SecureMed Test Suite Documentation

What is a Test Suite?

A test suite is a collection of automated tests that verify your code works correctly. Instead of manually testing every feature, you run one command and it checks everything for you.

Think of it like a checklist that automatically verifies:

- □ Does encryption work?
- □ Are passwords hashed correctly?
- □ Can users log in?
- □ Are we protected from SQL injection?

Running the Tests

Quick Start

```
# Make sure you're in the project directory
cd Project_dev1

# Run all tests
python3 test_webapp.py
```

Expected Output

When you run the tests, you should see something like:

```
============================================================
 SecureMed Test Suite
============================================================

test_ssn_encryption (__main__.TestEncryption)
Make sure SSN gets encrypted and decrypted correctly ...  SSN encryption/decryption works!
ok

test_password_hashing (__main__.TestPasswordHashing)
Passwords should be hashed with SHA-256 ...  Password hashing works!
ok

... (more tests) ...

----------------------------------------------------------------------
Ran 20 tests in 0.523s

OK
```

If all tests pass, you'll see **OK** at the end. If any fail, you'll see **FAILED** with details.

What Does Each Test Do?

1. Encryption Tests (`TestEncryption`)

Purpose: Make sure patient data (SSNs, PHI) is encrypted properly

Tests:

- `test_ssn_encryption`: Encrypts an SSN, checks it's different, then decrypts to verify it matches original
- `test_multiple_encryptions_different`: Ensures encryption is secure (same data encrypted twice gives different results)

Why This Matters: If encryption breaks, patient SSNs would be stored in plain text - HIPAA violation!

---

2. Password Hashing Tests (`TestPasswordHashing`)

Purpose: Verify passwords are hashed securely

Tests:

- `test_password_hashing`: Checks password is hashed with SHA-256 (64 character hex string)
- `test_same_password_same_hash`: Same password should always produce same hash
- `test_different_passwords_different_hashes`: Different passwords should have different hashes

Why This Matters: If passwords aren't hashed, a database leak would expose everyone's passwords

---

3. Database Tests (`TestDatabaseOperations`)

Purpose: Test database operations work correctly

Tests:

- `test_user_insertion`: Can we add users to database?
- `test_patient_insertion_with_encryption`: Can we add patients with encrypted SSNs?
- `test_sql_injection_prevention`: Are we protected from SQL injection attacks?

Why This Matters:

- Ensures data is stored correctly
- Verifies we're using parameterized queries (no SQL injection)
- Tests integration between encryption and database

Security Test Explained:

```
# This is an SQL injection attempt
malicious_username = "admin' OR '1'='1"

# If we used unsafe queries like this:
# query = f"SELECT * FROM users WHERE username='{malicious_username}'"
# The attacker would get all users!

# But we use parameterized queries:
cursor.execute("SELECT * FROM users WHERE username=?", (malicious_username,))
# This treats it as literal text, so the attack fails □
```

---

4. Flask Route Tests (`TestFlaskRoutes`)

Purpose: Test web routes and API endpoints

Tests:

- `test_login_page_loads`: Does the login page show up?
- `test_login_with_valid_credentials`: Can users log in with correct password?
- `test_login_with_invalid_credentials`: Are wrong passwords rejected?
- `test_protected_route_without_login`: Can't access dashboard without logging in?
- `test_api_returns_json`: Do API endpoints return JSON format?

Why This Matters: Ensures authentication works and unauthorized users can't access protected pages

---

5. Password Validation Tests (`TestPasswordValidation`)

Purpose: Test password strength requirements

Tests:

- `test_password_length_requirement`: Password must be 8+ characters
- `test_password_complexity_requirements`: Must have uppercase, lowercase, number, special char
- `test_common_passwords_blocked`: Blocks "Password123!", "Admin123!", etc.

Why This Matters: Weak passwords are the #1 cause of breaches. These tests ensure our requirements are enforced.

Example:

```
"abc123" → ❌ Too short, no uppercase, no special char
"Password" → ❌ No number, no special char
"Password123!" → ❌ Too common (blocked)
"MyP@ssw0rd2025" → ✅ Passes all requirements
```

---

6. Compliance Scoring Tests (`TestComplianceScoring`)

Purpose: Verify training score calculation is accurate

Tests:

- `test_score_increases_on_correct_answer`: Score goes up correctly
- `test_max_score_is_100`: Maximum score is 100%
- `test_score_cannot_go_negative`: Score never goes below 0%

Why This Matters: Training scores affect compliance reporting. If calculation is wrong, we're lying to auditors!

