НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
інформаційних систем та технологій

_____ Швиденко М.З., зав. каф., к.е.н. засл. проф
(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Аналіз використання графічних інтерфейсів у мобільних та десктопних додатках»

Спеціальність 122 «Комп'ютерні науки»

Гарант освітньої програми

д.е.н. доцент _____ Руденський Р.А.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

д.філос.н(спец.122), ст.викл. _____ Золотуха Р.А.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Єрофеєва А.В.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
інформаційних систем та технологій

_____/ Швиденко М.З., зав. Каф., к.е.н. засл. проф /

підпис

“ _ ” _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студент Єрофеєва Анастасія Віталіївна

Спеціальність 122 «Комп’ютерні науки»

Тема бакалаврської кваліфікаційної роботи: Аналіз використання графічних інтерфейсів у мобільних та десктопних додатках

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2246 “С”

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання “16” 12 2024р.

Керівник бакалаврської кваліфікаційної роботи

д.філос.н(спец.122), ст.викл. _____ Золотуха Р.А.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Єрофеєва А.В.

(підпис) (ПІБ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

– ІСППР — інформаційна система підтримки прийняття рішень

– БД — база даних

– API — прикладний програмний інтерфейс (Application Programming Interface)

– GUI — графічний інтерфейс користувача (Graphical User Interface)

– HTTP — протокол передавання гіпертексту (HyperText Transfer Protocol)

– JSON — формат обміну даними JavaScript Object Notation

– JS — мова програмування JavaScript

– Node.js — програмна платформа для виконання JavaScript-коду на стороні сервера

– Express.js — фреймворк для створення серверних веб-додатків на основі Node.js

– REST — архітектурний стиль побудови API (Representational State Transfer)

– MVC — архітектурна модель «Model–View–Controller»

– SQL — мова структурованих запитів (Structured Query Language)

– UI — інтерфейс користувача (User Interface)

– UX — користувацький досвід (User Experience)

– HTML — мова розмітки гіпертексту (HyperText Markup Language)

– CSS — каскадні таблиці стилів (Cascading Style Sheets)

– NPM — менеджер пакетів для середовища Node.js (Node Package Manager)

– CRUD — базові операції з даними: створення (Create), читання (Read), оновлення (Update), видалення (Delete)

– SPA — односторінковий вебзастосунок (Single Page Application)

– JWT — стандарт маркерів доступу (JSON Web Token)

ВСТУП

У сучасному інформаційному просторі графічний інтерфейс користувача (Graphical User Interface, GUI) є ключовим компонентом ефективної взаємодії між людиною та програмним забезпеченням. Зі стрімким розвитком цифрових технологій, зростанням кількості мобільних пристроїв і широким використанням десктопних систем виникає необхідність у ґрунтовному аналізі підходів до проєктування, реалізації та оцінювання графічних інтерфейсів. Якість реалізації UI (User Interface) безпосередньо впливає на зручність, доступність і продуктивність роботи кінцевих користувачів, що зумовлює актуальність дослідження особливостей побудови інтерфейсів у кросплатформних програмних системах.

Метою дипломної роботи є дослідження принципів побудови, критеріїв оцінки та порівняльного аналізу графічних інтерфейсів у мобільних і десктопних програмних додатках, а також розробка програмного засобу для автоматизованого збору та інтерпретації метрик інтерфейсів. У межах дослідження передбачено виявлення відмінностей у дизайні, логіці побудови та підходах до взаємодії користувача з інтерфейсами залежно від типу платформи.

Предметом дослідження є методи реалізації, структурування та оцінки графічних інтерфейсів у програмному забезпеченні.

Об’єктом дослідження виступають мобільні та десктопні програмні продукти з різними реалізаціями інтерфейсів користувача, що функціонують у середовищах Android, iOS, Windows та Linux.

У межах поставленої мети передбачається виконання таких основних **завдань:**

– здійснити системний аналіз предметної області з урахуванням сучасних підходів до побудови графічних інтерфейсів у мобільних та десктопних середовищах;

– дослідити чинні принципи та стандарти взаємодії людини з комп’ютером (HCI), особливості реалізації UI/UX на різних платформах, а також методи їхньої оцінки;

– сформулювати функціональні та нефункціональні вимоги до програмного засобу аналізу інтерфейсів;

– побудувати концептуальну модель системи із застосуванням інструментів структурного та об'єктно-орієнтованого моделювання (діаграми класів, прецедентів, компонентів, розгортання тощо);

– розробити програмний інструмент, що дозволяє здійснювати збір, аналіз та порівняння інтерфейсних елементів на основі визначених критеріїв (доступність, контрастність, адаптивність, відповідність UI-гайдам);

– реалізувати модулі формування текстових і графічних звітів за результатами аналізу інтерфейсів;

– провести тестування та оцінку ефективності програмного забезпечення в умовах реального середовища виконання;

– сформувати рекомендації щодо впровадження розробленого інструменту в освітній, дослідницький або професійний контекст.

Наукова новизна роботи полягає у створенні спеціалізованого інструменту, який дозволяє автоматизовано здійснювати аналіз графічних інтерфейсів додатків на різних платформах з урахуванням принципів доступності, юзабіліті та відповідності стандартам побудови UI. Запропонований підхід дозволяє комплексно оцінити ефективність і зручність інтерфейсу на основі структурованих критеріїв.

Структура дипломної роботи складається з чотирьох розділів. У першому розділі подано системний аналіз предметної області, обґрунтовано

актуальність теми, сформульовано завдання та визначено методи дослідження. Другий розділ присвячений проєктуванню інформаційного та програмного забезпечення системи аналізу інтерфейсів. У третьому розділі описано безпосередню реалізацію програмного засобу з деталізацією модулів та алгоритмів. У четвертому розділі наведено рекомендації щодо тестування, впровадження та експлуатації створеної системи. Завершенням роботи є загальні висновки, список використаних джерел та додатки з фрагментами реалізації.

.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сфера розробки графічних інтерфейсів є ключовою складовою сучасного програмного забезпечення, оскільки саме вона визначає зручність, ефективність і швидкість взаємодії користувача з системою. У контексті мобільних і десктопних додатків поняття інтерфейсу охоплює не лише візуальне представлення, а й логіку взаємодії, послідовність дій, реакцію на події та адаптивність до параметрів пристрою. Зростаюча кількість багатофункціональних програм, що працюють у різних середовищах (наприклад, Android, iOS, Windows, Linux), потребує уніфікованих, але при цьому контекстно чутливих рішень щодо побудови UI. Відповідно до цього у предметній області аналізу виділяються мобільні та десктопні інтерфейси, кожен із яких має власні особливості, обмеження і принципи побудови, що обґрунтовує потребу в їхньому системному порівнянні.

Платформні відмінності суттєво впливають на вибір архітектурних шаблонів, типових патернів, а також на очікування користувачів щодо функціональності та ергономіки інтерфейсу. Так, у мобільному середовищі основний акцент робиться на жестову навігацію, компактність і економне використання простору, тоді як у десктопному – на панелі інструментів, багатовіконність та широкі можливості кастомізації. Ці особливості формують окремі підходи до структуризації інтерфейсів, що враховуються при розробці UI-дизайну під конкретну платформу. Для системного узагальнення ключових характеристик інтерфейсів у різних середовищах у табл. 1.1 наведено порівняння мобільного та десктопного UI за основними Огляд інформаційних джерел та існуючих рішень

Таблиця 1.1

Порівняльна характеристика мобільних та десктопних графічних інтерфейсів

Параметр	Мобільні інтерфейси	Десктопні інтерфейси
Тип взаємодії	Сенсорне введення, жести	Клавіатура, миша
Розмір екрана	Обмежений, змінний	Більший, стабільний
Принцип навігації	Вертикальна прокрутка, вкладки	Меню, панелі інструментів
Обмеження на обсяг інформації	Високі (через розмір дисплея)	Низькі (можна виводити великі обсяги одночасно)
Пріоритет дизайну	Простота, мінімалізм	Функціональність, контроль
Адаптивність	Висока необхідність адаптивного дизайну	Часто фіксована розмітка
Частота оновлення UI	Частіша (через оновлення платформ)	Рідше змінюється

Як видно з таблиці, у межах мобільних платформ домінує підхід, орієнтований на сенсорне керування, лаконічність та узгодженість із гайдлайнами операційної системи, зокрема Material Design для Android і Human Interface Guidelines для iOS. У свою чергу, десктопні застосунки більше орієнтовані на багатофункціональність, глибоку взаємодію та складну навігаційну структуру. Це зумовлює потребу у диференційованому підході до аналізу їхніх графічних оболонок, який враховує не лише технічні параметри, а й контекст використання, тип користувача, цільові задачі та поведінкові патерни.

З огляду на вказане, предметна область дипломної роботи охоплює концепти, інструменти та реалізації, пов’язані з побудовою, інтерпретацією та оцінкою графічних інтерфейсів у двох типах цифрових середовищ. Вивчення подібностей і відмінностей у структурі, логіці та принципах взаємодії дозволяє сформулювати єдину систему критеріїв для об’єктивного порівняння та формування вимог до інтерфейсів незалежно від платформи. Це забезпечує

підґрунтя для подальшої розробки програмного інструменту аналізу UI-дизайнів з урахуванням кросплатформного контексту.

1.2 Огляд інформаційних джерел та існуючих рішень

На сьогодні не існує безпосередніх аналогів програмної системи, яка б забезпечувала комплексний автоматизований аналіз графічних інтерфейсів мобільних і десктопних додатків із можливістю порівняльного оцінювання їхньої структури, логіки побудови та візуальної відповідності сучасним стандартам. Саме тому доцільним є дослідження існуючих професійних інструментів, які забезпечують розробку та проектування графічних інтерфейсів, з метою виявлення застосовних концепцій і принципів для реалізації аналітичного функціоналу майбутньої системи.

Однією з найпоширеніших платформ для кросплатформного проєктування графічного інтерфейсу є Figma — хмарне середовище для дизайну інтерфейсів і спільної роботи над UI/UX-проєктами. Figma підтримує створення інтерактивних макетів, прототипів та забезпечує зручний доступ до інспекційних панелей, властивостей елементів, стилів і компонентів. Завдяки можливості автоматизованої верстки та підтримці режиму розробника, Figma використовується не лише для візуального дизайну, але й для технічного аналізу інтерфейсних рішень (рис. 1.1).

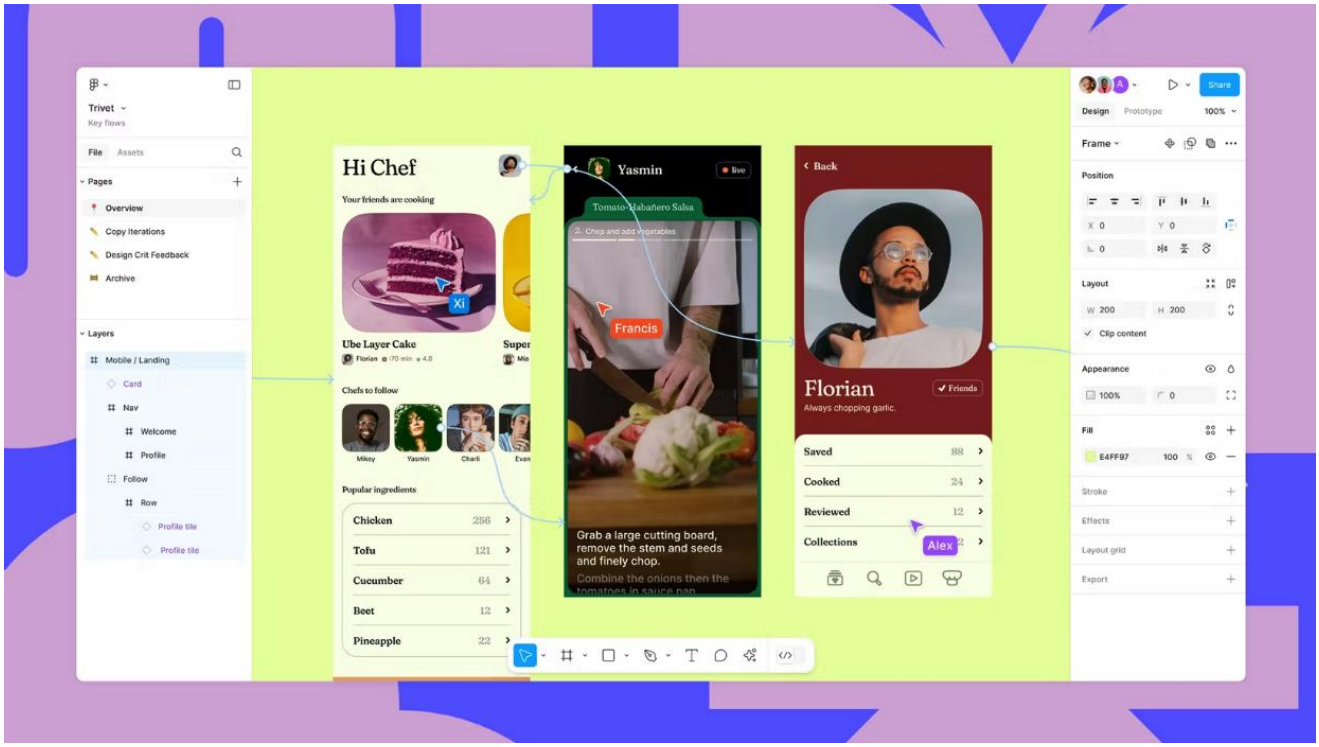

Рисунок 1.1 – Інтерфейс середовища Figma

Ще одним прикладом професійного середовища є Adobe XD — інтерактивний редактор для проектування інтерфейсів, який дозволяє реалізовувати складні багатовіконні прототипи, а також перевіряти логіку переходів між екранами. Adobe XD активно використовується у комерційному дизайні мобільних та десктопних застосунків, забезпечуючи відповідність створених UI-структур до принципів Material Design та HIG (рис. 1.2).

