Conestoga College

Assignment 1

Lei Chen-(8945274)

Code Coverage and Quality Control

PROG8840- Winter 2025- Section 1

Kumar Halder

2025/02/22

I. The coverage report

Coverage report: 83%

Files | Functions | Classes

coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

File ▲	class	statements	missing	excluded	coverage
test.py	TestDatabase	42	0	0	100%
test.py	*(no class)*	16	1	0	94%
user.py	User	3	3	0	0%
user.py	Database	14	0	0	100%
user.py	UserService	13	13	0	0%
user.py	*(no class)*	15	0	0	100%
Total		**103**	**17**	**0**	**83%**

coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

Coverage report: 83%

Files | Functions | Classes

coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

File ▲	statements	missing	excluded	coverage
test.py	58	1	0	98%
user.py	45	16	0	64%
Total	**103**	**17**	**0**	**83%**

coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

Coverage report: 83%

Files | Functions | Classes

coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

File ▲	function	statements	missing	excluded	coverage
test.py	TestDatabase.setUp	6	0	0	100%
test.py	TestDatabase.tearDown	1	0	0	100%
test.py	TestDatabase.test_init_creates_table	1	0	0	100%
test.py	TestDatabase.test_insert_user	5	0	0	100%
test.py	TestDatabase.test_get_user_found	5	0	0	100%
test.py	TestDatabase.test_get_user_not_found	4	0	0	100%
test.py	TestDatabase.test_update_user_success	5	0	0	100%
test.py	TestDatabase.test_update_user_not_found	5	0	0	100%
test.py	TestDatabase.test_delete_user_success	5	0	0	100%
test.py	TestDatabase.test_delete_user_not_found	5	0	0	100%
test.py	*(no function)*	16	1	0	94%
user.py	User.__init__	3	3	0	0%
user.py	Database.__init__	3	0	0	100%
user.py	Database.insert_user	3	0	0	100%
user.py	Database.get_user	2	0	0	100%
user.py	Database.update_user	3	0	0	100%
user.py	Database.delete_user	3	0	0	100%
user.py	UserService.__init__	1	1	0	0%
user.py	UserService.create_user	2	2	0	0%
user.py	UserService.get_user	4	4	0	0%
user.py	UserService.update_user	3	3	0	0%
user.py	UserService.delete_user	3	3	0	0%
user.py	*(no function)*	15	0	0	100%
Total		**103**	**17**	**0**	**83%**

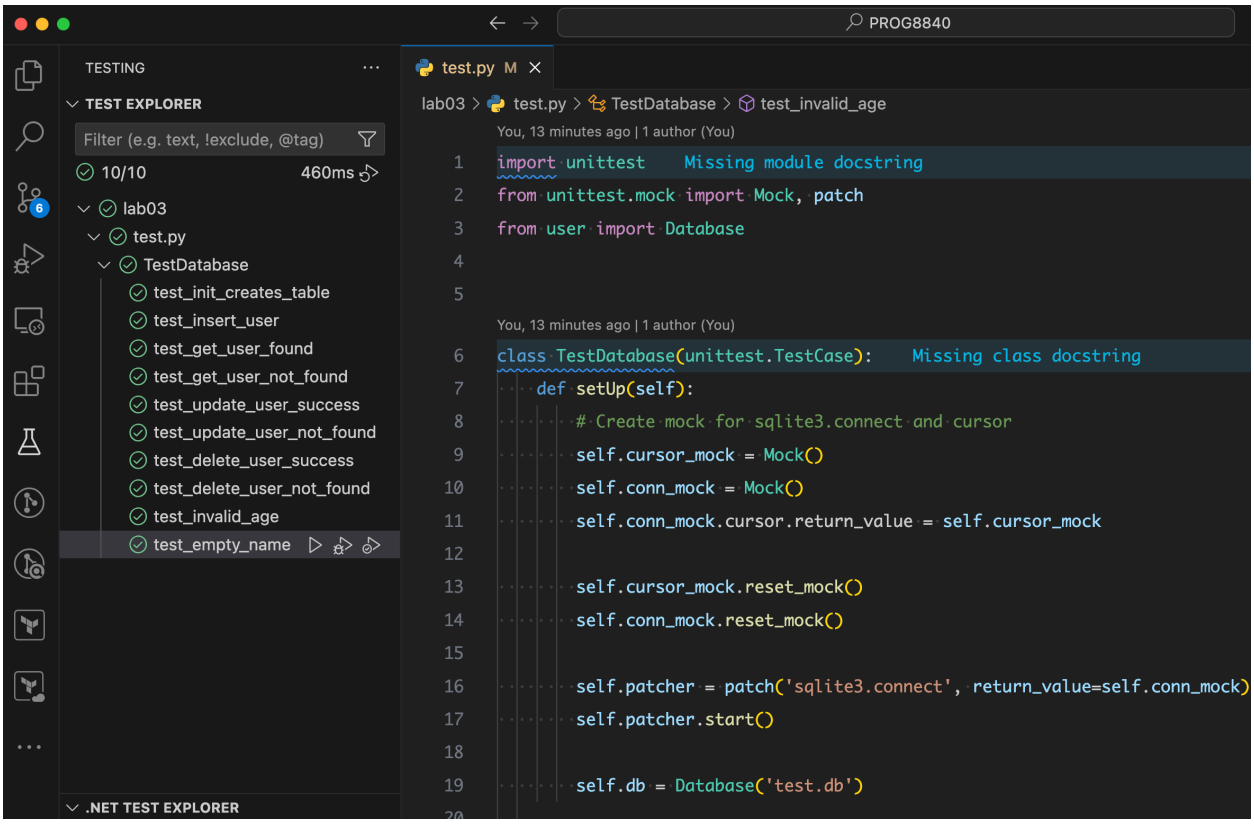
coverage.py v7.6.10, created at 2025-02-02 12:46 -0500

