

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка онлайн растрового редактору»

Виконав студент групи ПЗ-20-1
ДЕНИСЮК Нестор Геннадійович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
ГРИВЮК Олексій Петрович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ДЕНИСЮК Нестор Геннадійович**

Керівник роботи: **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Розробка онлайн растрового редактору»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **Методи та засоби розробки онлайн растрового редактору для редагування растрової графіки.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Дунєв С.С.	27.02.2024р.	27.02.2024р.
2	Дунєв С.С.	27.02.2024р.	27.02.2024р.
3	Дунєв С.С.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна робота бакалавра складається з комплексної програмної реалізації растрового графічного редактора, а також пояснювальної записки. Пояснювальна записка до дипломної роботи містить 56 сторінок, 26 ілюстрації, 7 таблиць.

Основною метою даної роботи є розробка алгоритмів та реалізація програмного продукту, що надаватиме широкий спектр інструментів для малювання, маніпуляції зображеннями, створення колажів.

В роботі визначено ключові завдання, що пов'язані з розробкою редактора, проведено аналіз існуючого інструментарію, подібного до розроблюваного, а також наведено сценарії роботи програмного продукту. Представлено огляд загальної структури додатку, описано алгоритми взаємодії між окремими компонентами, а також наведено опис об'єктної структури (діаграму класів) додатку.

В рамках випускної роботи проводились експерименти по оптимізації графічних перетворень в результаті яких були визначені оптимальні налаштування для швидкого редагування зображень.

КЛЮЧОВІ СЛОВА: ЗОБРАЖЕННЯ, РАСТРОВИЙ ГРАФІЧНИЙ РЕДАКТОР, КОМП'ЮТЕРНА ГРАФІКА

					ІПЗ.КР.Б – 121 – 24 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Денисюк Н. Г.			Розробка онлайн растрового редактора			Літ.	Арк.	Аркушів	
Керівник		Дунєв С.С.								3	
								Житомирська політехніка, гр. ІПЗ-20-1			
Рецензент		Гривюк О.П.									
Зав. каф.		Вакалюк Т.А.									
					Пояснювальна записка						

ABSTRACT

The bachelor's thesis consists of a complex software implementation of a raster graphics editor, as well as an explanatory note. The explanatory note for the thesis contains 56 pages, 26 illustrations, 7 tables.

The main method of this work is the development of algorithms and the implementation of a software product that provides a wide range of tools for painting, image manipulation, and collage creation.

In the work, the key points that are associated with the development of the editor are identified, an analysis of the existing tools, similar to those previously developed, is carried out, as well as scenarios for the work of the software product are outlined. An overview of the underlying structure of the addendum is presented, algorithms for interaction between adjacent components are described, and a description of the object structure (class diagram) of the addendum is provided.

As part of the final work, experiments were carried out to optimize graphic settings as a result of optimal adjustment for smooth image editing.

KEYWORDS: IMAGE, RASTER GRAPHIC EDITOR, COMPUTER GRAPHICS

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ОНЛАЙН РАСТРОВОГО РЕДАКТОРУ	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту.....	10
1.3 Обґрунтування вибору архітектури онлайн растрового графічного редактора.....	18
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ РАСТРОВОГО ГРАФІЧНОГО РЕДАКТОРА.....	20
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	20
2.2 Проектування та реалізація алгоритмів роботи систем	23
2.3 Реалізація додатку онлайн растрового редактора	25
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН РАСТРОВИМ РЕДАКТОРОМ.....	40
3.1 Порядок встановлення та налаштування параметрів системи	40
3.2 Структура інтерфейсу онлайн растрового редактора	41
3.3 Тестування роботи програмного комплексу.....	50
Висновки до третього розділу.....	52
ВИСНОВКИ	53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
ДОДАТКИ.....	56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (Прикладний програмний інтерфейс)

DOM – Document Object Model (Об'єктна модель документа);

GUI – Graphic User Interface (Графічний інтерфейс користувача);

UML – Unified Modeling Language (Уніфікована мова моделювання);

SDK – Software Development Kit (Набір засобів розробки);

ОС – Операційна система;

ПЗ – Програмне забезпечення;

					ПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми: через зростання веб-технологій багато людей починають використовувати різноманітні мережеві сервіси для розв'язку задач пов'язаних з редагуванням різноманітної графіки. Разом з цим у користувачів зростає запит на створення графіки внаслідок загального зростання доступності та вживаності мережових сервісів. У подібній ситуації онлайн растровий редактор стає зручним, доступним інструментом котрий дає можливість виконувати редагування з будь-якого підключеного до мережі Інтернет пристрою.

С кожним роком все більше людей знаходяться у пошуку більш сучасних рішень для редагування різноманітної растрової графіки, фотографій та інших цілей. Створення мережового сервісу для виконання подібних завдань може забезпечити більш швидку та зручну публікацію своїх робіт. Онлайн растрові редактори дають можливість для редагування зображень без необхідності встановлення складних програм на власному комп'ютері. Швидкий та доступний додаток для редагування растрової графіки є необхідним для різноманітних користувачів. Наразі платформи для онлайн редагування зображень не є розвинутими і не мають великої популярності, вони програють конкуренцію класичним десктопним редакторам. Основними проблемами існуючих мережових систем для редагування графіки є висока навантаженість інтерфейсу та складе знайомство з методикам роботи з додатками.

Хоча сучасні рішення для мережового редагування графіки не є розповсюдженими рішеннями, вони продовжують набирати популярність. Подібні редактори можуть бути доступні для широкого кола користувачів, включаючи як професіоналів у галузі дизайну, так і простих користувачів, які потребують базових інструментів для редагування зображень.

Метою кваліфікаційної роботи є створення високоякісного та універсального інструменту для редагування растрових зображень, який задовольняє потреби користувачів різних профілів та забезпечує їхній успішний та комфортний досвід роботи з графічним контентом у веб-середовищі.

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання кваліфікаційної роботи включає такі етапи:

1. аналіз задач, пов'язаних з користуванням растровим редактором;
2. визначення архітектури, узагальненої структури, ключових компонентів додатку;
3. обґрунтування та вибір засобів реалізації продукту;
4. проектування структурних складових та алгоритмів роботи додатку;
5. впровадження програмної реалізації

Об'єктом дослідження є інтерфейс та функціональність онлайн растрового редактора, а також способи його інтеграції та використання в онлайн середовищі.

Предметом дослідження є технічні аспекти розробки, такі як вибір технологій, архітектура додатку та оптимізація продуктивності.

Методи дослідження: було використано методи та принципи об'єктно-орієнтованого програмування та їх застосування у розробці редактора. З використанням фреймворку Blazor, бібліотеки Fabric.js та JavaScript JSRuntime.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ОНЛАЙН РАСТРОВОГО РЕДАКТОРУ

1.1 Постановка задачі

Основною метою створення додатку «онлайн растровий редактор» є надання користувачам веб інструментів для редагування, створення та обробки растрових зображень. Користувачі мають отримати можливість використовуючи елементи інтерфейсу проводити базові дії з растровими зображеннями, а саме: редагувати растрові зображення, зміни розміру та масштабу, застосування різних варіантів розміщення шарів, додавання та редагування тексту.

Для реалізації поставленого завдання, основними етапами є:

1. Аналіз проекту та збір вимог:

- Проведення аналізу практики використання класичних десктопних растрових редакторів;
- Визначення основних характеристик і функціональних можливостей редактора;
- Аналіз аналогів програмного продукту;

2. Вибір та інтеграція технологій:

- Вибір платформи розробки;
- Інтеграція стандартних інструментів та фреймворків, на яких буде працювати редактор;
- Впровадження розгортання програмного продукту на будь-який браузерний рушій;

3. Розробка та впровадження системи:

- Розробка інтуїтивно зрозумілого та зручного інтерфейсу, який включає різні інструменти малювання, панелі інструментів, меню, полотно для відображення зображення та інші елементи інтерфейсу;
- Реалізація функцій, малювання, обрізання, переміщення, масштабування, зображень.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

- Обробку користувацьких дій, таких як: кліки мишею, введення з клавіатури;
- Реалізація взаємодії з процесом редагування зображення: копіювання, вставка, видалення, перенесення зображень між шарами;
- Реалізація алгоритмів, що виконують дії над параметрами зображення.

4. Тестування та забезпечення якості:

- Проведення ретельного тестування, щоб гарантувати, що програмне рішення працює за призначенням і відповідає вимогам;

Успішна розробка та впровадження такого ресурсу дасть користувачам можливість швидкого та зручного редагування растрових зображень як у особистих, так і у корпоративних цілях.

1.2 Аналіз аналогів програмного продукту

Існує велика кількість різноманітних растрових редакторів для різних платформ, та з різними особливостями функціоналу. Для порівняння будемо використовувати такі растрові редактори як: Adobe Photoshop, MS Paint, GIMP, Photopea, Pixlr.

Adobe Photoshop - є стандартом у сфері растрового редагування і відомий своєю потужною функціональністю та безліччю інструментів, що дозволяють користувачам створювати та редагувати графічні зображення будь-якої складності. Він випускається компанією Adobe Systems і має багаторічну історію розвитку, що призвела до виникнення найбільш різноманітної та продуктивної програми для роботи з графікою (рис. 1.1) [1].

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

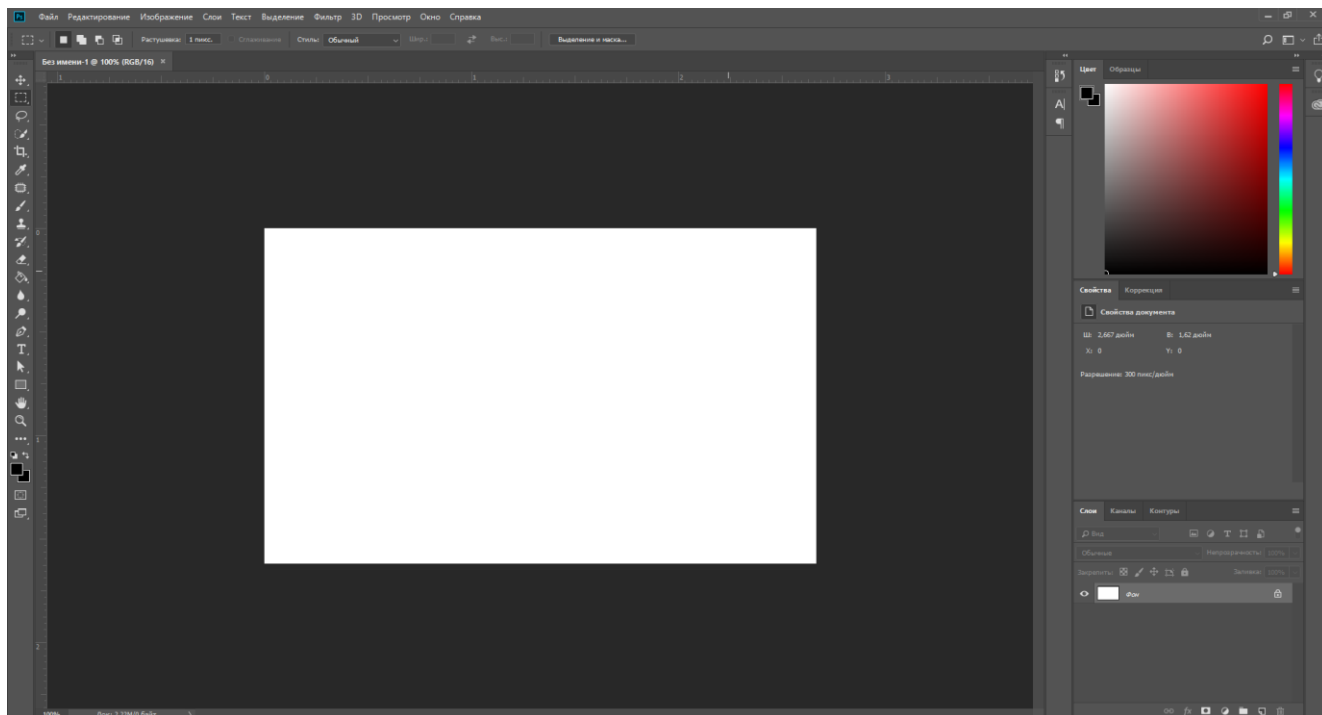


Рис. 1.1 Інтерфейс Adobe Photoshop

Переваги Adobe Photoshop:

- Photoshop має безліч інструментів та функцій для редагування растрових зображень, що дозволяє створювати якісні та професійні графічні проекти.
- Він використовується мільйонами професіоналів у сферах дизайну, фотографії, маркетингу та інших галузях, що свідчить про його високу якість та надійність.
- Adobe постійно випускає оновлення та нові версії Photoshop з покращеннями, новими функціями та виправленнями помилок, що дозволяє користувачам триматися в тренді та мати доступ до нових можливостей.
- Photoshop легко інтегрується з іншими програмами Adobe Creative Cloud, такими як Illustrator, InDesign, Premiere Pro та інші, що спрощує роботу з проектами та обмін ресурсами між програмами.

Недоліки Adobe Photoshop:

- Однією з основних недоліків Photoshop є його висока вартість. Підписка на Adobe Creative Cloud, яка включає Photoshop, може бути дорога для користувачів з обмеженим бюджетом.
- Photoshop вимагає потужного комп'ютера для плавної роботи з великими файлами та складними проектами. Це може стати проблемою для користувачів зі слабкими апаратними характеристиками.
- Для новачків Photoshop може здатися складним та перенасиченим функціями. Вивчення всіх можливостей програми може зайняти час та вимагати додаткових зусиль.

MS Paint – MS Paint, або Microsoft Paint, є одним з найпопулярніших та найпростіших растрових редакторів, випущеним компанією Microsoft. Він включений у стандартну поставку операційних систем Windows і відомий своєю легкістю використання та базовими функціями для створення та редагування графічних зображень (рис. 1.2).

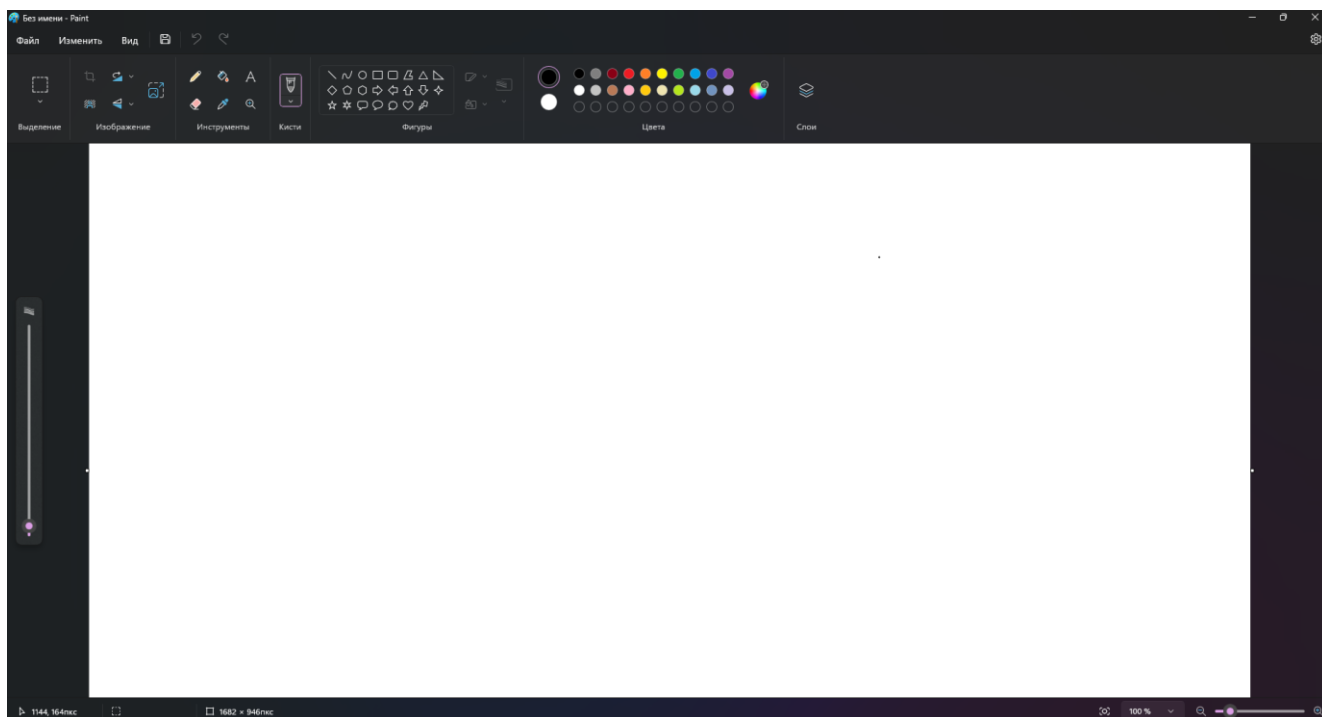


Рис. 1.2 Інтерфейс MS Paint

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

Переваги MS Paint:

- MS Paint має дуже простий та інтуїтивний інтерфейс, що робить його доступним для користувачів з будь-яким рівнем досвіду.
- Оскільки MS Paint включений у стандартну поставку операційних систем Windows, його можна знайти на більшості комп'ютерів під управлінням Windows без потреби встановлення додаткових програм.
- MS Paint доступний безкоштовно для користувачів Windows, що робить його доступним для всіх.

