
**Software Engineering and Testing. BSC Year 2, 2024/2025**
**(Assignment 4 - 25%, Demo -5%)**

# Assessment 4: Software Testing

**Submitted by: Danyil Tymchuk (B00167321) &**

**Artem Surzhenko (B00163362)**

**17/04/2025**

## Declaration

I herby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ordinary Degree in Computing in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated.

Author: Danyil Tymchuk Dated: 17/04/2025

Author: Artem Surzhenko Dated: 17/04/2025

# Overview

This document presents formal software tests performed on my PHP-based project as part of Assignment 4. All files are submitted in .php or .pdf as required.

# Tests

## (a) Unit Testing

https://github.com/DanyilT/WebDev-Project#unit-testing

- **Models/User/**
  - *UserTest*
    - *Test Cases*
    - *Tested Class*
  - *UserCreateTest*
    - *Test Cases*
    - *Tested Class*
  - *UserDeleteTest*
    - *Test Cases*
    - *Tested Class*
  - *UserReadTest*
    - *Test Cases*
    - *Tested Class*
  - *UserUpdateTest*
    - *Test Cases*
    - *Tested Class*

## (b) User Interface Testing

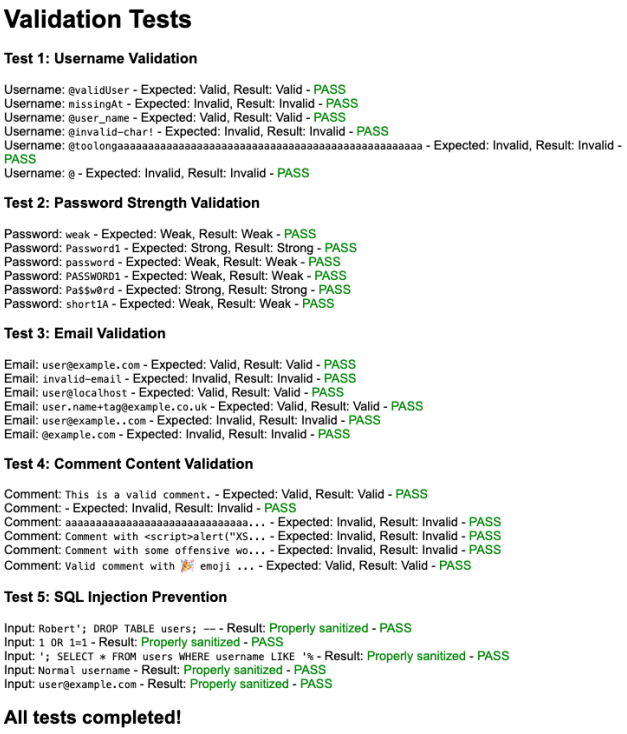
UI Testing Documentation.pdf

# (c) Validation Testing

https://github.com/DanyilT/WebDev-Project#validation-testing

*5-Validation Tests:* validation_tests.php

**Demo** (validation_tests.php):



## Validation Tests

**Test 1: Username Validation**

Username: `@validUser` - Expected: Valid, Result: Valid - PASS
Username: `missingAt` - Expected: Invalid, Result: Invalid - PASS
Username: `@user_name` - Expected: Valid, Result: Valid - PASS
Username: `@invalid-char!` - Expected: Invalid, Result: Invalid - PASS
Username: `@toolongaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa` - Expected: Invalid, Result: Invalid - PASS
Username: `@` - Expected: Invalid, Result: Invalid - PASS

**Test 2: Password Strength Validation**

Password: `weak` - Expected: Weak, Result: Weak - PASS
Password: `Password1` - Expected: Strong, Result: Strong - PASS
Password: `password` - Expected: Weak, Result: Weak - PASS
Password: `PASSWORD1` - Expected: Weak, Result: Weak - PASS
Password: `Pa$$w0rd` - Expected: Strong, Result: Strong - PASS
Password: `short1A` - Expected: Weak, Result: Weak - PASS

