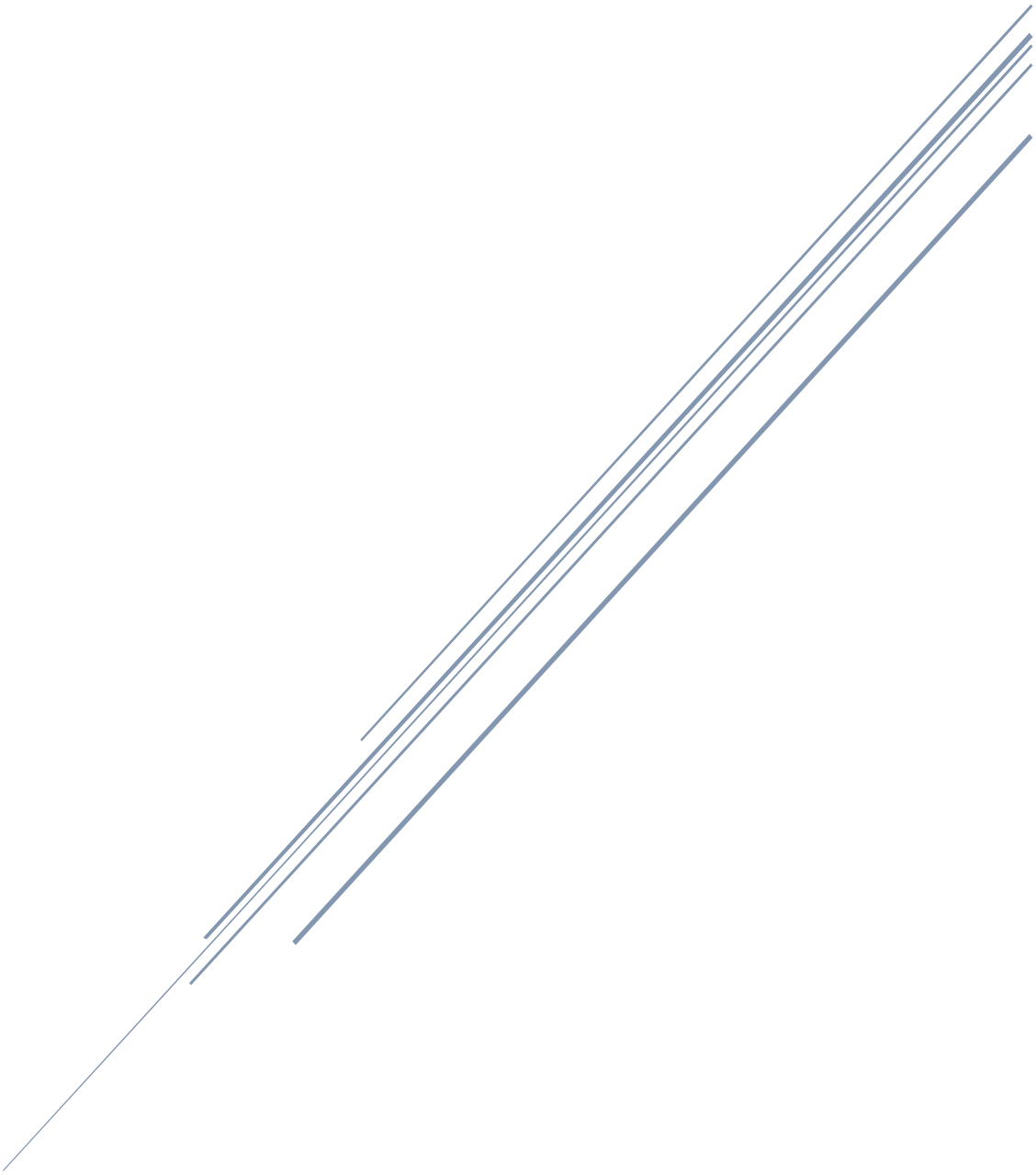
STUBBORN

Documentation



Barbey Mathiey

Sommaire

I/Introduction

Cette documentation permettra à tout utilisateur de pouvoir comprendre les fonctions du code derrière le site Stubborn ainsi que leurs fonctionnements sur chaque page du site. Ce document se divisera par page présente. Les pages n'ayant pas de fonction particulière tel que la page d'accueil ne seront pas prise en compte dans cette documentation. Une section service est présente à la fin de ce document pour expliquer les méthodes importantes présentes dedans.

