

Contents

1 Semester Project: Testing “The New Cantina” Web Application 1

1.1 Table of Contents 1

1.2 Introduction 1

1.2.1 Team Members 1

1.2.2 Description of Tested Functionality 2

1.3 Test Case Management Template 2

1.4 Testing Methodology and Execution 4

1.4.1 Testing Methods & Tools 4

1.4.2 Test Results Summary 5

1.5 Conclusion 6

1.6 Annexes 6

1.7 Annex A: Automated Test Execution Details 6

1.7.1 Model & Unit Tests 7

1.7.2 API Tests 33

1.7.3 End-to-End (E2E) Tests 65

1.7.4 Scenario & Integration Tests 68

1.7.5 Undocumented Test Cases 72

1.8 Annex B: Manual Test Execution Details 74

1 Semester Project: Testing “The New Cantina” Web Application

Project:	The New Cantina - System Testing
Course:	Web Programming
Team:	Ishan Baichoo, Gabriel Aumasson-Leduc, Clément De Simon
Submission Date:	10.07.2025

1.1 Table of Contents

1. Introduction

1. Team Members

2. Description of Tested Functionality

2. Test Case Management Template

3. Testing Methodology and Execution

1. Testing Methods & Tools

2. Test Results Summary

4. Conclusion

5. Annexes

1. Annex A: Automated Test Execution Details

2. Annex B: Manual Test Execution Details

1.2 Introduction

The primary objective of this project was to design and implement a robust testing strategy to validate the application’s functionality, stability, and data integrity.

This report details the testing methodologies employed, the test cases executed (both automated and manual), and a summary of the overall test plan and results.

1.2.1 Team Members

The following students contributed to this semester project:

- Ishan Baichoo
- Gabriel Aumasson-Leduc
- Clément De Simon

1.2.2 Description of Tested Functionality

“The New Cantina” is a web-based meal ordering system designed for a university environment. The application serves two primary user roles, each with distinct functionalities:

- **Student / Staff (User Role):**
 - **Authentication:** Users can log in with their credentials.
 - **Dashboard:** After logging in, users can view daily menus for different cafeterias. They can navigate between cafeterias and select different dates to view menus.
 - **Ordering:** Users can add items to a shopping cart, review their order, and place it. The total cost is deducted from their account balance.
 - **Account Management:** Users can view their current balance, top up their account with additional funds, and view their complete order history, filterable by month.
- **Administrator (Admin Role):**
 - **Authentication:** Admins log in through the same portal but are redirected to a dedicated admin dashboard.
 - **User Management:** Admins can view, search, filter, and delete all users in the system. They can also create new users.
 - **Cafeteria, Dish, and Menu Management:** Admins have full CRUD (Create, Read, Update, Delete) capabilities over cafeterias, individual dishes, and the daily menus that link them together. This includes a comprehensive interface for managing which dishes are served in which cafeterias on any given day.
 - **RESTful API:** All administrative actions are backed by a secure RESTful API, ensuring that data management can be performed programmatically and is subject to authorization checks.

The entire application is built on a Flask framework with a PostgreSQL database and a modern, responsive frontend using HTMX and Alpine.js.

1.3 Test Case Management Template

To effectively manage the testing process, our team designed a comprehensive and flexible template using Microsoft Excel. This approach was chosen for its universal accessibility, ease of use, and powerful features for data organization and tracking, requiring no specialized software for team members.

The system is built on a two-tiered structure:

1. A high-level **Test Summary Dashboard** for at-a-glance project management and task distribution.
2. Individual, detailed **Test Case Sheets** for every test, providing all necessary information for execution and recording.

1.3.0.1 The Test Summary Dashboard The “Test Summary” sheet serves as the central hub for monitoring the entire testing effort. Its primary purpose is to provide a quick overview of test status, feature coverage, and team workload.

The key columns and their functions are:

- **Test ID:** A unique identifier for each test case. A consistent naming convention (e.g., MANUAL-TC-XXX, API-SEC-XXX) is used to categorize tests by type and scope.
- **Title:** A concise, human-readable name for the test case.
- **Feature:** The application module or feature being tested (e.g., Account Management, Ordering, API Security). This allows for filtering to assess the test coverage of specific parts of the system.
- **Status:** The current state of the test case, which is color-coded for immediate visual feedback:
 - **Passed:** The test was executed successfully.
 - **Failed:** The test was executed, and a defect was found.

- **In Progress:** The test is currently being executed or debugged.
- **Not Executed / Draft:** The test case has been designed but not yet run.
- **Assigned To:** The name of the team member responsible for the test case. This is crucial for distributing work and establishing clear ownership.
- **Date Tested:** The date of the last execution, providing a timeline of testing activity.

This dashboard structure is essential for team coordination, allowing the project lead to quickly identify bottlenecks, track progress against features, and re-allocate tasks as needed.

1.3.0.2 The Detailed Test Case Sheet Each test case listed in the summary dashboard has its own dedicated sheet, named after its Test ID (e.g., “MANUAL-TC-001”). This sheet provides a standardized and exhaustive template for a tester to follow, ensuring that tests are repeatable and results are recorded consistently.

The detailed sheet is organized into the following logical sections:

- **Header & Metadata:** This top section captures the administrative details of the test case itself, including **Test Case ID**, **Description**, **Created By**, **Reviewed By**, and **Version**. This establishes a quality control process where tests can be drafted and peer-reviewed before execution. The **QA Tester's Log** provides space for notes on the test case's evolution.
- **Execution Details:** This block records the results of a specific run, with fields for **Tester's Name**, **Date Tested**, and the final **Test Case (Pass/Fail)** status.
- **Prerequisites & Test Data:** This section is critical for reproducibility. **Prerequisites** lists all conditions that must be met before starting the test (e.g., user must be logged in), while **Test Data** specifies any inputs or values needed (e.g., amount to add).
- **Test Scenario:** A clear, one-sentence goal for the test.
- **Test Steps:** The core of the test case, this table provides a script for the tester to follow with explicit columns for:
 - **Step #:** Sequential numbering.
 - **Step Details:** The action to be performed.
 - **Expected Results:** The specific outcome that should occur if the application is working correctly.
 - **Actual Results:** The observed outcome during the test.
 - **Pass / Fail / ...:** The status of the individual step.

1.3.0.3 Specific Example: MANUAL-TC-001 To illustrate how the template is used in practice, here is the record for MANUAL-TC-001.

1. View in the “Test Summary” Dashboard:

Test ID	Title	Feature	Status	Assigned To	Date Tested
MANUAL-TC-001	Verify Visual Feedback and State on Balance Top-Up	Account Management	Passed	Clément De Simon	2025-07-08

2. View in the “MANUAL-TC-001” Detailed Sheet:

Attribute	Value
Test Case ID	MANUAL-TC-001
Test Case Description	Verify that when a user adds funds, the UI provides clear feedback and all balance indicators update correctly without a full page reload.
Created By	Gabriel Aumasson-Leduc
Reviewed By	Ishan Baichoo
Version	1.0

Attribute	Value
QA Tester's Log	Initial draft for manual testing phase. Covers HTMX out-of-band swaps.
Tester's Name	Clément De Simon
Date Tested	2025-07-08
Test Case (Pass/Fail/Not Executed)	Pass

Prerequisites: 1. User is logged in as 'student1@example.com'. 2. Application is running in a modern web browser.

Test Data: 1. Amount to add: 25.50

Test Scenario: Verify that when a user adds funds, the UI provides clear feedback and all balance indicators update correctly without a full page reload.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Navigate to the 'Top Up Balance' page.	The 'Account Balance' page is displayed.	As Expected	Pass
2	Enter '25.50' into the amount field and click 'Add Money'.	A success message appears and the 'Current Balance' on the page updates.	As Expected	Pass
3	Observe the header of the application.	The balance in the top-right corner updates instantly without a page refresh.	As Expected	Pass
4	Navigate to the main 'Dashboard' page.	The balance in the header remains at the new, updated value.	As Expected	Pass

1.4 Testing Methodology and Execution

1.4.1 Testing Methods & Tools

A hybrid testing strategy was adopted to ensure comprehensive coverage, combining the efficiency of automation with the intuitiveness of manual testing.

Automated Testing: Automated tests form the backbone of our quality assurance process, focusing on the backend logic, API endpoints, and critical user workflows. This allows for rapid, repeatable, and consistent verification, which is essential for regression testing.

Manual Testing: Manual testing was employed to cover areas where human observation is paramount. This includes exploratory testing to find edge cases, verifying the user interface (UI) for visual consistency and responsiveness, and assessing the overall user experience (UX).

