Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Сервісне програмне забезпечення автоматизованого редагування файлів для ОС Windows»

Студента 4 курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» Волошина Ігоря Ігоровича

_____
підпис студента

Науковий керівник старший викладач кафедри інженерії програмного забезпечення та кібербезпеки Гнатченко Дмитро Дмитрович

_____
підпис керівника

Гарант освітньої програми доктор технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки Рзаєва Світлана Леонідівна

_____
підпис гаранта

КИЇВ – 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Волошину Ігорю Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Сервісне програмне забезпечення автоматизованого редагування файлів для ОС Windows»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту розробка сервісного програмного забезпечення для редагування XML файлів.

Об'єкт дослідження процес автоматизованого редагування XML файлу.

Предмет дослідження сервісне програмне забезпечення для редагування XML файлу.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис проблематики дослідження

1.2. Технічне завдання

1.3. Висновок до розділу 1

РОЗДІЛ 2. ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

2.1. Обґрунтування вибору мови програмування

2.2. Вибір середовища розробки

2.3. Висновок за розділом 2

РОЗДІЛ 3. АРХІТЕКТУРА ТА РОЗРОБКА ДОДАТКУ

3.1. Архітектура додатку

3.2. Розробка додатку

3.3. Висновок за розділом 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Аналіз предметної області*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Вибір інструментів розробки*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Архітектура та розробка додатку*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту _____ Гнатченко Д.Д.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми _____ Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент _____ Волошин І.І.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

Науковий керівник випускного кваліфікаційного проєкту

____
(підпис, дата)

Відмітка про попередній захист ____
(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента ____________Волошина І.І.____
(прізвище, ініціали)

може бути допущений до захисту екзаменаційній комісії.

Гарант освітньої програми _______________________________Рзаєва С.Л.______
(прізвище, ініціали, підпис)

Завідувач кафедри _______________________________________Криворучко О.В.
(підпис, прізвище, ініціали)

«_____»_________________20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження, дана кваліфікаційна робота присвячена дослідженню процесу автоматизованого редагування XML файлів та розробці сервісного програмного забезпечення для цієї задачі.

У першому розділі проводиться аналіз предметної області, досліджуються основні аспекти, пов'язані з редагуванням файлів, визначаються проблеми, що виникають при цьому процесі. У результаті даного аналізу сформовано технічне завдання, яке визначає вимоги до розроблюваного програмного забезпечення.

Другий розділ присвячений вибору інструментів розробки. Зокрема у ньому пояснюється обґрунтування вибору мови програмування для реалізації програмного забезпечення, визначається оптимальне середовище розробки для забезпечення ефективності та зручності роботи. Також було досліджено інструментарій розробки, необхідний для створення системи автоматизованого редагування файлів.

У третьому розділі роботи описано архітектуру створеного програмного забезпечення та сам процес розробки. Розробка серверної частини виконана у IDE JetBrains Rider, використовуючи технології .NET Core та .NET Framework. Результати роботи підтверджують досягнення поставлених цілей та виконання технічного завдання. Розроблений додаток може бути використаний широким спектром користувачів, які потребують ефективного та зручного редагування файлів на ОС Windows.

Ключові слова: XML, сервісне програмне забезпечення, JetBrains Rider, .NET Core, .NET Framework, C#.

ABSTRACT

In accordance with the research objective, this qualification work is devoted to the study of the process of automated editing of XML files and the development of service software for this task.

The first chapter analyses the subject area, examines the main aspects related to file editing, and identifies the problems that arise in this process. As a result of this analysis, a technical task is formed that defines the requirements for the software to be developed.

The second section is devoted to the choice of development tools. In particular, it explains the rationale for choosing a programming language for software implementation, determines the optimal development environment to ensure efficiency and convenience of work. It also examines the development tools required to create an automated file editing system.

The third section of the paper describes the architecture of the created software and the development process itself. The development of the server side was performed in the JetBrains Rider IDE, using .NET Core and .NET Framework technologies. The results of the work confirm the achievement of the set goals and the fulfilment of the technical task. The developed application can be used by a wide range of users who need efficient and convenient editing of files on Windows OS.

Keywords: XML, service software, JetBrains Rider, .NET Core, .NET Framework, C#.

ЗМІСТ

ВСТУП

В сучасному світі, де велика кількість програмного забезпечення використовує XML файли для зберігання налаштувань та конфігурацій, створення спеціалізованого редактора XML файлів є актуальним та корисним завданням. Це особливо стосується таких сфер, як комп'ютерні ігри та програмне забезпечення відеокарт, де від налаштувань залежить продуктивність та якість зображення на екрані. Розробка програми для редагування XML файлів, в яких зберігаються настройки відеокарт компанії AMD, є одним з цікавих напрямків розробки в галузі комп'ютерних технологій.

Метою даної роботи є створення програми, яка дозволить користувачам легко та швидко редагувати настройки відеокарт AMD, збережені у вигляді XML файлів.

Для досягнення даної мети необхідно:

- вивчити структуру XML файлів;
- вивчити специфіку роботи з налаштуваннями відеокарт AMD;
- провести аналіз існуючих рішень для редагування XML файлів;
- розробити алгоритми для зчитування, редагування та зберігання даних у цих файлах;
- розглянути можливість інтеграції з іншими програмами та системами для полегшення процесу налаштування відеокарт;

Об’єктом дослідження є програмне забезпечення для редагування XML файлів з налаштуванням відеокарт AMD. Предметом дослідження є розробка програмного забезпечення та дизайну, вибір технологій для створення

програми й реалізація функціоналу для роботи з XML файлами. Дослідження буде здійснено методами порівняльного аналізу й визначення функціональних вимог і технологій розробки.

На основі проведеного аналізу буде розроблено програмне забезпечення, що включатиме усі необхідні інструменти для редагування XML файлів з налаштуваннями відеокарт AMD. При цьому програма повинна мати користувацький інтерфейс, що спрощує роботу з файлами та надає зручний доступ до всіх параметрів налаштувань.

У результаті роботи над проектом очікується отримання програмного продукту, який значно спростить редагування налаштувань відеокарт AMD для користувачів, дозволить їм оптимізувати роботу своїх відеокарт та отримати кращі результати в роботі з графічними додатками та комп'ютерними іграми.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис проблематики дослідження

Аналіз предметної області процесу автоматизованого редагування XML файлів включав дослідження основних аспектів, пов'язаних з редагуванням таких файлів, а також виявлення проблем, що виникають у процесі редагування XML.

Основні аспекти редагування XML файлів, що були досліджені в ході аналізу даної предметної області:

1. Структура XML: Була проаналізована структура XML файлів, їхній синтаксис та особливості. Визначені основні теги, атрибути та елементи, які використовуються у XML.

2. Семантика XML: Досліджено семантику XML, тобто значення та використання різних елементів та атрибутів XML файлів. Визначено типові варіанти використання XML для зберігання та обміну даними.

3. Інструменти для редагування XML: Було проаналізовано наявні інструменти та редактори, які використовуються для редагування XML файлів. Визначено їхні переваги, недоліки та можливості.

В результаті дослідження визначено проблеми, що виникають при редагуванні XML файлів:

1. Великі розміри файлів: XML файли можуть бути досить великими, що ускладнює їхнє редагування та обробку. Великі розміри файлів можуть призводити до повільної роботи та обмежувати продуктивність редакторів XML.

2. Складна структура: XML файл може мати складну ієрархічну структуру з багатьма вкладеними елементами та атрибутами. Це може ускладнювати навігацію та зміну даних у файлі.

3. Валідація та перевірка синтаксису: XML файл повинен відповідати певним правилам та валідному синтаксису. Помилки у синтаксисі можуть призводити до невалідного XML файлу, що унеможливлює його правильну обробку.

4. Збереження зв'язків та посилань: XML файли можуть містити зв'язки та посилання між елементами. Під час редагування важливо забезпечити цілісність цих зв'язків та посилань.

Аналіз предметної області дозволив зрозуміти основні аспекти та проблеми, пов'язані з редагуванням XML файлів, що були враховані під час розробки сервісного програмного забезпечення для автоматизованого редагування файлів на ОС Windows.

Отже, проект «AMD Video Card Configuration Editor» націлений на розробку програмного забезпечення, яке забезпечує зручний та гнучкий інтерфейс для редагування XML-файлів. Головною метою проекту є надання користувачам зручного та ефективного інструменту для налаштування XML-файлів з мінімальними зусиллями. Програма включає в себе набір функцій, що дозволяють легко створювати, редагувати та зберігати XML-файли, що дозволяє зменшити час на ручну роботу з файлами та запобігти помилкам.

1.2. Технічне завдання

1. Загальні відомості

1.1. Найменування системи

Повне найменування системи: Сервісне програмне забезпечення автоматизованного редагування файлів для ОС Windows «AMD Video Card

Configuration Editor».

Скорочене найменування системи: ПЗ для ОС Windows «AMD Video Card Configuration Editor».

1.2. Планові терміни виконання роботи

Розробка була запланована на 30.01.2023 р., закінчення робіт до 31.06.2023 р.

1.3. Порядок оформлення і пред'явлення результатів робіт

Представляються результати робіт у вигляді відповідних документів та відповідному програмному продукті.

1.4. Головний бенефіціар та потенційні користувачі системи

Головним бенефіціаром є власник системи. Потенційними користувачами є всі бажаючі які прагнуть використовувати даний застосунок редагування XML-файлів.

2. Мета та призначення системи

2.1. Призначення системи

Основне призначення системи - це редагування XML-файлів, полягає у спрощенні процесу редагування та обробки даних в XML-форматі. Така система дозволяє користувачам змінювати та додавати дані до XML-файлів, що дозволяє легко та ефективно управляти даними.

2.2. Мета створення системи

Головна мета полягає в забезпеченні ефективної роботи з такими файлами, зокрема їх редагуванні, збереженні, валідації та іншій обробці. Іншими словами, мета полягає в полегшенні та прискоренні роботи з XML-файлами для користувачів.

3. Вимоги до системи

3.1. Вимоги до системи в цілому

3.1.1. Вимоги до структури та функціонування системи, перелік підсистем

- Необхідно, щоб структура програмного забезпечення була зрозумілою та логічною для користувача;

- У складі системи мають бути наявні такі компоненти, як зручний інтерфейс користувацької взаємодії та засіб зручного редагування XML-файлів.

3.1.2. Вимоги до взаємодії з користувачем

- Інтерфейс користувача повинен бути гнучким та зручним у використанні для легкого редагування XML-файлів;

- Програма повинна надавати користувачеві чітку та стислу інформацію про її призначення та функціональні можливості;

- Користувач повинен мати можливість не тільки зберігати свої дані в системі, але й відновлювати їх у разі потреби. Це забезпечить повну контрольованість користувачем процесу роботи з даними та захистить його від можливих втрат або помилкових дій;

- Програма має бути розроблена для безперебійної та ефективної роботи на різноманітних апаратних конфігураціях та операційних системах, включаючи настільні та портативні комп'ютери;

- Програма має бути розроблена таким чином, щоб мінімізувати використання системних ресурсів, таких як пам'ять і ЦП, щоб уникнути уповільнення роботи інших програм або спричинення нестабільності системи.

3.1.3. Вимоги до мов та технологій

- Проект побудовано з використанням мови програмування C# та .NET Framework;

- У проекті використовується архітектурний шаблон MVVM, щоб відокремити логіку презентації від бізнес-логіки;

- Проект використовує пакет Serilog для реєстрації подій програми;

- Проект використовує пакет Notification.Wpf для відображення сповіщень користувачеві;

- У проекті використовується адаптивний дизайн, щоб гарантувати, що програма виглядає та добре працює на екранах різних розмірів і роздільної здатності;

- У проекті використовується асинхронне програмування, щоб гарантувати, що програма залишається чуйною навіть під час виконання тривалих завдань.

3.2. Вимоги до функціональності системи

3.2.1. Гнучке налаштування XML-файлів

- Користувач повинен мати можливість змінювати налаштування XML-файлів з параметрами, такими як: GPU Min Clock, GPU Max Clock, GPU Voltage та GPU Memory Speed;

- Система повинна бути легко налаштованою, зручною та інтуїтивно зрозумілою для користувача;

- Користувач повинен мати можливість переглядати відповідні значення налаштувань, збережених в XML-файлах.

3.2.2. Просте додавання характеристик під час редагування файлів

- Користувач повинен мати можливість додавати нові параметри відеокарт до XML-файлів;

- Користувач повинен мати можливість змінювати існуючі параметри відеокарт;

- Система повинна бути легко редагованою та змінною.

3.2.3. Керування збереженням та завантаженням файлів

- Користувач повинен мати можливість зберігати змінені XML-файли та завантажувати їх знову в систему;

- Система повинна забезпечувати безпеку даних та зберігати файли з належними дозволами.

3.2.4. Інтерфейс користувача

- Система повинна мати зрозумілий та дружній інтерфейс

користувача;

- Користувач повинен мати можливість швидко знаходити та змінювати необхідні параметри;
- Інтерфейс повинен бути оптимізованим для роботи з XML-файлами та містити всі необхідні інструменти для редагування та збереження файлів.

3.3. Вимоги до надійності та продуктивності системи

3.3.1. Надійність

- Система повинна бути стійкою до помилок та відмов, які можуть виникнути під час роботи з XML-файлами;
- Можливо забезпечити контроль цілісності даних та захист від випадкових або навмисних змінених файлів.

3.3.2. Продуктивність

- Система повинна працювати достатньо швидко, щоб користувачі не відчували затримок під час взаємодії з нею;
- Завантаження даних із XML-файлів та їх збереження має відбуватися якомога швидше.