Недоліки MS Paint:

- Порівняно з більш продуктивними та розширеними растровими редакторами, такими як Adobe Photoshop або GIMP, MS Paint має обмежені можливості та функції.
- MS Paint не має таких просунутих функцій, як шари, фільтри або робота з текстом, що робить його менш придатним для складних графічних проєктів.

GIMP - є безкоштовним растровим редактором з відкритим вихідним кодом, розробленим для багатьох платформ, включаючи Windows, macOS та різні дистрибутиви Linux. Він володіє великою популярністю серед користувачів як професіоналів, так і початківців, завдяки своїм розширеним можливостям та безкоштовному доступу (рис. 1.3) [2].

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

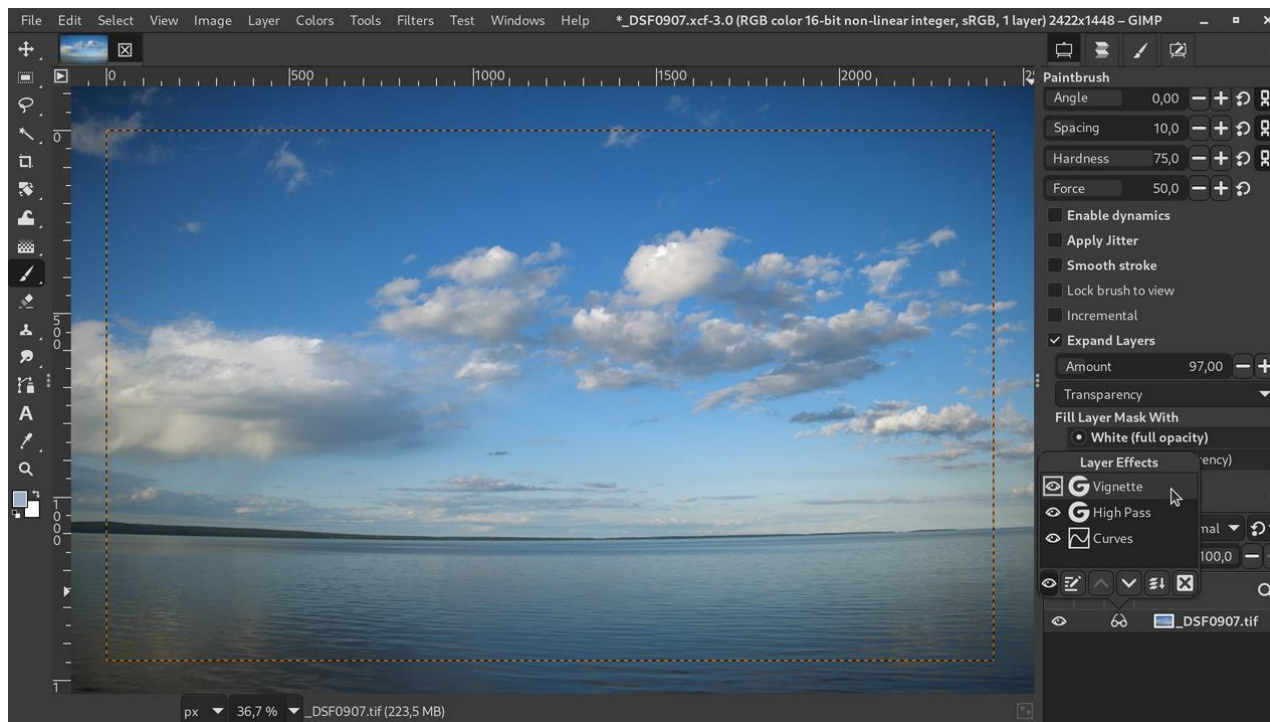


Рис. 1.3 Інтерфейс GIMP

Переваги GIMP:

- GIMP є безкоштовним програмним забезпеченням з відкритим вихідним кодом, що робить його доступним для всіх користувачів без витрат.
- широкий спектр функцій, які конкурують з комерційними редакторами, такими як Photoshop.
- GIMP дозволяє встановлювати різноманітні плагіни, що розширюють його функціональність. Це дає можливість користувачам додавати нові інструменти та функції за їх потребами.
- активна спільнота користувачів та розробників, які постійно вносять вдосконалення та розвиток програми.

Недоліки GIMP:

- Деякі функції GIMP можуть працювати повільно на менш потужних комп'ютерах або застарілих операційних системах.

Загалом, GIMP є потужним та безкоштовним растровим редактором, який може задовольнити потреби як початківців, так і досвідчених користувачів у роботі з графікою.

Photopea - це онлайн-редактор графіки, який надає потужні можливості для редагування растрових та векторних зображень безпосередньо у веб-браузері. Photopea отримав широку популярність завдяки своїй зручності використання, розширеним функціям та безкоштовному доступу (рис. 1.4) [3].

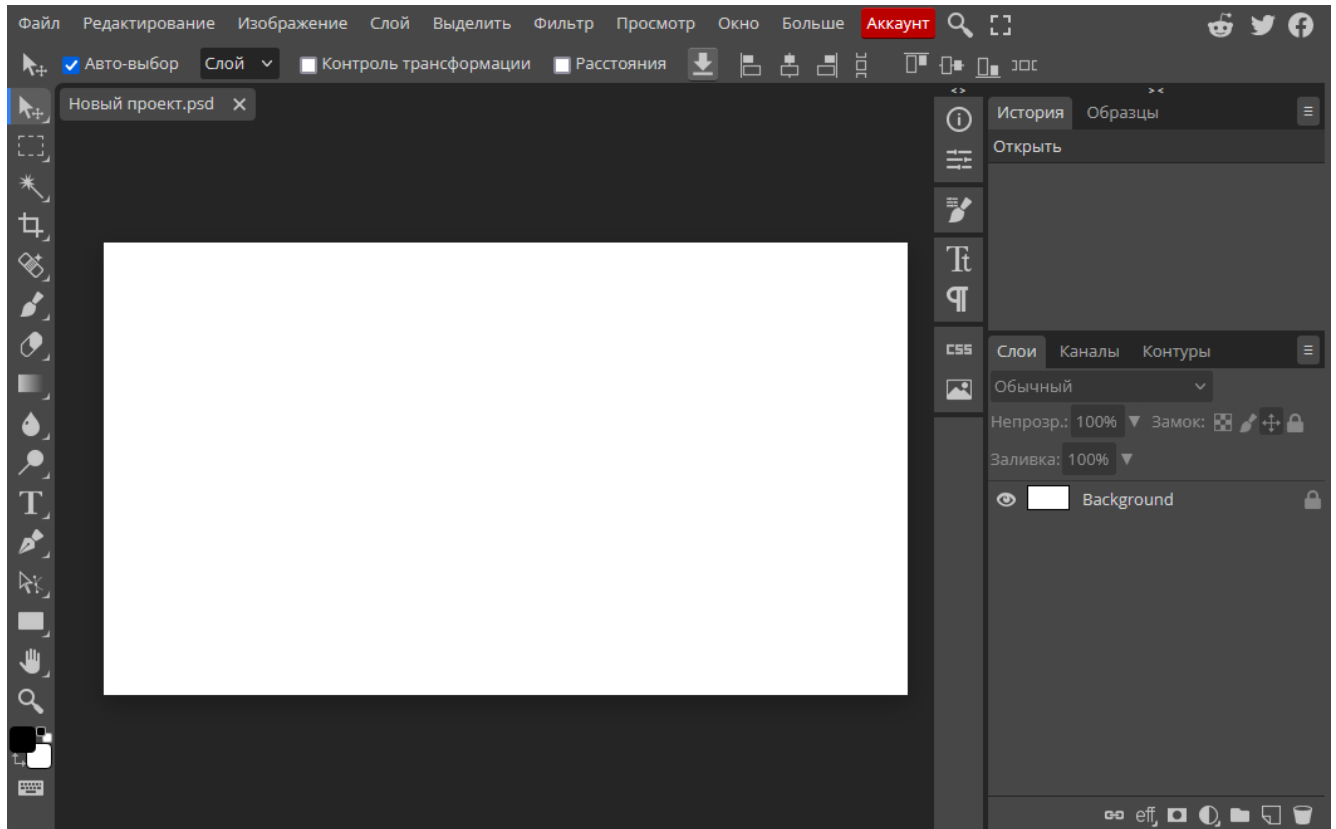


Рис. 1.4 Інтерфейс Photopea

Переваги Photopea:

- Photopea доступний безкоштовно для всіх користувачів, хоча він має можливість підписки на розширену версію з додатковими функціями.
- Інтерфейс Photopea інтуїтивно зрозумілий та зручний у використанні навіть для початківців у редакції графіки.
- Photopea надає потужні можливості для редагування та обробки растрових та векторних зображень, що робить його відмінним інструментом для різних завдань.
- Photopea працює у веб-браузері, що дозволяє користувачам працювати з графікою без необхідності встановлення додаткових програм на свій комп'ютер.

Недоліки Photoprea:

- Залежність від Інтернет-з'єднання: Photoprea потребує Інтернет-з'єднання для роботи, що може бути недоліком у випадках обмеженого доступу до Інтернету.
- У порівнянні з деякими десктопними редакторами графіки, як Adobe Photoshop, може бути менша продуктивність на великих та складних проектах.

Pixlr - це онлайн-редактор графіки, який надає користувачам можливість легко та ефективно редагувати растрові зображення безпосередньо у веб-браузері. Він є популярним серед користувачів завдяки своїм простоті використання, різноманіттю функцій та доступності для широкого кола користувачів (рис. 1.5) [4].

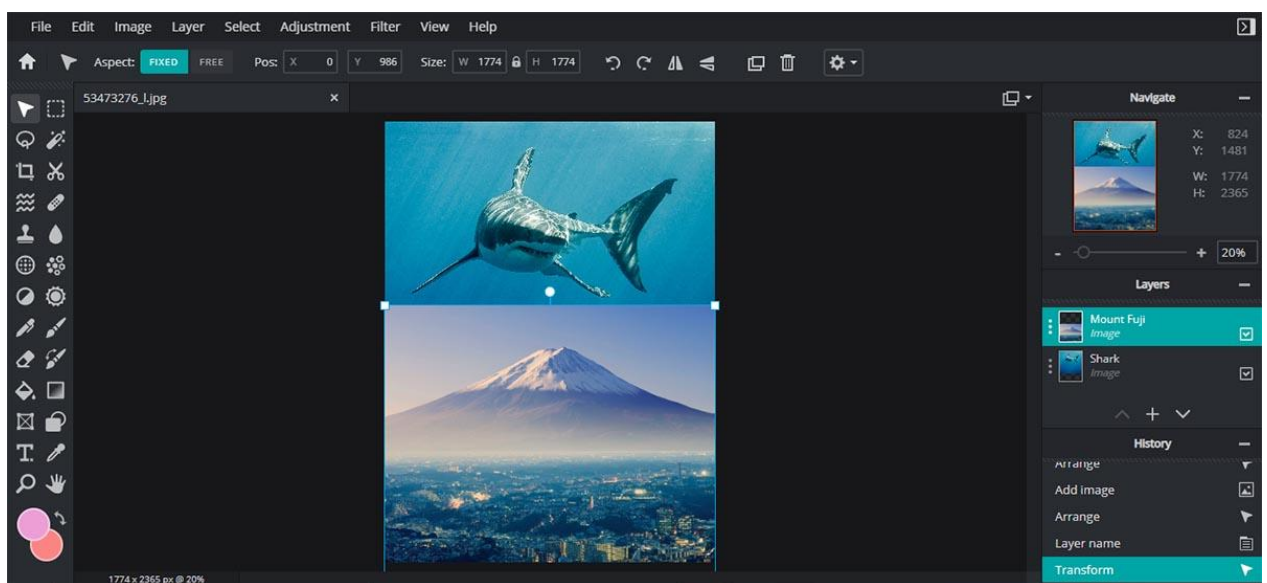


Рис. 1.5 Інтерфейс Pixlr

Переваги Pixlr:

- Pixlr доступний безкоштовно для всіх користувачів, хоча він має можливість підписки на преміум-версію з додатковими функціями.
- Інтерфейс Pixlr інтуїтивно зрозумілий та зручний у використанні навіть для початківців у редакції графіки.
- Pixlr має багато інструментів та функцій для редагування та обробки зображень, що робить його відмінним інструментом для різних завдань.

Недоліки Pixlr:

- Pixlr потребує Інтернет-з'єднання для роботи, що може бути недоліком у випадках обмеженого доступу до Інтернету.

- У порівнянні з деякими десктопними редакторами графіки, може бути менша продуктивність на великих та складних проектах.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця растрових графічних редакторів

Критерії	Adobe Photoshop	MS Paint	GIMP	Photopea	Pixlr
Вартість	Платний	Вбудований у Windows	Безкоштовний (Open Source)	Безкоштовний	Безкоштовний
Функціональність	Широкий спектр функцій	Базовий функціонал	Широкий спектр функцій	Інструменти редагування та обробки фото	Інструменти редагування та обробки фото
Інтерфейс	Професійний, складний для новачків	Простий, інтуїтивно зрозумілий	Застарілий, складний для новачків	Застарілий	Зручний, інтуїтивний
Підтримка форматів	Підтримує широкий спектр форматів, включаючи PSD, JPEG, PNG, TIFF, і багато інших	Обмежена підтримка, основні формати, такі як JPEG та BMP	Підтримує багато форматів, включаючи PSD, JPEG, PNG, GIF, TIFF, і багато інших	Підтримує широкий спектр форматів, включаючи PSD, JPEG, PNG, SVG, і багато інших	Підтримує широкий спектр форматів, включаючи PSD, JPEG, PNG, SVG, і багато інших
Доступність	Наявність мобільних версій для iOS та Android	Немає мобільних версій	Немає мобільних версій	Працює на більшості пристроїв через веб-браузер	Працює на більшості пристроїв через веб-браузер
Підтримка плагінів	Підтримка широкого спектру плагінів для розширення функціональності	Немає підтримки плагінів	Підтримка плагінів для розширення функціональності	Немає підтримки плагінів	Немає підтримки плагінів

Вивчаючи всі аспекти, ми можемо впровадити найкращі функції та можливості MS Paint, Adobe Photoshop та Krita у наш власний продукт. Такий підхід гарантує, що наш продукт не просто задовольняє різні потреби користувачів, але і використовує сильні сторони цих визнаних растрових графічних редакторів. Завдяки цьому, наш продукт гарантує доступ до потужного та зручного

інструменту, який надає широкий спектр можливостей для малювання та маніпуляції зображеннями.

1.3 Обґрунтування вибору архітектури онлайн растрового графічного редактора

Для реалізації функціоналу онлайн растрового графічного редактора на платформі ASP.NET було обрано Blazor. Даваймо розглянемо варіанти вибору мов програмування та інструментальних засобів:

JavaScript: Оскільки програма є браузерною, JavaScript є основною мовою для розробки. Він підтримує взаємодію зі стороною клієнта та може використовуватися для реалізації функціоналу веб-інтерфейсу [5, 6, 7].

Blazor: фреймворк для створення інтерфейсу користувача, який використовує Razor-синтаксис та може взаємодіяти з користувачем у веб-браузері. Компоненти можуть бути використані для різних елементів інтерфейсу, таких як панелі інструментів, області малювання та налаштування [8, 9, 10, 11].

JavaScript Interop: інфраструктура для взаємодії між .NET та JavaScript. Надає можливість викликати JavaScript-функції з .NET-методів та навпаки [12, 13, 14].

Fabric.js: потужна бібліотека для роботи з HTML5-холстом та маніпуляції SVG. Вона надає інтерактивну модель об'єктів поверхні холсту та підтримує роботу з геометричними фігурами, зображеннями, текстом, градієнтами, фільтрами та анімацією. Також дає можливість використовувати парсери SVG-to-canvas та canvas-to-SVG, систему подій та функції серіалізації [15, 16, 17, 18].

SortableJS: бібліотека для створення сортованих списків, у яких можна зручно встановлювати ієрархію елементів [19].

Вимоги до апаратного забезпечення: Для розробки на Blazor та використання Fabric.js необхідний комп'ютер з достатнім обсягом оперативної пам'яті. Важливо також мати сучасний веб-браузер.