The following tools were utilized to implement our testing strategy:

Tool	Purpose
pytest	The core testing framework for Python. Used to write, organize, and run all automated tests, including model, API, and E2E tests. Its powerful fixture system was used to create isolated test environments.
Selenium IDE	Used for the initial recording of complex user workflows in the browser. This provided a quick way to generate baseline scripts for key end-to-end scenarios.
pytest-selenium	This pytest plugin was used to integrate and run the browser automation scripts (exported from Selenium IDE) within our primary testing framework, allowing for unified test execution and reporting.
Docker & Docker Compose	The entire application stack (Python app, PostgreSQL database, pgAdmin) was containerized using Docker. Docker Compose was used to define and orchestrate the multi-container environment, ensuring a consistent and reproducible setup for both development and testing.
Custom Python Scripts	A custom script (report_converter.py) was developed to parse the JUnit XML output from pytest and a manual test Excel file to generate a professional, human-readable Markdown report with a summary and detailed annexes.
Microsoft Excel	Used as a centralized tool for designing, documenting, and tracking the execution of manual test cases.

1.4.2 Test Results Summary

The following tables provide a high-level overview of the entire testing effort, combining results from both automated and manual test executions.

1.4.2.1 Overall Test Statistics

Test Category	Total	Passed	Failed	Skipped	Pass Rate
Model & Unit Tests	40	37	3	0	92.5%
API Tests	52	50	2	0	96.2%
End-to-End (E2E) Tests	2	1	1	0	50.0%
Scenario & Integration Tests	7	7	0	0	100.0%
Uncategorized	60	60	0	0	100.0%
Manual Tests	3	1	1	0	33.3%
Total	164	156	7	0	95.1%

1.4.2.2 Key Findings (Failures) This section highlights all tests that failed or resulted in an error during execution, providing a quick reference for developers to address critical issues. For complete details, including stack traces and test steps, please refer to the corresponding test case in the Annexes.

Test ID	Description	Category	Details
API-SEC-001	Verify that a standard user receives a 403 Forbidden error when attempting to access admin-only API routes.	Manual Tests	See Details

Test ID	Description	Category	Details
API_SEC_001	test_user_api_permissions[endpoint_config13]	API	See
		Tests	Details
API_SEC_001	test_user_api_permissions[endpoint_config14]	API	See
		Tests	Details
E2E_ORDER_001	orderfoodinthepast	End-to-End (E2E)	See
		Tests	Details
MODEL_DB_001	menu_uniqueness_constraint_on_update	Model & Unit	See
		Tests	Details
MODEL_MENUITEM_004	create_menu_item_with_no_data	Model & Unit	See
		Tests	Details
MODEL_MENUITEM_005	all_menu_items_as_dicts	Model & Unit	See
		Tests	Details

1.5 Conclusion

The primary objective was to validate the core functionality of the application, encompassing both the student-facing user interface and the administrative-level RESTful APIs. The goal was to ensure the system's stability, data integrity, and adherence to specified business rules before deployment.

The scope of testing was comprehensive, covering critical components of the application. For the backend, this included full Create, Read, Update, and Delete (CRUD) lifecycle testing for the User, Dish, Cafeteria, and Daily Menu API endpoints. Authentication and authorization were rigorously checked to ensure standard users could not access admin-protected resources. On the frontend, testing focused on key user journeys: the complete student workflow from login to browsing menus, adding items to a cart, and placing an order, as well as the administrator's ability to log in and manage menus. Additionally, model-level tests were conducted to verify database constraints, such as the uniqueness of menus. Intentionally out of scope for this phase were performance and load testing, formal security vulnerability scanning, and usability testing with a formal user group.

A hybrid strategy combining automated and manual testing methodologies was employed to achieve broad and deep coverage. Critical API functionality, model-level data integrity, and repetitive UI workflows were automated to ensure consistent and repeatable verification. These automated scripts formed the core of our regression suite. Manual exploratory testing was used to supplement this, focusing on verifying visual layout, user experience edge cases, and scenarios not easily covered by automated scripts. This blended approach allowed for both efficient regression checking and flexible, human-driven validation.

To execute this strategy, a specific set of tools was chosen. The **pytest** framework served as the foundation for all Python-based testing due to its powerful fixture system and extensibility. For backend API and model-level tests, **pytest** was used with an in-memory SQLite database to ensure fast and isolated execution. End-to-end browser automation was accomplished using **Selenium IDE** for initial workflow recording, with the exported scripts managed and run via **pytest-selenium**. The entire application and its dependencies were orchestrated using **Docker** and **docker-compose**, which provided a consistent and reproducible testing environment across all stages.

1.6 Annexes

This section contains the detailed execution reports for all automated and manual test cases.

1.7 Annex A: Automated Test Execution Details

Metric	Value
Total Automated Tests	161
Passed	155
Failed/Error	6
Skipped	0

1.7.1 Model & Unit Tests

1.7.1.1 MODEL_CAFE_001: Verify Cafeteria model can create a new cafeteria.

Attribute	Value
Test Case ID	MODEL_CAFE_001
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_cafeteria

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call Cafeteria.create_cafeteria().	A new cafeteria is created and persisted to the database.	As Expected	Pass

1.7.1.2 MODEL_CAFE_002: Verify Cafeteria model can update a cafeteria's name.

Attribute	Value
Test Case ID	MODEL_CAFE_002
Version	1.0
Tester	Automation
Execution Time	0.3040s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_cafeteria

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_cafeteria() with a new name.	The cafeteria's name is successfully updated in the database.	As Expected	Pass

1.7.1.3 MODEL_CAFE_003: Verify get_all_dicts for Cafeteria returns all records.

Attribute	Value
Test Case ID	MODEL_CAFE_003
Version	1.0
Tester	Automation
Execution Time	0.3450s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_all_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create multiple cafeterias and call get_all_dicts().	A list containing all created cafeteria dictionaries is returned.	As Expected	Pass

1.7.1.4 MODEL_CAFE_004: Verify Cafeteria model can delete a cafeteria.

Attribute	Value
Test Case ID	MODEL_CAFE_004
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_delete_cafeteria

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call delete_cafeteria() on an instance.	The cafeteria is removed from the database.	As Expected	Pass

1.7.1.5 MODEL_CAFE_005: Verify update_cafeteria() with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_CAFE_005
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_cafeteria_with_no_data

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_cafeteria() with no parameters.	The method must return False.	As Expected	Pass

1.7.1.6 MODEL_DB_002: Verify database uniqueness constraint for user emails.

Attribute	Value
Test Case ID	MODEL_DB_002
Version	1.0
Tester	Automation
Execution Time	0.3950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_unique_email_constraint

Prerequisites: None

Test Scenario: The database should prevent the creation of two users with the same email address, raising an IntegrityError.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create and commit User A with a specific email.	User A is created successfully.	As Expected	Pass
2	Create User B with the same email.	User B instance is created in the session.	As Expected	Pass
3	Attempt to commit the session with User B.	The commit must fail and raise an IntegrityError.	As Expected	Pass

1.7.1.7 MODEL_DB_003: Verify that updating a user's email to an already existing email fails.

Attribute	Value
Test Case ID	MODEL_DB_003
Version	1.0
Tester	Automation
Execution Time	0.3830s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_app_user_update_to_existing_email_fails

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create two users. Attempt to update the second user's email to match the first user's email.	The update_user() method must return False due to the unique constraint violation.	As Expected	Pass

1.7.1.8 MODEL_DB_004: Verify a dish cannot be deleted if referenced by a DailyMenu-Item.

Attribute	Value
Test Case ID	MODEL_DB_004
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_dish_delete_fails_if_in_use_by_menu_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a dish and add it to a menu.	The setup is successful.	As Expected	Pass
2	Attempt to delete the dish.	The delete_dish() method must return False.	As Expected	Pass

1.7.1.9 MODEL_DB_005: Verify a dish cannot be deleted if referenced by an OrderItem.

Attribute	Value
Test Case ID	MODEL_DB_005
Version	1.0
Tester	Automation
Execution Time	0.3440s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_dish_delete_fails_if_in_use_by_order_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a dish and include it in a reservation.	The setup is successful.	As Expected	Pass
2	Attempt to delete the dish.	The delete_dish() method must return False.	As Expected	Pass

1.7.1.10 MODEL_DISH_001: Verify a Dish can be created.

Attribute	Value
Test Case ID	MODEL_DISH_001
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_dish

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call Dish.create_dish().	A new dish is created.	As Expected	Pass

1.7.1.11 MODEL_DISH_002: Verify a Dish can be updated from a dictionary.

Attribute	Value
Test Case ID	MODEL_DISH_002

Attribute	Value
Version	1.0
Tester	Automation
Execution Time	0.2990s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_from_dict

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_from_dict() with new data.	The dish's attributes are updated.	As Expected	Pass

1.7.1.12 MODEL_DISH_003: Verify a Dish can be retrieved by its ID.

Attribute	Value
Test Case ID	MODEL_DISH_003
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_by_id

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a dish and call get_by_id() with its ID.	The correct dish instance is returned.	As Expected	Pass