3.4. Вимоги до безпеки системи

- Програма повинна бути безпечною та не порушувати конфіденційність користувача.

3.5. Вимоги до інтерфейсу користувача

3.5.1. Зручність використання

- Інтерфейс повинен бути зрозумілим та інтуїтивно зрозумілим для користувачів будь-якого рівня технічної грамотності;

3.5.2. Надійність інтерфейсу

- Інтерфейс повинен працювати без збоїв та помилок;
- Інтерфейс повинен бути простим у використанні та надійним.

4. Вимоги до програмного забезпечення

- Для розробки необхідно використовувати актуальні технології та мови програмування, щоб забезпечити оптимальну продуктивність та ефективність системи;
- Система повинна бути масштабованою та готовою до легкого розширення, щоб задовольнити потреби користувачів у майбутньому;
- Інтерфейс системи повинен бути простим і зрозумілим для користувачів різного рівня технічної грамотності, щоб забезпечити зручне та ефективне використання системи.

5. Вимоги до технічного забезпечення

- Система повинна бути сумісною з будь-якими пристроями, що працюють на операційній системі Windows, та здатна працювати з різними конфігураціями. Одним з важливих елементів для оптимальної роботи системи є наявність SSD для швидкого зчитування та запису даних;
- Для забезпечення безпеки даних, система повинна мати можливість створювати резервні копії XML-файлів, або автоматично створювати копії під час редагування файлу.

1.3. Висновок до розділу 1

Сервісне програмне забезпечення має на меті допомогти користувачам у автоматизації редагування XML файлів. Додаток буде розроблений на базі .NET Framework і написаний мовою програмування C# для забезпечення високої продуктивності. Чіткі вимоги, які було визначено у технічномузавданні, дозволяють перейти до стадії розробки архітектури, логічних та фізичних моделей бази даних, а також макету веб-додатка.

РОЗДІЛ 2
ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

2.1. Обґрунтування вибору мови програмування

C# - це потужна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина фреймворку .NET. Є кілька основних причин, чому C# являється гарним вибором для розробки програмного забезпечення:

- Це сучасна мова, яка підтримує багато розширених функцій, таких як збір сміття, безпека типів та автоматичне керування пам'яттю, які допомагають розробникам писати ефективний та безпомилковий код;
- Має велику та активну спільноту розробників, які діляться своїми знаннями та досвідом через різні онлайн-форуми, блоги та соціальні мережі. Це забезпечує розробників великою кількістю ресурсів та підтримкою, які допомагають їм вирішувати проблеми та вдосконалювати свої навички;
- Це універсальна мова, яку можна використовувати для розробки широкого спектру додатків, від десктопних додатків до веб-додатків та мобільних додатків. Ви можете використовувати інші програмні платформи, такі як Windows, Linux і macOS;
- Легко інтегрується з новими технологіями від Microsoft, такими як SQL Server та Azure, що робить її ідеальним вибором для розробки відладчиків для асинхронного програмування;
- Ця мова являється строго типізованою мовою, що означає, що змінні повинні бути оголошені з певним типом даних. Це допомагає відстежувати помилки під час компіляції та гарантує, що код буде більш стійким та надійним.

Отже, C# - це сучасна, потужна та універсальна мова програмування, яка добре підходить для розробки широкого спектру додатків. Її велика спільнота та потужна підтримка роблять її чудовим вибором для розробки програмного забезпечення.

C# є чудовим вибором для розробки проекту “AMD Video Card Configuration Editor” з кількох причин:

- По-перше, C# є популярною мовою програмування. Це означає, що є велика кількість розробників та ресурсів, доступних для вирішення будь-яких проблем, що можуть виникнути під час розробки;
- По-друге, C# є мовою яку легко вивчити та використовувати. Вона має чіткий та зрозумілий синтаксис, що дозволяє розробникам швидко та ефективно створювати програми;
- По-третє, C# підтримує багатопоточність, що дозволяє розробникам створювати швидкі та ефективні програми, які можуть працювати з багатьма процесами одночасно;
- Нарешті, C# має ряд інструментів та фреймворків, таких як .NET Framework та .NET Core, що дозволяють розробникам швидко та легко створювати програми з високим рівнем функціональності та безпеки.

Отже, враховуючи всі ці переваги, C# являється ідеальним вибором для розробки проекту «AMD Video Card Configuration Editor». Всі переваги мови програмування C# будуть в повній мірі використовуватись в даному проекті та будуть демонструвати саму гнучкість самої мови.

2.2. Вибір середовища розробки

1. Вибір IDE для розробки проекту

На сьогоднішній день існує кілька популярних середовищ для розробки на мові C#, зокрема Visual Studio та Rider. Проведемо короткий аналіз для

визначення, яке середовище краще підійде для розробки проекту «AMD Video Card Configuration Editor», і обґрунтуємо свій вибір.

Visual Studio та Rider є дуже популярними середовищами розробки для мови програмування C#. Обидва середовища мають свої переваги та недоліки, тому вирішення про вибір кращої IDE для проекту «AMD Video Card Configuration Editor» варто робити на підставі конкретних потреб проекту.

Visual Studio є одним з найбільш повних та універсальних середовищ розробки, що підтримує різні мови програмування та платформи. Вона має багато різноманітних інструментів для розробки, таких як відлагоджувач, побудова інтерфейсів користувача, підтримка Git та інших систем контролю версій, інтеграція з різними серверами тестування, аналізатори коду, плагіни та інше.

Rider - це інтегроване середовище розробки (IDE) для програмістів, яке розробляється компанією JetBrains. Це середовище розробки для різних мов програмування, таких як C#, F#, Visual Basic .NET, JavaScript, TypeScript, HTML, CSS та інші. Rider пропонує широкі можливості, такі як підтримка розробки .NET Framework, .NET Core, Unity, Xamarin і середовища ASP.NET, підтримка відладки коду, вбудована система керування версіями, підсвічування синтаксису, автодоповнення, рефакторинг та багато іншого. Крім того, Rider працює на різних платформах, таких як Windows, macOS та Linux.

Переваги Pro версії Rider від інших IDE:

- Має додаткові функції та плагіни для підвищення продуктивності розробки;
- Забезпечує підтримку більшої кількості мов програмування та фреймворків;
- Має більш розширену функціональність для дебаггинга та тестування коду;

- Підтримує інтеграцію з багатьма іншими інструментами та сервісами розробки програмного забезпечення.

Rider є більш легкою та швидкодіючою IDE для розробки на C#. Її перевагами є низькі вимоги до системи, що робить Rider ідеальним вибором для розробки на старих комп'ютерах або машинах з обмеженими ресурсами. Також вона має зручний інтерфейс, розширення, включаючи ті, що дозволяють розробляти веб-додатки, та підтримку популярних систем контролю версій.

Rider надає можливість взаємодіяти з Git-репозиторієм безпосередньо в його інтерфейсі. Це дозволяє розробникам здійснювати коміти, відгалуження, злиття, перевірку стану репозиторію та багато іншого, не покидаючи середовище Rider. Основні функції взаємодії з Git в Rider:

- Інтегрована консоль Git: Rider надає доступ до консолі Git безпосередньо в його інтерфейсі. Це дозволяє розробникам виконувати Git-команди без необхідності відкривати окреме вікно терміналу;
- Git-коміти: Rider дозволяє розробникам здійснювати Git-коміти безпосередньо з його інтерфейсу. Це дозволяє внести зміни в код та зберегти їх із зазначенням опису коміту;
- Git-відгалуження: Rider дозволяє розробникам створювати нові Git-відгалуження безпосередньо з його інтерфейсу. Це дозволяє розробникам працювати з різними гілками коду та злити їх в головну гілку, коли робота над функціоналом буде завершена;
- Git-злиття: Rider дозволяє розробникам здійснювати Git-злиття безпосередньо з його інтерфейсу. Це дозволяє розробникам злити зміни з однієї гілки в іншу без необхідності використовувати консоль Git;

- Git-історія: Rider надає зручний інтерфейс для перегляду історії Git-репозиторію. Розробники можуть переглядати список комітів, зміни в конкретному коміті, гілки та багато іншого.

Крім цього, Rider має інтеграцію з різними сервісами для хостингу Git репозиторіїв, такими як GitHub, Bitbucket, GitLab тощо. Це дозволяє працювати з репозиторіями безпосередньо з IDE та спрощує процес збереження змін та взаємодії з іншими учасниками проекту.

Якщо порівнювати Visual Studio та Rider за рядом параметрів, то можна виділити наступне:

- Кросплатформеність: Rider підтримує розробку на різних операційних системах, включаючи Windows, macOS і Linux, тоді як Visual Studio доступна тільки для Windows. Це дає Rider перевагу, якщо мова йде про розробку на Linux або macOS;
- Швидкість роботи: Rider має вищу швидкість роботи зі значними проектами, особливо з тими, що містять багато файлів, в той час як Visual Studio може дещо уповільнює роботу з такими проектами;
- Інтеграція з Git: Rider має вбудований клієнт Git, який забезпечує зручну інтеграцію з Git-репозиторіями, тоді як для Visual Studio потрібно встановлювати відповідні плагіни;
- Функціональність: Visual Studio має більше розширень та додатків, що розширюють її функціональність, в той час як Rider віддає перевагу більш інтегрованій середовищу.

Загалом, обидва інструменти мають свої переваги та недоліки, і вибір між ними залежить від особистих уподобань та потреб проекту. Однак, з урахуванням крос-платформності та розширених функцій для роботи з Git, можна сказати, що Rider може бути кращим вибором для розробки проекту “AMD Video Card Configuration Editor”.

2. Обрання допоміжних інструментів та технологій

Для розробки проекту “AMD Video Card Configuration Editor”, крім IDE Rider, можуть знадобитись такі інструменти:

- Git: для контролю версій і спільної роботи над кодом в команді;
- .NET Framework/.NET Core: залежно від обраної платформи, на якій буде працювати додаток;

ReSharper: для підвищення продуктивності та якості коду, якщо буде потрібно.

2.3. Висновок до розділу 2

Цілком важливо ретельно пройти кожен етап і правильно визначити всі деталі, щоб під час розробки програмного забезпечення уникнути непередбачуваних ситуацій, таких як зміна взаємодії користувача з сервісом. Внесення змін у вже функціонуючий продукт може вимагати більше ресурсів, ніж розробка нового функціоналу. Зрозуміло, все залежить від конкретної функції, але підготовчий етап до розробки - це вкрай важливий процес, який не варто пропускати.

Отже, у цьому розділі було вибрано середовище розробки IDE Rider для «AMD Video Card Configuration Editor», обґрунтована придатність його використання. Також обрано допоміжні інструменти розробки, такі як .NET Framework для безпосередньої розробки програмного забезпечення, PowerShell для роботи зі скриптами та автоматизації операційної системи, Git для контролю версій програмного забезпечення та спільної роботи над проектом.

РОЗДІЛ 3
АРХІТЕКТУРА ТА РОЗРОБКА ДОДАТКУ

3.1. Архітектура додатку

1. Трикомпонентна архітектура

Трикомпонентна архітектура - це тип архітектури програмного забезпечення, який розділяє програму на три окремі компоненти: представлення (presentation), бізнес-логіку (business logic) та доступ до даних (data access).

Переваги використання трикомпонентної архітектури полягають в наступному:

- Забезпечує високу рівень розширюваності та підтримки коду, що дозволяє розробникам легко вносити зміни в окремі компоненти, не впливаючи на решту програми;
- Розділення коду на окремі компоненти дозволяє збільшити масштабованість програми та забезпечити можливість паралельної роботи розробників над окремими компонентами.

У даному проекті трикомпонентна архітектура використовується для організації програмного забезпечення на три окремі рівні. Представлення відповідає за взаємодію з користувачем та інтерфейс взаємодії, бізнес-логіка - за виконання бізнес-правил та обробку даних, а доступ до даних - за роботу з даними. Використання трикомпонентної архітектури дозволяє забезпечити легку розширюваність програми, покращити її ефективність та підтримку, а також спростити її розуміння та розробку. Як показано на рисунку 3.1.

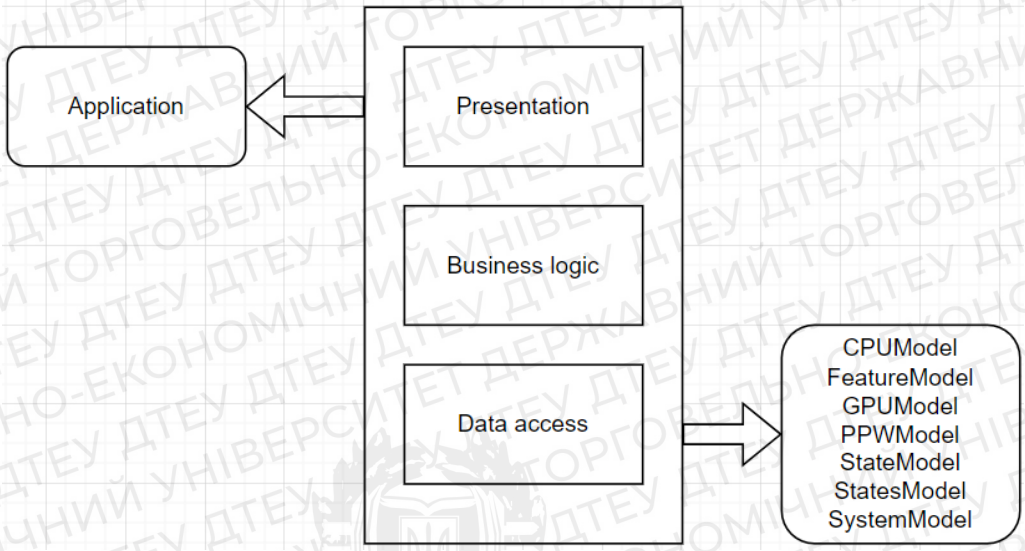


Рис. 3.1. Трикомпонентна архітектура

2. Use-Case Діаграма

Use-Case діаграма - це діаграма, що відображає функціональність системи з точки зору її користувачів та інших зацікавлених сторін. У ній показані взаємодії між системою та її користувачами, а також можливі дії, які можуть виконуватись користувачами системи.

