AL

TDD avec Laravel et PHPUnit

Prérequis

1. Laravel et PHPUnit installés.
2. Une base de données configurée pour l'environnement de tests.
3. Une copie fonctionnelle de l'application **Chirper (Vous pouvez cloner le projet d'un camarade)**

Exercice 1 : Création de "chirps"

Objectif

Permettre à un utilisateur connecté de publier un "chirp".

Consignes

- Simulez un utilisateur connecté avec `actingAs`.
- Effectuez une requête `POST` à la route `/chirps` avec un contenu.
- Vérifiez que le "chirp" est enregistré en base de données.

Exemple de test

```
public function test_un_utilisateur_peut_creer_un_chirp()
{
    // Simuler un utilisateur connecté
    $utilisateur = User::factory()->create();
    $this->actingAs($utilisateur);

    // Envoyer une requête POST pour créer un chirp
```

```
    $reponse = $this->post('/chirps', [
        'content' => 'Mon premier chirp !'
    ]);

    // Vérifier que le chirp a été ajouté à la base de donnée
s
    $reponse->assertStatus(201);
    $this->assertDatabaseHas('chirps', [
        'content' => 'Mon premier chirp !',
        'user_id' => $utilisateur->id,
    ]);
}
```

Exercice 2 : Validation des "chirps"

Objectif

S'assurer que les règles de validation empêchent :

- Un contenu vide.
- Un contenu de plus de 255 caractères.

Consignes

- Testez qu'une tentative avec un contenu vide échoue.
- Testez qu'une tentative avec un contenu trop long échoue.

Exemple de test

```
public function test_un_chirp_ne_peut_pas_avoir_un_contenu_vi
de()
{
    $utilisateur = User::factory()->create();
    $this->actingAs($utilisateur);

    $reponse = $this->post('/chirps', [
```

```
        'content' => ''
    ]);

    $reponse->assertSessionHasErrors(['contenu']);
}

public function test_un_chirp_ne_peut_pas_depasse_255_caracte
res()
{
    $utilisateur = User::factory()->create();
    $this->actingAs($utilisateur);

    $reponse = $this->post('/chirps', [
        'content' => str_repeat('a', 256)
    ]);

    $reponse->assertSessionHasErrors(['contenu']);
}
```

Exercice 3 : Affichage des "chirps"

Objectif

Afficher les "chirps" existants sur la page d'accueil.

Consignes

- Créez plusieurs "chirps".
- Testez que tous les "chirps" sont affichés dans la réponse d'une requête `GET` sur la page d'accueil.

Exemple de test

```
public function test_les_chirps_sont_affiches_sur_la_page_d_a
ccueil()
{
```

```
    $chirps = Chirp::factory()->count(3)->create();

    $reponse = $this->get('/');

    foreach ($chirps as $chirp) {
        $reponse->assertSee($chirp->contenu);
    }
}
```

Exercice 4 : Mise à jour d'un "chirp"

Objectif

Permettre à un utilisateur de modifier son propre "chirp".

Consignes

- Créez un utilisateur et un "chirp".
- Effectuez une requête `PUT` pour mettre à jour le contenu.
- Vérifiez que la modification est sauvegardée.

Exemple de test

```
public function test_un_utilisateur_peut_modifier_son_chirp()
{
    $utilisateur = User::factory()->create();
    $chirp = Chirp::factory()->create(['user_id' => $utilisat
eur->id]);

    $this->actingAs($utilisateur);

    $reponse = $this->put("/chirps/{$chirp->id}", [
        'content' => 'Chirp modifié'
    ]);
```

```
    $reponse->assertStatus(200);
    // Vérifie si le chirp existe dans la base de donnée.
    $this->assertDatabaseHas('chirps', [
        'id' => $chirp->id,
        'content' => 'Chirp modifié',
    ]);
}
```

Exercice 5 : Suppression d'un "chirp"

Objectif

Permettre à un utilisateur de supprimer son propre "chirp".

Consignes

- Créez un utilisateur et un "chirp".
- Effectuez une requête `DELETE` pour supprimer le "chirp".
- Vérifiez qu'il n'existe plus en base de données.

Exemple de test

```
public function test_un_utilisateur_peut_supprimer_son_chirp
()
{
    $utilisateur = User::factory()->create();
    $chirp = Chirp::factory()->create(['user_id' => $utilisat
eur->id]);

    $this->actingAs($utilisateur);

    $reponse = $this->delete("/chirps/{$chirp->id}");

    $reponse->assertStatus(200);
```

```
    $this->assertDatabaseMissing('chirps', [
        'id' => $chirp->id,
    ]);
}
```

Exercice 6 : Permissions pour les "chirps"

Objectif

Empêcher un utilisateur de modifier ou de supprimer le "chirp" d'un autre utilisateur.

Consignes

- Créez deux utilisateurs différents.
- Testez que l'utilisateur 2 ne peut ni modifier ni supprimer le "chirp" de l'utilisateur 1.

Exercice 7 : Validation lors de la mise à jour

Objectif

S'assurer que les mêmes règles de validation s'appliquent lors de la mise à jour d'un "chirp".

Consignes

- Testez qu'un contenu vide ou trop long n'est pas accepté lors de la modification.

Exercice 8 : Nombre maximum de "chirps" par utilisateur

Objectif

Limiter un utilisateur à un maximum de 10 "chirps".

Consignes

- Testez qu'un utilisateur avec 10 "chirps" ne peut pas en créer un 11e.
- Vérifiez que le système retourne une erreur appropriée.

Exercice 9 : Filtrage des "chirps" dans l'affichage

Objectif

Afficher uniquement les "chirps" créés dans les 7 derniers jours.

Consignes

- Créez des "chirps" avec différentes dates.
- Vérifiez que seuls les "chirps" récents sont affichés dans la réponse.

Exercice 10 : Fonctionnalité "like" pour les "chirps"

Objectif

Permettre aux utilisateurs d'aimer un "chirp".

Consignes

- Créez une fonctionnalité pour liker un "chirp".
- Testez qu'un utilisateur peut liker un "chirp" et que cela est enregistré en base de données.
- Testez qu'un utilisateur ne peut pas liker deux fois le même "chirp".

Commandes utiles

- **Créer un fichier de test** :

```
php artisan make:test ChirpTest
```