Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
(підпис)
«___»_____2025р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «Фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: «Розробка кросплатформного графічного застосунку»

Студент групи КН-41	Ярослав БІЛИК	_____ (підпис)
Керівник роботи	Надія ГАВРИШКІВ	_____ (підпис)
Консультанти: з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Наталя КУЛЬЧИНСЬКА	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

(підпис)

«___»_____2024р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студенту Білику Ярославу Анатолійовичу

1. Тема роботи Розробка кросплатформного графічного застосунку затверджено наказом по коледжу від “25” листопада 2024 р., № 253а-н
2. Термін здачі студентом завершеного проєкту “___” _____ 2025 р.
3. Вихідні дані до проєкту Результати дослідження кросплатформних графічних застосунків, аналіз ринку таких додатків та технологій їхньої реалізації
4. Перелік питань, які повинні бути розроблені:

а) основна частина: аналіз предметної області та постановка завдань; проєктування системи; реалізація та тестування системи.

б) техніко-економічнеобгрунтування: аналіз ринку; розрахунок витрат на проєктування; обґрунтування необхідності розробки.

5. Перелік графічного матеріалу: діаграма компонентів; діаграма послідовності; діаграма класів;

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
3 техніко-економічного обґрунтування	Меленчук Л. І. (вчена ступінь, звання П.І.Б _____ (консультант)		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, аналіз вимог до кваліфікаційної роботи	16.11.2024	25.11.2024
2.	Дослідження наявних графічних застосунків	26.11.2024	06.12.2024
3.	Розробка функціональних та нефункціональних вимог реалізації застосунку	05.12.2024	07.12.2024
4.	Дослідження технологій реалізації	08.12.2024	12.12.2024
5.	Встановлення бібліотек та налаштування середовища розробки	13.12.2024	14.12.2024
6.	Проєктування системи	15.12.2024	26.12.2024
7.	Реалізація основної логіки графічного застосунку	27.12.2024	02.05.2025
8.	Тестування застосунку, написання розділу реалізації	03.05.2025	09.05.2025
9.	Написання техніко-економічного обґрунтування	10.05.2025	11.05.2025
10.	Оформлення пояснювальної записки	12.05.2025	13.06.2025
11.	Попередній захист кваліфікаційної роботи	13.06.2025	13.06.2025
12.	Підготовка до захисту кваліфікаційної роботи	14.06.2025	23.06.2025
13.	Захист кваліфікаційної роботи	24.06.2025	24.06.2025

Дата видачі “___” _____ 2024 р. Керівник _____/ Гавришків Н.Г.

Завдання прийняв до виконання _____/ Білик Я.А.

Реферат

Кросплатформний графічний застосунок для створення зображень. Кваліфікаційна робота. Білик Ярослав. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. Спеціальність 122 «Комп’ютерні науки», 2025. Сторінок – 53, рисунків – 17, додатків – 45.

Об’єкт дослідження – програмні засоби для обробки растрової графіки та методи реалізації кросплатформних графічних застосунків.

Метою роботи є реалізація сучасного графічного застосунку для створення зображень із багатошаровою структурою підтримкою інструментів малювання та інтерактивним інтерфейсом користувача.

У роботі використовувалися мова програмування C++ і мова розмітки QML у поєднанні з фреймворком Qt. У якості середовища розробки застосовувались Neovide та система збірки CMake. Для керування версіями проєкту використовувався Git.

Результатом роботи став повноцінний застосунок із чіткою модульною архітектурою, який може бути використаний як основа для розширеного графічного редактора в майбутньому.

ГРАФІЧНИЙ ЗАСТОСУНОК, C++, QML, GIT, ПОЛОТНО, JSON, OPEN SOURCE, PROVIDER, , NEOVIDE, КРОСПЛАТФОРМНІСТЬ.

Abstract

Cross-platform graphic application for creating images. Qualification work. Yaroslav Bilyk. Vyacheslav Chornovil Galician Professional College, Department of Computer Technologies. Speciality 122 ‘Computer Science’, 2025. Pages – 53, figures – 17, appendices – 45.

The object of research is software for processing raster graphics and methods for implementing cross-platform graphics applications.

The aim of the work is to implement a modern graphics application for creating images with a multi-layer structure, support for drawing tools and an interactive user interface.

The work used the C++ programming language and the QML markup language in combination with the Qt framework. Neovide and the CMake build system were used as the development environment. Git was used for project version control.

The result of the work is a full-fledged application with a clear modular architecture that can be used as the basis for an advanced graphics editor in the future.

GRAPHICAL APPLICATION, C++, QML, GIT, CANVAS, JSON, OPEN SOURCE, PROVIDER, NEOVIDE, CROSS-PLATFORM.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface

JSON – JavaScript Object Notation

MVVM – Model-View-ViewModel

PDF – Portable Document Format

QML – Qt Modeling Language

SVG – Scalable Vector Graphics

ВСТУП

Особлива увага сьогодні приділяється створенню кросплатформних графічних додатків, які забезпечують зручність і ефективність роботи користувачів незалежно від операційної системи. Фреймворк Qt завдяки своїй універсальності, високій продуктивності та підтримці різних платформ став одним із найпоширеніших інструментів для розробки таких рішень. З огляду на зростаючу потребу в подібних додатках, ця тема залишається актуальною та має значну практичну цінність. Графічні додатки знаходять широке застосування в різних сферах, таких як бізнес, інженерія та розробка програмного забезпечення. Qt дозволяє створювати ефективні користувацькі інтерфейси, які вирізняються продуктивністю та гнучкістю. Незважаючи на значний обсяг досліджень у цій галузі, все ще існують виклики, пов’язані з оптимізацією продуктивності, інтеграцією з новітніми технологіями та вдосконаленням наявних підходів. Мета цієї кваліфікаційної роботи полягає в розробці кросплатформного графічного додатку з використанням фреймворку Qt. У процесі роботи планується проаналізувати існуючі рішення у сфері кросплатформної розробки, дослідити можливості Qt та інструментів для створення графічних інтерфейсів, розробити архітектуру додатку, застосувати кращі практики програмування, реалізувати основні функції, а також провести тестування й оптимізацію продуктивності. Крім того, будуть проаналізовані отримані результати та визначені перспективи подальшого розвитку. Основний акцент зроблено на методах і інструментах фреймворку Qt, які сприяють створенню продуктивних і зручних користувацьких інтерфейсів. Розроблений додаток може бути застосований в таких областях, як створення дизайну, створення зображень, редагування зображень. Отримані результати стануть у пригоді користувачам, які шукають гнучкі та ефективні програмні рішення.

Таким чином, ця робота вносить важливий вклад у сферу кросплатформної розробки, акцентуючи увагу на сучасних технологіях і підходах, що підвищують продуктивність і зручність графічних додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Опис предметної області

Сучасне програмування все частіше вимагає створення застосунків, які бездоганно працюють на різних операційних системах із єдиною кодовою базою. Процес розробки охоплює проєктування інтерфейсу користувача, написання логіки програми, інтеграцію з системними ресурсами, тестування та оптимізацію продуктивності. У світі, де користувачі очікують однаково якісного досвіду на Windows, macOS та на Linux, створення кросплатформенних рішень набуває стратегічного значення.

Розробка таких програм стикається з кількома викликами. Багато розробників не мають повного розуміння можливостей фреймворку, його бібліотек і інструментів. Складність проєктів зростає через різноманітність апаратного забезпечення, операційних систем і вимог до продуктивності. Відсутність систематичного підходу до проєктування може призводити до надмірного ускладнення коду та зниження його масштабованості.

У сучасному ритмі розробки програмного забезпечення ручне налаштування коду для кожної платформи є трудомістким і часто призводить до помилок. Водночас багато розробників віддають перевагу ручному контролю над кодом, щоб уникнути непередбачуваних результатів автоматичної генерації чи через недовіру до повністю абстрагованих рішень. Для таких фахівців критично важливо мати зручний набір інструментів, який полегшує розробку, мінімізує ризик помилок і забезпечує гнучкість у налаштуванні.

Основні процеси, які потребують оптимізації, включають:

– Проєктування зручного та адаптивного графічного інтерфейсу для всіх платформ.

– Розробку універсальної логіки, що забезпечує стабільну роботу програми.

– Налаштування доступу до унікальних можливостей кожної операційної системи.

– Організацію ефективного тестування в різних середовищах.

– Підвищення швидкості роботи й раціонального використання ресурсів.

– Створення чіткої документації для полегшення подальшої підтримки.

Розробка інтерфейсу вимагає створення єдиного дизайну, який адаптується до особливостей кожної платформи, що потребує від розробника уважності й послідовності. Кросплатформна логіка залежить від інструментів, які дозволяють швидко виявляти платформозалежні ділянки коду. Інтеграція з системними функціями передбачає роботу з файлами, мережею чи апаратними ресурсами з урахуванням специфіки ОС. Тестування потребує аналізу поведінки програми в різних умовах і чіткої візуалізації результатів.

Сучасні технології сприяють створенню таких застосунків. Завдяки потужним комп’ютерам і хмарним сервісам тестування й розгортання стають швидшими. Інструменти, такі як Qt Creator і QML, дозволяють розробляти інтерактивні інтерфейси та координувати роботу команд. Однак основною проблемою залишається брак зручного й ефективного підходу, який би спрощував написання коду, надавав повну видимість поведінки програми на різних платформах і допомагав створювати якісні рішення.

Запропонований підхід спрямований на розв’язання цієї проблеми через створення зручного середовища для розробки інтерфейсів, використання шаблонів для типових завдань, інструментів для аналізу продуктивності та спрощення тестування. Це дає змогу підвищити якість коду, краще розуміти поведінку програми, оптимізувати ресурси, швидше досягати цілей, зменшувати стрес від адаптації до кількох платформ і покращувати користувацький досвід, зберігаючи при цьому контроль над процесом розробки.

1.2 Аналіз наявних рішень

Сучасні потреби в розробці графічного контенту та підтримці кросплатформності підштовхують програмістів до використання потужних інструментів для роботи з графікою. Особливу роль відіграють програми, що дають змогу користувачам зручно створювати й редагувати графічні проєкти на будь-яких операційних системах.

«Krita» - є одним із відомих кросплатформних застосунків, який пропонує зрозумілий інтерфейс, широкий набір інструментів для цифрового малювання та підтримує численні формати файлів. Його розробила команда спеціалістів під егідою Krita Foundation, забезпечивши сумісність із Windows, macOS і Linux.

Серед особливостей «Krita» — безкоштовність і відкритий код, що дозволяє спільноті долучатися до розробки, а також наявність таких функцій, як різноманітні пензлі, стабілізатори ліній і інструменти для анімації. Інтерфейс можна налаштовувати, а програма працює стабільно навіть на середньому обладнанні.

Однак є й певні обмеження: велика кількість функцій може ускладнити освоєння для новачків, підтримка векторної графіки поступається спеціалізованим застосункам, а в нових версіях іноді виникають помилки, що потребує регулярних оновлень. Загалом, «Krita» поєднує широкі можливості з певними недоліками, які варто враховувати при її використанні (рис. 1.1).

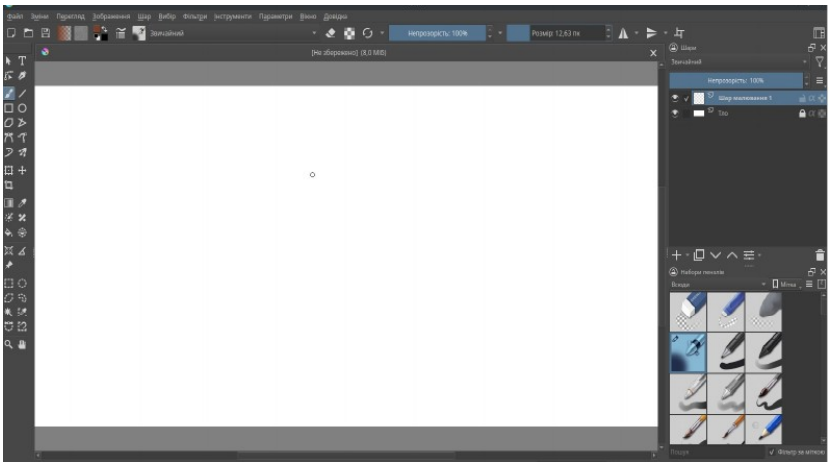

Рисунок 1.1 – Графічний застосунок Krita

«Inkscape» – це кросплатформенний редактор SVG векторної графіки з відкритим кодом, який забезпечує інтуїтивно зрозумілий інтерфейс та широкий набір інструментів для створення і редагування кривих. Розроблений командою Inkscape Developers, цей застосунок підтримує операційні системи Windows, macOS і Linux, що робить його доступним для широкого кола користувачів.

Серед основних переваг «Inkscape» — безкоштовність, підтримка відкритих форматів, таких як SVG, а також гнучкість у роботі з векторною графікою. Програма містить інструменти для створення складних фігур, роботи з шарами, використання градієнтів, ефектів і фільтрів. Вона підтримує імпорт та експорт у різні формати (PNG, PDF, EPS та інші), що забезпечує сумісність із професійними графічними редакторами.

Попри свої переваги, «Inkscape» має й певні недоліки. Програма може працювати повільно при обробці великих файлів або складних ілюстрацій, а деякі інструменти можуть здаватися менш зручними у використанні порівняно з комерційними альтернативами. Крім того, хоча інтерфейс є зрозумілим, він може виглядати дещо застарілим, а деякі функції потребують звикання. Проте для багатьох дизайнерів, художників і технічних ілюстраторів «Inkscape» залишається надійним вибором завдяки своїм широким можливостям і активній спільноті розробників (рис. 1.2).

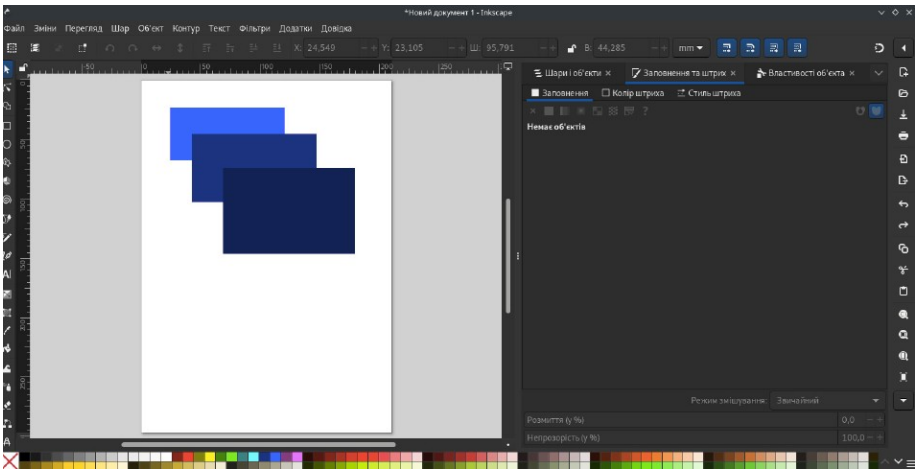

Рисунок 1.2 – Графічний застосунок Inkscape

GIMP – це потужний редактор растрової графіки з відкритим кодом, розроблений спільнотою під егідою проєкту GNU. Він підтримує операційні системи Windows, macOS і Linux, що робить його доступним для різних категорій користувачів, від любителів до професійних дизайнерів.

Основними перевагами GIMP є безкоштовність та відкритий код, що дозволяє спільноті постійно вдосконалювати програму та створювати власні доповнення. Програма підтримує численні формати файлів, зокрема PNG, JPEG, PSD (Adobe Photoshop), XCF та багато інших, забезпечуючи сумісність із різними графічними редакторами. GIMP пропонує широкий набір інструментів для редагування, включаючи пензлі, фільтри, шари, маски та засоби для корекції кольорів. Програма є гнучкою та розширюваною завдяки підтримці плагінів і скриптів, що дозволяє автоматизувати багато завдань та адаптувати програму під власні потреби. Активна спільнота розробників та користувачів регулярно випускає оновлення, що покращують стабільність роботи та додають нові можливості.

Попри численні переваги, GIMP має й певні недоліки. Інтерфейс програми може здаватися складним для новачків, особливо для тих, хто звик до Adobe Photoshop, оскільки логіка роботи деяких інструментів відрізняється. Крім того, GIMP має обмежену підтримку роботи з CMYK, що може створювати труднощі для дизайнерів, які працюють із поліграфічною продукцією. Деякі функції, зокрема розширене редагування та ефекти, потребують встановлення додаткових плагінів, що може ускладнити робочий процес. Також при обробці великих файлів продуктивність програми може знижуватися, що впливає на швидкість виконання складних операцій.

Незважаючи на усе перечислене, GIMP залишається одним із найпопулярніших безкоштовних інструментів для редагування зображень, ретуші фотографій та створення цифрового мистецтва, пропонуючи користувачам широкий спектр можливостей для творчої роботи (рис. 1.3).

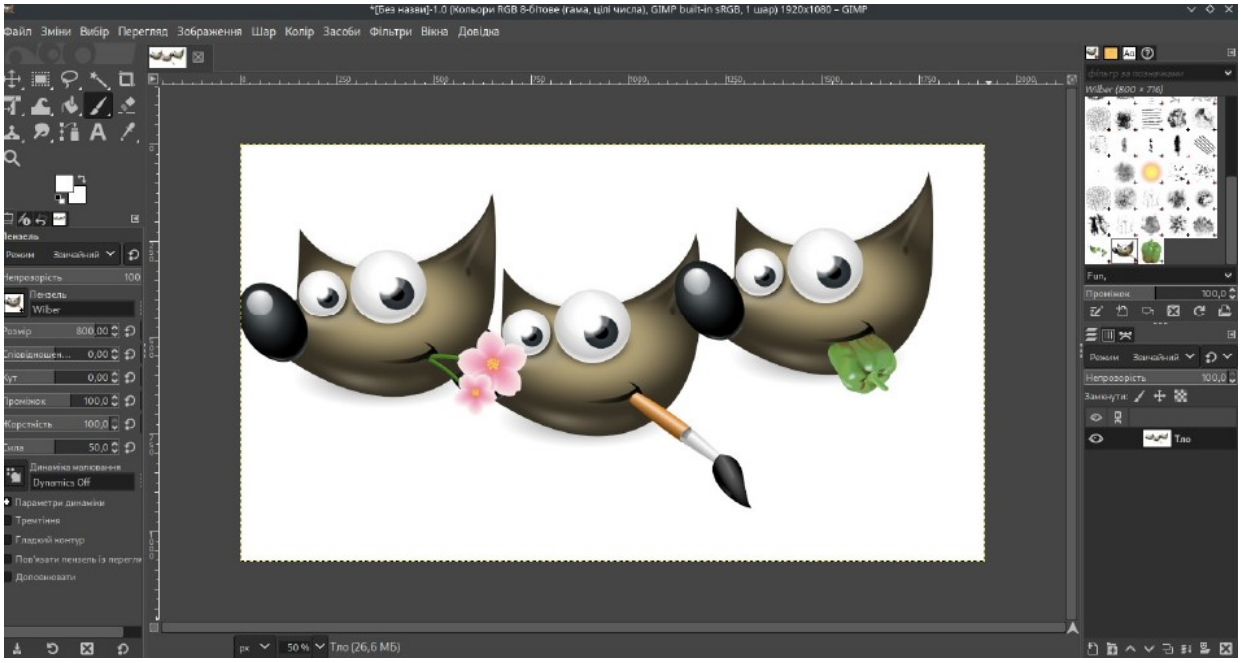

Рисунок 1.3 – Графічний застосунок GIMP

1.3 Аналіз вимог до програмного засобу та постановка завдання

На основі аналізу потреб у сфері кросплатформної розробки було виявлено потребу в створенні зручного інструменту для розробки графічних додатків, які однаково ефективно працюють на різних операційних системах. Метою цього проєкту є створення кросплатформного графічного додатку з використанням фреймворку Qt, який надасть користувачам зручний інтерфейс, високу продуктивність і можливість адаптації до їхніх потреб незалежно від платформи.