II/Documentation

Page d'inscription

La page d'inscription est composé de 2 fonctions. La première sert à vérifier et enregistrer les informations de l'utilisateur, et la seconde sert à valider l'email de l'utilisateur avec l'envoi d'un mail. Par ailleurs, il est important de noter que la table user en base de donnée est basée sur un modèle User fourni par Symphony permettant de gérer les rôles sans créer plusieurs tables.

a. Fonction register()

Cette fonction prend en entrée les informations du formulaire d'inscription (nom, mail, adresse de livraison et mot de passe).

```
public function register(Request $request, UserPasswordHasherInterface
$userPasswordHasher, EntityManagerInterface $entityManager): Response
    {
        $user = new User();
        $form = $this->createForm(RegistrationFormType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {

            /** @var string $password */
            $password = $form->get('password')->getData();

            // encode the plain password
            $user->setPassword($userPasswordHasher->hashPassword($user,
$password));
            $user->setRoles(['ROLE_USER']);

            $entityManager->persist($user);
            $entityManager->flush();
```

```
            // generate a signed url and email it to the user
            $this->emailVerifier->sendEmailConfirmation('app_verify_email',
$user,
                (new TemplatedEmail())
                    ->from(new Address('mathieu.barbey18@gmail.com', 'Creator
bot'))
                    ->to((string) $user->getEmail())
                    ->subject('Please Confirm your Email')
                    -
>htmlTemplate('registration/confirmation_email.html.twig')
            );

            // do anything else you need here, like send an email

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

        return $this->render('registration/register.html.twig', [
            'registrationForm' => $form,
        ]);
    }
```

Voici son fonctionnement :

On va d'abord créer un nouvel utilisateur, un nouveau formulaire Symfony et récupérer les informations du formulaire de la page d'inscription. On rentre ensuite dans une condition vérifiant si le formulaire est bien rempli et si celui-ci respecte les conditions (Regex, longueurs de caractères etc) définies pour ce formulaire (voir dossier Form). On récupère ensuite le mot de passe pour l'encoder et on enregistre le tout dans la base de donnée. Un mail de vérification est ensuite envoyé.

b. verifyUserEmail()

Cette fonction permet de valider l'utilisateur lorsque celui-ci clique sur le lien pour vérifier son compte.

```
    public function verifyUserEmail(Request $request, TranslatorInterface
$translator): Response
    {
        $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

        // validate email confirmation link, sets User::isVerified=true and
persists
        try {
            /** @var User $user */
            $user = $this->getUser();
            $this->emailVerifier->handleEmailConfirmation($request, $user);
```

```
        } catch (VerifyEmailExceptionInterface $exception) {
            $this->addFlash('verify_email_error', $translator-
>trans($exception->getReason(), [], 'VerifyEmailBundle'));

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

        // @TODO Change the redirect on success and handle or remove the flash
message in your templates
        $this->addFlash('success', 'Your email address has been verified.');

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

Cette fonction utilise le bundle Security de Symfony pour vérifier le compte et changer en base de donnée si le compte est bien validé ou non à l'aide d'un booléen (0 pour non, 1 pour oui).

Page de connexion

La page de connexion contient aussi deux fonctions. Seule la première sera décrite, la seconde étant la déconnexion gérée directement par le bundle Security de Symphony.

a. login()

Cette fonction sert tout simplement à connecter un utilisateur en se basant sur son mail et son mot de passe et en utilisant le bundle Security de Symphony.

```
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();

        // last username entered by the user
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('login/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }
```

Pages produits et page produit

Les pages produits donnant la liste de tous les sweatshirts vendus par Stubborn ainsi que la page donnant les détails d'un seul produit sont réunies dans le même controller,

ProductsController. Ce sont principalement des fonctions pour récupérer les produits en base de donnée à l'aide du ProductRepository.

a. productsAll()

```
    public function productsAll(ProductRepository
$productRepository): Response
    {

        $products = $productRepository->findAllProducts();

        return $this->render('products/productsList.html.twig', [
            'controller_name' => 'ProductsController',
            'products' => $products,
        ]);
    }
```

Cette fonction récupère tous les produits à l'aide de la méthode findAllProducts() du ProductRepository.

b. filterProducts()

Cette fonction va permettre de trier par tranche de prix les produits affichés sur la page.