Основною метою Use-Case діаграми є опис сценаріїв використання системи. Кожен сценарій використання описується одним або кількома use-case'ами, що розглядаються як окремі функціональні блоки системи.

Опис діаграми:

- Користувач: може використовувати наш проект в повному розмірі, а саме: читання, редагування та зберігання XML-файлів;
- Розробник: Покращує з середини проект, та збільшує функціонал проекту.

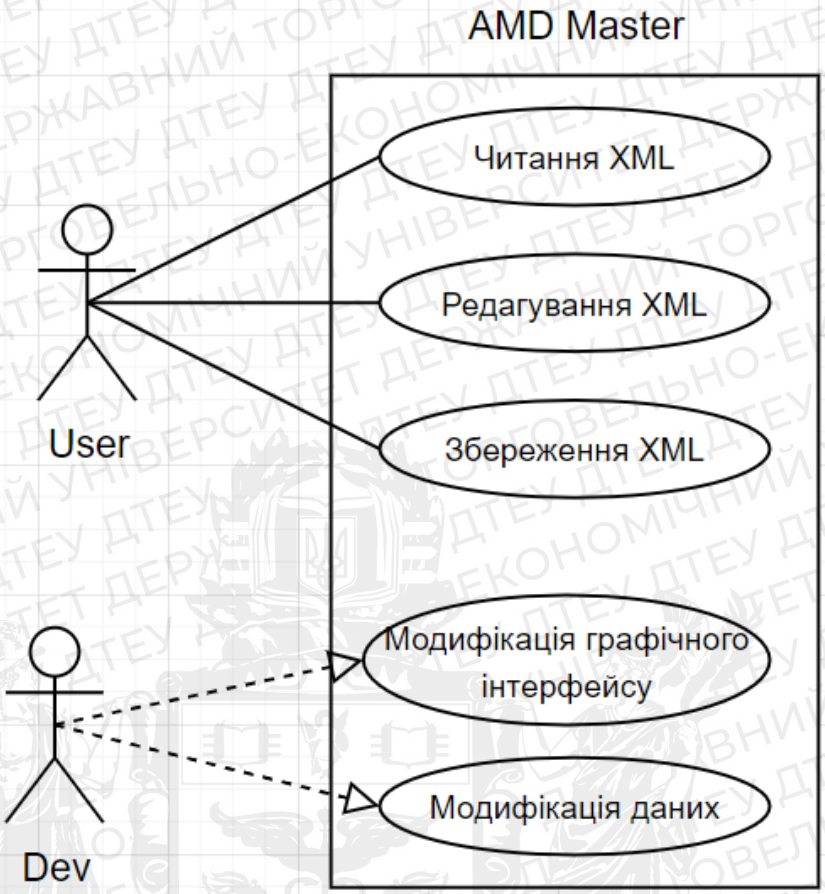


Рис. 3.2. Use- Case діаграма

3. Діаграма станів

Діаграма стану, є однією з важливих діаграм, що дозволяє описати поведінку об'єкта в різних станах. Вона показує, як об'єкт може переходити з одного стану в інший в залежності від вхідних подій або дій користувача.

Діаграма стану допомагає зрозуміти логіку роботи системи та може бути корисною при її розробці. Вона дозволяє виявити можливі помилки в логіці переходів та дій системи, що дозволяє уникнути їх у подальшому.

Діаграма стану складається з наступних елементів:

- Початковий стан (коло);
- Кінцевий стан (коло з маленьким колом всередині);
- Активні дії які відбуваються в даному стані (округлий прямокутник);
- Перехід між станами (стрілка).

Рис. 3.3. Діаграма стану

4. Діаграма класів

Діаграма класів - це модель, що відображає структуру класів, їх взаємодію та відносини у системі. Елементами діаграми класів є класи, інтерфейси, абстрактні класи, зв'язки між ними та атрибути класів.

Класи на діаграмі зображуються у вигляді прямокутників з назвою класу всередині. У прямокутнику також можуть бути вказані атрибути класу, такі як поля та методи. Зв'язки між класами зображуються за допомогою ліній з різними видами стрілок, які вказують на тип відносин між класами, такі як наслідування, агрегація, композиція та залежність.

Діаграма класів може допомогти розібратися зі структурою системи та зрозуміти, як класи взаємодіють між собою. Вона є важливим етапом проектування системи, оскільки дозволяє заздалегідь з'ясувати структуру та організацію класів у системі.

5. Блок-схема

Блок-схема - це графічне зображення послідовності дій або алгоритму, яке складається з різних блоків, які відображають різні етапи процесу, та зв'язків між ними.

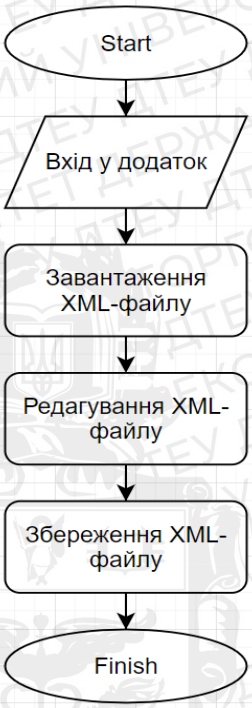


Рис. 3.4. Блок-схема алгоритму програми

6. Модель С4

Модель С4 - це абстрактна модель, вона дозволяє описувати архітектуру програмного забезпечення на різних рівнях деталей та комплексної структури. Модель допомагає розробникам програмного коду забезпечити легке спілкування про архітектуру, визначати відповідальності та обов'язки між компонентами, а також використовувати її для планування, аналізу та вдосконалення архітектури.

Модель С4 складається з 4 рівнів:

- Рівень контексту (Context level) - найвищий рівень абстракції, на якому описується роль системи та її залежності від зовнішніх сутностей;
- Рівень контейнерів (Container level) - на цьому рівні описуються основні компоненти системи, їх відносини та залежності;
- Рівень компонентів (Component level) - на цьому рівні описуються

деталі реалізації кожного компонента системи;

- Рівень коду (Code level) - найнижчий рівень, на якому описується реалізація кожного компонента на рівні коду.

Ці рівні допомагають розробникам максимально чітко та повно описувати архітектуру програмного забезпечення, використовуючи максимально доступні та зрозумілі техніки та інструменти.

Модель C4 визначає набір абстракцій, які допомагають описувати ваші програмні системи:

- Особа (Person) - кінцевий користувач, який використовує вашу систему;

- Програмна система (Software system) - найвищий рівень абстракції, що надає цінність кінцевим користувачам;

- Контейнер (Container) - програми та сховища даних, які складають систему;

- Компонент (Component) - будівельні блоки/модулі, з яких складається контейнер.

З даними концепціями високого рівня, давайте подивимося, їх детальніше на рисунках 3.5-3.7, які є спроектовані для нашого проекту.

6.1. Контекстна діаграма системи

Дана діаграма являється високорівневим представлення про мою систему. Діаграма демонструє систему як централізовану систему

На діаграмі для проекту “AMD Video Card Configuration Editor” можуть бути такі елементи:

• Система “AMD Video Card Configuration Editor”: вона містить різноманітні компоненти, такі як клієнтський додаток;

• Клієнти: це зовнішні сутності, які взаємодіють з системою через клієнтський додаток;

• Інші системи: система “AMD Video Card Configuration Editor”

може взаємодіяти з іншими системами, які надають додатковий функціонал для взаємодії з клієнтським додатком.

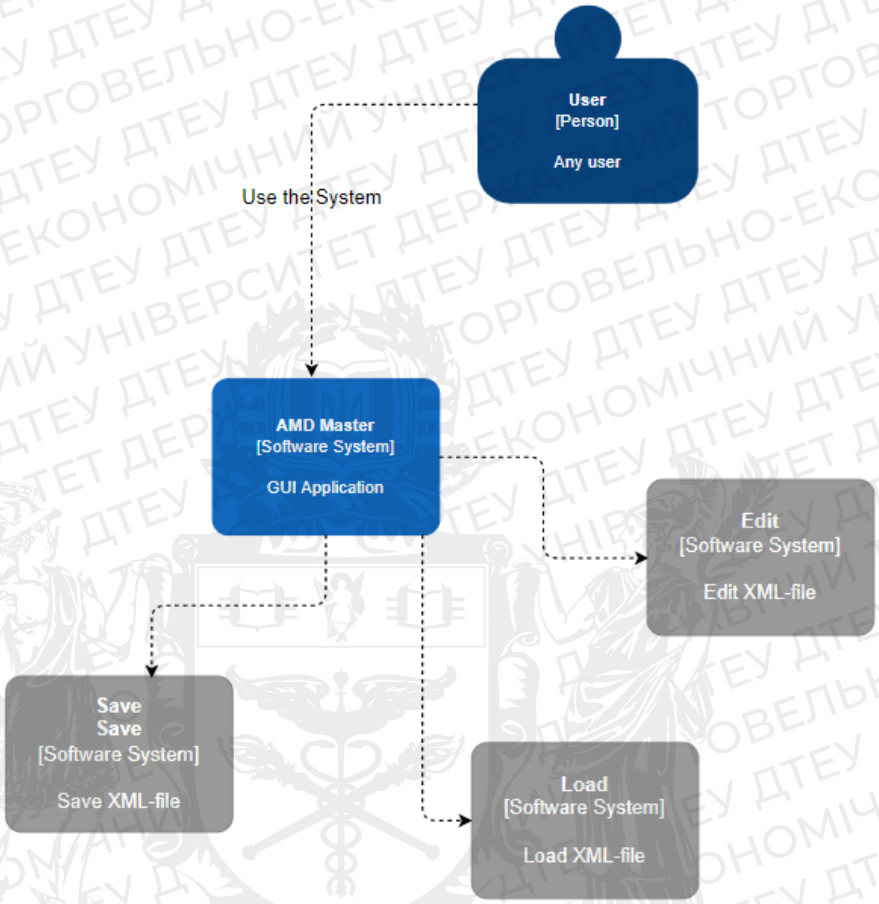


Рис. 3.5. Контекстна діаграма системи

6.2. Діаграма контейнера

Це діаграма, яка дозволяє описати взаємодію між компонентами системи на рівні контейнерів. Контейнер може бути представлений як логічна або фізична група компонентів, які мають спільну функціональність та можуть взаємодіяти між собою.

Діаграма контейнера для “AMD Video Card Configuration Editor” може включати такі елементи:

- Клієнтський додаток: це компонент, який відповідає за взаємодію з користувачем та надання йому інтерфейсі для роботи з системою. Проект “AMD Video Card Configuration Editor” містить графічний інтерфейс користувача (GUI);

- Додатковий сервіс: це компонент, який взаємодіє з даними під час редагування XML-файлу.

Таким чином, діаграма контейнера для “AMD Video Card Configuration Editor” може включати в себе компоненти, що забезпечують взаємодію з користувачем та обробкою даних у XML-файлі.

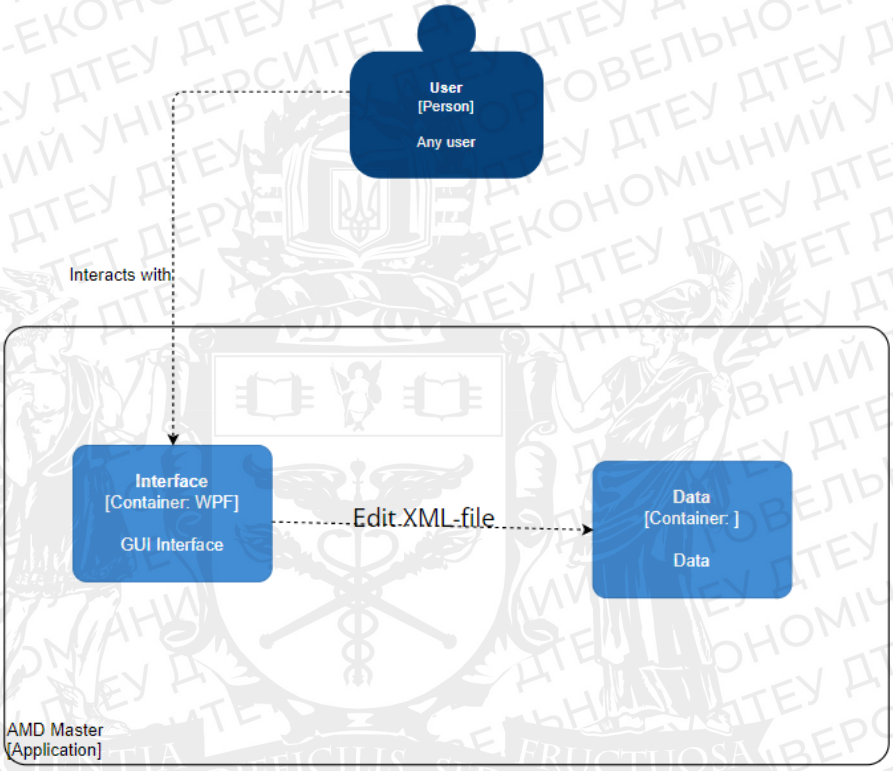


Рис. 3.6. Діаграма контейнера

6.3. Діаграма компонентів

Це структурна діаграма, що показує компоненти системи та їх зв'язки. Компоненти можуть бути фізичними (наприклад, сервери, бази даних) або логічними (наприклад, модулі програмного забезпечення).

