**Задание 5. Тестирование**

**Тест-кейсы**

|  |  |
| --- | --- |
| **Название проекта** | ProjectSchool |
| **Рабочая версия** | 1.0 |
| **Имя тестирующего** | Подосенов Тихон Романович |
| **Дата(ы) теста** | 20.12.2024 |

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_UI\_1 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка заполнения полей при авторизации |
| **Краткое изложение теста** | Проверка возможности войти в аккаунт, не ввода пароль или логин |
| **Этапы теста** | 1. Открыть страницу авторизации. 2. Пропустить ввод пароля и логина. 3. Нажать кнопку "Вход". |
| **Тестовые данные** | * Логин: "рябка" * Пароль: "98765" |
| **Ожидаемый результат** | Появление сообщения под полем Логин «Введите логин» и Пароль «Введите пароль» |
| **Фактический результат** | Сообщение под полем Логин «Введите логин» и Пароль «Введите пароль» |
| **Статус** | Зачет (Приложение А. Рисунок 1) |
| **Предварительное условие** | Открыта страница авторизации |
| **Постусловие** | Появление сообщений о необходимости заполнения полей |
| **Примечания/комментарии** | - |

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_UI\_2 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Фильтрация таблицы Отзывы по полю Должность |
| **Краткое изложение теста** | Фильтрация таблицы Отзывы по полю Должность путем выбора элемента из выпадающего списка |
| **Этапы теста** | 1. Открыть страницу авторизации. 2. Войти в личный аккаунт. 3. Перейти на страницу «Отзывы». 4. Выбрать вариант из выпадающего списка Должность |
| **Тестовые данные** | * Логин: "рябка" * Пароль: "98765" * Вариант «Учитель» |
| **Ожидаемый результат** | Фильтрация таблицы Отзывы с выводом записей с должностью учитель |
| **Фактический результат** | Фильтрация таблицы Отзывы с выводом записей с должностью учитель |
| **Статус** | Зачет (Приложение А. Рисунок 2) |
| **Предварительное условие** | Открыта страница Отзывы. |
| **Постусловие** | Данные в таблице смогли пройти фильтрацию |
| **Примечания/комментарии** | - |

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_UI\_3 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Удаление записи из таблицы Расписание |
| **Краткое изложение теста** | Удаление записи из таблицы Расписание путем выбора варианта в модальном окне |
| **Этапы теста** | 1. Открыть страницу авторизации. 2. Войти в личный аккаунт под логином и паролем администратора. 3. Перейти на страницу «Расписание». 4. Нажать на кнопку «Изменить/удалить» 5. Выбрать вариант записи 6. Нажать на кнопку «Удалить» |
| **Тестовые данные** | 1. Логин: " тамаз" 2. Пароль: " 2ощы9" 3. Id: 50 |
| **Ожидаемый результат** | Исчезновение записи из таблицы |
| **Фактический результат** | Исчезновение записи из таблицы |
| **Статус** | Зачет (Приложение А. Рисунок 3, Рисунок 4) |
| **Предварительное условие** | Открыта страница «Расписание». |
| **Постусловие** | Запись из базы данных исчезла. |
| **Примечания/комментарии** | - |

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_UI\_4 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Добавление новой записи в таблицу «Расписание» |
| **Краткое изложение теста** | Добавление новой записи в таблицу «Расписание» путем выбора данных из выпадающих списков |
| **Этапы теста** | 1. Открыть страницу авторизации. 2. Войти в личный аккаунт под логином и паролем администратора. 3. Перейти на страницу «Расписание». 4. Нажать на кнопку «Добавить» 5. Выбрать варианты данных новой записи 6. Нажать на кнопку «Ок» |
| **Тестовые данные** | 1. Логин: " тамаз" 2. Пароль: " 2ощы9" 3. Класс: 5А   День недели: Пятница  Номер урока: 3  Предмет: Геометрия |
| **Ожидаемый результат** | Появление новой записи |
| **Фактический результат** | Появление новой записи |
| **Статус** | Зачет (Приложение А. Рисунок 5, Рисунок 6) |
| **Предварительное условие** | Открыта страница «Расписание». |
| **Постусловие** | Данные новой записи добавлены в базу данных. |
| **Примечания/комментарии** | - |