```
    public function filterProducts(ProductRepository $productRepository,
Request $request)
    {
        $priceRange = $request->query->get('price_range');
        $products = $productRepository->findByPriceInterval($priceRange);

        return $this->render('products/productsList.html.twig', [
            'products' => $products,
        ]);
    }
```

Pour cela, la méthode findByPriceInterval() qui se base sur une requête SQL faite avec le queryBuilder de Symfony est utilisée.

c. productDetails()

Cette fonction permet d'afficher les détails d'un produit à partir de son id.

```
    public function productDetails(?Product $product, EntityManagerInterface
$em): Response
    {
        $sizes = $em->getRepository(Size::class)->findAll();
        return $this->render('products/product.html.twig',[
```

```
            'product' => $product,
            'sizes'=> $sizes
        ]);
    }
```

Page Panier

Cette partie prendra en compte les fonctions liées à la page panier et donc au controller CartController, mais aussi les fonctions du PaymentController celles-ci étant directement liées à cette page.

a. Fonctions du panier

Les fonctions du panier utilisent les méthodes disponibles dans le service CartService. Celles-ci seront décrite à la fin de cette documentation.

1. add()

Cette fonction récupère les informations du produit choisi pour l'ajouter dans le panier.

```
    public function add(int $id, Request $request, ProductRepository
$productRepository, CartService $cartService): Response
    {
        $product = $productRepository->find($id);
        if (!$product) {
            throw $this->createNotFoundException("Produit introuvable");
        }

        $size = $request->query->get('size', 'M');
        $cartService->addProduct($id, $size);

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

On récupère le produit à l'aide de son id dans un premier temps. Ensuite, une condition vérifie si la variable $product n'est pas null ce qui signifie que le produit n'a pas été trouvé. On utilise ensuite une méthode addProduct() avec comme paramètres l'id et la taille.

2. show()

cette fonction permet de récupérer tous les éléments dans le panier pour les afficher ensuite.

```
    public function show(CartService $cartService, ProductRepository
$productRepository): Response
    {
        $stripePublicKey = $this->parameterBag->get('env(STRIPE_PUBLIC_KEY)');
        $cart = $cartService->getCart();
        $cartWithDetails = [];
```

```
        foreach ($cart as $item) {
            $product = $productRepository->find($item['productId']);
            if ($product) {
                $cartWithDetails[] = [
                    'product' => $product,
                    'size' => $item['size'],
                    'quantity' => $item['quantity']
                ];
            }
        }

        $totalPrice = $cartService->getTotalPrice();

        return $this->render('cart/cart.html.twig', [
            'cart' => $cartWithDetails,
            'totalPrice' => $totalPrice,
            'stripePublicKey' => $stripePublicKey
        ]);
    }
```

On utilise pour ça une autre méthode du service CartService pour récupérer le contenu du panier. On a ensuite une boucle pour enregistrer dans un array les informations de chaque produit ajouté au panier. On retrouve aussi une méthode getTotalPrice() venant aussi du service CartService permettant de calculer le prix total du panier.

3. *remove()*

Cette fonction se base sur la méthode removeProduct() du CartService et prenant en paramètre l'id et la taille du produit dans le panier pour supprimer un élément du panier.

```
    public function remove(int $id, string $size, CartService $cartService):
Response
    {
       $cartService->removeProduct($id, $size);
       return $this->redirectToRoute('cart_show');
    }
```

b. Fonctions du paiement

Seule la fonction du paiement sera décrite ici, la seconde fonction n'étant qu'une réorientation en cas d'erreur ou d'annulation durant le paiement.

La fonction createPaymentSession utilise l'API de Stripe pour créer une session de paiement.

```
    public function createPaymentSession( CartService $cartService,
ProductRepository $productRepository, Product $product): JsonResponse
    {
        $stripeSecretKey = $this->stripeService->getStripeSecretKey();
```

```
        $successUrl = $this->urlGenerator->generate('app_home', [],
UrlGeneratorInterface::ABSOLUTE_URL);
        $cancelUrl = $this->urlGenerator->generate('checkout_cancel', [],
UrlGeneratorInterface::ABSOLUTE_URL);
        $cart = $cartService->getCart();

        //retrieve products in the cart
        foreach ($cart as $item) {
            $product = $productRepository->find($item['productId']);
            if($product) {
                $cartWithDetails[] = [
                    'product' => $product,
                    'size' => $item['size'],
                    'quantity' => $item['quantity']
                ];
            }
        }

        // Stripe config
        Stripe::setApiKey($stripeSecretKey);

        //Payment session creation try
        try {
            $session = Session::create([
                'payment_method_types' => ['card'],
                'line_items' => array_map(function ($cartItem) {
                    $product = $cartItem['product'];
                    return [
                        'price_data' => [
                            'currency' => 'eur',
                            'product_data' => [
                                'name' => $product->getName(),
                            ],
                            'unit_amount' => $product->getPrice() * 100, //to
convert centime in euro
                        ],
                        'quantity' => $cartItem['quantity'],
                    ];
                }, $cartWithDetails),
                'mode' => 'payment',
                'success_url' => $successUrl,
                'cancel_url' => $cancelUrl,
            ]);
            $cartService->clearCart();

            return new JsonResponse(['id' => $session->id]);
        } catch (\Exception $e) {
            return new JsonResponse(['error' => $e->getMessage()], 500);
        }
```

```
    }
```