Розроблюваний додаток призначений для спрощення роботи користувачів шляхом створення інтуїтивного графічного інтерфейсу, який підтримує введення даних, їх обробку та візуалізацію результатів. Система має забезпечувати легкість використання, швидке реагування на дії користувача та гнучкість налаштувань для роботи на Windows, macOS, Linux та інших платформах.

Функціональні вимоги до програми включають кілька ключових аспектів кросплатформної розробки. У частині графічного інтерфейсу система повинна забезпечувати створення адаптивного дизайну, вибір тем оформлення,

налаштування розмірів вікон і підтримку різних роздільних здатностей екранів. Для обробки даних додаток має надавати можливість введення інформації, створення шаблонів для повторюваних операцій, редагування та категоризацію введених даних із швидким доступом до основних функцій.

Нефункціональні вимоги стосуються зручності, продуктивності, безпеки та сумісності. Інтерфейс має бути інтуїтивно зрозумілим і адаптивним до різних пристроїв, додаток — запускатися швидко й працювати без затримок.

Основними користувачами додатку є люди, які потребують зручного кросплатформного рішення. Серед них можна виділити: початківців, які шукають прості інструменти для роботи; досвідчених розробників, які аналізують продуктивність і тестують складні проєкти; компанії, що потребують універсальних рішень для командної роботи; а також ентузіастів, які експериментують із графічними додатками.

Завдання розробки полягає в створенні кросплатформного графічного додатку на базі Qt, який дозволить користувачам адаптувати інтерфейс до своїх потреб. Додаток має бути продуктивним, безпечним і відповідати запитам різних груп користувачів. Успішна реалізація цього проєкту стане цінним внеском у розвиток графічних рішень, підвищуючи ефективність роботи користувачів і демонструючи потенціал фреймворку Qt.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування структури застосунку

Структура кросплатформного графічного застосунку для цифрового малювання буде побудована відповідно до сучасних принципів модульної розробки з чітким розподілом функціональності за окремими компонентами. Архітектура застосунку орієнтована на багаторазове використання коду та його масштабованість, із підтримкою як десктопних (Windows, Linux, macOS)платформ. Основною моделлю архітектури стане поєднання C++ та QML, де C++ відповідатиме за логіку та взаємодію з апаратними ресурсами, а QML — за побудову гнучкого та адаптивного інтерфейсу користувача.

Для впорядкування коду проєкту буде створено низку основних модулів: логіка взаємодії з полотнами, обробка шарів, інструменти малювання, компоненти інтерфейсу користувача, конфігурація середовища, обробка зображень та системні ресурси. Така структура дозволить забезпечити логічне розділення відповідальностей, мінімізувати зв’язки між модулями та суттєво спростити процес розробки, тестування і подальшого розширення функціоналу.

Компоненти, що відповідають за керування шарами, надаватимуть користувачеві можливість створювати, дублювати, переміщувати, видаляти та змінювати порядок шарів. Це дозволить будувати складні графічні композиції без втрати зручності. Інструменти малювання включатимуть кисті, ластик, фігури, заливку та інші базові інструменти, які будуть реалізовані на C++ для досягнення високої продуктивності.

Інтерфейс користувача формуватиметься з окремих QML-компонентів, які відповідатимуть за певні частини UI — панелі інструментів, бокові меню, індикатори, елементи навігації тощо. Для зручності стилізації буде передбачено систему тем оформлення, яка дозволить користувачам перемикатися між різними візуальними стилями. Компоненти інтерфейсу

також підтримуватимуть адаптивну верстку, що забезпечить коректне відображення на екранах різних розмірів.

Графічні ресурси застосунку (іконки, шрифти, зображення) керуватимуться централізовано за допомогою механізму ресурсних файлів Qt (QRC), що забезпечить легке оновлення вмісту без змін у структурі коду.

Модуль обробки зображень відповідатиме за збереження, експорт та конвертацію файлів. Користувач зможе зберігати свої роботи у внутрішньому форматі застосунку або експортувати до популярних форматів (PNG, JPEG тощо).

Уся структура проєкту буде організована відповідно до принципів високої модульності та слабкої зв’язаності між компонентами. Кожен функціональний блок відповідатиме за окрему частину логіки, що дозволить легко орієнтуватися у коді, підтримувати його в актуальному стані, розширювати функціонал у майбутньому та забезпечити стабільну роботу застосунку на різних платформах. На рисунку 2.1 зображено діаграму компонентів, яка ілюструє основні частини системи та їхню взаємодію.

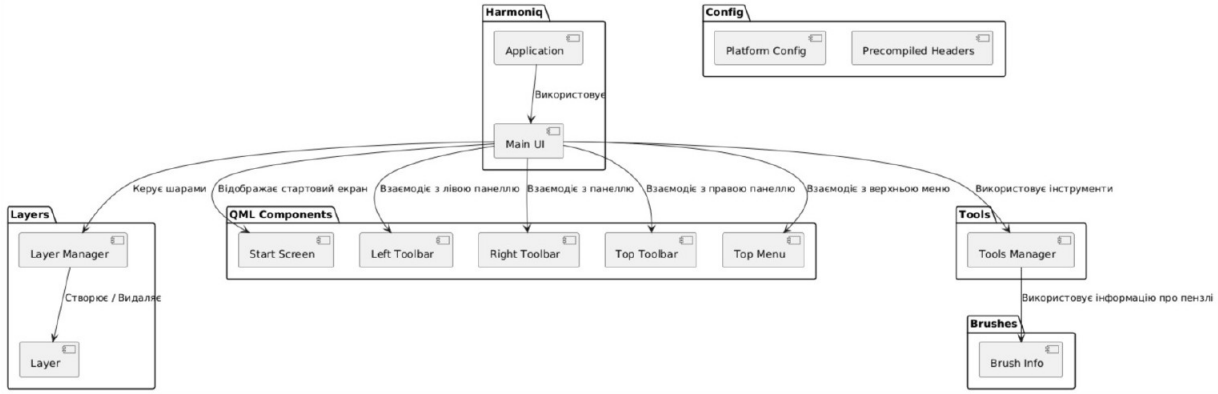

Рисунок 2.1 – Діаграма послідовності

Запропонована структура не лише гарантує дотримання високих стандартів якості програмного коду, але й створює надійну основу для ефективної колективної розробки, оперативного внесення змін та адаптації застосунку до змінних вимог користувачів.

2.2 Проєктування архітектури застосунку

Під час проєктування графічного застосунку планується використання модульного підходу з інтеграцією елементів архітектури MVC (Model-View-Controller) для чіткої організації коду та розділення логіки відображення й обробки даних. Графічний інтерфейс реалізовуватиметься за допомогою QML, що дозволить створити динамічний та адаптивний UI, а обробка графіки здійснюватиметься за допомогою вбудованих Qt класів та пов’язаних з ним механізмів.

Використання QML-компонентів як основних будівельних блоків користувацького інтерфейсу надасть суттєві переваги у вигляді кращої модульності та гнучкості. Це дозволить незалежно розробляти та тестувати окремі частини інтерфейсу графічного редактора, а також ефективно перевикористовувати компоненти в різних контекстах. Наприклад, компонент для управління шарами можна використовувати як у панелі інструментів, так і в окремому вікні редагування.

Інтеграція об'єктно-орієнтованого підходу з використанням менеджерів ресурсів забезпечить ефективне управління графічними даними та відображенням. Кожен інструмент (наприклад, пензель або гумка) працює з відповідними класами, що відповідають за обробку параметрів та взаємодію з полотном. Такий підхід дозволить гнучко керувати зображенням, забезпечуючи швидке оновлення без необхідності зберігати стан у глобальних структурах. Оскільки інструменти керують лише відображенням, збереження проєкту може здійснюватися через механізм експорту у файли різних форматів.

Передача даних між модулями застосунку здійснюватиметься за допомогою сигналів і слотів Qt, що дозволить ефективно реагувати на дії користувача та оновлювати інтерфейс у реальному часі.

Основним механізмом обробки графіки буде використання QPainter, що забезпечить швидке малювання на різних типах поверхонь. Також у проєкті передбачена можливість роботи з багатошаровими зображеннями, для чого

реалізовано систему керування шарами, кожен з яких є автономним і може зберігати власне зображення в пам’яті.

На рисунку 2.2 зображена діаграма послідовності, яка демонструє взаємодію між користувачем, компонентами інтерфейсу, менеджером шарів та інструментами редагування під час роботи із зображенням.

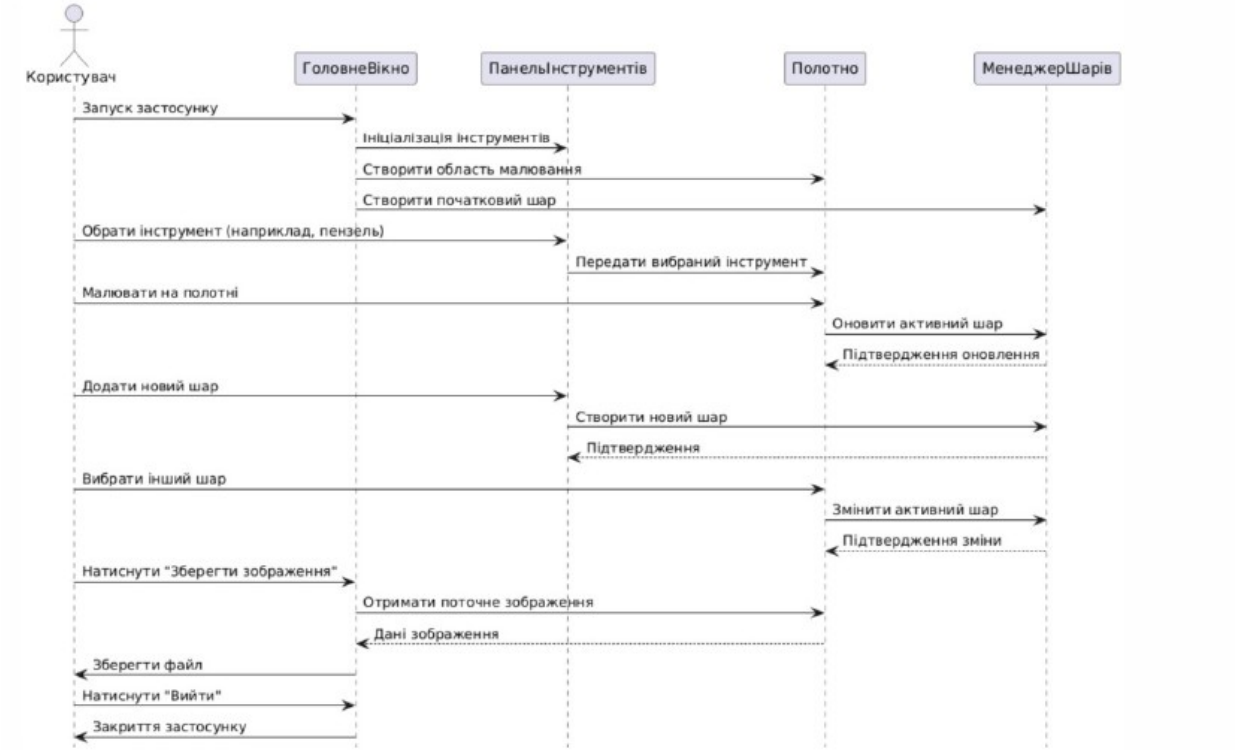

Рисунок 2.2 – Діаграма послідовності

У програмі реалізовано чітку об'єктно-орієнтовану структуру, яка базується на взаємодії між основними класами, зокрема Application, LayerManager, Layer та Tools.

Клас Application відповідає за ініціалізацію та запуск основної логіки застосунку, а також координує роботу з іншими компонентами системи. Через нього здійснюється взаємодія з класами LayerManager та Tools.

Клас LayerManager відповідає за керування всіма шарами. Він зберігає колекцію об'єктів класу Layer, дозволяє додавати нові шари, видаляти або отримувати доступ до конкретного шару за ідентифікатором. Такий підхід

дозволяє легко масштабувати проєкт при зростанні кількості шарів у документі.

Клас Layer містить основні параметри одного шару, зокрема: ідентифікатор, назву шару, статус видимості та блокування. Це дозволяє здійснювати незалежне керування кожним шаром, що є важливим для гнучкого графічного редагування.

Клас Tools реалізує набір інструментів для взаємодії з шарами (наприклад, пензель, гумка, тощо). Він дозволяє обрати активний інструмент і застосувати його до певної області зображення.

На рисунку 2.3 представлена діаграма класів, яка ілюструє основні сутності, їх атрибути, методи та взаємозв'язки між ними. Така структура дозволяє досягти високого рівня модульності, розширюваності та підтримуваності системи, що є ключовим аспектом для сучасних кросплатформених графічних застосунків.

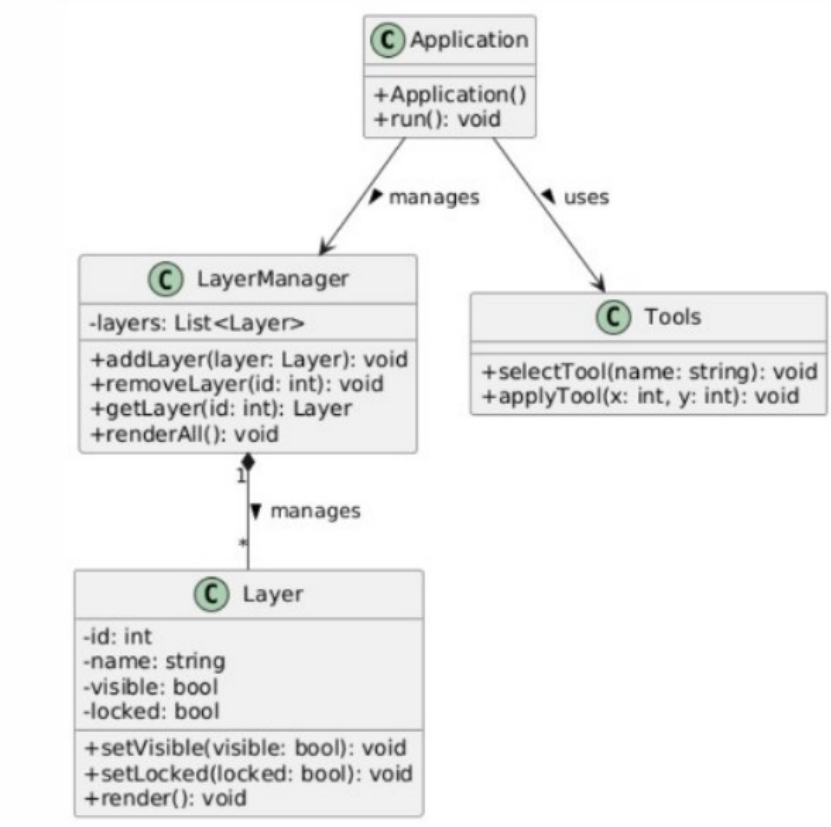

Рисунок 2.3 – Діаграма класів

Загалом, у програмі буде реалізована гнучка та масштабована архітектура, що забезпечить зручне керування шарами, інструментами та взаємодію між ними. Такий підхід дозволить легко розширювати функціональність застосунку, спрощуючи роботу користувачів із графічними елементами.

2.3 Проєктування функціональних модулів

Модуль керування полотнами забезпечував створення, редагування, видалення, переміщення та відображення окремих незалежних елементів зображення. Кожне полотно мало власні властивості: розміри, прозорість, фоновий колір, видимість, стан блокування та активності. Один з елементів завжди вважався активним, і лише з ним відбувалася взаємодія інструментів. Менеджер полотен також відповідав за збереження порядку їх накладання та передачу обраного інструменту до активного полотна.

Модуль документів відповідав за організацію роботи з кількома проєктами одночасно. Кожен документ включав власний набір полотен, історію змін, назву, шлях збереження, стан модифікації та був інтегрований у загальну модель даних. Підтримувалося збереження та завантаження документа у вигляді структурованого формату JSON, з повною реконструкцією полотен і вмісту. Активний документ міг змінюватися за запитом користувача, а всі зміни автоматично відображалися в інтерфейсі.

Модуль інструментів реалізовував основну взаємодію користувача з полотнами. До нього входили базові дії, такі як малювання, стирання, заливка, вибір кольору, побудова геометричних фігур, переміщення частин зображення, побудова ліній та виділення області. Кожен інструмент мав власну логіку обробки подій введення (натискання, переміщення, відпускання), працював з вибраним полотном і враховував поточні параметри — розмір, колір, прозорість, вибрану область тощо. Перемикання інструментів відбувалося динамічно, а всі параметри зберігалися глобально.

Модуль історії відповідав за збереження станів полотна після кожної дії користувача. Реалізовано механізм збереження, скасування та повтору змін. Історія прив’язувалася до активного полотна та очищалася при перемиканні між документами. Завдяки цьому користувач мав можливість безпечно експериментувати з редагуванням, повертаючись до попереднього стану або повторюючи останню дію.

