Bonnes pratiques Git - EcoRide

Projet de covoiturage écoresponsable

VI. Tests et bonnes pratiques GIT

A. Tests

i. Tests du back-end

Les deux tests fonctionnels suivants réalisés avec PHPUnit couvrent la soumission d'un formulaire d'inscription :

- Un test avec des données valides et 2FA activé,
- Un test avec des données invalides, pour vérifier la validation du formulaire.

Avant de créer ces deux tests, il convient d'installer les packages nécessaires aux tests :

```
$  composer require --dev symfony/test-pack
```

Puis, nous testons dans la console :

```
$  php bin/phpunit
```

Nous pouvons créer notre test :

```
$  php bin/console make:test
```

```
Which test type would you like?:
 [TestCase       ] basic PHPUnit tests
 [KernelTestCase ] basic tests that have access to Symfony services
 [WebTestCase    ] to run browser-like scenarios, but that don't execute JavaScript code
 [ApiTestCase    ] to run API-oriented scenarios
 [PantherTestCase] to run e2e scenarios, using a real-browser or HTTP client and a real web server
```

```
Which test type would you like?:
> WebTestCase

The name of the test class (e.g. BlogPostTest):
 > Controller\RegisterUserTest
```

Cela va créer la classe de test.

Symfony Flex crée automatiquement **phpunit.xml.dist** et **tests/bootstrap.php**. Si ces fichiers sont manquants, nous pouvons essayer d'exécuter la recette en utilisant :
composer recipes:install phpunit/phpunit --force -v

Les tests qui interagissent avec la base de données doivent utiliser leur propre base de données distincte pour ne pas perturber les bases de données utilisées dans les autres environnements de configuration.

```
$  php bin/console --env=test doctrine:database:create
   php bin/console --env=test doctrine:schema:create
```

1. Test d'inscription avec 2FA activé

Test : « testSomething() »

Ce test vérifie que le formulaire d'inscription fonctionne avec des données valides. Il suit les étapes suivantes :

1. Accès à la page « /inscription »
2. Soumission du formulaire avec un email et pseudo uniques
3. Redirection attendue vers « /login/success »
4. Vérification du contenu de la redirection

Code utilisé :

```php
    public function testSomething(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/inscription');

        $this->assertResponseIsSuccessful();
        $this->assertSelectorExists('form[name="registration_form"]');

        $randomSuffix = uniqid();

        $client->submitForm("S'enregistrer", [
            'registration_form[pseudo]' => 'testuser_' . $randomSuffix,
            'registration_form[email]' => 'testuser_' . $randomSuffix . '@example.com',
            'registration_form[plainPassword]' => '%&dMpD5#K#<btSG1SCQrBgpL',
            'registration_form[firstname]' => 'Test',
            'registration_form[lastname]' => 'User',
            'registration_form[phone]' => '0612345678',
            'registration_form[adress]' => '12 rue de la Victoire',
            'registration_form[postalCode]' => '75001',
            'registration_form[city]' => 'Paris',
            'registration_form[agreeTerms]' => true,
        ]);

        $this->assertResponseRedirects('/login/success');
        $client->followRedirect();

        // Facultatif
        $this->assertSelectorTextContains('body', 'Redirecting');
    }
```

2. Test de validation des erreurs de formulaire

Test : « testRegistrationFormValidationErrors() »

Ce test soumet un formulaire avec des données invalides pour déclencher les erreurs de validation Symfony. Il vérifie que chaque champ retourne bien un message d'erreur attendu.

