**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**

**Honey pot /**login throttling

**Fullcalendar**

**Commande (x2)**

**login throttling**

**slug**

**composer require symphony/var-dumper**

**creation d’un validateur personnalisé pour vérifier que l’heure sélectionnée est une heure pleine**

**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

**STRIPE :**

**1. : Install : composer require stripe/stripe-php**

>

> symphony/dotenv (

**Sécuriser private key:**

Controller

Service.ymal

Env

**HONEY POT :**

**1. : champ caché + switch name**

>

**CAPTCHA :**

**1. : reCAPTCHA v3 / install : composer require karser/karser-recaptcha3-bundle**

>

**LIMITATION TENTATIVES D’INSCRIPTION :**

**RATE LIMITER / LOGIN THROTTLING**

1. **: install: composer require symfony/rate-limiter**
2. **Add (security.yaml)**
3. # by default, the feature allows 5 login attempts per minute
4. login\_throttling: null
5. # configure the maximum login attempts
6. login\_throttling:
7. max\_attempts: 3 # per minute ...
8. # interval: '15 minutes' # ... or in a custom period

**3: custom parameters**composer require symfony/lock

>

**COMMAND (isActive subscription)**

> create a symphony console command:

> register the command as a service

> scheduled the command to run daily using cron:

# app/config/schedule.yaml

# Run the command every day at midnight

app:update-subscription-status:

cron: '0 0 \* \* \*'

**TARTE AU CITRON:**

> ajout de tarteaucitron.js

**AUTH with Google,facebook etc.**

**SLUGIFY :**

1. Install : composer require cocur/slugify

2. Ajouter champ slug dans entité + method generateSlug() – ajouter le use slugify

3. Changer route dans controller + path dans views

4. dans form, set slug

**CALENDAR :**

1. Install full calendar (cdn) + calendar.js + id=calendar dans la page
2. Dans le controller pour les events, faire une fonction qui récupère les events, et formatte les résultats pour que ça soit interprétable par full calendar
3. Configurez une route pour cette méthode dans le fichier routes.yaml :
4. Modifier le script JS pour utiliser cette nouvelle route

**GOOGLE AUTH:**

1. Install : composer require league/oauth2-client

composer require symfony/http-client

Google identity platform > Oauth 20. Access Goggle APIs

(google renvoie token d’accès, chaine de caractère)

> Identifiants >**Créer un ID client OAuth >**  [Configurer](https://developers.google.com/identity/protocols/oauth2/?hl=fr)  l’écran de consentement > récupérer les clefs (config.php)

> Définir : **Redirect\_uri**, **Scope (ici email – API google sign in)**, **Access\_type (spécifier le type de token qu’on va recevoir : 2 types : token valable pour une courte durée, token offline permettant des appels à l’api beaucoup plus tard dans le temps dans notre cas on utilise du online).**

> Définir ces paramètres puis rediriger l’utilisateur

<https://developers.google.com/identity/protocols/oauth2/web-server?hl=fr>

Ajouter le lien fournis par google sur la page login par exemple, créer une page connect, où l’on revoie le google id dans le controller

> dans la page connect, via le controller récupérer le code de l’url ($\_get)

Step 3 : google prompts for consent (déjà fait)

Step 4 : handle the Pauth 2.0 server response (déjà fait)

Step 5: exchange authorization code for refresh and access token

Documentation openId Connect > the discovery document

> install : composer require guzzlehttp/guzzle

Dans le controller new client https://docs.guzzlephp.org/en/stable/quickstart.html

url : <https://developers.google.com/identity/openid-connect/openid-connect?hl=fr#createxsrftoken>

rajouter file cacert.pem

> composer : composer require league/oauth2-client

**Validateur personnalisé (heure pleine):**

1. création d’un constraint validator (symfony\Component\Validator\Constraint

> Créer un répertoire Validator dans votre répertoire src s'il n'existe pas déjà. Créez un répertoire **Validator** dans votre répertoire **src** s'il n'existe pas déjà. **FullHourValidator.php**

> dans entity ajouter #[CustomAssert\FullHour] et use App\Validator\Constraints as CustomAssert;

**FRONT END**

**FONT STACKS WILL IMPROVE YOUR SITE PERFORMANCE :**

**1. :**

>

composer dump-autoload