Phase 1: Library Management System Functional Testing

Project Overview:

Technology: .NET Core 7.0

Project Description: The Library Management System is a web-based application designed to manage books in a library. It utilizes Entity Framework Core for data management and connects to a SQL Server database for data storage.

Key Features in Phase 1:

Books Management: This phase focuses on the core functionality of managing books within the library. It includes features for adding, editing, and deleting books.

Upcoming Features in Phase 2:

Customer Functionality: Phase 2 of the project will introduce customer-related functionalities, such as user accounts, book borrowing, and return management.

The Library Management System is built using the latest .NET Core technology, ensuring efficiency, security, and scalability. By leveraging Entity Framework Core and SQL Server, it provides robust data management capabilities for seamless book management.

NuGet Packages

NUnit: Used as the testing framework for writing and executing test cases.

Entity Framework Core: Used to set up and manage the in-memory database for testing.

Fluent Assertions: Used to make assertions in a more readable and fluent manner.

Test Case 1: Create a Book

Test Case Description

Test Method: CreateBook_ShouldAddBookToDatabase_WhenBookIsValid

Test Scenario: This test case confirms that when a valid book entry is created using the Create method, it is correctly added to the database. It ensures the "Create" functionality works as expected.

Test Steps

Arrange: Create a new instance of the BooksAdmin class with the following properties:

BookId = 4

BookCode = "B001"

BookName = "Test Book"

BookPrice = 9.99

Act: Add the book to the in-memory database and save changes.

Assert:

Check if the retrieved book from the database is not null.

Verify that the retrieved book matches the expected book based on property values.

Expected Results

The book with the specified properties should be successfully added to the database.

The retrieved book from the database should not be null.

The retrieved book's properties should match the expected values.

Test Case 2: Edit a Book

Test Case Description

Test Method: EditBook_ShouldUpdateBookInDatabase_WhenModelStateIsValid

Test Scenario: This test case confirms that when a book's details are updated using the Edit method with valid ModelState, the book's data is correctly updated in the database. It ensures the "Edit" functionality works as expected.

Test Steps

Arrange: Create a new instance of the BooksAdmin class with the following properties:

BookId = 1

BookCode = "B001"

BookName = "Test Book"

BookPrice = 9.99

Act:

Add the book to the in-memory database and save changes.

Create a new instance of the BooksAdminController and set the context.

Call the Edit method with the book's updated details.

Assert:

Check if the edit result is not null.

Verify that the action name in the result is "Index," indicating a successful update.

Check if the book is updated in the database by comparing property values.

Expected Results

The book with the specified properties should be successfully added to the database initially.

After calling the Edit method, the edit result should not be null.

The action name in the result should be "Index."

The book's properties in the database should be updated to match the new values.

Test Case 3: Delete a Book

Test Case Description

Test Method: DeleteBook_ShouldRemoveBookFromDatabase_WhenBookExists

Test Scenario: This test case confirms that when a book is deleted using the DeleteConfirmed method, it is correctly removed from the database. It ensures the "Delete" functionality works as expected.

Test Steps

Arrange: Create a new instance of the BooksAdmin class with the following properties:

BookId = 1

BookCode = "B001"

BookName = "Test Book"

BookPrice = 9.99

Add the book to the in-memory database.

Act: Call the DeleteConfirmed method with the book's ID (1).

Assert:

Check if the delete result is not null.

Verify that the action name in the result is "Index," indicating a successful delete.

Check if the book with the specified ID (1) is null in the database.

Expected Results

The book with the specified properties should be successfully added to the database initially.

After calling the DeleteConfirmed method, the delete result should not be null.

The action name in the result should be "Index."

The book with ID 1 should be removed from the database.

This detailed test case documentation outlines the test steps, expected results, and test scenarios for each of the three test cases in phase 1 of the Library Management System Functional Testing.

Test Cases for Books Details

Test Case 1: View Book Details

Test Case Description

Test Method: Details_ShouldReturnViewWithBook_WhenIdIsValid

Test Scenario: This test case verifies that when a valid book ID is provided to the Details method, it correctly returns a View displaying the book's details.

Test Steps

Arrange:

Create a new instance of the BooksAdmin class with valid properties.

Add the book to the in-memory database.

Create a new instance of the BooksAdminController and set the context.

Act:

Call the Details method with the book's valid ID.

Assert:

Check if the result is not null, indicating a successful response.

Verify that the model in the View is of type BooksAdmin.

Ensure that the model in the View matches the expected book based on property values using FluentAssertions.

Expected Results

The Details method should return a View displaying the book's details.

The model in the View should be of type BooksAdmin.

The model in the View should match the expected book.

Test Case 2: Invalid Book ID

Test Case Description

Test Method: Details_ShouldReturnNotFound_WhenIdIsNull

Test Scenario: This test case verifies that when a null book ID is provided to the Details method, it correctly returns a "NotFound" result.

Test Steps

Arrange:

Create a new instance of the BooksAdminController and set the context.

Act:

Call the Details method with a null book ID.

Assert:

Check if the result is not null, indicating a "NotFound" response.

Expected Results

The Details method should return a "NotFound" result.

Test Case 3: Non-Existent Book ID

Test Case Description

Test Method: Details_ShouldReturnNotFound_WhenIdDoesNotExist

Test Scenario: This test case verifies that when a non-existent book ID is provided to the Details method, it correctly returns a "NotFound" result.

Test Steps

Arrange:

Create a new instance of the BooksAdminController and set the context.

Act:

Call the Details method with a book ID (99) that does not exist in the database.

Assert:

Check if the result is not null, indicating a "NotFound" response.

Expected Results

The Details method should return a "NotFound" result.

This detailed test case documentation outlines the test steps, expected results, and test scenarios for each of the three test cases in the BooksDetailsTests class.

Functional testing

Functional testing is a critical aspect of software quality assurance that focuses on evaluating the system's functionality against the specified requirements. In the context of the Library Management System project, functional testing is employed to ensure that the application's core features work as expected.

The primary goal of functional testing is to validate that each function or component of the system performs its intended tasks accurately and consistently. This type of testing assesses the system's behavior and its ability to produce the desired outcomes when users interact with it.

PHASE 2 WILL CONTINUE SOON .....