Довідка
про відсутність плагіату у випускній кваліфікаційній роботі

Міністерство освіти і науки України
Український державний університет науки та технологій

Кафедра «Комп'ютерні інформаційні технології»

ДОВІДКА

За результатами перевірки випускної кваліфікаційної роботи здобувача вищої освіти *Сироти Олександра Анатолійовича*
(прізвище, ім'я, по батькові)
на тему: *Дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS*
в роботі не виявлено порушень академічної доброчесності.

Керівник ВКР _____ Олександра ГОРБОВА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

_____ /Вадим ГОРЯЧКІН/

«20» 12 2021 р.

ДИПЛОМНА РОБОТА

на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS**

Theme **Research of the consequences of using patterns and common approaches in building the architecture of cross-platform applications for Android and iOS**

Керівник дипломної роботи доц. _____ Олександра ГОРБОВА

Нормоконтролер доц. _____ Олена КУРОП'ЯТНИК

Студент групи ПЗ2021 _____ Олександр СИРОТА

Student Oleksandr SYROTA

Дніпро – 2021

2

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій та систем
Кафедра Комп'ютерні інформаційні технології
Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ /В.І. ШИНКАРЕНКО/
«____» _____ 20___ р.

ЗАВДАННЯ
до дипломної роботи на здобуття ОС ___магістр___
(освітній ступінь)
студента групи ПЗ32021 (961 М) Сирота Олександр Анатолійович
(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS
затверджена наказом по університету від «18» листопада 2020 р. № 690 ст.

2 Термін подання студентом закінченої роботи 3 грудня 2021 р.

3 Вихідні дані до дипломної роботи _____

4 Зміст пояснювальної записки (перелік питань до розробки) проведення дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS.

5 Перелік демонстраційного матеріалу презентація на тему дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS, результати проведених досліджень, демонстраційні матеріали роботи розробленого програмного забезпечення для проведення дослідження за темою дипломної роботи.

3

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	Микола ГНЕННИЙ		
Охорона праці	Олег САБЛІН		

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва розділів дипломної роботи	Термін виконання розділів роботи	Примітка
1	Вступ		
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами		від 70 джерел
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження		
4	Постановка задачі, технічне завдання	13.10.21 – 21.10.21	30%
5	Техніко-економічні показники		
6	Розробка інструментальних засобів дослідження		
7	Виконання досліджень	08.11.21 – 19.11.21	60%
8	Оформлення тез доповідей		
9	Оформлення статті у фаховий журнал		
10	Оформлення пояснювальної записки		
11	Розробка демонстраційних матеріалів	25.11.21 – 03.12.21	100%

Дата видачі завдання «18» листопада 2021р.

Керівник дипломної роботи _____ (підпис) Гордова О.В. (ПІБ)

Завдання прийняв до виконання _____ (підпис) О.А. Сирота (ПІБ)

РЕФЕРАТ

В даній магістерській роботі об’єктом дослідження виступає один із етапів життєвого циклу розробки програмного продукту – проектування архітектури.

Предметом дослідження є патерни та ефективність їх використання при проектуванні мобільного додатку.

За мету роботи вважається аналіз підходів до розробки з надмірним або недостатнім використанням принципів та шаблонів проектування, а також аналіз результатів їх використання.

Використані методі дослідження: теоретичний метод - моделювання, аналіз, порівняння та абстрагування, а також емпіричний метод дослідження.

Результати досліджень та практичне значення: Результати виконаних досліджень можуть бути використані викладачами вищих навчальних закладів, студентами та програмістами на комерційних проектах.

Пояснювальна записка містить вступ до роботи, 5 розділів, список використаних джерел та п’яти додатків. Вступ складається з 4 сторінок та описує суть роботи. Перший розділ складається з 26 сторінок та відображає аналіз сучасного стану дослідження. Другий розділ складається з 7 сторінок та обґрунтовує експериментальний метод дослідження. Третій розділ включає 20 сторінок, відображає етапи розробки і проектування програмного забезпечення. Четвертий розділ складається з 18 сторінок та описує проведення експериментів. П’ятий розділ включає в себе 15 сторінок та висвітлює інформацію стосовно безпечної праці. Додатки складаються з програмного коду, технічного завдання і керівництва користувача. Рисунків – 43, таблиць – 12, літературних джерел – 50. Ключові слова: патерн, архітектура, статичний аналізатор, програмний продукт, проектування.

Зміст

Список скорочень

1. MVP – продукт з мінімальною, але достатню реалізацію функціоналу.
2. MVVM (Model-View-ViewModel) – паттерн, для відокремлення бізнес логіки додатку від візуальної частини.
3. IoC (Inversion of Control) – важливий принцип об'єктно-орієнтованого програмування.
4. Фабричний метод (Factory Method) – шаблон, який надає можливість дочірнім класам створювати екземпляри певного класу.
5. SOLID – список принципів дизайну програмного забезпечення в об'єктно-орієнтованому програмуванні.
6. KISS-принцип – принцип проектування.
7. MSDN – це офіційна бібліотека технічної документації.
8. SAAM – метод аналізу архітектури програмного забезпечення.
9. Android, IOS – операційні системи.
10. ATAM – метод аналізу архітектурних компромісів.
11. CBAM – метод аналізу рентабельності.
12. FAAM – метод оцінки сімейства архітектур.
13. ALMA – метод аналізу модифікованості на рівні архітектури.
14. XAML – мова на основі XML.

ВСТУП

Актуальність роботи. Кожного дня з’являються все нові й нові ідеї для створення різних програмних продуктів. Беручи до уваги той факт, що в наш час важко уявити людину яка не користувалася б смартфоном, то основним напрямком при створенні багатьох додатків є саме такі операційні системи як iOS та Android. В таких мобільних маркетах як App Store або Google Play можна знайти все що заманеться: різні трекери, ігри, фото та відео редактори, додатки які орієнтовані на мед-послуги, бронювання столиків в ресторані та замовлення їжі, операції з нерухомістю, розваги та інше. Одним із невід’ємних етапів життєвого циклу крос – платформних додатків для iOS та Android є побудова архітектури. Використання патернів при побудові архітектури є фундаментом майбутнього додатку.

Виходячи з того, що патерн проектування – це найбільш поширене вирішенні певної задачі при проектування архітектури програми, то можна сказати, що ця тема є досить важливою, так як кожного року створюється все більше і більше програм різного розміру і з різним функціоналом, які в майбутньому можуть допрацьовуватись або змінюватись. При їх проектуванні використовуються різні шаблони проектування, деякі із задач, які вирішують ці програми, досить часто вже вирішувались раніше.

Не завжди використання патернів та загально-прийнятих практик позитивно впливає на кінцевий програмний продукт. Воно може бути надмірне або недостатнє, тому і вплив на кінцевий результат, тобто існування, функціонування, підтримку та розширення можливостей програмного продукту в майбутньому може бути різний при будь-якому із підходів. Важливо не тільки вміти будувати архітектуру, використовуючи відомі для цього «інструменти», але

й розуміти в якій мірі це повинно бути реалізовано і який вплив це матиме на програмний продукт в подальшому.

Об’єктом дослідження є один із процесів життєвого циклу розробки програмного продукту.

Предметом дослідження є патерни та ефективність їх використання при проектуванні мобільного додатку.

Мета роботи полягає в аналізі підходів до розробки з надмірним або недостатнім використанням принципів та шаблонів проектування, а також аналіз результатів кінцевого продукту, крос-платформного програмного забезпечення для операційних систем iOS та Android. Та їх вплив на складність розуміння реалізації, існування, функціонування, підтримку та розширення можливостей програмного продукту з точки зору розробника.

Поставлена мета зумовила необхідність вирішення такого комплексу взаємопов’язаних завдань:

1. ознайомлення з часто – використовуваними патернами;
2. ознайомлення з основними загально – відомими практиками побудови архітектури;
3. перегляд Open – source проектів на GitHub, GitLab, Bitbucket;
4. оцінка реалізації проектів та впливу на них вживаних підходів;
5. виведення критеріїв для розуміння в якій мірі слід використовувати той, чи інший «інструмент».

Для аналізу підходів до розробки з надмірним або недостатнім використанням принципів та шаблонів проектування, а також аналізу результатів кінцевого продукту, крос-платформного програмного забезпечення для операційних систем iOS та Android було використано теоретичний метод дослідження: моделювання, аналіз, порівняння та абстрагування.

Наприклад при використанні методу порівняння було переглянуто та протиставлено декілька проектів з застосуванням або ігноруванням такого SOLID принципу як «Dependency Inversion». При чому в той же час було використано абстрагування від інших присутніх або відсутніх підходів в проектуванні ПЗ.

Окрім теоретичного методу дослідження було використано емпіричний метод для розбору складності швидкого розуміння логіки та редагування того чи іншого програмного модулю.

Наукова новизна роботи полягає у визначенні необхідної міри та наслідків використання шаблонів проектування. Приведення користі та прикладів використання тих чи інших патернів та підходів в проектуванні крос-платформних додатків відбувається досить давно, але рідко йде річ про те, в якій мірі вони повинні бути реалізовані на проектах різного масштабу та з різною функціональністю. Вперше було проведено аналіз необхідної міри використання шаблонів проектування на різних за розміром та призначенням мобільних додатках.

З точки зору наукової новизни було виконано доповнення, розширення та уточнення відомих даних.

Практичне значення. Результати виконаних досліджень дозволять програмісту краще розуміти як проектувати додатки для операційних систем iOS та Android. В якому разі слід застосувати якийсь з відомих шаблонів проектування, а в якому краще цього не робити, аби не нашкодити.

Отримана інформація може бути використана викладачами вищих навчальних закладів, як засіб для надання практичних прикладів та демонстрації для учнів, а також студентами при виконанні практичних робіт та власне програмістами на реальних комерційних проектах.

Апробація результатів дослідження. Основні положення магістерської роботи доповідалися та були схвалені:

- на чотирнадцятій міжнародній науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті», яка відбулася в Дніпровському національному університеті залізничного транспорту імені академіка В. Лазаряна (ДНУЗТ) 15 – 16 грудня 2020 року;
- на 81 Всеукраїнській науково-технічній конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту» яка відбулася в Дніпровському національному університеті залізничного транспорту імені академіка В. Лазаряна (ДНУЗТ) 28 жовтня 2021 року.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ДОСЛІДЖЕННЯ НАСЛІДКІВ ВИКОРИСТАННЯ ПАТЕРНІВ І ЗАГАЛЬНОПРИЙНЯТИХ ПІДХОДІВ В ПОБУДОВІ АРХІТЕКТУРИ КРОС-ПЛАТФОРМНИХ ДОДАТКІВ ПІД ANDROID І IOS.

1.1 Аналіз предметної області

Загальноприйнятими підходами в побудові програмних додатків слід вважати принципи та рекомендації побудови архітектури, які багато разів перевірені на практиці та зарекомендували себе як спосіб покращення програмного забезпечення на різних етапах життєвого циклу.

В області розробки мобільних додатків або інших продуктів патерном або шаблоном проектування (Design Pattern) називається вже знайдене рішення задачі, що досить часто зустрічаються на практиці. Це рішення використовується повторно знову і знову, тому і набуло таку назву як шаблон. Зазвичай вони представленні у вигляді схеми або діаграми. Нижче представлене визначення цього поняття, яке можна знайти в мережі інтернет.
У програмній інженерії шаблон проектування є загальним повторюваним рішенням поширеної проблеми в розробці програмного забезпечення. Шаблон дизайну – це не готовий дизайн, який можна перетворити безпосередньо в код. Це опис або шаблон для вирішення проблеми, який можна використовувати в багатьох різних ситуаціях [1].

Основна ідея використання патернів полягає в створенні ієрархій класів певного виду при розв’язанні певної задачі. Також треба сказати, що застосування патернів або загальноприйнятих не “прив’язується” до конкретної мови програмування, яка використовується, а отже підтримується усіма об’єктно - орієнтованими мовами. Основними причинами широкого використання різних шаблонів є наступні:

- з’являється можливість для повторного використання коду (code reuse) у цьому ж або іншому програмному додатку;
- змога легко супроводжувати продукт та надає можливість додавати, змінювати або видаляти різні частини незалежно одна від одної з мінімальними витратами в часі, кількості розробників та з мінімальною складністю;
- полегшення комунікації розробників завдяки абстракціям які вносяться патернами [1];
- підтримує можливість легко та швидко редагувати всю систему вцілому;

Повертаючись до питання проектування мобільного додатку з використанням патернів можна сказати, що одним із головних завдань на цьому етапі є не лише розуміння того, що потрібно взяти той чи інший відомий патерн та застосувати його при проектуванні, а й розуміння, прогнозування наслідків його використання та аналіз того, в якій мірі це повинно бути. Простіше говорячи програміст повинен розуміти, а чи потрібно взагалі використовувати на даному конкретному проекті той, чи інший патерн, підхід. Чи можливо потрібно взагалі уникнути цього і зробити все якомога простіше та швидше. В такому разі йому потрібно опиратися на такі пункти, як:

1. вплив на складність розуміння реалізації проекту в подальшому;
2. вплив на функціонування програмного продукту;
3. вплив на етап тестування;
4. можливість для розширення функціоналу або заміни деяких модулів мобільного додатку;
5. бізнес цілі проекту;

Треба підкреслити, що можуть виникати ситуації коли ці пункти пожуть конфліктувати між собою або виключати один одного, наприклад п.1 та п.5. Такою ситуацією може бути розробка MVP продукту.

Minimum viable product або MVP – це продукт, що має мінімальну, але достатню реалізацію функціоналу для можливості користування і задоволення потреб споживачів. Метою його створення є отримання зворотного звя’язку від користувачів, планування майбутнього розвитку проекту та мінімізація ризиків на основі отриманих відгуків [2].

Розробка та аналіз MVP продукту дешевша ніж розробка повноцінного програмного додатку з максимально продуманою архітектурою та великим обсягом різних функцій. Такий додаток повинен бути створений та якнайшвидше випущений в магазин мобільних додатків, як макет з певним функціоналом лише для аналізу попиту на нього, виявлення зацікавленості користувачів у ньому. І якщо при аналізі стає зрозуміло, що користувачам цікавий такий продукт, то в майбутньому створюється повністю новий додаток, який вирішує ту ж саму проблему та буде підтримуватись та розширюватись [2].

Етапи життєвого циклу такого програмного продукту зображено на рис. 1.1 [3].

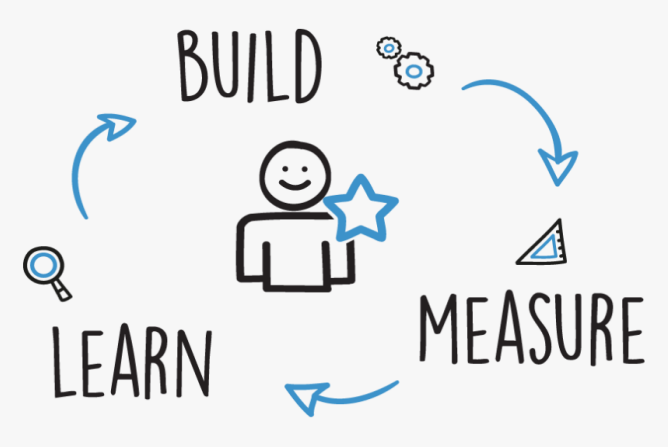


Рисунок 1.1 – Етапи життєвого циклу продукту з мінімально реалізацією

Слід зазначити, що незважаючи на те, що при створенні Minimum viable product додатку основний фокус не на побудові довготривалого проекту, а на аналізі можливого попиту це не означає, що розробник може на дизайн архітектури не звертати уваги взагалі. На рис. 1.2 показано якої уваги потребує кожна складова такого продукту [4].

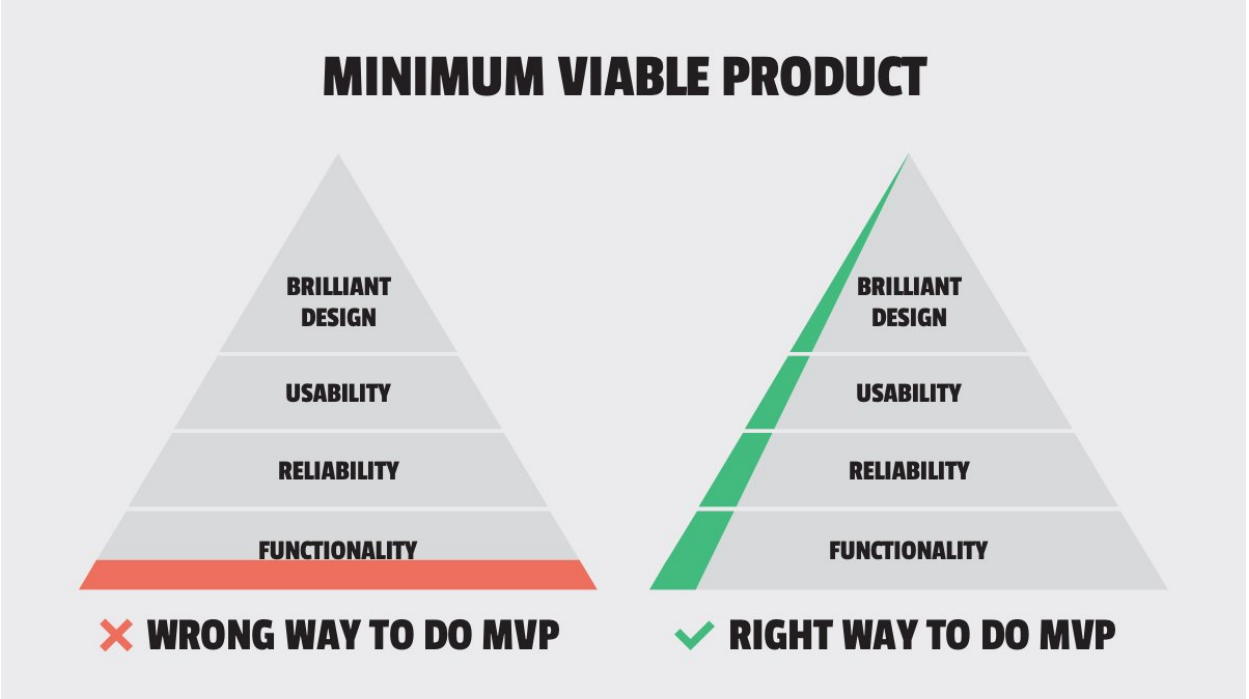


Рисунок 1.2 – Схема варіантів побудови продукту з мінімально реалізацією

Звичайно, більшість програмістів прагне мати ідеально – спроєктовану систему в якій були б враховані всі нюанси та розумно застосовані відомі підходи, що дало б свої плоди в майбутньому. Та чи було б виконане поставлене завдання? Розробник повинен в конкретній ситуації та на конкретному проекті зорієнтуватися чи потрібно його реалізовувати, чи ні. Досить часто бувають ситуації коли шаблон проектування може нашкодити внаслідок збільшення складності проекту, замість того, щоб покращити розробку і підтримку продукту.

Можна виокремити декілька найбільш поширених патернів та підходів, що використовуються при проектуванні крос-платформних додатків:

1.1.1 Патерн Model-View-ViewModel

MVVM (Model-View-ViewModel) – паттерн, що дозволяє відокремити бізнес логіку додатку від візуальної частини - інтерфейсу користувача. Даний патерн є архітектурним, тобто він задає загальну архітектуру програми та надає змогу тестувати модулі незалежного один від одного, вносити зміни до модулю незалежно від інших, повторно використовувати реалізовано логіку. Взаємодію між компонентами продемонстровано на рис. 1.3 [5, 6].

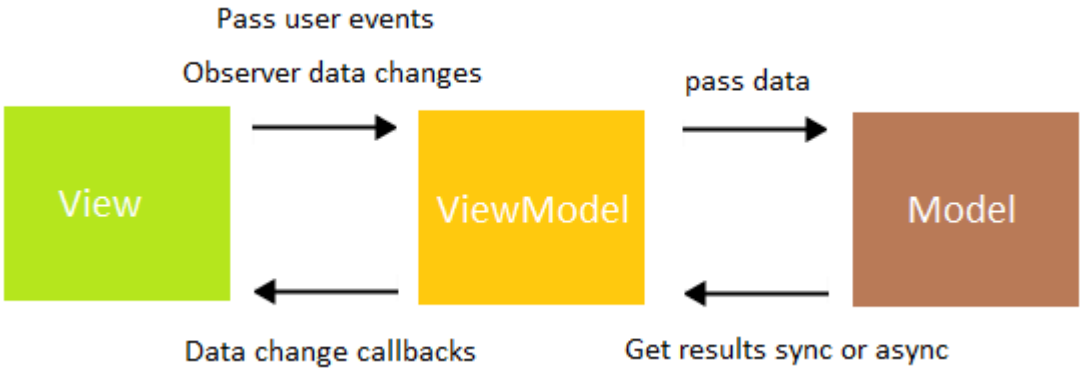


Рисунок 1.3 – Взаємодія MVVM компонентів

1.1.2 Принцип Inversion of Control

IoC (Inversion of Control) - важливий принцип об'єктно-орієнтованого програмування, що використовується для написання слабо зв'язаного коду. Суть якого в тому, що потік управління програми контролюється фреймворком. Також архітектурне рішення інтеграції, що спрощує розширення можливостей системи. Dependency Injection, Factory Method, Service Locator – можливі реалізації принципу Inversion of Control що продемонстровані на рис. 1.4 [7, 8, 9].

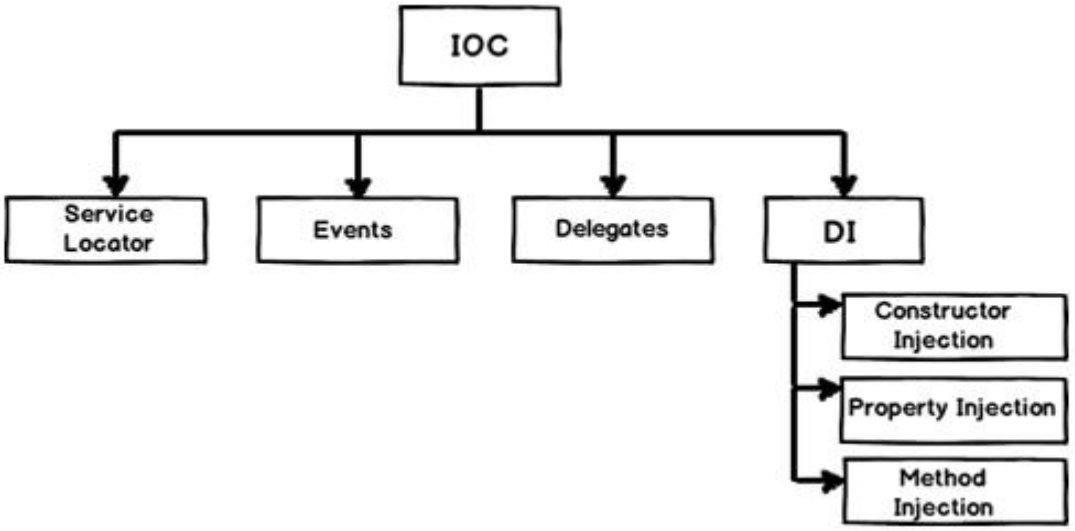


Рисунок 1.4 – Варіанти реалізації принципу IoC

1.1.3 Патерн Factory method

Фабричний метод (Factory Method) - шаблон, який надає можливість дочірнім класам створювати екземпляри певного класу за допомогою абстрактного інтерфейсу, що дозволяє використовувати абстракцію замість конкретної реалізації деяких сутностей. Схематичне представлення фабричного методу показане на рис. 1.5 [10].

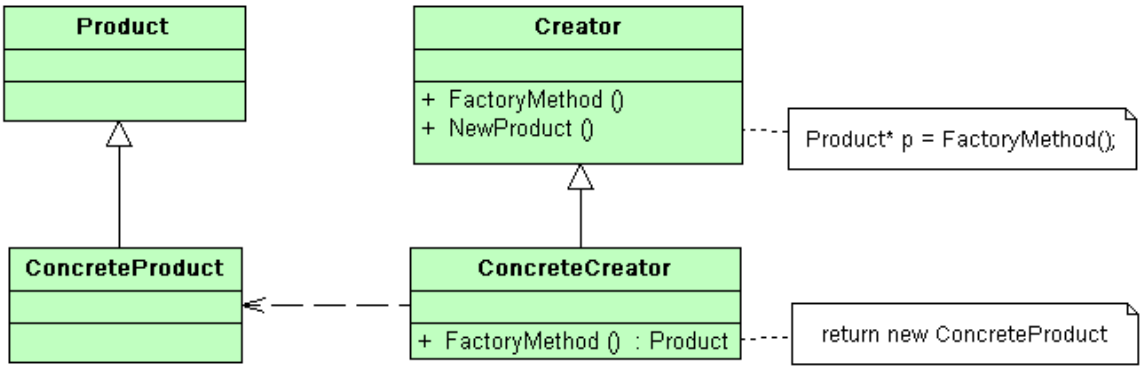


Рисунок 1.5 – зображення патерну Фабричний метод

1.1.4 Принципи SOLID

SOLID — це список шаблонів або принципів дизайну програмного забезпечення в об'єктно-орієнтованому програмуванні.

- S — принцип єдиної відповідальності (англ. *single responsibility principle*) говорить про те, що кожен модуль повинен мати лише одну відповідальність.
- O — принцип відкритості – закритості (англ. *open-closed principle*) - програмні сутності повинні бути відкриті для розширення і закриті для модифікації.
- L — принцип підстановки Барбари Лісков (англ. *Liskov substitution*) - підкласи можуть виступати заміною для супер класів без порушень роботи програми.
- I — принцип розподілу інтерфейсів (англ. *interface segregation principle*) - декілька вузько-направлених інтерфейсів краще, ніж один універсальний.
- D — принцип інверсії залежностей (англ. *dependency inversion principle*) – залежності між модулями програми повинні будуватися на основі абстракцій і уникати залежностей від конкретної реалізації.

Дотримання цих принципів дозволяє спроектувати таку систему, яку в подальшому буде легко змінювати, розширювати та підтримувати [11, 12].

1.1.5 Принцип KISS

KISS-принцип (англ. *KISS principle*) – принцип проектування, який говорить про те, що програмна система буде працювати краще якщо буде залишатися максимально простою, а не ускладнюватись. Ця абревіатура розшифровується як: “Keep it short and simple” [13].

При розробці програмного забезпечення треба уникати збільшення складності, яке досить часто трапляється внаслідок черезмірного або непотрібного використання патернів або інших принципів проектування.

1.2 Огляд останніх досліджень і публікацій

В наш час для розробників найбільш дорогоцінним ресурсом є час, тому при вирішенні якоїсь проблеми або необхідності певної підказки програміст буде використовувати саме ту літературу, яку йому надає мережа інтернет. Головною причиною цього є те, що пошук необхідної інформації проходить в декілька разів швидше ніж це було б при роботі з книгою. Також переважна більшість статей базуються на інформації з книг, але в більш зжатому варіанті, з розкриттям головного питання. Саме через це при огляді літературних джерел в основному увага приділялася різним ресурсам в мережі інтернет з інформацією на тему розробки та проектування мобільних додатків. Далі буде розглянуто найбільш популярні із них.

1.2.1 Огляд Medium

Medium – це швидка і проста в обігу блог-платформа від творців Blogger і Twitter. Місце, де своїми думками діляться користувачі (як в Twitter) та сам Medium (подібно BuzzFeed). На тему розробки мобільних додатків під платформи IOS та Android існує безліч написаних статей, бо досить велика частина аудиторії це розробники програмного забезпечення зі знанням різних технологій та різним рівнем досвіду. Інформація, яка пропонується авторами цієї платформи, покриває широкий спектр запитань з якими інший розробник потрапляє на неї. В основному це новини зі світу розробки мобільних додатків, опис нового функціоналу, який був реалізований в новій версії фреймворку, стислий переказ офіційної документації, приклади використання тієї чи іншої бібліотеки, приклади реалізації різного функціоналу і звичайно

статі на тему конструювання архітектури, огляд патернів та загальноприйнятих підходів при конструюванні крос-платформних мобільних додатків [14].

До плюсів літератури, яка пропонується авторами цього сайту можна віднести стисле викладення інформації, що дозволяє не витрачати багато часу на її вивчення. Також в переважній більшості пояснення матеріалу супроводжується схемами, ілюстраціями та прикладами простих проектів, що сприяє швидкому розумінню того, про що йде мова. Ще одним із плюсів є те, що можна знайти декілька статей на одну й ту ж саму тему, але від різних авторів, підходи яких можуть різнитися або доповнювати один одного.

Мінусами можна вважати те, що інформація на тему користі того, чи іншого патерну, необхідності його реалізації в розрізі бізнес цілі проекту, його розміру, бюджету та dead-line'ів приводиться також в неповному обсязі або не наводиться взагалі. Досить рідко можна знайти інформацію про те, які були наслідки використання певного патерну, чи призвело це до якихось проблем, чи насправді його реалізація була так необхідна та чи можна було його не використовувати.

1.2.2 Огляд Stack Overflow

Stack Overflow – система питань і відповідей про програмування, розроблена Джоелем Спольскі і Джеффом Етвуд (англ.) в 2008 році. В наш час цей ресурс є одним із найпопулярніших і найвідоміших серед розробників програмного забезпечення. Хоча це не варто називати документацією або навіть статтею та залишити його без уваги неправильно [15].

Плюсом є те, що в цій "базі знань" створений шляхом запитань і відповідей можна знайти інформацію не тільки про якийсь специфічний баг

на певній платформі, Android наприклад, а й пояснення патернів, приклади і ситуації необхідності їх використання, наслідки. І навіть якщо щось не вдалось знайти, то можна створити питання і отримати на нього відповідь з поясненням та посиланнями на інші літературні джерела. Вірогідність отримати більше детальної інформації значно вище ніж в звичайних статтях на сайтах. До того ж, розробники які відповідають на питання, або ставлять його, скоріш за все використовували різні підходи на різних проектах і можуть поділитися своїми спостереженнями.

До мінусів можна віднести той факт, що на поставлене запитання можна взагалі не отримати відповідь. Воно може бути проігнороване.

1.2.3 Огляд Habr

Habr – веб-сайт в форматі системи тематичних колективних блогів (іменованих хабами) з елементами новинного сайту, створений для публікації новин, аналітичних статей, думок, пов'язаних з інформаційними технологіями, бізнесом та інтернетом [16].

Переважна більшість статей на тему розробки програмного забезпечення досить об'ємні та несуть в собі багато інформації, яка базується на інших літературних джерелах, таких як книги, офіційна документація, інші статті та власний досвід. На даному сайті можна знайти багато даних стосовно побудови архітектури додатку, пояснення та приклади застосування патернів.

Трапляються доклади як розробників, які працюють у невеликих компаніях над невеликими за розміром проектах так і тих, хто працює у досить великих та відомих компаніях над не менш відомими програмними продуктами. Це дає можливість побачити та проаналізувати, як та чи інша команда програмістів використовує шаблони проектування в свої проектах та

приблизно уявити необхідність реалізації чогось подібного на своєму проекті. Також в деяких статтях приділяється увага питанню необхідності використання певного патерну на певному проекті. Все це супроводжується різними зображеннями, схемами та прикладами окремих частин коду, що може доповнюватись посиланням на GitHub, GitLab або Bitbucket.

1.2.4 Огляд офіційної документації

Існує багато офіційних джерел з документацією, одним з них є сервіс від компанії Microsoft.

Бібліотека MSDN (англ. *MSDN Library*) – бібліотека офіційної технічної документації для розробників під ОС Microsoft Windows. MSDN розшифровується як Microsoft Developer Network. Бібліотека MSDN містить документацію на API продуктів Microsoft, а також код прикладів, технічні статті та іншу корисну для розробників інформацію. Крім цього вона також включає різні приклади коду, технічні статті та іншу інформацію про програмування. Вся ця інформація доступна безкоштовно або за платної підписки. Крім звичайних статей на своєму сайті Microsoft також пропонує завантажити книгу [17].

Звичайно, що ця бібліотека також містить інформацію про патерни проектування. Викладення інформації високого рівня доповнюється схемами, прикладами коду, детальним поясненням за що відповідає той, чи інший компонент в реалізації і також нотатками із порадами та попередженнями про типові помилки. Також надається більш детальне пояснення навіщо взагалі використовувати патерн, про який йде річ у документації та з яких сутностей буде складатися проект у разі його реалізації.

1.3 Огляд програмних аналогів

Зазвичай аналіз побудови архітектури проекту, правильного вибору патернів та їх реалізації проводиться проводиться або на етапі проектування або безпосередньо в процесі ревізії коду (англ. *code review*) або інспекції коду(англ. *code inspection*). Code review – це перевірка програмного коду з метою виявлення не лише помилок реалізації функціоналу, розроблених алгоритмів, але й з метою аналізу архітектури проекту, що створюється. Цим займаються розробники різного рівню, з різними знаннями та різним досвідом, що надає цьому процесу деякої суб’єктивності. Суб’єктивності намагаються уникнути залучаючи декілька розробників та використовуючи програмне забезпечення для автоматизації процесу ревізії коду, що полегшує роботу та дозволяє аналізувати великі кількістю строк коду файли та навіть цілі проекти, що в свою чергу називають статичним аналізом коду.

Статичний аналіз коду (англ. *static code analysis*) – це процес аналізу програмного забезпечення за допомогою програмних засобів, що називаються статичними аналізаторами коду. Процедура перевірки коду проводиться без виконання досліджуваної програми, тому і використовується термін “статичний”. Точність, детальність статичного аналізу залежить від програмного продукту, який для цього використовується та заданих критеріїв аналізу. Типи помилок які при цьому виявляються такі, що у більшості випадків не стосуються використання патернів: неініціалізовані змінні, використання null-вказівників, переповнення буферу, незмінний вхідний параметр функції, витік пам’яті та інші [18].

На відміну від аналізаторів, що шукають потенціальні помилки в коді, тих що аналізують архітектурну складову проекту досить мало.

1.3.1 Огляд Klocwork

Klocwork – статичний аналізатор коду для мов C, C++, C#, Java, JavaScript і Python, що визначає проблеми безпеки, якості та надійності програмного забезпечення, допомагаючи забезпечити відповідність стандартам. Він масштабується для проектів будь-якого розміру, інтегрується з великими складними середовищами, широким набором інструментів розробки програмного забезпечення, а також забезпечує контроль та звітність. Це зробило Klocwork таким статичним аналізатором, який підтримує високу швидкість розробки, забезпечуючи безперервну відповідність вимогам безпеки та якості. Функція постійного аналізу забезпечує автоматичне виявлення проблем і виділення помилок під час роботи та при одразу при відкритті нового файлу. Приклад процесу аналізу коду показано на рис. 1.6 [19].

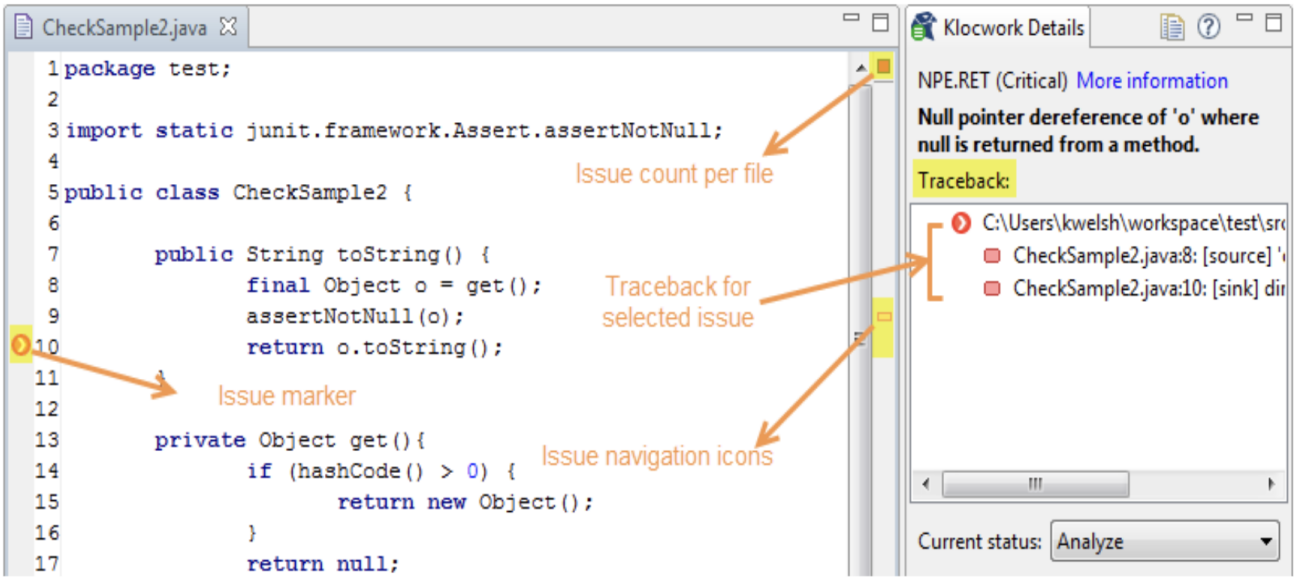


Рисунок 1.6 – Процес аналізу коду

За замовчуванням Klocwork Desktop виконує безперервний аналіз лише тих файлів, які зараз відкриті. При використанні Klocwork Desktop, папка

проекту сканується та аналізується будь-який змінений файл, незалежно від того, відкритий файл чи ні [20, 21].

Функціонал аналізу коду доповнюється створенням підсумкового звіту. Підсумковий звіт – це інформаційна панель проекту, яка надає швидку діагностичну інформацію про стан цього проекту. Підсумковий звіт містить чотири звіти за замовчуванням:

1. Посилаючись на результати аналізу, показує проблеми двох видів - в статусі “Analyze” та проблеми в статусі “in Fix”(буде виправлено).
2. Найпопулярніші проблеми, показує основні проблеми останньої збірки.
3. Тенденція складності, що показує кількість методів зі складністю понад 20.
4. Усі проблеми за станом, показуючи різницю станів для всіх проблем (наприклад: нові, відкриті, виправлені) [22].

Звіт підтримує редагування, тобто до нього можна додавати інші звіти за замовчуванням або звіти які вже були створені. У звіті з відомостями зображено: таблиці, що показують розповсюдження помилок або розповсюдження попереджень, завдяки чому можна побачити, як проблеми розподіляються за категоріями в поточному проекті; таблицю з яка називається “Top 10 Files/Classes” (зображено на рис. 1.7) із найбільшою кількістю проблем перелічених в порядку спадання для кожної категорії, яка містить виявлені проблеми. Є можливість задати кількість файлів/класів, які відображаються [22].

Незважаючи на те, що статичний аналізатор Klocwork просунутий рівень функціоналу для аналізу програмного коду, він є платним, але що є його не є його єдиним мінусом. Головним недоліком даного програмного забезпечення є те, що воно не включає в себе функціона аналізу архітектури,

тобто не надає можливості проаналізувати наявність та якість реалізації патернів проектування.

C and C++: Buffer Overflow

Issues density: 1.08

Issues Distribution

Type	#	%
ABR: Buffer Overflow - Array Index Out of Bounds	5	83.33
ABV.STACK: Buffer Overflow - Local Array Index Out of Bounds	1	16.67
Total	6	100

Top 10 Files/Classes

File/Class	Issues	Density
trees.c	5	9.78
minigzip.c	1	6.49

Рисунок 1.7 – Зміст підсумкового звіту

1.3.2 Огляд VisualCodeGrepper

VisualCodeGrepper – це автоматизований інструмент перевірки безпеки коду з відкритим вихідним кодом, який працює з C++, C#, VB, PHP, Java та PL/SQL, щоб відстежувати недоліки та різні проблеми в коді та прискорити процес перевірки майбутнього програмного продукту шляхом виявлення поганого/небезпечного коду. Цей інструмент перевіряє та детально описує виявлені ним проблеми, пропонуючи простий у використанні інтерфейс, який показано нижче на рис. 1.8 [23, 24, 25].

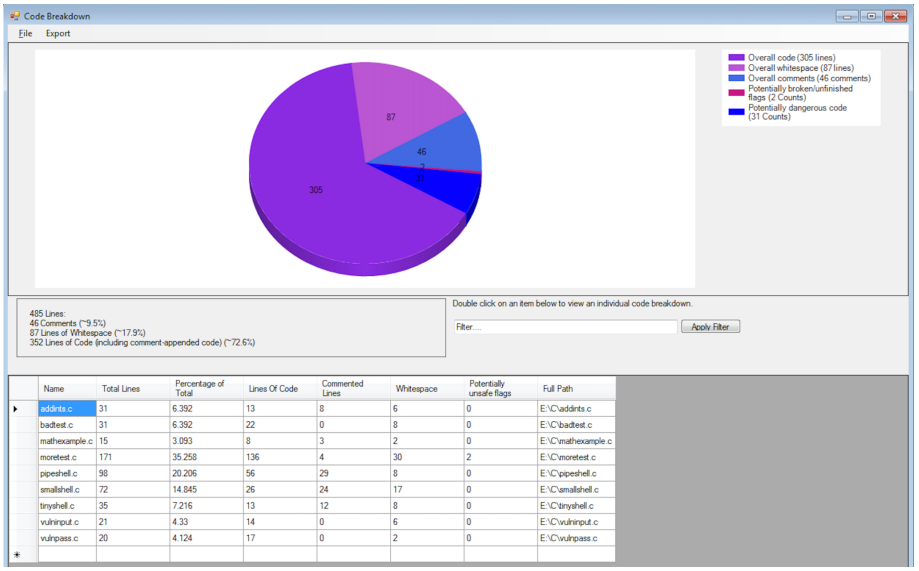


Рисунок 1.8 – інтерфейс аналізатору коду VisualCodeGrepper

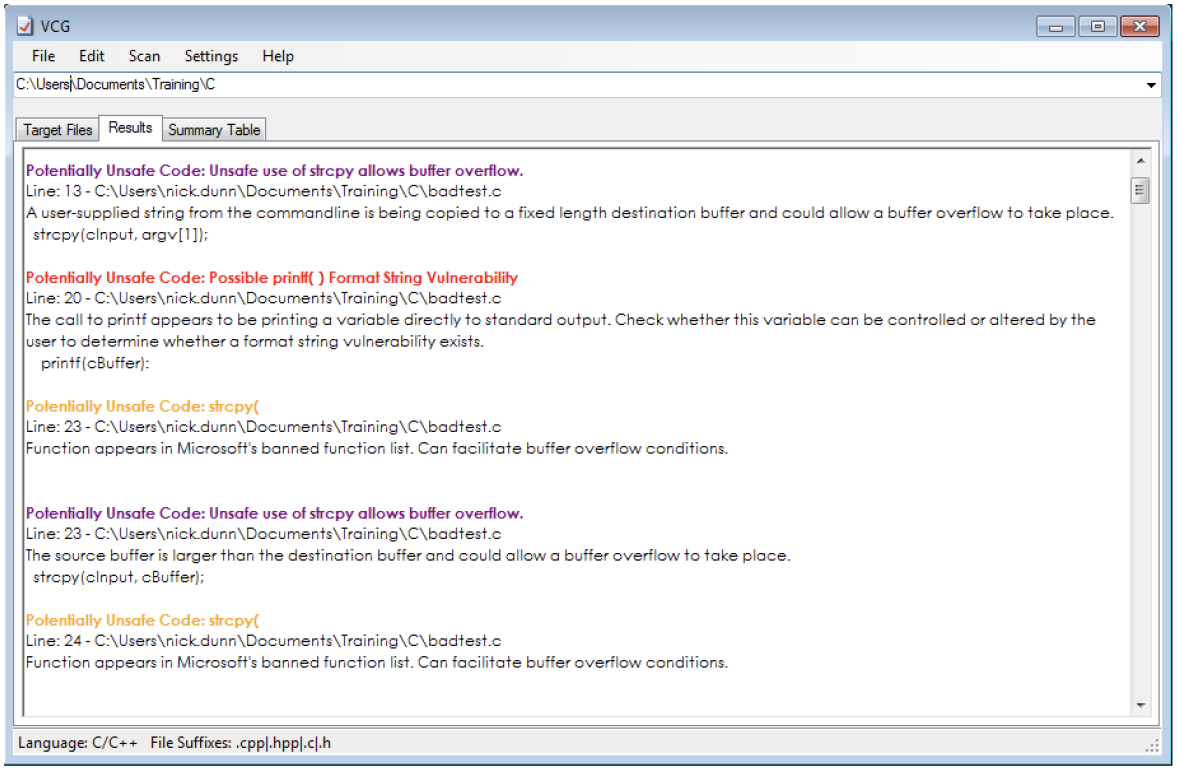


Рисунок 1.9 – Інтерфейс користувача з помилки знайденими при аналізі коду

Даний аналог надає такий функціонал:

1. На додаток до виконання деяких складних перевірок, він має файл конфігурації для кожної мови, який в основному дозволяє розробникам додавати будь-які необхідні функції (або інший текст), який необхідно перевірити в проекті.
2. Знаходження діапазону приблизно з 20 фраз у коментарях, які можуть вказувати на зламаний код (“ToDo”, “FixMe”, “Kludge” тощо).
3. Забезпечує гарну кругову діаграму для всієї кодової бази та для окремих файлів, що показує відносні пропорції коду, пробілів, коментарів, коментарі у стилі «ToDo» та поганого коду.
4. Експорт метаданих коду як XML файл. Отриманий XML містить деталі щодо кількості рядків коду, коментарів, пробілів тощо.