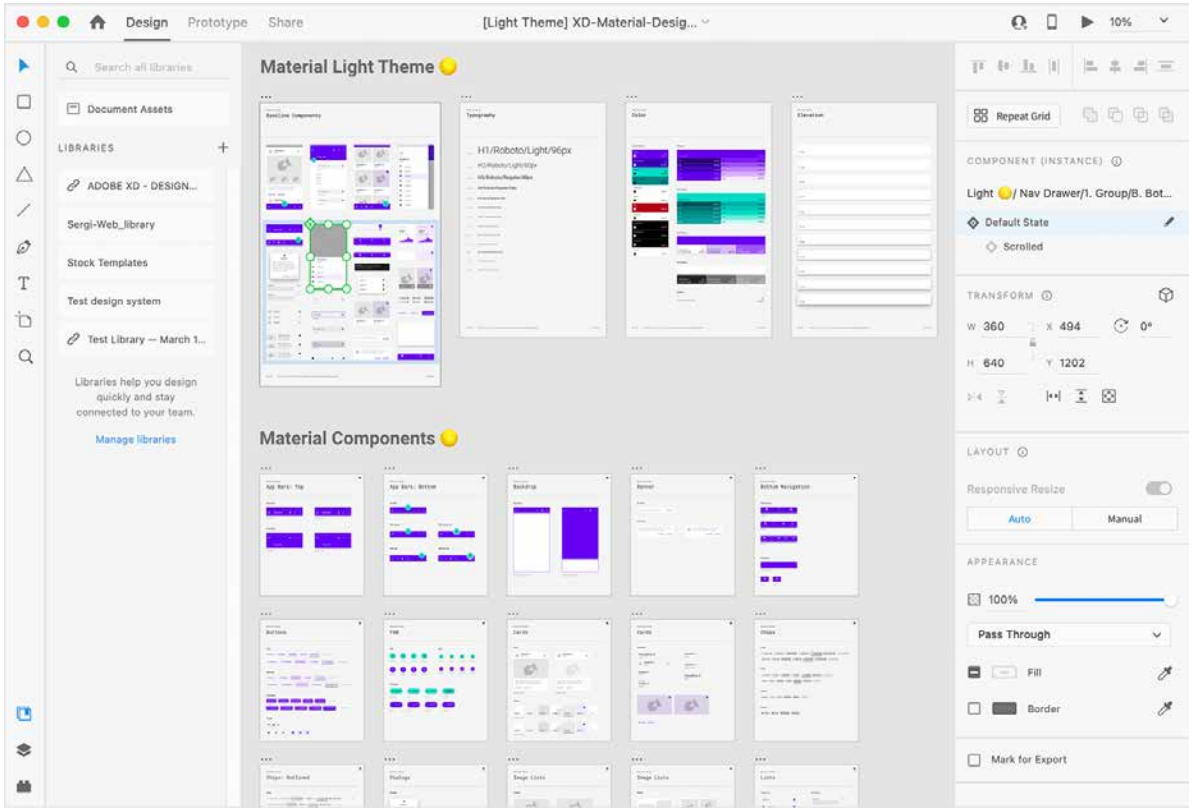

Рисунок 1.2 – Робоча область Adobe XD із темним шаблоном елементів

У випадку створення десктопних додатків із графічним інтерфейсом за допомогою мови Python актуальним є застосування Qt Designer — графічного редактора з пакету Qt, який дає змогу створювати інтерфейси на основі компонентів бібліотеки PyQt. Це середовище реалізує подання UI у форматі XML, підтримує WYSIWYG-підхід, дозволяє налаштовувати ієрархію елементів, їхні властивості, сигнали і слоти (рис. 1.3). Такий підхід зручний для інтеграції з Python-кодом, що дозволяє будувати UI-орієнтовані десктопні системи з високим ступенем модульності.

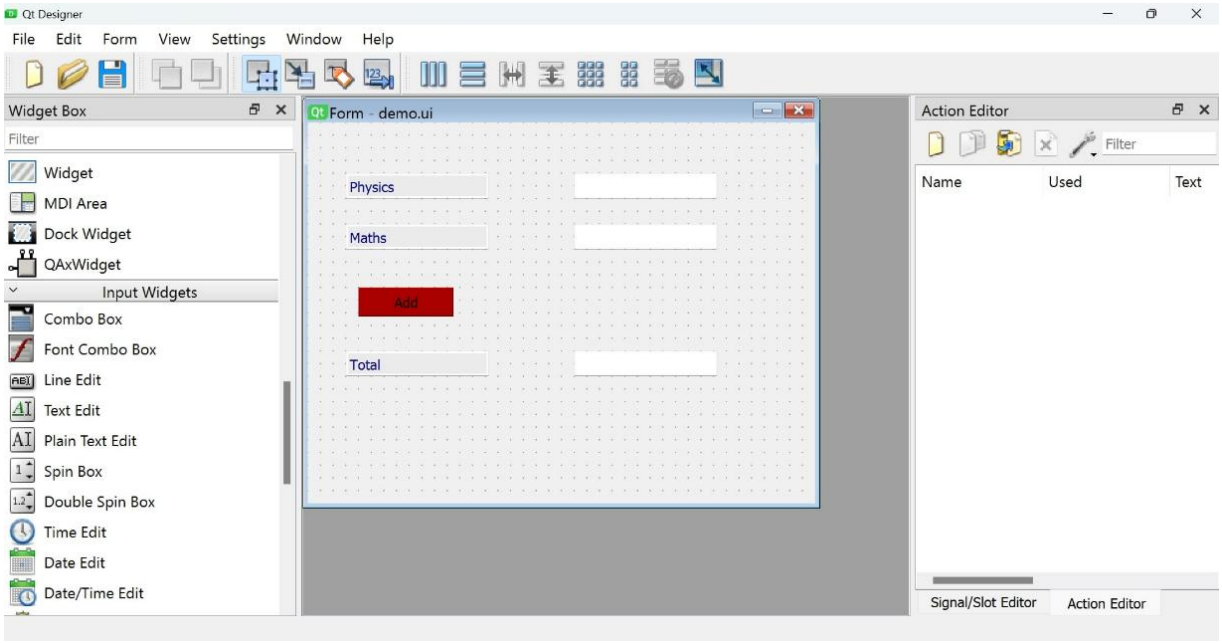

Рисунок 1.3 – Середовище візуального компонування Qt Designer

Для мобільної платформи Android найбільш поширеним інструментом є Android Studio Layout Editor — візуальний редактор інтерфейсів, інтегрований до офіційного середовища розробки Android. Цей інструмент підтримує створення адаптивних макетів за допомогою ConstraintLayout, автоматичне попереднє відображення елементів UI у різних конфігураціях екрана, а також надає розробнику можливість налаштовувати прив’язки, параметри та взаємозв’язки між елементами (рис. 1.4). Це дозволяє реалізовувати інтерфейси, які коректно функціонують на широкому спектрі пристроїв і роздільних здатностей дисплея.

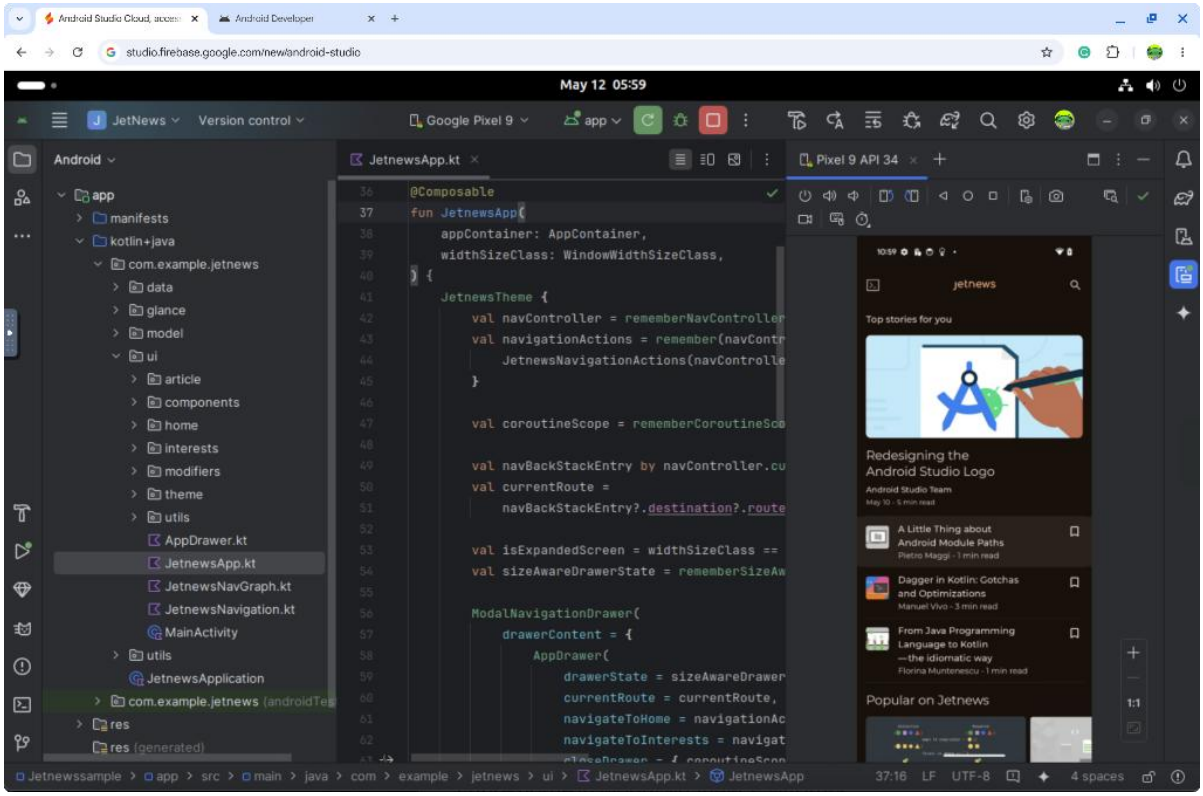

Рисунок 1.4 – Приклад візуального редактора макетів Android Studio

Огляд зазначених платформ свідчить про високий рівень розвитку інструментарію для створення інтерфейсів, проте жоден з наведених інструментів не має вбудованого механізму об’єктивної оцінки ефективності UI чи його відповідності визначеним критеріям. Існуючі рішення зосереджені передусім на дизайні та попередньому перегляді, а не на глибокому аналізі структурних характеристик інтерфейсу, взаємозв’язків між компонентами або їхньої поведінки під час реальної взаємодії. Це створює передумови для розробки нової системи, що не конкурує з розглянутими інструментами, а доповнює їх у частині формалізованого аналізу якості графічного інтерфейсу.

З метою систематизації результатів аналізу наведених інструментальних рішень доцільно провести їх порівняння за основними критеріями, що мають безпосереднє значення для подальшої розробки аналітичної системи: тип підтримуваного середовища, орієнтація на розробника або дизайнера, доступність інспекційних функцій, підтримка адаптивності та можливість

експорту структури інтерфейсу у форматі, придатному для аналізу. Порівняльна характеристика подана в табл. 1.2.

Таблиця 1.2

Порівняльна характеристика існуючих інструментів проектування інтерфейсів

Інструмент	Платформа	Орієнтація	Інспекція атрибутів	Підтримка адаптивності	Експорт структури UI
Figma	Веб, мультиплатформна	UI/UX-дизайн	Так	Так	Так (JSON, SVG)
Adobe XD	Windows/macOS	UI/UX-дизайн	Так	Обмежено	Так (SVG, PNG)
Qt Designer	Десктоп (Windows/Linux/macOS)	Розробка GUI	Так	Обмежено	Так (UI/XML)
Android Studio Layout Editor	Android	Розробка GUI	Так	Так	Так (XML)

Як показано в таблиці, усі платформи забезпечують підтримку візуального редагування, проте лише частина з них надає засоби структурного експорту, що є критично важливим для побудови інструменту, орієнтованого на аналіз графічного інтерфейсу. Зокрема, Figma дозволяє експортувати окремі компоненти у форматі JSON, що відкриває можливості для програмного парсингу й обробки; Android Studio підтримує декларативний опис інтерфейсів у XML, що також може бути використано як вхідні дані для аналізу. Qt Designer генерує UI-файли на основі XML-схеми, що дозволяє будувати логічне дерево елементів. Adobe XD, попри зручність для дизайнерських завдань, має обмеження щодо адаптивності та структурного експорту.

