

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using WebApplication1.Controllers;

using WebApplication1.DB;

using WebApplication1.Models;

using Xunit;

namespace WebApplication1.Tests

{

[Collection("Sequential")]

public class BooksControllerTests

{

private ApplicationDbContext GetContext()

{

var options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

return new ApplicationDbContext(options);

}

[Fact]

public async Task Index\_ReturnsBookName()

{

using(var context = GetContext()) {

context.books.AddRange(new List<book> {

new book { name = "Book 1", description = "1", isbn = "123", path\_to\_book\_cover=""},

new book { name = "Book 2", description = "2", isbn = "321", path\_to\_book\_cover = ""}

});

context.SaveChanges();

var controller = new BooksController(context);

// Act (Действие)

var result = await controller.Index() as ViewResult;

var model = result?.Model as List<book>;

// Assert (Проверка)

Assert.NotNull(result);

Assert.NotNull(model);

Assert.Contains(model, b => b.name == "Book 1");

Assert.Contains(model, b => b.name == "Book 2");

}

}

[Fact]

public async Task Index\_ReturnsBookPath()

{

using (var context = GetContext())

{

context.books.AddRange(new List<book> {

new book { name = "Book 3", description = "3", isbn = "123", path\_to\_book\_cover="../../img/books/idiot.jpg"},

new book { name = "Book 4", description = "4", isbn = "321", path\_to\_book\_cover = "../../img/books/zemlya.jpg"}

});

context.SaveChanges();

var controller = new BooksController(context);

// Act (Действие)

var result = await controller.Index() as ViewResult;

var model = result?.Model as List<book>;

// Assert (Проверка)

Assert.NotNull(result);

Assert.NotNull(model);

Assert.Contains(model, b => b.path\_to\_book\_cover == "../../img/books/idiot.jpg");

Assert.Contains(model, b => b.path\_to\_book\_cover == "../../img/books/zemlya.jpg");

}

}

[Fact]

public void Details\_ReturnsViewWithBook()

{

using (var context = GetContext())

{

// Заполняем тестовую базу данных

var book = new book { name = "Book 5", description = "5", isbn = "3211", path\_to\_book\_cover = "" };

context.books.Add(book);

context.SaveChanges();

var controller = new BooksController(context);

var id = book.book\_id;

// Act

var result = controller.Details(id) as ViewResult;

var model = result?.Model as book;

// Assert

Assert.NotNull(result);

Assert.NotNull(model);

Assert.Equal("Book 5", model.name);

}

}

[Fact]

public void Details\_ReturnsViewWithDescription()

{

using (var context = GetContext())

{

// Заполняем тестовую базу данных

var book = new book { name = "Book 6", description = "Описание книги", isbn = "3211", path\_to\_book\_cover = "" };

context.books.Add(book);

context.SaveChanges();

var controller = new BooksController(context);

var id = book.book\_id;

// Act

var result = controller.Details(id) as ViewResult;

var model = result?.Model as book;

// Assert

Assert.NotNull(result);

Assert.NotNull(model);

Assert.Equal("Описание книги", model.description);

}

}

[Fact]

public void Details\_ReturnsViewWithIsbn()

{

using (var context = GetContext())

{

// Заполняем тестовую базу данных

var book = new book { name = "Book 7", description = "Описание книги", isbn = "3211", path\_to\_book\_cover = "" };

context.books.Add(book);

context.SaveChanges();

var controller = new BooksController(context);

var id = book.book\_id;

// Act

var result = controller.Details(id) as ViewResult;

var model = result?.Model as book;

// Assert

Assert.NotNull(result);

Assert.NotNull(model);

Assert.Equal("3211", model.isbn);

}

}

[Fact]

public void Details\_NonExistingBookId\_ReturnsNotFound()

{

// Arrange

using (var context = GetContext())

{

var controller = new BooksController(context);

// Act

var result = controller.Details(999);

// Assert

Assert.IsType<NotFoundResult>(result);

}

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using WebApplication1.Controllers;

using WebApplication1.Models;

using Xunit;

using Microsoft.AspNetCore.Http;

using WebApplication1.DB;

namespace WebApplication1.Tests

{

public class ReaderAcountControllerTests : IDisposable

{

private DbContextOptions<ApplicationDbContext> \_options;

private ApplicationDbContext \_context;

public ReaderAcountControllerTests()