5. Можливість налаштування VCG так, щоб повідомляти лише про помилки, які перевищують певний рівень серйозності, що зображені на рис. 1.9. Наприклад при виборі середнього рівню, у звіті буде відображено лише середній, високий і критичний рівень серйозності помилок в коді [25].

Перевірка коду може проводитись наступними способами:

1. Лише коментарі – VCG намагається визначити будь-які коментарі, які вказують на зламаний або незавершений код на основі списку з приблизно 16 фраз, які зазвичай містяться в таких коментарях (“ToDo”, “FixMe” тощо).
2. Тільки код – VCG сканує та повідомляє про потенційні проблеми з безпекою коду та будь-які небезпечні функції, які міститься в коді використовуючи вказані в конфігураційному файлі налаштування.
3. Тільки небезпечні функції – VCG сканує та повідомляє лише про будь-які небезпечні функції в коді. Для цього також використовуються налаштування вказані в конфігураційному файлі.
4. Повна перевірка – код, небезпечні функції та коментарі – це комбіноване сканування коду та коментарів, що охоплює все вищезазначене.

Крім описаних вище налаштувань є стандартні налаштування, які показано на рис. 1.10, що коригуються безпосередньо через графічний інтерфейс користувача [25].

Конфігураційні файли існують для кожної з шести мов, з якими працює аналізатор. Вони забезпечують додатковий рівень перевірки, щоб доповнити вбудовані комплексні сканування для кожної мови [24, 25]. Крім того VisualCodeGrepper є програмним забезпеченням open-source типу, що дає змогу розробникам прийняти участь і доповнити або змінювати функціонал,

наприклад реалізувати аналіз використання патернів та загальноприйнятих підходів проектування, бо в поточній версії цей функціонал нереалізований, що є недоліком даного програмного аналогу.

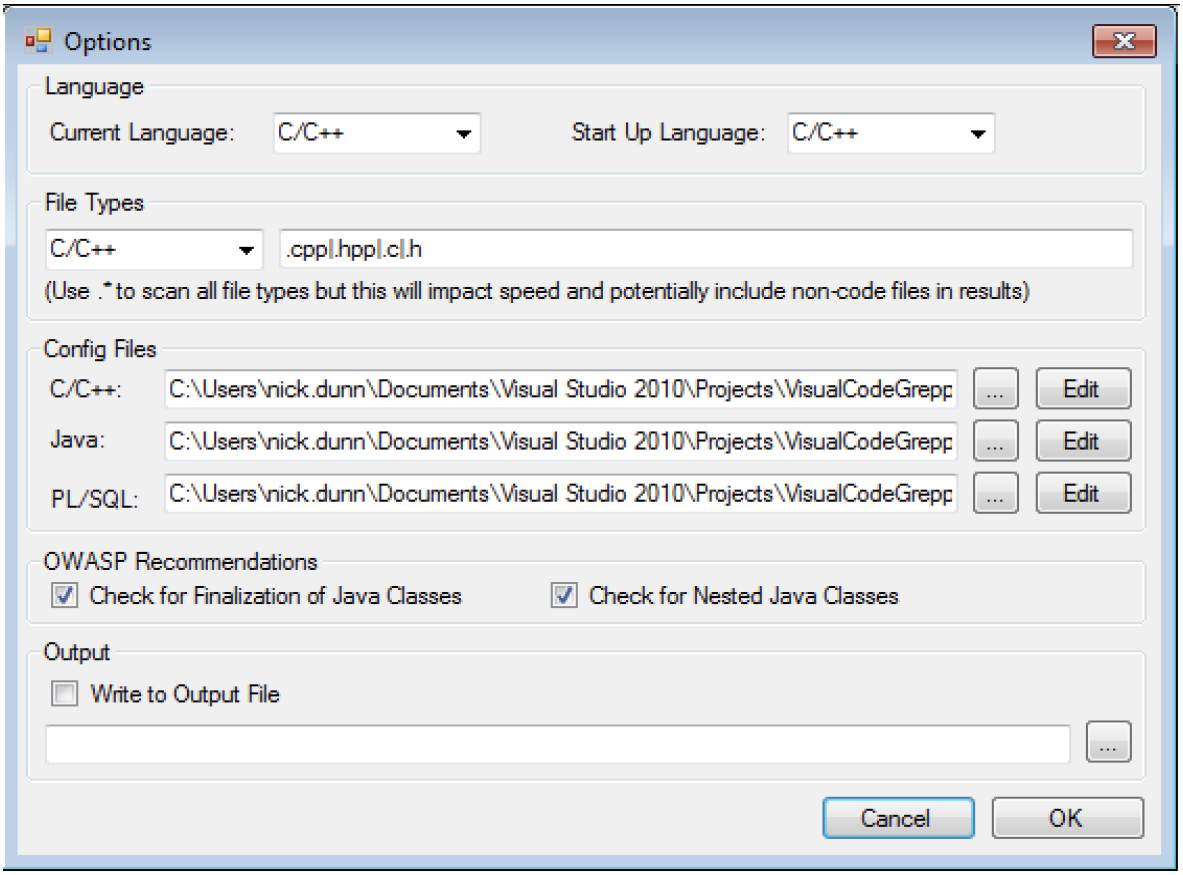


Рисунок 1.10 – Налаштування VCG

1.3.3 Огляд NDepend

NDepend – це програмний продукт для статичного аналізу коду, який орієнтований на перевірку архітектури. Програма працює зі складними метриками та може виявити недоліки виду поганої реалізації принципу інкапсуляції, проблеми пов’язані із зв’язанністю програмних модулів, тощо. Метрик нараховується приблизно 82 і їх можна використовувати для опису нових метрик користувача [26].

Крім того функціонал дозволяє побудувати графік залежностей (англ. dependency graph) між програмними модулями, що аналізуються. Завдяки цьому можна візуально зрозуміти які модулі в програмі реалізовані некоректно. Приклад такого графіку зображено на рис. 1.11 [26, 27].

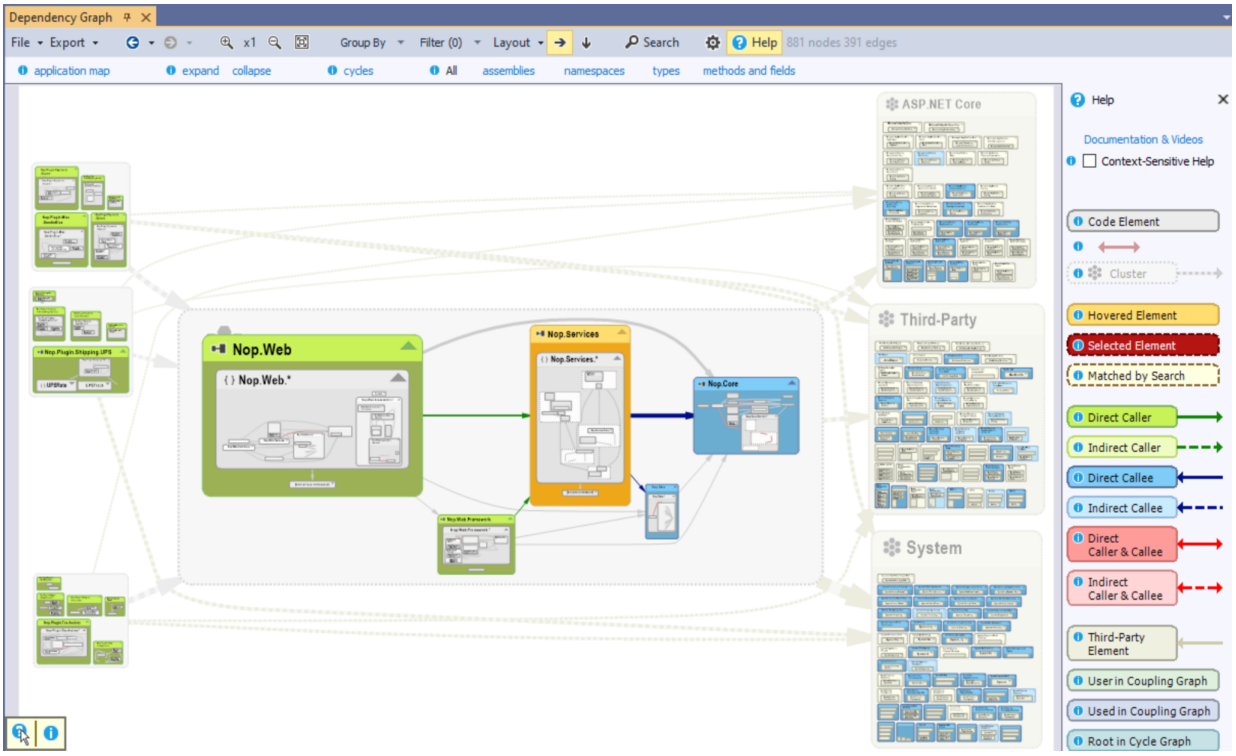


Рисунок 1.11 – Графік залежностей

Графік залежностей синхронізується з матрицею залежностей (англ. *dependency matrix*), яка демонструє загальну кількість залежностей між модулями. Матриця залежностей зображена на рис. 1.12 [26, 27].

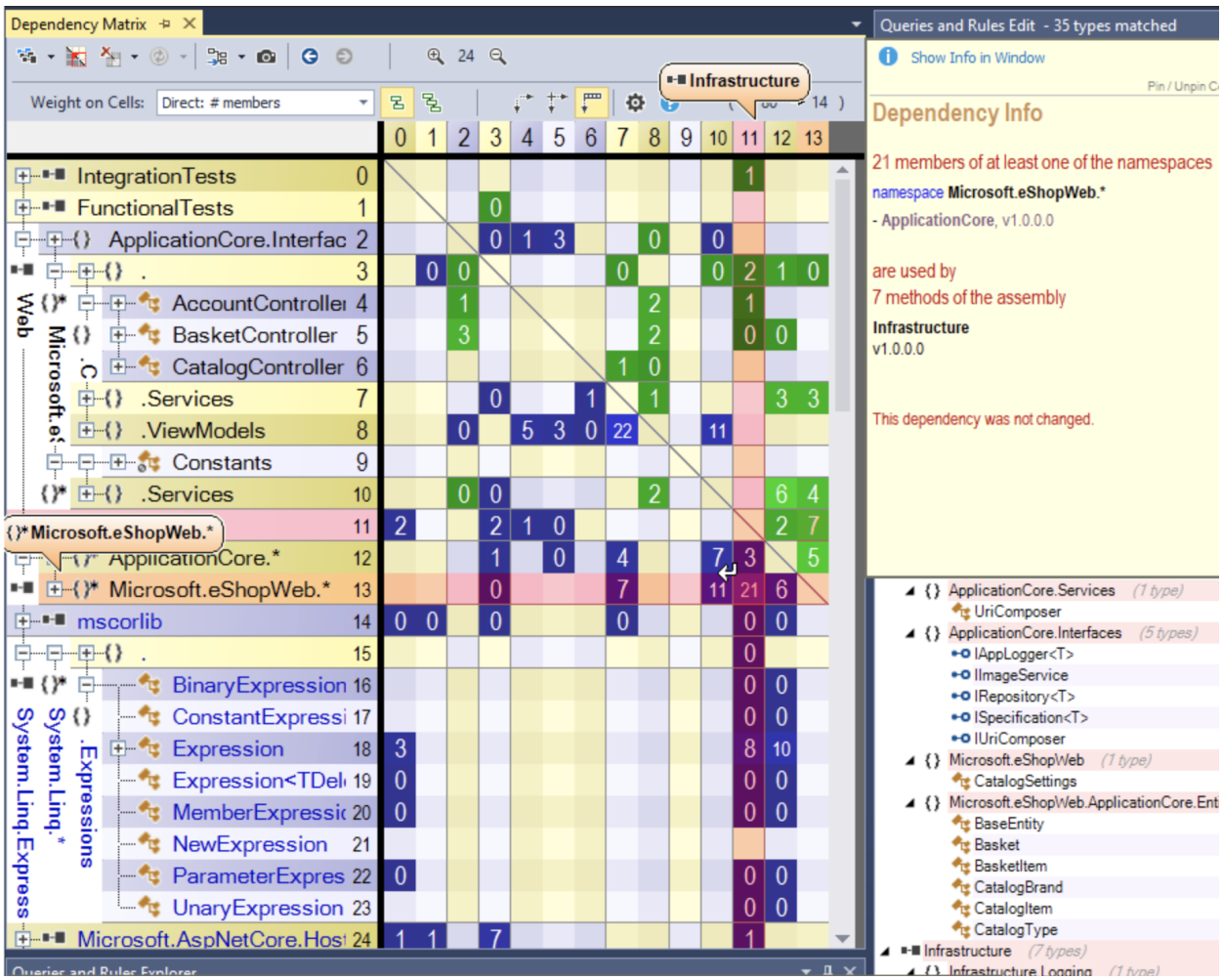


Рисунок 1.12 – Матриця залежностей

Основою NDepend є мова подібна SQL запитам до бази даних - Code Query Language або CQL. Цей аналізатор можна інтегрувати в процес білду (англ. build process), наприклад на боці CI (Continuous Integration). Також є можливість інтегрувати в якості плагіна в Visual Studio, що показано на рис. 1.13 [26, 27].

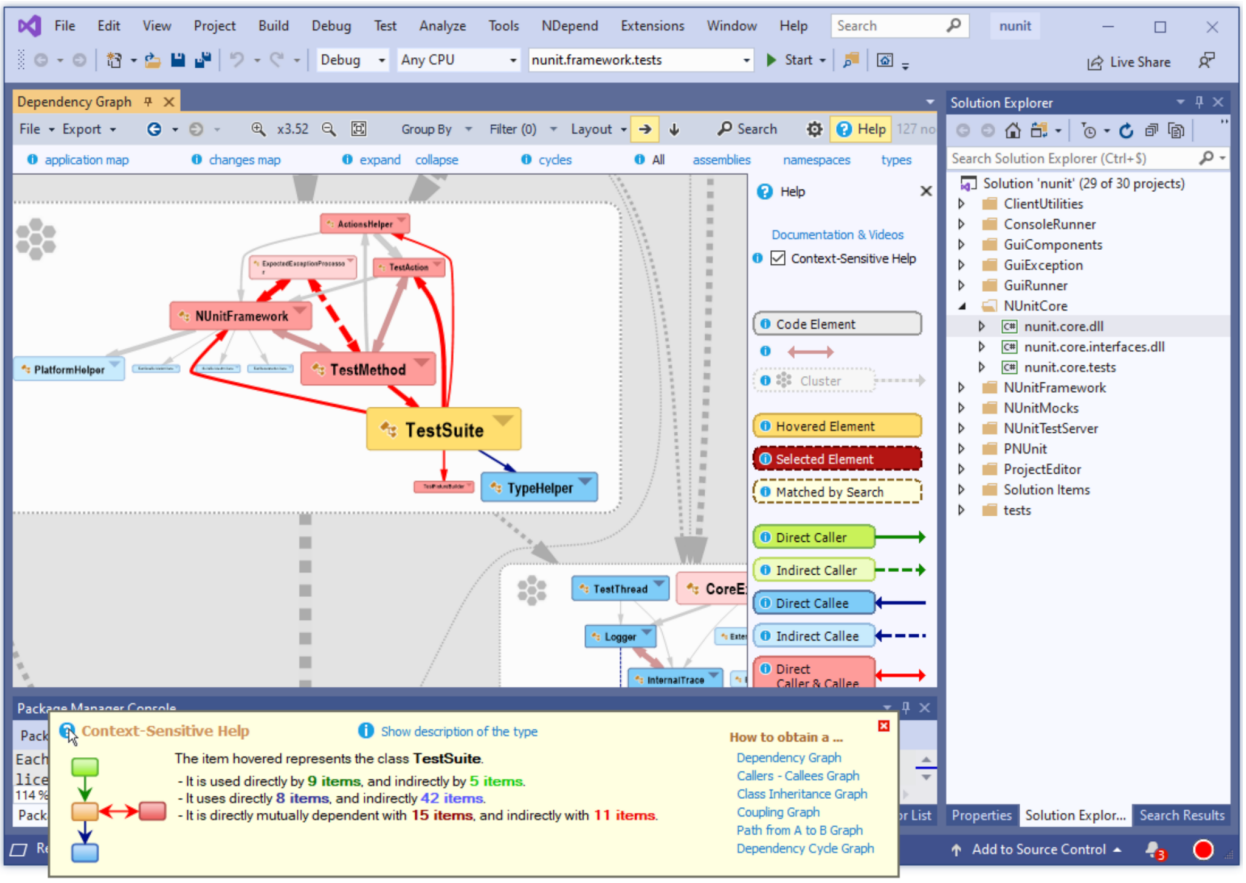


Рисунок 1.13 – Інтеграція NDepend з Visual Studio

Недоліками можна вважати, що даний програмний продукт вимагає від користувача хороших знань стосовно побудови архітектури, тому розробники - початківці навряд чи мають набратися досвіду цьому питанні. Окрім цього NDepend потребує гарного розуміння своєї термінології, а також є платним.

1.3.4 Огляд Sonargraph

Sonargraph – це статичний аналізатор коду, який використовується для моніторингу якості програмної системи та архітектури програмного забезпечення. Цей аналізатор підтримує такі мови як C, C++, Java, Kotlin, C#, Python та включає в себе механізм написання сценаріїв на основі DSL (domain specific language) для опису архітектури програмних продуктів. Крім цього Sonargraph складається з декількох модулів одними з яких є

Sonargraph-Explorer, який є повністю безкоштовним та Sonargraph-Architect, що поставляється в комерційному варіанті. Обидва є аналізаторами [28].

Sonargraph-Architect надає можливість автоматизованої перевірки з використанням DSL, механізму сценаріїв на основі Groovy, віртуального рефакторингу, засобу перевірки повторюваного коду, а також включає в себе всі функції Sonargraph-Explorer [29].

Sonargraph-Explorer – це інструмент статичного аналізу з акцентом на метриках та візуалізацію залежностей. Граф залежностей, приклад якого наведено на рис. 1.14, показує залежності між елементами. Усі вузли, що входять до циклічної групи об’єднуються в один вузол і його можна додатково вивчати за допомогою засобу перегляду циклів [30].

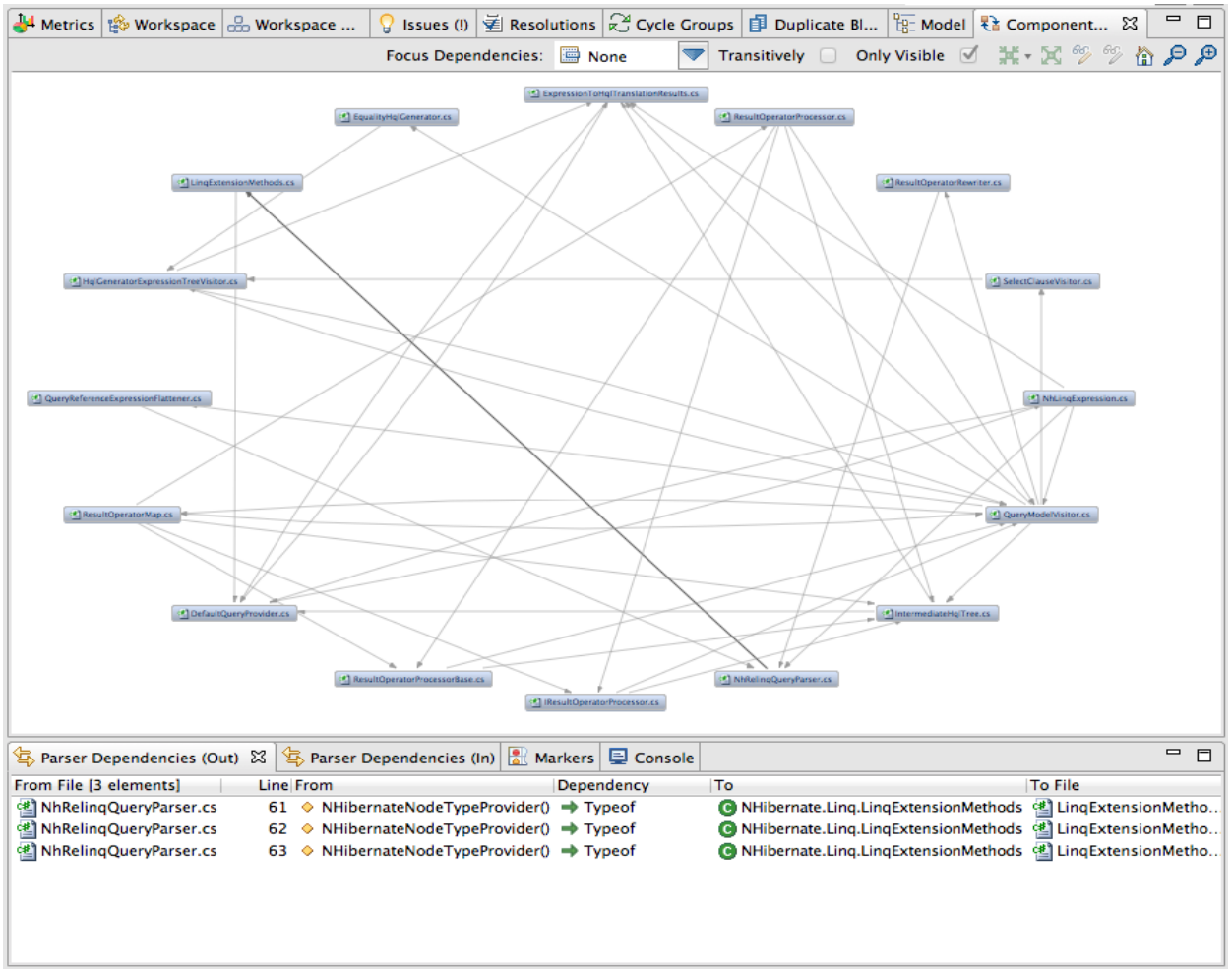
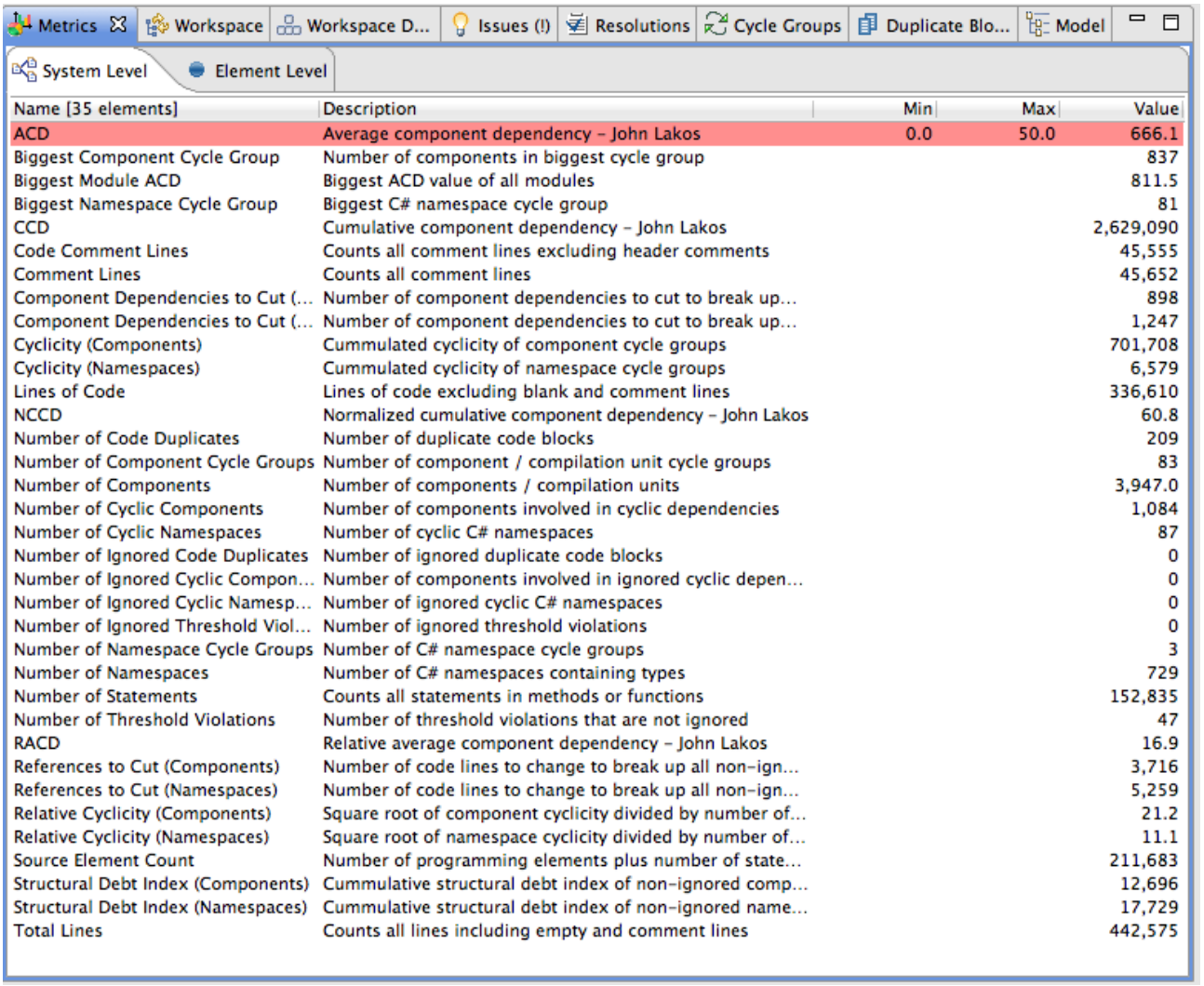


Рисунок 1.14 – Граф залежностей

Sonargraph-Explorer обчислює сотні метрик якості, завдяки яким можна оцінити якість будь-якої програмної системи, а завдяки візуалізації залежностей можна зрозуміти програмну систему навіть якщо вона не має документації. Інтерфейс для перегляду метрик зображено на рис. 1.15 [30].



Name [35 elements]	Description	Min	Max	Value
ACD	Average component dependency – John Lakos	0.0	50.0	666.1
Biggest Component Cycle Group	Number of components in biggest cycle group			837
Biggest Module ACD	Biggest ACD value of all modules			811.5
Biggest Namespace Cycle Group	Biggest C# namespace cycle group			81
CCD	Cumulative component dependency – John Lakos			2,629,090
Code Comment Lines	Counts all comment lines excluding header comments			45,555
Comment Lines	Counts all comment lines			45,652
Component Dependencies to Cut (...	Number of component dependencies to cut to break up...			898
Component Dependencies to Cut (...	Number of component dependencies to cut to break up...			1,247
Cyclicity (Components)	Cummulated cyclicity of component cycle groups			701,708
Cyclicity (Namespaces)	Cummulated cyclicity of namespace cycle groups			6,579
Lines of Code	Lines of code excluding blank and comment lines			336,610
NCCD	Normalized cumulative component dependency – John Lakos			60.8
Number of Code Duplicates	Number of duplicate code blocks			209
Number of Component Cycle Groups	Number of component / compilation unit cycle groups			83
Number of Components	Number of components / compilation units			3,947.0
Number of Cyclic Components	Number of components involved in cyclic dependencies			1,084
Number of Cyclic Namespaces	Number of cyclic C# namespaces			87
Number of Ignored Code Duplicates	Number of ignored duplicate code blocks			0
Number of Ignored Cyclic Compon...	Number of components involved in ignored cyclic depen...			0
Number of Ignored Cyclic Namesp...	Number of ignored cyclic C# namespaces			0
Number of Ignored Threshold Viol...	Number of ignored threshold violations			0
Number of Namespace Cycle Groups	Number of C# namespace cycle groups			3
Number of Namespaces	Number of C# namespaces containing types			729
Number of Statements	Counts all statements in methods or functions			152,835
Number of Threshold Violations	Number of threshold violations that are not ignored			47
RACD	Relative average component dependency – John Lakos			16.9
References to Cut (Components)	Number of code lines to change to break up all non-ign...			3,716
References to Cut (Namespaces)	Number of code lines to change to break up all non-ign...			5,259
Relative Cyclicity (Components)	Square root of component cyclicity divided by number of...			21.2
Relative Cyclicity (Namespaces)	Square root of namespace cyclicity divided by number of...			11.1
Source Element Count	Number of programming elements plus number of state...			211,683
Structural Debt Index (Components)	Cummulative structural debt index of non-ignored comp...			12,696
Structural Debt Index (Namespaces)	Cummulative structural debt index of non-ignored name...			17,729
Total Lines	Counts all lines including empty and comment lines			442,575

Рисунок 1.15 – Інтерфейс перегляду метрик

Незважаючи на те, що інструментарій аналізатору архітектури пропонує різний функціонал, на жаль він не може бути використаний окремо від статичного аналізатору і безкоштовним є лише одна з його складових -

Sonargraph-Explorer, а повноцінний функціонал є платним. Ці пункти можна віднести до недоліків даного програмного забезпечення.

Висновки до розділу 1

Ознайомлюючись з інформацією, яка пропонується, можна дізнатися дуже багато про патерни та загальні підходи побудови крос-платформних мобільних додатків, побачити різні приклади їх реалізації, тощо. Майже вся інформація - це переказ якоїсь частини книги, з накладанням певного досвіду. Всі літературні ресурси поєднує те, що більшість інформації припадає на пояснення шаблону, його реалізації та прикладам самої реалізації і досить мала доля припадає на пояснення ситуацій коли , в якій мірі необхідно використовувати і чи потрібно взагалі використовувати той чи інший патерн/підхід, на проекті якого розміру чи типу, з яким функціоналом, бюджетом, замовником.

Проаналізувавши програмні продукти орієнтовані на аналіз архітектури, наявності реалізації патернів, якості цієї реалізації та використання загальноприйнятих підходів в певному проекті стає зрозуміло, що більшість програмних аналогів займаються пошуком програмних помилок і лише декілька надають функціонал для перевірки архітектурної складової та є платними. Досить часто аналіз архітектури неавтоматизовано.

Досить важко із впевненістю сказати, що аналізатор будуть використовувати в невеликих компаніях і на невеликих проектах, бо з фінансової точки зору це дорогий процес, а безкоштовні версії являють собою недостатній набір функцій та обмежені в часі використання. Крім того всі виявлені проблеми є можливими, а не точними. Це говорить про те, що даний інструмент не гарантує точно, чи є ця помилка в коді насправді проблемою, тому часто результат аналізу коду є помилковим.

Перевагами статичного аналізу коду є відсутність необхідності в астрономічних прогонах програм за різних умов функціонування та можливість домогтися більшої міри автоматизації перевірок на наявність дефектів програм виходячи з їх конструктивних ознак [31].

Підводячи підсумок можна сказати, що існуючі програмні рішення та літературні джерела не відповідають повною мірою поставленим вимогам. Отже, дослідження є досить актуальним.

2. ОБҐРУНТУВАННЯ ЕКСПЕРИМЕНТАЛЬНОГО МЕТОДУ ДОСЛІДЖЕННЯ НАСЛІДКІВ ВИКОРИСТАННЯ ПАТЕРНІВ І ЗАГАЛЬНОПРИЙНЯТИХ ПІДХОДІВ В ПОБУДОВІ АРХІТЕКТУРИ КРОС-ПЛАТФОРМНИХ ДОДАТКІВ ПІД ANDROID І IOS.

Задачею експериментального методу дослідження є доведення або спростування гіпотези про те, що архітектура крос-платформного додатку яка розроблена із надмірним застосуванням патернів та загальноприйнятих підходів проектування несе не менш негативні наслідки на підтримку, розширення, повторне використання окремих модулів, вартість мобільного додатку, ніж архітектура яка розроблялася з нехтуванням необхідних патернів та принципів проектування.

При розробці алгоритму використовуються тенденції, методики, методи та підходи, що можуть використовуватись в сукупності. Виходячи з того, що для доведення чи спростування гіпотези необхідно розробити алгоритм для аналізу архітектури програмного забезпечення, то слід приділити увагу вже існуючим методам та підходам.

Аналіз архітектури додатків – критично важлива задача, оскільки дозволяє скоротити витрати на виправлення помилок, як можна раніше виявити та виправити можливі проблеми. Аналіз архітектури виконується часто по завершенню основних етапів проекту та у відповідь на істотні зміни в архітектурі.

Хоча архітектура програмного забезпечення стає все більш важливою темою дослідження в останні роки, недостатньо уваги приділяється методам оцінки архітектури. Оцінити архітектуру складно з двох основних причин. По-перше, не існує спільної мови для опису різних архітектур. По-друге, немає чіткого способу розуміння архітектури стосовно проблем якості прог -

рамного забезпечення, таких як ремонтопридатність, переносимість, модульність, можливість повторного використання тощо.

Основна задача аналізу архітектури – підтвердження відповідності базової архітектури та її можливих варіантів, а також перевірка відповідності запропонованих технічних рішень функціональними вимогами та параметрами якості. Крім того, аналіз допомагає виявити проблеми та області, потребуючі виправлень або доопрацювань [32].

Оцінка на основі сценаріїв – це потужний метод аналізу дизайну архітектури. При такому оцінювані основну увагу спрямовано на сценарії найважливіші з точки зору бізнесу, а також на ті, що мають великий вплив на архітектуру. Розглянемо наступні типові методики [32].

2.1 Метод аналізу архітектури програмного забезпечення (SAAM)

Метод аналізу архітектури програмного забезпечення SAAM (англ. *Software Architecture Analysis Method, SAAM*) – це методологія, яка використовується для визначення того, як були досягнуті конкретні атрибути якості програми та як можливі зміни в майбутньому вплинуть на атрибути якості на основі гіпотетичних ситуацій. Загальні атрибути якості, які можна використовувати за допомогою цієї методології, включають можливість модифікації, надійність, переносимість і розширюваність [32, 33].

Метод аналізу архітектури програмного забезпечення SAAM можна застосувати для двох різних аналізів: для порівняння двох або більше варіантів дизайну, щоб зрозуміти який із них відповідає вимогам до якості краще; для оцінки дизайну, щоб виявити місця, де архітектура не відповідає вимогам до якості [34].

2.1.1 Атрибут якості “Можливість модифікації програми”

Атрибут “Можливість модифікації програми” (англ. *Application Modifiability*) вказує на те, наскільки легко буде змінити систему в майбутньому. Для того, щоб SAAM можна було належним чином застосувати для перевірки цього атрибута, необхідно створити конкретні гіпотетичні приклади. Наприклад заміна модулю для оплати продуктів або додавання додаткового виду платежу в мобільному додатку, який надає змогу проводити покупки онлайн. Суть цього атрибуту якості має спільні риси з принципами SOLID [33].

2.1.2 Атрибут якості “Надійність застосування”

Атрибут якості “Надійність застосування” (англ. *Application Robustness*) відноситься до того, як програма справляється з несподіванками. Несподіванкою можна назвати те, що не передбачено в початковому проекті системи. Як поведе себе програмний продукт в такій ситуації. Наприклад: обробка некоректних даних, відсутність підключення до мережі інтернет або будь-які неочікувані випадки в програмі [33].

2.1.3 Атрибут якості “Переносимість програми”

Атрибут якості “Переносимість програми” (англ. *Application Portability*) відноситься до простоти перенесення програми для запуску в новій операційній системі або на новому пристрої. Наприклад, набагато простіше змінити веб-сайт ASP.net, щоб він був доступним для Windows, Mac, IPhone, Android в порівнянні з настільною програмою, написаною на VB.net, тому що було б необхідно провести набагато більше роботи, щоб довести програму до стану коли вона зможе використовуватися на різних платформах.

Цей атрибут якості повинен бути оцінений на основі кожного нового середовища, для якого проведено гіпотетичне дослідження. Особливої уваги потребують:

- питання портативності;
- апаратні залежності;
- залежності операційної системи;
- залежність від джерела даних;
- залежність від доступу до мережі [33].

2.1.4 Атрибут якості “Розширюваність програми”

Атрибут якості “Розширюваність програми” (англ. *Application Extensibility*) відноситься до простоти додавання нових функцій до існуючої програми без впливу на наявну функціональність. Аспекти, що впливають на цей атрибут якості це: наявність жорстко закодованих змінних, наявність документації програмного продукту, використання принципів SOLID [33].

2.2 Метод аналізу архітектурних компромісів ATAM

Метод аналізу архітектурних компромісів (англ. *Architecture Tradeoff Analysis, ATAM*) - це доопрацьована та покращена версія SAAM, яка дозволяє переглядати архітектурні рішення щодо вимог параметрів якості та того, наскільки добре ці рішення відповідають конкретним цільовим показникам якості. ATAM є найбільш корисним, коли використовується на ранніх етапах життєвого циклу програмного забезпечення, коли вартість змін в архітектурі є мінімальною. Проводити аналіз архітектурних компромісів ATAM треба: початку життєвого циклу; при побудові системи; при придбанні системи. Схематичне зображення цього методу показано на рис. 2.1 [32, 35, 36, 37].

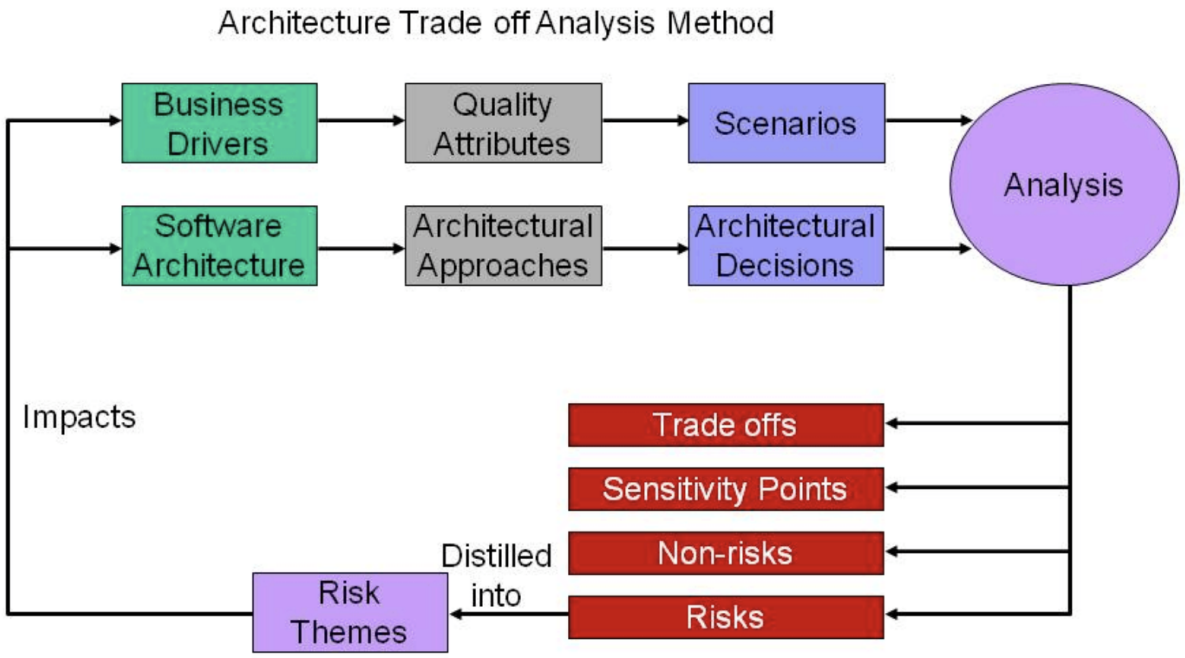


Рисунок 2.1 - Схема методу аналізу архітектурних компромісів

Плюси використання методу аналізу архітектурних компромісів:

- виявлення ризиків на початку життєвого циклу;
- наявність конкретних вимог до атрибутів якості;
- покращена архітектурна документація;
- документована основа архітектурних рішень.

Процес ATAM складається зі збору зацікавлених сторін для аналізу бізнес-драйверів (функціональність системи, цілей, обмежень, бажаних нефункціональних властивостей) і вилучення з цих драйверів атрибутів якості, які використовуються для створення сценаріїв. Ці сценарії потім використовуються в поєднанні з архітектурними підходами та архітектурними рішеннями для створення аналізу компромісів, точок чутливості та ризиків (або не ризиків). Цей аналіз можна перетворити на теми ризику та їх вплив, після чого процес можна повторити. З кожним циклом аналізу процес аналізу переходить від більш загального до більш конкретного, вивчаючи питання, які були виявлені в попередньому циклі, до

тих пір, поки архітектура не буде точно налаштована і не будуть розглянуті теми ризику [35].

2.3 Метод аналізу рентабельності CBAM

Метод аналізу рентабельності (англ. *Cost Benefit Analysis Method, CBAM*). Метод CBAM основну увагу приділяє аналізу витрат, вигод та планування наслідків архітектурних рішень. Складові методу відображені на рис. 2.2 [32, 38].

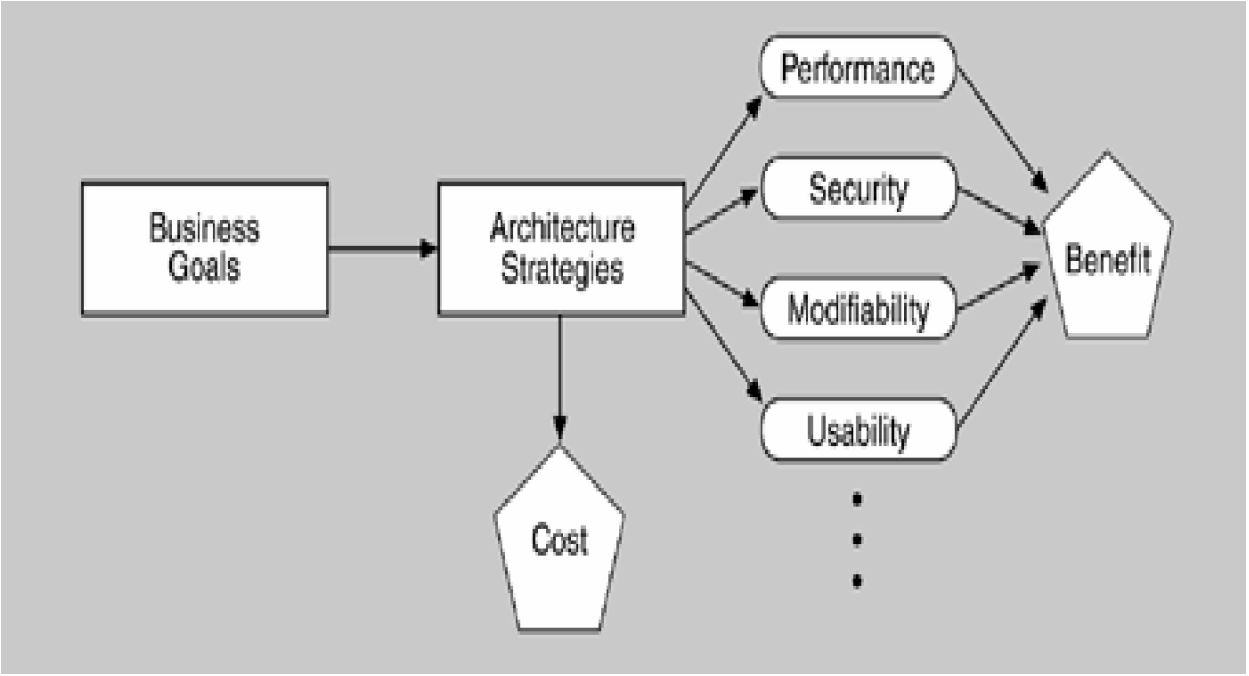


Рисунок 2.2 - Складові методу аналізу рентабельності

2.4 Метод оцінки сімейства архітектур FAAM

Метод оцінки сімейства архітектур (англ. *Family Architecture Assessment Method, FAAM*) оцінює архітектури сімейства інформаційних систем з погляду можливості взаємодії та розширюваності [32].

