Team name: “`QWERTY`” LAB10

Members:

Danyil Tymchuk

Artem Surzhenko

Currently Executable Tests

This document outlines the tests that are currently executable and repeatable from our test case list. We've implemented Unit tests for our User classes (Create, Read, Update, Delete).

Unit Testing

We have successfully implemented the following unit tests:

1. UserCreateTest

- **testCreateUserWithValidData**: Tests if a user can be created with valid data
- **testIsUsernameExistReturnsTrue**: Tests if the system correctly identifies an existing username
- **testIsUsernameExistReturnsFalse**: Tests if the system correctly identifies a non-existing username

2. UserReadTest

- **testGetUserProfile**: Tests if the system can correctly retrieve a user profile
- **testGetUserId**: Tests if the system can retrieve a user ID by username
- **testIsUsernameExist**: Tests if the system can check if a username exists

3. UserUpdateTest

- **testUpdateUser**: Tests if a user's information can be updated
- **testUpdateUserPassword**: Tests if a user's password can be updated
- **testUpdateFollowersNewFollow**: Tests if a user can follow another user
- **testIsUsernameExist**: Tests if the system can check if a username exists

4. UserDeleteTest

- **testDeleteUser**: Tests if a user can be soft deleted
- **testActuallyDeleteUser**: Tests if a user can be permanently deleted when authenticated
- **testActuallyDeleteUserNotAuthenticated**: Tests if the system prevents deletion when not authenticated
- **testIsUsernameExist**: Tests if the system can check if a username exists

Test Implementation Approach

Our unit tests use PHPUnit's mocking capabilities to simulate database connections and queries rather than connecting to a real database. This approach makes the tests faster, more reliable, and independent of the actual database state.

Each test focuses on a specific functionality and tests both valid and invalid scenarios to ensure robustness.

Diary Entry: Implementation Modifications Needed

Based on our testing experience, we've identified the following parts of our main implementation that need to be modified and adjusted:

1. **Autoloader Configuration**: We need to properly set up an autoloader to handle class loading. Currently, the test system cannot find our classes due to namespace and path issues.
2. **Namespace Consistency**: We need to ensure consistent namespacing across all files. Some files use `Models\User` while others might be using different namespaces.
3. **Database Abstraction**: The current direct database queries make testing difficult. We should consider implementing a repository pattern or data access layer to make mocking easier.
4. **Error Handling**: Some methods lack proper error handling, leading to potential issues during testing. We need to implement consistent error handling across all classes.
5. **Method Visibility**: Some methods that should be protected are public, making the class interface less clear. We should review and adjust method visibility.
6. **Code Documentation**: While the existing code has good documentation, some newer methods lack proper documentation, making testing more challenging.
7. **Test Bootstrap**: We need to create a proper bootstrap file for our tests to ensure all dependencies are loaded correctly.
8. **Validation Logic**: Some validation logic is duplicated across classes. We should consider extracting this to a separate validation class.

Calculations for Basis Path Testing

For the `UserCreate::validate()` method:

- Control flow graph nodes:
    1. Entry → Check username validity → Check email validity → Check password validity → Return result
- Edges = 4, Nodes = 5
- Cyclomatic complexity = E - N + 2 = 4 - 5 + 2 = 1
- Independent paths to test:
    1. Valid data path (all validations pass)
    2. Invalid username path
    3. Invalid email path
    4. Invalid password path

Calculations for Equivalence Partition Testing

For username validation in `UserCreate::isValidUsername()`:

- **Valid partitions**:
    - Alphanumeric usernames between 3-20 characters with at least one letter
    - Usernames that don't exist in the database
    - Usernames that aren't reserved words
- **Invalid partitions**:
    - Empty usernames
    - Usernames shorter than 3 characters
    - Usernames longer than 20 characters
    - Usernames without letters
    - Usernames that already exist in the database
    - Usernames that are reserved words