II. Test report

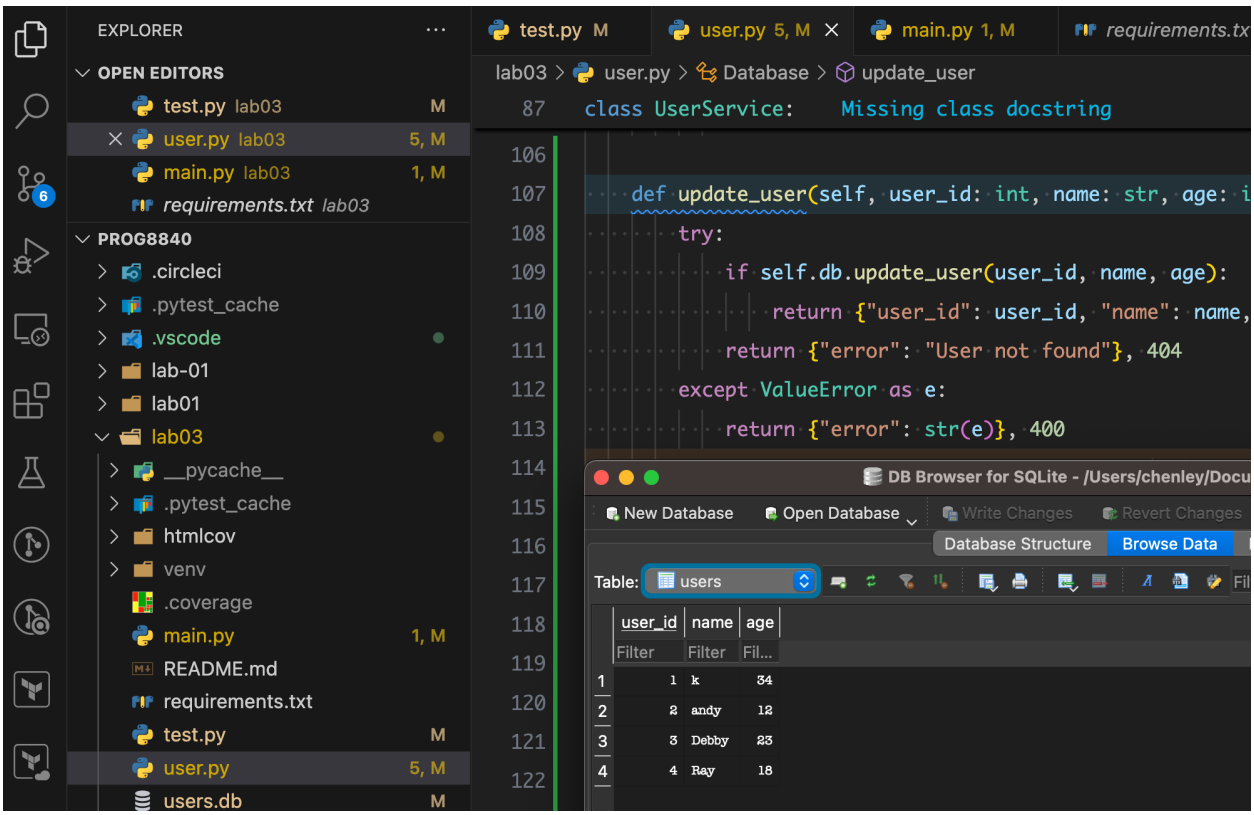
```
(venv) venv➜  lab03 git:(assignment-01) ✗ ll
total 88
-rw-r--r--@  1 chenley  staff   2.5K Feb  2 11:33 README.md
drwxr-xr-x@  7 chenley  staff   224B Feb  2 12:56 __pycache__
drwxr-xr-x@ 14 chenley  staff   448B Feb  2 12:35 htmlcov
-rw-r--r--@  1 chenley  staff   2.8K Feb  2 12:50 main.py
-rw-r--r--@  1 chenley  staff     9B Feb  2 11:33 requirements.txt
-rw-r--r--@  1 chenley  staff   4.9K Feb  2 12:56 test.py
-rw-r--r--   1 chenley  staff   4.9K Feb  2 12:55 test_database.py
-rw-r--r--@  1 chenley  staff   4.5K Feb  2 12:53 user.py
-rw-r--r--@  1 chenley  staff   8.0K Feb  2 11:33 users.db
drwxr-xr-x@  7 chenley  staff   224B Feb  2 12:06 venv
(venv) venv➜  lab03 git:(assignment-01) ✗ pytest ./test.py
============================================= test session starts =============================================
platform darwin -- Python 3.13.1, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/chenley/Documents/Life/Immi/Conestoga/ConestogaProgram/PROG8840 Code Coverage and Quality Control/lab/PROG8840
/lab03
collected 10 items

test.py ..........                                                                                      [100%]

============================================== 10 passed in 0.11s ==============================================
(venv) venv➜  lab03 git:(assignment-01) ✗
```