2.5 Аналіз модифікованості лише на рівні архітектури ALMA

Аналіз модифікованості лише на рівні архітектури (англ. *Architecture Level*

Modifiability Analysis, ALMA) оцінює модифікованість архітектури для систем бізнес-аналітики (англ. *business information systems, BIS*) [32].

Висновки до розділу 2

У розділі розглянуто та проаналізовано основні методики аналізу архітектури програмного забезпечення. Виділено завдання які вони вирішують, що пояснює необхідність їх використання при розробці алгоритму системи аналізу реалізації патернів та загально-прийнятих підходів в побудові архітектури крос-платформних додатків.

3. ПРОЕКТУВАННЯ І РОЗРОБКА ІНСТРУМЕНТАЛЬНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ НАСЛІДКІВ ВИКОРИСТАННЯ ПАТЕРНІВ І ЗАГАЛЬНОПРИЙНЯТИХ ПІДХОДІВ В ПОБУДОВІ АРХІТЕКТУРИ КРОС-ПЛАТФОРМНИХ ДОДАТКІВ ПІД ANDROID І IOS

3.1 Формалізація задачі

Для формалізації задачі була розроблена діаграма варіантів використання, на якій відображено можливі варіанти використання системи актором, що представляє користувача.

Варіанти використання системи показані на рис. 3.1.

Рисунок 3.1 – Схема варіантів використання

3.2 Опис базової архітектури системи

При проектуванні інструментального забезпечення було вирішено використовувати популярний шаблон MVVM - “Модель-вигляд-модель вигляду” (англ. *Model-View-ViewModel)* , що використовується для розподілу архітектури програмного забезпечення “Static architecture analyzer” на три рівні. Схему розподілу зображено на рис. 3.2.

Розподіл архітектури на три рівні дозволяє розподілити обов’язки кожного модулю, надає можливість паралельної розробки модулів незалежно один від одного, а також полегшує процес тестування, бо кожен з модулів може бути протестовано окремо один від одного. В свою чергу модулі патерну MVVM несуть за собою наступну функціональність:

Модель (англ. *Model*) являє собою бізнес логіку проекту, що є основною його функціональністю, а також моделі даних, які використовуються бізнес логікою проекту. Ці моделі даних можуть бути збережені в базі даних, використовуватися для обміну даними між сервісами або можуть бути надані рівню ViewModel в якості результату якоїсь операції. Бізнес логіка на рівні Модель ділиться на сервіси (англ. *Services*), які є підмодулями з реалізацією певного функціоналу. Прикладом може бути сервіс по реалізації CRUD (Create-Read-Update-Delete) операцій над замітками користувача і результатом виконання якого буде модель “замітка” [5].

У шаблоні MVVM залежність рівнів між собою однонаправлена, тобто влаштована так, що Модель Вигляду має залежність від моделі, а Модель в свою чергу немає залежності від жодного з рівнів. На рівні Модель не допустимо описувати логіку інтерфейсу користувача або логіку взаємодії з ним [5].

Вигляд (англ. *View*) описує реалізацію інтерфейсу користувача (англ. *User Interface*). Це може бути сторінка мобільного, десктопного або веб додатку. Рівень інтерфейсу користувача часто поділяється на підмодулі, залежить від рівню Модель Вигляду, але не повинен реалізовувати бізнес логіку чи логіку по роботі з даними [5].

Модель Вигляд (англ. *ViewModel*) займається обробкою команд від інтерфейсу користувача та запитами до рівню Модель аби потім отриманні дані були відображені на рівні Вигляд. Цей компонент є мостом між рівнем представлення даних та роботою з ними. Передача інформації до інтерфейсу користувача реалізується завдяки механізму зв'язування (англ. *Binding*). На даному рівні не повинно міститись реалізації роботи з даними [5].

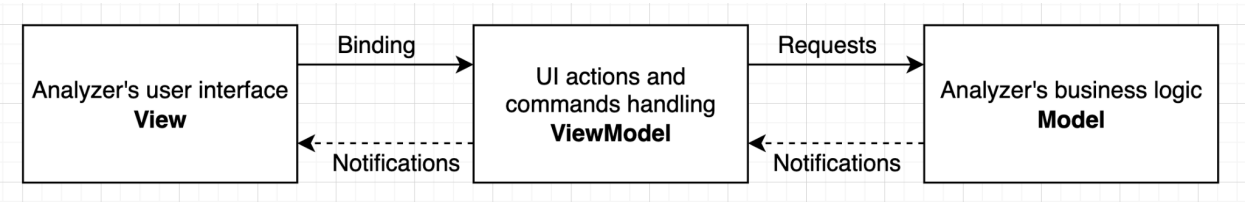


Рисунок 3.2 – Схема архітектури "Static architecture analyzer" за паттерном MVVM

Парадигмою програмування була обрана об'єктно-орієнтована.

3.3 Внутрішнє проектування

3.3.1 Вибір мови програмування

При розробці статичного аналізатору було застосовано об'єктно-орієнтовану мову програмування C# та мову розмітки для додатків XAML.

Розширювана мова розмітки додатків (XAML) — це мова на основі XML, створена компанією Microsoft як альтернатива програмному коду для створення екземплярів та ініціалізації об'єктів, а також організації цих об'єктів в ієрархії "parent-child". XAML був адаптований до кількох технологій у рамках .NET, але він представляє свою найбільшу користь у створення розмітки інтерфейсів користувача у Windows Presentation Foundation (WPF), Silverlight, Windows Runtime та Universal Windows Platform (UWP) [39].

XAML дозволяє розробникам визначати інтерфейси користувача в програмах Xamarin.Forms, використовуючи розмітку, а не код. Xamarin.Forms додатки можуть бути розроблені і без застосування XAML, але він часто є більш стислим і візуально узгодженим, ніж еквівалентний код. Ця мова добре підходить для використання з популярною архітектурним патерном MVVM (Model-View-ViewModel): за допомогою XAML реалізується View рівень, який пов'язаний з кодом рівня ViewModel через прив'язки даних [39].

У файлі розмітки розробник може описувати інтерфейс користувача, використовуючи всі представлення, макети та сторінки Xamarin.Forms, а також власноруч створені UI компоненти. Файл з розміткою може бути скомпільований або вбудований у виконуваний файл. Інтрфейс розроблений за допомогою XAML має кілька переваг перед еквівалентним інтерфейсом описаним за допомогою коду: XAML часто є більш стислим і читабельним, ніж еквівалентний код; ієрархія “parent-child”, властива XML, дозволяє з більшою візуальною чіткістю імітувати ієрархію об’єктів інтерфейсу користувача; розмітка може бути легко написана від руки програмістами, але також може бути сгенерована за допомогою конструкторів для візуального проектування користувача інтерфейсу [39].

Мова програмування C# - це об'єктно-орієнтована мова програмування. Він був створений у період з 1998 по 2002 рік командою інженерів Microsoft під керівництвом Андерса Хейлсберга та Скотта Вільтаумота. Мова входить у сім'ю C-подібних мов. Синтаксис наближений до Java та C++. Ця мова має наступні особливості: статистична типізація, підтримується поліморфізм; підтримується перевантаження операторів; доступна делегація; атрибути, події, узагальнені типи та анонімні функції. Розробка Microsoft багато особливостей успадкувала у Delphi, Smalltalk та Java [40].

C# популярний за рахунок своєї «простоти». Простоти для сучасних програмістів і великих команд розробників, щоб ті могли в стислий термін створювати функціональні та продуктивні програми. Цьому сприяють нетипові конструкції мови та специфічний синтаксис, що допомагає максимально органічно реалізувати намічені функції. Популярність мови – ще одна важлива перевага, бо цу сприятливо впливає на зростання кількості вакансій, пов'язаних із розробкою на мові компанії Microsoft. Програмісти, добре знайомі з C#, затребувані в індустрії, а попит на таких розробників продовжує зростати. Також його популярності сприяє те, що даний інструмент дозволяє створювати різні типи програмних продуктів [40].

Мова використовується для розробки ігор під Windows, MacOS, Android та iOS. Вся справа в Unity – платформі для роботи з 3D-графікою, а C# краще інших мов адаптований під роботу з цим двигуном. Тому програмісти зазвичай не вибирають, а одразу використовують зв'язку Unity та C# [40].

Безпека програм та операційних систем забезпечується завдяки потужним утилітам на базі C#. Колосальна кількість вірусів, що на щоденній основі атакують комп'ютери користувачів, блокується інструментами, створеними за допомогою мови Microsoft. Аналогічна ситуація спостерігається у великому бізнесі – світові корпорації захищаються від атак хакерів за допомогою ПЗ, розробленого на C# [40].

Практично вся операційна система Microsoft існує завдяки C#. Звичні утиліти та програми створені з використанням цієї мови та фреймворків, розроблених для неї. До цієї категорії потрапляє месенджер Skype, браузер Internet Explorer, середовище для розробки Visual Studio , Microsoft Office [40].

Розробляти кросплатформні додатки в тому числі і для операційних систем Android та iOS стало можливо завдяки C# та Xamarin.Forms [40].

3.3.2 Технологічна платформа

Xamarin.Forms — це UI-фреймворк з відкритим вихідним кодом. Фреймворк дозволяє розробникам створювати програми Xamarin.Android, Xamarin.iOS, Windows та macOS з єдиної спільної кодової бази та надає змогу створення інтерфейсу користувача за допомогою XAML та коду на C#. UI-елементи відображаються як нативні (специфічні для певної платформи) елементи інтерфейсу користувача на кожній платформі. Xamarin.Forms надає API для створення елементів інтерфейсу користувача на різних платформах. Цей API може бути реалізований в XAML або C# і підтримує прив’язування даних для таких шаблонів, як Model-View-ViewModel (MVVM) [41].

Під час виконання Xamarin.Forms використовує засоби візуалізації платформи для перетворення міжплатформних елементів інтерфейсу користувача у нативні елементи керування на Xamarin.Android, Xamarin.iOS, UWP та macOS. Програми створені на цьому з використанням цього фреймворку зазвичай складаються із спільної бібліотеки .NET Standard та окремих проектів платформи. Спільна бібліотека містить представлення XAML або C# і будь-яку бізнес-логіку, моделі чи інший код. Проекти платформи містять будь-яку специфічну для платформи логіку або пакети, які потрібні додатку [41].

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від компанії Microsoft. Воно використовується для розробки комп’ютерних програм, а також веб-сайтів, веб-додатків, веб-сервісів та мобільних додатків [42].

Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Він може створювати як рідний код, так і керований код. Середовище розробки включає редактор коду, який підтримує IntelliSense (компонент завершення коду), функціонал для рефакторингу коду,

вбудовані інструменти включають профайлер коду, конструктор для створення додатків з графічним інтерфейсом, веб-дизайнер, конструктор класів і конструктор схем баз даних. Visual Studio підтримує 36 різних мов програмування. Вбудовані мови включають C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML і CSS. Підтримка інших мов, таких як Python, Ruby, Node.js і M, серед інших, доступна через плагіни [42].

3.3.4 Ієрархія та взаємодія класів системи

Проектування системи відбувалось за допомогою створення діаграми класів. Взаємодія різних рівнів показана на рис. 3.3 – 3.5.

3.3.5 Застосовані підходи проектування

Під час проектування програмного забезпечення застосовано наступні підходи.

Інверсія залежностей (англ. *Dependency inversion*) – залежності між модулями реалізовані за допомогою абстракцій сутностей, що в той же час гарантує дотримання достатнього рівню абстракції та інкапсуляції. Цей принцип є одним із принципів SOLID [11].

Розділення інтерфейсів (англ. *Interface segregation*) – модулі розподілені на підмодулі, що гарантує декілька вузько-направлених замість універсальних інтерфейсів. Даний підхід ще говорить про те, що дотримується такий принцип як розподіл відповідальностей (англ. *Separation of concerns*) [11].

Принцип єдиного обов’язку (англ. *Single responsibility*) – гарантує, що кожен модуль в системі виконує лише одну вузько-направлену функцію. Так само як і попередній принцип він також є схожим з принципом розподілу відповідальностей [11].

Рисунок 3.3 – Взаємодія сервісів та моделей на рівні Model

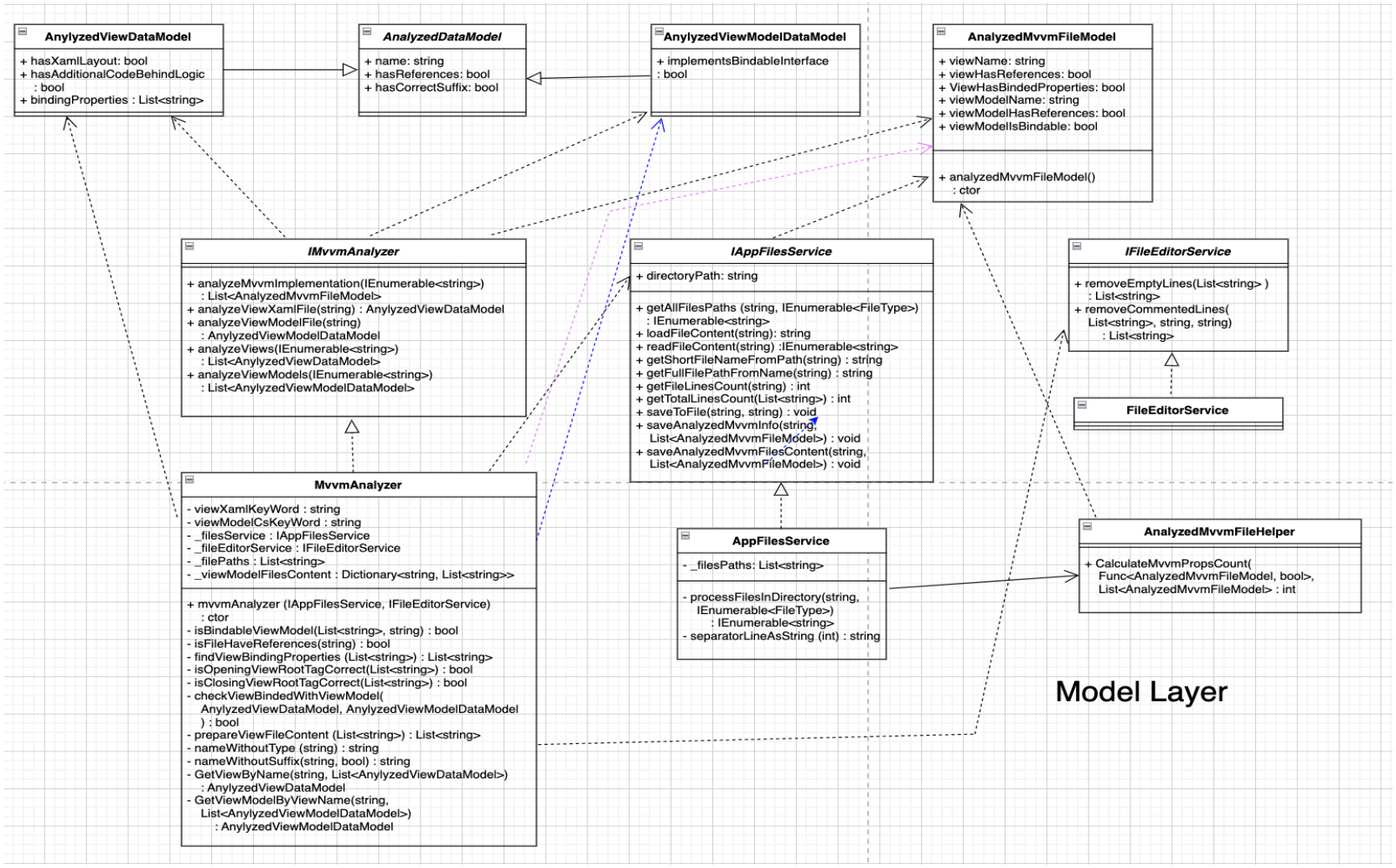


Рисунок 3.4 – Взаємодія між рівнями Model та ViewModel

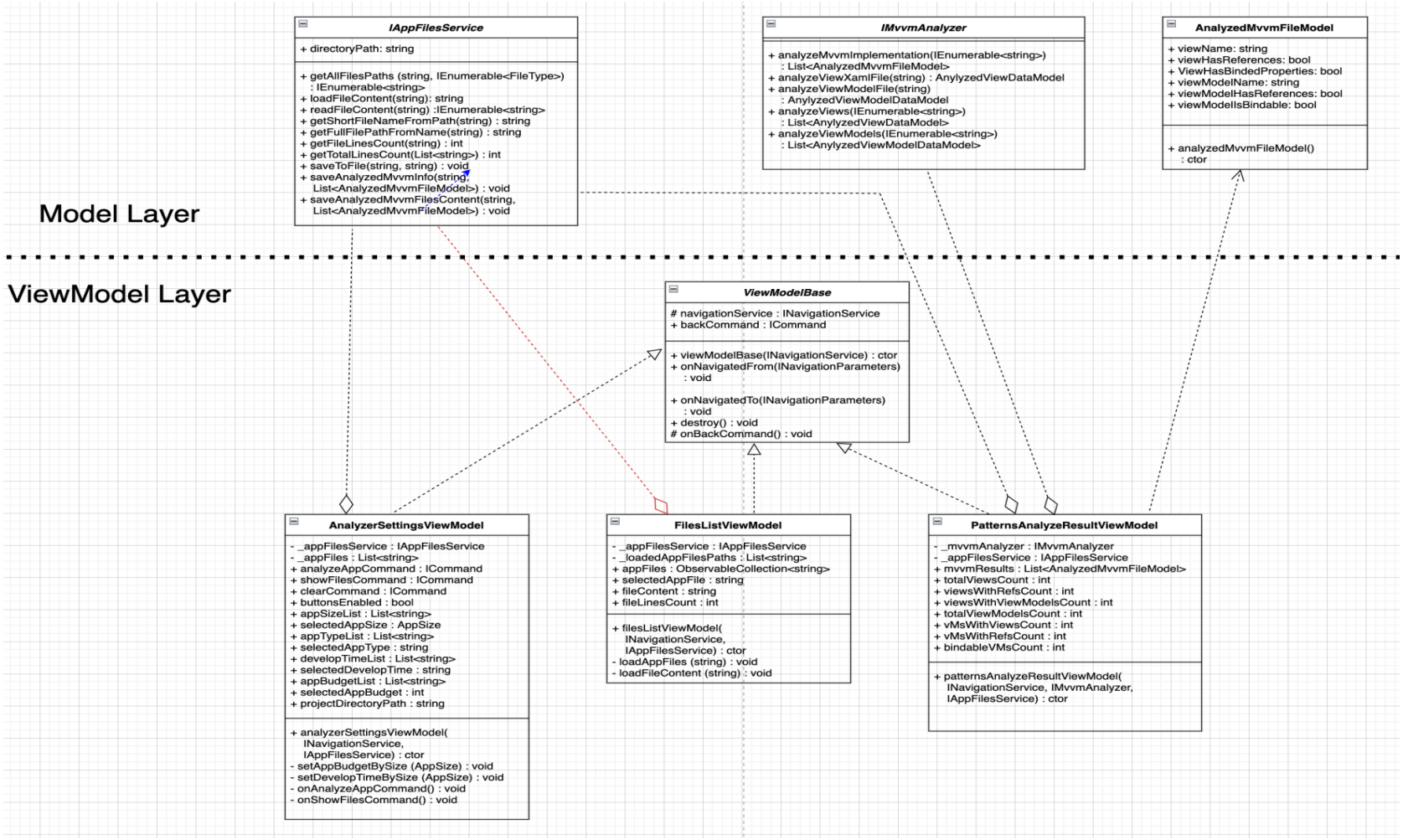
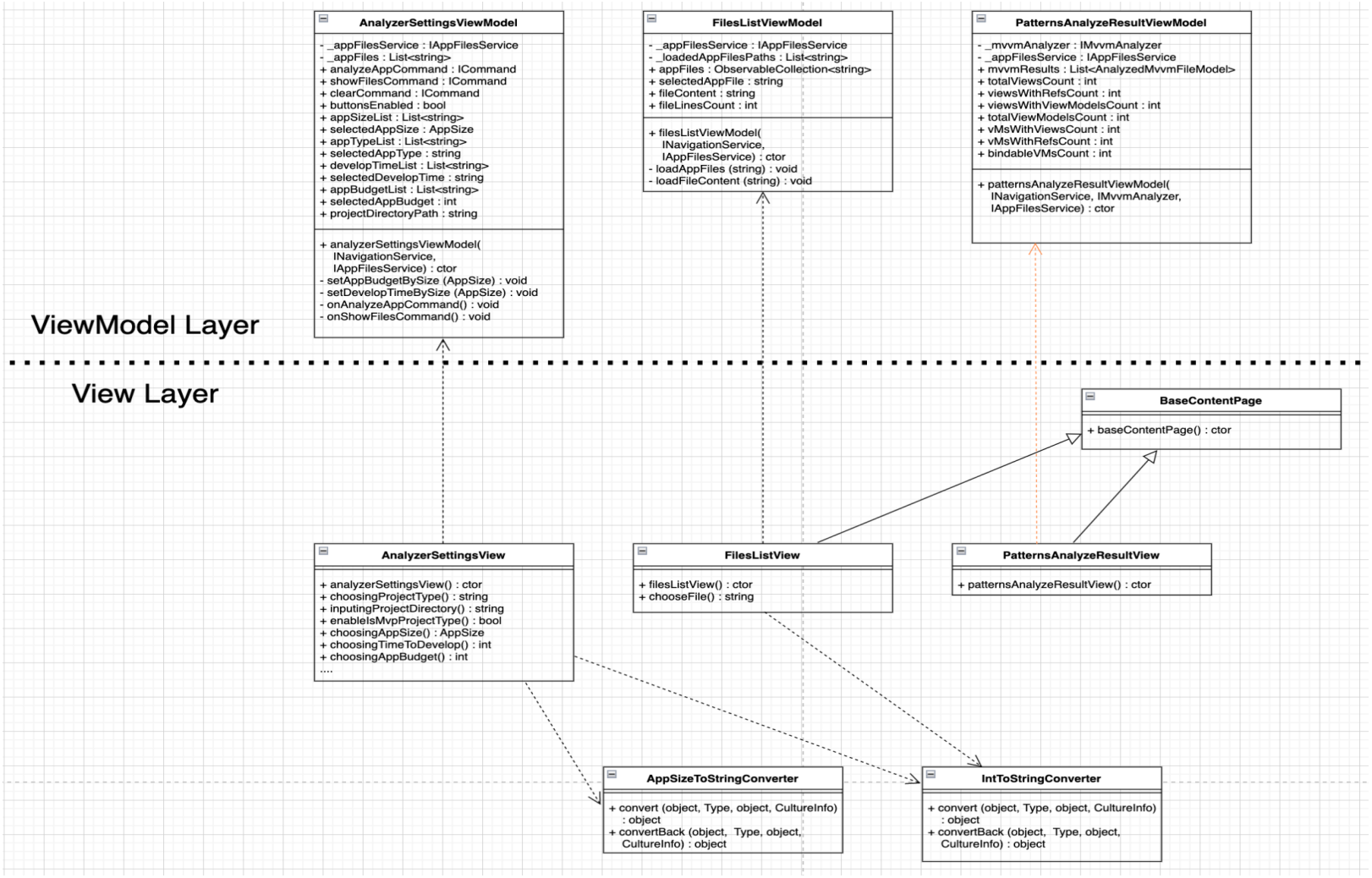


Рисунок 3.5 - Взаємодія між рівнями ViewModel та View



Відкритості – закритості (англ. *Open-closed*) – функціонал сутностей програми може бути розширюється завдяки поліморфізму, але початкові сутності залишаються незмінними. Цьому сприяє такий ООП-принцип як поліморфізм.

Підстановка Барбари Лісков (англ. *Liskov substitution*) – в спроектованій архітектурі наслідники базової моделі можуть використовуватись замість базової і правильність виконання програми при цьому не постраждає.

Принцип простоти (англ. *KISS*) – розроблені модулі та сам програмний додаток максимально просто реалізовані, не перевантажені непотрібними архітектурними рішеннями та функціоналом.

Абстракція та інкапсуляція – принципи які мають йти в парі один з одним, бо лише в парі вони гарантують відділення деталей реалізації від інтерфейсу певного модулю. Завдяки цим принципам програмні інтерфейси сервісів не несуть ніяких деталей про те як вони реалізовані на тому рівні, на якому ці інтерфейси використовуються. Інтерфейс кожного з сервісів лише говорить про те, що він робить і нічого про те як.

Поліморфізм – завдяки якому одні моделі програми розширюють функціонал інших моделей, тим самим створюючи декілька реалізацій одного інтерфейсу батьківського класу та дозволяють при опису залежностей між модулями абстрагуватися завдяки використанню базової моделі - суперкласу, замість використання конкретної.

3.4 Розробка інтерфейсу користувача

3.4.1 Розробка прототипів екранних форм

Прототип екранної форми з налаштуванням аналізатору зображено на рис. 3.6. Також було створено прототип екранної форми для перегляду файлів проекту, що буде проаналізовано. Прототип показано на рис. 3.7.

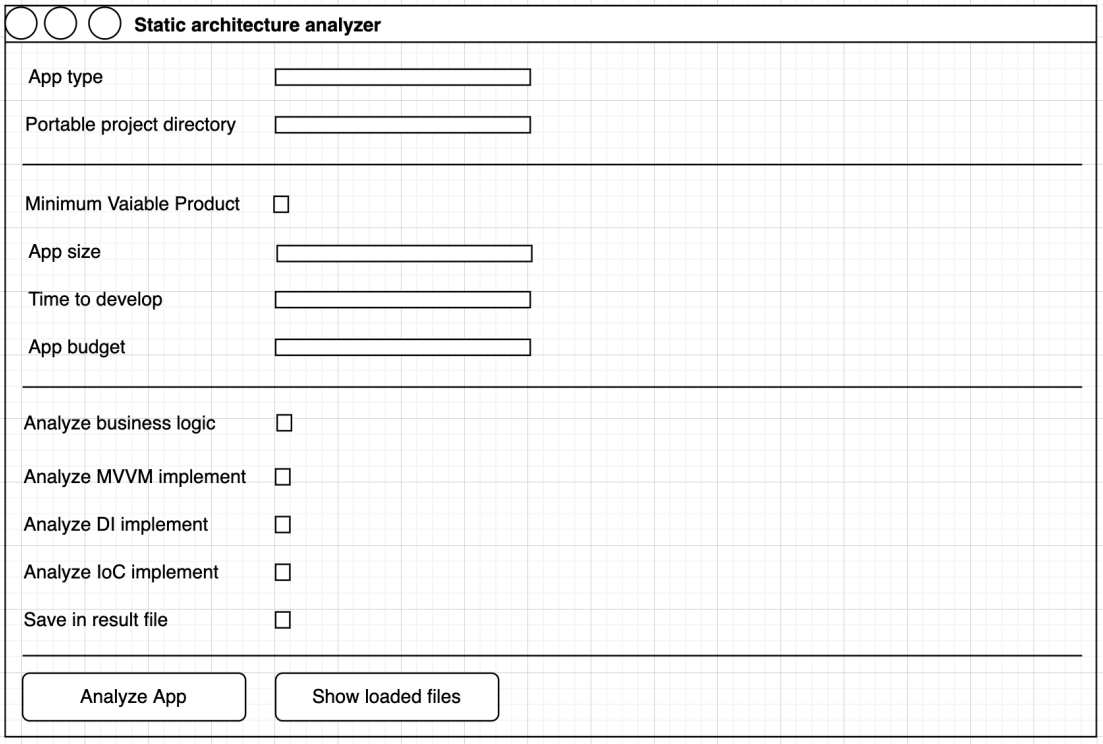


Рисунок 3.6 – Прототип екранної форми з опціями

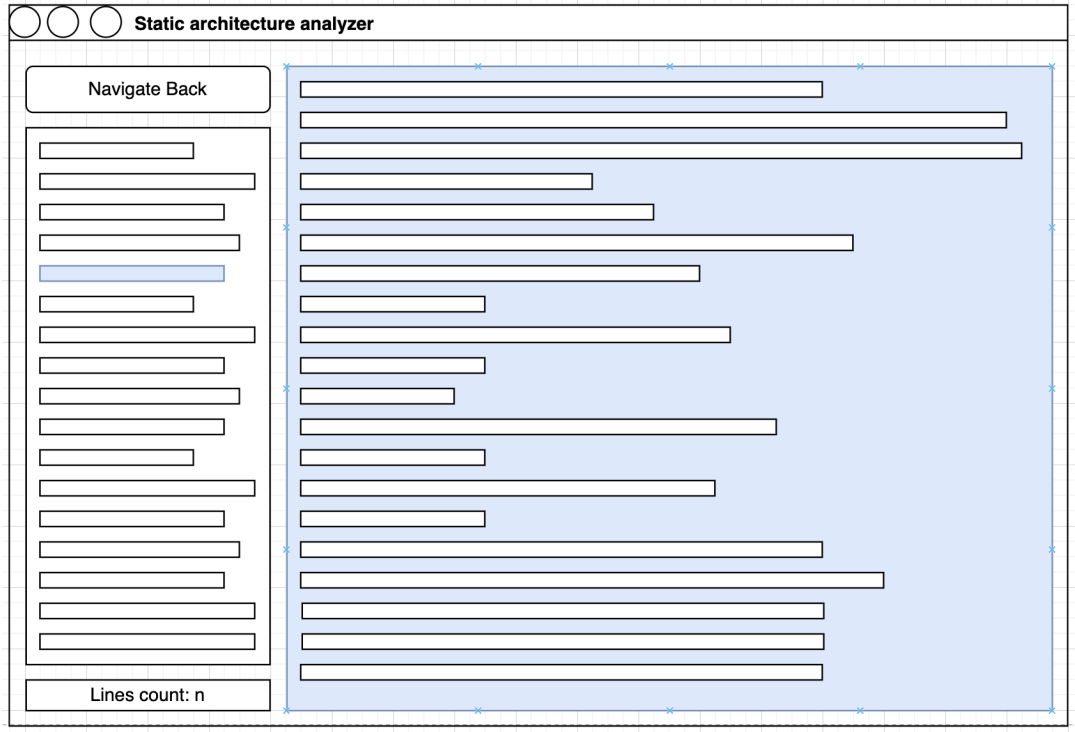


Рисунок 3.7 – Прототип сторінки для перегляду файлів, що аналізуються

Для перегляду результатів аналізу архітектури проекту необхідно окрема сторінка, оскільки на ній відображено багато інформації стосовно використаних патернів. Тому для цього було розроблено прототип екранної форми з результатами аналізу, який показано на рис. 3.8.

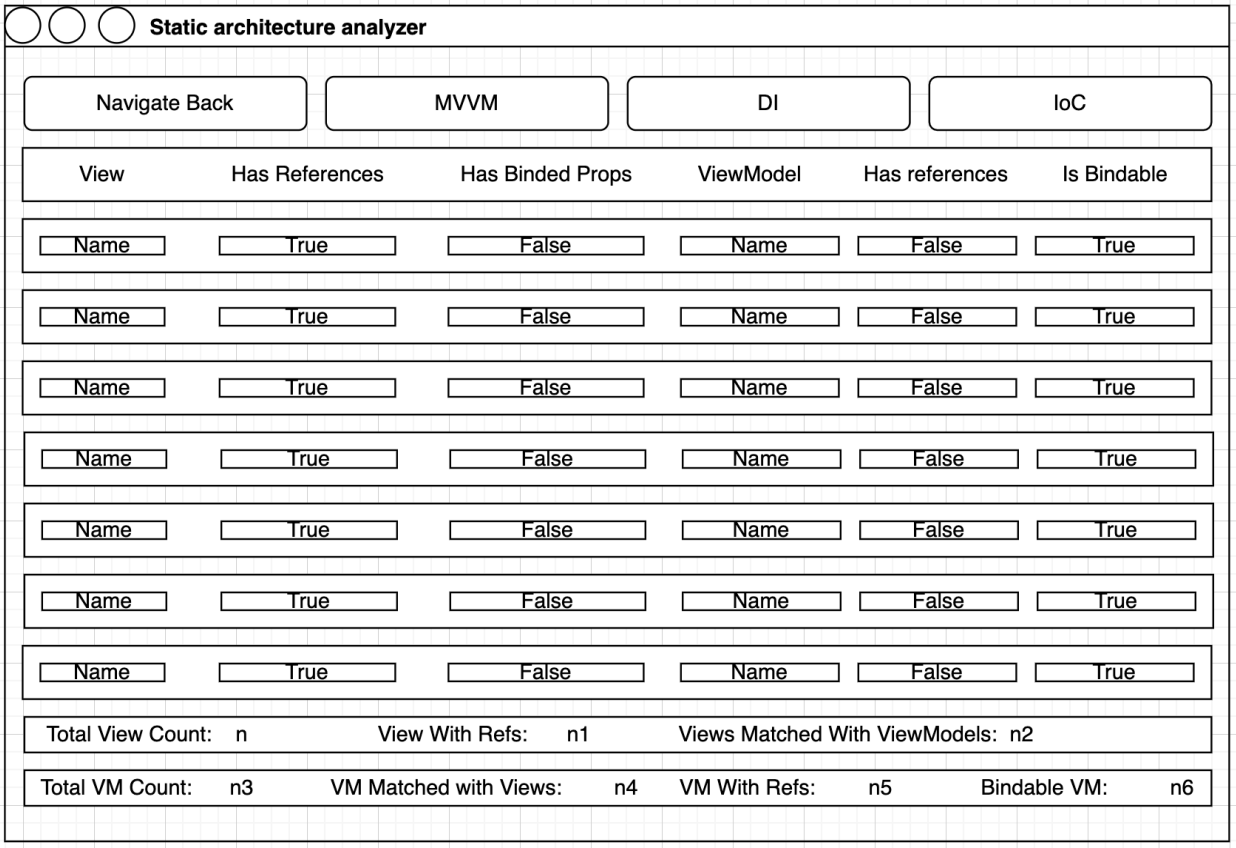


Рисунок 3.8 – Прототип екранної форми з результатами аналізу архітектури

Для створення інтерфейсу користувача буде використано мову Xaml та наступні UI елементи Xamarin.Froms фреймворку: Label, Button, StackLayout, Grid, ListView, CheckBox, Entry, Picker, ContentPage.

3.4.2 Реалізація інтерфейсу користувача

Результати розробки інтерфейсу користувача за прототипами показано на рис. 3.9 – 3.11.

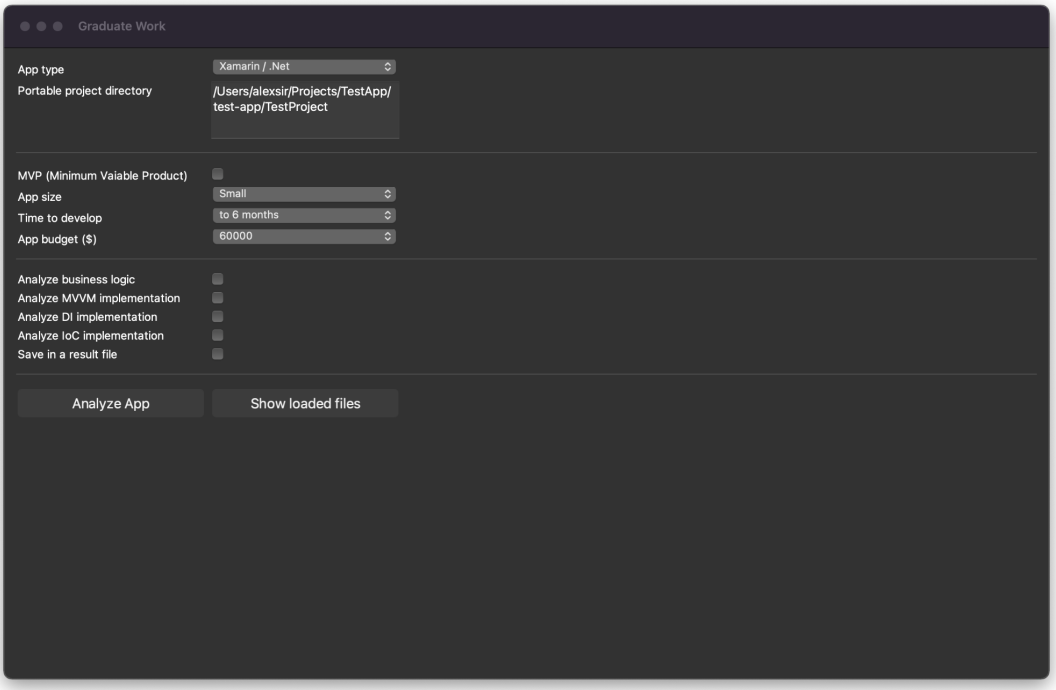


Рисунок 3.9 – Екранна форма з опціями

На рис. 3.9 показана екранна форма на якій вказується мова, фреймворк та директорія проекту, архітектура якого буде аналізуватися та поля з додатковими опціями.

На рис. 3.10 відображена екранна форма на якій можна переглянути всі завантажені файли проекту, що будуть аналізуватись. Крім цього є можливість переглянути їх контент.

На рис. 3.11 зображена екранна форма з таблицею в якій наведено результатами аналізу архітектури деякого проекту, а також додаткову інформацію щодо проаналізованої архітектури. Наприклад загальна кількість файлів певного типу, що були оброблені, кількість посилань на ті чи інші файли, взаємозв’язок між реалізацією певних модулів.

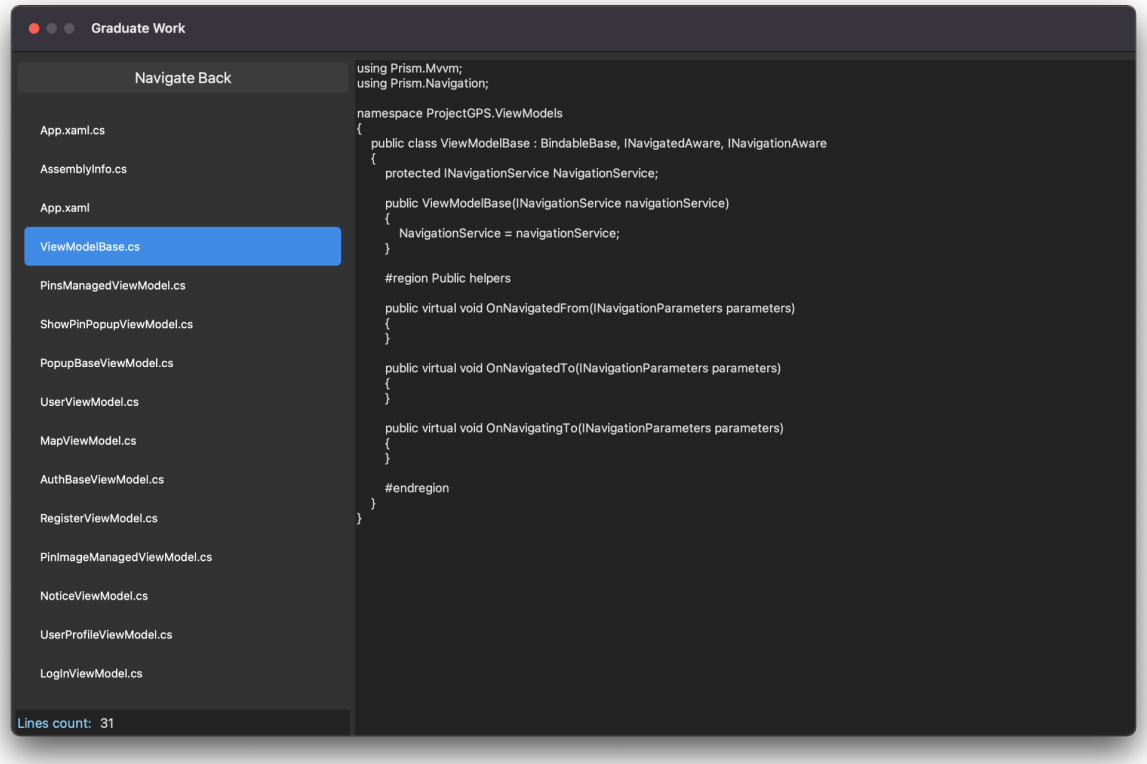


Рисунок 3.10 – Екранна форма для перегляду файлів, що аналізуються

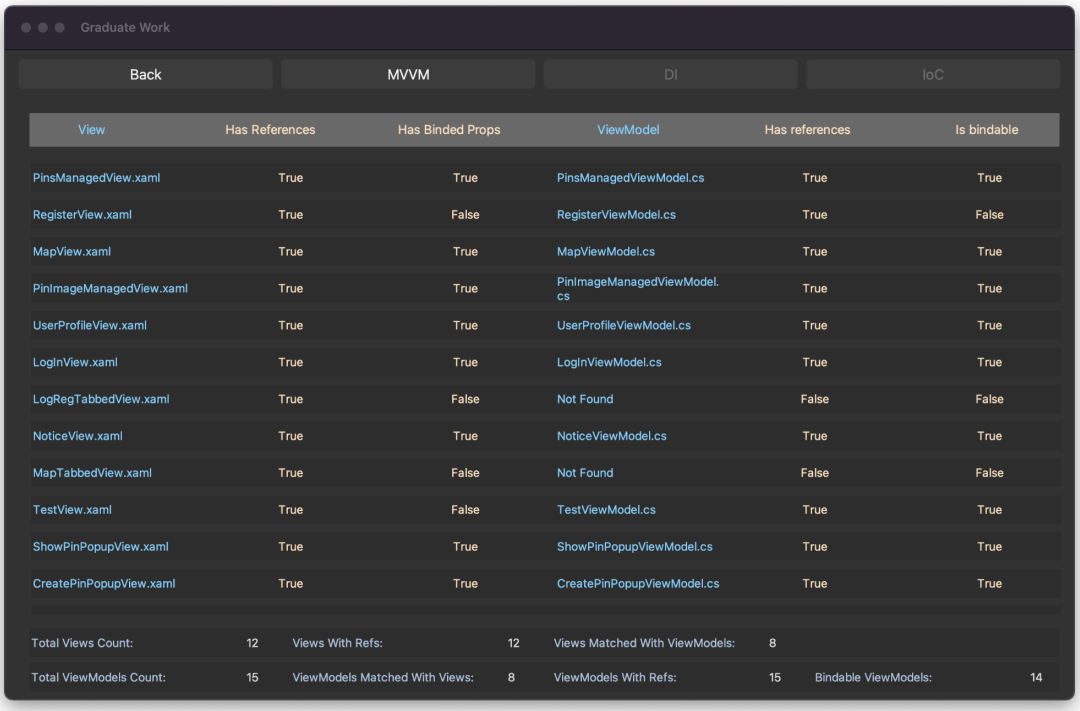


Рисунок 3.11 – Екранна форма з результатами аналізу архітектури

3.5 Тестування та налагодження програми

3.5.1 Аналіз методів тестування

На етапі тестування, найчастіше тестувальникам, потрібно переконатися, що програмний продукт не містить помилок та працює правильно - згідно за документацією проекту. Інколи на даному етапі знаходиться такі особливості які не були враховані раніше в коді та документації. В таких випадках продукт потребує додаткового проектування або навіть перепроектування, розробки та повторного тестування з урахуванням нового функціоналу.

Етап тестування не менш важливий ніж інші етапи життєвого циклу розробки програми, але він також займає багато часу, тому найчастіше процес тестування, як і розробки, ітеративний, тобто проводиться поетапно за кожною реалізованою функцією. І як говорилось раньше у разі виявлення помилок, проводиться їх виправлення та повторне тестування.

Одними із найпоширеніших та ефективних методів тестування є: тестування методом “білої/чорної скриньки”, метод “покриття операторів” та “припущення про похибку”. Існує також регресивне тестування, яке часто проводиться перед релізом програми. Воно проводиться для повторного поверхневого тестування всіх реалізованих функцій або випадків використання при яких раніше було знайдено помилки.

3.5.2 Тестування за методом покриття операторів

Для тестування системи за методом покриття операторів використовувались юніт-тести (англ. *Unit tests*). Було написано декілька тестів, завдяки чому було здійснене покриття операторів та знайдено декілька критичних помилок, переважна більшість яких була в модулі аналізу файлів.

Окрім модулю для аналізу файлів та модулю с реалізацією інтерфейсу користувача, що мав деякі недоліки в шрифтах, інші не містили ніяких помилок. Після виправлення цих помилок було проведено повторне тестування і нових помилок не було виявлено. Це говорить про те, що даний метод тестування системи є досить корисним і саме тому дуже популярний серед розробників програмного забезпечення.

