

# SYMFONY

# INTRODUCTION

- Framework PHP open-source
- Crée par le lillois Fabien Potencier
- Largement utilisé



# HISTORIQUE



# CARACTÉRISTIQUES CLÉS

- Composants Réutilisables
- Architecture MVC
- Doctrine ORM
- Console
- Tests Automatisés
- Bundles
- Twig



# SYMFONY VS LARAVEL

- Modularité et Composants
- Maturité et Stabilité
- Flexibilité
- Communauté et Documentation
- Intégration et Écosystème

	Laravel	Symfony
Modularité et Composants	✓	✓
Maturité et Stabilité	✓	✓
Flexibilité	✗	✓
Communauté et Documentation	✗	✓
Intégration et Écosystème	✗	✓

SYMFONY



= SPRING  
BOOT



.NET

# Matrice de Pugh

Alternative Symfony

Critère	Poids	Symfony	Spring boot	.NET
Performance	1	0	1	1
Gestion des formulaires	2	0	-1	-1
Moteur de template	2	0	0	0
facilité d'apprentissage	3	0	+1	-1
documentation	3	0	-1	-1
Sécurité	1	0	-1	1
<b>Score</b>		<b>0</b>	<b>-2</b>	<b>-6</b>

# ARCHITECTURE MODULAIRE



# PRINCIPAUX CONCEPTS

Http  
Foundation

Dependency  
Injection

Routing

Security

Form

Validator



# HTTP FOUNDATION

- Manipulation des requêtes et des réponses HTTP.
- Simplification de la gestion des données provenant du client et les interactions entre le serveur et l'application.

# ROUTING

- Système de routage pour mapper les URL aux actions des contrôleurs .
- Définition des règles pour le traitement des requêtes HTTP .

# FORM

- Gestion simplifiée des formulaires (création + validation).
- Abstraction puissante pour la manipulation.

# VALIDATOR

- Outils pour la validation des données
- Garant de la qualité des données traitées

# SECURITY

- Gestion de l'authentification et de l'autorisation dans l'application.
- Protection des ressources sensibles et gestion d'accès des utilisateurs.

# DEPENDENCY INJECTION

- Gestion de la création et l'injection des services.
- Favorisant la réutilisation du code et l'organisation modulaire.

# SYNTAXE DE BASE



# DÉCLARATION DE VARIABLES

```
<?php  
  
$a_bool = TRUE;      // un booléen  
$a_str  = "foo";    // une chaîne de caractères  
$a_str2 = 'foo';    // une chaîne de caractères  
$an_int = 12;       // un entier  
  
echo gettype($a_bool); // affiche : boolean  
echo gettype($a_str); // affiche : string
```

# STRUCTURES DE CONTRÔLE

## 1. CONDITIONS

```
if ($condition) {  
    // Code à exécuter si la condition est vraie  
} elseif ($autre_condition) {  
    // Code à exécuter si l'autre condition est vraie  
} else {  
    // Code à exécuter si aucune condition n'est vraie  
}
```

# STRUCTURES DE CONTRÔLE

## 2. BOUCLES

```
for ($i = 0; $i < 5; $i++) {  
    // Code à répéter  
}
```

```
foreach ($tableau as $valeur) {  
    // Code à répéter pour chaque élément du tableau  
}
```

```
while ($condition) {  
    // Code à répéter tant que la condition est vraie  
}
```

# FONCTIONS

```
function maFonction($parametre1, $parametre2) {  
    // Code de la fonction  
    return $resultat;  
}  
  
// Appel de la fonction  
$resultat = maFonction($valeur1, $valeur2);
```

# CLASSES ET OBJETS

```
class MaClasse {  
    public $attribut;  
  
    public function __construct($valeur) {  
        $this->attribut = $valeur;  
    }  
  
    public function maMethode() {  
        // Code de la méthode  
    }  
}
```



# FLUENT SETTER

```
$user = new User();
$user->setFullName("GLOP")->setEmail("glop@shoploc.com");
```

# DOCTRINE QUERY LANGUAGE

```
public function findByUsername($username)
{
    return $this->createQueryBuilder('u')
        ->where('u.username = :username')
        ->setParameter('username', $username)
        ->getQuery()
        ->getResult();
}
```

