# Aptitude test

A new project will be implemented in the company to cover the product lifecycle from product definition to product sale via e-store. The following components are the project's deliverables:

1. Product management. Module should allow procurement to:
    a. Create new products (Product id, product type, product name, provider, sale price)
    b. Update product stocks (product id, current stock)
    c. Update sale campaign (product id, discount, from date, to date). Should be allowed for all products, regardless of stock
2. E-store. Website will:
    a. Display all available products (products currently in stock)
    b. Allow customers to add (remove) products in (from) basket. Current order value will be calculated at each basket update.
    c. Allow user to create an account. For account creation, mandatory information is email, name and surname. Optional information: delivery address, card details.
    d. Allow customers to place order:
        i. As guest. As guest customer will be requested to provide the following information:
            - Name and surname
            - Email
            - Delivery address
            - Card details
        ii. As logged in user. For the logged in user details will be loaded as currently defined on the account. The user will be allowed to update delivery address and or payment details and to save modifications for future use.

    Notes:
        - Card payments are processed by external provider. The module sends out the request and receives the validation/ rejection of payment. There is no validation done by the application in terms of card details validity, only that all fields required are provided.
        - On order placement, the product stock is automatically updated by decreasing the current stock with the ordered amount. For each order details saved are: Order ID, client details, order details (product id, quantity).
3. Back-office management. The module will:
    a. Allow sales administrators to modify order details
        i. Change product and/ or quantity
        ii. Change delivery address
    b. Allow sales administrators to cancel order. On cancelation, stocks will be reverted to previous state.

Project will be delivered in as iterative waterfall and includes 2 phases.

Delivery schedule:

| Phase 1 | Phase 2 |
|---|---|
| • Module 1<br>• Module 2 (a, b, d (i))<br>• Module 3 (b) | • Module 2 (c, d(ii))<br>• Module 3 |

# Aptitude test

Question 1. What testing strategy would you apply for first phase and why?
Detail applicable test levels, scope of each test level, entry and exist criteria for each test level, any additional information you consider relevant.

The strategy for Phase 1 will be based on a **Continuous Quality** using the **Test Pyramid** model. The objective is to ensure the robustness of the system's core functionalities, focusing on automation for rapid feedback and mitigating the key business risks identified for this phase.

**1. Unit Tests**

- **Scope:** Validate the smallest units of code (functions, methods) in isolation. The focus is on the correctness of the business logic for each component.
    - **Module 1:** Business rules for product creation, field validation, stock update logic, and application of discount campaigns.
    - **Module 2:** Logic for calculating the basket total and validations for the guest checkout form fields.
    - **Module 3:** Logic for reverting stock levels upon order cancellation.
- **Owner:** Developers.
- **Entry Criteria:** The code for the feature has been developed.
- **Exit Criteria:** A minimum of 80% code coverage for new business logic and 100% pass rate in the test suite.
- **Justification:** This is the fastest and cheapest way to find defects. It ensures the application's "building blocks" are solid before integration, forming the foundation of quality and reducing the complexity of tests at higher levels.

**2. Integration Tests (API / Service Level)**

- **Scope:** Verify the communication and correct data exchange between the system's different modules and with external services. This is the most critical layer for this project.
    - **Module 2 (E-store) ↔ Module 1 (Product Mgmt):** Ensure the e-store correctly fetches the list of in-stock products and that stock is correctly updated via APIs after a sale.
    - **Module 3 (Back-office) ↔ Module 1:** Validate that an order cancellation in the back-office correctly triggers the stock reversal in the product module.
    - **Module 2 (E-store) ↔ Payment Gateway:** Mock the submission of payment requests and validate that the system correctly handles both success and failure responses from the external provider.
- **Owner:** Developers and QAs (with QAs focusing on API automation).
- **Entry Criteria:** Modules are deployed to an integrated test environment; unit tests have passed.
- **Exit Criteria:** All Phase 1 API endpoints and contracts are covered by automated tests. 100% pass rate in the integration test suite.
- **Justification:** Most of the business logic resides in the interaction between services. Testing at this layer is much faster and more stable than through the UI, allowing for precise and efficient isolation of communication failures.

**3. System & Acceptance Tests (End-to-End - E2E)**

- **Scope:** Validate the complete business flows from the end-user's perspective, ensuring the integrated system meets the requirements.
    - **Critical Success Flow:**
        1. An admin creates a product and updates its stock (Module 1).
        2. A guest customer browses the e-store, sees the available product, and adds it to the basket (Module 2).
        3. The customer fills in their details and successfully completes the purchase (Module 2).

