Bien sûr 👍 Je vais te donner des **exemples concrets et simples en PHP**, comme si tu les utilisais dans un petit projet web (formulaire d'inscription / login). Je pars du principe que tu utilises **Composer** et que l'autoload est déjà en place.

```
require __DIR__ . '/vendor/autoload.php';
```

1️⃣ `respect/validation` — Validation des données utilisateur

👉 Sert à **vérifier** les entrées (email valide, longueur d'un mot de passe, champs requis, etc.)

Exemple : validation d'un formulaire d'inscription

```php
use Respect\Validation\Validator as v;

$data = [
    'email' => $_POST['email'] ?? '',
    'password' => $_POST['password'] ?? '',
    'age' => $_POST['age'] ?? ''
];

// Règles de validation
$emailValidator = v::email()->notEmpty();
$passwordValidator = v::stringType()->length(8, null);
$ageValidator = v::intType()->between(18, 99);

// Vérifications
$errors = [];

if (!$emailValidator->validate($data['email'])) {
    $errors['email'] = "Email invalide";
}

if (!$passwordValidator->validate($data['password'])) {
    $errors['password'] = "Mot de passe trop court (min 8 caractères)";
}

if (!$ageValidator->validate($data['age'])) {
    $errors['age'] = "Âge invalide";
}

if (!empty($errors)) {
    // Gestion des erreurs
    print_r($errors);
} else {
    echo "Données valides ☑";
}
```

💡 **Avantage** : lisible, clair, réutilisable.

---

2 `symfony/security-core` — Hashage et vérification des mots de passe

👉 Sert à **hasher proprement** les mots de passe et à les vérifier au login.

Hasher un mot de passe (inscription)

```php
use Symfony\Component\PasswordHasher\Hasher\PasswordHasherFactory;

// Création de la factory
$factory = new PasswordHasherFactory([
    'default' => ['algorithm' => 'auto']
]);

$passwordHasher = $factory->getPasswordHasher('default');

// Mot de passe utilisateur
$plainPassword = $_POST['password'];

// Hash
$hashedPassword = $passwordHasher->hash($plainPassword);

// À stocker en base de données
echo $hashedPassword;
```

Vérifier un mot de passe (connexion)

```php
// Mot de passe soumis par l'utilisateur
$inputPassword = $_POST['password'];

// Mot de passe hashé récupéré depuis la base
$storedHash = $user['password'];

if ($passwordHasher->verify($storedHash, $inputPassword)) {
    echo "Connexion réussie ☑";
} else {
    echo "Mot de passe incorrect ❌";
}
```

💡 **Pourquoi Symfony plutôt que `password_hash()` ?**

- Algorithmes modernes (Argon2, bcrypt)
- Facilement configurable
- Propre pour projets évolutifs

3 `slim/csrf` — Protection CSRF pour formulaires

👉 Empêche qu'un formulaire soit soumis depuis un site malveillant.

Exemple avec Slim (PSR-7)

Middleware CSRF

```php
use Slim\Factory\AppFactory;
use Slim\Csrf\Guard;

$app = AppFactory::create();

$responseFactory = $app->getResponseFactory();
$csrf = new Guard($responseFactory);

$app->add($csrf);
```

Générer les tokens dans un formulaire

```php
$csrfNameKey = $request->getAttribute('csrf_name');
$csrfValueKey = $request->getAttribute('csrf_value');

$csrfName = $request->getAttribute($csrfNameKey);
$csrfValue = $request->getAttribute($csrfValueKey);
```

```html
<form method="POST">
    <input type="hidden" name="<?= $csrfNameKey ?>" value="<?= $csrfName ?>">
    <input type="hidden" name="<?= $csrfValueKey ?>" value="<?= $csrfValue ?>">

    <input type="email" name="email">
    <input type="password" name="password">
    <button type="submit">Envoyer</button>
</form>
```

Vérification automatique

👉 Si le token est invalide :

- la requête est **bloquée**
- Slim retourne une erreur 400

💡 Tu n'as **rien à vérifier manuellement**, le middleware s'en charge.

🧠 Résumé rapide

Dépendance	Sert à	Quand l'utiliser
`respect/validation`	Vérifier les données	Formulaires, API
`symfony/security-core`	Hasher / vérifier mdp	Login, inscription
`slim/csrf`	Protection CSRF	Formulaires POST

Si tu veux, je peux :

- te faire un **mini projet complet** (inscription + login)
- montrer l'intégration avec **PDO + MySQL**
- ou adapter les exemples **sans Slim**

Dis-moi 😉