Black-box Test Design

Revenue Simulator

The Revenue Simulator calculates expected revenue, cost, and profit for a list of products over a given number of days.
The simulation is influenced by random daily sales, controlled optionally by a seed value.

Equivalence Partitioning

Products Input

- Valid – Non-empty product list
  - List contains one or more products
  - Each product has:
    - name
    - quantity
    - price_per_unit
    - selling_price
- Valid – Empty product list
  - No products provided
  - System should return a summary with zero values and empty details

Product Stock (quantity)

- Valid – Positive stock
  - Quantity > 0
  - Sales may occur depending on simulation
- Valid – Zero stock
  - Quantity = 0
  - No sales should occur
- Invalid – Negative stock
  - Not supported (would indicate invalid input)
  -

Days Parameter

- Valid
  - Positive integers (1, 7, 30, 365)
- Invalid
  - 0

    - Negative integers
    - Non-integer values (float, string, null)

Seed Parameter

- Valid
    - Positive integer
    - None (seed not provided)
- Invalid
    - Negative integers
    - Non-integer values (float, string, list, object)

Price Relationship (Decision-based Partition)

- Selling price > buying price
    - Positive profit expected
- Selling price < buying price
    - Negative profit expected
- Selling price = buying price
    - Zero profit expected

Boundary Value Analysis

Stock Boundaries

Boundary	**Description**	**Expected result**
0	No stock available	Sold units = 0
1	Minimal stock	Sales <= 1
Very large stock	10.000 units	Sales remain within stock

Days Boundaries

Boundary	**Description**	**Expected result**

1	Minimum valid days	Simulation runs
365	Upper realistic limit	Simulation runs
0	Invalid	ValueError
-1	invalid	ValuError

Seed Boundaries

Boundary	**Description**	**Expected result**
None	Seed not provided	Simulation runs
1	Small positive integer	Detrministic results
Larger integer	Hisg seed value	Determistic result
0	Invalid	ValueErrror
Negative	invalid	ValueError

generate_random_sales() Boundaries

Boundary	**Description**	**Expected result**
Stock = 0	No inventory	Always returns 0
Stock = 1	Minimal inventory	Result = 0 or 1
Large stock	High inventory	Result <= stock

Decision Tables

Decision Table 1: Days + Seed Validation

Rule	**Days valid**	**Seed valid**	**Expected outcome**
R1	Yes	Yes	Simulation runs

R2	Yes	Not provided	Simulation runs
R3	Yes	No	ValueError
R4	No	Yes	ValueError
R5	No	No	ValueError

Decision Table 2: Profit Outcome

Rule	Buy price	Sell price	Expected profit
R1	<	>	Positive
R2	>	<	Negative
R3	=	=	Zero

Inventory Spending Calculator

The Inventory Spending Calculator computes the total amount spent on inventory for a given month and year.
It provides a category-wise breakdown of spending and identifies the category with the highest cost contribution.

The function accepts:

- A list of orders
- A target year
- A target month

And each order must contain a date, quantity, cost, and category.

Equivalence Partitioning

Year Parameter

- Valid
  - Integer between from 2000 and up to 2100
- Invalid
  - Non-integer values (string, float, null)
  - Integer < 2000
  - Integer > 2100

Month Parameter

- Valid
  - Integer between 1 and 12
- Invalid
  - Non-integer values
  - Integer < 1
  - Integer > 12

Orders List

- Valid
  - Empty list (no orders)
  - List of valid order objects
- Invalid
  - Orders with invalid or missing required fields

Order Date

- **Valid**
  - datetime object
- Invalid
  - String
  - Null
  - Any non-datetime type

Quantity

- **Valid**
  - Integer or float >= 0
- **Invalid**
  - Negative numbers
  - Non-numeric values (string, object)

Cost

- Valid
  - Integer or float >= 0
- Invalid
  - Negative numbers
  - Non-numeric values

Category

- **Valid**
  - Any string value
  - Missing category (defaults to "Unknown")

Boundary Value Analysis

Year Boundaries

Boundary	**Description**	**Expected result**
2000	Minimum allowed year	Accepted
2100	Maximun allowed year	Accepted
1999	Below lower boundary	ValueError
2101	Above upper boundary	ValueError