4. Verify that the product stock has been correctly decremented (Module 1).
5. An admin cancels the order (Module 3).
6. Verify that the stock was reverted to its original value (Module 1).

- **Negative Scenarios:** Attempting to buy a product that just went out of stock; payment failure; required field validation on checkout.

- **Owner:** QAs.
- **Entry Criteria:** The system is fully integrated in a stable, production-like test environment. All integration tests are passing.
- **Exit Criteria:** 100% pass rate for test cases covering critical business flows. No open blocker or major defects.
- **Justification:** This is the final validation that ensures the system as a whole delivers business value and provides a good user experience. Automating these flows protects against future regressions in the most important functionalities.

Question 2: For one module (at choice), define the test plan to cover functionalities delivered in the first phase. Define test cases for each test level included in your test strategy.

**Module Choice and Justification**

For Phase 1, the most critical module to focus the test plan on is **Module 2: E-store**, specifically the guest checkout flow.

- **Justification:** It is the **central point of integration** for all Phase 1 features, it delivers **direct business value** (revenue generation), and it carries the **highest risk** due to its complexity and customer-facing nature.

**Test Plan: Module 2 (E-store - Phase 1)**

- **1. Test Objective:** To ensure the E-store functionalities delivered in Phase 1 (product display, basket management, and guest checkout) work as per requirements, providing a stable and correct shopping experience.
- **2. Scope:**
  - **In Scope:** Display of in-stock products, adding/removing from basket, automatic total order calculation, guest checkout flow, integration with payment gateway, and automatic stock updates.
  - **Out of Scope:** User account creation, user login, logged-in user checkout.
- **3. Applied Test Strategy:** The Test Pyramid model will be used, with a focus on automation at the Unit and Integration (API) levels, and a combination of exploratory and automated testing for critical E2E flows.
- **4. Key Risks:**
  - Incorrect price or discount calculations in the basket.
  - Stock is not updated correctly after a sale.
  - Payment gateway failures are not handled gracefully.
  - Poor usability in the checkout form leads to basket abandonment.

**Test Cases by Test Level (Module 2)**

**1. Unit Tests**

| Target Component | Test ID | Description | Expected Result |
|---|---|---|---|
| **Basket Service** | TU-01 | Add a product to an empty basket. | Product is added with quantity 1. Total value equals the product's price. |

| | | | |
|---|---|---|---|
| **Basket Service** | TU-02 | Add a product with an active discount campaign. | The total basket value reflects the price with the discount applied. |
| **Checkout Validator** | TU-03 | Attempt to validate a checkout form missing the "email" field. | Validation fails, returning a specific error for the email field. |

**2. Integration Tests (API)**

| Integration | Test ID | Description | Expected Result |
|---|---|---|---|
| **E-store ↔ Prod. Mgmt** | TI-01 | Call the GET /api/products endpoint. | The API returns status 200 (OK) and a list of products where stock > 0. |
| **E-store → Payment GW** | TI-02 | Submit an order (POST /api/orders) that should be approved by the payment gateway mock. | The system processes the success response, creates the order, and returns status 201 (Created). |
| **E-store → Prod. Mgmt** | TI-03 | After a successful order, verify the PATCH /api/products/{id}/stock endpoint was called correctly. | The stock of the sold product is correctly decremented via the API call. |

**3. System Tests (End-to-End)**

| User Flow | Test ID | Execution Steps | Expected Result |
|---|---|---|---|
| **Successful Guest Purchase** | E2E-01 | 1. Admin creates "Product A" with stock 10.<br>2. Guest customer adds 2 units of "Product A" to the basket.<br>3. Proceeds to checkout, fills in all guest data, and uses valid payment details.<br>4. Submits the order. | 1. "Order Placed Successfully" message is displayed.<br>2. The stock for "Product A" is updated to 8. |
| **Checkout with Payment Failure** | E2E-02 | 1. Guest customer adds a product to the basket.<br>2. Proceeds to checkout and uses payment details that will be rejected by the gateway.<br>3. Submits the order. | 1. "Payment Declined" message is displayed.<br>2. The product's stock is not changed.<br>3. The order is not created in the system. |

Question 3: For the same module, define the test plan and the test cases to be executed in the second phase.

