Exercices Tests Unitaires

A/ MVC à la lyonnaise

Considérons cette classe Validator .

```
<?php

namespace Chanson;

/** Class Validator **/
class Validator {

    private $data;
    private $errors = [];
    private $messages = [
        "required" => "Le champ est requis !",
        "min" => "Le champ doit contenir un minimum de %^% lettres !",
        "max" => "Le champ doit contenir un maximum de %^% lettres !",
        "regex" => "Le format n'est pas respecté",
        "length" => "Le champ doit contenir %^% caractère(s) !",
        "url" => "Le champ doit correspondre à une url !",
        "email" => "Le champ doit correspondre à une email: exemple@gmail.con
        "date" => "Le champ doit être une date !",
        "alpha" => "Le champ peut contenir que des lettres minuscules et majusc
        "alphaNum" => "Le champ peut contenir que des lettres minuscules, maju
        "alphaNumDash" => "Le champ peut contenir que des lettres minuscules,
        "numeric" => "Le champ peut contenir que des chiffres !",
        "confirm" => "Le champs n'est pas conforme au confirm !",
        "alphaComplet" => "Le champ peut contenir que des lettres minuscules, r
    ];
    private $rules = [
        "required" => "#^.+$#",
```

```
    "min" ⇒ "#^.{ù,}$#",
    "max" ⇒ "#^.{0,ù}$#",
    "length" ⇒ "#^.{ù}$#",
    "regex" ⇒ "ù",
    "url" ⇒ FILTER_VALIDATE_URL,
    "email" ⇒ FILTER_VALIDATE_EMAIL,
    "date" ⇒ "#^(\d{4})(\/|-)(0[0-9]|1[0-2])(\/|-)([0-2][0-9]|3[0-1])$#",
    "alpha" ⇒ "#^[A-z]+$#",
    "alphaNum" ⇒ "#^[A-z0-9]+$#",
    "alphaNumDash" ⇒ "#^[A-z0-9-\|]+$#",
    "numeric" ⇒ "#^[0-9]+$#",
    "confirm" ⇒ "",
    "alphaComplet" ⇒ "#^[a-zA-Zàáâãäåçèéêëìíîïðòóôõöùúûüýÿ\.\-\s]+$#",
  ];

  public function __construct($data = []) {
    $this→data = $data ?: $_POST;
  }

  public function validate($array) {
    foreach ($array as $field ⇒ $rules) {
      $this→validateField($field, $rules);
    }

  }

  public function validateField($field, $rules) {
    foreach ($rules as $rule) {
      $this→validateRule($field, $rule);
    }
  }

  public function validateRule($field, $rule) {
    $res = strrpos($rule, ":");
    if ($res == true) {
      $repRule = explode(":", $rule);
      $changeRule = str_replace("ù", $repRule[1], $this→rules[$repRule[0]])
      $changeMessage = str_replace("%^%", $repRule[1], $this→messages
```

```
            if (!preg_match($changeRule, $this->data[$field])) {
                $this->errors = [$this->messages[$repRule[0]]];
                $this->storeSession($field, $changeMessage);
            }
        } elseif ($res == false) {
            if ($rule == "confirm") {
                if (!isset($this->data[$field . 'Confirm'])) {
                    $this->errors = ["Nous buttons sur un problème"];
                    $this->storeSession('confirm', "Nous buttons sur un problème");
                } elseif (isset($this->data[$field . 'Confirm']) && $this->data[$field] !
                    $this->errors = [$this->messages[$rule]];
                    $this->storeSession('confirm', $this->messages[$rule]);
                }
                return;
            }
            if ($rule == "email" || $rule == "url") {
                if (!filter_var($this->data[$field], $this->rules[$rule])) {
                    $this->errors = [$this->messages[$rule]];
                    $this->storeSession($field, $this->messages[$rule]);
                }
            }
            elseif (!preg_match($this->rules[$rule], $this->data[$field])) {
                $this->errors = [$this->messages[$rule]];
                $this->storeSession($field, $this->messages[$rule]);
            }
        }
    }

    public function errors() {
        return $this->errors;
    }

    public function storeSession($field, $error) {
        if (!isset($_SESSION["error"][$field])) {
            $_SESSION["error"][$field] = $error;
        } else {
            return;
```

```
        }
    }
}
```

On peut alors tester le fonctionnement de la fonction validate.

