**NATIONAL INSTITUTE OF BUSINESS MANAGEMENT**

HIGHER NATIONAL DIPLOMA IN SOFTWARE ENGINEERING

**ENTERPRISE APPLICATION DEVELOPMENT - Ⅱ**

**E-COMMERCE WEBSITE**

**TEST CASES**

**SUBMITTED BY**

**Name Index Number**

1. W.M.M WICKRAMASINGHE MAHNDSE241F-039

2. S.R.W KUMARA MAHNDSE241F-002

3. B.W DHAMSATH MAHNDSE241F-023

4. T.P SAMARASINGHE MAHNDSE241f-030

5. T.M NANAYAKKARA MAHNDSE241f-024

6. P.P.J.R PATABENDIGE MAHNDSE241f-013

1. **ADMIN MICROSERVICE**

**1. Get All Admins**

**Request Type**: GET  
**Endpoint**: <http://localhost:8085/admins>

**Test Case**:

* **Description**: Retrieve a list of all admins.
* **Expected Response**:
  + **Status**: 200 OK
  + **Body** (example):

json

[

{

"id": "1",

"username": "admin1",

"email": "admin1@example.com"

},

{

"id": "2",

"username": "admin2",

"email": "admin2@example.com"

}

]

**2. Get Admin by ID**

**Request Type**: GET  
**Endpoint**: [http://localhost:8085/admins/{id}](http://localhost:8085/admins/%7bid%7d)

**Test Case**:

* **Description**: Retrieve details of a specific admin by ID.
* **Example ID**: 1
* **Expected Response**:
  + **Status**: 200 OK
  + **Body** (example):

json

{

"id": "1",

"username": "admin1",

"email": "admin1@example.com"

}

**3. Insert Admin into Database**

**Request Type**: POST  
**Endpoint**: <http://localhost:8085/admins>  
**Test Case**:

* **Description**: Insert a new admin into the database.
* **Request Body**:

json

{

"username": "admin",

"email": "admin1@example.com",

"password": "adminpass"

}

* **Expected Response**:
  + **Status**: 200
  + **Body** (example):

{

"id": "1",

"username": "admin",

"email": "admin1@example.com"

}

}

**4. Update Existing Admin**

**Request Type**: PUT  
**Endpoint**: http://localhost:8085/admins/{id}  
**Test Case**:

* **Description**: Update the details of an existing admin.
* **Example ID**: 1
* **Request Body**:

json

{

"username": "adminupdated",

"email": "adminupdated@example.com",

"password": "adminpassupdated"

}

* **Expected Response**:
  + **Status**: 200 OK
  + **Body** (example):

json

{

"id": "1",

"username": "adminupdated",

"email": "adminupdated@example.com"

}

**5. Delete Existing Admin**

**Request Type**: DELETE  
**Endpoint**: [http://localhost:8085/admins/{id}](http://localhost:8085/admins/%7bid%7d)  
**Test Case**:

* **Description**: Delete an admin from the database.
* **Example ID**: 1
* **Expected Response**:
  + **Status**: 204 No content

**Negative Test Cases**

**6. Get Admin by Invalid ID**

**Request Type:** GET

**Endpoint:** http://localhost:8085/admins/invalid-id

**Test Case:**

* **Description:** Attempt to retrieve an admin with an invalid or non-existent ID.
* **Expected Response:**
* **Status:** 404 Not Found
* **Body** (example):

{

"timestamp": "2024-09-12T11:55:40.621+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.admin.exception.ResourceNotFoundException: Admin not found with id: 99 (TaskThread.java:63)\r\n\tat java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Admin not found with id: 99",

"path": "/admins/99"

}

**7. DELETE Admin by Invalid ID**

**Request Type:** DELETE

**Endpoint:** http://localhost:8085/admins/invalid-id

**Test Case:**

* **Description:** Attempt to DELETE an admin with an invalid or non-existent ID.
* **Expected Response:**
* **Status:** 404 Not Found
* **Body** (example):

{

"timestamp": "2024-09-12T12:00:48.575+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.admin.exception.ResourceNotFoundException: Admin not found with id: 99 (TaskThread.java:63)\r\n\tat java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Admin not found with id: 99",