Модуль графічного інтерфейсу реалізовував користувацьке середовище на основі сучасного UX/UI-дизайну. Інтерфейс включав головне меню, робоче полотно з масштабуванням і прокруткою, бічні панелі з інструментами та кольорами, верхню панель із параметрами інструменту, а також систему вкладок для перемикання між відкритими документами. Уся логіка була побудована на двосторонньому зв’язку з ядром на C++, що дозволяло миттєво реагувати на дії користувача та забезпечувало високий рівень інтерактивності.

Модуль налаштувань інструментів забезпечував централізоване зберігання поточних параметрів активного інструменту: розмір, колір, прозорість, інтенсивність, режим заливки. Завдяки реалізації у вигляді єдиного доступного об'єкта, ці параметри можна було змінювати з інтерфейсу та одночасно використовувати в логіці інструментів, що забезпечувало послідовність налаштувань і зручність у роботі.

Модуль ресурсів відповідав за організацію графічних елементів, таких як іконки, текстури та стилізовані компоненти інтерфейсу. Усі ресурси підключалися централізовано, з підтримкою кількох розмірів для різної щільності екранів. Завдяки цьому інтерфейс залишався чітким і адаптивним на будь-яких пристроях, а структура ресурсів спрощувала оновлення й підтримку дизайну.

Таким чином проєкт представляє собою чітко структуровану систему, де кожен модуль виконує окрему, добре ізольовану функцію. Такий підхід гарантує масштабованість, гнучкість у розширенні функціоналу та легкість супроводу коду, а також забезпечує злагоджену взаємодію між компонентами застосунку.

2.4 Аргументація вибору засобів та середовища

Фреймворк Qt виступатиме основою для побудови графічного інтерфейсу, оскільки він забезпечує кросплатформність, високу продуктивність і широкі можливості інтеграції. Завдяки цьому застосунок зможе стабільно працювати на різних операційних системах. Для розробки інтерфейсу буде використовуватись QML — декларативна мова програмування, яка дозволить створювати адаптивні та інтерактивні UI-компоненти, що сприятиме покращенню користувацького досвіду.

QML — це високорівнева декларативна мова, що буде застосовуватись для створення динамічних і анімованих графічних інтерфейсів. Вона базується на JavaScript і підтримує реактивне програмування, що спростить управління змінами у стані застосунку. QML забезпечить просту інтеграцію векторної графіки, шейдерів, анімацій та взаємодії з користувачем, що зробить її ідеальним вибором для створення сучасного UI. Крім того, завдяки інтеграції з C++ за допомогою механізму контекстних об’єктів (context properties) та моделей даних, вдасться ефективно організувати зв’язок між бізнес-логікою та інтерфейсом.

Для рендерингу та роботи з графікою буде розглянуто декілька підходів, зокрема QPainter, QQuickPaintedItem та QAbstractListModel. QPainter застосовуватиметься для низькорівневої обробки зображень та векторної графіки, що дозволить досягти високої якості візуалізації. QQuickPaintedItem надасть можливість інтегрувати власні методи рендерингу в QML-інтерфейси, що забезпечить гнучке керування графічним вмістом. QAbstractListModel буде використовуватись для організації даних та забезпечення зв’язку між логікою застосунку та інтерфейсом, що спростить роботу з динамічними елементами UI.

CMake виступатиме як кросплатформна система автоматизованого збирання, яка спростить конфігурацію, компіляцію та управління залежностями проєкту. Вона дозволить розробникам описувати структуру застосунку за допомогою файлів CMakeLists.txt, у яких визначатимуться цілі

збірки, бібліотеки та необхідні залежності. CMake підтримуватиме різні генератори (наприклад, Makefiles, Ninja, Visual Studio), що дозволить адаптувати його до різних середовищ розробки. Завдяки своїй гнучкості та інтеграції з Qt, CMake стане ефективним інструментом для створення масштабованих і портативних програмних рішень.

У даному розділі проведено детальне проєктування графічного застосунку з багатошаровою структурою та інструментами для редагування зображень. Обрано компонентний підхід із чітким розподілом відповідальності між класами, що забезпечить модульність і розширюваність системи. Основу архітектури складатимуть класи Application, LayerManager, Layer та Tools, які визначають логіку роботи програми та взаємодію між елементами. Для розробки використано технологічний стек на базі Qt та QML, що дозволить створити кросплатформене рішення з ефективним рендерингом графіки. Реалізація системи управління шарами дасть змогу зручно додавати, редагувати та організовувати вміст, а набір інструментів забезпечить повноцінну взаємодію користувача з кожним шаром. Спроєктовано загальну структуру застосунку, розроблено UML-діаграми класів, послідовності та взаємодії компонентів, що дозволить легко розширювати функціонал у майбутньому.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Засоби реалізації застосунку

Для реалізації кросплатформного графічного застосунку було використано комплекс програмного забезпечення, що охоплював системне, прикладне та інструментальне ПЗ. Системне програмне забезпечення включало операційні системи Windows, Linux та macOS, які забезпечували тестування та стабільну роботу застосунку на основних десктопних платформах.

Основною мовою програмування був обраний C++, що забезпечувало високу продуктивність, низькорівневий контроль над пам’яттю та сумісність із фреймворком Qt, який використовувався для створення графічного інтерфейсу користувача з використанням QML.

Як основне середовище розробки використовувався редактор Neovim, запущений через графічну оболонку Neovide, що надала розширений функціонал (включаючи підтримку шрифтів, анімацій і GUI-інтеграції), з можливістю гнучкого налаштування через конфігураційні файли. Такий підхід забезпечив швидкість, мінімалізм і максимальний контроль над середовищем розробки.

Інструментальне ПЗ включало CMake як систему автоматичного збирання, що дозволила налаштувати незалежне складання окремих модулів (document, layer, tools, history тощо), а також Git у поєднанні з платформою GitHub для зберігання, версіювання та колективної роботи над кодом.

Прикладне програмне забезпечення базувалося на компонентах Qt Quick, Qt Core та Qt GUI, які забезпечили створення адаптивного інтерфейсу, підтримку інтерактивної роботи з полотнами, анімацій та інструментів малювання. Графічні ресурси у форматах SVG та PNG інтегрувалися через механізм ресурсів Qt за допомогою .qrc файлів.

Проєкт мав чітко визначену модульну структуру, що підтримувалася завдяки правильній організації файлів, централізованому конфігураційному

файлу CMakeLists.txt і структурованим підмодулям. Такий підхід забезпечив високу підтримуваність, масштабованість та зручність у подальшій розробці.

3.2 Реалізація програмного застосунку

Розробка графічного застосунку була комплексним процесом, що охоплював проєктування архітектури, реалізацію логіки управління документами та полотнами, створення інструментів малювання, а також побудову інтерфейсу користувача із застосуванням QML. Основу проєкту становить фреймворк Qt, який забезпечує кросплатформність та високу продуктивність. У межах реалізації застосунку були розроблені модулі для роботи з багатошаровими документами, історією змін, інструментами малювання та налаштуванням інтерфейсу.

Основною точкою входу в застосунок є функція main, яка ініціалізує об’єкт QApplication, реєструє типи та синглтони для використання в QML, а також виконує початкове завантаження інтерфейсу. На початку задаються назва організації та назва самого застосунку, після чого встановлюється стиль інтерфейсу Material через QQuickStyle::setStyle. Подальше налаштування включає підключення стилів, реєстрацію перерахування інструментів (ToolType), синглтона з налаштуваннями інструментів (ToolSettings), а також типу Layer, доступного для використання у QML.

Окремим кроком є створення об'єкта DocumentManager, що відповідає за керування відкритими документами та їхній життєвий цикл. Він передається до контексту QML через setContextProperty, що дозволяє використовувати його безпосередньо у QML-коді. Після цього завантажується основний інтерфейс користувача із ресурсу qrc:/Qml/main.qml, а також налаштовується обробка помилок створення об'єктів на випадок, якщо завантаження QML зазнає невдачі.

Таким чином, функція main виконує централізовану ініціалізацію ядра програми та передає керування в інтерфейсну частину. Фрагмент коду, що демонструє запуск застосунку, наведено в лістингу 3.1.

Лістинг 3.1 – Ініціалізація ядра програми

```
        const QUrl url(QStringLiteral("qrc:/Qml/main.qml"));
        QObject::connect(
            &engine, &QQmlApplicationEngine::objectCreated,
&app,
            [url](QObject *obj, const QUrl &objUrl) {
                if (!obj && url == objUrl)
                    QCoreApplication::exit(-1);
            },
            Qt::QueuedConnection);
        engine.load(url);
        return app.exec();
```

Керування документами було реалізовано у класах Document та DocumentManager, що відповідали за зберігання даних про відкриті файли, активний документ, його назву, шлях, полотна та стан змін. Основні функції охоплювали серіалізацію, десеріалізацію, перемикання між документами та збереження їх у файл.

У класі Document зберігалася інформація про назву, шлях, стан змін (modified) та вказівники на менеджери полотен і історії. Всі дані документа серіалізувалися в JSON за допомогою методу toJson, а відновлення з файлу виконувалося методом fromJson.

Функція toJson формувала JSON-об’єкт із даними про всі полотна, включаючи зображення, яке кодувалося у Base64. Це наведено в лістингу 3.2.

Лістинг 3.2 – Серіалізація полотна у форматі JSON

```
    QByteArray byteArray;
    QBuffer buffer(&byteArray);
    buffer.open(QIODevice::WriteOnly);
    layer->layer->image().save(&buffer, "PNG");
    layerObj["image"] =
Qstring::fromLatin1(byteArray.toBase64());
```

Для відновлення документа з файлу використовувався метод fromJson, який створював новий об’єкт Document, зчитував дані з JSON-файлу, додавав відповідні полотна та завантажував їхні зображення. У лістингу 3.3 показано додавання полотна з відновленням зображення.

Лістинг 3.3 – Відновлення зображення полотна з JSON

```
    QString imageBase64 = layerObj["image"].toString();
    QByteArray imageData =
QByteArray::fromBase64(imageBase64.toLatin1());
    QImage image;
    image.loadFromData(imageData, "PNG");
    layerWrapper->layer->setImage(image);
```

Клас DocumentManager виконував роль моделі даних у QML, реалізуючи QAbstractListModel. Через нього здійснювалося додавання, видалення та вибір активного документа. Усі методи були доступні у QML за допомогою макросу Q_INVOKABLE.

Для додавання документа використовувався метод addDocument, який створював новий об'єкт Document, задавав йому назву та шлях, а також додавав до колекції відкритих документів, це представлено в лістингу 3.4.

Лістинг 3.4 – Додавання нового документа

```
Document *doc = new Document();
doc->setName(name);
doc->setPath(path);
m_documents.append(doc);
```

Для збереження документа у файл був реалізований метод saveToFile. Він зчитував шлях, серіалізував документ у формат JSON, зберігав файл на диск і оновлював шлях у документі, це представлено в лістингу 3.5.

Лістинг 3.5 – Збереження документа у файл

```
QFile file(localPath);
file.open(QIODevice::WriteOnly);
QJsonDocument doc(m_currentDocument->toJson());
file.write(doc.toJson(QJsonDocument::Indented));
file.close();
m_currentDocument->setPath(localPath);
```

Клас Layer представляв об’єкт полотна, який відповідав за відображення зображення, обробку подій миші, застосування графічних інструментів і збереження результатів малювання. Полотно містило параметри фону, зображення, а також ознаки видимості, активності та блокування. За відтворення вмісту на екрані відповідав метод paint, що використовував об’єкт QPainter для рендерингу.

Для обробки користувацьких подій використовувалися методи mousePressEvent, mouseMoveEvent і mouseReleaseEvent. Вони викликали відповідні методи активного інструменту, що зберігався в ToolHandler. При натисканні, переміщенні або відпусканні миші передавалася подія, після чого оновлювалося зображення.

У лістингу 3.6 наведено фрагмент, що обробляв подію натискання миші:

Лістинг 3.6 – Обробка події миші у класі Layer

```
if (m_toolHandler) {
    m_toolHandler->press(event);
    update();
}
```

Інструменти малювання ініціалізувалися у методі setCurrentTool. Залежно від типу інструменту створювався відповідний об’єкт (BrushTool, EraseTool, MoveTool тощо) та прив’язувався до полотна. Крім того, одразу встановлювалися параметри інструменту: розмір, колір, прозорість, заливка та виділення. У лістингу 3.7 показано приклад логіки вибору інструменту:

Лістинг 3.7 – Вибір інструменту у полотні

```
switch (m_currentTool) {
    case ToolType::Brush:
        m_toolHandler = std::make_unique<BrushTool>();
        break;
    case ToolType::Eraser:
        m_toolHandler = std::make_unique<EraseTool>();
        break;
    // ...
}
m_toolHandler->setImage(&m_canvas);
```

Клас LayerManager відповідав за зберігання та керування колекцією полотен. Він реалізовував QAbstractListModel, завдяки чому всі полотна відображалися в QML. Кожен елемент зберігав назву, стан блокування, видимість, розміри та об’єкт Layer.

Додавання нового полотна виконувалося методом addLayer, який створював екземпляр Layer, встановлював розміри, фон і активний інструмент. Це представлено у лістингу 3.8.

Лістинг 3.8 – Додавання нового полотна

```
newLayer.layer = new Layer();
newLayer.layer->setWidth(widthToUse);
newLayer.layer->setHeight(heightToUse);
newLayer.layer->setBackgroundColor(background);
newLayer.layer->updateCanvasSize();
```

Керування активністю здійснювалося методом setCurrentIndex. Поточне активне полотно отримувало інструмент і приймало події миші. Всі інші полотна переходили у неактивний стан. Це дозволяло працювати лише з одним полотном одночасно.

Отримання фінального зображення з усіх видимих полотен реалізовувалося в методі getMergedImage. Всі зображення об’єднувалися поверх одне одного в об'єкті QImage через Qpainter, це представлено у лістингу 3.9.

Лістинг 3.9 – Об’єднання всіх полотен

```
QPainter painter(&finalImage);
for (const auto &layerData : m_layers) {
    if (layerData.visible) {
        painter.drawImage(0, 0, layerData.layer->image());
    }
}
```

Усі інструменти в проєкті наслідували абстрактний клас ToolHandler, який визначав базовий інтерфейс для обробки дій миші (press, move, release), встановлення зображення та властивостей інструмента через структуру ToolInfo. Основний перелік інструментів був представлений у переліку ToolType, оголошеному в класі ToolWrapper.

Клас BrushTool відповідав за малювання ліній пензлем. Після натискання кнопки миші координата зберігалася, а в методі move створювалася лінія між поточним положенням та попередньою точкою. Малювання здійснювалося за допомогою QPainter. У лістингу 3.10 показано фрагмент, який відповідав за малювання лінії.

Лістинг 3.10 – Малювання лінії пензлем

```
painter.setPen(QPen(m_tool.color, m_tool.size,
Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));
painter.drawLine(QPointF(m_tool.position.x(),
m_tool.position.y()), event->position());
```

Для вибору кольору з полотна використовувався клас ColorPickerTool. У момент натискання миші визначалась точка на зображенні, і через метод pixelColor зчитувався відповідний колір. Цей колір зберігався в об’єкті ToolSettings, як показано у лістингу 3.11.

Лістинг 3.11 – Вибір кольору з полотна

```
QColor pickedColor = canvas->pixelColor(pos);
ToolSettings::instance()->setColor(pickedColor);
```

Інструмент EraseTool працював за аналогією до пензля, але замість кольору використовував прозорість. Завдяки композиційному режиму QPainter::CompositionMode_Clear, фрагменти зображення видалялися. Основна логіка представлена в лістингу 3.12.

Лістинг 3.12 – Стирання фрагменту зображення

```
painter.setPen(QPen(Qt::transparent, m_tool.size,
Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));
painter.setCompositionMode(QPainter::CompositionMode_Clear);
painter.drawLine(QPointF(m_tool.position.x(),
m_tool.position.y()), event->position());
```

Клас FillTool реалізовував алгоритм заливки з використанням пошуку в глибину (stack). При натисканні миші інструмент запускав рекурсивне заповнення суміжних пікселів, які мали схожий колір. Обчислення подібності кольорів здійснювалося з використанням порогової функції. У лістингу 3.13 наведено перевірку схожості кольорів.

Лістинг 3.13 – Перевірка подібності кольорів

```
return std::abs(qRed(c1) - qRed(c2)) <= tolerance &&
       std::abs(qGreen(c1) - qGreen(c2)) <= tolerance &&
       std::abs(qBlue(c1) - qBlue(c2)) <= tolerance;
```

Інструмент LineTool дозволяв створювати прямі лінії між двома точками. Користувач визначав початкову і кінцеву координати, після чого малювалася

лінія із заданими параметрами. Рендеринг здійснювався через QPainter з використанням згладжування (antialiasing), як показано в лістингу 3.14.

Лістинг 3.14 – Побудова прямої лінії

```
QPen pen(m_currentInfo.color);
pen.setWidthF(m_currentInfo.size);
painter.setPen(pen);
painter.drawLine(m_startPoint, m_endPoint);
```

Для переміщення області зображення застосовувався клас MoveTool. При натисканні миші з області виділення копіювався фрагмент зображення, після чого полотно очищалося. У момент відпускання миші цей фрагмент переміщувався в нову позицію. Логіка очищення і вставки представлена в лістингу 3.15.

Лістинг 3.15 – Переміщення виділеного фрагмента

```
painter.setCompositionMode(QPainter::CompositionMode_Clear);
painter.fillRect(m_moveRect, Qt::transparent);
painter.setCompositionMode(QPainter::CompositionMode_SourceO
ver);
painter.drawImage(targetRect.topLeft(), m_movedContent);
```

Інструмент SelectionTool дозволяв користувачу створювати прямокутну область виділення. Під час переміщення миші обчислювалася нова прямокутна зона, яка застосовувалася до полотна. При відпусканні кнопки результат оновлювався. Лістинг 3.16 демонструє обчислення прямокутника виділення.

Лістинг 3.16 – Формування прямокутника виділення

```
QRect selectionRect = QRect(m_startPoint,
m_endPoint).normalized();
m_layer->setSelection(selectionRect);
```

Всі інструменти використовували структуру ToolInfo, яка містила параметри — колір, розмір, прозорість, інтенсивність, режим заливки та координати. Зміна активного інструмента і його параметрів здійснювалася через клас ToolSettings, який був реалізований як синглтон і доступний у QML через qmlRegisterSingletonType.

Для збереження та відновлення змін на полотнах було реалізовано окремий модуль історії — CanvasHistoryManager. Цей клас відповідав за управління історією дій користувача шляхом збереження станів зображення та надання функціоналу для скасування (undo) або повтору (redo) останніх операцій.