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

Висновки до першого розділу

Проведено аналіз предметної області за допомогою якого були визначені основні вимоги до онлайн растрового редактору, основною задачею якої є створення користувачами растрової графіки. В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані та порівняні різні платформи з різними принципами роботи, основним функціоналом яких є редагування растрової графіки. Крім цього, були сформовані основні вимоги для власної платформи. На основі відповідних вимог та критеріїв був сформований набір основних технологій і засобів реалізації для розробки вебдодатку.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ РАСТРОВОГО ГРАФІЧНОГО РЕДАКТОРА

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури

В процесі розробки онлайн растрового редактора важливим аспектом є визначення основних варіантів використання та об'єктно-орієнтованої структури системи. Ці елементи дозволяють краще зрозуміти функціональні вимоги до системи, забезпечити логічну організацію коду та полегшити подальшу підтримку і розширення функціональності редактора.

Редактор надає вебінтерфейс, через взаємодії з яким користувач може створювати та редагувати растрову графіку. Функціональність онлайн растрового редактора залежать від вимог користувачів та типів проектів, які вони реалізують. Ми можемо виділити дві основні категорії проектів:

Прості проекти:

Ці проекти не вимагають збереження у хмарі і можуть бути завершені за один сеанс роботи. Користувачі можуть бути не зацікавлені у збереженні проекту на локальному пристрої.

Проекти, що потребують довготривалої роботи:

Якщо користувач планує працювати над проектом довготривалий час, проект має бути доступним після завершення сеансу роботи з ним. Подібні проекти мають зберігатися у локальному сховищі браузера користувача.

Також важливим фактором для використання редактору є функціонал роботи з растровою графікою.

Для візуалізації взаємодії між користувачем та редактором було створено діаграму використання (рис 2.1).

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис. 2.1 Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.6

Таблиця 2.1

Опис прецеденту «Створення проекту»

Назва варіанту використання	Створення проекту
Короткий опис	Користувач створює новий проект після відкриття додатку. В залежності від потреб користувача, він може продовжити роботу з проектом зі збереженням у локальному сховищі браузера, або взагалі не зберігаючи проект.

Таблиця 2.2

Опис прецеденту «Збереження проекту на локальному пристрої»

Назва варіанту використання	Збереження проекту на локальному пристрої
Короткий опис	Користувача він має можливість завантажити файл свого проекту, або ж зображення яке являє собою результат роботи користувача.

Таблиця 2.3

Опис прецеденту «Створення або завантаження растрових зображень»

Назва варіанту використання	Створення або завантаження растрових зображень
Короткий опис	Користувач має можливість додавати зображення до робочого простору напряду з локального пристрою, або з буферу обміну.

Таблиця 2.4

Опис прецеденту «Редагування, корекція, ретушування растрової графіки»

Назва варіанту використання	Редагування, корекція, ретушування растрової графіки
Короткий опис	Користувач редагує растрові зображення за допомогою наданих інструментів редагування

Опис прецеденту «Простий проект»

Назва варіанту використання	Простий проект
Короткий опис	Використовується якщо користувач має намір простого та швидкого редагування растрової графіки.

Опис прецеденту «Проект на локальному пристрої»

Назва варіанту використання	Проект на локальному пристрої
Короткий опис	Для довготривалої роботи над проектом користувач має можливість зберегти проект у локальному сховищі, а саме у IndexedDb

2.2 Проектування та реалізація алгоритмів роботи систем

Додаток реалізовує основні функції для редагування растрової графіки. Алгоритм роботи з додатком зображено на рисунку 2.2.

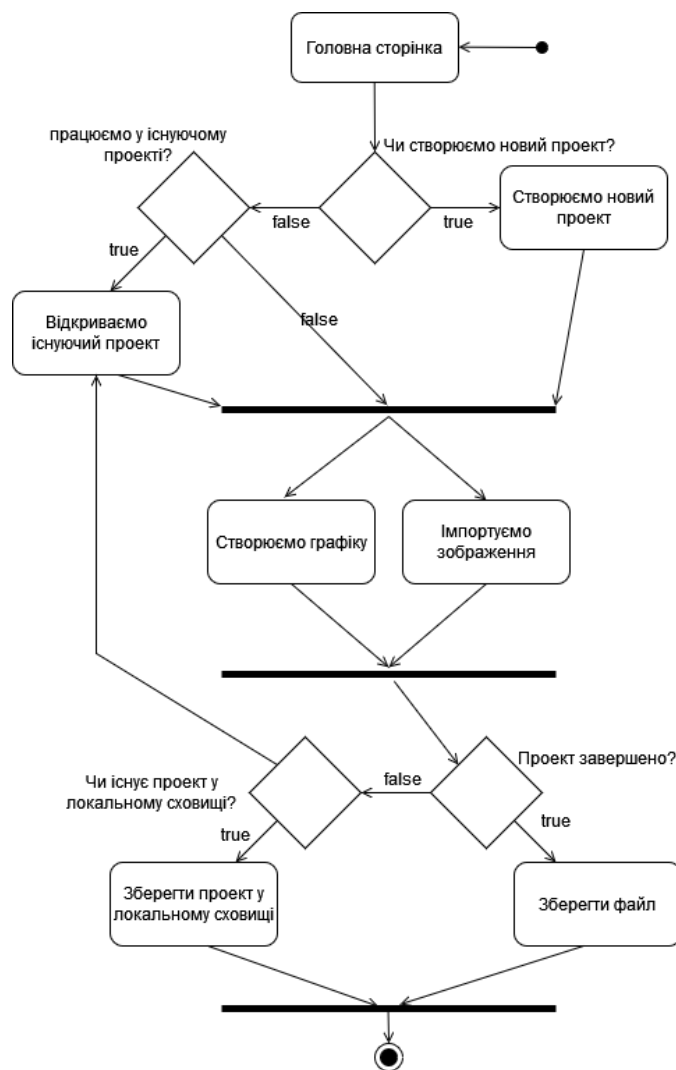


Рис. 2.2 Діаграма активності

Додаток має такі основні елементи та функції як:

Реалізація області малювання:

- Функції вибору інструментів малювання
- Масштабування та переміщення холсту у робочому просторі
- Збереження та завантаження проекту до локального сховища
- Структуризація шарів відповідно заданої користувачем структури

Робота з зовнішніми файлами:

- Завантаження та вставки зображення
- Збереження зображення

Редагування зображення:

- Налаштування параметрів функцій редагування зображень

Редагування системи шарів:

- Редагування ієрархії шарів
- Передача ієрархії шарів до сховища проекту

2.3 Реалізація додатку онлайн растрового редактора

Додаток являє собою систему обміну інформацією між вікнами, котрі утримують у собі інтерфейс для взаємодії з різними функціями потрібними для редагування проекту.

Розглянемо для початку функцію котра формує область редагування у відповідному для неї вікні:

Глобальні змінні робочої області:

```
var viewport = document.getElementById('viewport');

var canvasContainer = document.getElementById('canvasContainer');
var canvasWidth = 600;
var canvasHeight = 500;

var data;
data = localStorage.getItem('layers');
if (!data) {
    data = JSON.stringify({
        "selected": "canvas1",
        "layers": [
            { "id": "canvas1", "name": "Layer 1" }
        ]
    });
}

//змінні масштабування та переміщення
var scale = 1.0; // Початковий масштаб
var minscale = 0.1;
var maxscale = 5.0;
const zoomSpeed = 0.1; // Швидкість масштабування

var canvasPosX = - canvasWidth / 2, canvasPosY = - canvasHeight / 2;
var oldPosX = - canvasWidth / 2, oldPosY = - canvasHeight / 2;

var isDragging = false;
var startX, startY;

var painting = false;

canvasTransform.translate(oldPosX, oldPosY);

//Список шарів, та активний шар
```

```
var canvases = [];
var activeCanvas;
var canvasData = {};
```

Вводяться змінні котрі відповідають за позицію та масштаб холсту, отримуються дані про шари. Для зміни позиція та масштабу холсту використовується метод refactor, огорнутий у методи translate та scale, класу canvasTransform.

```
class canvasTransform {
    static transform = '';
    static #translate = '';
    static #scale = '';
    static translate(valueX, valueY) {
        this.#translate = `translate(${valueX}px,
            ${valueY}px)`;
        this.refactor();
    }
    static scale(value) {
        this.#scale = `scale(${value})`;
        this.refactor();
    }
    static refactor() {
        this.transform = this.#translate + '\n' + this.#scale;
        canvasContainer.style.transform = this.transform;
    }
}
```

Метод scale присвоюється події прокрутки колеса миші, завдяки чому відбувається мастшабування холсту. Також метод translate присвоюється події переміщення миші, під час виконання події перевіряється чи натиснута середня кнопка миші, якщо це так, метод виконується.

Лістинг:

```
viewport.addEventListener('mousedown', function (e) {
    switch (e.button) {
        case 1:
            isDragging = true;
            startX = e.clientX;
            startY = e.clientY;
            oldPosX = canvasPosX;
            oldPosY = canvasPosY;
            break;
    }
}); //MouseDown

viewport.addEventListener('mousemove', function (e) {
    if (isDragging) {
        var currentX = e.clientX;
        var currentY = e.clientY;
        canvasPosX = oldPosX + currentX - startX;
        canvasPosY = oldPosY + currentY - startY;
        canvasTransform.translate(canvasPosX, canvasPosY);
    }
}); //MouseMove
```

```

viewport.addEventListener('wheel', function (e) {
    e.preventDefault();
    var x = e.deltaX;
    var y = e.deltaY;

    if (y > 0) {
        if (scale - zoomSpeed > minscale) {
            scale = scale - zoomSpeed;
            canvasTransform.scale(scale);
        }
    }
    else {
        if (scale + zoomSpeed < maxscale) {
            scale = scale + zoomSpeed;
            canvasTransform.scale(scale);
        }
    }
}); //Wheel

```

Для вибору інструментів використовується функція selectTool(). Ця функція відповідає за вибір інструменту в редакторі графіки. Перед виконанням перевіряє, чи існує активний холст (activeCanvas).

Лістинг:

```

function selectTool(tool) {
    if (!activeCanvas) return;

    window.removeEventListener('storage', readOptionsEvent);
    readOptionsEvent = (e) => {
        onLocalStorageChange(e, 'options', () => {
            localStorage.setItem(tool,
                localStorage.getItem('options'));
            selectTool(tool);
        });
    };
    window.addEventListener('storage', readOptionsEvent);
}

```

Функція очищає минулі обробники подій холсту (activeCanvas) та налаштовує нові для вибору об'єктів, створення об'єктів та оновлення параметрів. Видаляє попередні обробники подій зі сховища (localStorage) та додає новий обробник для зчитування параметрів інструменту. Вибирає інструмент в залежності від переданого параметра tool.

```

clearCanvasEvents();
const storage = localStorage.getItem(tool); //Для перевірки наявності

activeCanvas.on('selection:created', (e) => objectSelection(e));
activeCanvas.on('selection:updated', (e) => objectSelection(e));
activeCanvas.on('selection:cleared', (e) => setOptions());

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Для кожного інструменту встановлює відповідний режим роботи холсту та параметри малювання (колір, ширина, прозорість). Якщо вибраним інструментом є “селектор” ('select'), функція вимикає будь-які інструменти малювання та не передає ніяких даних про свої значення до localStorage.

```
//Вибір інструменту
if (tool === 'select') {
    localStorage.removeItem('options');
    localStorage.removeItem('select');
    activeCanvas.isDrawingMode = false; // Вимикаємо малювання
    activeCanvas.selection = true; // Вмикаємо вибір об'єктів
    activeCanvas.forEachObject(function (obj) {
        obj.selectable = true;
        obj.evented = true;
    });
}
```

Якщо вибраним інструментом є “кисть” ('brush'), функція вмикає інструмент малювання кистю та передає дані про параметри кисті до localStorage.

```
if (tool === 'brush') {
    activeCanvas.isDrawingMode = true;
    const brush = new fabric.PencilBrush(activeCanvas);
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({
            "color": { "type": "color", "value": "#0000FF" },
            "width": { "type": "float", "value": 10 },
            "opacity": { "type": "float", "value": 0.8 }
        }));
    }; //Стандартні значення

    const values = JSON.parse(storage);
    brush.color = hexToRgba(values["color"]["value"], values["opacity"]["value"]);
    brush.width = parseFloat(values["width"]["value"]);
    activeCanvas.freeDrawingBrush = brush;
}
```

Якщо вибраним інструментом є “ластик” ('eraser'), функція вмикає інструмент малювання ластиком та передає дані про параметри ластика до localStorage.

```
if (tool === 'eraser') {
    activeCanvas.isDrawingMode = true;
    const eraser = new fabric.EraserBrush(activeCanvas);
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({
            "width": { "type": "float", "value": 15 }
        }));
    }; //Стандартні значення

    const values = JSON.parse(storage);
    eraser.width = parseFloat(values["width"]["value"]);
    activeCanvas.freeDrawingBrush = eraser;
}
```

Якщо вибраним інструментом є “прямокутник” ('rect'), функція створює прямокутник та передає його до холста. Також вона передає дані про параметри прямокутника до localStorage. Під час створення прямокутника функція створює обробники подій натискання мишею та переміщення миші, ці події передають інформацію про позицію миші, з цих даних функція створює позицію та розміри прямокутника.

```

if (tool === 'rect') {
  if (!storage) {
    localStorage.setItem(tool, JSON.stringify({
      "color": { "type": "color", "value": "#0000FF" },
      "opacity": { "type": "float", "value": 0.8 }
    }));
  }; //Стандартні значення

  const values = JSON.parse(storage);
  activeCanvas.on('mouse:down', function (opt) {
    isDrawing = true;
    const pointer = activeCanvas.getPointer(opt.e);
    const origX = pointer.x;
    const origY = pointer.y;
    drawingShape = new fabric.Rect({
      left: origX,
      top: origY,
      fill: values["color"]["value"],
      opacity: values["opacity"]["value"],
      width: 1,
      height: 1
    });
    activeCanvas.add(drawingShape);

    activeCanvas.on('mouse:move', function (opt) {
      if (!isDrawing) return;
      const pointer = activeCanvas.getPointer(opt.e);
      var width = pointer.x - origX;
      var height = pointer.y - origY;

      if (width < 0) {
        drawingShape.set({ left: pointer.x });
        width = Math.abs(width);
      }
      if (height < 0) {
        drawingShape.set({ top: pointer.y });
        height = Math.abs(height);
      }
      drawingShape.set({ width: width, height: height });
      activeCanvas.renderAll();
    });

    activeCanvas.on('mouse:up', function (opt) {
      isDrawing = false;
      drawingShape.setCoords();
    });
  });
}

```

Якщо вибраним інструментом є “коло” ('circle'), функція створює коло та передає це коло до холста. Також вона передає дані про параметри кола до localStorage. Під час створення кола функція створює обробники подій натискання мишею та переміщення миші, ці події передають інформацію про позицію миші, з цих даних функція створює позицію та радіус кола.

```

if (tool === 'circle') {
  if (!storage) {
    localStorage.setItem(tool, JSON.stringify({
      "color": { "type": "color", "value": "#0000FF" },
      "opacity": { "type": "float", "value": 0.8 }
    }));
  }; //Стандартні значення

  const values = JSON.parse(storage);
  activeCanvas.on('mouse:down', function (opt) {
    isDrawing = true;
    const pointer = activeCanvas.getPointer(opt.e);
    drawingShape = new fabric.Circle({
      left: pointer.x,
      top: pointer.y,
      fill: values["color"]["value"],
      opacity: values["opacity"]["value"],
      radius: 1,
      originX: 'center',
      originY: 'center'
    });
    activeCanvas.add(drawingShape);
  });

  activeCanvas.on('mouse:move', function (opt) {
    if (!isDrawing) return;
    const pointer = activeCanvas.getPointer(opt.e);
    const radius = Math.sqrt(Math.pow(pointer.x - drawingShape.left, 2) +
Math.pow(pointer.y - drawingShape.top, 2));
    drawingShape.set({ radius: radius });
    activeCanvas.renderAll();
  });

  activeCanvas.on('mouse:up', function (opt) {
    isDrawing = false;
    drawingShape.setCoords();
  });
}

```

Якщо вибраним інструментом є “текст” ('text'), функція створює об'єкт текст та передає це об'єкт до холста. Також вона передає дані про параметри тексту до localStorage.

