

Table des matières

INTRODUCTION	3
Remerciements.....	3
Présentation personnelle.....	3
I Cahier des charges	3
1. Présentation du projet.....	3
2. Organisation du site	4
3. Champs d'action.....	5
4. Minimum Viable Product (MVP)	7
II Gestion de projet.....	8
1. Outils et méthodes.....	8
2. Langages	10
3. Technologies	11
4. Environnement	13
III Protection des données et sécurisation	14
1. Gestion des droits et des données personnelles.....	14
2. Sécurisation du code et des données	16
IV Search Engine Optimization	23
1. Structure du contenu et balisage HTML.....	23
2. Optimisation technique.....	24
3. Interactivité maîtrisée.....	26
V Conception du projet.....	26
1. Modélisation des données.....	26
2. Maquettage.....	27
3. Expérience utilisateur, interface et accessibilité.....	29
VI Architecture de l'application	32
1. Programmation Orientée Objet (POO).....	32
2. Requêtes http (HyperText Transfer Protocol)	34
3. Symfony 7 et son modèle MVC (Model, View, Controller)	36
VII Fonctionnalité phare	38
VIII Axes d'améliorations.....	58

1. Planificateur	58
2. Tests Unitaires	58
3. Blog	58
CONCLUSION	59
ANNEXES.....	60
Annexe 1 : Back-office.....	61
Annexe 2 : Trello.....	62
Annexe 3 : MCD	63
Annexe 4 : MLD.....	64
Annexe 5 : Maquettes mobile first.....	65
Annexe 6 : Maquettes desktop.....	66
Annexe 7 : Benchmark	67
Annexe 8 : Code de la commande	68

INTRODUCTION

Remerciements

Je souhaite remercier mon conjoint pour son soutien constant depuis mes premières réflexions autour de ma reconversion. Il m'a encouragée à suivre cette voie et a su adapter son emploi du temps en fonction des exigences de chaque formation, me permettant ainsi de me consacrer pleinement à ce projet.

Je remercie également ma fille pour sa patience et sa compréhension. Elle a dû s'adapter à mon absence plus fréquente, notamment les week-ends, et je lui suis reconnaissante pour cela.

Enfin, je tiens à remercier un apprenant, devenu un ami, et grâce à qui certains troubles dans mes connaissances techniques ont été rapidement dissipés. Ainsi que l'équipe pédagogique pour son accompagnement tout au long de cette formation. Leur disponibilité, leur écoute et leur professionnalisme ont grandement contribué à la qualité de ce parcours.

Je remercie également les membres du jury pour l'attention portée à la lecture de ce dossier.

Présentation personnelle

Mon intérêt pour l'informatique a débuté dès mon adolescence, que ce soit en stage de première lors de la création d'un logiciel avec Access, ou encore à travers mes cours d'informatique en BTS d'assistante de gestion où j'ai appris la modélisation de tables de données, entre autres.

Cependant, on m'a souvent dit que je pouvais, parfois, ne pas avoir assez de patience pour ce domaine. Manquant de confiance en moi, ces remarques m'ont fait prendre d'autres chemins. J'ai alors exploré d'autres univers tout aussi intéressants et variés. Ces expériences m'ont permis de développer une confiance en moi et une persévérance qui, aujourd'hui, me permettent de me reconvertis avec légitimité dans ce domaine.

Pour cela, j'ai suivi une formation de découverte du numérique puis une formation où j'ai acquis les bases de la programmation ce qui m'a permis d'intégrer sereinement le cursus actuel d'Elan Formation.

I Cahier des charges

1. Présentation du projet

Je me suis rendu compte que de nombreux conflits restent dans l'ombre, souvent parce qu'ils sont jugés peu historiques ou peu glorieux pour certains pays. Parfois, certains

événements sont même passés sous silence dans les cours d'histoire, alors que certains combats méritent pourtant d'être connus. Bien que je ne sois pas spécialiste en Histoire, j'ai souhaité partager quelques-unes de mes recherches.

Regards de guerre est une association fictive dédiée à l'organisation d'expositions sur le thème de la guerre. Chaque exposition met en lumière un conflit particulier, en utilisant le pouvoir de l'art pour susciter l'intérêt et la réflexion. Au-delà de sa vocation culturelle, l'association a pour objectif de faciliter l'accès à ses événements par le biais d'une plateforme de e-commerce proposant l'achat de e-tickets.

2. Organisation du site

Header

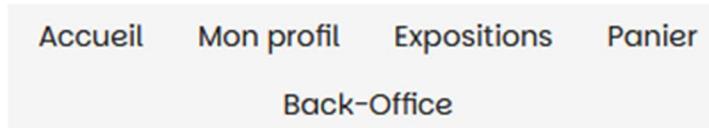
L'en-tête, situé en haut de chaque page, joue un rôle crucial dans **l'expérience utilisateur** et l'identité visuelle du site. Il doit être à la fois fonctionnel, esthétique et adapté à tous les appareils.

Logo et liens de connexion/ inscription :

Placés à gauche on les trouve aisément.



Menu de navigation : Clair et intuitif permet aux utilisateurs d'accéder aux pages principales du site (accueil, expositions, tickets, panier).



Accueil	X
Mon profil	
Expositions	
Panier	
Back-Office	

Pages

Accueil : Permet de connaître la dernière exposition programmée, d'avoir accès à l'agenda de toutes les expositions, de présenter l'accessibilité au lieu d'exposition.

Exposition : Permet de connaître le détail de l'exposition (sujet, artistes/type d'art, salle où sera représenté l'artiste) mais aussi de connaître les tickets disponibles.

Panier : Permet de réserver l'ensemble des tickets.

Back-office : Permet aux administrateurs de gérer l'ensemble du site à un seul endroit et sans avoir besoin de passer par le développeur (facturation, gestion des utilisateurs, expositions, artistes, tickets, stock)

Footer

Le pied de page, situé en bas de chaque page, regroupe les informations importantes et les liens utiles. Il contribue à la crédibilité et à la transparence du site.

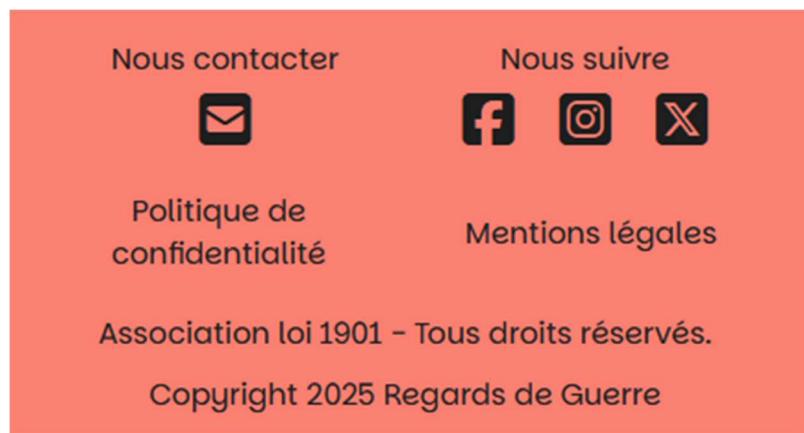
Liens vers les réseaux sociaux : Des icônes discrètes mènent aux pages des réseaux sociaux de l'association.

Informations de contact : Un lien permet d'envoyer directement un mail à un administrateur.

Autres informations : Page sur l'association et les locaux (café, boutique)

Liens vers les pages légales : Des liens mènent aux mentions légales et à la politique de confidentialité.

Copyright : La mention du type d'association affichée.



Cette présentation permet de respecter la **sémantique** et le **référencement SEO** (Search Engine Optimization). Respect de la hiérarchie des titres grâce aux **balises** h1, h2, section principale.

3. Champs d'action

Les champs d'action de l'utilisateur en tant que :

Visiteur

Un visiteur est un utilisateur non connecté qui peut naviguer librement sur le site et consulter les informations mises à disposition.

- Consultation des pages publiques :

Accueil : Présentation du site, agenda des expositions, plan d'accès et coordonnées.

Expositions : Informations sur les billets disponibles et les conditions d'achats. Ajout de tickets dans le panier en vue d'une commande future.

Utilisateur connecté

Un utilisateur connecté est un utilisateur authentifié qui dispose de fonctionnalités supplémentaires lui permettant d'interagir avec le site et de personnaliser son expérience.

- Effectuer une commande :

Finalisation d'un achat de tickets :

- Possibilité d'ajouter, modifier ou de supprimer des tickets dans son panier avant de finaliser sa commande.
- Confirmation effectuée par email incluant le détail de la commande.

Paramétrier son compte :

- Consulter son historique de commande : Accéder à la liste des commandes passées et aux détails de chaque commande. Il est également possible de recevoir la facture liée à une commande par mail.
- Modifier ses informations : Mettre à jour ses coordonnées ou son mot de passe.
- Supprimer son compte : Le membre peut demander la suppression de son compte et de ses données personnelles.

Root

Il dispose des droits les plus élevés. Ses actions se limitent à la gestion des autres administrateurs et à la configuration globale du système.

Ses responsabilités incluent :

- Attribution et modification des rôles administrateurs.
- Suppression de comptes administrateurs.

Administrateur : (Annexe 1)

Il intervient à différents niveaux, que ce soit sur les profils utilisateurs, les pages d'artistes, les expositions. Son rôle est essentiel pour garantir le bon fonctionnement de la plateforme et une expérience fluide pour tous les utilisateurs.

- Utilisateur :

Profil utilisateur : Consultation et suppression (à la demande ou si nécessaire).

Historique des commandes : Consulter les commandes effectuées et envoie de facture sur demande.

- Artistes :

Renseignement des données des artistes en vue de la création de la page d'exposition.

- Expositions :

Création des pages complètes des nouvelles expositions (titre, description, prix, artistes, salle)

➤ Tickets :

Gestion des tickets: Ajouter, modifier ou supprimer des tickets (type, prix, nombre).

Gestion des stocks: Mise en place d'un stock d'alerte pour les expositions presque épuisées et épuisées.

4. Minimum Viable Product (MVP)

L'ensemble des fonctionnalités présentées constituent le **Minimum Viable Product (MVP)**, visant à démontrer la faisabilité d'une plateforme de gestion d'expositions et de commandes de tickets.

Ce projet répond aux besoins essentiels des utilisateurs tout en garantissant une expérience utilisateur fluide et une gestion optimisée des utilisateurs, des expositions et des communications.

Ce MVP pose les bases d'une plateforme évolutive, permettant d'ajouter de nouvelles fonctionnalités en fonction des besoins futurs des utilisateurs et des exigences du marché.

Liste des compétences couvertes par REAC

➤ Développer la partie front-end d'une application web ou web mobile sécurisée

- Installer et configurer son environnement de travail en fonction du projet web ou web mobile
- Maquetter des interfaces utilisateur web ou web mobile
- Réaliser des interfaces utilisateur statiques web ou web mobile
- Développer la partie dynamique des interfaces utilisateur web ou web mobile

➤ Développer la partie back-end d'une application web ou web mobile sécurisée

- Mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL et NoSQL
- Développer des composants métier coté serveur
- Documenter le déploiement d'une application dynamique web ou web mobile

II Gestion de projet

La gestion de projet moderne exige des outils flexibles et des méthodologies agiles. Dans ce contexte, **Trello**, la **méthode MoSCoW** et le **Kanban**, au sein d'une **approche agile**, constituent une combinaison puissante pour organiser, prioriser et suivre l'avancement du projet de manière efficace.

1. Outils et méthodes

Méthode MoSCoW : Priorisation des exigences

Elle est utilisée pour prioriser les exigences du projet en fonction de leur importance. Elle classe les exigences en quatre catégories :

- **Must have (Doit avoir)** : Les exigences indispensables pour que le projet soit considéré comme un succès.

Exemples : BACK-OFFICE Gestion des expos

- Creation d'une page expo
- Développer le formulaire de création page expo
- Proposer l'achat de ticket s'il en reste sinon pas de bouton cta
- Supprimer de l expo seulement si ya pas eu de commande

- **Should have (Devrait avoir)** : Les exigences importantes, mais qui peuvent être reportées si nécessaire.

Exemples : BACK-OFFICE Admin

- Page historique de commandes
- Permettre de voir les commandes déjà effectuées par les clients
- Mep du suivi de nb de places restantes
- Alerte pour stock de sécurité

- **Could have (Pourrait avoir)** : Les exigences souhaitables, mais qui ont une priorité plus faible.

Exemples : BACK-OFFICE Gestion des stocks

- Mep bandeau admin en évidence : stock alerte
- Mep image presque épuisé/épuisé sur l'expo+agenda
- Précision du stock sur ticket (épuisé ou presque avec conditions)
- Vérif stock lors de la commande / msg flash
- Mep alerte mail : stock alerte
- Mep filtre dans gestion des stocks
- Mep condition de stock pour envoi de mail alerte que quand cest nécessaire

- **Won't have (N'aura pas)** : Les exigences qui ne seront pas incluses dans cette version du projet.

Exemples : FRONT-END / BACK-END : Blog

- Création page présentation du blog
- Création/récupération page expo
- crud de commentaires par les clients
- Gestion des accès aux commentaires (visiteurs - auteur - admin)
- Gestion des expos passées -> blog

Cette méthode permet de se concentrer sur les fonctionnalités essentielles pour la réalisation d'un MVP viable.

Kanban : Un flux de travail continu

Le Kanban est une **méthode de gestion de flux de travail** qui vise à optimiser la productivité en limitant le travail en cours.

Il repose sur un tableau visuel qui représente les différentes étapes du processus de travail

("À faire", "En cours", "A tester", "Terminé"). Les tâches sont déplacées d'une étape à l'autre, ce qui permet de visualiser l'avancement du projet et d'identifier les obstacles.

Le Kanban encourage l'amélioration continue en permettant à l'équipe de s'adapter aux changements et d'optimiser son flux de travail.

Trello (*Annexe 2*) : Un tableau de bord visuel

L'intégration de ces méthodes est facilitée par Trello, un tableau de bord centralisé offrant une vue d'ensemble claire de l'état d'avancement.

Son interface intuitive, basée sur des cartes (chacune définissant un **sprint** d'une période de 3 jours à 2 semaines) et des listes, permet de visualiser les tâches et les échéances.

Chaque carte représente une tâche, classée selon la méthode MoSCoW, et peut être déplacée entre les listes représentant les étapes du projet (Kanban).

Méthodologie Agile

L'approche agile a permis de structurer mon travail et de maintenir une organisation rigoureuse, même en travaillant seule.

Elle a favorisé l'adaptation aux changements et aux imprévus, la concentration sur les tâches prioritaires et la limitation de la dispersion.

Au quotidien, une liste de tâches était établie, réajustée en fin de journée ou servant de base à la planification du lendemain.

Intégration et avantages :

Cette combinaison d'outils et de méthodes a permis de :

- Visualiser l'avancement du projet en temps réel.
- Identifier et résoudre les problèmes rapidement.
- S'adapter aux changements et aux imprévus.
- Réaliser le MVP dans les délais impartis.

2. Langages

Côté client

- **HTML (HyperText Markup Language)** : Langage de balisage utilisé pour structurer et organiser le contenu des pages web. Il définit la signification et la structure des éléments.

- **CSS (Cascading Style Sheets)** : Langage de style utilisé pour définir la présentation et l'apparence des pages web. Il permet de contrôler la mise en page, les couleurs, les polices, etc.
- **JS (JavaScript)** : Langage de programmation permettant d'ajouter de l'interactivité et des fonctionnalités dynamiques aux pages web (menu burger).

Côté serveur

- **PHP (Hypertext Preprocessor)** : Langage de programmation principal qui gère la logique métier, la communication avec la base de données, traiter des formulaires, personnaliser les pages web et la génération de contenu dynamique.
- **SQL (Structured Query Language)** : Langage de requête utilisé pour interagir avec des bases de données relationnelles. Il permet de récupérer, de modifier et de gérer des données stockées dans des tables.

3. Technologies

Symfony

Framework PHP doté d'une architecture **Modèle-Vue-Contrôleur (MVC)** qui favorise une organisation claire du code, facilitant ainsi la séparation des responsabilités et la maintenance.

Ses composants réutilisables (formulaire, routes, sécurité) et sa configuration flexible permettent de l'adapter précisément aux besoins de chaque projet.

Il permet grâce aux lignes de commandes de créer un projet aisément :

```
PS C:\laragon\www\regards_guerre> symfony console make:user
```

Permet de créer les entités, ses champs, leur format, un champ de mot de passe sécurisé, le nom de l'entité unique, la création d'un repository et la mise en place d'un rôle. Il met également en place les **accesseurs** et **mutateurs**.

```
PS C:\laragon\www\regards_guerre> symfony console make:entity NameEntity
```

Permet de créer ses champs, leur format et son repository. Il ne faut pas oublier de créer les relations ManyToMany qui deviennent des tables associatives.

```
PS C:\laragon\www\regards_guerre> symfony console make:controller NameController
```

Permet de créer le contrôleur d'une entité avec une route et une fonction par défaut ainsi qu'un dossier de template possédant un index.