**Test Plan: Module 2 (E-store - Phase 2)**

- **1. Test Objective:**
  - To ensure the new Phase 2 functionalities (account creation, login, and logged-in user checkout) work as required.
  - To validate that user data is managed securely and correctly. Additionally, to ensure Phase 1 functionalities have not been impacted (**regression testing**).

- **2. Scope:**
  - **In Scope:** User account creation, user login, logged-in user checkout with pre-filled data, updating address/payment during checkout, and persistence of user data changes. **Regression testing of all Phase 1 flows.**

  - **Out of Scope:** Full user profile management (e.g., password change, order history).
- **3. Applied Test Strategy:** Continue using the Test Pyramid. The E2E automation suite will be expanded to cover the new logged-in user flows. The full Phase 1 automated regression suite will be run on every build.
- **4. Key Risks:**
  - Security flaws in account creation or session management.
  - User data (address, payment) is not loaded or saved correctly during checkout.
  - Conflicts between guest and logged-in user sessions.
  - **Regression:** New features break the guest checkout flow.

**Test Cases by Test Level (Module 2 - Phase 2)**

**1. Unit Tests**

| Target Component | Test ID | Description | Expected Result |
|---|---|---|---|
| **Registration Service** | TU-04 | Attempt to create a user with an invalid email format. | Validation fails, returning an invalid email format error. |
| **User Service** | TU-05 | Validate the logic for updating a user's address. | The function correctly updates the address within the user's data structure. |

**2. Integration Tests (API)**

| Integration | Test ID | Description | Expected Result |
|---|---|---|---|
| **E-store → Users API** | TI-04 | Make a POST /api/users call with an email that already exists. | The API returns status 409 (Conflict) with an appropriate error message. |
| **E-store → Auth API** | TI-05 | Make a POST /api/auth/login call with correct credentials. | The API returns status 200 (OK) and an authentication token (e.g., JWT). |
| **E-store → Orders API** | TI-06 | Submit an order (POST /api/orders) with a valid auth token in the header. | The order is created successfully and is associated with the authenticated user's ID. |

**3. System Tests (End-to-End)**

| User Flow | Test ID | Execution Steps | Expected Result |
|---|---|---|---|
| **Successful Registration & Login** | E2E-03 | 1. Customer navigates to "Create Account".<br>2. Fills out the registration form with valid data.<br>3. Submits the form.<br>4. Attempts to log in with the new credentials. | 1. Account is created successfully.<br>2. Login is successful, and the user's name is displayed in the header. |
| **Purchase as Logged-In User** | E2E-04 | 1. User logs in.<br>2. Adds a product to the basket.<br>3. Proceeds to checkout.<br>4. Finishes the purchase. | 1. Checkout page is pre-filled with the user's name, email, and address.<br>2. The order is completed successfully and linked to the user's account. |
| **Update Data During Checkout** | E2E-05 | 1. Logged-in user proceeds to checkout.<br>2. Edits the delivery address and checks "Save for future | 1. The order is sent to the new address.<br>2. The next checkout pre-fills the newly saved address. |

| | | | |
|---|---|---|---|
| | | use".<br>3. Completes the order.<br>4. Starts a new checkout. | |
| **Regression: Guest Purchase** | REG-01 | 1. Without logging in, add a product to the basket.<br>2. Complete the purchase as a guest. | 1. The guest checkout flow works exactly as it did in Phase 1, with no errors. |

Question 4: Would the change in delivery methodology (from Waterfall to Agile) influence your testing strategy? If yes, how would you adapt your test strategy?

Yes, the change from Waterfall to Agile would influence the testing strategy. The reactive, phase-based approach of Waterfall would be replaced by a proactive, continuous, and fully integrated strategy.

Here is how the strategy would be adapted:

1. **Adoption of 'Shift-Left Testing': Quality as a Whole Team Responsibility**
    - **In Waterfall:** QA acts as a gatekeeper at the end of the cycle.
    - **In Agile:** Quality is everyone's responsibility. The QA is involved from the very beginning of each sprint, refining user stories and identifying risks before development starts. The focus shifts from **finding defects** to **preventing defects**.
2. **Short Test Cycles and Rapid Feedback**
    - **In Waterfall:** Testing happens in one long phase covering a large set of features.
    - **In Agile:** The strategy is broken down into short cycles aligned with sprints (e.g., 2 weeks). Tests for a user story are planned, automated, and executed within the same sprint, providing almost immediate feedback.
3. **Automation as a Central Pillar of the CI/CD Pipeline**
    - **In Waterfall:** Automation is often a separate project run during the test phase.
    - **In Agile:** Automation is essential and integrated into the CI/CD pipeline. Unit and integration tests run on every code commit, and the E2E regression suite runs automatically on every deployment to a test environment.
4. **Lightweight Documentation and Focus on Collaboration**
    - **In Waterfall:** The strategy relies on extensive, formal Test Plans and test cases.
    - **In Agile:** Documentation becomes lighter. **Acceptance Criteria** on user stories serve as the primary test basis. Constant communication and collaboration between QAs, Developers, and Product Owners replace heavy, formal documentation.