На рис. 3.12 – 3.13 наведено приклад тесту для аналізу файлу інтерфейсу користувача та файлу логіки інтерфейсу користувача, що виявив критичну помилку.

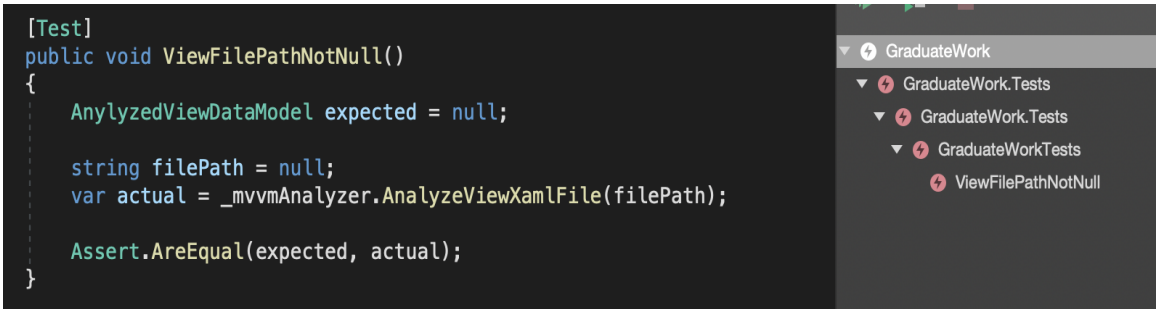


Рисунок 3.12 – Приклад тесту на коректність шляху до файлу інтерфейсу користувача

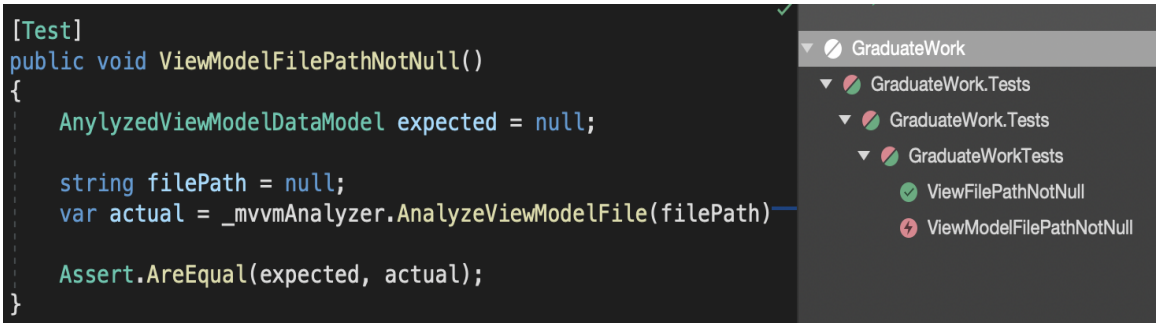


Рисунок 3.13 – Приклад тесту на коректність шляху до файлу логіки інтерфейсу користувача

3.5.3 Тестування за методом припущення про помилку

Окрім тестування за методом покриття операторів програмний засіб також було протестовано за допомогою методу припущення про похибку.

Оскільки найскладнішим в створенні даного продукту є реалізація модулю аналізу файлів, то найбільша увага на етапі тестування даним методом була приділена саме цьому модулю. Було декілька припущень про помилку в сервісах та методах аналізу які найбільш можливо могли мати некоректну реалізацію. Проте милок не було виявлено, так як всі можливі було виправлено на попередньому етапі. Але використовуючи цей метод вдалось знайти та виправити помилки в збереженні файлів з результатами аналізу архітектури деякої програми. Як і при тестуванні модулю аналізу файлів, було виявлено помилки пов'язані з null вказівником, що призводять до появи виключень (англ. *Exception*) в програмі, які призводять до її завершення. Помилки даного типу можуть повністю унеможливити використання деякого функціоналу програмного забезпечення. Приклад тесту, модулю збереження інформації про архітектуру показано на рис. 3.14.

```
[Test]
public void TotalLinesCountZeroIfFilesAreNull()
{
    int expected = 0;

    List<string> filesPaths = null;
    int actual = _appFilesService.GetTotalLinesCount(filesPaths);

    Assert.AreEqual(expected, actual);
}
```

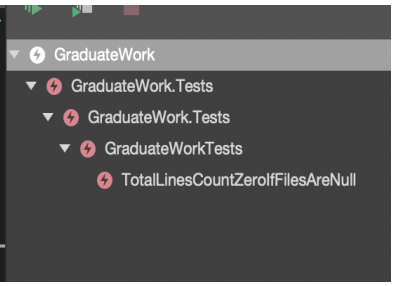


Рисунок 3.14 – Приклад тесту для методу підрахування кількості рядків коду

Висновки до розділу 3

Спроектовано програмне забезпечення для дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS. При проектуванні були розроблені діаграми класів та діаграма використання за допомогою яких було спрощено процес внутрішнього проектування інструментального забезпечення. Програмний засіб було спроектовано з дотриманням популярних принципів, підходів та патернів. Для розподілу сутностей за обов'язками було використано

принципи SOLID та патерн MVVM. Для спрощення реалізації залежностей між модулями було використано підхід, що називається ін’єкція залежностей (англ. *Dependency Injection*). Внаслідок чого модулі можуть бути протестовані, змінені або перевикористані на інших проектах незалежно від інших.

Крім цього в розділі описано процес проектування інтерфейсу користувача, який розпочався з його візуалізації у вигляді прототипів, а після реалізований засобами обраної мови програмування та фреймворку. Завдяки Api Xamarin.Forms програма може легко використовуватися на таких платформах як Windows, macOS, Android та iOS, використовуючи вже написану логіку і реалізований інтерфейс користувача. Тобто не знадобиться додаткова реалізація.

Після проектування та етапу розробки було проведено тестування та відлагодження програми, що дозволило виявити помилки та виправити їх. Це говорить про те, що всі функції працюють коректно та не мають помилок. Для тестування було використано методи покриття операторів та припущення про помилку.

4. ДОСЛІДЖЕННЯ НАСЛІДКІВ ВИКОРИСТАННЯ ПАТЕРНІВ І ЗАГАЛЬНОПРИЙНЯТИХ ПІДХОДІВ В ПОБУДОВІ АРХІТЕКТУРИ КРОС-ПЛАТФОРМНИХ ДОДАТКІВ ПІД ANDROID І IOS

4.1 Підготовка до експерименту

Невід'ємною частиною будь-якого експерименту є етап підготовки даних, що будуть використовуватися відповідно до предметної області стосовно якої проводиться дослідження. В даній ситуації вхідними даними є набір проектів реалізації деяких крос-платформних програмних продуктів для операційних систем Android та IOS. Приклад структури одного з проектів наведено на рис. 4.1. Фактичний розмір S кожного проекту на етапі аналізу оцінено згідно моделі COCOMO в рядках коду KLOC.

Розмір проектів, що будуть використані як вхідні дані для експериментів:

1. KLOC першого проекту дорівнює 4270 рядків;
2. KLOC другого рівне 3710 рядків;
3. KLOC третього 3674 рядки;
4. KLOC четвертого складає 2425;
5. KLOC четвертого складає 4303;

Всі вони реалізовані на об'єктно орієнтованій мові програмування C# та з використанням фреймворку Xamarin.Forms, що має необхідність реалізувати патерн MVVM.

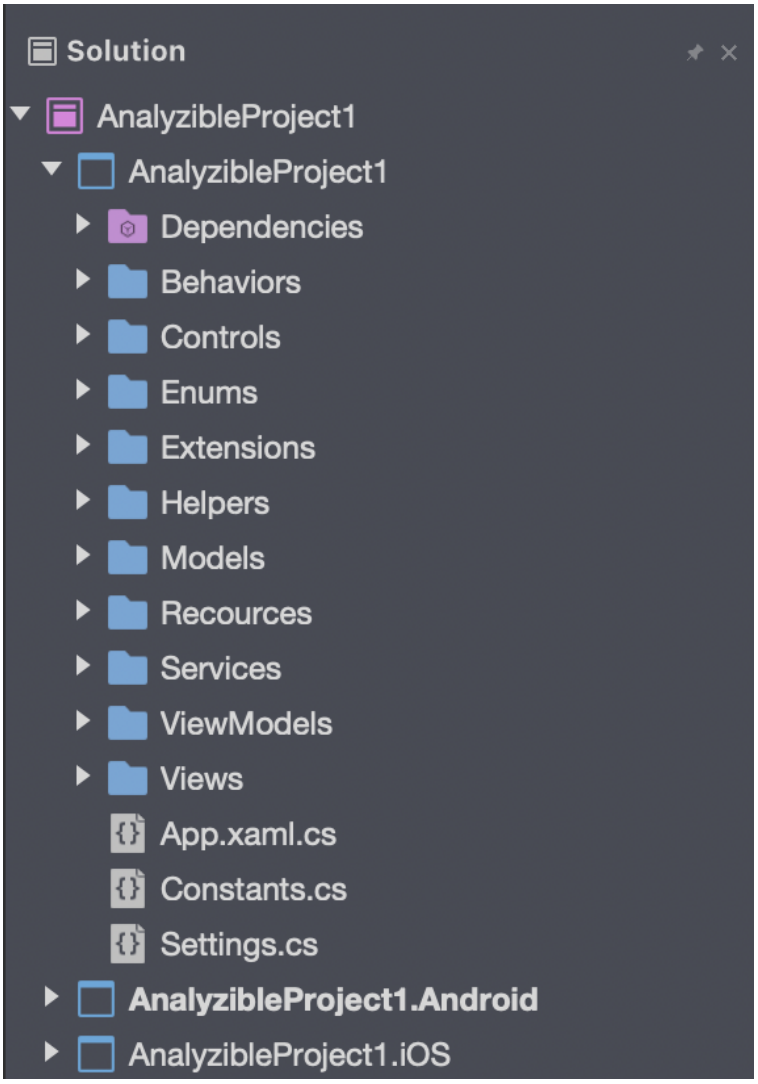


Рисунок 4.1 – Типова структура проекту, що використовується як вхідні дані

4.1.1 Опис програмно-апаратного середовища необхідного для проведення експерименту

Для проведення експерименту використовується персональний компьютер на операційній системі macOS Monterey (версія 12.0.1) з характеристиками наведеними у табл. 4.1. В таблиці наведено лише важливі характеристики для цього експерименту.

Таблиця 4.1 – Характеристики персонального комп’ютеру

Процесор	Intel Core i9 (8 ядер, частота: 2.3 GHz)
ОЗУ	16 GB (тип: DDR4, швидкість: 2667 MHz)
Тип накопичувачу	SSD

4.1.2 Опис підходу до аналізу наслідків використання патернів в архітектурі крос-платформних додатків

При аналізі якості архітектури приділяють увагу тому, як вона впливає на складність програми. При оцінці складності програм, як правило, виділяють три основні групи метрик: метрики розміру програм, метрики складності потоку управління програм та метрики складності потоку даних програм. Це впливає на можливість подальшої підтримки, розширення програмного продукту та найголовніше на його вартість. Складність програми можна було б оцінити в KLOC, тобто чим нижче якість розробленої архітектури тим більшим буде цей показник, але це не є дійсним. Бо використання патернів допомагає зменшити кількість строк коду, але дуже просто може збільшити складність підтримки програмного засобу [43, 44].

При аналізі наслідків використання патернів буде взято до уваги такі показники як:

- вплив на структуру проекту;
- потенційне збільшення або зменшення кількості програмних сутностей або модулів;
- потенційне збільшення або зменшення складності розуміння модулів системи;

- можливе збільшення чи зменшення підсумкової вартості мобільного додатку;
- розміри програмного додатку.

Так як всі додатки побудовані з використанням Xamarin.Forms, то основна увага буде приділятися аналізу якості побудованої архітектури саме з цим патерном.

4.2 Проведення експерименту

Суть проведення експерименту полягає в аналізі архітектури різних крос-платформних проектів за допомогою статичного аналізатору коду з використанням підготовлених вхідних даних та подальшим аналізом результатів цього експерименту. Проведення експерименту складається з наступних дій:

1. запуск статичного аналізатору (рис. 4.2);
2. вказання директорії проекту, що потрібно проаналізувати (рис. 4.3);
3. проведення додаткового налаштування аналізатору (рис. 4.4);
4. перегляд результатів аналізу архітектури проекту (рис. 4.5);

Також додатково можна повернутися до сторінки з завантаженими файлами для аналізу архітектури програмного забезпечення, де їх можна побачити у вигляді списку та ознайомитись з вихідним кодом кожного з таких файлів. При проведенні експериментів даний функціонал використовувався, для ознайомлення з проектом та вмістом файлів, для більш чіткого розуміння отриманих результатів аналізу. Та слід зазначити, що можна провезти іншу послідовність експериментів без використання цього функціоналу, але результати будуть менш точними ніж ті, що будуть отримані завдяки повному застосуванню програмного забезпечення.

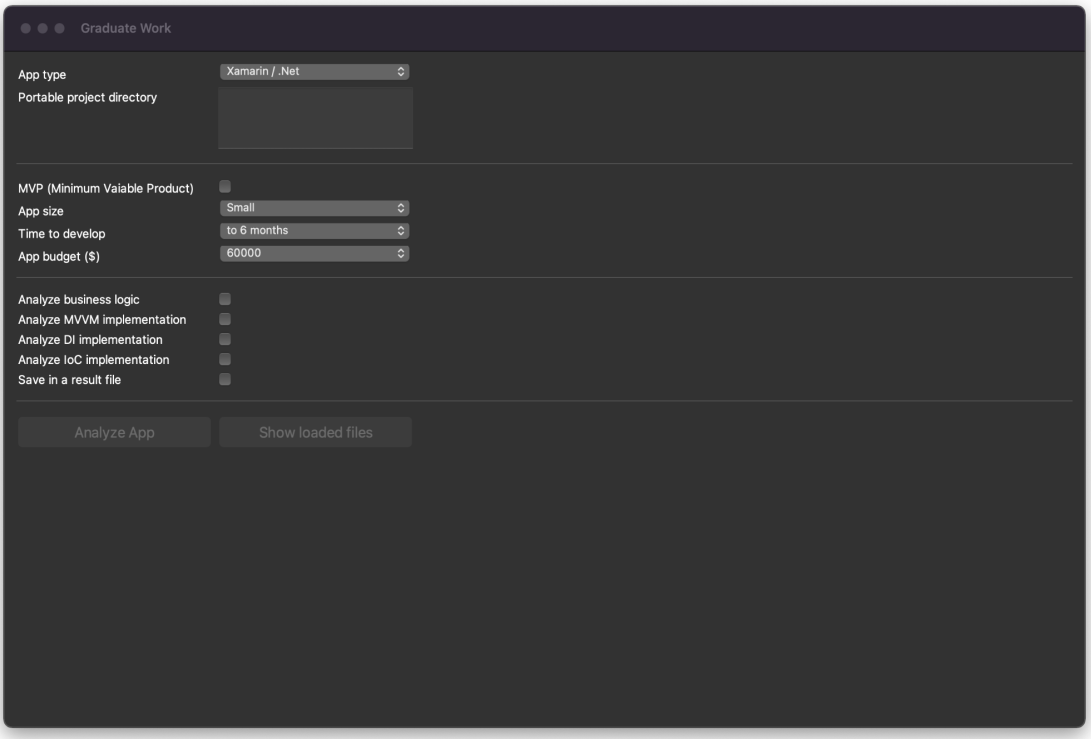


Рисунок 4.2 – Запуск статичного аналізатору

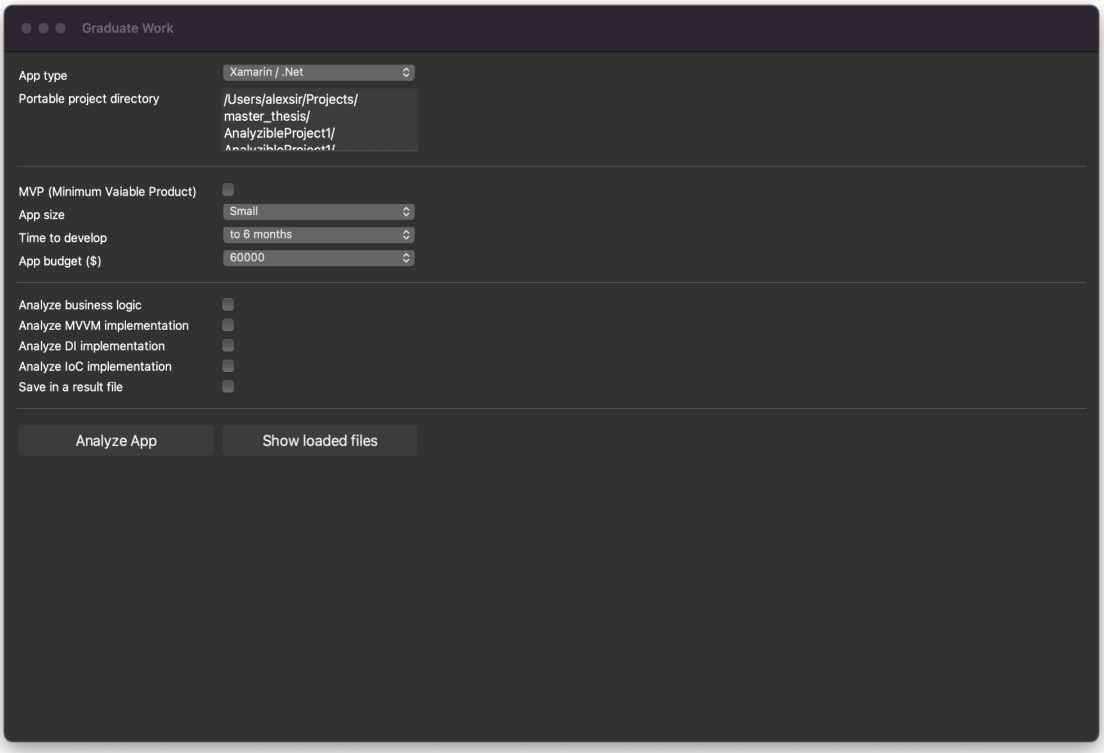


Рисунок 4.3 Вказання директорії проекту

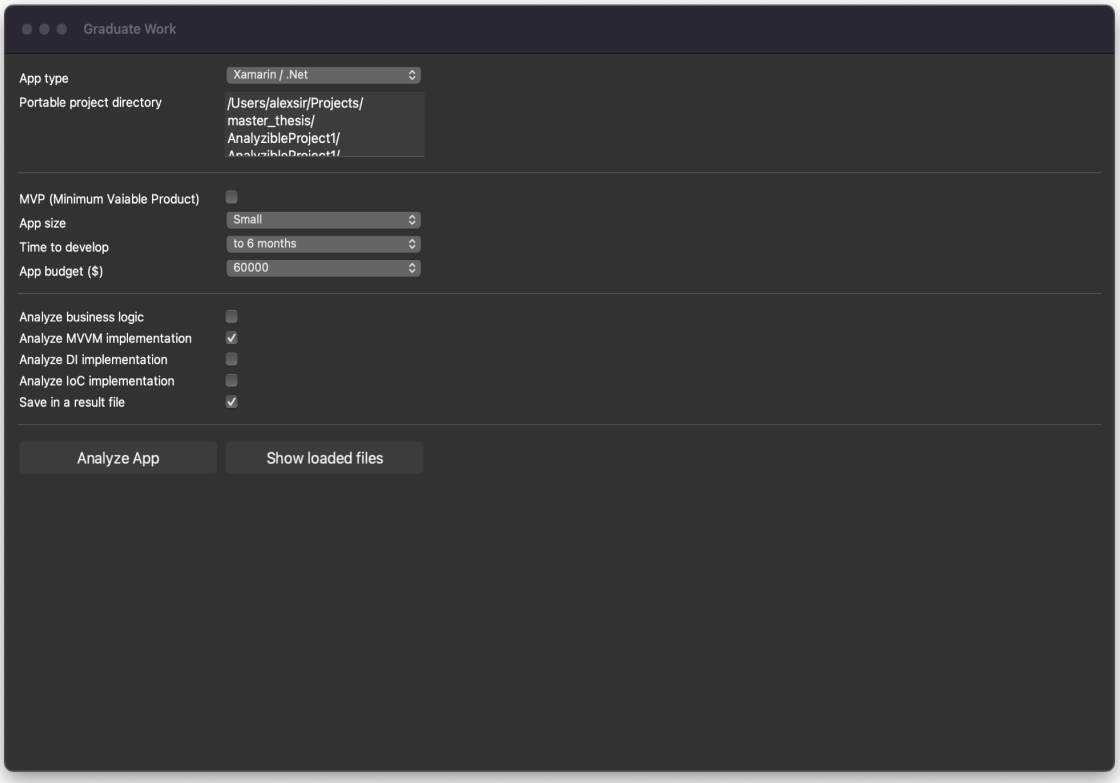


Рисунок 4.4 - Проведення додаткового налаштування статичного аналізатору

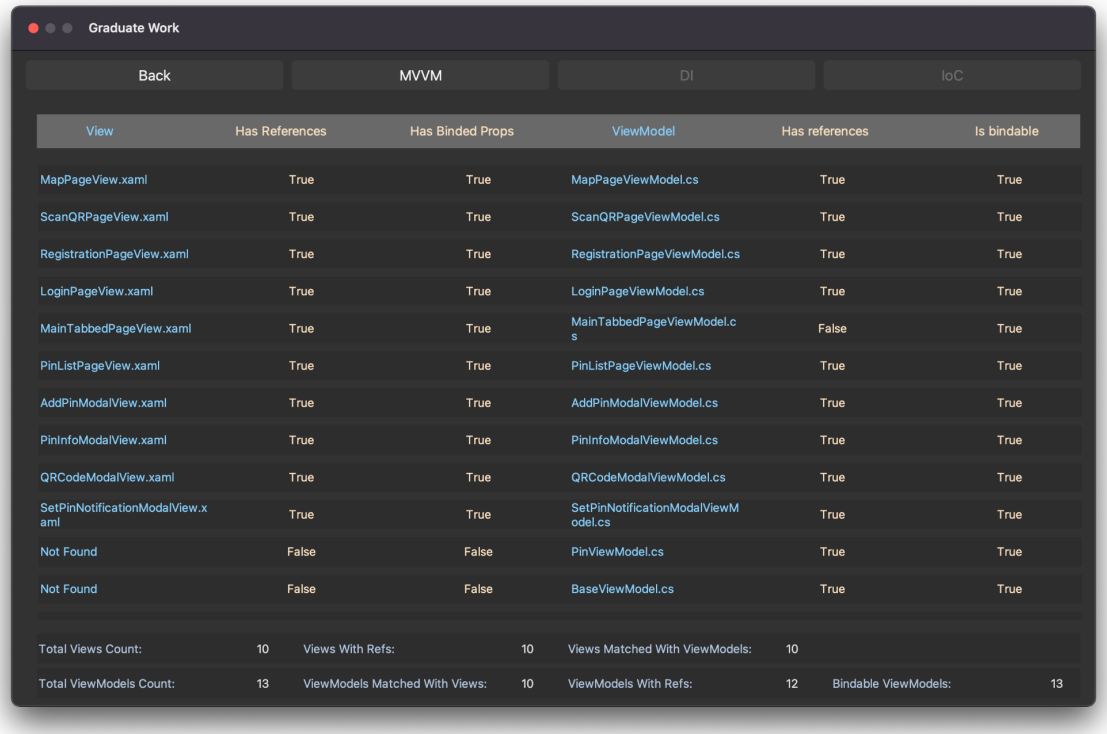


Рисунок 4.5 – Перегляд результатів аналізу архітектури

5. Аналіз результатів експерименту;

4.3 Результати експерименту

Оскільки результатами аналізу є файли з основними і додатковими даними аналізу, а також візуальне їх представлення, то результати першого та всіх наступних експериментів буде представлено на рисунках та у виді двох таблиць з основними результатами аналізу та додатковими.

Результати першого експерименту показано на рис. 4.6 та подано у виді табл. 4.2 – 4.3.

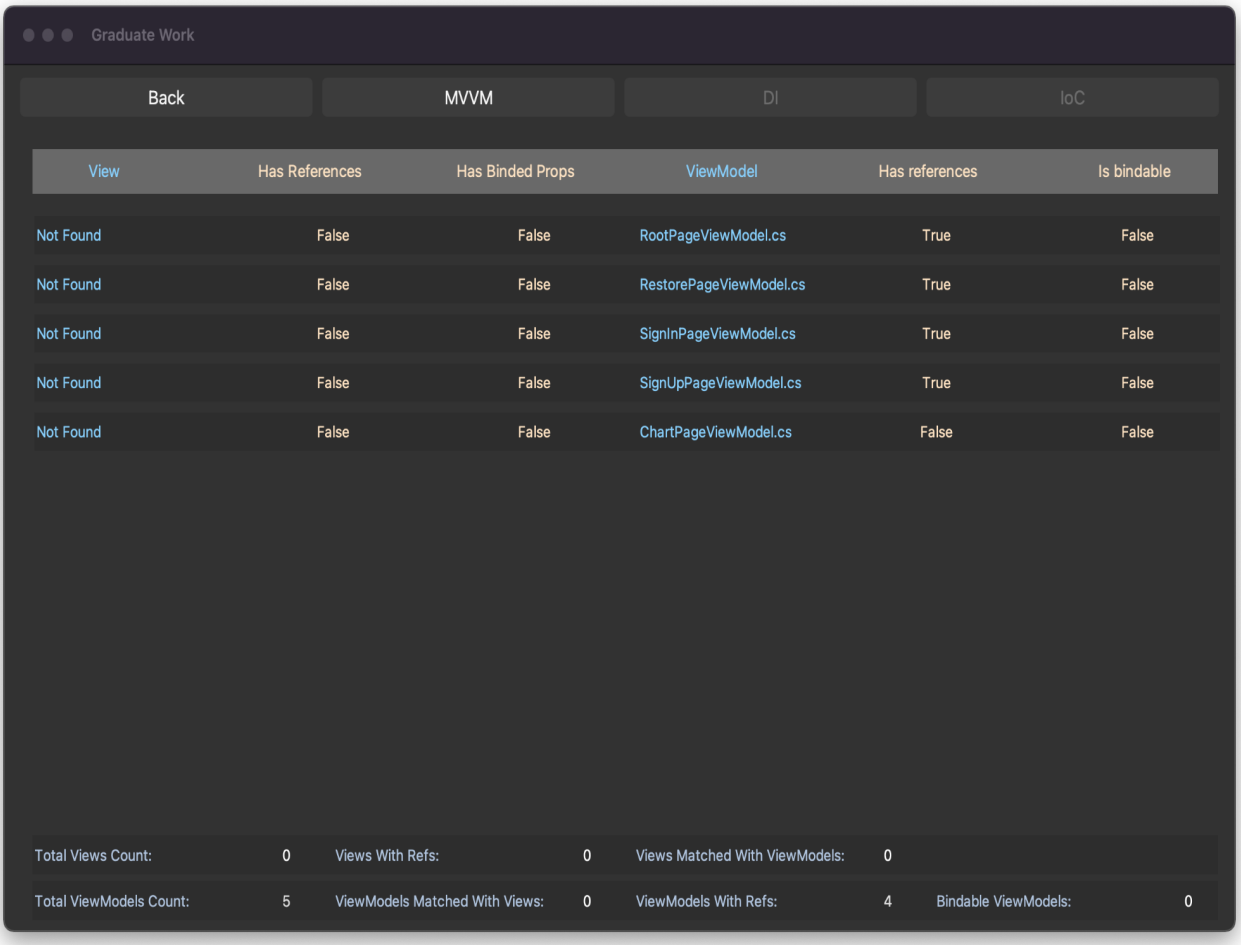


Рисунок 4.6 – Результат першого експерименту

Таблиця 4.2 – Результати першого експерименту (основний аналіз)

Назва View файлу	Має посилання на себе	Зв’язаний із ViewModel файлом	Назва файлу ViewModel	Має посилання на себе	Підтримує зв’язазку із View
Не знайдено	Ні	Ні	RootPageViewModel.cs	Так	Ні
Не знайдено	Ні	Ні	RestorePageViewModel.cs	Так	Ні
Не знайдено	Ні	Ні	SignInPageViewModel.cs	Так	Ні
Не знайдено	Ні	Ні	SignUpPageViewModel.cs	Так	Ні
Не знайдено	Ні	Ні	ChartPageViewModel.cs	Ні	Ні

Таблиця 4.3 – Результати першого експерименту (додатковий аналіз)

Кількість View файлів	0
Кількість View файлів з посиланнями на них	0
Кількість View пов’язаних із ViewModel файлами	0
Кількість ViewModel файлів	5
Кількість ViewModel пов’язаних із View	0
Кількість ViewModel файлів з посиланнями на них	4
Кількість ViewModel, що підтримують зв’язку із View	0

Результати другого експерименту показано на рис. 4.7 та подано у виді табл. 4.4 – 4.5.

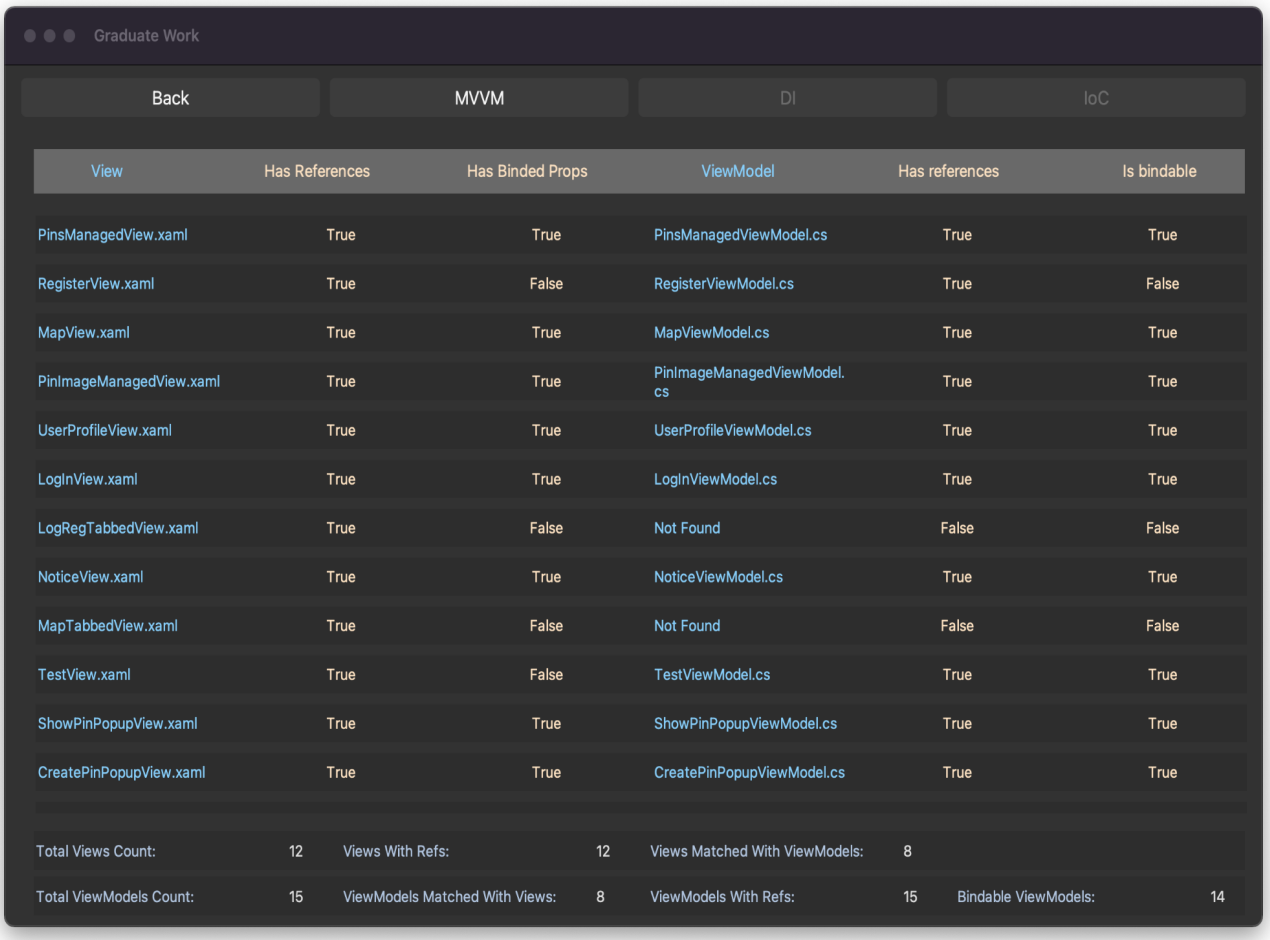


Рисунок 4.7 – Результати другого експерименту

Таблиця 4.4 – Результати другого експерименту (основний аналіз)

Назва View файлу	Має посилання на себе	Зв’язаний із ViewModel файлом	Назва файлу ViewModel	Має посилання на себе	Підтримує зв’язазку із View
1	2	3	4	5	6
PinsManagedView.xaml	Так	Так	PinsManagedViewModel.cs	Так	Так

Продовження таблиці 4.4

1	2	3	4	5	6
RegisterView.xaml	Так	Ні	RegisterViewModel.cs	Так	Ні
MapView.xaml	Так	Так	MapViewModel.cs	Так	Так
PinImageManagedView.xaml	Так	Так	PinImageManagedViewModel.cs	Так	Так
UserProfileView.xaml	Так	Так	UserProfileViewModel.cs	Так	Так
LogInView.xaml	Так	Так	LogInViewModel.cs	Так	Так
LogRegTabbedView.xaml	Так	Ні	Не знайдено	Ні	Ні
NoticeView.xaml	Так	Так	NoticeViewModel.cs	Так	Так
MapTabbedView.xaml	Так	Ні	Не знайдено	Ні	Ні
TestView.xaml	Так	Ні	TestViewModel.cs	Так	Так
ShowPinPopupView.xaml	Так	Так	ShowPinPopupViewModel.cs	Так	Так
CreatePinPopupView.xaml	Так	Так	CreatePinPopupViewModel.cs	Так	Так
Not Found	Ні	Ні	PopupBaseViewModel.cs	Так	Так

Продовження таблиці 4.4

1	2	3	4	5	6
Not Found	Hi	Hi	UserView Model.cs	Так	Так
Not Found	Hi	Hi	AuthBaseV iewModel.c s	Так	Так
Not Found	Hi	Hi	SearchPin ViewModel .cs	Так	Так
Not Found	Hi	Hi	CustomPin ViewModel .cs	Так	Так

Таблиця 4.5 – Результати другого експерименту (додатковий аналіз)

Кількість View файлів	12
Кількість View файлів з посиланнями на них	12
Кількість View пов’язаних із ViewModel файлами	8
Кількість ViewModel файлів	15
Кількість ViewModel пов’язаних із View	8
Кількість ViewModel файлів з посиланнями на них	15
Кількість ViewModel, що підтримують зв’язку із View	14

Результати третього експерименту представлено на рис. 4.8 та подано у виді табл. 4.6 - 4.7.

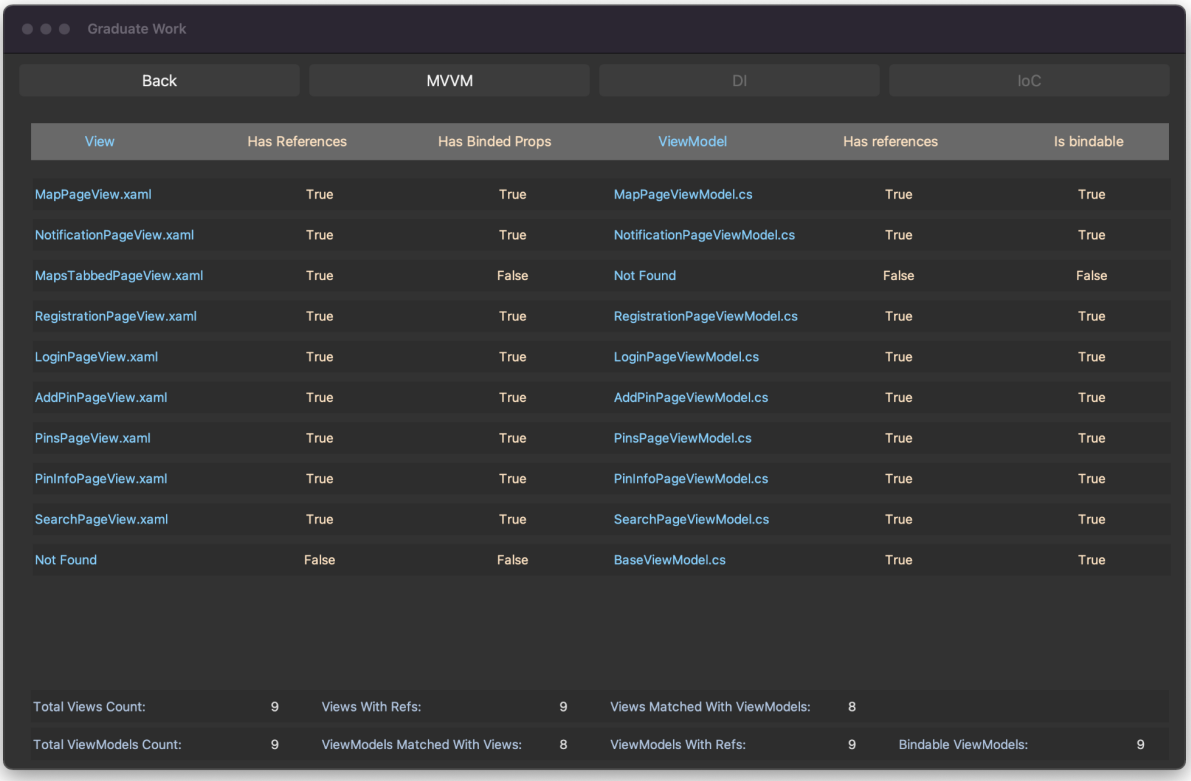


Рисунок 4.8 – Результати третього експерименту

Таблиця 4.6 – Результати третього експерименту (основний аналіз)

Назва View файлу	Має посилання на себе	Зв’язаний із ViewModel файлом	Назва файлу ViewModel	Має посилання на себе	Підтримує зв’язазку із View
1	2	3	4	5	6
MapPageView.xaml	Так	Так	MapPageViewModel.cs	Так	Так
NotificationPageView.xaml	Так	Так	NotificationPageViewModel.cs	Так	Так
MapsTabbedPageView.xaml	Так	Ні	Не знайдено	Ні	Ні

Продовження таблиці 4.6

1	2	3	4	5	6
RegistrationPageView.xaml	Так	Так	RegistrationPageViewModel.cs	Так	Так
LoginPageView.xaml	Так	Так	LoginPageViewModel.cs	Так	Так
AddPinPageView.xaml	Так	Так	AddPinPageViewModel.cs	Так	Так
PinsPageView.xaml	Так	Так	PinsPageViewModel.cs	Так	Так
PinInfoPageView.xaml	Так	Так	PinInfoPageViewModel.cs	Так	Так
SearchPageView.xaml	Так	Так	SearchPageViewModel.cs	Так	Так
Не знайдено	Ні	Ні	BaseViewModel.cs	Так	Так

Таблиця 4.7 – Результати третього експерименту (додатковий аналіз)

Кількість View файлів	9
Кількість View файлів з посиланнями на них	9
Кількість View пов’язаних із ViewModel файлами	8
Кількість ViewModel файлів	9
Кількість ViewModel пов’язаних із View	8
Кількість ViewModel файлів з посиланнями на них	9

Продовження таблиці 4.7

Кількість ViewModel, що підтримують зв’язку із View	9

Результати четвертого експерименту показано на рис. 4.9 та подано у виді табл. 4.8 – 4.9.

В ході даного експерименту програмним забезпеченням було виявлено лише декілька файлів і це є однією із ситуацій коли в нагоді стане функція перегляду всіх файлів проекту та їх вмісту.

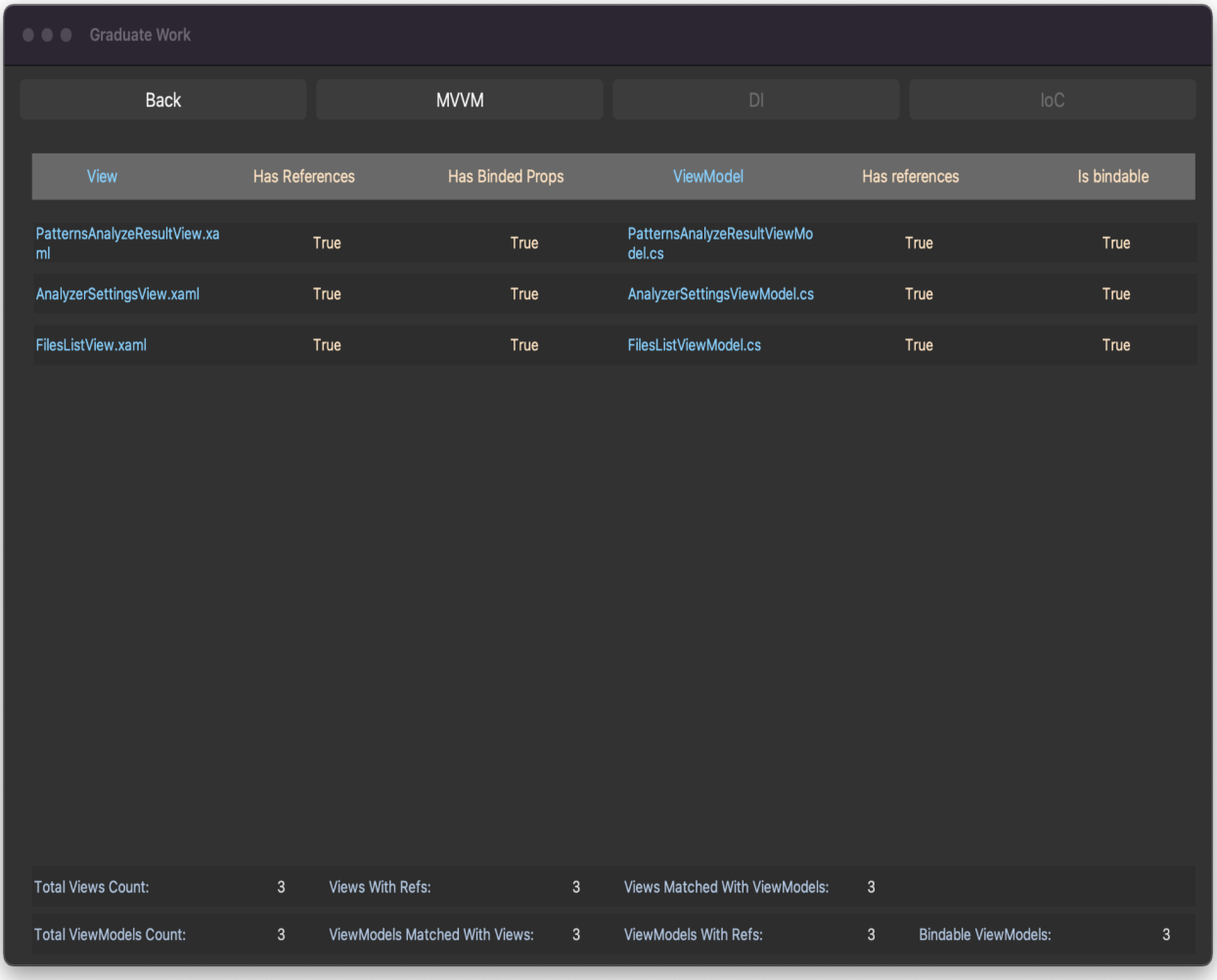


Рисунок 4.9 – Результати четвертого експерименту

Таблиця 4.8 – Результати четвертого експерименту (основний аналіз)

Назва View файлу	Має посилання на себе	Зв’язаний із ViewModel файлом	Назва файлу ViewModel	Має посилання на себе	Підтримує зв’язазку із View
PatternsAnalyzeResultView.xaml	Так	Так	PatternsAnalyzeResultViewModel.cs	Так	Так
AnalyzerSettingsView.xaml	Так	Так	AnalyzerSettingsViewModel.cs	Так	Так
FilesListView.xaml	Так	Так	FilesListViewModel.cs	Так	Так

Таблиця 4.9 – Результати четвертого експерименту (додатковий аналіз)

Кількість View файлів	3
Кількість View файлів з посиланнями на них	3
Кількість View пов’язаних із ViewModel файлами	3
Кількість ViewModel файлів	3
Кількість ViewModel пов’язаних із View	3
Кількість ViewModel файлів з посиланнями на них	3
Кількість ViewModel, що підтримують зв’язку із View	3

Виходячи з розмірів проекту, що використовувався в якості вхідних даних для четвертого експерименту, можна помітити як це впливає на результати.