Cette fonction, très longue, va d'abord définir dans des variables des informations tel que la secret key de stripe ou les url à suivre en cas de succès ou d'échec.

Elle récupère ensuite les produits dans le panier à l'aide de la méthode getCart() et met les données en forme dans la boucle qui suit.

On crée ensuite la session de paiement (cf documentation Stripe).

La redirection se fait en fonction de si un échec ou un succès à lieu, la fonction cleanCart() est appelée pour vider le panier dans le cas d'un succès, la méthode ne s'étant pas arrêtée comme dans le cas d'un échec.

Page administrateur

Cette page est composée de 3 fonctions pour créer, éditer ou supprimer un produit. Ces fonctions sont regroupées dans le SweatshirtController.

a. list()

La fonction list() englobe 2 fonctions, celle pour afficher la liste des éléments ainsi que celle pour les créer, le formulaire de création étant directement intégré à la page

```
 public function list(EntityManagerInterface $em, Request $request,
SluggerInterface $slugger): Response
    {
        $products = $em->getRepository(Product::class)->findAll();

        $newProduct = new Product();
        $sizes = $em->getRepository(Size::class)->findAll();

        foreach ($sizes as $size) {
            $stock = new Stock();
            $stock->setSize($size);
            $newProduct->addStock($stock);
        }

        $createForm = $this->createForm(SweatshirtType::class, $newProduct, [
            'is_edit' => false,
        ]);
        $createForm->handleRequest($request);

        if ($createForm->isSubmitted() && $createForm->isValid()) {
            $imageFile = $createForm->get('image')->getData();

            if (!$imageFile) {
                $this->addFlash('error', 'L\'image est obligatoire pour créer
un produit.');
```

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

            $originalFilename = pathinfo($imageFile->getClientOriginalName(),
PATHINFO_FILENAME);
            $safeFilename = $slugger->slug($originalFilename);
            $newFilename = $safeFilename . '-' . uniqid() . '.' . $imageFile-
>guessExtension();

            try {
                $imageFile->move($this->getParameter('uploads_directory'),
$newFilename);
            } catch (FileException $e) {
                $this->addFlash('error', 'Erreur lors de l\'upload de
l\'image.');
                return $this->redirectToRoute('admin');
            }

            $newProduct->setImage($newFilename);
            $em->persist($newProduct);
            $em->flush();

            $this->addFlash('success', 'Produit créé avec succès.');
            return $this->redirectToRoute('admin');
        }

        $editForms = [];
        foreach ($products as $product) {
            $editForms[$product->getId()] = $this-
>createForm(SweatshirtType::class, $product, [
                'is_edit' => true,
            ])->createView();
        }

        return $this->render('admin/list.html.twig', [
            'products' => $products,
            'createForm' => $createForm->createView(),
            'editForms' => $editForms
        ]);
    }
```

Comme on le voit sur le début de la fonction, la seule ligne pour afficher les produits est la première, le reste étant pour récupérer les informations du formulaire, les manipuler et les enregistrer en base de donnée.

La création du produit commence par la création d'une nouvelle entité product se basant sur la classe Product (voir ProductEntity) ainsi que des stocks pour chaque taille lié à ce produit.

Le formulaire est ensuite créé et une condition vérifiant si le formulaire est bien envoyé et s'il respecte les règles (Regex, longueur de caractère etc) sont bien respectées.

La partie suivante est principalement liée à l'enregistrement de l'image. On va alors standardiser le nom de l'image pour la retrouver plus facilement lors que l'on en aura besoin.

On peut ensuite enregistrer le nouveau produit dans la base de donnée.

La méthode editForm() permet de créer les formulaires d'édition sur la page d'administration pour chaque produit déjà existant.

b. edit()

Cette fonction est sensiblement la même que celle vu précédemment. Elle est là pour récupérer les nouvelles informations venant du formulaire d'édition.