Результати цього аналізу дозволяють зробити висновок, що жоден з розглянутих інструментів не виконує функції автоматизованого оцінювання параметрів інтерфейсу відповідно до формалізованих критеріїв якості, таких як узгодженість, читабельність, адаптивність, контрастність, навігаційна складність тощо. Усі наявні рішення орієнтовані на етапи проектування або реалізації, але не передбачають етапу об’єктивного порівняльного аналізу. Враховуючи виявлені сильні та слабкі сторони кожного середовища, на основі узагальнених характеристик можна сформулювати вимоги до майбутньої системи автоматизованого аналізу графічних інтерфейсів, що представлені у пункті 1.3.

1.3 Аналіз вимог до програмної системи

Формування вимог до програмної системи аналізу графічних інтерфейсів здійснюється на основі результатів огляду предметної області, аналізу існуючих рішень (див. п. 1.2), а також з урахуванням стандартів, рекомендацій і практик побудови UI. Вимоги поділяються на функціональні та нефункціональні. Першочергово визначаються загальні очікування щодо системи: вона повинна мати можливість завантаження макетів або структурованих UI-файлів, автоматичного розпізнавання основних елементів інтерфейсу, аналізу їхньої структури, атрибутів і взаємозв’язків, а також генерації текстового або графічного звіту. У табл. 1.3 узагальнено основні функціональні вимоги до системи.

Таблиця 1.3

Функціональні вимоги до системи

№	Вимога	Короткий опис
1	Завантаження макету	Імпорт UI-файлу (наприклад, у форматі JSON, XML, UI)
2	Аналіз структури	Визначення ієрархії елементів, типів, вкладеності

3	Обчислення метрик	Вимірювання щільності елементів, контрасту, повторюваності
4	Побудова логічної карти інтерфейсу	Візуалізація структури з виділенням ключових компонентів
5	Генерація звіту	Формування звіту у форматах PDF, TXT або HTML
6	Підтримка мультимовності	Інтерфейс має підтримувати зміну мови

Значну роль у процесі розробки відіграють також нефункціональні вимоги, що визначають загальні властивості системи: продуктивність, портативність, масштабованість, безпеку та зручність використання. З огляду на плановану архітектуру (локальна десктопна програма з можливістю офлайн-аналізу), система повинна бути придатною для розгортання на різних ОС, мати мінімальні вимоги до ресурсів, забезпечувати швидку обробку вхідних даних і збереження результатів у структурованому вигляді. У табл. 1.4 наведено нефункціональні вимоги, сформовані відповідно до ISO/IEC 25010 [18].

Таблиця 1.4

Нефункціональні вимоги до системи

№	Категорія	Вимога
1	Продуктивність	Аналіз середнього макету не перевищує 2 секунд
2	Портативність	Робота на Windows, macOS та Linux
3	Юзабіліті	Інтерфейс не потребує попереднього навчання користувача
4	Надійність	Система має обробляти помилки вводу без аварійного завершення
5	Безпека	Локальне збереження результатів без зовнішньої передачі даних

Крім загальних вимог, що охоплюють функціональні блоки та системні характеристики, необхідно враховувати специфічні вимоги, пов’язані із забезпеченням сумісності формальних критеріїв якості інтерфейсу. Йдеться про підтримку набору метрик, які дозволяють кількісно оцінити властивості UI: кількість унікальних елементів, глибину вкладеності, використання контрасту, наявність відступів, порушення адаптивності тощо. Ці метрики мають бути реалізовані як незалежні модулі з можливістю розширення та налаштування. У табл. 1.5 подано перелік рекомендованих метрик.

Таблиця 1.5

Ключові метрики для аналізу графічного інтерфейсу

№	Метрика	Призначення
1	Глибина вкладеності	Визначає складність ієрархії елементів
2	Середній відступ	Аналізує відповідність до рекомендацій щодо просторової сітки
3	Коефіцієнт контрасту	Обчислює читаємість тексту на тлі
4	Щільність розміщення	Оцінює перевантаження елементами у межах одного екрану
5	Повторюваність компонентів	Виявляє шаблонність або недотримання уніфікованих стилів
6	Адаптивність	Перевіряє зміну інтерфейсу залежно від розмірів або орієнтації екрана

Аналіз функціональних, нефункціональних і метричних вимог дозволяє сформувати цілісну модель очікуваної системи, яка об’єднує засоби автоматизованого розбору інтерфейсів, обробки структурних атрибутів, візуалізації результатів і оцінювання відповідності сучасним UI-гайдам. Сформульовані вимоги будуть покладені в основу при створенні логічної моделі програмного засобу, що викладена у наступному розділі (див. п. 2.1).

1.4 Постановка завдання

На основі проведеного аналізу предметної області (див. п. 1.1), порівняння існуючих рішень та інструментів (див. п. 1.2), а також результатів формалізації функціональних і нефункціональних вимог (див. п. 1.2 та табл. 1.3–1.5), сформульовано технічне завдання на розробку інформаційної системи підтримки аналізу графічних інтерфейсів мобільних та десктопних додатків. Метою системи є забезпечення автоматизованого аналізу структури інтерфейсів на основі вхідних UI-файлів або макетів, оцінювання їх відповідності до заданих критеріїв якості, візуалізація результатів та формування технічного звіту для

подальшого використання в освітньому, аналітичному чи розробницькому середовищі.

Розроблювана система повинна функціонувати локально, без інтеграції із зовнішніми API, що дозволить гарантувати автономність роботи та конфіденційність обробки інтерфейсних даних. Основні вхідні параметри системи включають структуровану інформацію про графічний інтерфейс: список елементів, їхню вкладеність, типову ідентифікацію (наприклад, кнопка, текстове поле, контейнер), атрибути візуального представлення, а також відомості про екранну роздільність, платформу, тип компонування та орієнтацію інтерфейсу. Структура основних вхідних даних подана в таблиці 1.6.

Таблиця 1.6

Вхідні параметри для аналізу інтерфейсу

Категорія	Параметри
Тип платформи	Android, iOS, Windows, Linux, macOS
Формат вхідних даних	JSON, XML, UI (Qt), макет у Figma
Елементи інтерфейсу	Назва, тип, позиція, розмір, вкладеність, відступи, атрибути стилю
Візуальні параметри	Колір тла, колір тексту, шрифт, товщина ліній, наявність іконок
Поведінкові властивості	Події onClick, onHover, логіка навігації
Конфігурація екрану	Ширина, висота, орієнтація, масштабування

На основі обробки зазначених параметрів система повинна формувати структуровані результати аналізу інтерфейсу. Це передбачає обчислення метрик складності, адаптивності, візуальної узгодженості та читабельності, виявлення потенційних порушень рекомендацій з проєктування інтерфейсів, а також генерацію рекомендацій щодо вдосконалення UI. Результати, які система повинна формувати після аналізу, наведено в таблиці 1.7.

Таблиця 1.7

Очікувані результати роботи системи

Компонент результату	Опис
Карта структури інтерфейсу	Візуалізоване дерево елементів із вкладеністю, типами і ключовими властивостями
Аналітичний звіт	Підсумкові значення метрик (глибина, щільність, контраст, повторюваність тощо)
Попередження	Повідомлення про виявлені порушення (надмірна вкладеність, низький контраст тощо)
Рекомендації	Список порад щодо покращення структури, доступності та адаптивності
Збереження результатів	Можливість експорту результатів у PDF, CSV або HTML
Повідомлення для користувача	Інформування про завершення аналізу, помилки або недостатність вхідних даних

Архітектура інформаційної системи повинна мати модульну структуру з чітким поділом на компоненти: обробку вхідного формату, логіку аналізу, обчислення метрик, формування звітів та виведення результатів. Програмна реалізація має бути побудована на базі десктопної технології Python з графічним інтерфейсом на PyQt6, що дозволяє реалізувати кросплатформну локальну систему без потреби в браузері або зовнішньому сервері. У таблиці 1.8 наведено основні програмні компоненти системи.

Таблиця 1.8

Основні компоненти інформаційної системи

Компонент	Призначення
Модуль імпорту	Завантаження вхідних UI-файлів (JSON, XML, .ui)
Аналізатор структури	Визначення ієрархії, класифікація елементів, побудова дерева інтерфейсу
Обчислювач метрик	Розрахунок глибини, відступів, контрастності, повторюваності тощо
Генератор звітів	Формування PDF/HTML-документів на основі результатів аналізу
Модуль рекомендацій	Генерація пропозицій щодо оптимізації UI
Візуальний інтерфейс (GUI)	Інтерактивний графічний інтерфейс для взаємодії користувача із системою

З технічної точки зору система має реалізовуватись з урахуванням принципів архітектурної моделі MVC (Model–View–Controller), що забезпечує відокремлення логіки аналізу, відображення результатів та збереження даних, а також спрощує процес тестування окремих компонентів. Структура задачі охоплює повний життєвий цикл побудови програмного засобу — від імпорту вхідних даних до генерації формалізованого висновку — і забезпечує цілісну логіку подальшого проєктування концептуальної моделі, яка буде детально представлена у наступному розділі (див. п. 2.1).

1.5 Моделювання предметної області

У межах моделювання предметної області проєктованої інформаційної системи аналізу графічних інтерфейсів було здійснено формалізацію типових сценаріїв взаємодії користувача із системою, визначення послідовності інформаційних процесів та структури логічних зв’язків між основними компонентами. Застосування стандартних засобів UML-моделювання дало змогу наочно описати функціональну поведінку системи, ролі суб’єктів, черговість викликів методів та загальну динаміку обробки запитів.

На основі аналізу функціональних вимог було побудовано діаграму прецедентів, що відображає основні дії, які виконують учасники процесу – користувач, системний аналітик та UI-експерт. Користувач ініціює процес завантаження інтерфейсного макета, запуск аналізу та передачу його на обробку, що включає визначення платформи та розширену оцінку юзабіліті. Системний аналітик відповідає за генерацію звіту, а UI-експерт може додатково здійснювати порівняння макета з іншими проєктами та експортувати результати (рис. 1.5). У діаграмі передбачено механізми розширення через зв’язки типу «extend», що дає змогу масштабувати функціональність за потреби.

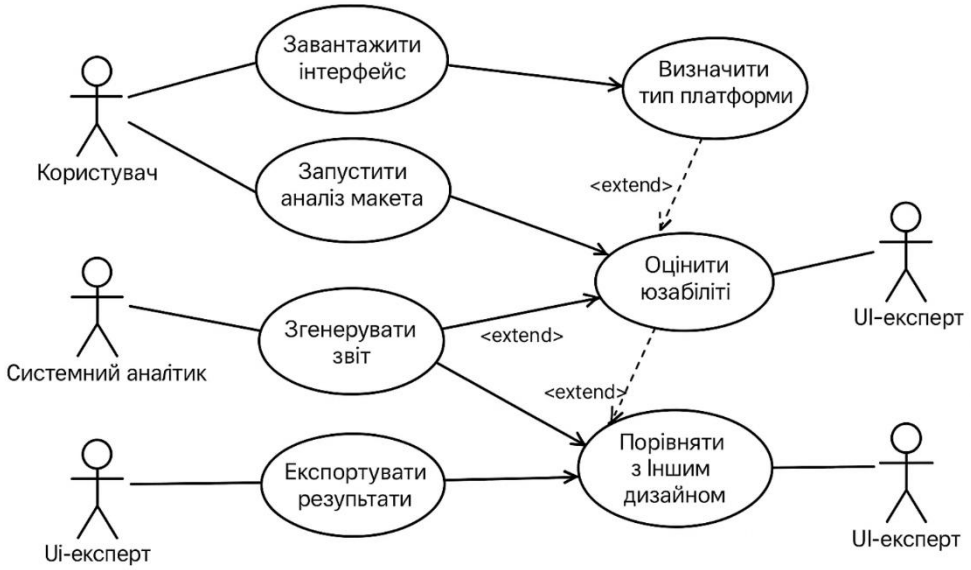

Рисунок 1.5 – Діаграма прецедентів взаємодії із системою

Для відображення динаміки взаємодії між логічними модулями системи побудовано діаграму послідовності, яка моделює процес запуску аналізу UI-макету. Вона ілюструє послідовність повідомлень між користувачем, графічним інтерфейсом, контролером, модулем аналізу та генератором звітів. Після вибору файлу через UI викликається метод ініціалізації аналізу, який передає файл у контролер, далі — в аналітичний модуль, де здійснюється виклик методу обчислення метрик, після чого результат повертається у зворотному напрямку до інтерфейсу користувача (рис. 1.6). Така модель дозволяє чітко розмежувати функції компонентів та простежити інформаційні потоки у системі.

