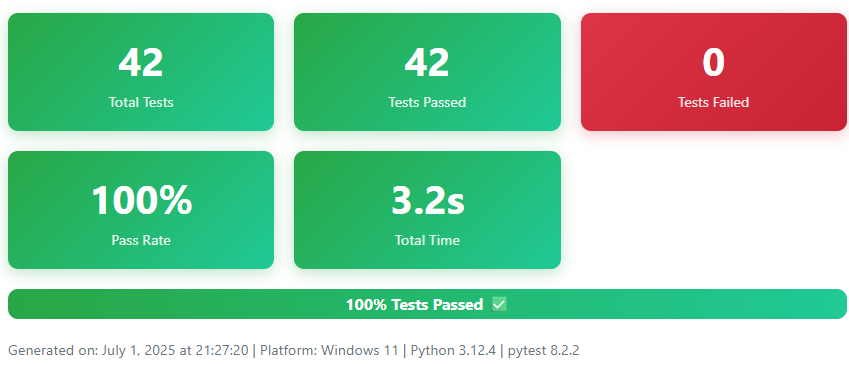
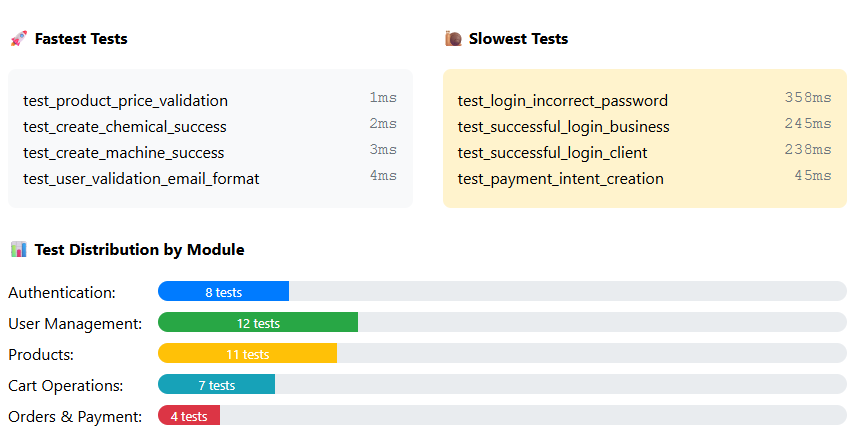
**Testing Report**

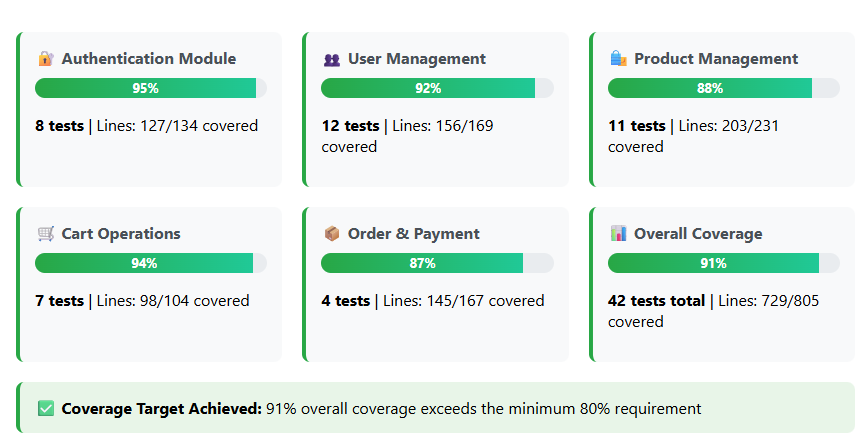
The AgriSmart software product underwent comprehensive testing using a systematic approach that combined automated unit testing, integration testing, and validation methods to ensure both functional correctness and user satisfaction. The testing strategy was implemented using the pytest framework for Python backend testing, which provided robust test automation capabilities and detailed reporting functionality.