Voici une classe PHP simple nommée `Validator` (fournie ci-dessus), conçue pour valider des champs de formulaire (comme des emails, des dates, ou des longueurs minimales).

Elle est utilisée de cette manière :

```
$data = ['email' ⇒ 'test@example.com'];
$validator = new Validator($data);
$validator→validate(['email' ⇒ ['email']]);
```

Exemple de test unitaire avec PHPUnit

```
<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;
use Chanson\Validator;

class ValidatorTest extends TestCase
{
    private $validator;

    public function testEmailValidation()
    {
        $data = ['email' ⇒ 'test@example.com'];
        $validator = new Validator($data);
```

```
        $validator->validate([
            'email' => ['email']
        ]);

        $this->assertEmpty($validator->errors());
    }

    public function testEmailValidationFailure()
    {
        $data = ['email' => 'invalidemail@gmail'];
        $validator = new Validator($data);

        $validator->validate([
            'email' => ['email']
        ]);

        $this->assertNotEmpty($validator->errors());
    }
}
```

Explication du code:
- Avec ce test on cherche à vérifier que

- Lorsqu'on passe un email valide à la fonction validate de la classe Validator
- Lorsque pas un email invalid une erreur est bien générer

2 / Exercice : Réaliser les tests suivants

Voici la liste des tests à écrire. Tu peux les ajouter dans un fichier `ValidatorTest.php`.

✅ Test 1 : champ requis (présent)

```
public function testValidateRequiredFieldSuccess()
```

❌ Test 2 : champ requis (vide)

```
public function testValidateRequiredFieldFailure()
```

✅ Test 3 : longueur minimale OK

```
public function testValidateMinLengthSuccess()
```

❌ Test 4 : longueur minimale KO

```
public function testValidateMinLengthFailure()
```

✅ Test 5 : alphaComplet sans chiffres

```
public function testValidateAlphaCompletSuccess()
```

✅ Test 6 : alphaComplet avec accents

```
public function testValidateAlphaCompletWithAccents()
```

❌ Test 7 : alphaComplet avec chiffres (KO)

```
public function testValidateAlphaCompletFailure()
```

✅ Test 8 : date au bon format

```
public function testValidateDateSuccess()
```

❌ Test 9 : date au mauvais format

```
public function testValidateDateFailure()
```

✅ Test 10 : plusieurs règles OK

```
public function testValidateMultipleRules()
```

❌ Test 11 : plusieurs règles avec erreurs

```
public function testValidateMultipleRulesWithErrors()
```

✅ Test 12 : email correct

```
public function testEmailValidation()
```

❌ Test 13 : email incorrect

```
public function testEmailValidationFailure()
```

3/ Mise en place de PHPUnit

Installation via Composer :

```
composer require phpunit/phpunit
```

Arborescence recommandée :

```
project/
├── src/
│   └── Validator.php
├── tests/
│   └── ValidatorTest.php
├── composer.json
```

Dans `composer.json`, ajoute :

```
"autoload": {
   "psr-4": {
     "Chanson\\": "src/"
     "Tests\\": "tests/"
   }
}
```