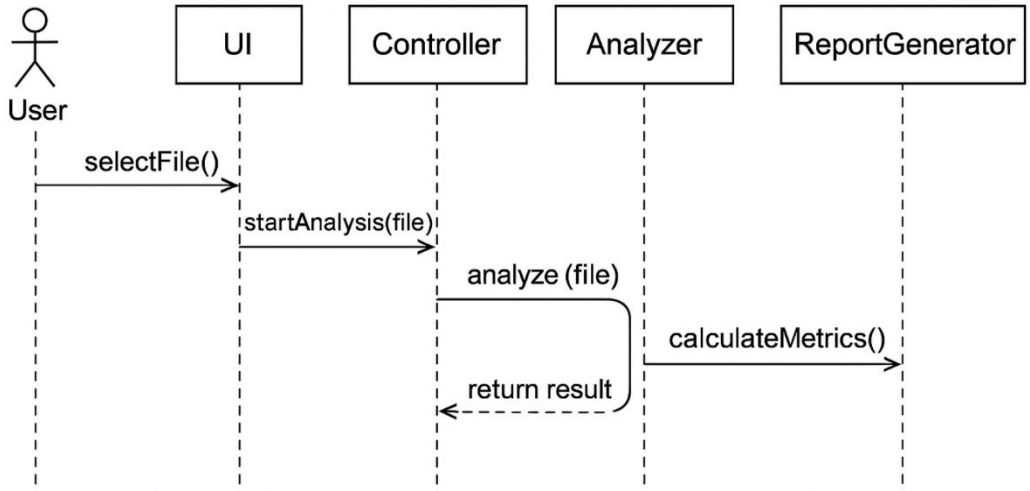

Рисунок 1.6 – Діаграма послідовності запуску аналізу макета

Для узагальнення логіки функціонування системи також побудовано діаграму активності, що відображає основні етапи від моменту запуску програми до формування результату. Кожен крок подано у вигляді дії: завантаження макета, визначення типу платформи, обробка структури інтерфейсу, обчислення метрик, генерація звіту та виведення результату. Послідовність процесів відповідає логіці обробки вхідних даних у програмі й відображає лінійну структуру сценарію без умовних переходів (рис. 1.7). Модель акцентує на ролі внутрішньої логіки системи, яка опрацьовує дані відповідно до принципу єдиної відповідальності на кожному етапі.

Рисунок 1.7 – Діаграма активності обробки UI-макета

Здійснене моделювання предметної області дозволяє забезпечити повне охоплення логічних сценаріїв системи, виявити основні залежності між компонентами, структурувати потоки обробки даних і візуалізувати очікувану

функціональність на ранньому етапі проєктування. Отримані діаграми будуть покладені в основу архітектурного проєктування та реалізації прототипу системи, відповідно до вимог, сформульованих у попередньому пункті (див. п. 1.3).

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Побудова логічної моделі даних є ключовим етапом у структурному проєктуванні інформаційної системи, оскільки дозволяє формалізувати сутності предметної області, їхні атрибути та зв’язки між ними. У розроблюваній системі аналізу графічних інтерфейсів передбачається обробка макетів UI, збереження структури елементів, фіксація розрахованих метрик, генерація звітів та зберігання інформації про користувацькі дії. З огляду на ці функціональні вимоги сформовано логічну модель даних, відображену у вигляді ER-діаграми (рис. 2.1), яка охоплює всі основні сутності, що беруть участь у взаємодії системи, та зв’язки між ними.

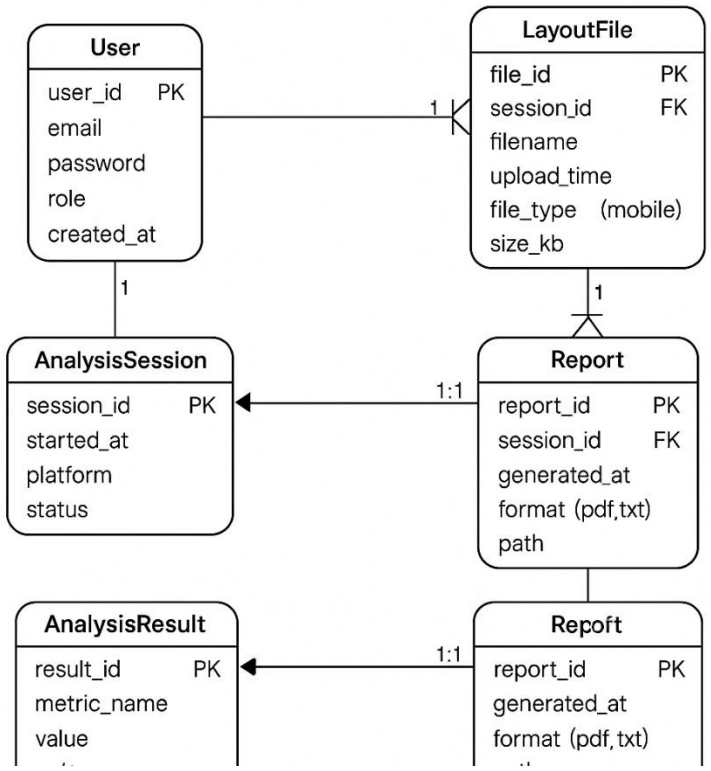

Рисунок 2.1 – ER-діаграма логічної моделі даних системи аналізу графічного інтерфейсу

Модель включає базові сутності: User, Interface, Component, Metric, Report, Platform. Кожна сутність має набір атрибутів, що ідентифікують її у системі та містять релевантні властивості для подальшої обробки. Наприклад, сутність Component описує елемент інтерфейсу (наприклад, кнопка, поле, панель) та пов’язана з конкретним макетом, а Metric зберігає розраховані показники для одного або кількох компонентів. Для уніфікованого подання цієї структури у табл. 2.1 наведено стислий опис сутностей та їх ключових атрибутів.

Таблиця 2.1

Основні сутності логічної моделі даних

Сутність	Ключові атрибути	Опис
User	user_id, name, role	Ідентифікатор користувача, його ім’я, тип доступу
Interface	interface_id, file_path, upload_time, platform_id	Шлях до файлу макета, час завантаження, платформа
Component	component_id, interface_id, type, x, y, width, height	Елемент UI, координати, тип і розміри
Platform	platform_id, name, description	Назва і опис платформи (Android, iOS, Windows)
Metric	metric_id, component_id, name, value	Назва метрики (контраст, глибина тощо)
Report	report_id, user_id, interface_id, generation_time, path	Зв’язок із користувачем, збережений звіт

Відповідно до моделі, між сутностями встановлюються логічні зв’язки: один користувач може створити багато звітів (User – Report), один інтерфейс може містити багато компонентів (Interface – Component), один компонент може бути пов’язаний із кількома метриками (Component – Metric), кожен інтерфейс належить певній платформі (Interface – Platform). Таким чином, логічна модель забезпечує повну відповідність вимогам, визначеним у п. 1.3, та дозволяє ефективно зберігати й обробляти інформацію, необхідну для реалізації функціональності аналізу.

Запропонована ER-модель не лише структуризує збереження даних у межах системи, а й створює основу для фізичної реалізації схеми бази даних. Завдяки модульності структури, вона дозволяє масштабування при розширенні функціональності – наприклад, додавання підтримки нових типів метрик, компонентів або форматів вхідних макетів без необхідності зміни архітектурної основи. Таким чином, логічна модель слугує невіддільною частиною концептуального рівня проєктування і закладає підґрунтя для реалізації наступного етапу – структурування класів системи та їхньої поведінки.

2.2 Діаграма класів і кооперації

Моделювання архітектури програмної системи на рівні класів дозволяє чітко визначити структуру компонентів, їхню взаємодію, типи зв’язків і відповідальність кожного елемента в реалізації функціональності. Діаграма класів була побудована з урахуванням принципів об’єктно-орієнтованого проєктування та шаблону MVC (Model–View–Controller), який передбачає розподіл логіки на частини: інтерфейс, керування та обробка даних. У рамках даної системи визначено п’ять ключових класів: GUIWindow, Controller, Analyzer, JsonStorage, ReportGenerator і PlatformAdapter. Їх взаємозв’язки подано у вигляді UML-діаграми класів (рис. 2.2).

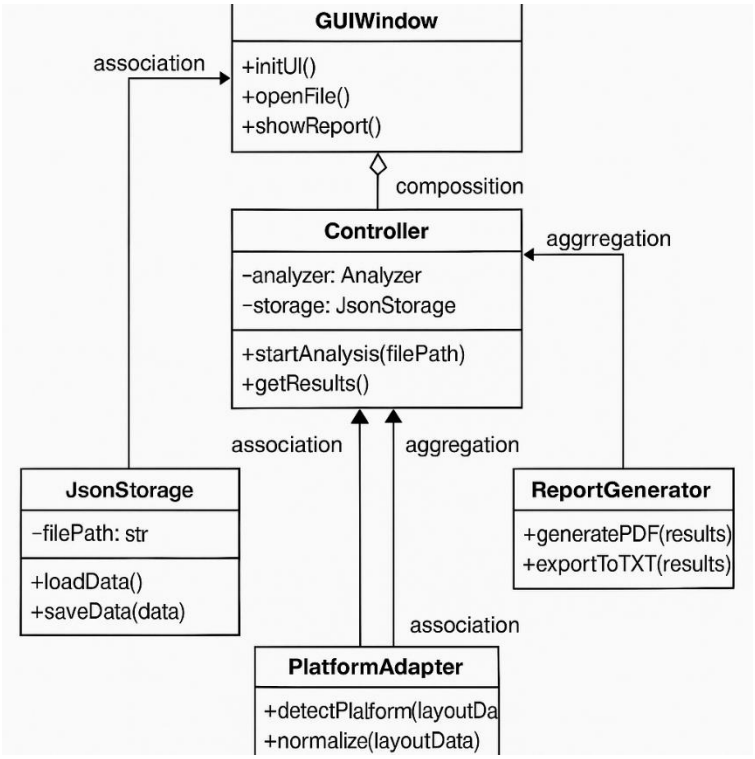

Рисунок 2.2 – Діаграма класів системи аналізу графічних інтерфейсів

Клас GUIWindow відповідає за взаємодію з користувачем, ініціалізацію інтерфейсу, завантаження файлу та виведення результатів. Він реалізує композиційний зв’язок із класом Controller, який є центральним координаційним компонентом. Контролер керує викликами методів модуля аналізу (Analyzer), зчитує дані через JsonStorage, ініціює формування звіту в ReportGenerator та взаємодіє з адаптером платформи для визначення типу середовища. Кожен із допоміжних класів реалізує власну відповідальність, що чітко відокремлена згідно з принципом єдиної відповідальності. У табл. 2.2 представлено коротку характеристику ключових класів.

Таблиця 2.2

Класи системи та їх основне призначення

Клас	Призначення
GUIWindow	Ініціалізація графічного інтерфейсу, обробка подій користувача
Controller	Координація роботи модулів, маршрутизація запитів
Analyzer	Обробка макета інтерфейсу, виклик обчислення метрик
PlatformAdapter	Визначення платформи, нормалізація структури відповідно до типу середовища

JsonStorage	Зчитування та збереження структурованих даних у JSON-файлах
ReportGenerator	Формування звітів у форматах PDF або TXT

Для моделювання обміну повідомленнями між об'єктами в межах одного сценарію взаємодії – запуску аналізу макета – побудовано UML-діаграму кооперації (діаграму комунікацій). У цій діаграмі відображено послідовність методів, які викликаються об'єктами під час взаємодії: від вибору файлу до генерації звіту. Користувач ініціює процес, контролер завантажує файл через JsonStorage, передає його аналізатору, який у свою чергу викликає адаптер платформи для виявлення контексту. Після обробки результат повертається до контролера, який спрямовує його в ReportGenerator, а потім відображає результат через інтерфейс (рис. 2.3).

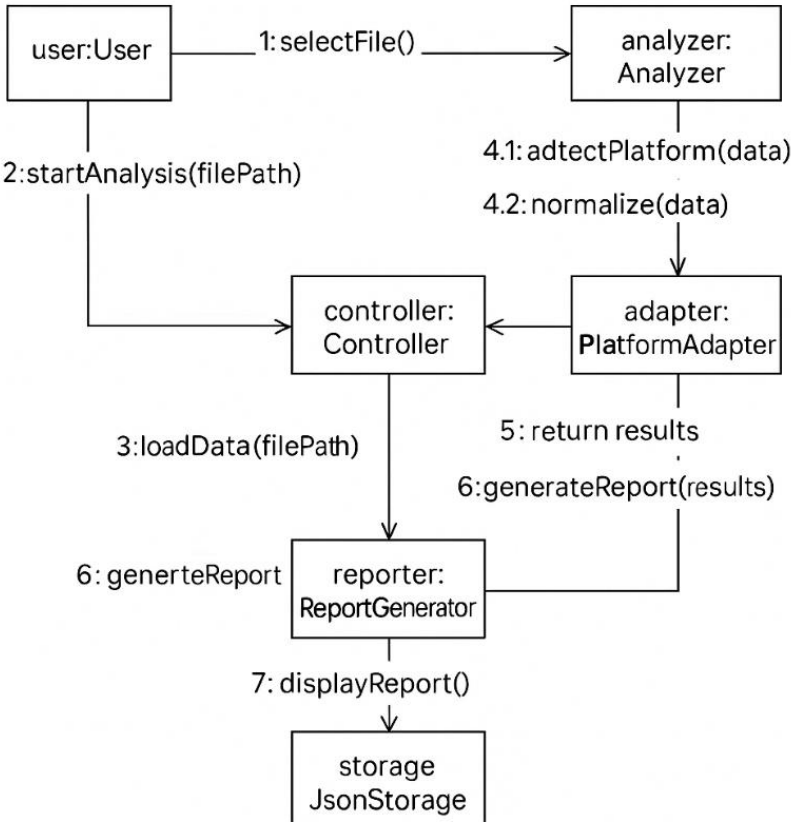

Рисунок 2.3 – Діаграма кооперації для сценарію аналізу макета

Дана діаграма кооперації демонструє об’єктну структуру системи під час виконання ключового сценарію та дозволяє побачити, як відбувається послідовна маршрутизація запитів між компонентами без порушення логічної

цілісності. Вона підтверджує відповідність реалізації архітектурним шаблонам і забезпечує прозорість для майбутнього тестування, підтримки та масштабування системи.