In summary, the Waterfall model treats testing as a "quality control gate" at the end of the production line, while the Agile model incorporates testing as a continuous and collaborative activity that ensures quality throughout the entire development process.

Question 5: Create the test data set necessary to test requirement 1.c. (include anything relevant for data generation - SQL statements, script or table data to be used)

To test requirement **1.c. Update sale campaign**, it is crucial to create a test data set that covers positive, negative, and boundary scenarios.

Below, I present the data structure, the test cases in a table format, and the SQL scripts to generate

# Aptitude test

this test data.

**1. Product Structure and Data (Prerequisite)**
First, we need a base of products on which the campaigns will be applied. It is essential to include products with and without stock to validate the "regardless of stock" rule.

**Table: Products**

| ProductID | ProductName | Provider | StockQuantity | SalePrice |
|---|---|---|---|---|
| **(auto)** | Laptop Pro X1 | TechCorp | 50 | 1200.00 |
| **(auto)** | Wireless Mouse | Peripherals Inc. | 200 | 25.00 |
| **(auto)** | Mechanical Keyboard | GamerGear | 0 | 80.00 |
| **(auto)** | 4K Monitor | Display Solutions | 15 | 350.50 |

**SQL Script to Insert and Update Products**
This script first inserts the products, then declares variables to hold their dynamically generated ProductIDs. Finally, it uses these variables to perform the stock updates in a safe and readable way.

sql
```
-- Clear existing data to ensure a clean state (optional)
-- DELETE FROM Products;

-- Step 1: Insert base products for the campaign tests
INSERT INTO Products (ProductType, ProductName, Provider, SalePrice) VALUES
('Electronics', 'Laptop Pro X1', 'TechCorp', 1200.00),
('Accessories', 'Wireless Mouse', 'Peripherals Inc.', 25.00),
('Accessories', 'Mechanical Keyboard', 'GamerGear', 80.00),
('Electronics', '4K Monitor', 'Display Solutions', 350.50);

-- Step 2: Declare variables and retrieve the ProductIDs for the newly inserted products.
-- This avoids hardcoding IDs and makes the script more robust.
DECLARE @LaptopID INT, @MouseID INT, @KeyboardID INT, @MonitorID INT;

SELECT @LaptopID = ProductID FROM Products WHERE ProductName = 'Laptop Pro X1';
SELECT @MouseID = ProductID FROM Products WHERE ProductName = 'Wireless Mouse';
SELECT @KeyboardID = ProductID FROM Products WHERE ProductName = 'Mechanical Keyboard';
SELECT @MonitorID = ProductID FROM Products WHERE ProductName = '4K Monitor';

-- Step 3: Update the stock for each product using the variables.
-- This is much safer and clearer than using hardcoded IDs or product names in the WHERE clause.
UPDATE Products SET StockQuantity = 50 WHERE ProductID = @LaptopID;
UPDATE Products SET StockQuantity = 200 WHERE ProductID = @MouseID;
UPDATE Products SET StockQuantity = 0 WHERE ProductID = @KeyboardID;
UPDATE Products SET StockQuantity = 15 WHERE ProductID = @MonitorID;
```

**2. Data Set for Testing Sale Campaigns**
To ensure the tests for the sale campaigns are resilient and reusable, the following test scenarios will be used. The accompanying SQL script uses the ProductID variables captured in the previous step and dynamic dates to ensure the scenarios remain valid regardless of when the tests are run.