1.7.1.13 MODEL_DISH_004: Verify a Dish can be deleted.

Attribute	Value
Test Case ID	MODEL_DISH_004
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_delete_dish

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call delete_dish() on an instance.	The dish is removed.	As Expected	Pass

1.7.1.14 MODEL_DISH_005: Verify get_all_dicts for Dish returns all records.

Attribute	Value
Test Case ID	MODEL_DISH_005
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_all_dishes_as_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create multiple dishes and call get_all_dicts().	A list of all created dish dictionaries is returned.	As Expected	Pass

1.7.1.15 MODEL_MENUITEM_001: Verify a DailyMenuItem can be created.

Attribute	Value
Test Case ID	MODEL_MENUITEM_001
Version	1.0
Tester	Automation
Execution Time	0.2980s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_menu_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call DailyMenuItem.create_menu_item().	A new menu item is created.	As Expected	Pass

1.7.1.16 MODEL_MENUITEM_002: Verify a DailyMenuItem can be updated.

Attribute	Value
Test Case ID	MODEL_MENUITEM_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_menu_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_menu_item() with new data.	The item's attributes are updated.	As Expected	Pass

1.7.1.17 MODEL_MENUITEM_003: Verify a DailyMenuItem can be deleted.

Attribute	Value
Test Case ID	MODEL_MENUITEM_003
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_delete_menu_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call delete_menu_item() on an instance.	The item is removed.	As Expected	Pass

1.7.1.18 MODEL_MENU_001: Verify DailyMenu model can create a menu.

Attribute	Value
Test Case ID	MODEL_MENU_001
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_menu

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call Daily-Menu.create_menu().	A new menu is created and persisted.	As Expected	Pass

1.7.1.19 MODEL_MENU_002: Verify DailyMenu model can update a menu.

Attribute	Value
Test Case ID	MODEL_MENU_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_menu

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_menu() with a new date.	The menu's date is updated.	As Expected	Pass

1.7.1.20 MODEL_MENU_003: Verify DailyMenu model can delete a menu.

Attribute	Value
Test Case ID	MODEL_MENU_003
Version	1.0
Tester	Automation

Attribute	Value
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_delete_menu

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call delete_menu() on a menu instance.	The menu is removed from the database.	As Expected	Pass

1.7.1.21 MODEL_MENU_004: Verify update_menu() with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_MENU_004
Version	1.0
Tester	Automation
Execution Time	0.3010s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_menu_with_no_data

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_menu() with no parameters.	The method must return False.	As Expected	Pass

1.7.1.22 MODEL_MENU_005: Verify get_all_dicts for DailyMenu returns all records.

Attribute	Value
Test Case ID	MODEL_MENU_005
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_all_menus_as_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create multiple menus and call get_all_dicts().	A list containing all created menu dictionaries is returned.	As Expected	Pass

1.7.1.23 MODEL_ORDERITEM_001: Verify an OrderItem can be created.

Attribute	Value
Test Case ID	MODEL_ORDERITEM_001
Version	1.0
Tester	Automation
Execution Time	0.3430s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_order_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call OrderItem.create_order_item().created.	A new order item is	As Expected	Pass

1.7.1.24 MODEL_ORDERITEM_002: Verify an OrderItem can be updated.

Attribute	Value
Test Case ID	MODEL_ORDERITEM_002
Version	1.0
Tester	Automation
Execution Time	0.3450s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_order_item

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call <code>update_order_item()</code> with new data.	The item's attributes are updated.	As Expected	Pass

1.7.1.25 MODEL_ORDERITEM_003: Verify an OrderItem can be deleted.

Attribute	Value
Test Case ID	MODEL_ORDERITEM_003
Version	1.0
Tester	Automation
Execution Time	0.3450s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	<code>test_delete_order_item</code>

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call <code>delete_order_item()</code> on an instance.	The item is removed.	As Expected	Pass

1.7.1.26 MODEL_ORDERITEM_004: Verify `update_order_item()` with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_ORDERITEM_004
Version	1.0
Tester	Automation
Execution Time	0.3410s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	<code>test_update_order_item_with_no_data</code>

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call <code>update_order_item()</code> with no parameters.	The method must return False.	As Expected	Pass

1.7.1.27 MODEL_ORDERITEM_005: Verify get_all_dicts for OrderItem returns all records.

Attribute	Value
Test Case ID	MODEL_ORDERITEM_005
Version	1.0
Tester	Automation
Execution Time	0.3420s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_all_order_items_as_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create items and call get_all_dicts().	A list of all created item dictionaries is returned.	As Expected	Pass

1.7.1.28 MODEL_RESERVATION_001: Verify a Reservation can be created.

Attribute	Value
Test Case ID	MODEL_RESERVATION_001
Version	1.0
Tester	Automation
Execution Time	0.3400s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_reservation

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call Reservation.create_reservation().	A new reservation is created.	As Expected	Pass

1.7.1.29 MODEL_RESERVATION_002: Verify a Reservation can be updated.

Attribute	Value
Test Case ID	MODEL_RESERVATION_002
Version	1.0

Attribute	Value
Tester	Automation
Execution Time	0.3400s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_reservation

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_reservation() with new data.	The reservation's attributes are updated.	As Expected	Pass

1.7.1.30 MODEL_RESERVATION_003: Verify a Reservation can be deleted.

Attribute	Value
Test Case ID	MODEL_RESERVATION_003
Version	1.0
Tester	Automation
Execution Time	0.3400s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_delete_reservation

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call delete_reservation() on an instance.	The reservation is removed.	As Expected	Pass

1.7.1.31 MODEL_RESERVATION_004: Verify update_reservation() with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_RESERVATION_004
Version	1.0
Tester	Automation
Execution Time	0.3410s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_update_reservation_with_no_data

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_reservation() with no parameters.	The method must return False.	As Expected	Pass

1.7.1.32 MODEL_RESERVATION_005: Verify get_all_dicts for Reservation returns all records.

Attribute	Value
Test Case ID	MODEL_RESERVATION_005
Version	1.0
Tester	Automation
Execution Time	0.3520s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_get_all_reservations_as_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create reservations and call get_all_dicts().	A list of all created reservation dictionaries is returned.	As Expected	Pass

1.7.1.33 MODEL_USER_001: Verify AppUser model can create a user with a hashed password.

Attribute	Value
Test Case ID	MODEL_USER_001
Version	1.0
Tester	Automation
Execution Time	0.4230s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_create_user

Prerequisites: None

Test Scenario: The create_user class method should correctly instantiate a user, hash their password, and add them to the database session.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call AppUser.create_user with valid details.	An AppUser instance is returned.	As Expected	Pass
2	Commit the session.	The user is saved to the database with a user_id.	As Expected	Pass
3	Verify the user's password.	The verify_password method returns True for the correct password and False for an incorrect one.	As Expected	Pass

1.7.1.34 MODEL_USER_002: Verify AppUser model can update user attributes.

Attribute	Value
Test Case ID	MODEL_USER_002
Version	1.0
Tester	Automation
Execution Time	0.3420s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_update_user

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a user, then call update_user() with new data (e.g., last_name, balance).	The user's attributes are updated in the database and the changes are persisted.	As Expected	Pass

1.7.1.35 MODEL_USER_003: Verify AppUser model can delete a user.

Attribute	Value
Test Case ID	MODEL_USER_003
Version	1.0
Tester	Automation
Execution Time	0.3480s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_delete_user

Attribute	Value
-----------	-------

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a user, retrieve their ID, then call delete_user().	The user is removed from the database and can no longer be retrieved by their ID.	As Expected	Pass

1.7.1.36 MODEL_USER_004: Verify calling update_user with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_USER_004
Version	1.0
Tester	Automation
Execution Time	0.3400s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_app_user_update_nothing_returns_false

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	On a user instance, call update_user() with no parameters.	The method must return False, indicating no update was performed.	As Expected	Pass

1.7.1.37 MODEL_USER_005: Verify that updating password and role works correctly.

Attribute	Value
Test Case ID	MODEL_USER_005
Version	1.0
Tester	Automation
Execution Time	0.4780s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_app_user_update_password_and_role

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_user() with a new password and role.	Method returns True, role is updated, and the new password can be verified while the old one cannot.	As Expected	Pass

1.7.1.38 MODEL_DB_001: Verify database uniqueness constraint for (cafeteria_id, menu_date) on update.

Attribute	Value
Test Case ID	MODEL_DB_001
Version	1.0
Tester	Automation
Execution Time	0.3050s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_menu_uniqueness_constraint_on_update

Prerequisites: None

Test Scenario: The system should prevent a menu from being updated to a date/cafeteria combination that already exists for another menu.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create Menu A for Cafeteria 1 on Date X.	Menu is created.	As Expected	Pass
2	Create Menu B for Cafeteria 2 on Date Y.	Menu is created.	As Expected	Pass
3	Attempt to update Menu B to use Cafeteria 1 and Date X.	The update operation must fail and return False due to the unique constraint violation.	Execution failed. See details below.	Failed