Formula:

```
Score = (correct_answers / 9 total questions) × 100%

Examples:
3 correct / 9 = 33.33%
6 correct / 9 = 66.67%
9 correct / 9 = 100%
```

---

7. Audit Logging Tests (`TestAuditLogging`)

Purpose: Ensure all actions are logged for HIPAA compliance

Tests:

- `test_activity_logging`: Verify actions get written to audit log

Why This Matters: HIPAA requires complete audit trail. If logging breaks, we can't prove who accessed what.

Understanding Test Output

When Tests Pass

```
test_ssn_encryption ... □ SSN encryption/decryption works!
ok
```

- `ok` = test passed
- Green checkmark shows what was tested

When Tests Fail

```
test_password_hashing ... FAIL

======================================================================
FAIL: test_password_hashing (__main__.TestPasswordHashing)
----------------------------------------------------------------------
AssertionError: 'Admin123!' == 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855'

Password was not hashed!
```

This tells you:

- Which test failed
- What went wrong
- What values were expected vs. what you got

Test Coverage

We test these security-critical areas:

Area	Tests	Why Critical
Encryption	2	PHI protection (HIPAA §164.312)
Passwords	6	Prevent unauthorized access
SQL Injection	1	#1 OWASP vulnerability
Authentication	5	Access control (HIPAA §164.312(a)(2))
Compliance Scoring	3	Accurate reporting
Audit Logging	1	HIPAA §164.312(b) requirement

Total: 20 automated tests

Adding Your Own Tests

Want to add a new test? Here's the template:

```
def test_your_feature_name(self):
    """Brief description of what you're testing"""
    # Setup
    test_data = "something"

    # Execute
    result = some_function(test_data)

    # Assert (check result is correct)
    self.assertEqual(result, expected_value)
    print("□ Your feature works!")
```

Example - testing MRN generation:

```
def test_mrn_unique(self):
    """Test that MRN numbers are unique"""
    mrn1 = generate_unique_mrn()
    mrn2 = generate_unique_mrn()

    # MRNs should be different
    self.assertNotEqual(mrn1, mrn2)
    print("□ MRN generation is unique!")
```

Common Test Methods

Method	What It Does	Example
`assertEqual(a, b)`	Check a equals b	`assertEqual(2+2, 4)`
`assertNotEqual(a, b)`	Check a does NOT equal b	`assertNotEqual(encrypted, original)`
`assertTrue(x)`	Check x is True	`assertTrue(is_valid)`
`assertFalse(x)`	Check x is False	`assertFalse(is_weak_password)`
`assertIn(a, b)`	Check a is in b	`assertIn(b'Login', response.data)`
`assertIsNone(x)`	Check x is None	`assertIsNone(user)`
`assertIsNotNone(x)`	Check x is NOT None	`assertIsNotNone(result)`

Why Testing Matters for Capstone

1. **Proves Code Works**: "I tested it manually" vs. "Here are 20 automated tests that pass"
2. **Shows Professional Practice**: Real companies use test suites. This demonstrates you understand industry standards.
3. **Catches Regressions**: If you change code later, tests catch what you broke
4. **Satisfies Requirements**: Your capstone breakdown explicitly requires a "test suite" as a deliverable
5. **Impresses Professors**: Most student projects don't have automated tests. Yours does.

Troubleshooting

Error: "ModuleNotFoundError: No module named 'webapp'"

Solution: Make sure you're in the Project_dev1 directory when running tests

```
cd Project_dev1
python3 test_webapp.py
```

Error: "Can't connect to database"

Solution: Tests create temporary databases (test_securemed.db, test_audit.db). These are automatically deleted after tests run. Make sure you have write permissions in the directory.

Some Tests Fail

Common reasons:

1. Database schema changed (update test table creation)
2. Function names changed (update import statements)
3. Password requirements changed (update test expectations)

Read the error message - it tells you exactly what failed and why.

Running Individual Test Classes

Don't want to run all tests? Run specific ones:

```
# Just encryption tests
python3 -m unittest test_webapp.TestEncryption

# Just password tests
python3 -m unittest test_webapp.TestPasswordHashing

# Just one specific test
python3 -m unittest test_webapp.TestEncryption.test_ssn_encryption
```

Test Results for Documentation

When presenting your capstone, you can say:

> *"Our application includes 20 automated tests covering encryption, authentication, SQL injection prevention, password validation, compliance scoring, and audit logging. All tests pass, demonstrating the security and reliability of our HIPAA compliance platform."*

Then run: `python3 test_webapp.py` and show the green output!

Next Steps

1. **Run the tests**: `python3 test_webapp.py`
2. **Verify all pass**: You should see "OK" at the end
3. **Take a screenshot**: Use it in your presentation
4. **Add to README**: Mention you have automated tests

Last Updated: December 2025 For SecureMed Capstone Project