Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

**Бакалаврська кваліфікаційна робота на тему:**

# РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СХЕМ

Виконав: студент 4 курсу, групи 1КН-20Б
спеціальності 122 – Комп'ютерні науки
Полігас В.В.

Керівник: к.т.н., доцент каф.КН
Сілагін О. В.
«07» 06 2024 р.

Рецензент: к.т.н., доцент каф.АІІТ
Козачко О.М.
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А.П.
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

**ЗАТВЕРДЖУЮ**
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024 р.

**ЗАВДАННЯ**
**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
Полігасу Владиславу Володимировичу

1. Тема роботи: Розробка web-ресурсу для візуалізації алгоритмів та схем.

керівник роботи: Сілагін Олексій Віталійович, к.т.н., доц.
затверджені наказом вищого навчального закладу "11" 03 2024 року №80

2. Термін подання студентом роботи 27.05.2024

3. Вихідні дані до роботи :
Мова програмування – об'єктно-орієнтована;
Кількість варіантів аунтефікації – 3;
Інтерфейс користувача має бути інтуїтивно зрозумілий
Можливість командної роботи в реальному часі.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
- Обґрунтування доцільності розробки web-ресурсу для візуалізації алгоритмів та схем.
- Аналіз існуючих рішень та технологій web-ресурсів для візуалізації алгоритмів та схем.
- Проектування архітектури web-ресурсу для візуалізації алгоритмів та схем.
- Реалізація функціональних можливостей web-ресурсу.
- Тестування розробленого web-ресурсу для візуалізації алгоритмів та схем для забезпечення його ефективності та надійності.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): приклади реалізації інтерфейсу у готових застосунках, схеми можливих видів архітектур web-застосунку, схема основних модулів та компонентів, схема принципу UI/UX дизайну, приклад створення прототипу сайту .

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-3 | Сілагін О.В., к.т.н., доц. каф. КН | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.03.2024

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва та зміст етапу | Термін виконання | | Примітка |
|---|---|---|---|---|
| | | початок | закінчення | |
| 1 | Аналіз предметної області для візуалізації алгоритмів та схем | 06.05.24 | 12.05.24 | |
| 2 | Проектування web-ресурсу для візуалізації алгоритмів та схем | 13.05.24 | 20.05.24 | |
| 3 | Програмна реалізація web-ресурсу для візуалізації алгоритмів та схем | 21.05.24 | 30.05.24 | |
| 4 | Тестування web-ресурсу для візуалізації алгоритмів та схем | 31.05.24 | 02.06.24 | |
| 5 | Розробка інструкції користувача | 03.06.24 | 04.06.24 | |
| 6 | Оформлення матеріалів до захисту БДР | 05.06.24 | 06.06.24 | |

**Студент** _____ **Полігас В.В,**
(підпис) (прізвище та ініціали)

**Керівник проекту (роботи)** _____ **Сілагін О.В**
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Бакалаврська дипломна робота складається з 104 сторінок формату А4, на яких є 23 рисунки.

Метою роботи є доведення необхідності розширення функціональних можливостей веб-ресурсів для візуалізації алгоритмів та схем.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій доведена доцільність розробки веб-ресурсу для візуалізації алгоритмів та схем. Для веб-ресурсу розроблено алгоритм роботи, та вибрана архітектура.

Програмний продукт розроблений в інтегрованому середовищі Visual Studio Code за допомогою найновіших технологій, таких як Next.js, TypeScript, Node.js, Clerk, Convex та LiveBlocks.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції веб-ресурсу для візуалізації алгоритмів та схем реалізовано. Ключові слова: веб-ресурс, алгоритми, схеми.

## ABSTRACT

The bachelor's thesis consists of 104 A4 pages with 23 figures.

The purpose of the work is to prove the need to expand the functionality of web resources for visualizing algorithms and schemes.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies, the expediency of developing a web resource for visualizing algorithms and schemes is proved. An algorithm for the web resource has been developed and an architecture has been selected.

The software product was developed in the integrated Visual Studio Code environment using the latest technologies such as Next.js, TypeScript, Node.js, Clerk, Convex, and LiveBlocks.

The software product was developed and tested. The results of the development testing indicate that the main functions of the web resource for visualizing algorithms and schemes have been implemented. Keywords: web resource, algorithms, schemes.

ЗМІСТ

# ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у різних сферах діяльності людини. Однією з важливих задач є ефективне візуалізація алгоритмів та схем, що сприяє кращому розумінню та аналізу складних процесів. Розробка веб-ресурсів для візуалізації алгоритмів дозволяє інтерактивно і наочно представляти інформацію, що є надзвичайно актуальним для освітніх, наукових та комерційних потреб.

**Актуальність** дослідження обумовлена необхідністю розширення функціональних можливостей існуючих веб-ресурсів для візуалізації алгоритмів і схем. Це дозволить забезпечити більш ефективне навчання, аналіз даних та інтерактивну роботу з алгоритмами.

**Метою дослідження** є розширення функціональних можливостей веб-ресурсів для візуалізації алгоритмів та схем. Для досягнення цієї мети потрібно виконати наступні задачі:

- Обґрунтування доцільності розробки веб-ресурсу для візуалізації.
- Аналіз існуючих рішень та технологій у цій сфері.
- Проектування архітектури веб-ресурсу.
- Реалізація функціональних можливостей веб-ресурсу.
- Тестування розробленого веб-ресурсу для забезпечення його ефективності та надійності.

**Об’єктом дослідження** є процес розробки веб-ресурсів з візуалізації алгоритмів та схем. **Предметом дослідження** виступають програмні засоби, які використовуються для розробки таких веб-ресурсів.

Результати, одержані в процесі написання бакалаврської дипломної роботи, плануються до впровадження у розробки компанії ТОВ "Аграрна Платформа". Про це свідчить розміщення довідки про впровадження у додатку Д.

# РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-РЕСУРСУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СХЕМ

## 1.1 Обгрунтування доцільності створення веб-ресурсу для візуалізації алгоритмів та схем

Візуалізація алгоритмів і схем є важливим інструментом для покращення розуміння складних процесів і концепцій. Веб-ресурси, які надають можливість візуалізації, дозволяють користувачам інтерактивно взаємодіяти з алгоритмами, наочно представляючи їх роботу та структуру. Це особливо важливо для освітніх цілей, де студенти можуть навчатися і експериментувати з різними алгоритмами в режимі реального часу. Постановка задачі є ключовим етапом у створенні такого веб-ресурсу, оскільки вона визначає мету, основні вимоги та функціональні можливості, які повинні бути реалізовані.

Предметна область включає в себе різноманітні алгоритми та схеми, які використовуються в різних галузях знань. Алгоритми можуть бути поділені на кілька категорій, таких як сортування, пошук, графові алгоритми, алгоритми обробки даних та інші. Кожен тип алгоритму має свої особливості та вимоги до візуалізації. Наприклад, алгоритми сортування часто включають множинні кроки, які можуть бути відображені у вигляді анімацій, що показують зміну порядку елементів. Графові алгоритми, такі як алгоритм Дейкстри або алгоритм Пріма, можуть бути представлені у вигляді графів з вузлами та ребрами.

Важливою частиною предметної області є також методи візуалізації, які можуть використовуватися для відображення алгоритмів. Це можуть бути як стандартні способи, такі як блок-схеми або діаграми станів, так і більш складні інтерактивні візуалізації, які дозволяють користувачам взаємодіяти з алгоритмом, змінювати його параметри в режимі реального часу і бачити результати цих змін. Використання різних технологій для візуалізації, дозволяє створювати багатофункціональні інтерфейси, що роблять процес вивчення алгоритмів більш захопливим та ефективним.

На сьогоднішній день існує велика кількість алгоритмів, які застосовуються в різних галузях науки та техніки. Однак, багато з цих алгоритмів є досить складними для розуміння без відповідного візуального представлення. Веб-ресурси, які дозволяють візуалізувати алгоритми, надають можливість покращити процес навчання та дослідження, забезпечуючи інтуїтивно зрозуміле представлення складних процесів. Таким чином, створення такого веб-ресурсу є актуальним завданням, яке може сприяти покращенню якості освіти та професійного розвитку.

Метою даної роботи є створення веб-ресурсу, який дозволяє візуалізувати алгоритми та схеми. Цей ресурс повинен надавати користувачам можливість взаємодії з алгоритмами, включаючи їх створення, редагування, та детальне відображення кожного кроку алгоритму. Окрім цього, веб-ресурс повинен підтримувати різні типи схем, такі як блок-схеми, дерева рішень, графи та інші візуальні представлення алгоритмів.

Для досягнення поставленої мети буде використано методи аналізу, проектування та програмної реалізації. Аналіз включає в себе огляд існуючих рішень та визначення вимог до веб-ресурсу. Проектування охоплює розробку архітектури системи та інтерфейсу користувача. Програмна реалізація включає в себе розробку функціональності веб-ресурсу з використанням сучасних технологій та інструментів.

Аналіз існуючих рішень дозволить визначити, які підходи вже використовуються для візуалізації алгоритмів, та ідентифікувати їх сильні та слабкі сторони. Це допоможе уникнути помилок та недоліків, які можуть бути притаманні іншим системам, і розробити більш ефективний та зручний веб-ресурс.

Проектування архітектури системи включатиме вибір відповідних технологій, розробку структури даних та алгоритмів, які будуть використовуватися для візуалізації. Особлива увага буде приділена інтерактивності, оскільки це ключовий аспект для забезпечення користувачів можливістю взаємодіяти з алгоритмами в режимі реального часу.

Програмна реалізація включатиме в себе написання коду для фронтенд та бекенд частин веб-ресурсу. Фронтенд буде відповідати за відображення інформації та взаємодію з користувачем, тоді як бекенд забезпечуватиме обробку даних та зберігання інформації.

Для тестування веб-ресурсу будуть використані методи юніт-тестування, інтеграційного тестування та функціонального тестування, що дозволить забезпечити високу якість та надійність програмного забезпечення. Юніт-тестування допоможе перевірити окремі частини коду, інтеграційне тестування дозволить впевнитися в коректній взаємодії між різними модулями системи, а функціональне тестування гарантує, що веб-ресурс виконує всі необхідні функції відповідно до вимог.

В результаті виконання даної роботи очікується створення веб-ресурсу, який дозволить користувачам візуалізувати та взаємодіяти з різними алгоритмами та схемами. Цей ресурс повинен бути зручним у використанні, забезпечуючи високу продуктивність та безпеку даних. Очікується, що веб-ресурс буде корисним для студентів, викладачів та дослідників, надаючи їм інструмент для більш глибокого розуміння та аналізу алгоритмів.

## 1.2 Аналіз існуючих рішень в області створення веб-ресурсу для візуалізації алгоритмів та схем

На сьогоднішній день існує багато різноманітних інструментів та платформ для візуалізації алгоритмів та схем. Ці інструменти варіюються за функціональністю, цільовою аудиторією, використовуваними технологіями та рівнем інтерактивності. Вони можуть бути орієнтовані як на освітні потреби, так і на професійне використання в різних галузях науки та техніки. Розгляд існуючих рішень допоможе нам зрозуміти, які функції та особливості є найбільш важливими та корисними для користувачів, а також які підходи використовуються для ефективної візуалізації складних алгоритмів та процесів.

**Miro**

Miro – це інтерактивна онлайн-дошка для спільної роботи, яка дозволяє користувачам створювати різноманітні схеми, діаграми та візуалізації в реальному часі. Miro є чудовим прикладом інструменту, який забезпечує зручність використання та широкий функціонал для візуалізації алгоритмів та схем.

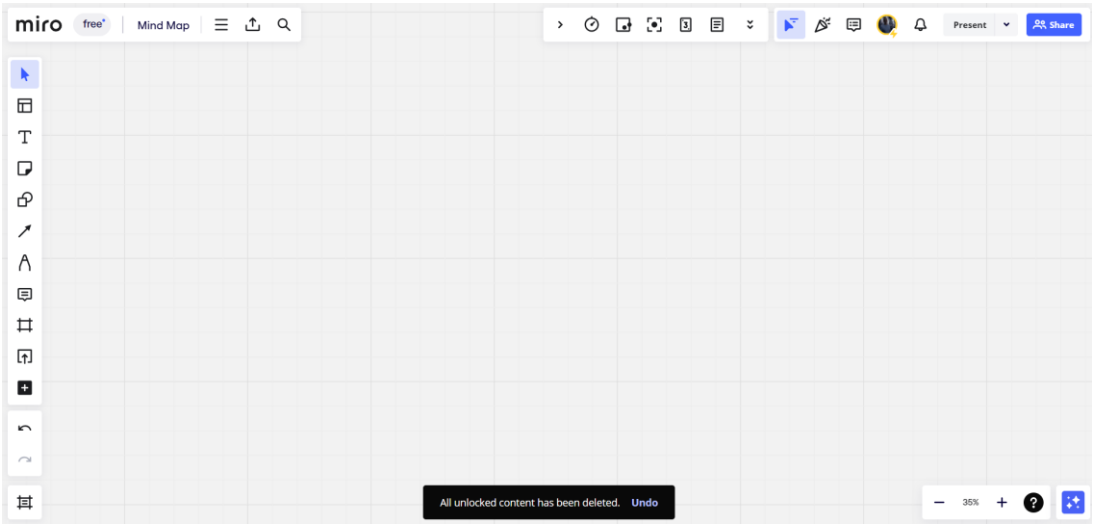


Рисунок 1.1 – інтерфейс Miro

Особливості:

- Miro дозволяє користувачам створювати та редагувати схеми одночасно з іншими учасниками, що сприяє ефективній командній роботі.
- Інструмент підтримує широкий спектр шаблонів та інструментів для створення блок-схем, діаграм, графів та інших типів візуалізацій.
- Miro легко інтегрується з іншими популярними сервісами та інструментами, такими як Google Drive, Slack, Jira та інші.

Переваги:

- Інтерфейс Miro інтуїтивно зрозумілий та простий у використанні, що дозволяє швидко створювати та редагувати візуалізації.
- Можливість спільної роботи в реальному часі робить Miro ідеальним інструментом для командних проєктів та навчання.
- Наявність великої кількості шаблонів та інструментів дозволяє створювати різноманітні типи візуалізацій, задовольняючи потреби різних користувачів.

Недоліки:

- Повний доступ до функціоналу Miro вимагає платної підписки, що може бути недосяжним для деяких користувачів.
- Для роботи з Miro необхідне стабільне інтернет-з'єднання, що може бути незручним у деяких ситуаціях.

**Lucidchart**

Lucidchart – це ще один популярний інструмент для створення схем, діаграм та візуалізацій, який широко використовується як в освітніх, так і в професійних цілях. Він надає зручний інтерфейс для створення різних типів візуалізацій, включаючи блок-схеми, організаційні діаграми, UML-діаграми та інші.

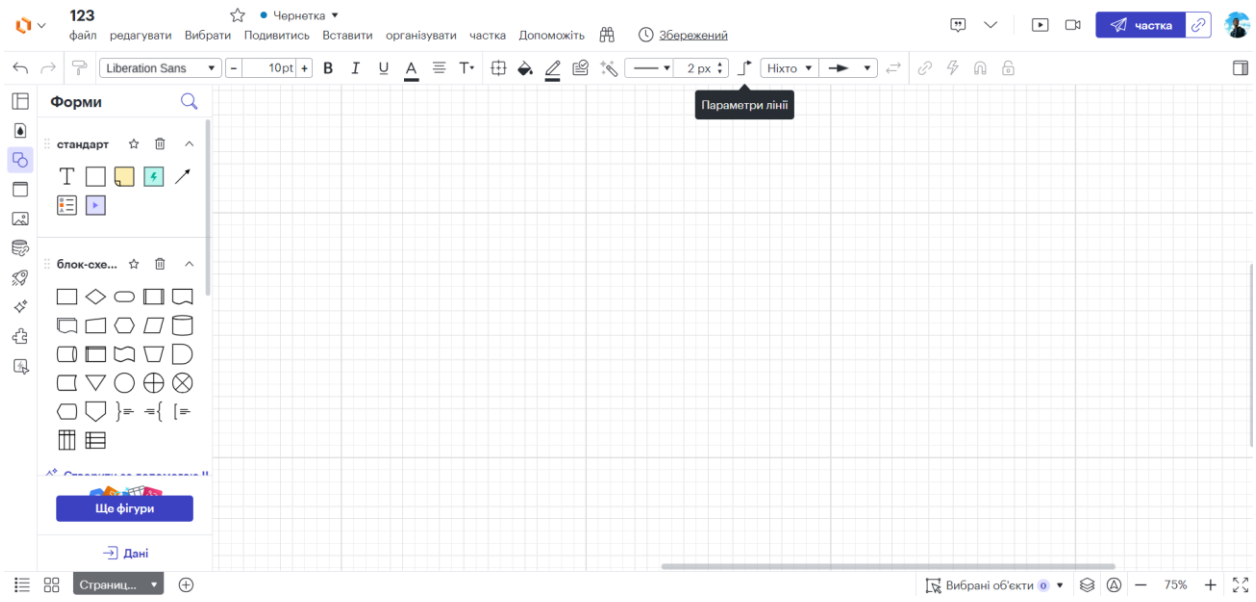


Рисунок 1.2 – інтерфейс Lucidchart

Особливості:

- Lucidchart підтримує створення різних типів діаграм та схем, що робить його універсальним інструментом для візуалізації.
- Користувачі можуть працювати над схемами разом в реальному часі, обмінюватися коментарями та редагувати візуалізації.
- Інструмент інтегрується з популярними сервісами, такими як Google Drive, Microsoft Office, Slack та іншими.

Переваги:

- Інтерфейс Lucidchart зручний та інтуїтивно зрозумілий, що дозволяє швидко створювати візуалізації.
- Велика кількість шаблонів та прикладів допомагає користувачам швидко розпочати роботу.
- Інструмент забезпечує високу безпеку даних, що важливо для корпоративних користувачів.

Недоліки:

- Як і у випадку з Miro, повний доступ до функціоналу Lucidchart вимагає платної підписки.
- Для роботи з Lucidchart потрібне стабільне інтернет-з'єднання.

**Creately**

Creately – це ще один популярний онлайн-інструмент для створення діаграм та візуалізацій, який використовується для роботи з блок-схемами, діаграмами Ганта, картами думок та іншими типами візуалізацій.

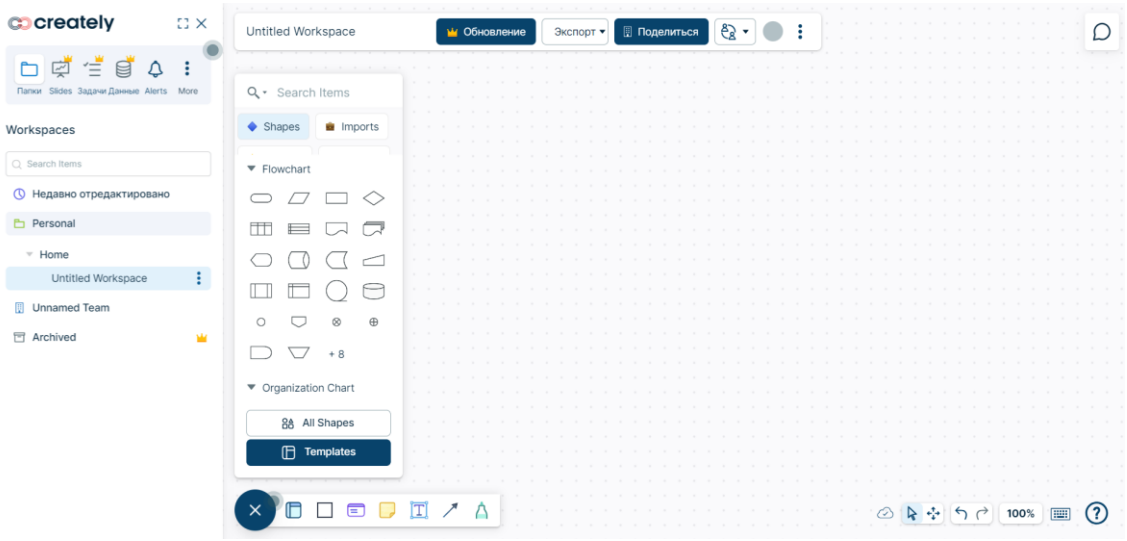


Рисунок 1.3 – інтерфейс Creately

Особливості:

- Creately підтримує одночасну роботу кількох користувачів над однією діаграмою.
- Інструмент пропонує широкий вибір шаблонів та прикладів для різних типів візуалізацій.

- Creately інтегрується з популярними інструментами, такими як Google Drive, Jira, Confluence та іншими.