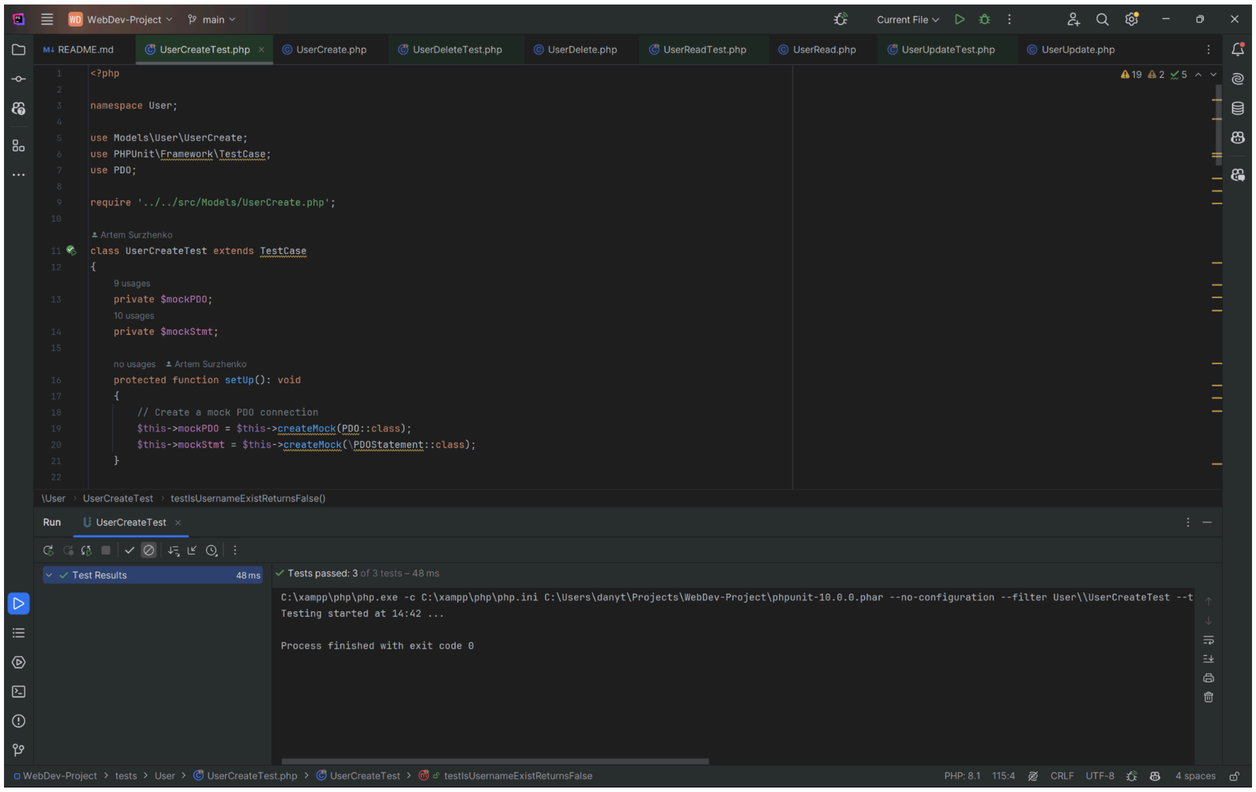
Validation Tests

Current validation rules implemented and tested:

- **Username Validation**:
    - Must be 3-20 characters
    - Alphanumeric characters and underscores only
    - Must have at least one letter
    - Cannot be a reserved word
    - Must be unique in the database
- **Email Validation**:
    - Cannot be empty
    - Must be a valid email format
- **Password Validation**:
    - Cannot be empty
    - Must be at least 6 characters long