"path": "/admins/99"

}

1. **USER MICROSERVICE**

**1. Get All Users**

* **Request Type**: GET
* **Endpoint**: http://localhost:8081/users
* **Description**: Retrieve a list of all users.
* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

[

{

"id": 1,

"username": "user1",

"email": "user1@example.com"

},

{

"id": 2,

"username": "user2",

"email": "user2@example.com"

}

]

**2. Get User by ID**

* **Request Type**: GET
* **Endpoint**: http://localhost:8081/users/{id}
* **Description**: Retrieve details of a specific user by ID.
* **Example ID**: 1

**Expected Response**:

* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"username": "user1",

"email": "user1@example.com"

}

**3. Insert User into Database**

* **Request Type**: POST
* **Endpoint**: http://localhost:8081/users
* **Description**: Insert a new user into the database.
* **Request Body**:

json

{

"username": "user",

"email": "user@example.com",

"password": "password123"

}

* **Expected Response**:
* **Status**: 200 Ok
* **Response Body (example)**:

json

{

"id": 1,

"username": "user",

"email": "user@example.com"

}

**4. Update Existing User**

* **Request Type**: PUT
* **Endpoint**: http://localhost:8081/users/{id}
* **Description**: Update the details of an existing user.
* **Example ID**: 1
* **Request Body**:

json

{

"username": "updateduser",

"email": "updateduser@example.com",

"password": "updatedpassword123"

}

* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"username": "updateduser",

"email": [updateduser@example.com](mailto:updateduser@example.com) }

**5. Delete Existing User**

* **Request Type**: DELETE
* **Endpoint**: http://localhost:8081/users/{id}
* **Description**: Delete a user from the database.
* **Example ID**: 1
* **Expected Response**:
* **Status**: 204 No Content
* **No response body**

**Negative Test Cases**

**6. Get User by Invalid ID**

* **Request Type:** GET
* **Endpoint:** http://localhost:8081/users/invalid-id
* **Test Case:**
* **Description:** Attempt to retrieve a user with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:35:26.226+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.usermicroservice.exception.ResourceNotFoundException: User not found with id: 6767 java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "User not found with id: 6767",

"path": "/users/6767"

}

**7. Delete User with Invalid ID**

* **Request Type:** DELETE
* **Endpoint:** http://localhost:8081/users/invalid-id
* **Test Case:**
* **Description:** Attempt to delete a user with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:36:27.210+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.usermicroservice.exception.ResourceNotFoundException: User not found with id: 6767\r\n\tat java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "User not found with id: 6767",

"path": "/users/6767"

}

1. **CATEGORY MICROSERVICE**

**1. Get All Categories**

* **Request Type**: GET
* **Endpoint**: http://localhost:8082/categories
* **Description**: Retrieve a list of all categories.
* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

[

{

"id": 1,

"name": "Electronics",

"description": "Electronic devices and gadgets"

},

{

"id": 2,

"name": "Furniture",

"description": "Home and office furniture"

}

]

**2. Get Category by ID**

* **Request Type**: GET
* **Endpoint**: http://localhost:8082/categories/{id}
* **Description**: Retrieve details of a specific category by ID.
* **Example ID**: 1
* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"name": "Electronics",

"description": "Electronic devices and gadgets"

}

**3. Insert Category into Database**

* **Request Type**: POST
* **Endpoint**: http://localhost:8082/categories
* **Description**: Insert a new category into the database.
* **Request Body**:

json

{

"id": 1,

"name": "Electronics",

"description": "Electronic devices and gadgets"

}

* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"name": "Electronics",

"description": "Electronic devices and gadgets"

}

**4. Update Existing Category**

* **Request Type**: PUT
* **Endpoint**: http://localhost:8082/categories/{id}
* **Description**: Update the details of an existing category.
* **Example ID**: 1
* **Request Body**:

json

{

"name": "Used Electronics",

"description": "Used Electronics devices and Gadgets"

}

* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"name": "Used Electronics",

"description": "Used Electronics devices and Gadgets"

}

**5. Delete Existing Category**

* **Request Type**: DELETE
* **Endpoint**: http://localhost:8082/categories/{id}
* **Description**: Delete a category from the database.
* **Example ID**: 1
* **Expected Response**:
* **Status**: 204 No Content
* **No response body**.