На рисунку 3.7 можемо побачити що система складається двох основних компонентів, таких як: користувацький інтерфейс та робота з даними.

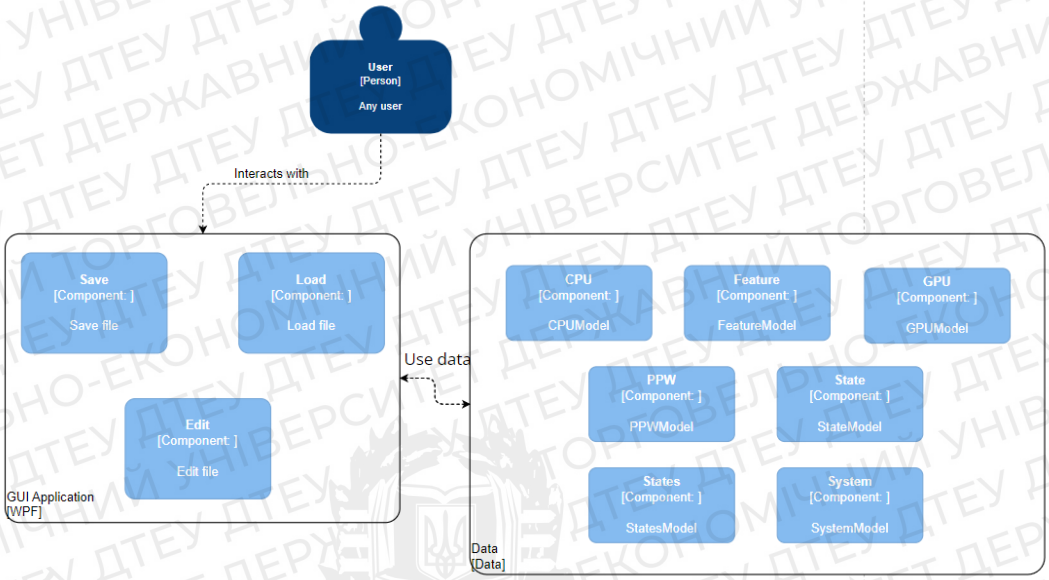


Рис. 3.7. Діаграма компонентів

6.4. Діаграма коду

Діаграма коду в подробицях розкриває кожен компонент нашого проекту, показуючи їх реалізацію у вигляді коду. Ця діаграма є важливим інструментом для демонстрації, але найкращим варіантом є детальне описання кожного файлу в майбутніх розділах диплому, де будуть розглянуті всі нюанси і деталі реалізації проекту. Там буде зосереджено на кожному файлі окремо, що дозволить заглибитися в кожну дрібницю.

3.2. Розробка додатку

1. Детальний опис проекту

У цьому розділі я планую описати загальну структуру проекту «AMD Video Card Configuration Editor» та розповісти про кожну складову проекту більш детально. Проект «AMD Video Card Configuration Editor» розроблений для редагування XML-файлів та має гнучкий інтерфейс, який дозволяє легко ознайомитись з функціоналом та налаштувати файли. У проекті використовуються багато нестандартних способів реалізації, що робить його цікавим для розгляду.

Структура проекту “AMD Video Card Configuration Editor” складається з наступних каталогів: AMD_Master, AMD_Master.Data_Layer, AMD_Master.Business_Layer, та AMD_Master.Extension_Methods. Каталог AMD_Master містить головні класи та моделі, які використовуються у всьому проекті. AMD_Master.Data_Layer містить класи, які працюють з даними, зокрема з XML-файлами. AMD_Master.Business_Layer містить бізнес-логіку проекту, в той час як AMD_Master.Extension_Methods містить розширення для деяких класів.

1.1. Головний каталог проекту “AMD Video Card Configuration Editor”

Код файлу App.cs який демонструється, який є головним файлом додатку. Він містить клас App, який є нащадком класу Application з Windows Presentation Foundation (WPF) і містить методи для ініціалізації та запуску додатку.

Метод OnExit викликається при завершенні роботи додатку та перевіряє, чи реалізує _serviceProvider інтерфейс IDisposable. Якщо це так, то він викликає метод Dispose(), який відповідає за звільнення ресурсів, використовуваних _serviceProvider.

Метод OnStartup викликається при запуску додатку та містить код для налаштування служб та вікна додатку. Він створює новий екземпляр LoggerConfiguration з бібліотеки Serilog для налаштування логування в файл. Далі, за допомогою ServiceCollection, він реєструє всі служби, необхідні для функціонування додатку, в тому числі клас MainWindow, який є головним вікном додатку. Після цього він будує ServiceProvider зі списку зареєстрованих служб, та зберігає його в змінну _serviceProvider. Він викликає метод ShowDialog () для головного вікна, щоб відобразити його на екрані користувача.

У Файлі DependencyInjection.cs знаходиться клас DependencyInjection, який містить статичний метод AddServices. Цей метод розширює інтерфейс

IServiceCollection, який дозволяє зареєструвати та налаштувати сервіси, що використовуються в проекті.

Цей файл містить декілька різних служб, що реалізують інтерфейси IXmlService, INotificationService, IUserSettings та інші. Також зареєстровані ViewModel та View, які будуть використовуватись в проекті.

Найбільш відомий принцип реєстрації сервісів, що використовується в цьому файлі - це Dependency Injection (DI) - це процес надання залежностей для об'єктів з іншого об'єкту, замість того, щоб об'єкти створювали свої власні залежності. Це дозволяє полегшити тестування, підтримку та розширення коду. В даному проекті DI реалізований за допомогою фреймворка Microsoft.Extensions.DependencyInjection, який дозволяє легко реєструвати та вирішувати залежності.

Файл UserSettings.cs містить код опис якого інтерфейс IUserSettings та його реалізацію у вигляді класу UserSettings. Давайте розглянемо його основні складові:

Інтерфейс IUserSettings визначає властивості та методи для збереження та завантаження користувацьких налаштувань. Він містить такі елементи:

- ConfigurationFilePath: Рядок, що отримує або задає шлях до файлу конфігурації;
- LastModified: Об'єкт типу DateTimeOffset, що отримує або задає дату та час останньої зміни налаштувань;
- Save(): Метод, який зберігає користувацькі налаштування до файлу;
- SaveAsync(CancellationToken cancellationToken): Асинхронний метод, який зберігає користувацькі налаштування до файлу з можливістю відміни операції за допомогою об'єкта CancellationToken.

Клас UserSettings реалізує інтерфейс IUserSettings та містить логіку для збереження та завантаження користувацьких налаштувань. Деякі основні

елементи цього класу:

- FilePath: Рядок, що містить шлях до файлу налаштувань (за замовчуванням settings.json в директорії ApplicationData);
- Lock: Об'єкт для блокування доступу до файлу налаштувань для запобігання конкурентному доступу з боку декількох потоків.
- ObservableProperty: Атрибут, який використовується з бібліотекою CommunityToolkit.Mvvm.ComponentModel для автоматичного сповіщення про зміни властивостей.
- Конструктор UserSettings(bool firstTime): Створює новий екземпляр класу UserSettings. Якщо параметр firstTime має значення true, перевіряється наявність файлу налаштувань, завантажуються налаштування, якщо файл існує, і створюється файл з налаштув.

Файл INavigableView.cs визначає інтерфейс під назвою INavigableView. Інтерфейс представляє представлення, яке може переходити до інших представлень і має пов’язану модель представлення. Він має єдиний загальний параметр типу T, який представляє тип моделі перегляду.

Інтерфейс має одну загальнодоступну властивість ViewModel типу T. Ця властивість отримує модель представлення, пов’язану з представленням. Ключове слово out перед параметром типу вказує, що загальний тип T є коваріантним.

Код файлом якого є NotificatonService.cs визначає клас NotificationService та інтерфейс INotificationService у просторі імен AMD_Master.Presentation_Layer.Services. Клас NotificationService реалізує інтерфейс INotificationService, надаючи методи для відображення різних типів сповіщень, таких як помилка, інформація, сповіщення, успіх і попередження.

Клас NotificationService приймає екземпляр INotificationManager як параметр конструктора, який використовується для відображення сповіщень.

Інтерфейс INotificationService визначає методи для відображення різних

типів сповіщень, кожне з яких приймає повідомлення та необов'язковий заголовок як параметри. Інтерфейс також містить метод ShowAsync, який приймає повідомлення, заголовок і тип сповіщення як параметри, забезпечуючи спосіб відображення спеціальних сповіщень. Кожен метод повертає об'єкт Task, який представляє асинхронну операцію відображення сповіщення.

Файл PageService.cs визначає інтерфейс IPageService та клас PageService, який реалізує інтерфейс. Метою цих класів є надання послуги для отримання сторінок із контейнера ін'єкції залежностей.

Інтерфейс IPageService визначає два загальнодоступних методи:

- GetPage<T>(): цей метод приймає параметр типу T, який має бути типом Page, і повертає FrameworkElement, який представляє сторінку типу T.
- GetPage(Type pageType): цей метод приймає параметр типу pageType, який представляє тип сторінки для отримання, і повертає FrameworkElement, який представляє сторінку вказаного типу.

Клас PageService реалізує інтерфейс IPageService і забезпечує конкретні реалізації двох загальнодоступних методів, визначених в інтерфейсі. Клас має приватне поле _serviceProvider, яке представляє контейнер ін'єкції залежностей, з якого будуть отримані сторінки. Клас має відкритий конструктор, який приймає параметр IServiceProvider, який буде використовуватися для ініціалізації поля _serviceProvider.

Файл ConfigurationViewModel.cs містить Клас ConfigurationViewModel який містить конструктор і деякі приватні та публічні методи, які обробляють події, викликані інтерфейсом користувача. Конструктор приймає три параметри типів IXmlService, IUserSettings і INotificationService та ініціалізує цими об'єктами деякі приватні поля.

Приватні поля включають цілі числа, які представляють різні значення та об'єкта класу SystemModel. Приватні методи включають

AssignPropertiesToBindablePropeties, який призначає значення відповідним прив'язуваним властивостям на основі значень у завантаженій SystemModel; LoadXmlButtonClicked, який обробляє подію натискання кнопки для завантаження XML-файлу та присвоєння значень у файлі зв'язуваним властивостям; OnSaveAsButtonClicked, який обробляє подію натискання кнопки для збереження завантаженої SystemModel у файл XML; і UpdateModelValues, який оновлює значення в завантаженій SystemModel на основі поточних значень прив'язуваних властивостей.

Клас ConfigurationViewModel розширює клас під назвою ObservableObject із простору імен CommunityToolkit.Mvvm.ComponentModel. Це базовий клас, який реалізує інтерфейс INotifyPropertyChanged і надає атрибут ObservableProperty для позначення властивостей, які можна прив'язати до інтерфейсу користувача.

Файл коду MainWindowViewModel.cs для класу моделі перегляду під назвою MainWindowViewModel у програмі WPF. Модель представлення реалізує клас ObservableObject із бібліотеки CommunityToolkit.Mvvm для підтримки двостороннього зв'язування даних із представленням.

Модель перегляду має конструктор, який приймає інтерфейс IPageService як параметр. Інтерфейс використовується для отримання початкової сторінки HomePage для відображення в головному кадрі програми. Властивість FrameContent типу FrameworkElement використовується для прив'язки поточної сторінки, що відображається в основному фреймі.

Метод NavigateToPage — це приватний метод з атрибутом [RelayCommand], який використовується для переходу на нову сторінку, коли користувач натискає кнопку або пункт меню в поданні. Метод приймає параметр Type, який визначає тип сторінки для переходу. Перед переходом на нову сторінку перевіряється, чи поточна сторінка має той самий тип, що й сторінка, на яку потрібно перейти. Якщо поточна сторінка та нова сторінка

мають різні типи, метод використовує інтерфейс IPageService для отримання нової сторінки та встановлює її як властивість FrameContent. Клас Logger використовується з бібліотеки Serilog для реєстрації інформації про подію навігації.

Файл SettingsViewModel.cs містить клас ObservableObject із простору імен CommunityToolkit.Mvvm.Componекий реалізує модель перегляду для сторінки налаштувань у програмі WPF. Він імпортує необхідні простори імен і визначає клас SettingsViewModel, який успадковntModel.

Клас має два приватних поля: _notificationService типу INotificationService та _userSettings типу IUserSettings, які передаються конструктору під час створення екземпляра моделі перегляду.

Клас має два приватні методи, прикрашені атрибутом RelayCommand: BrowseConfigurationFilePath() і SaveSettings().

Метод BrowseConfigurationFilePath() відкриває діалогове вікно файлу, щоб дозволити користувачеві шукати та вибирати файл конфігурації, а потім оновлює властивість ConfigurationFilePath об’єкта _userSettings вибраним шляхом до файлу. Нарешті, він відображає сповіщення про те, чи було вибрано файл чи ні.

Метод SaveSettings() зберігає налаштування користувача асинхронно за допомогою методу SaveAsync() об’єкта _userSettings. Після завершення операції збереження відображається сповіщення про успіх. Метод також реєструє інформаційне повідомлення про те, коли було збережено налаштування.

Код у файлі WelcomeViewModel.cs визначає клас WelcomeViewModel, який успадковує клас ObservableObject із простору імен CommunityToolkit.Mvvm.ComponentModel. Клас WelcomeViewModel містить одну властивість Title, доступну лише для читання, яка має тип string і повертає значення «Home».

1.2. AMD_Master.Data_Layer

Файл CPUModel.cs визначає клас CpuModel, який представляє модель процесора і містить список функцій, які присутні в цьому процесорі.