Un contrôleur et une vue Home a également été mise en place.

```
PS C:\laragon\www\regards_guerre> symfony console doctrine:database:create  
DATABASE_URL="mysql://root:@127.0.0.1:3306/regardsGuerre"  
  
doctrine:  
| dbal:  
| | url: '%env(resolve:DATABASE_URL)%'
```

Permet de créer la base de données en fonction du nom (.env) et des paramètres par défaut (doctrine.yaml)

```
PS C:\laragon\www\regards_guerre> symfony console make:migration  
PS C:\laragon\www\regards_guerre> symfony console doctrine:migrations:migrate
```

Permet de préparer puis d'appliquer la mise à jour du schéma de la base de données à partir des entités et de leurs relations.

Doctrine

Object Relational Mapping (ORM) qui établit une passerelle entre PHP et la base de données relationnelle MySQL.

Son approche orientée objet permet de manipuler les données avec **Doctrine Query Language (DQL)**, un langage de requête intuitif, évitant ainsi l'écriture de **Structured Query Language (SQL)** complexe.

Les migrations facilitent la gestion des schémas de base de données, assurant la cohérence et la traçabilité des changements.

Twig

Moteur de **templates** de Symfony, il favorise la séparation entre la **logique de présentation** et la **logique métier**, améliorant ainsi la lisibilité et la maintenabilité du code.

L'héritage de templates permet de créer des mises en page réutilisables, réduisant la duplication de code et assurant une cohérence visuelle.

De plus, il intègre des fonctionnalités de sécurité robustes pour protéger les applications web contre les attaques courantes.

4. Environnement

VS Code (Visual Studio Code)

Cet éditeur de code a été choisi pour sa légèreté, sa performance et sa vaste gamme d'extensions qui en font un outil indispensable.

Exemple d'extensions utilisées :

- **Prettier** : automatise le formattage du code
- **PHP intelephense** : autocomplétion et vérification de refactoring
- **Twig Pack** : Templates réutilisables, Fonctionnalités avancées (héritage de templates, fonctions (ex : path()))

Son terminal intégré permet d'exécuter les commandes Symfony et Composer directement depuis l'éditeur.

Laragon

C'est un environnement de développement local portable, isolé, rapide et facile à utiliser pour Windows. Il simplifie considérablement la configuration d'un serveur web, d'une base de données.

HeidiSQL

C'est un **outil de gestion de base de données** qui, grâce à son interface graphique intuitive et à ses fonctionnalités complètes, facilite grandement la manipulation et la gestion des données.

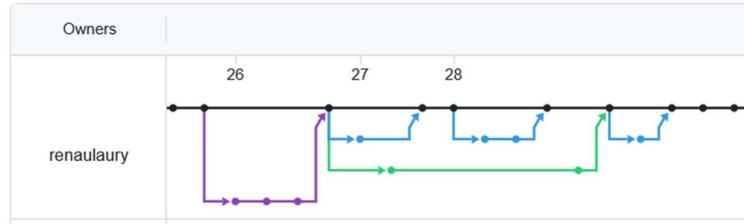
Il permet de visualiser et de modifier les données, de créer et de modifier les tables, d'exécuter et de tester des requêtes SQL avant de les adapter pour DQL.

Github Desktop et GitHub

Fort de son interface intuitive, **GitHub Desktop** simplifie le workflow de développement en offrant un suivi clair des modifications de code grâce aux **commits** (assurant un historique fiable et la possibilité de restaurer des états antérieurs). Il permet également une gestion aisée des **branches**, favorisant le travail en parallèle et l'expérimentation sans risque, tout en rendant la résolution des conflits de fusion plus accessible.

Merge pull request #8 from renaulaury/paiement
Laury • 7 days ago
mep stripe ok
Lily • 7 days ago
Merge pull request #7 from renaulaury/paiement
Laury • 7 days ago
maj
Lily • 7 days ago
Merge pull request #6 from renaulaury/restructuration-page-ticket
Laury • 7 days ago
délocalisation des tickets dans page ok
Lily • 7 days ago

GitHub, quant à lui, c'est la plateforme de **contrôle de version** et de collaboration utilisée pour ce projet et permet à mes formateurs de suivre l'avancement de mon projet en temps réel.



Composer

Gestionnaire de dépendances qui simplifie l'installation et la mise à jour des bibliothèques et des composants nécessaires à une application PHP, assurant la cohérence des versions et évitant les conflits.

Il utilise un fichier « *composer.json* » pour définir les **dépendances du projet** (doctrine, bundles (ex : la sécurité), **packages** (ex : les formulaires), **bibliothèques** (ex : Symfony mailer)).

Bundle

Ensemble structuré de fichiers PHP, de configurations, de **routes**, de **contrôleurs**, de **templates** et d'autres ressources qui fournissent une fonctionnalité spécifique. (exs : SecurityBundle, SymfonyCastsResetPasswordBundle, domPdf).

Looping

Logiciel de **modélisation conceptuelle des données (MCD)** utilisé pour créer des **diagrammes relationnels**. Il permet de concevoir des bases de données en structurant les informations sous forme **d'entités** et de **relations**.

L'outil facilite la création et l'édition de diagrammes avec des fonctionnalités intuitives. Looping aide à définir les **clés primaires, étrangères**, et les **cardinalités** entre les entités. Il génère automatiquement les **scripts SQL** pour la création des bases de données.

III Protection des données et sécurisation

1. Gestion des droits et des données personnelles

Dans le cadre du **Règlement Général sur la Protection des Données (RGPD)** et en accord avec les directives de la **Commission Nationale de l'Informatique et des Libertés (CNIL)** et sur les conseils de l'**Open Web Application Security Project (OWASP)**, j'ai implémenté

un ensemble de mesures de sécurité rigoureuses. Ces mesures visent à protéger les données personnelles de nos utilisateurs à chaque étape de leur cycle de vie, depuis leur collecte initiale jusqu'à leur suppression définitive.

Licéité, loyauté et transparence

Chaque utilisateur est informé clairement des informations collectées et un **consentement explicite** est requis via une case à cocher lors de l'inscription. Une **politique de confidentialité** détaillée est accessible depuis la navigation du site, garantissant une communication transparente sur l'usage des données.

J'accepte que mes données personnelles soient collectées et traitées conformément à **la politique de confidentialité**.

Limitation des finalités

Les données personnelles sont collectées uniquement pour des **finalités déterminées**, explicites et légitimes. Ces informations servent exclusivement à gérer les commandes de billets.

Minimisation des données

Seules les informations **strictement nécessaires** à la gestion des services sont collectées et conservées (email pour la connexion et l'envoi de commande, nom et prénom pour commander). Ce **principe de minimisation** limite les risques en cas de compromission.

Exactitude – droits d'accès/rectification

Les utilisateurs peuvent consulter, modifier ou corriger leurs données personnelles via un espace dédié. En cas de difficulté, ils ont également la possibilité de nous solliciter directement. Cette approche garantit que les informations traitées restent précises et à jour.

Mon profil

L'icône  vous permet de modifier votre identité et vos coordonnées.

Email : a.dupont@regardsguerre.fr 

Mot de passe : 

Nous contacter

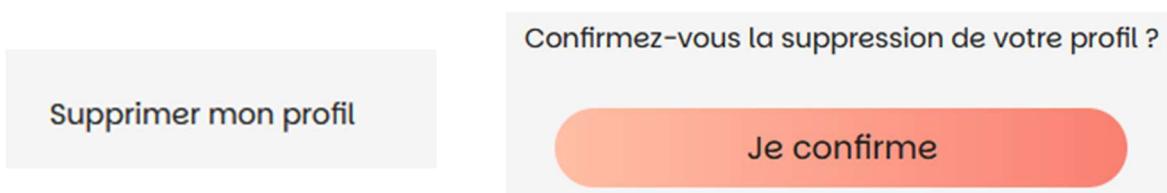


Limitation de la conservation

Les données personnelles sont **conservées uniquement pendant la durée nécessaire aux finalités** pour lesquelles elles sont collectées. Les informations liées aux commandes et aux factures sont supprimées après 10 ans. Ces mesures garantissent le respect des obligations légales en matière de durée de **conservation comptable**.

Droit à l'oubli

Les utilisateurs peuvent supprimer leur compte et toutes les données associées. Une suppression définitive est effectuée sur demande explicite, garantissant la disparition totale des informations.



Anonymisation des utilisateurs

Afin de préserver la confidentialité des informations personnelles, une fois la demande de suppression effectuée. L'ensemble des données personnelles associées à ce compte seront alors supprimées. Les informations relatives aux commandes passées seront accessibles via les factures correspondantes.

2. Sécurisation du code et des données

La gestion de l'inscription (Register)

Les **informations minimales sont demandées** : email + mot de passe
Le nom et prénom sont demandés mais sont facultatifs. En effet, ils seront obligatoires pour une commande.

A screenshot of a registration form titled "Création de compte". The form has a light beige background. It contains two input fields: "Nom¹" with placeholder "Votre nom" and "Prénom¹" with placeholder "votre prénom". Below the fields is a small note: "¹Ces informations sont facultatives mais seront nécessaires pour passer une commande.".

Validation des données

Les données personnelles (telles que l'email ou le nom) sont validées à l'aide des **contraintes Assert** de Symfony. Cela permet de garantir la conformité et la validité des informations entrées par l'utilisateur.

Exemple : Contrainte permettant de vérifier que l'email est unique en base de données

```
'constraints' => [ //Evite les doublons
    new Callback([$this, 'validateUniqueEmail']),
],
```

Exemple : Contrainte permettant de gérer le format des images téléchargées

```
'constraints' => [
    new Image([ //Gestion taille, format et erreur de téléchargement
        'maxSize' => '2G',
        'mimeType' => ['image/jpeg', 'image/png', 'image/webp'],
        'mimeTypeMessage' => 'Veuillez télécharger une image valide au format JPEG, PNG ou WEBP et de moins de 2Go.',
    ]),
],
```

Protection contre les attaques courantes

- Attaques **Cross-Site Request Forgery** (CSRF)

Définition : Elle force le navigateur d'un utilisateur authentifié à exécuter une action non désirée sur un site web, à son insu, en exploitant sa session active.

Exemple : L'utilisateur ouvre un email et clique sur un lien qui soumet silencieusement un formulaire effectuant une action non voulue sur un site où il est connecté.

Protection : Tous les formulaires manipulant des données sensibles sont protégés contre les attaques CSRF grâce à l'activation native de **csrf_protection** dans Symfony.

```
framework:
  form:
    csrf_protection:
      token_id: submit

    csrf_protection:
      stateless_token_ids:
        - submit
        - authenticate
        - logout
```

Lors de la soumission du formulaire, un jeton CSRF (un identifiant unique) est envoyé avec la requête. Ce jeton est généralement inclus dans le formulaire sous forme d'un champ caché.

```
<input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>
```

Lors de la réception de la requête sur le serveur, Symfony vérifie que le jeton soumis avec le formulaire correspond à celui généré pour cet utilisateur et cette session. Si le jeton est absent, expiré ou incorrect, Symfony rejette la requête comme étant potentiellement malveillante.

➤ Attaques **Cross-Site Scripting** (xss)

Définition : Elle consiste à injecter du code malveillant (souvent JavaScript) dans des pages web consultées par d'autres utilisateurs. Ce code injecté peut alors exécuter des actions malveillantes dans le navigateur de la victime, comme voler des informations de session ou modifier le contenu de la page.

Exemple : Un attaquant injecte du code malveillant via le formulaire de nom, et ce code s'exécute dans le navigateur d'autres utilisateurs lorsque les données contaminées sont affichées sans être sécurisées.

Protection :

La protection contre les attaques XSS est assurée à la fois en entrée et en sortie.

En entrée, les **FormTypes** normalisent et contraignent les données soumises par l'utilisateur à travers des **règles de validation configurées**. Ces filtres de validation préviennent l'introduction de structures syntaxiques interprétables comme du code exécutable côté client.

```
$builder
    ->add('userName', TextType::class, [
        'label' => 'Nom',
        'required' => true,
        'trim' => true,
        'constraints' => [
            new NotBlank(['message' => 'Le nom est obligatoire.']),
            new Length([
                'min' => 2,
                'max' => 50,
                'minMessage' => 'Le nom doit contenir au moins {{ limit }} caractères.',
                'maxMessage' => 'Le nom ne peut pas dépasser {{ limit }} caractères.',
            ]),
            new Regex([
                'pattern' => '/^([a-zA-Zéèçàùïöë -]+$/i', // Autorise lettres, espaces, tirets
                'message' => 'Le nom ne peut contenir que des lettres, des espaces et des tirets.',
            ]),
        ],
    ])
]
```

En sortie, Twig s'occupe de **l'échappement automatique** des variables dynamiques, ce qui empêche l'injection de scripts malveillants dans les pages web. Par conséquent, toute tentative d'injection de balises `<script>` est neutralisée. Cette double protection garantit que seules des données sûres et validées sont affichées aux utilisateurs.

```

<p>
    Email : {{ app.user.userEmail }}
</p>
|

```

➤ Attaques de force brute et par dictionnaire

Définitions/Exemples :

- Essai systématique de toutes les combinaisons possibles (mots de passe, clés, etc.) jusqu'à trouver la bonne.
- Tentative d'accès en utilisant une liste préétablie de mots de passe courants.

Protection :

Un système de limitation des tentatives de connexion est mis en place à l'aide de la fonctionnalité **Login Throttling** de Symfony.

```

main: #security.yaml #
    login_throttling:
        max_attempts: 10
        interval: "3 minutes"

```

Cette fonctionnalité permet de bloquer temporairement l'accès après un certain nombre d'échecs de connexion consécutifs. Cette mesure empêche les attaques de force brute, où un attaquant tente de deviner un mot de passe par essais successifs.

De plus, les mots de passe des utilisateurs sont sécurisés grâce à une **empreinte numérique** qui est générée en utilisant un algorithme de hachage sécurisé comme **bcrypt**.

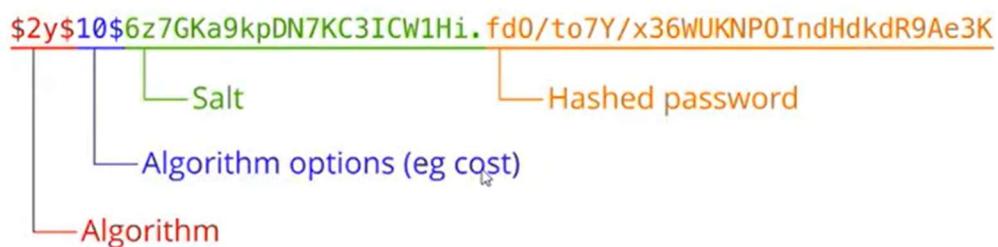
```

// Empreinte numérique du mot de passe security.yaml :
// Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto' (auto = default)
$user->setPassword($userPasswordHasher->hashPassword($user, $password));

```

Ce processus consiste à transformer le mot de passe en une valeur fixe de longueur déterminée, ce qui rend l'original impossible à retrouver, même si la base de données est compromise.

Type de **hachage fort** :



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 à des lettres, chiffres et caractères spéciaux.

Enfin, afin de renforcer la sécurité des mots de passe, une validation par **expression régulière (regex)** est appliquée. La regex suivante impose des critères stricts pour les mots de passe :

```
$regex = '/^(?=.*[A-Z])(?=.*\d)(?=.*[\w_]).{12,}$/';
```

Le mot de passe doit contenir au moins :

(A-Z) : Une lettre majuscule.

(\d) : Un chiffre.

([\w_]) : Un caractère spécial.

{12} : Douze caractères minimum.

➤ Injections SQL

Définition : C'est une technique d'attaque qui consiste à insérer du code SQL malveillant dans les requêtes envoyées à une base de données, souvent via des formulaires.

Exemple : Un attaquant utilise la chaîne de caractères « ; *DROP TABLE users ;* » dans un champ de saisie pour tenter de supprimer la table des utilisateurs de la base de données.

Protection : Symfony utilise **Doctrine ORM (Object Relationnel Mapping)** pour gérer l'accès à la base de données.

Pour éviter les attaques par injection SQL, qui visent à manipuler les **requêtes SQL** en injectant des instructions malveillantes, Doctrine prépare systématiquement les requêtes en utilisant des **requêtes paramétrées**.

Ce mécanisme empêche l'insertion de code malveillant en séparant la logique de la requête et les données.

Lors de la création des **DQL (Doctrine Query Language)**, le QueryBuilder est utilisé pour construire dynamiquement les requêtes SQL de manière sécurisée.

La méthode *setParameter()* permet de lier les valeurs des paramètres à la requête sans les injecter directement dans la chaîne SQL.

