1.2. Выбор средств тестирования

Тестирование информационных систем - процесс установления соответствия между реальным и ожидаемым поведением системы в целом или отдельных ее функций. Тестирование ИС является важным этапом производства ПО, направленным на детальное исследование программного кода в выявление ошибок в работе системы.

В рамках данного курсового проекта необходимо провести функциональное тестирование, которое проверяет функции и методы системы на корректность их выполнения. Целью функционального тестирования является проверка системы на соответствие функциональным требованиям.

Преимущества функционального тестирования:

* Имитирует реальное пользование системой
* Выявляет ключевые системные ошибки, «узкие места»
* Позволяет сэкономить за счет своевременного обнаружения и исправления дефектов

Функциональное тестирование будет проводится с использованием метода «белого ящика». Метод «белого ящика» - это подход к тестированию, при котором у тестировщика есть доступ к исходному коду и окружению. Как правило, такое тестирование осуществляется разработчиками, так как предполагает знание программирования. Кроме прохождения по шагам тестового сценария и выявления ошибок, тестирование методом «белого ящика» предполагает просмотр программного кода с целью выявления причин неполадок.

Преимущества тестирования «белого ящика»:

* Оптимизация кода путём поиска скрытых ошибок
* Возможность автоматизировать тестовые случаи
* Тестирование можно начать на раннем этапе разработки, даже если графический интерфейс ещё недоступен

Недостатки тестирования «белого ящика»:

* Требуется профессиональная подготовка, понимание программирования и реализации проекта
* Сложность тестирования может сильно возрасти с увеличением размера приложения, от нескольких минут для небольшого проектов до недель для более крупных

Первый недостаток нерелевантен, так как тестирование проекта будет проводиться его же разработчиком. Второй недостаток незначителен, так как размер тестируемого приложения небольшой.

Таким образом было принято решение использовать метод «белого ящика».

Для успешного тестирования необходимо составить тестовые случаи или тест-кейсы. При тестировании методом «белого ящика» тест-кейсы создаются преимущественно на основе кода тестируемых функций. После этого определяют наборы тестовых данных.

Данный проект написан на языке программирования C# с использованием фреймворка ASP.NET Core для серверной части приложения и платформы ASP.NET Core Blazor, использующей C# для создания графических веб-интерфейсов, для клиентской части. Для тестирования будет использоваться фреймворк для написания модульных тестов NUnit, доступный для всех языков платформы .NET.

Модульное тестирование - это тип тестирования программного обеспечения (ПО), при котором тестируются отдельные компоненты или модули ПО. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом.

Модульное тестирование обладает рядом преимуществ:

* Позволяет тестировать отдельные части проекта, не дожидаясь завершения других
* Позволяет проводить рефакторинг кода на любой стадии проекта

Также у него есть недостатки:

* Не выявит всех ошибок, из-за невозможности оценить все пути выполнения даже в простейших программах
* Не сможет выявить ошибки интеграции или ошибки системного уровня, так как ориентировано на единицу кода

Часть инструментов для проведения модульного тестирования, предлагаемых фреймворком NUnit:

Атрибуты:

* Test - помечает метод как тестовый;
* TestCase - позволяет передавать тестовые данные в тестовый метод. С его помощью можно создавать несколько тестовых случаев для одного теста;
* TestCaseSource - позволяет указать источник тестовых случаев.

Методы для проведения тестов класса Assert:

* Assert.True - проверяет истинность переданного выражения;
* Assert.False - проверяет ложность переданного выражения;
* Assert.Null - проверяет, что переданный объект равен null;
* Assert.NotNull - проверяет, что переданный объект не равен null;
* Assert.AreEqual - проверяет два переданных аргумента на равенство;
* Assert.AreNotEqual - проверяет два переданных аргумента на неравенство;
* Assert.AreSame - проверяет, что две переданные ссылки ссылаются на один и тот же объект;
* Assert.AreNotSame - проверяет, что две переданные ссылки не ссылаются на один и тот же объект;
* Assert.Throws - проверяет, «выбрасывает» ли переданный делегат при его вызове заданное исключение;
* Assert.DoesNotThrow - проверяет, что переданный делегат при вызове не «выбрасывает» исключение;
* Assert.Pass - тест автоматически считается пройденным;
* Assert.Fail - тест автоматически считается непройденным.

Также для проведения модульного тестирования потребуется создание моков (mock). Мок - это настраиваемый объект, который имитирует поведение реального объекта. Моки нужны для имитации зависимостей тестируемых модулей ПО. Для языка C# существует фреймворк Moq. Moq – это простой и легковесный изоляционный фреймворк, который построен на основе анонимных методов и деревьев выражений. Для создания моков он использует кодогенерацию, поэтому позволяет создавать моки интерфейсов, классов, подменять их методы (в том числе защищенные), однако не позволяет подменять статические методы.

Так как в данном проекте присутствует работа с файловой системой, ее тоже необходимо протестировать. При использовании стандартной библиотеки языка C# System.IO протестировать функционал по работе с файловой системой сложно, а в некоторых случаях невозможно. Но для C# существует сторонняя библиотека System.IO.Abstractions, она предоставляет абстракцию для файловой системы (интерфейс IFileSystem), что позволяет:

1. Внедрять файловую систему как зависимость с помощью DI-контейнера
2. Легко тестировать функционал по работе с файловой системой

Также System.IO.Abstractions предоставляет готовые моки для тестирования файловой системы.

Таким образом использование фреймворков NUnit, Moq и библиотеки System.IO.Abstractions позволит провести полное модульное тестирование приложения. Поэтому было принято решение использовать их.

Также необходимо будет протестировать REST API, предоставляемый серверной частью приложения. Это позволяет сделать приложение Postman. Postman позволяет создавать коллекции запросов к API и позволяет писать автоматизированные тесты для методов API на языке программирования JavaScript с использованием специальных встроенных библиотек.

Приложение Postman способно своим функционалом удовлетворить потребность в тестировании REST API, поэтому будет использоваться в тестировании данного проекта.