Результати п’ятого експерименту продемонстровано на рис. 4.10 та подано у виді табл. 4.10 – 4.11.

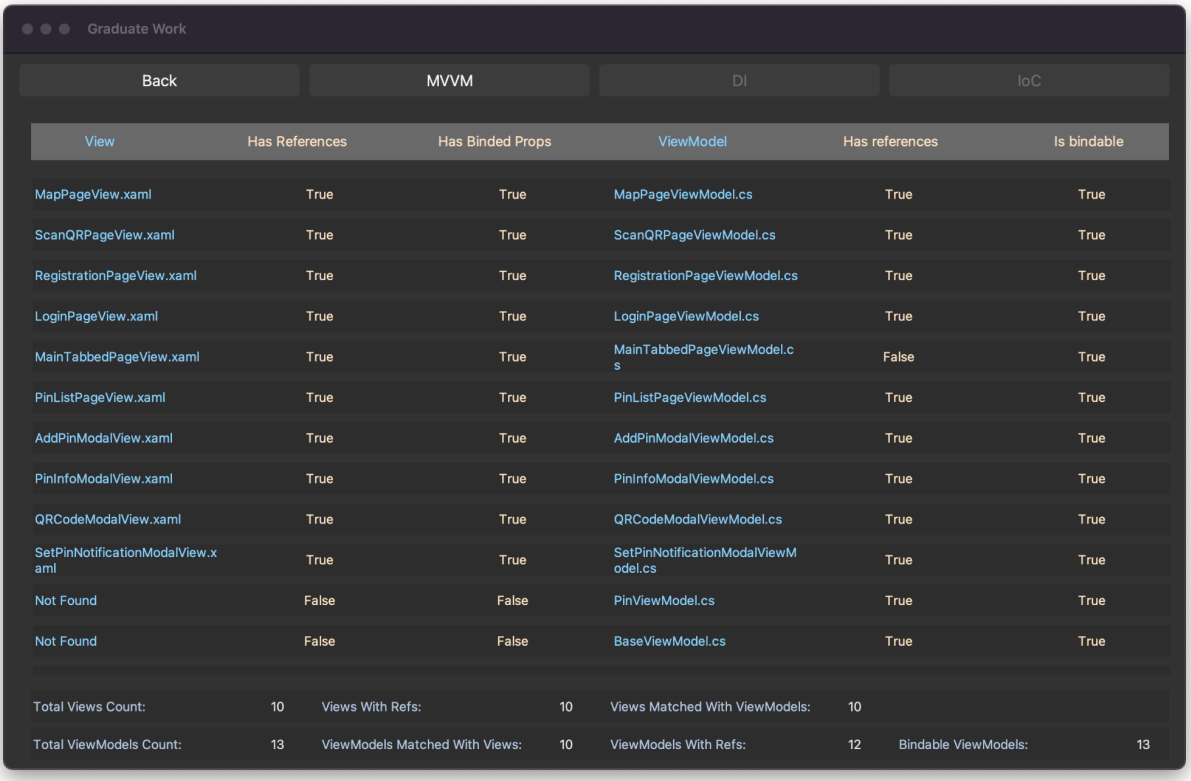


Рисунок 4.10 – Результати п’ятого експерименту

Таблиця 4.10 – Результати п’ятого експерименту (основний аналіз)

Назва View файлу	Має посилання на себе	Зв’язаний із ViewModel файлом	Назва файлу ViewModel	Має посилання на себе	Підтримує зв’язазку із View
1	2	3	4	5	6
MapPageView.xaml	Так	Так	MapPageViewModel.cs	Так	Так
ScanQRPageView.xaml	Так	Так	ScanQRPageViewModel.cs	Так	Так
RegistrationPageView.xaml	Так	Так	RegistrationPageViewModel.cs	Так	Так

Продовження таблиці 4.10

1	2	3	4	5	6
LoginPage View.xaml	Так	Так	LoginPage ViewModel .cs	Так	Так
MainTabbe dPageView. xaml	Так	Так	MainTabbe dPageView Model.cs	Ні	Так
PinListPag eView.xaml	Так	Так	PinListPag eViewMod el.cs	Так	Так
AddPinMo dalView.xa ml	Так	Так	AddPinMo dalViewMo del.cs	Так	Так
PinInfoMo dalView.xa ml	Так	Так	PinInfoMo dalViewMo del.cs	Так	Так
QRCodeM odalView.x aml	Так	Так	QRCodeM odalViewM odel.cs	Так	Так
SetPinNotif icationMod alView.xam l	Так	Так	SetPinNotif icationMod alViewMod el.cs	Так	Так
Не знайдено	Ні	Ні	PinViewM odel.cs	Так	Так
Не знайдено	Ні	Ні	BaseView Model.cs	Так	Так
Не знайдено	Ні	Ні	BaseModal ViewModel .cs	Так	Так

Таблиця 4.11 – Результати п’ятого експерименту (додатковий аналіз)

Кількість View файлів	10
Кількість View файлів з посиланнями на них	10
Кількість View пов’язаних із ViewModel файлами	10
Кількість ViewModel файлів	13
Кількість ViewModel пов’язаних із View	10
Кількість ViewModel файлів з посиланнями на них	12
Кількість ViewModel, що підтримують зв’язку із View	13

При проведенні четвертого експерименту в якості вхідних даних було використано проект, з найменшим значенням KLOC, для другого та третього – проекти середнього розміру, а для проведення першого та п’ятого експериментів було використано проекти з найбільшим показником KLOC, що надає можливість порівняти різні реалізації паттернів, в цьому разі патерну MVVM, на різних за розмірами проектах та виявити певні закономірності. Крім цього це дозволяє розгялнути можливі типи проектів які трапляються під час розробки програмного забезпечення в реальному житті.

Кожен з експериментів має свій результат, що являє собою інформацію про стан архітектури для кожного з проаналізованих проектів, тобто чи реалізовано патерн MVVM для проекту, в якій мірі, чи мають файли проекту посилання на них, кількість файлів інтерфейсу користувача, кількість файлів логіки інтерфейсу користувача, їх назви та чи пов’язані вони між собою за допомогою механізму зв’язування графічного інтерфейсу з логікою цього інтерфейсу, а також інформацію про те, чи підтримує певний файл логіки інтерфейсу користувача можливість зв’язки з файлом представлення.

Висновки до розділу 4

В даному розділі продемонстровано п’ять експериментів, а також їх результати стосовно питання дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS. Враховуючи той факт, що патернів проектування існує дуже велика кількість, то аналіз результатів буде проведено базуючись на результатах реалізації патерну MVVM, який є одним із найпоширеніших патернів при побудові мобільних додатків. При аналізі було знехтувано файлами з базовою реалізацією логіки інтерфейсу користувача або базовою реалізацією цього інтерфейсу, бо ці файли в цілому не можуть залежати один від одного.

З результатів експерименту, де в якості вхідних даних виступав найменший проект з KLOCK рівним 2425 рядків коду (четвертий експеримент) можна зробити такі висновки, що при наявності трьох файлів інтерфейсу користувача та трьох файлів логіки інтерфейсу користувача проект реалізує патерн MVVM на 100 %, а файлів, що не використовуються взагалі немає. Можемо зробити висновок, що складність підтримки або розширення даного проекту низька, а отже і вартість даного процесу теж не перевищує розумні показники. Проте реалізація патерну займає t годин часу, що залежать від кваліфікації розробника, а у разі нехтування його реалізації час на розробку проекту зменшується на ті ж самі t годин, що певною мірою знижує його вартість. Та насправді ціна проекту лише зросте через те, що процес підтримки стане набагато складніше з плином часу, якщо програмний продукт планується розвивати в майбутньому. Станеться це через те, що з ростом кількості сутностей програмний код буде ускладнюватися, бо в результаті модулі буде майже неможливо протестувати або розширити незалежно один від одного. Зростаюча кількість помилок буде заважати

реалізації нового функціоналу, а збільшення вартості буде рости з кожною новою задачою.

За результатами експериментів для яких використовувалися проекти середнього розміру (другий та третій експеримент) з KLOCK рівним 3710 та 3674 рядки видно, що у другому проекті патерн реалізовано приблизно на 80 – 90%, бо деякі з файлів інтерфейсу та логіки інтерфейсу користувача не зв’язані між собою за механізмом прив’язки (англ. *Binding*). Третій проект реалізує MVVM шаблон на 100%. Як і в попередньому випадку, такий відсоток дає гарантію того, що складність підтримки продукту, як і його вартість, буде невеликою навіть у разі росту проекту.

Аналізуючи результати першого та п’ятого експериментів де вхідними даними були проекти найбільшого розміру з показниками KLOCK 4720 та 4303 рядків вихідного коду видно, що для першого проекту патерн MVVM реалізовано приблизно на 0%, тобто не реалізовано взагалі. Спираючись на результати попередніх експериментів відразу стає зрозуміло, що складність розширення і підтримки даного програмного продукту буде досить швидко зростати збільшуючи вартість розробки. Якщо не провести рефакторинг, то це може призвести до великих збитків або навіть провалу ідеї і роботи компанії. У випадку з п’ятим проектом патерн реалізовано на 100%, що на перший погляд спрощує етапи тестування, підтримки та розширення функціоналу програмного додатку. Але спираючись на розміри проекту та можливий його ріст в майбутньому не важко зрозуміти, що і кількість сутностей буде зростати. Цей факт буде ускладнювати процес розробки, адже ці сутності в проекті утворюють граф залежностей одна від одної і чим він більше тим складніше розробнику тримати його в розумі під час роботи над проектом. Щонайменше це призведе до більш часових витрат. Також це може стати основою для майбутніх помилок на етапі розробки програмного засобу.

На практиці великі проекти реалізують не один патерн або архітектурний підхід, що збільшує складність на всіх етапах розробки. Як результат збільшення витрат неминуче.

З отриманих результатів стали зрозумілими наступні тенденції:

- зменшення вартості продукту шляхом зменшення необхідного часу на його розробку за рахунок нехтування патернами проектування, насправді лише ілюзія. В майбутньому складність розробки буде збільшуватись в геометричній прогресії, що без сумніву вплине на його вартість в гіршу сторону;
- зменшення вартості проекту за рахунок зменшення необхідного часу на розробку нехтуючи реалізацією певного патерну можна допускати лише в тому разі, коли проект на планується підтримувати та супроводжувати в майбутньому. Прикладом такої ситуації може бути розробка MVP додатку. Найчастіше це трапляється на проектах невеликого розміру;
- реалізація шаблонів проектування може спрощувати процес підтримки програмного засобу та утримувати вартість цього процесу на адекватному рівні в майбутньому;
- збільшення кількості реалізованих патернів та архітектурних підходів на проекті з часом ускладнює його розуміння, сповільнює розробку та слугує основою для майбутніх помилок в функціоналі, що також призводить до збільшення його вартості. Бажання покращити архітектуру завдяки імплементації додаткового шаблону, може призвести до його погіршення.

5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Вимоги безпеки праці під час виконання робіт з розробки програмного забезпечення.

Наказом Мінсоцполітики 14.02.2018 № 207, який вступив в силу 18.05.2018 р., затверджені Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями. Ці Вимоги поширюються на всіх суб’єктів господарювання незалежно від форм власності, організаційно-правової форми і видів діяльності та встановлюють мінімальні вимоги безпеки та захисту здоров’я під час здійснення роботи, пов’язаної з використанням екранних пристроїв незалежно від їхнього типу та моделі. Ці Вимоги не обмежують права роботодавця встановлювати більш жорсткі та/або спеціальні вимоги безпеки і захисту здоров’я та життя працівників під час роботи з екранними пристроями, якщо це не суперечить чинному законодавству [45].

5.1.1 Шкідливі виробничі фактори при роботі з ЕОМ

Працівники, задіяні на роботах, пов'язаних з періодичною або постійною роботою за комп'ютером, піддаються впливу шкідливих виробничих факторів, основними з яких є:

- тривале сидяче положення, малорухомість;
- монотонність праці в окремих випадках;
- електромагнітне випромінювання;
- напруга зору, пам’яті, уваги;
- нерівномірний розподіл яскравості в полі зору;
- запиленість повітря робочого приміщення.

Всі перераховані чинники можуть призводити до таких проблем зі здоров'ям як ожиріння, остеохондроз, сколіоз, проблем із зором, головний біль, стрес, що є причиною інфаркту, захворювання органів дихання (алергія) [46].

Для підтримання безпечних умов праці необхідно дотримуватися заданих стандартів, що регулюються: ст. 153 Кодексу законів про працю України та ст. 6 Закону України "Про охорону праці", ст. 158 Кодексу законів про працю України, ч. 1 ст. 13 Закону України "Про охорону праці", "Правилами охорони праці під час експлуатації електронно-обчислювальних машин", затверджених Наказом Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010 року № 65 та "Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин", затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 (ДСанПіН 3.3.2-007-98). Зазначені нормативно-правові акти встановлюють санітарно-гігієнічні вимоги до приміщення, в якому розташоване робоче місце, власне до робочого місця, освітлення, рівнів вібрації і шуму, мікроклімату в приміщенні тощо [47, 57].

Під час організації робочого місця та обладнання робочого місця висота робочої поверхні робочого столу має регулюватися в межах 680-800 мм, а ширина і глибина – забезпечувати можливість виконання операцій у зоні досяжності моторного поля (рекомендовані розміри: 600-1400мм, глибина – 800-1000мм). Робочий стіл повинен мати простір для ніг заввишки не менше ніж 600мм, завширшки не менше ніж 500мм, завглибшки (на рівні колін) не менше ніж 450мм, на рівні простягнутої ноги не менше ніж 650мм. Робочий стілець має бути підйомно-поворотним, регульованим за висотою, з кутом і нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння поверхня сидіння має бути плоскою, передній край – заокругленим. Висота поверхні сидіння має регулюватися в межах 400-500мм, а ширина і глибина становити не

менше ніж 400мм. Кут нахилу сидіння – до 15 градусів вперед і до 5 градусів назад. Висота спинки стільця має становити (300+-20) мм, ширина – не менше ніж 380 мм, радіус кривизни горизонтальної площини – 400мм. Кут нахилу спинки має регулюватися в межах 1-30 градусів від вертикального положення. Відстань від спинки до переднього краю сидіння має регулюватися в межах 260-400мм. Для зниження статичного напруження м'язів верхніх кінцівок слід використовувати стаціонарні або змінні підлокітники завдовжки не менше ніж 250мм, завширшки 50-70мм, що регулюються за висотою над сидінням у межах 230-260мм і відстанню між підлокітниками в межах 350-500мм. Монітор має розташовуватися на оптимальній відстані від очей користувача, що становить 600-700мм, але не ближче ніж за 600мм з урахуванням розміру літерно-цифрових знаків і символів. Розташування екрана монітору має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 градусів до нормальної лінії погляду працівника. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю. У конструкції клавіатури має передбачатися опорний пристрій, який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5-15 градусів. Висота середнього рядка клавіш має не перевищувати 30мм. Поверхня клавіатури має бути матовою з коефіцієнтом відбиття 0,4. Розташування пристрою введення – виведення інформації має забезпечувати добру видимість монітору, зручність ручного керування в зоні досяжності моторного поля і за висотою – 900-1300мм, за шириною 400-500мм [47].

Приміщення для роботи з персональними комп'ютерами мають бути обладнані системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури,

відносної вологості й рухливості повітря у відповідності до ГОСТ 12.1.005-88, СН 4088-86, що наведено в табл. 5.1 [47].

Таблиця 5.1 – Значення параметрів мікроклімату у приміщенні

Пора року	Категорія робіт	Температура повітря, град. С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		оптимальна	оптимальна	оптимальна
Холодна	легка-1 а	22 – 24	40 – 60	0,1
	легка-1 б	21 – 23	40 – 60	0,1
Тепла	легка-1 а	23 – 25	40 – 60	0,1
	легка-1 б	22 – 24	40 – 60	0,2

Рівні позитивних і негативних іонів у повітрі мають відповідати санітарно-гігієнічним нормам № 2152-80 які показано в табл. 5.2 [47].

Таблиця 5.2 – Допустимі рівні позитивних і негативних іонів у повітрі

Рівні	Кількість іонів в 1 см куб. повітря	
	n +	n –
Мінімально необхідні	400	600
Оптимальні	1500 – 3000	3000 – 5000
Максимально допустимі	50000	50000

Приміщення, в яких встановлені персональні комп’ютери, повинні мати природне та штучне освітлення відповідно до ДБН В.2.5 – 28 – 2006. Зазначення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк. При цьому світильники місцевого освітлення слід

встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300лк. Необхідно обмежувати відбитий блиск на робочих поверхнях відносно джерел природного і штучного освітлення. При цьому яскравість відблисків на екрані ВДТ має не перевищувати 40 кд/м2, а яскравість стелі в разі застосування системи відбитого освітлення – 200 кд/м [47].

Значення напруженості електростатичного поля на робочих місцях, як у зоні екрана дисплея, так і на поверхнях обладнання, клавіатури, друкувального пристрою, мають не перевищувати гранично допустимих за ГОСТ 12.1.045-84, СН 1757-77. Значення напруженості електромагнітних полів на робочих місцях з ВДТ мають відповідати нормативним значенням (ГДР № 3206-85, ГДР № 4131-86, СН № 5802-91, ГОСТ 12.1.006-84). Інтенсивність потоків інфрачервоного випромінювання має не перевищувати допустимих значень відповідно до СН 4088-86, ГОСТ 12.1.005-88. Інтенсивність потоків ультрафіолетового випромінювання має не перевищувати допустимих значень відповідно до СН 4557-88. Допустимі параметри наведено в табл. 5.1 [47].

Таблиця 5.3 –Допустимі параметри електромагнітних неіонізуючих випромінювань і електростатистичного поля

Допустимі параметри електромагнітних неіонізуючих випромінювань і електростатистичного поля			
Види поля	Допустимі параметри поля		Допустима поверхнева щільність потоку енергії (інтенсивність потоку енергії), Вт/кв.м
	за електричною складовою (Е), В/м	за магнітною складовою (Н), А/м	
1	2	3	4

Продовження таблиці 5.3

1	2	3	4
Напруженість електромагнітного поля 60 кГц до 3 мГц	50	5	
Напруженість електромагнітного поля3 кГц до 30 мГц	20		
Напруженість електромагнітного поля 30 кГц до 50 мГц	10	0,3	
Напруженість електромагнітного поля 30 кГц до 300 мГц 5	5		
Напруженість електромагнітного поля 300 кГц до 300 гГц			10Вт/кв. м
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектруУФ-С (220 – 280 мм)			0,001

Продовження таблиці 5.3

1	2	3	4
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектруУФ-В (280 – 320 мм)			0,01
Електромагнітне поле оптичного діапазону в ультрафіолетовій частині спектруУФ-А (320 – 400 мм)			10,0
Електромагнітне поле оптичного діапазону в видимій частині спектру400 – 760 мм			10,0
Електромагнітне поле оптичного діапазону в інфрачервоній частині спектру0,76 – 10,0 мкм			35,0 – 70,0
Напруженість електричного поля відеодисплейного терміналу			20кВ/м

При організації праці, що пов’язана з використанням персональних

комп’ютерів, для збереження здоров’я працюючих, запобігання професійним захворювання і підтримки працездатності слід передбачити внутрішньозмінні регламентовані перерви для відпочинку. Внутрішньозмінні режими праці і відпочинку мають передбачати додаткові нетривалі перерви в періоди, що передують появі об’єктивних і суб’єктивних ознак стомлення і зниження працездатності. За основну роботу з персональним комп’ютером слід вважати таку, що займає не менше 50% часу впродовж робочої зміни. Тривалість обідньої перерви визначається чинним законодавством про працю і Правилами внутрішнього трудового розпорядку. Встановлюються такі внутрішньозмінні режими праці та відпочинку при роботі з ЕОМ при 8-годинній денній робочій зміні в залежності від характеру праці:

– для розробників програм слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за персональним комп’ютером;
– для операторів персональних комп’ютерів слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;
– для операторів комп’ютерного набора слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин після кожної години роботи за персональним комп’ютером [47].

5.1.2 Загальні обов’язки роботодавців

Роботодавець повинен:

– проінформувати працівників під розписку про умови праці та наявність на їх робочих місцях небезпечних та шкідливих виробничих факторів, які виникають під час роботи з екранними пристроями та ще не усунуто, а також про можливі наслідки їх впливу на здоров’я працівників;

- забезпечити навчання і перевірку знань працівників з питань охорони праці та безпечного використання екранних пристроїв до початку роботи з ними;
- вжити відповідних заходів, щоб забезпечити відповідність робочого місця працівника до цих Вимог;
- під час облаштування робочого місця працівника з екранними пристроями необхідно обирати таке устаткування, яке не створює зайвого шуму та не виділяє надлишкового тепла;
- за рахунок тривалості робочої зміни організувати внутрішні регламентовані перерви для відпочинку;
- забезпечити за свій рахунок проведення медичних оглядів працівників. За результатами цих оглядів роботодавець за потреби повинен забезпечити виконання відповідних оздоровчих заходів;
- за необхідності проводити лабораторні дослідження умов праці працівників з метою виявлення шкідливих і небезпечних факторів виробничого середовища, важкості та напруженості трудового процесу [48].

5.1.3 Вимоги безпеки до робочих місць працівників

Робочі місця працівників з екранними пристроями мають бути

- спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів;
- для забезпечення безпеки та захисту здоров’я працівників усе випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня з погляду безпеки та охорони здоров’я працівників;
- організація робочого місця працівника з екранними пристроями має забезпечувати відповідність усіх елементів робочого місця та їх

розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт;

- освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98;
- мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [56];
- робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування;
- робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння - як по висоті, так і по нахилу. Слід передбачати підніжку для тих, кому це необхідно для зручності [48].

5.1.4 Мінімальні вимоги безпеки під час роботи

Мінімальні вимоги безпеки під час роботи працівників на підприємстві:

- щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень;
- після закінчення роботи екранні пристрої слід відключати від електричної мережі;
- у разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі [48].

Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності [48].

5.1.5 Мінімальні вимоги безпеки до екранних пристроїв

Вимоги до екранних пристроїв наступні:

- екранні пристрої не мають бути джерелом ризику для працівників;
- усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров’я працівників;
- символи на екранних пристроях мають бути чіткими, відповідного розміру. Між символами і рядками символів має бути належна відстань;
- зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності;
- яскравість та/або контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов;
- вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника;
- за необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана;

- екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями;
- вибираючи клавіатуру, слід надавати перевагу такій клавіатурі, яка відкидається і є автономною (відокремленою від екрана), щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук (кисті і верхньої частини руки);
- поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання. Розташування клавіш і самі клавіші мають полегшувати роботу із клавіатурою. Позначення клавіш повинно бути достатньо контрастним і розбірливим;
- устаткування, яке входить до робочої станції, не має виділяти надлишкового тепла, що може спричинити незручності працівникам під час роботи з екранними пристроями;
- під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно - адаптованим до рівня знань і досвіду працівника [48].

5.2 Дії працівників в надзвичайних ситуаціях

Відповідно до Кодексу цивільного захисту України, підготовка персоналу на підприємствах незалежно від форм власності до дій у надзвичайних ситуаціях здійснюється за спеціально розробленою схемою заходів захисту населення та територій. Стаття 130 Кодексу цивільного захисту України передбачає, що на підприємствах розробляються та затверджуються інструкція щодо дій при

загрозі або виникненні надзвичайних ситуацій, деякі положення якої приведені далі [49].

5.2.1 Положення типової інструкції щодо дій персоналу при загрозі або виникненні надзвичайних ситуацій

Додержання протиепідемічних заходів при загрозі розповсюдження небезпечних інфекційних захворювань:

1. якщо на території підприємства виникла небезпека розповсюдження особливо небезпечних інфекційних захворювань, усі працівники повинні суворо виконувати вимоги санітарно-епідеміологічної служби.
2. при необхідності працівники, які прибули на роботу, повинні проходити санітарну обробку , дезінфекцію або міняти одяг.

Особливості дій працівників при деяких надзвичайних ситуаціях:

1. при загрозі хімічного ураження оповіщаються всі працівники та відвідувачі, які знаходяться на території підприємства.
2. при виявленні у приміщенні, де укриваються працівники, хімічно небезпечної речовини працівники повинні залишити зону забруднення, рухаючись в напрямку, перпендикулярному напрямку вітру в засобах індивідуального захисту.
3. при радіоактивному забрудненні території підприємства або при загрозі забруднення всі працівники повинні уважно слідкувати за мовним повідомленням управління з питань надзвичайних ситуацій.
4. при загрозі або виникненні катастрофічних стихійних лих працівник підприємства по розпорядженню адміністрації повинен зупинити виробництво, виконати необхідні протипожежні заходи, відключити від

електромережі електрообладнання, підготуватися до евакуації або вивезення до безпечного місця найбільш цінних матеріальних засобів.

5. при надходженні анонімної інформації про загрозу на території підприємства або поблизу нього терористичного акту працівник, який прийняв її, повинен терміново доповісти керівнику підприємства та до правоохоронних органів і діяти згідно з розпорядженнями та рекомендаціями [49].

5.2.2 Забезпечення електробезпеки на робочому місці

Мінімальні вимоги до безпеки під час роботи з екранними пристроями та використання побутових електроприладів. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності [50].

Заходи щодо виконання вимог електробезпеки офісних працівників регламентують наступні нормативні документи:

- типове положення про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 0.00-4.12-05), затверджене наказом Державного комітету України з нагляду за охороною праці України від 26 січня 2005 р. № 15;

- правила технічної експлуатації електроустановок споживачів, затверджені наказом Міністерства палива та енергетики України від 25 липня 2006 р. № 258 (далі — ПТЕЕС);
- правила безпечної експлуатації електроустановок споживачів, затверджені наказом Міністерства праці та соціальної політики України, Комітету по нагляду за охороною праці від 9 січня 1998 р. № 4;
- правила пожежної безпеки в Україні, затверджені наказом Міністерства внутрішніх справ України від 30 грудня 2014 р. № 1417;
- типове положення про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 0.00-4.12-05), затверджене наказом Державного комітету України з нагляду за охороною праці України від 26 січня 2005 р. № 15 [50, 58].

При ураженні струмом необхідно ізолювати потерпілого від джерела електричного струму шляхом вимкнення щитку, у разі якщо це неможливо, потрібно розірвати контакт людини зі струмом, але ні в якому разі не можна торкатися тіла потерпілого, лише його речей, використовуючи гумові чоботи, рукавиці та предмети, що не проводять електричний струм. Після цього необхідно викликати лікаря.

До прибуття лікаря необхідно:

- якщо потерпілий при свідомості, то його необхідно покласти на підстилку з тканини чи одягу, створити приплив свіжого повітря, розстібнути одяг, що стискає та перешкоджає диханню, розтерти та зігріти тіло і забезпечити спокій до прибуття лікаря;
- потерпілому, що знаходиться в непритомному стані, треба дати понюхати нашатирний спирт або збризнути обличчя холодною водою. Коли

потерпілий прийде до тями, дати йому випити 15-20 крапель настоянки валеріани та гарячого чаю;

- за відсутності ознак життя потрібно негайно розпочати серцево-легеневу реанімацію (СЛР). Попередньо потерпілого необхідно покласти спиною на тверду рівну поверхню;
- у випадку зупинки серця, яку можна визначити за відсутністю у потерпілого пульсу на сонній артерії, необхідно провести непрямий масаж серця;

Штучне дихання та непрямий масаж серця необхідно виконувати доти, поки у потерпілого повністю не відновиться дихання та робота серця або поки не прибуде швидка медична допомога [51].

5.2.3 Забезпечення пожежної безпеки на робочому місці

Правила пожежної безпеки в кабінетах, адміністративних приміщеннях, лабораторіях повинні відповідати основним вимогам будівельних норм. Після закінчення роботи, перед закриттям кабінетів та адміністративних приміщень всі електроустановки та персональні комп'ютери слід вимкнути з мережі електроживлення [52].

На робочих місцях не дозволяється:

- влаштовувати перегородки на шляхах евакуації, а також загромаджувати сходові майданчики різними предметами і обладнанням;
- проводити роботи з ремонту комп'ютерів безпосередньо на робочому місці;
- зберігати постійно на робочому місці дискети, магнітні диски і інші носії інформації, запасні блоки й деталі від комп'ютерів;
- користуватися побутовими електронагрівальними приладами;
- захаращувати евакуаційні виходи та проходи;
- встановлювати на вікнах глухі грати;

- користуватися відкритим вогнем;
- залишати без нагляду ввімкнену в мережу електронну апаратуру, яка використовується для випробування та контролю ЕОМ, комп’ютерів;
- кошики та ящики для паперу повинні регулярно спорожнятися, а сміття виноситися за межі будівлі в спеціально відведені місця [52].

Відповідно до Кодексу цивільного захисту України та Правил пожежної безпеки в Україні, затверджених наказом Міністерства внутрішніх справ України 30.12.2014 № 1417, зареєстрованих в Міністерстві юстиції України 05 березня 2015 р. за № 252/26697, та на виконання доручення Прем'єр-міністра України О. Гончарука від 14.01.2020 № 47730/1/1-19 рекомендації щодо дій у разі виникнення пожежі (надзвичайної ситуації) наступні: необхідно заздалегідь знати де і які засоби пожежогасіння, зв'язку розміщені і як ними користуватися. В жодному разі не слід панікувати. Якщо пожежа застала у середині приміщенні, слід дотримуватись таких правил:

- не панікувати і перш за все, викликати рятувальну службу за телефоном № 101;
- до дверей приміщення рухатися швидко в залежності від рівня загрози і відстані до виходу;
- якщо двері не гарячі, то відчиняти їх та швидко виходити;
- якщо двері гарячі від джерела загоряння із зовнішнього боку, їх не можна відчиняти, а навпаки щільно закрити, а всі щілини та отвори заткнути тканиною. Повернутись повзком у глибину приміщення і вжити належних заходів, викликаючи рятувальників, голосом, світловими, шумовими сигналами тощо;
- розкрити вікно або розбити скло твердим предметом та поклилкати на допомогу;

- не використовувати ліфти під час пожежі;
- у висотному будинку з осередком загоряння на нижчому поверсі, не можна бігти до низу крізь вогнище, врятуватися можна на даху будівлі;
- долайти межу вогню проти вітру (протягу), закривши голову і обличчя одягом;
- якщо евакуюватися з місця пожежі неможливо, слід прикрити рот і ніс тканиною та дихати повітрям, що знаходиться ближче до підлоги – повітря там менш задимлене [53, 54, 55].

Висновки до розділу 5

В даному розділі було розглянуто питання охорони праці та безпеки праці робітників підприємства та їх дії в разі настання надзвичайних ситуацій відповідно до існуючих нормативних актів. Під час робіт з розробки програмного забезпечення на ЕОМ було наведено шкідливі виробничі фактори які впливають на розробників програмних систем та проблеми зі здоров'ям до яких ці фактори можуть призвести.

Також було наведено:

- загальні обов'язки роботодавців;
- вимоги до робочих місць працівників;
- мінімальні вимоги безпеки під час роботи;
- мінімальні вимоги безпеки до екранних пристроїв;
- дії працівників в надзвичайних ситуаціях;
- положення інструкції щодо дій персоналу при загрозі виникнення або настання надзвичайних ситуацій;
- питання електробезпеки на робочому місці;
- питання пожежної безпеки на робочому місці.

ЗАГАЛЬНІ ВИСНОВКИ ЗА РОБОТОЮ

В процесі виконання дипломної роботи проводилося дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS. Для розуміння сучасного стану питання, встановлення наукової новизни та обґрунтування необхідності дослідження було проведено аналіз наявних літературних джерел та виявлено їх плюси та мінуси. Як виявилось проаналізовані літературні аналоги не розкривають досліджуване питання повною мірою.

Окрім вивчення стану літературних джерел було проаналізовано програмні аналоги, що дозволило виявити недоліки в існуючому програмному забезпеченні та допомогло зрозуміти аспекти які вимагають удосконалення. Переважна більшість статичних аналізаторів досліджує лише помилки коду, а не архітектури, а ті що стосуються аналізу реалізації патернів мають різні нюанси, які треба покращувати.

Для покращення існуючих програмних засобів було спроектовано та реалізовано статичний аналізатор, який орієнтовано на опрацювання архітектури в програмних засобах різного розміру та типу. Для створення програмного продукту було використано лише необхідні шаблони та підходи проектування, що дозволило реалізувати необхідний функціонал, не ускладнити систему та гарантувати легку підтримку, тестування та розширення функціоналу у разі необхідності. Інтерфейс користувача було спроектовано та реалізовано так, щоб в процесі використання програми все було інтуїтивно зрозуміло. Тому кількість використаних елементів мінімальна, але достатня щоб відповідати поставленим задачам.

Завдяки розробленому програмному забезпеченню було проведено декілька експериментів, що допомогли провести більш детальне дослідження за обраною темою. В результаті досліджень було виявлено, що

не лише нехтування патернами та загальноприйнятими підходами проектування, але й надмірне їх використання ускладнює розробку, розширення, підтримку та тестування програмних засобів. Накладає додаткове навантаження на мозок розробників, що призводить до помилок в коді. Все це в сукупності призводить до погіршення архітектури проекту в цілому та збільшує його вартість, що може бути досить критичним для певної компанії. Слід пам’ятати, що архітектура проекту залежить не тільки від вміння використовувати різні шаблони, але й від навичок прогнозування росту продукту, аналізу поточного стану архітектури, бюджету проекту, цілей на подальший розвиток та вплив зміни розміру продукту на його архітектуру.

Список використаних джерел

1. Design Patterns [Електронний ресурс] / Design Patterns – Режим доступу: URL: https://sourcemaking.com/design_patterns [Дата звертання: 10.11.2021].
2. Minimum viable product [Електронний ресурс] / WikipediA – Режим доступу: URL: https://en.wikipedia.org/wiki/Minimum_viable_product [Дата звертання: 10.11.2021].
3. MVP [Електронний ресурс] / DEVELOPER EXPERIENCE.io – Режим доступу: URL: https://developerexperience.io/practices/minimum-viable-product [Дата звертання: 10.11.2021].
4. What is Minimum Viable Product And How To Make It Right [Електронний ресурс] / Medium – Режим доступу: URL: https://medium.com/anoda-mobile-development-agency/what-is-minimum-viable-product-and-how-to-make-it-right-8b885001c6f5 [Дата звертання: 10.11.2021].
5. Определение паттерна MVVM [Електронний ресурс] / METANIT.COM – Режим доступу: URL: https://metanit.com/sharp/wpf/22.1.php [Дата звертання: 10.11.2021].
6. Model View ViewModel MVVM Android Example [Електронний ресурс] / ZEFTINO.COM – Режим доступу: URL: https://www.zoftino.com/model-view-viewmodel-mvvm-android-example [Дата звертання: 10.11.2021].
7. Инверсия управления [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B5%D1

%80%D1%81%D0%B8%D1%8F_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F [Дата звертання: 10.11.2021].

8. Inversion of Control vs Dependency Injection [Електронний ресурс] / stack overflow – Режим доступу: URL: https://stackoverflow.com/questions/6550700/inversion-of-control-vs-dependency-injection [Дата звертання: 10.11.2021].
9. IoC, DI, IoC-контейнер — Просто о простом [Електронний ресурс] / Хабр – Режим доступу: URL: https://habr.com/ru/post/131993/
10. Фабричный метод (шаблон проектирования) [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B1%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Дата звертання: 10.11.2021].
11. SOLID (об'єктно-орієнтоване програмування) [Електронний ресурс] / ВікіпедіЯ – Режим доступу: URL: https://uk.wikipedia.org/wiki/SOLID_(%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [Дата звертання: 10.11.2021].
12. The Importance of SOLID Design Principles [Електронний ресурс] / BMC Blogs – Режим доступу: URL:

https://www.bmc.com/blogs/solid-design-principles/# [Дата звертання: 10.11.2021].

13. KISS (принцип) [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/KISS_(%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF) [Дата звертання: 10.11.2021].
14. Блог-платформа Medium: Подробное руководство по использованию для маркетологов [Електронний ресурс] / vc.ru – Режим доступу: URL: https://vc.ru/flood/7935-medium-how-to [Дата звертання: 10.11.2021].
15. Stack Overflow [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/Stack_Overflow [Дата звертання: 10.11.2021].
16. Хабр [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%B1%D1%80 [Дата звертання: 10.11.2021].
17. Библиотека MSDN [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_MSDN [Дата звертання: 10.11.2021].
18. Статический анализ кода [Електронний ресурс] / ВикипедиЯ – Режим доступу: URL: https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%BA%D0%BE%D0%B4%D0%B0 [Дата звертання: 10.11.2021].

19. Klocwork: Best Static Code Analyzer for Developer Productivity, SAST, and DevOps/DevSecOps [Електронний ресурс] / PERFORCE – Режим доступу: URL: https://www.perforce.com/products/klocwork?utm_leadsource=cpc-googleadwords&utm_source=googleadwords&utm_medium=cpc&utm_campaign=KlocworkEMEABrand&utm_adgroup=BrandTerms&gclid=CjwKCAiAqIKNBhAIEiwAu_ZLDlS4SCJQp8qYsyJPHoqAn_Tf3hP4Lt3L0CtBWtMy-nw0Zi12uB_fChoCdD4QAvD_BwE [Дата звертання: 10.11.2021].
20. Getting started with Klocwork Desktop Plug-in for Visual Studio [Електронний ресурс] / RogueWave – Режим доступу: URL: https://docs.roguewave.com/en/klocwork/2016/gettingstartedwithklocworkdesktoppluginforvisualstudio#concept750 [Дата звертання: 10.11.2021].
21. Getting started with Klocwork Desktop for C/C++ [Електронний ресурс] / RogueWave – Режим доступу: URL: https://docs.roguewave.com/en/klocwork/current/gettingstartedwithklocworkdesktopforcc [Дата звертання: 10.11.2021].
22. Default reports in Klocwork Static Code Analysis [Електронний ресурс] / RogueWave – Режим доступу: URL: https://docs.roguewave.com/en/klocwork/current/defaultreportsinklocworksca
23. Top 10 Static Code Analysis Tool | Best Static Code Analysis Tools List [Електронний ресурс] / scmGalaxy – Режим доступу: URL: https://www.scmgalaxy.com/tutorials/top-10-static-code-analysis-tool/ [Дата звертання: 10.11.2021].
24. VisualCodeGrepper V2.2.0 [Електронний ресурс] / SOURCEFORGE – Режим доступу: URL: https://sourceforge.net/projects/visualcodegrepp/ [Дата звертання: 10.11.2021].

25. VisualCodeGrepper - Code security scanning tool [Електронний ресурс] / GitHub – Режим доступу: URL: https://github.com/nccgroup/VCG [Дата звертання: 10.11.2021].
26. NDepend [Електронний ресурс] / WikipediA – Режим доступу: URL: https://en.wikipedia.org/wiki/NDepend [Дата звертання: 10.11.2021].
27. NDepend Screenshots [Електронний ресурс] / ndepend – Режим доступу: URL: https://www.ndepend.com/screenshots#Poster [Дата звертання: 10.11.2021].
28. Sonargraph Overview [Електронний ресурс] / HELLO2MORROW – Режим доступу: URL: https://www.hello2morrow.com/products/sonargraph [Дата звертання: 10.11.2021].
29. Sonargraph-Architect [Електронний ресурс] / HELLO2MORROW – Режим доступу: URL: https://www.hello2morrow.com/products/sonargraph/architect9 [Дата звертання: 10.11.2021].
30. Sonargraph-Explorer (free license for any use) [Електронний ресурс] / HELLO2MORROW – Режим доступу: URL: https://www.hello2morrow.com/products/sonargraph/explorer [Дата звертання: 10.11.2021].
31. Концептуальные основы построения анализатора безопасности программного кода [Електронний ресурс] / ПРОГРАММНЫЕ ПРОДУКТЫ И СИСТЕМЫ – Режим доступу: URL: http://www.swsys.ru/index.php?page=article&id=3379 [Дата звертання: 10.11.2021].
32. Руководство Microsoft по проектированию архитектуры приложений [Текст] / J.D. Meier, David Hill, Alex Homer, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr., Akshay Bogawat. — 2-е изд. —

2009. — 63 - 64 с. – Режим доступу: URL: http://www.dut.edu.ua/uploads/l_1507_99407341.pdf [Дата звертання: 10.11.2021].

33. Software Architecture Analysis Method (SAAM) [Електронний ресурс] / DZone – Режим доступу: URL: https://dzone.com/articles/software-architecture-analysis [Дата звертання: 10.11.2021].
34. Software Architecture Analysis Method (SAAM) [Електронний ресурс] – Режим доступу: URL: http://www.peter-lo.com/Teaching/U08182/L07A.pdf [Дата звертання: 10.11.2021].
35. Architecture tradeoff analysis method [Електронний ресурс] / WikipediA – Режим доступу: URL: https://en.wikipedia.org/wiki/Architecture_tradeoff_analysis_method [Дата звертання: 10.11.2021].
36. Architecture analysis: – The SAAM – ATAM [Електронний ресурс] / WikipediA – Режим доступу: URL: https://www.softwareresearch.net/fileadmin/src/docs/teaching/SS02/SWA/SWAslides3.pdf [Дата звертання: 10.11.2021].
37. Architecture tradeoff analysis method (ATAM) [Електронний ресурс] / CONCISE SOFTWARE – Режим доступу: URL: https://concisesoftware.com/architecture-tradeoff-analysis-method-atam/ [Дата звертання: 10.11.2021].
38. The CBAM (Cost Benefit Analysis Method) [Електронний ресурс] – Режим доступу: URL: http://ce.sharif.ac.ir/courses/85-86/2/ce924/resources/root/Presentations/Chap%2012.pdf [Дата звертання: 10.11.2021].

39. Xamarin.Forms XAML Basics [Електронний ресурс] / Microsoft – Режим доступу: URL: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/ [Дата звертання: 10.11.2021].
40. Язык программирования C#: краткая история, возможности и перспективы [Електронний ресурс] / timeweb – Режим доступу: URL: https://timeweb.com/ru/community/articles/chto-takoe-csharp [Дата звертання: 10.11.2021].
41. What is Xamarin.Forms? [Електронний ресурс] / Microsoft – Режим доступу: URL: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms#how-xamarinforms-works [Дата звертання: 10.11.2021].
42. Microsoft Visual Studio [Електронний ресурс] / WikipediA – Режим доступу: URL: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio [Дата звертання: 10.11.2021].
43. Сложность программной системы [Електронний ресурс] / StudRef – Режим доступу: URL: https://studref.com/316134/informatika/slozhnost_programmnoy_sistemy [Дата звертання: 10.11.2021].
44. Насколько плох SLOC (исходные строки кода) как метрика? [Електронний ресурс] / CodeRoad – Режим доступу: URL: https://coderoad.ru/3769716/%D0%9D%D0%B0%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%BE-%D0%BF%D0%BB%D0%BE%D1%85-SLOC-%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B5-%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8-%D0%BA%D0%BE%D0%B4%D0%B0-%D0%BA%D0%B0

%D0%BA-%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0 [Дата звертання: 10.11.2021].

45. Наказ № 207 Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями від 14.02.2018;
46. Шкідливі та небезпечні виробничі чинники при роботі з комп’ютерною технікою [Електронний ресурс] / Національний університет Львівська політехніка. StudFiles – Режим доступу: URL: https://studfile.net/preview/5211197/page:3/ [Дата звертання: 10.11.2021].
47. Охорона праці в офісі. Вимоги до робочого місця офісного працівника [Електронний ресурс] / GCC – Режим доступу: URL: https://gc.ua/uk/oxorona-praci-v-ofisi-vimogi-do-robochogo-miscya-ofisnogo-pracivnika/ [Дата звертання: 10.11.2021].
48. Вступили в силу нові вимоги до роботи за комп’ютером [Електронний ресурс] / medoc – Режим доступу: URL: https://medoc.ua/blog/vstupili-v-silu-novi-vimogi-do-roboti-za-kompjuterom [Дата звертання: 10.11.2021].
49. Як діяти персоналу підприємства в надзвичайній ситуації [Електронний ресурс] / Головне управління Держпраці у Львівській області – Режим доступу: URL: https://lviv.dsp.gov.ua/pytannia-vidpovidi/iak-diiaty-personalu-pidpryiemstva-v-nad/ [Дата звертання: 10.11.2021].
50. Про заходи з електробезпеки для працівників офісу [Електронний ресурс] / Охорона праці – Режим доступу: URL: https://oppb.com.ua/articles/pro-zahody-z-elektrobezpeky-dlya-pracivnykiv-ofisu [Дата звертання: 10.11.2021].