Cela garantit que les valeurs sont correctement échappées et ne risquent pas d'être interprétées comme des instructions SQL.

```

SELECT o.*  

FROM `order` o  

INNER JOIN order_detail od ON o.id = od.order_id  

INNER JOIN exhibition e ON od.exhibition_id = e.id  

INNER JOIN ticket t ON od.ticket_id = t.id  

WHERE o.user_id = 30  

ORDER BY e.date_exhibit ASC;

public function findOrdersDetailByUser(int $userId): array  

{  

    // Récupération de l'EntityManager pour interagir avec la base de données  

    $entityManager = $this->getEntityManager();  

    // Création du QueryBuilder (spécifique Symfony) pour construire la requête DQL  

    $queryBuilder = $entityManager->createQueryBuilder();  

    $queryBuilder->select('o')  

        ->from('App\Entity\Order', 'o')  

        ->innerJoin('o.orderDetail', 'od')  

        ->innerJoin('od.exhibition', 'e')  

        ->innerJoin('od.ticket', 't')  

        ->where('o.user = :userId')  

        ->setParameter(':userId', $userId)  

        ->orderBy('e.dateExhibit', 'ASC');  

    //Renvoie du résultat  

    // getQuery() retourne l'objet Query Doctrine qui permet d'exécuter la requête construite  

    // getResult() exécute la requête et retourne les résultats sous forme d'un tableau d'entités  

    return $queryBuilder->getQuery()->getResult();
}

```

➤ Faille Upload

Définition : Elle se produit lorsqu'une application web permet à un utilisateur de télécharger des fichiers sans vérifier correctement leur type, contenu ou origine.

Exemple : Un attaquant télécharge un fichier PHP contenant un virus lors de la création d'une exposition ou d'un artiste. Ce fichier serait ensuite exécuté, infectant le serveur.

Protection : Pour cela, une limite des fichiers autorisés a été mise en place :

- Dans le *FormType*:

```

'constraints' => [  

    new Image([ //Gestion taille, format et erreur de téléchargement  

        'maxSize' => '2G',  

        'mimeType' => ['image/jpeg', 'image/png', 'image/webp'],  

        'mimeTypeMessage' => 'Veuillez télécharger une image valide au format  

        JPEG, PNG ou WEBP et de moins de 2Go.',  

    ]),

```

- Dans le *Controller*: \$allowedMimeTypes = ['image/jpeg', 'image/webp', 'image/png'];

➤ Injection dans le code source

Définition : Elle survient lorsqu'un lien s'exécute dans un nouvel onglet. Exposant alors la page à des interactions non désirées permettant de modifier la page originale.

Exemple : L'attribut `target="_blank"`, ouvre un lien dans un nouvel onglet tout en laissant la page ouverte vulnérable à une modification via `window.opener`.

Protection : Ajouter dans le lien « `rel="noopener noreferrer"` » :

`noopener`: empêche la nouvelle page ouverte d'accéder à la page d'origine via `window.opener`

`noreferrer`: empêche le navigateur de transmettre des informations de référence HTTP (Referer) au site cible. Cela protège la confidentialité des utilisateurs en empêchant le site de destination de savoir d'où provient la requête.

```
<a href="https://www.facebook.com" target="_blank" rel="noopener noreferrer" aria-label="Notre page Facebook" title="Suivez-nous sur Facebook (nouvelle fenêtre)">
|   <i class="iconNetworks fa-brands fa-square-facebook"></i>
</a>
```

Responsabilité

➤ Droits d'accès :

La gestion des droits d'accès est strictement encadrée par l'utilisation des rôles (`ROLE_USER`, `ROLE_ADMIN`, `ROLE_ROOT`).

```
access_control:
- { path: ^/root, roles: ROLE_ROOT }
- { path: ^/backOffice, roles: IS_AUTHENTICATED_FULLY }
- { path: ^/admin, roles: ROLE_ADMIN }
- { path: ^/profile, roles: ROLE_USER }
- { path: ^/user, roles: ROLE_USER }
- { path: ^/orderHistory, roles: ROLE_USER }
```

Les contrôles sont effectués via la méthode `isGranted()`

```
if (!$this->isGranted('ROLE_ADMIN') && !$this->isGranted('ROLE_ROOT')) {
|   return $this->redirectToRoute('home');
}
```

La gestion des **login** et **register** est essentielle pour offrir une expérience utilisateur complète et sécurisée. Ces fonctionnalités permettent aux utilisateurs de créer un compte (inscription) et de se connecter à une application (connexion). Symfony utilise **SecurityBundle**.

IV Search Engine Optimization

Il regroupe un ensemble de techniques destinées à améliorer la visibilité d'un site web dans les résultats des moteurs de recherche. L'objectif est d'optimiser le contenu, la structure et les performances du site afin de favoriser un meilleur positionnement sur des requêtes pertinentes. Cela permet d'augmenter le trafic organique, c'est-à-dire les visites provenant naturellement des moteurs sans passer par de la publicité.

1. Structure du contenu et balisage HTML

Structure sémantique

Une **structure sémantique et hiérarchique** bien définie est cruciale pour une bonne indexation des pages par les moteurs de recherche et pour offrir une navigation claire aux utilisateurs. Une structure correcte aide à comprendre l'importance relative des éléments et à mieux organiser le contenu.

- Balise **<header>** : Elle doit inclure les éléments essentiels comme le logo, la navigation principale et les informations primaires relatives à la page. Cette section informe les moteurs de recherche et les utilisateurs sur le contenu principal du site.
- Balise **<footer>** : Située en bas de la page, cette balise contient généralement les informations de contact, les liens de navigation secondaires, les mentions légales et la politique de confidentialité. Elle aide les moteurs de recherche à comprendre la structure du site tout en étant utile pour les utilisateurs.
- Balises **<section>** : Utilisées pour structurer les différentes sections d'une page. Chaque section devrait contenir un sujet distinct.
- Balises **<div>** : Utilisées en complément des balises sémantiques comme `<section>` pour éviter une structure trop « plate » qui pourrait compliquer l'indexation par les moteurs de recherche.

Balisés métas définies

Les balises métas sont des éléments HTML qui fournissent des informations supplémentaires sur le contenu d'une page et doivent toujours être adaptées à chaque page pour garantir un maximum de pertinence.

- Balise **<title>** : Le titre de la page qui apparaît dans les résultats des moteurs de recherche. Il doit être concis, contenir des mots-clés pertinents et décrire le contenu de la page. En général, la longueur idéale est entre 50 et 60 caractères.

```
{% block title %}Back-Office{% endblock %}
```

- Balise <**meta description**> : Elle permet de donner un aperçu du contenu de la page dans les résultats de recherche. Bien que la meta-description n'influence pas directement le classement, elle impacte le taux de clics, donc elle doit être attrayante et inclure des mots-clés.

```
{% block description %}  
    Le back-office permet aux administrateurs de gérer les expositions, les artistes, les utilisateurs, la gestion des stocks et l'accès aux factures.  
{% endblock %}
```

Attributs Alt Complets sur Toutes les Images

Les **attributs alt** (textes alternatifs) sont essentiels pour le SEO des images. Ils permettent aux moteurs de recherche de comprendre le contenu de l'image, ce qui est particulièrement important pour l'accessibilité et l'indexation du contenu visuel. Un attribut alt complet décrit l'image de manière précise et naturelle.

```

```

Ils permettent :

Accessibilité : Les utilisateurs ayant des handicaps visuels utilisent des lecteurs d'écran qui lisent ces balises pour décrire les images.

SEO : Les moteurs de recherche ne "voient" pas les images, mais ils peuvent comprendre le texte des attributs alt. Cela améliore l'indexation des images dans les résultats de recherche, notamment dans la Google Image Search.

2. Optimisation technique

URLs optimisées

La structure des URLs joue un rôle fondamental pour améliorer la visibilité et l'accessibilité du site. Une URL optimisée doit être claire, descriptive et contenir des mots-clés pertinents afin de faciliter l'indexation par les moteurs de recherche et d'offrir une meilleure expérience utilisateur.

Dans ce contexte, j'ai mis en place un système de génération automatique de **slugs** en utilisant la bibliothèque **Cocur/Slugify**. Cette bibliothèque permet de convertir les titres dynamiques en slugs normalisés, en supprimant les caractères spéciaux, en remplaçant les espaces par des tirets (-) et en convertissant tous les caractères en minuscules. Cette approche garantit une structure d'URL cohérente et améliore leur lisibilité, et leur convivialité les rendant **user-friendly** à la fois pour les utilisateurs et les moteurs de recherche.

Méthode créée dans l'entité :

```
public function createSlugArtist(): string
{
    $slugify = new Slugify();

    // Construction du slug à partir de l'ID, du prénom et du nom
    $slugSource = $this->id . '-' . $this->artistFirstname . '-' . $this->artistName;

    return $slugify->slugify($slugSource);
}
```

Puis création du slug dans le contrôleur lors de la création d'un nouvel enregistrement.

```
//Générer le nouveau slug
$slug = $artist->createSlugArtist();           slug
$artist->setSlug($slug);                      4-jean-moribon

// Définir le slug sur l'entité
$artist->setSlug($slug);
```

Cela permet d'assurer une meilleure indexation des pages, de favoriser leur partage et d'améliorer l'expérience utilisateur tout en garantissant une compatibilité optimale avec les moteurs de recherche.

Chargement des pages grâce à la compression des images

La vitesse de chargement des pages est un facteur clé pour l'expérience utilisateur et le SEO. Des pages plus rapides sont favorisées par les moteurs de recherche et génèrent un meilleur taux de conversion. Les utilisateurs sont de plus en plus impatients, et un site lent risque de les faire fuir, entraînant ainsi un **taux de rebond** élevé (pourcentage de visiteur quittant un site après avoir visité une seule page). Une méthode importante pour améliorer la vitesse de votre site est la compression des images.

Les formats **WebP** sont à privilégier et c'est la raison de la mise en place d'un convertisseur.

```
// Convertir l'image en WebP directement
$webpFileName = '00_main_image.webp';
$imageService->convertToWebP($file->getPathname(), $uploadDirectory . '/' . $webpFileName);

// Mettre à jour le chemin de l'image dans l'entité
$exhibition->setMainImage('images/events/' . $exhibition->getDateExhibit()->format('Ymd') . '/' .
$webpFileName);

// Renommer le dossier si la date a été modifiée
if ($oldDate && $oldDate != $exhibition->getDateExhibit()) {
    $oldDirectory = $this->getParameter('kernel.project_dir') . '/public/images/events/' .
    $oldDate->format('Ymd');

    if ($filesystem->exists($oldDirectory)) {
        $filesystem->rename($oldDirectory, $uploadDirectory);
    }
}
```

3. Interactivité maîtrisée

Une interactivité légère améliore l'expérience utilisateur sans nuire à la performance du site. Une interactivité trop complexe peut alourdir la page, ralentir son temps de chargement et affecter le SEO. Les scripts qui n'ont d'intérêt sur une unique page sont appelés uniquement sur celle-ci.

Utilisation de JavaScript pour une meilleure expérience utilisateur :

Côté client : Un menu burger.

Côté serveur : Sur l'ajout d'artiste dans la création de l'exposition.

V Conception du projet

1. Modélisation des données

Méthode Merise

La modélisation des données est une étape essentielle pour organiser et gérer les informations de manière optimale. J'ai choisi d'utiliser la **méthode Merise** qui offre une approche structurée et rigoureuse pour la modélisation des données et des traitements, permettant de passer d'une vision métier à une implémentation technique.

Modèle Conceptuel de Données (MCD) (*Annexe 3*)

Il représente une vue globale et indépendante des contraintes techniques. Ce modèle décrit les entités (objets ou concepts manipulés), les associations (relations entre ces entités) et leurs cardinalités (contraintes numériques sur ces relations). L'objectif est de structurer les informations en se concentrant sur leur signification métier.

Les relations et cardinalités

Dans la méthode Merise, les cardinalités sont essentielles pour définir la nature des relations entre les entités. Elles précisent combien d'occurrences d'une entité peuvent être associées à une autre.



ENTITES : Order – Invoice

OneToOne vers l'entité Order

OneToOne vers l'entité Invoice

En effet, chaque commande est associée à une seule facture.

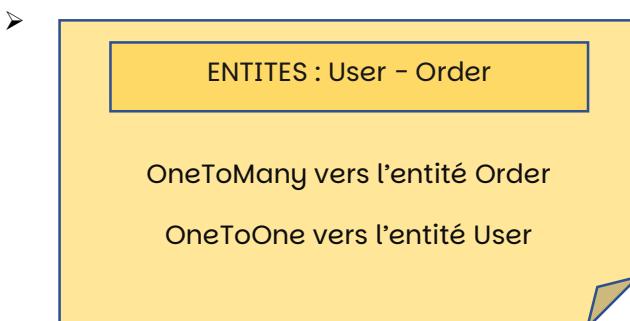
Chaque facture est associée à une seule commande.

- En effet, une exposition peut avoir 0 ou plusieurs tickets associés.
Un ticket peut être associé à 0 ou plusieurs expositions.

ENTITES : Exhibition – Ticket

ManyToMany vers l'entité Ticket

ManyToMany vers l'entité Exhibition



En effet, un utilisateur peut avoir 0 ou plusieurs commandes.
Une et une seule commande appartient à un seul utilisateur.

Modèle Logique de Données (MLD) (*Annexe 4*)

Il traduit le MCD en une structure directement exploitable par un **système de gestion de base de données (SGBD)**.

À ce niveau, les entités deviennent des tables, les associations se traduisent par des clés étrangères et les cardinalités définissent les types de relations. Les relations ayant un ManyToMany de part et d'autre deviennent une table associative.

Dans Symfony, les tables associatives deviennent une entité à part entière.

La relation Exhibition – Artist – Room donnera naissance à la table Show et présentera des attributs complémentaires (artistPhoto, artistPhotoAlt, artistTextArt) qui n'existeront que dans cette nouvelle table.

La relation Exhibition – Ticket donnera naissance à la table ticketPricing et présentera des attributs complémentaires (unitPrice, quantity) qui n'existeront que dans cette nouvelle table.

La relation Exhibition – Order donnera naissance à la table orderDetail et présentera un attribut complémentaire (standardPrice) qui n'existera que dans cette nouvelle table.

2. Maquettage

Une fois la structure de données définie, l'étape suivante a consisté à concevoir l'interface utilisateur en tenant compte des contraintes identifiées, des entités à afficher, et de l'expérience de navigation souhaitée.

Le **maquettage** permet de donner un visuel au MVP et permet de visualiser la structure, l'agencement et l'interaction des éléments avant de passer au développement. **Figma**, avec ses fonctionnalités intuitives, est un outil de choix pour cette phase.

Version Mobile – First (*Annexe 5*)

J'ai commencé par concevoir la version mobile pour assurer une compatibilité optimale sur les petits écrans. Cette approche est essentielle car la majorité des utilisateurs accèdent aux sites via leur smartphone. Cette méthode présente plusieurs avantages :

- Priorisation du contenu essentiel : Les contraintes des petits écrans obligent à se concentrer sur les informations les plus importantes.

Amélioration de l'expérience utilisateur (ux) : Un design optimisé pour le mobile offre une navigation fluide et intuitive.

Optimisation du référencement (SEO) : Google privilégie les sites adaptés aux mobiles.

Version Desktop (*Annexe 6*)

Après la validation de la version mobile, j'ai adapté les maquettes pour les résolutions plus larges en optimisant l'affichage sur ordinateur.

Moodboard : L'Univers Visuel

Le **moodboard** est un tableau d'inspiration visuelle regroupant des éléments clés pour définir l'identité du site.

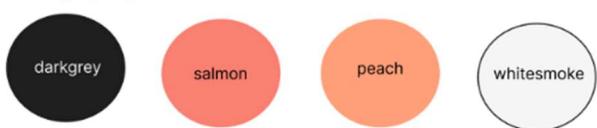
- Inspirations Principales :

Des tons doux pour mettre en lumière la mémoire.

Interface épurée, facilitant la lecture et l'interaction.

Contraste suffisant et hiérarchisation de l'information pour une expérience optimale.

Charte graphique :



Logos :



Polices :

Cormorant Garamoud

Icônes :



Les camps d'Algérie, créés pendant la guerre d'indépendance, sont devenus des symboles de l'exil, de la souffrance et du déracinement, où les conditions de vie des populations déplacées étaient marquées par l'humiliation et l'abandon.

Produits :



Inspi :

[Awwwards Nominees](#)

Charte Graphique

La **charte graphique** assure la cohérence visuelle et identitaire du site. Elle comprend :

➤ Palette de Couleurs :

Salmon Puff (`#FFA07A` et ses déclinaisons) : Symbolise la chaleur et la joie.
Gris Anthracite (`#1e1e1e`) : Pour les textes, apportant contraste et lisibilité.
Whitesmoke (`#f5F5F5`) : Favorise la clarté sans agresser l'œil pour l'arrière-plan du site.
Brun roux (`#944C43`) : Pour mettre en valeur les titres et les icônes.