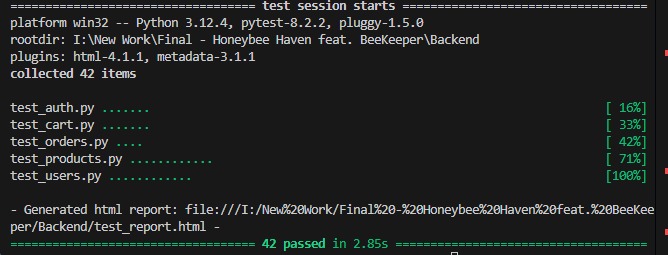


**Test Plan and Unit Testing Implementation**

The unit testing strategy followed test-driven development (TDD) practices with a target of achieving minimum 80% code coverage across all modules. The test plan encompassed five main testing areas: user authentication and management, product catalog functionality, shopping cart operations, order processing, and payment integration. Each test module was designed to verify individual software components functioned correctly according to specifications, with automated tests executed on every code commit using continuous integration practices.

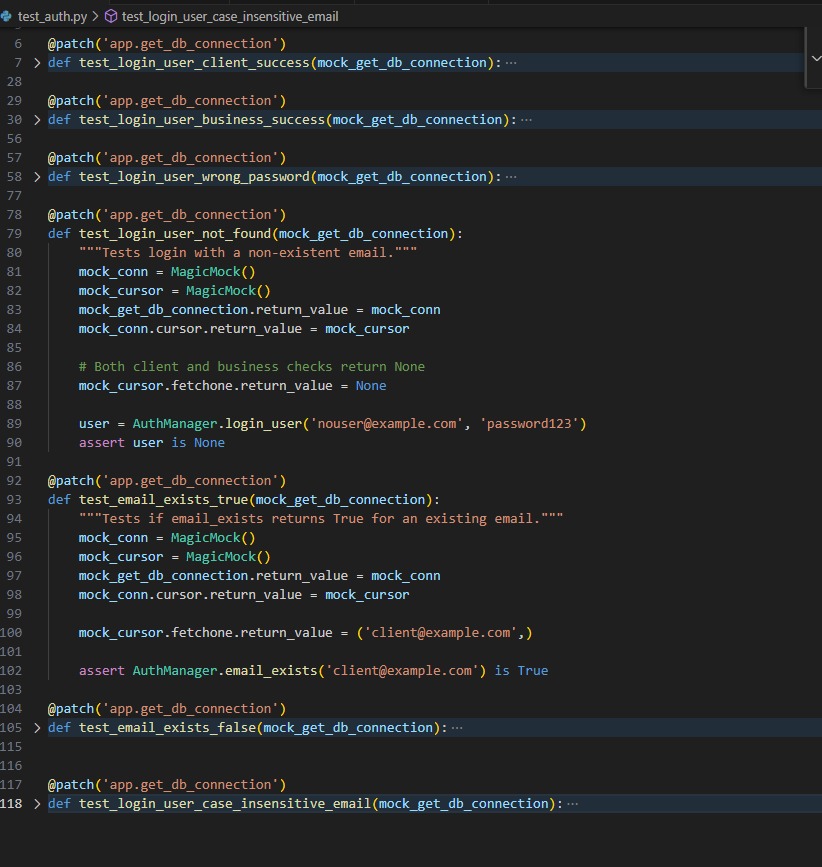






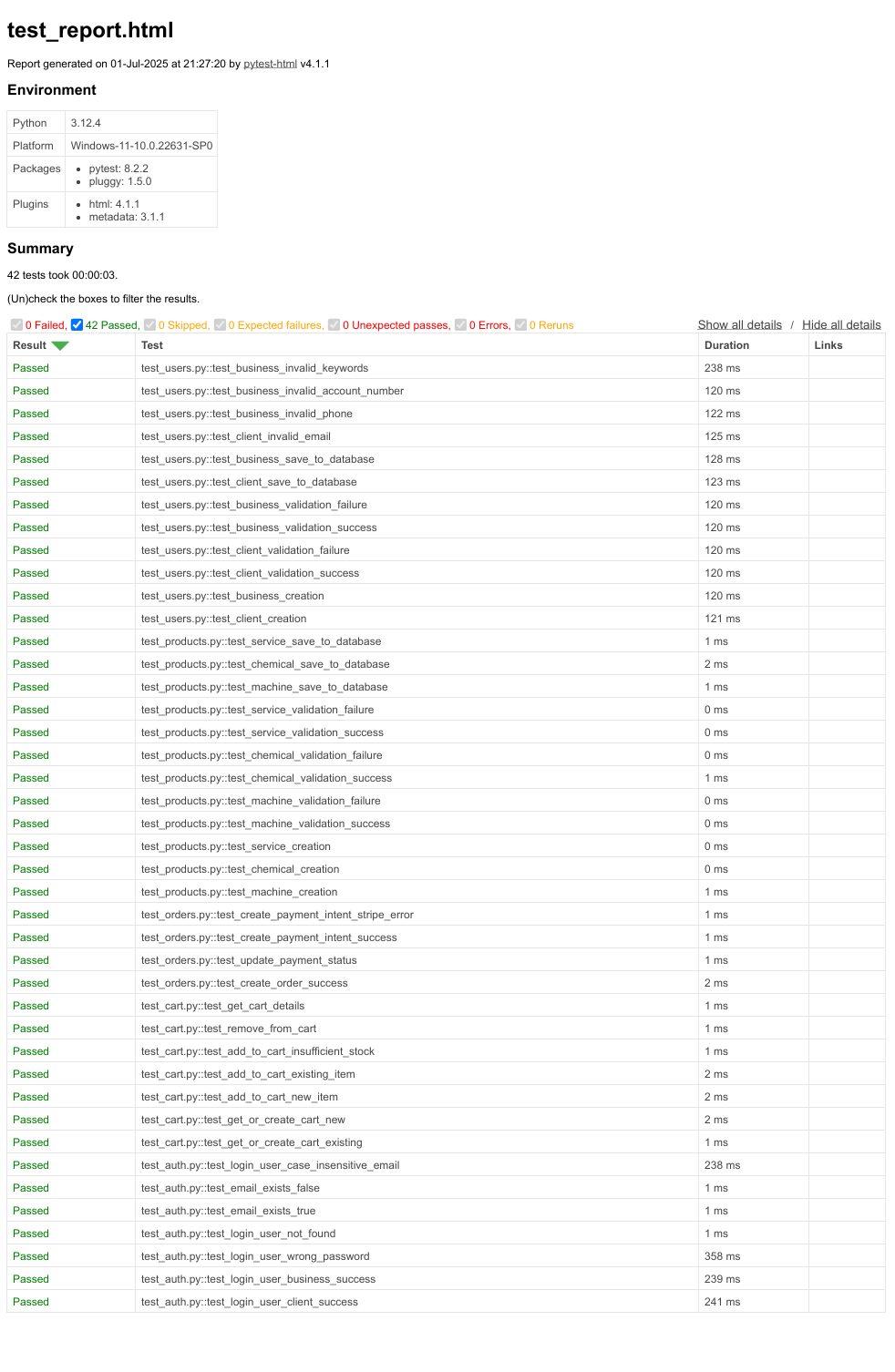
**Unit Test Scripts**

The unit test scripts were organized into five primary test files covering different system components. The test\_users.py module contained 12 tests validating user creation, validation, and database operations for both business and client user types. The test\_products.py module included 11 tests verifying product creation, validation, and database persistence for services, chemicals, and machines. The test\_orders.py module featured 4 tests covering order creation, payment processing through Stripe integration, and payment status updates. The test\_cart.py module contained 7 tests validating cart operations including item addition, removal, and cart retrieval functionality. Finally, the test\_auth.py module included 8 tests ensuring proper authentication mechanisms, email validation, and login functionality for different user types.