Failure Details:

```
app = <Flask 'app.controller.controller'>
```

```
def test_menu_uniqueness_constraint_on_update(app):
    """Vérifie la contrainte d'unicité (cafeteria_id, menu_date) lors d'une mise à jour."""
    with app.app_context():
        caf1 = Cafeteria.create_cafeteria("Caf 1")
        caf2 = Cafeteria.create_cafeteria("Caf 2")
        db.session.commit()

        # Menu existant pour caf1 à une date donnée
```



```

DailyMenu.create_menu(cafeteria_id=caf1.cafeteria_id, menu_date=date(2030, 1, 1))

# Autre menu pour caf2 qu'on va essayer de déplacer
menu_to_update = DailyMenu.create_menu(cafeteria_id=caf2.cafeteria_id, menu_date=date(20
db.session.commit()

# Tenter de déplacer le menu vers une date/cafeteria déjà prise
ok = menu_to_update.update_menu(cafeteria_id=caf1.cafeteria_id, menu_date=date(2030, 1,
> assert ok is False # La méthode doit retourner False en cas d'échec d'intégrité
~~~~~
E assert True is False

tests/test-python/models/test_daily_menu.py:61: AssertionError

```

1.7.1.39 MODEL_MENUITEM_004: Verify update_menu_item() with no arguments returns False.

Attribute	Value
Test Case ID	MODEL_MENUITEM_004
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_update_menu_item_with_no_data

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call update_menu_item() with no parameters.	The method must return False.	Execution failed. See details below.	Failed

Failure Details:

```

self = <sqlalchemy.engine.base.Connection object at 0x10dc084d0>, dialect = <sqlalchemy.dialects.sql
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc904d0>
statement = <sqlalchemy.dialects.sqlite.base.SQLiteCompiler object at 0x1074be850>, parameters = [(N

def _exec_single_context(
    self,
    dialect: Dialect,
    context: ExecutionContext,
    statement: Union[str, Compiled],
    parameters: Optional[_AnyMultiExecuteParams],
) -> CursorResult[Any]:
    """continue the _execute_context() method for a single DBAPI
    cursor.execute() or cursor.executemany() call.

    """
    if dialect.bind_typing is BindTyping.SETINPUTSIZES:

```

```

generic_setinputsizes = context._prepare_set_input_sizes()

if generic_setinputsizes:
    try:
        dialect.do_set_input_sizes(
            context.cursor, generic_setinputsizes, context
        )
    except BaseException as e:
        self._handle_dbapi_exception(
            e, str(statement), parameters, None, context
        )

cursor, str_statement, parameters = (
    context.cursor,
    context.statement,
    context.parameters,
)

effective_parameters: Optional[_AnyExecuteParams]

if not context.executemany:
    effective_parameters = parameters[0]
else:
    effective_parameters = parameters

if self._has_events or self.engine._has_events:
    for fn in self.dispatch.before_cursor_execute:
        str_statement, effective_parameters = fn(
            self,
            cursor,
            str_statement,
            effective_parameters,
            context,
            context.executemany,
        )

if self._echo:
    self._log_info(str_statement)

    stats = context._get_cache_stats()

    if not self.engine.hide_parameters:
        self._log_info(
            "[%s] %r",
            stats,
            sql_util._repr_params(
                effective_parameters,
                batches=10,
                ismulti=context.executemany,
            ),
        )
    else:
        self._log_info(
            "[%s] [SQL parameters hidden due to hide_parameters=True]",
            stats,
        )

evt_handled: bool = False

```

```

try:
    if context.execute_style is ExecuteStyle.EXECUTEMANY:
        effective_parameters = cast(
            "_CoreMultiExecuteParams", effective_parameters
        )
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_executemany:
                if fn(
                    cursor,
                    str_statement,
                    effective_parameters,
                    context,
                ):
                    evt_handled = True
                    break
        if not evt_handled:
            self.dialect.do_executemany(
                cursor,
                str_statement,
                effective_parameters,
                context,
            )
    elif not effective_parameters and context.no_parameters:
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_execute_no_params:
                if fn(cursor, str_statement, context):
                    evt_handled = True
                    break
        if not evt_handled:
            self.dialect.do_execute_no_params(
                cursor, str_statement, context
            )
    else:
        effective_parameters = cast(
            "_CoreSingleExecuteParams", effective_parameters
        )
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_execute:
                if fn(
                    cursor,
                    str_statement,
                    effective_parameters,
                    context,
                ):
                    evt_handled = True
                    break
        if not evt_handled:
            self.dialect.do_execute(
                cursor, str_statement, effective_parameters, context
            )

```

../../../../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p

```

self = <sqlalchemy.dialects.sqlite.pysqlite.SQLiteDialect_pysqlite object at 0x1074be0d0>, cursor =
statement = 'INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)', paramet
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc904d0>

```

```

    def do_execute(self, cursor, statement, parameters, context=None):
>         cursor.execute(statement, parameters)
E         sqlite3.IntegrityError: NOT NULL constraint failed: daily_menu.cafeteria_id

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/default

The above exception was the direct cause of the following exception:

app = <Flask 'app.controller.controller'>

    def test_update_menu_item_with_no_data(app):
        with app.app_context():
            caf = Cafeteria.create_cafeteria("DMI4")
            dish = Dish.create_dish("D4", "", 1, "main_course")
            menu = DailyMenu.create_menu(caf.cafeteria_id, date.today())
>         db.session.commit()

tests/test-python/models/test_daily_menu_item.py:52:
-----
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/scoping.p
    return self._proxied.commit()
    ~~~~~
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    trans.commit(_to_root=True)
<string>:2: in commit
    ???
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/state_cha
    ret_value = fn(self, *arg, **kw)
    ~~~~~
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self._prepare_impl()
<string>:2: in _prepare_impl
    ???
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/state_cha
    ret_value = fn(self, *arg, **kw)
    ~~~~~
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self.session.flush()
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self._flush(objects)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    with util.safe_reraise():
    ~~~~~
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/util/langhelp
    raise exc_value.with_traceback(exc_tb)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    flush_context.execute()
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/unitofwor
    rec.execute(self)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/unitofwor
    util.preloaded.orm_persistence.save_obj(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/persisten
    _emit_insert_statements(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/persisten
    result = connection.execute(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
    return meth(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/sql/elements.

```

```

        return connection._execute_clauseelement(
../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
        ret = self._execute_context(
../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
        return self._exec_single_context(
../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
        self._handle_dbapi_exception(
../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
        raise sqlalchemy_exception.with_traceback(exc_info[2]) from e
../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
        self.dialect.do_execute(
- - - - -

self = <sqlalchemy.dialects.sqlite.pysqlite.SQLiteDialect_pysqlite object at 0x1074be0d0>, cursor =
statement = 'INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)', paramet
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc904d0>

    def do_execute(self, cursor, statement, parameters, context=None):
>         cursor.execute(statement, parameters)
E         sqlalchemy.exc.IntegrityError: (sqlite3.IntegrityError) NOT NULL constraint failed: daily_me
E         [SQL: INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)]
E         [parameters: (None, '2025-07-08', '2025-07-08 14:33:30.047358')]
E         (Background on this error at: https://sqlalche.me/e/20/gkpj)

../.../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/default

```

1.7.1.40 MODEL_MENUITEM_005: Verify get_all_dicts for DailyMenuItem returns all records.

Attribute	Value
Test Case ID	MODEL_MENUITEM_005
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_get_all_menu_items_as_dicts

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create multiple items and call get_all_dicts().	A list containing all created item dictionaries is returned.	Execution failed. See details below.	Failed

Failure Details:

```

self = <sqlalchemy.engine.base.Connection object at 0x10dc91790>, dialect = <sqlalchemy.dialects.sql
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc92750>
statement = <sqlalchemy.dialects.sqlite.base.SQLiteCompiler object at 0x1070bf250>, parameters = [(N

```

```

def _exec_single_context(
    self,
    dialect: Dialect,
    context: ExecutionContext,
    statement: Union[str, Compiled],
    parameters: Optional[_AnyMultiExecuteParams],
) -> CursorResult[Any]:
    """continue the _execute_context() method for a single DBAPI
    cursor.execute() or cursor.executemany() call.