XmlElement атрибут, що прикріплений до властивості Features, вказує на те, що ця властивість повинна серіалізуватися в XML-елемент з іменем «FEATURE».

Клас реалізує інтерфейс IEquatable<CpuModel>, що дозволяє порівнювати дві моделі процесора за значеннями їхніх властивостей. Інтерфейс містить методи Equals, які порівнюють два об'єкти за значеннями їхніх властивостей, GetHashCode, який повертає хеш-код об'єкта, і оператори == та !=, які також порівнюють два об'єкта за значеннями їхніх властивостей.

Файл FeatureModel.cs визначає клас FeatureModel, який представляє модель особливостей системи з ID, статусом та списком станів. У класі також оголошено перерахування Definition, яке містить визначення відомих особливостей системи.

Клас містить такі елементи:

- Enabled - відображає статус особливості в рядковому форматі;
- Id - ідентифікатор особливості;
- States - модель стану особливості.

Клас також реалізує інтерфейс IEquatable<FeatureModel>, щоб забезпечити можливість порівняння об'єктів типу FeatureModel. Метод Equals порівнює Id та Enabled з іншим об'єктом. Метод GetHashCode генерує хеш-код на основі Id та Enabled.

В класі використовується атрибут XmlAttribute, щоб відобразити Enabled та Id як атрибути XML, та XmlElement, щоб відобразити список станів як елемент XML. Клас також містить вкладений елемент Definition, який описує заздалегідь визначені особливості.

Файл GPUModel.cs містить опис класу GpuModel, який представляє

модель графічного процесора (GPU). Клас містить наступні властивості:

- DevId (типу string) - ідентифікатор пристрою GPU.
- RevId (типу string) - ідентифікатор ревізії пристрою GPU.
- Features (типу List<FeatureModel>) - список особливостей (features), що визначають модель GPU.
- Ppw (типу PpwModel) - модель performance per watt (PPW) для даної моделі GPU.

Крім того, клас містить перевизначені методи Equals і GetHashCode для порівняння об'єктів типу GpuModel. Метод Equals порівнює всі властивості об'єктів GpuModel, а метод GetHashCode обчислює хеш-код для об'єкту типу GpuModel, використовуючи значення його властивостей.

Для роботи з XML клас використовує атрибути XmlAttribute і XmlElement для серіалізації і десеріалізації об'єктів. Атрибут XmlAttribute використовується для вказівки імені властивості у вихідному XML-файлі, а XmlElement - для вказівки тегу XML, що містить значення властивості.

Файл PPWModel.cs визначає клас PpwModel, який представляє модель потужності, продуктивності та ватності (PPW) зі значенням. Цей клас містить одну властивість, що називається Value, яка є цілим числом і відображає значення моделі PPW. Крім того, клас реалізує інтерфейс IEquatable<PpwModel>, що дозволяє порівнювати об'єкти типу PpwModel. Для порівняння об'єктів, реалізовані методи Equals(), GetHashCode() і перевизначений оператор ==.

Файл StateModel.cs містить клас StateModel, який представляє модель стану з ідентифікатором, станом ввімкнення та значенням. Цей клас реалізує інтерфейс IEquatable<StateModel>, що дозволяє порівнювати екземпляри класу за допомогою методу Equals.

У класі є три властивості:

- Enabled - отримує або задає стан ввімкнення стану як рядок;

- Id - отримує або задає ідентифікатор стану;
- Value - отримує або задає значення стану.

У класі є також перевизначені методи Equals, GetHashCode та ToString. Метод Equals порівнює два об'єкти класу за допомогою їх ідентифікатора, значення та стану ввімкнення. Метод GetHashCode повертає хеш-код об'єкту, що обчислюється на основі ідентифікатора, значення та стану ввімкнення. Метод ToString повертає рядок, який представляє об'єкт класу StateModel.

Файл StatesModel.cs містить клас StatesModel, який представляє колекцію об'єктів StateModel. StateModel в свою чергу містить три властивості - ідентифікатор, статус включення і значення.

В класі StatesModel визначено публічну властивість StateList, яка є списком об'єктів StateModel. Цей список серіалізується як XML-вузли з іменем «STATE» за допомогою атрибуту XmlElement.

Крім того, клас містить перевизначені методи Equals, GetHashCode і ToString, що використовуються для порівняння двох об'єктів StatesModel. Особливість Equals полягає в порівнянні всіх елементів в списку об'єктів StateModel. Метод GetHashCode генерує хеш-код за допомогою списку об'єктів StateModel в своєму вмісті.

Файл SystemModel.cs визначає клас під назвою SystemModel, який представляє модель системи з моделлю CPU та моделлю GPU. Він має дві загальнодоступні властивості: Cpu типу CpuModel і Gpu типу GpuModel, обидва прикрашені атрибутом [XmlElement] для визначення імені елемента XML під час серіалізації об’єкта. Клас також реалізує інтерфейс IEquatable, який дозволяє порівнювати екземпляри класу на рівність.

Метод Equals порівнює властивості Cpu та Gpu двох екземплярів і повертає значення true, якщо вони однакові. Метод GetHashCode повертає XOR хеш-кодів CPU та GPU.

1.3. Логіка бізнес-процесів AMD_Master.Business_Layer

Файл XmlService.cs визнача клас під назвою XmlService, який реалізує інтерфейс IXmlService. Цей клас надає два публічні методи: Deserialize і SerializeToXML.

Метод Deserialize приймає рядок XML як вхідні дані та використовує LINQ to XML для десеріалізації в об’єкт SystemModel. Рядок XML спочатку аналізується в об’єкт XDocument, який потім використовується для заповнення властивостей об’єкта SystemModel. Властивості Cpu та Gpu SystemModel заповнюються даними, отриманими з вхідних даних XML.

Метод SerializeToXML приймає об’єкт SystemModel як вхідні дані та використовує LINQ to XML, щоб серіалізувати його в рядок XML. Об’єкт SystemModel використовується для заповнення об’єкта XDocument, який потім перетворюється на рядкове представлення.

Код також містить деякі перевірки нуля, щоб переконатися, що введення не є нульовим або порожнім.

Інтерфейс IXmlService визначає ці два методи та служить контрактом для реалізації класів, яким слід слідувати.

1.4. AMD_Master.Extension_Methods

Файл ObjectExtensions.cs містить розширювальний метод (extension method) для класу Object. Розширювальний метод можна використовувати для додавання нової функціональності до існуючого класу без його наслідування або зміни його вихідного коду.

Клас ObjectExtensions містить лише один метод DeepCopy, який створює глибоку копію об'єкта вхідного параметру. Для того щоб використовувати цей метод, необхідно передати об'єкт, який буде скопійовано, як параметр методу.

Метод DeepCopy має типовий параметр T, який вказує на тип об'єкта, який буде скопійовано. Метод використовує DataContractSerializer для копіювання об'єкта. Після того, як об'єкт було скопійовано, він повертається як результат методу.

2. Проведення тестування проекту

В даному розділі хочу описати тестування власного проекту, для тестування використовую фреймворк xUnit, він зручний для написання і виконання автоматичних тестів в програмному забезпеченні. В проекті “AMD Video Card Configuration Editor” xUnit може бути корисним з кількох причин:

- Простота використання: xUnit надає простий та зрозумілий синтаксис для написання тестів;
- Автоматизація тестування: xUnit дозволяє автоматизувати виконання тестів, що полегшує процес тестування проекту “AMD Video Card Configuration Editor”. Ви можете створити набір тестів, які будуть виконуватися автоматично при кожному збиранні або релізі проекту, забезпечуючи надійну перевірку функціональності;
- Покриття коду: xUnit дозволяє виміряти покриття коду тестами. Це допомагає вам переконатися, що ваш код добре покритий тестами, що зменшує ризик виявлення помилок в майбутньому і поліпшує якість програмного забезпечення;
- Інтеграція з іншими інструментами: xUnit легко інтегрується з іншими інструментами для автоматизованого тестування, такими як засоби збирання, системи неперервної інтеграції та інші. Це дозволяє вам налаштувати повний пайплайн тестування для проекту “AMD Video Card Configuration Editor”;
- Підтримка розширень: xUnit має багато розширень, які дозволяють розширити його функціональність та пристосувати його до вашого проекту. Ви можете використовувати розширення для генерації звітів, вимірювання продуктивності, мокування об'єктів та іншого.

2.1. BusinessLayer_Test

Наведений файл XmlServiceTests.cs представляє метод XmlService_SerializeDeserialize_ReturnsEqualObjects класу XmlServiceTests.

Даний тест використовує фреймворк Xunit для перевірки коректності серіалізації та десеріалізації об'єктів типу SystemModel за допомогою XmlService.

У тесті спочатку створюється очікуваний об'єкт expectedModel типу SystemModel, який містить в собі дані про CPU та GPU. Далі за допомогою XmlService об'єкт expectedModel серіалізується у рядок xmlString, а потім цей рядок десеріалізується назад у об'єкт actualModel.

У розділі Assert проводяться перевірки, що поля та об'єкти expectedModel співпадають з відповідними полями та об'єктами actualModel.

Цей тест переконується, що процес серіалізації та десеріалізації виконується коректно, тобто після серіалізації та десеріалізації отримуємо однакові об'єкти.

2.2. DataLayerTest

Код який є у файлі CpuModelTests.cs він тестує властивість Features класу CpuModel. Тест перевіряє, чи повертається очікуване значення expectedFeatures, після того як властивість Features була задана за допомогою model.Features = expectedFeatures. Якщо перевірка пройшла успішно, тест вважається пройденим.

Цей код який є у файлі FeatureModelTests.cs тестує різні властивості класу FeatureModel.

- Тест FeatureModel_GetSetEnabled_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Enabled була задана за допомогою model.Enabled = expectedValue;
- Тест FeatureModel_GetSetId_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Id була задана за допомогою model.Id = expectedValue;
- Тест FeatureModel_GetSetStates_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedStates після того, як властивість

States була задана за допомогою model.States = expectedStates.

Цей код який є у файлі GpuModelTests.cs тестує різні властивості класу GpuModel:

- Тест GpuModel_GetSetDevId_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість DevId була задана за допомогою model.DevId = expectedValue;
- Тест GpuModel_GetSetFeatures_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedFeatures після того, як властивість Features була задана за допомогою model.Features = expectedFeatures;
- Тест GpuModel_GetSetPpw_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedPpw після того, як властивість Ppw була задана за допомогою model.Ppw = expectedPpw;
- Тест GpuModel_GetSetRevId_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість RevId була задана за допомогою model.RevId = expectedValue.

Цей код який є у файлі PpwModelTests.cs тестує властивість Value класу PpwModel.

Тест PpwModel_GetSetValue_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Value була задана за допомогою model.Value = expectedValue.

Цей код який є у файлі StateModelTests.cs тестує властивості Enabled, Id і Value класу StateModel.

- Тест StateModel_GetSetEnabled_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Enabled була задана за допомогою model.Enabled = expectedValue;
- Тест StateModel_GetSetId_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Id була задана за допомогою model.Id = expectedValue;

- Тест StateModel_GetSetValue_ReturnsCorrectValue перевіряє, чи повертається очікуване значення expectedValue після того, як властивість Value була задана за допомогою model.Value = expectedValue.

Цей код який є у файлі StatesModelTests.cs тестує властивість StateList класу StatesModel.

- Тест StatesModel_GetSetStateList_ReturnsCorrectValue перевіряє, чи повертається очікуваний список expectedStateList після того, як властивість StateList була задана за допомогою model.StateList = expectedStateList.

Цей код який є у файлі SystemModelTests.cs тестує властивості Cpu та Gpu класу SystemModel.

- Тест SystemModel_GetSetCpu_ReturnsCorrectValue перевіряє, чи повертається очікуваний об'єкт expectedCpu після того, як властивість Cpu була задана за допомогою model.Cpu = expectedCpu;

- Тест SystemModel_GetSetGpu_ReturnsCorrectValue перевіряє, чи повертається очікуваний об'єкт expectedGpu після того, як властивість Gpu була задана за допомогою model.Gpu = expectedGpu.

2.3. ExtensionMethodsTests

Цей код який є у файлі ObjectExtensionsTests.cs тестує розширений метод DeepCopy класу ObjectExtensions.

- Тест DeepCopy_NullInput_ThrowsArgumentNullException перевіряє, чи викидається виняток ArgumentNullException, коли вхідний параметр input є null. Це перевіряє, якщо намагаємося зробити глибоке копіювання null об'єкта;

- Тест DeepCopy_ValueType_ReturnsCopy перевіряє, чи повертається глибока копія значимого типу. В цьому випадку, вхідний параметр input є типом int, і тест перевіряє, чи копія (copy) має таке ж значення, як і вхідний параметр input;

- Тест DeepCopy_ReferenceType_ReturnsCopy перевіряє, чи

повертається глибока копія посилального типу. В цьому випадку, вхідний параметр input є екземпляром List<int>, і тест перевіряє, чи копія (copy) має такий самий перший елемент, як і вхідний параметр input;

- Тест DeepCopy_MutableReferenceType_ReturnsCopy перевіряє, чи повертається глибока копія для змінного посилального типу. В цьому випадку, вхідний параметр input є екземпляром List<int>, копія (copy) створюється і зберігає початковий стан input. Після зміни вхідного параметра input шляхом додавання нового елемента, тест перевіряє, чи зміни не впливають на копію (copy). Також перевіряється, чи input та copy є різними об'єктами за допомогою Assert.NotSame.