Переваги:

- Зручний інтерфейс дозволяє легко створювати та редагувати візуалізації.
- Creately пропонує великий набір інструментів для створення різних типів діаграм.
- Інструмент доступний як у вигляді веб-додатку, так і в мобільних додатках.

Недоліки:

- Повний доступ до всіх функцій Creately вимагає платної підписки.
- Для роботи з Creately потрібне стабільне інтернет-з'єднання.

**Cacoo**

Cacoo – це онлайн-інструмент для створення діаграм та візуалізацій, який також підтримує спільну роботу. Він дозволяє створювати різні типи діаграм, включаючи блок-схеми, діаграми баз даних, сіткові діаграми та інші.

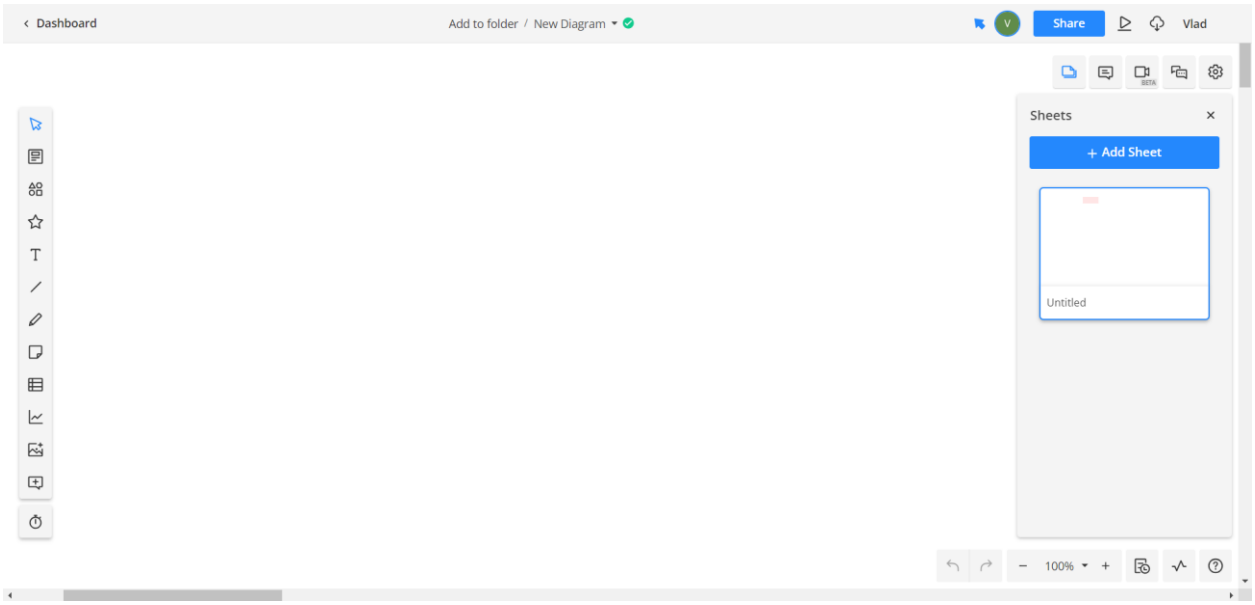


Рисунок 1.4 – інтерфейс Cacoo

Особливості:

- Cacoo дозволяє кільком користувачам одночасно працювати над однією діаграмою.

- Інструмент інтегрується з такими сервісами, як Google Drive, Dropbox, Slack та іншими.
- Cacoo пропонує широкий вибір шаблонів для різних типів діаграм.

Переваги:

- Простота використання: Інтерфейс Cacoo зручний та інтуїтивно зрозумілий.
- Користувачі можуть одночасно редагувати діаграми та обмінюватися коментарями.
- Велика кількість шаблонів та інструментів дозволяє створювати різноманітні типи візуалізацій.

Недоліки:

- Повний доступ до функціоналу Cacoo вимагає платної підписки.
- Для роботи з Cacoo потрібне стабільне інтернет-з'єднання.

Огляд існуючих рішень для візуалізації алгоритмів та схем показує, що кожен інструмент має свої переваги та недоліки. Вибір конкретного інструменту залежить від цілей та вимог користувача. Miro, Lucidchart, Creately та Cacoo є чудовими прикладами інструментів для командної роботи та створення різноманітних візуалізацій. Вони забезпечують зручність використання, інтерактивність та широкий функціонал, що робить їх ідеальними для професійного та освітнього використання.

При розробці власного веб-ресурсу для візуалізації алгоритмів та схем необхідно враховувати найкращі практики та досвід існуючих рішень. Це дозволить створити ефективний та зручний інструмент, який буде корисним для широкої аудиторії.

## 1.3 Формулювання вимог до веб-ресурсу для візуалізації алгоритмів та схем

Аналіз потреб та вимог до веб-ресурсу для візуалізації алгоритмів та схем є критично важливим етапом у процесі його розробки. Цей етап дозволяє

зрозуміти, які функції та характеристики повинен мати веб-ресурс, щоб задовольнити очікування та потреби користувачів. Збір та аналіз потреб користувачів допомагають визначити функціональні та нефункціональні вимоги, які забезпечать зручність використання, ефективність та надійність ресурсу.

**Визначення цільової аудиторії**

Першим кроком у процесі аналізу потреб є визначення цільової аудиторії. Це важливо, оскільки різні групи користувачів можуть мати різні вимоги та очікування щодо веб-ресурсу.


Рисунок 1.5 – портрет цільової аудиторії

Основними користувачами веб-ресурсу для візуалізації алгоритмів та схем можуть бути:

1. Студенти та викладачі, тобто веб-ресурс може використовуватися в навчальних закладах для викладання та вивчення алгоритмів. Студенти можуть використовувати ресурс для самостійного навчання, а викладачі – для демонстрації алгоритмів у навчальному процесі.
2. Дослідники та науковці можуть використовувати застосунок для аналізу та тестування нових алгоритмів, а також для публікації результатів своїх досліджень у зручному форматі.

3. Розробники програмного забезпечення використовують для розуміння та оптимізації алгоритмів у своїх проєктах, а також для демонстрації роботи алгоритмів своїм клієнтам або колегам.
4. Люди, які хочуть покращити свої знання про алгоритми та їхню візуалізацію, можуть використовувати ресурс для самостійного навчання та експериментів.

**Функціональні вимоги:**

1. Веб-ресурс повинен забезпечувати можливість візуалізації різних типів алгоритмів, включаючи сортування, пошук, графові алгоритми та інші.
2. Користувачі повинні мати можливість взаємодіяти з алгоритмами, змінюючи їхні параметри та спостерігаючи за змінами в реальному часі.
3. Веб-ресурс повинен підтримувати різні типи схем, такі як блок-схеми, дерева рішень, графи тощо.
4. Важливо забезпечити можливість спільної роботи, де кілька користувачів можуть одночасно працювати над однією схемою або алгоритмом, обмінюючись коментарями та редагуючи в реальному часі.
5. Користувачі повинні мати можливість зберігати свої проекти та завантажувати їх для подальшого редагування.

**Нефункціональні вимоги:**


Рисунок 1.6 – нефукціональні вимоги

1. Веб-ресурс повинен забезпечувати швидке виконання алгоритмів та оперативний відгук на дії користувачів, навіть при обробці великих обсягів даних.
2. Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким у використанні, щоб забезпечити позитивний користувацький досвід.
3. Важливо забезпечити безпеку даних користувачів, включаючи захист від несанкціонованого доступу та втрати даних. Це включає реалізацію механізмів аутентифікації та авторизації.
4. Ресурс повинен бути масштабованим, щоб забезпечити можливість обробки збільшеної кількості користувачів та даних без втрати продуктивності.
5. Веб-ресурс повинен бути доступним з різних пристроїв та браузерів, забезпечуючи коректну роботу на мобільних пристроях та настільних комп'ютерах.

**Збір та аналіз вимог користувачів**

Важливо отримати зворотний зв'язок від потенційних користувачів на ранніх етапах розробки, щоб врахувати їхні потреби та очікування. Для збору вимог користувачів можна використовувати різні методи, такі як опитування, інтерв'ю, фокус-групи та аналіз існуючих рішень:

1. Проведення опитувань серед потенційних користувачів допоможе зібрати інформацію про їхні потреби та очікування щодо веб-ресурсу.
2. Інтерв'ю з представниками цільової аудиторії дозволять отримати детальну інформацію про специфічні вимоги та побажання користувачів.
3. Проведення фокус-груп допоможе виявити спільні проблеми та потреби користувачів, а також обговорити можливі рішення.
4. Вивчення та аналіз існуючих інструментів для візуалізації алгоритмів дозволить виявити їх сильні та слабкі сторони, що допоможе уникнути помилок та недоліків у власному веб-ресурсі.

**Аналіз зібраних вимог**

Після збору вимог необхідно провести їх аналіз, щоб визначити ключові функції та пріоритети. Це дозволить створити веб-ресурс, який відповідатиме потребам користувачів та забезпечуватиме їм необхідні інструменти для візуалізації алгоритмів та схем.

Етапи аналізу:

1. Розподілити вимоги на функціональні та нефункціональні для більш зручного управління.
2. Визначити, які вимоги є найбільш важливими для цільової аудиторії та повинні бути реалізовані в першу чергу.
3. Провести оцінку можливостей та складності реалізації кожної вимоги, щоб скласти план розробки.

Аналіз потреб та вимог до веб-ресурсу для візуалізації алгоритмів та схем є важливим кроком у його розробці. Визначення цільової аудиторії, функціональних та нефункціональних вимог дозволяє створити ресурс, який буде зручним та ефективним у використанні. Збір зворотного зв'язку від користувачів та врахування їхніх потреб допоможе забезпечити успішне впровадження веб-ресурсу та його подальший розвиток. Забезпечення високої продуктивності, безпеки, масштабованості та інтеграції з іншими сервісами є ключовими факторами для створення конкурентоспроможного та корисного інструменту.

## 1.4 Висновок

У першому розділі було проведено всебічний аналіз для розробки веб-ресурсу, призначеного для візуалізації алгоритмів та схем. На початку ми визначили основні задачі, встановили мету створення ресурсу, яка полягає в наданні користувачам інструментів для інтерактивної візуалізації та покрокового розуміння алгоритмів.

Проаналізувавши існуючі рішення, такі як Miro, Lucidchart, Creately та Cacoo, ми виділили їхні переваги та недоліки, що дозволило нам зрозуміти, які

функціональні можливості і підходи є найбільш ефективними. Це включає інтерактивність, спільну роботу та інтеграцію з іншими сервісами.

Подальший аналіз потреб та вимог користувачів дозволив сформувати чітке уявлення про необхідні функціональні та нефункціональні вимоги до нашого веб-ресурсу. Це включає підтримку різних типів алгоритмів та схем, забезпечення продуктивності, зручності використання, безпеки та масштабованості.

Загалом, проведений аналіз допоміг закласти міцну основу для розробки веб-ресурсу, який буде відповідати очікуванням користувачів та забезпечувати їм ефективні інструменти для вивчення та взаємодії з алгоритмами. Подальші етапи роботи включатимуть проектування та програмну реалізацію ресурсу з урахуванням визначених вимог.

# 2 ПРОЕКТУВАННЯ ВЕБ-РЕСУРСУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СХЕМ

## 2.1 Архітектура веб-ресурсу для візуалізації алгоритмів та схем

У розробці веб-додатків існує кілька типових архітектур, які мають свої особливості, переваги та недоліки. Основні з них включають:

**Монолітна архітектура**

Монолітна архітектура є традиційним підходом до розробки додатків, де всі компоненти, такі як користувацький інтерфейс, бізнес-логіка та дані, інтегровані в один єдиний додаток. Такий підхід простий у реалізації для невеликих проектів, оскільки всі функції зосереджені в одному місці, що полегшує розробку та розгортання. Однак зі зростанням проекту монолітна архітектура стає важко масштабованою, а її підтримка та оновлення ускладнюються через високу зв'язаність компонентів.

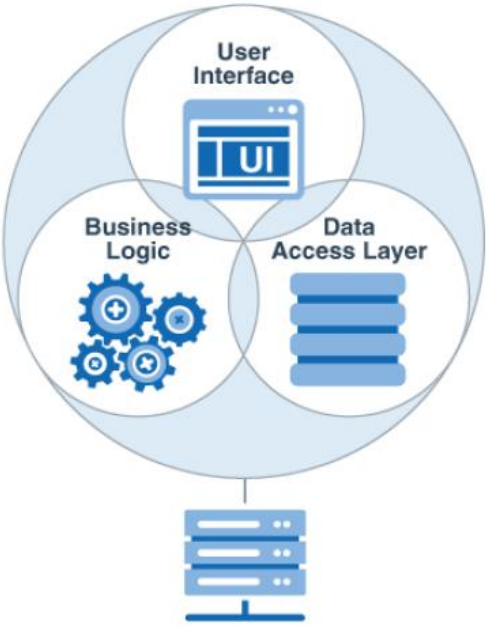


Рисунок 2.1 – схема монолітної архітектури

**Клієнт-серверна архітектура**

Клієнт-серверна архітектура розділяє компоненти додатку на клієнтську та серверну частини. Клієнтська частина відповідає за взаємодію з користувачем, забезпечуючи інтерфейс та обробку введених даних. Серверна частина відповідає за обробку запитів, управління даними та виконання

бізнес-логіки. Такий підхід дозволяє легко масштабувати додаток, оскільки клієнт і сервер можуть розвиватися незалежно один від одного. Це також забезпечує гнучкість у підтримці та розширенні функціональності.

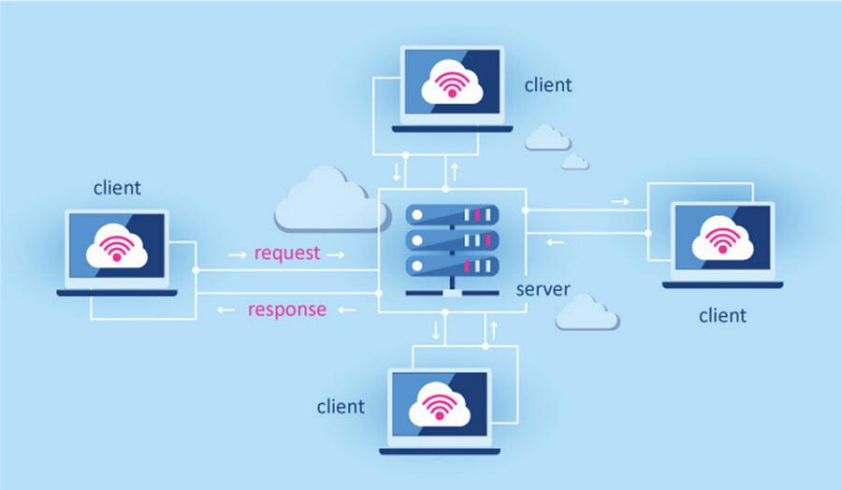


Рисунок 2.2 – схема клієнт-серверної архітектури

**Трирівнева архітектура**

Трирівнева архітектура включає три окремі рівні, кожен з яких виконує свою окрему роль в системі:

- Рівень представлення (Presentation layer), який відповідає за взаємодію з користувачем, відображення інтерфейсу та обробку введених даних, використовуючи технології HTML, CSS, JavaScript, фреймворки як-от Vue, React тощо, забезпечуючи інтерфейс користувача і передаючи дані до рівня логіки;
- Рівень логіки (Application layer), який відповідає за обробку бізнес-логіки додатку, обробляє запити від клієнтської частини, виконання бізнес-правил та управління даними, містить API, веб-сервіси та інші сервіси для обробки даних;
- Рівень даних (Data layer), який відповідає за управління даними додатку, включає базу даних, систему управління базами даних (DBMS) та інші сервіси для зберігання та обробки даних, використовуючи технології як-от SQL, NoSQL, ORM (Object-Relational Mapping) тощо.


Рисунок 2.3 – схема трирівневої архітектури

**Мікросервісна архітектура**

Мікросервісна архітектура розбиває додаток на набір невеликих, незалежних сервісів, кожен з яких виконує певну функцію. Кожен сервіс може розгортатися і масштабуватися окремо, що забезпечує високу гнучкість та масштабованість системи. Цей підхід дозволяє командам працювати незалежно один від одного над різними сервісами, що прискорює розробку та впровадження нових функцій. Однак, мікросервісна архітектура може бути складнішою у початковій розробці та управлінні через необхідність оркестрації та забезпечення взаємодії між сервісами.

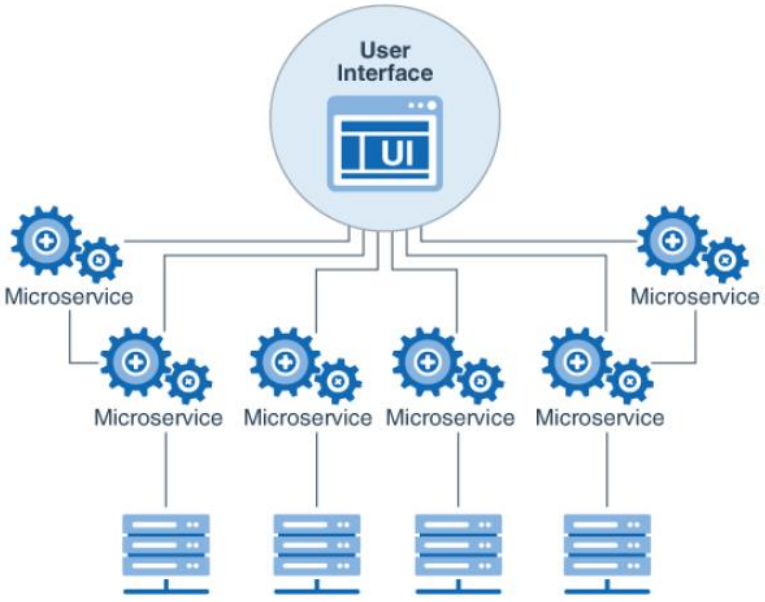


Рисунок 2.4 – схема мікросервісної архітектури

**Архітектура без сервера (Serverless)**

Архітектура без сервера використовує функції як сервіс (FaaS), які запускаються на вимогу і виконують окремі задачі. Це дозволяє розробникам зосередитися на бізнес-логіці без необхідності управління інфраструктурою. Serverless підхід забезпечує ефективне використання ресурсів, оскільки функції виконуються тільки при необхідності. Це може бути вигідним для проектів з нерівномірним навантаженням. Однак, налагодження та тестування таких додатків можуть бути складнішими через розподілену природу функцій.

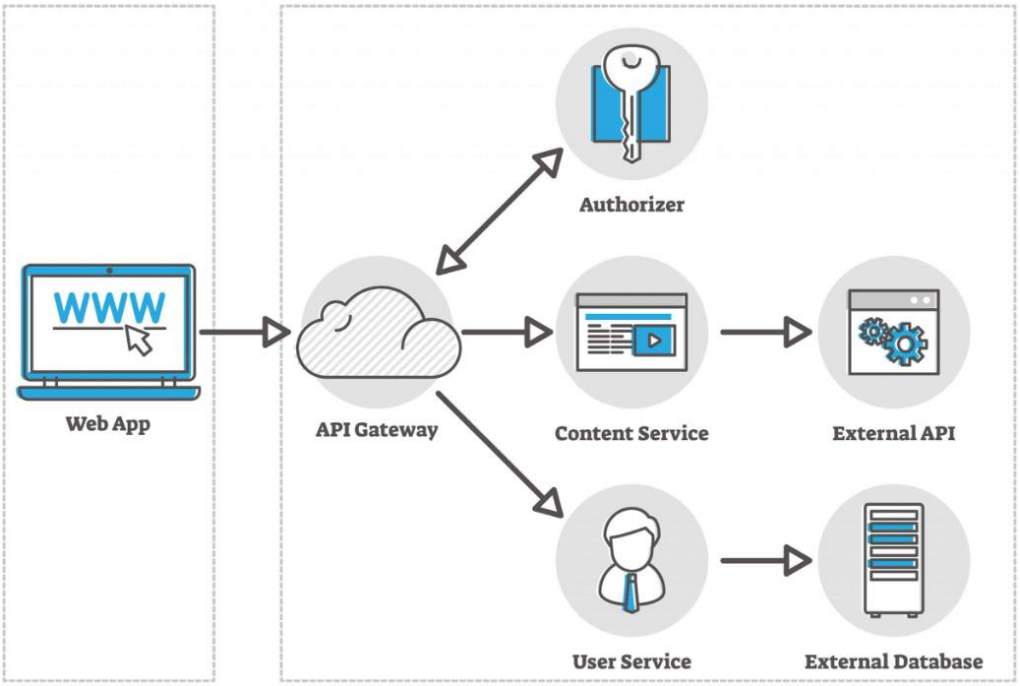


Рисунок 2.5 – схема архітектури без сервера

**Вибір архітектури для проекту**

Для нашого проекту ми обрали клієнт-серверну архітектуру. Цей вибір обумовлений наступними причинами:

1. Клієнтська частина проекту відповідає за взаємодію з користувачем, забезпечуючи інтерактивність та зручність використання. Вона включає в себе такі компоненти, як інтерактивна дошка, панель інструментів, функціональність шарів, система кольорів, функціонал "Скасувати" та "Повторити", гарячі клавіші, функціонал улюблених елементів, реалізація інтерфейсу користувача та управління станом додатку. Це дозволяє

використовувати сучасні технології, такі як React, TailwindCSS, ShadcnUI та TypeScript, для створення ефективного та адаптивного інтерфейсу.