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_UI\_5 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Изменение данных в таблице «Расписание» |
| **Краткое изложение теста** | Изменение данных в таблице «Расписание» путем выбора необходимого id и изменение его данных. |
| **Этапы теста** | 1. Открыть страницу авторизации. 2. Войти в личный аккаунт под логином и паролем администратора. 3. Перейти на страницу «Расписание». 4. Нажать на кнопку «Изменить/удалить» 5. Выбрать id записи и изменить ее данные 6. Нажать на кнопку «Изменить» |
| **Тестовые данные** | * Логин: " тамаз" * Пароль: " 2ощы9" * Id: 56 * Класс: 5А * День недели: Понедельник * Номер урока: 4 * Предмет: Геометрия |
| **Ожидаемый результат** | Изменение данных записи в таблице |
| **Фактический результат** | Изменение данных записи в таблице |
| **Статус** | Зачет (Приложение А. Рисунок 7, Рисунок 8) |
| **Предварительное условие** | Открыта страница «Расписание». |
| **Постусловие** | Данные записи изменены в базу данных. |
| **Примечания/комментарии** | - |

Приложение А  


Рисунок 1 – предупреждение о необходимости заполнения полей

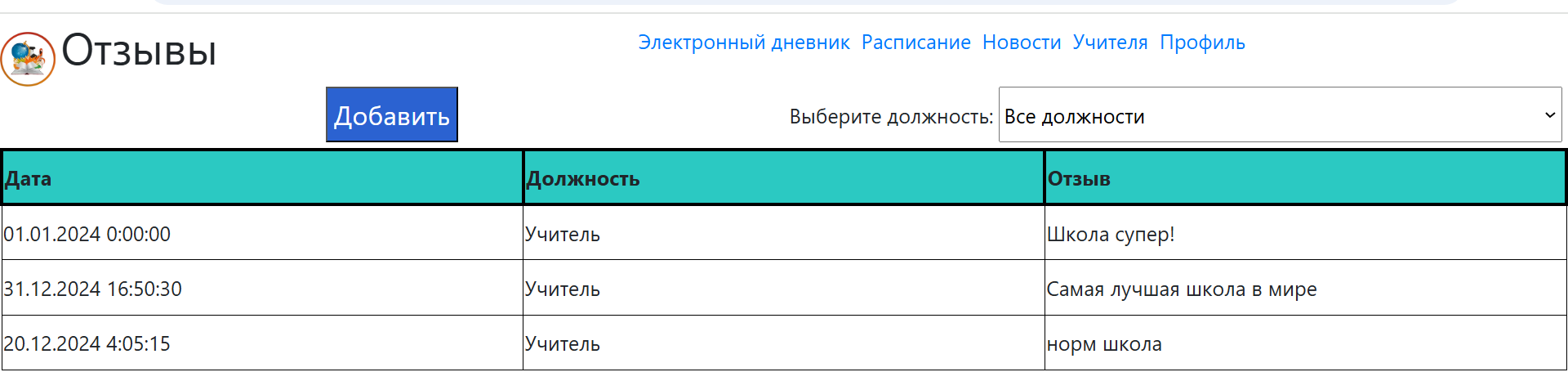


Рисунок 2 – успешная фильтрация таблицы «Отзывы»