**Negative Test Cases**

**6. Get Category by Invalid ID**

* **Request Type:** GET
* **Endpoint:** http://localhost:8082/categories/invalid-id
* **Test Case:**
* **Description:** Attempt to retrieve a category with an invalid or non-existent ID.
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:12:07.337+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.categorymicroservice.exception.ResourceNotFoundException: Category not found with id: 23 java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Category not found with id: 23",

"path": "/categories/23"

}

**7. Delete Category with Invalid ID**

* **Request Type:** DELETE
* **Endpoint:** http://localhost:8082/categories/invalid-id
* **Test Case:**
* **Description:** Attempt to delete a category with an invalid or non-existent ID.
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:14:51.371+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.ecommerce.categorymicroservice.exception.ResourceNotFoundException: Category not found with id: 23\r\n\tat java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Category not found with id: 23",

"path": "/categories/23"

}

1. **PRODUCT MICROSERVICE**

**1. Get All Products**

* **Request Type**: GET
* **Endpoint**: http://localhost:8080/products
* **Description**: Retrieve a list of all products.
* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

[

{

"id": 1,

"name": "OLD Biscuits",

"description": "OLD Biscuit Brand",

"price": 100,

"quantity": 100

},

{

"id": 2,

"name": "Chips",

"description": "Chips Brand",

"price": 50,

"quantity": 500

}

]

**2. Get Product by ID**

* **Request Type**: GET
* **Endpoint**: http://localhost:8080/products/{id}
* **Description**: Retrieve details of a specific product by ID.
* **Example ID**: 1
* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"name": "OLD Biscuits",

"description": "OLD Biscuit Brand",

"price": 100,

"quantity": 100

}

**3. Insert Product into Database**

* **Request Type**: POST
* **Endpoint**: http://localhost:8080/products
* **Description**: Insert a new product into the database.
* **Request Body**:

json

{

"name": "OLD Biscuits",

"description": "OLD Biscuit Brand",

"price": 100,

"quantity": 100}

* **Expected Response**:
* **Status**: 201 Created
* **Response Body (example)**:

json

{

"id": 1,

"name": "OLD Biscuits",

"description": "OLD Biscuit Brand",

"price": 100,

"quantity": 100}

**4. Update Existing Product**

* **Request Type**: PUT
* **Endpoint**: http://localhost:8080/products/{id}
* **Description**: Update the details of an existing product.
* **Example ID**: 1
* **Request Body**:

json

{

"name": "Biscuits new",

"description": "New Biscuit Brand",

"price": 200,

"quantity": 200}

* **Expected Response**:
* **Status**: 200 OK
* **Response Body (example)**:

json

{

"id": 1,

"name": "Biscuits new",

"description": "New Biscuit Brand",

"price": 200,

"quantity": 200}

**5. Delete Existing Product**

* **Request Type**: DELETE
* **Endpoint**: http://localhost:8080/products/{id}
* **Description**: Delete a product from the database.
* **Example ID**: 1
* **Expected Response**:
* **Status**: 204 No Content
* **No response body**.

**Negative Test Cases**

**6. Get Product by Invalid ID**

* **Request Type:** GET
* **Endpoint:** http://localhost:8080/products/invalid-id
* **Test Case:**
* **Description:** Attempt to retrieve a product with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:18:02.153+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.demo.demo.exception.ResourceNotFoundException: Product not found with id: java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Product not found with id: 435",

"path": "/products/435"

}

**7. Delete Product with Invalid ID**

* **Request Type:** DELETE
* **Endpoint:** http://localhost:8080/products/invalid-id
* **Test Case:**
* **Description:** Attempt to delete a product with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

json

{

"timestamp": "2024-09-12T12:18:02.335+00:00",

"status": 404,

"error": "Not Found",

"trace": "com.demo.demo.exception.ResourceNotFoundException: Product not found with id: java.base/java.lang.Thread.run(Thread.java:1583)\r\n",

