Feature Tests for CRUD Operations with Authentication

Here's a comprehensive guide to writing PHPUnit feature tests for CRUD operations with authentication in Laravel.

1. Setup Test Environment

First, ensure your `phpunit.xml` is properly configured:

```
<php>
    <env name="APP_ENV" value="testing"/>
    <env name="DB_CONNECTION" value="sqlite"/>
    <env name="DB_DATABASE" value=":memory:"/>
    <env name="SESSION_DRIVER" value="array"/>
    <env name="QUEUE_CONNECTION" value="sync"/>
</php>
```

2. Test Case Base Class

Create a base test case class for authentication:

```
// tests/Feature/AuthenticatedTestCase.php

namespace Tests\Feature;

use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;

abstract class AuthenticatedTestCase extends TestCase
{
    use RefreshDatabase, WithFaker;

    protected $user;
    protected $authToken;

    protected function setUp(): void
    {
        parent::setUp();

        // Create and authenticate a test user
        $this->user = User::factory()->create();
        $this->authToken = $this->user->createToken('test-token')-
>plainTextToken;
```

PROF

```
    }

    protected function authenticatedJson($method, $uri, array $data =
[], array $headers = [])
    {
        $headers['Authorization'] = 'Bearer ' . $this->authToken;
        return $this->json($method, $uri, $data, $headers);
    }
}
```

3. User Registration Test

```
// tests/Feature/Auth/RegistrationTest.php

namespace Tests\Feature\Auth;

use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class RegistrationTest extends TestCase
{
    use RefreshDatabase;

    public function test_new_users_can_register()
    {
        $response = $this->postJson('/api/register', [
            'name' => 'Test User',
            'email' => 'test@example.com',
            'password' => 'password',
            'password_confirmation' => 'password',
        ]);

        $response->assertStatus(201)
            ->assertJsonStructure([
                'data' => [
                    'user' => [
                        'id',
                        'name',
                        'email',
                    ],
                    'access_token'
                ]
            ]);

        $this->assertDatabaseHas('users', [
            'email' => 'test@example.com'
        ]);
    }
```

```
    public function test_registration_validation()
    {
        // Test required fields
        $response = $this->postJson('/api/register', []);
        $response->assertStatus(422)
            ->assertJsonValidationErrors(['name', 'email', 'password']);

        // Test password confirmation
        $response = $this->postJson('/api/register', [
            'name' => 'Test User',
            'email' => 'test@example.com',
            'password' => 'password',
            'password_confirmation' => 'wrong',
        ]);
        $response->assertStatus(422)
            ->assertJsonValidationErrors(['password']);

        // Test unique email
        User::factory()->create(['email' => 'test@example.com']);
        $response = $this->postJson('/api/register', [
            'name' => 'Test User',
            'email' => 'test@example.com',
            'password' => 'password',
            'password_confirmation' => 'password',
        ]);
        $response->assertStatus(422)
            ->assertJsonValidationErrors(['email']);
    }
}
```

4. Authentication Test

PROF

```
// tests/Feature/Auth/AuthenticationTest.php

namespace Tests\Feature\Auth;

use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class AuthenticationTest extends TestCase
{
    use RefreshDatabase;

    public function test_users_can_authenticate()
    {
        $user = User::factory()->create();

        $response = $this->postJson('/api/login', [
            'email' => $user->email,
```

```
            'password' => 'password',
        ]);

        $response->assertStatus(200)
            ->assertJsonStructure([
                'data' => [
                    'access_token',
                    'token_type'
                ]
            ]);
    }

    public function
test_users_can_not_authenticate_with_invalid_password()
    {
        $user = User::factory()->create();

        $response = $this->postJson('/api/login', [
            'email' => $user->email,
            'password' => 'wrong-password',
        ]);

        $response->assertStatus(401)
            ->assertJson(['message' => 'Invalid credentials']);
    }

    public function test_authenticated_user_can_get_their_profile()
    {
        $user = User::factory()->create();
        $token = $user->createToken('test-token')->plainTextToken;

        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->getJson('/api/me');

        $response->assertStatus(200)
            ->assertJson([
                'data' => [
                    'email' => $user->email
                ]
            ]);
    }

    public function test_users_can_logout()
    {
        $user = User::factory()->create();
        $token = $user->createToken('test-token')->plainTextToken;

        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->postJson('/api/logout');

        $response->assertStatus(200)
```

```
            ->assertJson(['message' => 'Successfully logged out']);

        $this->assertCount(0, $user->tokens);
    }
}
```