➤ Logos et Icônes :

Logo : bien qu'il ne soit pas totalement minimalistique, il met en avant, dès le premier regard, un œil, symbole essentiel du projet. À côté de celui-ci, on retrouve des éléments liés à la guerre, comme le heaume, les épées, et les plumes, qui illustrent à la fois la lutte et les pertes humaines. Les ailes, quant à elles, ajoutent une dimension de mémoire et de souffrance, évoquant les nombreuses vies perdues au cours du conflit.

Icônes : simples et reconnaissables pour faciliter la navigation.

➤ Images et Visuels :

Les photographies immersives utilisées pour illustrer les expositions proviennent principalement de Freepik, sous licence gratuite avec attribution. À l'exception des expositions « L'Ukraine en résistance » (œuvre de Seth) et « L'incident de Kyujo » (extrait du film *Japan's Longest Day*), qui sont soumises à d'autres droits d'auteur.

Un traitement visuel sobre et épuré pour rester en accord avec le thème mémoriel.

3. Expérience utilisateur, interface et accessibilité

L'**Expérience Utilisateur (UX)** se concentre sur la facilité d'utilisation et l'efficacité, tandis que l'**Interface Utilisateur (UI)** concerne l'aspect visuel et l'esthétique.

User Experience : Améliorer le Parcours Utilisateur

L'objectif principal de l'UX est de garantir une navigation intuitive et efficace. Pour cela, j'ai adopté plusieurs bonnes pratiques :

➤ Navigation fluide : Un menu clair et un panier accessible sur toutes les pages.

- Cohérence visuelle : Utilisation de repères visuels (boutons, icônes) pour guider l'utilisateur.

User Interface :Créer une Interface Claire et Esthétique

L'interface utilisateur se concentre sur l'aspect visuel du site. J'ai adopté une présentation épurée avec des éléments visuels cohérents :

- Typographie lisible :

La police *Cormorant Garamond*, choisie pour son esthétique élégante, est utilisée pour les titres et certains paragraphes peu importants.

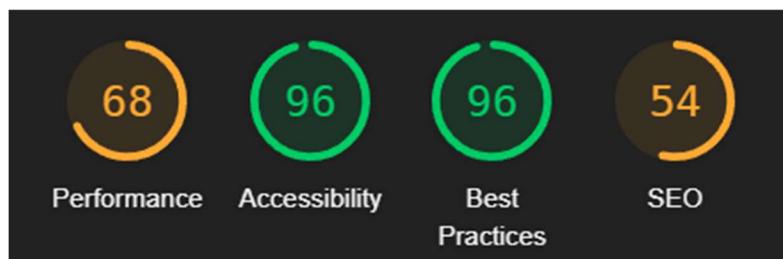
La police *Parkinsans*, plus lisible, est utilisée pour le corps de texte principal.
- Espacements équilibrés : Pour une lecture confortable et une hiérarchie claire.
- Boutons et interactions : Des boutons bien contrastés pour encourager l'action.

Outils utilisés pour optimiser le rendu :

- Whocanuse : Permet de tester les contrastes entre les couleurs de l'interface pour s'assurer qu'ils sont lisibles pour les personnes atteintes de déficiences visuelles (daltonisme, basse vision, etc.).
- #F5F5F5 – #1e1e1e : Textes
- #F5F5F5 – 944C43 : Titres

Contrast Ratio	15.29:1	Contrast Ratio	5.71:1
----------------	----------------	----------------	---------------

- Lighthouse : Le fait d'effectuer des audits de manière régulière m'a permis de diriger les efforts concernant l'optimisation de l'expérience utilisateur.



Accessibilité :Un site pour tous

L'accessibilité est notamment régie par les règles déjà évoquées dans la partie SEO comme le respect de la structure sémantique, les balises métadéfinies, les attributs alt complets pour les images ou encore le contraste des couleurs argumenté dans la partie UI.

Mais en France elle est également guidée par le **Référentiel Général d'Amélioration de l'Accessibilité (RGAA)**, basé sur les standards internationaux des **Web Content Accessibility Guidelines (WCAG)**.

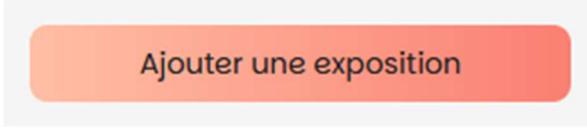
Mais également :

- Les formulaires accessibles : étiquette « label » correspondant à l'attribut « for »
- Langage de la page défini via l'attribut « lang="fr" », afin que les lecteurs d'écran puissent s'adapter correctement.
- « Title » mis en place principalement sur les boutons et les icônes.

Disponibilité des fonctionnalités

L'**affordance** d'un utilisateur désigne les indices visuels qui indiquent comment interagir avec un élément de l'interface.

Suivant la **loi de Jakob** un bouton doit ressembler à un bouton ("Panier", "Ajouter") car les utilisateurs sont familiers avec ces conventions. Selon la **loi de Fitts** il doit avoir une taille et un emplacement adaptés pour un clic aisément.



Ajouter une exposition

Les icônes doivent être immédiatement compréhensibles grâce à des symboles universels, comme trois lignes pour un menu. Notamment, pour la partie back-office, l'administrateur trouvera une notice d'explication concernant les différentes icônes.

Cette page offre une vue d'ensemble des stocks de tickets pour chaque exposition.

L'icône  signifie que les tickets sont épuisés

L'icône  signifie que les tickets sont presque épuisés

Cliquez sur l'icône  pour modifier le stock de tickets d'une exposition.

La navigation doit être fluide, avec des éléments comme des liens soulignés ou des menus déroulants qui sont familiers et facilement identifiables. Une bonne affordance rend l'interface intuitive, minimisant la nécessité d'explications supplémentaires et offrant une expérience utilisateur agréable et efficace.

L'accessibilité des fonctionnalités à tout moment et pour tous les utilisateurs est essentielle. Pour cela, il est important de mettre en place :

- Interface intuitive : L'interface utilisateur est simple et claire, avec des éléments visibles et compréhensibles. Cela inclut des boutons, des liens et des icônes qui indiquent clairement leurs fonctions.
- Retours visuels : Afin que l'utilisateur sache ce qui se passe quand il interagit avec l'interface (par ex, lors du clic sur un bouton ou lors de la soumission d'un formulaire) un *hover* est mis en place que ce soit en version mobile ou en desktop.
- Visibilité des informations importantes : Les informations clés (prix, date d'exposition, boutons, call to action) sont mis en évidence sur les pages adéquates.

Responsive design

J'ai adopté une approche **mobile-first**, en concevant d'abord pour les écrans mobiles, avant d'adapter progressivement le design aux tablettes et aux ordinateurs. Cette méthodologie répond aux usages actuels, où la navigation mobile représente une part majoritaire du trafic web.

Benchmark (*Annexe 7*)

Il permet de comparer les pratiques et les caractéristiques de sites similaires afin de mieux comprendre les attentes des utilisateurs et adapter le projet en conséquence. Cela aide à définir des besoins tout en préservant les habitudes des utilisateurs.

Je me suis inspirée du site du [Musée Beyerler](#) pour la gestion des tickets, afin de faciliter l'expérience utilisateur tout en intégrant une logique simple et fluide.

Par ailleurs, le design du site [Festival Film Festival](#), récompensé d'une mention honorable en 2023 sur Awwwards, m'a servi de référence pour l'aspect visuel et l'interactivité.

VI Architecture de l'application

Le développement de ce projet s'appuie sur plusieurs principes fondamentaux du développement web moderne : la **programmation orientée objet (POO)**, la gestion des échanges client-serveur via les requêtes HTTP, et une architecture en couches.

Ces concepts permettent de structurer le code, de garantir sa maintenabilité, et d'assurer une organisation claire entre les différentes responsabilités de l'application.

1. Programmation Orientée Objet (POO)

C'est un **paradigme de programmation** qui permet de structurer le code autour de "l'objet", une entité regroupant des données et des comportements. Dans la POO, tout est **objet**, et

chaque objet appartient à une **classe**. Ce paradigme vise à rendre le code plus modulaire, réutilisable, et facile à maintenir.

Les principaux concepts de la POO :

- Objet et Classe

Classe : Une classe est un plan pour créer des objets. Elle définit les attributs (titleExhibit, dateExhibit, mainImage, etc) et les **méthodes** (getters et setters ainsi que par exemple un getter spécialisé dans le formatage de date) que les objets de cette classe auront.

Exemple : class Exhibition

```
public function getDateExhibitFr()
{
    $date = $this->dateExhibit;

    // Create a DateTimeFormatter for French locale
    $dateFormat = new \IntlDateFormatter(
        'fr_FR', //Pays
        \IntlDateFormatter::LONG, //Format long
        \IntlDateFormatter::NONE); //Fuseau horaire ou heure

    return $dateFormat->format($date);
}
```

Objet : Un objet est une **instance d'une classe**. Il possède les propriétés définies dans la classe et peut exécuter les méthodes définies pour cette classe.

Exemple : Ajout d'une nouvelle exposition, avec des tickets

```
$exhibition = new Exhibition();
// Gestion des TicketPricing
foreach ($exhibition->getTicketPricings() as $ticketPricing) {
    $ticketPricing->setExhibition($exhibition);
    $entityManager->persist($ticketPricing);
}
```

➤ Encapsulation

Elle vise à protéger les données d'un objet en rendant ses attributs privés et accessibles uniquement via des méthodes dédiées (getters/setters). Cela permet de centraliser la **logique métier**, d'ajouter des règles de validation et de préserver **l'intégrité des données**.

Exemple : Dans l'entité Exhibition, la méthode *getTicketsRemaining()* en est un exemple : elle calcule le nombre de billets restants sans exposer les détails du calcul. Le contrôleur utilise cette méthode sans connaître la logique interne, ce qui illustre le principe d'encapsulation.

```

//Nb de tickets restants
public function getTicketsRemaining(): int
{
    return $this->getStockMax() - $this->getTicketsReserved();
}

```

➤ L'héritage

C'est un principe fondamental de la POO permettant à une **classe enfant** (ou sous-classe) de réutiliser les attributs et méthodes d'une classe parente. Cela favorise la réutilisation du code et sa spécialisation sans duplication.

Dans Symfony, les contrôleurs héritent de la classe *AbstractController*, qui fournit des méthodes utilitaires comme *render()* pour afficher des vues Twig, ou encore des fonctions liées aux formulaires, services ou autorisation (*createForm()*, *isGranted()*).

Exemple : *ExhibitionBOController* use *AbstractController* permet d'utiliser *render()*

```

final class ExhibitionBOController extends AbstractController
{
    /***** Liste des expositions *****/
    #[Route('/backOffice/exhibitListBO', name: 'exhibitListBO')]
    public function exhibitListBO(ExhibitionShareRepository $exhibitionShareRepo): Response
    {
        // Vérif de l'accès
        if (!$this->isGranted('ROLE_ADMIN') && !$this->isGranted('ROLE_ROOT')) {
            return $this->redirectToRoute('home');
        }

        $exhibitions = $exhibitionShareRepo->findAllNextExhibition();

        return $this->render('/backOffice/exhibition/exhibitListBO.html.twig', [
            'exhibitions' => $exhibitions,
        ]);
    }
}

```

2. Requêtes http (HyperText Transfer Protocol)

Les **requêtes HTTP** jouent un rôle essentiel dans la communication entre le client et le serveur. Chaque interaction entre l'utilisateur et le site repose sur ces échanges, garantissant l'affichage des pages, la récupération des données et l'envoi d'informations.

➤ GET : Utilisée pour récupérer des ressources depuis le serveur, comme des pages HTML, des images ou encore des données d'expositions. Lorsqu'un utilisateur demande à voir une page ou une image, une requête GET est envoyée pour récupérer ces éléments

Exemple : L'affichage d'une liste d'artistes depuis la base de données.

```
$allArtists = $artistRepo->findArtists();
```

- POST : Employée pour envoyer des données au serveur, notamment lors de l'inscription d'un utilisateur ou de la réservation d'un ticket.

Exemple :

- Dans le template : La méthode peut être précisée mais sur Symfony, la méthode par défaut est 'post'.

```
<form method="post" action="{{ path('artistAddBO') }}">
{{ form_start(form) }}
```

- Dans le contrôleur, les données du formulaire sont récupérées

```
// Cr ation du form
$form = $this->createForm(ArtistAddEditB0Type::class, $artist);

// Traiter la requ te HTTP
$form->handleRequest($request);

// V rif du formulaire
if ($form->isSubmitted() && $form->isValid()) {
    //G n rer le nouveau slug
    $slug = $artist->createSlugArtist();
    $artist->setSlug($slug);

    // D finir le slug sur l'entit 
    $artist->setSlug($slug);

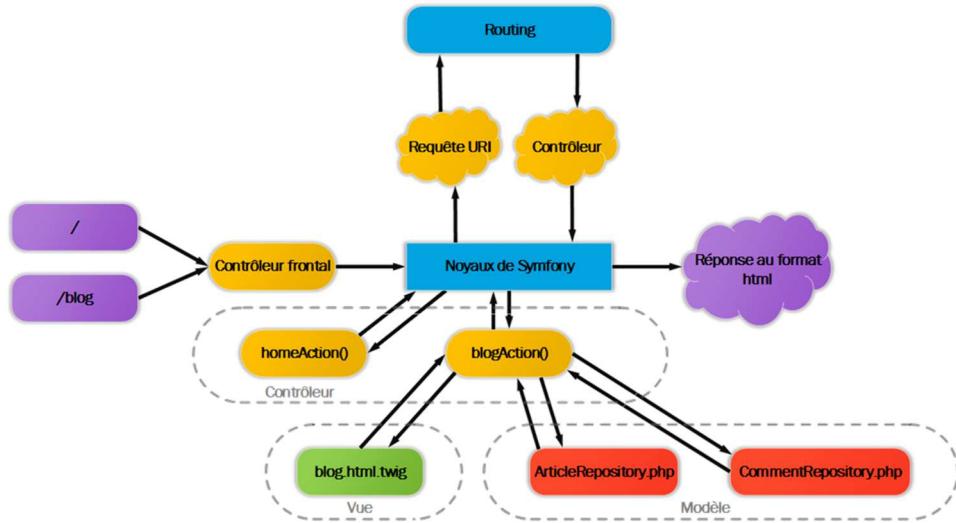
    // Persister les modifications dans la base de donn es
    $entityManager->persist($artist);
    $entityManager->flush();

    // Rediriger vers la liste des artistes
    return $this->redirectToRoute('artistListBO');
}
```

Symfony structure ces requ tes via des routes d finies dans les **contr leurs**. Lorsqu'un utilisateur effectue une action (exemple : cliquer sur un bouton de r servation), une requ te HTTP est envoy e au serveur, qui traite la demande et renvoie une **R ponse au format HTML**.

3. Symfony 7 et son design pattern

Le **MVC (Model – Vue – Controller)** est le **design pattern** utilisé par Symfony.



Il permet de structurer le projet en 3 couches distinctes :

Lorsque l'utilisateur est sur la page « home », une requête HTTP est envoyée, elle est d'abord interceptée par le **contrôleur frontal** (`index.php`), qui agit comme point d'entrée unique. Cette requête est ensuite transmise au **noyau de Symfony**, qui utilise le **système de routing** pour identifier la route correspondante et déterminer le **contrôleur approprié**.

- **Controller (couche logique)** : Il exécute alors la méthode liée à cette route, interroge le **modèle** pour récupérer les données nécessaires et renvoie une réponse HTML à la vue pour qu'elles soient affichées à l'utilisateur.

Exemple d'utilisation :

Le contrôleur rassemble les informations récupérées depuis la base de données via le modèle puis les transmet à la vue pour que le détail de l'agenda soit affiché.

```
#[Route('/home', name: 'home')]
public function index(ExhibitionShareRepository $exhibitShareRepo): Response
{
    $exhibitions = $exhibitShareRepo->findNextExhibition(); //3 dernières expos
    $agenda = $exhibitShareRepo->findAllNextExhibition(); //agenda des expos

    return $this->render('home.html.twig', [
        'exhibitions' => $exhibitions,
        'agenda' => $agenda,
    ]);
}
```

- **Model (couche infrastructure)** : Le model est **responsable de la gestion des données et de la logique métier** de l'application. Dans Symfony, cela inclut la gestion des entités (objets représentant des tables de la base de données) et l'interaction avec Doctrine pour récupérer ou manipuler les données.

