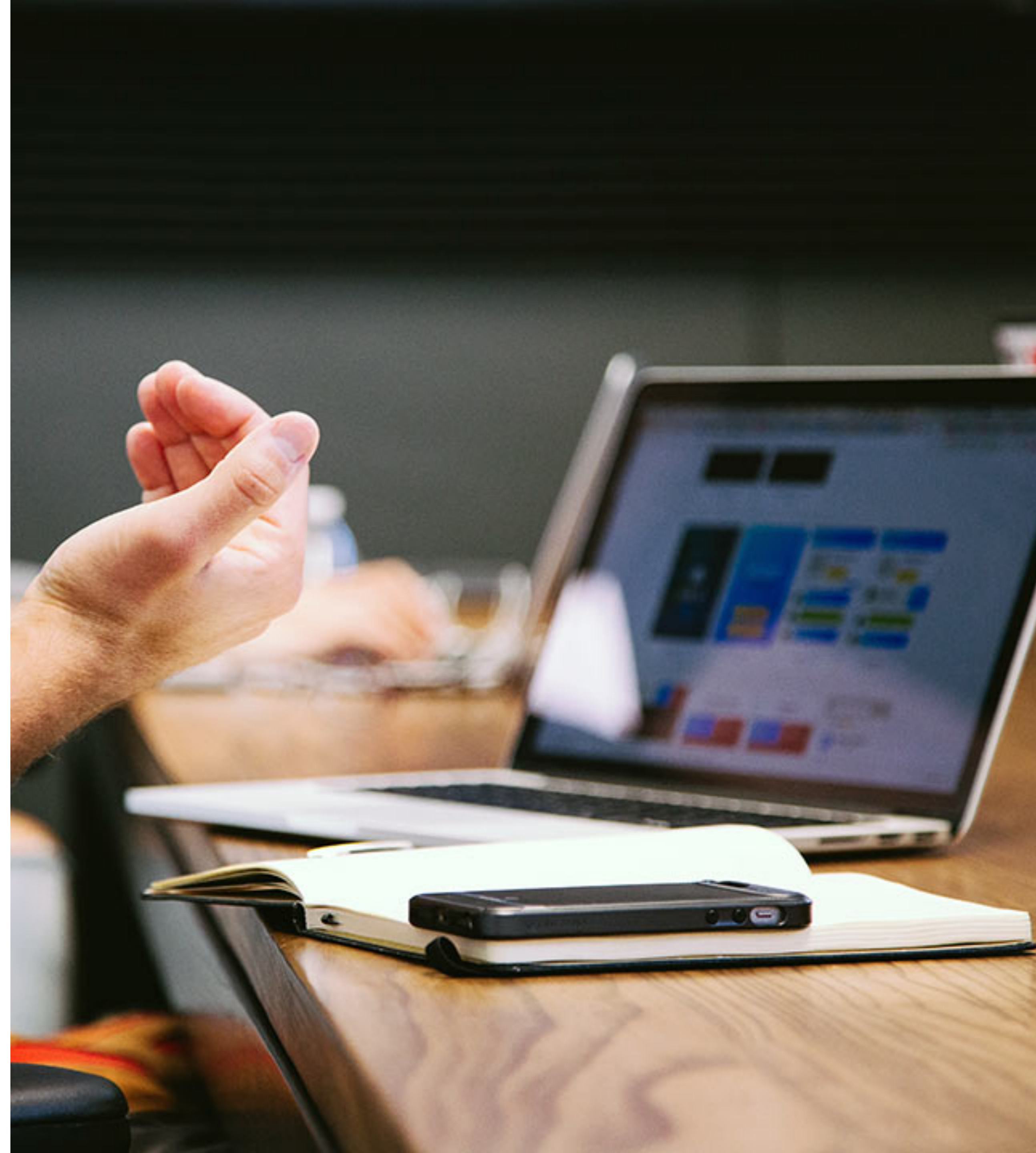


# Yoann Coualan

CTO et Chef d'entreprise

# Mon parcours

- Développeur web
- Tech Lead / Directeur technique
- Jedy Formation
- Yoda Formation
- Jedy Agency





High Performance PHP Framework for Web Development

Yoann Coualan

# Repo du cours

[https://github.com/yoanncoualan/IIM\\_202324\\_Symfony\\_cours](https://github.com/yoanncoualan/IIM_202324_Symfony_cours)



