Sakura Sushi - Test Plan

1. Scope & Goals

Verify that the Sakura Sushi web app meets functional and nonfunctional requirements:

- Functional: guest reservation requests, menu search, admin CRUD, report generation

- Nonfunctional: security (authN/authZ), validation, basic performance on typical data size.

2. Test Types & Justification

- Unit tests (xUnit) - Validate core business rules in isolation (e.g., reservation cannot be in the past, menu price cannot be negative).

Justification: prevents regression in critical domain logic

- Manual Integration/system tests - Exercise controller + EF Core + Identity end-to-end (guest Reserve flot, admin CRUD, admin Report).

Justification: confirms the app behaves correctly through the full stack.

- User acceptance testing - Validate that the Owner/Staff roles can perform required tasks and that public users see only public features.

Justification: ensures the app meets customer/user needs.

3. Environments

* Local dev on Windows 11, .NET 8, SQLite
* Browser: Chrome latest version
* Seed users: ‘[owner@sakura.local](mailto:owner@sakura.local)’ / ‘ChangeMe1!’, ‘[staff@sakura.local](mailto:staff@sakura.local)’ / ‘ChangeMe1!’

4. Test Data

* Seeder inserts multiple MenuItems (Nigiri/Sashimi?roll) and sample Reservations
* Additional records created during CRUD tests.

5. Unit Test Cases (xUnit)

| ID | Name | Preconditions | Steps | Expected |
| --- | --- | --- | --- | --- |
| UT-01 | Create reservation succeeds | None | Call `Reservation.Create("Test",2, future, SeatType.SushiBar, "555-555-5555")` | Object returned, Name=”Test”, PartySize=2, Date=future, SeatType=SushiBar, Phone=”555-555-5555” |
| UT-02 | Create reservation with past date throws | None | Call ‘Reservation.Create(“Test”,2,past, SeatType.Table, “555-555-5555”)’ | ‘ArgumentException’ thrown |
| UT-03 | Update reservation changes values | Reservation created | Call ‘Update(name, size, future, seat, phone)’ | Properties update |
| UT-04 | MenuItem does not allow negative price | Nigiri created | Call ‘UpdateDetails(..., price: -1, …)’ | ‘ArgumentOutOfRangeException’ thrown |

6. Manual Integration / System Tests

| ID | Flow | Role | Steps | Expected |
| --- | --- | --- | --- | --- |
| IT-01 | Guest reservation request | Public | Navigate to Reserve -> Enter valid details -> submit request | Thank you page should display |
| IT-02 | Menu search | Public | Navigate to Menu -> Enter “salmon” in the search bar and press enter | Table shows multiple results with name/description/price |
| IT-03 | Admin reservations CRUD | Owner/Staff | Login -> Navigate to Admin -> Manage Reservations ->Create/Edit/Delete | Operations Succeed, Validation errors shown if details are invalid |
| IT-04 | Admin menu CRUD | Owner/Staff | Login -> Navigate to Admin -> Manage Menu -> Create/Edit/Delete | Operations Succeed, menu item shows up under menu, validation errors shown if details are invalid |
| IT-05 | Reservations Report | Owner/Staff | Login -> Navigate to Admin -> Reservations Report -> Enter date range -> Run | Table appears with title and generated timestamp |
| IT-06 | Security (nav + auth) | Public/Staff | Look for Admin link in the nav bar | Link is hidden when logged out, link is shown when logged in as Owner/Staff |

7. Defect Management Workflow

* Log defect with steps, expected vs actual, and severity
* Root cause analysis (trace through controller, domain, EF)
* Fix in a feature branch, add/extend unit test that reproduces the issue
* Pull request & review, merge to ‘working’, test in the app
* Retest & Close when passing

8. Completion/Exit Criteria

* 100% pass for UT-01 - UT-04
* All integration flows IT-01 - IT-06 pass
* No open High/Critical defects

9. Test Results

10. Changes Resulting From Tests

* Added enum dropdown for SeatType to fix invalid input on Reservation Create
* Adjusted Reports controller to materialized and order in memor to handle SQLite limitation
* Added server-side validation in ViewModels and domin ‘UpdateDetails()’ to reject negative prices