Министерство образования Республики Беларусь

Учреждение образования  
 БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
 ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Рефакторинг и оптимизация программного кода

Отчет

по лабораторной работе №4

на тему:

**МЕТОДЫ ТЕСТИРОВАНИЯ**

|  |  |  |
| --- | --- | --- |
| Проверил |  | А.В. Шелест |
|  | (подпись) |  |
| зачтено |  |  |
|  | (дата защиты) |  |
|  |  |  |
| Выполнил |  | Ю-Ф.А. Тюленев  гр. 114301 |
|  | (подпись) |  |

Минск, 2025

**Цель:** *разработать unit- и интеграционные тесты для оценки работоспособности реализованной функциональности ПС*

Ссылка на репозиторий GitHub: <https://github.com/FranGranDev/RSAS>

**Модульное тестирование**

Модульное тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

xUnit — это популярная библиотека для модульного тестирования, которая широко используется в экосистеме .NET, включая проектирование и тестирование приложений ASP.NET Core. Она предоставляет мощный, простой и современный подход к написанию тестов и является одной из рекомендуемых библиотек для тестирования в ASP.NET Core.

Пример разработанных модульных тестов для сервиса рассчета аналитики представлены на рисунках 1-2.

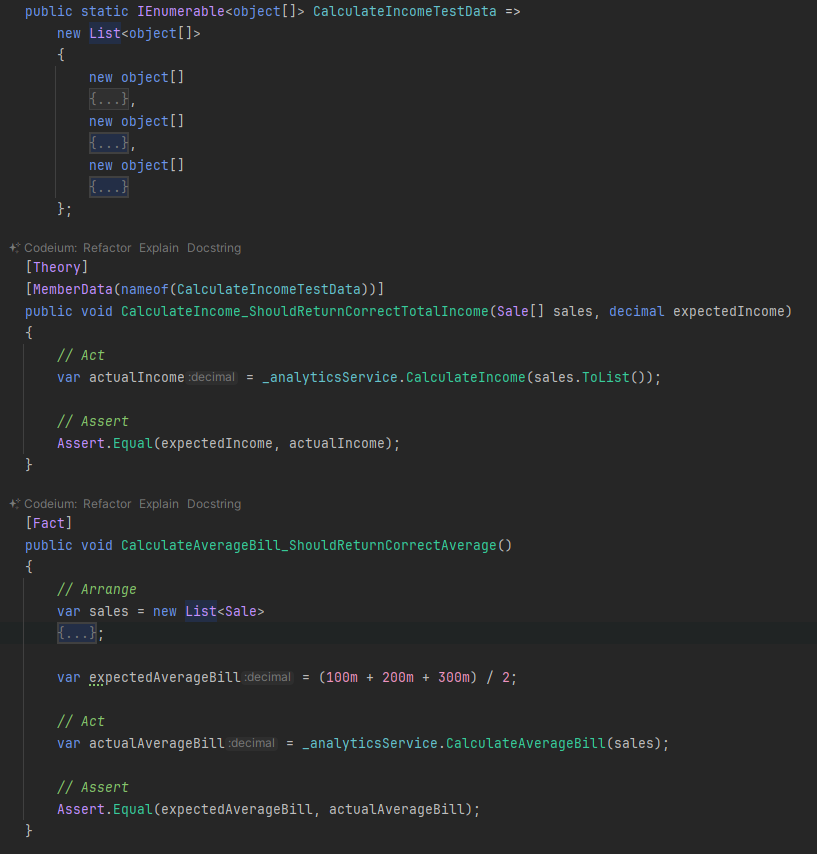


Рисунок 1 – Разработанные unit-тесты

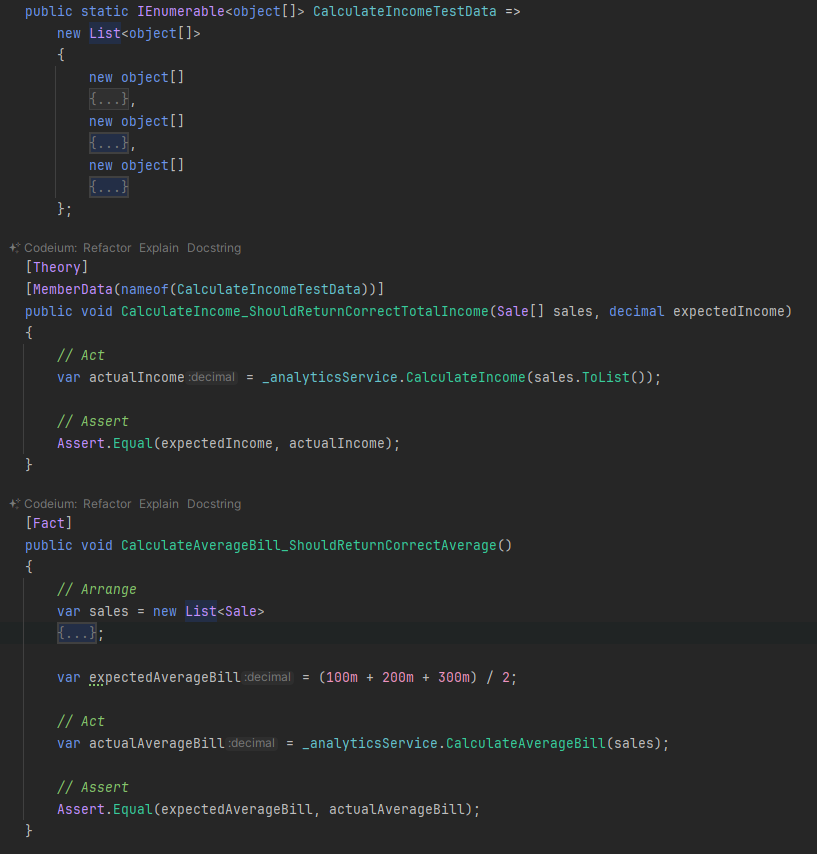


Рисунок 2 – Разработанные unit-тесты

Unit тесты, как правило, делятся на три логические части: Arrange, Act и Assert. Каждая из них выполняет определенную задачу для проверки корректности кода.

1. **Arrange (Подготовка)**

В этой части теста происходит настройка начального состояния, необходимого для выполнения теста.