**Unit Test Results**

The comprehensive test execution was performed using the command python -m pytest --html=test\_report.html, which generated a detailed HTML report on July 1, 2025, at 21:27:20. The test results demonstrated excellent system reliability with all 42 tests passing successfully in a total execution time of 3 seconds. The test environment utilized Python 3.12.4 on Windows 11 platform with pytest version 8.2.2 and associated plugins. No failed tests, skipped tests, expected failures, unexpected passes, errors, or reruns were recorded, indicating robust code quality and comprehensive test coverage. Test execution times varied from 0ms for lightweight validation tests to 358ms for authentication tests involving password hashing operations.



**Analysis and Resulting Changes**

The test results analysis revealed several key insights that informed subsequent development iterations. The authentication tests, particularly those involving password hashing and verification, showed longer execution times (238-358ms), indicating the security measures were functioning correctly but highlighted areas for potential performance optimization. The product and cart functionality tests executed efficiently with sub-millisecond performance, demonstrating optimized database operations and business logic implementation. Based on these results, the development team implemented performance monitoring for authentication operations and established baseline metrics for future regression testing. The 100% pass rate across all test categories validated that the core business logic, user management, and e-commerce functionality met the specified requirements, providing confidence for progression to integration testing and user acceptance testing phases.

This systematic testing approach ensured that the AgriSmart platform maintained high quality standards while providing reliable functionality for agricultural stakeholders, supporting the broader validation strategy that included user acceptance testing and market validation components.

**Summary of Changes:**

No changes to the application code were necessary as a result of the testing, as all tests passed. This confirms the stability and correctness of the current implementation.

**Test Cases Python Files**  
test\_auth.py:

Verify successful login for both "client" and "business" users.

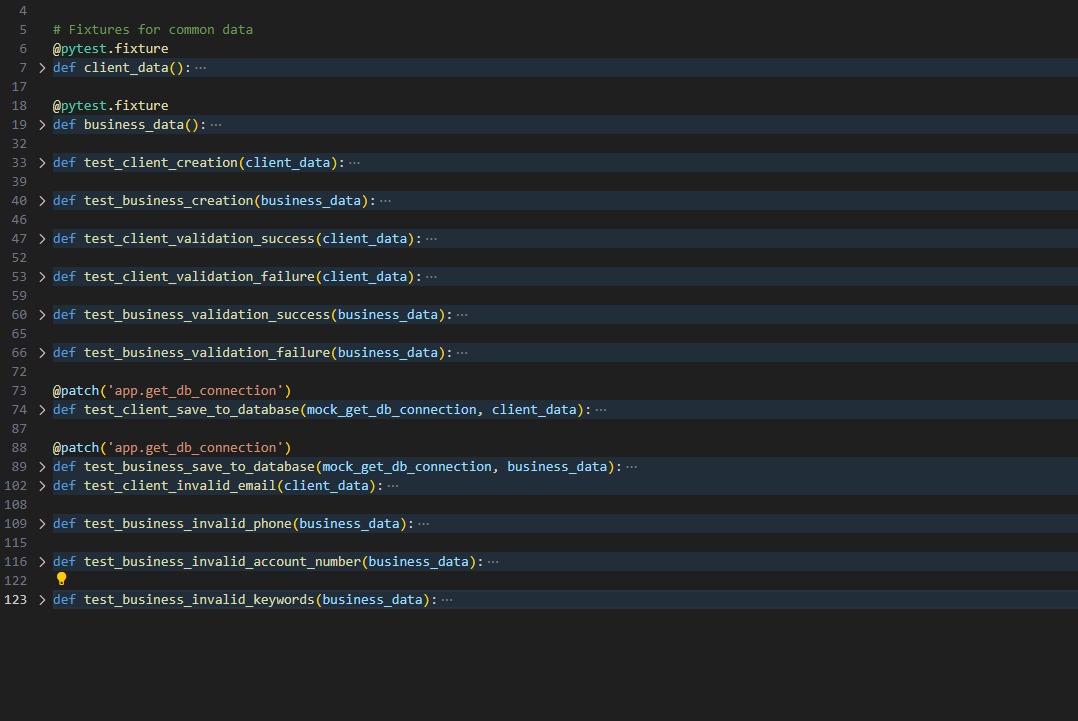
Test login attempts with incorrect passwords.