```

if (tool === 'text') {
  if (!storage) {
    localStorage.setItem(tool, JSON.stringify({
      "color": { "type": "color", "value": "#000000" },
      "font size": { "type": "float", "value": 50 },
      "font weight": { "type": "float", "value": 'normal' },
      "font family": { "type": "float", "value": 'Times New Roman' },

```

```

        "opacity": { "type": "float", "value": 0.8 }
    }));
}; //Стандартні значення

const values = JSON.parse(storage);
function createTextObject(x, y) {
    const text = new fabric.IText('', {
        left: x,
        top: y,
        fill: values["color"]["value"],
        fontSize: values["font size"]["value"],
        fontWeight: values["font weight"]["value"],
        fontFamily: values["font family"]["value"],
        opacity: values["opacity"]["value"],
        selectable: true,
        editable: true
    });

    activeCanvas.add(text).setActiveObject(text);
    text.enterEditing();
    activeCanvas.renderAll();
}

activeCanvas.on('mouse:down', function (opt) {
    const pointer = activeCanvas.getPointer(opt.e);
    createTextObject(pointer.x, pointer.y);
});

function clearCanvasEvents() {
    activeCanvas.isDrawingMode = false;
    activeCanvas.off('mouse:down');
    activeCanvas.off('mouse:move');
    activeCanvas.off('mouse:up');
    activeCanvas.off('selection:created');
    activeCanvas.off('selection:updated');
    activeCanvas.off('selection:cleared');
    activeCanvas.selection = false;
    activeCanvas.forEachObject(function (obj) {
        obj.selectable = false;
        obj.evented = false;
    });
}

```

Для збереження та завантаження проекту до локального сховища використовуються функції `saveCanvasDataToProject()` та `loadCanvasDataFromProject()`. Вони отримують доступ до локальної бази даних за допомогою функції `getProjectById()`, та передають у відповідне сховище проекту серіалізовані дані про шари та їх наповнення.

Лістинг:

```

function saveCanvasDataToProject(projectId) {
    saveCanvasData(); // Збереження структури canvasData

    console.log(getProjectById(projectId));
    getProjectById(projectId).then(project => {

```

```

        project.data.canvasData = canvasData; // Оновлення даних у IndexedDB
        return updateProject(project);
    });
}

function loadCanvasDataFromProject(projectId) {
    return getProjectById(projectId).then(project => {
        canvasData = project.data.canvasData || {}; // Завантаження даних canvas
        з IndexedDB
        loadCanvasData(); // Завантаження даних на canvas
    });
}

```

Для збереження структури шарів відповідно до структури створеної користувачем використовується функція `extractLayers()`. Функція призначена для витягування ієрархії шарів (`layers`) з деревоподібної структури елементів.

Вона рекурсивно обходить всі дочірні елементи `element` та визначає, чи вони є шарами чи папками. Якщо елемент має клас `'layer'`, він вважається шаром. Функція отримує ідентифікатор шару (або генерує новий), назву та додає цей шар до списку `layers`. Якщо елемент має клас `'folder'`, він вважається папкою. Функція створює об'єкт папки з назвою та рекурсивно викликає себе для обробки вкладених шарів. Параметр `element` - це кореневий елемент дерева шарів.

Функція повертає об'єкт зі списком шарів та ідентифікатором обраного шару.

Лістинг:

```

function extractLayers(element) {
    var layers = [];

    function traverse(element) {
        const children = Array.from(element.children);

        children.forEach(child => {
            if (child.classList.contains('layer')) {
                var id = layerIdMap.get(child);
                if (!id) {
                    id = generateLayerId();
                    layerIdMap.set(child, id);
                }
                const name = child.querySelector('input[id="title"]').value;
                layers.push({ id, name });

                if (child.classList.contains('selected')) {
                    selectedLayer = id;
                }
            } else if (child.classList.contains('folder')) {
                const folder = {
                    name: child.querySelector('input[id="title"]').value,
                    items: []
                };

                const innerExplorer = child.querySelector('#explorer');
                if (innerExplorer) {
                    traverse(innerExplorer);
                }
            }
        });
    }

    traverse(element);
    return layers;
}

```

```

        const subLayers = extractLayers(innerExplorer);
        folder.items = subLayers.layers;
        if (subLayers.selected && !selectedLayer) {
            selectedLayer = subLayers.selected;
        }
    }

    layers.push(folder);
}
});
}

traverse(element);

return { selected: selectedLayer, layers };
}

```

Для завантаження зображень використовуються два обробники подій. Перший обробник подій, обробляє подію копіювання у робочій області. При вставці даних з буфера обміну перевіряється, чи є серед них зображення. Якщо так, створюється об'єкт Image, який завантажується з даних, отриманих з буфера обміну. Завантажене зображення додається на активний холст (activeCanvas) у центрі. Після додавання зображення холст перерисовується.

Лістинг:

```

viewport.addEventListener('paste', function () {
    const items = event.clipboardData.items;
    for (let i = 0; i < items.length; i++) {
        if (items[i].type.indexOf('image') !== -1) {
            const blob = items[i].getAsFile();
            const reader = new FileReader();
            reader.onload = function (event) {
                const img = new Image();
                img.src = event.target.result;
                img.onload = function () {
                    const image = new fabric.Image(img);
                    image.set({
                        left: activeCanvas.width / 2,
                        top: activeCanvas.height / 2,
                        originX: 'center',
                        originY: 'center'
                    });
                    activeCanvas.add(image);
                    activeCanvas.renderAll();
                };
            };
            reader.readAsDataURL(blob);
        }
    }
});

```

Другий обробник подій, обробляє подію кидання зображення у робочу область. Вона перевіряє, чи файли є зображеннями. Для кожного зображення

створюється об'єкт Image, який завантажується з файлу. Завантажене зображення масштабується, якщо його розмір перевищує розміри холста. Зображення додається на активний холст у центрі та перерисовується.

Лістинг:

```
viewport.addEventListener('drop', function (event) {
  event.preventDefault();
  console.log('fileAdded');
  const files = event.dataTransfer.files;
  for (let i = 0; i < files.length; i++) {
    const file = files[i];
    if (file.type.indexOf('image') !== -1) {
      const reader = new FileReader();
      reader.onload = function (e) {
        const img = new Image();
        img.src = e.target.result;
        img.onload = function () {
          const image = new fabric.Image(img);

          // Масштабуємо зображення, якщо його розмір перевищує
размери холста

          const scaleX = canvasWidth / img.width;
          const scaleY = canvasHeight / img.height;
          const scale = Math.min(scaleX, scaleY, 1); // Ограничуємо
масштаб до 1 (100%)

          image.set({
            left: activeCanvas.width / 2,
            top: activeCanvas.height / 2,
            originX: 'center',
            originY: 'center',
            scaleX: scale,
            scaleY: scale
          });
          activeCanvas.add(image);
          activeCanvas.renderAll();
        };
      };
      reader.readAsDataURL(file);
    }
  }
});
```

Для збереження кінцевого зображення на локальному пристрої використовується функція saveCombinedImage(). функція призначена для збереження об'єднаного зображення, яке складається з декількох холстів. Спочатку створюється новий елемент <canvas> з контекстом 2D. Розміри цього об'єднаного холсту встановлюються такі ж, як розміри вхідних холстів (canvasWidth та canvasHeight). Далі для кожного холсту зі списку canvases виконується наступне: Встановлюється глобальна прозорість (значення 1.0). Зображення нижнього холсту (canvas.lowerCanvasEl) копіюється на об'єднаний холст. Після об'єднання всіх

зображень, отримується URL зображення у форматі PNG. Створюється посилання (<a>), яке має URL зображення та ім'я файлу 'image.png'. Викликається дія “зберегти” для цього посилання.

Лістинг:

```
function saveCombinedImage() {
  const mergedCanvas = document.createElement('canvas');
  const ctx = mergedCanvas.getContext('2d');

  mergedCanvas.width = canvasWidth;
  mergedCanvas.height = canvasHeight;

  canvases.forEach(canvas => {
    ctx.globalAlpha = 1.0; // Adjust transparency as needed
    ctx.drawImage(canvas.lowerCanvasEl, 0, 0);
  });

  const image = mergedCanvas.toDataURL('image/png');
  const link = document.createElement('a');
  link.href = image;
  link.download = 'image.png';
  link.click();
}
```

Для оновлення ієрархії шарів використовуються такі функції addFolder(), addLayer(), deleteSelected(), clearSelection(). Функція addFolder() відповідає за додавання нової папки до дерева об'єктів. Створюється елемент <div> для папки та іконка (іконка папки). При натисканні на іконку папки відбувається зміна стану (розкриття або згортання) папки. Додається поле введення для назви папки. Папка може містити вкладені об'єкти (шари або підпапки). Всі об'єкти сортується за допомогою бібліотеки SortableJs.

Лістинг:

```
function addFolder() {
  const selectedFolder = document.querySelector('.selected'); // Поточна папка
  const explorer = document.getElementById('explorer');
  const folderDiv = document.createElement('div');
  const folderExplorer = document.createElement('div');

  folderDiv.classList.add('folder');

  const icon = document.createElement('i'); //Створення іконки
  icon.classList.add('bi', 'bi-folder-fill');
  icon.onclick = function () { //Бінд колапсу та розкриття при натисканні на іконку
    if (folderExplorer.classList.contains('collapsed')) {
      icon.classList.remove('bi-folder');
      icon.classList.add('bi-folder-fill');

      folderExplorer.classList.remove('collapsed');
      folderExplorer.classList.add('expanded');
    } else {
      icon.classList.remove('bi-folder-fill');
    }
  }
}
```

```

        icon.classList.add('bi-folder');

        folderExplorer.classList.remove('expanded');
        folderExplorer.classList.add('collapsed');
    }

    var title = document.createElement('input'); //Назва папки
    title.type = "text";
    title.id = "title";
    title.classList = "title";
    title.value = getNextFolderName();
    title.onfocus = function () { renameElement(this); };

    folderExplorer.classList.add('expanded');
    folderDiv.appendChild(icon);
    folderDiv.appendChild(title);

    folderDiv.onclick = function () {
        event.stopPropagation();
        clearSelection();
        this.classList.add('selected');
    };
    folderExplorer.classList.add('explorer');
    folderExplorer.id = 'explorer';

    Sortable.create(folderExplorer, {
        group: 'nested',
        animation: 150,
        fallbackOnBody: true,
        swapThreshold: 0.65
    });
    folderDiv.appendChild(folderExplorer);

    if (selectedFolder != null && selectedFolder.className == "folder selected") {
        selectedFolder.querySelector('#explorer').appendChild(folderDiv);
    } else {
        explorer.appendChild(folderDiv);
    }

    new Sortable(explorer, {
        group: 'nested',
        animation: 150,
        fallbackOnBody: true,
        swapThreshold: 0.65
    });
}

```

Функція addLayer() відповідає за додавання нового шару до дерева об'єктів. Створюється елемент <div> для шару з іконкою (іконка файлу). Додається поле введення для назви шару. Шар може бути вкладений в папку або бути на одному рівні з папками. Всі об'єкти сортується за допомогою бібліотеки SortableJs.

Лістинг:

```

function addLayer() {
    const selectedFolder = document.querySelector('.selected');
    const explorer = document.getElementById('explorer');

    const layerDiv = document.createElement('div');
    layerDiv.classList.add('layer');
}

```

```

layerDiv.innerHTML = '<i class="bi bi-file-earmark"></i>';
layerDiv.onclick = function (event) {
    event.stopPropagation();
    clearSelection();
    this.classList.add('selected');
};

var title = document.createElement('input'); //Назва шару
title.type = "text";
title.id = "title";
title.classList = "title";
title.value = getNextLayerName();
title.onfocus = function () { renameElement(this); };

layerDiv.appendChild(title);

if (selectedFolder != null && selectedFolder.className == "folder selected") {
    selectedFolder.querySelector('#explorer').appendChild(layerDiv);
} else {
    explorer.appendChild(layerDiv);
}

new Sortable(explorer, {
    group: 'nested',
    animation: 150,
    fallbackOnBody: true,
    swapThreshold: 0.65
});
}

```

Функція deleteSelected() видаляє об'єкт, який був вибраний користувачем (має клас 'selected'). Функція clearSelection() скасовує вибір об'єкта, який був вибраний користувачем (має клас 'selected').

Лістинг:

```

function deleteSelected() {
    const selected = document.querySelector('.selected');
    selected.remove();
}

function clearSelection() {
    const selectedFolder = document.querySelector('.selected');
    if (selectedFolder) {
        selectedFolder.classList.remove('selected');
    }
}

```

Функція setStructure() відповідає за збереження структури об'єктів (шарів та папок) у локальному сховищі (localStorage). Спочатку отримуємо посилання на елемент з ідентифікатором 'explorer' (це, очевидно, дерево об'єктів). Викликаємо функцію extractLayers(), яка витягує ієрархію шарів та папок з цього елемента. Отриманий результат (структура об'єктів) виводимо у консоль для перевірки. Зберігаємо цю структуру у локальному сховищі під ключем 'layers'.

Лістинг:

```
function setStructure() {  
    const explorerElement = document.getElementById('explorer');  
    console.log(explorerElement);  
    const result = extractLayers(explorerElement);  
    console.log(result);  
    console.log(JSON.stringify(result, null, 2));  
  
    localStorage.setItem('layers', JSON.stringify(result));  
}
```

Функція loadLayersData() завантажує дані про шари для проекту за його ідентифікатором (projectId). Використовується функція getProjectById(), яка отримує дані про проект. Після отримання даних створюється дерево об'єктів (шарів та папок) на основі елемента з ідентифікатором 'explorer'. Виводиться інформація про обрані шари та їх параметри у консоль.

Лістинг:

```
function loadLayersData(projectId) {  
    getProjectById(projectId)  
        .then(project => {  
            const rootElement = document.getElementById('explorer');  
            console.log('Got');  
            console.log(project.data.selected);  
            console.log(project.data.layers);  
            createLayers(rootElement, { project.data.selected, project.data.layers  
});  
        })  
        .catch(error => {  
            console.error('Помилка:', error);  
        });  
}
```

Подія котра відбувається при зміні даних в області зберігання (localStorage або sessionStorage) в контексті іншого документа, викликає цю функцію. При зміні даних в сховищі викликається ця подія, і ви можете відслідковувати зміни. Відбувається перевірка, чи відбулася зміна проекту (змінна 'changeProject' у sessionStorage). Якщо так, отримується ідентифікатор поточного проекту (currentProjectId) з localStorage. Викликається функція loadLayersData(currentProjectId), яка завантажує дані про шари для цього проекту.

Лістинг:

```
// Завантаження проекту при його зміні  
window.addEventListener('storage', (e) => {  
    if (e.storageArea === sessionStorage && e.key === 'changeProject') {  
        if (sessionStorage.getItem('changeProject')) {  
            currentProjectId = parseInt(localStorage.getItem('currentProjectId'));  
            if (currentProjectId) {  
                loadLayersData(currentProjectId);  
            }  
        }  
    }  
});
```

