Database Testing with SQL and .NET Core - Lab Assignment

Overview

In this lab, you will learn how to implement database testing for a .NET Core application. You will create a Bookstore application with a SQL Server database backend and write comprehensive tests to ensure its reliability and correctness.

Learning Objectives

- Set up a .NET Core solution with proper project structure
- Create Entity Framework Core models and context
- Implement repository pattern for data access
- Write different types of database tests
- Use transactions for test isolation
- Implement the builder pattern for test data management
- Test stored procedures and direct SQL operations
- Evaluate database performance and security

Prerequisites

- Visual Studio 2022 or Visual Studio Code
- .NET 6.0 SDK or later
- SQL Server LocalDB or SQL Express
- Basic knowledge of C#, SQL, and Entity Framework Core

Submission Requirements

- Complete solution with all source code
- SQL scripts for database setup
- README.md file with setup instructions and explanation of your approach
- Screenshots of test results

Grading Criteria

- Functionality (40%): All tests pass and correctly verify database operations
- Code Quality (25%): Clean, maintainable code with proper separation of concerns
- Test Coverage (25%): Comprehensive testing of all database functionality
- Documentation (10%): Clear documentation and code comments

Assignment Tasks

Part 1: Project Setup

1. Create a new solution with the following projects:

- `BookstoreApp.Core` (.NET 6 Class Library)
- `BookstoreApp.Data` (.NET 6 Class Library)
- `BookstoreApp.Tests` (.NET 6 NUnit Test Project)

2. Add the following NuGet packages:

- BookstoreApp.Data:
  - Microsoft.EntityFrameworkCore.SqlServer
  - Microsoft.EntityFrameworkCore.Design
- BookstoreApp.Tests:
  - Microsoft.EntityFrameworkCore.InMemory
  - NUnit
  - NUnit3TestAdapter
  - Microsoft.NET.Test.Sdk
  - Moq
  - Dapper

3. Set up project references:

- BookstoreApp.Data → BookstoreApp.Core
- BookstoreApp.Tests → BookstoreApp.Core, BookstoreApp.Data

Part 2: Domain Model Implementation

1. In BookstoreApp.Core, create the following models:

- Author:
  - Id (int)
  - Name (string)
  - Biography (string)
  - DateOfBirth (DateTime)
- Book:
  - Id (int)
  - Title (string)
  - ISBN (string)
  - PublicationDate (DateTime)
  - Price (decimal)
  - AuthorId (int)
  - Author (navigation property)
- Customer:
  - Id (int)
  - Name (string)
  - Email (string)
  - RegistrationDate (DateTime)
- Order:

        - Id (int)
        - CustomerId (int)
        - OrderDate (DateTime)
        - TotalAmount (decimal)
        - Customer (navigation property)
        - OrderItems (collection navigation property)
    - OrderItem:
        - Id (int)
        - OrderId (int)
        - BookId (int)
        - Quantity (int)
        - Price (decimal)
        - Order (navigation property)
        - Book (navigation property)

2. Create repository interfaces in BookstoreApp.Core:
    - IAuthorRepository
    - IBookRepository
    - ICustomerRepository
    - IOrderRepository

Part 3: Data Layer Implementation

1. In BookstoreApp.Data, create the `BookstoreDbContext` class:
    - Configure entity relationships
    - Set up required DbSets
    - Implement OnModelCreating with Fluent API
2. Implement the repository interfaces:
    - AuthorRepository
    - BookRepository
    - CustomerRepository
    - OrderRepository
3. Create a SQL script (`database-setup.sql`) that includes:
    - Database schema creation
    - A stored procedure to get top selling books
    - A trigger to update book inventory on order placement

Part 4: Basic Database Tests

1. Create a `DatabaseFixture` class in BookstoreApp.Tests for test setup:
    - Implement database initialization
    - Add test data seeding

- Set up proper cleanup

2. Write tests for the Author repository:

   - Test GetById
   - Test GetAll
   - Test Create
   - Test Update
   - Test Delete

3. Write tests for the Book repository:

   - Test GetByAuthorId
   - Test searching for books by title
   - Test price range filtering

Part 5: Advanced Database Tests

1. Create test data builders:

   - Implement AuthorBuilder
   - Implement BookBuilder
   - Implement CustomerBuilder

2. Write transaction-based tests:

   - Test Order creation with OrderItems
   - Test rollback scenario for failed operations

3. Implement direct SQL tests:

   - Test stored procedure for top selling books
   - Test database schema validation
   - Test trigger functionality

Part 6: Performance and Security

1. Write performance tests:

   - Measure query execution time
   - Compare in-memory vs. SQL Server performance
   - Test stored procedure performance

2. Implement security tests:

   - Test input validation
   - Test error handling for edge cases