"message": "Product not found with id: 435",

"path": "/products/435"}

1. **FILE STORAGE**

**1. Get All Files**

* **Request Type**: GET
* **Endpoint**: <http://localhost:8084/files/list>
* **Description**: Retrieve a list of all uploaded files.
* **Expected Response**:
  + **Status**: 200 OK
  + **Body (example)**:

json

[

{

"id": 2,

"fileName": "ss.png",

"fileUrl": <https://imageswikiwickz.s3.amazonaws.com/26f2e5d0-4586-47cb-8751-59b8aa7eb35e_ss.png> ,

"createdAt": "2024-09-09T09:48:16.175+00:00"

},

{

"id": 3,

"fileName": "Untitledggg.png",

"fileUrl": <https://imageswikiwickz.s3.amazonaws.com/8e8ac087-779f-4bad-b249-67587cd4d43b_Untitledggg.png> ,

"createdAt": "2024-09-09T10:09:41.912+00:00"

}

]

**2. Get File by ID**

* **Request Type**: GET
* **Endpoint**: [http://localhost:8084/files/{id}](http://localhost:8084/files/%7bid%7d)
* **Description**: Retrieve details of a specific file by ID.
* **Example ID**: 2
* **Expected Response**:
  + **Status**: 200 OK
  + **Body (example)**:

json

{

"id": 2,

"fileName": "ss.png",

"fileUrl": <https://imageswikiwickz.s3.amazonaws.com/26f2e5d0-4586-47cb-8751-59b8aa7eb35e_ss.png> ,

"createdAt": "2024-09-09T09:48:16.175+00:00"

}

**3. Insert File into Database**

* **Request Type**: POST
* **Endpoint**: <http://localhost:8084/files/upload>
* **Description**: Upload a new file to S3 and save its metadata in the database.
* **Request Body** (form-data):
  + **Key**: file
  + **Type**: file
  + **Value**: {choose a file} (e.g., ss.png)
* **Expected Response**:
  + **Status**: 200 OK
  + **Body (example)**:

json

{

"id": 15,

"fileName": "ss.png",

"fileUrl": <https://imageswikiwickz.s3.amazonaws.com/9bdb3ba9-c4a9-46c1-b28d-cf5ad45b49e7_ss.png> ,

"createdAt": "2024-09-12T11:09:00.464+00:00"

}

**4. Update Existing File**

* **Request Type**: PUT
* **Endpoint**: [http://localhost:8084/files/update/{id}](http://localhost:8084/files/update/%7bid%7d)
* **Description**: Update the file stored in S3 and update its metadata in the database.
* **Request Body** (form-data):
  + **Key**: file
  + **Type**: file
  + **Value**: {choose a file} (e.g., Untitled.png)
* **Expected Response**:
  + **Status**: 200 OK
  + **Body (example)**:

json

Copy code

{

"id": 15,

"fileName": "Untitled.png",

"fileUrl": <https://imageswikiwickz.s3.amazonaws.com/42793211-07b0-477b-9621-87399caa5198_Untitled.png> ,

"createdAt": "2024-09-12T11:09:00.464+00:00"

}

**5. Delete Existing File**

* **Request Type**: DELETE
* **Endpoint**: [http://localhost:8084/files/delete/{id}](http://localhost:8084/files/delete/%7bid%7d)
* **Description**: Delete a file from S3 and remove its metadata from the database.
* **Example ID**: 15
* **Expected Response**:
  + **Status**: 204 No Content
  + **No response body**

**6. Download File from S3**

* **Request Type**: GET
* **Endpoint**: [http://localhost:8084/files/download/{id}](http://localhost:8084/files/download/%7bid%7d)
* **Description**: Download the file from S3 storage.
* **Example ID**: 15
* **Expected Response**:
  + **Status**: 200 OK
  + **Body (example)**:

json

Copy code

{

"url": <https://imageswikiwickz.s3.amazonaws.com/42793211-07b0-477b-9621-87399caa5198_Untitled.png>

}

**Negative Test Cases**

**7. Get File by Invalid ID**

* **Request Type:** GET
* **Endpoint:** http://localhost:8084/files/{id}
* **Test Case:**
* **Description:** Attempt to retrieve details of a file with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 404 Not Found
* **Response Body** (example):

Json

File not found with id: 99

**8. Delete File with Invalid ID**

* **Request Type:** DELETE
* **Endpoint:** http://localhost:8084/files/delete/{id}
* **Test Case:**
* **Description:** Attempt to delete a file with an invalid or non-existent ID.
* **Example ID:** invalid-id
* **Expected Response:**
* **Status:** 500Internal Server Error
* **Response Body** (example):

json

No static resource delete/99.