```

        sessionStorage.setItem('changeProject', false);
    }
});

```

Висновки до другого розділу

В процесі проектування онлайн растрового редактору проаналізовано варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована структура платформи. Було описано алгоритм роботи з додатком та описані послідовні пункти варіантів його використання. Також був наведений опис основного функціоналу додатку. Було описано реалізацію основних алгоритмів функціонування програмного додатку.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН РАСТРОВИМ РЕДАКТОРОМ

3.1 Порядок встановлення та налаштування параметрів системи

Встановлення та налаштування платформи вимагає розгортання всіх його компонентів. Кожен компонент повинен бути належним чином інтегрований в загальну систему, щоб забезпечити оптимальну взаємодію та функціонування. Для детального опису архітектури платформи було створено діаграму розгортання (рис. 3.1). Ця діаграма наочно ілюструє розташування та зв'язки між різними компонентами системи, включаючи сервери, бази даних, клієнтські додатки та інші важливі елементи. Вона допомагає зрозуміти, як компоненти платформи взаємодіють між собою та з зовнішніми системами, забезпечуючи цілісність і стабільність роботи всієї платформи.

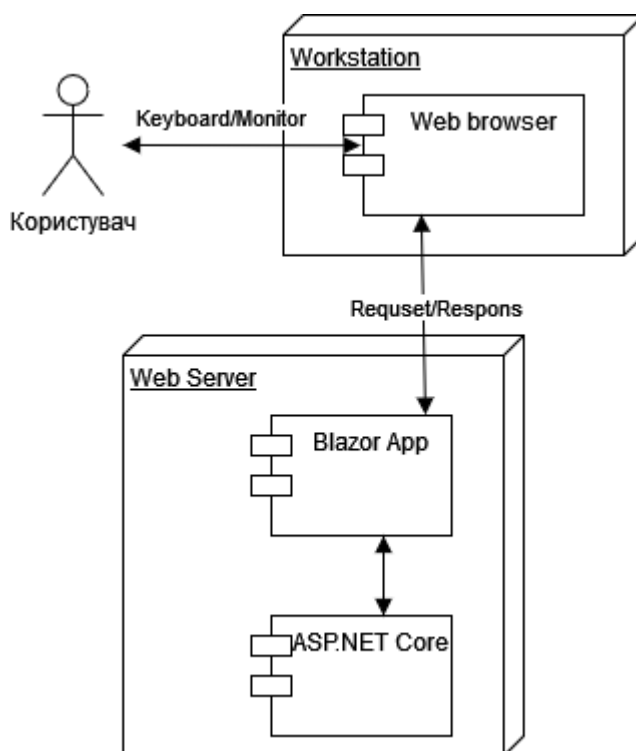


Рис. 3.1 Діаграма розгортання

Опис процесу розгортання:

Використання серверної моделі розгортання Blazor відбувається на сервері з підтримкою ASP.NET Core, що надає численні переваги та можливості для розробки сучасних веб-додатків. У цій моделі всі компоненти додатку виконуються на сервері, що дозволяє використовувати повну потужність серверного обладнання для обробки даних та виконання логіки програми. Це також означає, що клієнтська частина додатку залишається легкою, оскільки основна обчислювальна робота виконується на сервері.

Оновлення елементів користувацького інтерфейсу, обробка подій та виклики JavaScript здійснюються через підключення SignalR. SignalR – це бібліотека для ASP.NET, яка спрощує процес додавання функціоналу в реальному часі до веб-додатків. Вона забезпечує двостороннє спілкування між сервером і клієнтом, дозволяючи миттєво передавати дані та оновлювати інтерфейс без необхідності перезавантаження сторінки.

3.2 Структура інтерфейсу онлайн растрового редактора

Для використання онлайн растрового редактору необхідно перейти на веб-сайт <https://ipz-dng-graduate.somee.com/>. Головна сторінка має такий вигляд (рис. 3.2).

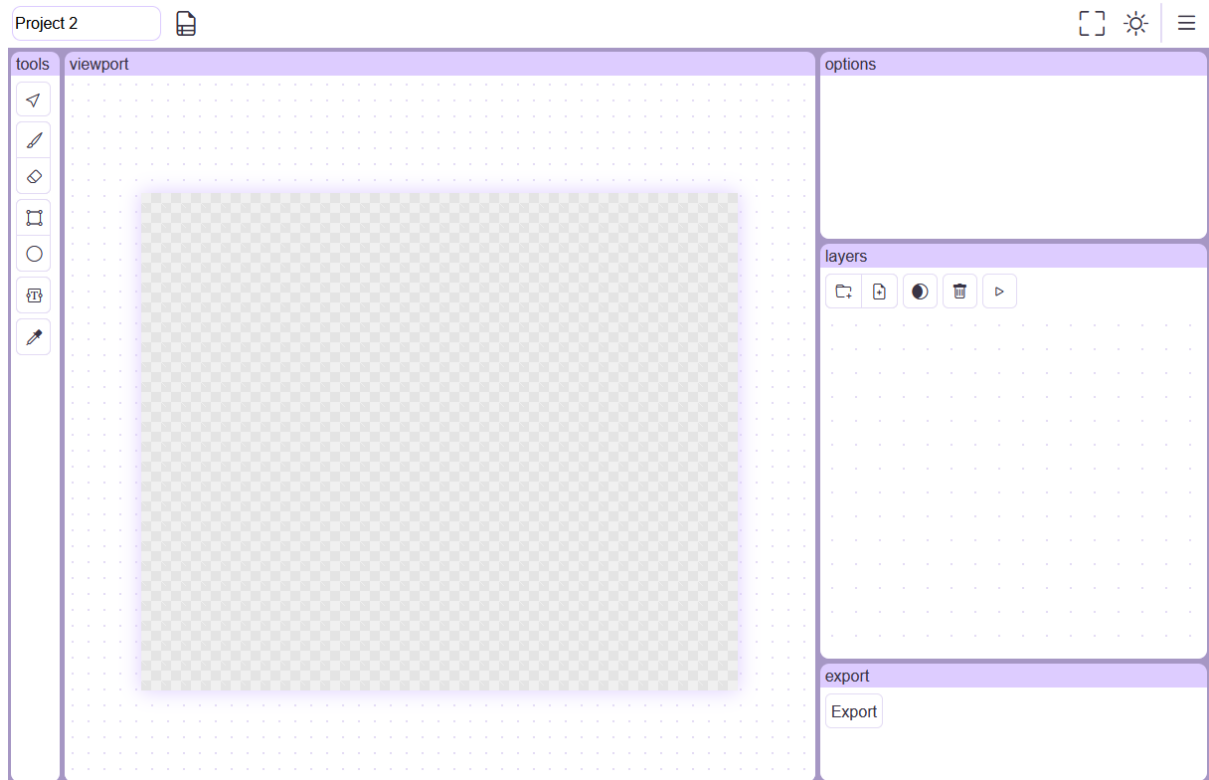


Рис. 3.2 Головна сторінка додатку

Користувач має доступ до декількох екранів на головній сторінці. Розглянемо функціонал та способи взаємодії з кожним екраном. Найголовнішим екраном у даному випадку є екран робочої області (рис 3.3).

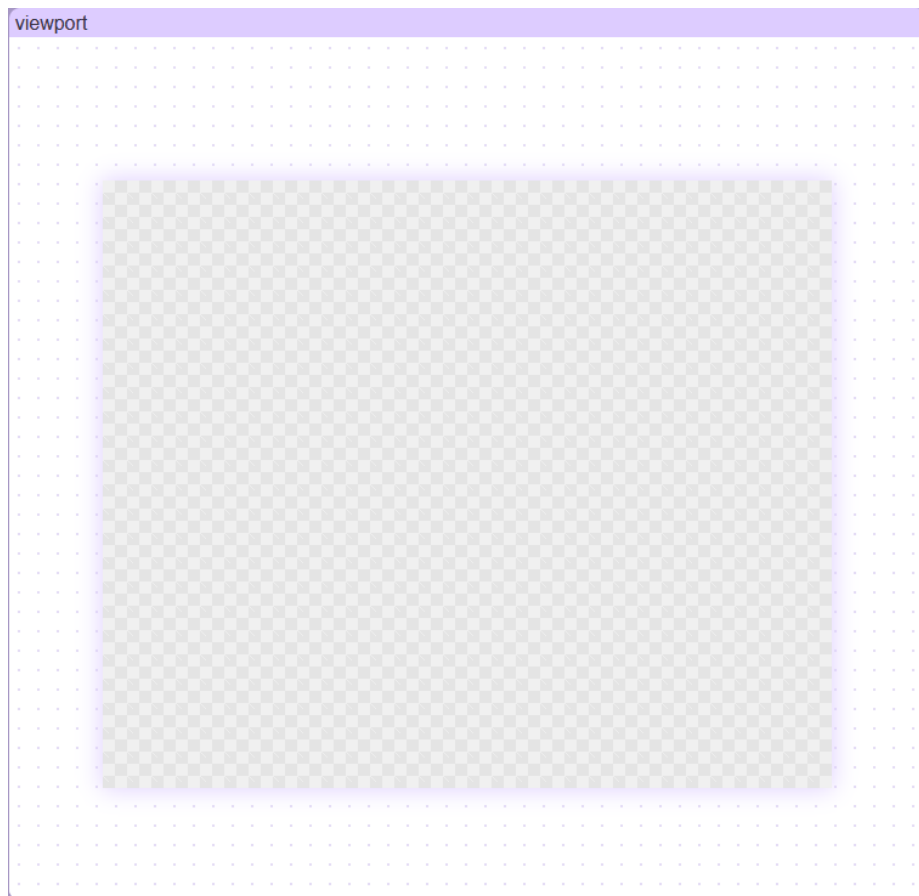


Рис. 3.3 Екран робочої області

При взаємодії з робочою областю за допомогою миші можна змінити координати та розміри холсту (рис. 3.4).

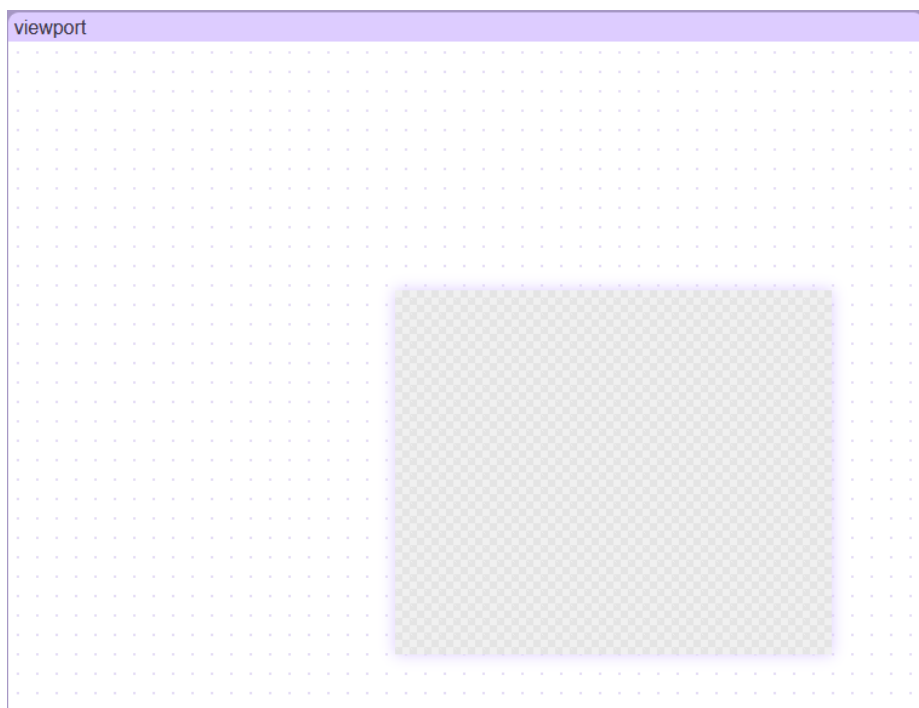


Рис. 3.4 Трансформований екран робочої області

Для вибору між інструментами малювання використовується екран з панеллю інструментів, цей екран налічує інструменти пензлик, гумка, малювання фігур, створення тексту (рис. 3.5).

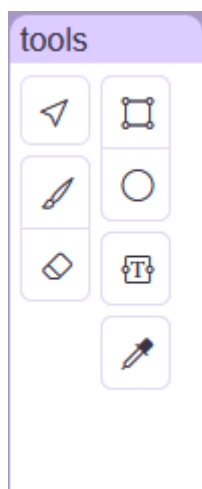


Рис. 3.5 Екран вибору інструментів

Для редагування структури шарів використовується екран «Шари» (рис. 3.6). Цей екран містить у собі кнопки для додавання нових шарів або нових груп, видалення шарів та груп та оновлення їх структури.

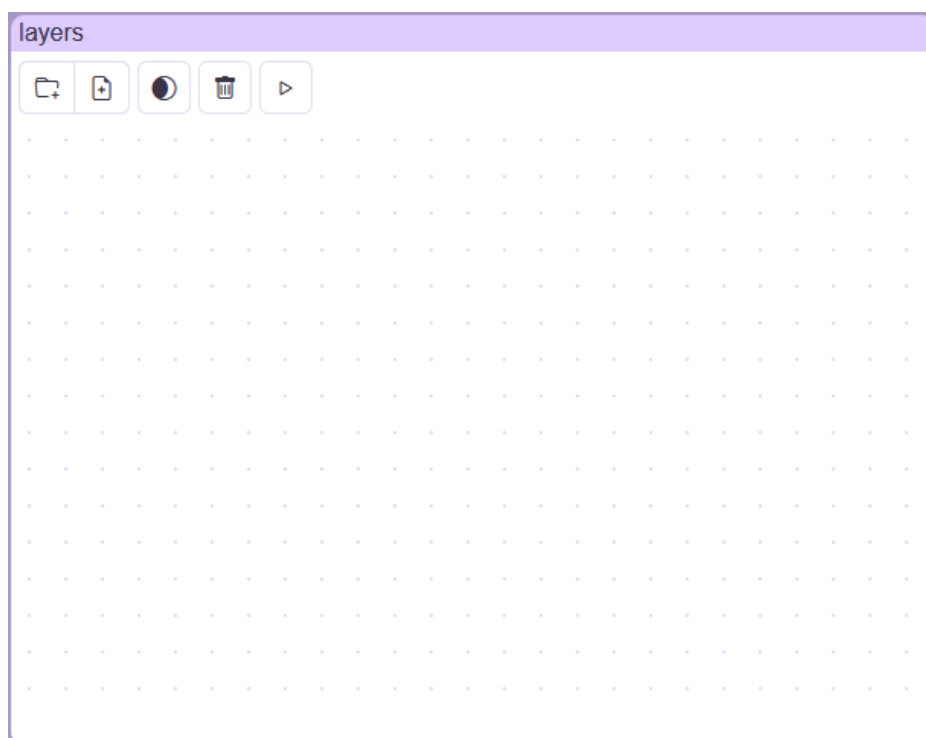


Рис. 3.6 Екран редактору шарів

При натисканні кнопок відповідаючих за створення нових шарів та груп, у провіднику системи шарів з'являються нові елементи. У елемента «група» є власний провідник, у який ми так само можемо переміщувати шари та групи. У результаті взаємодії з екраном виникає подібна структура (рис. 3.7)

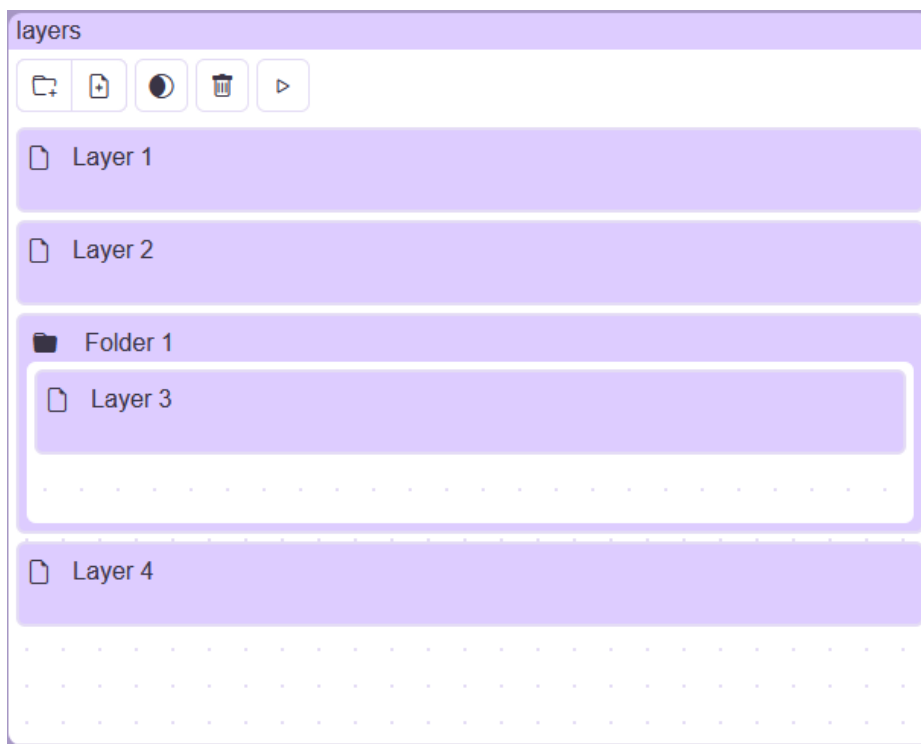


Рис. 3.7 Відредагована структура шарів

За допомогою бібліотеки Sortablejs можна виконувати сортування структури за допомогою переміщення елементів мишею (рис. 3.8).

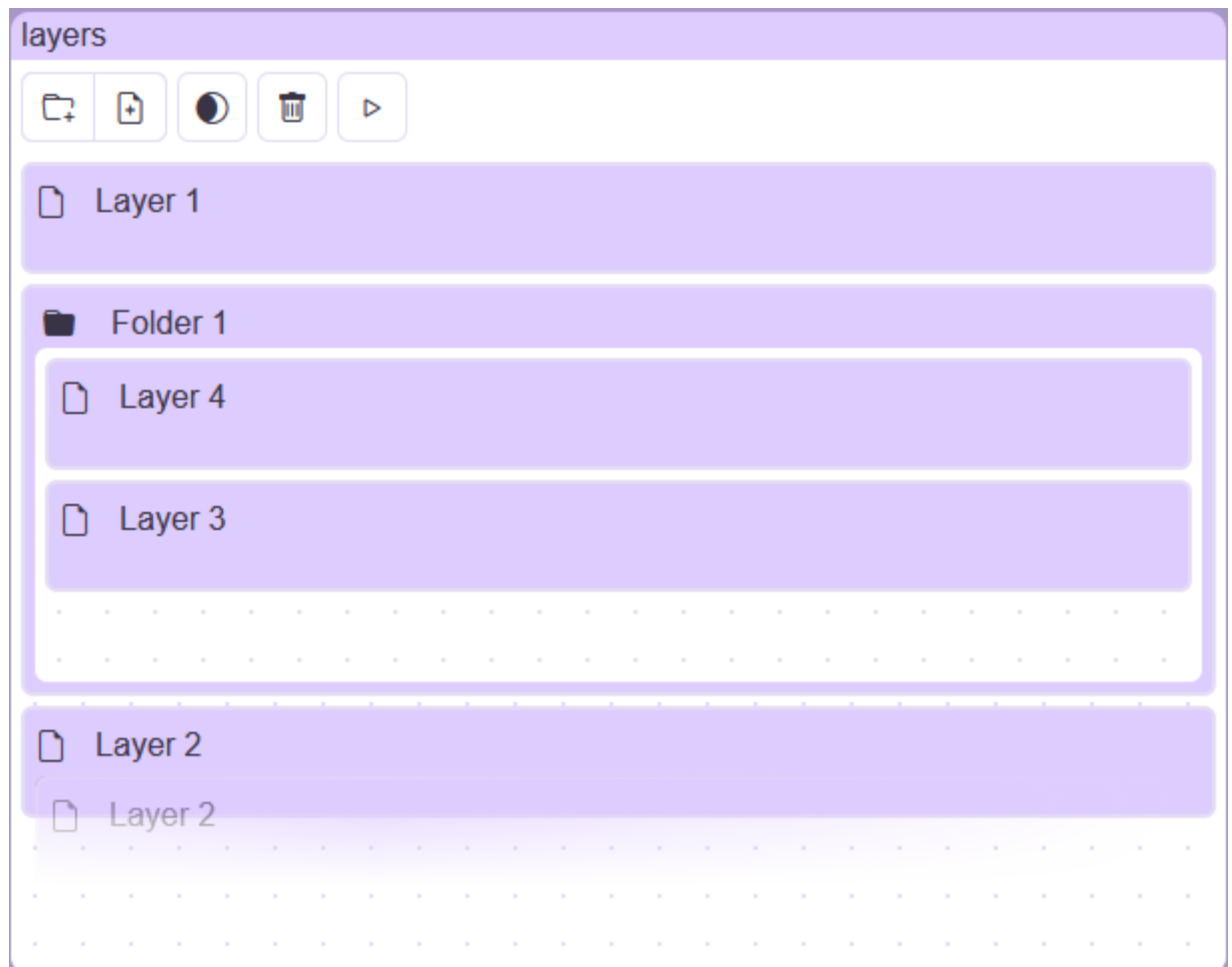


Рис. 3.8 Сортування елементів

В залежності від змін структури шарів буде змінюватися структура субхолстів у робочому просторі (рис. 3.9)(рис. 3.10).

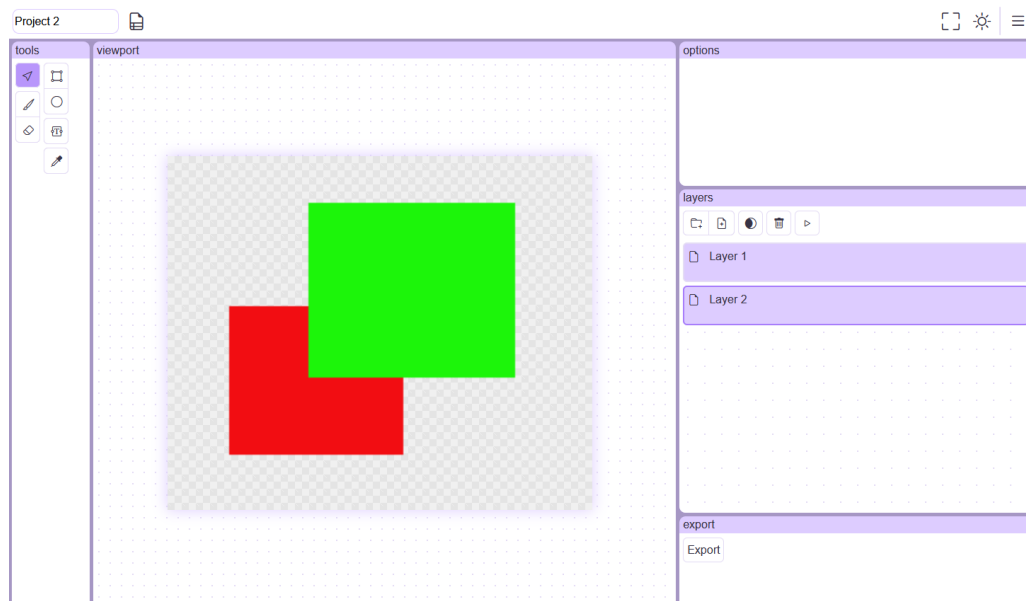


Рис. 3.9 Два об'єкти знаходяться у різних шарах

Після зміни порядку холстів, об'єкти змінили свою ієрархію у робочому просторі (рис. 3.10).

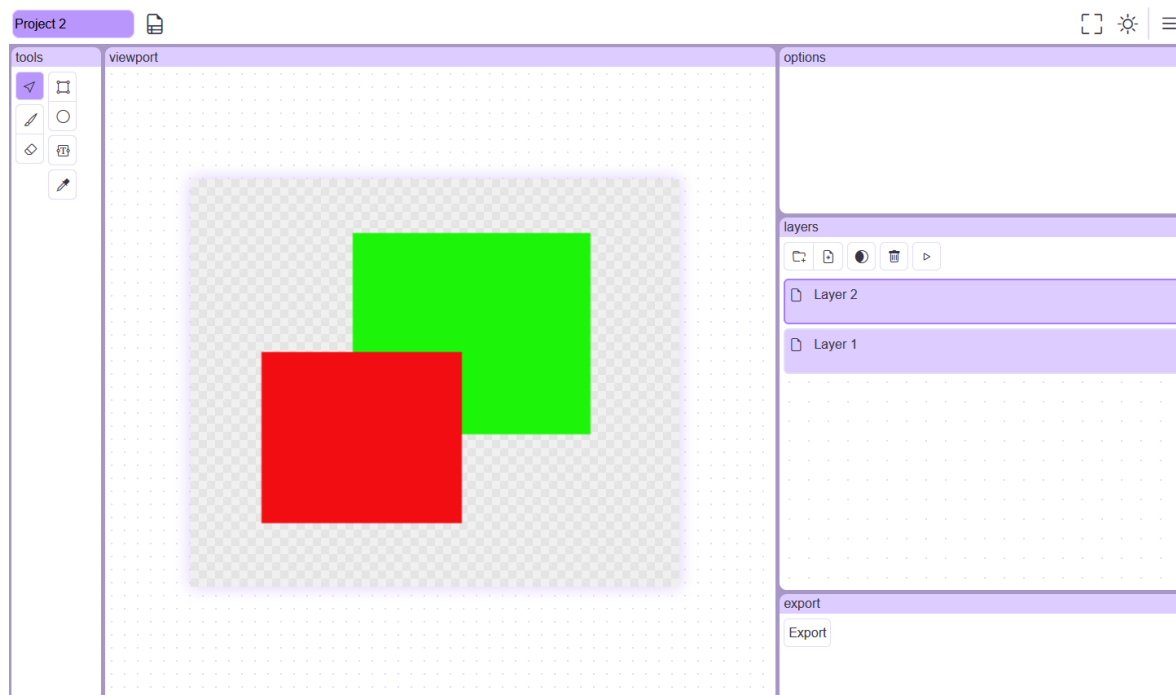


Рис. 3.10 Результат зміни порядку шарів

Для налаштування параметрів інструментів використовується панель опцій (рис. 3.11). Цей екран отримує дані про вибраний інструмент, співвідносить їх з вже існуючими параметрами у localStorage, та оновлює їх при внесенні змін користувачем (рис. 3.12).

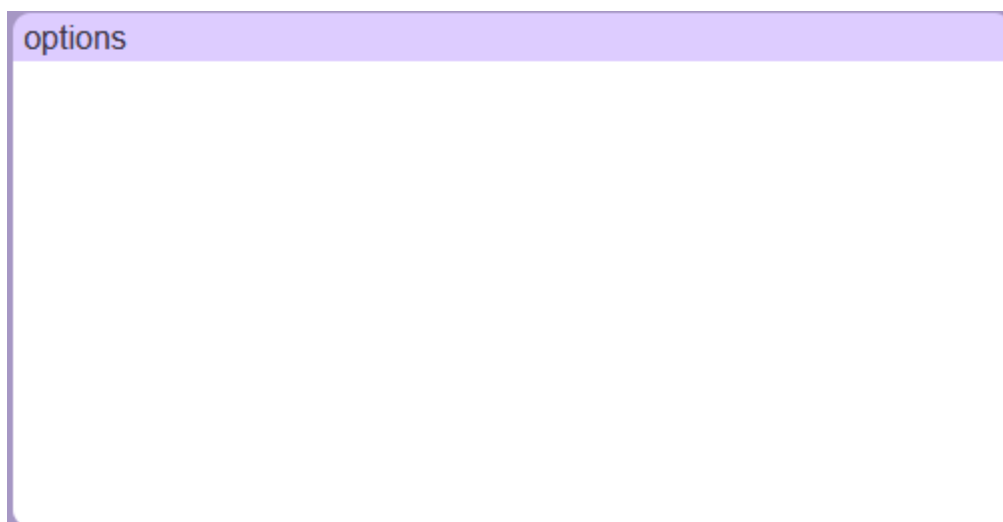


Рис. 3.11 Екран опцій

Після вибору інструменту панель опцій оновлюється

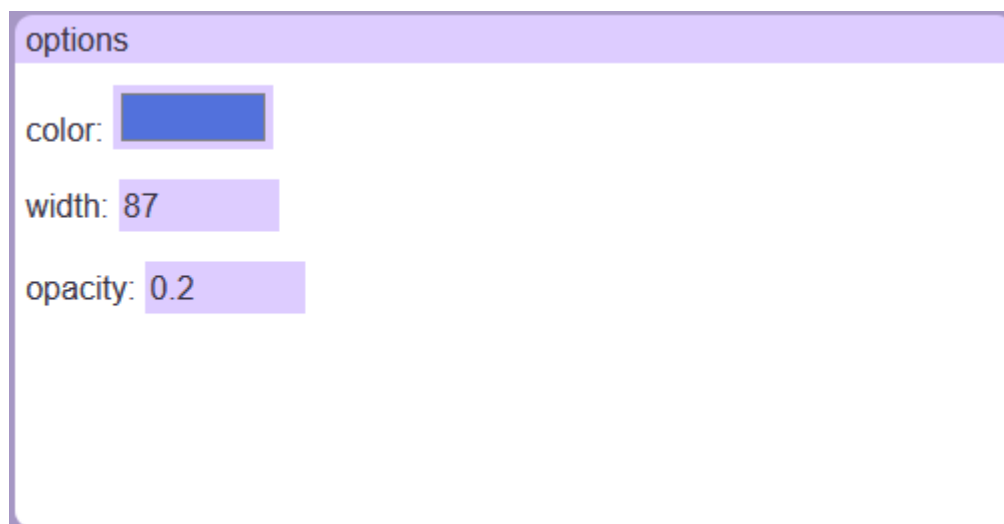


Рис. 3.12 Екран опцій з параметрами інструменту

Після оновлення параметрів інструменту, користувач може використовувати дії за допомогою оновленого інструменту, наприклад робити вставку різноманітних текстових об'єктів (рис. 3.13).



Рис. 3.13 Приклад функціонування зміни опцій інструменту

У шапці сторінки знаходиться селектор проектів, кнопка зміни теми, кнопка повноекранного режиму, кнопка для створення нового проекту та поле для зміни назви проекту (рис. 3.14).

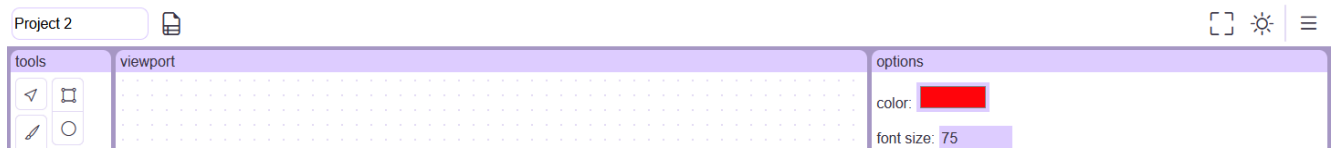


Рис. 3.14 Шапка сторінки

Відкривши селектор проектів можна вибрати інший проект, або ж видалити непотрібний (рис. 3.15).

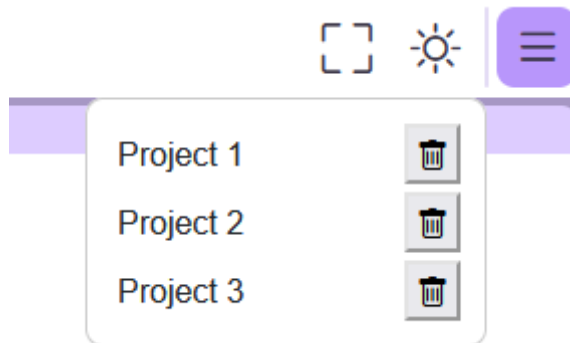


Рис. 3.15 Селектор проектів

Функціонування зміни кольорової схеми показано на рисунку (рис. 3.16) .

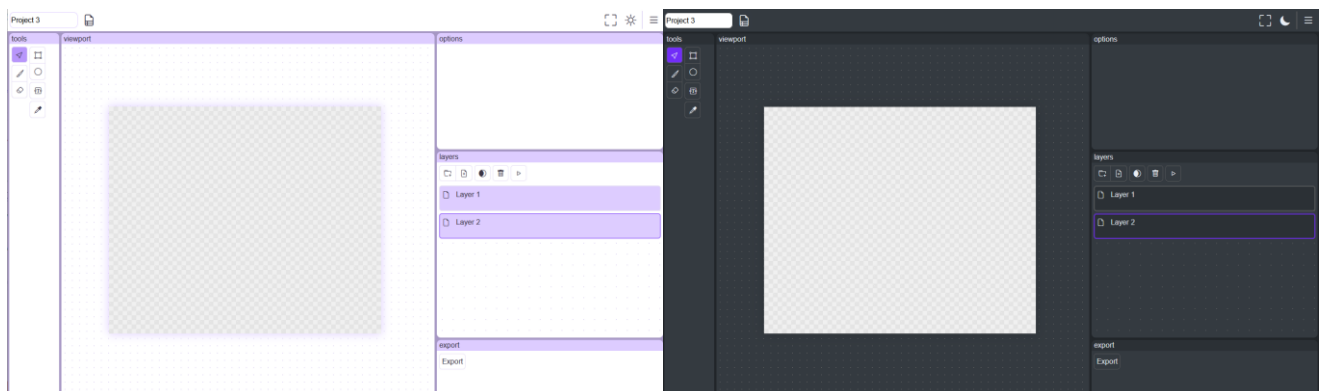


Рис. 3.16 Зміна кольорової схеми