Что включает:

* Создание тестируемого объекта или вызов методов инициализации.
* Подготовка входных данных.
* Настройка зависимостей (например, использование заглушек, моком или фейков).
* Указание ожидаемого результата.

1. **Act (Действие)**

В этой части теста выполняется непосредственное действие, которое вы хотите проверить. Обычно это вызов метода или выполнение некоторой операции над подготовленными данными.

Что включает:

* Вызов тестируемого метода или функции.
* Сохранение результата выполнения метода, если это необходимо для дальнейшей проверки.

**3. Assert (Утверждение)**

В этой части теста проверяется, что результат действия соответствует ожиданиям, заданным на этапе Arrange.

Что включает:

* Сравнение фактического результата с ожидаемым.
* Использование утверждений для проверки условий: равенства, выброса исключений, выполнения логических операций и т.д.

**Интеграционное тестирование**

Интеграционное тестирование (англ. Integration testing) — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Интеграционное тестирование в ASP.NET Core проверяет взаимодействие различных компонентов приложения, таких как контроллеры, middleware, базы данных и внешние сервисы. Оно выполняется с использованием встроенного тестового веб-сервера (TestServer) или класса WebApplicationFactory, который позволяет запускать приложение в изолированном окружении. Такие тесты оценивают функциональность конечных точек API, валидность маршрутизации и корректность всей системы в целом, заменяя реальные зависимости (например, базы данных) на тестовые или in-memory реализации.

Пример разработанных интеграционных тестов представлены на рисунке 3.