51. Електробезпека [Електронний ресурс] / Київський національний торговельно-економічний університет. Житомирський торговельно-економічний коледж – Режим доступу: URL: http://www.ztec.com.ua/ztec/e-lib/%D0%9E%D1%85%D0%BE%D1%80%D0%BE%D0%BD%D0%B0%20%D0%BF%D1%80%D0%B0%D1%86%D1%96/%D0%A2%D0%B5%D0%BC%D0%B0%2017%20%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0.pdf [Дата звертання: 10.11.2021].
52. Заходи пожежної безпеки на робочому місці [Електронний ресурс] / Одеський державний екологічний університет. StudFiles – Режим доступу: URL: https://studfile.net/preview/5163303/page:2/ [Дата звертання: 10.11.2021].
53. Кодекс цивільного захисту України Документ 5403-VI, чинний, поточна редакція — Редакція від 10.06.2021, підстава - 1217-IX;
54. Наказ № 1417 Про затвердження Правил пожежної безпеки в Україні від 30.12.2014;
55. Рекомендації щодо дій з правил пожежної безпеки та порядку дій в разі виникнення пожежі / ДЕРЖАВНА СЛУЖБА МОРСЬКОГО ТА РІЧКОВОГО ТРАНСПОРТУ УКРАЇНИ – Режим доступу: URL: https://marad.gov.ua/ua/gromadskosti/bezpeka-zhittyediyalnosti/rekomendaciyi-shchodo-dij-z-pravil-pozhezhnoyi-bezpeki-ta-poryadku-dij-v-razi-viniknennya-pozhezhi [Дата звертання: 10.11.2021].
56. Постанова № 42 від 01.12.99 Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99;
57. Закон України «Про охорону праці» № 2695-XII від 14.10.92;

58. Наказ від 26.01.2005 № 15 Про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці та Переліку робіт з підвищеною небезпекою;

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис БОДНАР

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01215-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

_____

Керівник розробки
Олександра ГОРБОВА

_____

Виконавець
Олександр СИРОТА

_____

Нормоконтролер
Олена КУРОП’ЯТНИК

_____

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01215-01

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технічне завдання

Аркушів 23

2021

АНОТАЦІЯ

Документ 1116130.01215-01 «Статичний аналізатор архітектури крос-платформного програмного забезпечення. Технічне завдання» є однією з частин програмної документації до системи.

У даному документі описано призначення системи, область її застосування, основні вимоги, стадії та строки розробки проекту, технічні та техніко-економічні показники.

ЗМІСТ

ВСТУП

Для дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS було розроблено статичний аналізатор, який в процесі аналізу проекту робить акцент на якості створеної архітектури. Завдяки реалізованому алгоритму аналізатор під час виконання перевіряє наявність того, чи іншого патерну шляхом перевірки правил його реалізації, а також надає додаткову інформацію, щодо посилань на модулі, її кількості та інш.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ № 690 ст. від 18.11.2020 ректора Дніпровського національного університету залізничного транспорту імені академіка В. Лазаряна «Про затвердження керівників та затвердження тем магістерських робіт».

Тема дипломної роботи: Дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS.

Керівник дипломної роботи– Горбова О.В.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення розробки

У якості функціонального призначення системи виступає статичний аналіз архітектури крос-платформного програмного забезпечення.

2.2 Експлуатаційне призначення розробки

Експлуатаційним призначенням розробки програми є можливість її використання при аналізі архітектури проектів різної складності, розміру та бюджету. Це надає змогу перевірити якість сконструйованої системи незалежно від її характеристик, наприклад розміру.

3 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вимоги до функціональних характеристик системи

Вимоги до функціональних характеристик системи наступні:

- перегляд завантажених файлів проекту, архітектура якого буде проаналізована;
- може бути здійснений перегляд контенту завантажених файлів проекту для аналізу, а його форматування має відповідати оригінальному форматуванню, заданому при розробці в певному середовищі для розробки програмних засобів;
- налаштування аналізатору коду повинні дозволяти налаштувати програмне середовище під проекти різного розміру, вартості та часу розробки, а також з необхідною мовою та фреймворком, що були використані при реалізації проекту, що аналізується;
- результати аналізу архітектури програмного забезпечення мають бути представлені за допомогою сконструйованого графічного інтерфейсу користувача та збережені у результуючі файли;
- результуючі файли мають містити контент файлів, що був проаналізований та результати аналізу архітектури.

В якості вхідних даних для роботи повинні використовуватись файли з вихідним кодом різного типу, залежно від використаної мови програмування.

В якості вихідних даних виступають візуальне представлення результатів аналізу, а також текстові файли з цими результатами та назвами і вихідним кодом, що було оброблено аналізатором.

3.2 Вимоги до надійності програмної системи

Нижче представлено список сформованих вимог до надійності програмної системи:

- вхідні дані, тобто файли вихідного коду проетку не повинні змінюватись. Вони використовуються лише для аналізу і їх зміни заборонені;
- програма мусить контролювати процес виділення та вивільнення пам’яті;
- програмна система мусить не допускати до аналізу та перегляду файлів, доки проект не було завантажено;
- якість системи повинна бути така, що запустивши її 2000 разів значення максимальної кількість відмов буде рівне 1.

3.3 Умови експлуатації програмного забезпечення

Для роботи з програмним забезпеченням повинні дотримуватись вимоги, що описані в документах [1, 2]. Окрім них повинні виконуватись наступні вимоги:

- користувач повинен вивчити такі документи як керівництво користувача та опис програми;
- користувач програмного продукту повинен мати мінімальний досвід роботи з програмним забезпеченням такого типу;
- персональний комп’ютер, що використовуються для роботи з статичним аналізатором повинен відповідати встановленим вимогам;
- мікроклімат в робочих приміщеннях має задовольняти санітарним нормам мікроклімату виробничих приміщень, які наводяться в документі [3];

3.4 Вимоги до параметрів технічних засобів

Для того щоб програмане забезпечення функціонувало на персональному комп’ютері він повинен відповідати наступним вимогам:

- процессор: Intel Core i5-3210M, 2 ядра, 2.5 ГГц;
- оперативна пам'ять: 4 GB DDR2;
- вільний дисковий простір 100 Mb на накопичувачі;
- роз’єм USB для завантаження додаткового програмного забезпечення;

- операційна система: macOS Mojave - Monterey;
- монітор роздільна здатність якого не менше ніж 1024x768;
- клавіатура та «миша».

3.5 Вимоги до програмної сумісності

При конструюванні програмного забезпечення необхідно використовувати об’єктно орієнтовану мову програмування C#, мову розробки графічних інтерфейсів Xaml, крос-платформний фреймворк Xamarin.Forms та інтегроване середовище розробки Visual Studio.

3.6 Вимоги до маркування

В випадку релізу та поширення програмного продукту в електронному форматі сам продукт та відповідна документація повинні мати відповідне маркування, що показано на рис. 3.1.

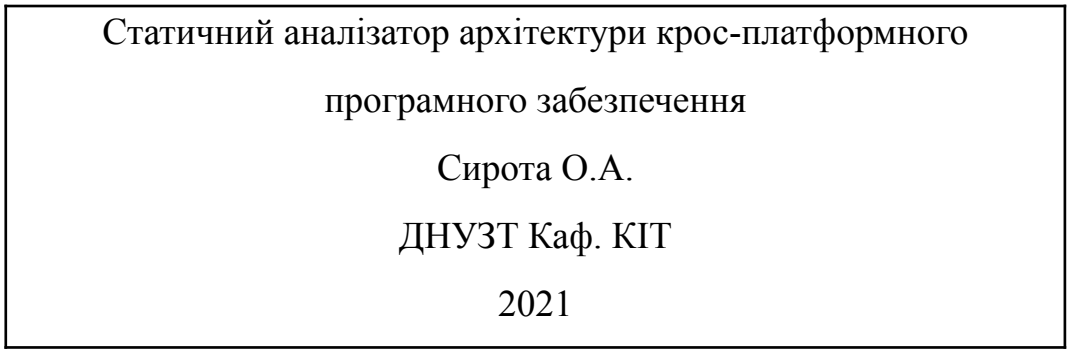


Рисунок 3.1 – Маркування програмного продукту

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація має відповідати вимогам державного стандарту до оформлення програмних документів ГОСТ 19.101-77 «Єдина система програмної документації. Види програм та програмних документів» [3] та складатися з таких документів як:

- специфікація;
- опис програми;
- керівництво користувача. Керівництво з використання статичного аналізатору архітектури крос-платформного забезпечення.
- текст програми.

5 ТЕХНІКО-ЕКОНОМІЧНІ ПОКАЗНИКИ

При розробці будь-якого програмного забезпечення необхідно оцінити можливий майбутній прибуток з проекту, його корисність та прорахувати передбачувані витрати, що в результаті дасть повну економічну картину. Для досягнення цих цілей існує розрахунок, що називається техніко-економічним обґрунтуванням (ТЕО).

Перш за все необхідно оцінити трудові витрати розробників оцінивши розмір програмного продукту, що планується до розробки. Методики які використовуються для оцінки трудовитрат полягають в використаному типі критерію: кількісний або якісний.

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях.

$$E = a \cdot S^b \cdot EAF \tag{1}$$

де E – витрати праці на проект в людино-місяцях;

S^b – розмір коду в KLOC;

EAF – фактор уточнення витрат (англ. *Effort adjustment factor*);

Для простих систем, $a = 2{,}4$; $b = 1{,}05$

Розмір програмного коду розробленого програмного засобу становить 2838 рядків:

$$E = 2{,}4 \cdot 2{,}838^{1{,}05} \cdot 1 = 7 \tag{2}$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складають приблизно 6,8 людино-місяців.

Нижче наведені розрахунки вартості розробки «Статичного аналізатору архітектури крос-платформних додатків». Основними витрати припадають на:

– основна заробітна плата (ОЗП);

- відрахування на соціальні потреби;
- накладні витрати;
- витрати на базовий персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата використовується для оцінки праці інженера програміста зі створення програмного продукту. Для її визначення необхідно розуміти загальну кількість розробників, що потрібно залучити до розробки, часу виконання в годинах та заробітної плати одного розробника за одну годину роботи.

Оскільки програмний додаток було розроблено використовуючи мову програмування C#, то при розрахунках буде використана заробітня плата C# розробника. Згідно статистики сайту з пошуку роботи - Work.ua, середня заробітна плата C# розробника варіюється в межах 18 000 грн [4], але при розрахунках буде використана плата в сумі 15 000 грн, виходячи з початкового досвіду розробника. Розрахунок заробітної плати проводиться по формі табл. 5.1.

Таблиця 5.1 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	15000	1	6,8	102 000

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 05.04.21 до 05.10.21, що складає 154 дні або 31 робочий тиждень. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка розробника складає 93 грн/год.

Витрачений робочий час розраховується за наступною формулою:

$$t_{\text{розробки}} = N_{\text{чол}} \cdot N_{\text{тиж}} \cdot N_{\text{год}}, \quad (3)$$

де $N_{\text{чол}}$ – кількість виконавців, *чол*;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, год;

$$t_{\text{розробки}} = 1 \cdot 31 \cdot 40 = 1240 \text{ чол/год.} \quad (4)$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5)$$

Де $t_{\text{розробки}}$ – витрати праці у чол/год;

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, приймається 0,75.

ОЗП складає:

$$\text{ОЗП} = 1240 \cdot 93 \cdot 0,75 = 86490\text{грн.} \quad (6)$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати:

$$C_{\text{соц}} = \frac{\text{ОЗП} \cdot 22\%}{100\%}$$

$$C_{\text{соц}} = \frac{86490 \cdot 22\%}{100\%} = 19027\text{грн.} \quad (7)$$

Для визначення основних прямих витрат необхідно підсумувати отримані результати розрахунків (6) та (7), отримуємо 105517 грн.

Накладні витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в відсотках (30 – 40%) від суми прямих витрат:

$$\text{С}_{\text{накл}} = \frac{(\text{ОЗП} + \text{С}_{\text{соц}}) \cdot 35\%}{100\%}; \quad (8)$$

$$\text{С}_{\text{накл}} = \frac{(86490 + 19027) \cdot 35\%}{100\%} = 36930\text{грн.} \quad (9)$$

На протязі усього терміну використання техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на комп’ютер залежать від його вартості та розраховуються протягом терміну розробки програмного продукту До цих витрат можна віднести:

- заробітна плата ремонтника;
- амортизаційні витрати на персональний комп'ютер і програмне забезпечення;
- витрати на ремонт;
- вартість витратних матеріалів;
- оренда приміщення;
- додаткові витрати, такі як: прибирання приміщення, охорона, оренда, комунальні послуги;
- витрати на електроенергію ($С_{ел}$),

Витрати на електроенергію визначаються за формулою:

$$С_{ел} = P \cdot B \cdot T_{розр}, \quad (10)$$

де Р – потужність комп’ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/година, складає 1,68 грн [5];

$T_{розр}$ – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{ел} = 0,45 \cdot 1,68 \cdot 1240 = 937 грн. \quad (11)$$

Для розробки програмного забезпечення достатньо комп’ютеру базової комплектації - об’єм ОЗУ 8 - 16GB, SSD накопичувач об’ємом 256 GB, двух’ядровий процессор з частотою процесору 2.2 Гц. Після аналізу цін на сайті OLX за середню вартість комп’ютеру вважається 18000 грн. [6], а термін експлуатації – 5 років. Витрати на витратні матеріали ($С_{вм}$) протягом всього терміну експлуатації приблизно 10% від вартості комп’ютеру і для їх визначення використовується наступна формула:

$$\mathrm{С}_{\text{вм}} = \mathrm{В}_{\text{ком}} \cdot \frac{N_{\text{Д}}}{N_{\text{експ}} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (12)$$

де $B_{КОМ}$ – вартість комп'ютеру;

$N_{\text{Д}}$ – кількість днів розробки програмного продукту;

$N_{\text{експ}}$ – термін експлуатації комп'ютеру.

Витрати на витратні матеріали визначаються наступним чином:

$$C_{\text{вм}} = 18000 \cdot \frac{154}{5 \cdot 365} \cdot \frac{10}{100} = 144 \text{ грн.} \quad (13)$$

Заробітна плата ремонтника ($C_{\text{рем}}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік, середньомісячна заробітна плата якого приймається за 18000 грн, відповідно до статистики заробітних плат сайту Work.ua [7]. Тоді відносно одного комп'ютеру, його заробітна плата дорівнює:

$$C_{\text{рем}} = \frac{C_{\text{рем}}'}{N_{КОМ}} \cdot T_{\text{міс}}, \quad (14)$$

де $C_{\text{рем}}'$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{мес}$ – час розробки програмного продукту, міс.

Заробітна плата ремонтника ($C_{\text{рем}}$) буде складати:

$$C_{\text{рем}} = \frac{18000}{50} \cdot 7 = 2520 \text{ грн.}$$

За статистикою витрати на комплектуючі вироби ($\mathrm{С}_{КОМ}$) для ремонту персонального комп'ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{КОМ} = C_{ВМ} = 144 \text{ грн.} \quad (16)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального

старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$\text{АКП} = \text{В}_{КОМ} \cdot \frac{\text{Тміс}}{N_{\text{експ}} \cdot 365}; \quad (17)$$

$$\text{АКП} = 18000 \cdot \frac{7}{3 \cdot 12} = 3600$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для macOS 5 років та Visual Studio за 1 рік, то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Розробка програмного забезпечення відбувалась на персональному комп'ютері за підтримки операційної системи macOS Catalina, для написання програмного забезпечення використовувалось програмне середовище Visual Studio for Mac.

Виходячи з того, що операційна система macOS Catalina постачається безкоштовно, бо вона входить в вартість персонального комп'ютеру, то:

$$\text{АКП}_w = 0 \quad (18)$$

Згідно цін вказаних на офіційному сайті з продажу Visual Studio, вартість Visual Studio підписки типу Enterprise коштує 250 $/міс [8]. Курс долару НБУ на момент розробки програмного забезпечення становить 27.10 грн. [9] Отже загальна вартість Visual Studio Enterprise наступна:

$$250 \cdot 7 \cdot 27.10 = 47425 \text{ грн.} \quad (19)$$

$$\text{АКП}_w = 47425 \cdot \frac{7}{12} = 28455.$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2.

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, *грн*	Джерело придбання	Амортизаційні відрахування, *грн*
macOS Catalina	Поставляється безкоштовно з комп'ютером	—	0
Visual Studio (підписка Enterprise)	47425	https://visualstudio.microsoft.com/ru/vs/pricing/?tab=business	28455
Всього:	47425	—	28455

Додаткові витрати ($C_{\text{дод}}$), такі як прибирання приміщень, охорона, комунальні послуги важко оцінити точно, тому прийнято рівними 50% заробітної плати інженера-програміста, тобто 7500 гривень на місяць.

За даними сайту r24.ua вартість оренди офісного приміщення площею в 40 кв.м. складає 6000 - 12000 грн. [10], тому вартість оренди приймемо рівною 6000 гривень на місяць.

Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (20)$$

$$C_{\text{експ}} = 937 + 144 + 2520 + 144 + \\ + 3600 + 28455 + 42000 + 7500 = 85300 \text{ грн.} \quad (21)$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	937
Вартість витратних матеріалів	144
Витрати на ремонт	2664
Амортизація персонального комп’ютера	3600
Амортизація програмного забезпечення	28455
Оренда приміщення	42000
Додаткові витрати	7500
Всього	85300

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (22)$$

$$C_{\text{розробки}} = 86490 + 19027 + 36930 + 85300 = 227747 \text{ грн} \quad (23)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	86490
Відрахування на соціальні потреби	19027
Накладні витрати	36930
Експлуатаційні витрати	85300
Всього	227747

За отриманими значеннями техніко-економічних показників проекту складено

кошторис витрат на розробку сучасного програмного забезпечення для оцінки схожості програм. За результатами розрахунків, приблизна вартість розробки складає 227747 грн.

6 СТАДІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Всі стадії розробки програмного продукту показано в табл. 6.1.

Таблиця 6.1 – Стадії розробки програмного забезпечення

Стадії	Зміст робіт	Терміни виконання
1	2	3
1. Технічне завдання	Постановка завдання Збір початкових матеріалів Вибір і обґрунтування критеріїв ефективності і якості програми, що розробляється Обґрунтування необхідності проведення науково-дослідницьких робіт Визначення структури вхідних і вихідних даних Попередній вибір методів рішення задач Обґрунтування доцільності застосування раніше розроблених програм Визначення вимог до технічних засобів Обґрунтування принципової можливості рішення поставленої задачі Визначення вимог до програми Техніко-економічне обґрунтування розробки програми Визначення стадій, етапів і термінів розробки програми і документації на неї Вибір мов програмування Визначення необхідності проведення науково-дослідницьких робіт на подальших стадіях Узгодження і затвердження технічного завдання	28.01.2021 – 15.03.2021

Продовження таблиці 6.1

1	2	3
2. Робочий проект	Програмування і налагодження програми Розробка програмних документів відповідно до вимог ГОСТ 19.101-77 Розробка, узгодження і затвердження програми і методики випробувань Проведення випробувань Коригування програми і програмної документації за наслідками випробувань	16.03.2021 – 28.05.2021
3. Впровадження	Підготовка і передача програми і програмної документації для супроводу і (або) виготовлення	29.05.2021 – 15.06.2021

7 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контролем виконання роботи займається керівник розробки, а прийом програмного продукту проводиться керівником розробки разом з прийомною комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Наказ № 207 Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями від 14.02.2018.
2. Охорона праці в офісі. Вимоги до робочого місця офісного працівника [Електронний ресурс] / GCC – Режим доступу: URL: https://gc.ua/uk/oxorona-praci-v-ofisi-vimogi-do-robochogo-miscya-ofisnogo-pracivnika/ [Дата звертання: 10.07.2021].
3. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст]/ Постанова Головного Державного санітарного лікаря України від 01.12.1999 № 42 – К., 1999.
4. C# программист: средняя зарплата в Украине [Електронний ресурс] / WORK.ua – Режим доступу: URL: https://www.work.ua/ru/salary-C%23+%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82/?count=by-resumes [Дата звертання: 10.07.2021].
5. Увага! Інформація про ціну на електричну енергію для населення з 1 вересня та з 1 жовтня 2021р. [Електронний ресурс] / ЗАПОРІЖЖЯЕЛЕКТРОПОСТАЧАННЯ – Режим доступу: URL: https://zpep.com.ua/2021/08/26/%D1%83%D0%B2%D0%B0%D0%B3%D0%B0-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%8F-%D0%BF%D1%80%D0%BE-%D1%86%D1%96%D0%BD%D1%83-%D0%BD%D0%B0-%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87/ [Дата звертання: 10.07.2021].
6. Вартість персонального комп’ютеру [Електронний ресурс] / OLX – Режим доступу: URL: https://www.olx.ua/elektronika/noutbuki-i-aksesuary/noutbuki/apple/q-macbook-pro-%D0%B1%D1%83/?search%5Bfilter_float_price%3Afrom%5D=16000&search

%5Bfilter_float_price%3Ato%5D=20000&search%5Bfilter_enum_display_size%5D%5B0%5D=3 [Дата звертання: 10.07.2021].

7. Системотехник: средняя зарплата в Украине [Електронний ресурс] / WORK.ua – Режим доступу: URL: https://www.work.ua/ru/salary-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA/ [Дата звертання: 10.07.2021].
8. Купить Visual Studio [Електронний ресурс] / Micrisoft – Режим доступу: URL: https://visualstudio.microsoft.com/ru/vs/pricing/?tab=business [Дата звертання: 10.7.2021].
9. Курс долар НБУ [Електронний ресурс] / Мінфін – Режим доступу: URL: https://minfin.com.ua/ua/currency/nbu/usd/ [Дата звертання: 10.07.2021].
10. Ціни на оренду офісного приміщення [Електронний ресурс] / R24 – Режим доступу: URL: https://r24.ua/%D1%81%D0%BD%D1%8F%D1%82%D1%8C-%D0%BE%D1%84%D0%B8%D1%81-%D0%B4%D0%BD%D0%B5%D0%BF%D1%80?area_min=40&area_max=40 [Дата звертання: 10.07.2021].

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис БОДНАР

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Робочий проект

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01215-01-ЛЗ

Завідувача кафедри КІТ
Вадим ГОРЯЧКІН

_____

Керівник розробки
Олександра ГОРБОВА

_____

Виконавець
Олександр СИРОТА

_____

Нормоконтролер
Олена КУРОП’ЯТНИК

_____

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01215-ЛЗ

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Специфікація

Аркушів 1

2021

Позначення	Найменування	Примітка
1116130.01215-01-ЛЗ	Лист затвердження	
1116130.01215-01 12 01-ЛЗ	Лист затвердження	
1116130.01215-01 12 01	Текст програми	
1116130.01215-01 13 01-ЛЗ	Лист затвердження	
1116130.01215-01 13 01	Опис програми	
1116130.01215-01 І3 01-ЛЗ	Лист затвердження	
1116130.01215-01 І3 01	Керівництво користувача. Керівництво з використання статичного аналізатору архітектури крос-платформного забезпечення.	

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01215-01 12 01-ЛЗ

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Текст програми

Листів 32

2021

АНОТАЦІЯ

Документ 1116130.01215-01 12 01 «Статичний аналізатор арзітектури крос-платформного програмного забезпечення. Текст програми» входить до складу документації для програми статичного аналізатору, що застосовується для аналізу наслідків використання патернів та загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS.

Документ містить текст програми. Програмне забезпечення розроблене з використанням об’єктно орієнтованої мови C#, мови створення інтерфейсу користувача Xaml, крос-платформного фреймворку Xamarin.Forms в інтегрованому середовищі розробки Visual Studio.

ЗМІСТ

1 СТРУКТУРА ПРОГРАМИ

Розроблене програмне забезпечення має наступні три модулі: модуль для налаштування статичного аналізатору; модуль для перегляду файлів та їх коду, що будуть проаналізовані; модуль аналізу архітектури. Ці модулі в свою чергу складаються з окремих підмодулів.

Модуль налаштування статичного аналізатору складається з: модулю інтерфейсу користувача; модулю, що опрацьовує запити від модулю інтерфейсу користувача; модулю для збереження та редагування налаштувань.

Модуль перегляду файлів та їх вмісту складається з: модулю інтерфейсу користувача; модулю, що опрацьовує запити від інтерфейсу користувача; модулю для обробки завантажених файлів.

Модуль аналізу архітектури використовує такі підмодулі: модулю інтерфейсу користувача; модулю, що опрацьовує запити від інтерфейсу користувача; модуль аналізу архітектури та збереження отриманих результатів.

На рис 1.1 зображено діаграму описаних модулів.

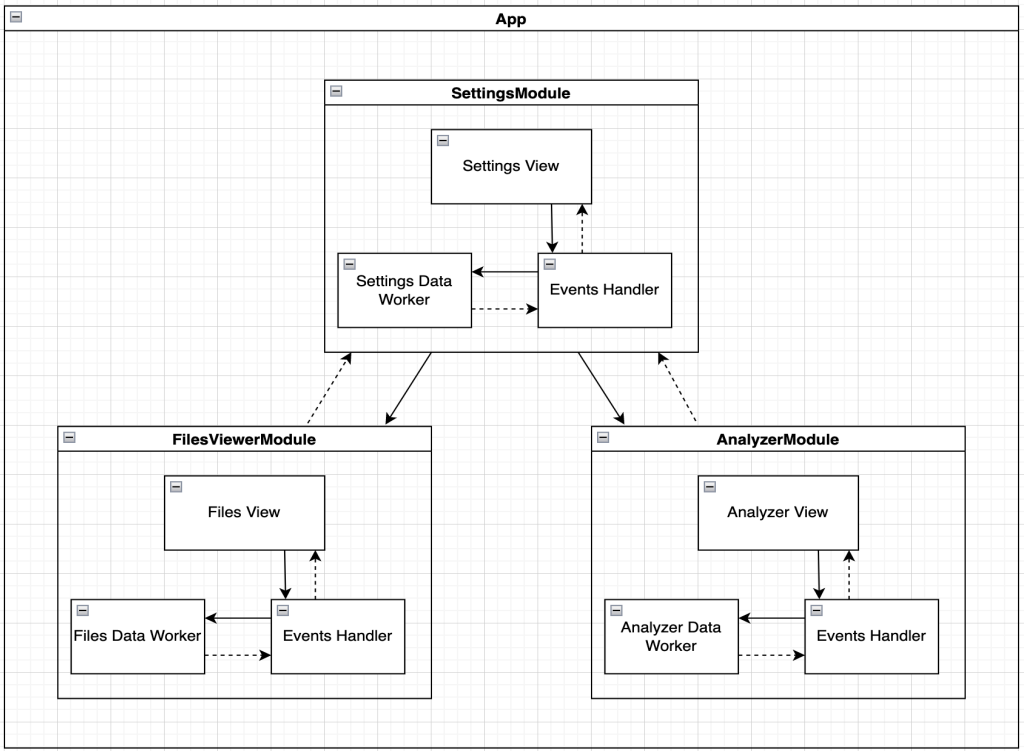


Рисунок 1.1 – Діаграма модулів

ТЕКСТ ПРОГРАМИ

2.1 Реалізація моделей та структур даних

AnalyzedDataModel.cs

```
using System;

namespace GraduateWork.Models
{
   // Базова модель, що містить базові властивості необхідні для аналізу певного класу
   public abstract class AnalyzedDataModel
   {
      public string Name;

      public bool HasReferences;
      public bool HasCorrectSuffix;
   }
}
```

AnylyzedViewDataModel.cs

```
using System;
using System.Collections.Generic;

namespace GraduateWork.Models
{
   // Модель, що використовується для аналізу реалізації користувацького інтерфейсу (UI),
   // який представлений на рівні View у вигляді файлів з розширенням .xaml.
   // Використовується сутністю MvvmAnalyzer для збереження результатів аналізу файлу руалізації UI.
   public class AnylyzedViewDataModel : AnalyzedDataModel
   {
      public bool HasXamlLayout;
      public bool HasAdditionalCodeBehindLogic;

      public List<string> BindingProperties;
   }
}
```

AnylyzedViewModelDataModel.cs

```
using System;

namespace GraduateWork.Models
{
   // Модель, що використовується для аналізу реалізації логіки користувацького інтерфейсу (UI),
   // що представлена на рівні ViewModel у вигляді файлів з розширенням .cs
   // Використовується сутністю MvvmAnalyzer для збереження результатів аналізу файлу руалізації логіки UI.
   public class AnylyzedViewModelDataModel : AnalyzedDataModel
   {
      public bool ImplementsBindableInterface;
   }
}
```

AnalyzedMvvmFileModel.cs

```
using System.Collections.Generic;
using Prism.Mvvm;

namespace GraduateWork.Models
{
   // Результуюча модель, що містить дані аналізу відповідних View та ViewModel файлів.
   // Використовується сутностями FileServiceEditor, MvvmAnalyzer для збереження результатів аналізу,
   // та AnalyzedMvvmFileHelper і PatternsAnalyzeResultViewModel для відображення в UI
   public class AnalyzedMvvmFileModel : BindableBase
   {
      public string ViewName { get; set; }
      public bool ViewHasReferences { get; set; }
```

```
    public bool ViewHasBindedProperties { get; set; }

    public string ViewModelName { get; set; }
    public bool ViewModelHasReferences { get; set; }
    public bool ViewModelIsBindable { get; set; }

    public AnalyzedMvvmFileModel()
        {
            ViewName = "Not Found";
            ViewModelName = "Not Found";
        }
    }
}
```

AppSize.cs

```
using System;

namespace GraduateWork.Enums
{
    // Структура даних для опису розмірів мобільних додатків.
    // Використовується сутностями AnalyzerSettingsViewModel та AppSizeToStringConverter для відображення в UI.
    public enum AppSize
    {
        Small,
        Medium,
        Big
    }
}
```

FileType.cs

```
using System;
namespace GraduateWork.Enums
{
    // Структура данних для опису типу файлів, що аналізуються.
    // Використовується сутностями AppFileService та FilesListViewModel
    // для завантаження та відображення в Ui списку файлів проекту.
    public enum FileType
    {
        cs,
        xaml
    }
}
```

AppConstants.cs

```
using System;
using System.Collections.Generic;
using GraduateWork.Enums;

namespace GraduateWork
{
    public static class AppConstants
    {
        public static readonly List<string> AppTypes = new List<string>
        {
            "Xamarin / .Net",
            "Flutter / Dart",
            "React Native / JavaScript"
        };

        public static Dictionary<AppSize, string> TimeToDevelop =
            new Dictionary<AppSize, string>()
            {
                { AppSize.Small, "to 6 months" },
                { AppSize.Medium, "6 months - year" },
                { AppSize.Big, "more than year" },
            };

        public static Dictionary<AppSize, int> AppBudget =
            new Dictionary<AppSize, int>()
            {
                { AppSize.Small, 60000 },
                { AppSize.Medium, 150000 },
                { AppSize.Big, 300000 },
            };
    }
```

```
public static class NavigationConstants
{
    public static readonly string AppDirectoryPath =
"app_directory_path";
    public static readonly string AppFilesPaths =
"app_files_paths";
}
```

2.2 Реалізація бізнес логіки

IAppFilesService.cs

```
using System.Collections.Generic;
using GraduateWork.Enums;
using GraduateWork.Models;

namespace GraduateWork.Services
{
    public interface IAppFilesService
    {
        string DirectoryPath { get; }

        IEnumerable<string> GetAllFilesPaths(string
directiryPath, IEnumerable<FileType> fileTypes =
null);

        string LoadFileContent(string filePath);

        IEnumerable<string> ReadFileContent(string
filePath);

        string GetShortFileNameFromPath(string
filePath);

        string GetFullFilePathFromName(string
fileName);

        int GetFileLinesCount(string filePath);

        int GetTotalLinesCount(List<string> filesPaths);

        void SaveToFile(string path, string text);

        void SaveAnalyzedMvvmInfo(string path,
List<AnalyzedMvvmFileModel>
analyzedMvvmResults);
        void SaveAnalyzedMvvmFilesContent(string
path, List<AnalyzedMvvmFileModel>
analyzedMvvmResults);
    }
}
```

AppFilesService.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using GraduateWork.Enums;
using GraduateWork.Helpers;
using GraduateWork.Models;
using Xamarin.Forms.Internals;

namespace GraduateWork.Services
{
    public class AppFilesService : IAppFilesService
    {
        public string DirectoryPath { get; private set; }

        List<string> _filesPaths;

        public IEnumerable<string>
GetAllFilesPaths(string directoryPath,
IEnumerable<FileType> fileTypes = null)
        {
            DirectoryPath = directoryPath;

            List<string> files = new List<string>();

            if (File.Exists(directoryPath))
            {
                // This path is a file
                files.Add(directoryPath);

                return files;
```

```
        }

        if (Directory.Exists(directoryPath))
        {
            // This path is a directory

files.AddRange(ProcessFilesInDirectory(directoryPat
h, fileTypes));
        }
        else
        {
            Console.WriteLine("{0} is not a valid file
or directory.", directoryPath);
        }

        _filesPaths = new List<string>(files);

        return files;
    }

    public string GetFullFilePathFromName(string
fileName)
    {
        foreach (var path in _filesPaths)
        {
            if (path.Substring(path.LastIndexOf("/") +
1).Contains(fileName))
            {
                return path;
            }
        }

        return null;
    }

    public int GetFileLinesCount(string filePath)
    {
        var fileContent = ReadFileContent(filePath);

        return fileContent != null ?
fileContent.Count() : 0;
    }

    public int GetTotalLinesCount(List<string>
filesPaths)
    {
        int linesCount = 0;

        foreach (var filePath in filesPaths)
        {
            var fileContent =
ReadFileContent(filePath);
            linesCount += fileContent.Count();
        }

        return linesCount;
    }

    public string GetShortFileNameFromPath(string
filePath)
    {
        return
filePath.Substring(filePath.LastIndexOf("/") + 1);
    }

    public string LoadFileContent(string filePath)
    {
        string text = File.ReadAllText(filePath);

        return text;
    }

    public IEnumerable<string>
ReadFileContent(string filePath)
    {
        var text = File.ReadLines(filePath);

        return text;
    }

    // Process all files in the directory passed in,
recurse on any directories
    // that are found, and process the files they
contain.
    private IEnumerable<string>
ProcessFilesInDirectory(string targetDirectory,
IEnumerable<FileType> fileTypes)
    {
        List<string> filesListByType = new
List<string>();

        // Process the list of files found in the
directory.
        var loadedFiles =
Directory.GetFiles(targetDirectory);

        if (fileTypes != null &&
fileTypes.ToList().Count > 0)
        {
            fileTypes.ForEach((type) => {
```

```
            var fileTypeAsString =
type.ToString().ToLower();

            var files = loadedFiles.Where(
               fileName => fileName.EndsWith("." +
fileTypeAsString)
            ).ToList();

            filesListByType.AddRange(files);
         });
      }
      else
      {
         filesListByType.AddRange(loadedFiles);
      }

      // Recurse into subdirectories of this directory.
      var subdirectoryEntries =
Directory.GetDirectories(targetDirectory);

      foreach (string subdirectory in
subdirectoryEntries)
      {
         bool needProccesSubdirectory =
!subdirectory.EndsWith("obj")
            && !subdirectory.EndsWith("bin");

         if (needProccesSubdirectory)
         {

filesListByType.AddRange(ProcessFilesInDirectory(
subdirectory, fileTypes));
         }
      }

      return filesListByType;
   }

   public void SaveToFile(string path, string text)
   {
      //Open the stream and write to it.
      using (FileStream fs = File.OpenWrite(path))
      {
         Byte[] info = new
UTF8Encoding(true).GetBytes(text);

         // Add some information to the file.
         fs.Write(info, 0, info.Length);
      }
   }

   public void SaveAnalyzedMvvmFilesContent(
      string path,
      List<AnalyzedMvvmFileModel>
analyzedMvvmResults)
   {
      string fileContent = string.Empty;

      foreach (var mvvm in analyzedMvvmResults)
      {
         if (fileContent == string.Empty)
         {
            string currentDateAsString =
DateTime.Now.ToLocalTime().ToLongDateString();

            string header = "Analysed MVVM files
content.";

            fileContent += currentDateAsString +
"\n";
            fileContent += header + "\n";
            fileContent +=
separatorLineAsString(header.Length) + "\n\n";

            string totalLinesCount = "Total lines
count of the project: "
               + GetTotalLinesCount(new
List<string>(GetAllFilesPaths(DirectoryPath))).ToStr
ing();

            fileContent += totalLinesCount + "\n";
            fileContent +=
separatorLineAsString(totalLinesCount.Length) +
"\n\n";
         }

         var viewFilePath =
GetFullFilePathFromName(mvvm.ViewName);

         if (viewFilePath != null)
         {
            string header = mvvm.ViewName;

            fileContent += header + "\n\n";
            fileContent +=
LoadFileContent(viewFilePath) + "\n";
            fileContent +=
separatorLineAsString(80) + "\n\n";
         }
```

```
            var viewModelFilePath =
GetFullFilePathFromName(mvvm.ViewModelName)
;

            if (viewModelFilePath != null)
            {
               string header = mvvm.ViewModelName;

               fileContent += header + "\n\n";
               fileContent +=
LoadFileContent(viewModelFilePath) + "\n";
               fileContent +=
separatorLineAsString(80) + "\n\n";
            }
         }

         if (fileContent != string.Empty)
         {
            SaveToFile(path, fileContent);
         }
      }

      public void SaveAnalyzedMvvmInfo(
         string path,
         List<AnalyzedMvvmFileModel>
analyzedMvvmResults)
      {
         string currentDateAsString =
DateTime.Now.ToLocalTime().ToLongDateString();

         const string hasReferences = "Has
references";
         const string hasNoReferences = "Has No
references";

         const string hasBindedProperties = "Has
binded properties";
         const string hasNoBindedProperties = "Has
No binded properties";

         const string IsBindable = "Is bindable";
         const string IsNotBindable = "Is Not
bindable";

         string mvvmText = null;

         int linesCount = 1;

         foreach (var result in analyzedMvvmResults)
         {
            if (mvvmText == null)
            {
               string header = "MVVM pattern
analysing info.";

               mvvmText += currentDateAsString +
"\n";
               mvvmText += header + "\n";
               mvvmText +=
separatorLineAsString(header.Length) + "\n\n";

               int viewsWithViewModelsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewHasBindedProperties,
analyzedMvvmResults);

               string totalViewsCount = "Total Views
count: "
                  +
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewName.Contains(".xaml"),
analyzedMvvmResults);

               string viewsWithRefsCount = "Views
with refs count: "
                  +
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewHasReferences,
analyzedMvvmResults);

               string viewsMatchedViewModelsCount
= "Views matched with ViewModels count: " +
viewsWithViewModelsCount;

               string totalViewModelsCount = "Total
View models count: "
                  +
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelName.Contains(".cs"),
analyzedMvvmResults);

               string viewModelsMatchedViewsCount
= "ViewModels matched with Views count: " +
viewsWithViewModelsCount;

               string viewModelsWithRefsCount =
"ViewModels with refs count: " +

AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelHasReferences,
analyzedMvvmResults);
```

```
            string bindableViewModelsCount =
"Bindable ViewModels count: " +

AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelIsBindable,
analyzedMvvmResults);

            string mvvmShortInfo =
totalViewsCount + "\n"
                + viewsWithRefsCount + "\n"
                + viewsMatchedViewModelsCount +
"\n"
                + totalViewModelsCount + "\n"
                + viewModelsMatchedViewsCount +
"\n"
                + viewModelsWithRefsCount + "\n"
                + bindableViewModelsCount + "\n";

            mvvmText += mvvmShortInfo;
            mvvmText +=
separatorLineAsString(bindableViewModelsCount.L
ength) + "\n\n";
        }

        string mvvmInfoLine = linesCount + ". "
            + result.ViewName + ": "
            + (result.ViewHasReferences ?
hasReferences : hasNoReferences) + ", "
            + (result.ViewHasBindedProperties ?
hasBindedProperties : hasNoBindedProperties) + ", "
            + result.ViewModelName + ": "
            + (result.ViewModelHasReferences ?
hasReferences : hasNoReferences) + ", "
            + (result.ViewModelIsBindable ?
IsBindable : IsNotBindable) + "\n\n";

            mvvmText += mvvmInfoLine;
            mvvmText +=
separatorLineAsString(mvvmInfoLine.Length) +
"\n\n";

            linesCount++;
        }

        if (mvvmText != null && mvvmText !=
string.Empty)
        {
            SaveToFile(path, mvvmText);
        }
    }

    private string separatorLineAsString(int length)
    {
        string separator = string.Empty;

        for (int i = 0; i < length; i++)
        {
            separator += "_";
        }

        return separator;
    }
  }
}
```

IFileEditorService.cs

```
using System;
using System.Collections.Generic;
using GraduateWork.Models;

namespace GraduateWork.Services
{
  public interface IFileEditorService
  {
    List<string> RemoveEmptyLines(List<string>
fileContent);

    List<string> RemoveCommentedLines(
        List<string> fileContent,
        string startCommentSymbol,
        string endCommentSymbol);
  }
}

using System;
using System.Collections.Generic;
using GraduateWork.Helpers;
using GraduateWork.Models;
using GraduateWork.Services;

namespace GraduateWork.Services
{
  public class FileEditorService : IFileEditorService
  {
```

```
        private IAppFilesService _appFilesService;

        public FileEditorService(IAppFilesService
appFilesService)
        {
            _appFilesService = appFilesService;
        }

        public List<string>
RemoveEmptyLines(List<string> fileContent)
        {
            List<string> fileWithoutEmptyLines = new
List<string>();

            fileContent.ForEach(line =>
            {
                if (line != string.Empty)
                {
                    fileWithoutEmptyLines.Add(line);
                }
            });

            return fileWithoutEmptyLines;
        }

        public List<string> RemoveCommentedLines(
            List<string> fileContent,
            string startCommentSymbol,
            string endCommentSymbol)
        {
            List<string> fileWithoutComments = new
List<string>();

            bool isCommentedLine = false;

            foreach (var line in fileContent)
            {
                if (line.StartsWith(startCommentSymbol))
                {
                    isCommentedLine = true;
                }
                else if
(line.EndsWith(endCommentSymbol))
                {
                    isCommentedLine = false;
                    continue;
                }

                if (isCommentedLine)
                {
                    continue;
                }
                else
                {
                    fileWithoutComments.Add(line);
                }
            }

            return fileWithoutComments;
        }

        private List<string>
RemoveSingleCommentedLines(
            List<string> fileContent,
            string commentSymbol)
        {
            List<string> fileWithoutComments = new
List<string>();

            foreach (var line in fileContent)
            {
                if (line.StartsWith(commentSymbol))
                {
                    continue;
                }
                else
                {
                    fileWithoutComments.Add(line);
                }
            }

            return fileWithoutComments;
        }
    }
}
```

IMvvmAnalyzer.cs

```
using System;
using System.Collections.Generic;
using GraduateWork.Models;

namespace GraduateWork.Services
{
```

```
   public interface IMvvmAnalyzer
   {
      List<AnalyzedMvvmFileModel>
AnalyzeMvvmImplementation(IEnumerable<string>
filePaths);