{

\_options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new ApplicationDbContext(\_options);

}

[Fact]

public async Task Index\_AuthorizedUser\_ReturnsViewWithUserData()

{

// Arrange

var username = "testuser";

var user = new user { username = username, password ="123321", email = "test@example.com", role = "reader" };

\_context.Users.Add(user);

\_context.SaveChanges();

var controller = new ReaderAcountController(\_context);

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, username)

};

var identity = new ClaimsIdentity(claims, "CookieAuth");

var principal = new ClaimsPrincipal(identity);

var httpContext = new Mock<HttpContext>();

httpContext.Setup(x => x.User).Returns(principal);

controller.ControllerContext = new ControllerContext { HttpContext = httpContext.Object };

// Act

var result = controller.Index() as ViewResult;

// Assert

Assert.NotNull(result);

Assert.IsType<user>(result.Model);

var viewUser = Assert.IsType<user>(result.Model);

Assert.Equal(username, viewUser.username);

Assert.Equal("test@example.com", viewUser.email);

Assert.Equal("reader", viewUser.role);

}

public void Dispose()

{

\_context.Dispose();

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using WebApplication1.Controllers;

using WebApplication1.Models; // Изменено пространство имен

using Xunit;

using Npgsql;

using WebApplication1.DB;

using System.Data;

using Microsoft.AspNetCore.Connections;

using static WebApplication1.Tests.RegistrationControllerTests;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Primitives;

namespace WebApplication1.Tests

{

public class RegistrationControllerTests : IDisposable

{

private DbContextOptions<ApplicationDbContext> \_options;

private ApplicationDbContext \_context;

public RegistrationControllerTests()

{

\_options = new DbContextOptionsBuilder<ApplicationDbContext>()

.UseInMemoryDatabase(databaseName: "TestDb", builder => builder.EnableNullChecks(false))

.Options;

\_context = new ApplicationDbContext(\_options);

}

private Mock<IDbConnection> CreateMockConnection()

{

var mockConnection = new Mock<IDbConnection>();

mockConnection.Setup(conn => conn.Open()).Verifiable();

mockConnection.Setup(conn => conn.Close()).Verifiable();

return mockConnection;

}

[Fact]

public async Task InsertData\_ValidData\_ReturnsOk()

{

// Arrange

var mockConnection = CreateMockConnection();

var mockCommand = new Mock<IDbCommand>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

var controller = new RegistrationController();

var username = "testuser";

var email = "test@example.com";

var password = "testpass";

var formCollection = new Dictionary<string, StringValues>

{

{ "username", username },

{ "email", email },

{ "password", password }

};

var form = new FormCollection(formCollection);

var httpContext = new Mock<HttpContext>();

// Store data in the context

httpContext.SetupGet(x => x.Request.Form).Returns(form);

controller.ControllerContext = new ControllerContext { HttpContext = httpContext.Object };

// Act

var result = controller.InsertData(username, password, email) as OkObjectResult;

// Assert

Assert.NotNull(result);

Assert.Equal("Данные успешно добавлены.", result.Value);

var user = \_context.Users.FirstOrDefault(u => u.username == username);

Assert.NotNull(user);

Assert.Equal(username, user.username);

Assert.Equal(email, user.email);

Assert.Equal("читатель", user.role);

}

[Fact]

public void InsertData\_SqlInjection\_ReturnsBadRequest()

{

// Arrange

var controller = new RegistrationController();

// Act

var result = controller.InsertData("testuser", "DROP TABLE users;", "test@example.com") as BadRequestObjectResult;

// Assert

Assert.NotNull(result);

Assert.Equal("Попытка SQL-инъекции обнаружена.", result.Value);

}

[Fact]

public void InsertData\_Exception\_ReturnsBadRequest()

{

// Arrange

var controller = new RegistrationController();

// Act

var result = controller.InsertData("testuser", "testpass", "test@example.com") as BadRequestObjectResult;

// Assert

Assert.NotNull(result);

Assert.Contains("Ошибка при вставке данных", result.Value.ToString());

}

public void Dispose()

{

// Очищаем контекст и удаляем базу данных в памяти

\_context.Dispose();

}

}

}