# Qu'est ce que Sf ?

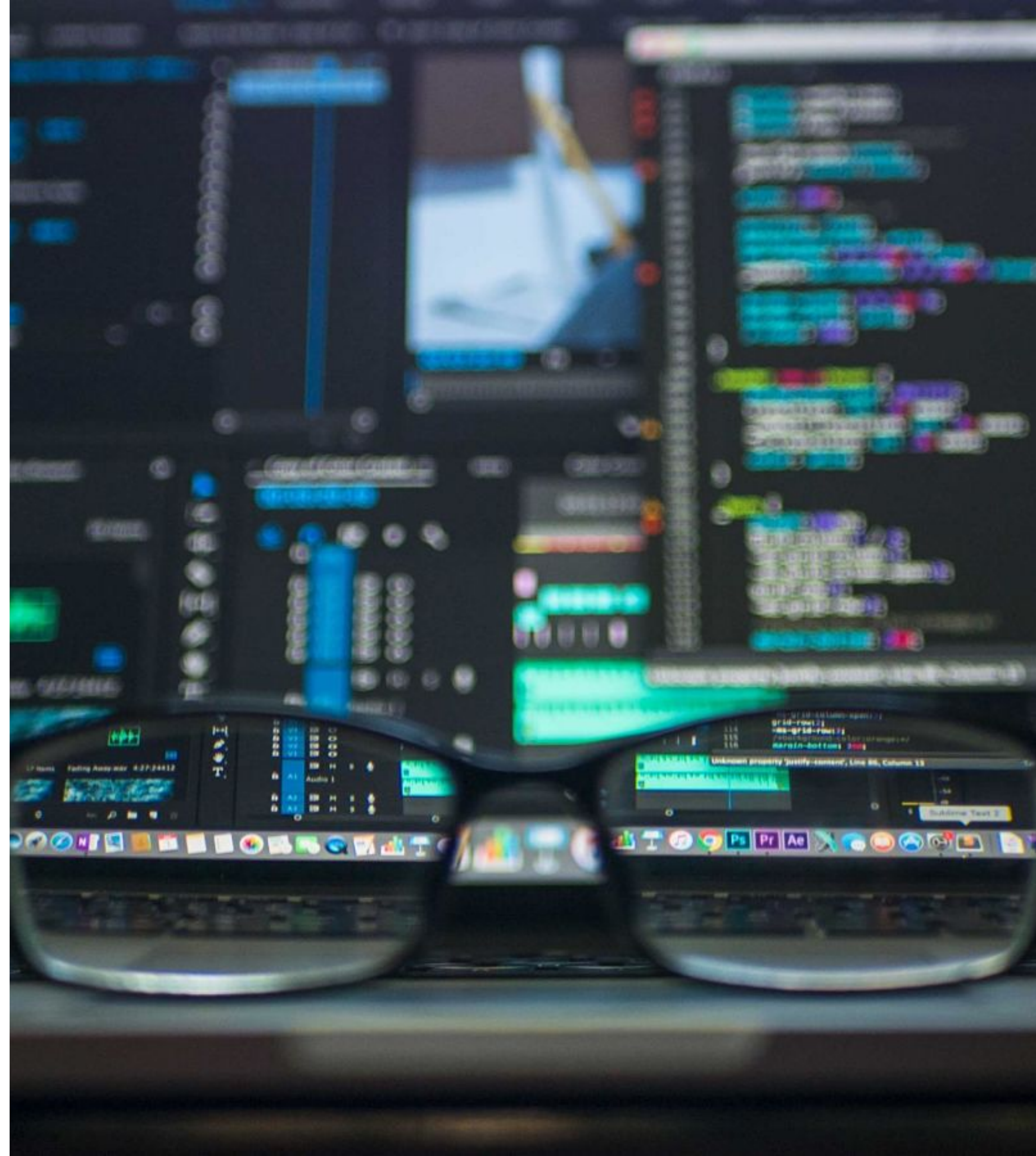
- **Un ensemble de composants PHP**
- Un modèle MVC prêt à l'emploi
- Une grande communauté
- Une maintenance régulière





# Les composants phares

- **Composer** : gestionnaire de dépendances
- **Make** : générateur de code
- **Doctrine** : gestionnaire de BDD
- **Form** : gestionnaire de formulaires
- **Routing** : uniformise les requêtes HTTP
- **Security** : gère les autorisations
- **Translation** : gère les traductions
- **Twig** : moteur de template



# Démarrer un projet

1. Avoir **PHP**  $\geq$  8.2
2. Installer **Composer**  
<https://getcomposer.org/download/>
3. Installer **Symfony CLI**  
<https://symfony.com/download>



# Démarrer un projet

5. Créer un **nouveau projet** Symfony

```
symfony new project_name --webapp
```

*ou*

```
composer create-project symfony/skeleton project_name  
cd project_name  
composer require webapp
```

6. Lancer Symfony

```
cd project_name  
symfony server:start
```



# Organisation des fichiers

**Bin** : executables de Symfony

**Config** : configuration du projet

**Migrations** : migrations

**Public** : CSS, JS, images, etc

**Src** : modèles, contrôleurs, formulaires

**Templates** : vues

**Tests** : tests automatisés

**Translations** : traductions

**Var** : logs, cache

**Vendor** : librairies





# Vocabulaire

**Bundle** : brique, contient tout le code qui gère une fonctionnalité

**Entité** : représente une table dans la BDD

**Route** : URL

**Repository** : Modèle

**Service** : utilitaire, class accessible depuis n'importe quel fichier

**Namespace** : compartiment virtuel





## Erreur

```
<?php
class maClass{
    // ...
}
?>

<?php
class maClass{
    // ...
}
?>
```

VS

## Succès

```
<?php
    namespace namespace1;

    class maClass{
        // ...
    }
?>

<?php
    namespace namespace2;

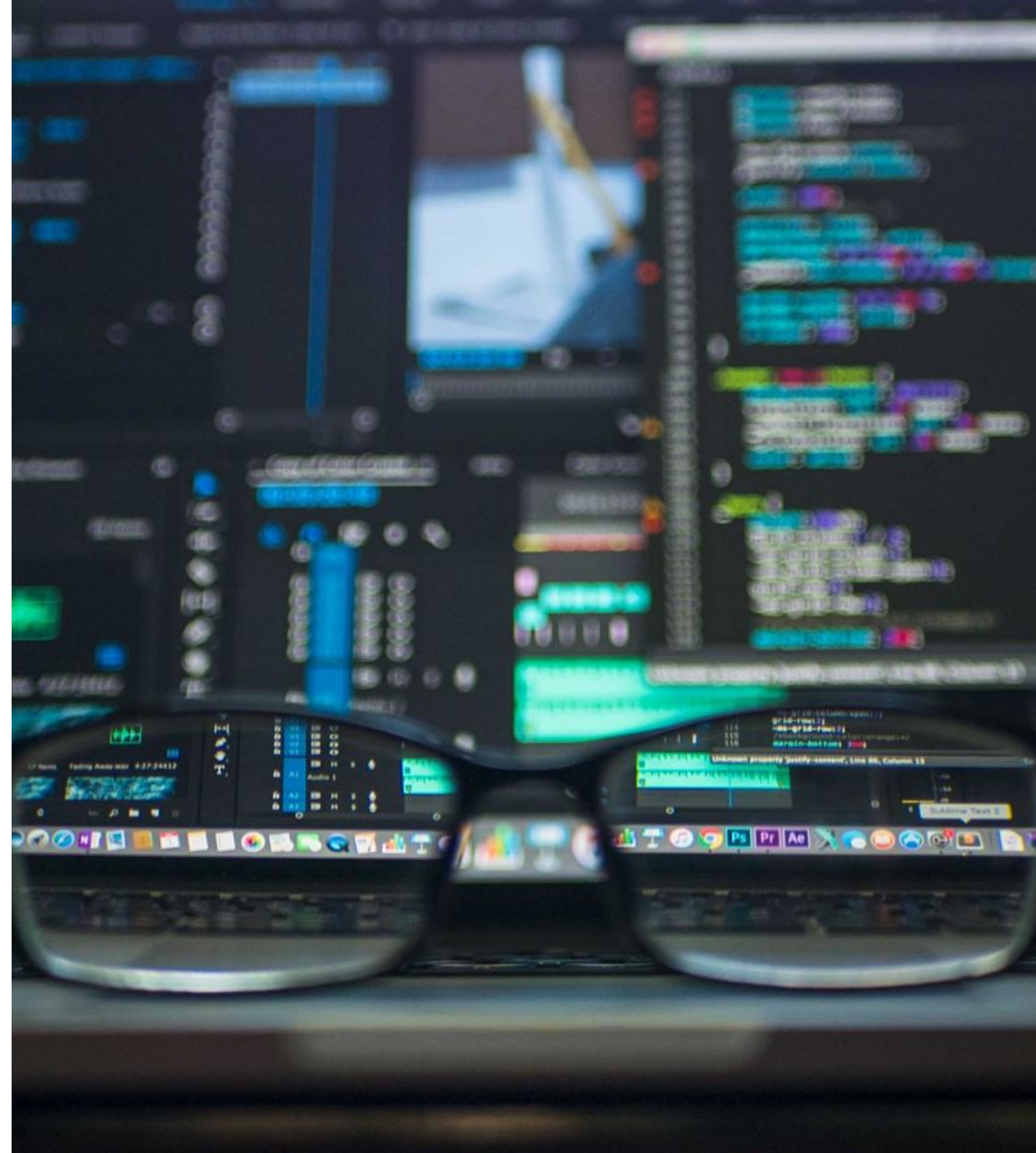
    class maClass{
        // ...
    }
?>
```