В основі зберігання змін використовувалися два стеки: undoStack і redoStack. Кожен стан представлявся структурою HistoryState, яка містила копію зображення (QImage) та вказівник на відповідне полотно (Layer). Новий стан зберігався через слот saveState, який викликався сигналом stateChanged, прив’язаним до кожного нового полотна. Це було реалізовано в класі Document через метод connectLayerSignals, як показано у лістингу 3.17.

Лістинг 3.17 – Підключення історії до нового полотна

```
connect(layer, &layer::Layer::stateChanged,
m_historyManager.get(), &CanvasHistoryManager::saveState);
```

При виклику saveState відбувалося збереження копії поточного зображення полотна, якщо воно було активним. Стан додавався у стек undoStack, а стек redoStack залишався недоторканим, що відповідало класичній моделі редакторів. У лістингу 3.18 наведено збереження нового стану.

Лістинг 3.18 – Збереження стану полотна

```
HistoryState state;
state.image = image.copy();
state.layer = layer;
undoStack.push(state);
```

Функція undo витягувала останній стан з undoStack і застосовувала збережене зображення до відповідного полотна. Поточний стан при цьому переміщувався до redoStack, щоб мати змогу виконати повтор. Оновлення полотна здійснювалося методом setImage, після чого зображення перерендерювалося. Основна логіка undo представлена в лістингу 3.19.

Лістинг 3.19 – Скасування останньої дії

```
HistoryState state = undoStack.pop();
redoStack.push(state);
state.layer->setImage(state.image);
state.layer->update();
```

Метод redo діяв аналогічно, але в протилежному напрямку — він витягував стан з redoStack, застосовував його до полотна та знову зберігав у undoStack, як видно з лістингу 3.20.

Лістинг 3.20 – Повтор останньої скасованої дії

```
HistoryState state = redoStack.pop();
undoStack.push(state);
state.layer->setImage(state.image);
state.layer->update();
```

Для побудови інтерфейсу користувача було реалізовано окремий модуль qml, який відповідав за візуальне оформлення та взаємодію з користувачем. Модуль містив компоненти для створення початкового екрана, керування документами, вибору кольору, управління полотнами та інструментами, а також верхнє меню з основними функціями. Всі елементи були розміщені у відповідних QML-файлах, які взаємодіяли з бекендом через контекстні властивості та сигнали. Детальна структура і логіка кожного елементу наведена нижче.

У файлі StartScreen.qml було реалізовано початковий екран програми, який автоматично відображався при запуску. Він включав центральну колонку contentColumn з заголовком "Harmoniq", кнопкою для створення нового проєкту з параметрами (розмір, колір фону) та кнопкою для відкриття наявного документа через файловий діалог. Обробка введених даних та виклик методів бекенду здійснювалися через сигнали та слоти. Детальніша реалізація подій подана в додатку А.

Компонент Layer.qml відповідав за відображення полотен у робочій області редактора. За прокрутку відповідав Flickable, а Repeater генерував елементи на основі моделі documentManager.currentLayerManager.

Масштабування полотна здійснювалося за допомогою WheelHandler і Scale, де зміна масштабу відбувалася відносно центру полотна. Деталі реалізації наведено в додатку Б.

У файлі DocumentTab.qml реалізовувалась система вкладок для перемикання між відкритими документами. TabBar разом з Repeater створював динамічні кнопки для кожного відкритого документа. Вкладки показували назву документа з індикатором змін ("*"), а також містили кнопку для закриття. Логіка оновлення вкладок та зв’язок з моделлю були описані в додатку В.

Файл ColorPicker.qml реалізовував інструмент вибору кольору для малювання. Він складався з трьох основних частин: смуги відтінків (HUE), поля насиченості/яскравості (SV) та регулятора прозорості (ALPHA). Зміни здійснювались через MouseArea, а вибраний колір передавався до ToolSettings. Історія обраних кольорів зберігалася у моделі colorHistory, що обмежувалася 10 унікальними записами. Деталі взаємодії описані у додатку Г.

У LeftBar.qml було реалізовано ліву панель з кнопками для вибору інструментів. Компонент ToolButton відповідав за вигляд кожної кнопки, яка мала відповідну іконку та сигнал на активацію інструменту. Кнопки групувалися в сітки, відповідно до функціонального призначення (малювання, заливка, переміщення тощо). Структура кнопок наведена в Додатку Ґ.

Файл RightBar.qml забезпечував праву панель з вибором кольору та списком полотен. Для вибору кольору використовувався Loader, який завантажував ColorPicker.qml динамічно. Список полотен реалізовувався через ListView, де кожен елемент дозволяв змінити назву, видимість, блокування, або перемістити/видалити полотно. Візуальна анімація панелі реалізовувалась за допомогою Behavior. Опис структури наведено в додатку Д.

У TopBar.qml було розміщено елементи керування властивостями активного інструменту. Слайдер у блоці "Size" дозволяв змінювати розмір інструменту, а чекбокс у блоці "Fill" — увімкнути або вимкнути заливку. Всі елементи були об’єднані в RowLayout, який забезпечував компактне горизонтальне розміщення. Повна логіка описана в додатку Е.

В файлі TopMenu.qml було реалізовано верхнє меню із пунктами File, Edit, View та Help. Вони запускали основні дії — створення, відкриття, збереження, експорт документа, скасування (undo) і повтор (redo) через historyManager, а також викликали діалоги файлів. Деталі реалізації меню наведено в додатку Є.

3.3 Тестування застосунку

Тестування є завершальним і надзвичайно важливим етапом у процесі розробки програмного забезпечення. Саме на цьому етапі виявлялися можливі помилки, збої в логіці роботи застосунку та перевірялася відповідність функціоналу очікуваним результатам.

У першу чергу було проведено перевірку кросплатформеності та базової функціональності графічного застосунку. Програму запускали та тестували в середовищах операційних систем Windows і Linux, що дало змогу впевнитися у стабільній роботі на різних платформах (рис. 3.1, рис. 3.2).

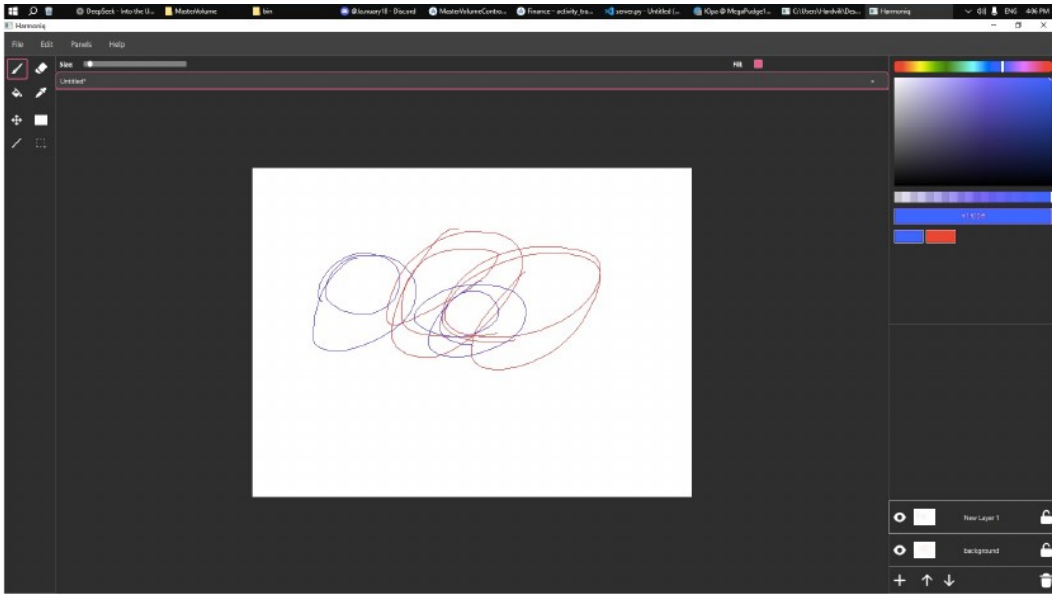

Рисунок 3.1 – Тестування запуску на програми на ОС Windows

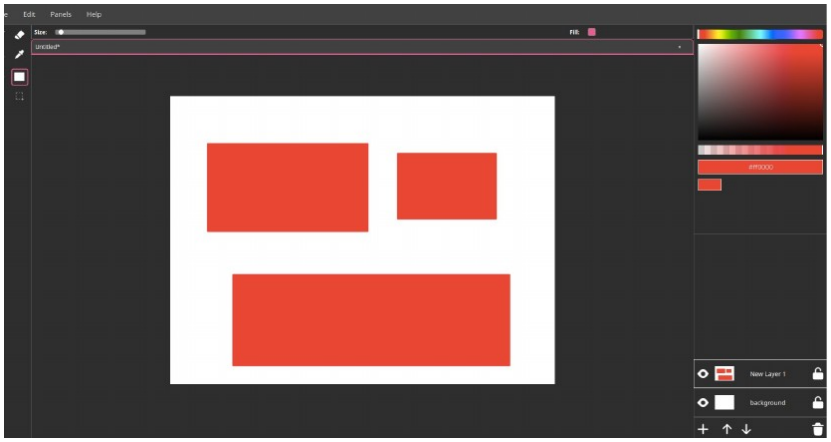

Рисунок 3.2 – Тестування запуску на програми на ОС Linux

Далі було перевірено всі інструменти графічного застосунку, включаючи BrushTool FillTool, LineTool, MoveTool, SelectionTool та ShapeTool з метою оцінки їхньої коректності, стабільності та відповідності специфікаціям. Спочатку було перевірено інструмент «Пензель» (BrushTool), який коректно малює лінії з урахуванням заданих параметрів розміру, кольору та прозорості (рис. 3.3).

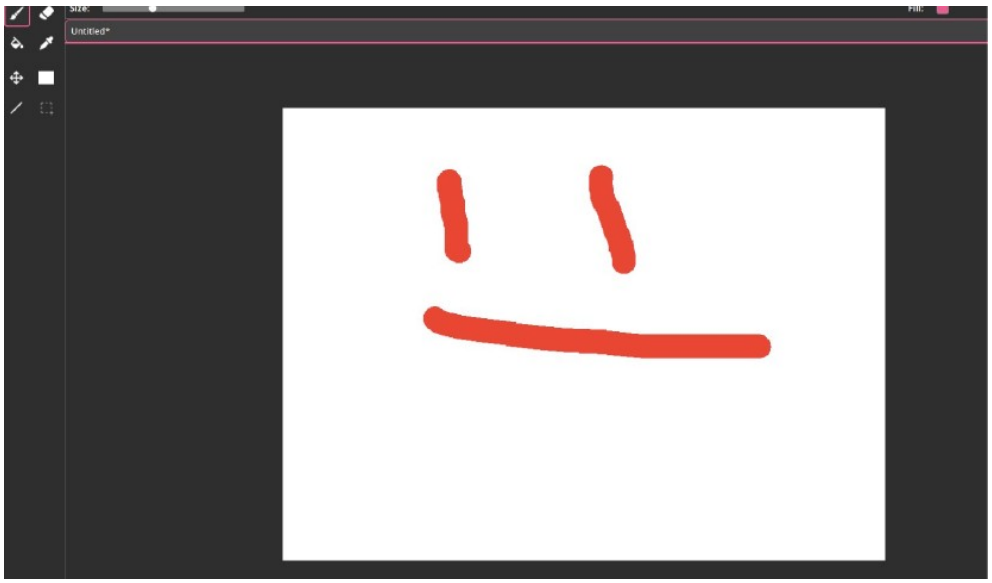

Рисунок 3.3 – Тестування інструменту пензлик

Інструмент «Заливка» (FillTool) забезпечує заповнення замкнених областей обраним кольором, демонструючи стабільну роботу на різних розмірах полотна (рис. 3.4).

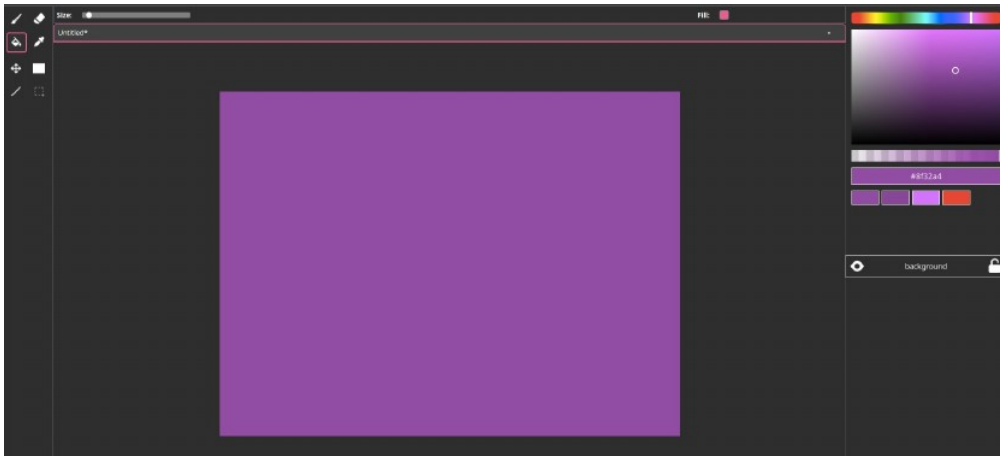

Рисунок 3.4 – Тестування інструменту заливка

Інструмент «Фігури» (ShapeTool) успішно створює геометричні фігури, такі як прямокутники та кола, з можливістю налаштування контуру та заповнення (рис. 3.5).

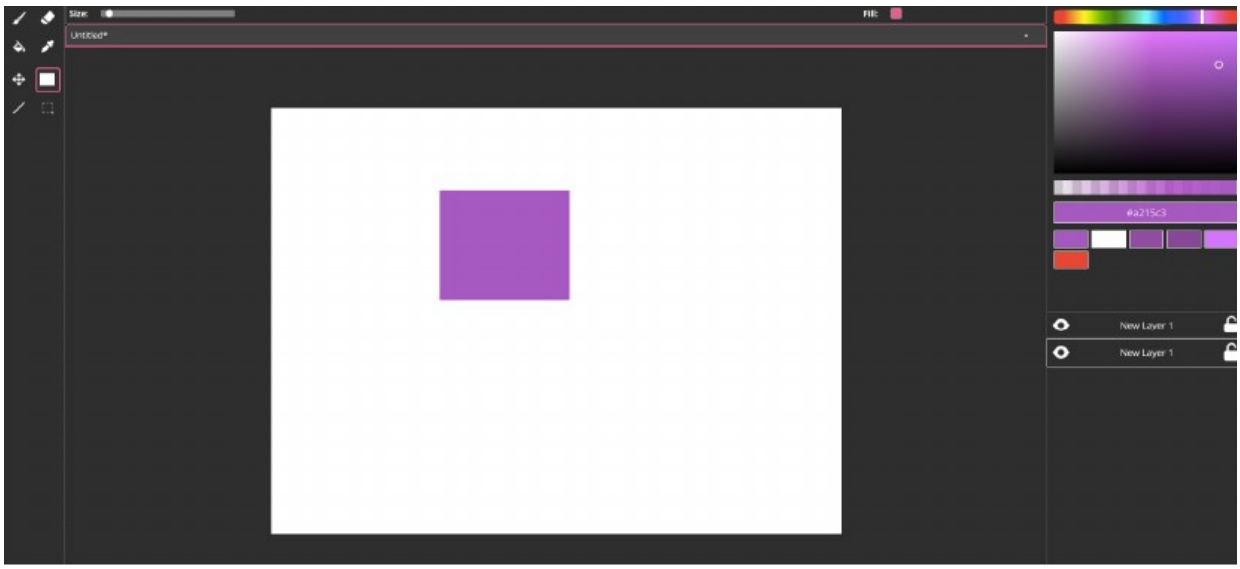

Рисунок 3.5 – Тестування інструменту фігура

Інструмент «Переміщення» (MoveTool) коректно переміщує виділені об’єкти або шари по полотну (рис. 3.6).

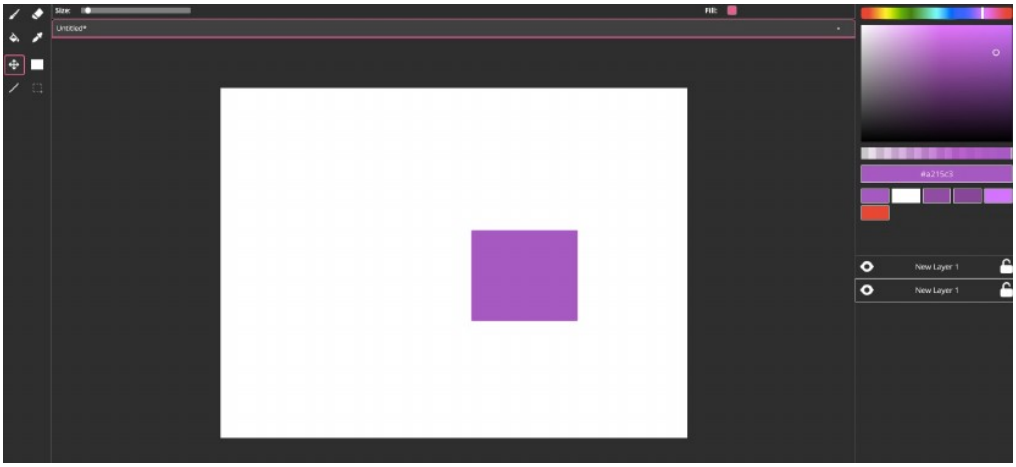

Рисунок 3.6 – Тестування інструменту переміщення

Інструмент «Лінія» (LineTool) дозволяє створювати прямі лінії з налаштуванням товщини та стилю, функціонуючи без помилок (рис. 3.7).