Month Boundaries

Boundary	**Description**	**Expected result**
1	Janurary (min)	Accepted
12	December (max)	Accepted
0	Below range	ValueError
13	Above range	ValueError

Quantity & Cost Boundaries

Boundary	Description	Expected result
0	Zero quantity or cost	Spend = 0
Large value	Very high quantity	Spend calculated correctly
negative	Invalid	ValueError

Orders List Boundary

Case	Description	Expected result
Emty list	No orders	Zero totals, no changes

Decision Tables

Decision Table 1: Order Inclusion by Date

Rule	Order year	Order month	Incluced in calculation
R1	Matches	Matches	Yes
R2	Matches	Different	No
R3	Different	Matches	No
R4	Different	Different	No

Decision Table 2: Quantity + Cost Validation

Rule	Quantity valid	Cost valid	Expected outcome
R1	yes	Yes	Included in calculation
R2	No	yes	ValueError
R3	Yes	No	ValueError
R4	No	No	ValueError

Decision Table 3: Category Cost Driver

Rule	Category Spend	Highest Cost Driver
R1	Single category	Only that category
R2	Multiple categories	The category with max spend
R3	No orders	None

Products Module

Scope

The following black-box test design covers the products module, specifically the functionality for:

- Retrieving products
- Adding a new product
- Updating an existing product
- Deleting a product

Equivalence Partitioning

Get All Products (get_all_products)

Input	Equivalence	Expected output
Database contains valid product rows	Valid	List of product objects returned
Database contains no products	Valid	Empty list returned

Covered by tests:

- test_get_all_products_returns_list
- test_get_all_products_empty_result

Insert New Product (insert_new_product)

Product Fields and Equivalence Classes

Field	**Valid Equivalence class**	**Invalid equivalence class**

Name	Non-empty string	Empty string
UOM	Positive integer	Non-integer or negativ
Price per unit	Positive number	Non-numeric
Selling price	Provided	Invalid numeric value
Quantity	Integer between 0 and 1000	Integer <0 or >1000

Covered by tests:

- test_insert_new_product_success
- test_insert_new_product_type_casting
- test_insert_product_invalid_quantity_boundaries

Update Product (update_product)

Input	**Equivalence**	**Expected output**
Existing product ID with valid fields	Valid	Product updated
Non existing product ID	Valid	0 rows affected
Quantity within allowed range	Valid	Updated succeeds
Quantity < 0 or > 1000	Invalid	Error raised

Covered by tests:

- test_update_product_success
- test_update_product_no_match
- test_update_product_rejects_large_quantity

Delete Product (delete_product)

Product id	Equivalence	Expected output
Existing product ID	Valid	1 row deleted
Non-existing product ID	Valid	0 rows deleted

Covered by tests:

- test_delete_product_existing
- test_delete_product_non_existing

Boundary Value Analyses

Quantity Field (Insert & Update)

The quantity field has explicit boundaries:

Value	Boundary type	Expected result
0	Lower valid boundary	Accepted
1000	Upper valid boundary	Accepted
-1	Just below lower boundary	Rejected
1001	Just above upper boundary	Rejected

Covered by tests:

- test_insert_product_invalid_quantity_boundaries
- test_update_product_rejects_large_quantity

Decision Tables

Insert Product (Quantity Validation and Commit Behavior)

Quantity valid?	**DB insert executed**	**Commit executed**	**result**
yes	Yes	Yes	Product inserted
No (<0 or >1000)	No	No	Error raised

Covered by test:

- test_insert_product_does_not_commit_on_invalid_quantity

Delete Product (Do the product exist)

Product exists	**Rows afftected**	**result**
Yes	1	Success
No	0	Nothing happens

Covered by tests:

- test_delete_product_existing
- test_delete_product_non_existing

Update Product (Do the product exist)

Product exists	**Quantity valid**	**Rows updated**	**Result**
Yes	**yes**	**1**	**Update successful**
No	**Yes**	**0**	**No update**
Any	**No**	**0**	**Error raised**

Covered by tests:

- test_update_product_success

- test_update_product_no_match
- test_update_product_rejects_large_quantity

Black-box Test Design

Revenue Simulator

The Revenue Simulator calculates expected revenue, cost, and profit for a list of products over a given number of days.
The simulation is influenced by random daily sales, controlled optionally by a seed value.