Exemple d'utilisation :

Suite à la demande du contrôleur, le **repository** effectue une **requête DQL (Doctrine Query Language)** afin de récupérer les informations des expositions uniquement nécessaires pour l'agenda, et les renvoie au contrôleur pour qu'il les envoie à la vue.

```
***** Sélectionne toutes les dernières expositions prévues *****
public function findAllNextExhibition() {
    // Récupération de l'EntityManager pour interagir avec la base de données
    $entityManager = $this->getEntityManager();
    // Création du QueryBuilder (spécifique Symfony) pour construire la requête DQL
    $queryBuilder = $entityManager->createQueryBuilder();

    $queryBuilder->select('e')
        ->from('App\Entity\Exhibition', 'e')
        ->where('e.dateExhibit > :now') //now() ne fonctionne pas seul, il faut setParameter
        ->setParameter('now', new \DateTime())
        ->orderBy('e.dateExhibit', 'ASC');

    //Renvoie du résultat
    // getQuery() retourne l'objet Query Doctrine qui permet d'exécuter la requête construite
    // getResult() exécute la requête et retourne les résultats sous forme d'un tableau d'entités
    return $queryBuilder->getQuery()->getResult();
}
```

- **View (couche de présentation)** : La vue est la partie du site que l'utilisateur voit et avec laquelle il interagit. Elle est **responsable de la présentation** des données, c'est-à-dire de l'affichage de l'interface utilisateur.

25 mai 2025 - 09h00 à 16h00

ÉPUISÉ

Les camps d'Algérie
Lieux de souffrance et de résistance (1954 - 1962)

Sujet sensible encore aujourd'hui, ils font désormais partie d'une mémoire collective partagée entre les Algériens et les Français.

> Fiche détaillée

3 juin 2025 - 09h00 à 16h00

L'Ukraine en résistance
Chronique d'une guerre : l'Ukraine au cœur d'une crise européenne (2022 - à aujourd'hui)

L'invasion russe a déclenché un conflit majeur en Europe, bouleversant l'équilibre géopolitique et provoquant une crise humanitaire de grande ampleur.

> Fiche détaillée

Gérée par Twig, le moteur de templates de Symfony, permet de séparer la logique de présentation de la logique métier.

Exemple d'utilisation :

Pour afficher le détail de l'agenda sur le fichier de template Twig, j'ai fait appel à une **boucle foreach** afin d'afficher les informations nécessaires (date et horaires de l'exposition, texte d'information etc) Ces informations sont transmises au template par le contrôleur.

```

{% for agendas in agenda %}

    <div class="blockAgenda">

        <div class="exhibitAgenda">
            <div class="imgAgendaContainer">
                
            </div>

            <div class="infoAgendaContainer">
                <div class="blockDateAgenda">
                    <p><strong>{{ agendas.dateExhibitFr }}</strong> - {{ agendas.hourBeginFr }} à {{ agendas.hourEndFr }}</p>
                </div>

                <div class="blockImgStockHome">
                    {% if agendas.ticketsRemaining <= 10 and agendas.ticketsRemaining > 0 %}
                        
                    {% elseif agendas.ticketsRemaining <= 10 %}
                        
                    {% endif %}
                </div>

                <div class="blockTextAgenda">
                    <p class="cgBold">{{ agendas.titleExhibit }}</p>
                    <p class="cgBold">{{ agendas.subTitleExhibit }}</p>
                    {{ agendas.dateWarBeginFr }} -
                    {% if agendas.dateWarEndFr is null %}
                        à aujourd'hui
                    {% else %}
                        {{ agendas.dateWarEndFr }}
                    {% endif %}
                </p>

                    <p class="hookAgenda">{{ agendas.hookExhibit }}</p>
                </div>

                <a class="parkinsanBrown textHover" href="{{ path('exhibition', {'slug': agendas.slug}) }}> Fiche détaillée</a>
            </div>
        </div>

        {% if not loop.last %}
            <hr>
        {% endif %}

    </div>
{% endfor %}

```

VII Fonctionnalité phare

La fonctionnalité clé de ce projet se concentre sur la gestion des commandes et des paiements, assurée par le **CartService** et le **PaymentController**. Ces contrôleur facilite le suivi des commandes et garantissent des transactions sécurisées, offrant ainsi aux utilisateurs une expérience fluide et conforme aux normes de sécurité des paiements en ligne.

1. Les Services

Les **services** sont des objets réutilisables qui **encapsulent** une logique métier spécifique (mise en session, gestion du panier, envoi d'email). Ils favorisent un code modulaire, maintenable et centralisé. Ils respectent le principe de **Single Responsibility (S)** de **SOLID**.

- **CartService** : Il est le cœur de la gestion du panier d'achat. Il centralise la logique d'ajout, de suppression et de modification des articles. Il assure également le stockage et la récupération du panier, et calcule les totaux.

Voici les **méthodes** utilisées pour la fonctionnalité phare :

Le constructeur : Il utilise **l'injection de dépendances** pour recevoir et stocker des **instances** de *RequestStack* (pour gérer la session), *TicketRepository* (pour accéder aux données des tickets) et *ExhibitionShareRepository* (pour accéder aux données des expositions). Cela permet au service d'accéder à ces composants sans avoir à les instancier lui-même.

```
public function __construct(
    RequestStack $requestStack,
    TicketRepository $ticketRepo,
    ExhibitionShareRepository $exhibitionShareRepo)
{
    $this->ticketRepository = $ticketRepo;
    $this->requestStack = $requestStack;
    $this->exhibitionShareRepository = $exhibitionShareRepo;
}
```

Méthode getSession() : Elle récupère l'objet *SessionInterface* via *RequestStack*. Déclarée *private*, elle encapsule l'accès à la session, contrôlant ainsi son utilisation et masquant la complexité de sa récupération. Cette encapsulation renforce la modularité du code : seul *CartService* interagit directement avec la gestion de la session. En limitant l'accès à cette méthode, on applique le **principe de moindre privilège** (droits d'accès minimaux), améliorant la sécurité et la robustesse.

```
private function getSession()
{
    return $this->requestStack->getCurrentRequest()->getSession();
}
```

Méthode getCart() : Elle récupère le contenu du panier de l'utilisateur à partir de la **session**. Elle extrait la valeur associée à la clé *\$cart* sous forme de tableau. Si le panier n'existe pas encore en session, elle retourne un tableau vide, ce qui évite les erreurs lorsqu'on essaie de manipuler un panier qui n'a pas encore été initialisé. Cela facilite la manipulation du panier dans d'autres parties de l'application.

```
public function getCart(): array
{
    return $this->getSession()->get('cart', []);
}
```

Méthode `setCart(array $cart)` : Elle met à jour le tableau `$cart` en session sous la clé 'cart' à chaque action de l'utilisateur qui affecte le contenu de son panier (ajout, suppression, modification de quantité). Cela permet de **persister** ainsi le contenu du panier de l'utilisateur à travers ses requêtes. Le terme "set" indique l'assignation du tableau `$cart` à la clé 'cart' dans la session.

```
public function setCart(array $cart): void
{
    $this->getSession()->set('cart', $cart);
}
```

Méthode `addCart()` : Elle permet d'ajouter un ticket au panier en cliquant sur le plus.

```
***** Ajouter un produit au panier *****
public function addCart(Ticket $ticket, int $exhibitionId, int $qty = 1)
{
```

On récupère le panier actuel depuis la session. `$cart = $this->getCart();`

Mais aussi les détails du ticket via le `TicketRepository`, et l'exposition via `l'ExhibitionShareRepository`.

```
if ($ticket) {
    // Récupérer les informations via le repository
    $ticketId = $ticket->getId();
    $ticketDetails = $this->ticketRepository->findTicketDetails($ticketId);
```

On utilise une clé unique (`$exhibitionId.'_'.$ticketId`) pour identifier l'article dans le panier.

```
// Créer une clé unique combinant exhibitionId et ticketId
$cartKey = $exhibitionId.'_'.$ticketId;
```

Si l'article existe déjà, j'incrémente la quantité. Sinon, j'ajoute un nouvel élément au panier avec les informations du ticket, de l'exposition, la quantité et le prix.

```
// Si le ticket est déjà dans le panier pour cette exposition, on met à jour la quantité
if (isset($cart[$cartKey])) {
    $cart[$cartKey]['qty'] += $qty;
} else {
    // Sinon, on ajoute le ticket au panier avec ses informations
    $cart[$cartKey] = [
        'ticket' => $ticket,
        'ticketId' => $ticketId,
        'exhibition' => $exhibition,
        'exhibitionId' => $exhibitionId,
        'qty' => $qty,
        'price' => $price,
    ];
}
```

On calcule également le total de la ligne pour cet article (*totalLine*).

```
// Ajouter le total de la ligne  
$cart[$cartKey]['totalLine'] = $cart[$cartKey]['price'] * $cart[$cartKey]['qty'];
```

Puis on met à jour le panier dans la session et recalcule le total du panier.

```
// Sauvegarde du panier  
$this->setCart($cart);  
$this->updateCartTotal($cart);
```

Méthode **removeCart()** : Elle supprime 1 ticket en cliquant sur le moins.

```
***** Soustraire un produit *****  
public function removeCart(Ticket $ticket, int $exhibitionId, int $qty = 1)  
{
```

Récupère le panier et la clé de l'article par le biais de la clé unique lié au panier.

```
$cart = $this->getCart();  
$ticketId = $ticket->getId();  
$cartKey = $exhibitionId.'_'.$ticketId;
```

Si l'article existe, décrémente la quantité.

```
// Vérif si le ticket est présent dans le panier  
if (isset($cart[$cartKey])) {  
    $cart[$cartKey]['qty'] -= $qty;
```

Si la quantité devient inférieure ou égale à zéro, l'article est supprimé du panier.

```
//Suppression totale du ticket <= 0  
if ($cart[$cartKey]['qty'] <= 0) {  
    unset($cart[$cartKey]);  
}
```

Met à jour le panier dans la session.

```
$this->setCart($cart);
```

Méthode **clearCart()** : Elle permet de supprimer le panier complet.

```
public function clearCart(): void  
{  
    $this->getSession()->remove('cart');  
}
```

- **EmailService** : Il centralise l'envoi d'e-mails dans Symfony. Il utilise **l'injection de dépendances** pour utiliser *MailerInterface* pour l'envoi et Twig pour la mise en forme du contenu.

Voici les **méthodes** utilisées pour la fonctionnalité phare :

Méthode `send()` : Elle crée un objet *Email*, le configure (expéditeur, destinataire, sujet, corps HTML, pièce jointe) et l'envoie via *MailerInterface*. La méthode `renderTemplate()`

```
public function send(
    string $to, string $subject, string $body, ?array $attachment = null,
    string $from = 'noreply@regardsguerre.fr'
): void
{
    $email = (new Email())
        ->from($from)
        ->to($to)
        ->subject($subject)
        ->html($body);

    //Si le mail a des PJ alors on rajoute au mail
    if (is_array($attachment) && isset($attachment['content'], $attachment['filename'])) {

        $email->attach($attachment['content'], $attachment['filename'], $attachment
            ['mimeType'] ?? null);
    }

    $this->mailer->send($email);
}
```

Méthode `renderTemplate()` : Elle utilise *Twig* pour générer le contenu HTML de l'e-mail à partir d'un template et de variables. Ces méthodes encapsulent la logique d'envoi et de mise en forme des e-mails.

```
public function renderTemplate(string $templatePath, array $context = []): string
{
    return $this->twig->render($templatePath, $context);
}
```

- **StockAlertEmailService** : Il permet d'envoyer un mail à l'administrateur dès qu'un stock est presque épuisé ou épuisé.

```
public function sendStockAlertEmail(array $soonOutStockExhibits, array
    $outOfStockExhibitions): void
{
    $body = $this->emailService->renderTemplate('emails/stockAlertEmail.html.
    twig', [
        'soonOutStockExhibits' => $soonOutStockExhibits,
        'outOfStockExhibitions' => $outOfStockExhibitions,
    ]);

    $this->emailService->send('admin@regardsguerre.fr', 'Alerte de stock',
    $body);
}
```

- **OrderConfirmationEmailService** : Il se charge de générer le PDF de la commande et de l'envoyer à l'utilisateur.

Voici les **méthodes** utilisées pour la fonctionnalité phare :

Méthode

generateTicketPdf() :

```
public function generateTicketPdf(Order $order): ?array
{
    // Init du tabl pour les commandes regroupées
    $groupedOrderDetails = [];

    // Parcours des détails de la commande pour les regrouper par exposition
    foreach ($order->getOrderDetails() as $orderDetail) {
        $exhibitionId = $orderDetail->getExhibition()->getId();

        if (!isset($groupedOrderDetails[$exhibitionId])) {
            $groupedOrderDetails[$exhibitionId] = [
                'exhibition' => $orderDetail->getExhibition(),
                'orderDetails' => [],
            ];
        }
        // Ajoute le ticket à la liste
        $groupedOrderDetails[$exhibitionId]['orderDetails'][] = $orderDetail;
    }

    // Génère le contenu PDF (pdf service + template)
    $pdfContent = $this->pdfService->generatePdf(
        'pdf/eticketPdf.html.twig',
        [
            'order' => $order,
            'groupedOrderDetails' => $groupedOrderDetails,
        ]
    );

    // Retourne le contenu et le nom du fichier PDF
    return [
        'content' => $pdfContent,
        'filename' => 'eticket.pdf',
    ];
}
```

Méthode sendTicketEmail() :

```
public function sendTicketEmail(Order $order): void
{
    //Génère le pdf de la facture
    $orderPdf = $this->generateTicketPdf($order);

    if ($orderPdf) {
        $attachment = [
            'content' => $orderPdf['content'],
            'filename' => $orderPdf['filename'],
            'mimeType' => 'application/pdf',
        ];

        //Envoi l'email
        $this->emailService->send(
            $order->getCustomerEmail(),
            'Vos tickets',
            $this->emailService->renderTemplate(
                'emails/eticketEmail.html.twig',
                ['order' => $order]
            ),
            $attachment
        );
    }
}
```

➤ **OrderExportPdfService :**

Méthode `generateTicketsPdf()` : Elle génère les documents PDF (e-tickets) qui seront envoyés par e-mail. Elle récupère les informations de la commande et du panier, puis utilise un service PDF et un template Twig pour mettre en page chaque e-ticket. Enfin, elle retourne un tableau contenant le contenu PDF et le nom de fichier de chaque e-ticket.

```
public function generateTicketsPdf(int $orderId, array $groupedCart): ?array
{
    // Récupère l'entité Order à partir de son ID
    $order = $this->orderRepository->find($orderId);

    // Vérifie si la commande existe et est associée à un utilisateur
    if (!$order || !$order->getUser()) {
        return null; // Retourne null si la commande n'est pas trouvée
    }

    $eTickets = [];

    // Vérifie si le panier regroupé contient des articles
    if (isset($groupedCart['items'])) {
        // Parcours les articles du panier regroupé
        foreach ($groupedCart['items'] as $item) {
            // Vérifie si l'article est un ticket et est associé à une exposition
            if ($item['type'] === 'ticket' && isset($item['exposition'])) {

                // Génère le contenu PDF (pdf service + template)
                $pdfContentTicket = $this->pdfService->generatePdf(
                    'pdf/eticketPdf.html.twig',
                    ['item' => $item, 'order' => $order]
                );

                // Add le contenu et le nom du fichier de l'e-ticket au tableau
                $eTickets[] = [
                    'content' => $pdfContentTicket,
                    'filename' => 'e_ticket_' . $item['exposition'][ 'slug' ] . '_' . $orderId . '.pdf',
                ];
            }
        }
    }

    // Retourne le tableau des e-tickets
    return $eTickets;
}
```

➤ **PdfService :**

```
public function generatePdf(
    string $templatePath,
    array $data): string
{
    // Configuration des options de Dompdf
    $pdfOptions = new Options();
    $pdfOptions->set('defaultFont', 'Arial');
    $pdfOptions->set('isRemoteEnabled', true); // Pour placer le logo

    $dompdf = new Dompdf($pdfOptions); // Crée une instance de Dompdf avec les options

    // Génération du HTML à partir du template Twig
    $html = $this->twig->render(
        $templatePath,
        $data
    );

    // Mise en forme dompdf
    $dompdf->loadHtml($html); // Chargement du HTML dans Dompdf
    $dompdf->setPaper('A4', 'portrait'); // Config du format de papier et de l'orientation
    $dompdf->render(); // Génération du PDF

    return $dompdf->output(); // Retourne le contenu du PDF
}
```

Méthode `generatePdf()` :

Elle génère un PDF à partir d'un *template Twig*. Elle configure *Dompdf*, une bibliothèque PHP de conversion HTML en PDF, charge le HTML généré par Twig, définit le format du papier et retourne le contenu du PDF.

2. La commande

(Annexe 8 : Code de la commande)

L'utilisateur se rend sur la fiche de l'exposition pour effectuer sa **commande**.

Ce template présente la vue détaillée d'une entité *Exhibition*, offrant une interface pour la consultation et l'interaction avec les *TicketPricings* associés.

Tickets :				
Adulte	10.00 €	-	2	+
Enfant	8.00 €	-	3	+
Enfant -6ans	Gratuit	-	0	+
Panier				

La section des tickets itère (`{% for pricing in ticketPricings %}`) sur les différents types, affichant leur standardPrice. L'interaction se fait via des liens générant des requêtes *GET* vers les routes *removeTicketFromCart* et *addTicketToCart*.