Screenshots

UserCreateTest.php

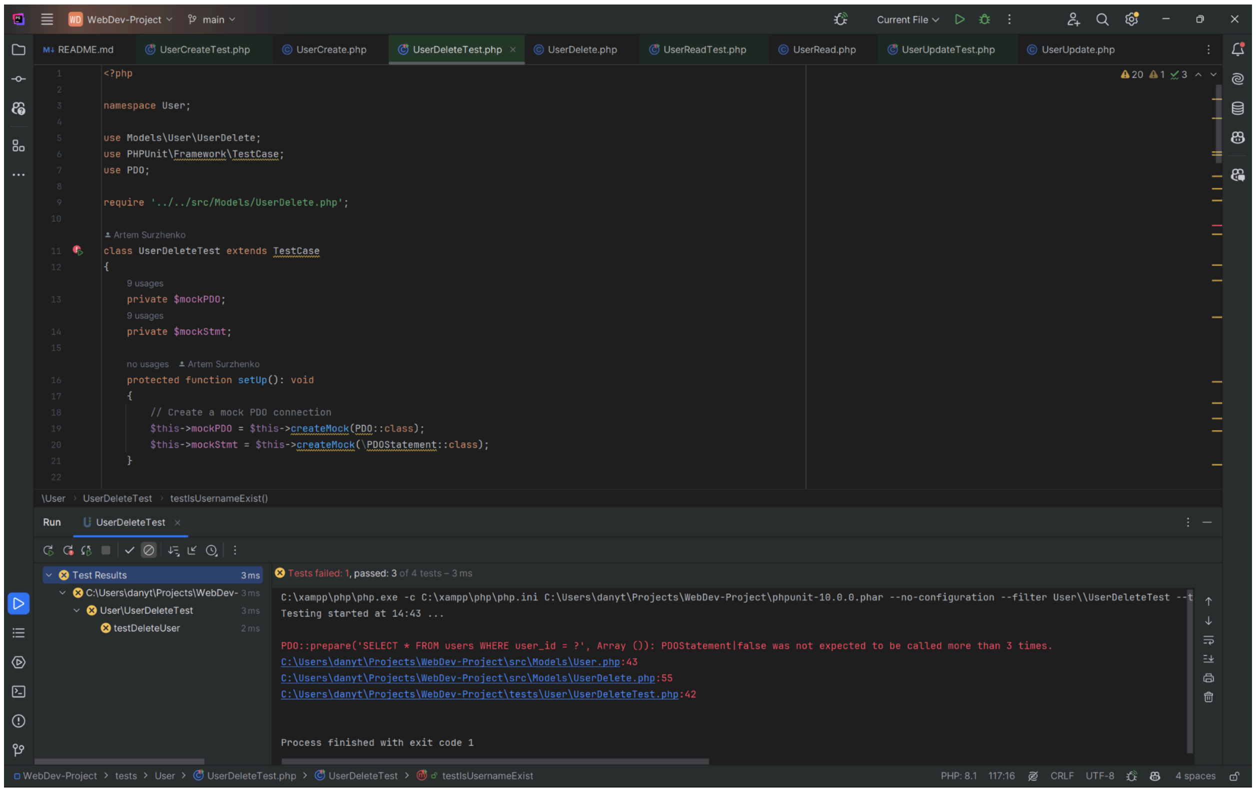


```php
<?php

namespace User;

use Models\User\UserCreate;
use PHPUnit\Framework\TestCase;
use PDO;

require '../../src/Models/UserCreate.php';

class UserCreateTest extends TestCase
{
    private $mockPDO;
    private $mockStmt;

    protected function setUp(): void
    {
        // Create a mock PDO connection
        $this->mockPDO = $this->createMock(PDO::class);
        $this->mockStmt = $this->createMock(\PDOStatement::class);
    }

```

Tests passed: 3 of 3 tests – 48 ms

```
C:\xampp\php\php.exe -c C:\xampp\php\php.ini C:\Users\danyt\Projects\WebDev-Project\phpunit-10.0.0.phar --no-configuration --filter User\\UserCreateTest --t
Testing started at 14:42 ...

Process finished with exit code 0
```

