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).