Equivalence Partitioning

Products Input

- Valid – Non-empty product list
  - List contains one or more products
  - Each product has:
    - name
    - quantity
    - price_per_unit
    - selling_price
- Valid – Empty product list
  - No products provided
  - System should return a summary with zero values and empty details

Product Stock (quantity)

- Valid – Positive stock
  - Quantity > 0
  - Sales may occur depending on simulation
- Valid – Zero stock
  - Quantity = 0
  - No sales should occur
- Invalid – Negative stock
  - Not supported (would indicate invalid input)
  -

Days Parameter

- Valid
  - Positive integers (1, 7, 30, 365)
- Invalid
  - 0
  - Negative integers
  - Non-integer values (float, string, null)

Seed Parameter

- Valid
  - Positive integer
  - None (seed not provided)
- Invalid
  - Negative integers
  - Non-integer values (float, string, list, object)

Price Relationship (Decision-based Partition)

- Selling price > buying price
  - Positive profit expected
- Selling price < buying price
  - Negative profit expected
- Selling price = buying price
  - Zero profit expected

Boundary Value Analysis

Stock Boundaries

Boundary	**Description**	**Expected result**
0	No stock available	Sold units = 0
1	Minimal stock	Sales <= 1
Very large stock	10.000 units	Sales remain within stock

Days Boundaries

Boundary	Description	Expected result
1	Minimum valid days	Simulation runs
365	Upper realistic limit	Simulation runs
0	Invalid	ValueError
-1	invalid	ValuError

Seed Boundaries

Boundary	Description	Expected result
None	Seed not provided	Simulation runs
1	Small positive integer	Detrministic results
Larger integer	Hisg seed value	Determistic result
0	Invalid	ValueErrror
Negative	invalid	ValueError

generate_random_sales() Boundaries

Boundary	Description	Expected result
Stock = 0	No inventory	Always returns 0
Stock = 1	Minimal inventory	Result = 0 or 1
Large stock	High inventory	Result <= stock

Decision Tables

Decision Table 1: Days + Seed Validation

Rule	Days valid	Seed valid	Expected outcome
R1	Yes	Yes	Simulation runs
R2	Yes	Not provided	Simulation runs
R3	Yes	No	ValueError
R4	No	Yes	ValueError
R5	No	No	ValueError

Decision Table 2: Profit Outcome

Rule	Buy price	Sell price	Expected profit
R1	<	>	Positive
R2	>	<	Negative
R3	=	=	Zero

Inventory Spending Calculator

The Inventory Spending Calculator computes the total amount spent on inventory for a given month and year.
It provides a category-wise breakdown of spending and identifies the category with the highest cost contribution.

The function accepts:

- A list of orders
- A target year
- A target month

And each order must contain a date, quantity, cost, and category.

Equivalence Partitioning Analysis

Year Parameter

- Valid
  - Integer between from 2000 and up to 2100
- Invalid
  - Non-integer values (string, float, null)
  - Integer < 2000
  - Integer > 2100

Month Parameter

- Valid
  - Integer between 1 and 12
- Invalid
  - Non-integer values
  - Integer < 1
  - Integer > 12

Orders List

- Valid
  - Empty list (no orders)
  - List of valid order objects
- Invalid
  - Orders with invalid or missing required fields

Order Date

- **Valid**
  - datetime object
- Invalid
  - String
  - Null
  - Any non-datetime type

Quantity

- **Valid**

    - Integer or float >= 0
- **Invalid**
    - Negative numbers
    - Non-numeric values (string, object)

Cost

- Valid
    - Integer or float >= 0
- Invalid
    - Negative numbers
    - Non-numeric values

Category

- **Valid**
    - Any string value
    - Missing category (defaults to "Unknown")

Boundary Value Analysis

Year Boundaries

Boundary	Description	Expected result
2000	Minimum allowed year	Accepted
2100	Maximun allowed year	Accepted
1999	Below lower boundary	ValueError
2101	Above upper boundary	ValueError

Month Boundaries

Boundary	Description	Expected result
1	Janurary (min)	Accepted
12	December (max)	Accepted