UserDeleteTest.php



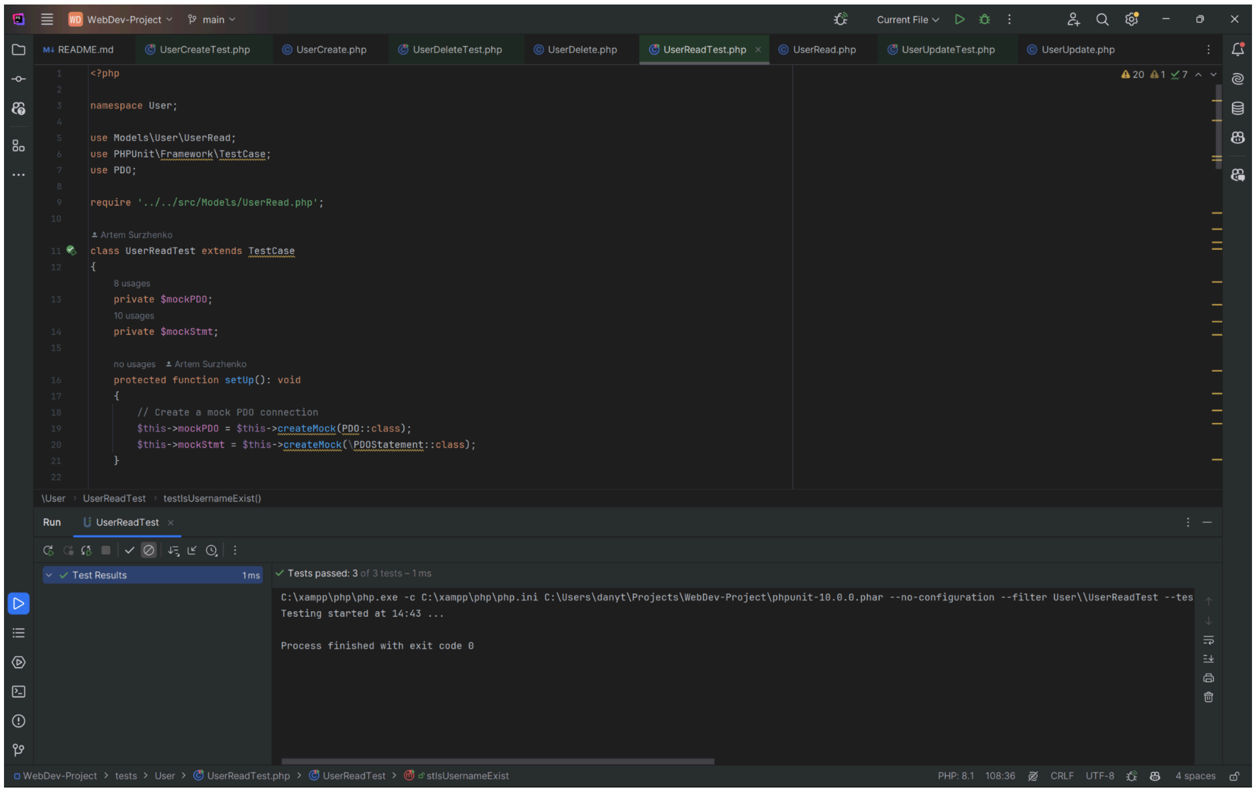
```php
<?php

namespace User;

use Models\User\UserDelete;
use PHPUnit\Framework\TestCase;
use PDO;

require '../../src/Models/UserDelete.php';

class UserDeleteTest extends TestCase
{
    private $mockPDO;
    private $mockStmt;

    protected function setUp(): void
    {
        // Create a mock PDO connection
        $this->mockPDO = $this->createMock(PDO::class);
        $this->mockStmt = $this->createMock(\PDOStatement::class);
    }

```

Tests failed: 1, passed: 3 of 4 tests – 3 ms

```
C:\xampp\php\php.exe -c C:\xampp\php\php.ini C:\Users\danyt\Projects\WebDev-Project\phpunit-10.0.0.phar --no-configuration --filter User\\UserDeleteTest --t
Testing started at 14:43 ...

PDO::prepare('SELECT * FROM users WHERE user_id = ?', Array ()): PDOStatement|false was not expected to be called more than 3 times.
C:\Users\danyt\Projects\WebDev-Project\src\Models\User.php:43
C:\Users\danyt\Projects\WebDev-Project\src\Models\UserDelete.php:55
C:\Users\danyt\Projects\WebDev-Project\tests\User\UserDeleteTest.php:42


Process finished with exit code 1
```