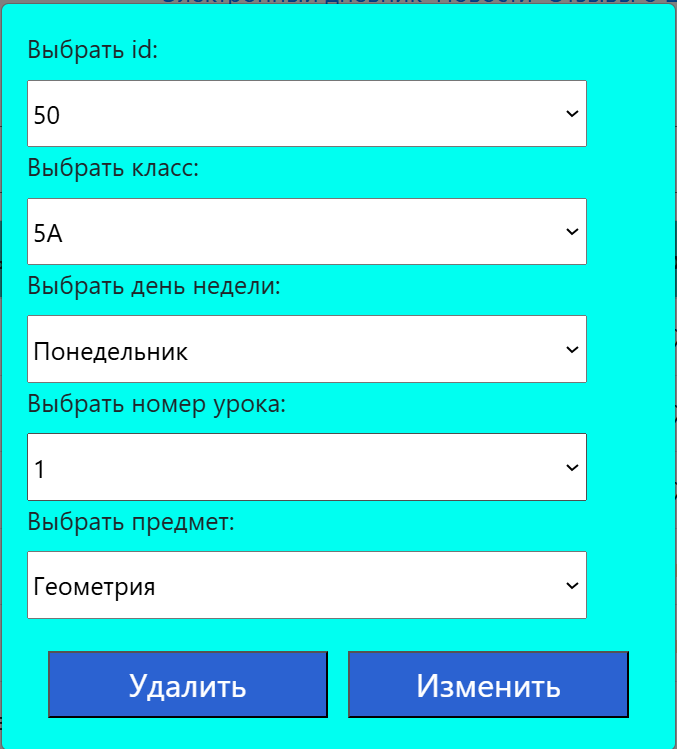


Рисунок 3 – открытие окна для удаления данных и выбор id = 50

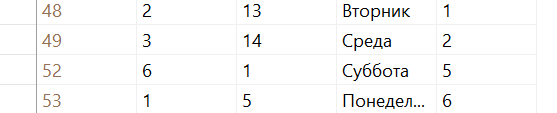


Рисунок 4 – Успешное удаление записи из базы данных

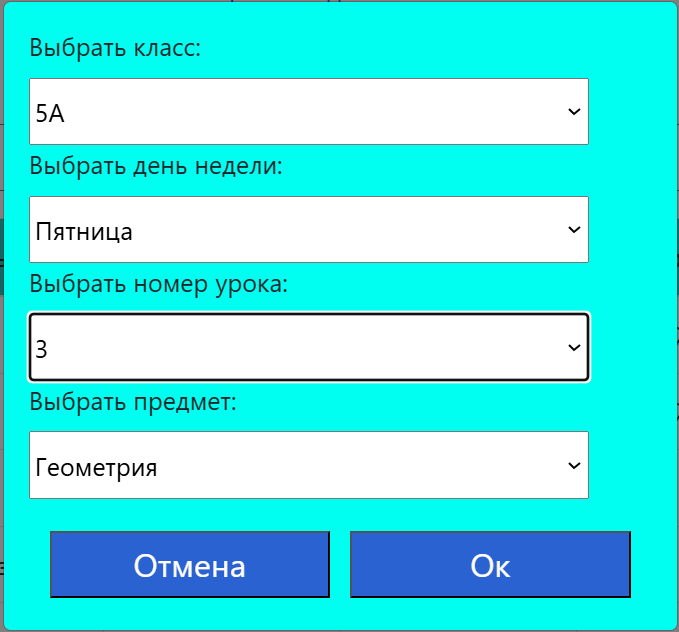


Рисунок 5 - открытие окна для добавления новых данных



Рисунок 6 – успешное добавление новой записи

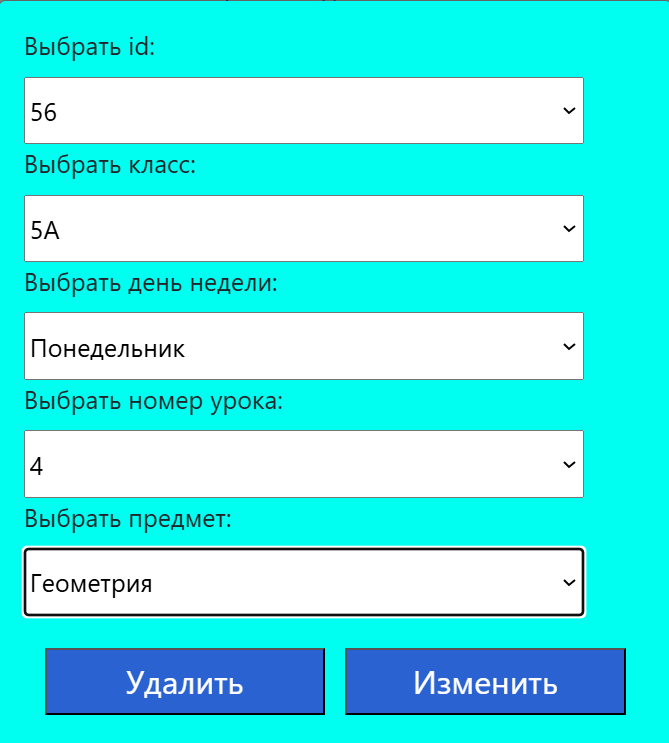


Рисунок 7 – открытие окна для изменения данных записи с id = 56

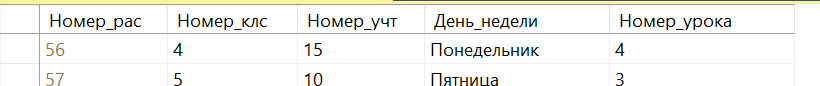


Рисунок 8 – успешное изменение данных в базе данных

**Unit-tests**

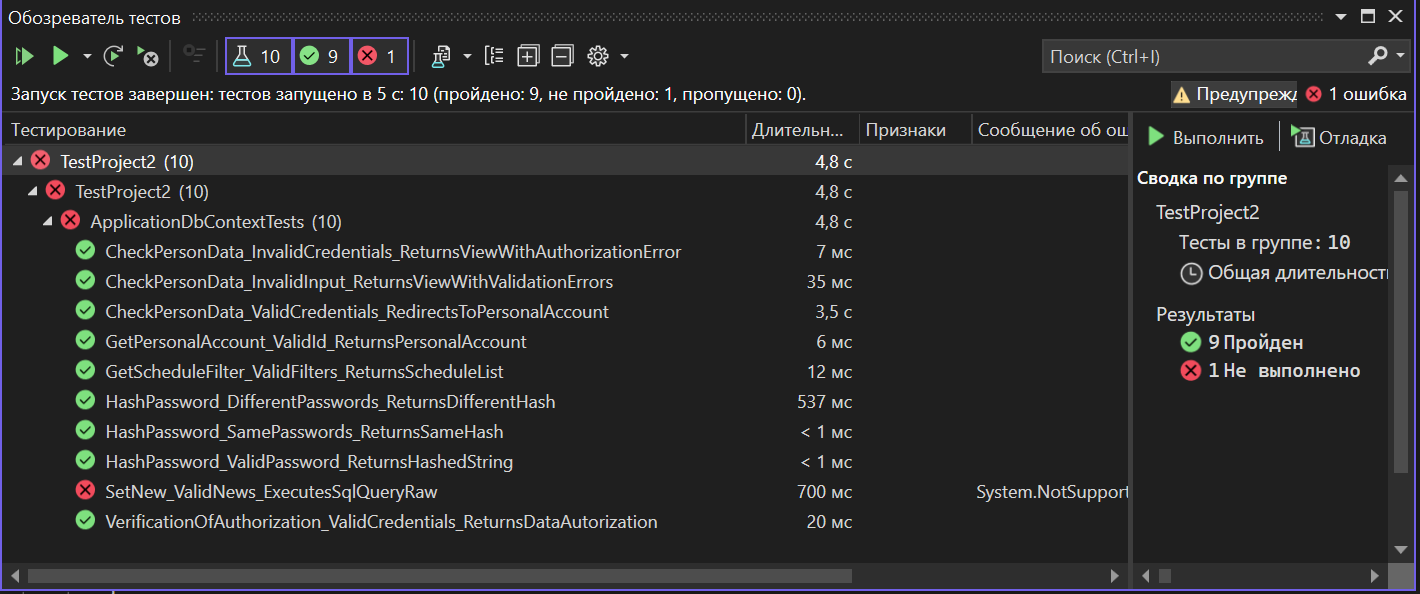
****

Рисунок 9 – результат тестов

Код класса ApplicationDbContextTests:

using Microsoft.EntityFrameworkCore;

using ProjectSchool.Models;

using Moq;

using ProjectSchool.DB;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.Data.SqlClient;

using Microsoft.AspNetCore.Mvc;

using ProjectSchool.Controllers;

using ProjectSchool;

using ProjectSchool.Tests;

namespace TestProject2

{

public class ApplicationDbContextTests

{

[Fact]

public void GetPersonalAccount\_ValidId\_ReturnsPersonalAccount()

{

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

int id\_aut = 45;

var expectedPersonalAccount = new List<PersonalAccount>

{

new PersonalAccount { Name = "Алексей", Surname = "Смирнов", Patronymic = "Викторович", Group = "5В", Number = "89123456789" }

};

sqlQueryExecutorMock.Setup(executor => executor.ExecuteQuery<PersonalAccount>(

"EXEC SelectStudent @id\_aut",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 1 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@id\_aut" &&

(int)((SqlParameter)parameters[0]).Value == id\_aut)))

.Returns(expectedPersonalAccount);

// Act

var result = dbContext.GetPersonalAccount(id\_aut);

// Assert

Assert.NotNull(result);

Assert.Single(result);

Assert.Equal("Алексей", result[0].Name);

Assert.Equal("Смирнов", result[0].Surname);

Assert.Equal("Викторович", result[0].Patronymic);

Assert.Equal("5В", result[0].Group);

Assert.Equal("89123456789", result[0].Number);