2.4. PresentationLayerTests

Цей код який є у файлі UnitTest1.cs тестує клас UserSettings і його методи:

- Тест Constructor_WithTrueArgument_CreatesNewSettingsFile перевіряє, чи створюється новий файл налаштувань при створенні об'єкту UserSettings з аргументом true. В першу чергу, тест перевіряє, чи видаляється наявний файл налаштувань, якщо він існує. Потім створюється новий об'єкт UserSettings з аргументом true, і перевіряється, чи файл налаштувань був створений, чи дата останньої модифікації відповідає поточній даті і чи поле ConfigurationFilePath має порожнє значення. На кінці тесту файл налаштувань видаляється;

- Тест Save_SavesSettingsToFile перевіряє, чи налаштування зберігаються в файл. Спочатку створюється об'єкт UserSettings з аргументом true. Далі, присвоюються значення полям ConfigurationFilePath та LastModified. Після цього викликається метод Save, який зберігає налаштування в файл. На кінці тесту зчитується вміст файлу та перевіряється, чи відповідають збережені налаштування очікуваним значенням. Файл налаштувань видаляється;

- Тест SaveAsync_SavesSettingsToFileAsync перевіряє, чи налаштування зберігаються в файл асинхронно. Схожий на попередній тест спосіб, але використовується асинхронний метод SaveAsync для збереження налаштувань у файл. Перевірка проводиться наявністю та вмістом файлу збережених налаштувань.

3. Результати роботи

Проект “AMD Video Card Configuration Editor” - це додаток, розроблений на платформі WPF з використанням мови програмування C# та трьохслойної архітектури для редагування XML-файлів. Цей додаток надає користувачам зручний інтерфейс для редагування XML-структури та вміє зберігати зміни у відповідних файлових ресурсах.

Основний функціонал який був розроблений в додатку буде включати наступне:

1. Головна сторінка додатку (Home)

Ця сторінка містить ознайомчий характер для користувачів які тільки почали користуватись даним додатком.

Рис. 3.8. Головна сторінка

2. Налаштування (Settings)

Тут містяться прості налаштування даного додатку, власне отримання необхідного XML-файлу.

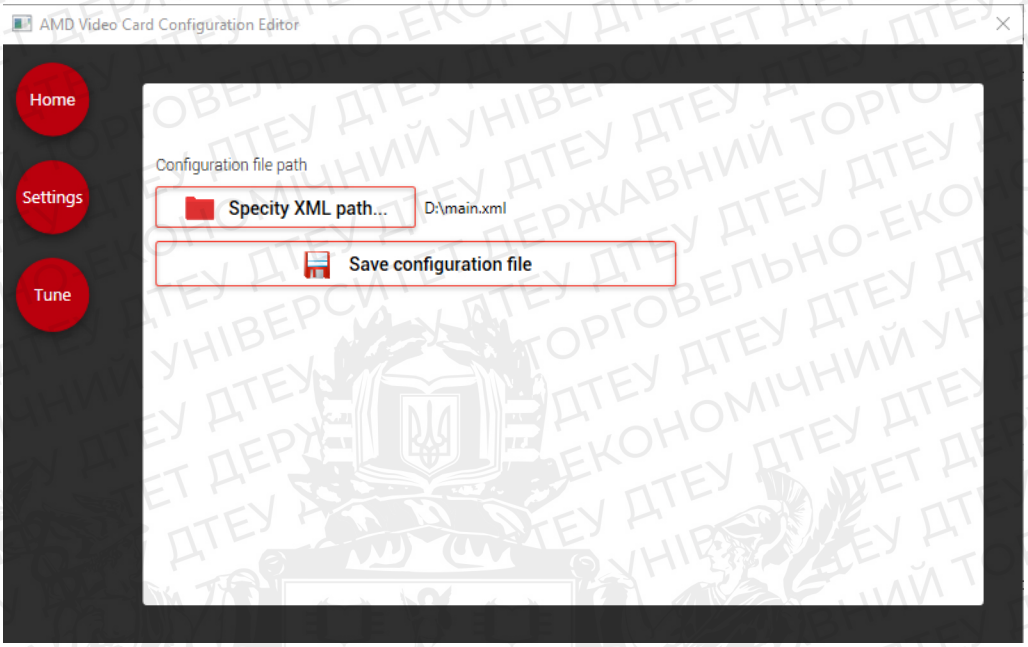


Рис. 3.9. Сторінка з налаштуваннями

3. Редагування XML-файлу (Tune)

Містяться основні поля над якими ми виконуємо редагування. Також зручне використання збереження, завантаження та оновлення даних після редагування.

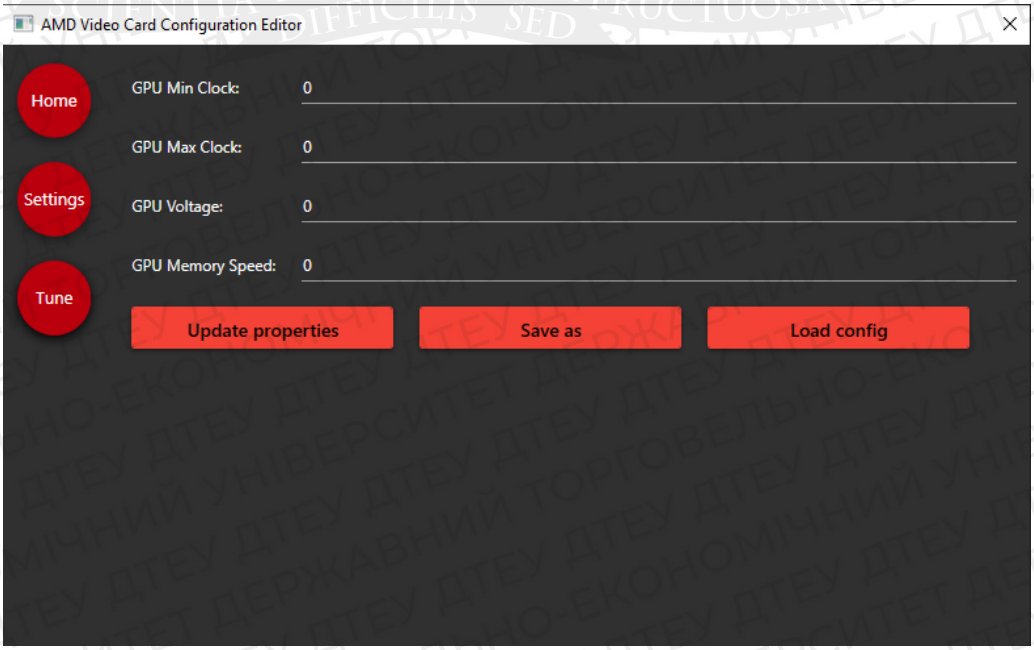


Рис. 3.10. Сторінка з налаштуваннями

Демонстрація виконання даного функціоналу додатку “AMD Video Card Configuration Editor”. Після того, як ми завантажили з відповідного XML-файлу конфігурації (рисунок 3.11).

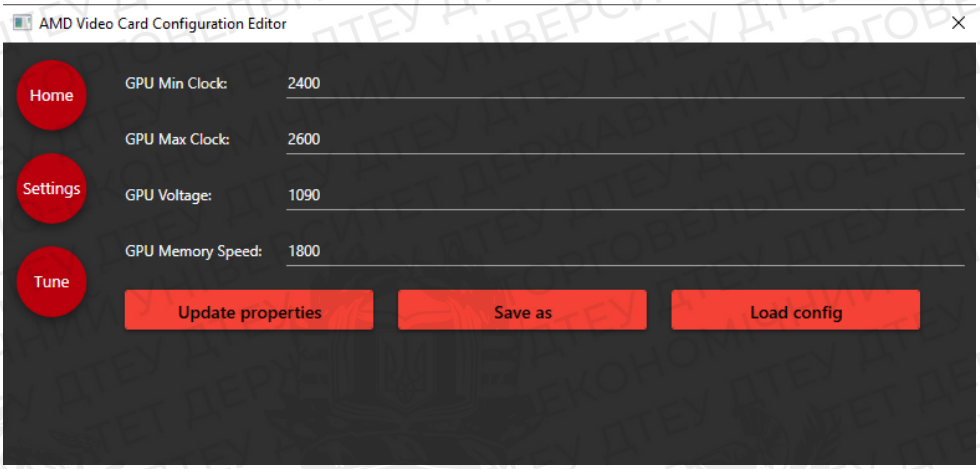


Рис. 3.11. Завантаження конфігурацій з XML-файлу

Ми можемо виконувати деякі маніпуляції з цими значеннями, для прикладу давайте змінемо GPU Min Clock = 2700 та GPU Max Clock = 3000. Та проведемо оновлення та збереження цих конфігурацій, як показано на рисунку 3.12.

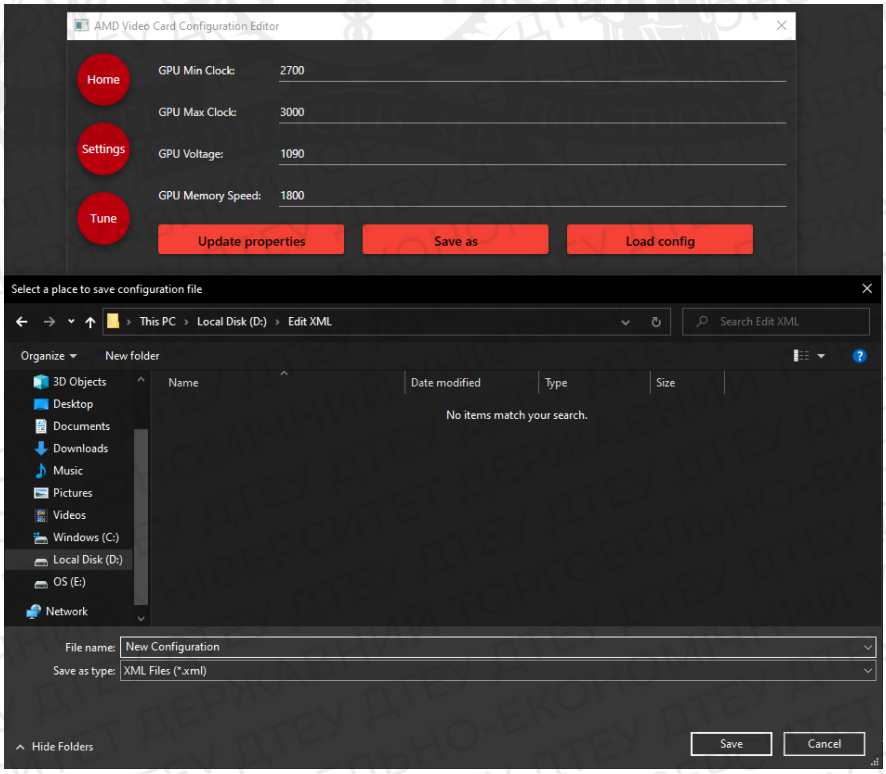


Рис. 3.12. Редагування та збереження конфігурацій

Давайте перейдемо до порівняння двох XML- файлів для наглядного прикладу виконання коректності функціоналу додатку (рисунок 3.13).

Рис. 3.13. Переглядаємо два конфігураційних файла

Також ключовою частиною чому мій інтерфейс є зручний, тим що маю великий різновид сповіщень, які будуть легко демонструвати що наші операції відбулись успішні або зазнали помилок. Демонструю рисунки 3.14-3.18.

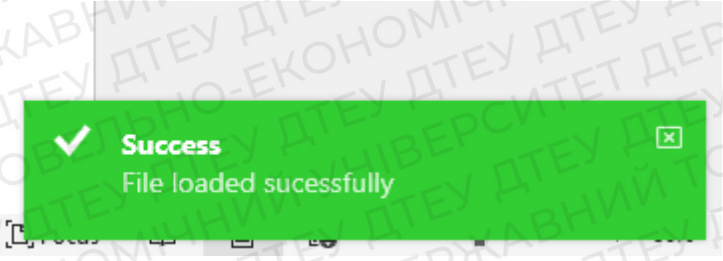


Рис. 3.14. Успішне виконання завантаження файлу

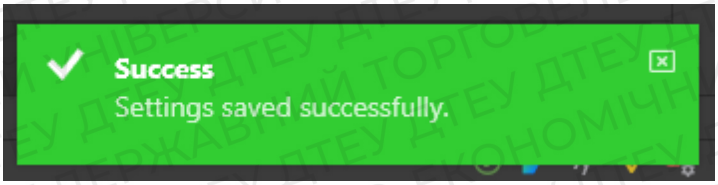


Рис. 3.15. Успішне виконання збереження файлу

Рис. 3.16. Успішне виконання оновлення значень

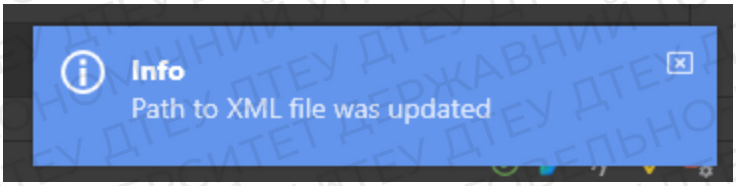


Рис. 3.17. Сповіщення що файл був доданий до додатку і з ним можна виконувати будь які взаємодії

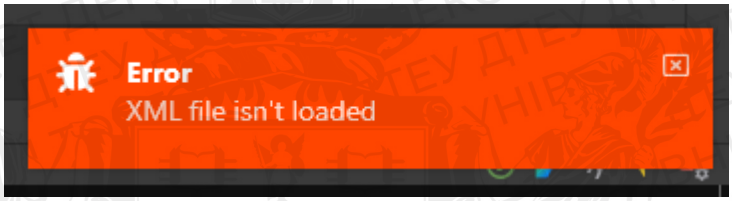


Рис. 3.18. Сповіщення що файл не був завантажений

3.3. Висновок до розділу 3

В результаті розробки додатку було успішно реалізовано сервісне програмне забезпечення для автоматизованого редагування файлів на ОС Windows. Застосування трикомпонентної архітектури та різних діаграм дозволило забезпечити модульність та гнучкість системи. Описаний алгоритм дії додатка дозволив швидко і без додаткових поправок розробити клієнт-серверне програмне забезпечення. А проведені тестування підтвердили відповідність функціональних вимог та коректність роботи застосунку.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Було проведено аналіз предметної області клієнт-серверного додатка для автоматизованого редагування XML файлів. В результаті аналізу була вибрана мова програмування C#, яка дозволяє оптимально реалізувати і забезпечити високу продуктивність майбутнього сервісного ПЗ.
2. В якості платформи для роботи з файлами було обрано .NET Framework, що робить сервіс сумісним з різними версіями Windows.
3. У процесі виконання проєкту було створено архітектуру сервісу (use case, класів, еталонну модель, блок-схему, діаграму послідовності, діаграму стану, діаграму розгортання та компонентів), а також розроблено сервісний додаток «XML Editor» з урахуванням оптимізації для різних ширин екрана.
4. Розроблена система повністю відповідає вимогам, які були поставлені на початку розробки.