Експорт зображень відбувається за допомогою панелі «Експорт». Після натискання кнопки експорту, додаток запитує у користувача місце збереження, та зберігає результуючий файл зображення (рис. 3.17).

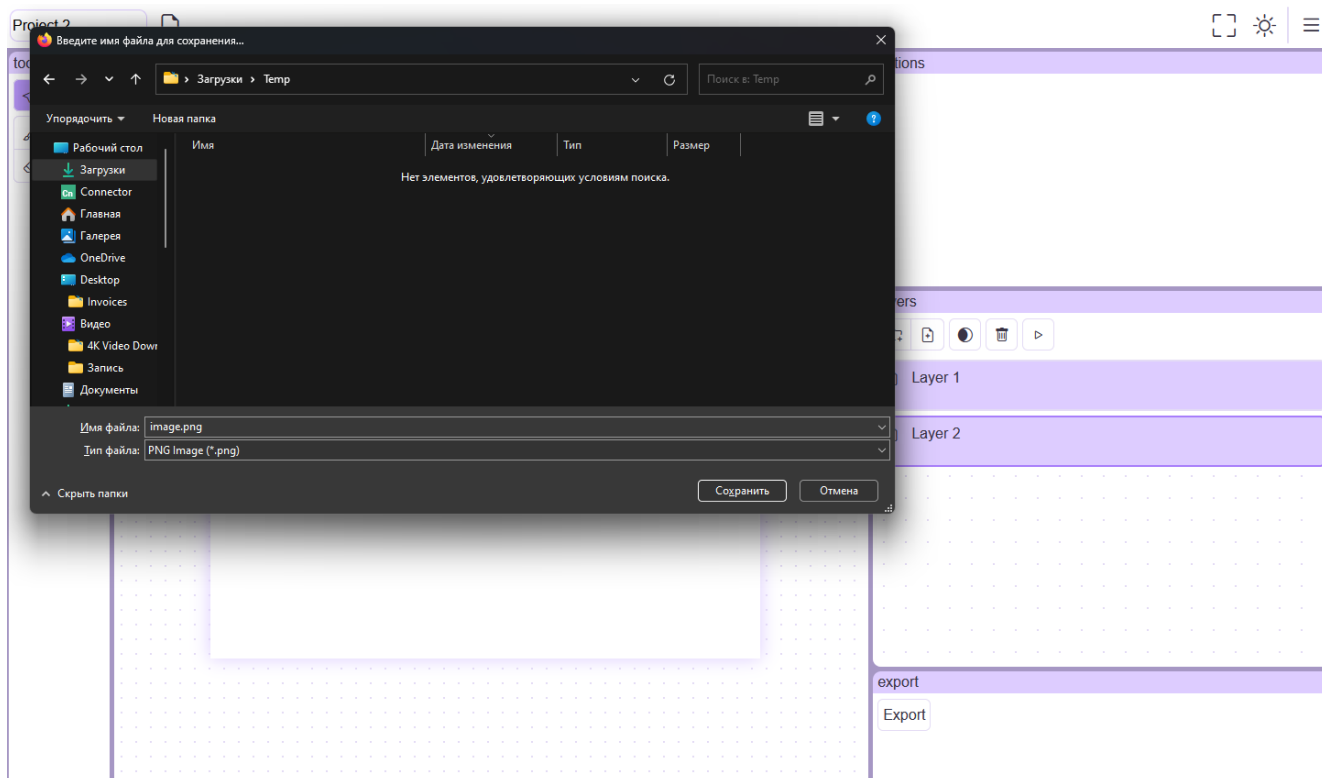


Рис. 3.17 Збереження зображення

3.3 Тестування роботи програмного комплексу

За допомогою онлайн-сервісу було проведено тестування веб-додатку. Для цього використовувався інструмент GTMetrix. GTMetrix – це сервіс, який дозволяє перевірити продуктивність та швидкість завантаження веб-сторінок. Він надає звіти та аналіз продуктивності вашого веб-сайту на основі різних метрик, таких як час завантаження сторінки, розмір сторінки, кількість запитів до сервера та інші параметри. Після завершення тестування було отримано звіт, який містить різноманітну інформацію про продуктивність сайту. (рис. 3.17)

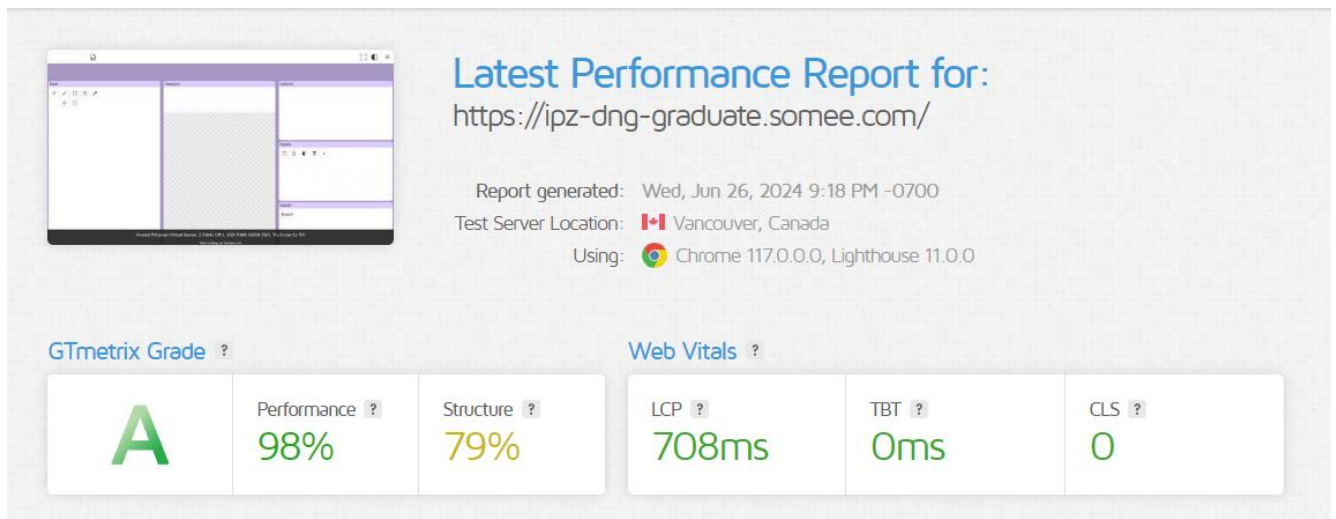


Рис. 3.18 Тестування вебдодатку за допомогою GTMetrix

Звіти GTMetrix містять різноманітну інформацію про продуктивність веб-сайту. Сервіс надає звіти та аналіз на основі різних метрик, таких як загальний час завантаження сторінки, час завантаження вмісту, кількість запитів до сервера, розмір сторінки та швидкість завантаження на різних типах з'єднань. Ця інформація допомагає зрозуміти, як наш веб-сайт працює з точки зору продуктивності та швидкості, і дозволяє вжити заходів для поліпшення цих показників (рис. 3.18).

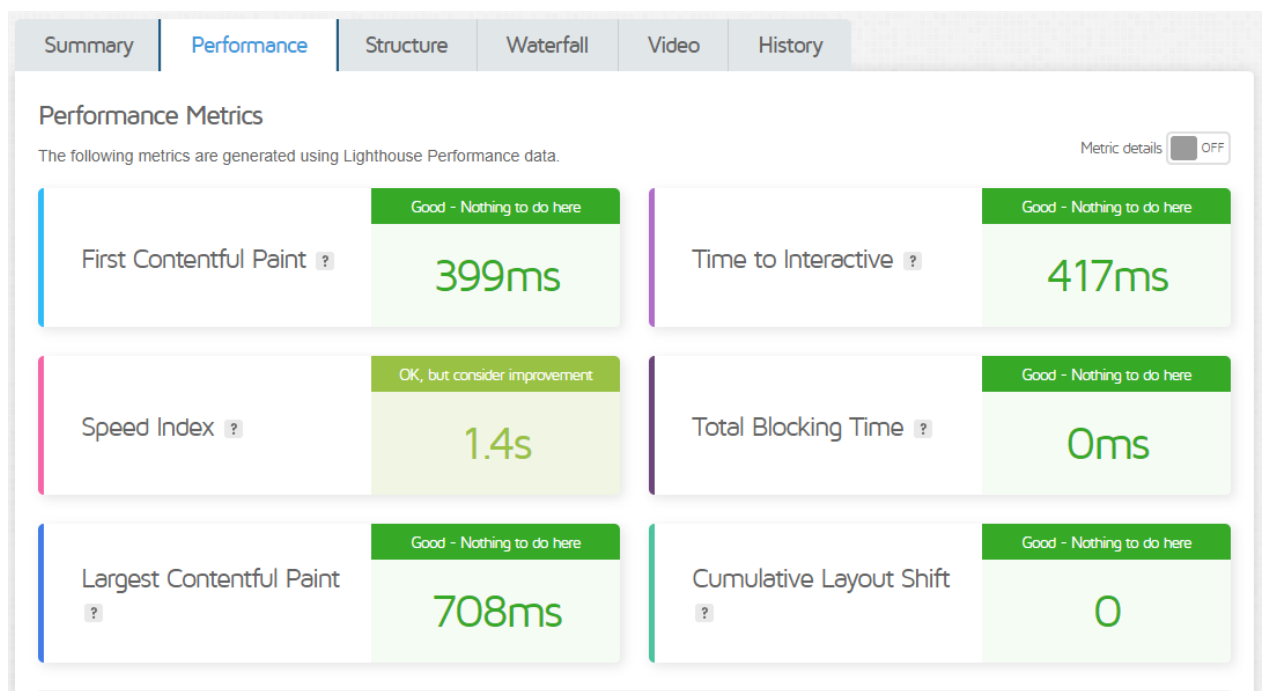


Рис. 3.19 Тестування швидкості вебдодатку за допомогою GTMetrix

Висновки до третього розділу

В даному розділі була наведена схема розгортання платформи. Було надано опис інтерфейсу розробленого програмного продукту. Були розглянуті всі функціональні можливості додатку. Було проведено аналіз взаємодій з інтерфейсом додатку. Тести швидкодії дали розуміння швидкості та якості роботи програмного додатку.

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра був розроблений онлайн растровий редактор. Спершу було проведено аналіз предметної області, який дозволив визначити основні вимоги до онлайн растрового редактора, основною задачею якого є створення користувачами растрової графіки. В процесі дослідження аналогічних та схожих за концепцією продуктів було проаналізовано і порівняно різні платформи, основним функціоналом яких є редагування растрової графіки. Крім того, були визначені основні вимоги для власної платформи. На основі цих вимог та критеріїв був обраний набір основних технологій і засобів реалізації для розробки веб-додатку.

В процесі проектування онлайн растрового редактора були розглянуті варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманих результатів було спроектовано структуру платформи. Було описано алгоритм роботи з додатком та надано послідовний опис варіантів його використання. Також наведено опис основного функціоналу додатку та реалізацію основних алгоритмів його функціонування.

Створено інтуїтивно зрозумілий інтерфейс додатку. Було розглянуто всі функціональні можливості додатку і проведено аналіз взаємодій з його інтерфейсом. Тести швидкодії дозволили оцінити швидкість та якість роботи програмного додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adobe Photoshop [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.adobe.com/ua/products/photoshop.html>
2. GIMP [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.gimp.org/>
3. Photopea [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.photopea.com/>
4. Pixlr [Електронний ресурс] – Режим доступу до ресурсу: <https://pixlr.com>
5. JavaScript Info [Електронний ресурс] – Режим доступу до ресурсу:
<https://javascript.info/>
6. MDN Web Docs – JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
7. W3schools [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.w3schools.com/js/DEFAULT.asp>
8. ASP.NET Core Blazor [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0>
9. .NET documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/dotnet/?view=aspnetcore-8.0>
10. Blazor tutorial [Електронний ресурс] – Режим доступу до ресурсу:
<https://dotnet.microsoft.com/en-us/learn/aspnet/blazor-tutorial/intro>
11. Blazor University [Електронний ресурс] – Режим доступу до ресурсу:
<https://blazor-university.com/>
12. ASP.NET Core Blazor JavaScript interoperability [Електронний ресурс] –
Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/?view=aspnetcore-8.0>
13. Call JavaScript functions from .NET methods in ASP.NET Core Blazor
[Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-8.0>

14. JavaScript [JSImport]/[JSExport] interop with ASP.NET Core Blazor
[Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interopability/import-export-interop?view=aspnetcore-8.0>
15. Fabric.js Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: <http://fabricjs.com/docs/>
16. Fabric.js Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <http://fabricjs.com/articles/>
17. Fabric.js Demos [Електронний ресурс] – Режим доступу до ресурсу: <http://fabricjs.com/demos/>
18. Fabric.js CustomBuild [Електронний ресурс] – Режим доступу до ресурсу: <http://fabricjs.com/build/>
19. SortableJs [Електронний ресурс] – Режим доступу до ресурсу: <https://sortablejs.github.io/Sortable/>
20. Learn Razor Pages [Електронний ресурс] – Режим доступу до ресурсу: <https://www.learnrazorpages.com/>
21. Introduction to Razor Pages in ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio>
22. ASP.NET Core Blazor render modes [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0>
23. "Blazor Revealed" by Peter Himschoot, 2019 – 247с.
24. "Blazor in Action" by Chris Sainty, 2022 – 380с.
25. "C# 9 and .NET 5 – Modern Cross-Platform Development" by Mark J. Price, (2020) – 822с.
26. "ASP.NET Core 5 for Beginners" by Andreas Helland, 2020 – 602с.
27. "Blazor WebAssembly by Example" by Toi B. Wright and Scott Hanselman, 2021 – 438с.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

Основна сторінка додатку, Viewport:

```
@page "/viewport"
@rendermode InteractiveServer
@inject IJSRuntime JSRuntime

<link rel="stylesheet" href="pages/Viewport.css"/>

@using BlazorApp.Components.Layout
@layout ComponentLayout

@*<div class="resize"></div>*@
<div id="viewport">
    <div id="canvasContainer" class="canvasContainer"></div>
    @*<canvas id="canvas" width="600" height="600" style="background-color:
white"></canvas>*@
</div>

<script src="Components/Pages/Viewport.razor.js"></script>
<script src="js/fabricjs/fabric.js"></script>
<script src="js/SetLocalStorage.js"></script>

@*
<button @onclick="SavePng">Save</button>
<button @onclick="ClearCanvas">Clear</button>
*@

<script>
    function initializeCanvas() {
        var viewport = document.getElementById('viewport');

        var canvasContainer = document.getElementById('canvasContainer');
        var canvasWidth = 600;
        var canvasHeight = 500;

        var data;
        data = localStorage.getItem('layers');
        if (!data) {
            data = JSON.stringify({
                "selected": "canvas1",
                "layers": [
                    { "id": "canvas1", "name": "Layer 1" }
                ]
            });
        }

        //змінні масштабування та переміщення
        var scale = 1.0; // Початковий масштаб
        var minscale = 0.1;
        var maxscale = 5.0;
        const zoomSpeed = 0.1; // Швидкість масштабування

        var canvasPosX = - canvasWidth / 2, canvasPosY = - canvasHeight / 2;
        var oldPosX = - canvasWidth / 2, oldPosY = - canvasHeight / 2;

        var isDragging = false;
        var startX, startY;

        var painting = false;

        class canvasTransform {
```

```

static transform = '';
static #translate = '';
static #scale = '';
static translate(valueX, valueY) {
    this.#translate = `translate(${valueX}px,
        ${valueY}px)`;
    this.refactor();
}
static scale(value) {
    this.#scale = `scale(${value})`;
    this.refactor();
}
static refactor() {
    this.transform = this.#translate + '\n' + this.#scale;
    canvasContainer.style.transform = this.transform;
}
}
canvasTransform.translate(oldPosX, oldPosY);

//Список шарів, та активний шар
var canvases = [];
var activeCanvas;
var canvasData = {};

createCanvases(data);
updateCanvasSize();

function saveCanvasDataToProject(projectId) {
    saveCanvasData(); // Збереження структури canvasData

    console.log(getProjectById(projectId));
    getProjectById(projectId).then(project => {
        project.data.canvasData = canvasData; // Оновлення даних у IndexedDB
        return updateProject(project);
    });
}

function loadCanvasDataFromProject(projectId) {
    return getProjectById(projectId).then(project => {
        canvasData = project.data.canvasData || {}; // Завантаження даних canvas
        loadCanvasData(); // Завантаження даних на canvas
    });
}

//Функція для зміни шарів, їх структури та їх властивостей
function createCanvases(data) {
    data = JSON.parse(data);
    const layers = data["layers"];

    const canvasContainer = document.getElementById('canvasContainer');
    canvasContainer.innerHTML = ''; // Clear existing canvases

    // Функція для обробки кожного шару або папки
    function processLayer(layer) {
        if (layer.items) {
            // Якщо папка
            layer.items.forEach(processLayer);
        } else {
            // Якщо шар
            createCanvas(layer);
        }
    }
}

```

```

    }

    // Створення шарів, та тригер зміни IndexedDB
    function createCanvas(layer) {
        // Create canvas wrapper
        const canvasWrapper = document.createElement('div');
        canvasWrapper.classList.add('canvas_wrapper');
        canvasWrapper.style.width = canvasWidth + 'px';
        canvasWrapper.style.height = canvasHeight + 'px';
        canvasContainer.appendChild(canvasWrapper);

        // Create canvas element
        const canvasElement = document.createElement('canvas');
        canvasElement.id = layer.id;
        canvasElement.width = canvasWidth;
        canvasElement.height = canvasHeight;

        canvasWrapper.appendChild(canvasElement);

        // Initialize Fabric.js canvas
        const fabricCanvas = new fabric.Canvas(layer.id);
        canvases.push(fabricCanvas);

        //Оновлення IndexedDB при змінах у шарі
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        function sendToIDB() {
            var id = parseInt(localStorage.getItem('currentProjectId'));
            if (id) {
                saveCanvasDataToProject(id);
            }
        }
        fabricCanvas.on('mouse:down', sendToIDB);
        fabricCanvas.on('mouse:leave', sendToIDB);
        fabricCanvas.on('mouse:up:before', sendToIDB);
        fabricCanvas.on('object:modified', sendToIDB);

