Ce template présente la vue détaillée d'une entité Exhibition, offrant une interface pour la consultation et l'interaction avec les TicketPricings associés. L'utilisateur visualise les attributs de l'exposition (titre, dates, etc.) et son asset image principal. Des indicateurs signalent l'état du stock (ticketsRemaining). La section des tickets itère ({% for pricing in ticketPricings %}) sur les différents types, affichant leur standardPrice. L'interaction se fait via des liens (<a>) 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 (<button>) avec un lien hypertexte vers la route cart permet la navigation vers le récapitulatif. Des informations sur les Show et leurs Artist associés sont également affichées via une boucle.

--- cart controller + cart service  
Le contrôleur CartController 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).

CartController :

\_\_construct(CartService $cartService) : Le constructeur injecte une instance du service CartService, rendant ses méthodes accessibles dans le contrôleur via la propriété $this->cartService.

showCart(?User $user): Response (Route :/order/cart, nom : cart) :

Cette action affiche le contenu du panier. Elle récupère le panier, le total et le panier regroupé par exposition en utilisant les méthodes correspondantes du CartService.

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).

Enfin, elle rend le template order/cart.html.twig en passant les informations du panier, le total et le formulaire. Le paramètre optionnel ?User $user permet d'accéder à l'utilisateur connecté, bien qu'il ne soit pas directement utilisé dans la logique d'affichage du panier dans ce code.

addTicketToCart(TicketRepository $ticketRepo, int $exhibitionId, int $ticketId, string $origin, ExhibitionShareRepository $exhibitionRepo): Response (Route : /ticket/{exhibitionId}/addTicketToCart/{ticketId}/{origin}, nom : addTicketToCart) :

Cette action ajoute 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).

Elle utilise le TicketRepository pour récupérer l'entité Ticket

correspondant à l'$ticketId et l'ExhibitionShareRepository pour récupérer l'entité ExhibitionShare correspondant à l'$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.

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.

removeTicketFromCart(TicketRepository $ticketRepo, int $exhibitionId, int $ticketId, string $origin, ExhibitionShareRepository $exhibitionRepo) (Route :/ticket/{exhibitionId}/removeTicketFromCart/{ticketId}/{origin}, nom : removeTicketFromCart) :

Cette action soustrait un ticket du panier. Elle reçoit les mêmes paramètres que addTicketToCart.

Elle récupère les entités Ticket et ExhibitionShare en utilisant leurs repositories.

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

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'.

deleteCartConfirm(): Response (Route :/order/deleteCartConfirm, nom : deleteCartConfirm) :

Cette action affiche une page de confirmation avant de vider complètement le panier. Elle rend le template order/deleteCartConfirm.html.twig.

deleteCart(): Response (Route : /order/cart/delete, nom : deleteCart) :

Cette action vide complètement le panier en appelant la méthode clearCart() du CartService.

Elle redirige ensuite l'utilisateur vers la page du panier.

removeProductToCart(string $id): Response (Route :/order/cart/remove/{id}, nom : removeProduct) :

Cette action retire un produit spécifique du panier en utilisant sa clé unique

($id, qui correspond à $exhibitionId.'\_'.$ticketId).

Elle appelle la méthode removeProduct() du CartService pour supprimer l'élément du

panier.

Elle redirige l'utilisateur vers la page du panier.

orderSuccess(): Response (Route : /orderSuccess, nom : orderSuccess) :

Cette action affiche une page indiquant que la commande a été validée avec succès. Elle rend le template order/orderSuccess.html.twig.

CartService :

\_\_construct(RequestStack $requestStack, TicketRepository $ticketRepo, ExhibitionShareRepository $exhibitionShareRepo) : Le constructeur injecte le RequestStack pour accéder à la session, le TicketRepository pour récupérer les informations des tickets, et l'ExhibitionShareRepository pour récupérer les informations des expositions.

getSession() : Une méthode privée pour obtenir l'objet Session courant.

getCart(): array : Récupère le contenu du panier depuis la session (clé 'cart'). Retourne un tableau vide si le panier n'existe pas.

setCart(array $cart): void : Enregistre le tableau $cart dans la session sous la clé 'cart'.

addCart(Ticket $ticket, int $exhibitionId, int $qty = 1) :

Ajoute un ticket au panier. Récupère le panier actuel, les détails du ticket via le TicketRepository, et l'exposition via l'ExhibitionShareRepository.