0	Below range	ValueError
13	Above range	ValueError

Quantity & Cost Boundaries

Boundary	**Description**	**Expected result**
0	Zero quantity or cost	Spend = 0
Large value	Very high quantity	Spend calculated correctly
negative	Invalid	ValueError

Orders List Boundary

Case	**Description**	**Expected result**
Emty list	No orders	Zero totals, no changes

Decision Tables

Decision Table 1: Order Inclusion by Date

Rule	Order year	Order month	Incluced in calculation
R1	Matches	Matches	Yes
R2	Matches	Different	No
R3	Different	Matches	No
R4	Different	Different	No

Decision Table 2: Quantity + Cost Validation

Rule	Quantity valid	Cost valid	Expected outcome
R1	yes	Yes	Included in calculation
R2	No	yes	ValueError

R3	Yes	No	ValueError
R4	No	No	ValueError

Decision Table 3: Category Cost Driver

Rule	Category Spend	Highest Cost Driver
R1	Single category	Only that category
R2	Multiple categories	The category with max spend
R3	No orders	None

Products Module

Scope

The following black-box test design covers the products module, specifically the functionality for:

- Retrieving products
- Adding a new product
- Updating an existing product
- Deleting a product

Equivalence Partitioning (EP)

Get All Products (get_all_products)

Input	Equivalence	Expected output
Database contains valid product rows	Valid	List of product objects returned
Database contains no products	Valid	Empty list returned

Covered by tests:

- test_get_all_products_returns_list
- test_get_all_products_empty_result

Insert New Product (insert_new_product)

Product Fields and Equivalence Classes

Field	Valid Equivalence class	Invalid equivalence class
Name	Non-empty string	Empty string
UOM	Positive integer	Non-integer or negativ
Price per unit	Positive number	Non-numeric
Selling price	Provided	Invalid numeric value
Quantity	Integer between 0 and 1000	Integer <0 or >1000

Covered by tests:

- test_insert_new_product_success
- test_insert_new_product_type_casting
- test_insert_product_invalid_quantity_boundaries

Update Product (update_product)

Input	Equivalence	Expected output
Existing product ID with valid fields	Valid	Product updated
Non existing product ID	Valid	0 rows affected
Quantity within allowed range	Valid	Updated succeeds
Quantity < 0 or > 1000	Invalid	Error raised

Covered by tests:

- test_update_product_success
- test_update_product_no_match
- test_update_product_rejects_large_quantity

Delete Product (delete_product)

Product id	**Equivalence**	**Expected output**
Existing product ID	Valid	1 row deleted
Non-existing product ID	Valid	0 rows deleted

Covered by tests:

- test_delete_product_existing
- test_delete_product_non_existing

Boundary Value Analyses (BVA)

Quantity Field (Insert & Update)

The quantity field has explicit boundaries:

Value	**Boundary type**	**Expected result**
0	Lower valid boundary	Accepted
1000	Upper valid boundary	Accepted
-1	Just below lower boundary	Rejected
1001	Just above upper boundary	Rejected

Covered by tests:

- test_insert_product_invalid_quantity_boundaries

- test_update_product_rejects_large_quantity

Decision Tables

Insert Product (Quantity Validation and Commit Behavior)

Quantity valid?	**DB insert executed**	**Commit executed**	**result**
yes	Yes	Yes	Product inserted
No (<0 or >1000)	No	No	Error raised

Covered by test:

- test_insert_product_does_not_commit_on_invalid_quantity

Delete Product (Do the product exist)

Product exists	**Rows afftected**	**result**
Yes	1	Success
No	0	Nothing happens

Covered by tests:

- test_delete_product_existing
- test_delete_product_non_existing

Update Product (Do the product exist)

Product exists	**Quantity valid**	**Rows updated**	**Result**

Yes	yes	1	Update successful
No	Yes	0	No update
Any	No	0	Error raised

Covered by tests:

- test_update_product_success
- test_update_product_no_match
- test_update_product_rejects_large_quantity

State Transition Testing

Identify States (order system):

For the order system the realistic states are:

State ID	Name	Description
S0	**Empty**	No order exists yet
S1	**Creating**	User enters order data (not saved yet)
S2	**Saved**	Order stored in database
S3	**Updated**	Order modified and saved again
S4	**Deleted**	Order removed (terminal state)