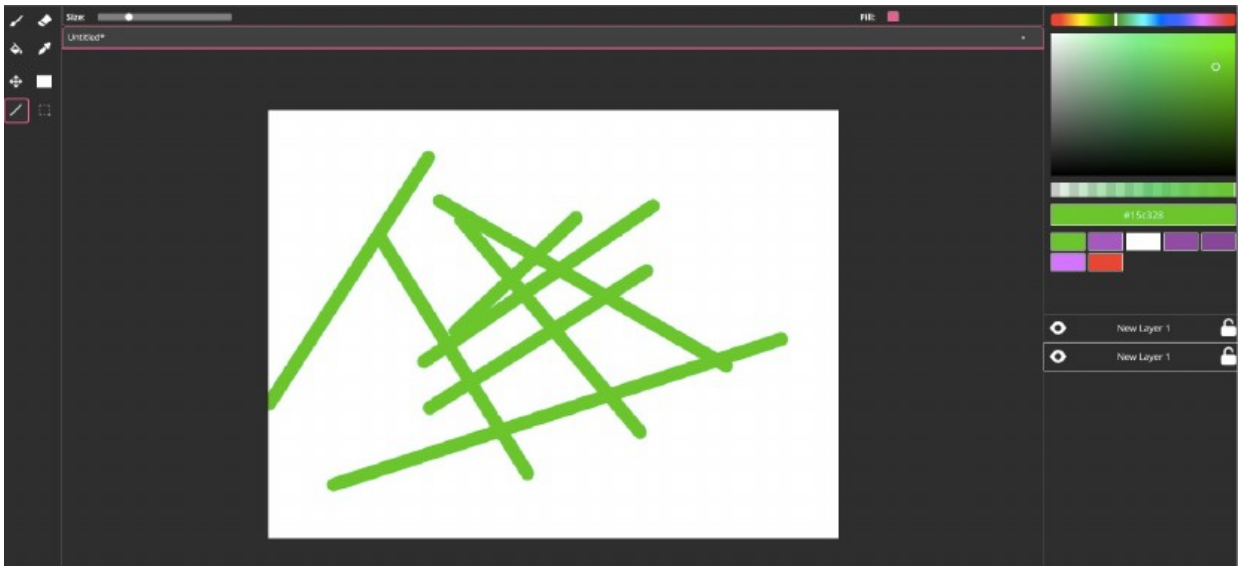

Рисунок 3.7 – Тестування інструменту лінія

Інструмент «Виділення» (SelectionTool) забезпечує точне виділення областей для подальшого редагування, для прикладу було виділено прямокутну область, і після цього був використаний інструмент заливка в виділеній області, це представлено на рисунку 3.8.

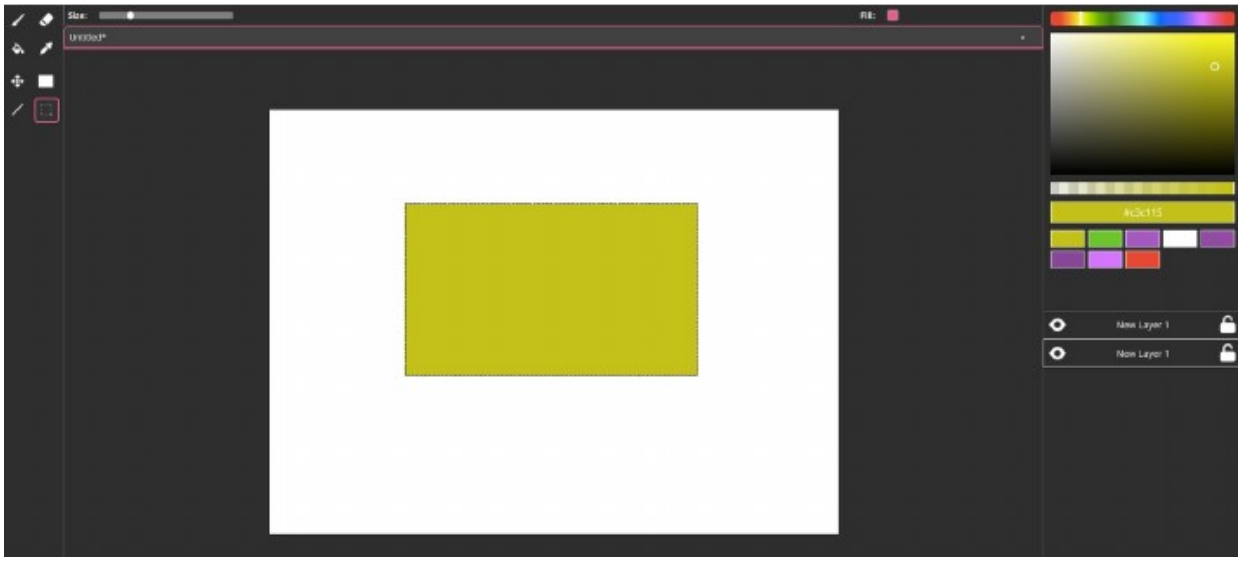

Рисунок 3.8 – Тестування інструменту виділення

Далі було перевірено функціональність роботи з полотнами в графічному застосунку, включаючи створення, видалення та переміщення полотен, з метою оцінки їхньої коректності, стабільності та відповідності специфікаціям. Тестування охоплювало базову функціональність операцій із полотнами та поведінку в різних сценаріях. Спочатку було перевірено створення полотна, яке дозволяє користувачу задавати розміри (від 100x100 до 10000x10000 пікселів) і створювати новий документ без помилок (рис. 3.9).

Рисунок 3.9 – Тестування створення полотна

Функція видалення полотна коректно видаляє активне полотно з інтерфейсу користувача, залишаючи усі інші, для прикладу було видалено «New Layer 1», це представлено на рисунку 3.10.

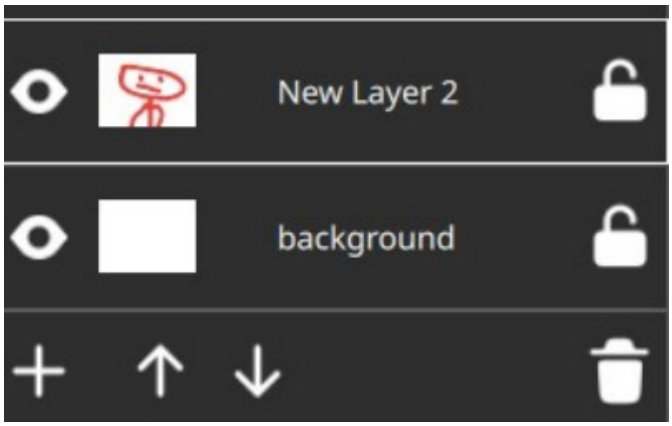

Рисунок 3.10 – Тестування видалення полотна

Переміщення полотна забезпечує плавне зміщення вмісту без артефактів чи втрати даних, для прикладу було переміщено 3 полотно, це преставлено на рисунку 3.11.

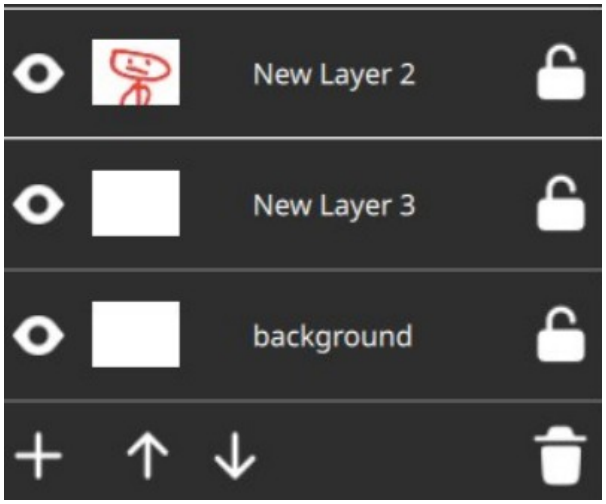

Рисунок 3.11 – Тестування переміщення полотна

Таким чином було розроблено повноцінний графічний застосунок для створення зображень, який надає користувачам зручний і функціональний інструмент для створення, редагування та керування графічним контентом.

4 ТЕХНІЧНО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Розроблений кросплатформний графічний застосунок є сучасним інструментом для створення та редагування зображень, орієнтованим на широку аудиторію користувачів — студентів, фрилансерів, освітян, дизайнерів-початківців та представників креативних індустрій. Застосунок поєднував у собі достатній набір функціональних можливостей, інтуїтивний інтерфейс та високу продуктивність, що забезпечувало його ефективне використання як у побутових, так і в напівпрофесійних сценаріях.

З технічного боку продукт вирізнявся такими характеристиками:

– система збереження з можливістю відновлення при аварійному завершенні роботи.

– тісна інтеграція між C++ і QML, що забезпечувала високу швидкість обробки подій та оновлення інтерфейсу.

– підтримка англійської мови з можливістю подальшої локалізації.

– відкритий вихідний код і безкоштовна ліцензія на використання.

З економічного погляду, застосунок мав низький поріг входу як для кінцевого користувача, так і для розробників. Завдяки використанню вільного програмного забезпечення (Qt, CMake, Git, Neovide) не потребувалося жодних витрат на ліцензування чи обслуговування сторонніх компонентів, що дозволяло суттєво скоротити витрати на розробку та підтримку.

Цільовою аудиторією продукту були користувачі віком від 16 до 50 років, яким потрібен зручний і доступний інструмент для щоденних задач — створення ілюстрацій, макетів, редагування зображень або підготовки графіки для освітніх та соціальних проєктів. Зокрема, програму орієнтовано на студентів, фрилансерів, освітні заклади, SMM-спеціалістів та користувачів з невисокими технічними навичками.

Основним ринком збуту визначено Україну, однак завдяки багатоплатформності (Windows, Linux, macOS), відкритій архітектурі та

гнучкій системі локалізації, застосунок мав потенціал для міжнародного розповсюдження. Прогнозована кількість користувачів у перший рік становила 500–2000 осіб. Реалізація планувалась через відкриті платформи (Flathub, Microsoft Store, GitHub, офіційний вебсайт), що зменшувало маркетингові витрати.

Серед основних конкурентів:

– Krita — професійний редактор, орієнтований переважно на художників, менш зручний для початківців.

– GIMP — складний у користуванні через перевантажений інтерфейс та застарілий підхід до роботи з шарами.

– Paint.NET — обмежений функціонально та доступний лише для Windows.

– Canva — працює лише онлайн і потребує платної підписки для розширених функцій.

Цінова політика конкурентів здебільшого базується на підписній моделі, де повний функціонал надається лише після оплати (від 50 до 200 грн/міс.). Запропонований застосунок є повністю безкоштовним, що створює суттєву перевагу для користувачів із обмеженим бюджетом.

Проведений аналіз підтверджує доцільність створення графічного застосунку як конкурентоспроможного та суспільно корисного продукту з потенціалом до масштабування та подальшого розвитку.

4.2 Розрахунок витрат на проєктування

Матеріальна частина охоплює витрати, пов’язані з розробкою програмного забезпечення, зокрема — грошову винагороду членам команди, які брали участь у створенні кросплатформного графічного застосунку з використанням фреймворку Qt. Розмір оплати праці залежав від складності реалізованих модулів, ефективності виконання поставлених завдань та внутрішніх можливостей проєкту.

Заробітна плата працівників умовно поділяється на основну та додаткову (неосновну) частину. Основна частина включає винагороду за реалізацію ключових функцій застосунку, тоді як неосновна може включати оплату за позапланові задачі, які не передбачалися початковим технічним завданням.

Згідно з чинним законодавством, мінімальна заробітна плата з 01.04.2024 року становить 8000 грн/місяць, і заробітна плата спеціалістів не може бути нижчою за цю суму.

У розробці застосунку брали участь такі фахівці: C++/Qt-developer, Проєктувальник, Тестувальник (QA engineer), QML-developer і UI/UX designer.

C++/Qt-developer за повний відпрацьований місяць отримав 16000 гривень.

Рахуємо податок на доходи фізичних осіб: 16000 * 18% = 2880 грн.

Рахуємо військовий збір: 16000 * 1.5% = 240 грн.

Рахуємо єдиний соціальний внесок: 16000 * 22% = 3520 грн.

Рахуємо загальну суму утримань: 2880 + 240 = 3120 грн.

До виплати працівникові – 16000 - 3120 = 12880 грн.

QML-developer за повний відпрацьований місяць отримав 14000 гривень.

Рахуємо податок на доходи фізичних осіб: 14000 * 18% = 2520 грн.

Рахуємо військовий збір: 14000 * 1.5% = 210 грн.

Рахуємо єдиний соціальний внесок: 14000 * 22% = 3080 грн.

Рахуємо загальну суму утримань: 2520 + 210 = 2730 грн.

До виплати працівникові – 14000 - 2730 = 11270 грн.

QA engineer за повний відпрацьований місяць отримав 12000 гривень.

Рахуємо податок на доходи фізичних осіб: 12000 * 18% = 2160 грн.

Рахуємо військовий збір: 12000 * 1.5% = 180 грн.

Рахуємо єдиний соціальний внесок: 12000 * 22% = 2640 грн.

Рахуємо загальну суму утримань: 2160 + 180 = 2340 грн.

До виплати працівникові – 12000 - 2340 = 9660 грн.

UI/UX designer за повний відпрацьований місяць отримав 10000 гривень.

Рахуємо податок на доходи фізичних осіб: 10000 * 18% = 1800 грн.

Рахуємо військовий збір: 10000 * 1.5% = 150 грн.

Рахуємо єдиний соціальний внесок: 10000 * 22% = 2200 грн.

Рахуємо загальну суму утримань: 1800 + 150 = 1950 грн.

До виплати працівникові – 10000 - 1950 = 8050 грн.

Проєктувальник за повний відпрацьований місяць отримав 15000 гривень.

Рахуємо податок на доходи фізичних осіб: 15000 * 18% = 2700 грн.

Рахуємо військовий збір: 15000 * 1.5% = 225 грн.

Рахуємо єдиний соціальний внесок: 15000 * 22% = 3300 грн.

Рахуємо загальну суму утримань: 2700 + 225 = 2925 грн.

До виплати працівникові – 15000 - 2925 = 12075 грн.

Результати обрахунків окладів представлені в таблиці 4.1.

Таблиця 4.1 – Відрахування заробітніх плат

№ П/П	Посада	Оклад, грн./міс	Відрахування, грн./міс	Кількість		Сума, грн
1	C++/Qt-developer	16000	3120	1	6 міс.	96000
2	QML-developer	14000	2730	1	6 міс.	84000
3	QA engineer	12000	2340	1	6 міс.	72000
4	UI/UX designer	10000	1950	1	2 міс.	20000
5	Проєктувальник	15000	2925	1	2 міс.	30000
6		Усього:				302000

Сума відрахувань від зарплати на соціальні потреби складає: 3120+2730+2340+1950+2925 = 13065 грн.

Під час розробки застосунку використовувалося стандартне обладнання, яке представлено в таблиці 4.2.

Таблиця 4.2 – Використане обладнання

Пристрій	Потужність (Вт)	Час роботи (год)	Вартість (грн)
Комп'ютер	150	320	207,36
Монітор	50	320	69,12
Освітлення	40	320	55,3
Смартфон	15	160	10,4

Розрахунок вартості електроенергії за тарифом 4,32 грн/кВт·год

- Комп'ютер 0,25·320·4,32=345,6 грн0,25·320·4,32=207,36 грн
- Монітор 0,05·320·4,32=82,9 грн0,06·320·4,32=69,12 грн
- Освітлення 0,04·320·4,32=55,3 грн0,04·320·4,32=55,3 грн
- Смартфон 0,015·160·4,32=10,4 грн0,015·160·4,32=10,4 грн

Загальні витрати на електроенергію 207,36+69,12+55,3+10,4=342,18 грн.

Для розробки програмного продукту було використано виключно вільне та безкоштовне програмне забезпечення з відкритими або умовно вільними ліцензіями. Це дозволило мінімізувати витрати на розробку, не порушуючи умов ліцензування. Перелік основних інструментів:

- Qt Framework — фреймворк для розробки графічних інтерфейсів, використано у версії з відкритою ліцензією (LGPL).
- CMake — кросплатформна система складання, що використовується для побудови проєкту.
- Git / GitHub — система керування версіями та віддалене сховище для командної роботи над кодом.
- Neovide — GUI-версія редактора Neovim, використана як основне середовище розробки.

Витрати на зв'язок представлено на таблиці 4.3.

Таблиця 4.3 – Розрахунок витрат на зв'язок

Зв'язок	Кількість місяців	Вартість за місяць (грн)	Сума (грн)
Мобільний зв'язок	3	200	600
Інтернет	3	200	600

Загальна сума прямих витрат з урахуванням соціальних відрахувань, витрат на електроенергію та зв’язок становить 14607,18 грн.

Накладні витрати за один місяць складають 40% від суми прямих витрат – 14607,18 * 40% = 5842,87 гривені.

Планові накопичення складають 25% від суми прямих та накладних витрат – (14607,18 + 5842,87)*25% = 5127,51 гривню.

Кошторисна ціна додатка: 14607,18 + 5842,87 + 5127,51 = 25577,56 гривень.

Податок на додаткову ціну: 25577,56*20% = 5115,51 гривеня.

Ціна за договором: 25577,56 + 5115,51 = 30693,07 гривень.

Кошторис розходів у проєктуванні зведено у таблиці 4.4

Таблиця 4.4 – Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн	Обґрунтування
1. Зарплата розробників	302000	
2. Відрахування на соціальні потреби	13065	
3. Контрагентські роботи і послуги	–	Не відбулися
4. Відрядження	–	Не відбулися
5. Інші прямі витрати	1542,18	Електроенергія — 342,18 грн; зв’язок — 1200 грн
6. Усього прямих витрат	14607,18	
7. Накладні витрати	5842,87	40% від прямих витрат
8. Передбачувані зібрання	5127,51	25% від суми (прямі + накладні витрати)
9. Загальна вартість проєкту	25577,56	
10. Податок на додаткову вартість	5115,51	20% від вартості проєкту
11. Загалом, ціна за договором	30693,07	

Впровадження кросплатформного графічного застосунку дає змогу зекономити ресурси та підвищити продуктивність роботи користувачів. Зокрема, можна оцінити ефект за рахунок зменшення часу на виконання графічних задач, особливо для освітніх закладів та фрилансерів.

Припустимо, середній користувач витрачає 1 годину на створення базового зображення в іншому редакторі, а у новому застосунку — 0.5 години, за рахунок швидшого інтерфейсу. Вартість години роботи становить 120 грн, кількість таких задач на рік — 335:

(1 - 0.5) × 120 × 335 = 0.5 × 120 × 335 = 20100 грн/рік.

При традиційному редагуванні користувач може втратити прогрес, стерти шари чи помилково перезаписати зміни. Застосунок має інтуїтивну систему збереження та керування шарами, що знижує кількість критичних помилок.

Припустимо, у середньому користувач допускає 60 таких помилок на рік, а з новим інтерфейсом — 30. Вартість помилки оцінюється в 120 грн:

(60 - 30) × 120 = 3600 грн/рік.

Багато професійних редакторів потребують платної підписки або ліцензії (наприклад, Affinity Photo, CorelDraw тощо), вартість яких у середньому складає 1500 грн на рік. Створений застосунок має відкриту ліцензію і не потребує додаткових витрат: 2500 грн/рік.