      AnylyzedViewDataModel
AnalyzeViewXamlFile(string filePath);
      AnylyzedViewModelDataModel
AnalyzeViewModelFile(string filePath);
      List<AnylyzedViewDataModel>
AnalyzeViews(IEnumerable<string> filePaths);
      List<AnylyzedViewModelDataModel>
AnalyzeViewModels(IEnumerable<string>
filePaths);
   }
}
```

MvvmAnalyzer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using GraduateWork.Models;

namespace GraduateWork.Services
{
   public class MvvmAnalyzer : IMvvmAnalyzer
   {
      const string ViewXamlKeyWord = "View.xaml";
      const string ViewModelCsKeyWord =
"ViewModel.cs";

      private IAppFilesService _filesService;
      private IFileEditorService _fileEditorService;

      private List<string> _filePaths;

      private Dictionary<string, List<string>>
_viewModelFilesContent;

      public MvvmAnalyzer(
         IAppFilesService filesService,
         IFileEditorService fileEditorService)
      {
         _filesService = filesService;
         _fileEditorService = fileEditorService;

         _viewModelFilesContent = new
Dictionary<string, List<string>>();
      }

      public List<AnalyzedMvvmFileModel>
AnalyzeMvvmImplementation(IEnumerable<string>
filePaths)
      {
         _filePaths = filePaths.ToList();

         var anylizedViews =
AnalyzeViews(filePaths);

         var anylizedViewModels =
AnalyzeViewModels(filePaths);

         var analyzedMvvmFiles = new
List<AnalyzedMvvmFileModel>();

         if (anylizedViews != null &&
anylizedViews.Count > 0)
         {
            foreach (var view in anylizedViews)
            {
               var analyzedMvvmFile = new
AnalyzedMvvmFileModel();

               analyzedMvvmFile.ViewName =
view.Name;
               analyzedMvvmFile.ViewHasReferences
= view.HasReferences;

               if (anylizedViewModels != null &&
anylizedViewModels.Count > 0)
               {
                  var viewModel =
GetViewModelByViewName(view.Name,
anylizedViewModels);

                  if (viewModel != null)
                  {

analyzedMvvmFile.ViewModelName =
viewModel.Name;

analyzedMvvmFile.ViewModelHasReferences =
viewModel.HasReferences;
```

```
analyzedMvvmFile.ViewHasBindedProperties =
CheckViewBindedWithViewModel(
                        GetViewByName(view.Name,
anylizedViews), viewModel);


analyzedMvvmFile.ViewModelIsBindable =
viewModel.ImplementsBindableInterface;
                    }
                }



analyzedMvvmFiles.Add(analyzedMvvmFile);
            }
        }

        if (anylizedViewModels != null &&
anylizedViewModels.Count > 0)
        {
            foreach (var analyzedViewModel in
anylizedViewModels)
            {
                bool viewModelAdded =
analyzedMvvmFiles.FirstOrDefault(
                    item => item.ViewModelName ==
analyzedViewModel.Name) == null ? false : true;

                if (!viewModelAdded)
                {
                    var analyzedMvvmFile = new
AnalyzedMvvmFileModel();

                    analyzedMvvmFile.ViewModelName
= analyzedViewModel.Name;

analyzedMvvmFile.ViewModelHasReferences =
analyzedViewModel.HasReferences;

analyzedMvvmFile.ViewModelIsBindable =
analyzedViewModel.ImplementsBindableInterface;


analyzedMvvmFiles.Add(analyzedMvvmFile);
                }
            }
        }

        return analyzedMvvmFiles;
    }

    public List<AnylyzedViewDataModel>
AnalyzeViews(IEnumerable<string> filePaths)
    {
        List<AnylyzedViewDataModel>
anylizedViews = new
List<AnylyzedViewDataModel>();

        foreach (var path in filePaths)
        {
            if (path.EndsWith(ViewXamlKeyWord))
            {
                var anylyzedView =
AnalyzeViewXamlFile(path);
                anylizedViews.Add(anylyzedView);
            }
        }

        return anylizedViews;
    }

    public List<AnylyzedViewModelDataModel>
AnalyzeViewModels(IEnumerable<string> filePaths)
    {
        List<AnylyzedViewModelDataModel>
anylizedViewModels = new
List<AnylyzedViewModelDataModel>();

        foreach (var path in filePaths)
        {
            if (path.Contains(ViewModelCsKeyWord))
            {
                var anylyzedViewModel =
AnalyzeViewModelFile(path);

anylizedViewModels.Add(anylyzedViewModel);
            }
        }

        return anylizedViewModels;
    }

    public AnylyzedViewModelDataModel
AnalyzeViewModelFile(string filePath)
    {
        AnylyzedViewModelDataModel
anylyzedViewModel = new
AnylyzedViewModelDataModel();
```

```
        anylyzedViewModel.HasCorrectSuffix =
filePath.EndsWith(ViewModelCsKeyWord);

        if (anylyzedViewModel.HasCorrectSuffix)
        {
            var content =
_fileEditorService.RemoveEmptyLines(

_filesService.ReadFileContent(filePath).ToList()
            );

            anylyzedViewModel.Name =
_filesService.GetShortFileNameFromPath(filePath);

            anylyzedViewModel.HasReferences =
isFileHaveReferences(anylyzedViewModel.Name);


anylyzedViewModel.ImplementsBindableInterface =
isBindableViewModel(content,
anylyzedViewModel.Name);


_viewModelFilesContent.Add(anylyzedViewModel.
Name, content);
        }

        return anylyzedViewModel;
    }

    private bool isBindableViewModel
(List<string> viewModelContent, string fileName)
    {
        const string InotifyPropertyChanged =
"INotifyPropertyChanged";
        const string BindableBase = "BindableBase";

        string viewModelName =
NameWithoutType(fileName);

        bool isBindableViewmodel = false;

        foreach (var line in viewModelContent)
        {
            bool viewModelHasInheritance =
line.Contains("class")
                && line.Contains(viewModelName)
                && line.Contains(":") &&
!line.Contains("base");

            if (viewModelHasInheritance)
            {
                string inheritedInterfaceName =
line.Substring(line.LastIndexOf(":") + 2);

                for (int itemIndex = 0; itemIndex <
inheritedInterfaceName.Length; itemIndex++)
                {
                    if (inheritedInterfaceName[itemIndex]
== ',')
                    {
                        inheritedInterfaceName =
inheritedInterfaceName.Remove(itemIndex);
                        break;
                    }
                }

                bool isBindableInheritedInterface =
inheritedInterfaceName == InotifyPropertyChanged
                    || inheritedInterfaceName ==
BindableBase;

                if (isBindableInheritedInterface)
                {
                    isBindableViewmodel = true;
                    break;
                }
                else
                {
                    string fullFilePath = null;

                    foreach(var path in _filePaths)
                    {
                        var filePathWithSpaces =
path.Replace('/', ' ');

                        if
(filePathWithSpaces.EndsWith(inheritedInterfaceNa
me + ".cs"))
                        {
                            fullFilePath = path;
                            break;
                        }
                    }

                    if (fullFilePath != null)
                    {
                        var content =
_fileEditorService.RemoveEmptyLines(

_filesService.ReadFileContent(fullFilePath).ToList()
                        );
```

```
                    isBindableViewmodel =
isBindableViewModel(content,
inheritedInterfaceName);
                }
            }
        }
    }

    return isBindableViewmodel;
}

public AnylyzedViewDataModel
AnalyzeViewXamlFile(string filePath)
{
    AnylyzedViewDataModel anylyzedView =
new AnylyzedViewDataModel();

    anylyzedView.HasCorrectSuffix =
filePath.EndsWith(ViewXamlKeyWord);

    if (anylyzedView.HasCorrectSuffix)
    {
        List<string> content =
PrepareViewFileContent(_filesService.ReadFileCont
ent(filePath).ToList());

        bool isOpeningTagCorrect =
isOpeningViewRootTagCorrect(content);
        bool isClosingTagCorrect =
isClosingViewRootTagCorrect(content);

        anylyzedView.Name =
_filesService.GetShortFileNameFromPath(filePath);

        anylyzedView.HasReferences =
isFileHaveReferences(anylyzedView.Name);

        anylyzedView.HasXamlLayout =
isOpeningTagCorrect && isClosingTagCorrect;

        anylyzedView.BindingProperties = new
List<string>(FindViewBindingProperties(content));
    }

    return anylyzedView;
}

private bool isFileHaveReferences(string name)
{
    string formattedName =
NameWithoutType(name);

    bool hasReferences = false;

    if (_filePaths != null)
    {
        foreach (var path in _filePaths)
        {
            var content =
_filesService.LoadFileContent(path);

            if (path.EndsWith(name) ||
path.EndsWith(name + ".cs"))
            {
                continue;
            }

            if (content.Contains(formattedName))
            {
                hasReferences = true;
                break;
            }
        }
    }

    return hasReferences;
}

private List<string> FindViewBindingProperties
(List<string> fileContent)
{
    const string BindingKeyWord = "Binding";

    var bindingProperties = new List<string>();

    fileContent.ForEach(line =>
    {
        if (line.Contains(BindingKeyWord + " "))
        {
            int from =
line.LastIndexOf(BindingKeyWord + " " );
            string lineWithPropertyName =
line.Substring(from);

            string[] splittedLine =
lineWithPropertyName.Split(' ');

            string propertyName = string.Empty;

            // A property name index == 1
```

```
            if (splittedLine[1].Contains(","))
            {
                propertyName =
splittedLine[1].Remove(splittedLine[1].LastIndexOf(
","));
            }
            else if (splittedLine[1].Contains("}"))
            {
                propertyName =
splittedLine[1].Remove(splittedLine[1].LastIndexOf(
"}"));
            }
            else
            {
                propertyName = splittedLine[1];
            }

            Console.WriteLine("Property name: " +
propertyName);

            bindingProperties.Add(propertyName);
        }
    });

    return bindingProperties;
}

private bool
isOpeningViewRootTagCorrect(List<string>
fileContent)
{
    bool firstOpeningTagSymbolFound = false;
    bool lastOpeningTagSymbolFound = false;

    bool hasXamarinFormsLink = false;
    bool hasXamlLink = false;

    // Usually an opening tag contains several
lines
    foreach (var line in fileContent)
    {
        // Skip a line with xml version which
usually looks like this:
        // <?xml version="1.0"
encoding="UTF-8"?>
        if (line.StartsWith("<?xml version="))
        {
            continue;
        }

        // All Xamarin.Forms opening page tags
contain 'Page' word and <, > symbols
        if (!firstOpeningTagSymbolFound)
        {
            firstOpeningTagSymbolFound =
line.StartsWith("<")
                && line.Contains("Page>");
        }

        if (firstOpeningTagSymbolFound &&
lastOpeningTagSymbolFound)
        {
            break;
        }

        if (!hasXamarinFormsLink)
        {
            hasXamarinFormsLink =
line.Contains("http://xamarin.com/schemas/")
                && line.Contains("/forms");
        }

        bool lineContainsXamlLink =
line.Contains("http://schemas.microsoft.com/winfx/")
            && line.Contains("/xaml");

        if (!hasXamlLink)
        {
            hasXamlLink = lineContainsXamlLink;
        }

        lastOpeningTagSymbolFound =
line.EndsWith(">");
    }

    return firstOpeningTagSymbolFound
        && lastOpeningTagSymbolFound
        && hasXamarinFormsLink
        && hasXamlLink;
}

private bool
isClosingViewRootTagCorrect(List<string>
fileContent)
{
    // All Xamarin.Forms closing page tags
contain 'Page' word and </, > symbols
    return fileContent.Last().StartsWith("</")
        && fileContent.Last().Contains("Page>");
}
```

```
        private bool
CheckViewBindedWithViewModel(AnylyzedViewD
ataModel view, AnylyzedViewModelDataModel
viewModel)
        {
            if (view != null && viewModel != null &&
viewModel.ImplementsBindableInterface)
            {
                var viewModelContent =
_viewModelFilesContent[viewModel.Name];

                foreach (var property in
view.BindingProperties)
                {
                    foreach (var line in viewModelContent)
                    {
                        if (line.Contains(property))
                        {
                            return true;
                        }
                    }
                }
            }

            return false;
        }

        private List<string> PrepareViewFileContent
(List<string> fileContent)
        {
            List<string> updatedFileContent =
_fileEditorService.RemoveEmptyLines(fileContent);
            updatedFileContent =
_fileEditorService.RemoveCommentedLines(updated
FileContent, "<!--", "-->");

            return updatedFileContent;
        }

        private string NameWithoutType (string
fileName)
        {
            string formattedName = fileName;

            if (formattedName.EndsWith(".xaml") ||
formattedName.EndsWith(".cs"))
            {
                formattedName =
formattedName.Remove(formattedName.LastIndexO
f("."));
            }

            return formattedName;
        }

        private string NameWithoutSuffix(string
fileName, bool isViewSuffix)
        {
            var simpleName =
NameWithoutType(fileName);

            int substringLength = isViewSuffix ? 4 : 9;

            string fileNameWithoutSuffix =
simpleName.Substring(0, simpleName.Length -
substringLength);

            return fileNameWithoutSuffix;
        }

        private AnylyzedViewDataModel
GetViewByName(string viewName,
List<AnylyzedViewDataModel> views)
        {
            foreach (var view in views)
            {
                if (NameWithoutType(view.Name) ==
NameWithoutType(viewName))
                {
                    return view;
                }
            }

            return null;
        }

        private AnylyzedViewModelDataModel
GetViewModelByName(
            string viewModelName,
            List<AnylyzedViewModelDataModel>
viewModels)
        {
            foreach (var viewModel in viewModels)
            {
                if (NameWithoutType(viewModel.Name)
== NameWithoutType(viewModelName))
                {
                    return viewModel;
                }
            }

            return null;
```

```
        }

        private AnylyzedViewModelDataModel
GetViewModelByViewName(
            string viewName,
            List<AnylyzedViewModelDataModel>
viewModels)
        {
            foreach (var viewModel in viewModels)
            {
                string viewSimpleName =
NameWithoutSuffix(viewName, true);
                string viewModelSimpleName =
NameWithoutSuffix(viewModel.Name, false);
                if (viewSimpleName ==
viewModelSimpleName)
                {
                    return viewModel;
                }
            }

            return null;
        }
    }
}
```

AnalyzedMvvmFileHelper.cs

```
using System;
using System.Collections.Generic;
using GraduateWork.Models;

namespace GraduateWork.Helpers
{
    public static class AnalyzedMvvmFileHelper
    {
        public static int
CalculateMvvmPropsCount(Func<AnalyzedMvvmFi
leModel, bool> expression,
List<AnalyzedMvvmFileModel> mvvmResults)
        {
            int count = 0;

            foreach (var mvvmItem in mvvmResults)
            {
                if (expression(mvvmItem))
                {
                    count++;
                }
            }

            return count;
        }
    }
}
```

2.3 Реалізація логіки інтерфейсу користувача

AnalyzerSettingsViewModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Windows.Input;
using GraduateWork.Enums;
using GraduateWork.Services;
using GraduateWork.Views;
using Prism.Navigation;
using Xamarin.Forms;

namespace GraduateWork.ViewModels
{
    public class AnalyzerSettingsViewModel :
ViewModelBase
    {
        private IAppFilesService _appFilesService;

        private List<string> _appFiles;

        public AnalyzerSettingsViewModel(
            INavigationService navigationService,
            IAppFilesService appFilesService
        ) : base(navigationService)
        {
```

```
        _appFilesService = appFilesService;

        AnalyzeAppCommand = new
Command(OnAnalyzeAppCommand);
        ShowFilesCommand = new
Command(OnShowFilesCommand);

        _selecteAppBudget =
AppConstants.AppBudget.Values.ToList().First();
        _selectedDevelopTime =
AppConstants.TimeToDevelop.Values.ToList().First(
);

        _appFiles = new List<string>();
    }

    public ICommand AnalyzeAppCommand { get;
}

    public ICommand ShowFilesCommand { get; }

    public ICommand ClearCommand { get; }

    private bool _buttonsEnabled;
    public bool ButtonsEnabled
    {
        get => _buttonsEnabled;
        set => SetProperty(ref _buttonsEnabled,
value);
    }

    public List<string> AppSizeList
    {
        get =>
Enum.GetNames(typeof(AppSize)).ToList();
    }

    private AppSize _selectedAppSize;
    public AppSize SelectedAppSize
    {
        get => _selectedAppSize;
        set
        {
            SetProperty(ref _selectedAppSize, value);

            SetDevelopTimeBySize(value);
            SetAppBudgetBySize(value);
        }
    }

    public List<string> AppTypeList
    {
        get => AppConstants.AppTypes;
    }

    private string _selectedAppType;
    public string SelectedAppType
    {
        get => _selectedAppType;
        set
        {
            SetProperty(ref _selectedAppType, value);

            // As this app logic implemented only for
Xamarin / .Net project type
            // we always set the Xamarin / .Net project
type as selected
            SetProperty(ref _selectedAppType,
AppConstants.AppTypes.First());
        }
    }

    public List<string> DevelopTimeList
    {
        get =>
AppConstants.TimeToDevelop.Values.ToList();
    }

    private string _selectedDevelopTime;
    public string SelectedDevelopTime
    {
        get => _selectedDevelopTime;
        set => SetProperty(ref
_selectedDevelopTime, value);
    }

    public List<string> AppBudgetList
    {
        get
        {
            List<string> budgetList = new
List<string>();

            foreach(var value in
AppConstants.AppBudget.Values)
            {
                budgetList.Add(value.ToString());
            }

            return budgetList;
        }
    }
```

```
        private int _selecteAppBudget;
        public int SelectedAppBudget
        {
            get => _selecteAppBudget;
            set => SetProperty(ref _selecteAppBudget,
value);
        }

        private string _projectDirectoryPath;
        public string ProjectDirectoryPath
        {
            get => _projectDirectoryPath;
            set => SetProperty(ref _projectDirectoryPath,
value);
        }

        protected override void
OnPropertyChanged(PropertyChangedEventArgs
args)
        {
            base.OnPropertyChanged(args);

            if (args.PropertyName ==
nameof(ProjectDirectoryPath))
            {
                // ToDo: fix the case when the string
contains only whitespaces
                ButtonsEnabled =
ProjectDirectoryPath.Length > 0 &&
ProjectDirectoryPath != string.Empty;
            }
        }

        private void SetAppBudgetBySize(AppSize
projectSize)
        {
            SelectedAppBudget =
AppConstants.AppBudget[projectSize];
        }

        private void SetDevelopTimeBySize(AppSize
projectSize)
        {
            SelectedDevelopTime =
AppConstants.TimeToDevelop[projectSize];
        }

        private async void OnAnalyzeAppCommand()
        {
            _appFiles =
_appFilesService.GetAllFilesPaths(_projectDirectory
Path).ToList();

            if (_appFiles.Count > 0)
            {
                var navigationParams = new
NavigationParameters
                {
                    { NavigationConstants.AppFilesPaths,
_appFiles }
                };

                await
NavigationService.NavigateAsync($"{nameof(Patter
nsAnalyzeResultView)}", navigationParams);
            }
        }

        private async void OnShowFilesCommand()
        {
            if (_projectDirectoryPath.Length > 0)
            {
                var navigationParams = new
NavigationParameters
                {
                    {
NavigationConstants.AppDirectoryPath,
_projectDirectoryPath }
                };

                await
NavigationService.NavigateAsync($"{nameof(FilesL
istView)}", navigationParams);
            }
        }
    }
}
```

FilesListViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
```

```
using System.Linq;
using System.Windows.Input;
using GraduateWork.Enums;
using GraduateWork.Services;
using Prism.Navigation;
using Xamarin.Forms;

namespace GraduateWork.ViewModels
{
   public class FilesListViewModel :
ViewModelBase
   {
      private IAppFilesService _appFilesService;

      private List<string> _loadedAppFilesPaths;

      public FilesListViewModel(
         INavigationService navigationService,
         IAppFilesService appFilesService
      ) : base(navigationService)
      {
         _appFilesService = appFilesService;

         _loadedAppFilesPaths = new List<string>();
         _appFiles = new
ObservableCollection<string>();
      }

      private ObservableCollection<string>
_appFiles;
      public ObservableCollection<string> AppFiles
      {
         get { return _appFiles; }
         set => SetProperty(ref _appFiles, value);
      }

      private string _selectedAppFile;
      public string SelectedAppFile
      {
         get { return _selectedAppFile; }
         set => SetProperty(ref _selectedAppFile,
value);
      }

      private string _fileContent;
      public string FileContent
      {
         get { return _fileContent; }
         set => SetProperty(ref _fileContent, value);
      }

      private int _fileLinesCount;
      public int FileLinesCount
      {
         get { return _fileLinesCount; }
         set => SetProperty(ref _fileLinesCount,
value);
      }

      protected override void
OnPropertyChanged(PropertyChangedEventArgs
args)
      {
         base.OnPropertyChanged(args);

         if (args.PropertyName ==
nameof(SelectedAppFile))
         {
            LoadFileContent(SelectedAppFile);
         }
      }

      public override void
OnNavigatedTo(INavigationParameters parameters)
      {
         base.OnNavigatedTo(parameters);

         if
(parameters.ContainsKey(NavigationConstants.App
DirectoryPath))
         {
            string path =
parameters.GetValue<string>(NavigationConstants.A
ppDirectoryPath);
            LoadAppFiles(path);
         }
      }

      private void LoadAppFiles (string path)
      {
         _loadedAppFilesPaths
=_appFilesService.GetAllFilesPaths(
            path, new List<FileType>() { FileType.cs,
FileType.xaml }
         ).ToList();

         if (_loadedAppFilesPaths.Count > 0)
         {
            AppFiles = new
ObservableCollection<string>();
```

```
            foreach (var fileNamePath in
_loadedAppFilesPaths)
            {
                string name =
_appFilesService.GetShortFileNameFromPath(fileNa
mePath);
                AppFiles.Add(name);
            }
        }
    }

    private void LoadFileContent(string fileName)
    {
        try
        {
            string path = string.Empty;

            foreach (var filePath in
_loadedAppFilesPaths)
            {
                if (filePath.EndsWith(fileName))
                {
                    if
(_appFilesService.GetShortFileNameFromPath(fileP
ath) == fileName)
                    {
                        path = filePath;
                    }
                }
            }

            FileContent =
_appFilesService.LoadFileContent(path);

            FileLinesCount =
_appFilesService.GetFileLinesCount(path);
        }
        catch(Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
  }
}
```

PatternsAnalyzeResultViewModel.cs

```
using System;
using System.Collections.Generic;
using GraduateWork.Helpers;
using GraduateWork.Models;
using GraduateWork.Services;
using Prism.Navigation;

namespace GraduateWork.ViewModels
{
  public class PatternsAnalyzeResultViewModel :
ViewModelBase
  {
    private IMvvmAnalyzer _mvvmAnalyzer;
    private IAppFilesService _appFilesService;

    public PatternsAnalyzeResultViewModel(
      INavigationService navigationService,
      IMvvmAnalyzer mvvmAnalyzer,
      IAppFilesService appFilesService)
      : base(navigationService)
    {
      _mvvmAnalyzer = mvvmAnalyzer;
      _appFilesService = appFilesService;
    }

    private List<AnalyzedMvvmFileModel>
_mvvmResults;
    public List<AnalyzedMvvmFileModel>
MvvmResults
    {
      get { return _mvvmResults; }
      set => SetProperty(ref _mvvmResults, value);
    }

    private int _totalViewsCount;
    public int TotalViewsCount
    {
      get { return _totalViewsCount; }
      set => SetProperty(ref _totalViewsCount,
value);
    }

    private int _viewsWithRefsCount;
    public int ViewsWithRefsCount
    {
      get { return _viewsWithRefsCount; }
      set => SetProperty(ref _viewsWithRefsCount,
value);
```

```
        }

        private int _viewsWithViewModelsCount;
        public int ViewsWithViewModelsCount
        {
            get { return _viewsWithViewModelsCount; }
            set => SetProperty(ref
_viewsWithViewModelsCount, value);
        }

        private int _totalViewModelsCount;
        public int TotalViewModelsCount
        {
            get { return _totalViewModelsCount; }
            set => SetProperty(ref
_totalViewModelsCount, value);
        }

        private int _vMsWithViewsCount;
        public int VMsWithViewsCount
        {
            get { return _vMsWithViewsCount; }
            set => SetProperty(ref
_vMsWithViewsCount, value);
        }

        private int _vMsWithRefsCount;
        public int VMsWithRefsCount
        {
            get { return _vMsWithRefsCount; }
            set => SetProperty(ref _vMsWithRefsCount,
value);
        }

        private int _bindableVMsCount;
        public int BindableVMsCount
        {
            get { return _bindableVMsCount; }
            set => SetProperty(ref _bindableVMsCount,
value);
        }

        public override void
OnNavigatedTo(INavigationParameters parameters)
        {
            base.OnNavigatedTo(parameters);

            if
(parameters.ContainsKey(NavigationConstants.AppF
ilesPaths))
            {
                var paths =
parameters.GetValue<List<string>>(NavigationCons
tants.AppFilesPaths);

                MvvmResults = new
List<AnalyzedMvvmFileModel>(

_mvvmAnalyzer.AnalyzeMvvmImplementation(path
s)
                );

                TotalViewsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewName.Contains(".xaml"),
MvvmResults);
                ViewsWithRefsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewHasReferences,
MvvmResults);
                ViewsWithViewModelsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewHasBindedProperties,
MvvmResults);

                TotalViewModelsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelName.Contains(".cs"),
MvvmResults);
                VMsWithViewsCount =
ViewsWithViewModelsCount;
                VMsWithRefsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelHasReferences,
MvvmResults);
                BindableVMsCount =
AnalyzedMvvmFileHelper.CalculateMvvmPropsCou
nt((item) => item.ViewModelIsBindable,
MvvmResults);



_appFilesService.SaveAnalyzedMvvmInfo(_appFiles
Service.DirectoryPath + "/AnalysedMvvmInfo.txt",
MvvmResults);

_appFilesService.SaveAnalyzedMvvmFilesContent(_
appFilesService.DirectoryPath +
"/AnalysedMvvmFiles.txt", MvvmResults);
            }
        }
    }
```

```
}
```

ViewModelBase.cs

```
using System.Windows.Input;
using Prism.Mvvm;
using Prism.Navigation;
using Xamarin.Forms;

public abstract class ViewModelBase :
BindableBase, INavigationAware, IDestructible
{
   protected INavigationService NavigationService {
get; }

   public ViewModelBase(INavigationService
navigationService)
   {
      NavigationService = navigationService;

      BackCommand = new
Command(OnBackCommand);
   }

   public ICommand BackCommand { get; }

   // INavigationAware
   public virtual void
OnNavigatedFrom(INavigationParameters
parameters)
   {
   }

   // INavigationAware
   public virtual void
OnNavigatedTo(INavigationParameters parameters)
   {
   }

   // IDestructible
   public virtual void Destroy()
   {
   }

   protected virtual async void OnBackCommand()
   {
      await NavigationService.GoBackAsync();
   }
}
```

2.4 Реалізація інтерфейсу користувача

AnalyzerSettingsView.xaml

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage
   xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/
xaml"

xmlns:local="clr-namespace:GraduateWork.Converte
rs"

x:Class="GraduateWork.Views.AnalyzerSettingsVie
w">

   <ContentPage.Resources>
      <ResourceDictionary>
         <local:AppSizeToStringConverter
x:Key="AppSizeToString" />
         <local:IntToStringConverter
x:Key="IntToString" />
      </ResourceDictionary>
   </ContentPage.Resources>

   <StackLayout>
```

```
    <!--App type and app folder-->
    <Grid Margin="15, 15, 15, 10">
      <Grid.RowDefinitions>
        <RowDefinition Height=".5*"/>
        <RowDefinition Height=".5*"/>
        <RowDefinition Height=".5*"/>
        <RowDefinition Height=".5*"/>
      </Grid.RowDefinitions>

      <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>

      <!-- Project type -->
      <Label Text="App type"
          FontSize="14"
          Grid.Row="0"
          Grid.Column="0"
          Margin="0"/>

      <Picker Grid.Row="0"
           Grid.Column="1"
           ItemsSource="{Binding AppTypeList}"
           SelectedItem="{Binding
SelectedAppType}"
           Margin="0"/>

      <!-- Project folder -->
      <Label Text="Portable project directory"
          FontSize="14"
          Grid.Row="1"
          Grid.Column="0"
          Margin="0"/>

      <Entry Grid.Row="1"
          Grid.Column="1"
          Grid.RowSpan="3"
          Text="{Binding ProjectDirectoryPath}"
          FontSize="15"/>

    </Grid>

    <BoxView BackgroundColor="Gray"
         HeightRequest=".5"
         Margin="15, 0, 15, 0"/>

    <!-- Project Settings -->
    <Grid Margin="15, 10, 15, 10">
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>

      <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>

       <!-- MVP (Minimum Vaiable Product) -->
      <Label Text="MVP (Minimum Vaiable
Product)"
          FontSize="14"
          Grid.Row="0"
          Grid.Column="0"
          Margin="0"/>

      <CheckBox Grid.Row="0"
           Grid.Column="1"
           Margin="0"/>

      <!-- Project size -->
      <Label Text="App size"
           FontSize="14"
           Grid.Row="1"
           Grid.Column="0"
           Margin="0"/>

      <Picker Grid.Row="1"
          Grid.Column="1"
          ItemsSource="{Binding AppSizeList}"
          SelectedItem="{Binding
SelectedAppSize, Converter={StaticResource
AppSizeToString}}"/>

      <!-- Time to develop -->
      <Label Text="Time to develop"
           FontSize="14"
           Grid.Row="2"
           Grid.Column="0"
           Margin="0"/>

      <Picker Grid.Row="2"
          Grid.Column="1"
          ItemsSource = "{Binding
DevelopTimeList}"
          SelectedItem="{Binding
SelectedDevelopTime}"/>
```

```
        <!-- App budget -->
        <Label Text="App budget ($)"
               FontSize="14"
               Grid.Row="3"
               Grid.Column="0"
               Margin="0"/>

        <Picker Grid.Row="3"
                Grid.Column="1"
                ItemsSource="{Binding
AppBudgetList}"
                SelectedItem="{Binding
SelectedAppBudget, Converter={StaticResource
IntToString}}"/>

    </Grid>

    <BoxView BackgroundColor="Gray"
             HeightRequest=".5"
             Margin="15, 0, 15, 0"/>

    <!-- Architecture analyzer settings -->
    <Grid Margin="15, 10, 15, 10">
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>

      <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width=".3*"/>
        <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>

      <!-- Analyze business logic -->
      <Label Text="Analyze business logic"
             FontSize="14"
             Grid.Row="0"
             Grid.Column="0"
             Margin="0"/>

      <CheckBox Grid.Row="0"
                Grid.Column="1"
                Margin="0"/>

      <!-- Analyze MVVM implementation -->
      <Label Text="Analyze MVVM
implementation"
             FontSize="14"
             Grid.Row="1"
             Grid.Column="0"
             Margin="0"/>

      <CheckBox Grid.Row="1"
                Grid.Column="1"
                Margin="0"/>

      <!-- Analyze DI implementation -->
      <Label Text="Analyze DI implementation"
             FontSize="14"
             Grid.Row="2"
             Grid.Column="0"
             Margin="0"/>

      <CheckBox Grid.Row="2"
                Grid.Column="1"
                Margin="0"/>

      <!-- Analyze IoC implementation -->
      <Label Text="Analyze IoC implementation"
             FontSize="14"
             Grid.Row="3"
             Grid.Column="0"
             Margin="0"/>

      <CheckBox Grid.Row="3"
                Grid.Column="1"
                Margin="0"/>

      <!-- Save PDF -->
      <Label Text="Save in a result file"
             FontSize="14"
             Grid.Row="4"
             Grid.Column="0"
             Margin="0"/>

      <CheckBox Grid.Row="4"
                Grid.Column="1"
                Margin="0"/>

    </Grid>

    <BoxView BackgroundColor="Gray"
             HeightRequest=".5"
             Margin="15, 0, 15, 0"/>

    <Grid Margin="15, 10, 15, 10">
```

```
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>

        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width=".3*"/>
            <ColumnDefinition Width=".3*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>

        <Button Text="Analyze App"
                TextColor="WhiteSmoke"
                Padding="10"
                Grid.Row="0"
                Grid.Column="0"
                Command="{Binding
AnalyzeAppCommand}"
                IsEnabled="{Binding
ButtonsEnabled}"/>

        <Button Text="Show loaded files"
                TextColor="WhiteSmoke"
                Padding="10"
                Grid.Row="0"
                Grid.Column="1"
                Command="{Binding
ShowFilesCommand}"
                IsEnabled="{Binding
ButtonsEnabled}"/>

            <!--<Button Text="Clear"
                TextColor="WhiteSmoke"
                Padding="10"
                Grid.Row="1"
                Grid.Column="0"
                IsEnabled="{Binding
ButtonsEnabled}"/>-->

        </Grid>

    </StackLayout>

</ContentPage>
```

AnalyzerSettingsView.cs

```
using System;
using System.Collections.Generic;

using Xamarin.Forms;

namespace GraduateWork.Views
{
    public partial class AnalyzerSettingsView :
ContentPage
    {
        public AnalyzerSettingsView()
        {
            InitializeComponent();
        }
    }
}
```

BaseContentPage.cs

```
using Xamarin.Forms;

namespace GraduateWork.Views
{
    public class BaseContentPage : ContentPage
    {
        public BaseContentPage()
        {
            NavigationPage.SetHasBackButton(this,
false);
        }
    }
}
```

FilesListView.xaml

```
<?xml version="1.0" encoding="UTF-8"?>
<local:BaseContentPage
```

```
   xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/
xaml"

xmlns:local="clr-namespace:GraduateWork.Views"

xmlns:converters="clr-namespace:GraduateWork.Co
nverters"
   x:Class="GraduateWork.Views.FilesListView">

   <ContentPage.Resources>
      <ResourceDictionary>
         <converters:IntToStringConverter
x:Key="IntToString" />
      </ResourceDictionary>
   </ContentPage.Resources>

   <ContentPage.Content>

      <StackLayout>
         <Grid Margin="5, 10, 0, 0">

            <Grid.ColumnDefinitions>
               <ColumnDefinition Width=".3*"/>
               <ColumnDefinition Width=".7*"/>
            </Grid.ColumnDefinitions>

            <StackLayout Grid.Column="0">
               <Button Text="Navigate Back"
                     TextColor="WhiteSmoke"
                     Command="{Binding
BackCommand}"
                     Padding="10"
                     VerticalOptions="End"
                     Margin="0"/>

               <ListView ItemsSource="{Binding
AppFiles}"
                      SelectedItem="{Binding
SelectedAppFile}"
                      BackgroundColor="Transparent">
               </ListView>
            <Grid HeightRequest="80">

               <StackLayout Padding="20"
                        BackgroundColor="Black"
                        Opacity="0.3"/>

               <StackLayout
Orientation="Horizontal"
                        WidthRequest="300"
                        Margin="0, 0, 0, 5">

                  <Label Text="Lines count:"
                       TextColor="LightSkyBlue"
                       FontSize="15"/>

                  <Label Text="{Binding
FileLinesCount,
                        Converter={StaticResource
IntToString}}"
                       FontSize="15"/>
               </StackLayout>
            </Grid>
         </StackLayout>

         <StackLayout Orientation="Horizontal"

HorizontalOptions="FillAndExpand"
                     Padding="20"
                     BackgroundColor="Black"
                     Opacity="0.3"
                     Grid.Column="1"/>

         <ScrollView Grid.Column="1">
            <Label Text="{Binding FileContent}"
                 FontSize="14"
                 VerticalOptions="Start"/>
         </ScrollView>

      </Grid>
   </StackLayout>
  </ContentPage.Content>
</local:BaseContentPage>
```

FilesListView.cs

```
using System;
using System.Collections.Generic;

using Xamarin.Forms;

namespace GraduateWork.Views
{
```

```
   public partial class FilesListView :
BaseContentPage
   {
      public FilesListView()
      {
         InitializeComponent();
      }
   }
}
```

PatternsAnalyzeResultView.cs

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace GraduateWork.Views
{

[XamlCompilation(XamlCompilationOptions.Compil
e)]
   public partial class PatternsAnalyzeResultView :
BaseContentPage
   {
      public PatternsAnalyzeResultView()
      {
         InitializeComponent();
      }
   }
}
```

PatternsAnalyzeResultView.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<views:BaseContentPage
   xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/
xaml"

xmlns:views="clr-namespace:GraduateWork.Views"

x:Class="GraduateWork.Views.PatternsAnalyzeResu
ltView">

   <StackLayout>

      <!-- MVVM, DI, IoC -->
      <Grid Margin="15, 10, 15, 0">
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width=".3*"/>
            <ColumnDefinition Width=".3*"/>
            <ColumnDefinition Width=".3*"/>
            <ColumnDefinition Width=".3*"/>
         </Grid.ColumnDefinitions>

         <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
         </Grid.RowDefinitions>

        <Button Text="Back"
              TextColor="WhiteSmoke"
              Grid.Column="0"
              Command="{Binding BackCommand}"
              WidthRequest="100"/>

        <Button Text="MVVM"
              TextColor="WhiteSmoke"
              Grid.Column="1"
              Command="{Binding value}"
              Padding="10"
              WidthRequest="200"/>

        <Button Text="DI"
              TextColor="WhiteSmoke"
              Grid.Column="2"
              Command="{Binding value}"
              WidthRequest="200"
              IsEnabled="False"/>

        <Button Text="IoC"
              TextColor="WhiteSmoke"
              Grid.Column="3"
              Command="{Binding value}"
              WidthRequest="200"
              IsEnabled="False"/>
      </Grid>

      <ListView Grid.Column="0"
            ItemsSource="{Binding MvvmResults}"
            BackgroundColor="Transparent"
            SeparatorColor="Transparent"
            IsEnabled="False">
```

```
        <ListView.Header>
          <Grid Margin="0, 10, 0, 10">
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width=".3*"/>
              <ColumnDefinition Width=".3*"/>
              <ColumnDefinition Width=".3*"/>
              <ColumnDefinition Width=".3*"/>
              <ColumnDefinition Width=".3*"/>
              <ColumnDefinition Width=".3*"/>
            </Grid.ColumnDefinitions>

            <Grid.RowDefinitions>
              <RowDefinition Height="40"/>
            </Grid.RowDefinitions>

            <StackLayout
BackgroundColor="DimGray"
                         Grid.ColumnSpan="6"
                         Margin="30, 0, 18, 0"/>
            <!--Opacity="0.8"-->

            <Label Text="View"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   FontSize="15"
                   TextColor="LightSkyBlue"
                   Grid.Column="0"/>

            <Label Text="Has References"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   TextColor="Bisque"
                   FontSize="15"
                   Grid.Column="1"/>

            <Label Text="Has Binded Props"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   TextColor="Bisque"
                   FontSize="15"
                   Grid.Column="2"/>

            <Label Text="ViewModel"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   FontSize="15"
                   TextColor="LightSkyBlue"
                   Grid.Column="3"/>

            <Label Text="Has references"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   TextColor="Bisque"
                   FontSize="15"
                   Grid.Column="4"/>

            <Label Text="Is bindable"
                   HorizontalOptions="Center"
                   VerticalTextAlignment="Center"
                   TextColor="Bisque"
                   FontSize="15"
                   Grid.Column="5"/>
          </Grid>
        </ListView.Header>

        <ListView.ItemTemplate>
          <DataTemplate>
            <ViewCell>
              <Grid Margin="0, 0, 0, 10">
                <Grid.ColumnDefinitions>
                  <ColumnDefinition
Width=".3*"/>
                  <ColumnDefinition
Width=".3*"/>
                  <ColumnDefinition
Width=".3*"/>
                  <ColumnDefinition
Width=".3*"/>
                  <ColumnDefinition
Width=".3*"/>
                  <ColumnDefinition
Width=".3*"/>
                </Grid.ColumnDefinitions>

                <StackLayout
BackgroundColor="Black"
                             Opacity="0.1"
                             Grid.ColumnSpan="6"/>

                <Label Text="{Binding
ViewName}"
                       FontSize="14"
                       TextColor="LightSkyBlue"

VerticalTextAlignment="Center"
                       Grid.Column="0"/>

                <Label Text="{Binding
ViewHasReferences}"

HorizontalTextAlignment="Center"
```

```
VerticalTextAlignment="Center"
                        TextColor="Bisque"
                        FontSize="14"
                        Grid.Column="1"/>

                    <Label Text="{Binding
ViewHasBindedProperties}"
                        TextColor="Bisque"

VerticalTextAlignment="Center"

HorizontalTextAlignment="Center"
                        FontSize="14"
                        Grid.Column="2"/>

                    <Label Text="{Binding
ViewModelName}"
                        FontSize="14"

VerticalTextAlignment="Center"
                        TextColor="LightSkyBlue"
                        Grid.Column="3"/>

                    <Label Text="{Binding
ViewModelHasReferences}"

HorizontalTextAlignment="Center"

VerticalTextAlignment="Center"
                        TextColor="Bisque"
                        FontSize="14"
                        Grid.Column="4"/>

                    <Label Text="{Binding
ViewModelIsBindable}"

HorizontalTextAlignment="Center"

VerticalTextAlignment="Center"
                        TextColor="Bisque"
                        FontSize="14"
                        Grid.Column="5"/>

                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

<Grid Margin="30, 10, 0, 10"
                ColumnSpacing="0">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".1*"/>
        <ColumnDefinition Width="55"/>
        <ColumnDefinition Width=".1*"/>
        <ColumnDefinition Width="55"/>
        <ColumnDefinition Width=".1*"/>
        <ColumnDefinition Width="55"/>
        <ColumnDefinition Width=".1*"/>
        <ColumnDefinition Width="55"/>
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="35"/>
        <RowDefinition Height="35"/>
    </Grid.RowDefinitions>

    <StackLayout BackgroundColor="Black"
            Opacity="0.1"
            Grid.Row="0"
            Grid.ColumnSpan="8"
            Margin="0, 0, 18, 0"/>

    <StackLayout BackgroundColor="Black"
            Opacity="0.1"
            Grid.Row="1"
            Grid.ColumnSpan="8"
            Margin="0, 0, 18, 0"/>

    <Label Text="Total Views Count:"
        TextColor="LightSteelBlue"
        FontSize="14"
        VerticalTextAlignment="Center"
        Grid.Row="0"
        Grid.Column="0"/>

    <Label Text="{Binding
TotalViewsCount}"
            FontSize="14"
            VerticalTextAlignment="Center"
            Grid.Row="0"
            Grid.Column="1"/>

    <Label Text="Views With Refs:"
        TextColor="LightSteelBlue"
        FontSize="14"
        VerticalTextAlignment="Center"
        Grid.Row="0"
        Grid.Column="2"/>
```

```
            <Label Text="{Binding
ViewsWithRefsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="0"
                Grid.Column="3"/>

            <Label Text="Views Matched With
ViewModels:"
                TextColor="LightSteelBlue"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="0"
                Grid.Column="4"/>

            <Label Text="{Binding
ViewsWithViewModelsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="0"
                Grid.Column="5"/>

            <Label Text="Total ViewModels Count:"
                TextColor="LightSteelBlue"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="0"/>

            <Label Text="{Binding
TotalViewModelsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="1"/>

            <Label Text="ViewModels Matched With
Views:"
                TextColor="LightSteelBlue"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="2"/>

            <Label Text="{Binding
VMsWithViewsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="3"/>

            <Label Text="ViewModels With Refs:"
                TextColor="LightSteelBlue"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="4"/>

            <Label Text="{Binding
VMsWithRefsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="5"/>

            <Label Text="Bindable ViewModels: "
                TextColor="LightSteelBlue"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="6"/>

            <Label Text="{Binding
BindableVMsCount}"
                FontSize="14"
                VerticalTextAlignment="Center"
                Grid.Row="1"
                Grid.Column="7"/>
        </Grid>
    </StackLayout>