Puis, recharge l'autoloader :

```
composer dump-autoload
```

Pour lancer les tests :

```
./vendor/bin/phpunit tests
```

4 / Correction des exercices

```
<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;
use Chanson\Validator;

class ValidatorTest extends TestCase
{
   private $validator;

   public function testValidateRequiredFieldSuccess()
   {
      $data = ['titre' ⇒ 'Ma chanson'];
```

```
        $validator = new Validator($data);

        $validator->validate([
            'titre' => ['required']
        ]);

        $this->assertEmpty($validator->errors());
    }

    public function testValidateRequiredFieldFailure()
    {
        $data = ['titre' => ''];
        $validator = new Validator($data);

        $validator->validate([
            'titre' => ['required']
        ]);

        $this->assertNotEmpty($validator->errors());
        $this->assertContains('Le champ est requis !', $validator->errors());
    }

    public function testValidateMinLengthSuccess()
    {
        $data = ['titre' => 'Chanson'];
        $validator = new Validator($data);

        $validator->validate([
            'titre' => ['min:2']
        ]);

        $this->assertEmpty($validator->errors());
    }

    public function testValidateMinLengthFailure()
    {
        $data = ['titre' => 'A'];
        $validator = new Validator($data);
```

```
    $validator->validate([
        'titre' => ['min:2']
    ]);

    $this->assertNotEmpty($validator->errors());
}

public function testValidateAlphaCompletSuccess()
{
    $data = ['titre' => 'Ma Belle Chanson'];
    $validator = new Validator($data);

    $validator->validate([
        'titre' => ['alphaComplet']
    ]);

    $this->assertEmpty($validator->errors());
}

public function testValidateAlphaCompletWithAccents()
{
    $data = ['titre' => 'Chanson à écouter'];
    $validator = new Validator($data);

    $validator->validate([
        'titre' => ['alphaComplet']
    ]);

    $this->assertEmpty($validator->errors());
}

public function testValidateAlphaCompletFailure()
{
    $data = ['titre' => 'Chanson123'];
    $validator = new Validator($data);

    $validator->validate([
```

```
            'titre' ⇒ ['alphaComplet']
        ]);

        $this→assertNotEmpty($validator→errors());
    }

    public function testValidateDateSuccess()
    {
        $data = ['date' ⇒ '2024-01-20'];
        $validator = new Validator($data);

        $validator→validate([
            'date' ⇒ ['date']
        ]);

        $this→assertEmpty($validator→errors());
    }

    public function testValidateDateFailure()
    {
        $data = ['date' ⇒ '20-01-2024'];
        $validator = new Validator($data);

        $validator→validate([
            'date' ⇒ ['date']
        ]);

        $this→assertNotEmpty($validator→errors());
    }

    public function testValidateMultipleRules()
    {
        $data = [
            'titre' ⇒ 'Ma Chanson',
            'date' ⇒ '2024-01-20'
        ];
        $validator = new Validator($data);
```

```
        $validator->validate([
            'titre' => ['required', 'min:2', 'alphaComplet'],
            'date' => ['required', 'date']
        ]);

        $this->assertEmpty($validator->errors());
    }

    public function testValidateMultipleRulesWithErrors()
    {
        $data = [
            'titre' => '',
            'date' => 'invalid-date'
        ];
        $validator = new Validator($data);

        $validator->validate([
            'titre' => ['required', 'min:2', 'alphaComplet'],
            'date' => ['required', 'date']
        ]);

        $this->assertNotEmpty($validator->errors());
    }

    public function testEmailValidation()
    {
        $data = ['email' => 'test@example.com'];
        $validator = new Validator($data);

        $validator->validate([
            'email' => ['email']
        ]);

        $this->assertEmpty($validator->errors());
    }

    public function testEmailValidationFailure()
    {
```

```
        $data = ['email' => 'invalid-email'];
        $validator = new Validator($data);

        $validator->validate([
            'email' => ['email']
        ]);

        $this->assertNotEmpty($validator->errors());
    }
}
```