Отже, було створено якісну систему для автоматизованого редагування XML файлів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visualize Your Software Architecture With The C4 Model [Електронний ресурс]. - Режим доступу: https://www.milanjovanovic.tech/blog/visualize-your-software-architecture-with-the-c4-model .Дата звернення: 13.05.2023 р.
2. How to build and deploy a three-layer architecture application with C# [Електронний ресурс]. - Режим доступу: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-layer-architecture-application-with-c-sharp-net-in-practice.html .Дата звернення: 13.05.2023 р.
3. Windows Presentation Foundation documentation [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0 .Дата звернення: 5.05.2023 р.
4. C# programming guide [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/ .Дата звернення: 7.05.2023 р.
5. Online C4 Model Software [Електронний ресурс]. - Режим доступу: https://online.visual-paradigm.com/diagrams/features/c4-model-tool/ .Дата звернення: 13.05.2023 р.
6. Diagrams [Електронний ресурс]. - Режим доступу: https://app.diagrams.net/ .Дата звернення: 13.05.2023 р.

ДОДАТКИ

ДОДАТОК А

Основний файл App.cs

```
using ...

namespace AMD_Master.Presentation_Layer;

///     Interaction logic for App.xaml ...
 Ihor Ihorevych
public partial class App : Application
{
    Private Fields

    #region Protected Methods

    /// <inheritdoc />
     Ihor Ihorevych
    protected override void OnExit(ExitEventArgs e){...}

    /// Handles the startup process of the application, including configuring and initializing services, creating the ...
     Ihor Ihorevych
    protected override void OnStartup(StartupEventArgs e){...}

    #endregion
}
```

ДОДАТОК Б

Файл DependencyInjection.cs

```
using ...

namespace AMD_Master.Presentation_Layer;

/// <summary>
/// Provides methods for configuring dependency injection in the application.
/// </summary>
Ihor Ihorevych *
internal static class DependencyInjection
{
    #region Public Methods

    /// <summary>
    /// Adds and configures application services to the specified IServiceCollection.
    /// </summary>
    /// <param name="services">The IServiceCollection to add services to.</param>
    1 usage   Ihor Ihorevych *
    public static void AddServices(this IServiceCollection services){...}

    #endregion
}
```

Файл UserSettings.cs

```
public interface IUserSettings
{
    #region Public Properties

    /// <summary>
    /// Gets or sets the configuration file path.
    /// </summary>
    5 usages  1 implementation  Ihor Ihorevych
    string ConfigurationFilePath { get; set; }

    /// <summary>
    /// Gets or sets the last modified date and time of the settings.
    /// </summary>
    2 usages  1 implementation  Ihor Ihorevych *
    DateTimeOffset LastModified { get; set; }

    #endregion

    #region Public Methods

    /// <summary>
    /// Saves the user settings to a file.
    /// </summary>
    1 implementation  Ihor Ihorevych
    void Save();

    /// <summary>
    /// Saves the user settings to a file asynchronously.
    /// </summary>
    /// <param name="cancellationToken">An optional cancellation token to cancel the operation.</param>
    /// <returns>A task that represents the asynchronous operation.</returns>
    1 usage  1 implementation  Ihor Ihorevych
    Task SaveAsync(CancellationToken cancellationToken = default);

    #endregion
}

/// <inheritdoc cref="IUserSettings"/>
9 usages  Ihor Ihorevych *
public partial class UserSettings : ObservableObject, IUserSettings
{
    Private Fields

    #region Public Constructors

    /// Creates a new instance of the <see cref="UserSettings"/> class, optionally creating a default settings file i ...
    5 usages  Ihor Ihorevych
    public UserSettings(bool firstTime){...}

    #endregion

    #region Private Constructors

    /// Creates a new instance of the <see cref="UserSettings"/> class, loading the settings from the file if it exis ...
    1 usage  Ihor Ihorevych
    private UserSettings() : this( firstTime: false){...}

    #endregion

    Public Methods

    #region Private Methods

    ///  Creates default instance of <see cref="UserSettings"/> ...
    2 usages  Ihor Ihorevych *
    private static IUserSettings CreateDefault(){...}

    /// Creates a default configuration file if it doesn't already exist. ...
    1 usage  Ihor Ihorevych
    private void CreateDefaultIfNotExists(){...}

    /// <summary>
    /// Loads the user settings from the file at the specified file path, if it exists.
    /// </summary>
    /// <remarks>
    /// If the file does not exist, this method does nothing. Otherwise, it deserializes the contents of the file
    /// using the JSON format and sets the properties of this object to match the deserialized data.
    /// </remarks>
    1 usage  Ihor Ihorevych
    private void LoadIfExists(){...}

    #endregion
}
```

ДОДАТОК Г

Файл INavigableView.cs

```
namespace AMD_Master.Presentation_Layer.Pages

    /// <summary>
    /// Represents a view that can navigate to other views and has an associated view model.
    /// </summary>
    /// <typeparam name="T">The type of the view model.</typeparam>
    2 usages  4 inheritors  Ihor Ihorevych
    internal interface INavigableView<out T>
    {
        #region Public Properties

        /// <summary>
        /// Gets the view model associated with this view.
        /// </summary>
        2 implementations  Ihor Ihorevych
        T ViewModel { get; }

        #endregion
    }
```

Файл NotificatonService.cs

```
internal class NotificatonService : INotificationService
{
    Private Fields

    Public Constructors

    #region Public Methods

    5 usages  new *
    public Task ShowAsync(string message, string title, NotificationType type){...}

    /// <inheritdoc />
    0+7 usages  Ihor Ihorevych
    public Task ErrorAsync(string message, string title = "Error") => ShowAsync(message, title, NotificationType.Error);

    /// <inheritdoc />
    0+1 usages  Ihor Ihorevych
    public Task InfoAsync(string message, string title = "Info") => ShowAsync(message, title, NotificationType.Information);

    /// <inheritdoc />
    Ihor Ihorevych
    public Task NotificationAsync(string message, string title = "Notification") => ShowAsync(message, title, NotificationType.Notification);

    /// <inheritdoc />
    0+3 usages  Ihor Ihorevych
    public Task SuccessAsync(string message, string title = "Success") => ShowAsync(message, title, NotificationType.Success);

    /// <inheritdoc />
    0+2 usages  Ihor Ihorevych
    public Task WarningAsync(string message, string title = "Warning") => ShowAsync(message, title, NotificationType.Warning);

    #endregion
}
```

```
/// <summary>
/// Defines an interface for displaying various types of notifications.
/// </summary>
6 usages  1 inheritor  Ihor Ihorevych
public interface INotificationService
{
    #region Public Methods

    /// Shows an error notification with a specified message and an optional title. ...
    7 usages  1 implementation  Ihor Ihorevych
    Task ErrorAsync(string message, string title = "Error");

    /// Shows an info notification with a specified message and an optional title. ...
    1 usage  1 implementation  Ihor Ihorevych
    Task InfoAsync(string message, string title = "Info");

    /// Shows a notification with a specified message and an optional title. ...
    1 implementation  Ihor Ihorevych
    Task NotificationAsync(string message, string title = "Notification");

    /// Shows a notification with a specified message, title, and type. ...
    1 implementation  Ihor Ihorevych
    Task ShowAsync(string message, string title, NotificationType type);

    /// Shows a success notification with a specified message and an optional title. ...
    3 usages  1 implementation  Ihor Ihorevych
    Task SuccessAsync(string message, string title = "Success");

    /// Shows a warning notification with a specified message and an optional title. ...
    2 usages  1 implementation  Ihor Ihorevych
    Task WarningAsync(string message, string title = "Warning");

    #endregion
}
```

ДОДАТОК Д

Файл PageService.cs

```
/// <summary>
/// Provides a service for retrieving pages from the dependency injection container.
/// </summary>
1 usage  Ihor Ihorevych
public class PageService : IPageService
{
    Private Fields

    #region Public Constructors

    /// Initializes a new instance of the <see cref="PageService"/> class with the specified dependency injection con ...
    Ihor Ihorevych
    public PageService(IServiceProvider serviceProvider){...}

    #endregion

    #region Public Methods

    /// Gets the page of type <typeparamref name="T"/> from the dependency injection container. ...
    0+1 usages  Ihor Ihorevych
    public FrameworkElement GetPage<T>() where T : Page => _serviceProvider.GetRequiredService<T>();

    /// Gets the page of the specified type from the dependency injection container. ...
    0+1 usages  Ihor Ihorevych
    public FrameworkElement GetPage(Type pageType) => (FrameworkElement)_serviceProvider.GetRequiredService(pageType);

    #endregion
}
```

ДОДАТОК Е

Файл ConfigurationViewModel.cs

```
public partial class ConfigurationViewModel : ObservableObject
{
    Private Fields

    Public Constructors

    #region Private Methods

    /// Assigns the properties of the loaded SystemModel to the corresponding bindable properties. ...
    1 usage  Ihor Ihorevych
    private void AssignPropertiesToBindablePropeties(SystemModel model){...}

    /// Handles the Load XML button click event, allowing the user to load an XML model from a specified location. ...
    [RelayCommand]
    2 usages  Ihor Ihorevych
    private async Task LoadXmlButtonClicked(){...}

    /// Handles the Save As button click event, allowing the user to save the loaded XML model to a specified locatio ...
    [RelayCommand]
    2 usages  Ihor Ihorevych
    private async Task OnSaveAsButtonClicked(){...}

    /// Updates the values of the loaded SystemModel based on the current bindable properties. ...
    [RelayCommand]
    2 usages  Ihor Ihorevych
    private async Task UpdateModelValues(){...}

    #endregion
}
```

ДОДАТОК Є

Файл MainWindowViewModel.cs

```
4 usages   Ihor Ihorevych   21 exposing API
public partial class MainWindowViewModel : ObservableObject
{
    #region Private Fields

    private readonly IPageService _pageService;

    [ObservableProperty] private FrameworkElement _frameContent;

    #endregion

    #region Public Constructors

    Ihor Ihorevych
    public MainWindowViewModel(IPageService pageService){...}

    #endregion

    #region Private Methods

    [RelayCommand]
    2 usages   Ihor Ihorevych
    private void NavigateToPage(Type type){...}

    #endregion
}
```

ДОДАТОК Ж

Файл SettingsViewModel.cs

```
3 usages  Ihor Ihorevych  1 exposing API
public partial class SettingsViewModel : ObservableObject
{
    #region Private Fields

    private readonly INotificationService _notificationService;

    [ObservableProperty] private IUserSettings _userSettings;

    #endregion

    Public Constructors

    #region Private Methods

    /// Opens a file dialog to allow the user to browse for and select a configuration file. ...
    [RelayCommand]
    2 usages  Ihor Ihorevych
    private Task BrowseConfigurationFilePath(){...}

    /// Saves user settings asynchronously and displays a success notification. ...
    [RelayCommand]
    2 usages  Ihor Ihorevych
    private Task SaveSettings(){...}

    #endregion
}
```

ДОДАТОК 3

Файл WelcomeViewModel.cs

```
using CommunityToolkit.Mvvm.ComponentModel;

namespace AMD_Master.Presentation_Layer.ViewModels;

3 usages  Ihor Ihorevych  1 exposing API
public partial class WelcomeViewModel : ObservableObject
{
    #region Public Properties

    public string Title { get; } = "Home";

    #endregion
}
```

ДОДАТОК И

Файл CPUModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a CPU model with a list of features. ...
14 usages  Ihor Ihorevych  1 exposing API
public class CpuModel : IEquatable<CpuModel>
{
    #region Public Properties

    /// Gets or sets the list of features for the CPU model. ...
    [XmlElement("FEATURE")]
    8 usages
    public List<FeatureModel> Features { get; set; }

    #endregion

    #region IEquatable Implementation

    2 usages  Ihor Ihorevych
    public bool Equals( [CanBeNull] CpuModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj) => Equals(obj as CpuModel);

    1 usage  Ihor Ihorevych
    public override int GetHashCode() => HashCode.Combine(Features);

    Ihor Ihorevych
    public static bool operator ==(CpuModel left, CpuModel right) => Equals(left, right);

    Ihor Ihorevych
    public static bool operator !=(CpuModel left, CpuModel right) => !Equals(left, right);