# TWIG ET LES TEMPLATES

# POURQUOI TWIG ?

SÉPARATION CLAIRE

SÉCURITÉ  
INTÉGRÉE

MODULARITE

# CONTENU D'UN TEMPLATE TWIG

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1>{{ page_title }}</h1>

    {% if user.isLoggedIn %}
      Hello {{ user.name }}!
    {% endif %}

    {# ... #}
  </body>
</html>
```

# CONTROLLER ET LES ROUTES

```
class UserController extends AbstractController {  
  
    #[Route(path:"/user", name:"user.index", methods: ["GET"])]  
    0 references | 0 overrides  
    public function index() : Response {  
        return $this->render("users/index.html.twig");  
    }  
  
    #[Route(path:"/user/create", name:"recipe.index")]  
    0 references | 0 overrides  
    public function create(Request $request) : RedirectResponse {  
        // do something  
        return $this->redirectToRoute('homepage');  
    }  
}
```

# CONSOLE

# UTILITÉ

- **Productivité** : simplifie les tâches courantes de développement, automatisant des opérations et économisant du temps

# UTILITÉ

- **Productivité** : simplifie les tâches courantes de développement, automatisant des opérations et économisant du temps
- **Génération de code** : Le code est généré rapidement grâce à des commandes telles que make:controller et make:entity, accélérant la création de nouvelles fonctionnalités

# UTILITÉ

- **Productivité** : simplifie les tâches courantes de développement, automatisant des opérations et économisant du temps
- **Génération de code** : Le code est généré rapidement grâce à des commandes telles que make:controller et make:entity, accélérant la création de nouvelles fonctionnalités
- **Gestion de la base de données**

# UTILITÉ

- **Productivité** : simplifie les tâches courantes de développement, automatisant des opérations et économisant du temps
- **Génération de code** : Le code est généré rapidement grâce à des commandes telles que make:controller et make:entity, accélérant la création de nouvelles fonctionnalités
- **Gestion de la base de données**
- **Tests et débogage**

# PRINCIPALES COMMANDES SYMFONY

- Gestion du Serveur de Développement :

```
php bin/console server:run
```

- Génération de Contrôleur :

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

- Génération d'Entité (Doctrine) :

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

# PRINCIPALES COMMANDES SYMFONY

- Migrations (Doctrine) :

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

- Liste des Routes :

```
php bin/console debug:router
```

- Installation de Bundle :

```
php bin/console require nom_du_bundle
```

# VALIDATION ET CONTRAINTES

```
#[UniqueEntity('email')]  
#[ORM\Entity(repositoryClass: UserRepository::class)]  
13 references | 0 implementations  
class User {  
    #[ORM\Id]  
    #[ORM\GeneratedValue]  
    #[ORM\Column(type: 'integer')]  
    1 reference  
    private ?int $id;  
  
    #[ORM\Column(type: 'string', length: 50)]  
    #[Assert\NotBlank()]  
    #[Assert\Length(min: 2, max: 50)]  
    2 references  
    private string $name;  
  