2. Клієнт-серверна архітектура дозволяє легко масштабувати додаток. Серверна частина обробляє запити клієнта, управляє базою даних та забезпечує синхронізацію в реальному часі. Використання Liveblocks для реальної синхронізації даних та Convex для бази даних забезпечує надійне зберігання даних та їх синхронізацію. Це забезпечує високу продуктивність та можливість розширення системи в майбутньому.

3. Клієнтська та серверна частини можуть розроблятися та оновлюватися незалежно одна від одної, що забезпечує гнучкість у розширенні функціональності та адаптації додатку до змінних вимог користувачів. Такий підхід також полегшує тестування та налагодження окремих компонентів системи.

4. Використання Clerk для аутентифікації та управління організаціями забезпечує високий рівень безпеки та контроль доступу до додатку. Це критично важливо для захисту даних користувачів та забезпечення надійності системи.

5. Liveblocks що відповідає за синхронізацію даних у реальному часі дозволяє кільком користувачам одночасно працювати над однією схемою або алгоритмом, що підвищує ефективність командної роботи та забезпечує безперервність робочого процесу.

Вибір клієнт-серверної архітектури для вашого проекту забезпечує оптимальний баланс між гнучкістю, масштабованістю, безпекою та зручністю використання. Такий підхід дозволяє ефективно розробляти, підтримувати та розширювати функціональність вашого веб-додатку, відповідаючи на потреби користувачів та адаптуючись до змінних умов ринку.

## 2.2 Вибір технологій проектування

Вибір технологій є одним з ключових етапів у розробці веб-застосунку для візуалізації алгоритмів та схем. Від правильного вибору технологічного стеку залежить не тільки функціональність і продуктивність, але й зручність

використання, масштабованість та безпека застосунку. У цьому розділі буде розглянуто аналіз технологічних стеків, а також обґрунтування вибору конкретних технологій, які найкраще відповідають вимогам проекту.

### 2.2.1 Аналіз технологічних стеків

Перш ніж зупинитися на конкретних технологіях, ми розглянули кілька популярних фреймворків та інструментів для розробки веб-застосунків, які забезпечують необхідну функціональність та продуктивність. Серед них були React, Angular та Vue.js для фронтенду, а також Node.js, Django та Ruby on Rails для бекенду.

React був обраний через його компонентну архітектуру, яка забезпечує високу продуктивність та гнучкість. React має велику спільноту підтримки та багатий екосистемний набір інструментів, що робить його одним з найпопулярніших фреймворків для розробки користувацьких інтерфейсів. Відмінною особливістю React є можливість легкого інтегрування з іншими інструментами та фреймворками, що підвищує його гнучкість і розширюваність.

Angular був розглянутий як альтернатива завдяки своїй повноцінній архітектурі та багатству вбудованих функцій. Angular пропонує двостороннє зв'язування даних, вбудовані інструменти для тестування та потужну систему модулів, що дозволяє розробляти складні веб-додатки. Однак, його освоєння може бути більш складним і тривалим порівняно з React, що може стати перешкодою для швидкого старту проекту.

Vue.js також був розглянутий як можливий варіант завдяки своїй легкості та гнучкості. Vue.js поєднує найкращі властивості React та Angular, пропонуючи зручний синтаксис та високу продуктивність. Vue.js має зрозумілу документацію та активну спільноту, що робить його привабливим вибором для багатьох розробників. Однак, ми вирішили зупинитися на React через його ширшу популярність та кращу інтеграцію з іншими інструментами.

Для бекенд-розробки було розглянуто кілька технологій, включаючи Node.js, Django та Ruby on Rails.

Node.js був обраний завдяки своїй асинхронній архітектурі, яка дозволяє ефективно обробляти велику кількість запитів одночасно. Це робить його ідеальним вибором для реальних часів додатків, що потребують швидкого відгуку. Node.js забезпечує високу продуктивність та масштабованість, що особливо важливо для веб-додатків з високим навантаженням.

Django надає багато готових рішень для типових задач, таких як аутентифікація користувачів та управління базою даних, що значно скорочує час розробки. Однак, його монолітна архітектура може бути менш гнучкою порівняно з Node.js, особливо для проектів, що потребують високої масштабованості та адаптивності.

Ruby on Rails був розглянутий завдяки своїй швидкості розробки та спрощенню створення веб-додатків. Rails забезпечує високу продуктивність та гнучкість, пропонуючи багато вбудованих інструментів для розробки. Однак, популярність Rails останніми роками дещо знизилась у порівнянні з іншими фреймворками, і його екосистема не така велика, як у Node.js.

Для стилізації інтерфейсу ми обрали TailwindCSS та ShadcnUI. TailwindCSS забезпечує утилітарний підхід до стилізації, що дозволяє швидко створювати адаптивні та добре структуровані стилі без необхідності написання великої кількості власного CSS-коду. Це дозволяє зосередитися на функціональності додатку, не витрачаючи багато часу на стилізацію. TailwindCSS також надає можливість легкого налаштування та розширення стилів, що дозволяє створювати унікальні дизайни, що відповідають специфічним вимогам проекту. ShadcnUI додає додаткові компоненти для користувацького інтерфейсу, що допомагає створювати елегантні та функціональні інтерфейси. Використання цих технологій разом забезпечує швидкість розробки та високу якість кінцевого продукту.

Щоб забезпечити можливості співпраці в реальному часі було обрано Liveblocks. Цей сервіс дозволяє створювати спільні робочі середовища, де кілька користувачів можуть одночасно працювати над одним документом або проектом, забезпечуючи синхронізацію даних у реальному часі. Реальний час

співпраці дозволяє користувачам бачити зміни, внесені іншими учасниками, у режимі реального часу, що підвищує ефективність роботи та забезпечує безперервність робочого процесу.

### 2.2.2 Обґрунтування вибору компонентної технології проектування

На основі проведеного аналізу було обрано наступний технологічний стек для розробки веб-застосунку для візуалізації алгоритмів та схем:
Next.js був обраний через його інтеграцію з React, забезпечення серверного рендерингу та генерації статичних сайтів. Це покращує продуктивність додатку та його SEO-оптимізацію. Крім того, Next.js надає високу гнучкість для інтеграції з іншими технологіями. Серверний рендеринг дозволяє зменшити час завантаження сторінок та покращити користувацький досвід, що є критично важливим для сучасних веб-додатків. Додатково, Next.js підтримує такі функції, як автоматичне розбиття коду та інкрементальна статична генерація, що робить його ідеальним вибором для розробки високопродуктивних та масштабованих додатків.

TailwindCSS та ShadcnUI були обрані для стилізації інтерфейсу завдяки їхньому утилітарному підходу та гнучкості. TailwindCSS дозволяє швидко створювати адаптивні та добре структуровані стилі, що значно скорочує час розробки. ShadcnUI додає додаткові компоненти, що допомагають створювати елегантні та функціональні інтерфейси. Використання цих технологій разом забезпечує високу якість кінцевого продукту та зручність у використанні.

Node.js з використанням Convex був обраний для бекенд-розробки завдяки своїй асинхронній архітектурі та високій продуктивності. Node.js дозволяє ефективно обробляти велику кількість запитів одночасно, що робить його ідеальним для реальних часів додатків. Convex спрощує інтеграцію з різними сервісами зберігання даних, забезпечуючи надійність обробки даних у реальному часі. Це дозволяє створювати масштабовані та гнучкі рішення, що відповідають потребам проекту.

Clerk був обраний для аутентифікації користувачів завдяки своїй потужній системі управління користувачами. Clerk забезпечує надійну

аутентифікацію, управління організаціями та запрошеннями, що дозволяє легко керувати доступом та забезпечити безпеку даних користувачів. Використання Clerk дозволяє швидко реалізувати систему управління користувачами, що включає підтримку різних методів аутентифікації, двофакторну аутентифікацію та управління сесіями користувачів.

Liveblocks був обраний для забезпечення можливості реального часу співпраці завдяки своїй здатності створювати спільні робочі середовища. Liveblocks дозволяє кільком користувачам одночасно працювати над одним документом або проектом, забезпечуючи синхронізацію даних у реальному часі. Це підвищує ефективність роботи та забезпечує безперервність робочого процесу, що є критично важливим для нашого проекту.

Вибір цих технологій обумовлений необхідністю створення сучасного, швидкого та гнучкого веб-застосунку, який задовольняє потреби користувачів у ефективній візуалізації алгоритмів та схем. Використання Next.js, TailwindCSS, ShadcnUI, Node.js з Convex, Clerk та Liveblocks забезпечує високий рівень інтерактивності, зручності використання та надійності, що є ключовими аспектами для успішної реалізації проекту.

## 2.3 Проектування інтерфейсу користувача веб-ресурсу для візуалізації алгоритмів та схем

Створення інтерфейсу користувача є важливим етапом у розробці веб-застосунку для візуалізації алгоритмів та схем. Від того, наскільки зручним і інтуїтивно зрозумілим буде інтерфейс, залежить досвід користувачів і загальна ефективність роботи з застосунком. У цьому розділі розглянемо основні принципи UX/UI дизайну та процес створення прототипів.

### 2.3.1 Принципи UX/UI дизайну

При розробці інтерфейсу користувача важливо дотримуватися основних принципів UX/UI дизайну, щоб забезпечити зручність використання та позитивний користувацький досвід.

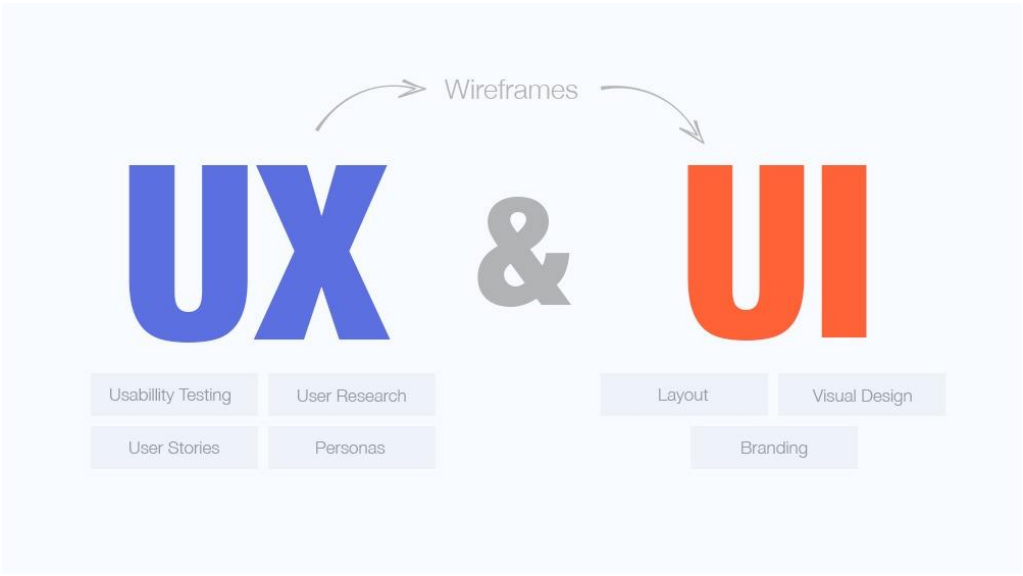


Рисунок 2.6 – принципи UX/UI дизайну

Зрозумілість і простота є ключовими аспектами успішного дизайну інтерфейсу. Інтерфейс повинен бути інтуїтивно зрозумілим, щоб користувачі могли легко знаходити необхідні функції та виконувати завдання. Використання простих і зрозумілих елементів управління знижує криву навчання і дозволяє швидше освоїти застосунок. Консистентність дизайну забезпечує однаковий вигляд і поведінку елементів на всіх сторінках застосунку. Це допомагає користувачам створити ментальну модель інтерфейсу, що покращує зручність використання і зменшує ймовірність помилок.

Візуальна ієрархія відіграє важливу роль у дизайні інтерфейсу, допомагаючи користувачам швидко орієнтуватися на сторінці. Важливо правильно розташовувати елементи, щоб вони привертали увагу користувача відповідно до їх важливості. Використання розміру, кольору та контрасту допомагає створити чітку візуальну ієрархію, яка сприяє легкому сприйняттю інформації.

Доступність є ще одним важливим аспектом UX/UI дизайну. Інтерфейс повинен бути доступним для всіх користувачів, включаючи людей з обмеженими можливостями. Це включає використання відповідних кольорів, шрифтів, а також підтримку навігації з клавіатури та екранних читалок.

Забезпечення доступності покращує загальну зручність використання і робить застосунок більш інклюзивним.

Зворотний зв'язок користувачів є критичним для успішного дизайну інтерфейсу. Користувачі повинні отримувати миттєвий зворотний зв'язок на свої дії. Це може бути у формі повідомлень про успішне виконання завдання, попереджень про помилки або підказок щодо подальших кроків. Зворотний зв'язок допомагає користувачам зрозуміти, що їх дії були сприйняті системою і що відбувається далі.

Мінімізація когнітивного навантаження є важливим принципом, що забезпечує легкість використання інтерфейсу. Інтерфейс повинен бути спроєктований таким чином, щоб мінімізувати кількість інформації, яку користувачі повинні обробляти одночасно. Використання простих і зрозумілих елементів допомагає зменшити когнітивне навантаження і полегшує взаємодію з системою.

Адаптивність інтерфейсу є необхідною для забезпечення зручності роботи на різних пристроях і розмірах екранів. Інтерфейс повинен коректно відображатися як на настільних комп'ютерах, так і на мобільних пристроях. Використання адаптивного дизайну дозволяє забезпечити позитивний користувацький досвід незалежно від того, який пристрій використовується.

### 2.3.2 Створення прототипів

Створення прототипів є важливим кроком у процесі розробки інтерфейсу користувача. Прототипи дозволяють візуалізувати та тестувати ідеї до того, як вони будуть реалізовані у фінальному продукті. Це допомагає уникнути потенційних проблем та покращити користувацький досвід на ранніх етапах розробки.

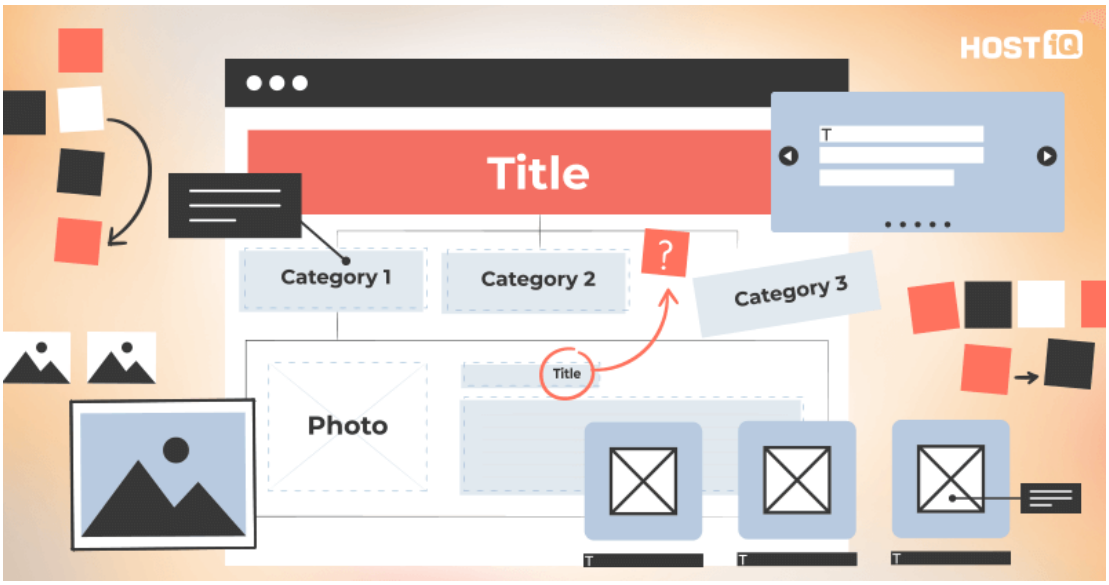


Рисунок 2.7 – створення прототипу сайту

На початковому етапі створення прототипів важливо розробити скетчі та вайрфрейми. Скетчі - це прості ручні малюнки, які допомагають швидко візуалізувати основні ідеї та компонування інтерфейсу. Вони дозволяють експериментувати з різними макетами та структурами, не витрачаючи багато часу на деталі. Вайрфрейми, навпаки, є більш деталізованими ілюстраціями, які показують структуру сторінок та розташування ключових елементів. Вони допомагають визначити, як різні частини інтерфейсу взаємодіють між собою і як користувачі будуть взаємодіяти з ними.

Після створення вайрфреймів, наступним кроком є розробка інтерактивних прототипів. Інтерактивні прототипи дозволяють користувачам взаємодіяти з інтерфейсом, що допомагає краще зрозуміти його функціональність та зручність використання. Інструменти, такі як Figma, Adobe XD або InVision, дозволяють створювати інтерактивні прототипи, які імітують поведінку реального додатку. Це допомагає виявити можливі проблеми та отримати зворотний зв'язок від користувачів ще до початку розробки.

Тестування прототипів з користувачами є критичним етапом у процесі створення інтерфейсу. Важливо проводити тестування з реальними користувачами, щоб отримати зворотний зв'язок і виявити можливі проблеми. Тестування допомагає зрозуміти, як користувачі взаємодіють з інтерфейсом,

які у них виникають труднощі та які аспекти потребують покращення. Це може бути як модераторське тестування, де фасилітатор проводить користувачів через конкретні завдання, так і віддалене тестування, де користувачі взаємодіють з прототипом самостійно.

На основі зворотного зв'язку від тестування необхідно вносити зміни в прототипи та повторювати процес тестування. Ітераційний підхід дозволяє поступово покращувати інтерфейс, враховуючи потреби та очікування користувачів. Кожна ітерація повинна фокусуватися на вирішенні виявлених проблем та вдосконаленні користувацького досвіду. Цей процес може включати кілька раундів тестування та внесення змін, поки не буде досягнуто оптимального дизайну.

Після кількох ітерацій тестування та внесення змін можна перейти до створення високо-деталізованих прототипів, які максимально наближені до фінального дизайну. Вони включають всі візуальні елементи, кольори, шрифти та інтерактивні елементи. Високо-деталізовані прототипи допомагають краще зрозуміти, як буде виглядати та працювати кінцевий продукт, і дозволяють виявити останні недоліки перед початком розробки.

Останнім етапом є документування дизайну, що включає створення стилістичних гайдлайнів та документації, яка детально описує всі елементи інтерфейсу, їх взаємодію та правила використання. Це допомагає забезпечити консистентність дизайну при реалізації та подальшому розширенні продукту. Документування також сприяє комунікації між дизайнерами та розробниками, забезпечуючи чітке розуміння вимог та очікувань щодо інтерфейсу.

Розробка інтерфейсу користувача є складним і багатокроковим процесом, який вимагає ретельного планування, тестування та ітераційного вдосконалення. Використання принципів UX/UI дизайну та створення прототипів допомагає створити інтерфейс, який буде зручним, інтуїтивно зрозумілим та ефективним для користувачів. Це забезпечує високий рівень задоволеності користувачів та успішність веб-застосунку в цілому.

## 2.4 Основні модулі та компоненти

Впровадження функціональності є ключовим етапом у розробці веб-застосунку для візуалізації алгоритмів та схем. Цей етап включає розробку основних модулів та компонентів, що забезпечують необхідні функціональні можливості застосунку, такі як інструменти для малювання, підтримка реального часу, аутентифікація користувачів, збереження даних та інші важливі аспекти.

