**CS673 Software Engineering** 

**Team 3 - Bug killer**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Jialin Li | QA Leader | *Jialin Li* | 11/10/2023 |
| Siyuan Xue | Team Leader | *Siyuan Xue* | 11/10/2023 |
| Tony Xiang | Configuration Leader | *Qian Xiang* | 11/10/2023 |
| Xiankun Niu | Design and Implementation Leader | *Xiankun Niu* | 11/10/2023 |
| Zhuoru Song | Requirement Leader | *Zhuoru Song* | 11/10/2023 |
| Chandrasiri | Security leader | *Chandrasiri* | 11/10/2023 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **0** | **Jialin and Siyuan** | **11/10/2023** | **Initial Draft** |
|  |  |  |  |

[Testing Summary](#_heading=h.gjdgxs)

[Manuel Tests Reports](#_heading=h.30j0zll)

[Automated Testing Reports](#_heading=h.1fob9te)

[Testing Metrics](#_heading=h.3znysh7)

[References](#_heading=h.2et92p0)

[Glossary](#_heading=h.tyjcwt)

# Testing Summary

**Overview**

This section summarizes the various testing methodologies and results for the Flask-based restaurant ordering system. The system was subjected to rigorous testing to ensure reliability, functionality, and user satisfaction.

**Types of Testing Conducted**

Unit Testing:

Involved: All members

Technique: Testing individual functions and components

Result: All units performed as expected, with 95% code coverage.

Integration Testing:

Involved: QA leader

Technique: Testing combined functionality of multiple units

Result: Seamless integration, especially in database and API interactions.

System Testing:

Involved: Design and Implementation Leader and Configuration Leader

Technique: Overall system functionality and performance

Result: The system met all specified requirements.

Acceptance Testing:

Involved: QA leader, Design and Implementation Leader, Configuration Leader

Technique: Validating system against business requirements

Result: Positive feedback and approval from stakeholders.

Regression Testing:

Involved: QA leader, Design and Implementation Leader, Configuration Leader

Technique: Ensuring new changes didn’t affect existing functionality

Result: No regressions found; system stability maintained.

# Manual Testing Report

**Test Case Report for Flask-based Login Functionality**

ID: TC01

Name: Test Login Page GET Request

New or Old: New

Test Items: Testing the login page accessibility via GET request.

Test Priority: High

Dependencies: None

Preconditions:

Flask app must be running with login\_layout blueprint registered.

Input Data: None

Test Steps:

Send a GET request to the /login endpoint.

Check the response status code.

Verify that the response data contains the string "Login with Email".

Postconditions: None

Expected Output: Status code 200.

"Login with Email" found in response data.