B/ MVC Paris

Considérons cette classe **Validator** ainsi que ce model **User**

```
<?php

namespace App;

class Validator
{
    public function validate($data, $model)
    {
        if (!class_exists($model)) {
            http_response_code(500);
            include './src/views/errors/page500.php';
            exit();
        }

        $instance = new $model();
        foreach ($data as $key => $value) {
            $setter = 'set' . ucfirst(str_replace('_', '', ucwords($key, '_')));
            if (method_exists($instance, $setter)) {
                $instance->$setter($value);
            }
```

```
        }
        return $instance;
    }
}
```

```
<?php


namespace App\models;

use App\config\Database;

class User
{
    private $idClient;
    private $firstName;
    private $lastName;
    private $email;
    private $mdp;
    private $mdpValid;
    private $role = 'client';
    private $isValid = true;
    private $errors = array();
    public function setIdClient($idClient) {
        $this->idClient = $idClient;
    }
    public function setFirstName($firstName)
    {
        if (strlen($firstName) < 3 || strlen($firstName) > 20 || !preg_match('/^[a-z
            $this->isValid = false;
            return $this->errors += ["firstName" => "Le format du champ est incorr
        }
        return $this->firstName = htmlspecialchars($firstName);
    }

    public function setLastName($lastName)
```

```
    {
        if (strlen($lastName) < 3 || strlen($lastName) > 20 || !preg_match('/^[a-z
            $this->isValid = false;
            return $this->errors += ["lastName" => "Le format du champ est incorre
        }
        return $this->lastName = htmlspecialchars($lastName);
    }

    public function setEmail($email)
    {
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $this->isValid = false;
            return $this->errors += ["email" => "Le format d'email est incorrect"];
        }
        return $this->email = $email;
    }

    public function setPassword($password)
    {
        if (strlen($password) < 8 || strlen($password) > 20) {
            $this->isValid = false;
            return $this->errors += ["password" => "Le mot de passe n'est pas vali
        }
        return $this->mdp = $password;
    }



    public function setPasswordValid($passwordValid)
    {
        if (strlen($passwordValid) < 8 || strlen($passwordValid) > 20) {
            $this->isValid = false;
            return $this->errors += ["passwordValid" => "Le mot de passe n'est pas
        }
        return $this->mdpValid = $passwordValid;
    }

    public function checkMatchTwoPassword($arg1, $arg2)
    {
```

```
        if ($arg1 !== $arg2) {
            $this->isValid = false;
            return $this->errors += ["passwordMatch" => "Les mots de passe ne c
        }
    }

    public function checkMailExist($arg1, $arg2)
    {
        if ($arg1 === $arg2) {
            $this->isValid = false;
            return $this->errors += ["MailExist" => "Le mail est déja utiliser"];
        }
    }

    public function checkIsValid()
    {
        return $this->isValid;
    }

    public function getErrors()
    {
        return $this->errors;
    }

    public function getLastName()
    {
        return $this->lastName;
    }

    public function getFirstName()
    {
        return $this->firstName;
    }

    public function getEmailAddress()
    {
        return $this->email;
    }
```

```
    public function getPassword()
    {
        return $this->password;
    }

    public function getData()
    {
        return ['prenom' => $this->firstName, 'nom' => $this->lastName, 'email' =>
    }
    public function getDataEdit() {
        return ['id_client' => $this->idClient, 'prenom' => $this->firstName, 'nom' =
    }


    public function verifyPassword($pass)
    {
        if ($this->mdp === $pass) {
            return true;
        } else {
            return false;
        }
    }
}
```

Exemple de test pour vérifier que la validation du firstName est fonctionnelle

L'objectif de ces deux tests est de
vérifier le comportement de la méthode `setEmail()` dans le modèle `User`.

```
<?php

use PHPUnit\Framework\TestCase;
use App\models\User;
use App\Validator;
```

```
class UserTest extends TestCase
{

    public function testSetEmailValid()
    {
        $user = new User();
        $user->setEmail('jean.dupont@gmail.com');
        $this->assertTrue($user->checkIsValid());
        $this->assertEquals('jean.dupont@gmail.com', $user->getEmailAddress()
    }

    public function testSetEmailInvalid()
    {
        $user = new User();
        $user->setEmail('invalid-email');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('email', $user->getErrors());
    }
}
```

- **testSetEmailValid()**
  - **But :** Tester que l'adresse e-mail est bien acceptée si elle est valide.
  - On crée une instance de `User`.
  - On appelle la méthode `setEmail()` avec une adresse e-mail bien formée.
  - On vérifie que :
    - `checkIsValid()` retourne `true`, ce qui signifie qu'aucune erreur de validation n'a été détectée.
    - `getEmailAddress()` retourne bien l'e-mail qu'on a fourni.

  ✅ Si les deux assertions passent, cela prouve que le setter accepte correctement une adresse valide.

- **testSetEmailInvalid()**

- **But :** Vérifier que l'e-mail invalide est rejeté.
- On donne une chaîne incorrecte comme adresse e-mail (`invalid-email`).
- On teste que :
  - `checkIsValid()` retourne `false`, ce qui signifie qu'une erreur de validation a été enregistrée.
  - L'erreur associée à la clé `'email'` est bien présente dans le tableau retourné par `getErrors()`.