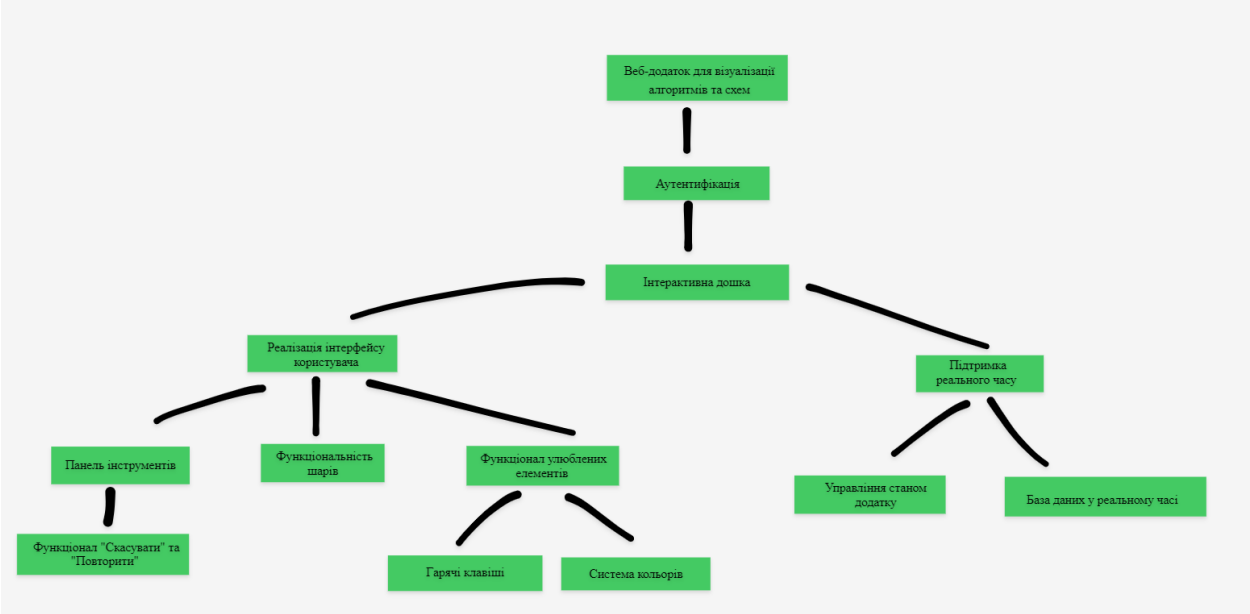


Рисунок 2.8 – схема основних модулів та компонентів веб-застосунку для візуалізації алгоритмів та схем виконана у реалізованій програмі

На основі розробленої практичної частини були впроваджені наступні ключові модулі та компоненти:

1. Інтерактивна дошка (Whiteboard) дозволяє користувачам створювати та редагувати схеми та алгоритми в режимі реального часу. Цей модуль включає різноманітні інструменти для малювання, такі як текстові поля, форми, стікери та олівець. Реалізація цієї функціональності базується на сучасних веб-технологіях, таких як HTML5 Canvas, що забезпечує високу продуктивність та гнучкість у відображенні графічних елементів.
2. Панель інструментів (Toolbar) надає користувачам доступ до різних інструментів для малювання та редагування. Панель включає кнопки для

додавання текстових полів, форм, стікерів, а також інструменти для малювання олівцем. Важливо, що панель інструментів реалізована з інтуїтивно зрозумілим розташуванням елементів, що дозволяє користувачам легко знаходити необхідні інструменти та швидко виконувати завдання.

3. Функціональність шарів (Layering functionality) дозволяє користувачам організовувати елементи на дошці у вигляді шарів, що значно полегшує управління складними схемами та алгоритмами. Користувачі можуть змінювати порядок шарів, приховувати або показувати їх, а також групувати елементи для зручності роботи. Ця функціональність реалізована за допомогою спеціалізованих бібліотек, що забезпечують гнучке управління шарами.

4. Система кольорів (Coloring system) дозволяє користувачам змінювати кольори елементів на дошці, що допомагає виділяти важливі частини схем та алгоритмів. Для цього реалізовано широкий вибір кольорів та зручний інтерфейс для їх вибору, що дозволяє користувачам швидко та легко змінювати кольори елементів, покращуючи візуальне сприйняття схем.

5. Функціонал "Скасувати" та "Повторити" (Undo & Redo functionality) є важливим компонентом, який дозволяє користувачам виправляти помилки та переглядати попередні зміни. Реалізація цієї функціональності включає зберігання історії змін та забезпечення можливості повернення до попередніх станів. Це значно підвищує зручність використання застосунку та дозволяє користувачам експериментувати з різними варіантами без ризику втрати даних.

6. Гарячі клавіші (Keyboard shortcuts) забезпечують швидкий доступ до основних функцій застосунку, що підвищує ефективність роботи користувачів. Реалізовано підтримку найбільш популярних та інтуїтивно зрозумілих комбінацій клавіш для виконання часто використовуваних операцій, таких як скасування, повторення, копіювання, вставка та видалення елементів.

7. Підтримка реального часу (Real-time collaboration) є критично важливою функцією, яка дозволяє кільком користувачам одночасно працювати над однією схемою або алгоритмом. Для реалізації цієї функціональності використовується Liveblocks, що забезпечує синхронізацію даних у реальному часі та дозволяє користувачам бачити зміни, внесені іншими учасниками, у режимі реального часу. Це підвищує ефективність роботи та забезпечує безперервність робочого процесу.

8. База даних у реальному часі (Real-time database) забезпечує зберігання даних користувачів та схем у реальному часі. Використання Convex дозволяє легко інтегрувати базу даних з іншими сервісами та забезпечує надійне зберігання даних. Це забезпечує високу продуктивність бази даних, що дозволяє користувачам швидко зберігати та завантажувати свої проекти.

9. Аутентифікація, управління організаціями та запрошення (Auth, organisations and invites) є критично важливими для забезпечення безпеки та управління доступом до застосунку. Для реалізації цієї функціональності використовується Clerk, що забезпечує потужну систему аутентифікації користувачів, управління організаціями та запрошеннями. Це дозволяє легко керувати доступом та забезпечити безпеку даних користувачів.

10. Функціонал улюблених елементів (Favoriting functionality) дозволяє користувачам зберігати улюблені елементи та швидко до них повертатися. Це підвищує зручність використання застосунку та дозволяє користувачам ефективно організовувати свої проекти.

11. Реалізація інтерфейсу користувача (UI implementation) базується на використанні TailwindCSS та ShadcnUI, що дозволяє створювати сучасний, адаптивний та зручний інтерфейс. TailwindCSS забезпечує утилітарний підхід до стилізації, що дозволяє розробникам швидко створювати адаптивні та добре структуровані стилі без написання великої кількості власного CSS-коду. ShadcnUI додає додаткові компоненти, що допомагають створювати елегантні та функціональні інтерфейси.

12. Для забезпечення управління станом додатку (state management) використовуються такі інструменти, як Redux або Context API в поєднанні з React. Це дозволяє ефективно управляти станом додатку та забезпечувати синхронізацію даних між різними компонентами. Управління станом є критично важливим для забезпечення узгодженості даних та підтримки складної логіки бізнес-процесів.

Впровадження всіх цих модулів та компонентів забезпечує створення повнофункціонального веб-застосунку для візуалізації алгоритмів та схем, який відповідає сучасним вимогам до продуктивності, безпеки та зручності використання. Кожен з цих модулів грає важливу роль у забезпеченні комплексного рішення, яке задовольняє потреби користувачів та допомагає досягти поставлених цілей.

## 2.5. Висновок

Розробка веб-ресурсу для візуалізації алгоритмів та схем є складним і багатокроковим процесом, що включає в себе аналіз предметної області, проектування архітектури, розробку інтерфейсу користувача та впровадження основних модулів і компонентів. Кожен з цих етапів є важливим і відіграє ключову роль у створенні повнофункціонального та зручного для користувачів продукту.

На етапі проектування архітектури веб-ресурсу було розроблено структуру системи, що забезпечує високий рівень продуктивності, масштабованості та надійності. Це дозволило створити базу для подальшого розвитку веб-ресурсу та його адаптації до змінних потреб користувачів.

Розробка інтерфейсу користувача включала дотримання основних принципів UX/UI дизайну, що забезпечило створення інтуїтивно зрозумілого та зручного інтерфейсу. Створення прототипів дозволило перевірити та вдосконалити дизайн до його реалізації. Використання сучасних інструментів для дизайну, таких як TailwindCSS та ShadcnUI, дозволило створити адаптивний та стильний інтерфейс.

Впровадження функціональності включало реалізацію ключових модулів та компонентів, таких як інтерактивна дошка, панель інструментів, система шарів, система кольорів, функціонал "Скасувати" та "Повторити", підтримка гарячих клавіш, реального часу співпраці, база даних у реальному часі, аутентифікація та управління організаціями, а також функціонал улюблених елементів. Це забезпечило створення повнофункціонального веб-ресурсу, що відповідає сучасним вимогам до продуктивності та безпеки.

Таким чином, реалізація проекту веб-ресурсу для візуалізації алгоритмів та схем включала всі необхідні етапи, від аналізу та проектування до впровадження функціональності. Використання сучасних технологій та інструментів дозволило створити продукт, що відповідає високим стандартам якості та забезпечує зручність і ефективність для кінцевих користувачів.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-РЕСУРСУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СХЕМ

## 3.1 Програмна реалізація веб-ресурсу для візуалізації алгоритмів та схем

Сучасний веб-застосунок для візуалізації алгоритмів і схем — це не просто проектування і кодування; це мистецтво створення інтуїтивно зрозумілих та візуально привабливих інтерфейсів, які можуть ефективно взаємодіяти з користувачем та обробляти складні дані. Використовуючи передові технології, розробники можуть перетворити складні технічні процеси на зрозумілі візуальні представлення, що значно спрощує взаємодію з користувачем та покращує загальний досвід використання продукту.

**Технологічний стек**

Для досягнення цих цілей ми обрали потужний і гнучкий набір інструментів:

- React — наш фундамент для фронтенду. Ця бібліотека ідеально підходить для створення динамічних, інтерактивних користувацьких інтерфейсів завдяки своєму компонентному підходу. React дозволяє нам ізолювати різні частини інтерфейсу в самодостатні блоки, що легко тестувати та підтримувати.
- Node.js — це серце нашого бекенду. Ця платформа вибрана за її високу продуктивність при роботі з асинхронними подіями та здатність ефективно обробляти численні запити, що є невід'ємною частиною обробки великих обсягів даних.

Ці технології формують основу нашого застосунку, дозволяючи створювати рішення, яке є одночасно потужним, ефективним та простим у використанні.

Приклад компоненту в React:

Файл: `app/(dashboard)/_components/board-list.tsx`

```
"use client";


import { useQuery } from "convex/react";
```

```
import { api } from "@/convex/_generated/api";

import { BoardCard } from "./board-card";
import { EmptySearch } from "./empty-search";
import { EmptyBoards } from "./empty-boards";
import { EmptyFavorites } from "./empty-favorites";
import { NewBoardButton } from "./new-board-button";

interface BoardListProps {
  orgId: string;
  query: {
    search?: string;
    favorites?: string;
  };
};

export const BoardList = ({
  orgId,
  query,
}: BoardListProps) => {
  const data = useQuery(api.boards.get, {
    orgId,
    ...query,
  });

  if (data === undefined) {
    return (
      <div>
        <h2 className="text-3xl">
          {query.favorites ? "Улюблені дошки" : "Командні дошки"}
        </h2>
        <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 lg:grid-cols-4 xl:grid-cols-5 2xl:grid-cols-6 gap-5 mt-8 pb-10">
```

```
        <NewBoardButton orgId={orgId} disabled />
        <BoardCard.Skeleton />
        <BoardCard.Skeleton />
        <BoardCard.Skeleton />
        <BoardCard.Skeleton />
      </div>
    </div>
  )
}

if (!data?.length && query.search) {
  return <EmptySearch />;
}

if (!data?.length && query.favorites) {
  return <EmptyFavorites />
}

if (!data?.length) {
  return <EmptyBoards />
}

return (
  <div>
    <h2 className="text-3xl">
      {query.favorites ? "Улюблені дошки" : "Командні дошки"}
    </h2>
    <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 lg:grid-cols-4 xl:grid-cols-5 2xl:grid-cols-
6 gap-5 mt-8 pb-10">
      <NewBoardButton orgId={orgId} />
      {data?.map((board) => (
        <BoardCard
          key={board._id}
          id={board._id}
```

```
        title={board.title}
        imageUrl={board.imageUrl}
        authorId={board.authorId}
        authorName={board.authorName}
        createdAt={board._creationTime}
        orgId={board.orgId}
        isFavorite={board.isFavorite}
      />
    ))}
   </div>
  </div>
 );
};
```

Цей компонент демонструє, як динамічно завантажувати та відображати дані з використанням React. Він використовує хук `useQuery` для отримання даних з сервера та відображає їх через компонент `BoardCard`.

useQuery - це кастомний хук, який виконує запит до API для отримання списку дошок (boards) для конкретної організації (orgId). Запит також може враховувати параметри пошуку (query).

Компонент використовує умовний рендеринг для відображення різних станів. Якщо дані ще не завантажені (`data === undefined`), відображаються скелетони дошок. Якщо не знайдено дошок відповідно до запиту, відображаються компоненти `EmptySearch`, `EmptyFavorites` або `EmptyBoards`.

Після успішного завантаження даних, кожна дошка відображається за допомогою компонента `BoardCard`.

**Інтеграція між фронтендом та бекендом**

Вибудовуючи зв'язок між користувацьким інтерфейсом, створеним на React, та серверними операціями на Node.js, ми гарантуємо, що всі користувацькі запити обробляються швидко та ефективно.

Використання React дозволяє створювати динамічні користувацькі інтерфейси. Завдяки сучасній архітектурі JavaScript і фреймворку React, ми можемо ефективно завантажувати дані на вимогу, без необхідності перезавантаження сторінки.

**Приклад компонента для створення нової дошки:**

Файл: `app/(dashboard)/_components/new-board-button.tsx`

```
"use client";

import { toast } from "sonner";
import { Plus } from "lucide-react";
import { useRouter } from "next/navigation";

import { cn } from "@/lib/utils";
import { api } from "@/convex/_generated/api";
import { useProModal } from "@/store/use-pro-modal";
import { useApiMutation } from "@/hooks/use-api-mutation";

interface NewBoardButtonProps {
  orgId: string;
  disabled?: boolean;
};

export const NewBoardButton = ({
  orgId,
  disabled,
}: NewBoardButtonProps) => {
  const router = useRouter();
  const { onOpen } = useProModal();
  const { mutate, pending } = useApiMutation(api.board.create);
```

```
 const onClick = () => {
  mutate({
   orgId,
   title: "Untitled"
  })
   .then((id) => {
    toast.success("Board created");
    router.push(`/board/${id}`);
   })
   .catch(() => {
    toast.error("Failed to create board");
    onOpen();
   });
 }


 return (
  <button
   disabled={pending || disabled}
   onClick={onClick}
   className={cn(
    "col-span-1 aspect-[100/127] bg-blue-600 rounded-lg hover:bg-blue-800 flex flex-col items-center
justify-center py-6",
    (pending || disabled) && "opacity-75 hover:bg-blue-600 cursor-not-allowed"
   )}
  >
   <div />
   <Plus className="h-12 w-12 text-white stroke-1" />
   <p className="text-sm text-white font-light">
    Нова дошка
   </p>
  </button>
```

```
  );
};
```

Пояснення:

Цей компонент використовує хук `useApiMutation` для асинхронного створення нової дошки через бекенд API та оновлення інтерфейсу після успішного створення.

- useRouter - використовується для перенаправлення користувача на сторінку новоствореної дошки після успішного створення.
- useApiMutation - це кастомний хук для виконання мутацій (змін) даних через API. Він керує станом запиту (`pending`) та обробляє результат запиту.
- toast - використовується для відображення повідомлень про успіх чи помилку при створенні дошки.

**Обробка та відгук на запити**

На бекенді, Node.js з Express забезпечує потужну платформу для обробки HTTP запитів. Роутинг в Express організовано так, щоб кожен маршрут міг ефективно обробляти конкретні запити, повертаючи потрібні дані у відповідь.

Приклад API маршруту для обробки запитів аутентифікації:

Файл: `app/api/liveblocks-auth/route.ts`

```
import { NextApiRequest, NextApiResponse } from "next";
import { getSession } from "@auth0/nextjs-auth0";

export default async function handler(req: NextApiRequest, res: NextApiResponse) {

 const session = getSession(req, res);

 if (!session) {
   res.status(401).json({ error: "Unauthorized" });
```

```
  return;
 }

 const { user } = session;
 res.status(200).json({ name: user.name });
}
```

Пояснення:

Цей маршрут обробляє запити аутентифікації, перевіряючи наявність сесії користувача і відповідним чином повертаючи інформацію про користувача або помилку.

- getSession – використовується для отримання сесії користувача. Якщо сесія не знайдена, маршрут повертає помилку 401 (Unauthorized).
- Якщо сесія існує, маршрут повертає ім'я користувача у відповіді.

**Використання хуків для управління станом**

Хуки в React дозволяють легко управляти станом і виконувати побічні ефекти у функціональних компонентах. В цьому розділі розглянемо приклади використання кастомних хуків з вашого проекту.

Приклад використання хука для асинхронних операцій:

Файл: `hooks/use-api-mutation.ts`

```
import { useState } from "react";
import { ApiClient } from "@/lib/api";

export function useApiMutation(apiEndpoint) {
 const [pending, setPending] = useState(false);

 const mutate = async (data) => {
  setPending(true);
  try {
   const response = await ApiClient.post(apiEndpoint, data);
   return response.data;
```

```
  } finally {
    setPending(false);
  }
 };

 return { mutate, pending };
}
```

Пояснення:

Цей хук `useApiMutation` дозволяє виконувати асинхронні операції з API, забезпечуючи управління станом запиту (pending) та обробку відповіді.

- useState - використовується для управління станом запиту (`pending`), що вказує на те, чи запит виконується.
- mutate – функція, яка виконує POST-запит до API та повертає відповідь. Вона також управляє станом запиту, встановлюючи його як `pending` під час виконання.

Приклад використання хука для видалення шарів:

Файл: `hooks/use-delete-layers.ts`

```
import { useCallback } from "react";
import { useApiMutation } from "@/hooks/use-api-mutation";
import { api } from "@/convex/_generated/api";

export function useDeleteLayers() {
 const { mutate, pending } = useApiMutation(api.layers.delete);

 const deleteLayers = useCallback(async (layerIds) => {
  await mutate({ ids: layerIds });
 }, [mutate]);

 return { deleteLayers, pending };
}
```

Пояснення:

Цей хук `useDeleteLayers` використовує `useApiMutation` для виконання асинхронного видалення шарів, забезпечуючи зручний API для видалення декількох елементів.

- useApiMutation - використовується для виконання асинхронного запиту до API для видалення шарів.
- useCallback - використовується для запобігання непотрібним рендерингам, запам'ятовуючи функцію `deleteLayers`, яка викликає `mutate` для видалення шарів.

**Інтеграція сторонніх сервісів**

Для розширення функціональності веб-застосунку часто використовуються сторонні сервіси. У вашому проекті прикладом може бути інтеграція зі Stripe для обробки платежів.

Приклад інтеграції зі Stripe:

Файл: `app/(dashboard)/_components/org-sidebar.tsx`

```
"use client";

import { toast } from "sonner";
import { useState } from "react";
import { useAction, useQuery } from "convex/react";
import { OrganizationSwitcher, useOrganization } from "@clerk/nextjs";
import { api } from "@/convex/_generated/api";
import { Button } from "@/components/ui/button";

export const OrgSidebar = () => {
 const { organization } = useOrganization();
 const isSubscribed = useQuery(api.subscriptions.getIsSubscribed, {
  orgId: organization?.id,
 });
```

```
const pay = useAction(api.stripe.pay);
const portal = useAction(api.stripe.portal);
const [pending, setPending] = useState(false);

const onClick = async () => {
 if (!organization?.id) return;

 setPending(true);

 try {
  const action = isSubscribed ? portal : pay;
  const redirectUrl = await action({
   orgId: organization.id,
  });

  window.location.href = redirectUrl;
 } catch {
  toast.error("Something went wrong");
 } finally {
  setPending(false);
 }
};

return (
 <div className="hidden lg:flex flex-col space-y-6 w-[206px] pl-5 pt-5">
  <OrganizationSwitcher hidePersonal />
  <div className="space-y-1 w-full">
   <Button
    onClick={onClick}
    disabled={pending}
```

```
        variant="ghost"
        size="lg"
      >
        {isSubscribed ? "Налаштування оплати" : "Покращити план"}
      </Button>
    </div>
  </div>
 );
};
```

Пояснення:

Цей компонент демонструє, як інтегрувати функціональність Stripe для обробки платежів і підписок у веб-застосунку.