}

[Fact]

public void VerificationOfAuthorization\_ValidCredentials\_ReturnsDataAutorization()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

string login = "test\_login";

string password = "test\_password";

var expectedDataAutorization = new List<DataAutorization>

{

new DataAutorization { Id\_aut = 1, Post = "Test Post" }

};

sqlQueryExecutorMock.Setup(executor => executor.ExecuteQuery<DataAutorization>(

"EXEC VerificationOfAuthorization @login, @password",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 2 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@login" &&

(string)((SqlParameter)parameters[0]).Value == login &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@password" &&

(string)((SqlParameter)parameters[1]).Value == password)))

.Returns(expectedDataAutorization);

// Act

var result = dbContext.VerificationOfAuthorization(login, password);

// Assert

Assert.NotNull(result);

Assert.Single(result);

Assert.Equal(1, result[0].Id\_aut);

Assert.Equal("Test Post", result[0].Post);

}

[Fact]

public async Task CheckPersonData\_InvalidInput\_ReturnsViewWithValidationErrors()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

var controller = new HomeController(dbContext, new GlobalDataService());

controller.ModelState.AddModelError("Login", "Введите логин");

controller.ModelState.AddModelError("Password", "Введите пароль");

var model = new Home { Login = null, Password = null };

// Act

var result = await controller.CheckPersonData(model) as ViewResult;

// Assert

Assert.NotNull(result);

Assert.False(result.ViewData.ModelState.IsValid);

Assert.True(result.ViewData.ModelState.ContainsKey("Login"));

Assert.True(result.ViewData.ModelState.ContainsKey("Password"));

Assert.Equal("Введите логин", result.ViewData.ModelState["Login"].Errors[0].ErrorMessage);

Assert.Equal("Введите пароль", result.ViewData.ModelState["Password"].Errors[0].ErrorMessage);

}

[Fact]

public async Task CheckPersonData\_InvalidCredentials\_ReturnsViewWithAuthorizationError()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

var controller = new HomeController(dbContext, new GlobalDataService());

string login = "wrong\_login";

string password = "wrong\_password";

var model = new Home { Login = login, Password = password };

sqlQueryExecutorMock.Setup(executor => executor.ExecuteQuery<DataAutorization>(

"EXEC VerificationOfAuthorization @login, @password",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 2 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@login" &&

(string)((SqlParameter)parameters[0]).Value == login &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@password" &&

(string)((SqlParameter)parameters[1]).Value == HomeController.HashPassword(password))))

.Returns(new List<DataAutorization>());

// Act

var result = await controller.CheckPersonData(model) as ViewResult;

// Assert

Assert.NotNull(result);

Assert.False(result.ViewData.ModelState.IsValid);

Assert.True(result.ViewData.ModelState.ContainsKey(string.Empty));

Assert.Equal("Неправильные данные", result.ViewData.ModelState[string.Empty].Errors[0].ErrorMessage);

}

[Fact]

public async Task CheckPersonData\_ValidCredentials\_RedirectsToPersonalAccount()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

var globalDataService = new GlobalDataService();

var controller = new HomeController(dbContext, globalDataService);

string login = "test\_login";

string password = "test\_password";

var model = new Home { Login = login, Password = password };

var expectedDataAutorization = new List<DataAutorization>

{

new DataAutorization { Id\_aut = 1, Post = "Ученик" }

};

sqlQueryExecutorMock.Setup(executor => executor.ExecuteQuery<DataAutorization>(

"EXEC VerificationOfAuthorization @login, @password",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 2 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@login" &&

(string)((SqlParameter)parameters[0]).Value == login &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@password" &&

(string)((SqlParameter)parameters[1]).Value == HomeController.HashPassword(password))))

.Returns(expectedDataAutorization);

// Act

var result = await controller.CheckPersonData(model) as RedirectToActionResult;

// Assert

Assert.NotNull(result);

Assert.Equal("Index", result.ActionName);

Assert.Equal("PersonalAccount", result.ControllerName);

Assert.Equal(1, globalDataService.Id\_aut);

Assert.Equal("Ученик", globalDataService.Post);

}

[Fact]

public void GetScheduleFilter\_ValidFilters\_ReturnsScheduleList()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new ApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object);

string subject = "Алгебра";

string group = "10A";

string day = "Понедельник";

var expectedSchedule = new List<Schedule>

{

new Schedule { Id = 1, Group = group, Day = day, Subject = subject, NumberSubject = 1 }