Code utilisé :

```
    public function testRegistrationFormValidationErrors(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/inscription');

        $crawler = $client->submitForm("S'enregistrer", [
            'registration_form[pseudo]' => '',
            'registration_form[email]' => 'not-an-email',
            'registration_form[plainPassword]' => 'abc',
            'registration_form[firstname]' => '',
            'registration_form[lastname]' => '',
            'registration_form[phone]' => '',
            'registration_form[adress]' => '',
            'registration_form[postalCode]' => '123',
            'registration_form[city]' => '',
            'registration_form[agreeTerms]' => false,
        ]);

        $this->assertResponseStatusCodeSame(422);
        $this->assertSelectorExists('.is-invalid');

        // Vérification par champ
        $this->assertSelectorTextContains('#registration_form_pseudo + .invalid-feedback',
'Merci d\'indiquer votre pseudo');
        $this->assertSelectorTextContains('#registration_form_email + .invalid-feedback',
'adresse e-mail');
        $this->assertSelectorTextContains('#registration_form_postalCode + .invalid-
feedback', 'code postal');
        $this->assertSelectorTextContains('#registration_form_city + .invalid-feedback',
'ville');
    }
```

Ces tests couvrent deux aspects essentiels : le bon déroulement du processus d'inscription, et la robustesse de la validation côté serveur. Les tests utilisent la classe « WebTestCase » de Symfony et simulent le parcours réel d'un utilisateur dans un navigateur via le client HTTP de test.

```
$  php bin/phpunit
   OK (2 tests, 16 assertions)
```

B. Utilisation de Git et gestion des branches

i. Contexte

Dans le cadre de cet ECF, j'ai dû travailler seul sur mon dépôt GitHub, en raison de contraintes professionnelles m'empêchant de collaborer avec d'autres développeurs.
En conséquence, aucune branche de développement distincte n'a été créée : l'ensemble du développement a été réalisé directement sur la branche principale (**main**).

Toutefois, afin d'illustrer et appliquer les bonnes pratiques de gestion de versions, une branche de test a été créée spécifiquement pour intégrer deux tests fonctionnels (enregistrement d'un utilisateur et validation d'un formulaire), puis fusionnée dans main.

ii. Procédure suivie

Objectif :

Mettre en place une branche de test à partir de la branche principale, y intégrer les tests fonctionnels, puis effectuer une fusion vers main après validation.

Étapes réalisées :

1. Vérification de la branche principale

```
$  git checkout main
   git status
```

Le terminal doit afficher : On branch main.

2. Création de la branche de test

```
$  git checkout -b test
```

Cette commande crée et bascule directement sur la branche test.

3. Ajout et commit des modifications

```
$  git add .
   git commit -m "Ajout de deux tests fonctionnels pour l'enregistrement d'un
   utilisateur et la validation du formulaire"
```

4. Envoi de la branche test vers GitHub

```
$  git push -u origin test
```

Cette étape crée la branche sur GitHub avec le contenu commité.

5. Retour sur la branche principale

```
$  git checkout main
```

6. Fusion des modifications de test dans main

```
$  git merge test
```

Un message de confirmation s'affiche si la fusion est effectuée avec succès.

```
Updating e388f87..4ed3fe1
Fast-forward
 phpunit.dist.xml                          |   3 --
 public/images/avatars/6895bfd71996c.png | Bin 0 -> 3733 bytes
 tests/Controller/RegisterUserTest.php     |  66 ++++++++++++++++++++++++++++++++++
 3 files changed, 66 insertions(+), 3 deletions(-)
 create mode 100644 public/images/avatars/6895bfd71996c.png
 create mode 100644 tests/Controller/RegisterUserTest.php
```

7. Mise à jour de main sur GitHub

```
$  git push origin main
```

8. Suppression de la branche de test (optionnel)

```
$  git branch -d test
   git push origin --delete test
```

iii. Synthèse

- L'utilisation de git **checkout -b** permet de créer une branche à partir de l'historique complet, garantissant la sécurité du code.
- Cette approche est recommandée pour tout développement, test ou correctif.
- L'utilisation de **git checkout --orphan** est déconseillée en dehors de cas spécifiques (réinitialisation complète, documentation séparée, etc.).

iv. Schéma illustratif