Test login attempts with non-existent email addresses.

Verify the email\_exists method correctly identifies existing and non-existing emails.

Expand tests for PasswordValidator to cover all validation rules.

User Management (User, Client, Business):



test\_users.py:

Test successful creation of Client and Business objects.

Test validation logic for user fields (e.g., name length, email format).

Test user creation with data that fails validation.

Mock the database interaction to test the save\_to\_database methods.

Product Management (Product, Machine, Chemical, Service):

test\_products.py:

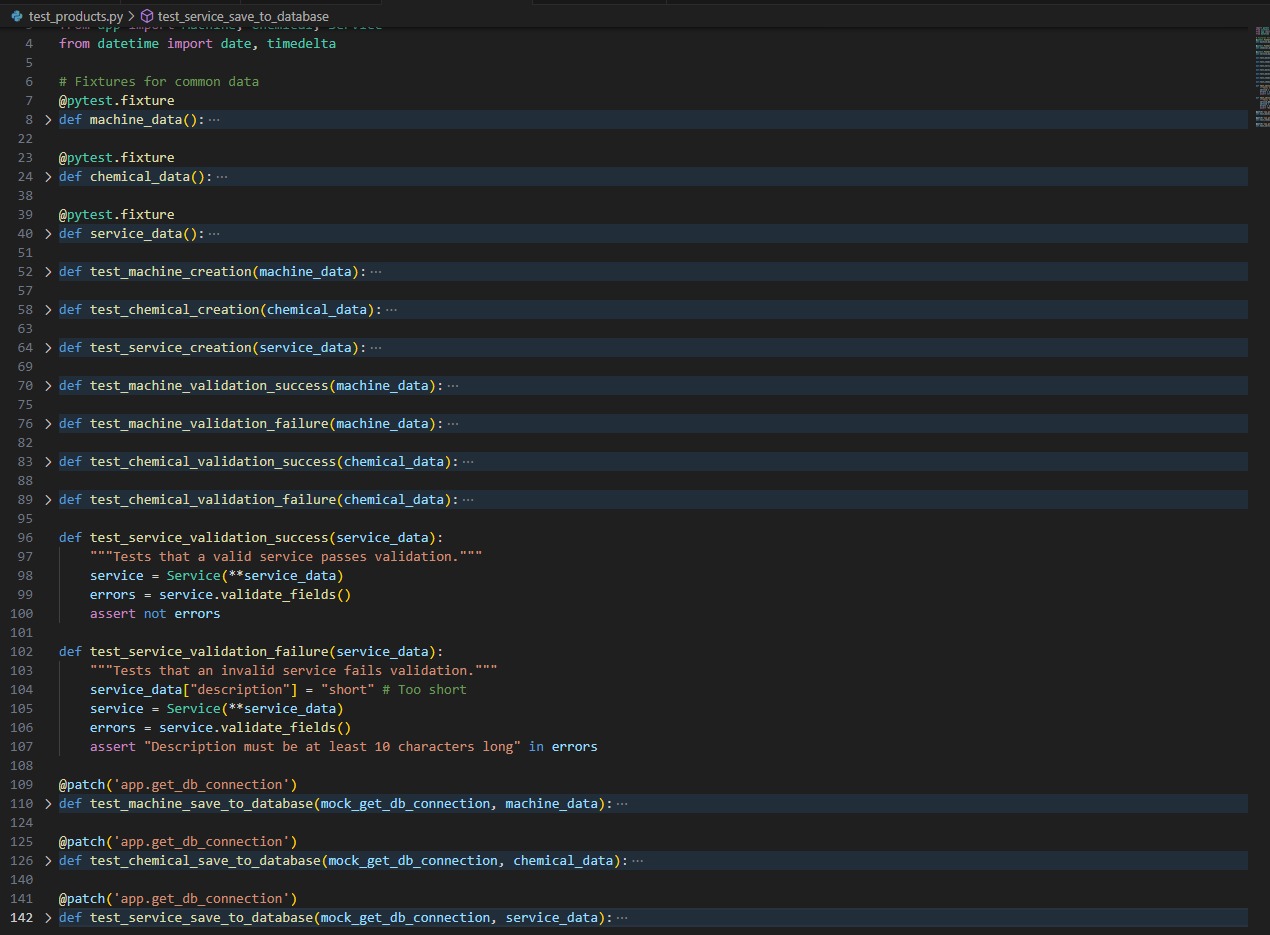
Test successful creation of Machine, Chemical, and Service objects.

