pour ce qui pourrait avoir. Et le won’t have en gris pour les fonctionnalités qui pourront être réalisées ultérieurement.

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

Concernant la modélisation des données, j’ai opté pour l’utilisation de la méthode Merise qui permet une analyse structurée et rigoureuse de mes données.

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 créé 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

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. Menu burger en haut a droite, logo a son opposé cliquable. Agenda dans la page, plan pour acceder au local. Footer en pied de page avec les différents menus textuels. Le panier est également visible à tout moment.

UI

L’aspect du site a facilité le respect des contrastes comme j’ai pu le constater sur Whocanuse. Ce sont les 4 couleurs utilisées pour le site.

SEO

J’ai suivi les normes du rgaa (référentiel général d’amélioration de l accessibilité) basées sur le wcag (Web content accessibilité guidelines)

Ainsi j’ai respecté une structure sémantique et hiérarchique optimisée pour améliorer l'indexation des pages par les moteurs de recherche. Un respect de l’ordre des balises h1 h2 par exemple.

J’ai utilisé les attributs aria-label et title pour rendre certaines icones et boutons plus accessibles.

J’ai également mis en place des slugs pour avoir des urls plus conviviale.

6. Architecture Logicielle

Symfony permet d’utiliser le design pattern mvc.

Pour illustrer mes propos, nous allons prendre les dernières expositions programmées sur la page home.

Lorsque l’utilisateur arrive sur la page home, une requête HTTP est envoyée. Elle est réceptionnée par le controller frontal puis passe par le noyau de Symfony qui l’envoie dans le système de routing afin d identifier la route et donc le bon controller. Le controller exécutera la méthode index. Ici il s agit du home controller dans lequel, j’ai injecté la dépendance ExhibitionShareRepo

A l’intérieur de celle-ci, il fera appel au model qui se charge d’appliquer la requête SQL auprès de la base de données et d’y récupérer les 4 dernières expositions.

Le model renvoie le résultat de la requête au controller qui lui envoie également une reponse html à la vue et permet d’afficher les dernieres expositions programmées.

7. Fonctionnalité phare

Diapo 6a

Quand l’utilisateur souhaite ajouter un ticket, il clique sur l icone du plus qui envoie une requete http controler qui va grâce a la route activer la méthode addTicketToCart.

-diapo 6b

Le **controller** interroge le **model** pour recupèrer le ticket souhaité via le repository. Une fois ces informations récupérées il 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 et effectuer le paiement.

Diapo 6c

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 associatif 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 6d

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. J’hydrate avec les informations blablabla. 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. Puis elle instancie une nouvelle facture.

7. Sécurité

J’ai suivi les recommandations de l’owaps afin de mettre en place des protections de sécurité contre les failles.

**Faille xss :**

Mise en place de contraintes lors de la saisie du formulaire. Ici, nous en avons 3 :

Obligation de rentrer le nom de l’artiste

Longueur maximale de 50 chars

Une régex acceptant que les lettres de différents types, des espaces et des tirets.

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)

Ce jeton est généralement inclus dans le formulaire sous forme d'un champ caché pour les formulaires de sécurité comme login.

Pour les autres, comme ici le formulaire de nom prénom dans le profile de l’utilisateur, il faut mettre en place une contrainte de token qui permet d’activer la protection CSRF.

Attaque par force brute et par dictionnaire :

ajout d’une regex

Prise de l’empreinte numérique du mot de passe coté serveur

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.

Ici c’est un exemple de regex pour un mot de passe qui respecte les recommandations de la cnil.

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 compilée pour trouver les commandes par utilisateur.

Elle selectionne toutes les commandes de l’entité Order. Elle fait une jointure avec les les entités orderdetail exhibition et ticket   
Ou la clause est l id de l utilisateur.

Le setparameter lie de manière sécurisé l’id de l’utilisateur

Le résultat est ordonné par ordre alphabétique des utilisateurs.

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.

D’un point de vue plus commercial, il serait intéressant de mettre en place un listing des personnes ayant acheté des billets pour un meilleur suivi le jour J et ainsi éviter les fraudes.

D’autre pour root, mettre en place, un dashboard avec des statistiques sur les ventes, le chiffre d’affaires par mois et des diagrammes pour que ça soit plus parlant.