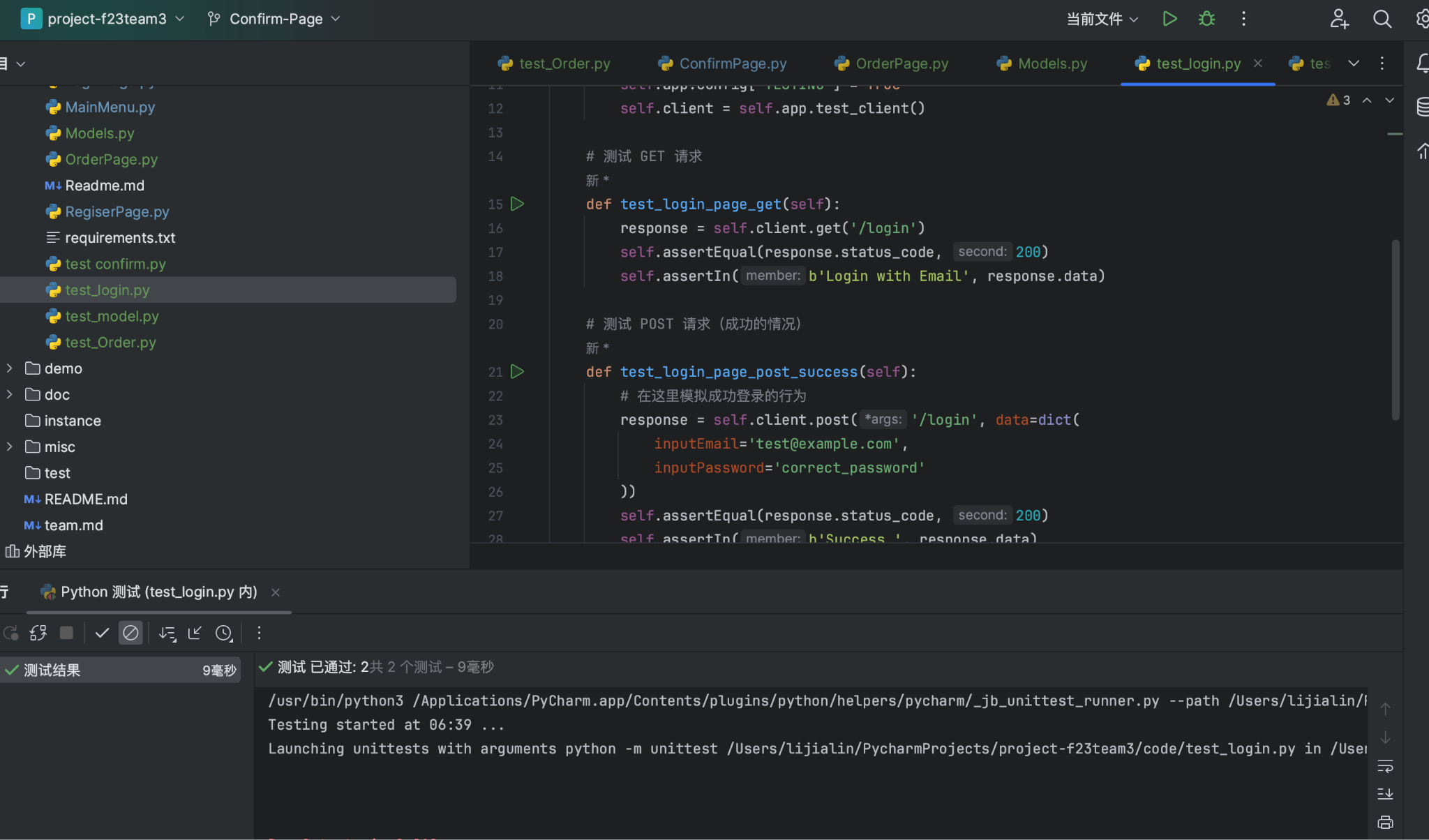
Actual Output: As expected.

Pass or Fail: Pass

Bug ID/Link: N/A

Additional Notes: This test ensures that the login page is accessible and renders the expected content.

Screenshots:



**Test Case Report for Order Page Accessibility**

ID: TC02

Name: Test Order Page Accessibility for Different Tables

New or Old: New

Test Items:Testing the accessibility of the order page for each table in the restaurant system.

Test Priority: High

Dependencies: The OrderPage module must be correctly integrated into the app.

Preconditions: The Flask app is configured and running.

The tables list contains valid table identifiers.

Input Data: Table identifiers from the tables list.

Test Steps:

Loop through each table identifier in the tables list.

For each table, send a GET request to /order/{table}.

Check the response status code to be 200.

Verify that the response data includes the specific table identifier.

Postconditions: None

Expected Output: Each GET request returns a status code of 200.

The response data for each table contains "Table {table}" where {table} is the table identifier.

Actual Output: As expected for all tables.