Utilise une clé unique ($exhibitionId.'\_'.$ticketId) pour identifier l'article dans le panier.

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

Calcule le total de la ligne pour cet article (totalLine).

Met à jour le panier dans la session et recalcule le total du panier.

removeCart(Ticket $ticket, int $exhibitionId, int $qty = 1) :

Soustrait une quantité spécifiée d'un ticket du panier.

Récupère le panier et la clé de l'article.

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

Met à jour le panier dans la session.

clearCart(): void : Supprime complètement le panier de la session en utilisant $this->getSession()->remove('cart').

removeProduct(string $cartKey): void : Supprime un article spécifique du panier en utilisant sa clé $cartKey. Met à jour le panier dans la session.

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

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

cartCount(): int : Retourne le nombre total d'articles dans le panier en additionnant les quantités de chaque article.

groupCartByExhibition(array $cart): array : Regroupe les articles du panier par exposition. Le tableau résultant a pour clés les exhibitionId, et chaque entrée contient les informations de l'exposition et un tableau des tickets associés à cette exposition dans le panier (avec leur quantité et prix).

--- payment controller

Une fois que l'utilisateur a constitué son panier, il est redirigé vers la page de paiement. 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. Dans le cas contraire, un formulaire basé sur le UserIdentityCartFormType s'affiche, lui permettant de saisir son identité. Ces informations sont obligatoires pour des raisons de conformité légale. 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::stripeCheckout). Ce contrôleur réceptionne la requête, crée une instance du UserIdentityCartFormType, effectue la validation des données soumises et les enregistre en session. Simultanément, le panier de l'utilisateur est récupéré via le CartService.

Une vérification du stock disponible pour chaque article du panier est effectuée en comparant la quantité demandée avec le stock maximal moins les tickets déjà réservés pour l'exposition concernée. Si un stock insuffisant est détecté pour un ou plusieurs articles, un message flash de type danger est affiché pour chaque erreur, informant l'utilisateur du problème, et il est redirigé vers la page du panier pour modifier sa commande.

Si le stock est suffisant pour tous les articles, un tableau contenant les informations nécessaires pour l'API Stripe ($productStripe) est initialisé. Pour chaque article du panier, les informations du ticket (nom) et le prix unitaire sont récupérés, et un tableau structuré pour Stripe est créé, incluant la devise (EUR) et la quantité. Une session de paiement Stripe est alors créée via l'API Stripe, incluant l'adresse e-mail de l'utilisateur ($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). Enfin, l'utilisateur est redirigé

vers l'URL de paiement fournie par Stripe. 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.

En cas de succès du paiement (retour via success\_url et la route paymentSuccess), les informations du formulaire stockées en session sont récupérées. 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. Ensuite, le contenu du panier est récupéré via le CartService.

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.

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

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).

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. 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. On boucle à nouveau sur les éléments du panier pour récupérer chaque article et alimenter un tableau $invoiceDetails contenant les informations nécessaires à la facture (titre de l'exposition et du ticket, prix unitaire, quantité). Ce tableau est ensuite attribué à la propriété invoiceDetails de l'entité Invoice.

Ensuite, l'entité Order et l'entité Invoice sont persistées et enregistrées en base de données via l'EntityManager. 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 potentiellement un service de génération de PDF pour les e-tickets (bien que cela ne soit pas explicitement montré dans ce contrôleur).

L'envoi d'une alerte de stock à l'administrateur est géré par le service StockAlertEmailService via sa méthode sendStockAlertEmail(). Avant cet envoi, l'entité ExhibitionShare est rafraîchie pour obtenir les dernières informations de stock. 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. Le service StockAlertEmailService utilise ensuite le EmailService pour envoyer un e-mail à l'adresse

'alerte\_stock@regardsguerre.fr' si l'un de ces tableaux n'est pas vide.

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).