Le paramètre origin dans l'URL permet un traitement contextuel par le contrôleur (*CartController*). La quantité dans le panier est rendue dynamiquement via la variable `$cart`, un tableau associatif basé sur une clé unique. Un bouton avec un lien hypertexte vers la route cart permet la navigation vers le récapitulatif.

3. Le panier

- **CartController** : Il est responsable de la gestion du panier d'achat de l'utilisateur. Il utilise le service *CartService* pour effectuer les opérations liées au panier (afficher, ajouter, supprimer, vider, calculer le total, regrouper les articles).

Le constructeur : Il injecte une instance du service *CartService*, rendant ses méthodes accessibles dans le contrôleur via la propriété `$this->cartService`.

```
public function __construct(  
    CartService $cartService)  
{  
    $this->cartService = $cartService;  
}
```

Méthode **showCart()** : Elle affiche le contenu du panier.

```
#[Route('/order/cart', name: 'cart')]  
public function showCart(?User $user): Response
```

Elle récupère le panier, le total et le panier regroupé par exposition en utilisant les méthodes correspondantes du *CartService*.

```
// Récupérer le panier depuis le service
$cart = $this->cartService->getCart();
$total = $this->cartService->getTotal();
$groupedCart = $this->cartService->groupCartByExhibition($cart);
```

Elle crée également une instance du formulaire *UserIdentityCartFormType*, qui est utilisé pour recueillir les informations d'identité de l'utilisateur avant la commande (si elles ne sont pas déjà renseignées).

```
//Obt pour commander
$form = $this->createForm(UserIdentityCartFormType::class);
```

Enfin, elle rend le template *order/cart.html.twig* en passant les informations du panier, le total et le formulaire. Le paramètre *?User \$user* permet d'accéder à l'utilisateur connecté.

```
return $this->render('order/cart.html.twig', [
    'groupedCart' => $groupedCart,
    'cart' => $cart,
    'total' => $total,
    'form' => $form->createView(),
]);
```

Méthode `addTicketToCart()`: Elle permet d'ajouter un ticket spécifique au panier. Elle reçoit l'ID de l'exposition (*exhibitionId*) et l'ID du ticket (*ticketId*) via les paramètres de la route, ainsi que l'origine de la requête (*origin*).

```
***** Ajoute un ticket au panier *****
#[Route('/ticket/{exhibitionId}/addTicketToCart/{ticketId}/{origin}', name: 'addTicketToCart')]
public function addTicketToCart()
```

Elle utilise le *TicketRepository* pour récupérer l'entité *Ticket* correspondant à l'*\$ticketId* et l'*ExhibitionShareRepository* pour récupérer l'entité *Exhibition* correspondant à l'*\$exhibitionId*.

```
TicketRepository $ticketRepo,
int $exhibitionId,
int $ticketId,
string $origin,
ExhibitionShareRepository $exhibitionShareRepo): Response
{
    // Récup du ticket via le repository
    $ticket = $ticketRepo->find($ticketId);

    // Récup de l'exposition via le repository en utilisant l'ID de la route
    $exhibition = $exhibitionShareRepo->find($exhibitionId);
```

Elle appelle la méthode `addCart()` du `CartService` pour ajouter le ticket à la session du panier, en fournissant l'entité `Ticket` et l'`$exhibitionId`.

```
// Ajout du ticket au panier via le service
$this->cartService->addCart($ticket, $exhibitionId);
```

Elle effectue une redirection en fonction de la valeur du paramètre `$origin`. Si `$origin` est `'listExhibit'`, l'utilisateur est redirigé vers la page de l'exposition. Sinon, il est redirigé vers la page du panier.

```
// Redirection en fonction de l'origine
if ($origin === 'listExhibit') {
    return $this->redirectToRoute('exhibition', ['slug' => $exhibition->getSlug()]);
}

// Redirection par défaut
return $this->redirectToRoute('cart', ['exhibition' => $exhibitionId]);
```

Méthode `removeTicketFromCart()`: Elle supprime un ticket du panier. Elle reçoit les mêmes paramètres que `addTicketToCart`.

```
***** Soustrait un produit au panier *****/
#[Route('/ticket/{exhibitionId}/removeTicketFromCart/{ticketId}/{origin}', name:
'removeTicketFromCart')]
public function removeTicketFromCart(
    TicketRepository $ticketRepo,
    int $exhibitionId,
    int $ticketId,
    string $origin,
    ExhibitionShareRepository $exhibitionRepo) : Response
{
    // Récupération du ticket via le repository
    $ticket = $ticketRepo->find($ticketId);

    // Récup de l'exposition via le repository en utilisant l'ID de la route
    $exhibition = $exhibitionRepo->find($exhibitionId);
```

Elle appelle la méthode `removeCart()` du `CartService` pour diminuer la quantité du ticket dans le panier pour l'exposition donnée.

```
// Soustraction au panier via le service
$this->cartService->removeCart($ticket, $exhibitionId);
```

Elle effectue une redirection vers la page de l'exposition si l'origine est `'listExhibit'`, ou vers la page du panier si l'origine est `'cart'`.

```
if ($origin === 'listExhibit') { // Si l'origine est 'listExhibit', rediriger vers la page de
    // ventes des tickets
    return $this->redirectToRoute('exhibition', ['slug' => $exhibition->getSlug()]);
}

if ($origin === 'cart') { // Si l'origine est 'cart', rediriger vers le panier
    return $this->redirectToRoute('cart', [
        'exhibition' => $exhibitionId,
    ]);
}

// Par défaut : rediriger vers le panier
return $this->redirectToRoute('cart');
```

Méthode **getTotal()** : Calcule le total du panier en parcourant tous les articles et en multipliant leur prix par leur quantité.

```
***** Calcul total du panier *****
public function getTotal(): float
{
    $cart = $this->getCart(); // Récupération du panier depuis la session
    $total = 0;

    foreach ($cart as $product) {
        $total += $product['price'] * $product['qty'];
    }

    return $total;
}
```

Méthode **updateCartTotal()** : Recalcule le total du panier en se basant sur le `$totalLine` de chaque article et met à jour la valeur '`cartTotal`' dans la session.

```
// Fonction pour mettre à jour le total du panier
public function updateCartTotal($cart)
{
    $total = 0;

    foreach ($cart as $product) {
        $total += $product['totalLine']; // Total de la ligne du ticket
    }

    // Mettre à jour le total dans la session
    $session = $this->requestStack->getCurrentRequest()->getSession();
    $session->set('cartTotal', $total);
}
```

Méthode **groupCartByExhibition()** : Elle regroupe les articles du panier par exposition.

```
***** Regroupement des achats *****
public function groupCartByExhibition(array $cart): array
```

On initialise un tableau. On boucle sur les expositions du panier. Si l'exposition n'existe pas on l'ajoute au tableau avec toutes ses informations et on initialise un tableau de tickets, sinon on passe à la suivante.

```
// Init tableau de regroupement
$groupedCart = [];

foreach ($cart as $product) {
    // Récupération des IDs pour plus de lisibilité
    $exhibitionId = $product['exhibitionId'];
    $ticketId = $product['ticketId'];

    // Si expo n'existe pas on la crée
    if (!isset($groupedCart[$exhibitionId])) {
        $groupedCart[$exhibitionId] = [
            'exhibition' => $product['exhibition'], // Add infos expo
            'tickets' => [] // Init tabl tickets
        ];
    }
}
```

Maintenant on vérifie si le ticket n'existe pas ou le crée avec toutes les informations (quantité, prix, nom) sinon on le crée.

```
// Si ticket n'existe pas ou le crée
if (!isset($groupedCart[$exhibitionId]['tickets'][$ticketId])) {
    $groupedCart[$exhibitionId]['tickets'][$ticketId] = [
        'ticket' => $product['ticket'], // Add infos ticket
        'quantity' => $product['qty'],
        'price' => $product['price'],
        // Ajout optionnel du total ligne si nécessaire
        'totalLine' => $product['price'] * $product['qty']
    ];
} else {
    // Si ticket existe on incrémente qty
    $groupedCart[$exhibitionId]['tickets'][$ticketId]['quantity'] += $product['qty'];
    // Mise à jour du total ligne si vous l'avez ajouté
    if (isset($groupedCart[$exhibitionId]['tickets'][$ticketId]['totalLine'])) {
        $groupedCart[$exhibitionId]['tickets'][$ticketId]['totalLine'] += $product['price'] *
            $product['qty'];
    }
}
```

4. Le paiement

Une fois que l'utilisateur a constitué ses achats, il se rend dans son panier.

Si les champs *userName* et *userFirstname* sont déjà renseignés dans son entité *User*, il visualise directement le récapitulatif de sa commande ainsi que le bouton de paiement.

Si son identité (nom et prénom) n'est pas renseignée dans son profil, un formulaire sera présent. Il est obligatoire de renseigner son identité pour commander, notamment en raison des obligations légales liées à la vente de tickets nominatifs et à la facturation. En effet, la vente de tickets nominatifs requiert l'identification de l'acheteur pour des raisons de conformité et de traçabilité.

Veuillez renseigner votre nom et prénom pour commander.

Nom

Prénom

Enregistrer mes informations pour mes prochaines commandes

L'utilisateur a également la possibilité de choisir de persister ou non ces données via la case à cocher *saveIdentity*, conformément au *RGPD* et au principe de *minimisation des données*.

Lors de la soumission de ce formulaire (*méthode HTTP POST*), les données sont transmises au contrôleur.

➤ PaymentController

Stripe est l'Application Programming Interface (API) qui a été utilisée. On envoie une *requête HTTP* à *l'API de Stripe*, mais la communication est immédiatement sécurisée et transite via le protocole HTTPS une fois la connexion établie avec les serveurs de Stripe. Cette requête contient les données (détails de paiement). *Stripe* traite la requête, effectue les opérations (débiter une carte), et renvoie une *réponse HTTP*. La réponse contient des données structurées indiquant le résultat (réussite/échec). *L'API de Stripe* est sécurisée : elle utilise des **clés d'authentification** et **chiffre** les données sensibles. Ce processus permet d'intégrer des paiements complexes sans gérer les détails techniques et de sécurité.

Voici les **méthodes** utilisées pour la fonctionnalité phare :

Méthode stripeCheckout() :

```
#Route('/order/create-session-stripe', name: 'paymentStripe')
public function stripeCheckout()
```

Ce contrôleur crée une instance du *UserIdentityCartFormType*.

```
$form = $this->createForm(UserIdentityCartFormType::class, $this->getUser());
```

Si les champs ont été remplis alors on réceptionne la requête.

```
//Si nom prenom !bdd
if (!$this->getUser()->getUserName() && !$this->getUser()->getUserFirstname()) {
    $form->handleRequest($request);
```

Ensuite si les données saisies sont valides à l'issue de la soumission du formulaire alors on sauvegarde les informations en session.

```
if ($form->isSubmitted() && $form->isValid()) {
    //Save en session car paiement non validé
    $session = $this->requestStack->getCurrentRequest()->getSession();
    $session->set('customerName', $form->get('userName')->getData());
    $session->set('customerFirstname', $form->get('userFirstname')->getData());
    $session->set('saveIdentity', $form->get('saveIdentity')->getData());
```

Sinon on notifie l'utilisateur d'un problème par le biais d'un message flash et on lui réaffiche son panier (via le *CartService*) ainsi que le formulaire.

```

} else {
    // Ajouter un message flash d'erreur
    $this->addFlash(
        'error',
        'Veuillez renseigner votre nom et prénom pour continuer la commande.'
    );

    // Réafficher le form avec les erreurs
    return $this->render('order/cart.html.twig', [
        'form' => $form->createView(), //Converti l'objet form en format Twig
        'groupedCart' => $this->cartService->groupCartByExhibition($this->cartService->getCart()),
        'cart' => $this->cartService->getCart(),
        'total' => $this->cartService->getTotal(),
    ]);
}

```

Une vérification du stock disponible est effectuée pour cela on récupère le panier.

```

//Récup panier pour vérif stock + préparation donnée pour Stripe
$cart = $this->cartService->getCart();

```

On initialise un tableau pour y stocker les expositions dont le stock ne correspond pas à la quantité demandée. Puis on effectue les vérifications en comparant la quantité demandée avec le stock maximal moins les tickets déjà réservés pour l'exposition concernée.

```

// Vérification init du stock et collecte des erreurs de stock
foreach ($cart as $item) {
    $exhibition = $exhibitionShareRepo->find($item['exhibitionId']); //Récup id expo like id dans panier
    if ($exhibition) {
        //Vérif si les tickets commandés sont dispos
        $ticketsAvailable = $exhibition->getStockMax() - $exhibition->getTicketsReserved();
        $qtyRequested = $item['qty']; //Qté demandée

        if ($qtyRequested > $ticketsAvailable) {
            $stockErrors[] = [
                'exhibitionTitle' => $exhibition,
                'ticketsAvailable' => $ticketsAvailable,
            ];
        }
    }
}

```

Si un stock insuffisant est détecté pour un ou plusieurs articles, un message flash de type danger est affiché pour chaque erreur (nom de l'exposition ainsi que le nombre de ticket restant disponible), informant l'utilisateur du problème, et il est redirigé vers la page du panier pour modifier sa commande.

```

if (!empty($stockErrors)) {
    foreach ($stockErrors as $error) {
        $this->addFlash(
            'danger',
            sprintf( //Permet concaténation string + variable
                'Stock insuffisant pour l\'exposition "%s". (%d restants).',
                $error['exhibitionTitle'],
                $error['ticketsAvailable']
            )
        );
    }
}

```

Si le stock est suffisant pour tous les articles, un tableau contenant les informations nécessaires pour l'API Stripe (`$productStripe`) est initialisé.

Pour chaque article du panier, les informations du ticket (nom) et le prix unitaire sont récupérés, et un tableau structuré pour Stripe est créé, incluant la devise (EUR) et la quantité.

```
$productStripe = [];

foreach ($cart as $product) {
    $ticket = $ticketRepo->find($product['ticketId']);
    $exhibition = $exhibitionShareRepo->find($product['exhibitionId']);
    $qty = $product['qty'];
    $price = $product['price'];

    $productStripe[] = [
        'price_data' => [
            'currency' => 'eur',
            'product_data' => [
                'name' => $ticket->getTitleTicket(),
            ],
            'unit_amount' => $price * 100, // Prix en centimes
        ],
        'quantity' => $qty,
    ];
}
```

On configure la clé API de Stripe pour l'authentification (clé à sécuriser dans le fichier `.env` comme pour toutes les données sensibles)

```
Stripe::setApiKey($_ENV['STRIPE_SECRET_KEY']);
```

Et on génère les URLs de succès et d'échec de paiement.

```
$successUrl = $urlGenerator->generate('paymentSuccess', [], UrlGeneratorInterface::ABSOLUTE_URL);
$cancelUrl = $urlGenerator->generate('paymentError', [], UrlGeneratorInterface::ABSOLUTE_URL);
```

Une session de paiement Stripe est alors créée via l'API Stripe, incluant l'adresse e-mail de l'utilisateur (`$this->getUser()->getUserIdentifier()`), le mode de paiement (`payment_method_types` réglé sur `'card'`), les `line_items` représentant les produits du panier (basés sur le tableau `$productStripe`) et les URLs de succès (`success_url` générée pour la route `paymentsuccess`) et d'annulation (`cancel_url` générée pour la route `paymentError`).

```
$checkoutSession = Session::create([
    'customer_email' => $this->getUser()->getUserIdentifier(),
    'payment_method_types' => ['card'],
    'line_items' => $productStripe,
    'mode' => 'payment',
    'success_url' => $successUrl,
    'cancel_url' => $cancelUrl,
]);
```

Enfin, l'utilisateur est redirigé vers l'URL de paiement fournie par Stripe.

```
return new RedirectResponse($checkoutSession->url, 303);
```

En cas d'échec du paiement (retour via *cancel_url* et la route *paymentError*), l'utilisateur est redirigé vers la page du panier avec un message flash de type danger indiquant une erreur de paiement.

```
#[Route('/order/error', name: 'paymentError')]
public function stripeError(
    CartService $cartService,
): Response
{
    $this->addFlash('danger', 'Une erreur est survenue lors du paiement. Veuillez réessayer.');

    return $this->redirectToRoute('cart');
}
```

En cas de succès du paiement (retour via *success_url* et la route *paymentsuccess*), les informations du formulaire stockées en session sont récupérées.

```
//Récup des infos du form stockés en session
$session = $requestStack->getCurrentRequest()->getSession();
$customerName = $session->get('customerName');
$customerFirstname = $session->get('customerFirstname');
$saveIdentity = $session->get('saveIdentity');
```