5. CRUD Operations Test

Assuming we have a `Post` model with CRUD operations:

```
// tests/Feature/PostCrudTest.php

namespace Tests\Feature;

use App\Models\Post;
use Tests\Feature\AuthenticatedTestCase;

class PostCrudTest extends AuthenticatedTestCase
{
    public function test_user_can_create_post()
    {
        $postData = [
            'title' => 'Test Post',
            'content' => 'This is a test post content',
        ];

        $response = $this->authenticatedJson('POST', '/api/posts',
$postData);

        $response->assertStatus(201)
            ->assertJsonStructure([
                'data' => [
                    'id',
                    'title',
                    'content',
                    'user_id',
                    'created_at',
                ]
            ])
            ->assertJson([
                'data' => [
                    'title' => 'Test Post',
                    'user_id' => $this->user->id,
                ]
            ]);

        $this->assertDatabaseHas('posts', [
            'title' => 'Test Post',
            'user_id' => $this->user->id,
        ]);
```

```
    }

    public function test_user_can_view_their_posts()
    {
        $post = Post::factory()->create(['user_id' => $this->user->id]);

        $response = $this->authenticatedJson('GET', '/api/posts');

        $response->assertStatus(200)
            ->assertJsonStructure([
                'data' => [
                    '*' => [
                        'id',
                        'title',
                        'content',
                        'user_id',
                    ]
                ]
            ])
            ->assertJsonFragment([
                'id' => $post->id,
                'title' => $post->title,
            ]);
    }

    public function test_user_can_view_single_post()
    {
        $post = Post::factory()->create(['user_id' => $this->user->id]);

        $response = $this->authenticatedJson('GET', "/api/posts/{$post->id}");

        $response->assertStatus(200)
            ->assertJson([
                'data' => [
                    'id' => $post->id,
                    'title' => $post->title,
                    'content' => $post->content,
                ]
            ]);
    }

    public function test_user_can_update_their_post()
    {
        $post = Post::factory()->create(['user_id' => $this->user->id]);

        $updateData = [
            'title' => 'Updated Title',
            'content' => 'Updated content',
        ];

        $response = $this->authenticatedJson('PUT', "/api/posts/{$post->id}", $updateData);
```

```
        $response->assertStatus(200)
            ->assertJson([
                'data' => [
                    'id' => $post->id,
                    'title' => 'Updated Title',
                    'content' => 'Updated content',
                ]
            ]);

        $this->assertDatabaseHas('posts', [
            'id' => $post->id,
            'title' => 'Updated Title',
        ]);
    }

    public function test_user_can_delete_their_post()
    {
        $post = Post::factory()->create(['user_id' => $this->user->id]);

        $response = $this->authenticatedJson('DELETE',
"/api/posts/{$post->id}");

        $response->assertStatus(204);

        $this->assertDatabaseMissing('posts', [
            'id' => $post->id,
        ]);
    }

    public function test_user_cannot_access_other_users_posts()
    {
        $otherUser = User::factory()->create();
        $post = Post::factory()->create(['user_id' => $otherUser->id]);

        // Try to view
        $response = $this->authenticatedJson('GET', "/api/posts/{$post-
>id}");
        $response->assertStatus(403);

        // Try to update
        $response = $this->authenticatedJson('PUT', "/api/posts/{$post-
>id}", [
            'title' => 'Unauthorized Update',
        ]);
        $response->assertStatus(403);

        // Try to delete
        $response = $this->authenticatedJson('DELETE',
"/api/posts/{$post->id}");
        $response->assertStatus(403);

        // Verify no changes were made
```

```
        $this->assertDatabaseHas('posts', [
            'id' => $post->id,
            'title' => $post->title,
        ]);
    }

    public function test_validation_rules_for_post_operations()
    {
        // Create validation
        $response = $this->authenticatedJson('POST', '/api/posts', []);
        $response->assertStatus(422)
            ->assertJsonValidationErrors(['title', 'content']);

        // Update validation
        $post = Post::factory()->create(['user_id' => $this->user->id]);
        $response = $this->authenticatedJson('PUT', "/api/posts/{$post-
>id}", []);
        $response->assertStatus(422)
            ->assertJsonValidationErrors(['title', 'content']);
    }
}
```

6. Testing Invalid Authentication

```
// tests/Feature/Auth/InvalidAuthenticationTest.php

namespace Tests\Feature\Auth;

use Tests\TestCase;

class InvalidAuthenticationTest extends TestCase
{
    public function
test_unauthenticated_users_cannot_access_protected_routes()
    {
        // Try to access protected route without token
        $response = $this->getJson('/api/me');
        $response->assertStatus(401);

        // Try with invalid token
        $response = $this->withHeaders([
            'Authorization' => 'Bearer invalid-token',
        ])->getJson('/api/me');
        $response->assertStatus(401);
    }

    public function test_invalid_token_format()
    {
        $response = $this->withHeaders([
            'Authorization' => 'InvalidFormat',
```

```
        ])->getJson('/api/me');
        $response->assertStatus(401);
    }
}
```

7. Running the Tests

Run your tests with:

```
php artisan test
```

Or run specific test classes:

```
php artisan test --filter RegistrationTest
php artisan test --filter PostCrudTest
```

8. Additional Testing Tips

1. **Test Coverage**: Add `@covers` annotations to document what each test covers
2. **Data Providers**: Use data providers for testing multiple input scenarios
3. **Database Transactions**: Consider using `DatabaseTransactions` instead of `RefreshDatabase` for faster tests
4. **Factories**: Create comprehensive factories for all your models
5. **HTTP Tests**: Use Laravel's HTTP test helpers for API testing

Example with data provider:

PROF

```
/**
 * @dataProvider invalidRegistrationDataProvider
 */
public function test_registration_validation($input, $errors)
{
    $response = $this->postJson('/api/register', $input);
    $response->assertStatus(422)
        ->assertJsonValidationErrors($errors);
}

public function invalidRegistrationDataProvider()
{
    return [
        'missing name' => [
            ['email' => 'test@test.com', 'password' => 'password',
'password_confirmation' => 'password'],
            ['name']
        ],
```

```
        'invalid email' => [
            ['name' => 'Test', 'email' => 'not-an-email', 'password' =>
'password', 'password_confirmation' => 'password'],
            ['email']
        ],
        'password mismatch' => [
            ['name' => 'Test', 'email' => 'test@test.com', 'password' =>
'password', 'password_confirmation' => 'different'],
            ['password']
        ],
    ];
}
```

These tests provide comprehensive coverage for CRUD operations with authentication, ensuring your API endpoints are secure and function as expected.