2.3 Діаграма пакетів

ля забезпечення модульності, масштабованості та структурної організації програмного коду системи аналізу графічного інтерфейсу доцільним є розбиття логіки на окремі пакети. Використання діаграми пакетів дозволяє формалізовано подати залежності між логічними модулями, визначити напрямки викликів функцій і зберегти незалежність ключових компонентів у процесі розробки та підтримки системи. Модель пакету також відображає розподіл відповідальностей між елементами системи згідно з принципом розділення обов'язків (Separation of Concerns), де кожен пакет виконує окрему роль: інтерфейс користувача, маршрутизація, обчислення, форматування, адаптація та допоміжна логіка.

У результаті аналізу структури проєктованої системи побудовано діаграму пакетів (рис. 2.4), яка відображає взаємозалежності між основними модулями: ui, controller, analyzer, reporting, platform та utils. Стрілки вказують на напрямки логічної залежності, де один пакет використовує інтерфейси або сервіси іншого.

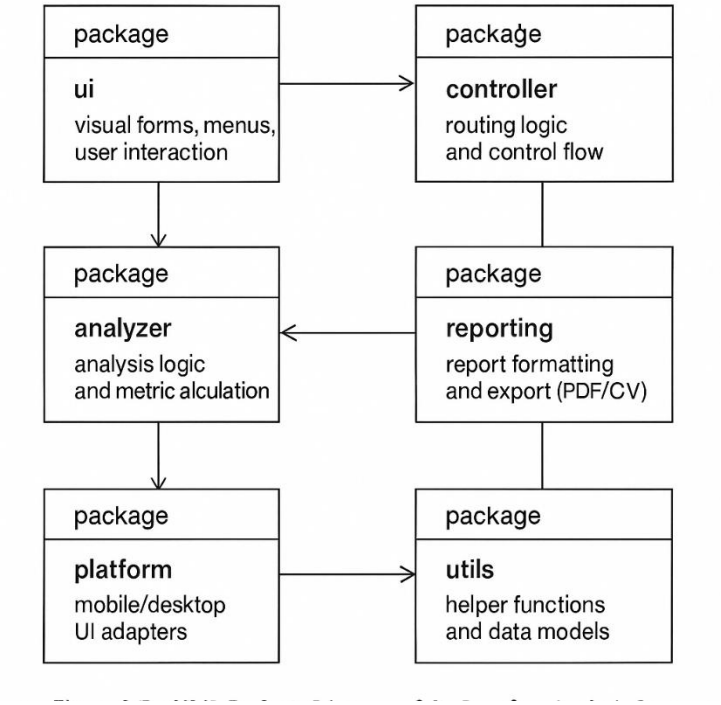

Рисунок 2.4 – Діаграма пакетів системи аналізу інтерфейсів

Пакет ui відповідає за реалізацію графічного інтерфейсу користувача та ініціалізацію основних подій, взаємодіє з controller, який координує роботу системи та спрямовує запити до analyzer. Сам analyzer виконує основні обчислення метрик і звертається до platform для врахування специфіки мобільного чи десктопного середовища. Пакет reporting форматує результати у вигляді файлів звітів і використовує вихідні дані з analyzer. Допоміжні функції, які не належать до основного функціонального навантаження, реалізовані в пакеті utils. Таблиця 2.3 узагальнює функціональну роль кожного з цих пакетів.

Таблиця 2.3

Опис призначення пакетів

Пакет	Призначення
ui	Графічні форми, меню, діалоги взаємодії з користувачем
controller	Логіка керування запитами, маршрутизація, виклики функціональних модулів
analyzer	Основна обчислювальна логіка, аналіз структури, розрахунок метрик
reporting	Формування та експорт результатів у PDF або текстові формати
platform	Адаптація логіки до мобільних або десктопних середовищ
utils	Допоміжні методи, утиліти, типи даних і моделі

Структура пакетів забезпечує розділення відповідальностей на рівні модулів, що спрощує реалізацію окремих частин системи незалежно одна від одної. Така архітектура знижує зв’язаність між компонентами, покращує читаність коду та полегшує подальше тестування, модифікацію і розширення функціональності системи. Запропонована модель також дає змогу реалізувати часткову повторну використуваність окремих модулів – зокрема, компонентів reporting і utils, які можуть бути інтегровані в інші системи без значних змін.

2.4 Діаграма компонентів

Компонентна модель системи дозволяє описати її структуру на рівні функціональних частин, кожна з яких виконує окреме завдання й може бути

незалежно реалізована, протестована та замінена без порушення цілісності архітектури. Діаграма компонентів наочно демонструє, з яких частин складається система, якими інтерфейсами ці частини взаємодіють та які залежності між ними існують. Для розроблюваної системи аналізу графічного інтерфейсу було виділено шість ключових компонентів, які реалізують основні функції: зчитування макетів, аналіз структури, нормалізація залежно від платформи, формування звіту, обробка даних і взаємодія з користувачем.

На рисунку 2.5 зображено структуру компонентів у вигляді UML-діаграми, де кожен блок позначено як окремий компонент із позначенням залежностей (лінії з круглими інтерфейсами). Центральним компонентом є Controller, що координує інші частини системи та забезпечує передачу команд. Компонент GUI_Module надає користувацький інтерфейс, Analyzer обробляє структуру макету, PlatformAdapter адаптує її до типу середовища, JsonStorage відповідає за зчитування та збереження даних, а generatePDF() формує підсумковий звіт.

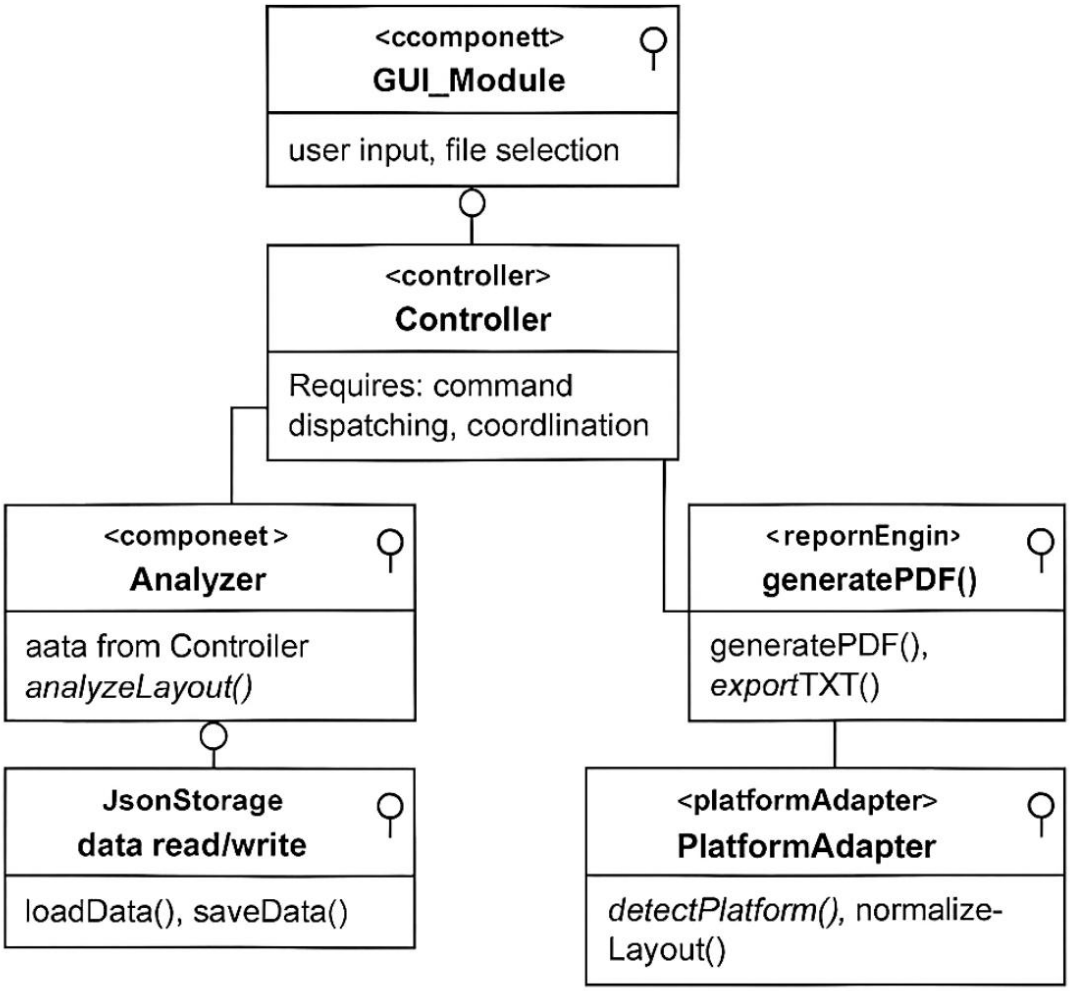

Рисунок 2.5 – Діаграма компонентів системи аналізу інтерфейсів

Для забезпечення повного уявлення про функціональну роль кожного компонента, у таблиці 2.4 наведено їх опис із зазначенням основних інтерфейсів, що надаються або використовуються.

Таблиця 2.4

Основні компоненти системи та їх функції

Компонент	Призначення	Інтерфейси / Методи
GUI_Module	Реалізація графічного інтерфейсу, обробка подій користувача	initUI(), openFile(), showReport()
Controller	Обробка запитів, маршрутизація між модулями	startAnalysis(filePath), getResults()
Analyzer	Аналіз структури макету, виклик обчислень	analyzeLayout()
PlatformAdapter	Визначення типу платформи, нормалізація структури	detectPlatform(), normalizeLayout()

JsonStorage	Зчитування та збереження даних, робота з JSON-файлами	loadData(), saveData()
generatePDF()	Формування PDF або TXT-звіту	generatePDF(), exportTXT()

Компонентна структура системи побудована з урахуванням принципів повторного використання (reusability), мінімальної залежності між модулями (low coupling) і максимальної відповідальності всередині кожного модуля (high cohesion). Такий підхід дає змогу окремо тестувати логіку аналізу, платформену адаптацію або генерацію звітів, а також легко масштабувати систему в майбутньому – наприклад, додати новий тип інтерфейсного експорту або підтримку іншого формату даних без зміни основного контролера. Запропонована модель логічно пов’язана з діаграмами класів і кооперацій (див. п. 2.2) і відповідає загальній архітектурі, що реалізується у проєкті.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

Для зберігання, доступу та обробки структурованих даних у розроблюваній системі аналізу графічних інтерфейсів обрана модель на основі файлової інформаційної бази у форматі JSON, яка функціонує локально та не потребує підключення до зовнішнього СУБД-сервера. Такий підхід повністю узгоджується з вимогами до автономної роботи системи, забезпечує конфіденційність обробки вхідних макетів інтерфейсів і гарантує стабільність у середовищах без доступу до мережі. Механізм взаємодії з інформаційною базою реалізовано у вигляді спеціалізованого модуля JsonStorage, який забезпечує завантаження UI-файлів, збереження результатів аналізу та обробку звітів. Кожен запис представляє собою окремий документ, що містить інформацію про компоненти інтерфейсу, обчислені метрики, платформу, час аналізу та структуру результатів

Обґрунтування вибору такої моделі збереження даних подано у таблиці 3.1, де представлено порівняння основних характеристик JSON-файлової бази з класичною реляційною СУБД у контексті цілей і обмежень цього програмного проєкту.

Таблиця 3.1

Порівняння моделей збереження даних у системі

Критерій	JSON-файлова модель	Реляційна СУБД (SQL)
Підтримка структури	Ієрархічна, вкладені об’єкти	Таблична, нормалізована
Простота розгортання	Висока, не потребує серверу	Потребує налаштування БД та сервісу
Продуктивність у малих проєктах	Висока	Надлишкова для локальних задач

Продовження таблиці 3.1

Підтримка автономності	Повна, можлива офлайн-робота	Обмежена, потребує клієнт-серверної моделі
Масштабованість	Обмежена без зовнішнього рушія	Висока при інтеграції з серверними СУБД
Гнучкість структури	Висока, необов'язкові поля	Жорстка схема, типізація
Швидкість доступу	Висока при прямому зчитуванні файлів	Швидка при індексованому доступі

Як видно з таблиці, файлово-орієнтований підхід на основі JSON цілком виправданий для програмної системи, що функціонує як настільний застосунок, має обмежений обсяг вхідних даних і не вимагає паралельного доступу до БД. Усі операції із зчитування, фільтрації та оновлення даних реалізуються безпосередньо через методи класу JsonStorage, який інкапсулює логіку роботи з файловою системою. Такий підхід також дозволяє зберігати кожен проаналізований макет у вигляді самостійного документу з єдиною структурою, що сприяє простому резервному копіюванню, обміну між системами та підтримці довготривалого зберігання результатів. У разі масштабування або перенесення системи у мережеве середовище структура бази може бути розширена до повноцінного сховища документів, наприклад на основі MongoDB, без необхідності зміни логіки взаємодії у вищих рівнях архітектури.