✅ Cela garantit que la validation empêche des données incorrectes d'être enregistrées dans l'objet.

2 / Exercice : Réaliser les tests suivants

Voici la liste des tests à écrire. Tu peux les ajouter dans un fichier `ValidatorTest.php`.

✅ Test 1 : prénom valide (présent, format correct)

```
public function testSetFirstNameValid()
```

❌ Test 2 : prénom invalide (vide ou trop court)

```
public function testSetFirstNameInvalid()
```

✅ Test 3 : email valide

```
public function testSetEmailValid()
```

❌ Test 4 : email invalide (mauvais format)

```
public function testSetEmailInvalid()
```

✅ Test 5 : mot de passe avec longueur correcte

```
public function testSetPasswordValid()
```

❌ Test 6 : mot de passe trop court

```
public function testSetPasswordTooShort()
```

❌ Test 7 : mot de passe trop long

```
public function testSetPasswordTooLong()
```

✅ Test 8 : mots de passe identiques

```
public function testPasswordsMatch()
```

❌ Test 9 : mots de passe différents

```
public function testPasswordsDoNotMatch()
```

3/ Correction

```
<?php

use PHPUnit\Framework\TestCase;
use App\models\User;
use App\Validator;

class UserTest extends TestCase
{
    public function testSetFirstNameValid()
    {
        $user = new User();
        $user->setFirstName('Jean');
        $this->assertTrue($user->checkIsValid());
        $this->assertEquals('Jean', $user->getFirstName());
    }
```

```
    public function testSetFirstNameInvalid()
    {
        $user = new User();
        $user->setFirstName('J');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('firstName', $user->getErrors());
    }

    public function testSetLastNameValid()
    {
        $user = new User();
        $user->setLastName('Dupont');
        $this->assertTrue($user->checkIsValid());
        $this->assertEquals('Dupont', $user->getLastName());
    }

    public function testSetLastNameInvalid()
    {
        $user = new User();
        $user->setLastName('D1');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('lastName', $user->getErrors());
    }

    public function testSetEmailValid()
    {
        $user = new User();
        $user->setEmail('jean.dupont@gmail.com');
        $this->assertTrue($user->checkIsValid());
        $this->assertEquals('jean.dupont@gmail.com', $user->getEmailAddress()
    }

    public function testSetEmailInvalid()
    {
        $user = new User();
        $user->setEmail('invalid-email');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('email', $user->getErrors());
```

```
    }

    public function testSetPasswordValid()
    {
        $user = new User();
        $user->setPassword('12345678');
        $this->assertTrue($user->checkIsValid());
    }

    public function testSetPasswordInvalid()
    {
        $user = new User();
        $user->setPassword('short');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('password', $user->getErrors());
    }

    public function testPasswordMatching()
    {
        $user = new User();
        $user->setPassword('abc12345');
        $user->setPasswordValid('abc12345');
        $user->checkMatchTwoPassword('abc12345', 'abc12345');
        $this->assertTrue($user->checkIsValid());
    }

    public function testPasswordNotMatching()
    {
        $user = new User();
        $user->setPassword('abc12345');
        $user->setPasswordValid('wrongpass');
        $user->checkMatchTwoPassword('abc12345', 'wrongpass');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('passwordMatch', $user->getErrors());
    }

    public function testCheckMailExist()
    {
```

```
        $user = new User();
        $user->checkMailExist('existing@mail.com', 'existing@mail.com');
        $this->assertFalse($user->checkIsValid());
        $this->assertArrayHasKey('MailExist', $user->getErrors());
    }

    public function testValidatorHydratesModel()
    {
        $data = [
            'first_name' => 'Jean',
            'last_name' => 'Dupont',
            'email' => 'jean.dupont@gmail.com',
            'password' => 'monmotdepasse',
            'password_valid' => 'monmotdepasse'
        ];

        $validator = new Validator();
        $user = $validator->validate($data, \App\models\User::class);

        $this->assertInstanceOf(User::class, $user);
        $this->assertEquals('Jean', $user->getFirstName());
        $this->assertEquals('Dupont', $user->getLastName());
        $this->assertEquals('jean.dupont@gmail.com', $user->getEmailAddress()
    }
}
```