- useAction - використовується для виклику асинхронних дій через API, таких як `pay` і `portal`, що відповідають за обробку платежів та управління підписками.
- useQuery - використовується для перевірки статусу підписки організації (`isSubscribed`).
- toast - використовується для відображення повідомлень про успіх чи помилку під час обробки платежів.

**Модальні вікна та підтвердження дій**

Модальні вікна використовуються для підтвердження важливих дій користувача та надання додаткової інформації. У цьому розділі розглянемо приклад використання модального вікна для підтвердження дій.

Приклад використання модального вікна для підтвердження дій:

Файл: `components/confirm-modal.tsx`

```
"use client";


import { useState } from "react";
```

```
import { Modal, Button } from "@/components/ui";

export const ConfirmModal = ({ isOpen, onClose, onConfirm }) => {
 const [loading, setLoading] = useState(false);

 const handleConfirm = async () => {
  setLoading(true);
  await onConfirm();
  setLoading(false);
  onClose();
 };

 return (
  <Modal isOpen={isOpen} onClose={onClose}>
   <div className="p-6">
    <h2 className="text-lg font-bold">Are you sure?</h2>
    <p className="text-sm text-gray-600 mt-2">
     This action cannot be undone.
    </p>
    <div className="mt-4 flex justify-end space-x-2">
     <Button onClick={onClose} variant="secondary">Cancel</Button>
     <Button onClick={handleConfirm} loading={loading}>
      Confirm
     </Button>
    </div>
   </div>
  </Modal>
 );
};
```

Пояснення:

Цей компонент `ConfirmModal` використовує стан для управління процесом підтвердження дії та завантаженням, забезпечуючи зручний інтерфейс для підтвердження важливих дій.

- useState - використовується для управління станом завантаження (`loading`), що вказує на те, чи дію підтверджують.
- handleConfirm - функція, яка виконується при підтвердженні дії. Вона викликає `onConfirm`, управляючи станом завантаження під час виконання.

Розробка сучасного веб-застосунку для візуалізації алгоритмів та схем є складним, але цікавим завданням, яке вимагає ретельного підходу до вибору технологій та архітектурних рішень. Використання передових інструментів, таких як React, Node.js та MongoDB, дозволяє створювати високопродуктивні, масштабовані та зручні у використанні додатки.

В рамках цього розділу було розглянуто технологічний стек, що використовується у проекті, а також детальні приклади коду, які демонструють ключові аспекти реалізації. Використання компонентного підходу, асинхронних операцій, кастомних хуків та інтеграція сторонніх сервісів забезпечують гнучкість та ефективність розробки. Також важливою частиною є створення зручного користувацького інтерфейсу, який дозволяє легко взаємодіяти з системою та отримувати необхідну інформацію у зручній формі.

## 3.2 Тестування та аналіз результатів роботи веб-ресурсу для візуалізації алгоритмів та схем

### Функціональне тестування


Рисунок 3.1 – тестування роботи панелі інструментів

На рисунку 3.1 зображено процес тестування панелі інструментів веб-ресурсу. Тут перевіряється функціональність різних інструментів, доступних користувачам, таких як олівець, форми та стікери. Користувач обирає кожен інструмент та застосовує його для створення елементів на інтерактивній дошці.

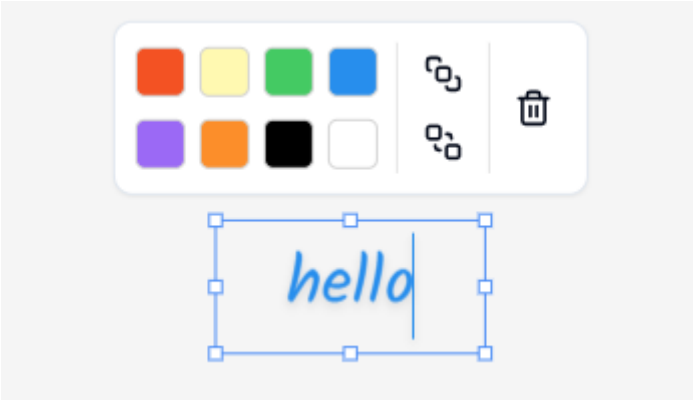


Рисунок 3.2 – тестування функціоналу додавання тексту

Рисунок 3.2 ілюструє тестування функціоналу додавання текстових полів. Користувач додає нове текстове поле на інтерактивну дошку, вводить текст та змінює його параметри (шрифт, розмір, колір). Перевіряється, чи зберігаються зміни та чи відображаються вони коректно.

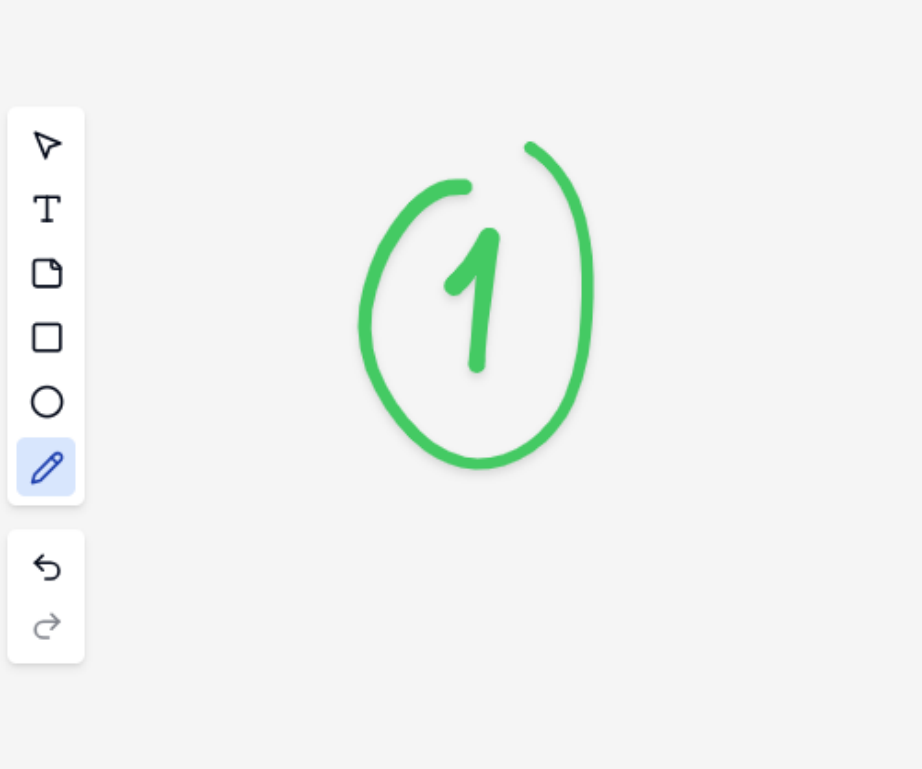

Рисунок 3.3 – зображення ДО використання функції “Скасувати”

На рисунку 3.3 представлено початковий стан інтерактивної дошки перед використанням функції "Скасувати". Це дозволяє порівняти зміни, внесені після натискання кнопки "Скасувати".

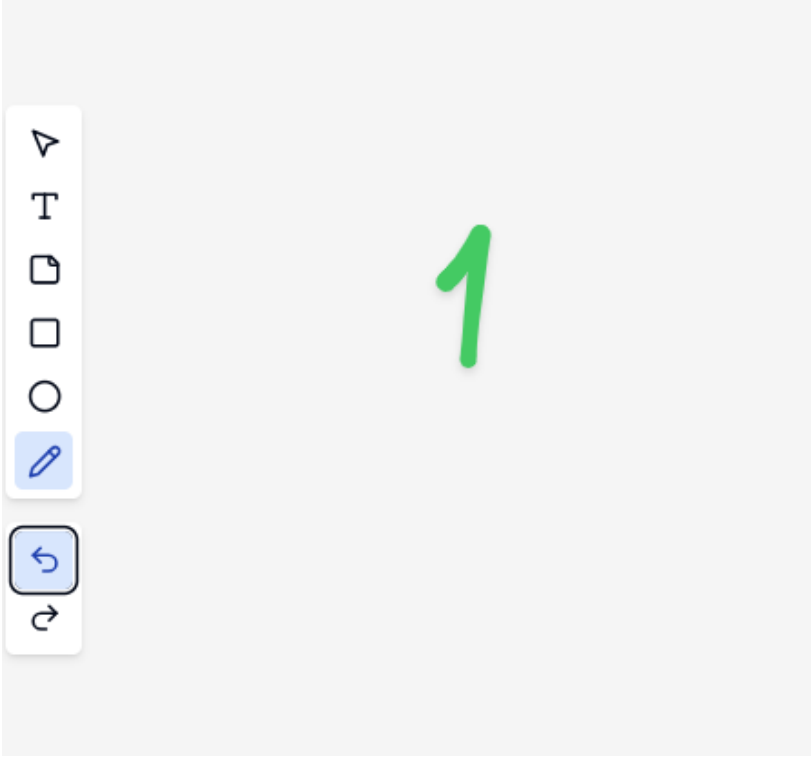

Рисунок 3.4 – зображення ПІСЛЯ використання функції “Скасувати”

Рисунок 3.4 демонструє інтерактивну дошку після використання функції "Скасувати". Користувач натискає кнопку "Скасувати", щоб повернутися до попереднього стану, що дозволяє оцінити коректність роботи цієї функції.

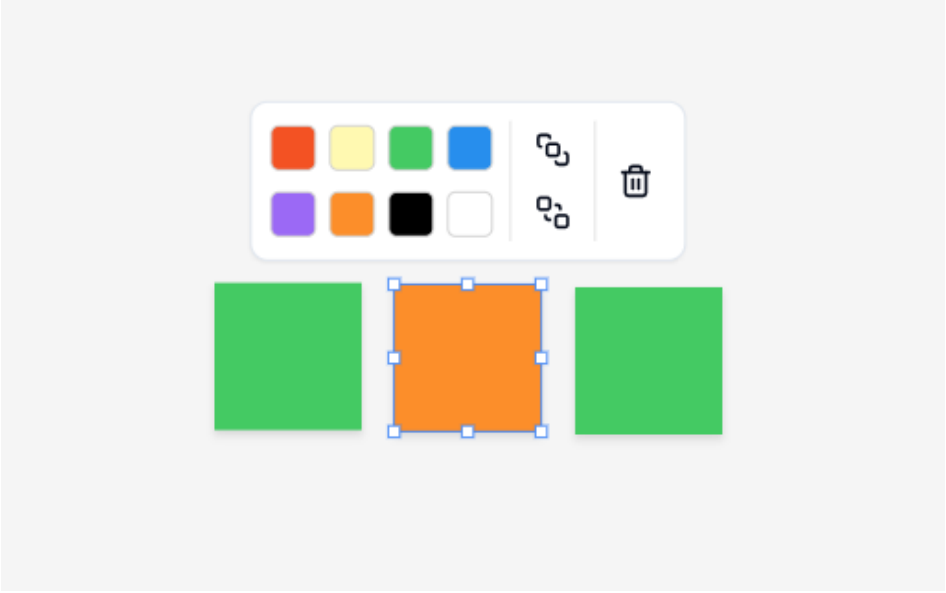

Рисунок 3.5 – тестування функціоналу зміни кольору елемента

На рисунку 3.5 показано процес тестування зміни кольору елементів. Користувач обирає елемент на інтерактивній дошці та змінює його колір за

допомогою доступних інструментів. Перевіряється, чи застосовуються зміни коректно та чи відображаються вони в режимі реального часу.

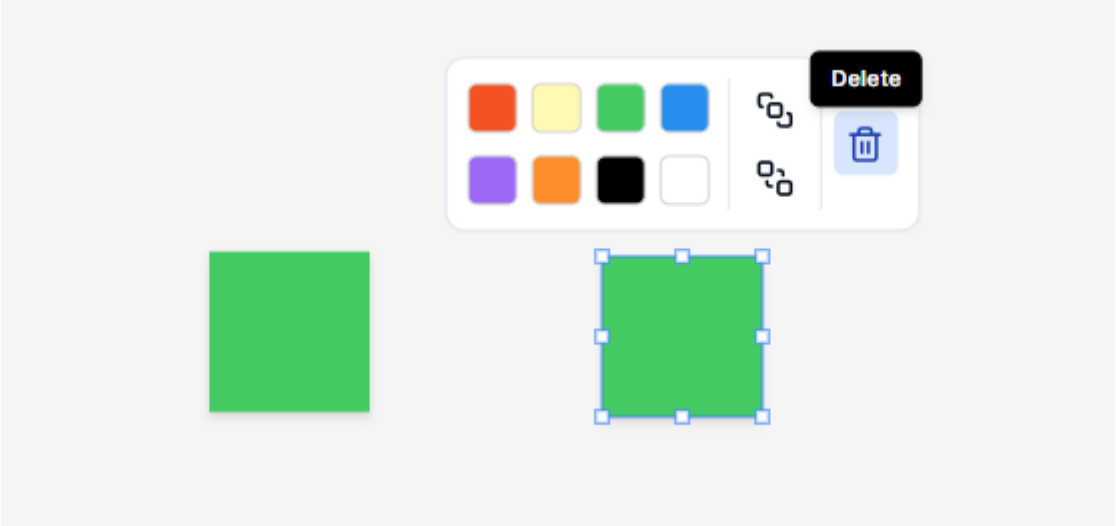


Рисунок 3.6 – тестування кнопки “Видалити”

Рисунок 3.6 ілюструє тестування функції видалення елементів. Користувач обирає елемент на інтерактивній дошці та натискає кнопку "Видалити". Перевіряється, чи видаляється елемент коректно та чи зникає він з дошки.

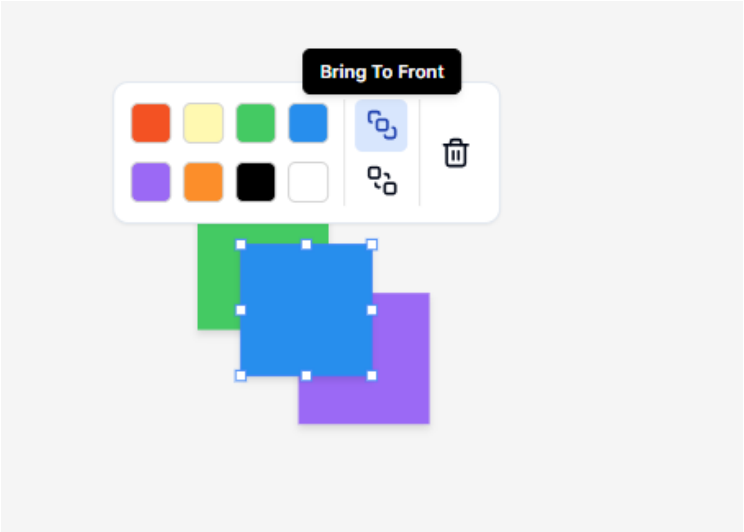


Рисунок 3.7 – тестування функціоналу “Шарів”

На рисунку 3.7 показано процес тестування функціоналу шарів. Користувач додає нові шари, змінює їх порядок, групує елементи та

приховує/показує окремі шари. Перевіряється, чи працює цей функціонал відповідно до очікувань.

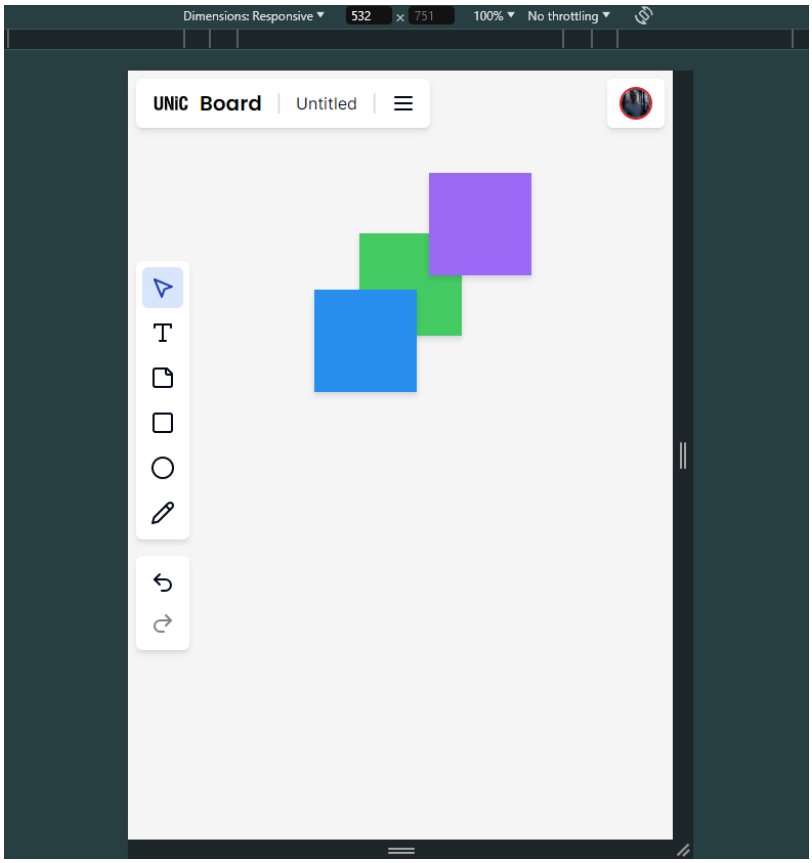


Рисунок 3.8 – Тестування адаптивності веб-застосунку

Рисунок 3.8 демонструє тестування адаптивності веб-застосунку на різних пристроях. Веб-ресурс відкривається на десктопі, планшеті та мобільному телефоні. Перевіряється, чи відображаються елементи інтерфейсу коректно на різних екранах та чи зберігається функціональність.

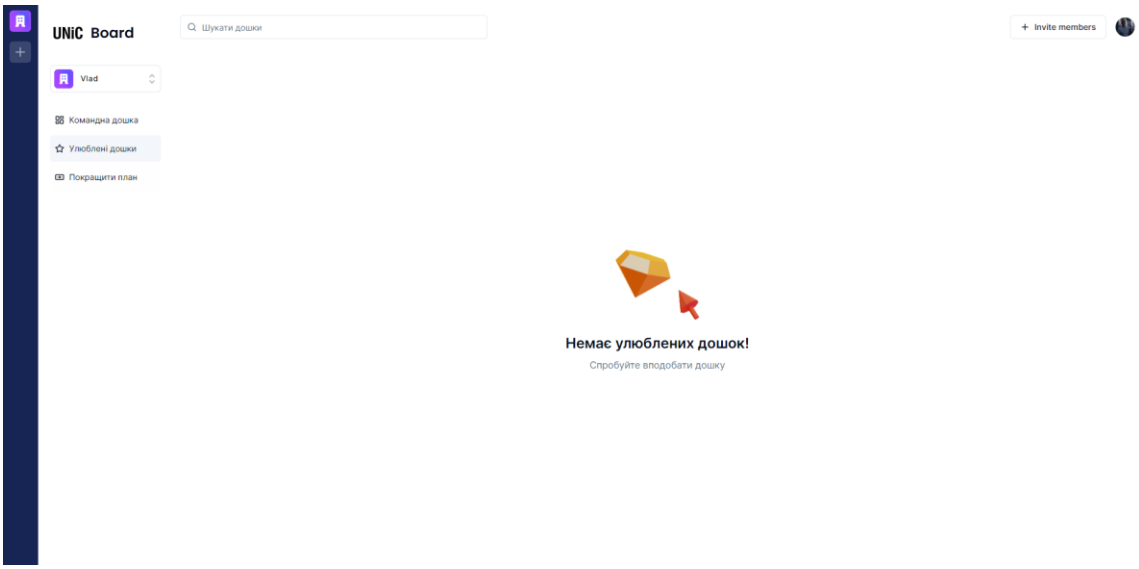


Рисунок 3.9 – тестування зручності інтерфейсу

На рисунку 3.9 представлено тестування зручності інтерфейсу користувача. Оцінюється логічність розташування елементів, легкість навігації та загальна зручність використання веб-ресурсу. Перевіряється, чи можуть користувачі легко знаходити необхідні функції та чи інтуїтивно зрозумілий інтерфейс.

# ВИСНОВКИ

У процесі виконання даної бакалаврської кваліфікаційної роботи було успішно розроблено веб-ресурс для візуалізації алгоритмів та схем. Робота включала аналіз предметної області, проектування архітектури, розробку інтерфейсу користувача та впровадження основних модулів і компонентів. Кожен з цих етапів відіграв важливу роль у створенні повнофункціонального та зручного для користувачів продукту.