Si l'utilisateur a activé l'option d'enregistrement (*saveIdentity* est *true*), les propriétés *userName* et *userFirstname* de l'entité User sont mises à jour avec les données de session, puis l'*EntityManager Doctrine* est utilisé pour persister et enregistrer ces modifications en base de données. Un message flash de type *success* informe l'utilisateur de cet enregistrement.

```
$user = $this->getUser(); //Récup user co

//Si $saveIdentity = true alors enregistrement dans user
if ($saveIdentity && $user) {
    $user->setUserName($customerName);
    $user->setUserFirstname($customerFirstname);
    $this->entityManager->persist($user);
    $this->entityManager->flush();
    $this->addFlash('success', 'Vos informations ont été enregistrées pour vos prochaines commandes.');
}
```

Ensuite, le contenu du panier est récupéré via le *CartService*.

```
$cart = $this->cartService->getCart(); /
```

Une **nouvelle instance** de l'entité *Order* est créée et hydratée avec les informations pertinentes : *orderDateCreation* (avec un objet *\DateTimeImmutable*), l'entité User associée, *orderStatus* initialisé à 'Envoyé', le nom et prénom du client (priorité aux données de session, sinon celles de l'entité User), l'adresse e-mail de l'utilisateur et le *orderTotal* calculé par le *CartService*.

```
// Vérif si un user est co ET si son nom et prénom ne sont pas vides
$order = new Order();
$order->setOrderDateCreation(new \DateTimeImmutable()); // //Avec une date immuable (const)
$order->setUser($this->getUser()); // Associe la commande au user co
$order->setOrderStatus('Envoyé');

// Récup le nom et prénom du client. Priorité à la session (formulaire), sinon BDD de l'utilisateur connecté.
$order->setCustomerName($customerName);
$order->setCustomerFirstname($customerFirstname);

$order->setCustomerEmail($this->getUser()->getUserIdentifier()); // Enregistre l'email de l'utilisateur

// Enregistrement du total de la commande
$total = $cartService->getTotal(); // Récupère le total du panier
$order->setOrderTotal($total); // Enregistre le total dans la commande
```

L'*EntityManager Doctrine* prépare et exécute l'enregistrement de cette entité *Order* en base de données. On itère ensuite sur les éléments du panier pour créer une nouvelle instance de l'entité *OrderDetail* pour chaque article. Chaque *OrderDetail* est hydraté avec l'entité *Order* associée, l'entité *ExhibitionShare* et l'entité *Ticket* correspondantes (retrouvées via leurs repositories), la *quantity* et le *unitPrice*.

```
// Création des détails de la commande
foreach ($cart as $item) {
    $orderDetail = new OrderDetail();
    $orderDetail->setOrder($order);

    // Charger l'objet Exhibition à partir de l'ID
    $exhibition = $exhibitShareRepo->find($item['exhibitionId']);
    if ($exhibition) {
        $orderDetail->setExhibition($exhibition);
    }

    // Charger l'objet Ticket à partir de l'ID
    $ticket = $ticketRepo->find($item['ticketId']);
    if ($ticket) {
        $orderDetail->setTicket($ticket);
    }

    $orderDetail->setQuantity($item['qty']);
    $orderDetail->setUnitPrice($item['price']);

    $this->entityManager->persist($orderDetail);
}
```

Ces entités *OrderDetail* sont ensuite persistées via l'*EntityManager*.

```

// Persist de l'order AVANT de générer le numéro de facture
$this->entityManager->persist($order);
$this->entityManager->flush();

```

L'entité *Order* est rafraîchie (`$this->entityManager->refresh($order)`) afin de récupérer l'ID généré par la base de données, qui sera utilisé pour la création du numéro de facture unique.

```

// Rafraîchir l'entité pour récupérer l'ID généré par la bdd
$this->entityManager->refresh($order);

```

Une nouvelle instance de l'entité *Invoice* est **instanciée et hydratée** avec les informations de la commande : *customerName*, *customerFirstname*, *customerEmail*, *orderTotal* et *dateInvoice* (avec un objet `\DateTimeImmutable`).

```

// Création de la facture
$invoice = new Invoice();
$invoice->setCustomerName($order->getCustomerName());
$invoice->setCustomerFirstname($order->getCustomerFirstname());
$invoice->setCustomerEmail($order->getCustomerEmail());
$invoice->setOrderTotal($order->getOrderTotal());
$invoice->setDateInvoice(new \DateTimeImmutable()); // Utilise une date immuable

```

Un *numberInvoice* unique est généré en utilisant l'ID de la commande et la date de création de la commande (*orderDateCreation*). Ce numéro est ensuite attribué aux propriétés *numberInvoice* des entités *Order* et *Invoice*.

```

// Génération du numéro de facture unique
$orderId = $order->getId();
$orderDate = $order->getOrderDateCreation()->format('Ymd');

$invoiceNumber = sprintf('%s-%s', $orderDate, $orderId);
$order->setNumberInvoice($invoiceNumber);
$invoice->setNumberInvoice($invoiceNumber);

```

Un *slug* unique et lisible pour l'administrateur est créé en combinant l'ID de l'utilisateur, son nom et son prénom, et est attribué à la propriété *slug* de l'entité *Invoice*.

```

// Génération du slug avec l'ID, le nom et le prénom de l'utilisateur
$userId = $order->getUser()->getId(); // Récupère l'ID de l'utilisateur à partir de l'entité $order
$userName = $invoice->getCustomerName();
$userFirstname = $invoice->getCustomerFirstname();

$slug = sprintf('%d-%s-%s', $userId, $userName, $userFirstname);
$invoice->setSlug($slug);

```

On itère sur le panier pour extraire les détails de chaque article (titres de l'exposition et du ticket, prix, quantité) et les structurer dans un tableau `$invoiceDetails`. Ce tableau, converti en **JSON** par **sérialisation**, est ensuite affecté à la propriété `invoiceDetails` de l'entité `Invoice` pour être stocké en base de données. (Lors de l'affichage de la facture, ces données JSON seront **désérialisées** pour retrouver la structure PHP originale.)

```
$invoiceDetails = []; //Détail de la commande

//Récup les éléments du panier
foreach ($cart as $item) {
    $exhibition = $exhibitShareRepo->find($item['exhibitionId']);
    $ticket = $ticketRepo->find($item['ticketId']);
    $quantity = $item['qty'];
    $price = $item['price'];

    $invoiceDetails[] = [
        'exhibitionTitle' => $exhibition->getTitleExhibit(),
        'ticketTitle' => $ticket->getTitleTicket(),
        'standardPrice' => $price,
        'quantity' => $quantity,
    ];
}

$invoice->setInvoiceDetails($invoiceDetails);
```

Ensuite, l'entité `Order` et l'entité `Invoice` sont **stockées et enregistrées** en base de données via l'`EntityManager`.

```
$this->entityManager->persist($order);
$this->entityManager->persist($invoice);
$this->entityManager->flush();
```

L'envoi de la confirmation de commande à l'utilisateur est géré par le service `OrderConfirmationEmailService` via sa méthode `sendTicketEmail()`. Ce service utilise le `EmailService` pour l'envoi d'e-mails et un service de génération de PDF pour les e-tickets.

```
...
$this->orderConfirmEmailService->sendTicketEmail($order);
```

L'envoi d'une alerte de stock à l'administrateur est géré par le service `StockAlertEmailService` via sa méthode `sendStockAlertEmail()`.

Avant cet envoi, on regroupe les articles du panier l'entité `Exhibition` est rafraîchie pour obtenir les dernières informations de stock.

```
$groupedCart = $this->cartService->groupCartByExhibition($cart);

$this->entityManager->refresh($exhibition); //rafraîchit l'expo
```

Ensuite, on boucle sur le `$groupedCart` (le panier regroupé par exposition) pour vérifier le stock restant de chaque exposition. Si le stock restant est inférieur ou égal au seuil d'alerte (`stockAlert`) et supérieur à zéro, l'exposition est ajoutée au tableau `$soonOutStockExhibits`. Si le stock est à zéro, l'exposition est ajoutée au tableau `$outOfStockExhibitions`.

```

// Vérification des stocks APRÈS l'enregistrement de la commande en parcourant le groupedCart
foreach ($groupedCart as $exhibitionId => $items) { //Parcours groupedCart (tabl AM)
    $exhibition = $exhibitShareRepo->find($exhibitionId);

    if ($exhibition) {
        $remainingStock = $exhibition->getStockMax() - $exhibition->getTicketsReserved(); // Calcul du stock restant

        if ($remainingStock <= $exhibition->getStockAlert() && $remainingStock > 0 && !in_array($exhibition, $soonOutStockExhibits)) {
            $soonOutStockExhibits[] = $exhibition; // si stock presque épuisé alors expo dans le tableau
        }
        if ($remainingStock <= 0 && !in_array($exhibition, $outOfStockExhibitions)) {
            $outOfStockExhibitions[] = $exhibition; // si stock épuisé alors expo dans le tableau
        }
    }
}

```

Le service *StockAlertEmailService* utilise ensuite le *EmailService* pour envoyer un e-mail à l'administrateur si l'un de ces tableaux n'est pas vide.

```

if (!empty($soonOutStockExhibits) || !empty($outOfStockExhibitions)) {
    $this->stockAlertEmailService->sendStockAlertEmail(array_unique($soonOutStockExhibits),
    array_unique($outOfStockExhibitions));
}

```

Enfin, le panier de l'utilisateur est vidé en utilisant la méthode *clearCart()* du service *CartService*. L'utilisateur est ensuite redirigé vers une page de succès de commande (route *orderSuccess*).

```

    $this->cartService->clearCart();

    return $this->redirectToRoute('orderSuccess');
}

```

VII Tests manuels

Des tests manuels ont été effectués pour garantir le bon fonctionnement de l'application.

- **Objectif :** Simuler des scénarios réels pour repérer les dysfonctionnements et assurer la conformité avec les spécifications.
- **Méthodologie :** Les tests ont ciblé les interfaces utilisateurs et les fonctionnalités critiques, en simulant divers scénarios pour couvrir un large éventail de cas d'usage.
- **Test fonctionnels :** Vérification de la fonctionnalité des éléments clés (formulaire, envoi de mails, connexion, vérification des authentification, etc).
- **Résultats et ajustements :** Des problèmes d'affichage et d'expérience utilisateur ont été corrigés suite aux tests, avec des rapports générés pour suivre les ajustements.

Les tests manuels ont permis de détecter des erreurs non identifiées lors du développement et ont contribué à l'amélioration du code.

VIII Axes d'améliorations

1. Planificateur

- **Objectif :** Automatiser l'anonymisation des données à une date précise.
- **Avantages :** Pas de tâche répétitive pour l'humain à vérifier, avec risque potentiel d'oubli.
- **Pratiques courantes :** Le fait que cela soit automatique, la veille humaine demeure moins récurrente.
- **Mise en place :**
 - Installation du composant **Messenger** de Symfony.
 - Création de la Command de planification
 - Configuration de la tâche (**cron**)
 -

2. Tests Unitaires

- **Objectif :** Garantir que chaque composant du projet fonctionne de manière autonome.
- **Avantages :** Permet de détecter rapidement les erreurs de logique, de faciliter la maintenance lors des évolutions du code et d'éviter les régressions.
- **Pratiques courantes :** Tests sur les méthodes de services, les entités, les contrôleur ; vérification des cas standards mais aussi des cas limites et des erreurs.
- **Mise en place :**
 - Utilisation du framework **PHPUnit**.
 - Intégration dans le **pipeline d'Intégration Continue** (ex. Github Actions, Gitlab CI).

3. Blog

- **Objectif :** L'ajout d'un blog enrichirait l'expérience utilisateur en offrant des contenus réguliers et interactifs sur les expositions passées.

- **Avantages** : L'ajout d'un blog permettrait d'enrichir l'expérience utilisateur en proposant des articles et mises à jour réguliers, tout en encourageant la participation anonyme via des pseudonymes et en garantissant un environnement respectueux grâce à un système de modération.
- **Pratiques courantes** : L'implémentation de pseudonymes favorise l'engagement tout en garantissant l'anonymat, tandis qu'une charte d'utilisation établit des règles claires pour assurer un comportement approprié, soutenu par un système de modération pour surveiller les échanges.
- **Mise en place :**
 - Requête pour les expositions passées : Afficher facilement l'historique des événements.
 - Création de pseudonymes : Permettre aux utilisateurs de participer sans dévoiler leur identité.
 - Mise à jour de la charte : Obliger la prise de connaissance lors de la création du pseudonyme.
 - Attribution de rôles de modérateurs : Assurer la qualité et la sécurité des échanges.
 - Système de commentaires : Permettre des échanges sous modération.

Ainsi, en intégrant des tests unitaires rigoureux, en optimisant le processus de mise en production et en enrichissant l'expérience utilisateur avec un blog interactif, nous pourrions non seulement garantir la fiabilité du projet, mais aussi assurer sa stabilité et favoriser l'engagement des utilisateurs.

CONCLUSION

Ce dossier présente le processus de conception et de développement d'un projet fictif qui, je l'espère, pourrait se concrétiser à l'avenir. L'objectif était de créer une plateforme fluide, intuitive et accessible, tout en mettant l'accent sur l'expérience utilisateur, tant sur mobile que sur desktop. Chaque étape a été pensée pour répondre aux besoins d'une interface simple, claire et performante.

Bien que ce projet soit encore imaginaire et qu'il n'existe rien de tel à ma connaissance, j'aimerais vraiment qu'il prenne vie. Je trouve qu'il est important de mettre en lumière des artistes qui abordent des aspects souvent oubliés de la guerre. Cette plateforme pourrait offrir une perspective enrichissante sur le sujet, et je crois sincèrement qu'elle apporterait une nouvelle manière de comprendre et de vivre l'histoire à travers l'art.

Ce travail représente un pas vers une idée que j'espère un jour voir concrétiser, apportant une valeur ajoutée à la culture et à la réflexion collective.

ANNEXES

Annexe 1 : Back-office

Gestion des tickets

Cette page offre une vue d'ensemble des stocks de tickets pour chaque exposition.

L'icône ✘ signifie que les tickets sont épuisés.

L'icône ⚠ signifie que les tickets sont presque épuisés.

Cliquez sur l'icône 🖍 pour modifier le stock de tickets d'une exposition.

Filtrer par : Tout afficher ▾

Exposition	Date	Stock	Stock d'alerte	Réserve	Reste
Les camps d'Algérie	(25 mai 2025)	150	10	10	140
L'Ukraine en résistance	(3 juin 2025)	150	10	11	139

Liste des factures clients

Filtrer : Tous ▾

Client	Mail
Cra Moisi	cramoisi@gmail.com
Lou Foque	maxLaMenace@gmail.com
Lou Foque	marouan@gmail.com

Liste des artistes

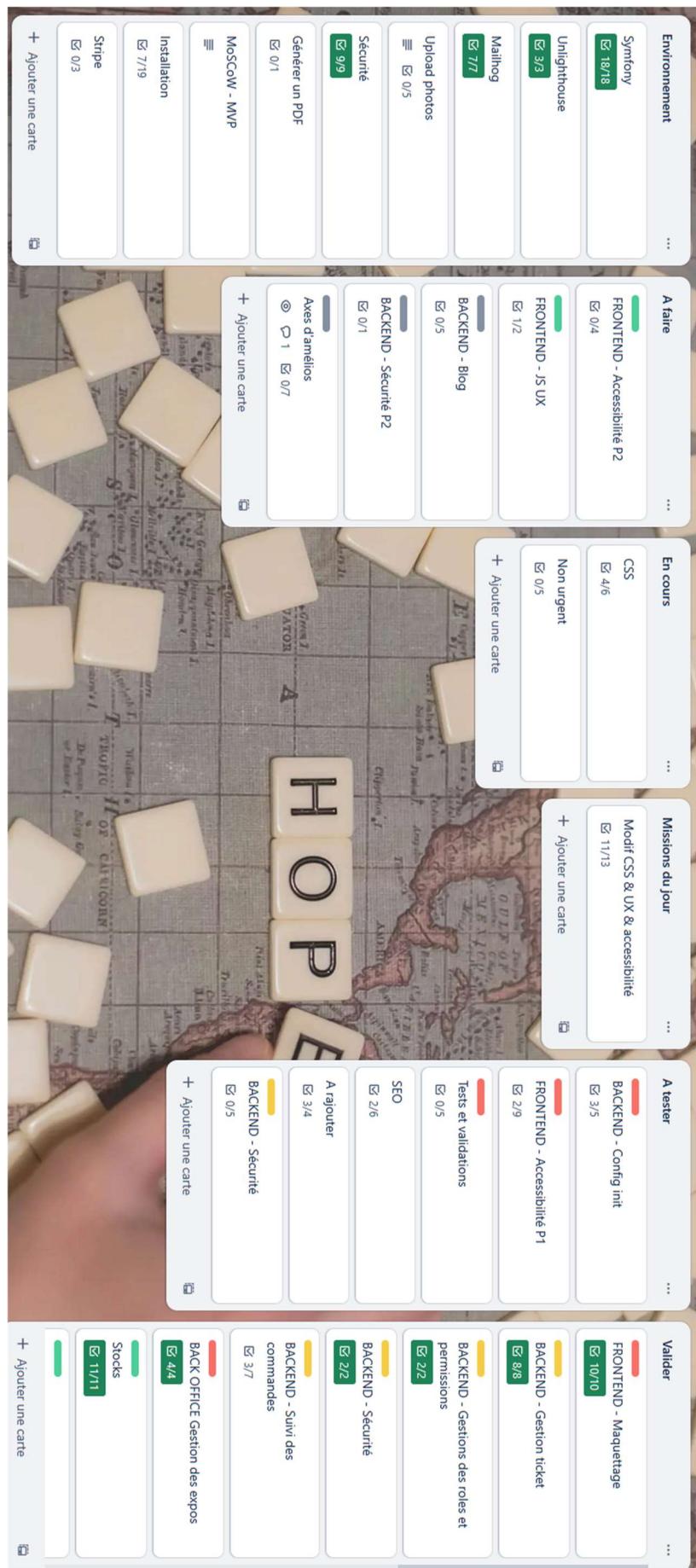
L'icône 🖍 permet de modifier les informations d'un artiste.