Identify Events:

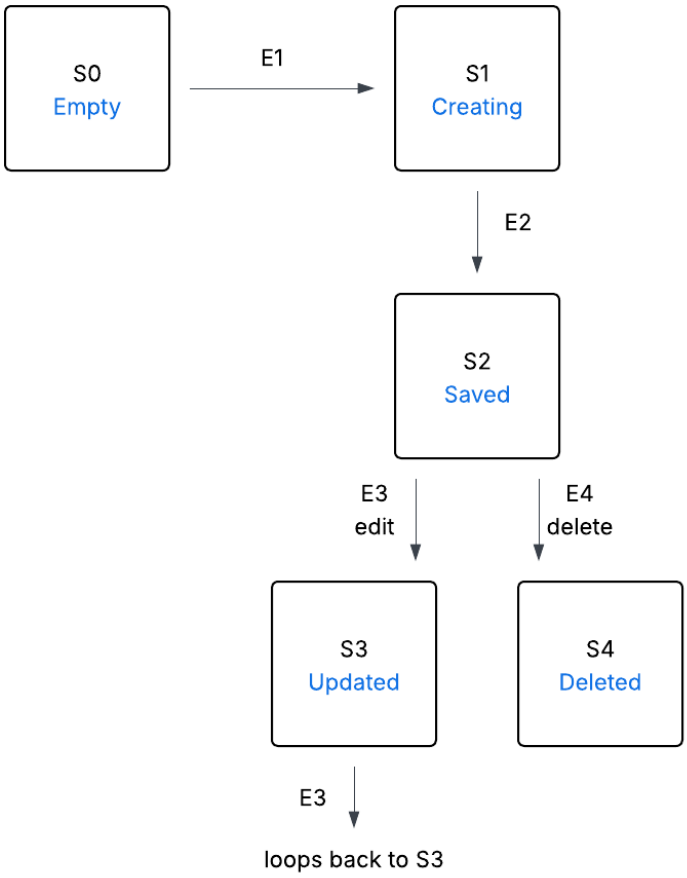
This are the actions that move the system from one state to another:

Event	Description
E1	User begins “Add Order”
E2	User submits a valid order
E3	User edits the order
E4	User deletes the order

State transition table for the Order System (Order lifecycle) :

Current State	Event Trigger	Next State	Expected Behavior
S0 Empty	**E1 Start new order**	S1 Creating	UI opens order form
S1 Creating	**E2 Submit valid order**	S2 Saved	Order written to DB, stock reduced
S1 Creating	**E4 Delete**	S0 Empty	Nothing saved, return to dashboard
S2 Saved	**E3 Edit order**	S3 Updated	Old quantities restored, new items saved
S2 Saved	**E4 Delete order**	S4 Deleted	Order deleted from DB, stock restored
S3 Updated	**E3 Edit again**	S3 Updated	Order continues updating
S3 Updated	**E4 Delete**	S4 Deleted	Order removed, stock restored
S4 Deleted	**Any event**	S4 Deleted	No further transitions allowed (terminal)

State transition diagram for order lifecycle:



Blackbox – State-based test cases

Test ID	Start State	Event	Expected Result
ST-01	S0	E1 Start new order	Order form is shown
ST-02	S1	E2 Submit valid order	Order is saved, moves to S2
ST-03	S1	E4 Delete	No DB entry, return to S0
ST-04	S2	E3 Edit	Order moves to S3 and stock restored then updated
ST-05	S2	E4 Delete	Order removed, moves to S4
ST-06	S3	E3 Edit again	Stays in S3
ST-07	S3	E4 Delete	Moves to S4
ST-08	S4	Any Event	No change — stays deleted

Identify states (Revenue simulator):

The modal goes through predictable UI/data states:

State ID	State Name	Description
RS0	**Closed**	Modal is not visible; no products loaded
RS1	**Opened**	Modal UI opened but products not loaded yet
RS2	**Loaded**	Products fetched and dropdown filled
RS3	**Simulated**	Simulation API called and results displayed

Identify events:

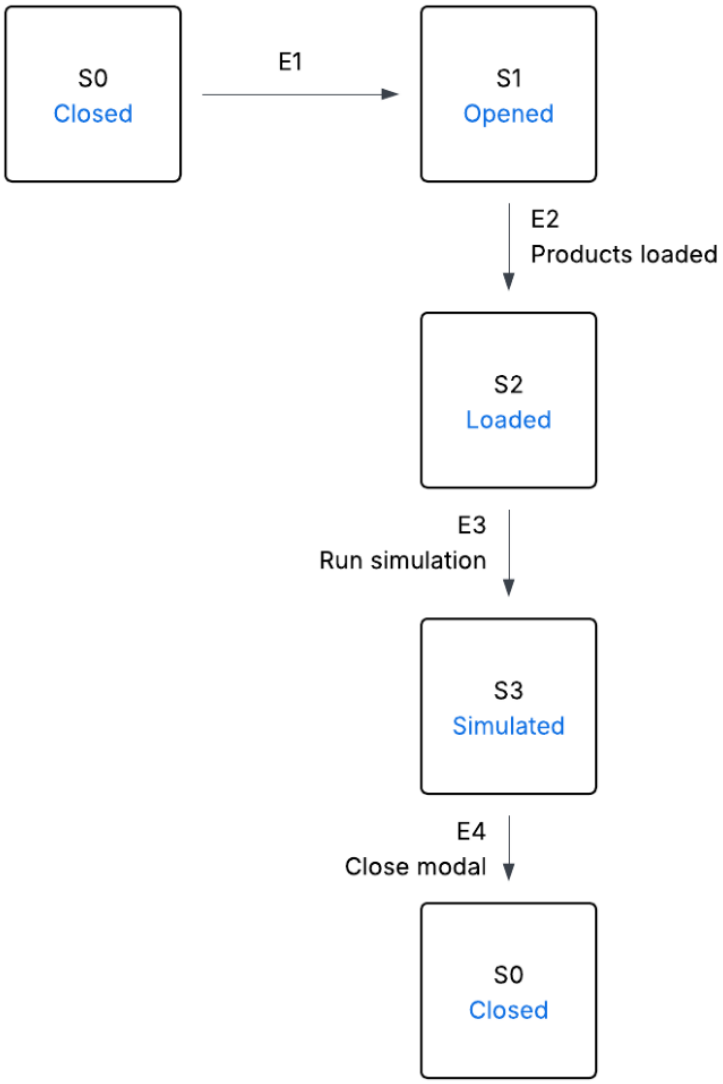
Event ID	Event Description
E1	User clicks **"Revenue Simulation"** button
E2	Products successfully fetched (/getProducts)
E3	User clicks **"Run Simulation"**
E4	Modal closed (X button or backdrop)

Transition table:

Current State	Event	Next State	Expected Behavior / Notes

RS0 Closed	E1 Open modal	**RS1 Opened**	Modal appears, UI visible
RS1 Opened	E2 Product list loaded	**RS2 Loaded**	Dropdown populates with product options
RS1 Opened	E3 Run simulation	**Stay RS1**	Should fail (no products loaded) — show error
RS2 Loaded	E3 Run simulation	**RS3 Simulated**	Revenue API runs, results rendered
RS2 Loaded	E4 Close modal	**RS0 Closed**	Modal closes; no persistence
RS3 Simulated	E3 Run simulation again	**RS3 Simulated**	Repeat simulation with updated input
RS3 Simulated	E4 Close modal	**RS0 Closed**	Modal closes; result disappears
RS0 Closed	E3 (Run simulation button cannot occur)	**Stay RS0**	No effect (button not visible)

State transition diagram for revenue simulator:



State-based test cases:

TC ID	Start State	Event	Input	Expected Result
RST-01	RS0	E1	Click "Revenue Simulation"	Modal opens - RS1
RST-02	RS1	E2	Fetch /getProducts returns success	Product dropdown lists items - RS2
RST-03	RS1	E3	Press Run Simulation too early	Error: "Please select at least one product"
RST-04	RS2	E3	Select product + Click Run	API call succeeds, results shown → RS3
RST-05	RS3	E3	Click Run again	Results update - stay RS3

RST-06	RS3	E4	Close modal	Modal disappears - RS0
RST-07	RS0	E3	Try to simulate while modal hidden	No UI element - no state change