У процесі дослідження були виконані всі поставлені задачі:

- Обґрунтування доцільності розробки веб-ресурсу для візуалізації

  Проведено аналіз, який підтвердив актуальність та необхідність створення веб-ресурсу для візуалізації алгоритмів та схем.

- Аналіз існуючих рішень та технологій у цій сфері

  Здійснено огляд та аналіз сучасних рішень і технологій, що дозволило визначити найкращі підходи для реалізації проекту.

- Проектування архітектури веб-ресурсу

  Розроблено архітектуру веб-ресурсу, що базується на клієнт-серверній архітектурі, яка забезпечує розділення між клієнтською частиною та серверною частиною, покращуючи масштабованість та гнучкість системи.

- Реалізація функціональних можливостей веб-ресурсу

  Успішно реалізовані всі необхідні функціональні можливості, включаючи інтерактивну дошку, панель інструментів, функціональність шарів, систему кольорів, функціонал "Скасувати" та "Повторити", гарячі клавіші, функціонал улюблених елементів, підтримку реального часу та управління станом.

- Тестування розробленого веб-ресурсу для забезпечення його ефективності та надійності

  Проведено комплексне тестування веб-ресурсу, що підтвердило його ефективність та надійність у використанні.

На етапі аналізу було визначено основні завдання та мету проекту, яка полягає в наданні користувачам інструментів для інтерактивної візуалізації та покрокового розуміння алгоритмів. Було проаналізовано існуючі рішення, такі як Miro, Lucidchart, Creately та Cacoo, їхні переваги та недоліки, що дозволило зрозуміти, які функціональні можливості та підходи є найбільш ефективними.

Проектування архітектури веб-ресурсу включало розробку структури системи, що забезпечує високий рівень продуктивності, масштабованості та надійності. Особлива увага приділялася інтерактивності, оскільки це ключовий аспект для забезпечення користувачів можливістю взаємодіяти з алгоритмами в режимі реального часу. У цьому контексті важливо було забезпечити можливість швидкої та ефективної обробки великих обсягів даних.

Розробка інтерфейсу користувача базувалася на принципах UX/UI дизайну, що забезпечило створення інтуїтивно зрозумілого та зручного інтерфейсу. Створення прототипів дозволило перевірити та вдосконалити дизайн до його реалізації. Використання сучасних інструментів для дизайну, таких як TailwindCSS та ShadcnUI, дозволило створити адаптивний та стильний інтерфейс.

Впровадження функціональності включало реалізацію основних модулів та компонентів, таких як інтерактивна дошка, панель інструментів, система шарів, система кольорів, функціонал "Скасувати" та "Повторити", підтримка гарячих клавіш, реального часу співпраці та база даних у реальному часі. Це забезпечило створення повнофункціонального веб-ресурсу, який відповідає сучасним вимогам до продуктивності та безпеки.

Особливу увагу було приділено впровадженню механізмів аутентифікації та авторизації користувачів для забезпечення безпеки даних. Використання сучасних технологій, таких як Clerk, забезпечило надійну аутентифікацію користувачів, управління організаціями та запрошеннями. Це дозволило створити безпечне середовище для роботи з даними та забезпечити захист від несанкціонованого доступу.

Також було реалізовано можливість спільної роботи в режимі реального часу, що підвищує ефективність командної роботи та дозволяє кільком користувачам одночасно працювати над однією схемою або алгоритмом. Це забезпечило синхронізацію даних у реальному часі та дозволило користувачам бачити зміни, внесені іншими учасниками, у режимі реального часу.

Загалом, реалізація проекту включала всі необхідні етапи від аналізу та проектування до впровадження функціональності. Використання сучасних технологій та інструментів дозволило створити продукт, що відповідає високим стандартам якості та забезпечує зручність і ефективність для кінцевих користувачів. Отримані результати свідчать про те, що розроблений веб-ресурс є конкурентоспроможним та корисним інструментом для широкої аудиторії, сприяючи покращенню якості освіти та професійного розвитку у сфері комп’ютерних наук.

Мета роботи – доведення необхідності розширення функціональних можливостей веб-ресурсів для візуалізації алгоритмів та схем – досягається за рахунок впровадження клієнт-серверної архітектури, що забезпечує покращену масштабованість, гнучкість та надійність у порівнянні з існуючими рішеннями.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Miro – інструмент для візуалізації [Електронний ресурс] – Режим доступу: https://miro.com
2. Lucidchart – платформа для створення схем [Електронний ресурс] – Режим доступу: https://www.lucidchart.com
3. Creately – онлайн-інструмент для візуалізації [Електронний ресурс] – Режим доступу: https://creately.com
4. React – бібліотека для створення інтерфейсів [Електронний ресурс] – Режим доступу: https://reactjs.org
5. Vue.js – прогресивний фреймворк [Електронний ресурс] – Режим доступу: https://vuejs.org
6. Angular – платформа для розробки додатків [Електронний ресурс] – Режим доступу: https://angular.io
7. Tailwind CSS – утилітарний CSS-фреймворк [Електронний ресурс] – Режим доступу: https://tailwindcss.com
8. Node.js – платформа для виконання JavaScript [Електронний ресурс] – Режим доступу: https://nodejs.org
9. Next.js – фреймворк для React [Електронний ресурс] – Режим доступу: https://nextjs.org
10. Convex – сервіс для роботи з даними [Електронний ресурс] – Режим доступу: https://convex.dev
11. Clerk – система аутентифікації [Електронний ресурс] – Режим доступу: https://clerk.dev
12. Liveblocks – платформа для реального часу співпраці [Електронний ресурс] – Режим доступу: https://liveblocks.io

# ДОДАТОК А

## ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка web-ресурсу для візуалізації алгоритмів та схем

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

**Показники звіту подібності Unicheck**

Оригінальність 98,07% Схожість 1,93%

Аналіз звіту подібності (відмітити потрібне):

- ☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Полігас В.В.

Керівник роботи Сілагін О.В.

# ДОДАТОК Б (обов’язковий)

## Лістинг програми

Файл board.ts:

```
import { v } from "convex/values";


import { mutation, query } from "./_generated/server";


const ORG_BOARD_LIMIT = 2;


const images = [
  "/placeholders/1.svg",
  "/placeholders/2.svg",
  "/placeholders/3.svg",
  "/placeholders/4.svg",
  "/placeholders/5.svg",
  "/placeholders/6.svg",
  "/placeholders/7.svg",
  "/placeholders/8.svg",
  "/placeholders/9.svg",
  "/placeholders/10.svg",
];


export const create = mutation({
  args: {
    orgId: v.string(),
    title: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
```

```
if (!identity) {
  throw new Error("Unauthorized");
}

const randomImage = images[Math.floor(Math.random() * images.length)];

const orgSubscription = await ctx.db
  .query("orgSubscription")
  .withIndex("by_org", (q) => q.eq("orgId", args.orgId))
  .unique();

const periodEnd = orgSubscription?.stripeCurrentPeriodEnd;

const isSubscribed = periodEnd && periodEnd > Date.now();

const existingBoards = await ctx.db
  .query("boards")
  .withIndex("by_org", (q) => q.eq("orgId", args.orgId))
  .collect();

if (
  !isSubscribed &&
  existingBoards.length >= ORG_BOARD_LIMIT
) {
  throw new Error("Organization limit reached");
}

const board = await ctx.db.insert("boards", {
  title: args.title,
  orgId: args.orgId,
  authorId: identity.subject,
```

```
    authorName: identity.name!,
    imageUrl: randomImage,
   });

   return board;
  },
});

export const remove = mutation({
  args: { id: v.id("boards") },
  handler: async (ctx, args) => {
   const identity = await ctx.auth.getUserIdentity();

   if (!identity) {
    throw new Error("Unauthorized");
   }

   const userId = identity.subject;

   const existingFavorite = await ctx.db
    .query("userFavorites")
    .withIndex("by_user_board", (q) =>
     q
      .eq("userId", userId)
      .eq("boardId", args.id)
    )
    .unique();

   if (existingFavorite) {
    await ctx.db.delete(existingFavorite._id);
   }
```

```
    await ctx.db.delete(args.id);
  },
});

export const update = mutation({
  args: { id: v.id("boards"), title: v.string() },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Unauthorized");
    }

    const title = args.title.trim();

    if (!title) {
      throw new Error("Title is required");
    }

    if (title.length > 60) {
      throw new Error("Title cannot be longer than 60 characters")
    }

    const board = await ctx.db.patch(args.id, {
      title: args.title,
    });

    return board;
  },
});
```

```
export const favorite = mutation({
 args: { id: v.id("boards"), orgId: v.string() },
 handler: async (ctx, args) => {
  const identity = await ctx.auth.getUserIdentity();

  if (!identity) {
   throw new Error("Unauthorized");
  }

  const board = await ctx.db.get(args.id);

  if (!board) {
   throw new Error("Board not found");
  }

  const userId = identity.subject;

  const existingFavorite = await ctx.db
   .query("userFavorites")
   .withIndex("by_user_board", (q) =>
    q
     .eq("userId", userId)
     .eq("boardId", board._id)
   )
   .unique();

  if (existingFavorite) {
   throw new Error("Board already favorited");
  }
```

```
    await ctx.db.insert("userFavorites", {
      userId,
      boardId: board._id,
      orgId: args.orgId,
    });

    return board;
  },
});

export const unfavorite = mutation({
  args: { id: v.id("boards") },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();

    if (!identity) {
      throw new Error("Unauthorized");
    }

    const board = await ctx.db.get(args.id);

    if (!board) {
      throw new Error("Board not found");
    }

    const userId = identity.subject;

    const existingFavorite = await ctx.db
      .query("userFavorites")
      .withIndex("by_user_board", (q) =>
        q
```

```
        .eq("userId", userId)
        .eq("boardId", board._id)
    )
    .unique();

  if (!existingFavorite) {
    throw new Error("Favorited board not found");
  }

  await ctx.db.delete(existingFavorite._id);

  return board;
 },
});

export const get = query({
 args: { id: v.id("boards") },
 handler: async (ctx, args) => {
  const board = ctx.db.get(args.id);

  return board;
 },
});
```

## Файл canvas.tsx

```
"use client";

import { nanoid } from "nanoid";
import { useCallback, useMemo, useState, useEffect } from "react";
import { LiveObject } from "@liveblocks/client";
```

```
import {
  useHistory,
  useCanUndo,
  useCanRedo,
  useMutation,
  useStorage,
  useOthersMapped,
  useSelf,
} from "@/liveblocks.config";
import {
  colorToCss,
  connectionIdToColor,
  findIntersectingLayersWithRectangle,
  penPointsToPathLayer,
  pointerEventToCanvasPoint,
  resizeBounds,
} from "@/lib/utils";
import {
  Camera,
  CanvasMode,
  CanvasState,
  Color,
  LayerType,
  Point,
  Side,
  XYWH,
} from "@/types/canvas";
import { useDisableScrollBounce } from "@/hooks/use-disable-scroll-bounce";
import { useDeleteLayers } from "@/hooks/use-delete-layers";

import { Info } from "./info";
```

```
import { Path } from "./path";
import { Toolbar } from "./toolbar";
import { Participants } from "./participants";
import { LayerPreview } from "./layer-preview";
import { SelectionBox } from "./selection-box";
import { SelectionTools } from "./selection-tools";
import { CursorsPresence } from "./cursors-presence";

const MAX_LAYERS = 100;

interface CanvasProps {
  boardId: string;
};

export const Canvas = ({
  boardId,
}: CanvasProps) => {
  const layerIds = useStorage((root) => root.layerIds);

  const pencilDraft = useSelf((me) => me.presence.pencilDraft);
  const [canvasState, setCanvasState] = useState<CanvasState>({
    mode: CanvasMode.None,
  });
  const [camera, setCamera] = useState<Camera>({ x: 0, y: 0 });
  const [lastUsedColor, setLastUsedColor] = useState<Color>({
    r: 0,
    g: 0,
    b: 0,
  });
```

```
useDisableScrollBounce();
const history = useHistory();
const canUndo = useCanUndo();
const canRedo = useCanRedo();

const insertLayer = useMutation((
 { storage, setMyPresence },
 layerType: LayerType.Ellipse | LayerType.Rectangle | LayerType.Text | LayerType.Note,
 position: Point,
) => {
 const liveLayers = storage.get("layers");
 if (liveLayers.size >= MAX_LAYERS) {
  return;
 }

 const liveLayerIds = storage.get("layerIds");
 const layerId = nanoid();
 const layer = new LiveObject({
  type: layerType,
  x: position.x,
  y: position.y,
  height: 100,
  width: 100,
  fill: lastUsedColor,
 });

 liveLayerIds.push(layerId);
 liveLayers.set(layerId, layer);

 setMyPresence({ selection: [layerId] }, { addToHistory: true });
```

```
  setCanvasState({ mode: CanvasMode.None });
}, [lastUsedColor]);

const translateSelectedLayers = useMutation((
  { storage, self },
  point: Point,
) => {
  if (canvasState.mode !== CanvasMode.Translating) {
    return;
  }

  const offset = {
    x: point.x - canvasState.current.x,
    y: point.y - canvasState.current.y,
  };

  const liveLayers = storage.get("layers");

  for (const id of self.presence.selection) {
    const layer = liveLayers.get(id);

    if (layer) {
      layer.update({
        x: layer.get("x") + offset.x,
        y: layer.get("y") + offset.y,
      });
    }
  }

  setCanvasState({ mode: CanvasMode.Translating, current: point });
},
```

```
[
  canvasState,
]);

const unselectLayers = useMutation((
  { self, setMyPresence }
) => {
  if (self.presence.selection.length > 0) {
    setMyPresence({ selection: [] }, { addToHistory: true });
  }
}, []);

const updateSelectionNet = useMutation((
  { storage, setMyPresence },
  current: Point,
  origin: Point,
) => {
  const layers = storage.get("layers").toImmutable();
  setCanvasState({
    mode: CanvasMode.SelectionNet,
    origin,
    current,
  });

  const ids = findIntersectingLayersWithRectangle(
    layerIds,
    layers,
    origin,
    current,
  );
```

```
  setMyPresence({ selection: ids });
}, [layerIds]);

const startMultiSelection = useCallback((
  current: Point,
  origin: Point,
) => {
  if (
    Math.abs(current.x - origin.x) + Math.abs(current.y - origin.y) > 5
  ) {
    setCanvasState({
      mode: CanvasMode.SelectionNet,
      origin,
      current,
    });
  }
}, []);

const continueDrawing = useMutation((
  { self, setMyPresence },
  point: Point,
  e: React.PointerEvent,
) => {
  const { pencilDraft } = self.presence;

  if (
    canvasState.mode !== CanvasMode.Pencil ||
    e.buttons !== 1 ||
    pencilDraft == null
  ) {
    return;
```

```
  }

  setMyPresence({
    cursor: point,
    pencilDraft:
      pencilDraft.length === 1 &&
      pencilDraft[0][0] === point.x &&
      pencilDraft[0][1] === point.y
        ? pencilDraft
        : [...pencilDraft, [point.x, point.y, e.pressure]],
  });
}, [canvasState.mode]);

const insertPath = useMutation((
  { storage, self, setMyPresence }
) => {
  const liveLayers = storage.get("layers");
  const { pencilDraft } = self.presence;

  if (
    pencilDraft == null ||
    pencilDraft.length < 2 ||
    liveLayers.size >= MAX_LAYERS
  ) {
    setMyPresence({ pencilDraft: null });
    return;
  }

  const id = nanoid();
  liveLayers.set(
    id,
```

```
    new LiveObject(penPointsToPathLayer(
      pencilDraft,
      lastUsedColor,
    )),
  );

  const liveLayerIds = storage.get("layerIds");
  liveLayerIds.push(id);

  setMyPresence({ pencilDraft: null });
  setCanvasState({ mode: CanvasMode.Pencil });
}, [lastUsedColor]);

const startDrawing = useMutation((
  { setMyPresence },
  point: Point,
  pressure: number,
) => {
  setMyPresence({
    pencilDraft: [[point.x, point.y, pressure]],
    penColor: lastUsedColor,
  })
}, [lastUsedColor]);

const resizeSelectedLayer = useMutation((
  { storage, self },
  point: Point,
) => {
  if (canvasState.mode !== CanvasMode.Resizing) {
    return;
  }
```

```
  const bounds = resizeBounds(
    canvasState.initialBounds,
    canvasState.corner,
    point,
  );

  const liveLayers = storage.get("layers");
  const layer = liveLayers.get(self.presence.selection[0]);

  if (layer) {
    layer.update(bounds);
  };
}, [canvasState]);

const onResizeHandlePointerDown = useCallback((
  corner: Side,
  initialBounds: XYWH,
) => {
  history.pause();
  setCanvasState({
    mode: CanvasMode.Resizing,
    initialBounds,
    corner,
  });
}, [history]);

const onWheel = useCallback((e: React.WheelEvent) => {
  setCamera((camera) => ({
    x: camera.x - e.deltaX,
    y: camera.y - e.deltaY,
```

```
  }));
}, []);

const onPointerMove = useMutation((
  { setMyPresence },
  e: React.PointerEvent
) => {
  e.preventDefault();

  const current = pointerEventToCanvasPoint(e, camera);

  if (canvasState.mode === CanvasMode.Pressing) {
    startMultiSelection(current, canvasState.origin);
  } else if (canvasState.mode === CanvasMode.SelectionNet) {
    updateSelectionNet(current, canvasState.origin);
  } else if (canvasState.mode === CanvasMode.Translating) {
    translateSelectedLayers(current);
  } else if (canvasState.mode === CanvasMode.Resizing) {
    resizeSelectedLayer(current);
  } else if (canvasState.mode === CanvasMode.Pencil) {
    continueDrawing(current, e);
  }

  setMyPresence({ cursor: current });
},
[
  continueDrawing,
  camera,
  canvasState,
  resizeSelectedLayer,
  translateSelectedLayers,
```

```
  startMultiSelection,
  updateSelectionNet,
]);

const onPointerLeave = useMutation(({ setMyPresence }) => {
  setMyPresence({ cursor: null });
}, []);

const onPointerDown = useCallback((
  e: React.PointerEvent,
) => {
  const point = pointerEventToCanvasPoint(e, camera);

  if (canvasState.mode === CanvasMode.Inserting) {
    return;
  }

  if (canvasState.mode === CanvasMode.Pencil) {
    startDrawing(point, e.pressure);
    return;
  }

  setCanvasState({ origin: point, mode: CanvasMode.Pressing });
}, [camera, canvasState.mode, setCanvasState, startDrawing]);

const onPointerUp = useMutation((
  {},
  e
) => {
  const point = pointerEventToCanvasPoint(e, camera);
```

```
  if (
    canvasState.mode === CanvasMode.None ||
    canvasState.mode === CanvasMode.Pressing
  ) {
    unselectLayers();
    setCanvasState({
      mode: CanvasMode.None,
    });
  } else if (canvasState.mode === CanvasMode.Pencil) {
    insertPath();
  } else if (canvasState.mode === CanvasMode.Inserting) {
    insertLayer(canvasState.layerType, point);
  } else {
    setCanvasState({
      mode: CanvasMode.None,
    });
  }

  history.resume();
},
[
  setCanvasState,
  camera,
  canvasState,
  history,
  insertLayer,
  unselectLayers,
  insertPath
]);

const selections = useOthersMapped((other) => other.presence.selection);
```

```
const onLayerPointerDown = useMutation((
  { self, setMyPresence },
  e: React.PointerEvent,
  layerId: string,
) => {
  if (
    canvasState.mode === CanvasMode.Pencil ||
    canvasState.mode === CanvasMode.Inserting
  ) {
    return;
  }

  history.pause();
  e.stopPropagation();

  const point = pointerEventToCanvasPoint(e, camera);

  if (!self.presence.selection.includes(layerId)) {
    setMyPresence({ selection: [layerId] }, { addToHistory: true });
  }
  setCanvasState({ mode: CanvasMode.Translating, current: point });
},
[
  setCanvasState,
  camera,
  history,
  canvasState.mode,
]);

const layerIdsToColorSelection = useMemo(() => {
```

```
  const layerIdsToColorSelection: Record<string, string> = {};

  for (const user of selections) {
    const [connectionId, selection] = user;

    for (const layerId of selection) {
      layerIdsToColorSelection[layerId] = connectionIdToColor(connectionId)
    }
  }

  return layerIdsToColorSelection;
}, [selections]);

const deleteLayers = useDeleteLayers();

useEffect(() => {
  function onKeyDown(e: KeyboardEvent) {
    switch (e.key) {
      // case "Backspace":
      //   deleteLayers();
      //   break;
      case "z": {
        if (e.ctrlKey || e.metaKey) {
          if (e.shiftKey) {
            history.redo();
          } else {
            history.undo();
          }
          break;
        }
      }
```

```
    }
  }

  document.addEventListener("keydown", onKeyDown);

  return () => {
    document.removeEventListener("keydown", onKeyDown)
  }
}, [deleteLayers, history]);

return (
  <main
    className="h-full w-full relative bg-neutral-100 touch-none"
  >
    <Info boardId={boardId} />
    <Participants />
    <Toolbar
      canvasState={canvasState}
      setCanvasState={setCanvasState}
      canRedo={canRedo}
      canUndo={canUndo}
      undo={history.undo}
      redo={history.redo}
    />
    <SelectionTools
      camera={camera}
      setLastUsedColor={setLastUsedColor}
    />
    <svg
      className="h-[100vh] w-[100vw]"
      onWheel={onWheel}
```

```
 onPointerMove={onPointerMove}
 onPointerLeave={onPointerLeave}
 onPointerDown={onPointerDown}
 onPointerUp={onPointerUp}
>
 <g
  style={{
   transform: `translate(${camera.x}px, ${camera.y}px)`
  }}
 >
  {layerIds.map((layerId) => (
   <LayerPreview
    key={layerId}
    id={layerId}
    onLayerPointerDown={onLayerPointerDown}
    selectionColor={layerIdsToColorSelection[layerId]}
   />
  ))}
  <SelectionBox
   onResizeHandlePointerDown={onResizeHandlePointerDown}
  />
  {canvasState.mode === CanvasMode.SelectionNet && canvasState.current != null && (
   <rect
    className="fill-blue-500/5 stroke-blue-500 stroke-1"
    x={Math.min(canvasState.origin.x, canvasState.current.x)}
    y={Math.min(canvasState.origin.y, canvasState.current.y)}
    width={Math.abs(canvasState.origin.x - canvasState.current.x)}
    height={Math.abs(canvasState.origin.y - canvasState.current.y)}
   />
  )}
  <CursorsPresence />
```

```
        {pencilDraft != null && pencilDraft.length > 0 && (
          <Path
            points={pencilDraft}
            fill={colorToCss(lastUsedColor)}
            x={0}
            y={0}
          />
        )}
      </g>
    </svg>
  </main>
 );
};
```