# Créer une page

[https://symfony.com/doc/current/page\\_creation.html](https://symfony.com/doc/current/page_creation.html)

1. Choisir / créer un controller
2. Créer une méthode
3. Créer une route
4. Créer une vue





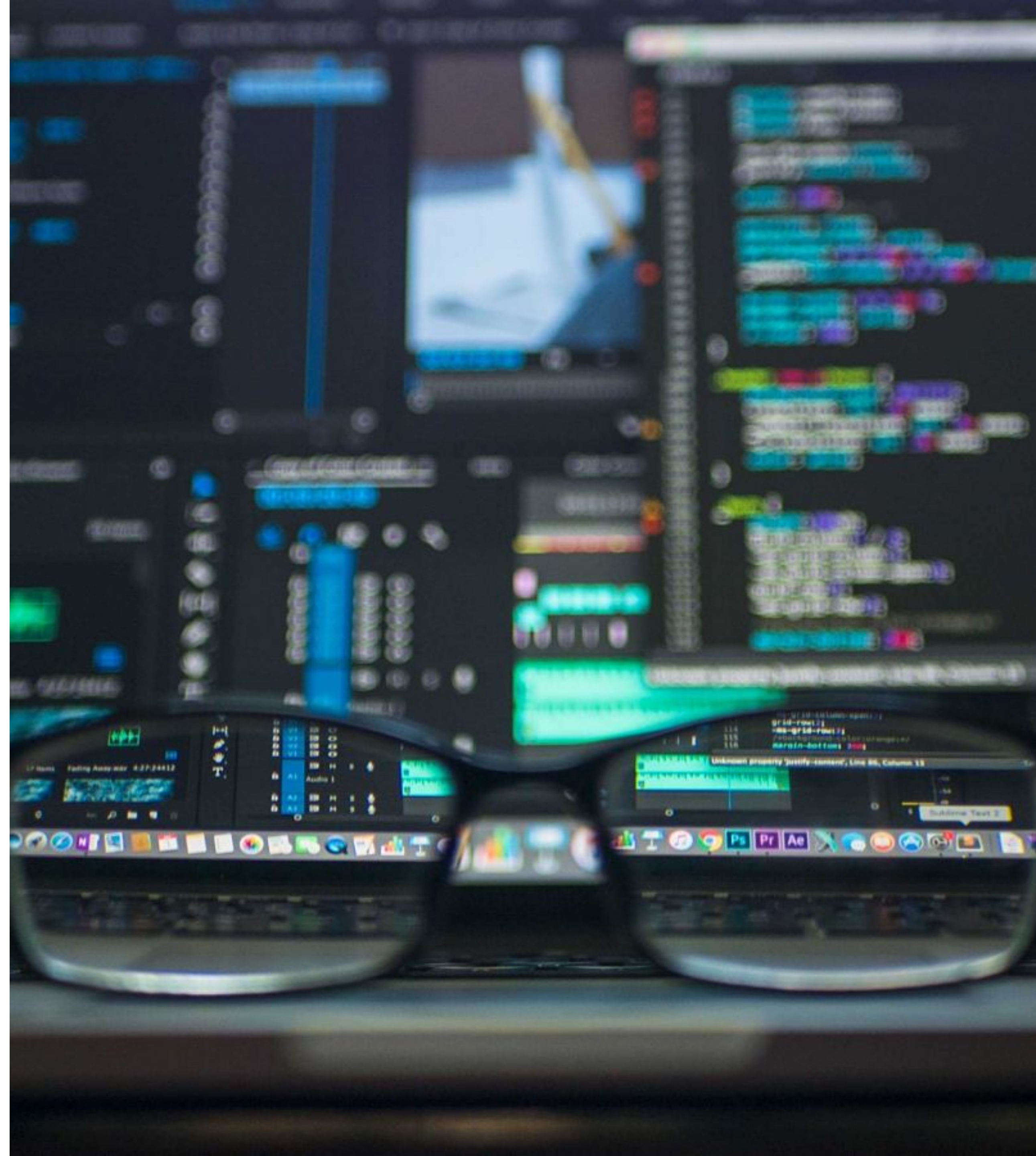
# Créer un controller

```
php bin/console make:controller
```

Génère :