    """
    if dialect.bind_typing is BindTyping.SETINPUTSIZES:
        generic_setinputsizes = context._prepare_set_input_sizes()

        if generic_setinputsizes:
            try:
                dialect.do_set_input_sizes(
                    context.cursor, generic_setinputsizes, context
                )
            except BaseException as e:
                self._handle_dbapi_exception(
                    e, str(statement), parameters, None, context
                )

    cursor, str_statement, parameters = (
        context.cursor,
        context.statement,
        context.parameters,
    )

    effective_parameters: Optional[_AnyExecuteParams]

    if not context.executemany:
        effective_parameters = parameters[0]
    else:
        effective_parameters = parameters

    if self._has_events or self.engine._has_events:
        for fn in self.dispatch.before_cursor_execute:
            str_statement, effective_parameters = fn(
                self,
                cursor,
                str_statement,
                effective_parameters,
                context,
                context.executemany,
            )

    if self._echo:
        self._log_info(str_statement)

        stats = context._get_cache_stats()

        if not self.engine.hide_parameters:
            self._log_info(
                "[%s] %r",
                stats,

```

```

        sql_util._repr_params(
            effective_parameters,
            batches=10,
            ismulti=context.executemany,
        ),
    )
else:
    self._log_info(
        "[%s] [SQL parameters hidden due to hide_parameters=True]",
        stats,
    )

evt_handled: bool = False
try:
    if context.execute_style is ExecuteStyle.EXECUTEMANY:
        effective_parameters = cast(
            "_CoreMultiExecuteParams", effective_parameters
        )
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_executemany:
                if fn(
                    cursor,
                    str_statement,
                    effective_parameters,
                    context,
                ):
                    evt_handled = True
                    break
        if not evt_handled:
            self.dialect.do_executemany(
                cursor,
                str_statement,
                effective_parameters,
                context,
            )
    elif not effective_parameters and context.no_parameters:
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_execute_no_params:
                if fn(cursor, str_statement, context):
                    evt_handled = True
                    break
        if not evt_handled:
            self.dialect.do_execute_no_params(
                cursor, str_statement, context
            )
    else:
        effective_parameters = cast(
            "_CoreSingleExecuteParams", effective_parameters
        )
        if self.dialect._has_events:
            for fn in self.dialect.dispatch.do_execute:
                if fn(
                    cursor,
                    str_statement,
                    effective_parameters,
                    context,
                ):
                    evt_handled = True

```

```

        break
    if not evt_handled:
>         self.dialect.do_execute(
            cursor, str_statement, effective_parameters, context
        )

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
-----

self = <sqlalchemy.dialects.sqlite.pysqlite.SQLiteDialect_pysqlite object at 0x1070bf4d0>, cursor = 
statement = 'INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)', paramet
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc92750>

    def do_execute(self, cursor, statement, parameters, context=None):
>         cursor.execute(statement, parameters)
E         sqlite3.IntegrityError: NOT NULL constraint failed: daily_menu.cafeteria_id

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/default

The above exception was the direct cause of the following exception:

app = <Flask 'app.controller.controller'>

    def test_get_all_menu_items_as_dicts(app):
        with app.app_context():
            db.session.query(DailyMenuItem).delete()
            caf = Cafeteria.create_cafeteria("DMI5")
            dish = Dish.create_dish("D5", "", 1, "main_course")
            menu = DailyMenu.create_menu(caf.cafeteria_id, date.today())
>         db.session.commit()

tests/test-python/models/test_daily_menu_item.py:63:
-----
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/scoping.p
    return self._proxied.commit()
           ~~~~~~

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    trans.commit(_to_root=True)
<string>:2: in commit
    ???
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/state_cha
    ret_value = fn(self, *arg, **kw)
               ~~~~~~

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self._prepare_impl()
<string>:2: in _prepare_impl
    ???
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/state_cha
    ret_value = fn(self, *arg, **kw)
               ~~~~~~

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self.session.flush()
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    self._flush(objects)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
    with util.safe_reraise():
           ~~~~~~

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/util/langhelp

```



```

        raise exc_value.with_traceback(exc_tb)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/session.p
flush_context.execute()
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/unitofwor
rec.execute(self)
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/unitofwor
util.preloaded.orm_persistence.save_obj(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/persisten
_emit_insert_statements(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/orm/persisten
result = connection.execute(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
return meth(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/sql/elements.
return connection._execute_clauseelement(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
ret = self._execute_context(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
return self._exec_single_context(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
self._handle_dbapi_exception(
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
raise sqlalchemy_exception.with_traceback(exc_info[2]) from e
../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/base.p
self.dialect.do_execute(
-----

self = <sqlalchemy.dialects.sqlite.pysqlite.SQLiteDialect_pysqlite object at 0x1070bf4d0>, cursor =
statement = 'INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)', paramet
context = <sqlalchemy.dialects.sqlite.base.SQLiteExecutionContext object at 0x10dc92750>

def do_execute(self, cursor, statement, parameters, context=None):
> cursor.execute(statement, parameters)
E sqlalchemy.exc.IntegrityError: (sqlite3.IntegrityError) NOT NULL constraint failed: daily_me
E [SQL: INSERT INTO daily_menu (cafeteria_id, menu_date, created_at) VALUES (?, ?, ?)]
E [parameters: (None, '2025-07-08', '2025-07-08 14:33:30.487919')]
E (Background on this error at: https://sqlalche.me/e/20/gkpb)

../../../../pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/sqlalchemy/engine/default

```

1.7.2 API Tests

1.7.2.1 API_CAFE_001: Verify full CRUD for the Cafeteria API endpoint.

Attribute	Value
Test Case ID	API_CAFE_001
Version	1.0
Tester	Automation
Execution Time	0.3590s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_cafeteria_crud

Prerequisites: 1. An authenticated admin client.

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Perform Create, Read, Update, Delete operations on the Cafeteria API.	All operations succeed with correct HTTP status codes.	As Expected	Pass

1.7.2.2 API_DISH_001: Verify full CRUD for the Dish API endpoint.

Attribute	Value
Test Case ID	API_DISH_001
Version	1.0
Tester	Automation
Execution Time	0.3520s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_dish_crud

Prerequisites: 1. An authenticated admin client.

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Perform Create, Read, Update, Delete operations on the Dish API.	All operations succeed with correct HTTP status codes.	As Expected	Pass

1.7.2.3 API_MENU_001: Verify creation of Menus and linking Menu Items via API.

Attribute	Value
Test Case ID	API_MENU_001
Version	1.0
Tester	Automation
Execution Time	0.3500s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_menu_and_item_crud

Prerequisites: 1. Authenticated admin client. 1. A Cafeteria and a Dish exist in the DB.

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a DailyMenu via POST request.	Receives HTTP 201 and menu data.	As Expected	Pass
2	Create a DailyMenuItem via POST, linking the menu and a dish.	Receives HTTP 201 and menu item data.	As Expected	Pass
3	Delete the parent DailyMenu.	The operation succeeds, and the child DailyMenuItem should be deleted by cascade.	As Expected	Pass

1.7.2.4 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3610s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions(endpoint_config0]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.5 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3520s

Attribute	Value
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config1]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.6 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3550s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config2]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.7 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3540s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config3]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.8 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3550s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config4]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.9 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3680s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions(endpoint_config5]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.10 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3610s

Attribute	Value
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config6]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.11 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3510s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config7]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.12 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3800s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config8]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.13 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3510s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config9]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.14 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3560s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config10]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.15 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3520s

Attribute	Value
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config11]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.16 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3490s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_api_permissions[endpoint_config12]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

1.7.2.17 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2920s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__user__(Li tous les utilisateurs)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.18 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__user_1_(R un utilisateur par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.19 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3050s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[POST__user__(C un utilisateur)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.20 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[PUT__user_1_(M un utilisateur)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.21 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__user_1 un utilisateur)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.22 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2980s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_api_unauthenticated_access_is_denied[POST__user_bal de l'argent au solde)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.23 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__cafeteria les_caf\xe9t\xe9rias)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.24 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__cafeteria une caf\%t%ria par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.25 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2990s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[POST__cafeteria une caf\%t%ria)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.26 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3070s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[PUT__cafeteria une caf\%t%ria)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.27 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__cafete une caf\%t%ria)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.28 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3080s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__dish__(Lil plat les plats)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.29 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__dish_1_(Roulette un plat par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.30 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[POST__dish__(Cun plat)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.31 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2980s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_api_unauthenticated_access_is_denied[PUT__dish_1_(Mun plat)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.32 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2980s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__dish_1_(Mun plat)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.33 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3060s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__daily-men tous les menus)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.34 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2930s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__daily-men un menu par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.35 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2930s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__daily-men un menu par caf\%t\%ria)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.36 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3060s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[POST__daily-me un menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.37 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2930s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[PUT__daily-men un menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.38 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2920s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__daily- un menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.39 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2920s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__daily-men les items d'un menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.40 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2960s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_api_unauthenticated_access_is_denied[POST__daily-menu item \xe0 un menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.41 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2940s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[PUT__daily-menu item de menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.42 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2920s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__daily-un item de menu)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.43 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2930s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__order-items les items de commande)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.44 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2950s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET__order-item un item de commande par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.45 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.3050s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[DELETE__order- un item de commande)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.46 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET_reservations r\reservations)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.47 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2920s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[POST_reservation r\reservation)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.48 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2910s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_api_unauthenticated_access_is_denied[GET_reservation r\reservation par ID)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.49 API_SEC_002: Verify that unauthenticated API access is denied for all protected endpoints.

Attribute	Value
Test Case ID	API_SEC_002
Version	1.0
Tester	Automation
Execution Time	0.2910s
Date Tested	08-Jul-2025
Final Status	Passed

Attribute	Value
Test Function	test_api_unauthenticated_access_is_denied[PUT_reservation(r\reservation)]

Prerequisites: 1. Application is running.

Test Scenario: An API client without a valid session cookie must be rejected with an HTTP 401 or 403 error when attempting to access any protected resource. This is a parametrized test that covers all protected API endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send a request (GET, POST, PUT, DELETE) to a protected API endpoint without authentication.	The server must respond with an HTTP 401 or 403 status code.	As Expected	Pass

1.7.2.50 API_USER_001: Verify full CRUD (Create, Read, Update, Delete) for the User API endpoint.

Attribute	Value
Test Case ID	API_USER_001
Version	1.0
Tester	Automation
Execution Time	0.4360s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_user_crud

Prerequisites: 1. An authenticated admin client.

Test Scenario: An authenticated admin must be able to fully manage users through the API.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Send POST to /api/v1/user/ to create a new user.	Receives HTTP 201 and user data.	As Expected	Pass
2	Send GET to /api/v1/user/ to list all users.	Receives HTTP 200 and the new user is in the list.	As Expected	Pass
3	Send PUT to /api/v1/user/{id} to update the user.	Receives HTTP 200 and updated data.	As Expected	Pass
4	Send DELETE to /api/v1/user/{id} to delete the user.	Receives HTTP 200.	As Expected	Pass

1.7.2.51 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3490s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_user_api_permissions(endpoint_config13]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

Failure Details:

```
authenticated_client = <FlaskClient <Flask 'app.controller.controller'>>
endpoint_config = {'allowed': False, 'desc': "Voir la réservation d'un autre", 'method': 'GET', 'url'
```

```
@pytest.mark.parametrize("endpoint_config", ENDPOINTS_PERMISSIONS)
def test_user_api_permissions(authenticated_client, endpoint_config):
    """
    Teste systématiquement les permissions d'un utilisateur standard connecté.
    """
    client = authenticated_client

    with client.application.app_context():
        # Récupère les objets depuis la BDD (créés par le seeder)
        current_user = AppUser.get_by_email("jakub.novak@example.com")
        other_user = AppUser.get_by_email("john.smith@example.com")

        # Crée une réservation pour 'l'autre utilisateur' si nécessaire pour le test
        other_reservation = Reservation(user_id=other_user.user_id, cafeteria_id=1, total=1.0)
        db.session.add(other_reservation)
        db.session.flush() # Utilise flush pour obtenir l'ID sans commit

        # Prépare l'URL finale en injectant les IDs
        url = endpoint_config["url"].format(
            current_user_id=current_user.user_id,
            other_user_id=other_user.user_id,
```

```

        other_user_reservation_id=other_reservation.reservation_id
    )

    # Exécute la requête de test
    method = endpoint_config["method"].lower()
    kwargs = {"json": endpoint_config.get("json")} if "json" in endpoint_config else {}
    response = getattr(client, method)(url, **kwargs)

    # Vérifie le code de statut
    allowed = endpoint_config["allowed"]
    debug_info = (
        f"Endpoint: {endpoint_config['method']} {url}\n"
        f"Description: {endpoint_config['desc']}\n"
        f"Attendu: {'Autorisé (2xx)' if allowed else 'Refusé (401/403)'}\n"
        f"Reçu: {response.status_code}\n"
        f"Réponse: {response.data.decode(errors='ignore')[:200]}"
    )

    if allowed:
        assert 200 <= response.status_code < 300, f"Accès REFUSÉ à un endpoint qui devait être a
    else:
>         assert response.status_code in {401, 403}, f"Accès AUTORISÉ à un endpoint qui devait être
E         AssertionError: Accès AUTORISÉ à un endpoint qui devait être refusé.
E             Endpoint: GET /api/v1/reservations/1
E             Description: Voir la réservation d'un autre
E             Attendu: Refusé (401/403)
E             Reçu: 404
E             Réponse: {"error": "Reservation not found."}
E
E         assert 404 in {401, 403}
E         + where 404 = <WrapperTestResponse 35 bytes [404 NOT FOUND]>.status_code

tests/test-python/controller/test_api_auth.py:93: AssertionError

```

1.7.2.52 API_SEC_001: Verify API permissions for a standard authenticated (non-admin) user.

Attribute	Value
Test Case ID	API_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.3470s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_user_api_permissions[endpoint_config14]

Prerequisites: None

Test Scenario: A standard user should be able to access their own data but be denied access to other users' data or admin-only endpoints. This is a parametrized test that covers multiple endpoints.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Attempt to access admin-only endpoints (e.g., list all users, create a dish).	Access is denied with HTTP 401/403.	As Expected	Pass
2	Attempt to access own user data (e.g., GET /api/v1/user/{own_id}).	Access is allowed with HTTP 200.	As Expected	Pass
3	Attempt to access another user's data (e.g., GET /api/v1/reservations/{other_user_reservation_id}).	Access is denied with HTTP 401/403/404.	Execution failed. See details below.	Failed

Failure Details:

```

authenticated_client = <FlaskClient <Flask 'app.controller.controller'>>
endpoint_config = {'allowed': False, 'desc': "Annuler la réservation d'un autre", 'method': 'PUT', '

@pytest.mark.parametrize("endpoint_config", ENDPOINTS_PERMISSIONS)
def test_user_api_permissions(authenticated_client, endpoint_config):
    """
    Teste systématiquement les permissions d'un utilisateur standard connecté.
    """
    client = authenticated_client

    with client.application.app_context():
        # Récupère les objets depuis la BDD (créés par le seeder)
        current_user = AppUser.get_by_email("jakub.novak@example.com")
        other_user = AppUser.get_by_email("john.smith@example.com")

        # Crée une réservation pour 'l'autre utilisateur' si nécessaire pour le test
        other_reservation = Reservation(user_id=other_user.user_id, cafeteria_id=1, total=1.0)
        db.session.add(other_reservation)
        db.session.flush() # Utilise flush pour obtenir l'ID sans commit

        # Prépare l'URL finale en injectant les IDs
        url = endpoint_config["url"].format(
            current_user_id=current_user.user_id,
            other_user_id=other_user.user_id,
            other_user_reservation_id=other_reservation.reservation_id
        )

        # Exécute la requête de test
        method = endpoint_config["method"].lower()
        kwargs = {"json": endpoint_config.get("json")} if "json" in endpoint_config else {}
        response = getattr(client, method)(url, **kwargs)

        # Vérifie le code de statut
        allowed = endpoint_config["allowed"]
        debug_info = (
            f"Endpoint: {endpoint_config['method']} {url}\n"
            f>Description: {endpoint_config['desc']}\n"
            f"Attendu: {'Autorisé (2xx)' if allowed else 'Refusé (401/403)'}\n"
            f"Reçu: {response.status_code}\n"
            f"Réponse: {response.data.decode(errors='ignore')[:200]}"
        )

    if allowed:

```



```

        assert 200 <= response.status_code < 300, f"Accès REFUSÉ à un endpoint qui devait être a
    else:
>         assert response.status_code in {401, 403}, f"Accès AUTORISÉ à un endpoint qui devait être
E         AssertionError: Accès AUTORISÉ à un endpoint qui devait être refusé.
E             Endpoint: PUT /api/v1/reservations/1/cancel
E             Description: Annuler la réservation d'un autre
E             Attendu: Refusé (401/403)
E             Reçu: 404
E             Réponse: {"error": "Reservation not found."}
E
E         assert 404 in {401, 403}
E         + where 404 = <WrapperTestResponse 35 bytes [404 NOT FOUND]>.status_code

tests/test-python/controller/test_api_auth.py:93: AssertionError

```

1.7.3 End-to-End (E2E) Tests

1.7.3.1 E2E_LOGIN_001: Verify Admin Login and Logout functionality through the UI.

Attribute	Value
Test Case ID	E2E_LOGIN_001
Version	1.0
Tester	Automation
Execution Time	3.4250s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_loginAdminLogOut

Prerequisites: 1. Application is running. 1. Default admin user exists.

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Navigate to login page.	Page loads successfully.	As Expected	Pass
2	Enter admin credentials and submit.	Redirected to the admin dashboard.	As Expected	Pass
3	Click the 'Logout' link.	Redirected back to the login page.	As Expected	Pass

1.7.3.2 E2E_ORDER_001: Verify the end-to-end student workflow for ordering a meal.

Attribute	Value
Test Case ID	E2E_ORDER_001
Version	1.0
Tester	Automation
Execution Time	3.2460s
Date Tested	08-Jul-2025
Final Status	Failed
Test Function	test_orderfoodinthepast

Attribute	Value
-----------	-------

Prerequisites: 1. Application is running. 1. Default student user exists. 1. Menus are available for the current month.

Test Scenario: A student logs in, navigates to a cafeteria, selects a date, adds a dish to their cart, places the order, and verifies the order in their history.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Navigate to login page and log in as a student.	Redirected to user dashboard.	As Expected	Pass
2	Navigate between different cafeteria links.	The main content updates to show the selected cafeteria's menu page.	As Expected	Pass
3	Open the date picker and select a date.	The menu table updates to show dishes for the selected date.	As Expected	Pass
4	Add an available item to the cart.	The item appears in the cart summary and the 'Add' button changes to 'Added'.	As Expected	Pass
5	Click 'Place Order'.	A success notification appears and the user is redirected to the 'Order History' page.	As Expected	Pass
6	Verify the new order appears on the 'Order History' page.	The order history list contains the newly placed order.	Execution failed. See details below.	Failed

Failure Details:

```
self = <test_orderfoodinthepast.TestOrderfoodinthepast object at 0x10657ad50>
```

```
def test_orderfoodinthepast(self):
    self.driver.get("http://localhost:8081/login")
    self.driver.set_window_size(1512, 888)
    self.driver.find_element(By.ID, "username").click()
    self.driver.find_element(By.ID, "username").send_keys("student1@example.com")
    self.driver.find_element(By.ID, "password").send_keys("pass123")
    self.driver.find_element(By.ID, "password").send_keys(Keys.ENTER)
> self.driver.find_element(By.LINK_TEXT, "Kafeteria").click()
~~~~~
```

```
tests/test-python/selenium-e2e/test_orderfoodinthepast.py:28:
```

```
-----
../../../../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/selenium/webdriver/remote
    return self.execute(Command.FIND_ELEMENT, {"using": by, "value": value})["value"]
~~~~~
../../../../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/selenium/webdriver/remote
    self.error_handler.check_response(response)
-----
```

```
self = <selenium.webdriver.remote.errorhandler.ErrorHandler object at 0x10e06c550>
```

```
response = {'status': 404, 'value': '{"value":{"error":"no such element"},"message":"Unable to locate
```

```

def check_response(self, response: dict[str, Any]) -> None:
    """Checks that a JSON response from the WebDriver does not have an
    error.

    :Args:
        - response - The JSON response from the WebDriver server as a dictionary
            object.

    :Raises: If the response contains an error message.
    """
    status = response.get("status", None)
    if not status or status == ErrorCode.SUCCESS:
        return
    value = None
    message = response.get("message", "")
    screen: str = response.get("screen", "")
    stacktrace = None
    if isinstance(status, int):
        value_json = response.get("value", None)
        if value_json and isinstance(value_json, str):
            try:
                value = json.loads(value_json)
                if isinstance(value, dict):
                    if len(value) == 1:
                        value = value["value"]
                    status = value.get("error", None)
                    if not status:
                        status = value.get("status", ErrorCode.UNKNOWN_ERROR)
                        message = value.get("value") or value.get("message")
                        if not isinstance(message, str):
                            value = message
                            message = message.get("message")
                    else:
                        message = value.get("message", None)
            except ValueError:
                pass

    exception_class: type[WebDriverException]
    e = ErrorCode()
    error_codes = [item for item in dir(e) if not item.startswith("__")]
    for error_code in error_codes:
        error_info = getattr(ErrorCode, error_code)
        if isinstance(error_info, list) and status in error_info:
            exception_class = getattr(ExceptionMapping, error_code, WebDriverException)
            break
    else:
        exception_class = WebDriverException

    if not value:
        value = response["value"]
    if isinstance(value, str):
        raise exception_class(value)
    if message == "" and "message" in value:
        message = value["message"]

    screen = None # type: ignore[assignment]
    if "screen" in value:

```

```

        screen = value["screen"]

    stacktrace = None
    st_value = value.get("stackTrace") or value.get("stacktrace")
    if st_value:
        if isinstance(st_value, str):
            stacktrace = st_value.split("\n")
        else:
            stacktrace = []
            try:
                for frame in st_value:
                    line = frame.get("lineNumber", "")
                    file = frame.get("fileName", "<anonymous>")
                    if line:
                        file = f"{file}:{line}"
                    meth = frame.get("methodName", "<anonymous>")
                    if "className" in frame:
                        meth = f"{frame['className']}.{meth}"
                    msg = "    at %s (%s)"
                    msg = msg % (meth, file)
                    stacktrace.append(msg)
            except TypeError:
                pass
    if exception_class == UnexpectedAlertPresentException:
        alert_text = None
        if "data" in value:
            alert_text = value["data"].get("text")
        elif "alert" in value:
            alert_text = value["alert"].get("text")
        raise exception_class(message, screen, stacktrace, alert_text) # type: ignore[call-arg]
> raise exception_class(message, screen, stacktrace)
E selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: Kafete
E Stacktrace:
E RemoteError@chrome://remote/content/shared/RemoteError.sys.mjs:8:8
E WebDriverError@chrome://remote/content/shared/webdriver/Errors.sys.mjs:199:5
E NoSuchElementException@chrome://remote/content/shared/webdriver/Errors.sys.mjs:552:5
E dom.find/</<@chrome://remote/content/shared/DOM.sys.mjs:136:16