**Test Scenarios Table**

# Aptitude test

| TestCaseID | Scenario Description | ProductID | Discount (%) | FromDate | ToDate | Justification / What is being tested |
|---|---|---|---|---|---|---|
| **CAM-01** | **Happy Path** | 101 | 15 | 2025-07-10 | 2025-07-20 | Valid campaign for a product with stock, starting today. |
| **CAM-02** | **Product without Stock** | 103 | 20 | 2025-07-11 | 2025-07-18 | Validate the rule that a campaign can be applied to products with zero stock. |
| **CAM-03** | **Boundary - Campaign Ends Today** | 102 | 10 | 2025-07-01 | 2025-07-10 | Test if the campaign is considered active until the end of today. |
| **CAM-04** | **Future Campaign** | 101 | 25 | 2025-08-01 | 2025-08-10 | Test the scheduling of a campaign that should not be active yet. |
| **CAM-05** | **Expired Campaign** | 104 | 30 | 2025-06-01 | 2025-06-30 | Test if a past campaign is not applied to the product's price today. |
| **CAM-06** | **Boundary - Maximum Discount** | 102 | 99.9 | 2025-07-15 | 2025-07-25 | Validate the application of a high discount (upper limit). |
| **CAM-07** | **One-Day Duration** | 104 | 5 | 2025-07-12 | 2025-07-12 | Test a campaign with a minimum duration (only one day). |
| **CAM-08** | **Invalid Data - Negative Discount** | 101 | -10 | 2025-07-10 | 2025-07-20 | The system must reject the creation/update with an invalid discount value. |
| **CAM-09** | **Invalid Data - ToDate before FromDate** | 102 | 10 | 2025-07-20 | 2025-07-10 | The system must display a date validation error. |
| **CAM-10** | **Invalid Data - Non-existent ProductID** | 999 | 50 | 2025-07-10 | 2025-07-20 | The system must return an error indicating that the product was not found. |

**SQL Script to Insert Campaigns**
This script creates the records in the SaleCampaigns table based on the scenarios defined above. It assumes that the variables @LaptopID, @MouseID, etc., are available in the current session.

**Note:** The syntax (GETDATE(), DATEADD) is for T-SQL (SQL Server). It may need adaptation for other database systems (e.g., NOW(), CURRENT_DATE).

# Aptitude test

sql

```
-- This script assumes the variables @LaptopID, @MouseID, @KeyboardID, and @MonitorID
-- have already been declared and populated with the correct IDs from the previous step.

-- Clear existing data to ensure a clean state (optional)
-- DELETE FROM SaleCampaigns;

-- Insert test campaigns (positive scenarios that should be inserted successfully)
-- using ID variables and dynamic dates.

-- CAM-01: Active campaign for 10 days starting today for the Laptop
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-01: Happy Path', @LaptopID, 15.00, GETDATE(), DATEADD(day, 10, GETDATE()));

-- CAM-02: Future campaign for the Keyboard (product without stock)
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-02: Product without Stock', @KeyboardID, 20.00, DATEADD(day, 1, GETDATE()),
DATEADD(day, 8, GETDATE()));

-- CAM-03: Campaign that ends today for the Mouse
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-03: Campaign Ends Today', @MouseID, 10.00, DATEADD(day, -9, GETDATE()), GETDATE());

-- CAM-04: Campaign scheduled for the future for the Laptop
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-04: Future Campaign', @LaptopID, 25.00, DATEADD(day, 22, GETDATE()), DATEADD(day, 32,
GETDATE()));

-- CAM-05: Campaign that has already expired for the Monitor
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-05: Expired Campaign', @MonitorID, 30.00, DATEADD(day, -40, GETDATE()), DATEADD(day, -
30, GETDATE()));

-- CAM-06: Campaign with a high discount for the Mouse
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-06: Maximum Discount', @MouseID, 99.90, DATEADD(day, 5, GETDATE()), DATEADD(day, 15,
GETDATE()));

-- CAM-07: One-day campaign scheduled for two days from now for the Monitor
INSERT INTO SaleCampaigns (CampaignName, ProductID, Discount, FromDate, ToDate) VALUES
('CAM-07: One-Day Duration', @MonitorID, 5.00, DATEADD(day, 2, GETDATE()), DATEADD(day, 2,
GETDATE()));
GO

/*
Final Note: The negative test cases (CAM-08, CAM-09, CAM-10) are not inserted via this SQL script.
They represent invalid data inputs that should be used in API or UI tests to verify that
the system's validation logic correctly rejects them and returns an appropriate error.
*/
```

# Aptitude test

Notes on completing the assignment:

- Attach any supporting documents required for full achievement of task.
- There are no predefined templates; the candidate can use any template that s/he considers best suited.