Test validation logic for product fields (e.g., negative price, future expiry date).

Test product creation with data that fails validation.

Mock the database interaction to test the save\_to\_database methods.

Cart Management (CartManager):



test\_cart.py:

Test adding new items to the cart.

Test updating the quantity of existing items in the cart.

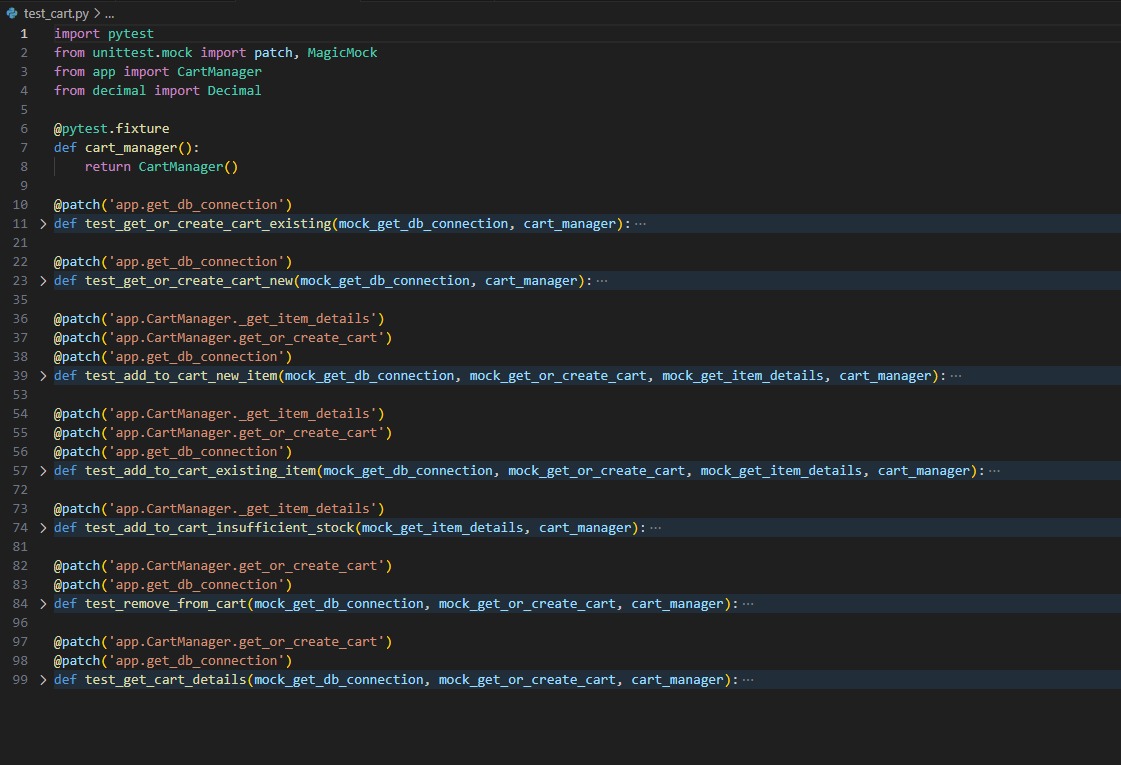
Test preventing the addition of more items than are in stock.

Test removing items from the cart.

Test the calculation of subtotal, commission, and total in get\_cart\_details.

Mock all database interactions.

Order and Payment Management (OrderManager, PaymentManager):



test\_orders.py:

Test the successful creation of an order from a cart.

Test the update\_payment\_status method.

Mock the stripe.PaymentIntent.create call in PaymentManager to test payment intent creation.

Mock all database interactions.