- Le controller
- Une méthode index
- Une route associée
- Une vue associée

=> Créer le controller `CategorieController`





# Twig et les templates

Extension :  
**\*.html.twig**

Afficher la valeur d'une variable :  
**{{ ... }}**

Définir quelque chose :  
**{% ... %}**





# Twig et les templates

Etendre un template :

```
{% extends 'base.html.twig' %}
```

Les blocs :

```
{% block nom_du_block %}
```

...

```
{% endblock %}
```





# Configuration

config/

└─ packages/

| Configuration des paquets installés

└─ routes/

| Configuration des routes

└─ services.yaml

| Paramètres et services

.env



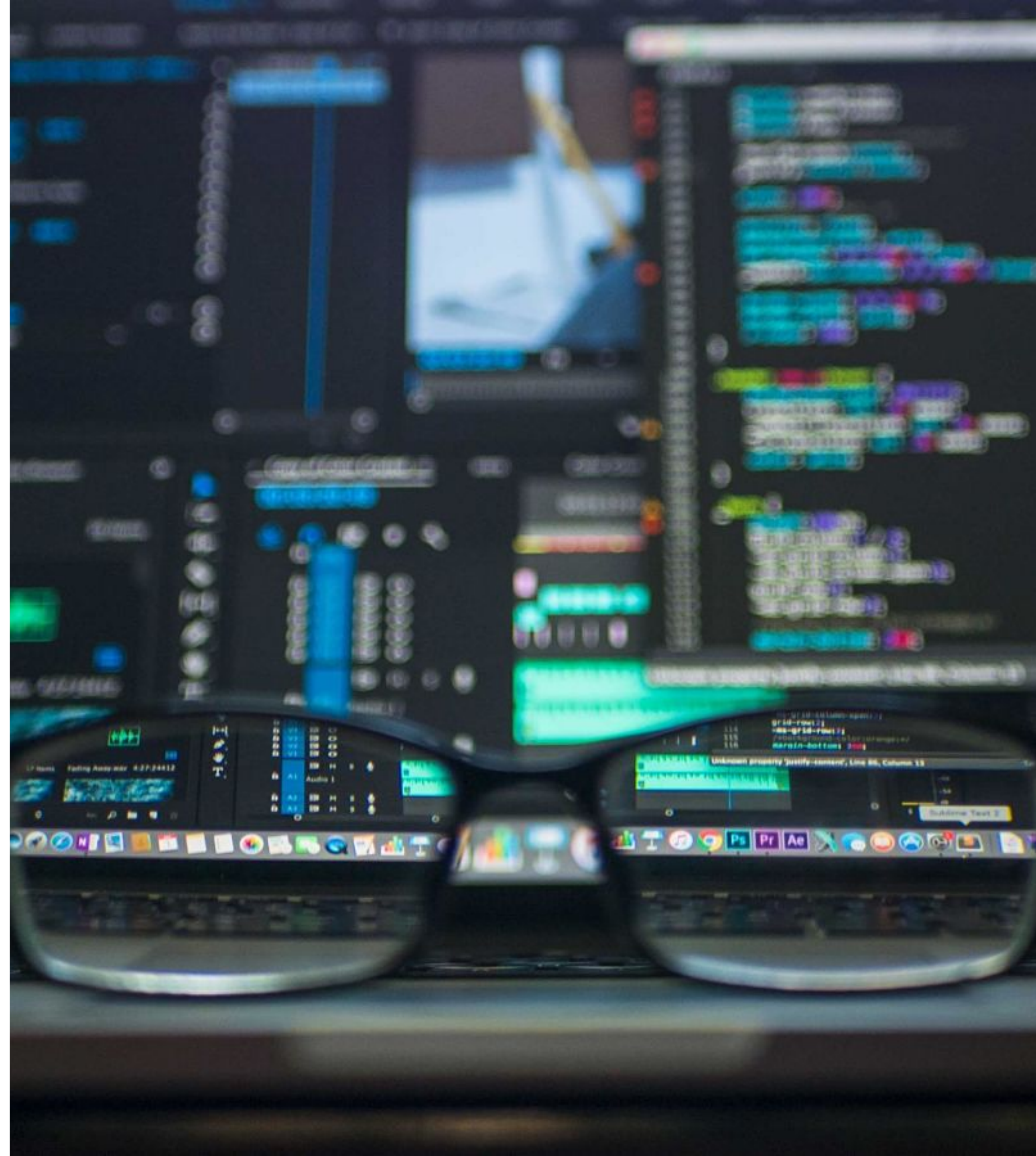


# La base de données

Doctrine

Créer une base de données :

```
Php bin/console  
doctrine:database:create
```





# Les entités - création

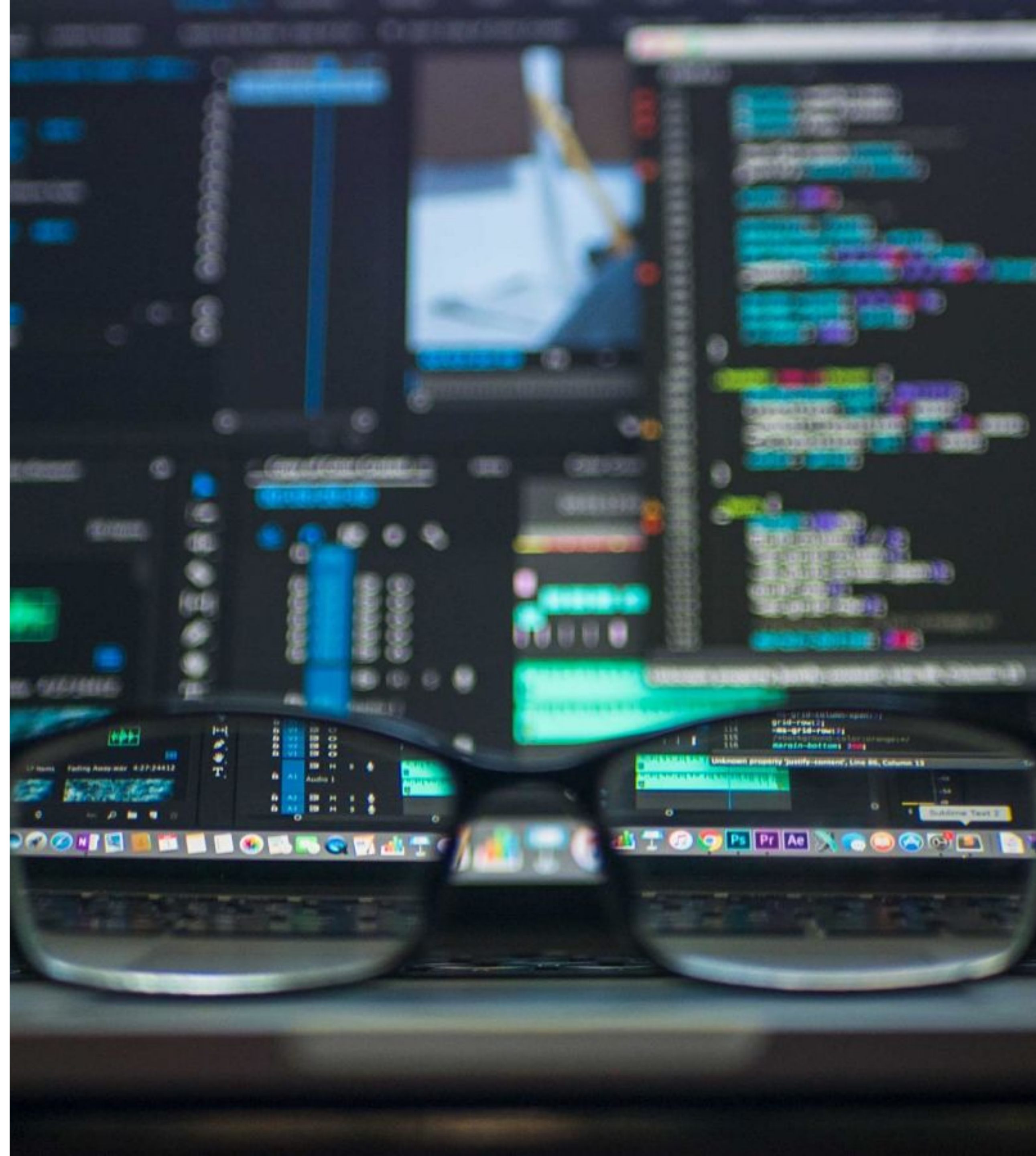
Créer une entité :

```
php bin/console make:entity Nom
```