../../../../.pyenv/versions/3.13.3/envs/devenvone/lib/python3.13/site-packages/selenium/webdriver/remot

```

1.7.4 Scenario & Integration Tests

1.7.4.1 SCEN_CAFE_001: Test the full lifecycle of cafeteria management at the model level.

Attribute	Value
Test Case ID	SCEN_CAFE_001
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_complete_cafeteria_lifecycle

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Execute create, update, and delete functions on the Cafeteria model.	All model methods execute without error and reflect correct state changes in the database.	As Expected	Pass

1.7.4.2 SCEN_DB_001: Verify multiple data integrity constraints across the database schema.

Attribute	Value
Test Case ID	SCEN_DB_001
Version	1.0
Tester	Automation
Execution Time	0.5670s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_referential_integrity_workflow

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Test AppUser email uniqueness constraint.	The database prevents duplicate emails.	As Expected	Pass
2	Test that a Dish in use by a menu cannot be deleted.	The deletion operation fails as expected.	As Expected	Pass
3	Perform cleanup in the correct order to respect foreign key constraints.	All entities are deleted successfully without integrity errors.	As Expected	Pass

1.7.4.3 SCEN_INT_001: Test the full system integration by combining model and API tests in a realistic sequence.

Attribute	Value
Test Case ID	SCEN_INT_001
Version	1.0
Tester	Automation
Execution Time	0.8710s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_full_system_integration

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Phase 1: System Setup (create users, cafeterias, dishes).	All base data is created via model methods.	As Expected	Pass
2	Phase 2: Menu Management (create menus and items).	Menus are created successfully.	As Expected	Pass
3	Phase 3 & 4: Order Processing and Management.	Orders are created and updated successfully.	As Expected	Pass
4	Phase 5: Data Management (update existing data).	Updates are successful.	As Expected	Pass
5	Phase 6: Security Verification (check unauthenticated access).	Access to a protected endpoint is denied.	As Expected	Pass
6	Phase 7: System Cleanup.	All created data is successfully deleted in the correct order.	As Expected	Pass

1.7.4.4 SCEN_MENU_001: Test the full menu and dish lifecycle, including referential integrity.

Attribute	Value
Test Case ID	SCEN_MENU_001
Version	1.0
Tester	Automation
Execution Time	0.3020s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_complete_menu_creation_workflow

Prerequisites: None

Test Scenario: Create all related entities (Cafeteria, Dish, Menu, MenuItem), then verify that an in-use dish cannot be deleted. Finally, clean up all entities in the correct order.

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Create a Cafeteria, Dish, DailyMenu, and DailyMenuItem.	All entities are created successfully.	As Expected	Pass
2	Attempt to delete the Dish while it is linked to the DailyMenuItem.	The deletion must fail, returning False.	As Expected	Pass

Step #	Step Details	Expected Results	Actual Results	Status
3	Delete the DailyMenuItem first, then delete the Dish.	Both deletions must now succeed.	As Expected	Pass

1.7.4.5 SCEN_ORDER_001: Test the complete order processing workflow at the model level.

Attribute	Value
Test Case ID	SCEN_ORDER_001
Version	1.0
Tester	Automation
Execution Time	0.7170s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_complete_order_workflow

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Execute model functions for creating users, dishes, menus, reservations, and order items in sequence.	All model methods execute without error, simulating a full order process.	As Expected	Pass

1.7.4.6 SCEN_SEC_001: Verify that a sample of protected API endpoints reject unauthenticated access.

Attribute	Value
Test Case ID	SCEN_SEC_001
Version	1.0
Tester	Automation
Execution Time	0.2970s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_unauthenticated_access_security[endpoint_subset0]

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Call a subset of protected API endpoints without a login session.	All calls must be rejected with a 401 or 403 status code.	As Expected	Pass

1.7.4.7 SCEN_USER_001: Test the full lifecycle of user management from creation to deletion at the model level.

Attribute	Value
Test Case ID	SCEN_USER_001
Version	1.0
Tester	Automation
Execution Time	0.6890s
Date Tested	08-Jul-2025
Final Status	Passed
Test Function	test_complete_user_lifecycle

Prerequisites: None

Test Scenario: N/A

Test Steps:

Step #	Step Details	Expected Results	Actual Results	Status
1	Execute create, update, and delete functions on the AppUser model, including constraint checks.	All model methods execute without error and reflect correct state changes in the database.	As Expected	Pass

1.7.5 Undocumented Test Cases

The following test cases were executed but have no metadata in the registry. Consider documenting them.

Test Function	Status	Time
test_api_unauthenticated_access_is_denied[DELETE__cafeteria_1_(Supprimer une caf\xe9teria)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[DELETE__daily-menu-item_1_(Supprimer un item de menu)]	Passed	0.2970s
test_api_unauthenticated_access_is_denied[DELETE__daily-menu_1_(Supprimer un menu)]	Passed	0.2980s
test_api_unauthenticated_access_is_denied[DELETE__dish_1_(Supprimer un plat)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[DELETE__order-item_1_(Supprimer un item de commande)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[DELETE__user_1_(Supprimer un utilisateur)]	Passed	0.3050s
test_api_unauthenticated_access_is_denied[GET__cafeteria_1_(R\xe9cup\xe9r une caf\xe9teria par ID)]	Passed	0.2940s

Test Function	Status	Time
test_api_unauthenticated_access_is_denied[GET__cafeteria__(Lister les caf\%t\%rias)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[GET__daily-menu-item_by-menu_id(Lister les items d'un menu)]	Passed	0.2980s
test_api_unauthenticated_access_is_denied[GET__daily-menu_1_(R\%cup\%order un menu par ID)]	Passed	0.2940s
test_api_unauthenticated_access_is_denied[GET__daily-menu__(Lister tous les menus)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[GET__daily-menu_by-cafeteria_id(R\%cup\%order un menu par caf\%t\%ria)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[GET__dish_1_(R\%cup\%dish un plat par ID)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[GET__dish__(Lister les plats)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[GET__order-item_1_(R\%cup\%order un item de commande par ID)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[GET__order-item__(Lister les items de commande)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[GET__reservations_1_(R\%cup\%order une r\%servation par ID)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[GET__reservations__(Lister les r\%servations)]	Passed	0.3010s
test_api_unauthenticated_access_is_denied[GET__user_1_(R\%cup\%dish un utilisateur par ID)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[GET__user__(Liste tous les utilisateurs)]	Passed	0.2940s
test_api_unauthenticated_access_is_denied[POST__cafeteria__(Cr\%er une caf\%t\%ria)]	Passed	0.2940s
test_api_unauthenticated_access_is_denied[POST__daily-menu-item__(Ajouter un item \%0 un menu)]	Passed	0.2970s
test_api_unauthenticated_access_is_denied[POST__daily-menu__(Cr\%er un menu)]	Passed	0.2940s
test_api_unauthenticated_access_is_denied[POST__dish__(Cr\%er un plat)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[POST__reservations__(Cr\%er une r\%servation)]	Passed	0.2980s
test_api_unauthenticated_access_is_denied[POST__user__(Cr\%er un utilisateur)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[POST__user_balance_(Ajouter de l'argent au solde)]	Passed	0.2980s
test_api_unauthenticated_access_is_denied[PUT__cafeteria_1_(Modifier une caf\%t\%ria)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[PUT__daily-menu-item_1_(Modifier un item de menu)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[PUT__daily-menu_1_(Modifier un menu)]	Passed	0.2960s
test_api_unauthenticated_access_is_denied[PUT__dish_1_(Modifier un plat)]	Passed	0.2970s
test_api_unauthenticated_access_is_denied[PUT__reservations_1_cancel_(Annuler une r\%servation)]	Passed	0.2950s
test_api_unauthenticated_access_is_denied[PUT__user_1_(Modifier un utilisateur)]	Passed	0.2950s
test_app_user_get_all_dicts	Passed	0.3820s
test_app_user_get_all_dicts	Passed	0.3830s
test_cafeteria_get_all_dicts	Passed	0.2970s
test_create_cafeteria	Passed	0.2950s

Test Function	Status	Time
test_create_dish	Passed	0.2950s
test_create_menu	Passed	0.2960s
test_create_menu_item	Passed	0.2980s
test_create_order_item	Passed	0.3420s
test_create_reservation	Passed	0.3390s
test_create_user	Passed	0.4290s
test_delete_cafeteria	Passed	0.2960s
test_delete_dish	Passed	0.2960s
test_delete_menu	Passed	0.2980s
test_delete_menu_item	Passed	0.2960s
test_delete_order_item	Passed	0.3440s
test_delete_reservation	Passed	0.3500s
test_delete_user	Passed	0.3400s
test_dish_delete_fails_if_in_use_by_menu_item	Passed	0.3600s
test_get_all_dishes_as_dicts	Passed	0.2960s
test_unique_email_constraint	Passed	0.3870s
test_update_cafeteria	Passed	0.2960s
test_update_from_dict	Passed	0.2940s
test_update_menu	Passed	0.3010s
test_update_menu_item	Passed	0.2960s
test_update_order_item	Passed	0.3410s
test_update_reservation	Passed	0.3420s
test_update_user	Passed	0.3390s

1.8 Annex B: Manual Test Execution Details

1.8.0.1 Manual Test Case: API-SEC-001

Attribute	Value
Test Case ID	API-SEC-001
Title	Verify that a standard user receives a 403 Forbidden error when attempting to access admin-only API routes.
Description	A non-admin user attempts to list all users via the API, which should be a protected action.
Tester	Ishan Baichoo
Date Tested	2025-07-10

Prerequisites: 1. A standard user is authenticated via API client.

Test Steps: | Step # | Action | Expected Result | Actual Result | Status | | :— | :—————- | :—————- | :—————- | :— || 1 | Authenticate as 'student1@example.com'. | Authentication successful. | As Expected | Pass || 2 | Send GET request to /api/v1/user/ | Server returns HTTP 403 Forbidden. | Server returned HTTP 404 Not Found. | Fail |

Final Result: FAILED

Notes/Comments: QA Tester's Log

1.8.0.2 Manual Test Case: E2E-ORDER-001

Attribute	Value
Test Case ID	E2E-ORDER-001

Attribute	Value
Title	An end-to-end test simulating a student logging in, selecting a meal, adding it to the cart, and placing the order.
Description	A student successfully orders a meal and verifies it in their order history.
Tester	Gabriel Aumasson-Leduc
Date Tested	

Prerequisites: 1. Application is running. 1. Default student user and menus exist.

Test Steps: | Step # | Action | Expected Result | Actual Result | Status | | :— | :—————- | :—————- | :—————- | :— | | 1 | Login as student. | Redirect to dashboard. | | | 2 | Add an item to the cart. | Cart updates correctly. | | | 3 | Click 'Place Order'. | Order success message appears. | | |

Final Result: IN PROGRESS

Notes/Comments: QA Tester's Log

1.8.0.3 Manual Test Case: MANUAL-TC-001

Attribute	Value
Test Case ID	MANUAL-TC-001
Title	Verify that when a user adds funds, the UI provides clear feedback and all balance indicators update correctly without a full page reload.
Description	Verify that when a user adds funds, the UI provides clear feedback and all balance indicators update correctly without a full page reload.
Tester	Clément De Simon
Date Tested	2025-07-08

Prerequisites: 1. User is logged in as 'student1@example.com'. 1. Application is running in a modern web browser.

Test Steps: | Step # | Action | Expected Result | Actual Result | Status | | :— | :—————- | :—————- | :—————- | :— | | 1 | Navigate to the 'Top Up Balance' page. | The 'Account Balance' page is displayed. | As Expected | Pass | | 2 | Enter '25.50' into the amount field and click 'Add Money'. | A success message appears and the 'Current Balance' on the page updates. | As Expected | Pass | | 3 | Observe the header of the application. | The balance in the top-right corner updates instantly without a page refresh. | As Expected | Pass | | 4 | Navigate to the main 'Dashboard' page. | The balance in the header remains at the new, updated value. | As Expected | Pass |

Final Result: PASSED

Notes/Comments: QA Tester's Log