Завдяки простому інтерфейсу нові користувачі опановують інструменти швидше. Якщо навчання іншого редактора займає 10 годин, а мого — лише 3, економія становить 7 годин на одного користувача. Для 35 користувачів при вартості години 120 грн:

(10 - 3) × 120 × 20 = 16800 грн

Загальний річний економічний ефект:

20100 + 3600 + 2500 + 16800 = 43000 грн/рік

Термін окупності проєкту

Срозр = 332693,07 грн (загальні витрати на розробку).

Е = 55000 грн/рік

Ток = 332693,07 / 55000 ≈ 72 місяці

Коефіцієнт економічної ефективності

Кеф = 43000 / 332693,07 ≈ 0.12.

Оскільки Кеф < 1, проект вважається економічно не доцільним та не ефективним, з поганими перспективами для подальшого впровадження та але з хорошими можливостями для масштабування.

4.3 Обґрунтування необхідності

Розробка кросплатформного графічного застосунку для створення та редагування зображень відповідає актуальним потребам сучасних користувачів. Серед основних потреб, які покриває продукт, слід виділити: доступність на всіх основних операційних системах (Windows, Linux, macOS), простоту інтерфейсу для новачків, підтримку багатошарового редагування та можливість розширення функціоналу.

Цей продукт особливо актуальний для освітніх закладів, фрилансерів та дизайнерських студій, які потребують ефективного інструменту без великих фінансових витрат. Він дозволяє зменшити витрати на придбання платних аналогів та економить час на створення графічних матеріалів.

Основні напрямки отримання ефекту від впровадження:

- економія часу на виконання графічних задач (вдвічі швидше);
- економія коштів на платне ПЗ;
- підвищення продуктивності працівників;
- поліпшення доступності інструментів для навчання.

З економічної точки зору, продукт має дуже низький коефіцієнт ефективності (0.12) та термін окупності (~72 місяці), що підтверджує не доцільність інвестицій у його розробку. Витрати на створення (капітальні інвестиції) не компенсуються отриманим ефектом протягом шести років використання.

Таким чином, розробка даного продукту не лише вирішує практичні завдання, але й забезпечує економічний ефект, сприяє цифровій трансформації навчальних та творчих процесів, та закладає основу для подальшого розвитку програмного забезпечення в Україні.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи на тему «Розробка кросплатформного графічного застосунку» було досягнуто поставленої мети – створено зручний і ефективний інструмент для створення, редагування та керування графічним контентом, що підтримує багатошарове редагування, інтуїтивний інтерфейс і кросплатформну сумісність. Робота виконана відповідно до визначених завдань, які включали аналіз предметної області, розробку архітектури застосунку, реалізацію функціональних модулів, тестування системи та техніко-економічне обґрунтування.

На етапі аналізу предметної області досліджено потреби користувачів у сфері графічного редагування, що підтвердило актуальність створення кросплатформного застосунку з підтримкою української мови та адаптацією до потреб локального ринку. Виявлено недоліки існуючих рішень, зокрема високу вартість ліцензій і складність інтерфейсів для новачків, що дозволило сформулювати вимоги до продукту: підтримка Windows, Linux і macOS, інтуїтивний інтерфейс, багатошарове редагування та захист даних.

Під час проєктування застосунку розроблено архітектуру з використанням Qt/QML для інтерфейсу та C++ для логіки, що забезпечило кросплатформність і продуктивність. Спроєктовано модулі для роботи з полотнами, шарами, інструментами редагування та історією дій, а також інтерфейс, оптимізований для зручності використання. Вибір інструментів розробки (Qt Creator, Neovim, Git) дозволив мінімізувати витрати завдяки використанню відкритих технологій.

На етапі реалізації та тестування створено основні функції застосунку: створення та видалення полотен, багатошарове редагування, інструменти для малювання та підтримка історії дій. Тестування на Windows і Linux підтвердило стабільність роботи, коректність функціоналу та відповідність вимогам, хоча виявлено незначні затримки на великих полотнах (>4000x4000 пікселів).

У рамках техніко-економічного обґрунтування розраховано витрати на розробку, які склали 30693,07 грн, включаючи зарплату розробників (30200 грн), відрахування на соціальні потреби (13065 грн), електроенергію, зв’язок (1542,18 грн), накладні витрати (5842,87 грн) і податок на додану вартість (5115,51 грн). Економічний ефект від впровадження оцінено в 43000 грн на рік для одного користувача за рахунок економії часу на виконання графічних задач (20100 грн), зменшення помилок (3600 грн), відсутності витрат на платні аналоги (2500 грн) і скорочення часу на навчання (16800 грн). Однак термін окупності проєкту становить 72 місяці, а коефіцієнт економічної ефективності (0,12) вказує на низьку економічну доцільність, що обмежує перспективи масштабування без додаткових інвестицій.

Практична цінність роботи полягає у створенні готового до використання кросплатформного графічного застосунку, який підходить для освітніх закладів, фрилансерів і дизайнерів. Застосунок вирізняється підтримкою української мови, інтуїтивним інтерфейсом і відкритою ліцензією, що дозволяє економити до 43000 грн на рік за рахунок автоматизації редагування та зменшення витрат на платне ПЗ. Проведено пілотне тестування серед представників Галицького фахового коледжу імені В’ячеслава Чорновола, що підтвердило зручність і працездатність продукту.

Застосунок підготовлено до розміщення в відкритих репозиторіях із базовою безкоштовною версією та потенціалом для додавання преміум-функцій. Незважаючи на низький коефіцієнт економічної ефективності, продукт відповідає сучасним вимогам ринку, сприяє підвищенню доступності графічних інструментів і закладає основу для розвитку подібних рішень в Україні, особливо в освітній та творчій сферах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. CMake reference documentation – cmake 4.0.3 documentation. cmake.org: вебсайт. URL: https://cmake.org/cmake/help/latest/ (дата звернення: 14.05.2025).

2. Gimp. gimp.org: вебсайт. URL: https://www.gimp.org (дата звернення: 14.06.2025).

3. Https://krita.org/en/. krita.org: вебсайт. URL: https://krita.org (дата звернення: 14.06.2025).

4. Inkscape - draw freely. inkscape.org: вебсайт. URL: https://inkscape.org (дата звернення: 14.05.2025).

5. Neovide - neovide. neovide.dev: вебсайт. URL: https://neovide.dev/index.html (дата звернення: 14.05.2025).

6. Neovim. neovim.io: вебсайт. URL: https://neovim.io/doc/ (дата звернення: 14.06.2025).

7. QML applications | qt 6.9. doc.qt.io: вебсайт. URL: https://doc.qt.io/qt-6/qmlapplications.html (дата звернення: 23.05.2025).

8. Qt 6.9. doc.qt.io: вебсайт. URL: https://doc.qt.io/qt-6/index.html (дата звернення: 23.05.2025).

9. The standard : standard C++. isocpp.org: вебсайт. URL: https://isocpp.org/std/the-standard (дата звернення: 06.06.2025).

10. QML tutorial | qt quick | qt 6.9.1. doc.qt.io: вебсайт. URL: https://doc.qt.io/qt-6/qml-tutorial.html (дата звернення: 06.06.2025).

11. Qt quick controls | qt 6.9.1. doc.qt.io: вебсайт. URL: https://doc.qt.io/qt-6/qtquickcontrols2-index.html (дата звернення: 06.06.2025).

12. Qt quick QML types | qt quick | qt 6.9.1. doc.qt.io: вебсайт. URL: https://doc.qt.io/qt-6/qtquick-qmlmodule.html (дата звернення: 06.06.2025).

14. GitHub - Jarlok17/Harmoniq. github.com: вебсайт. URL: https://github.com/Jarlok17/Harmoniq (дата звернення: 15.06.2025).

ДОДАТКИ

Додаток А

Лістинг програмного коду створення нового документа та відкриття існуючого у стартовому екрані:

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Dialogs

import Harmoniq 1.0

Rectangle {
    id: startScreen
    anchors.fill: parent
    color: Themes.currentTheme.background

    property int canvasWidth: 800
    property int canvasHeight: 600
    property string canvasBackgroundColor: "White"

    Column {
        id: contentColumn
        anchors.centerIn: parent
        spacing: 20
        width: parent.width * 0.8

        Text {
            text: "Harmoniq"
            font.pixelSize: Math.min(parent.width,
parent.height) * 0.2
            font.bold: true
            anchors.horizontalCenter: parent.horizontalCenter
            color: "white"
            horizontalAlignment: Text.AlignHCenter
        }

        Button {
            text: "Create New Project"
            font.pixelSize: Math.min(parent.width,
parent.height) * 0.05
            anchors.horizontalCenter: parent.horizontalCenter
```

```
            background: Rectangle {
                color: "#5cdb95"
                radius: 10
                implicitWidth: 250
                implicitHeight: 50
            }
            onClicked: dialogNewImage.open()
        }

        Button {
            text: "Open Existing Project"
            font.pixelSize: Math.min(parent.width,
parent.height) * 0.05
            anchors.horizontalCenter: parent.horizontalCenter
            background: Rectangle {
                color: "#05386b"
                radius: 10
                implicitWidth: 250
                implicitHeight: 50
            }
            onClicked: fileDialog.open()
        }
    }

    FileDialog {
        id: fileDialog
        title: "Open a Project"
        fileMode: FileDialog.OpenFile
        nameFilters: ["Harmoniq Projects (*.json)"]
        onAccepted: {
            console.log("OPEN DIALOG ACCEPTED:", selectedFile)
            documentManager.loadFromFile(selectedFile)

            startScreen.visible = false
            topBarLoader.visible = true
            leftBarLoader.visible = true
            rightBarLoader.visible = true
            layerLoader.visible = true
        }
    }

    ColorDialog {
        id: colorDialog
        title: "Select the color..."
        onAccepted: {
            selectedColor.color = colorDialog.selectedColor;
```

```
            canvasBackgroundColor = colorDialog.selectedColor;
        }
    }

    Popup {
        id: dialogNewImage
        modal: true
        focus: true
        closePolicy: Popup.CloseOnEscape |
Popup.CloseOnPressOutside
        visible: false
        anchors.centerIn: Overlay.overlay

        background: Rectangle {
            color: Themes.currentTheme.background
            radius: 10
        }

        contentItem: Column {
            spacing: 10
            padding: 10

            Label {
                text: "Width:"
            }

            TextField {
                id: widthField
                placeholderText: "Enter width"
                text: "800"
            }

            Label {
                text: "Height:"
            }

            TextField {
                id: heightField
                placeholderText: "Enter height"
                text: "600"
            }

            Label {
                text: "Background Color:"
            }
```

```
            Row {
                spacing: 10
                Button {
                    id: selectColor
                    text: "select color"
                    onClicked: {
                        colorDialog.open()
                    }
                }
                Rectangle {
                    id: selectedColor
                    width: 50
                    height: 50
                    color: canvasBackgroundColor
                }
            }

            Row {
                spacing: 10
                Button {
                    text: "Create"
                    onClicked: {
                        if (parseInt(widthField.text) > 0 &&
parseInt(heightField.text) > 0) {
                            canvasWidth =
parseInt(widthField.text);
                            canvasHeight =
parseInt(heightField.text);

                            dialogNewImage.close();
                            startScreen.visible = false;
                            leftBarLoader.visible = true;
                            rightBarLoader.visible = true;
                            topBarLoader.visible = true;
                            layerLoader.visible = true;


documentManager.addDocument("Untitled", "");

documentManager.currentLayerManager.addLayer("background",
canvasWidth, canvasHeight, canvasBackgroundColor, false);
                        } else {
                            console.log("Invalid dimensions");
                        }
                        dialogNewImage.visible = false;
                        dialogNewImage.close();
```

```
                    }
                }
                Button {
                    text: "Cancel"
                    onClicked: dialogNewImage.close()
                }
            }
        }
    }

    Component.onCompleted: {
        contentColumn.width = Math.min(parent.width * 0.8, 400)
    }

    Connections {
        target: parent

        function onWidthChanged() {
           contentColumn.width = Math.min(parent.width * 0.8,
400)
        }
    }
}
```

Додаток Б

Лістинг програмного коду реалізації прокрутки та масштабування полотен у робочій області застосунку:

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import Harmoniq_backend 1.0
import Harmoniq 1.0

Flickable {
    id: flickableArea
    anchors.fill: parent
    clip: true
    interactive: false
    visible: true

    property real targetScale: 1.0
    property real contentW: 800
    property real contentH: 600

    contentWidth: Math.max(contentW * targetScale, width)
    contentHeight: Math.max(contentH * targetScale, height)

    ScrollBar.horizontal: ScrollBar {
        visible: targetScale > 1 ? true : false
        policy: ScrollBar.AsNeeded
    }
    ScrollBar.vertical: ScrollBar {
        visible: targetScale > 1 ? true : false
        policy: ScrollBar.AsNeeded
    }

    onContentWidthChanged: centerContent()
    onContentHeightChanged: centerContent()
    onWidthChanged: centerContent()
    onHeightChanged: centerContent()

    function centerContent() {
        contentX = (contentWidth - width) / 2
        contentY = (contentHeight - height) / 2
        contentX = Math.max(0, Math.min(contentX, contentWidth -
width))
        contentY = Math.max(0, Math.min(contentY, contentHeight
- height))
```

```
        console.log("Centered content: contentX =", contentX,
"contentY =", contentY)
    }

    Item {
        id: container
        width: flickableArea.contentW
        height: flickableArea.contentH
        x: (flickableArea.contentWidth - width *
flickableArea.targetScale) / 2
        y: (flickableArea.contentHeight - height *
flickableArea.targetScale) / 2

        Repeater {
            model: documentManager && documentManager.current ?
documentManager.currentLayerManager : null

            delegate: Item {
                id: layerItem
                width: model.width
                height: model.height
                x: (container.width - width) / 2
                y: (container.height - height) / 2
                visible: documentManager.current !== null &&
model.visible

                children: [model.layerObject]

                Component.onCompleted: {
                    flickableArea.contentW = width
                    flickableArea.contentH = height
                    console.log("CANVAS WIDTH:",
flickableArea.contentW, "HEIGHT:", flickableArea.contentH)
                    flickableArea.centerContent()
                }

                WheelHandler {
                    id: wheelHandler
                    acceptedDevices: PointerDevice.Mouse |
PointerDevice.TouchPad
                    onWheel: function(wheel) {
                        const scaleStep = 0.05
                        const maxScale = 2.5
                        const minScale = 0.5

                        let mouseX = wheel.x
```

```
                        let mouseY = wheel.y

                        let contentMouseX = (mouseX +
flickableArea.contentX - container.x - layerItem.x) /
flickableArea.targetScale
                        let contentMouseY = (mouseY +
flickableArea.contentY - container.y - layerItem.y) /
flickableArea.targetScale

                        let oldScale = flickableArea.targetScale

                        if (wheel.angleDelta.y > 0) {
                            flickableArea.targetScale =
Math.min(flickableArea.targetScale + scaleStep, maxScale)
                        } else if (wheel.angleDelta.y < 0) {
                            flickableArea.targetScale =
Math.max(flickableArea.targetScale - scaleStep, minScale)
                        }

                        flickableArea.contentX = contentMouseX *
flickableArea.targetScale + container.x + layerItem.x - mouseX
                        flickableArea.contentY = contentMouseY *
flickableArea.targetScale + container.y + layerItem.y - mouseY

                        console.log("Zoomed to scale:",
flickableArea.targetScale, "contentX:", flickableArea.contentX,
"contentY:", flickableArea.contentY)
                    }
                }

                transform: Scale {
                    origin.x: 0
                    origin.y: 0
                    xScale: flickableArea.targetScale
                    yScale: flickableArea.targetScale
                }
            }
        }
    }

    Rectangle {
        width: parent.width
        height: parent.height
        color: "transparent"
        visible: documentManager && documentManager.current ===
null
```

Додаток В

Лістинг програмного коду динамічного створення вкладок

для відкритих документів:

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Dialogs
import Harmoniq 1.0
import QtQuick.Window

Rectangle {
    color: Qt.darker(Themes.currentTheme.background, 0.6)
    anchors.fill: parent

    Dialog {
        id: confirmCloseDialog
        title: "Save Changes?"
        width: 400
        height: 200
        modal: true
        property int indexToClose: -1
        anchors.centerIn: Overlay.overlay

        contentItem: Item {
            id: contentContainer
            anchors.fill: parent
            anchors.margins: 8 // Відступи для уникнення
обрізання
            visible: true
            opacity: 1.0

            Text {
                id: dialogText
                text: {
                    if (documentManager &&
confirmCloseDialog.indexToClose >= 0) {
                        var doc =
documentManager.get(confirmCloseDialog.indexToClose);
                        return "The document '" + (doc &&
doc.name ? doc.name : "Untitled") + "' has unsaved changes. Do
you want to save them before closing?";
                    }
```

```
                    return "The document has unsaved changes. Do
you want to save them before closing?";
                }
                color: "white"
                wrapMode: Text.WordWrap
                font.pixelSize: 14 // Явно задано розмір шрифту
                padding: 10 // Зменшено padding для кращого
балансу
                width: parent.width - 16 // Ширина з урахуванням
відступів
                height: Math.max(contentHeight + padding * 2,
50) // Динамічна висота з мінімальним значенням
                anchors.horizontalCenter:
parent.horizontalCenter
                anchors.verticalCenter: parent.verticalCenter
                visible: true
                opacity: 1.0
                onTextChanged: function(newText) {
                    console.log("Dialog text updated:",
newText);
                    console.log("Text size:", width,
implicitHeight, "Position:", x, y, "Visible:", visible,
"Opacity:", opacity);
                    console.log("Content container size:",
contentContainer.width, contentContainer.height);
                }
            }
        }

        background: Rectangle {
            color: Themes.currentTheme.background
            border.color: Themes.currentTheme.accent
            opacity: 1.0
        }

        footer: DialogButtonBox {
            Button {
                text: "Save"
                onClicked: confirmCloseDialog.accept()
            }
            Button {
                text: "Discard"
                onClicked: confirmCloseDialog.reject()
            }
            Button {
                text: "Cancel"
```

```
                onClicked: confirmCloseDialog.close()
            }
        }

        onAboutToShow: {
            console.log("Opening dialog for index:",
indexToClose, "Document:", documentManager.get(indexToClose));
            console.log("Dialog size:", width, height,
"Background color:", Themes.currentTheme.background, "Text
color:", dialogText.color);
            console.log("Text size:", dialogText.width,
dialogText.implicitHeight, "Position:", dialogText.x,
dialogText.y);
            console.log("Content container size:",
contentContainer.width, contentContainer.height);
        }

        onVisibleChanged: console.log("Dialog visible changed:",
visible)

        onAccepted: {
            console.log("Saving document before closing,
index:", indexToClose);
            var doc = documentManager.get(indexToClose);
            const currentPath = doc ? doc.path : "";
            console.log("Save triggered with path:",
currentPath);
            if (!currentPath || currentPath.trim() === "") {
                saveDialog.open();
            } else {
                const success =
documentManager.saveToFile(currentPath);
                if (success) {
                    console.log("Saved successfully to:",
currentPath);
                    if (doc) {
                        doc.setModified(false);

documentManager.removeDocument(indexToClose);
                    }
                } else {
                    console.warn("Save failed, closing cancelled
for index:", indexToClose);
                }
            }
        }
```