3.2 Розробка інформаційної бази

. Інформаційна база системи аналізу графічного інтерфейсу є ключовим елементом для зберігання результатів обробки макетів, обчислених метрик, звітів і сеансів взаємодії. У межах цієї системи реалізовано дві паралельні моделі: фізичну модель у вигляді JSON-файлів для автономної локальної роботи та логічну структуру у вигляді таблиць, яка відповідає класичному уявленню про реляційну базу даних. Таке подвійне представлення дозволяє гнучко адаптувати

збереження даних як для швидкого файлового доступу, так і для майбутньої міграції на SQL-сервер.

На рисунку 3.1 подано фізичну модель даних, яка містить п’ять основних таблиць: users, analysis_sessions, analysis_results, reports, files. Кожна з них має визначені зовнішні ключі, забезпечує зв’язки між сутностями та підтримує каскадне видалення, що сприяє цілісності даних.

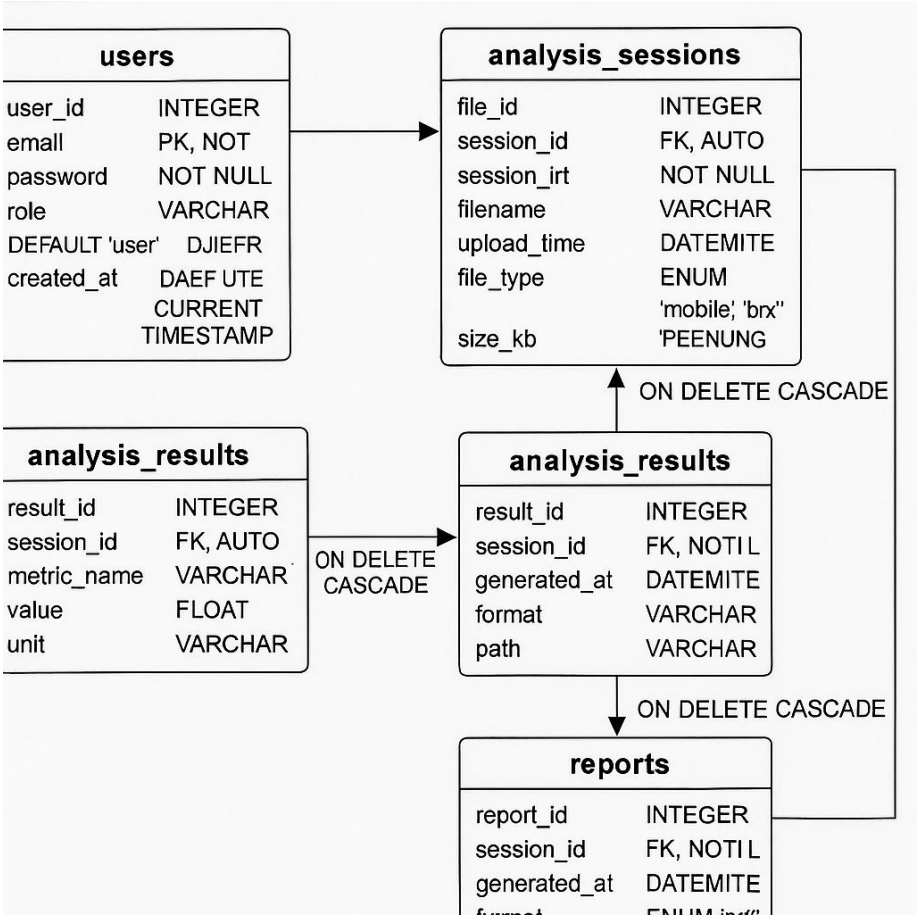

Рисунок 3.1 – Фізична модель даних системи (ER-схема)

Для практичної реалізації збереження даних у системі використано файлову модель на основі структури каталогів і REST-маршрутів, розгорнуту у вигляді Python-серверу з використанням Flask. У наведеному нижче фрагменті коду реалізовано базову логіку завантаження макетів інтерфейсів, збереження результатів аналізу, отримання компонентів і метрик, а також завантаження згенерованих звітів. Лістинінг коду представлений на рис.3.2.

```
# Ініціалізація каталогу для збереження
os.makedirs(DATA_DIR, exist_ok=True)

# Завантаження макета UI
@app.route('/upload', methods=['POST'])
def upload_interface():
    file = request.files.get('file')
    if not file:
        return jsonify({'error': 'No file provided'}), 400

    uid = str(uuid.uuid4())
    filepath = os.path.join(DATA_DIR, f"{uid}.json")
    file.save(filepath)
    return jsonify({'message': 'File uploaded', 'interface_id': uid}), 200

# Збереження результатів аналізу
@app.route('/save_results/<interface_id>', methods=['POST'])
def save_results(interface_id):
    results = request.json
    if not results:
        return jsonify({'error': 'No data received'}), 400

    result_path = os.path.join(DATA_DIR, f"{interface_id}_results.json")
    with open(result_path, 'w') as f:
        json.dump(results, f, indent=2)
    return jsonify({'message': 'Results saved'}), 200

# Отримання списку компонентів інтерфейсу
@app.route('/components/<interface_id>', methods=['GET'])
def get_components(interface_id):
    filepath = os.path.join(DATA_DIR, f"{interface_id}.json")
    if not os.path.exists(filepath):
        return jsonify({'error': 'Interface not found'}), 404

    with open(filepath) as f:
        data = json.load(f)
    return jsonify({'components': data.get('components', [])}), 200

# Отримання метрик для макета
@app.route('/metrics/<interface_id>', methods=['GET'])
def get_metrics(interface_id):
    result_path = os.path.join(DATA_DIR, f"{interface_id}_results.json")
    if not os.path.exists(result_path):
        return jsonify({'error': 'Results not found'}), 404

    with open(result_path) as f:
        data = json.load(f)
    return jsonify({'metrics': data.get('metrics', {})}), 200

# Завантаження сформованого звіту
@app.route('/report/<interface_id>', methods=['GET'])
def download_report(interface_id):
    report_path = os.path.join(DATA_DIR, f"{interface_id}_report.pdf")
    if not os.path.exists(report_path):
```

```
        return jsonify({'error': 'Report not found'}), 404
    return send_file(report_path, as_attachment=True)
```

Рис.3.2 – API-маршрути

У цій реалізації кожен інтерфейсний макет зберігається у вигляді окремого JSON-документа, прив’язаного до interface_id, а результати аналізу та звіт зберігаються у відповідних файлах із суфіксами _results.json і _report.pdf. Така структура дає змогу легко отримувати дані за запитом користувача без потреби в повноцінній базі даних, забезпечуючи швидкість, простоту резервування та незалежність від зовнішнього середовища.

Для формального представлення таблиць, що використовуються в моделі, у таблиці 3.2 наведено стислий опис структур відповідно до зображеної ER-схеми.

Таблиця 3.2

Структура таблиць фізичної моделі даних

Таблиця	Ключові поля та призначення
users	user_id (PK), email, password, role, created_at
analysis_sessions	session_id (PK), user_id (FK), filename, upload_time, file_type, size_kb
analysis_results	result_id (PK), session_id (FK), metric_name, value, unit
reports	report_id (PK), session_id (FK), generated_at, format, path
files	file_id (PK), session_id (FK), file_path, content_type, created_at

Реалізована модель інформаційної бази забезпечує ефективне зберігання, ізоляцію даних по сеансах і користувачах, підтримку цілісності структури та простий механізм розширення у випадку потреби масштабування або переходу до серверної СУБД. Така архітектура дозволяє гнучко поєднувати файлову природу макетів із логікою реляційного зберігання без втрати цілісності чи продуктивності.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Проєктування та реалізація інформаційної системи аналізу графічних інтерфейсів потребує ретельно підібраного інструментарію, який забезпечує повну відповідність функціональним вимогам, підтримує кросплатформенність, має достатній рівень стабільності, відкритості та дозволяє забезпечити візуалізацію, обробку та зберігання інтерфейсних даних. Основним критерієм при виборі технологій виступала можливість автономного запуску системи без залежності від зовнішніх сервісів, а також підтримка швидкої генерації графічного інтерфейсу та локальної обробки даних у форматі JSON.

Для реалізації графічного інтерфейсу було обрано бібліотеку PyQt6, яка забезпечує розробку повноцінного кросплатформного GUI-додатку із засобами взаємодії, обробки подій, виведення звітів і гнучкого компонування форм. Для побудови архітектури застосовано мову Python 3.11, яка поєднує високу читаність коду, підтримку об’єктно-орієнтованого підходу та широке розмаїття бібліотек для аналітики, обробки структурованих даних та роботи з файлами. Серверна логіка REST-запитів, збереження макетів і результатів аналізу реалізована за допомогою Flask — легкого вебфреймворку, що дозволяє швидко створити програмну обгортку навколо основної аналітичної логіки. У табл. 3.3 наведено обґрунтовану характеристику основного інструментарію, використаного при реалізації проєкту.

Таблиця 3.3

Обрані технології та їх функціональне призначення

Інструмент / Технологія	Версія	Призначення
Python	3.11.x	Основна мова розробки, логіка, обробка JSON, інтеграція компонентів

PyQt6	6.x	Графічний інтерфейс користувача (GUI), побудова форм, діалогів
Flask	2.x	Реалізація REST API, маршрути для збереження/завантаження даних
JSON	-	Формат обміну та збереження структури інтерфейсів і результатів
ReportLab	3.x	Генерація PDF-звітів на основі результатів аналізу
OS / UUID / Time	stdlib	Створення унікальних ID, керування каталогами та таймстемпами

Обраний інструментарій є відкритим, підтримується активною спільнотою та не потребує комерційних ліцензій для використання в навчальному та експериментальному середовищі. Крім того, всі компоненти легко інтегруються між собою, мають зрозумілу документацію та дозволяють зберігати результати у файловій системі без потреби в сторонніх сервісах. Такий підхід сприяє реалізації повністю ізольованої автономної системи, яка здатна працювати незалежно від наявності мережевого підключення та легко переноситься між платформами. Це дозволяє у майбутньому масштабувати функціональність без зміни фундаментального стеку технологій, розширювати API, підключати зовнішні модулі аналітики або автоматизації, зберігаючи при цьому початкову архітектурну цілісність.

3.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи аналізу графічних інтерфейсів реалізована відповідно до трирівневої моделі, що забезпечує чітке розмежування функціональних відповідальностей між презентаційним рівнем, рівнем логіки застосунку та рівнем збереження даних. Такий підхід підвищує масштабованість, забезпечує гнучкість модифікації окремих компонентів, спрощує тестування і дозволяє незалежно вдосконалювати кожен рівень без зміни загальної структури. В основі архітектури лежить шаблон MVC (Model–

View–Controller), адаптований до десктопної реалізації з використанням PyQt6 як інструменту побудови візуального інтерфейсу.

На рисунку 3.3 представлено загальну архітектурну модель системи, яка охоплює взаємозв'язки між користувачем, модулями GUI, контролером, аналітичним блоком, механізмом формування звітів та підсистемою збереження даних у форматі JSON. Прямі зв'язки між модулями відображають потокове передавання команд та результатів обробки між рівнями архітектури.

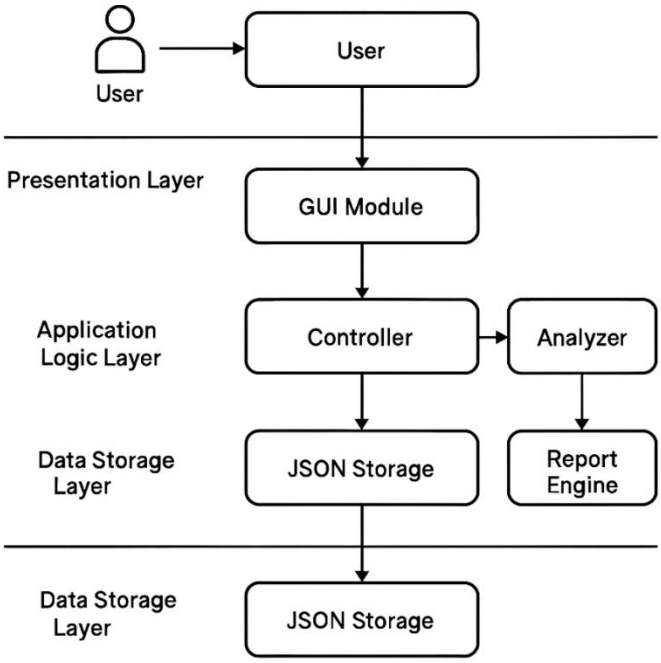

Рисунок 3.3 – Архітектура програмного забезпечення інформаційної системи

Верхній рівень архітектури — Presentation Layer — реалізує взаємодію з користувачем через GUI Module, який забезпечує завантаження макетів, ініціалізацію аналізу, перегляд результатів і запуск генерації звітів. Середній рівень — Application Logic Layer — представлений компонентом Controller, який виконує маршрутизацію запитів, взаємодіє з модулем Analyzer для обробки даних та викликає Report Engine для формування звітів. Нижній рівень — Data Storage Layer — реалізовано у вигляді модуля JSON Storage, який відповідає за зчитування і збереження інтерфейсних макетів, результатів обробки та готових звітів.

Для детального уявлення про ролі кожного компонента архітектури в таблиці 3.4 наведено короткий опис функцій, які виконує кожен з модулів.