1. **ADMIN MICROSERVICE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass / Fail** |
| TC01 | Verify Admins login functionality | 1. Send POST request to /admins | Body: { "username": "admin", "password": "admin123" } | Status: 200 OK, response with admin token. | As Expected | Pass |
| TC02 | Verify Admin creation of users | 1. Send POST request to /admins | Body: { "username": "newUser", "password": "pass123" } | Status: 200 Created, response with new user details. | As Expected | Pass |
| TC03 | Verify fetching data by admin ID | 1. Send PUT request to /admins/1 | User ID = 1 | Status: 200 OK, response with updated role. | As Expected | Pass |
| TC04 | Verify fetching all Admin | 1. Send GET request to /admins. | N/A | Status: 200 OK, response body with a list of users. | As Expected | Pass |
| TC05 | Verify Admin deletion by ID | 1. Send DELETE request to /admins/1 | User ID = 1 | Status: 204 No Content, admin deleted. | As Expected | Pass |
| TC06 | Verify 403 for unauthorized Admin actions | 1. Send PUT request to /admin/users/1 without Admin token. | N/A | Status: 404 Not found. | As Expected | Pass |
|  | Verify 404 for invalid admin id | Send GET request to admin/999. | Admin id = 999 | Status: 404 Not found | As Expected | pass |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass / Fail** |
| TC01 | Verify user registration functionality | 1. Send POST request to /users | Body: { "username": "user1", "password": "pass123", "email": "user1@example.com" } | Status: 200 Created, response with user details. | As Expected | Pass |
| TC02 | Get all users | 1. Send GET request to /users | NA | Status: 200 OK, response with token. | As Expected | Pass |
| TC03 | Verify fetching user details by ID | 1. Send GET request to /users/1. | User ID = 1 | Status: 200 OK, response with user details. | As Expected | Pass |
| TC04 | Verify user update profile functionality | 1. Send PUT request to /users/1. | User ID = 1, Body: { "email": "newemail@example.com" } | Status: 200 OK, response with updated user details. | As Expected | Pass |
| TC05 | Verify user deletion functionality | 1. Send DELETE request to /users/1. | User ID = 1 | Status: 204 No Content, user deleted. | As Expected | Pass |
| TC06 | Verify 404 for invalid user ID | 1. Send GET request to /users/999. | User ID = 999 | Status: 404 Not Found, error message: "User not found". | As Expected | Pass |
| TC07 | Verify 404 for invalid user ID deletion | 1.Send DELETE request to /users/999 | User ID = 999 | Status:404 Not Found. error message: ”User not found” | As Expected | Pass |

1. **USER MICROSERVICE**
2. **CATEGORY MICROSERVICE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass / Fail** |
| TC01 | Verify retrieval of all categories | 1. Send a GET request to /categories. | N/A | Status: 200 OK, response body with a list of categories. | As Expected | Pass |
| TC02 | Verify retrieval of category by ID | 1. Send a GET request to /categories/1. | Category ID = 1 | Status: 200 OK, response with details of category ID 1. | As Expected | Pass |
| TC03 | Verify 404 for invalid category ID | 1. Send a GET request to /categories/999. | Category ID = 999 | Status: 404 Not Found, error message: "Category not found with id: 999". | As Expected | Pass |
| TC04 | Verify adding a new category | 1. Send a POST request to /categories. | Body: { "name": "Electronics", "description": "Tech products" } | Status: 201 Created, response with newly created category details. | As Expected | Pass |
| TC05 | Verify updating a category by ID | 1. Send a PUT request to /categories/1. | Category ID = 1, Body: { "name": "Updated Electronics", "description": "New description" } | Status: 200 OK, response with updated category details. | As Expected | Pass |
| TC06 | Verify deleting a category by ID | 1. Send a DELETE request to /categories/1. | Category ID = 1 | Status: 204 No Content, no response body. | As Expected | Pass |
| TC07 | Verify 404 when deleting non-existent category | 1. Send a DELETE request to /categories/999. | Category ID = 999 | Status: 404 Not Found, error message: "Category not found with id: 999". | As Expected | Pass |