**Test 3: Email Validation**

Email: `user@example.com` - Expected: Valid, Result: Valid - PASS
Email: `invalid-email` - Expected: Invalid, Result: Invalid - PASS
Email: `user@localhost` - Expected: Valid, Result: Valid - PASS
Email: `user.name+tag@example.co.uk` - Expected: Valid, Result: Valid - PASS
Email: `user@example..com` - Expected: Invalid, Result: Invalid - PASS
Email: `@example.com` - Expected: Invalid, Result: Invalid - PASS

**Test 4: Comment Content Validation**

Comment: `This is a valid comment.` - Expected: Valid, Result: Valid - PASS
Comment: - Expected: Invalid, Result: Invalid - PASS
Comment: `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...` - Expected: Invalid, Result: Invalid - PASS
Comment: `Comment with <script>alert("XS...` - Expected: Invalid, Result: Invalid - PASS
Comment: `Comment with some offensive wo...` - Expected: Invalid, Result: Invalid - PASS
Comment: `Valid comment with 🎉 emoji ...` - Expected: Valid, Result: Valid - PASS

**Test 5: SQL Injection Prevention**

Input: `Robert'; DROP TABLE users; --` - Result: Properly sanitized - PASS
Input: `1 OR 1=1` - Result: Properly sanitized - PASS
Input: `'; SELECT * FROM users WHERE username LIKE '%` - Result: Properly sanitized - PASS
Input: `Normal username` - Result: Properly sanitized - PASS
Input: `user@example.com` - Result: Properly sanitized - PASS

**All tests completed!**

# Demo Section

## (d) Requirements Testing

- Unit Testing
    - Classes to test: `User`, `UserTest`, etc.
    - Objects to update: any logic classes impacted by new requirements.
    - Strategy: PHPUnit automation to cover methods like `getUsername()`, `getEmail()`, etc.

☑ UserCreateTest
    ☑ **testCreateUserWithValidData**: Tests if a user can be created with valid data
☑ 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
☑ UserUpdateTest
    ☑ **testUpdateUser**: Tests if a user's information can be updated
☑ UserDeleteTest
    ☑ **testDeleteUser**: Tests if a user can be soft deleted

# (e) Equivalence Partition

## 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

*Tests:* validation_tests.php
*Tests:* AuthControllerTest.php

**Demo** (*UserCreate.php*):

```
85
        Check if username is not empty, follows a valid pattern, and is not already taken or a reserved
        word

        Returns: bool

        Throws: Exception

        Package: Models\User
92      public function isValidUsername(): bool {
93          // Check if username is not empty
94          if (empty($this->username)) {
95              throw new Exception( message: "Username cannot be empty.");
96          }
97          // Allow only alphanumeric characters, numbers and underscores, 3 to 20 characters. At least one letter
98          if (!preg_match( pattern: '/^[a-zA-Z0-9_]{3,20}$/', $this->username) && !preg_match( pattern: '/[a-zA-Z]/', $this->username)) {
99              throw new Exception( message: "Username must be 3 to 20 characters long and can only contain letters, numbers, and underscores.");
100         }
101         // Check if username already exists
102         if ($this->isUsernameExist($this->username, $this->getConnection())) {
103             throw new Exception( message: "Username already exists.");
104         }
105         // Check if username is not a reserved word
106         $reservedWords = ['qwerty', 'dany', 'admin', 'root', 'user', 'test'];
107         if (in_array($this->username, $reservedWords)) {
108             throw new Exception( message: "Username is a reserved word.");
109         }
110         return true;
111     }
112
```

# (f) Basis Path Testing

## 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
- Cyclomaticcomplexity=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

*Tests:* AuthControllerTest.php

**Demo** (*UserCreate.php*):

```
75
        Validates all required data
        Returns: bool
        Throws: Exception
        Package: Models\User
82      private function validate(): bool {
83          return $this->isValidUsername() && $this->isValidEmail() && $this->isValidPassword();
84      }
85
```

## GitHub repo: https://github.com/DanyilT/WebDev-Project

**Last version:** https://github.com/DanyilT/WebDev-Project/tree/76ffdf689f52beb1df1ed0c66198be9d4381f3d8