Ma BDD est-elle à jour ?

```
php bin/console doctrine:schema:validate
```

=> Créer l'entité Catégorie (nom, description)





# Les entités - migration

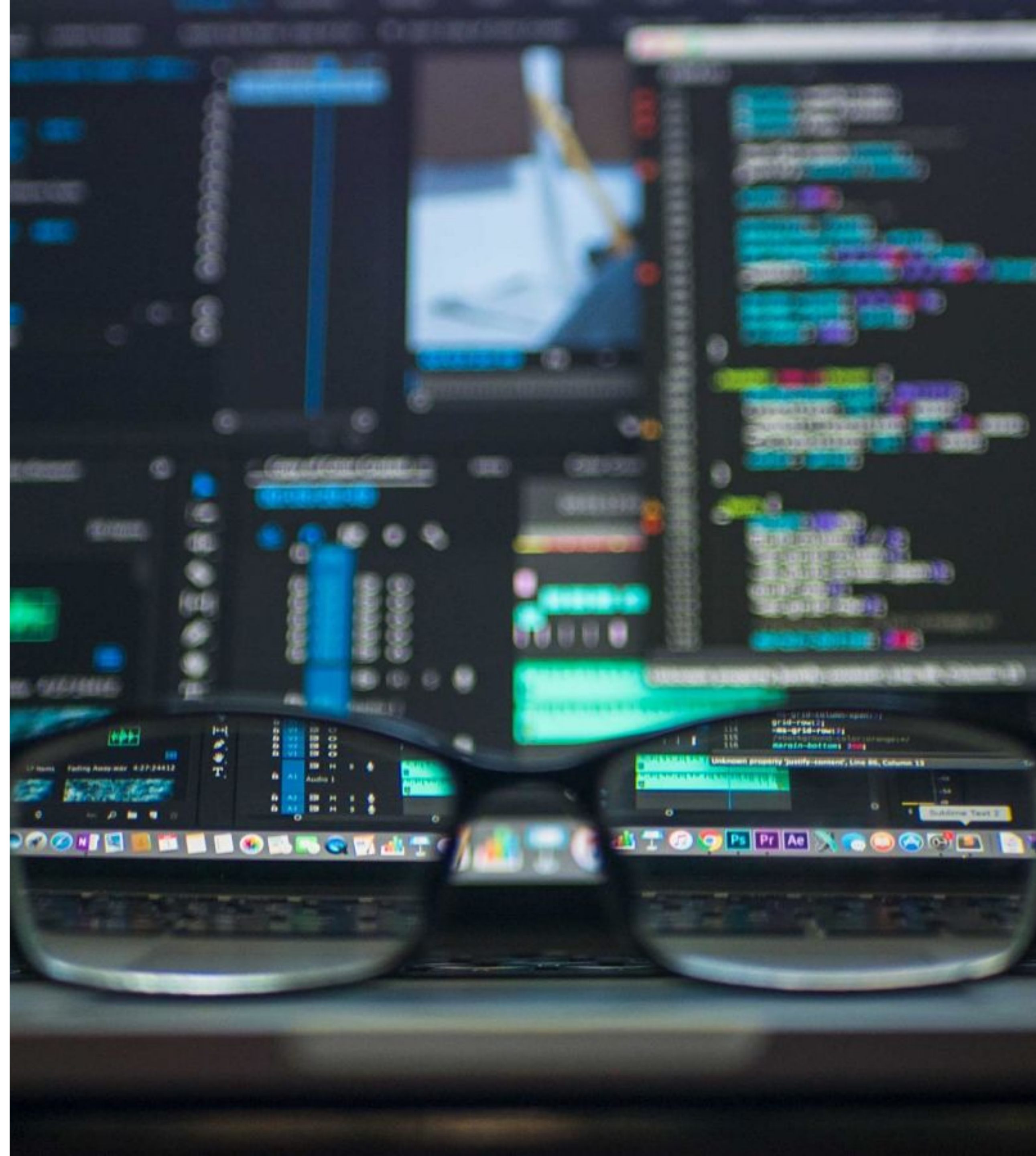
Préparer une migration :

```
php bin/console make:migration
```

Executer la migration

```
php bin/console  
doctrine:migration:migrate
```

⚠ Attention aux pertes de données





# Les entités - récupération

Récupérer Doctrine :

```
use Doctrine\ORM\EntityManagerInterface;  
  
public function index(EntityManagerInterface $em)
```