</views:BaseContentPage
```

ЗАТВЕРДЖЕНО

1116130.01215-01 13 01-ЛЗ

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Опис програми

Листів 19

2021

АНОТАЦІЯ

Документ 1116130.01215-01 13 01 «Статичний аналізатор архітектури крос-платформного програмного забезпечення. Опис програми» входить до складу документації для програми статичного аналізатору, який буде використано для аналізу наслідків використання патернів та загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS.

В цьому документі описано програмне забезпечення: загальні відомості, функціональне призначення, застосовані технічні засоби, вхідні та вихідні дані, опис логічної структури, опис інтерфейсу користувача, виклик програми.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Сконструйоване програмне забезпечення називається «Статичний аналізатор архітектури крос-платформного програмного забезпечення» та використовується розробниками програмних продуктів для аналізу архітектури створеного крос-платформного програмного забезпечення. Аналізатор розроблено із застосуванням об’єктно - орієнтованої мови програмування C#, мови Xaml для створення графічного інтерфейсу користувача та крос - платформного фреймворку Xamarin.Forms. У якості інтегрованого середовища для розробки використовувалось Visual Studio.

Програма розрахована на функціонування в операційних системах MacOS Mojave - MacOS Monterey.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Головним завданням програмного забезпечення є аналіз архітектури крос - платформного програмного забезпечення, збереження та відображення результатів аналізу архітектури.

Функціональним призначенням є:

- завантаження файлів проекту реалізації програмної системи;
- аналіз архітектури завантаженого проекту;
- збереження результатів аналізу архітектури.

Функціональними вимогами є:

- програма повинна надавати можливість перегляду завантажених файлів проекту, архітектура якого буде проаналізована;
- контент завантажених файлів проекту для аналізу може бути переглянутий, а його форматування має відповідати оригінальному форматуванні заданому при розробці в певному середовищі для розробки програмних засобів;
- налаштування аналізатору коду повинні дозволяти сконфігорувати програмне середовище під проекти різного розміру, вартості та часу розробки, а також з необхідною мовою та фреймворком, що були використані при реалізації проекту, що аналізується;
- результати аналізу архітектури програмного забезпечення мають бути представлені за допомогою сконструйованого графічного інтерфейсу користувача та збережені у результуючі файли;
- результуючі файли мають містити контент файлів, що був проаналізований та результати аналізу архітектури.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм роботи програми

На початку роботи програми необхідно завантажити файли проекту, що буде аналізуватися та задати налаштування статичному аналізатору, наприклад мова та фреймворк або увімкнути функцію збереження результатів аналізу архітектури проекту та файлів цього проекту в результуючі файли.

Після завантаження файлів проекту з’являється можливість переглянути або проаналізувати завантажені файли та їх контент.

При перегляді завантажених файлів програма відображає їх як список елементів та виділяє обраний файл. У разі вибору файлу зі списку на екран виводиться код, що описаний у цьому файлі з дотриманням форматування, що було застосоване при його реалізації.

У разі вибору функції аналізу файлу, програма опрацьовує вхідні файли, наприклад видаляє всі закоментовані блоки коду та перевіряє реалізацію деякого патерну за певними правилами його реалізації.

У випадку з архітектурним патерном MVVM, спочатку перевіряються всі файли та сортуються за їх видом, наприклад файли реалізації інтерфейсу користувача, файли логіки інтерфейсу користувача, тощо. Потім, якщо файли відповідають правилам реалізації патерну MVVM, то встановлюються пари View та ViewModel. Процес аналізу продовжується доки всі файли не буде оброблено.

Потім відбувається відображення результатів аналізу архітектури деякого проекту, а також збереження цих результатів у файли. На рисунку 3.1 схематично зображено роботу аналізатору.

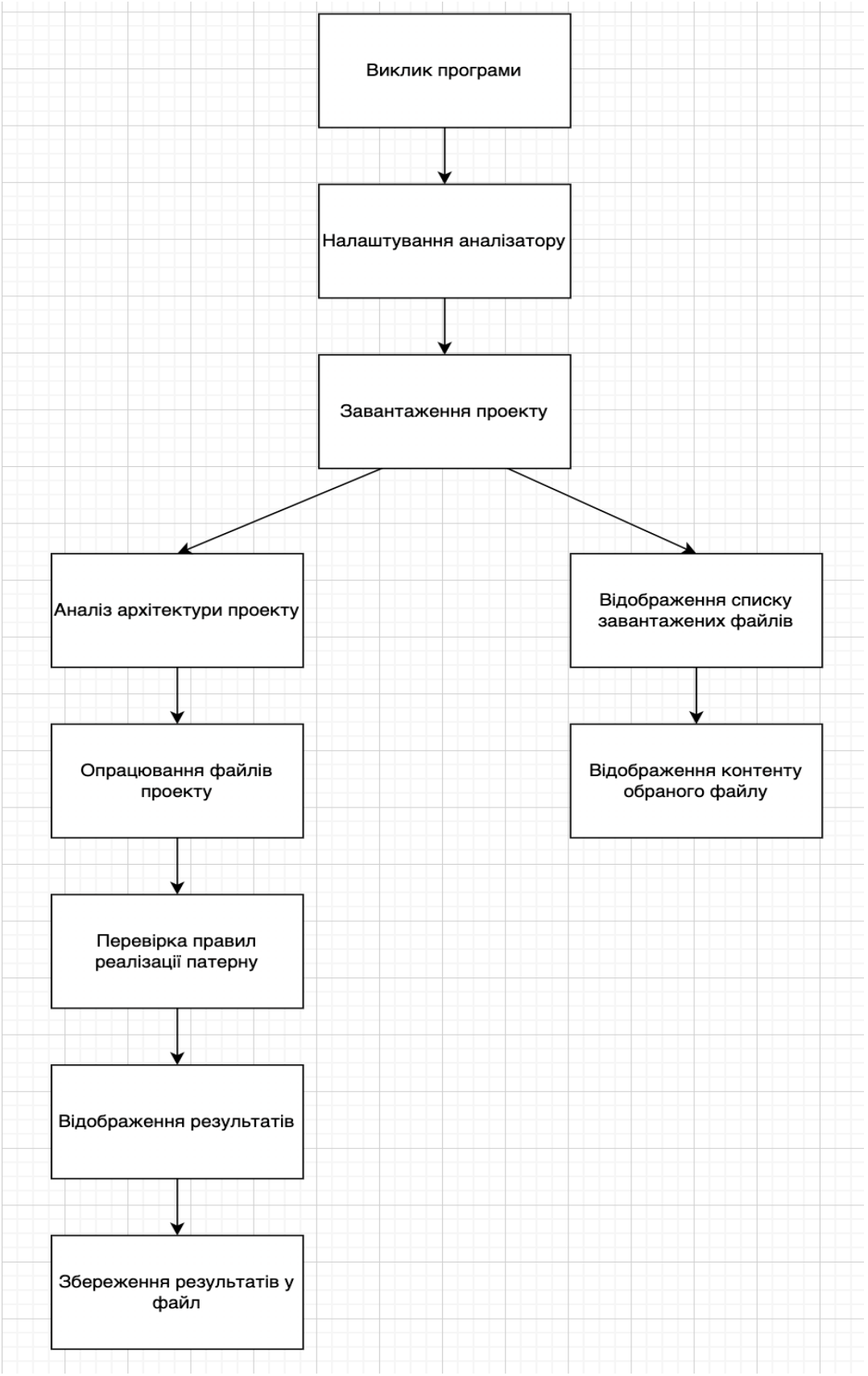


Рисунок 3.1 – Схематичне зображення роботи статичного аналізатору

3.2 Використані методи

На етапі аналізу проекту доводиться опрацьовувати велику кількість вхідних файлів, для чого використовується LinQ один з механізмів мови C#, що дозволяє опрацьовувати великі колекції файлів.

Для відображення даних на інтерфейсі користувача використовується механізм Binding, що повідомляє графічний інтерфейс про те, що необхідно оновитися завдяки події NotifyPropertyChanged. Програма складається з модулів, що описує патерн MVVM.

3.3 Структура програми з описом функцій

Програма розроблена таким чином, що вона складається з великих модулів, які в свою чергу складаються з модулів меншого розміру, а ті з ще менших модулів. Дивлячись на систему з найвищого рівню абстракції над усіма модулями, то можна виділити наступні:

– SettingsModule - модуль відповідальний за налаштування аналізатору та завантаження проекту, який необхідно проаналізувати. Підмодулі з яких складається цей SettingsModule можна побачити на рис. 3.2

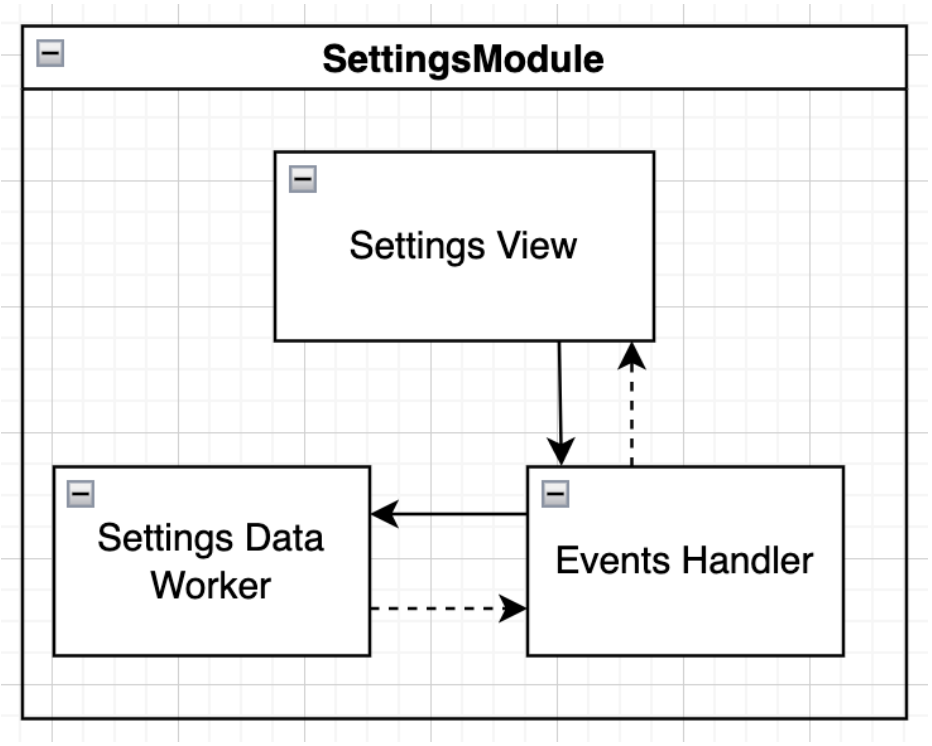


Рисунок 3.2 – Підмодулі модулю SettingsModule

– AnalyzerModule – модуль відповідальний за процес аналізу архітектури проекту. Його підмодулі показано на рис. 3.3

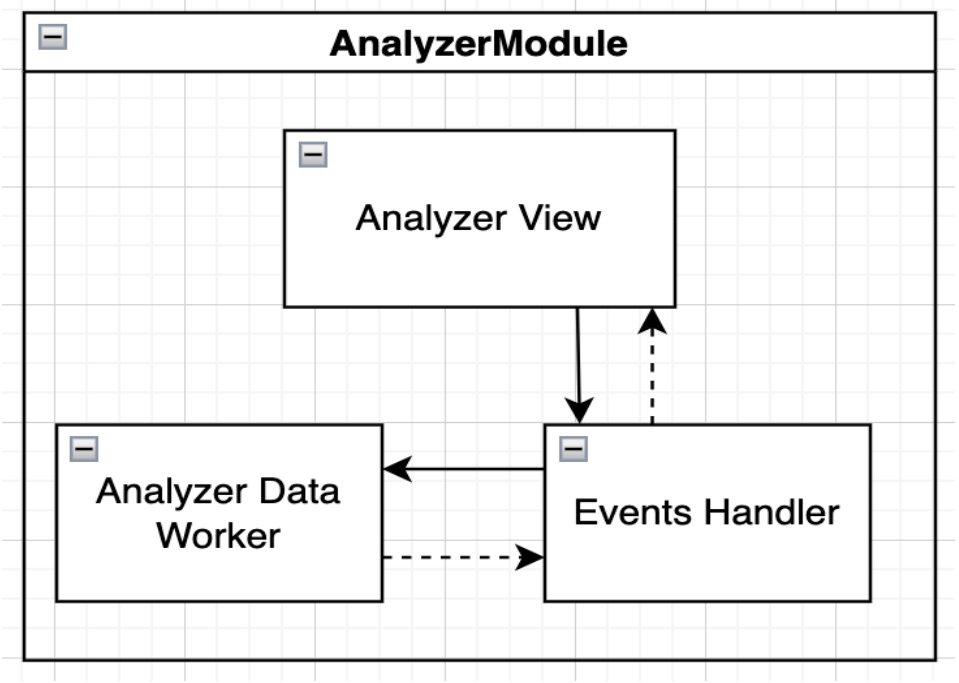


Рисунок 3.3 – Підмодулі модулю AnalyzerModule

– FilesViewerModule – модуль відповідальний за відображення завантажених файлів проекту та їх контенту. Підмодулі для нього зображено на рис. 3.4

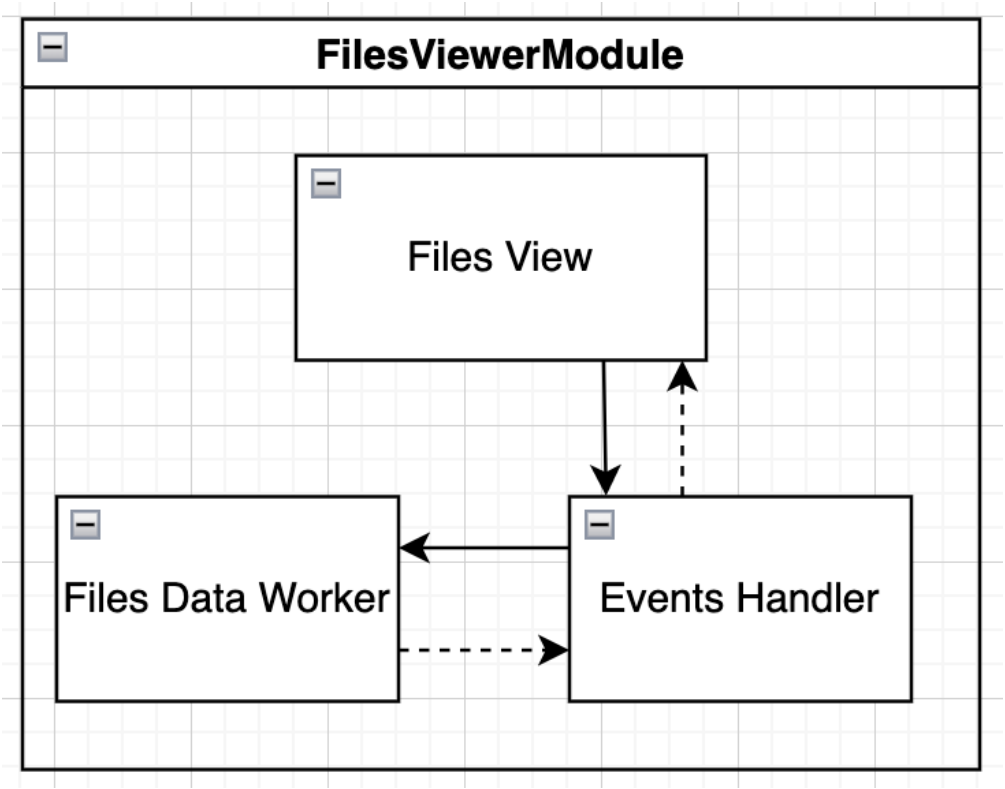


Рисунок 3.3 – Підмодулі модулю FilesViewerModule

3.4 Залежність від іншого програмного забезпечення

Розроблене програмне забезпечення може використовуватися незалежно від інших програм за винятком операційної системи, але аналізатор може генерувати файли з результатами аналізу архітектури певного проекту. В такому разі з’являється необхідність використання текстового редактору або редактору коду, наприклад Visual Studio Code. Сгенеровані текстові файли представлені в форматі «.txt», що надає можливість використовувати будь-який текстовий редактор для перегляду цього файлу.

4. ЗАСТОСОВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання статичного аналізатору архітектури крос-платформного програмного забезпечення необхідна наступна мінімальна конфігурація персонального комп’ютеру:

- процессор: Intel Core i5-3210M, 2 ядра, 2.5 ГГц;
- оперативна пам'ять: 4 GB DDR2;
- вільний дисковий простір 100 Mb на накопичувачі;
- роз’єм USB для завантаження додаткового програмного забезпечення;
- операційна система: macOS Mojave;
- монітор роздільна здатність якого не менше ніж 1024x768;
- клавіатура та «миша».

5. ВИКЛИК ПРОГРАМИ

Щоб почати користуватися програмним засобом необхідно викликати файл типу .exe, який називається «GraduateWork.exe». Після короткого завантаження програми на робочому столі з’явиться інтерфейс користувача для взаємодії з функціоналом програми.

6. ВХІДНІ ДАНІ

Вхідними даними є деякий проект реалізації крос-платформного програмного забезпечення, а точніше його файли з вихідним кодом. Формат файлів залежить від типу проекту та мови програмування яка використовувалася при розробці. Наприклад це можуть бути файли формату «.cs», якщо програмний додаток реалізований з використанням мови програмування C#. Приклад таких файлів зображено на рис. 6.1

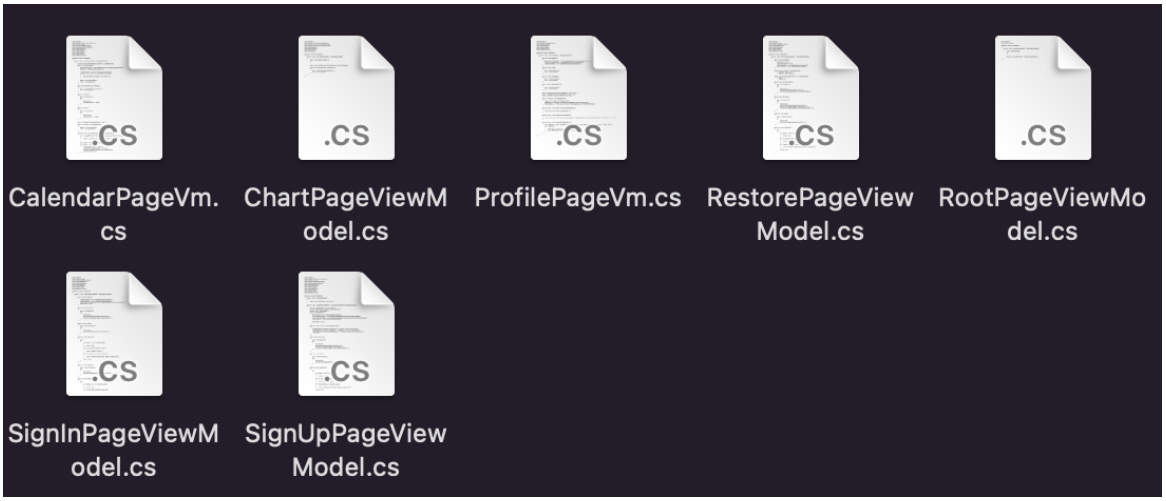


Рисунок 6.1 – Приклад вхідних даних

7. ВИХІДНІ ДАНІ

Вихідними файлами по закінченню роботи програми є таблиця результатів аналізу та файли з цими ж результатами та вихідним кодом та назвою файлів, що були проаналізовані. Приклади вихідних файлів продемонстровано на рис. 7.1 – 7.3.

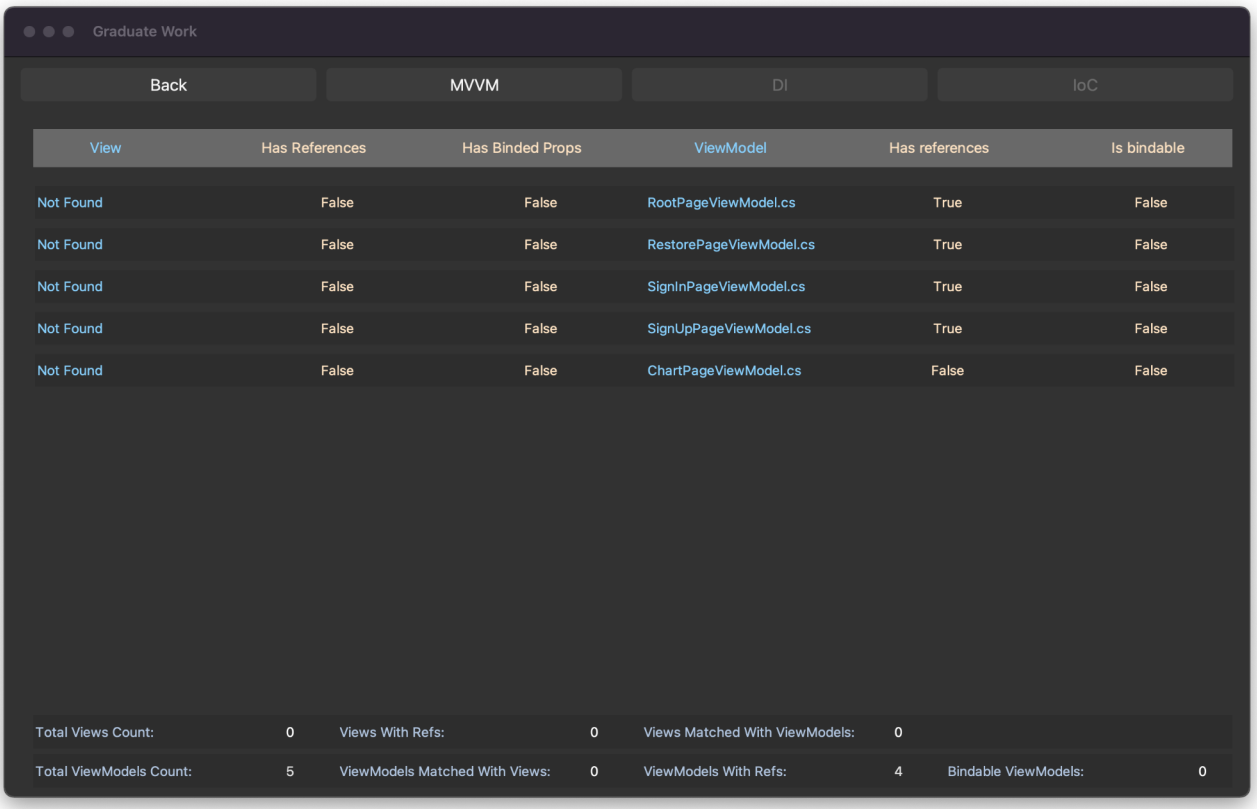


Рисунок 7.1 – Екрана форма вихідних даних у виді таблиці з результатами аналізу

Вихідні дані, які зображені завдяки елементам інтерфейсу користувача також можуть бути представлені у виді текстових документів з результатами аналізу архітектури програмного забезпечення створеного під операційні системи Android та IOS. Текстові файли будуть представлені в кількості двох, з відповідним вмістом.

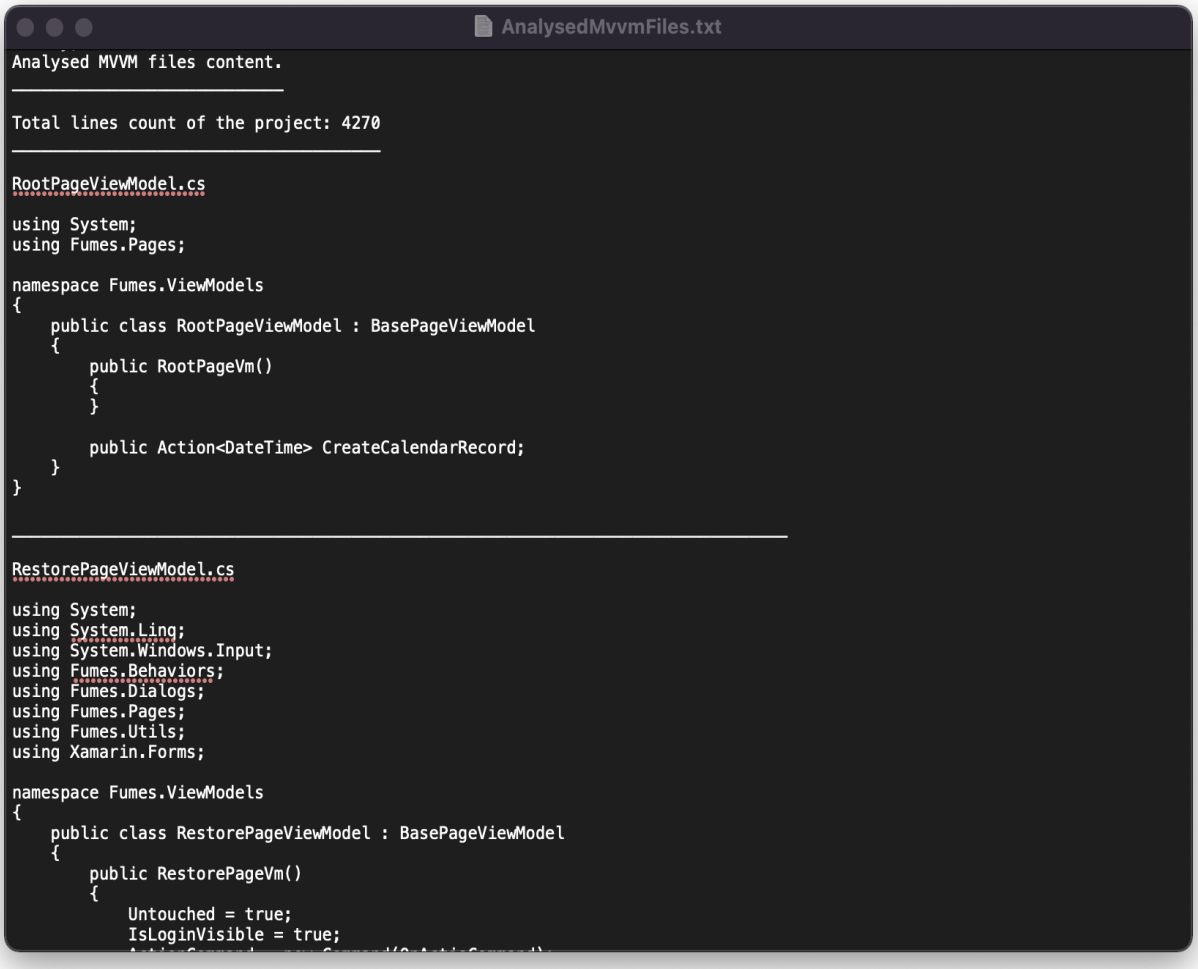


Рисунок 7.2 – Вихідні дані у виді файлу з кодом проаналізованих файлів

На представленному рисунку показано вихідні дані, що стосуються проаналізованих файлів програмного продукту, а точніше їх назви, вихідного коду та загальної кількості строк коду. Розмір вихідних даних залежить від розміру програмного забезпечення, що аналізується статичним аналізатором, тому від проекту до проекту може бути різним.

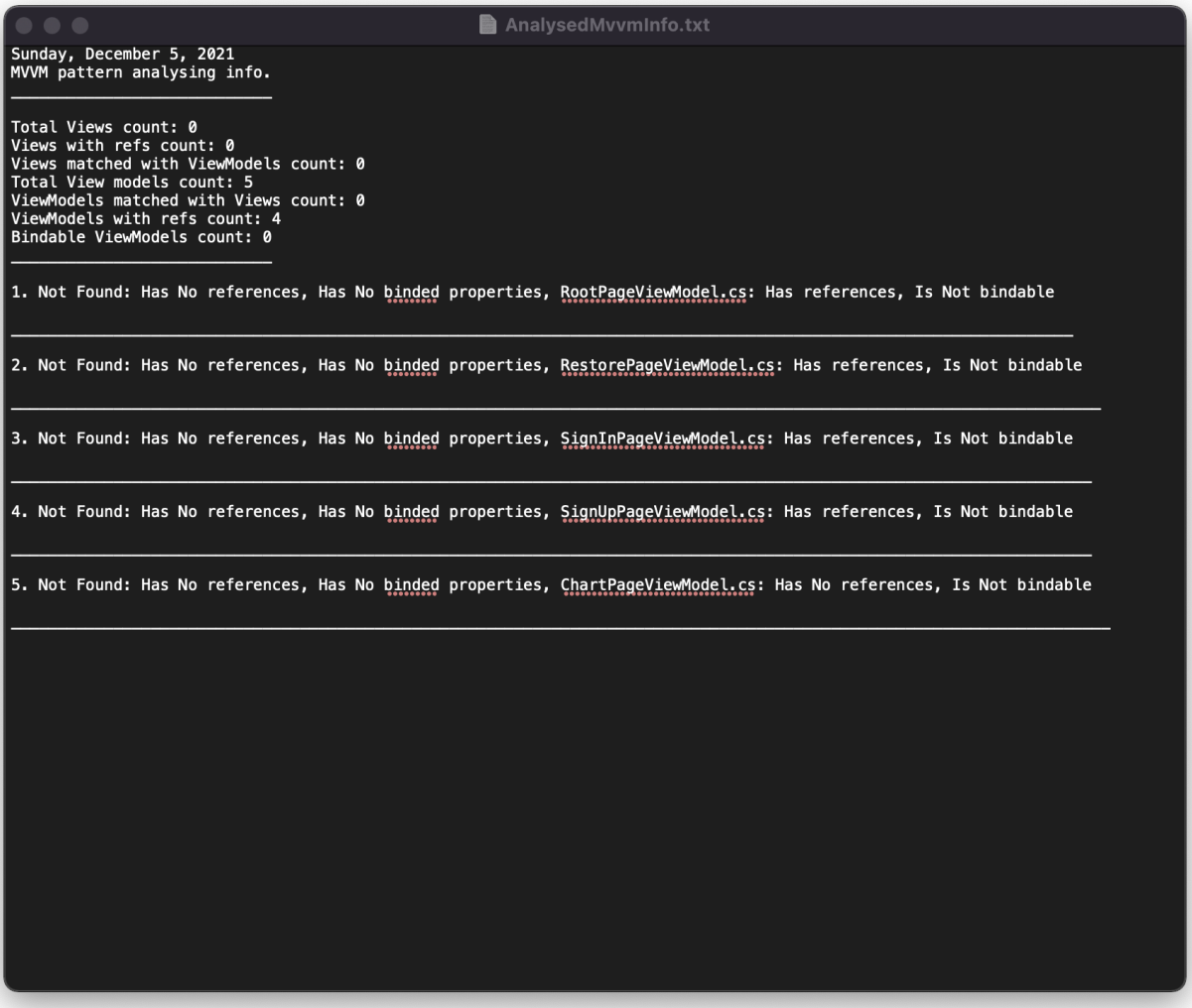


Рисунок 7.2 – Вихідні дані у виді файлу з результатами аналізу проекту

8. ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

В програмі присутні три сторінки для взаємодії з функціоналом аналізатору.

Сторінка налаштувань статичного аналізатору представляє графічний інтерфейс з різними графічними елементами для налаштування процесу аналізу, завантаження або збереження файлів тощо. Цю сторінка показано на рис. 8.1. Користувач може обрати лише необхідні налаштування, це надає йому можливість повного конфігурування процесу аналізу.

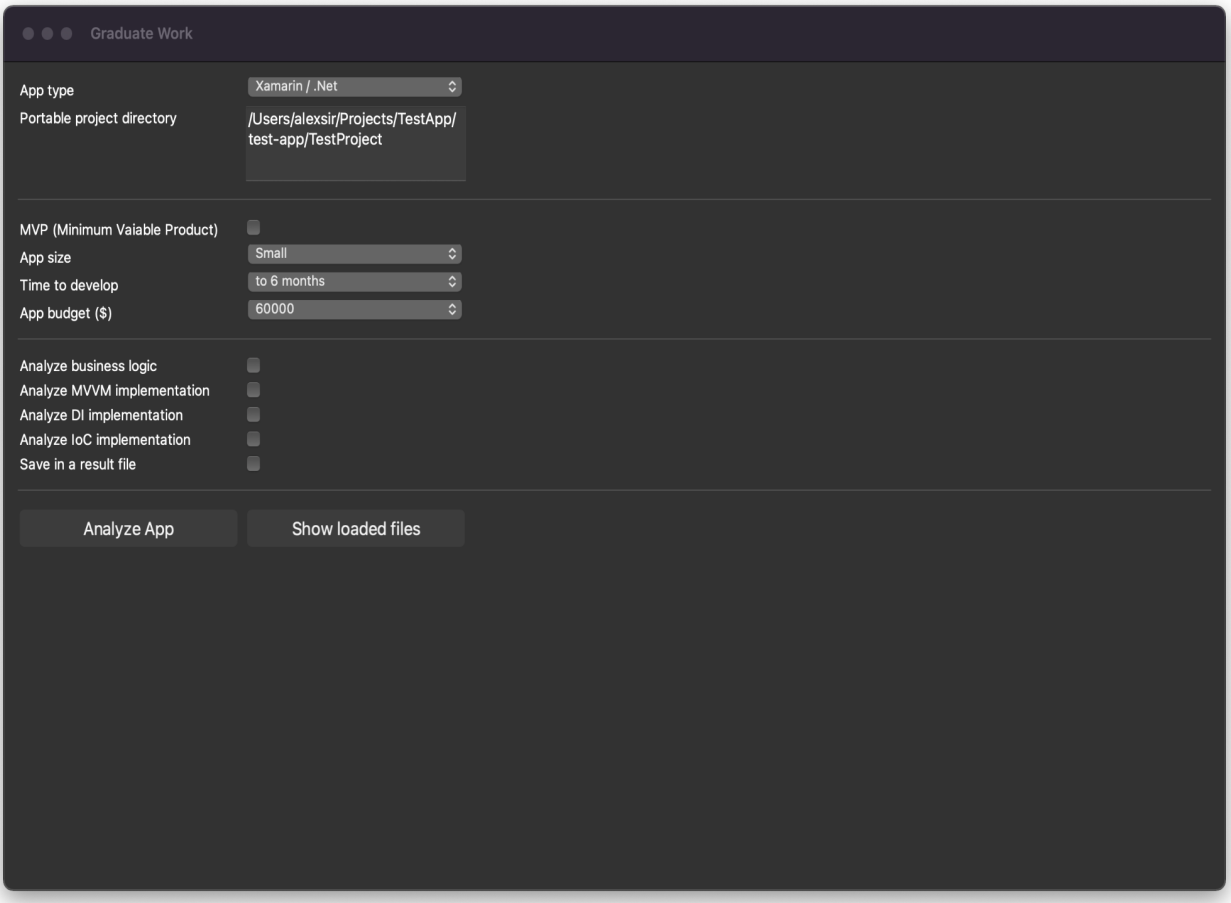


Рисунок 8.1 – Екранна форма з опціями

Сторінка для перегляду завантажених файлів та вихідного коду, який вони містять зображено на рис. 8.2. Завдяки цьому графічному інтерфейсу можна переглянути вихідний код деяких файлів та зрозуміти чому аналізатор показав ті чи інші результати перевірки архітектури, що стосується цих файлів.

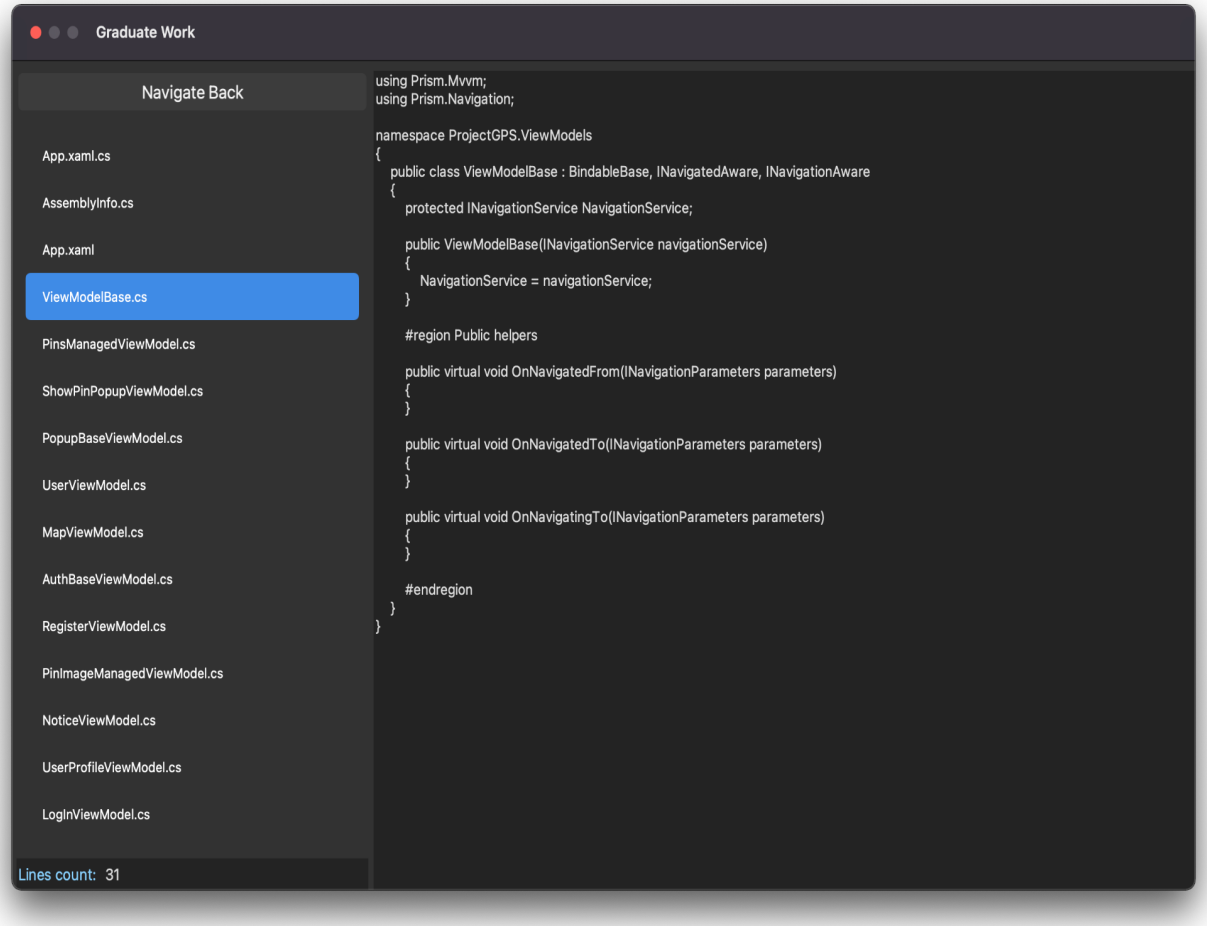


Рисунок 8.2 – Екранна форма для перегляду файлів, що аналізуються

Сторінка з результатами аналізу архітектури показана на рис. 8.3. Дана сторінка представляє себою таблицю з даними стосовно реалізації того, чи іншого патерну та додаткового надає дані про різні файли програмної системи, що аналізується статичним аналізатором архітектури. Додаткові дані подаються внизу сторінку програмного додатку та згруповані за типом.

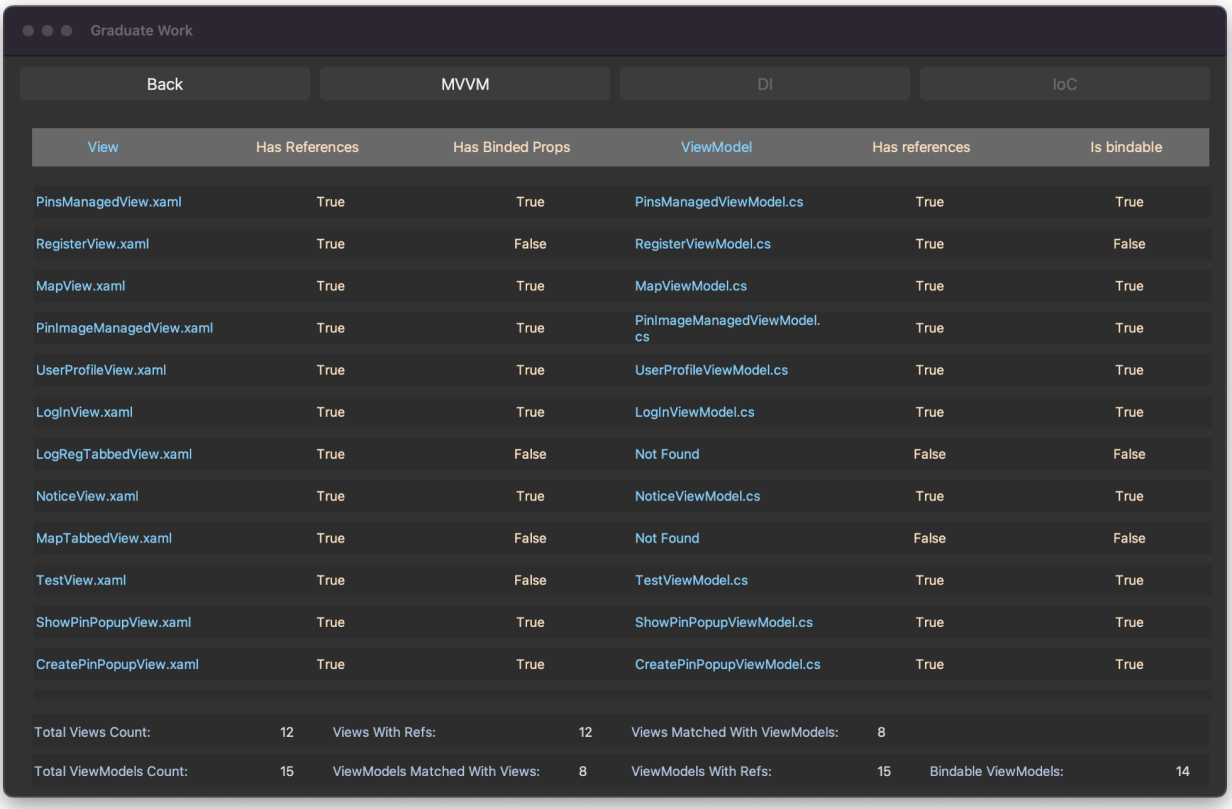


Рисунок 8.3 – Екранна форма з результатами аналізу архітектури

Досягти такої реалізації інтерфейсу користувача вдалося завдяки елементам, що є складовою фреймворку Xamarin.Froms.

9. ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Послідовність дій яку необхідно виконати при роботі з статичним аналізатором архітектури крос-платформних додатків:

- викликати програму;
- вказати директорію проекту, який необхідно проаналізувати;
- змінити налаштування аналізу (опціонально).

Аналіз архітектури проекту та перегляд файлів проекту може бути здійснений в будь-якій послідовності. Нижче описані кроки однієї з можливих послідовностей:

- клікнути по кнопці «Analyze App»;
- у відкритому вікні переглянути результати аналізу та отримані дані;
- клікнути по кнопці «Back;
- після відкриття головної сторінки клікнути по кнопці «Show loaded files»;
- переглянути необхідні файли;

Перелік дій описаних вище може бути повторений декілька разів для роботи з одним і тим же або іншим проектом.

ЗАТВЕРДЖЕНО

1116130.01215-01 І3 01-ЛЗ

СТАТИЧНИЙ АНАЛІЗАТОР АРХІТЕКТУРИ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Керівництво користувача. Керівництво з використання статичного аналізатору архітектури крос-платформного забезпечення.

Листів 6

2021

АНОТАЦІЯ

Документ 1116130.01215-01 І3 01 «Статичний аналізатор архітектури крос-платформного програмного забезпечення. Керівництво користувача Керівництво з використання статичного аналізатору архітектури крос-платформного забезпечення» є складовою документації на програму та входить в документацію на проект.

В документі описано призначення програмного забезпечення, умови його застосування, підготовку перед початком роботи, базові операції.

ЗМІСТ

ВСТУП

Програмне забезпечення під назвою «Статичний аналізатор архітектури крос-платформного програмного забезпечення» буде корисним для розробників програмних продуктів як початкового так і високого рівня при аналізі архітектури програмного забезпечення.

Даний програмний продукт потребує встановленої операційної системи macOS Mojave - Monterey на персональному ПК, а користувачам необхідно звернути увагу на керівництво користувача.

1 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Функціональне призначення програмного додатку

Функціональним призначення є:

- аналіз архітектури крос-платформного програмного забезпечення;
- збереження результатів аналізу архітектури.

1.2 Вимоги до технічних засобів

Щоб використовувати статичний аналізатор архітектури крос-платформного програмного забезпечення необхідно мати комп’ютер, який відповідає таким вимогам:

- процессор: Intel Core i5-3210M, 2 ядра, 2.5 ГГц;
- оперативна пам'ять: 4 GB DDR2;
- вільний дисковий простір 100 Mb на накопичувачі;
- роз’єм USB для завантаження додаткового програмного забезпечення;
- операційна система: macOS Mojave - Monterey;
- монітор роздільна здатність якого не менше ніж 1024x768;
- клавіатура та «миша».

2 ПІДГОТОВКА ПЕРЕД ПОЧАТКОМ РОБОТИ

Для того щоб користуватись програмним додатком пройдіть етап підготовки перед початком роботи.

Перш за все необхідно мати проект реалізації програмного забезпечення, яке потрібно проаналізувати. Зверніть увагу на те, що мова яка використовувалась при написанні програмного продукту, архітектура якого підлягає аналізу повинна підтримуватись статичним аналізатором архітектури. Після запуску аналізатору вкажіть шлях до проекту і переконайтеся, що вказаний шлях насправді до того проекту, який потрібен, тобто переконайтеся у відсутності помилок.

Крім того, при вказанні директорії проекту пам’ятайте, що назви папок або підпапок повинні бути складені з використанням латинського алфавіту.

3 ОПИС БАЗОВИХ ОПЕРАЦІЙ

Статичний аналізатор підтримує такі базові операції:

– завантаження файлів для аналізу - файли з обраного проекту будуть завантажені і підготовлені до аналізу;
– налаштування процесу аналізу - встановлення додаткових опцій або функціоналу перед аналізом;
– перегляд аналазуючих файлів - для встановлення відповідності між тим, який код написаний в ньому і між тим, які результати було отримано після аналізу стосовно якогось файлу, а не тільки архітектури взагалом;
– збереження файлів - збереження результатів аналізу програмного забезпечення у текстові файли. Для цього будуть створені два файли, де один містить код і назви проаналізованих файлів, а інший інформацію про стан архітектури, той що показано в інтерфейсі користувача.