L'icône 🗑 permet de supprimer un artiste.

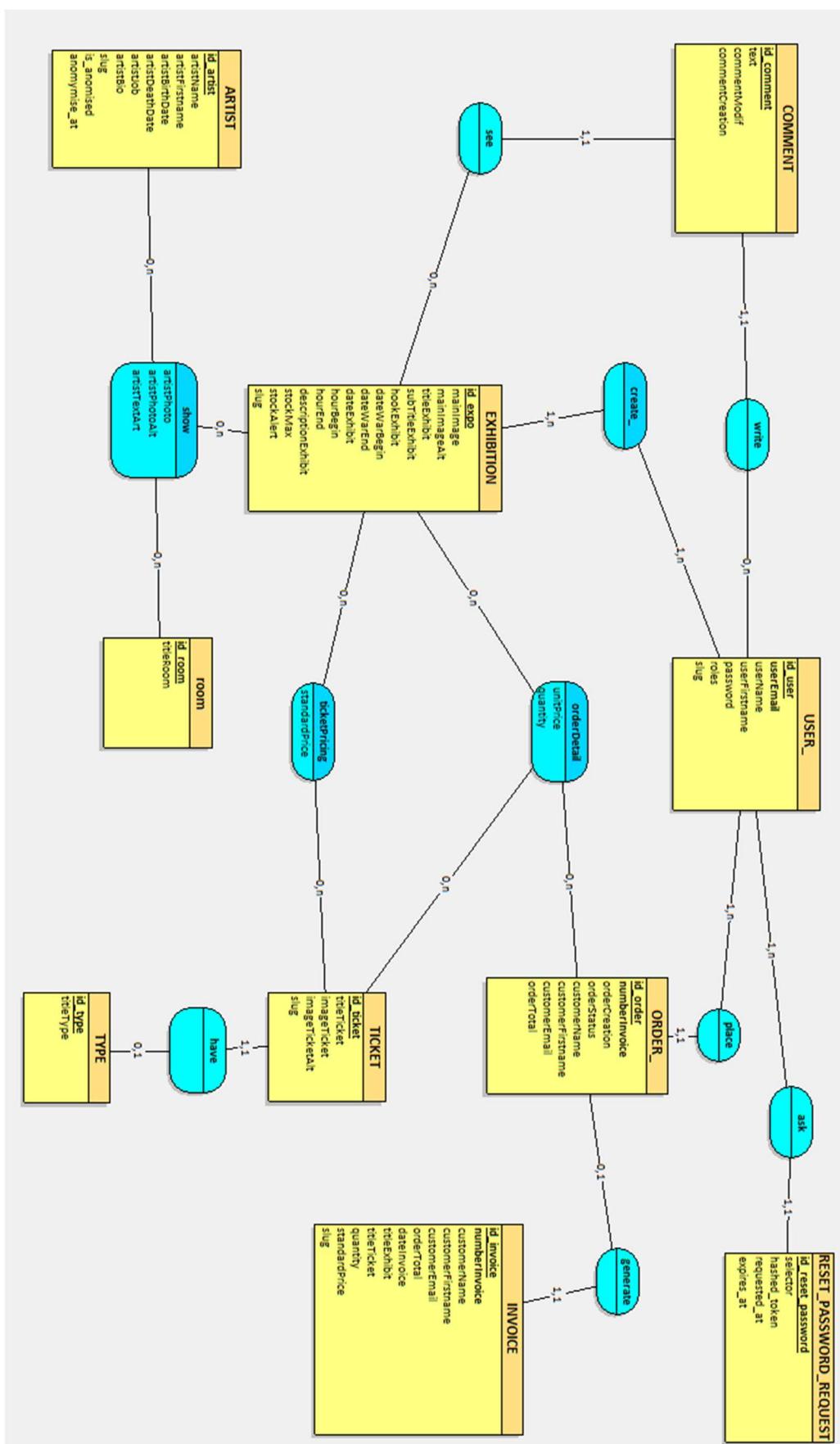
Filtrer par : Tout afficher ▾

Ajouter un artiste

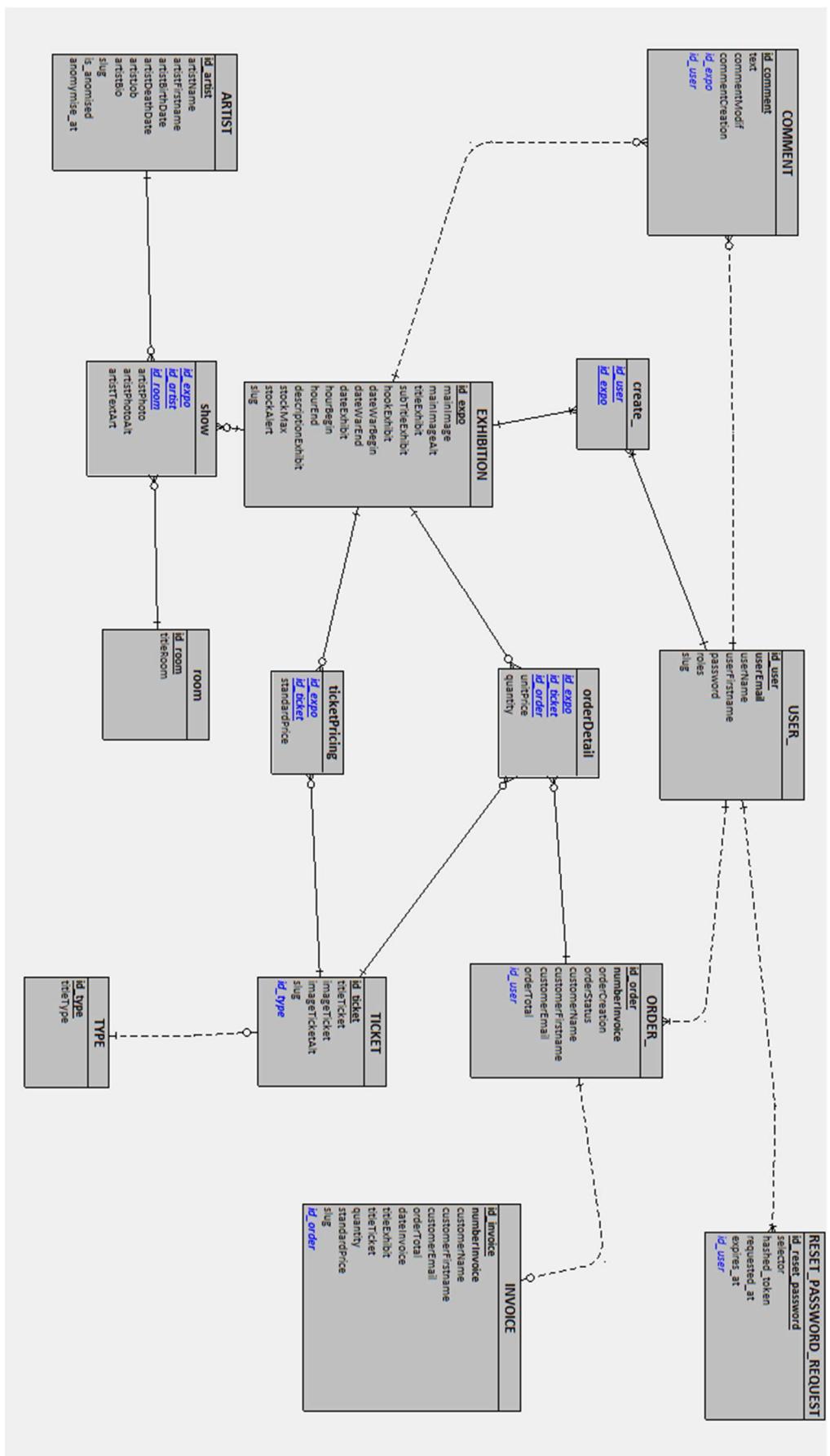
Annexe 2 : Trello



Annexe 3 : MCD



Annexe 4 : MLD



Annexe 5 : Maquettes mobile first

Planifiez votre visite

Rue de la Paix - 75000 PARIS

Nos locaux sont complètement accessibles pour les personnes en situation de handicap.



The map shows the Rue de la Paix area in Paris, with the exhibition space marked by a red pin. Other landmarks like the Palais Garnier, Opéra, and Ritz Paris are also visible.

Anna Garrigue - poétesse
(14 mars 1931 - 06 janvier 1966)



Née en Algérie, Anna Garrigue était une militante et poétesse engagée dans la lutte pour l'indépendance de l'Algérie. Emprisonnée pendant la guerre, elle a écrit des poèmes marqués par la douleur de l'exil et le combat pour la liberté.

Le travail poétique d'Anna Garrigue est profondément marqué par son engagement politique et son amour pour l'Algérie, qu'elle décrit à travers une écriture intense et émotive. Ses poèmes, souvent chargés de nostalgie et de résistance, abordent des thèmes de lutte, de mémoire et de réconciliation, avec une voix féminine forte et poignante.

Exposition : Salle Nicolas

Logo /
Regards de guerres



Panier

04 juin 2025 - La grande migration afro-américaine

Ticket adulte dématérialisé - 1 + 10 €

Ticket enfant dématérialisé - 1 + 8 €



Jean Moribon - photographe
(13 septembre 1925 - 03 novembre 2018)

Jean Moribon était un photographe suisse humaniste, reconnu pour ses reportages poignants sur les conflits et les crises sociales, notamment la guerre d'Algérie.

Son travail se distingue par une approche profondément humaniste, où il capte les souffrances et les émotions des civils dans des situations de guerre, notamment pendant la guerre d'Algérie. Ses photographies vont au-delà de l'image de la violence, en mettant l'accent sur la dignité et la résilience des personnes confrontées à des conditions extrêmes, offrant ainsi un témoignage puissant de leur réalité.

Exposition : Salle Sabandra

26 novembre 2025 - Les camps d'Algérie

Ticket enfant dématérialisé - 1 + 8 €

Total TTC 28 €

Commander

Annexe 6 : Maquettes desktop

Exposition

Les camps d'Algérie (1954 - 1962)

Les camps d'Algérie, créés pendant la guerre d'indépendance, sont devenus des symboles de l'exil, de la souffrance et du déracinement, où les conditions de vie des populations déplacées étaient marquées par l'humiliation et l'abandon.

09 septembre 2024
09h00 - 16h00

Tickets



Création de la page exposition

Titre Image

Dates de la guerre
Début Fin
Aujourd'hui

Dates de l'exposition
Date expo Heure début : Heure fin :

Description

Profil

[Modification de mes coordonnées](#)

[Historique de mes commandes](#)

[Modification de mon compte](#)

Nom

Renau

Prénom

Laury

Adresse

202 avenue de Colmar

Code Postal

67000 STRASBOURG

Ville

Annexe 7 : Benchmark

SHOP
EDITIONS
TICKETS

EXHIBITIONS



MON, 27.01. –
SUN, 25.05.
10H – 18H

NORTHERN LIGHTS
To visit the exhibition

EXHIBITION
E-TICKETS



FRI, 09.05. –
SUN, 18.05.
10H – 18H

SPECIAL-TICKET FOR THE ESC SHOW «OVER THE RAINBOW»
From 9–18 May 2025, a special collection presentation will be shown for the ESC in Basel. Admission for adults is reduced and includes access to the "Northern Lights" exhibition.

EXHIBITION
E-TICKETS

Musée Beyeler



**MON, 27.01. –
SUN, 25.05.
10h – 18h**

NORTHERN LIGHTS

To visit the exhibition "Over the Rainbow", which will be on display from 9 May to 18 May 2025, alongside the "Northern Lights" exhibition, please book only the special ticket.

The exhibition "Northern Lights" presents 74 landscape paintings by artists from Scandinavia and Canada produced between 1888 and 1937, among them masterpieces by Hilma af Klint and Edvard Munch. These artists all share the boreal forest as a common source of inspiration. The seemingly boundless expanses of the forest, the radiant light of endless summer days, the long winter nights and natural phenomena such as the northern lights gave rise to a specifically Nordic form of modern painting, which to this day exerts enduring appeal and fascination. The boreal forest, which stretches south and north of the polar circle, forming one of our planet's largest primeval forests, was increasingly represented as a spiritual landscape. The exhibition provides an opportunity to trace the development of Nordic landscape painting in modern art through selected works by Helmi Biese, Anna Boberg, Emily Carr, Prince Eugen, Gustaf Fjæstad, Akseli Gallen-Kallela, Lawren S. Harris, Hilma af Klint, J. E. H. MacDonald, Edvard Munch, Ivan Shishkin, Harald Sohlberg and Tom Thomson, in the process discovering artists likely still unknown to many visitors.

ADULTS	CHF 25.00	<input type="button" value="-"/>	<input type="button" value="0"/>	<input type="button" value="+"/>
ADMISSION IV*	CHF 20.00	<input type="button" value="-"/>	<input type="button" value="0"/>	<input type="button" value="+"/>
ADMISSION BASELCARD*	CHF 12.50	<input type="button" value="-"/>	<input type="button" value="0"/>	<input type="button" value="+"/>
ADMISSION UP TO 25*	CHF 0.00	<input type="button" value="-"/>	<input type="button" value="0"/>	<input type="button" value="+"/>

[ADD TO CART](#)



AUGUST 09-11 2024

A beloved local film festival that champions new voices and classic favorites.

2024 Programming

- The Supremes At Earl's All-You-Can-Eat**
- Daughters**
- My First Film**

Hudson Film Festival

Join us in Hudson for our second annual film festival this summer.



Building community through cinema.

[Get Your Tickets in MoviePass®](#)

Hudson Film Festival

Building community through cinema in Hudson, New York

Follow Us On Social

Annexe 8 : Code de la commande

```
{% if ticketPricings is not empty %}

    {% for pricing in ticketPricings %}
        <div class="ticketExhibit">
            <p class="titleTicket">{{ pricing.ticket.titleTicket }}</p>

            <div class="blockTitleQtyTicket">
                <div class="blockPriceTicket">
                    {% if pricing.standardPrice > 0 %}
                        <p class="price">{{ pricing.standardPrice }} €</p>
                    {% else %}
                        <p class="price">Gratuit</p>
                    {% endif %}
                </div>

                <div class="blockQtyTicket">

                    <a href="{{ path('removeTicketFromCart', {'exhibitionId': exhibition.id, 'ticketId': pricing.ticket.id, 'origin': 'listExhibit'}) }}" aria-label="{{ 'Retirer un ticket ' ~ pricing.ticket.titleTicket|lower ~ ' pour l\'exposition ' ~ exhibition.titleExhibit }}"
                    title="{{ 'Diminuer la quantité du ticket ' ~ pricing.ticket.titleTicket ~ ' pour l\'exposition ' ~ exhibition.titleExhibit }}"
                    ><i class="fa-solid fa-minus"></i>
                </a>

                    <div class="qtyProduct">
                        {# Crée une clé unique pour le ticket dans le panier (id expo + id ticket) #}
                        {% set cartKey = exhibition.id ~ '_' ~ pricing.ticket.id %}

                        {# Vérifie si le ticket est déjà dans le panier #}
                        {% if cart[cartKey] is defined %}
                            <span class="nbCart">{{ cart[cartKey].qty }}</span>
                        {% else %}
                            <span class="nbCart">0</span>
                        {% endif %}
                    </div>

                    <a href="{{ path('addTicketToCart', {'exhibitionId': exhibition.id, 'ticketId': pricing.ticket.id, 'origin': 'listExhibit'}) }}"
                    aria-label="{{ 'Ajouter un ticket ' ~ pricing.ticket.titleTicket|lower ~ ' pour l\'exposition ' ~ exhibition.titleExhibit }}"
                    title="{{ 'Augmenter la quantité du ticket ' ~ pricing.ticket.titleTicket ~ ' pour l\'exposition ' ~ exhibition.titleExhibit }}"
                    ><i class="fa-solid fa-plus"></i>
                </a>
            </div>
        </div>
    {% endfor %}
{% endif %}
```