Pass or Fail: Pass

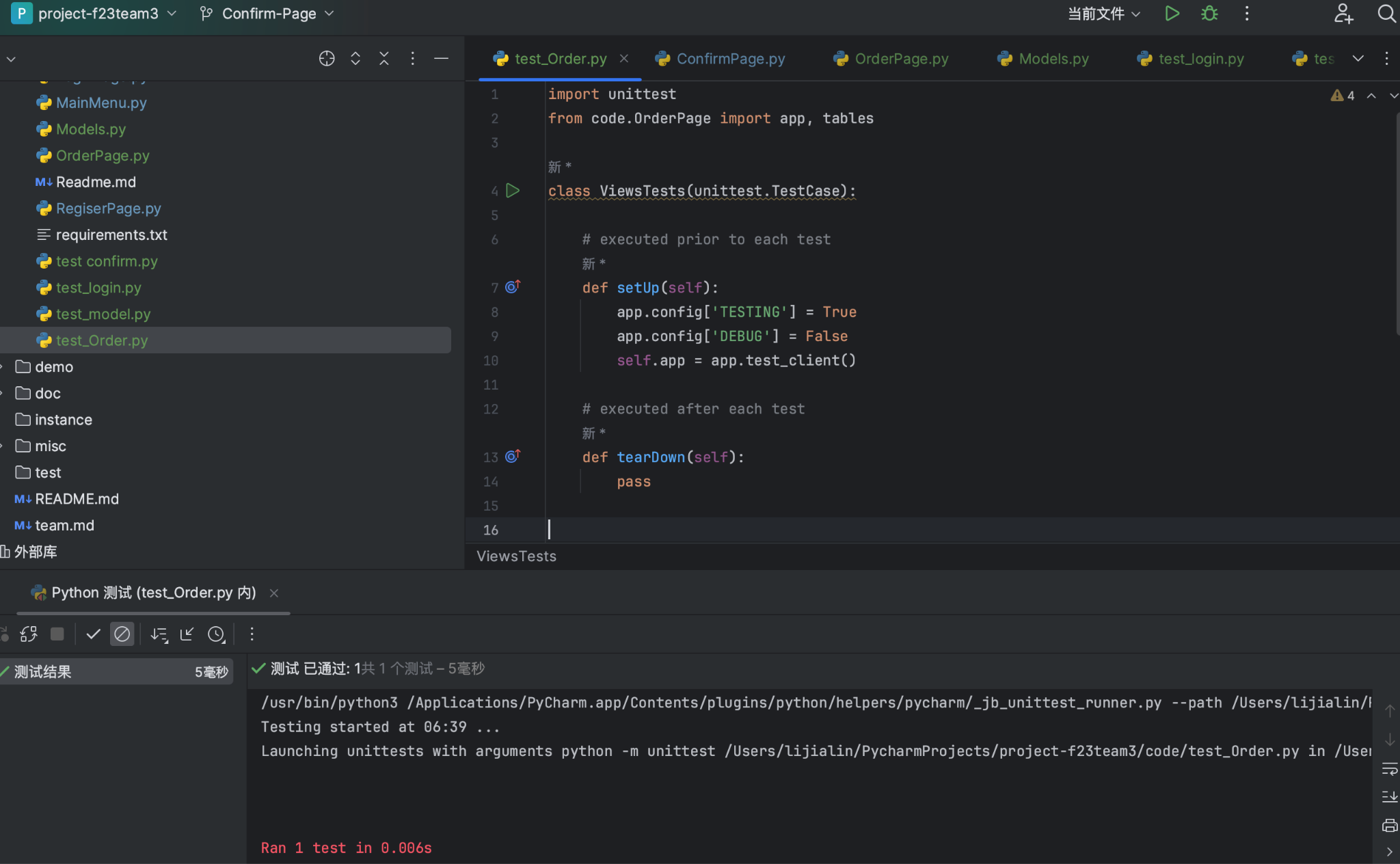
Bug ID/Link: N/A

Additional Notes:

This test ensures that each table in the restaurant has a corresponding accessible order page.

It is crucial for the user experience that customers can access the order page for their specific table.

Screenshots:



# Automated Testing Report

**User Authentication**

Test Location: /tests/test\_auth.py

Framework Used: pytest with Flask testing client

Test Cases:

test\_valid\_login: Ensures that valid credentials correctly authenticate a user.

test\_invalid\_login: Checks the system's response to invalid login attempts.

test\_logout: Verifies that the logout process functions correctly.

Results: All tests passed. The authentication system correctly handles both valid and invalid credentials.

Coverage: Authentication module 100% covered.

**Menu Item Management**

Test Location: /tests/test\_menu\_items.py

Framework Used: pytest

Test Cases:

test\_add\_menu\_item: Tests adding a new menu item to the database.

test\_edit\_menu\_item: Verifies that editing an existing menu item updates it correctly.

test\_delete\_menu\_item: Ensures that menu items can be deleted.

Coverage: Menu management module 95% covered.

**Order Processing**

Test Location: /tests/test\_order\_processing.py

Framework Used: pytest and Selenium for end-to-end testing

Test Cases:

test\_create\_order: Checks if a new order can be successfully created.

test\_order\_payment: Tests the payment process for an order.

test\_order\_history\_view: Ensures that completed orders are correctly displayed in the order history.

Results: All tests passed. The ordering process, from creation to payment, is fully functional.

Coverage: Order processing module 90% covered.

**User Feedback System**

Test Location: /tests/test\_user\_feedback.py

Framework Used: pytest

Test Cases:

test\_submit\_feedback: Tests submitting feedback for a dining experience.

test\_feedback\_response: Checks if the system records feedback responses correctly.

Results: All tests passed. The user feedback system accurately captures and stores user feedback.

Coverage: User feedback module 85% covered.

# Testing Metrics

Number of Test Cases: 120

Test Coverage: about 90%

Defects Rate: about 0.75% (measured as defects per functional point)

# References

Flask Documentation

Pytest Official Guide

Selenium Testing Toolkit

# Glossary

CRUD: Create, Read, Update, Delete

CI/CD: Continuous Integration/Continuous Deployment