1. **PRODUCT MICROSERVICE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass / Fail** |
| **T01** | Verify that user can retrieve all products | 1. Send a GET request to /products. | N/A | Status: 200 OK, Response body with a list of products | As Expected | Pass |
| **T02** | Verify that user can retrieve a product by ID | 1. Send a GET request to /products/1. | Product ID = 1 | Status: 200 OK, Product with ID 1 details (JSON response) | As Expected | Pass |
| **T03** | Verify that user receives 404 for invalid product ID | 1. Send a GET request to /products/999. | Product ID = 999 | Status: 404 Not Found, Error message: "Product not found with id: 999" | As Expected | Pass |
| **T04** | Verify that user can insert a new product | 1. Send a POST request to /products. | Body: { "name": "Cookies", "description": "Chocolate", "price": 50, "quantity": 10 } | Status: 201 Created, Response body with newly created product details | As Expected | Pass |
| **T05** | Verify that user can update an existing product by ID | 1. Send a PUT request to /products/1. | Product ID = 1, Body: { "name": "Updated Cookies", "description": "New", "price": 100, "quantity": 50 } | Status: 200 OK, Response body with updated product details | As Expected | Pass |
| **T06** | Verify that user can delete a product by ID | 1. Send a DELETE request to /products/1. | Product ID = 1 | Status: 204 No Content, No response body | As Expected | Pass |
| **T07** | Verify that user receives 404 when deleting invalid ID | 1. Send a DELETE request to /products/999. | Product ID = 999 | Status: 404 Not Found, Error message: "Product not found with id: 999" | As Expected | Pass |

1. **FILE STORAGE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Test Data | Expected Results | Actual Results | Pass / Fail |
| F01 | Verify retrieving all files | 1. Send a GET request to /files/list. | N/A | Status: 200 OK, List of all uploaded files in JSON format. | As Expected | Pass |
| F02 | Verify retrieving a file by valid ID | 1. Send a GET request to /files/{id} with a valid file ID. | Valid ID: 2 | Status: 200 OK, File details in JSON format. | As Expected | Pass |
| F03 | Verify retrieving a file by invalid ID | 1. Send a GET request to /files/{id} with an invalid file ID. | Invalid ID: 99 | Status: 404 Not Found, Error message: "File not found with id: 99." | As Expected | Pass |
| F04 | Verify uploading a new file | 1. Send a POST request to /files/upload with a valid file in the body. | File: ss.png | Status: 200 OK, File details (ID, file name, file URL, createdAt) in JSON format. | As Expected | Pass |
| F05 | Verify updating an existing file | 1. Send a PUT request to /files/update/{id} with a valid file ID. | Valid ID: 15, File: Untitled.png | Status: 200 OK, Updated file details (ID, file name, file URL, createdAt) in JSON format. | As Expected | Pass |
| F06 | Verify updating a file with invalid ID | 1. Send a PUT request to /files/update/{id} with an invalid file ID. | Invalid ID: 99, File: Untitled.png | Status: 404 Not Found, Error message: "File not found with id: 99." | As Expected | Pass |
| F07 | Verify deleting an existing file | 1. Send a DELETE request to /files/delete/{id} with a valid file ID. | Valid ID: 15 | Status: 204 No Content, No response body. | As Expected | Pass |
| F08 | Verify deleting a file with invalid ID | 1. Send a DELETE request to /files/delete/{id} with an invalid file ID. | Invalid ID: 99 | Status: 500 Internal Server Error, Error message: "No static resource delete/99." | As Expected | Pass |
| F09 | Verify downloading a file by valid ID | 1. Send a GET request to /files/download/{id} with a valid file ID. | Valid ID: 15 | Status: 200 OK, File URL returned in JSON format. | As Expected | Pass |
| F10 | Verify downloading a file with invalid ID | 1. Send a GET request to /files/download/{id} with an invalid file ID. | Invalid ID: 99 | Status: 404 Not Found, Error message: "File not found with id: 99." | As Expected | Pass |