**ДОДАТОК В (обов'язковий)**

**Графічна частина**

# «РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СХЕМ»

Виконав: студент 4 курсу, групи 1КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Полігас В.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О. В.
(прізвище та ініціали)
«07» 06 2024

Вінниця ВНТУ - 2024 рік

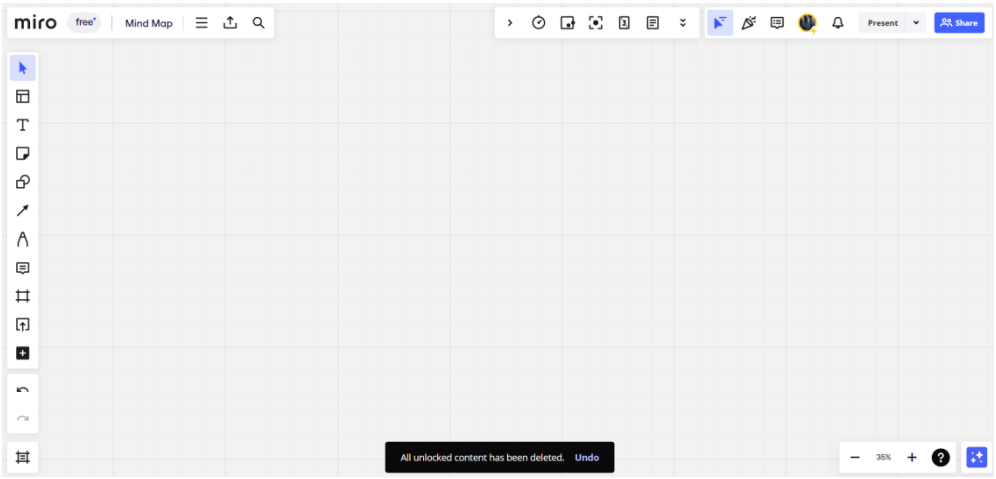


Рисунок В.1 – інтерфейс Miro

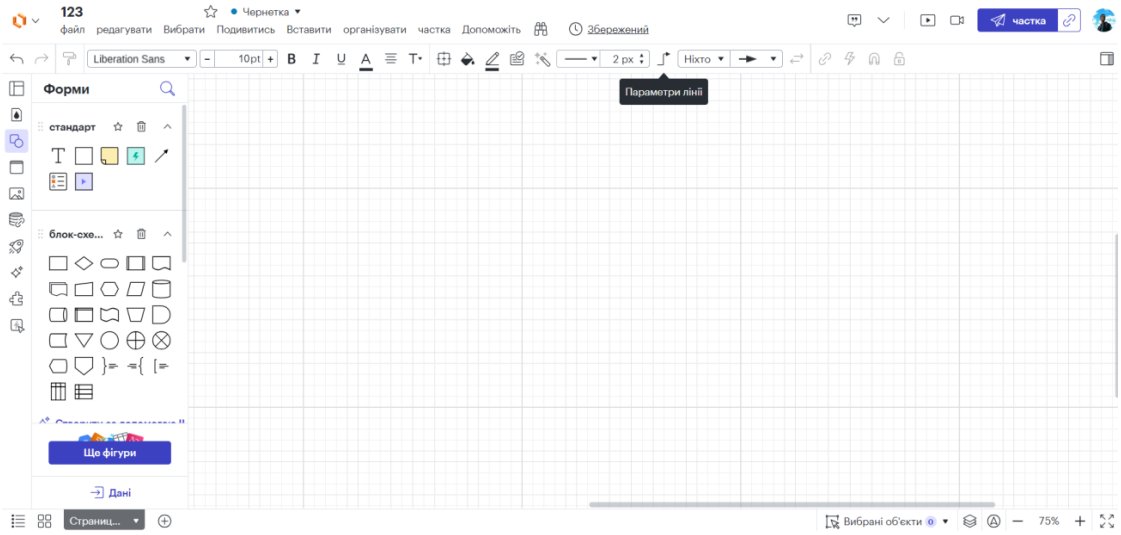


Рисунок В.2 – інтерфейс Lucidchart

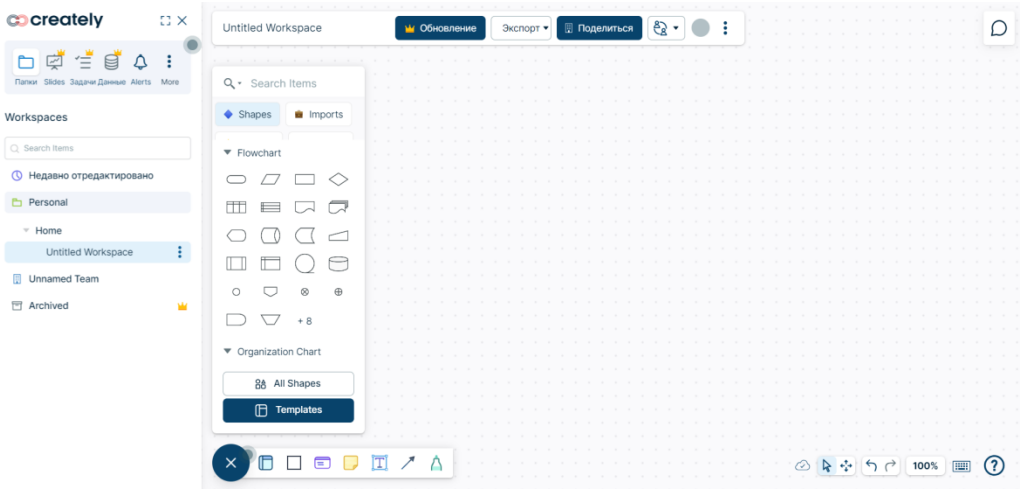


Рисунок В.3 – інтерфейс Creately

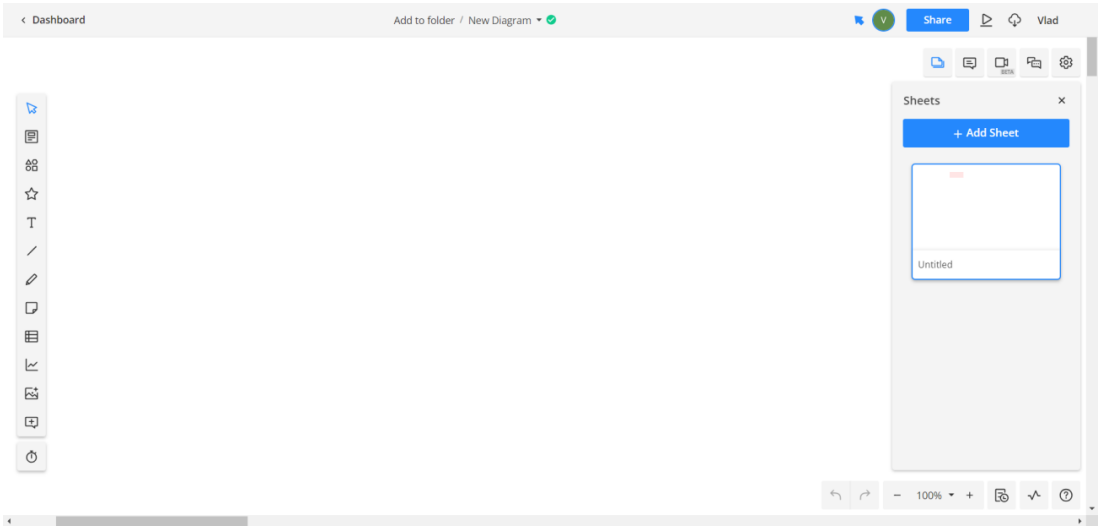


Рисунок В.4 – інтерфейс Cacoo


Рисунок В.5 – портрет цільової аудиторії


Рисунок В.6 – нефукціональні вимоги

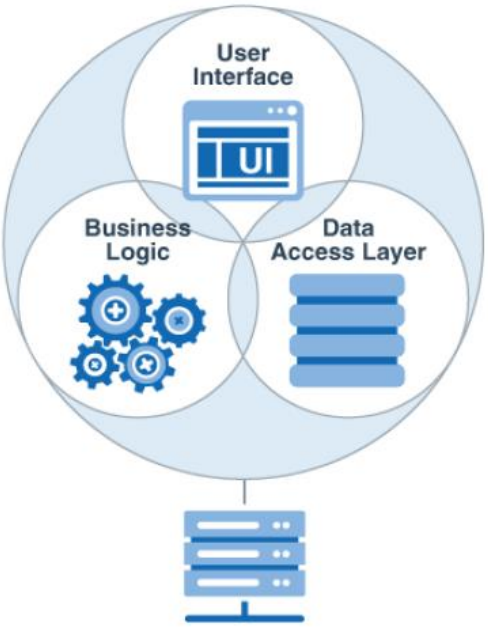


Рисунок В.7 – схема монолітної архітектури

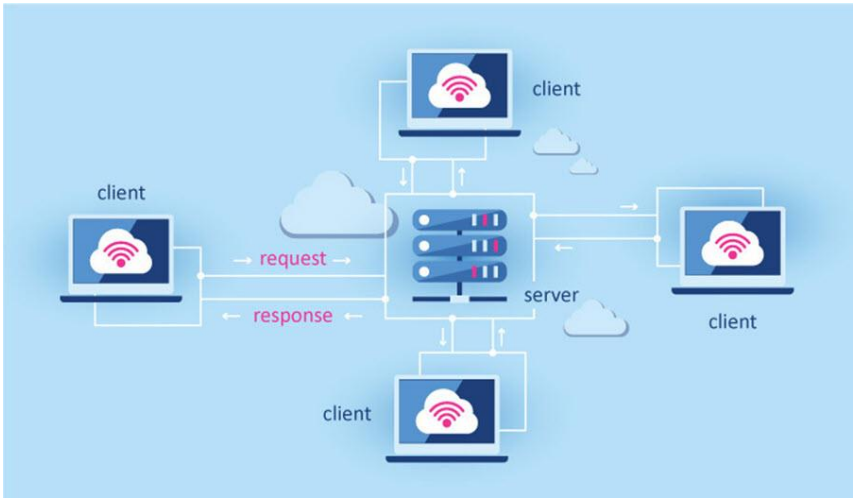


Рисунок В.8 – схема клієнт-серверної архітектури


Рисунок В.9 – схема трирівневої архітектури

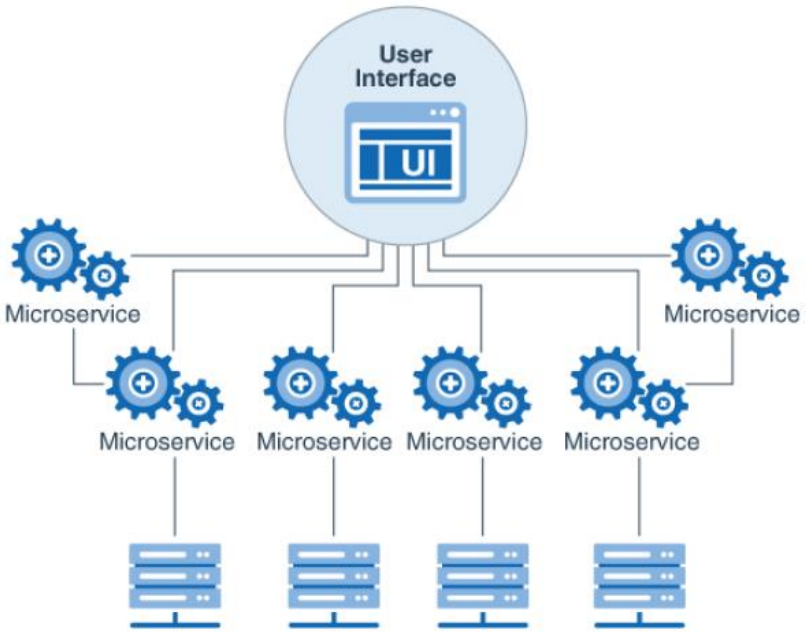


Рисунок В.10 – схема мікросервісної архітектури

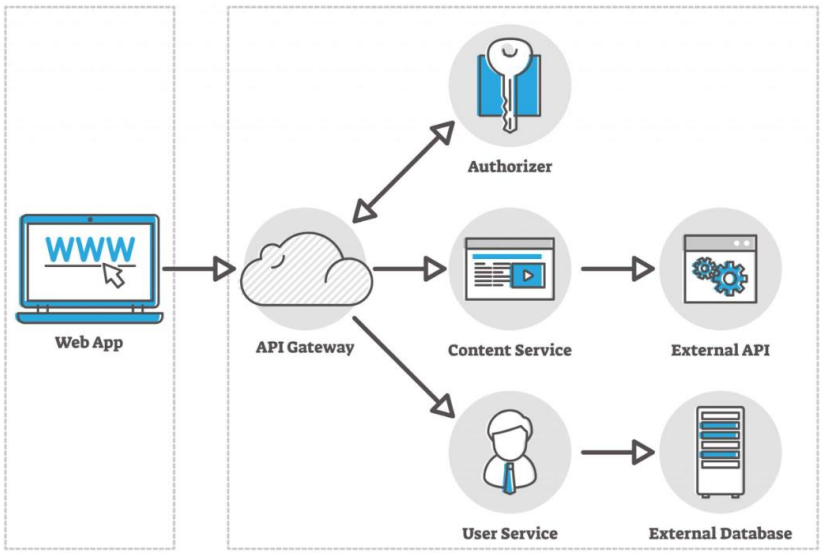


Рисунок В.11 – схема архітектури без сервера

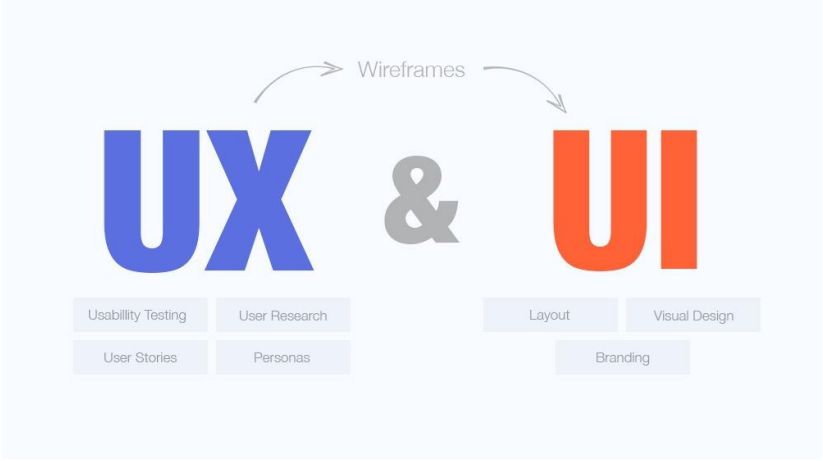


Рисунок В.12 – принципи UX/UI дизайну

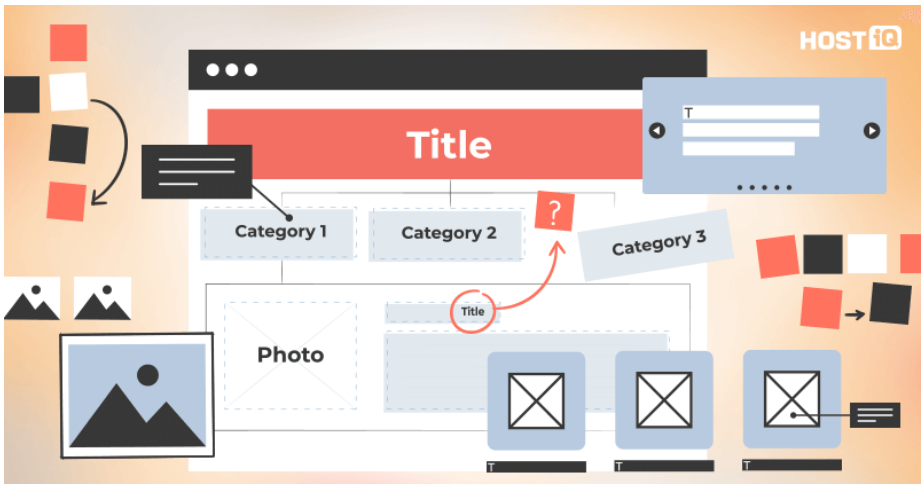


Рисунок В.13 – створення прототипу сайту

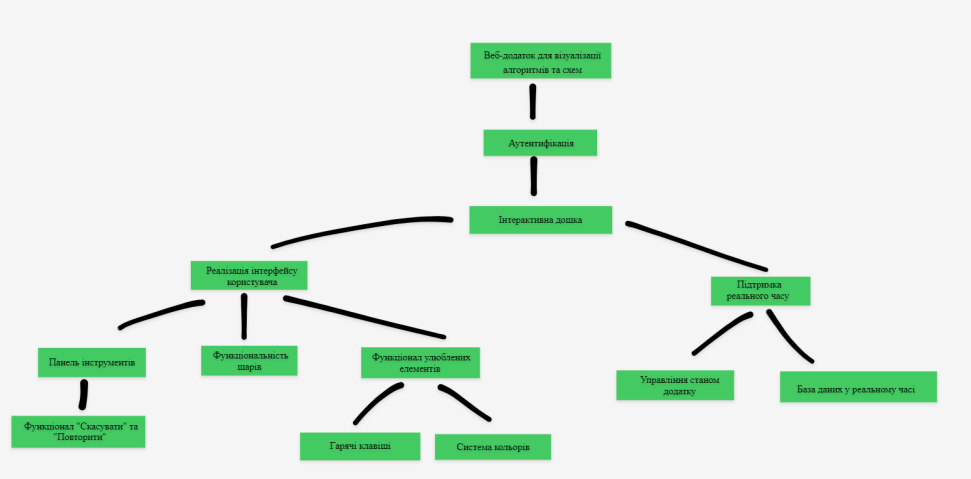


Рисунок В.14 – схема основних модулів та компонентів веб застосунку для візуалізації алгоритмів та схем виконана у реалізованій програмі