```
        onRejected: {
            console.log("Discarding changes, closing document,
index:", indexToClose);
            documentManager.removeDocument(indexToClose);
        }

        onClosed: {
            if (!accepted && !rejected) { // Аналог
Dialog.Cancel
                console.log("Closing cancelled for document,
index:", indexToClose);
            }
        }
    }
    FileDialog {
        id: saveDialog
        title: qsTr("Save Project As")
        fileMode: FileDialog.SaveFile
        nameFilters: ["Harmoniq Projects (*.json)", "All Files
(*)"]
        onAccepted: {
            if (selectedFile && selectedFile !== "") {
                var fullPath = selectedFile.toString();
                var localPath = fullPath.replace("file://", "");
                var fileName =
localPath.split('/').pop().split('.json')[0];
                if (documentManager && documentManager.current)
{
                    documentManager.current.setName(fileName);
                    documentManager.current.setPath(localPath);
                    documentManager.saveToFile(selectedFile);
                    documentManager.current.setModified(false);
                }
            }
        }
    }

    TabBar {
        id: tabBar
        width: parent.width

        Repeater {
            model: documentManager

            delegate: TabButton {
```

```
                id: tabButton
                property bool isModified:
documentManager.get(index) ? documentManager.get(index).modified
: false
                text: {
                    var baseName = name || "Untitled";
                    return isModified ? baseName + "*" :
baseName;
                }
                checked: documentManager.currentIndex === index
                checkable: true
                implicitHeight: 32
                implicitWidth: Math.max(100, text.length * 7 +
30)

                font.pixelSize: 12

                background: Rectangle {
                    radius: 6
                    border.color: tabButton.checked ?
Themes.currentTheme.accent : "transparent"
                    color: tabButton.checked ?
Qt.darker(Themes.currentTheme.background, 0.7) :
Themes.currentTheme.background
                }

                contentItem: Item {
                    anchors.fill: parent

                    Text {
                        id: tabText
                        text: tabButton.text
                        color: "white"
                        font.pixelSize: 12
                        anchors.verticalCenter:
parent.verticalCenter
                        anchors.left: parent.left
                        anchors.leftMargin: 8
                        anchors.right: closeButton.left
                        anchors.rightMargin: 8
                        elide: Text.ElideRight
                    }

                    Button {
                        id: closeButton
                        text: "×"
```

```
                        font.pixelSize: 10
                        anchors.verticalCenter:
parent.verticalCenter
                        anchors.right: parent.right
                        anchors.rightMargin: 2
                        background: Rectangle {
                            color: "transparent"
                        }
                        onClicked: {
                            if (isModified) {
                                confirmCloseDialog.indexToClose
= index;
                                confirmCloseDialog.open();
                            } else {
                                console.log("Closing unmodified
document, index:", index);

documentManager.removeDocument(index);
                            }
                        }
                    }
                }
                onClicked:
documentManager.setCurrentIndex(index)

                Connections {
                    target: documentManager.get(index)
                    function onModifiedChanged() {
                        tabButton.isModified =
documentManager.get(index).modified;
                    }
                }
            }
        }
    }
```

Додаток Г

Лістинг програмного коду компонента вибору кольору з історією кольорів:

```
import QtQuick 2.15
import QtQuick.Controls 2.15

import Harmoniq 1.0
import Harmoniq_backend 1.0

Item {
    id: colorPicker
    anchors.fill: parent

    property real hue: 0.0
    property real saturation: 1.0
    property real value: 1.0
    property real alpha: 1.0
    readonly property color selectedColor: Qt.hsva(hue,
saturation, value, alpha)

    ListModel {
        id: colorHistory
    }

    Timer {
        id: colorHistoryTimer
        interval: 500
        running: false
        repeat: false
        onTriggered: {
            addColorToHistory(colorPicker.selectedColor);
        }
    }

    function addColorToHistory(color) {
        var qmlColor = Qt.color(color);
        if (qmlColor.a === 0) {
            console.log("Skipping transparent color");
            return;
        }

        for (var i = 0; i < colorHistory.count; i++) {
```

```
            var existingColor = colorHistory.get(i).color;
            if (Math.abs(existingColor.r - qmlColor.r) < 0.01 &&
                Math.abs(existingColor.g - qmlColor.g) < 0.01 &&
                Math.abs(existingColor.b - qmlColor.b) < 0.01 &&
                Math.abs(existingColor.a - qmlColor.a) < 0.01) {
                console.log("Skipping near-duplicate color:",
qmlColor.toString());
                return;
            }
        }

        console.log("Adding color to history:",
qmlColor.toString());
        colorHistory.insert(0, { "color": qmlColor });
        if (colorHistory.count > 10) {
            colorHistory.remove(10);
        }
    }

    Column {
        anchors.fill: parent
        anchors.leftMargin: 10
        anchors.rightMargin: 10
        anchors.topMargin: 10
        spacing: 10

        // HUE BAR
        Rectangle {
            id: hueBar
            width: parent.width
            height: 20
            radius: 4
            gradient: Gradient {
                orientation: Gradient.Horizontal
                GradientStop { position: 0.0; color: "red" }
                GradientStop { position: 0.17; color: "yellow" }
                GradientStop { position: 0.33; color: "green" }
                GradientStop { position: 0.5; color: "cyan" }
                GradientStop { position: 0.67; color: "blue" }
                GradientStop { position: 0.83; color:
"magenta" }
                GradientStop { position: 1.0; color: "red" }
            }

            Rectangle {
                id: hueHandle
```

```
                width: 4
                height: parent.height
                x: colorPicker.hue * parent.width - width / 2
                color: "white"
            }

            MouseArea {
                anchors.fill: parent
                onPositionChanged: (mouse) => {
                    colorPicker.hue = Math.max(0, Math.min(1,
mouse.x / hueBar.width));
                    colorHistoryTimer.restart();
                }
                onPressed: (mouse) => {
                    colorPicker.hue = Math.max(0, Math.min(1,
mouse.x / hueBar.width));
                    colorHistoryTimer.restart();
                }
                onReleased: {
                    colorHistoryTimer.restart();
                }
            }
        }

        // SATURATION/VALUE FIELD
        Rectangle {
            id: svField
            width: parent.width
            height: 200
            clip: true

            Rectangle {
                anchors.fill: parent
                gradient: Gradient {
                    orientation: Gradient.Horizontal
                    GradientStop { position: 0.0; color: "white"
}
                    GradientStop { position: 1.0; color:
Qt.hsva(colorPicker.hue, 1, 1, 1) }
                }
            }

            Rectangle {
                anchors.fill: parent
                gradient: Gradient {
                    orientation: Gradient.Vertical
```

```
                    GradientStop { position: 0.0; color:
"transparent" }
                    GradientStop { position: 1.0; color: "black"
}
                }
            }

            Rectangle {
                id: selector
                width: 12
                height: 12
                radius: 6
                border.color: "white"
                border.width: 2
                color: "transparent"
                z: 2

                x: colorPicker.saturation * svField.width -
width / 2
                y: (1 - colorPicker.value) * svField.height -
height / 2
            }

            MouseArea {
                anchors.fill: parent
                onPositionChanged: (mouse) => {
                    colorPicker.saturation = Math.max(0,
Math.min(1, mouse.x / svField.width));
                    colorPicker.value = Math.max(0, Math.min(1,
1 - (mouse.y / svField.height)));
                    colorHistoryTimer.restart(); // Restart
timer on SV change
                }
                onPressed: (mouse) => {
                    colorPicker.saturation = Math.max(0,
Math.min(1, mouse.x / svField.width));
                    colorPicker.value = Math.max(0, Math.min(1,
1 - (mouse.y / svField.height)));
                    colorHistoryTimer.restart();
                }
                onReleased: {
                    colorHistoryTimer.restart();
                }
            }
        }
```

```
        // ALPHA BAR
        Rectangle {
            id: alphaBar
            width: parent.width
            height: 20
            radius: 4
            clip: true

            Repeater {
                model: 20
                delegate: Rectangle {
                    width: alphaBar.width / 20
                    height: alphaBar.height
                    color: (index % 2 === 0) ? "#ccc" : "#eee"
                    x: index * width
                }
            }

            Rectangle {
                anchors.fill: parent
                gradient: Gradient {
                    orientation: Gradient.Horizontal
                    GradientStop { position: 1.0; color:
Qt.hsva(colorPicker.hue, colorPicker.saturation,
colorPicker.value, 1.0) }
                    GradientStop { position: 0.0; color:
Qt.hsva(colorPicker.hue, colorPicker.saturation,
colorPicker.value, 0.0) }
                }
            }

            Rectangle {
                id: alphaHandle
                width: 4
                height: parent.height
                x: colorPicker.alpha * parent.width - width / 2
                color: "white"
            }

            MouseArea {
                anchors.fill: parent
                onPositionChanged: (mouse) => {
                    colorPicker.alpha = Math.max(0, Math.min(1,
mouse.x / alphaBar.width));
                    colorHistoryTimer.restart(); // Restart
timer on alpha change
```

```
                }
                onPressed: (mouse) => {
                    colorPicker.alpha = Math.max(0, Math.min(1,
mouse.x / hueBar.width));
                    colorHistoryTimer.restart();
                }
                onReleased: {
                    colorHistoryTimer.restart();
                }
            }
        }

        // COLOR PREVIEW
        Rectangle {
            width: parent.width
            height: 30
            color: colorPicker.selectedColor
            border.color: "white"

            Text {
                anchors.centerIn: parent
                color: "white"
                text: colorPicker.selectedColor
                font.pixelSize: 14
            }
        }

        // COLOR SWATCHES
        Grid {
            width: parent.width
            columns: 5
            rows: 2
            spacing: 5

            Repeater {
                model: colorHistory
                delegate: Rectangle {
                    width: (parent.width - (parent.columns - 1)
* parent.spacing) / parent.columns
                    height: 24
                    color: model.color
                    border.color: "white"
                    border.width: 1
```

Додаток Ґ

Лістинг програмного коду побудови лівої панелі інструментів:

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts 2.15

import Harmoniq 1.0
import harmoniq 1.0

Item {
    id: leftMenu
    width: Screen.width * 0.05
    height: parent.height

    property list<ToolButton> buttonManager: []

    Rectangle {
        id: menuBackground
        color: Themes.currentTheme.background
        anchors.fill: parent
        border.color: Qt.darker(Themes.currentTheme.background,
0.5)

        ColumnLayout {
            id: buttonContainer
            anchors.top: parent.top
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.topMargin: 5

            GridLayout {
                columns: 2
                Repeater {
                    model: [
                        {icon: "qrc:/Icons/64x64/brush.png",
action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Br
ush); console.log("Brush PNG"); }},
                        {icon: "qrc:/Icons/64x64/erase.png",
action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Er
aser); console.log("PEN PNG"); }}
```

```
                    ]

                    delegate: ToolButton {
                        width: leftMenu.width * 0.40
                        iconSource: modelData.icon
                        buttonManager: leftMenu.buttonManager
                        onClickedAction: modelData.action
                    }
                }
            }

            GridLayout {
                columns: 2
                Repeater {
                    model: [
                        {icon: "qrc:/Icons/64x64/color-
fill.png", action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Fi
ll); console.log("Fill PNG"); }},
                        {icon: "qrc:/Icons/64x64/color-
picker.png", action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Pi
cker); console.log("Color-Picker PNG"); }}
                    ]

                    delegate: ToolButton {
                        width: leftMenu.width * 0.40
                        iconSource: modelData.icon
                        buttonManager: leftMenu.buttonManager
                        onClickedAction: modelData.action
                    }
                }
            }

            Rectangle {
                height: 1
                width: parent.width * 0.9
                color:
Qt.lighter(Themes.currentTheme.background, 1.5)
                Layout.alignment: Qt.AlignHCenter
            }

            GridLayout {
                columns: 2
                Repeater {
                    model: [
```

```
                        {icon: "qrc:/Icons/64x64/move-
arrows.png", action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Mo
ve); console.log("Move Arrows PNG"); }},
                        {icon: "qrc:/Icons/64x64/shape-
tool.png", action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Sh
ape); console.log("Shape-Tool PNG"); }}
                    ]

                    delegate: ToolButton {
                        width: leftMenu.width * 0.40
                        iconSource: modelData.icon
                        buttonManager: leftMenu.buttonManager
                        onClickedAction: modelData.action
                    }
                }
            }

            GridLayout {
                columns: 2
                Repeater {
                    model: [
                        {icon: "qrc:/Icons/64x64/line-tool.png",
action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Li
ne); console.log("Line Tool PNG"); }},
                        {icon: "qrc:/Icons/64x64/selection-
tool.png", action: function()
{ documentManager.currentLayerManager.setCurrentTool(ToolType.Se
lection); console.log("Selection Tool PNG"); }}
                    ]

                    delegate: ToolButton {
                        width: leftMenu.width * 0.40
                        iconSource: modelData.icon
                        buttonManager: leftMenu.buttonManager
                        onClickedAction: modelData.action
                    }
                }
            }
        }
```

Додаток Д

Лістинг програмного коду відображення списку полотен та вибору кольору у правій панелі:

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts 2.15
import QtQuick.Dialogs

import Harmoniq 1.0
import Harmoniq_backend 1.0

Rectangle {
    id: rightBar
    width: Screen.width * 0.16
    height: parent.height
    color: Themes.currentTheme.background
    border.color: Qt.darker(Themes.currentTheme.background, 0.5)

    property bool isVisible: false
    property real hiddenX: parent.width
    property real visibleX: parent.width - width

    property double layerWidth: 800
    property double layerHeight: 600

    Rectangle {
        id: colorContainer
        width: rightBar.width
        height: 0
        color: Themes.currentTheme.background
        border.color: Themes.currentTheme.accent
        anchors.top: rightBar.top

        Loader {
            id: colorPickerLoader
            anchors.fill: parent
            source: "qrc:/UI/ColorPicker.qml"

            onLoaded: {
                if (colorPickerLoader.item) {
                    ToolSettings.color =
colorPickerLoader.item.selectedColor
```

```
                }
            }
        }

        Connections {
            target: colorPickerLoader.item
            onHueChanged: ToolSettings.color =
colorPickerLoader.item.selectedColor
            onSaturationChanged: ToolSettings.color =
colorPickerLoader.item.selectedColor
            onValueChanged: ToolSettings.color =
colorPickerLoader.item.selectedColor
            onAlphaChanged: ToolSettings.color =
colorPickerLoader.item.selectedColor
        }
    }

    Rectangle {
        id: bottomContainer
        width: rightBar.width
        height: rightBar.height / 2
        color: Themes.currentTheme.background
        border.color: Qt.darker(Themes.currentTheme.background,
0.5)
        anchors.bottom: rightBar.bottom

        Rectangle {
            id: layerContainerBottomMenu
            width: bottomContainer.width
            height: 50
            color: Themes.currentTheme.background
            border.color:
Qt.darker(Themes.currentTheme.background, 0.5)
            anchors.bottom: bottomContainer.bottom

            Rectangle {
                id: addButton
                width: 40
                height: 40
                color: "transparent"
                anchors.left: parent.left
                anchors.verticalCenter: parent.verticalCenter

                MouseArea {
                    anchors.fill: parent
                    onClicked: {
```

```
                        var layerName = "New Layer";

documentManager.currentLayerManager.addLayer(layerName + " " +
documentManager.currentLayerManager.rowCount(),
rightBar.layerWidth, rightBar.layerHeight, Qt.rgba(0, 0, 0, 0));
                    }
                    Image {
                        anchors.centerIn: parent
                        source: "qrc:/Icons/64x64/add.png"
                        sourceSize: Qt.size(32, 32)
                    }
                    cursorShape: Qt.PointingHandCursor
                }
            }

            Rectangle {
                Layout.fillWidth: true
                width: parent.width - 100
                height: 40
                anchors.centerIn: parent
                color: "transparent"

                Rectangle {
                    id: upArrowButton
                    width: 40
                    height: 40
                    color: "transparent"
                    anchors.left: parent.left
                    anchors.verticalCenter:
parent.verticalCenter

                    MouseArea {
                        anchors.fill: parent
                        onClicked: {
                            if (layersList.currentIndex <
layersList.count - 1) {

documentManager.currentLayerManager.moveLayer(layersList.current
Index, layersList.currentIndex + 1);
                            }
                        }
                        Image {
                            anchors.centerIn: parent
                            source: "qrc:/Icons/64x64/up-
arrow.png"
                            sourceSize: Qt.size(32, 32)
```

```
                        }
                        cursorShape: Qt.PointingHandCursor
                    }
                }
                Rectangle {
                    id: downArrowButton
                    width: 40
                    height: 40
                    color: "transparent"
                    anchors.left: upArrowButton.right
                    anchors.verticalCenter:
parent.verticalCenter

                    MouseArea {
                        anchors.fill: parent
                        onClicked: {
                            if (layersList.currentIndex > 0) {

documentManager.currentLayerManager.moveLayer(layersList.current
Index, layersList.currentIndex - 1);
                            }
                        }
                        Image {
                            anchors.centerIn: parent
                            source: "qrc:/Icons/64x64/down-
arrow.png"
                            sourceSize: Qt.size(32, 32)
                        }
                        cursorShape: Qt.PointingHandCursor
                    }
                }
            }

            Rectangle {
                id: trashButton
                width: 40
                height: 40
                color: "transparent"
                anchors.right: parent.right
                anchors.verticalCenter: parent.verticalCenter

                MouseArea {
                    anchors.fill: parent
                    onClicked: {
```

```
documentManager.currentLayerManager.removeLayer(layersList.curre
ntIndex);
                    }
                    Image {
                        anchors.centerIn: parent
                        source: "qrc:/Icons/64x64/trash.png"
                        sourceSize: Qt.size(32, 32)
                    }
                    cursorShape: Qt.PointingHandCursor
                }
            }
        }