```
 public function edit(Product $product, Request $request,
EntityManagerInterface $em, SluggerInterface $slugger): Response
    {
        $form = $this->createForm(SweatshirtType::class, $product, [
            'is_edit' => true,
        ]);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $imageFile = $form->get('image')->getData();

            if ($imageFile) {
                $originalFilename = pathinfo($imageFile-
>getClientOriginalName(), PATHINFO_FILENAME);
                $safeFilename = $slugger->slug($originalFilename);
                $newFilename = $safeFilename . '-' . uniqid() . '.' .
$imageFile->guessExtension();

                try {
                    $imageFile->move($this->getParameter('uploads_directory'),
$newFilename);
                } catch (FileException $e) {
                    $this->addFlash('error', 'Erreur lors de l\'upload de
l\'image.');
                    return $this->redirectToRoute('admin');
                }

                $product->setImage($newFilename);
            }

            $em->flush();
            $this->addFlash('success', 'Produit mis à jour avec succès.');
```

```
        } else {
            $this->addFlash('error', 'Erreur lors de la mise à jour du
produit.');
        }

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

On retrouve la même méthodologie avec la création du formulaire, puis la récupération et vérification du formulaire suivi de la standardisation de l'image si une nouvelle est ajoutée, et l'enregistrement en base de donnée.

c. delete()

Fonction pour supprimer un produit en base de donnée à l'aide d'un token csrf.

```
    public function delete(Product $product, EntityManagerInterface $em,
Request $request): Response
    {

        if ($this->isCsrfTokenValid('delete' . $product->getId(), $request-
>request->get('_token'))) {
            $filesystem = new Filesystem();

            if ($product->getImage()) {
                $imagePath = $this->getParameter('uploads_directory') . '/' .
$product->getImage();
                if ($filesystem->exists($imagePath)) {
                    $filesystem->remove($imagePath);
                }
            }

            $em->remove($product);
            $em->flush();

            $this->addFlash('success', 'Le sweatshirt a été supprimé avec
succès.');
        }

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

Cette fonction va vérifier si le token du formulaire est valide. Si oui, elle va récupérer toutes les informations liées au produit pour ensuite les supprimer.

Service du Panier CartService

Ici sera décrit les méthodes disponible dans le CartService. L'avantage d'un service est que la donnée est stockée sur le navigateur du client ce qui est intéressant dans le cas d'une connexion ou, dans notre cas, d'un panier.

Ce service ce divise en cinq méthodes.

a. getTotalPrice()

Cette méthode est aussi dans le CartController mais aussi dans le PaymentController. Elle permet de calculer la somme totale du panier.

```
    public function getTotalPrice(): float
    {
        $cart = $this->session->get('cart', []);
        $total = 0;

        foreach ($cart as $item) {
            $product = $this->productRepository->find($item['productId']);

            if ($product) {
                $total += $product->getPrice() * $item['quantity'];
            }
        }

        return $total;
    }
```

b. addProduct()

Cette méthode permet d'ajouter un produit dans le panier en récupérant le produit choisi ainsi que la taille souhaitée.

```
    public function addProduct(int $productId, string $size): void
    {
        $cart = $this->session->get('cart', []);

        $key = $productId . '_' . $size;

        if (!isset($cart[$key])) {
            $cart[$key] = [
                'productId' => $productId,
                'size' => $size,
                'quantity' => 1
            ];
        } else {
```

```
            $cart[$key]['quantity']++;
        }

        $this->session->set('cart', $cart);
    }
```

c. getCart()

Méthode permettant de récupérer le contenu du panier.

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

d. removeProduct()

Cette méthode permet de retirer un produit spécifique du panier à partir de son id.

```
    public function removeProduct(int $productId, string $size): void
    {
        $cart = $this->session->get('cart', []);

        $key = $productId . '_' . $size;
        if (isset($cart[$key])) {
            unset($cart[$key]);
        }

        $this->session->set('cart', $cart);
    }
```

e. cleanCart()

Enfin, cette méthode permet de vider le panier. On la retrouve dans le PaymentController.

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