Une fois la **commande** programmée, l’**utilisateur** se rend vers la **page du panier** afin de procéder au **paiement**.

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

Dans le cas contraire, un **formulaire** basé sur le **FormType** UserIdentityCartFormType sera affiché, lui permettant de saisir son identité. Ces informations sont obligatoires pour des raisons de conformité légale. L'utilisateur a également la possibilité de persister ou non ces données via la case à cocher saveIdentity, respectant ainsi le **RGPD** et le principe de **minimisation des données**.

Lors de la **soumission** de ce formulaire (méthode **HTTP POST**), les données sont transmises via une **requête HTTP**. Le **contrôleur** (PaymentController::stripeCheckoutAction) réceptionne cette requête, crée une instance du UserIdentityCartFormType, effectue la **validation** des données soumises et les enregistre en **session**. Simultanément, un **tableau** contenant les informations des **OrderItem** (éléments du panier) est initialisé.

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), les **line\_items** représentant les produits du panier et les **URLs de succès (success\_url) et d'annulation (cancel\_url)**.

Enfin, l'utilisateur est **redirigé** vers l'**URL de paiement Stripe**. En cas d'échec du paiement (retour via cancel\_url et route paymentError), l'utilisateur est redirigé vers la page du panier avec un **message flash** de type danger.

En cas de succès du paiement (retour via success\_url et 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 **hydratées** avec les données de session, puis l'**EntityManager** Doctrine est utilisé pour persister et flusher ces modifications en **base de données**. Un **message flash** de type success informe l'utilisateur de cet enregistrement.

Ensuite, le **service CartService** est utilisé pour récupérer le contenu du panier ($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.

L'**EntityManager** Doctrine prépare et exécute l'**enregistrement** de cette entité Order en **base de données**.

On itère ensuite sur les éléments du panier (foreach ($cart as $item)) 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 (user friendly) 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, la logique de **gestion des stocks** est mise en place. Trois **tableaux** sont initialisés : $stockErrors, $soonOutStockExhibits et $outOfStockExhibitions.

On commence par itérer sur les éléments du panier pour vérifier la disponibilité du stock avant de finaliser la commande. Pour chaque article, on récupère l'**entité ExhibitionShare** et on compare la quantité demandée (qty) avec le stock disponible (stockMax - ticketsReserved). Si le stock est insuffisant, un tableau $stockErrors est alimenté avec les informations de l'exposition et le stock restant. Si des erreurs de stock sont détectées, un **message flash** de type danger est ajouté pour chaque erreur, et l'utilisateur est redirigé vers la **page du panier**.

Ensuite, l'**entité Order** et l'**entité Invoice** sont persistées et flushées via l'**EntityManager**.

L'envoi de la confirmation de commande à l'utilisateur est géré par le **service OrderConfirmationEmailService**, et plus précisément sa méthode sendTicketEmail(). Ce service utilise le **service EmailService** pour l'envoi d'e-mails et le service PdfService (via generateEticketPdf et generatePdf) pour générer les **e-tickets** au format **PDF** qui sont joints à l'e-mail.

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 ($this->entityManager->refresh($exhibition)) 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).