SQLite Database



Code improvements

1. Implemented proper error handling with custom exceptions

```
class DatabaseError(Exception):
    """Custom exception for database operations"""
    Pass
```

2. Added input validation for age and name

```
def checkParams(age, name):
    if not isinstance(age, int) or age < 0:
        raise ValueError("Age must be a positive integer")
    if not name.strip():
```

```
            raise ValueError("Name cannot be empty")
```

3. Created a context manager for database connections

```
@contextmanager
def get_connection(self):
    """Context manager for database connections"""
    conn = sqlite3.connect(self.db_name)
    try:
        yield conn
    except sqlite3.Error as e:
        raise DatabaseError(f"Database operation failed: {e}")
    finally:
        conn.close()
```

4. Improved user interface with numbered menu

```
while True:
    print("\n=== User Management System ===")
    print("1. Get user")
    print("2. Create user")
    print("3. Update user")
    print("4. Delete user")
    print("5. Exit")
```

5. Added docstrings for better documentation
6. Added database constraints (NOT NULL, CHECK)

```
def _init_db(self) -> None:
    """Initialize the database with required tables"""
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            'CREATE TABLE IF NOT EXISTS users (user_id INTEGER PRIMARY KEY, name TEXT NOT NULL, age INTEGER CHECK (age >= 0))'
        )
        conn.commit()
```

---

7. Implemented safe integer input handling

```
def get_integer_input(prompt: str) -> Optional[int]:
    """Safely get integer input from user"""
    try:
        return int(input(prompt))
    except ValueError:
        print("Please enter a valid number")
        return None
```