Таблиця 3.4

Функціональні блоки архітектури програмного забезпечення

Рівень	Компонент	Функціональне призначення
Presentation Layer	GUI Module	Формування інтерфейсу, обробка подій, запуск процесів
Application Logic	Controller	Координація обробки, маршрутизація, взаємодія з аналітичними модулями
Application Logic	Analyzer	Аналіз структури UI, виклик платформеного адаптера, обчислення метрик
Application Logic	Report Engine	Побудова фінального звіту у форматі PDF або TXT
Data Storage Layer	JSON Storage	Локальне збереження UI-макетів, результатів аналізу та згенерованих звітів

Запропонована архітектура забезпечує гнучкість масштабування системи у випадку майбутнього розширення функціональності — зокрема, додавання нового формату зберігання, API-доступу або реалізації підтримки додаткових платформ UI. Водночас вона зберігає стабільність функціонування в автономному режимі, мінімізує залежність між модулями, сприяє повторному використанню коду та спрощує контроль версій. Така структурна модель є придатною як для навчального, так і для дослідницького впровадження, а її компонентна організація відповідає сучасним принципам архітектурного проєктування програмного забезпечення.

3.5 Алгоритмізація програмних модулів

У межах реалізації функціональних вимог до системи аналізу графічних інтерфейсів було здійснено повну алгоритмізацію ключових модулів, відповідальних за обробку вхідних даних, обчислення метрик, перевірку відповідності стандартам UI-дизайну, а також за формування висновків і збереження результатів. Побудована блок-схема (рис. 3.4) відображає

узагальнену логіку функціонування основного обчислювального ядра програми та є основою для програмної реалізації.

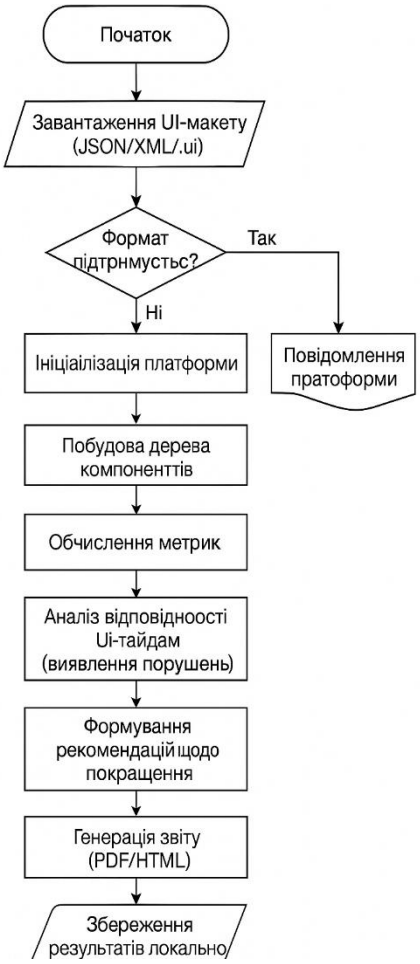

Рисунок 3.4 – Блок-схема логіки роботи програми

Згідно з блок-схемою (рис. 3.4), алгоритм починається з ініціалізації та завантаження UI-макету у підтримуваному форматі. На цьому етапі здійснюється первинна перевірка вхідного файлу, як показано на рис. 3.5.

```
@app.route('/upload', methods=['POST'])
def upload_interface():
    file = request.files.get('file')
    if not file or not file.filename.endswith(('.json', '.xml', '.ui')):
        return jsonify({'error': 'Непідтримуваний формат файлу'}), 400
    uid = str(uuid.uuid4())
    filepath = os.path.join(DATA_DIR, f"{uid}.json")
    file.save(filepath)
    return jsonify({'interface_id': uid, 'message': 'Файл завантажено'}), 200
```

Рисунок 3.5 – Перевірка вхідного файлу

У випадку, якщо формат не підтримується, згідно з умовною гілкою блок-схеми, система формує відповідне повідомлення про помилку й не переходить до наступних кроків.

Якщо формат коректний, відбувається визначення платформи — Android, iOS, Windows або інша. Ідентифікація здійснюється за вмістом структурованого документа, продемонстровано на рис.3.6.

```
def detect_platform(data: dict) -> str:
    platform = data.get('platform', '').lower()
    if 'android' in platform:
        return 'Android'
    elif 'ios' in platform:
        return 'iOS'
    elif 'windows' in platform:
        return 'Windows'
    return 'Unknown'
```

Рисунок 3.6 -Ідентифікація документу

Після ідентифікації платформи виконується побудова ієрархії елементів інтерфейсу. Компоненти аналізуються на предмет вкладеності, позиціонування, типів (кнопки, текстові поля тощо) і просторових властивостей.

Описані нижче функціональні блоки реалізують центральну логіку роботи модулів аналітичної обробки в програмній системі, яка здійснює оцінку графічного інтерфейсу користувача. Послідовність їх виклику повністю відповідає етапам, зображеним на блок-схемі (рис. 3.9), і охоплює обробку компонентів макету, обчислення критичних UI-метрик, перевірку на відповідність інтерфейсним гайдлайнам (Material Design, HIG), формування рекомендацій і генерацію звіту з подальшим збереженням результатів.

Кожен етап реалізовано як окрема функція у вигляді незалежного модуля, що підвищує гнучкість структури та полегшує тестування. Створена логіка забезпечує цілісний цикл обробки вхідного макету: від парсингу його структури до формування підсумкового PDF-звіту з виявленими порушеннями та рекомендаціями щодо покращення.

Нижче наведено інтегрований фрагмент коду який представлений на рис.3.7, що реалізує відповідну функціональність

```python
def parse_components(data: dict) -> list:
    components = []
    for el in data.get('components', []):
        components.append({
            'type': el['type'],
            'depth': el['depth'],
            'x': el['x'],
            'y': el['y'],
            'width': el['width'],
            'height': el['height'],
            'color': el['style'].get('color')
        })
    return components


def calculate_metrics(components: list) -> dict:
    depth = max([c['depth'] for c in components])
    contrast = compute_contrast(components)
    density = compute_density(components)
    return {
        'depth': depth,
        'contrast': contrast,
        'density': density
    }


def validate_ui_guidelines(metrics: dict) -> list:
    warnings = []
    if metrics['contrast'] < 4.5:
        warnings.append('Контраст тексту нижчий за рекомендоване значення
(4.5).')
    if metrics['depth'] > 5:
        warnings.append('Занадто глибока вкладеність UI-компонентів.')
    if metrics['density'] > 0.7:
        warnings.append('Інтерфейс перевантажено елементами.')
    return warnings


def generate_report(interface_id: str, metrics: dict, warnings: list) -> str:
    pdf_path = os.path.join(DATA_DIR, f"{interface_id}_report.pdf")
    pdf = ReportLabBuilder()
    pdf.add_heading("Звіт аналізу інтерфейсу")
    pdf.add_table(metrics)
    pdf.add_list(warnings)
    pdf.save(pdf_path)
    return pdf_path


def save_results(interface_id: str, metrics: dict, warnings: list):
    result_path = os.path.join(DATA_DIR, f"{interface_id}_results.json")
    data = {
        'metrics': metrics,
        'warnings': warnings
    }
    with open(result_path, 'w') as f:
        json.dump(data, f, indent=2)
```

Рисунок 3.7 – Лістинінг коду реалізації основних функціональних елементів програми

Усі зазначені дії чітко структуровані відповідно до етапів алгоритму, поданого на рисунку 3.4, і реалізуються в рамках багаторівневої архітектури програмного забезпечення (див. п. 3.4).

Для формального узагальнення реалізованої логіки використовується таблиця 3.5.

Таблиця 3.5 – Алгоритмічні модулі та їх функціональне призначення

№	Назва модуля	Призначення
1	upload_interface()	Завантаження макету, перевірка формату
2	detect_platform()	Визначення платформи (Android/iOS/Windows)
3	parse_components()	Побудова ієрархічного дерева UI-компонентів
4	calculate_metrics()	Обчислення глибини, контрасту, щільності
5	validate_ui_guidelines()	Перевірка відповідності метрик до вимог UX/UI стандартів
6	generate_report()	Побудова підсумкового звіту (PDF/HTML)
7	save_results()	Збереження результатів аналізу у файлову систему

Запропонований алгоритм є формалізованим, модульним і повністю відображає вимоги, закладені в логіку роботи системи. Усі етапи інтегровані в архітектуру системи з урахуванням принципів інкапсуляції, повторного використання та контролю помилок, що забезпечує стабільність і масштабованість реалізації.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування системи

У процесі розробки програмного забезпечення для аналізу графічних інтерфейсів було проведено повноцінне функціональне тестування з метою перевірки відповідності реалізованої логіки специфікації вимог, описаній у розділі 1.3. Тестування охоплювало ключові компоненти: інтерфейс взаємодії з користувачем, механізми обробки UI-макетів, обчислення метрик, генерацію звітів, а також збереження та читання даних у файловій інформаційній базі. Особливу увагу було приділено перевірці сценаріїв автономного використання, правильності обробки формату .ui та відображенню результатів у графічному інтерфейсі. Усі тести проводилися вручну у середовищі локального запуску застосунку, реалізованого з використанням PyQt6 та Flask.

У таблиці 4.1 представлено результати ключових тестових сценаріїв, які відображають відповідність роботи системи очікуваному функціоналу.

Таблиця 4.1

Результати тестування основних функцій системи

№	Назва тесту	Вхідні дані	Очікуваний результат	Результат
1	Завантаження UI-макету	Файл `layout.ui`	Відображення назви файлу в інтерфейсі	Успішно
2	Запуск аналізу	Завантажений макет	Генерація метрик, запис у файл `*_results.json`	Успішно
3	Виведення таблиці метрик	Структура з результатами	Відображення значень у таблиці GUI	Успішно
4	Формування PDF-звіту	Дані метрик	Створення та збереження файлу `*_report.pdf`	Успішно
5	Перевірка JSON-структури	Дані користувача	Валідність JSON та правильне збереження	Успішно
6	Повідомлення про збереження	Завершення аналізу	Виведення повідомлення «Звіт збережено»	Успішно

Функціональне тестування підтвердило стабільність роботи всіх модулів системи. Зокрема, під час виконання тесту №1 було підтверджено, що після вибору макету у форматі .ui його ім’я автоматично відображається у полі вибраного файлу (див. рис. 4.1).

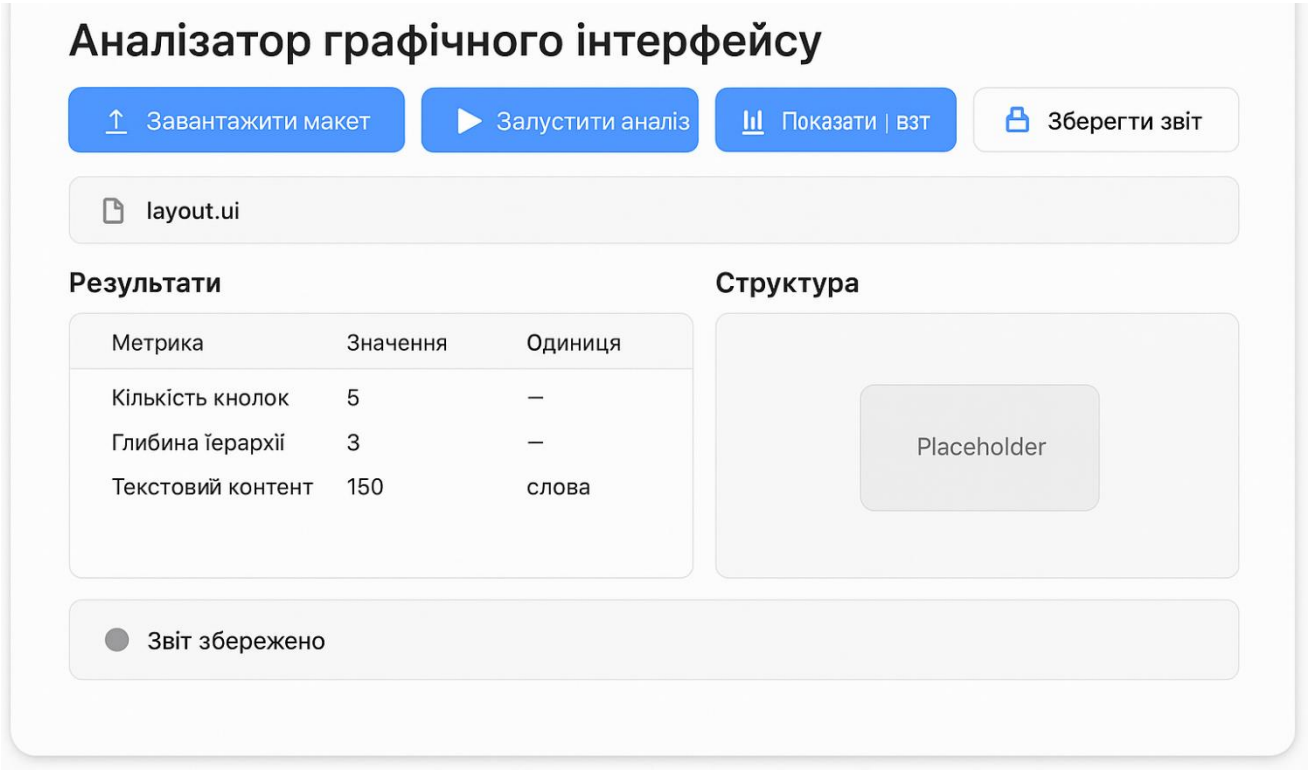

Рисунок 4.1 – Інтерфейс після успішного аналізу UI-макету

У тесті №3 була перевірена коректність відображення розрахованих метрик: кількість кнопок, глибина ієрархії та обсяг текстового контенту виводяться у відповідній таблиці інтерфейсу (див. рис. 4.1, секція «Результати»). При натисканні кнопки «Зберегти звіт» створюється відповідний PDF-файл, а користувач отримує сповіщення про успішне збереження (тест №4). Окремо було протестовано запис даних у форматі JSON із дотриманням структури користувача, зображеної на рис. 4.2.