Récupérer une entité :

```
$em->getRepository(Categorie::class)->findAll();
```

Afficher tout le contenu de la variable dans la vue :

```
{{ dump(var) }}
```



# Twig - conditions et boucles

SI

```
{% if var is not empty %}
```

```
{% else %}
```

```
{% endif %}
```

FOR ... IN

```
{% for element in var %}
```

```
{% endfor %}
```

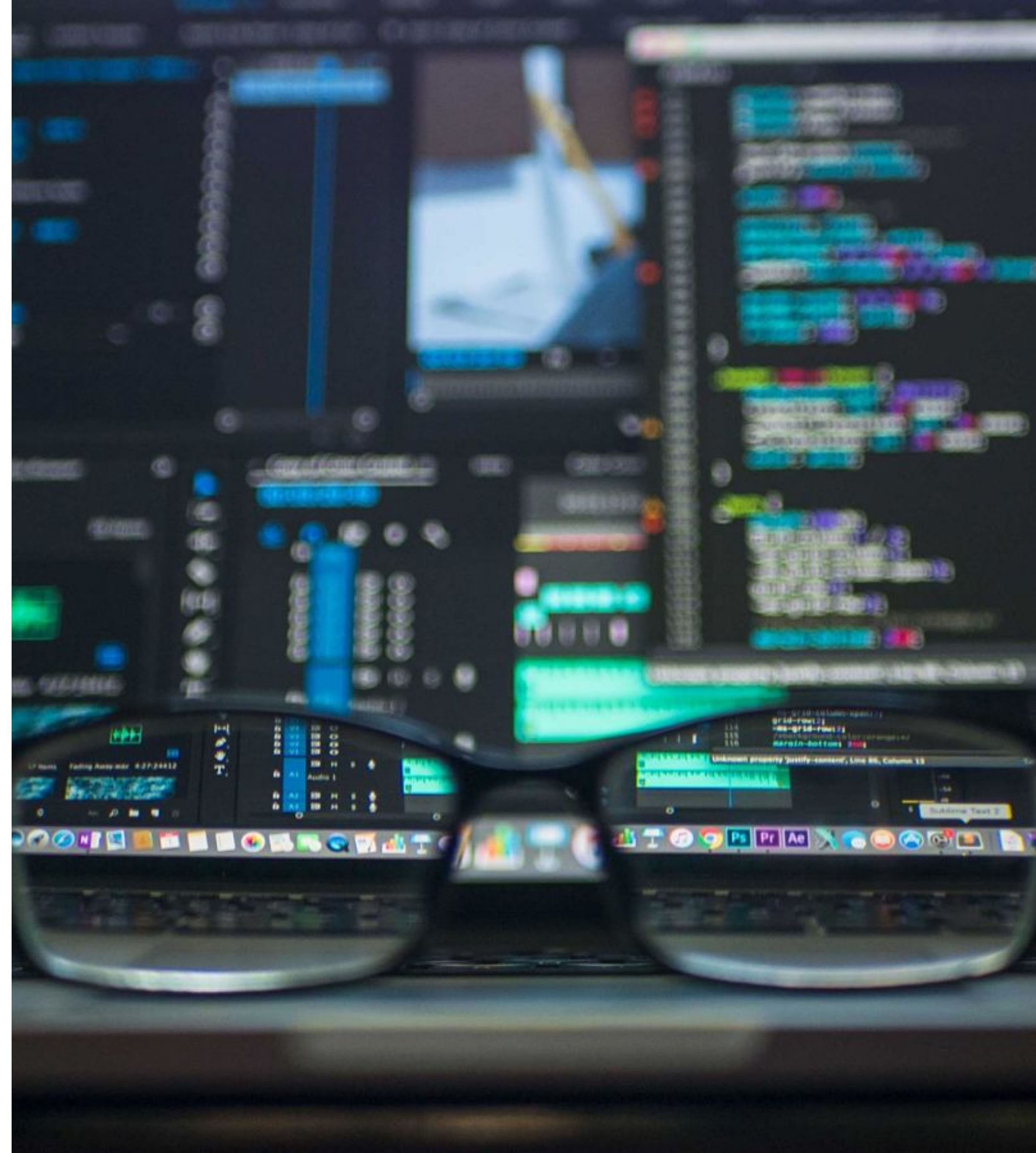


# Formulaires - création

<https://symfony.com/doc/current/forms.html>

```
php bin/console make:form
```

Externalise et automatise la création des formulaires en se basant sur une entité





# Formulaires - utilisation

Utilisation du formulaire :

```
$categorie = new Categorie();  
$form = $this->createForm(CategorieType::class, $categorie);
```

Envoyer le formulaire à la vue :

```
'form' => $form->createView()
```

Envoyer le formulaire et ses erreurs à la vue :