---

main.py

```
from user import Database, UserService, DatabaseError
import sys
```

```
from typing import Optional



def get_integer_input(prompt: str) -> Optional[int]:
    """Safely get integer input from user"""
    try:
        return int(input(prompt))
    except ValueError:
        print("Please enter a valid number")
        return None



def main():
    try:
        db = Database('users.db')
        user_service = UserService(db)

        while True:
            print("\n=== User Management System ===")
            print("1. Get user")
            print("2. Create user")
            print("3. Update user")
            print("4. Delete user")
            print("5. Exit")

            choice = get_integer_input("\nEnter your choice (1-5): ")
```

```
if not choice:
    continue

if choice == 5:
    break

try:
    if choice == 1:
        if user_id := get_integer_input("Enter user ID: "):
            user_data, status_code = user_service.get_user(user_id)
            print(f"Response: {
                user_data} (Status Code: {status_code})")

    elif choice == 2:
        name = input("Enter user name: ").strip()
        if age := get_integer_input("Enter user age: "):
            user_data, status_code = user_service.create_user(
                name, age)
            print(f"Response: {
                user_data} (Status Code: {status_code})")

    elif choice == 3:
        if user_id := get_integer_input("Enter user ID: "):
            name = input("Enter new name: ").strip()
            if age := get_integer_input("Enter new age: "):
                user_data, status_code = user_service.update_user(
```

```
                    user_id, name, age)
                print(f"Response: {
                    user_data} (Status Code: {status_code})")

            elif choice == 4:
                if user_id := get_integer_input("Enter user ID: "):
                    user_data, status_code = user_service.delete_user(
                        user_id)
                    print(f"Response: {
                        user_data} (Status Code: {status_code})")

        except DatabaseError as e:
            print("An error occurred while accessing the database" + str(e))

    except Exception as e:
        print("An unexpected error occurred: " + str(e))
        sys.exit(1)


if __name__ == '__main__':
    main()
```

user.py:

```
import sqlite3
from typing import Tuple, Optional, Dict, Any
```

```
from contextlib import contextmanager



class DatabaseError(Exception):
    """Custom exception for database operations"""
    pass



def checkParams(age, name):
    if not isinstance(age, int) or age < 0:
        raise ValueError("Age must be a positive integer")
    if not name.strip():
        raise ValueError("Name cannot be empty")



class User:
    def __init__(self, user_id: int, name: str, age: int):
        checkParams(age, name)

        self.user_id = user_id
        self.name = name
        self.age = age



class Database:
    def __init__(self, db_name: str):
```

```
        self.db_name = db_name
        self._init_db()

    @contextmanager
    def get_connection(self):
        """Context manager for database connections"""
        conn = sqlite3.connect(self.db_name)
        try:
            yield conn
        except sqlite3.Error as e:
            raise DatabaseError(f"Database operation failed: {e}")
        finally:
            conn.close()

    def _init_db(self) -> None:
        """Initialize the database with required tables"""
        with self.get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute(
                'CREATE TABLE IF NOT EXISTS users (user_id INTEGER PRIMARY KEY, name TEXT NOT
NULL, age INTEGER CHECK (age >= 0))'
            )
            conn.commit()

    def insert_user(self, name: str, age: int) -> int:
        """Insert a new user and return their ID"""
```

```
    checkParams(age, name)

    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            'INSERT INTO users (name, age) VALUES (?, ?)', (name, age))
        conn.commit()
        return cursor.lastrowid

def get_user(self, user_id: int) -> Optional[Tuple[int, str, int]]:
    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM users WHERE user_id = ?', (user_id,))
        return cursor.fetchone()

def update_user(self, user_id: int, name: str, age: int) -> bool:
    checkParams(age, name)

    with self.get_connection() as conn:
        cursor = conn.cursor()
        cursor.execute(
            'UPDATE users SET name = ?, age = ? WHERE user_id = ?', (name, age, user_id))
        conn.commit()
        return cursor.rowcount > 0

def delete_user(self, user_id: int) -> bool:
```

```
        with self.get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute('DELETE FROM users WHERE user_id = ?', (user_id,))
            conn.commit()
            return cursor.rowcount > 0



class UserService:
    def __init__(self, db: Database):
        self.db = db

    def create_user(self, name: str, age: int) -> Tuple[Dict[str, Any], int]:
        try:
            user_id = self.db.insert_user(name, age)
            return {"user_id": user_id, "name": name, "age": age}, 201
        except ValueError as e:
            return {"error": str(e)}, 400
        except DatabaseError as e:
            return {"error": "Internal server error"}, 500

    def get_user(self, user_id: int) -> Tuple[Dict[str, Any], int]:
        user = self.db.get_user(user_id)
        if user:
            return {"user_id": user[0], "name": user[1], "age": user[2]}, 200
        else:
            return {"error": "User not found"}, 404
```

```
    def update_user(self, user_id: int, name: str, age: int) -> Tuple[Dict[str, Any], int]:
        try:
            if self.db.update_user(user_id, name, age):
                return {"user_id": user_id, "name": name, "age": age}, 200
            return {"error": "User not found"}, 404
        except ValueError as e:
            return {"error": str(e)}, 400
        except DatabaseError as e:
            return {"error": "Internal server error"}, 500

    def delete_user(self, user_id: int) -> Tuple[Dict[str, Any], int]:
        try:
            if self.db.delete_user(user_id):
                return {"message": "User deleted successfully"}, 200
            return {"error": "User not found"}, 404
        except DatabaseError as e:
            return {"error": "Internal server error"}, 500
```