};

sqlQueryExecutorMock.Setup(executor => executor.ExecuteQuery<Schedule>(

"EXEC SelectScheduleFilter @subject, @group, @day",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 3 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@subject" &&

(string)((SqlParameter)parameters[0]).Value == subject &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@group" &&

(string)((SqlParameter)parameters[1]).Value == group &&

parameters[2] is SqlParameter &&

((SqlParameter)parameters[2]).ParameterName == "@day" &&

(string)((SqlParameter)parameters[2]).Value == day)))

.Returns(expectedSchedule);

// Act

var result = dbContext.GetScheduleFilter(subject, group, day);

// Assert

Assert.NotNull(result);

Assert.Single(result);

Assert.Equal(1, result[0].Id);

Assert.Equal(group, result[0].Group);

Assert.Equal(day, result[0].Day);

Assert.Equal(subject, result[0].Subject);

Assert.Equal(1, result[0].NumberSubject);

}

[Fact]

public void SetNew\_ValidNews\_ExecutesSqlQueryRaw()

{

// Arrange

var sqlQueryExecutorMock = new Mock<ISqlQueryExecutor>();

var databaseMock = new Mock<DatabaseFacade>(new Mock<DbContext>().Object);

var dbContextOptions = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestDb").Options;

var dbContext = new TestApplicationDbContext(dbContextOptions, sqlQueryExecutorMock.Object)

{

Database = databaseMock.Object

};

var model = new NewsAdministration { News\_ = "Test News", Date = DateTime.Now };

databaseMock.Setup(db => db.ExecuteSqlRaw(

"EXEC AddNew @new = @new, @date = @date",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 2 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@new" &&

(string)((SqlParameter)parameters[0]).Value == model.News\_ &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@date" &&

((DateTime)((SqlParameter)parameters[1]).Value).Date == model.Date.Date)));

// Act

dbContext.SetNew(model);

// Assert

databaseMock.Verify(db => db.ExecuteSqlRaw(

"EXEC AddNew @new = @new, @date = @date",

It.Is<object[]>(parameters => parameters != null &&

parameters.Length == 2 &&

parameters[0] is SqlParameter &&

((SqlParameter)parameters[0]).ParameterName == "@new" &&

(string)((SqlParameter)parameters[0]).Value == model.News\_ &&

parameters[1] is SqlParameter &&

((SqlParameter)parameters[1]).ParameterName == "@date" &&

((DateTime)((SqlParameter)parameters[1]).Value).Date == model.Date.Date)), Times.Once);

}

[Fact]

public void HashPassword\_ValidPassword\_ReturnsHashedString()

{

// Arrange

string password = "test\_password";

// Act

string hashedPassword = HomeController.HashPassword(password);

// Assert

Assert.NotNull(hashedPassword);

Assert.NotEmpty(hashedPassword);

//Проверка на то что метод работает, и что строка не пустая.

}

[Fact]

public void HashPassword\_SamePasswords\_ReturnsSameHash()

{

// Arrange

string password = "test\_password";

string password2 = "test\_password";

// Act

string hashedPassword1 = HomeController.HashPassword(password);

string hashedPassword2 = HomeController.HashPassword(password2);

// Assert

Assert.Equal(hashedPassword1, hashedPassword2);

}

[Fact]

public void HashPassword\_DifferentPasswords\_ReturnsDifferentHash()

{

// Arrange

string password = "test\_password";

string password2 = "test\_password2";

// Act

string hashedPassword1 = HomeController.HashPassword(password);

string hashedPassword2 = HomeController.HashPassword(password2);

// Assert

Assert.NotEqual(hashedPassword1, hashedPassword2);

}

}

}

Код интерфейса ISqlQueryExecutor:

using System.Collections.Generic;

using ProjectSchool.Models;

namespace ProjectSchool.DB

{

public interface ISqlQueryExecutor

{

List<T> ExecuteQuery<T>(string sql, params object[] parameters);

}

}

Код класса TestApplicationDbContext:

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using ProjectSchool.DB;

using ProjectSchool.DB;

namespace ProjectSchool.Tests

{

public class TestApplicationDbContext : ApplicationDbContext

{

public TestApplicationDbContext(DbContextOptions<ApplicationDbContext> options, ISqlQueryExecutor sqlQueryExecutor) : base(options, sqlQueryExecutor)

{

}

public new DatabaseFacade Database { get; set; }

}

}