POST-To create a product

```
// Test Case 1: Verify Successful Product Creation
pm.test("Product created successfully", function () {
    pm.response.to.have.status(201);
    pm.response.to.be.json;

    // Ensure the response contains the expected success message
    pm.expect(pm.response.json().message).to.equal("Product created successfully");

    // Ensure the response contains product details
    pm.expect(pm.response.json().data).to.not.be.null;
});

// Test Case 2: Verify Validation Errors Handling
pm.test("Validation errors handling", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);
});

// Test Case 3: Verify BadRequest for Duplicate Product Name
pm.test("BadRequest for Duplicate Product Name", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

    // Ensure the error message includes information about the duplicate product name
    pm.expect(pm.response.json().message).to.include("Name must be unique");
});

// Test Case 4: Verify BadRequest for Non-Positive Price
pm.test("BadRequest for Non-Positive Price", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

    // Ensure the error message includes information about the non-positive price
    pm.expect(pm.response.json().errors).to.include("Price must be greater than zero");
});
```

GET-To fetch a single product

```
// Test Case 1: Verify Successful Retrieval with Dynamic ID
pm.test("Product retrieved successfully", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected success message
    pm.expect(pm.response.json().message).to.equal("Product retrieved successfully");

    // Ensure the response contains product details
    pm.expect(pm.response.json().data).to.not.be.null;
});

// Test Case 2: Verify Not Found for Invalid Product ID with Dynamic ID
pm.test("Product not found for invalid ID", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");
});

// Test Case 3: Verify Proper Handling of Errors with Dynamic ID
pm.test("Proper handling of errors", function () {
    // Assume an error scenario in the service or repository
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);
});

PUT-To update a product
------------------------

// Test Case 1: Verify Successful Product Update
pm.test("Product updated successfully", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;

    // Ensure the response contains the expected success message
    pm.expect(pm.response.json().message).to.equal("Product updated successfully");

});
```

```
// Test Case 2: Verify Not Found for Non-existing Product with Dynamic ID
pm.test("Product Not found for the given Id", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);
});

// Test Case 3: Verify Validation Errors Handling
pm.test("Validation errors handling", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);
});

// Test Case 4: Verify BadRequest for Non-Positive Price
pm.test("BadRequest for Non-Positive Price", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

    // Ensure the error message includes information about the non-positive price
    pm.expect(pm.response.json().errors).to.include("Price must be greater than zero");
});
```

POST-To add review to a product

```
// Test Case 1: Verify Successful Review Addition with Dynamic ID
pm.test("Review added successfully", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected success message
```

```
    pm.expect(pm.response.json().message).to.equal("Reviews added successfully for product " + dynamicProductId);

});

// Test Case 2: Verify Not Found for Non-existing Product with Dynamic ID
pm.test("Product Not Found for the given Id", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");

});

// Test Case 3: Verify BadRequest for Validation Errors with Dynamic ID
pm.test("BadRequest for Validation Errors", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

});

// Test Case 4: Verify BadRequest for Non-Positive Rating
pm.test("BadRequest for Non-Positive Rating", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

    // Ensure the error messages include information about the non-positive rating
    pm.expect(pm.response.json().errors).to.include("Rating must be positive");
});

POST-To add offer to a product
-------------------------------

// Test Case 1: Verify Successful Offer Addition with Dynamic ID
pm.test("Offer added successfully", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();
```

```
    // Ensure the response contains the expected success message
    pm.expect(pm.response.json().message).to.equal("Offers added successfully for product " + dynamicProductId);

});

// Test Case 2: Verify Not Found for Non-existing Product with Dynamic ID
pm.test("Product Not Found for the given Id", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");

});

// Test Case 3: Verify BadRequest for Validation Errors with Dynamic ID
pm.test("BadRequest for Validation Errors", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

});

// Test Case 4: Verify BadRequest for Past Start Date
pm.test("BadRequest for Past Start Date", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Validation errors occurred. Please check your input");

    // Ensure the errors array is present and not empty
    pm.expect(pm.response.json().errors).to.be.an('array');
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);

    // Ensure the error messages include information about the past start date
    pm.expect(pm.response.json().errors).to.include("Start date must be in the present or future");
});

GET-To fetch all products
-------------------------

// Test Case 1: Verify Successful Retrieval of Product Summaries
pm.test("Successful Retrieval of Product Summaries", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;

    // Ensure the content array is present and not empty
    pm.expect(pm.response.json().PRODUCTS).to.be.an('array');
    pm.expect(pm.response.json().PRODUCTS.length).to.be.greaterThan(0);
```

```
});

// Test Case 2: Verify Invalid Page Number
pm.test("Invalid Page Number", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Error: Invalid page number. Page number must be greater than
});

// Test Case 3: Verify BadRequest for Invalid Page and Size Values
pm.test("BadRequest for Invalid Page and Size Values", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Error: Please provide valid values for size and page.");
});
```

GET-To fetch reviews for a product

```
// Test Case 1: Verify Successful Retrieval of Product Reviews
pm.test("Successful Retrieval of Product Reviews", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;
});

// Test Case 2: Verify Not Found for Non-existing Product with Dynamic ID
pm.test("Product Not Found for the given Id", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;
});

// Test Case 3: Verify Invalid Page Number
pm.test("Invalid Page Number", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Error: Invalid page number. Page number must be greater than
});

// Test Case 4: Verify BadRequest for Invalid Page and Size Values
pm.test("BadRequest for Invalid Page and Size Values", function () {
    pm.response.to.have.status(400);
    pm.response.to.be.json;

    // Ensure the response contains the expected error message
    pm.expect(pm.response.json().message).to.equal("Error: Please provide valid values for size and page.");
});
```

DELETE-To delete a product

```
// Test Case 1: Verify Successful Deletion of Product
pm.test("Successful Deletion of Product", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;
```

```
    // Ensure the response contains the expected success message
    pm.expect(pm.response.json().message).to.equal("Product deleted successfully");
});

// Test Case 2: Verify invalid ID of Product
pm.test("Product not found for invalid ID", function () {
    pm.response.to.have.status(404);
    pm.response.to.be.json;

    // Extract product ID from the URL path
    var urlPath = pm.request.url.getPath();
    var dynamicProductId = urlPath.split('/').pop();

    // Ensure the response contains the expected error message with dynamic product ID
    pm.expect(pm.response.json().message).to.equal("Product with ID " + dynamicProductId + " not found");
});

// Test Case 3: Ensure Errors Array is Present and Not Empty
pm.test("Ensure Errors Array is Present and Not Empty in case of error", function () {
    pm.response.to.be.json;

    // Ensure the errors array is present
    pm.expect(pm.response.json()).to.have.property('errors');

    // Ensure the errors array is an array
    pm.expect(pm.response.json().errors).to.be.an('array');

    // Ensure the errors array is not empty
    pm.expect(pm.response.json().errors.length).to.be.greaterThan(0);
});
```