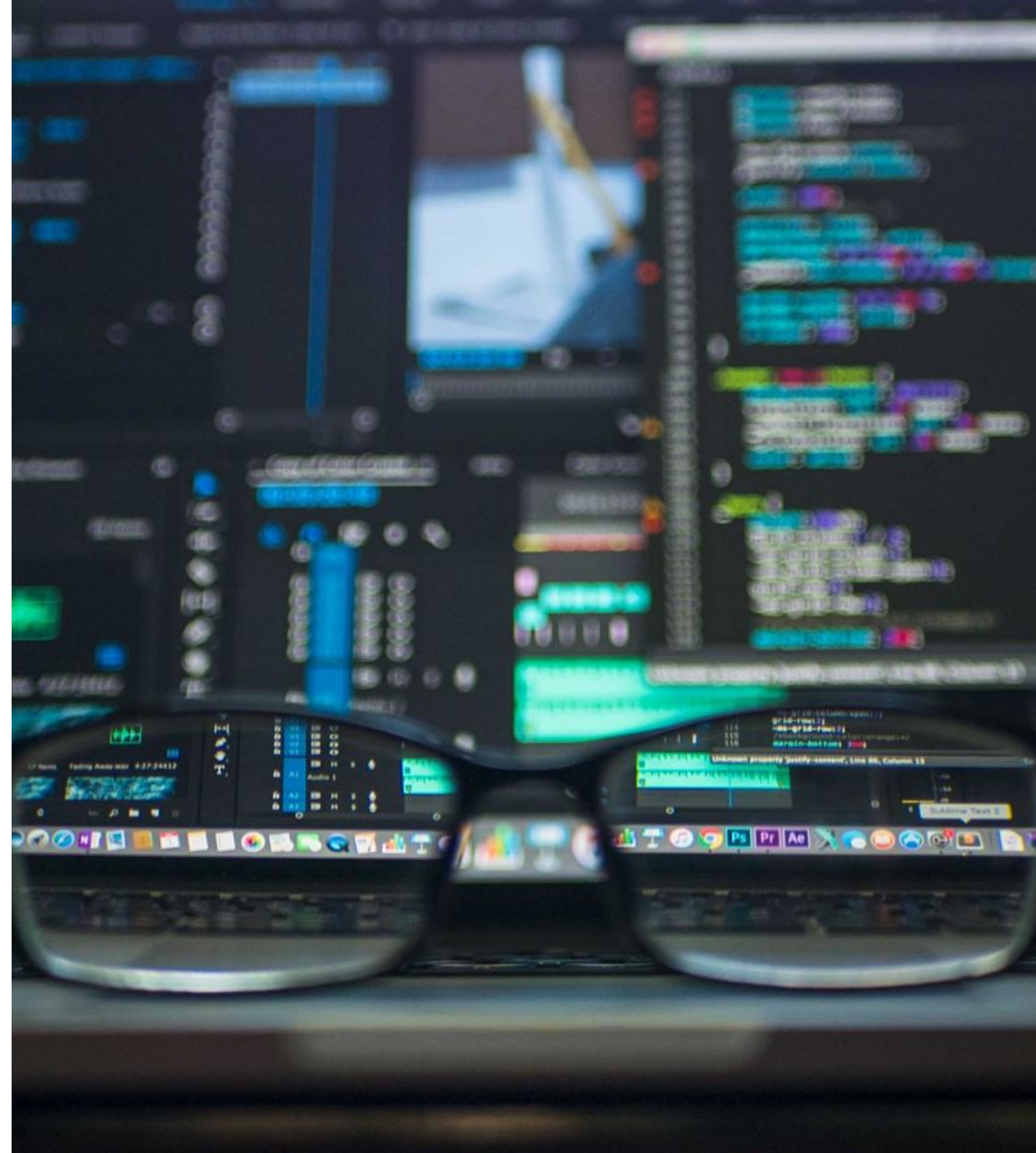
```
'form' => $form
```



# Formulaires - affichage

Afficher le formulaire dans le vue :

```
{{ form_start(form) }}  
    {{ form_errors(form) }}  
  
    {{ form_label(form.field1) }}  
    {{ form_widget(form.field1) }}  
  
    {{ form_row(form.field2) }}  
{{ form_end(form) }}
```





# Formulaires - soumission

Détecter les requêtes HTTP :

```
use Symfony\Component\HttpFoundation\Request;
```

Détecter l'envoi du formulaire :

```
public function index(Request $request)

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        // Le formulaire a été envoyé et est valide
    }
```



# Formulaires - validation et sauvegarde

Vérification des propriétés :

<https://symfony.com/doc/current/validation.html>

Sauvegarde du formulaire en base :

```
$em->persist($categorie);  
$em->flush();
```

Rediriger l'internaute :

```
return $this->redirectToRoute('app_categorie');
```



# Routes et path

Route simple (controller)

```
#[Route('/categories', name: 'categories')]
```

Lien (vue)

```
{{ path('categories') }}
```

Route avec paramètre (controller)

```
#[Route('/category/{id}', name: 'category')]
```

Lien (vue)

```
{{ path('category', {'id': element.id}) }}
```





# Les entités - récupération

Récupération d'une entité :

```
#[Route('/category/{id}', name: 'category')]
public function index(Categorie $categorie = null)
{
    if($categorie == null){
        // Aucune catégorie ne correspond à cet id
    }
}
```



# Un peu de pratique

Sur la page d'une catégorie, mettre en place le formulaire d'édition de la catégorie.



# Design des formulaires

Template de base :

[vendor/symfony/twig-bridge/Resources/views/Form/form\\_div\\_layout.html.twig](#)

Créer son propre fichier de rendu :

[/templates/form/fields.html.twig](#)

Puis, indiquer à Symfony d'utiliser notre fichier de rendu :

```
#config/packages/twig.yaml
```

```
twig:
```

```
    form_themes:
```

```
        - 'form/fields.html.twig'
```