    #[ORM\Column(type: 'string', length: 180, unique: true)]  
    #[Assert\Email()]  
    #[Assert\Length(min: 2, max: 180)]  
    3 references  
    private string $email;
```



# GESTION DES FORMULAIRES

# FORM TYPES

Pas de différenciation entre un formulaire et les champs d'un formulaire

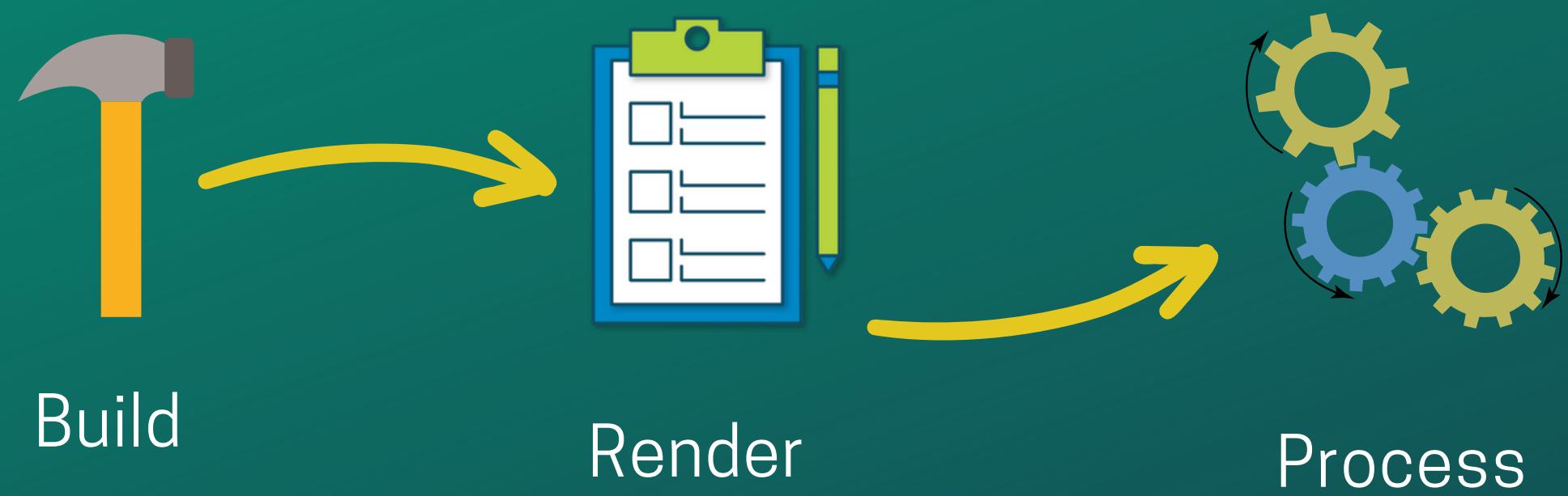
```
<input type="text">
```

InputText

```
<form>
```

UserType

# FORMULAIRE



# BUILD DANS LE CONTROLLER

```
class UserController extends AbstractController
{
    0 references | 0 overrides
    public function new(Request $request): Response
    {
        $user = new User();

        $form = $this->createFormBuilder($user)
            ->add('name', TextType::class)
            ->add('email', EmailType::class)
            ->add('save', SubmitType::class, ['label' => 'Create User'])
            ->getForm();

        // ...
    }
}
```

# BUILD DANS UNE CLASSE

```
class UserType extends AbstractType {  
    0 references | 0 overrides  
    public function buildForm(FormBuilderInterface $builder, array $options): void  
    {  
        $builder  
            ->add('name', TextType::class, ['attr' => [  
                'minlength' => '2',  
                'maxlength' => '50',  
            ],  
                'label' => 'Nom / Prénom')  
            ->add('plainPassword', PasswordType::class, ['label' => 'Mot de passe'])  
            ->add('submit', SubmitType::class);  
    }  
}
```

# RENDER

Dans le controller :

```
$form = $this->createForm(UserType::class, $user);
return $this->render('user/new.html.twig', [
    'form' => $form,
]);
```

Dans le template :

```
{# templates/user/new.html.twig #}
{{ form(form) }}
```

# PROCESS

Déroulement en 3 étapes :

1. Création et affichage du formulaire
2. Appel à la fonction handleRequest
3. Validation des données et actions



# SECURITY ET USER

# COMPOSANT DE SÉCURITÉ

Composant robuste offrant des fonctionnalités telles que l'authentification, l'autorisation, la gestion des rôles etc.

FIREWALLS

PROVIDERS

USER  
PROVIDERS

ENCODERS

VOTERS

# CONFIGURATION

```
security:
    encoders:
        App\Entity\User: bcrypt

    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email

    firewalls:
        secured_area:
            pattern: ^/secured
            anonymous: ~
            form_login:
                login_path: login
                check_path: login
            logout:
                path: /logout

    access_control:
        - { path: ^/secured, roles: ROLE_USER }

    role_hierarchy:
        ROLE_ADMIN:      ROLE_USER
        ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN]
```