test.py

```
import unittest
from unittest.mock import Mock, patch
from user import Database
```

```
class TestDatabase(unittest.TestCase):
    def setUp(self):
        # Create mock for sqlite3.connect and cursor
        self.cursor_mock = Mock()
        self.conn_mock = Mock()
        self.conn_mock.cursor.return_value = self.cursor_mock

        self.cursor_mock.reset_mock()
        self.conn_mock.reset_mock()

        self.patcher = patch('sqlite3.connect', return_value=self.conn_mock)
        self.patcher.start()

        self.db = Database('test.db')

    def reset_mock(self):
        self.cursor_mock.reset_mock()
        self.conn_mock.reset_mock()

    def tearDown(self):
        self.patcher.stop()

    def test_init_creates_table(self):
        self.cursor_mock.execute.assert_called_with(
            'CREATE TABLE IF NOT EXISTS users (user_id INTEGER PRIMARY KEY, name TEXT NOT
NULL, age INTEGER CHECK (age >= 0))'
```

```
    )

def test_insert_user(self):
    self.reset_mock()

    self.cursor_mock.lastrowid = 1

    result = self.db.insert_user("John", 30)

    self.cursor_mock.execute.assert_called_with(
        'INSERT INTO users (name, age) VALUES (?, ?)',
        ("John", 30)
    )
    self.conn_mock.commit.assert_called_once()
    self.assertEqual(result, 1)

def test_get_user_found(self):
    self.reset_mock()

    expected_user = (1, "John", 30)
    self.cursor_mock.fetchone.return_value = expected_user

    result = self.db.get_user(1)

    self.cursor_mock.execute.assert_called_with(
        'SELECT * FROM users WHERE user_id = ?',
```

```
        (1,)
    )
    self.assertEqual(result, expected_user)

def test_get_user_not_found(self):
    self.reset_mock()

    self.cursor_mock.fetchone.return_value = None

    result = self.db.get_user(1)

    self.cursor_mock.execute.assert_called_with(
        'SELECT * FROM users WHERE user_id = ?',
        (1,)
    )
    self.assertIsNone(result)

def test_update_user_success(self):
    self.reset_mock()

    self.cursor_mock.rowcount = 1

    result = self.db.update_user(1, "John Updated", 31)

    self.cursor_mock.execute.assert_called_with(
        'UPDATE users SET name = ?, age = ? WHERE user_id = ?',
```

```
        ("John Updated", 31, 1)
    )
    self.conn_mock.commit.assert_called_once()
    self.assertTrue(result)

def test_update_user_not_found(self):
    self.reset_mock()

    self.cursor_mock.rowcount = 0

    result = self.db.update_user(1, "John Updated", 31)

    self.cursor_mock.execute.assert_called_with(
        'UPDATE users SET name = ?, age = ? WHERE user_id = ?',
        ("John Updated", 31, 1)
    )
    self.conn_mock.commit.assert_called_once()
    self.assertFalse(result)

def test_delete_user_success(self):
    self.reset_mock()

    self.cursor_mock.rowcount = 1

    result = self.db.delete_user(1)
```

```
    self.cursor_mock.execute.assert_called_with(
        'DELETE FROM users WHERE user_id = ?',
        (1,)
    )
    self.conn_mock.commit.assert_called_once()
    self.assertTrue(result)

def test_delete_user_not_found(self):
    self.reset_mock()

    self.cursor_mock.rowcount = 0

    result = self.db.delete_user(1)

    self.cursor_mock.execute.assert_called_with(
        'DELETE FROM users WHERE user_id = ?',
        (1,)
    )
    self.conn_mock.commit.assert_called_once()
    self.assertFalse(result)

def test_invalid_age(self):
    with self.assertRaises(ValueError):
        self.db.insert_user("John", -1)

def test_empty_name(self):
```

```
        with self.assertRaises(ValueError):
            self.db.insert_user("", 30)



if __name__ == '__main__':
    unittest.main()
```