Рисунок 4.2 – Збережена структура користувача у форматі JSON

Усі ключові етапи обробки даних, передача між модулями та збереження результатів відпрацьовували згідно з вимогами. Не було виявлено критичних помилок, а візуальні компоненти інтерфейсу адаптивно реагували на сценарії взаємодії. Це дозволяє зробити висновок про функціональну готовність системи до використання як інструментального засобу для локального аналізу графічних інтерфейсів.

4.2 Впровадження системи

Впровадження інформаційної системи аналізу графічних інтерфейсів передбачає інсталяцію програмного забезпечення на робочі станції користувачів з подальшим локальним виконанням усіх операцій без необхідності мережевого з'єднання. Система розроблена відповідно до принципів ізольованої клієнтської інсталяції, що забезпечує високу автономність, простоту розгортання та низький поріг для первинної інтеграції. Уся логіка реалізована за допомогою скриптів Python із використанням PyQt6, а дані зберігаються у структурованих JSON-документах, які зчитуються та зберігаються у локальному сховищі.

На рисунку 4.3 подано модель розгортання системи у вигляді UML-діаграми, де візуалізовано основні компоненти середовища виконання, взаємодію з користувачем і зв'язки між модулями. Користувач взаємодіє із застосунком через графічний інтерфейс gui.py, який виконується всередині

середовища Python Interpreter. Основні функціональні блоки – controller.py, analyzer.py, report_engine.py та json_storage.py – працюють у рамках одного середовища виконання та забезпечують аналіз, генерацію звітів і файлову обробку. Зовнішнім джерелом збереження виступає LocalStorage, де у форматі *.json зберігаються як вхідні макети, так і результати аналізу.

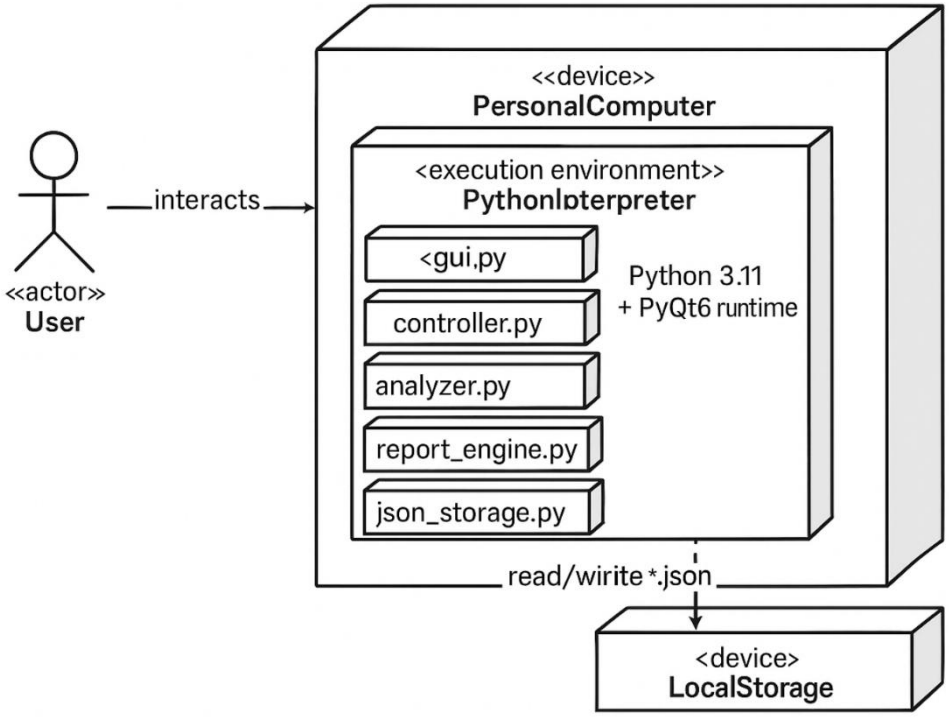

Рисунок 4.3 – Діаграма розгортання програмної системи

Для впровадження системи необхідно лише мати встановлений інтерпретатор Python версії 3.11 або новішої, а також залежності PyQt6 і стандартні модулі Python (uuid, os, json, datetime). Усі файли програми об'єднані у вигляді окремого каталогу, який може бути скопійований на будь-який комп’ютер користувача без змін. Така модель не потребує зовнішніх API, серверних баз даних чи адміністрування середовища, що значно спрощує процес поширення системи у малих або експериментальних організаціях. У таблиці 4.2 подано мінімальні вимоги до середовища для запуску системи.

Таблиця 4.2

Технічні вимоги для впровадження системи

Параметр	Мінімальне значення
Операційна система	Windows 10+, Linux (Ubuntu 20.04+), macOS
Python	Версія 3.11 або новіша
Графічна бібліотека	PyQt6
Обсяг оперативної пам’яті	2 ГБ
Місце на диску	50 МБ для програми + 10 МБ/макет
Права доступу	Локальний доступ до файлової системи

Завдяки модульному підходу система може бути легко перенесена на інші платформи, а також розширена шляхом додавання нових функціональних модулів без зміни основної архітектури. У разі потреби система може бути інтегрована в більші платформи аналізу як допоміжний локальний модуль для попередньої перевірки або генерації інтерфейсних звітів без підключення до серверного середовища. Таким чином, обраний підхід до впровадження забезпечує як функціональну ефективність, так і адаптивність до різних умов використання.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмної системи аналізу графічних інтерфейсів сформовано у вигляді локального каталогу з усіма необхідними файлами, модулями, залежностями та супровідною документацією. Такий формат дозволяє розгорнути систему на будь-якому сумісному пристрої без необхідності попереднього налаштування середовища або встановлення додаткових компонентів. Пакет є повністю автономним, не вимагає доступу до мережі та може бути розгорнутий вручну або за допомогою скрипту запуску. Усі файли структуровано за принципами функціонального розділення, що забезпечує простоту підтримки та внесення змін у подальшому.

У таблиці 4.3 представлено перелік основних складових інсталяційного пакету, а також їх призначення у системі.

Таблиця 4.3

Склад інсталяційного пакету програмної системи

Назва файлу / каталогу	Призначення
`main.py`	Головний файл запуску програми
`gui.py`	Модуль графічного інтерфейсу (PyQt6)
`controller.py`	Логіка маршрутизації подій між модулями
`analyzer.py`	Обробка макетів, обчислення метрик
`report_engine.py`	Генерація звітів у PDF або текстовому форматі
`json_storage.py`	Зчитування та збереження структурованих даних у форматі JSON
`requirements.txt`	Список залежностей для встановлення середовища
`data_storage/`	Каталог для зберігання завантажених макетів, результатів, звітів
`assets/`	Іконки, ресурси, стилі для візуального інтерфейсу
`README.md`	Коротка інструкція користувача та опис структури програми
`start.bat` / `start.sh`	Скрипти для запуску застосунку в середовищі Windows / Linux / macOS

У разі використання на сторонніх пристроях або в умовах навчального середовища, інсталяційний пакет може бути поширений у вигляді архіву (.zip або .tar.gz) або інтегрований у оболонку з компіляцією за допомогою інструментів, таких як PyInstaller або cx_Freeze, що забезпечують створення виконуваного файлу (.exe або .app). Структурованість пакету дозволяє легко виявляти, змінювати або доповнювати окремі компоненти, а також інтегрувати нові модулі з мінімальним втручанням у базову архітектуру. Завдяки мінімальній кількості зовнішніх залежностей та локальній обробці усіх даних, система забезпечує швидке розгортання, зручність супроводу та гнучкість адаптації до різних платформ.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено та впроваджено інформаційну систему для аналізу графічних інтерфейсів мобільних та десктопних додатків, яка дозволяє здійснювати автоматизовану обробку UI-макетів, обчислення метрик структури інтерфейсу, генерацію аналітичних звітів та локальне збереження результатів у стандартизованому форматі. Актуальність обраної теми обумовлена необхідністю підвищення ефективності оцінювання якості користувацьких інтерфейсів у проєктуванні програмного забезпечення, а також відсутністю універсальних автономних інструментів для попереднього аналізу макетів у середовищі розробника.

У першому розділі було виконано системний аналіз предметної області, проведено огляд існуючих рішень, сформульовано функціональні й нефункціональні вимоги до системи, а також побудовано логічну модель домену на основі типових сценаріїв користувацької взаємодії. На основі цього було визначено архітектурний підхід до реалізації, зокрема використання трирівневої структури з поділом на презентаційний рівень, рівень прикладної логіки та рівень збереження даних. У другому розділі здійснено проєктування інформаційного і програмного забезпечення системи: створено UML-діаграми, ER-модель, діаграми класів, кооперації, компонентів та розгортання, які в сукупності забезпечують цілісне уявлення про внутрішню організацію програмного продукту.

У третьому розділі реалізовано інформаційну базу системи у форматі локального JSON-сховища, обґрунтовано вибір технологічного стеку (Python 3.11, PyQt6, Flask, ReportLab), створено функціональну структуру каталогів, а також сформовано повноцінний інсталяційний пакет із супровідною документацією. На основі отриманих вимог і архітектурного шаблону було реалізовано компоненти інтерфейсу, контролера, модуля обробки макету,

генератора звітів і модуля збереження. У четвертому розділі проведено тестування реалізованої системи, яке підтвердило її працездатність у всіх ключових сценаріях: завантаження макетів, обробка структури, виведення метрик, формування та збереження результатів.

Таким чином, у дипломній роботі виконано повний цикл створення прикладної інформаційної системи — від аналізу предметної області до реалізації функціонального прототипу з підтримкою автономної роботи, що може бути використаний у практичних задачах UI-аналізу, освітніх цілях або як основа для подальшого наукового дослідження в галузі програмної інженерії. Результати роботи мають практичну цінність і можуть бути адаптовані до використання у командах розробників, UI/UX-аналітиків або інтегровані в комплексні системи перевірки інтерфейсів.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про інформацію» від 02.10.1992 № 2657-XII. URL: https://zakon.rada.gov.ua/laws/show/2657-12

2. ISO 9241-210:2010. Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems.

3. Шевченко С. В. Людино-машинна взаємодія: концепції, методи, практики. — К.: ДУІТ, 2020. — 256 с.

4. Смирнов С. А. Основи UX-дизайну: підходи, моделі, патерни. — Харків: ХНУРЕ, 2019. — 198 с.

5. Буч К. Основи проектування інтерфейсів. — К.: Либідь, 2021. — 272 с.

6. Веб-сайт Human Interface Guidelines (Apple Inc.). URL: https://developer.apple.com/design/human-interface-guidelines/

7. Веб-сайт Material Design (Google). URL: https://m3.material.io/

8. Nielsen J. Usability Engineering. — Boston: Morgan Kaufmann, 1994. — 362 p.

9. Norman D. The Design of Everyday Things. — Revised and Expanded Edition. — New York: Basic Books, 2013. — 384 p.

10. Бондаренко В. В., Струк І. В. Розробка кросплатформених GUI-додатків з використанням PyQt. // Вісник ХНУРЕ. — 2022. — №2(81). — С. 45–51.

11. Khamis M., Alt F., Bulling A. A survey on gaze interaction in mobile settings: challenges and opportunities. // ACM Computing Surveys (CSUR). — 2018. — Vol. 51, №4. — P. 1–33.

12. Мельник О. М. Особливості проєктування мобільних застосунків у порівнянні з десктопними: огляд практик. // Проблеми програмування. — 2023. — №2. — С. 122–129.

13. Веб-сайт Microsoft Fluent Design System. URL: https://learn.microsoft.com/en-us/windows/apps/design/

14. Ткаченко А. І., Боярчук А. В. Аналіз графічних рішень у розробці програм для ОС Android та Windows. // Наукові записки. — 2021. — №34. — С. 57–63.

15. Веб-сайт Figma — сучасне середовище для проєктування UI. URL: https://www.figma.com/

16. Веб-сайт Qt Documentation — GUI Design. URL: https://doc.qt.io/qt-6/topics-ui.html

17. Щербак Л. В. Взаємодія користувача з програмною системою: психологічні та технічні аспекти. — К.: КНЕУ, 2020. — 148 с.

18. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

19. Веб-сайт W3C Accessibility Guidelines. URL: https://www.w3.org/WAI/standards-guidelines/wcag/

20. Саєнко Р. В., Жарова Л. О. Порівняльна характеристика мобільних і десктопних UI-платформ. // Інформаційні технології в освіті. — 2022. — №45. — С. 91–98.