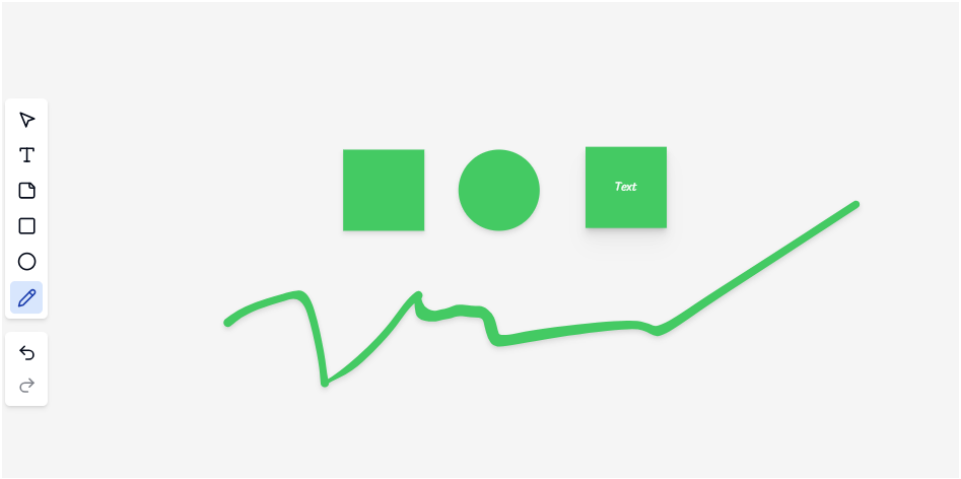


Рисунок В.15 – тестування роботи панелі інструментів

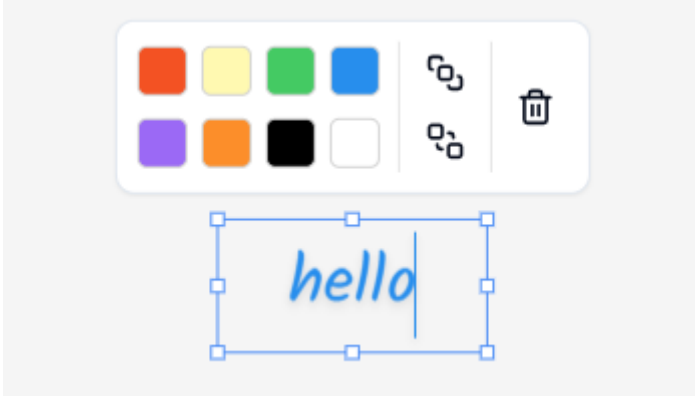


Рисунок В.16 – тестування функціоналу додавання тексту

Рисунок В.17 – зображення ДО використання функції “Скасувати”

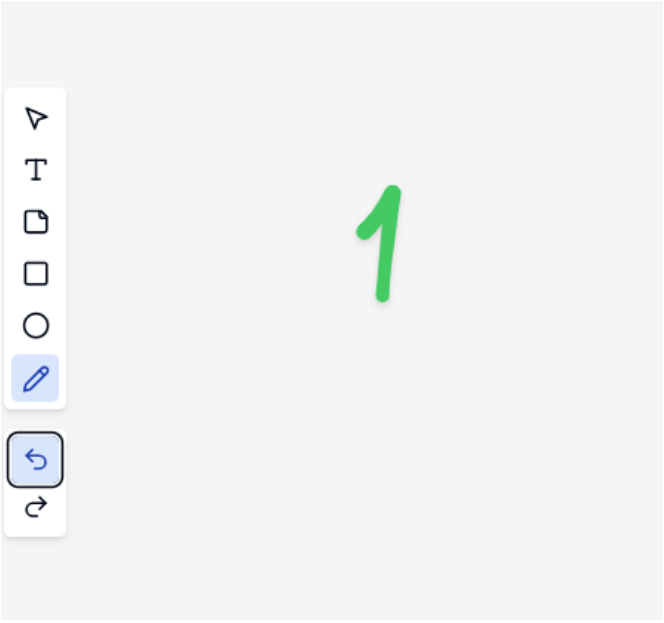

Рисунок В.18 – зображення ПІСЛЯ використання функції “Скасувати”

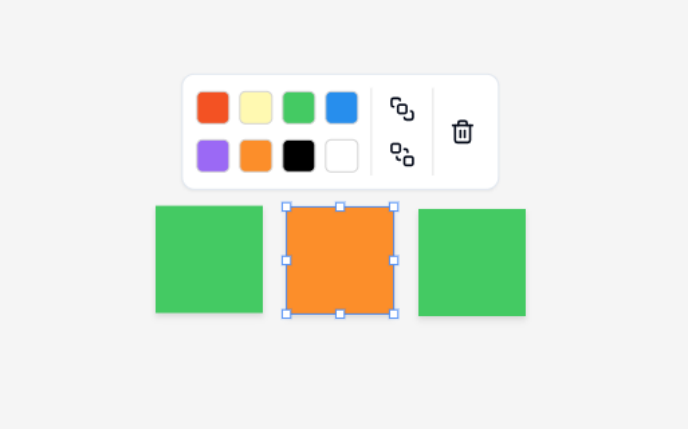

Рисунок В.19 – тестування функціоналу зміни кольору елемента

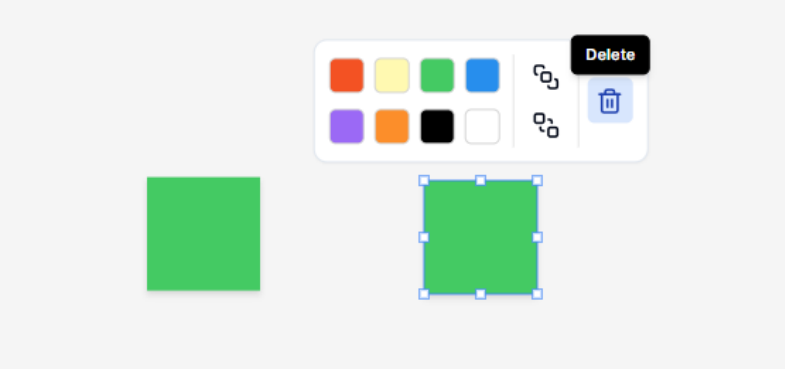


Рисунок В.20 – тестування кнопки “Видалити”

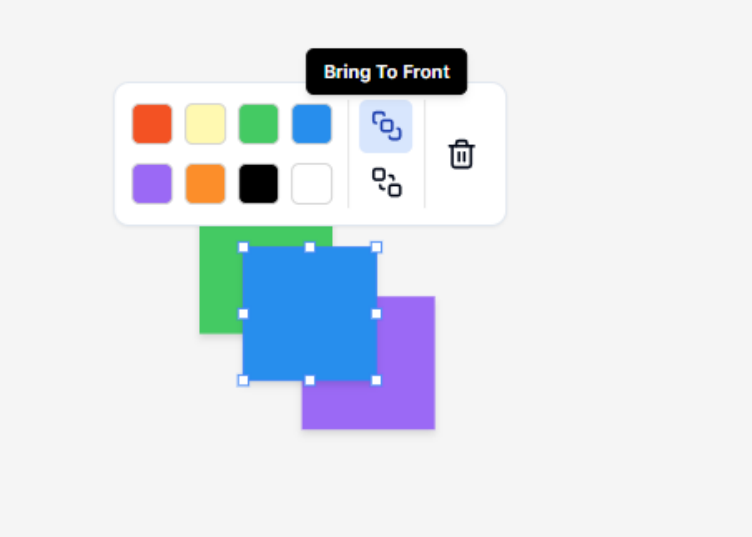


Рисунок В.21 – тестування функціоналу “Шарів”

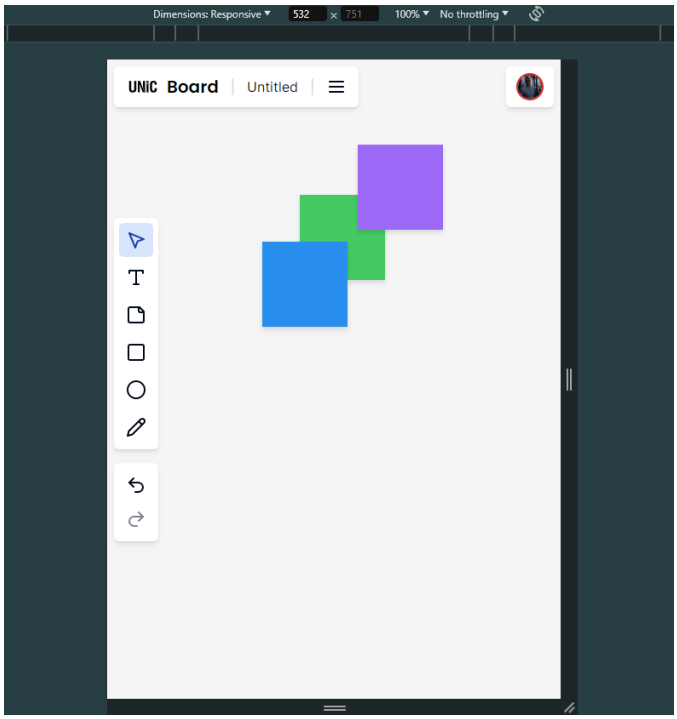


Рисунок В.22 – Тестування адаптивності веб-застосунку

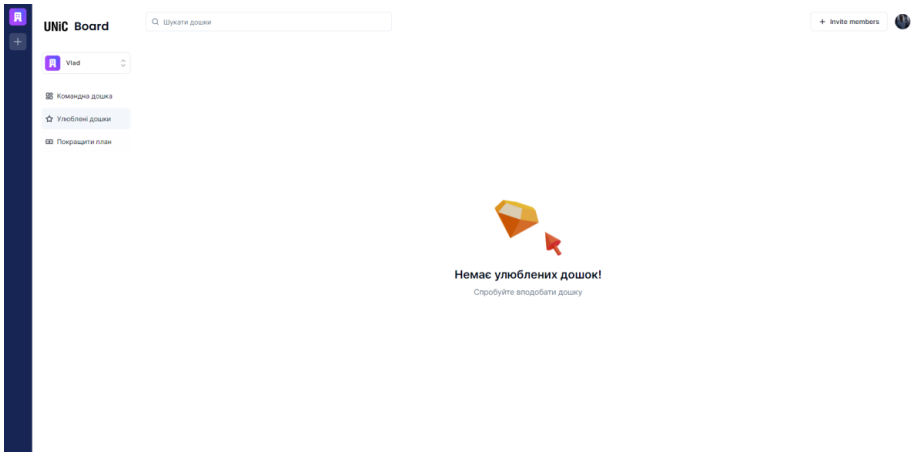


Рисунок В.23 – тестування зручності інтерфейсу

# ДОДАТОК Г (довідниковий)

## Інструкція користувача

1. Для початку роботи з додатком вам необхідно авторизуватися. Натисність на кнопку увійти за допомогою Google або Apple, також можете авторизуватись за допомогою email адресу.

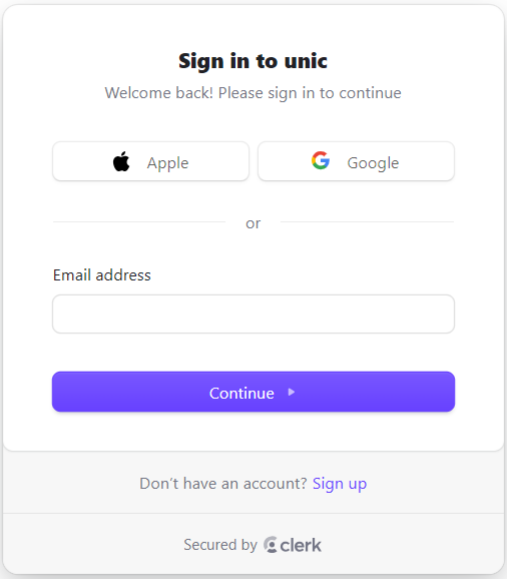


Рисунок Г.1 – Сторінка входу у веб-застосунок для візуалізації алгоритмів та схем

2. Далі потрібно створити нову дошку, натиснувши на відповідну кнопку

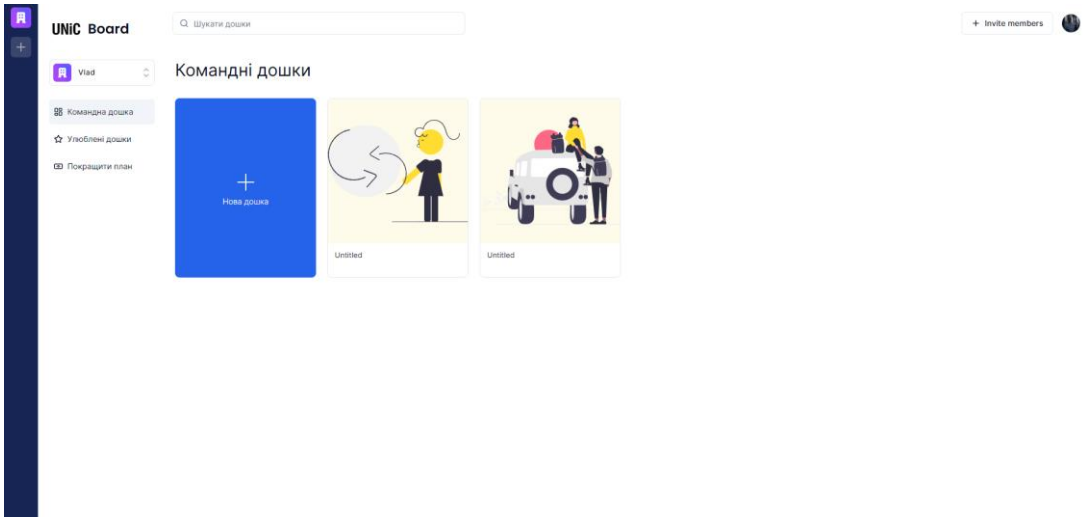

Рисунок Г.2 – Головна сторінка веб-застосунку для візуалізації алгоритмів та схем

3. Далі перед собою користувач побачить інтерфейс новоствореної дошки.

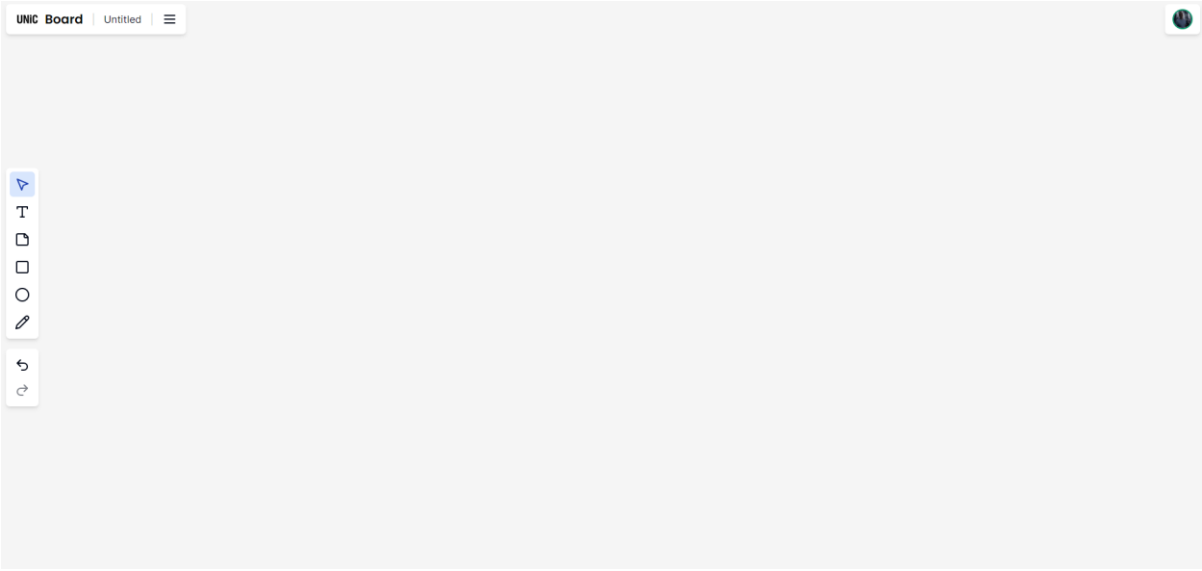


Рисунок Г.3 – Робоча область  веб-застосунку для візуалізації алгоритмів та схем

4. Ліва верхня панель містить наступні інструменти:

- Вибрати елемент
- Додати текст
- Додати замітку
- Додати квадрат/прямокутник
- Додати коло
- Ручка

Натомість у лівій нижні панелі можемо побачити інструменти, що допомагають повернути дію назад або вперед.

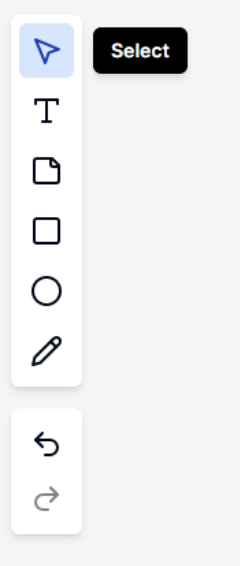


Рисунок Г.4 – Ліва бічна панель інструментів веб-застосунку для візуалізації алгоритмів та схем

5. Після додавання фігури, перед користувачем постає нова панель, а саме панель, що стосується конкретної фігури чи об’єкта.

На ній присутні наступні інструменти

- Змінити колір
- Перемістити елемент на передній план
- Перемістити елемент на задній план
- Видалити елемент

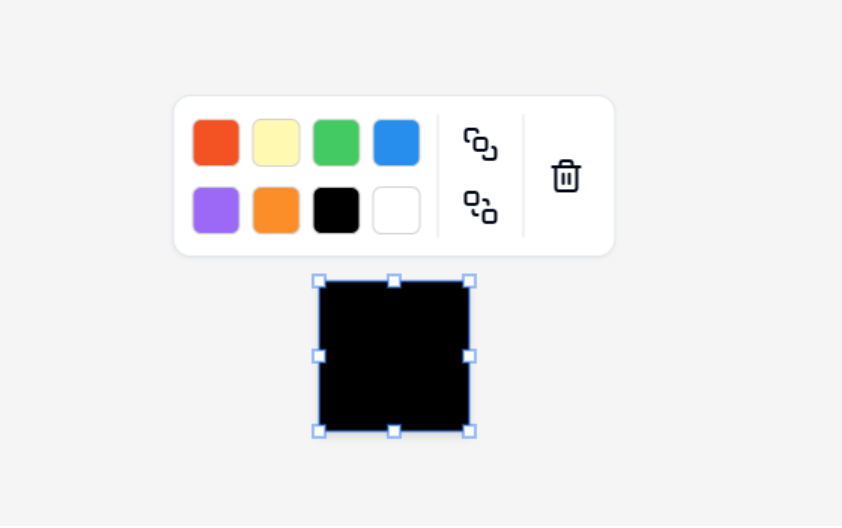

Рисунок Г.5 – Панель інструментів для певного елемента  веб-застосунку для візуалізації алгоритмів та схем

7. У верхій ліві частині користувач може змінити назву дошки. Її зберігання відбувається автоматично.

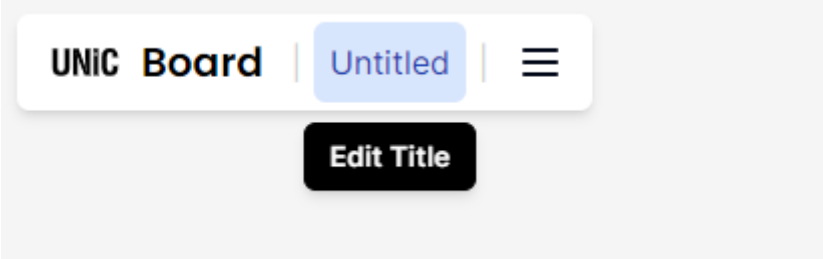


Рисунок Г.6 – Верхній блок програми, де користувач задає ім’я дошці веб-застосунку для візуалізації алгоритмів та схем

# ДОДАТОК Д (довідниковий)

## Довідка про впровадження

**ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «АГРАРНА ПЛАТФОРМА»**

*ЄДРПОУ 45317088, ІПН 453170802284,*
*UA923052990000026009046115049 в АТ КБ «ПриватБанк»,*
*Юридична адреса: 21015, м. Вінниця, вул. Лугова, будинок 57,*
*електронна адреса: aonlineplatform@gmail.com,*
*тел. 067-367-69-09*

Вих. № 26 від 14.06.2024р.

ДОВІДКА

Довідка видана студенту групи 1КН20б, Полігасу Владиславу Володимировичу, про те, що результати одержані ним в процесі виконання бакалаврської дипломної роботи, а саме алгоритми та програмні засоби, плануються до впровадження розробки ТОВ «Аграрна Платформа»

Довідка видана для пред'явлення за місцем вимоги .

Директор Олександр ЗАЄЦЬ