        ListView {
            id: layersList
            width: bottomContainer.width
            height: bottomContainer.height -
layerContainerBottomMenu.height
            model: documentManager && documentManager.current ?
documentManager.currentLayerManager : null
            anchors.top: bottomContainer.top
            clip: true
            verticalLayoutDirection: ListView.BottomToTop

            onCountChanged: {
                console.log("Layers count changed:",
layersList.count);
                forceThumbnailUpdate(); // Примусове оновлення
після зміни кількості шарів
            }

            function forceThumbnailUpdate() {
                console.log("Forcing thumbnail update for all
layers");
                for (var i = 0; i < layersList.count; i++) {
                    var item = layersList.itemAtIndex(i);
                    if (item && item.layerThumbnail) {
                        item.layerThumbnail.source = "";
                        item.layerThumbnail.source =
"image://thumbnail/" + i + "?v=" + new Date().getTime();
                        console.log("Updated thumbnail source
for index:", i);
                    } else {
                        console.log("Failed to update thumbnail
for index:", i, "Item:", item);
```

```
                    }
                }
            }

            Connections {
                target: documentManager
                function onLayerManagerChanged() {
                    console.log("QML: LayerManager set");
                    if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                        layersList.currentIndex =
documentManager.currentLayerManager.currentIndex;
                        console.log("QML: Initial currentIndex
set to:", layersList.currentIndex);
                        forceThumbnailUpdate();
                    }
                }
            }

            Connections {
                target: documentManager &&
documentManager.current ? documentManager.currentLayerManager :
null
                ignoreUnknownSignals: true
                function onCurrentIndexChanged() {
                    if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                        const index =
documentManager.currentLayerManager.currentIndex;
                        console.log("QML: LayerManager
currentIndexChanged to:", index);
                        layersList.currentIndex = index;
                    }
                }
            }

            delegate: Rectangle {
                id: nameContainer
                width: parent ? parent.width : rightBar.width
                height: 60
                color: Themes.currentTheme.background
                border.color: ListView.isCurrentItem ? "white" :
Qt.darker(Themes.currentTheme.background, 0.5)
```

```
                property bool isEditing: false

                Rectangle {
                    width: 40
                    height: 40
                    color: "transparent"
                    anchors.left: parent.left
                    anchors.verticalCenter:
parent.verticalCenter

                    MouseArea {
                        anchors.fill: parent
                        acceptedButtons: Qt.LeftButton
                        onClicked: {
                            if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                                console.log("Toggling visibility
for layer, UI index:", index);

documentManager.currentLayerManager.setLayerVisible(index, !
model.visible);
                            }
                        }
                        Image {
                            anchors.centerIn: parent
                            source: model ? (model.visible ?
"qrc:/Icons/64x64/open-eye.png" : "qrc:/Icons/64x64/closed-
eye.png") : ""
                            sourceSize: Qt.size(24, 24)
                        }
                        cursorShape: Qt.PointingHandCursor
                    }
                }

                Image {
                    id: layerThumbnail
                    width: 40
                    height: 40
                    anchors.left: parent.left
                    anchors.leftMargin: 45
                    anchors.verticalCenter:
parent.verticalCenter
                    source: documentManager &&
documentManager.current && documentManager.currentLayerManager ?
"image://thumbnail/" + index + "?v=" + new Date().getTime() : ""
```

```
                    fillMode: Image.PreserveAspectFit
                    cache: false
                    asynchronous: true

                    Connections {
                        target: documentManager &&
documentManager.current ? documentManager.currentLayerManager :
null
                        ignoreUnknownSignals: true
                        function
onLayerImageChanged(changedIndex) {
                            if (changedIndex === index) {
                                console.log("QML: Layer image
changed for index:", changedIndex);
                                layerThumbnail.source = "";
                                layerThumbnail.source =
"image://thumbnail/" + index + "?v=" + new Date().getTime();
                                console.log("Thumbnail updated
for index:", index);
                            }
                        }
                    }
                }

                Rectangle {
                    width: parent.width - 100 - 40 - 10
                    height: 40
                    anchors.left: layerThumbnail.right
                    anchors.leftMargin: 5
                    anchors.verticalCenter:
parent.verticalCenter
                    color: "transparent"

                    TextField {
                        id: layerNameField
                        visible: parent.parent.isEditing
                        anchors.fill: parent
                        anchors.margins: 3
                        text: model ? name : ""
                        color: "white"
                        background: Rectangle {
                            color:
Themes.currentTheme.background
                            border.color:
Themes.currentTheme.accent
                            radius: 4
```

```
                        }
                        onEditingFinished: {
                            if (documentManager &&
documentManager.current && documentManager.currentLayerManager
&& text.trim() !== "") {

documentManager.currentLayerManager.setLayerName(index,
text.trim());
                            }
                            console.log("Editing finished,
setting isEditing to false");
                            parent.parent.isEditing = false;
                            Qt.callLater(function() {
                                layerNameField.visible = false;
                            });
                        }
                        onFocusChanged: {
                            if (!focus) {
                                console.log("Focus lost, setting
isEditing to false");
                                parent.parent.isEditing = false;
                                Qt.callLater(function() {
                                    layerNameField.visible =
false;
                                });
                            }
                        }
                    }

                    MouseArea {
                        anchors.fill: parent
                        onDoubleClicked: {
                            if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                                console.log("Double-clicked on
layer, UI index:", index);
                                parent.parent.isEditing = true;
                                layerNameField.visible = true;

layerNameField.forceActiveFocus();
                                layerNameField.selectAll();
                            }
                        }
                        onClicked: {
```

```
                            if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                                console.log("Selecting layer, UI
index:", index);
                                layersList.currentIndex = index;

documentManager.currentLayerManager.setCurrentIndex(index);
                            }
                        }
                        cursorShape: Qt.PointingHandCursor

                        Text {
                            visible: !nameContainer.isEditing &&
model
                            anchors.centerIn: parent
                            text: model ? name : ""
                            color: "white"
                        }
                    }
                }

                Rectangle {
                    width: 40
                    height: 40
                    color: "transparent"
                    anchors.right: parent.right
                    anchors.verticalCenter:
parent.verticalCenter

                    MouseArea {
                        anchors.fill: parent
                        onClicked: {
                            if (documentManager &&
documentManager.current && documentManager.currentLayerManager)
{
                                console.log("Toggling lock for
layer, UI index:", index);

documentManager.currentLayerManager.setLayerLocked(index, !
locked);
                            }
                        }
                        Image {
                            anchors.centerIn: parent
```

```
                            source: model ? (locked ?
"qrc:/Icons/64x64/locked.png" : "qrc:/Icons/64x64/unlocked.png")
: ""
                            sourceSize: Qt.size(32, 32)
                        }
                        cursorShape: Qt.PointingHandCursor
                    }
                }

                Component.onCompleted: {
                    console.log("Delegate initialized for
index:", index);
                }
            }
        }
    }

    Behavior on x {
        NumberAnimation {
            duration: 300
            easing.type: Easing.InOutQuad
        }
    }
}
```

Додаток Е

Лістинг програмного коду керування розміром

інструменту та увімкненням заливки:

```
import QtQuick 2.15
import QtQuick.Window 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15

import Harmoniq 1.0
import Harmoniq_backend 1.0

Item {
    id: topBar
    width: parent.width
    height: 30
    anchors {
        left: parent.left
        right: parent.right
        top: parent.top
    }

    Rectangle {
        id: menuBackground
        color: currentTheme.background
        anchors.fill: parent
        border.color: Qt.darker(Themes.currentTheme.background,
0.5)

        RowLayout {
            id: topBarLayout
            anchors.fill: parent
            anchors.margins: 0
            spacing: 8

            // Блок для "Size" і Slider
            RowLayout {
                spacing: 4
                Layout.alignment: Qt.AlignVCenter

                Label {
                    text: "Size:"
                    color: "white"
```

```
                    font.pixelSize: 12
                    font.bold: true
                    Layout.preferredWidth: 40
                    Layout.preferredHeight: 30
                    Layout.alignment: Qt.AlignVCenter
                    verticalAlignment: Text.AlignVCenter
                    Layout.leftMargin: 6
                }

                Slider {
                    id: sizeSlider
                    from: 1
                    to: 100
                    value: ToolSettings.size
                    onMoved: ToolSettings.size = value
                    Layout.fillWidth: true
                    Layout.preferredWidth: 120
                    Layout.maximumWidth: 200
                    Layout.preferredHeight: 30
                    Layout.alignment: Qt.AlignVCenter

                    background: Rectangle {
                        anchors.verticalCenter:
parent.verticalCenter
                        width: parent.availableWidth
                        height: 10
                        color: "gray"
                        radius: 2
                    }

                    handle: Rectangle {
                        x: parent.leftPadding +
parent.visualPosition * (parent.availableWidth - width)
                        y: parent.topPadding +
parent.availableHeight / 2 - height / 2
                        implicitWidth: 12
                        implicitHeight: 12
                        radius: 6
                        color: parent.pressed ? "lightgray" :
"white"
                        border.color: "gray"
                    }
                }
            }

            RowLayout {
```

```
                spacing: 4
                Layout.alignment: Qt.AlignVCenter

                Label {
                    text: "Fill:"
                    color: "white"
                    font.pixelSize: 12
                    font.bold: true
                    Layout.preferredWidth: 30
                    Layout.preferredHeight: 30
                    Layout.alignment: Qt.AlignVCenter
                    verticalAlignment: Text.AlignVCenter
                }

                CheckBox {
                    checked: ToolSettings.fill
                    onToggled: ToolSettings.fill = checked
                    Layout.preferredWidth: 24
                    Layout.preferredHeight: 30
                    Layout.alignment: Qt.AlignVCenter

                    indicator: Rectangle {
                        anchors.centerIn: parent
                        implicitWidth: 16
                        implicitHeight: 16
                        radius: 3
                        border.color: "gray"
                        border.width: 1
                        color: parent.checked ?
Themes.currentTheme.accent : "transparent"

                        Rectangle {
                            anchors.centerIn: parent
                            width: 10
                            height: 10
                            color: "transparent"
                            border.color: "white"
                            border.width: 1
                            radius: 2
                            visible: !parent.parent.checked
                        }
                    }
```

Додаток Є

Лістинг програмного коду реалізації верхнього меню з діями над документом та історією:

```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs
import QtQuick.Window
import Harmoniq 1.0
import Harmoniq_backend 1.0

MenuBar {
    id: menuBar

    property int canvasWidth: 800
    property int canvasHeight: 600
    property string canvasBackgroundColor: "White"

    Menu {
        font.pixelSize: Themes.currentTheme.fontSize
        title: qsTr("File")
        MenuItem {
            text: qsTr("New...")
            onTriggered: dialogNewImage.open()
        }
        MenuItem {
            text: qsTr("Open...")
            onTriggered: openDialog.open()
        }
        Action {
            text: qsTr("Save...")
            enabled: documentManager &&
documentManager.current ? true : false
            shortcut: "Ctrl+S"
            onTriggered: {
                const currentPath = documentManager &&
documentManager.current ? documentManager.current.path : "";
                console.log("Save triggered with path:",
currentPath);
                if (!currentPath || currentPath === "") {
                    saveDialog.open();
                } else {
```

```
                    const success =
documentManager.saveToFile(currentPath);
                    if (documentManager &&
documentManager.current) {
                        if (!success) {
                            console.warn("Save failed, modified
state unchanged");
                        } else {

documentManager.current.setModified(false);
                            console.log("Saved successfully
to:", currentPath);
                        }
                    }
                }
            }
        }
        MenuItem {
            text: qsTr("Save as...")
            enabled: documentManager &&
documentManager.current ? true : false
            onTriggered: saveDialog.open()
        }
        MenuItem {
            text: qsTr("Export as image...")
            enabled: documentManager &&
documentManager.current ? true : false
            onTriggered: exportDialog.open()
        }
    }

    Connections {
        target: documentManager && documentManager.current ?
documentManager.current.historyManager : null
        function onStateChanged() {
            undoAction.enabled =
documentManager.current.historyManager.canUndo();
            redoAction.enabled =
documentManager.current.historyManager.canRedo();
        }
    }

    Menu {
        title: qsTr("Edit")
        font.pixelSize: Themes.currentTheme.fontSize
        Action {
```

```
            id: undoAction
            text: qsTr("Undo")
            shortcut: "Ctrl+Z"
            enabled: documentManager &&
documentManager.current ?
documentManager.current.historyManager.canUndo() : false
            onTriggered: {
                console.log("Undo triggered. Can undo:",
documentManager.current.historyManager.canUndo());
                documentManager.current.historyManager.undo();
            }
        }
        Action {
            id: redoAction
            text: qsTr("Redo")
            shortcut: "Ctrl+Shift+Z"
            enabled: documentManager &&
documentManager.current ?
documentManager.current.historyManager.canRedo() : false
            onTriggered: {
                console.log("Redo triggered. Can redo:",
documentManager.current.historyManager.canRedo());
                documentManager.current.historyManager.redo();
            }
        }
    }

    Menu {
        title: qsTr("View")
        MenuItem {
            text: qsTr("Toggle Left Toolbar")
            font.pixelSize: Themes.currentTheme.fontSize
            enabled: documentManager &&
documentManager.current ? true : false
            onTriggered: {
                leftBarLoader.visible = !leftBarLoader.visible
                console.log("Left Toolbar visibility:",
leftBarLoader.visible)
            }
        }
        MenuItem {
            text: qsTr("Toggle Right Toolbar")
            font.pixelSize: Themes.currentTheme.fontSize
            enabled: documentManager &&
documentManager.current ? true : false
            onTriggered: {
```

```
                rightBarLoader.visible = !rightBarLoader.visible
                console.log("Right Toolbar visibility:",
rightBarLoader.visible)
            }
        }
        MenuItem {
            text: qsTr("Toggle Top Panel")
            font.pixelSize: Themes.currentTheme.fontSize
            enabled: documentManager &&
documentManager.current ? true : false
            onTriggered: {
                topBarLoader.visible = !tpoBarLoader.visible
                console.log("Layers Panel visibility:",
layerLoader.visible)
            }
        }
    }

    Menu {
        title: qsTr("Help")
        MenuItem {
            text: qsTr("About")
            font.pixelSize: Themes.currentTheme.fontSize
            onTriggered: aboutWindow.visible = true
        }
    }

    Window {
        id: aboutWindow
        title: qsTr("Про Harmoniq")
        maximumWidth: 500
        maximumHeight: 200
        minimumWidth: 500
        minimumHeight: 200
        visible: false
        color: Themes.currentTheme.background
        modality: Qt.ApplicationModal
        flags: Qt.Dialog

        x: (Screen.width - width) / 2
        y: (Screen.height - height) / 2

        Column {
            anchors.centerIn: parent
            spacing: 10
            Text {
```

```
                text: qsTr("Harmoniq - кросплатформний графічний
застосунок\nВерсія: 1.0\nАвтор: Ярослав\nЛіцензія: MIT")
                font.pixelSize: Themes.currentTheme.fontSize
                color: "white"
            }
            Text {
                text: qsTr("© 2025 Harmoniq. Усі права
захищено.")
                font.pixelSize: Themes.currentTheme.fontSize - 2
                color: "white"
            }
            Button {
                text: qsTr("OK")
                anchors.horizontalCenter:
parent.horizontalCenter
                onClicked: aboutWindow.visible = false
            }
        }
    }
    FileDialog {
        id: saveDialog
        title: qsTr("Save Project As")
        fileMode: FileDialog.SaveFile
        nameFilters: ["Harmoniq Projects (*.json)", "All Files
(*)"]
        onAccepted: {
            if (selectedFile && selectedFile !== "") {
                var fullPath = selectedFile.toString();
                var localPath = fullPath.replace("file://", "");
                var fileName =
localPath.split('/').pop().split('.json')[0];
                if (documentManager && documentManager.current)
{
                    documentManager.current.setName(fileName);
                    documentManager.current.setPath(localPath);
                    documentManager.saveToFile(selectedFile);
                    documentManager.current.setModified(false);
                }
            }
        }
    }

    FileDialog {
        id: openDialog
        title: qsTr("Open Project or Image")
        fileMode: FileDialog.OpenFile
```

```
        nameFilters: ["Harmoniq Projects (*.json)", "Images
(*.png *.jpg *.jpeg)", "All Files (*)"]
        onAccepted: {
            console.log("OPEN DIALOG ACCEPTED:", selectedFile)

            startScreen.visible = false;
            leftBarLoader.visible = true;
            rightBarLoader.visible = true;
            topBarLoader.visible = true;
            layerLoader.visible = true;

            documentManager.loadFromFile(selectedFile)
        }
    }

    FileDialog {
        id: exportDialog
        title: qsTr("Export Image As")
        fileMode: FileDialog.SaveFile
        nameFilters: ["PNG (*.png)", "JPEG (*.jpg *.jpeg)", "All
Files (*)"]
        onAccepted: {
            if (selectedFile) {
                console.log("Exporting to:", selectedFile);

documentManager.exportImage(selectedFile.toString());
            }
        }
    }

    Popup {
        id: dialogNewImage
        modal: true
        focus: true
        closePolicy: Popup.CloseOnEscape |
Popup.CloseOnPressOutside
        anchors.centerIn: Overlay.overlay

        background: Rectangle {
            color: Themes.currentTheme.background
            radius: 10
        }

        contentItem: Column {
            spacing: 10
            padding: 10
```

```
Label {
    text: qsTr("Width:")
}

TextField {
    id: widthField
    placeholderText: qsTr("Enter width")
    text: "800"
}

Label {
    text: qsTr("Height:")
}

TextField {
    id: heightField
    placeholderText: qsTr("Enter height")
    text: "600"
}

Label {
    text: qsTr("Background Color:")
}

Row {
    spacing: 10
    Button {
        id: selectColor
        text: qsTr("Select color")
        onClicked: {
            colorDialog.open()
        }
    }
    Rectangle {
        id: selectedColor
        width: 50
        height: 50
        color: canvasBackgroundColor
    }
}

Row {
    spacing: 10
    Button {
        text: qsTr("Create")
```

```
                    onClicked: {
                        if (parseInt(widthField.text) > 0 &&
parseInt(heightField.text) > 0) {
                            canvasWidth =
parseInt(widthField.text);
                            canvasHeight =
parseInt(heightField.text);
                            canvasBackgroundColor =
selectedColor.color;

                            startScreen.visible = false;
                            leftBarLoader.visible = true;
                            rightBarLoader.visible = true;
                            topBarLoader.visible = true;
                            layerLoader.visible = true;


documentManager.addDocument("Untitled", "");

documentManager.setCurrentIndex(documentManager.count() - 1);

documentManager.currentLayerManager.addLayer("background",
canvasWidth, canvasHeight, canvasBackgroundColor, false);

                            dialogNewImage.close();
                        } else {
                            console.log("Invalid dimensions");
                        }
                    }
                }
```