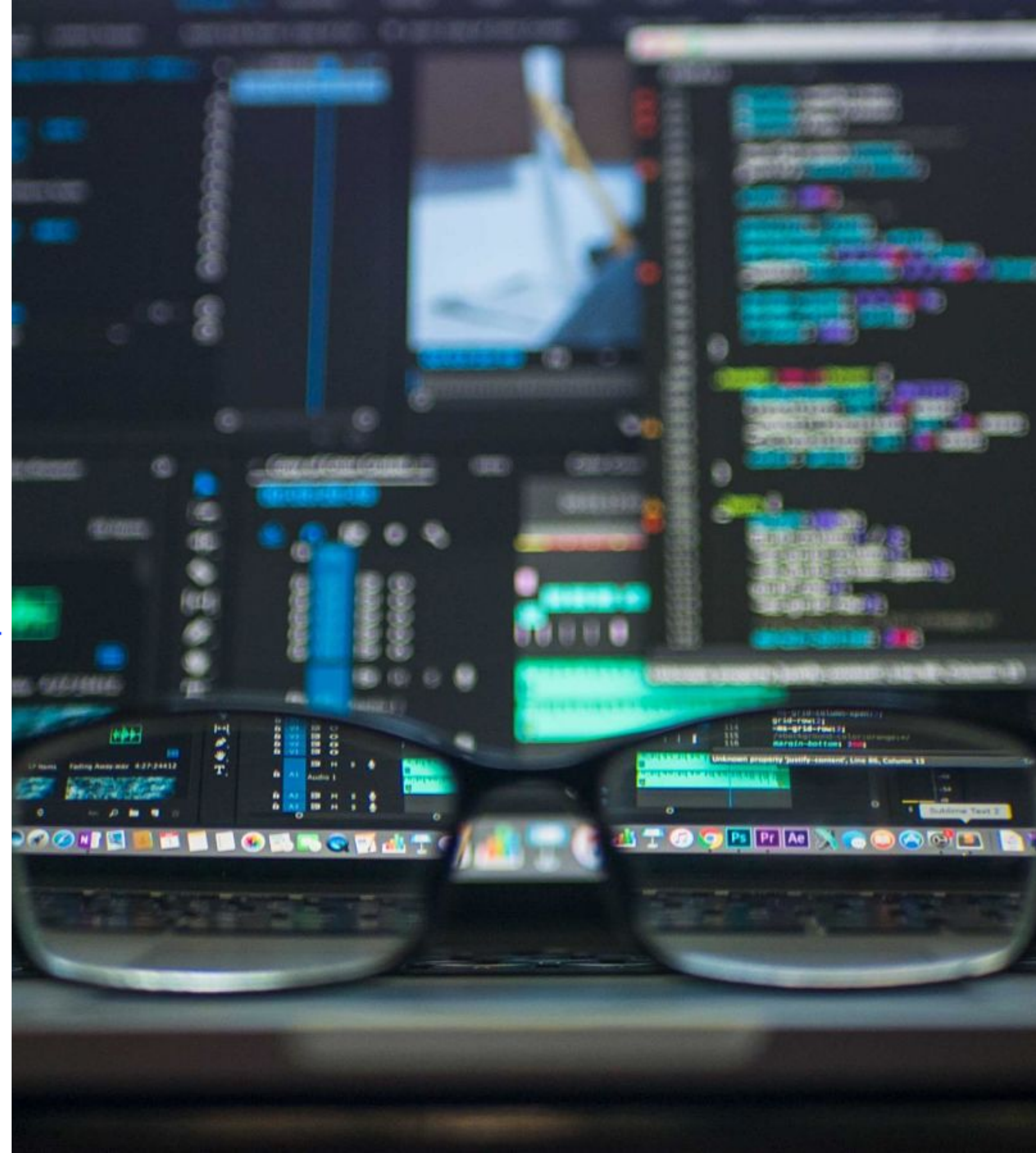
# Messages flash

Ajouter un message flash :

```
$this->addFlash('type', 'message');
```

Afficher les messages flash :

```
{% for type, messages in app.flashes %}  
    {% for msg in messages %}  
        <p>{{ type }} : {{ msg }}</p>  
    {% endfor %}  
{% endfor %}
```





# Les jointures

<https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html>

Un article ne peut avoir qu'une seule catégorie mais une catégorie peut avoir plusieurs articles :

- article.php : *ManyToOne*
- category.php : *OneToMany*

=> Mettre en place l'entité Produit (nom, description, prix, quantité) ayant une relation vers Catégorie.

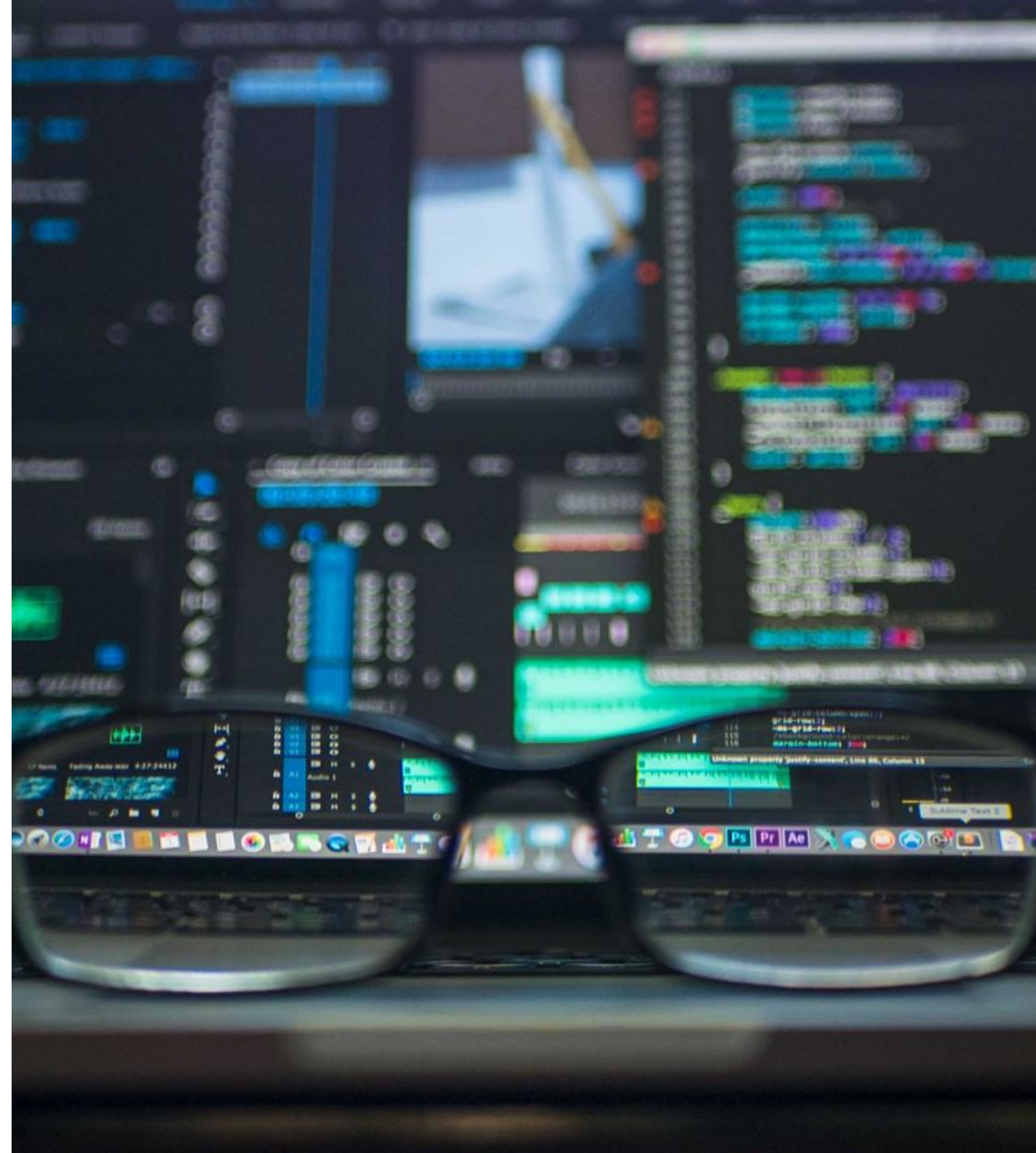
Un produit ne peut avoir qu'une seule Catégorie dans notre cas.



# Supprimer une instance

Supprimer un objet :

```
$em->remove($categorie);  
$em->flush();
```





# Un peu de pratique

Mettre en place le CRUD permettant d'interagir avec les produits.