**4 Тестирование и отладка программного продукта**

**4.1 Основные концепции тестирования и отладки программного подхода**

Тестирование программного обеспечения – процесс исследования, испытания [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов [13].

Отладка программного обеспечения – этап разработки [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), на котором обнаруживают, локализуют и устраняют ошибки [14].

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса.

Для исправления ошибки необходимо определить ее причину, т.е. найти оператор или фрагмент, содержащие ошибку.

Тестирование и отладка программ различаются тем, что при тестировании устанавливается факт ошибки, а отладка выявляет причину и предлагает способы ее устранения.

Для локализации ошибки и ее исправление чаще всего используются:

* разбиение программы на части и отладка по частям,
* точки останова,
* пошаговое выполнение программы,
* выполнение программы до оператора, указанного курсором,
* просмотр значений ряда переменных на различных шагах выполнения программы и т.д.

**4.2 Направления тестирования**

Выделяют следующие направления тестирования:

* тестирование «web»-ориентированных приложений,
* тестирование «desktop»-приложений,
* тестирование мобильных приложений.

«Web»-ориентированное приложение – приложение, которое размещается на удаленных серверах. Взаимодействие с пользователем осуществляется с помощью браузера, а обмен данных происходит по сети.

«Desktop»-приложение – приложение, которое размещается на компьютере пользователя. Оно не требует для работы подключение к интернету, взаимодействует с пользователем посредством стандартного интерфейса, зависит от используемой операционной системы и требует установку на каждый компьютер пользователя, желающего работать с данным приложением.

Мобильное приложение – это специально разработанное приложение под конкретную мобильную платформу («iOS», «Android», «Windows Phone»).

**4.3 Уровни тестирования**

Выделяют следующие уровни тестирования [15]:

* модульное тестирование («Unit Testing»),
* интеграционное тестирование («Integration Testing»),
* системное тестирование («System Testing»),
* приемочное тестирование («Acceptance Testing»).

Модульное тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по отдельности (модули программ, объекты, классы, функции и т.д.). Данное тестирование выполняют разработчики ПО. Обычно, модульное тестирование проводится при поддержке сред разработки, таких как фреймворки для модульного тестирования, или используя инструменты для отладки.

Фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Интеграционное тестирование – тестирование, при котором отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Системное тестирование – это тестирование ПО, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований к системе в целом.

Приемочное тестирование – тестирование, на котором проверяется соответствие системы требованиям. Обычно проводится по тест-кейсам, которые были разработаны для данного ПО.

**4.4 Автоматизированное тестирование**

Автоматизированное тестирование – это метод тестирования программного обеспечения, который выполняется с использованием специальных программных средств, которые, в свою очередь необходимы для выполнения набора тестовых примеров [16].

Для тестирования программного продукта использовался метод юнит-тестов.

Юнит-тесты позволяют быстро и автоматически протестировать отдельные компоненты приложения независимо от остальной его части.

Для создания юнит-тестов выбираются небольшие участки кода, которые нужно протестировать. Тестируемый участок, как правило, меньше класса. В большинстве случаев тестируется отдельный метод.

Для тестирования дипломного проекта был выбран фреймворк «xUnit.net» – наиболее популярный фреймворк тестирования для работы с «.NET Core» и «ASP.NET Core».

Для тестирования в решение был добавлен и настроен новый проект типа «xUnit Test Project (.NET Core)», который использует фреймворк «xUnit». В проект добавлена папка с классами, производящими тестирование контроллеров административной панели.

Код методов, отвечающих за тестирование вывода всех пользователей представлен в листинге 4.1.

**Листинг 4.1 – Тестирование вывода всех пользователей**

[Fact]

public void Index\_ReturnsView()

{

var result = \_controller.Index();

Assert.IsType<ViewResult>(result);

}

[Fact]

public void Index\_ReturnsExactNumberOfObjects()

{

var result = \_controller.Index();

var viewResult = Assert.IsType<ViewResult>(result);

var objects = Assert.IsType<List<User>>(viewResult.Model);

Assert.Equal(2, objects.Count);

}

[Fact]

public void Create\_ReturnsView()

{

var result = \_controller.Create();

Assert.IsType<ViewResult>(result);

}

Код методов, отвечающих за тестирование добавления новых пользователей представлен в листинге 4.2.

**Листинг 4.2 – Тестирование добавления нового пользователя**

[Fact]

public void Create\_InvalidModelState\_ReturnsView() {

\_controller.ModelState.AddModelError("Email", "Email уже используется");

var obj = new User

{

UserId = null, UserName = "TestName", Email = "test@mail.com", Password = "testPassword", RoleId = 2

};

var result = \_controller.Create(obj, null);

var viewResult = Assert.IsType<ViewResult>(result);

var testObj = Assert.IsType<User>(viewResult.Model);

Assert.Equal(obj.UserName, testObj.UserName);

Assert.Equal(obj.Email, testObj.Email);

}

[Fact]

public void Create\_RedirectsToIndex()

{

var obj = new User

{

UserId = null, UserName = "TestName", Email = "test@mail.com", Password = "testPassword", RoleId = 2

};

var result = \_controller.Create(obj, null);

var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);

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

}

Код методов, отвечающих за тестирование изменения существующего пользователя представлен в листинге В.3 приложения В.

Код методов, отвечающих за тестирование удаления пользователей представлен в листинге 4.3.

**Листинг 4.3 – Тестирование удаления пользователей**

[Fact]

public void Delete\_ReturnsView()

{

var result = \_controller.ConfirmDelete(2);

Assert.IsType<ViewResult>(result);

}

[Fact]

public void Delete\_InvalidId\_ReturnsNotFoundResult()

{

var result = \_controller.ConfirmDelete(-1);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public void Delete\_ActionExecuted\_RedirectsToIndexAction()

{

var result = \_controller.Delete(2);

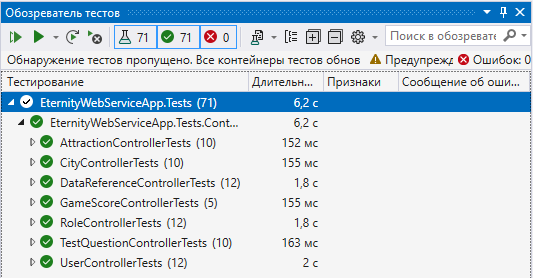
var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);

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

}

Методы других контроллеров протестированы схожим образом.

Тестирование программного продукта прошло успешно, ошибок не было выявлено. Результаты тестирование представлены на рисунке 4.1.



**Рисунок 4.1 – Результаты тестирования**

Протестировав программный продукт можно сделать вывод, что все необходимые функции функционируют и отвечают необходимым требованиям.