© Danyil Tymchuk & Artem Suzhenko

UserReadTest.php

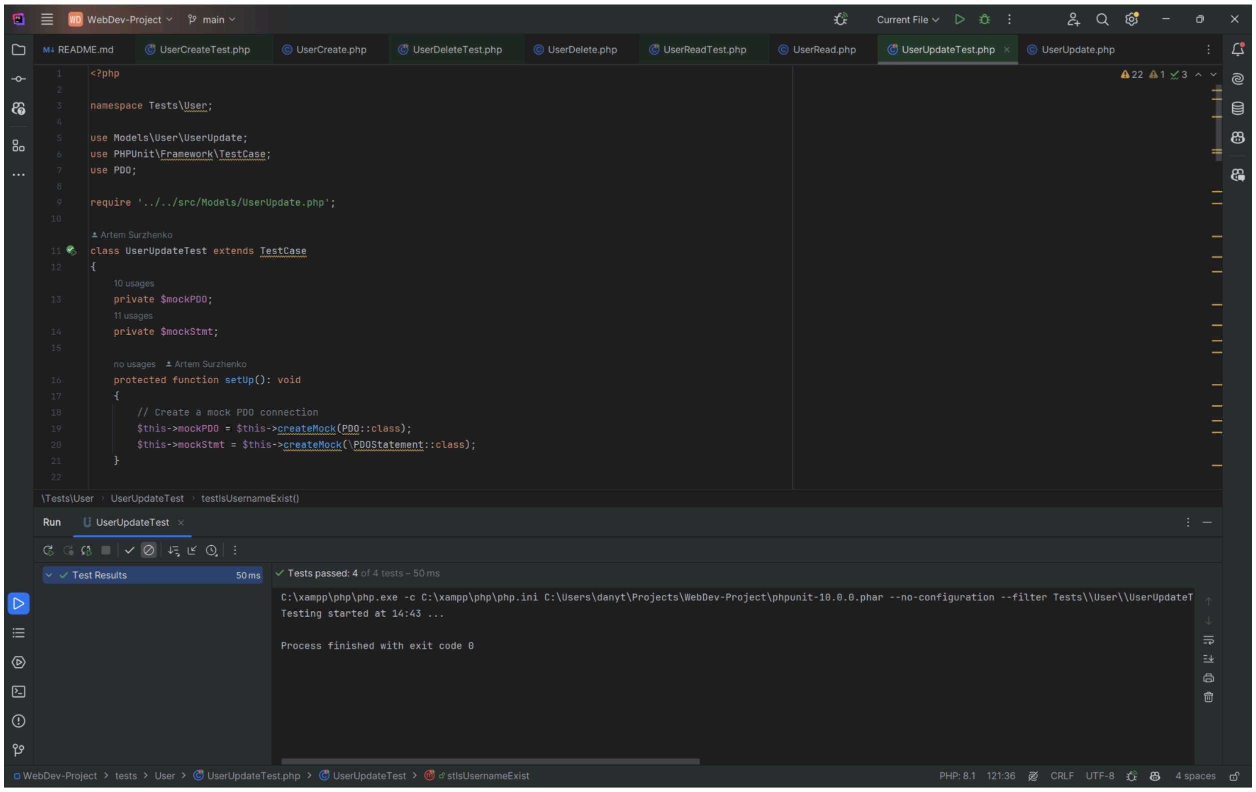


```php
<?php

namespace User;

use Models\User\UserRead;
use PHPUnit\Framework\TestCase;
use PDO;

require '../../src/Models/UserRead.php';

class UserReadTest extends TestCase
{
    private $mockPDO;
    private $mockStmt;

    protected function setUp(): void
    {
        // Create a mock PDO connection
        $this->mockPDO = $this->createMock(PDO::class);
        $this->mockStmt = $this->createMock(\PDOStatement::class);
    }

```

Tests passed: 3 of 3 tests – 1 ms

```
C:\xampp\php\php.exe -c C:\xampp\php\php.ini C:\Users\danyt\Projects\WebDev-Project\phpunit-10.0.0.phar --no-configuration --filter User\\UserReadTest --tes
Testing started at 14:43 ...

Process finished with exit code 0
```

UserUpdate.php



```php
<?php

namespace Tests\User;

use Models\User\UserUpdate;
use PHPUnit\Framework\TestCase;
use PDO;

require '../../src/Models/UserUpdate.php';

class UserUpdateTest extends TestCase
{
    private $mockPDO;
    private $mockStmt;

    protected function setUp(): void
    {
        // Create a mock PDO connection
        $this->mockPDO = $this->createMock(PDO::class);
        $this->mockStmt = $this->createMock(\PDOStatement::class);
    }

```

Tests passed: 4 of 4 tests – 50 ms

```
C:\xampp\php\php.exe -c C:\xampp\php\php.ini C:\Users\danyt\Projects\WebDev-Project\phpunit-10.0.0.phar --no-configuration --filter Tests\\User\\UserUpdateT
Testing started at 14:43 ...

Process finished with exit code 0
```

Conclusion

Our current test implementation focuses on Unit testing with PHPUnit. While we have successfully implemented tests for our User classes, we still need to address several issues to improve our testing framework. The most pressing issues are the autoloader configuration and namespace consistency.

Going forward, we plan to implement UI Testing, Requirements Testing, and expand our validation tests to cover more edge cases.

Source Code:

GitHub Repository: https://github.com/DanyilT/WebDev-Project

This Version on github: https://github.com/DanyilT/WebDev-Project/tree/f9df7eda41336416e9afba434b47677e9de0c3bc