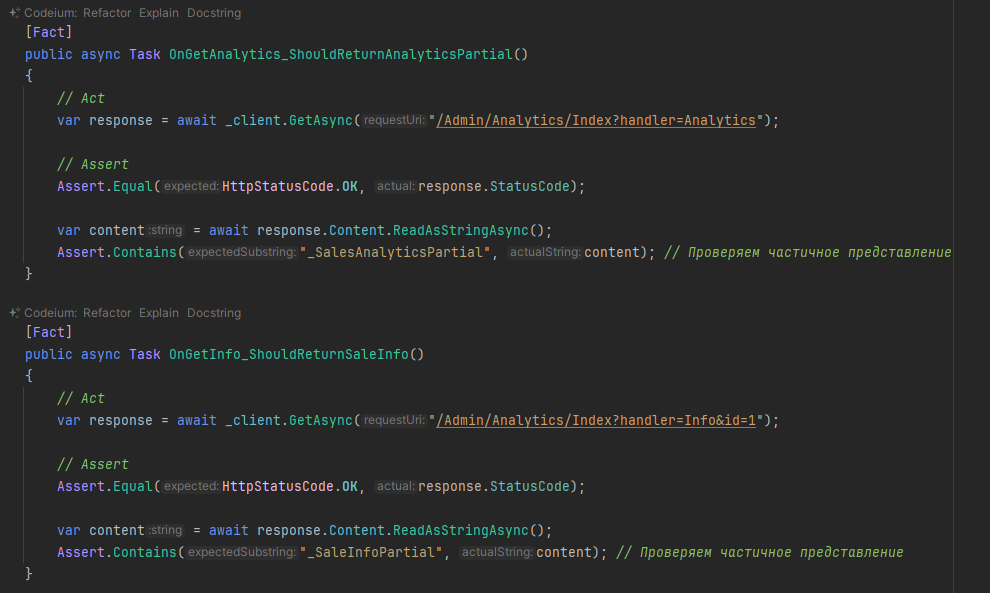
****

Рисунок 3 – Пример разработанных интеграционных тестов

Этот набор интеграционных тестов для приложения проверяет корректность работы страницы аналитики. Тесты охватывают различные сценарии взаимодействия с конечными точками и обработчиками Razor Pages:

* OnGet\_ShouldReturnPage: Проверяет успешную загрузку страницы аналитики с ожидаемыми элементами интерфейса.
* OnGetSales\_ShouldReturnSalesData: Убеждается, что обработчик GetSales возвращает корректное частичное представление для данных о продажах.
* OnPostSort\_ShouldSortSalesByPriceAsc: Тестирует сортировку продаж по цене в порядке возрастания, проверяя успешную обработку запроса и корректность результата.
* OnPostSetDate\_ShouldFilterSalesByDate: Валидирует фильтрацию данных о продажах по указанному диапазону дат, проверяя корректный рендеринг результата.
* OnGetAnalytics\_ShouldReturnAnalyticsPartial: Удостоверяется в возврате правильного частичного представления аналитики с обработчиком Analytics.
* OnGetInfo\_ShouldReturnSaleInfo: Проверяет, что обработчик Info возвращает корректное частичное представление для информации о конкретной продаже.

Все тесты используют WebApplicationFactory для изолированного запуска приложения, а также проверяют статус кодов HTTP, содержимое ответов и корректность загрузки представлений.

Результаты работы unit и интеграционных тестов представлены на рисунке 4.

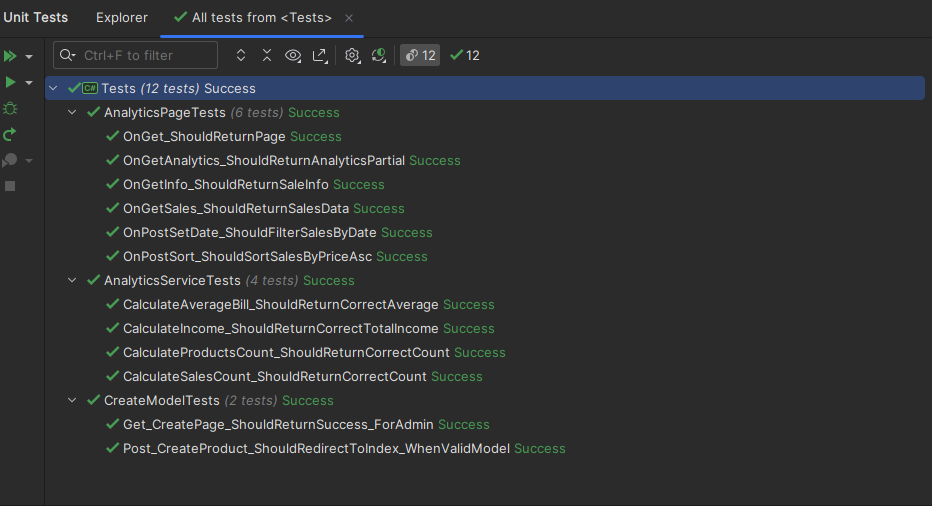


Рисунок 4 – Результаты работы unit и интеграционных тестов

**Вывод**: в ходе выполнения лабораторной работы были разработаны unit и интеграционные тесты для ключевой функции ПС, в ходе тестирование все методы и модули работали корректно.