        // Set the background color if available
        if (layer.background) {
            fabricCanvas.setBackgroundColor(layer.background,
            fabricCanvas.renderAll.bind(fabricCanvas));
        }

        // Load saved data if available
        if (canvasData[layer.id]) {
            fabricCanvas.loadFromJSON(canvasData[layer.id],
            fabricCanvas.renderAll.bind(fabricCanvas));
        }

        if (layer.id === data.selected) {
            activateCanvas(fabricCanvas);
        } else {
            deactivateCanvas(fabricCanvas);
        }
    }

    // Обробка всіх шарів та папок
    layers.forEach(processLayer);

    document.addEventListener('keydown', function (event) {
        if (event.key === 'Delete') {
            if (activeCanvas && activeCanvas.getActiveObject()) {
                activeCanvas.remove(activeCanvas.getActiveObject());
                activeCanvas.requestRenderAll();
            }
        }
    });

```

```

    });
  }

  function selectCanvas(canvas, data) {
    canvases.forEach(c => deactivateCanvas(c));
    activateCanvas(canvas);

    data.selected = canvas.lowerCanvasEl.id;
    const event = new CustomEvent('layerUpdate', { detail: JSON.stringify(data) });
  });
  document.dispatchEvent(event);
}

function activateCanvas(canvas) {
  activeCanvas = canvas;
  canvas.getElement().parentNode.style.pointerEvents = 'auto';

  canvas.forEachObject(function (obj) {
    obj.evented = true;
    obj.selectable = true;
  });

  canvas.discardActiveObject();
  canvas.requestRenderAll();
}

function deactivateCanvas(canvas) {
  canvas.isDrawingMode = false;
  canvas.getElement().parentNode.style.pointerEvents = 'none';

  canvas.forEachObject(function (obj) {
    obj.evented = false;
    obj.selectable = false;
  });

  canvas.discardActiveObject();
  canvas.requestRenderAll();
}

function saveCanvasData() {
  canvases.forEach(canvas => {
    canvasData[canvas.lowerCanvasEl.id] = canvas.toJSON();
  });
}

function saveCombinedImage() {
const mergedCanvas = document.createElement('canvas');
const ctx = mergedCanvas.getContext('2d');

mergedCanvas.width = canvasWidth;
mergedCanvas.height = canvasHeight;

canvases.forEach(canvas => {
  ctx.globalAlpha = 1.0; // Adjust transparency as needed
  ctx.drawImage(canvas.lowerCanvasEl, 0, 0);
});

const image = mergedCanvas.toDataURL('image/png');
const link = document.createElement('a');
link.href = image;
link.download = 'image.png';
link.click();
}

```

```

function updateCanvasSize() {
    canvasContainer.style.width = canvasWidth + 'px';
    canvasContainer.style.height = canvasHeight + 'px';

    canvases.forEach(canvas => {
        const wrapper = canvas.getElement().parentNode;
        wrapper.style.width = canvasWidth + 'px';
        wrapper.style.height = canvasHeight + 'px';
        canvas.setWidth(canvasWidth);
        canvas.setHeight(canvasHeight);
        canvas.renderAll();
    });
}

localStorage.removeItem('options');
localStorage.removeItem('select');
var readOptionsEvent;
function selectTool(tool) {
    if (!activeCanvas) return;

    window.removeEventListener('storage', readOptionsEvent);
    readOptionsEvent = (e) => {
        onLocalStorageChange(e, 'options', () => {
            localStorage.setItem(tool,
                localStorage.getItem('options'));
            selectTool(tool);
        });
    };
    window.addEventListener('storage', readOptionsEvent);

    clearCanvasEvents();
    const storage = localStorage.getItem(tool); //Для перевірки наявності

    activeCanvas.on('selection:created', (e) => objectSelection(e));
    activeCanvas.on('selection:updated', (e) => objectSelection(e));
    activeCanvas.on('selection:cleared', (e) => setOptions());

    //Вибір інструменту
    if (tool === 'select') {
        localStorage.removeItem('options');
        localStorage.removeItem('select');
        activeCanvas.isDrawingMode = false; // Вимикаємо малювання
        activeCanvas.selection = true; // Вмикаємо вибір об'єктів
        activeCanvas.forEachObject(function (obj) {
            obj.selectable = true;
            obj.evented = true;
        });
    }
    if (tool === 'brush') {
        activeCanvas.isDrawingMode = true;
        const brush = new fabric.PencilBrush(activeCanvas);
        if (!storage) {
            localStorage.setItem(tool, JSON.stringify({
                "color": { "type": "color", "value": "#0000FF" },
                "width": { "type": "float", "value": 10 },
                "opacity": { "type": "float", "value": 0.8 }
            }));
        }; //Стандартні значення

        const values = JSON.parse(storage);
        brush.color = hexToRgba(values["color"]["value"], values["opacity"]["value"]);
        brush.width = parseFloat(values["width"]["value"]);
        activeCanvas.freeDrawingBrush = brush;
    }
}

```

```

if (tool === 'eraser') {
    activeCanvas.isDrawingMode = true;
    const eraser = new fabric.EraserBrush(activeCanvas);
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({
            "width": { "type": "float", "value": 15 }
        }));
    }; //Стандартні значення

    const values = JSON.parse(storage);
    eraser.width = parseFloat(values["width"]["value"]);
    activeCanvas.freeDrawingBrush = eraser;
}
if (tool === 'rect') {
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({
            "color": { "type": "color", "value": "#0000FF" },
            "opacity": { "type": "float", "value": 0.8 }
        }));
    }; //Стандартні значення

    const values = JSON.parse(storage);
    activeCanvas.on('mouse:down', function (opt) {
        isDrawing = true;
        const pointer = activeCanvas.getPointer(opt.e);
        const origX = pointer.x;
        const origY = pointer.y;
        drawingShape = new fabric.Rect({
            left: origX,
            top: origY,
            fill: values["color"]["value"],
            opacity: values["opacity"]["value"],
            width: 1,
            height: 1
        });
        activeCanvas.add(drawingShape);

        activeCanvas.on('mouse:move', function (opt) {
            if (!isDrawing) return;
            const pointer = activeCanvas.getPointer(opt.e);
            var width = pointer.x - origX;
            var height = pointer.y - origY;

            if (width < 0) {
                drawingShape.set({ left: pointer.x });
                width = Math.abs(width);
            }
            if (height < 0) {
                drawingShape.set({ top: pointer.y });
                height = Math.abs(height);
            }
            drawingShape.set({ width: width, height: height });
            activeCanvas.renderAll();
        });

        activeCanvas.on('mouse:up', function (opt) {
            isDrawing = false;
            drawingShape.setCoords();
        });
    });
}
if (tool === 'circle') {
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({

```

```

        "color": { "type": "color", "value": "#0000FF" },
        "opacity": { "type": "float", "value": 0.8 }
    });
}; //Стандартні значення

const values = JSON.parse(storage);
activeCanvas.on('mouse:down', function (opt) {
    isDrawing = true;
    const pointer = activeCanvas.getPointer(opt.e);
    drawingShape = new fabric.Circle({
        left: pointer.x,
        top: pointer.y,
        fill: values["color"]["value"],
        opacity: values["opacity"]["value"],
        radius: 1,
        originX: 'center',
        originY: 'center'
    });
    activeCanvas.add(drawingShape);
});

activeCanvas.on('mouse:move', function (opt) {
    if (!isDrawing) return;
    const pointer = activeCanvas.getPointer(opt.e);
    const radius = Math.sqrt(Math.pow(pointer.x - drawingShape.left, 2) +
Math.pow(pointer.y - drawingShape.top, 2));
    drawingShape.set({ radius: radius });
    activeCanvas.renderAll();
});

activeCanvas.on('mouse:up', function (opt) {
    isDrawing = false;
    drawingShape.setCoords();
});
}
if (tool === 'text') {
    if (!storage) {
        localStorage.setItem(tool, JSON.stringify({
            "color": { "type": "color", "value": "#000000" },
            "font size": { "type": "float", "value": 50 },
            "font weight": { "type": "float", "value": 'normal' },
            "font family": { "type": "float", "value": 'Times New Roman' },
            "opacity": { "type": "float", "value": 0.8 }
        }));
    }; //Стандартні значення

    const values = JSON.parse(storage);
    // Функція для створення текстового об'єкта
    function createTextObject(x, y) {
        const text = new fabric.IText('', {
            left: x,
            top: y,
            fill: values["color"]["value"],
            fontSize: values["font size"]["value"],
            fontWeight: values["font weight"]["value"],
            fontFamily: values["font family"]["value"],
            opacity: values["opacity"]["value"],
            selectable: true,
            editable: true
        });
        activeCanvas.add(text).setActiveObject(text);
        text.enterEditing();
        activeCanvas.renderAll();
    }
}

```

```

    }

    activeCanvas.on('mouse:down', function (opt) {
        const pointer = activeCanvas.getPointer(opt.e);
        createTextObject(pointer.x, pointer.y);
    });
}

function clearCanvasEvents() {
    activeCanvas.isDrawingMode = false;
    activeCanvas.off('mouse:down');
    activeCanvas.off('mouse:move');
    activeCanvas.off('mouse:up');
    activeCanvas.off('selection:created');
    activeCanvas.off('selection:updated');
    activeCanvas.off('selection:cleared');
    activeCanvas.selection = false;
    activeCanvas.forEachObject(function (obj) {
        obj.selectable = false;
        obj.evented = false;
    });
}

function getObjectParameters(object) {
    var params = {};

    if (object.type === 'brush') {
        params = {
            "color": { "type": "color", "value": object.color },
            "opacity": { "type": "float", "value": object.opacity }
        };
    }
    if (object.type === 'rect') {
        params = {
            "color": { "type": "color", "value": object.fill },
            "width": { "type": "float", "value": object.width * object.scaleX },
            "height": { "type": "float", "value": object.height * object.scaleY }
        };
    }
    if (object.type === 'circle') {
        params = {
            "color": { "type": "color", "value": object.fill },
            "radius": { "type": "float", "value": object.radius * object.scaleX }
        };
    }
    if (object.type === 'i-text' || object.type === 'textbox') {
        params = {
            "color": { "type": "color", "value": object.color },
            "text": { "type": "string", "value": object.text },
            "fontSize": { "type": "float", "value": object.fontSize }
        };
    }

    return JSON.stringify(params);
}

function updateObjectParameters(object) {
    let params = getObjectParameters(object);
    localStorage.setItem('selectedObjectParams', params);

    // Optional: Update UI elements if you have any for displaying object parameters
    // Example:
    // document.getElementById('colorInput').value = object.fill || '#000';
    // document.getElementById('widthInput').value = object.width * object.scaleX ||
0;

```

```

|| 0; // document.getElementById('heightInput').value = object.height * object.scaleY

window.addEventListener("storage", function (e) {
    onLocalStorageChange(e, tool, () => {
        selectTool(tool);
    });
});

function objectSelection(event) {
    var selectedObject = event.selected;
    if (selectedObject) {
        updateObjectParameters(selectedObject);

        // Get parameters JSON string
        var objectParams = getObjectParameters(selectedObject);

        //localStorage.setItem('options', objectParams);
        // TODO !!!!!!!!!!!!!!!!!!!!!

        localStorage.removeItem('options');
        localStorage.setItem('options', objectParams);
        console.log('Object parameters JSON:', localStorage.getItem('options'));
    }
}

function hexToRGBA(hex, opacity) {
    var c;
    if (/^#([A-Fa-f0-9]{3}){1,2}$/.test(hex)) {
        c = hex.substring(1).split('');
        if (c.length == 3) {
            c = [c[0], c[0], c[1], c[1], c[2], c[2]];
        }
        c = '0x' + c.join('');
        return 'rgba(' + [(c >> 16) & 255, (c >> 8) & 255, c & 255].join(',') + ',' +
opacity + ')';
    }
    throw new Error('Bad Hex');
}

function readOptionsEvent(e, item, func) {
    onLocalStorageChange(e, 'options', () => {
        localStorage.setItem(item,
            localStorage.getItem('options'));
        func(item);
    });
}

viewport.addEventListener('paste', function () {
    const items = event.clipboardData.items;
    for (let i = 0; i < items.length; i++) {
        if (items[i].type.indexOf('image') !== -1) {
            const blob = items[i].getAsFile();
            const reader = new FileReader();
            reader.onload = function (event) {
                const img = new Image();
                img.src = event.target.result;
                img.onload = function () {
                    const image = new fabric.Image(img);
                    image.set({
                        left: activeCanvas.width / 2,
                        top: activeCanvas.height / 2,
                        originX: 'center',

```

```

        originY: 'center'
    });
    activeCanvas.add(image);
    activeCanvas.renderAll();
    };
    };
    reader.readAsDataURL(blob);
}
}
});

viewport.addEventListener('drop', function (event) {
    event.preventDefault();
    console.log('fileAdded');
    const files = event.dataTransfer.files;
    for (let i = 0; i < files.length; i++) {
        const file = files[i];
        if (file.type.indexOf('image') !== -1) {
            const reader = new FileReader();
            reader.onload = function (e) {
                const img = new Image();
                img.src = e.target.result;
                img.onload = function () {
                    const image = new fabric.Image(img);

// Масштабируем изображение, если его размер превышает
размеры холста

                    const scaleX = canvasWidth / img.width;
                    const scaleY = canvasHeight / img.height;
                    const scale = Math.min(scaleX, scaleY, 1); // Ограничиваем
масштаб до 1 (100%)

                    image.set({
                        left: activeCanvas.width / 2,
                        top: activeCanvas.height / 2,
                        originX: 'center',
                        originY: 'center',
                        scaleX: scale,
                        scaleY: scale
                    });
                    activeCanvas.add(image);
                    activeCanvas.renderAll();
                };
            };
            reader.readAsDataURL(file);
        }
    }
});

viewport.addEventListener('mousedown', function (e) {
    switch (e.button) {
        case 1:
            isDragging = true;
            startX = e.clientX;
            startY = e.clientY;
            oldPosX = canvasPosX;
            oldPosY = canvasPosY;
            break;
    }
}); //MouseDown

viewport.addEventListener('mousemove', function (e) {
    if (isDragging) {

```

```

        var currentX = e.clientX;
        var currentY = e.clientY;
        canvasPosX = oldPosX + currentX - startX;
        canvasPosY = oldPosY + currentY - startY;
        canvasTransform.translate(canvasPosX, canvasPosY);
    }
}); //MouseMove

viewport.addEventListener('mouseup', function (e) {
    isDragging = false;
}); //MouseUp

viewport.addEventListener('mouseleave', function (e) {
    isDragging = false;
}); //MouseLeave

viewport.addEventListener('wheel', function (e) {
    e.preventDefault();
    var x = e.deltaX;
    var y = e.deltaY;

    if (y > 0) {
        if (scale - zoomSpeed > minscale) {
            scale = scale - zoomSpeed;
            canvasTransform.scale(scale);
        }
    }
    else {
        if (scale + zoomSpeed < maxscale) {
            scale = scale + zoomSpeed;
            canvasTransform.scale(scale);
        }
    }
}); //Wheel

//Подія зміни інструменту
window.addEventListener("storage", function (e) {
    onLocalStorageChange(e, 'tool', () => {
        selectTool(localStorage.getItem('tool'));
        setOptions(localStorage.getItem('tool'));
    });
});

window.addEventListener("storage", function (e) {
    onLocalStorageChange(e, 'layers', () => {
        var data = localStorage.getItem('layers');
        saveCanvasData();
        createCanvases(data);
        loadCanvasData();
    });
});

document.addEventListener('DOMContentLoaded', () => {
    currentProjectId = parseInt(localStorage.getItem('currentProjectId'));
    if (currentProjectId) {
        loadCanvasDataFromProject(currentProjectId);
    }
    //displayProjects();
});

currentProjectId = parseInt(localStorage.getItem('currentProjectId'));
if (currentProjectId) {
    loadCanvasDataFromProject(currentProjectId);
}

```

```

    }

    // Експорт проекту, то завантаження проекту при його зміні
    window.addEventListener('storage', (e) => {
        if (e.storageArea === sessionStorage && e.key === 'exportProject') {
            if (sessionStorage.getItem('exportProject')) {
                saveCombinedImage();
                sessionStorage.setItem('exportProject', false);
            }
        }
        //console.log(e.key);
        if (e.storageArea === sessionStorage && e.key === 'changeProject') {
            if (sessionStorage.getItem('changeProject')) {
                currentProjectId =
                parseInt(localStorage.getItem('currentProjectId'));
                if (currentProjectId) {
                    loadCanvasDataFromProject(currentProjectId);
                }
                sessionStorage.setItem('changeProject', false);
            }
        }
    });
}
</script>

```

```

@code {
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            await JSRuntime.InvokeVoidAsync("initializeCanvas");
        }
    }
}

```