    #endregion
}
```

Файл FeatureModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a feature model with an ID, enabled status, and a list of states. ...
31 usages  Ihor Ihorevych
public class FeatureModel : IEquatable<FeatureModel>
{
    #region Public Enums

    /// Enumerates the known feature definitions for the system. ...
    6 usages  Ihor Ihorevych  4 exposing APIs
    public enum Definition{...}

    #endregion

    #region Public Properties

    /// Gets or sets the enabled status of the feature as a string. ...
    [XmlAttribute("Enabled")]
    15 usages
    public string Enabled { get; set; }

    /// Gets or sets the ID of the feature. ...
    [XmlAttribute("ID")]
    21 usages
    public int Id { get; set; }

    /// Gets or sets the list of states for the feature. ...
    [XmlElement("STATES")]
    14 usages
    public StatesModel States { get; set; }

    1 usage  Ihor Ihorevych
    public bool Equals(FeatureModel other){...}

    Ihor Ihorevych
    public override bool Equals(object obj) => Equals(obj as FeatureModel);

    Ihor Ihorevych
    public override int GetHashCode() => (Id, Enabled).GetHashCode();
    #endregion
}
```

Файл GPUModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a GPU model with a device ID, revision ID, PPW model, and a list of features. ...
16 usages  Ihor Ihorevych  1 exposing API
public class GpuModel : IEquatable<GpuModel>
{
    #region Public Properties

    /// Gets or sets the device ID of the GPU model. ...
    [XmlAttribute("DevID")]
    11 usages
    public string DevId { get; set; }

    /// Gets or sets the list of features for the GPU model. ...
    [XmlElement("FEATURE")]
    16 usages
    public List<FeatureModel> Features { get; set; }

    /// Gets or sets the PPW model for the GPU model. ...
    [XmlElement("PPW")]
    11 usages
    public PpwModel Ppw { get; set; }

    /// Gets or sets the revision ID of the GPU model. ...
    [XmlAttribute("RevID")]
    11 usages
    public string RevId { get; set; }

    #endregion

    #region Public Methods

    2 usages  Ihor Ihorevych
    public bool Equals( [CanBeNull] GpuModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj){...}

    1 usage  Ihor Ihorevych
    public override int GetHashCode()
    {
        unchecked{...}
    }
    #endregion
}
```

ДОДАТОК Й

Файл PPWModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a PPW (Power, Performance, and Wattage) model with a value. ...
10 usages  Ihor Ihorevych *  1 exposing API
public class PpwModel : IEquatable<PpwModel>
{
    #region Public Properties

    /// Gets or sets the value of the PPW model. ...
    [XmlAttribute("Value")]
    11 usages
    public int Value { get; set; }

    #endregion

    #region Public Methods

    2 usages  Ihor Ihorevych
    public bool Equals( [CanBeNull] PpwModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj){...}

    1 usage  Ihor Ihorevych
    public override int GetHashCode(){...}

    #endregion
}
```

ДОДАТОК К

Файл StateModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a state model with an ID, enabled status, and a value. ...
22 usages  Ihor Ihorevych
public class StateModel : IEquatable<StateModel>
{
    #region Public Properties

    /// Gets or sets the enabled status of the state as a string. ...
    [XmlAttribute("Enabled")]
    16 usages
    public string Enabled { get; set; }

    /// Gets or sets the ID of the state. ...
    [XmlAttribute("ID")]
    19 usages
    public int Id { get; set; }

    /// Gets or sets the value of the state. ...
    [XmlAttribute("Value")]
    23 usages
    public int Value { get; set; }

    #endregion

    #region Public Methods

    2 usages  Ihor Ihorevych
    public bool Equals( [CanBeNull] StateModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj){...}

    1 usage  Ihor Ihorevych
    public override int GetHashCode(){...}

    #endregion
}
```

ДОДАТОК К

Файл StatesModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a collection of state models in a list. ...
12 usages  Ihor Ihorevych  1 exposing API
public class StatesModel : IEquatable<StatesModel>
{
    #region Public Properties

    /// Gets or sets the list of state models. ...
    [XmlElement("STATE")]
    27 usages
    public List<StateModel> StateList { get; set; }

    #endregion

    #region Public Methods

    1 usage  Ihor Ihorevych
    public bool Equals( [CanBeNull] StatesModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj){...}

    Ihor Ihorevych
    public override int GetHashCode(){...}

    #endregion
}
```

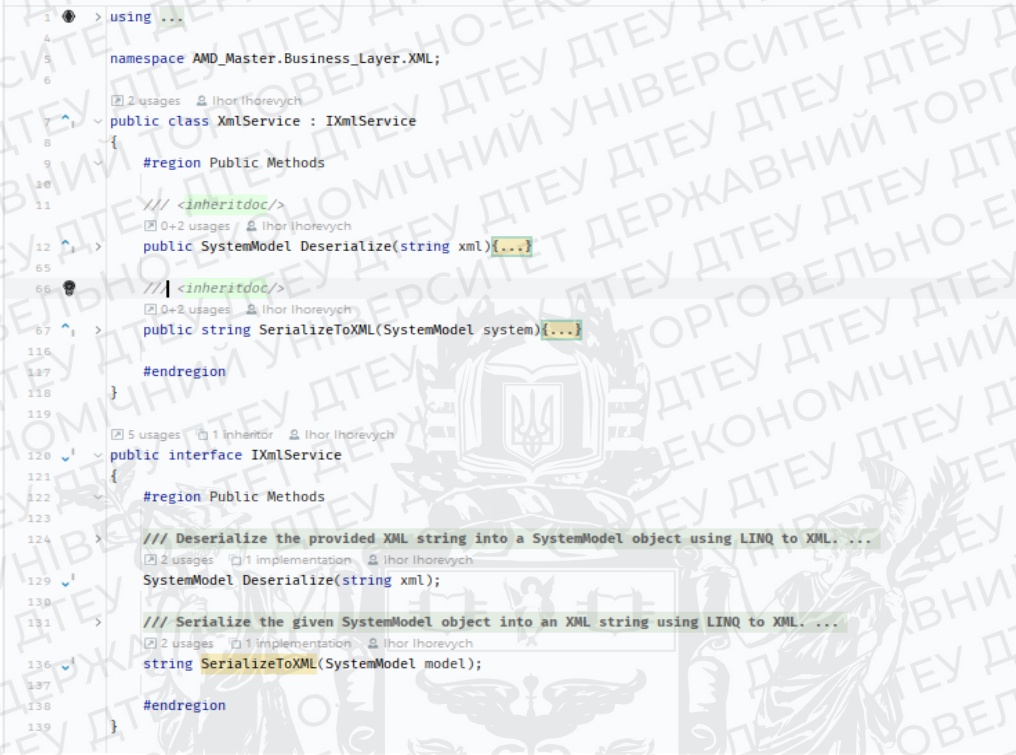
ДОДАТОК Л

Файл SystemModel.cs

```
using System.Xml.Serialization;

namespace AMD_Master.Data_Layer.XML;

/// Represents a system model with a CPU model and a GPU model. ...
[Serializable, XmlRoot( elementName: "SYSTEM")]
18 usages   Ihor Ihorevych   2 exposing APIs
public class SystemModel : IEquatable<SystemModel>
{
    #region Public Properties

    /// Gets or sets the CPU model for the system. ...
    [XmlElement("CPU")]
    10 usages
    public CpuModel Cpu { get; set; }

    /// Gets or sets the GPU model for the system. ...
    [XmlElement("GPU")]
    25 usages
    public GpuModel Gpu { get; set; }

    #endregion

    #region IEquatable Implementation

    1 usage   Ihor Ihorevych
    public bool Equals( [CanBeNull] SystemModel other){...}

    Ihor Ihorevych
    public override bool Equals( [CanBeNull] object obj){...}

    Ihor Ihorevych
    public override int GetHashCode(){...}

    #endregion
}
```

ДОДАТОК М

Файл XmlService.cs



```
using ...

namespace AMD_Master.Business_Layer.XML;

2 usages  Ihor Ihorevych
public class XmlService : IXmlService
{
    #region Public Methods

    /// <inheritdoc/>
    0+2 usages  Ihor Ihorevych
    public SystemModel Deserialize(string xml){...}

    /// <inheritdoc/>
    0+2 usages  Ihor Ihorevych
    public string SerializeToXML(SystemModel system){...}

    #endregion
}

5 usages  1 inheritor  Ihor Ihorevych
public interface IXmlService
{
    #region Public Methods

    /// Deserialize the provided XML string into a SystemModel object using LINQ to XML. ...
    2 usages  1 implementation  Ihor Ihorevych
    SystemModel Deserialize(string xml);

    /// Serialize the given SystemModel object into an XML string using LINQ to XML. ...
    2 usages  1 implementation  Ihor Ihorevych
    string SerializeToXML(SystemModel model);

    #endregion
}
```

ДОДАТОК Н

Файл ObjectExtensions.cs

```
using System.Runtime.Serialization;

namespace AMD_Master.Extension_Methods;

/// Provides extension methods for objects. ...
Ihor Ihorevych
public static class ObjectExtensions
{
    #region Public Methods

    /// Creates a deep copy of the input object of type T. ...
    4 usages   Ihor Ihorevych
    public static T DeepCopy<T>(this T input){...} using var stream

    #endregion
}
```

ДОДАТОК О

Файл XmlServiceTests.cs

```
using ...

namespace BusinessLayer_Tests;

Ihor Ihorevych *
public class XmlServiceTests
{
    [Fact]
    Ihor Ihorevych *
    public void XmlService_SerializeDeserialize_ReturnsEqualObjects()
    {
        // Arrange
        IXmlService service = new XmlService();
        SystemModel expectedModel = new SystemModel()
        {...};

        // Act
        string xmlString = service.SerializeToXML(expectedModel);
        SystemModel actualModel = service.Deserialize(xmlString);

        // Assert
        Assert.Equal( expected: expectedModel.Cpu.Features,  actual: actualModel.Cpu.Features);
        Assert.Equal( expected: expectedModel.Gpu.DevId,  actual: actualModel.Gpu.DevId);
        Assert.Equal( expected: expectedModel.Gpu.RevId,  actual: actualModel.Gpu.RevId);
        Assert.Equal( expected: expectedModel.Gpu.Ppw.Value,  actual: actualModel.Gpu.Ppw.Value);
        Assert.Equal( expected: expectedModel.Gpu.Features,  actual: actualModel.Gpu.Features);
    }
}
```

Файл CpuModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class CpuModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void CpuModel_GetSetFeatures_ReturnsCorrectValue()
    {
        // Arrange
        var model = new CpuModel();
        var expectedFeatures = new List<FeatureModel>()
        {...};

        // Act
        model.Features = expectedFeatures;

        // Assert
        Assert.Equal(expectedFeatures, actual: model.Features);
    }
}
```

ДОДАТОК Р

Файл FeatureModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class FeatureModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void FeatureModel_GetSetEnabled_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void FeatureModel_GetSetId_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void FeatureModel_GetSetStates_ReturnsCorrectValue(){...}
}
```

ДОДАТОК С

Файл GpuModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class GpuModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void GpuModel_GetSetDevId_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void GpuModel_GetSetFeatures_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void GpuModel_GetSetPpw_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void GpuModel_GetSetRevId_ReturnsCorrectValue(){...}
}
```

ДОДАТОК Т

Файл PpwModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class PpwModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void PpwModel_GetSetValue_ReturnsCorrectValue(){...}
}
```

ДОДАТОК У

Файл StateModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class StateModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void StateModel_GetSetEnabled_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void StateModel_GetSetId_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void StateModel_GetSetValue_ReturnsCorrectValue(){...}
}
```

ДОДАТОК Ф

Файл StatesModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;
 Ihor Ihorevych *
public class StatesModelTests
{
    [Fact]
     Ihor Ihorevych *
    public void StatesModel_GetSetStateList_ReturnsCorrectValue(){...}
}
```

ДОДАТОК Х

Файл SystemModelTests.cs

```
using AMD_Master.Data_Layer.XML;

namespace DataLayerTests;

Ihor Ihorevych *
public class SystemModelTests
{
    [Fact]
    Ihor Ihorevych *
    public void SystemModel_GetSetCpu_ReturnsCorrectValue(){...}

    [Fact]
    Ihor Ihorevych *
    public void SystemModel_GetSetGpu_ReturnsCorrectValue(){...}
}
```

ДОДАТОК Ц

Файл ObjectExtensionsTests.cs

```
using AMD_Master.Extension_Methods;

namespace ExtensionMethodsTests;

Ihor Ihorevych *
public class ObjectExtensionsTests
{
    [Fact]
    new *
    public void DeepCopy_NullInput_ThrowsArgumentNullException(){...}

    [Fact]
    Ihor Ihorevych *
    public void DeepCopy_ValueType_ReturnsCopy(){...}

    [Fact]
    Ihor Ihorevych *
    public void DeepCopy_ReferenceType_ReturnsCopy(){...}

    [Fact]
    Ihor Ihorevych *
    public void DeepCopy_MutableReferenceType_ReturnsCopy(){...}
}
```

ДОДАТОК Ч

Файл UnitTest1.cs

```
using AMD_Master.Presentation_Layer.Models.SettingsPage;

using Newtonsoft.Json;

namespace PresentationLayerTests;

Ihor Ihorevych *
public class UserSettingsTests
{
    private readonly string _filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "settings.json");

    [Fact]
    Ihor Ihorevych *
    public void Constructor_WithTrueArgument_CreatesNewSettingsFile(){...}

    [Fact]
    Ihor Ihorevych *
    public void Save_SavesSettingsToFile(){...}

    [Fact]
    Ihor Ihorevych *
    public async Task SaveAsync_SavesSettingsToFileAsync(){...}
```