**DONE :**

**Registration**

Is-verified (mail de confirmation)

**Reset password** (mail de demande)

Command pour supprimer les requêtes périmées

**Login / logout**

**Page profil**

-ajout avatar

-edit pseudo + email

-change password

**Création page error 404 personnalisée**

**Service pour upload plusieurs images (et les supprimer base de données + dossier images)**

composer require intervention/image

**Navbar : avoir le statut artiste ou éditer sa page artiste**

Edit info / ajouter ou supprimer images

**Flash message :**

après 5secondes ils disparaissent (js)

**Carrousel dynamique (artiste page)**

**Entity image (id user + id area)**

use Doctrine\Common\Collections\Criteria;

public function getPictures(): Collection

**formattage Date : twig extension**

**change password (DTO)**

**get last login (event listener)**

**date format : creation twig Extensions**

**envoi d’email de confirmation (service)**

**API leaflet**

**Command deletedExpiredResetPasswordRequest**

**Dom pdf (générer pdf + inclure après submit + mail)**

composer require dompdf/dompdf

template pdf + pdf controller

**Custom error page :**

**1. Require symfony/twig-bundle**

> This command installs the Symfony Twig bundle, which provides integration between Symfony and the Twig templating engine.

**2.** **Create the Twig Page for the Error (error.404.html.twig)**:

> put it in the **templates/bundles/TwigBundle/Exception/** directory. The contents of **error404.html.twig** would be your custom HTML for the 404 error page.

**3.** **Create an ErrorController with a show Method**:

> Create a controller that will handle errors and render the custom error pages.

4. **Add error\_controller to framework.yaml**:

> This configuration tells Symfony to use your **ErrorController::show** method as the custom error controller.

Command clean request table reset password

**Service UploadPicture**

**Pour éditer image**

**Dans vue : condition on affiche les images si édition**

**Penser à ajouter les images à la vue pour qu’il puisse les récupérer (via le controller)**

**MailerService :**

**1. Create a MailerService :** Créez un service Symfony qui encapsulera la logique d'envoi d'e-mails.

2. **Configurer le service :** Déclarez votre service dans le fichier de configuration des services (services.yaml), afin qu'il soit injecté lorsque vous en aurez besoin.

Assurez-vous que la classe **MailerService** est correctement associée à son namespace dans le fichier **services.yaml**.

3.**Utiliser le service dans votre contrôleur :**

Injectez le service dans votre contrôleur et utilisez les méthodes du service pour envoyer des e-mails spécifiques.

**Date format :**

To format dates consistently across your Symfony application, you can leverage the Twig templating engine and create a Twig extension filter. This allows you to define a custom filter that you can use in your Twig templates to format dates.

**1. Create a Twig Extensions :**

> Twig extensions are PHP classes that define custom functions or filters that can be used in your Twig templates.

Src/twig/AppExtension.php

**2. Register the Twig Extension :**

> Register the custom Twig extension as a service in your Symfony application. Add the following to your services.yaml file:

**3. Use the Custom Filter in Twig Templates:**

> you can use the formatDate filter in your Twig templates to format dates.

{{workshop.endDate|formatDate}}

{{ user.lastLoginDate|formatDate('d/m/Y H:i:s') }}

**API - Leaflet :**

**1. Install Leaflet :**

> Dans la vue > rajouter link stylesheet

> ajouter script

**2. Récupérer les données :**

Dans le contrôleur associé à la vue où vous afficherez la carte, récupérez les informations nécessaires depuis votre base de données.

**3. Convertir les adresses en coordonnées gps**

> Utiliser Leaflet GeoSearch

composer dump-autoload