МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК СТВОРЕННЯ ТА РЕДАГУВАННЯ БЛОК-СХЕМ І ДІАГРАМ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Олександра БАЛІНСЬКА**

«__» _____ 2026 р.

Керівник роботи
старша викладачка _____ **Світлана БОРОВЛЬОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувачки

Балінська Олександра

1. Тема кваліфікаційної роботи Вебзастосунок створення та редагування блок-схем і діаграм затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «18» червня 2026 р.
3. Очікуваний результат роботи: розроблений вебзастосунок відповідно до теми роботи.
4. Перелік питань, що підлягають розробці: предметна область створення та використання засобів візуального моделювання, аналіз існуючих програмних рішень для побудови блок-схем і діаграм, проєктування архітектури вебзастосунку та структури зберігання даних, реалізація модулів

бізнес-логіки та користувацького інтерфейсу для створення і редагування діаграм, розробка механізмів збереження, редагування та експорту створених графічних моделей.

5. Перелік графічних матеріалів: презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «26» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок створення та редагування блок-схем і діаграм**

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КБР	26.12.2025	26.12.2025	Виконано
2	Огляд літератури за темою роботи	29.12.2025	31.12.2025	Виконано
3	Складання календарного плану КБР	03.01.2026	06.01.2026	Виконано
4	Аналіз предметної області	09.01.2026	12.01.2026	Виконано
5	Розробка проєктних рішень	15.01.2026	18.01.2026	Виконано
6	Моделювання та конструювання ПЗ	21.01.2026	25.02.2026	Виконано
7	Кодування та тестування розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	28.03.2026	15.05.2026	Виконано
8	Відгук керівника КБР	18.05.2026	19.05.2026	Виконано
9	Оформлення КБР та презентації	22.05.2026	24.05.2026	Виконано
10	Попередній захист	27.05.2026	27.05.2026	Виконано
11	Завершення оформлення КБР та презентації	01.06.2026	06.06.2026	Виконано
12	Рецензування	09.06.2026	12.06.2026	Виконано
13	Захист кваліфікаційної роботи			

Здобувач _____ **Олександра БАЛІНСЬКА**

«26» грудня 2025 р.

Керівник роботи

ст. викладачка _____ **Світлана БОРОВЛЬОВА**

«26» грудня 2025 р.

АНОТАЦІЯ

До кваліфікаційної бакалаврської роботи

Вебзастосунок створення та редагування блок-схем і діаграм

Здобувачка 409 гр.: Балінська Олександра

Керівник: ст. викладачка Боровльова Світлана

Актуальність. Широке використання засобів візуального моделювання у сфері розробки ПЗ, бізнес-аналізу та проєктування інформаційних систем зумовлює необхідність створення ефективних інструментів для побудови діаграм. Блок-схеми, UML-діаграми та ER-моделі є важливими засобами аналізу та документування програмних рішень. Водночас існуючі інструменти не завжди поєднують зручність використання, гнучкість і підтримку різних типів діаграм в одному застосунку.

Метою роботи є розробка вебзастосунку для створення та редагування діаграм із підтримкою локальної роботи та серверного зберігання даних.

Об’єктом роботи є процеси проєктування та розробки вебзастосунку для створення і редагування блок-схем та діаграм.

Предметом роботи є методи та засоби програмної інженерії для реалізації клієнт-серверного застосунку візуального моделювання.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі визначено актуальність, мету, завдання, об’єкт і предмет роботи. У першому розділі проаналізовано існуючі рішення. У другому розглянуто технології та сформульовано вимоги до системи. У третьому описано проєктування системи з використанням діаграм. У четвертому розділі розглянуто розробку основних модулів застосунку. У висновках узагальнено результати роботи.

КРБ викладена на 72 сторінки, вона містить 4 розділи, 27 ілюстрацій, 21 таблиць, 22 джерел в переліку посилань.

Ключові слова: вебзастосунок, редактор діаграм, блок-схеми, React, TypeScript, Node.js, Express, PostgreSQL, SVG.

ABSTRACT

to the qualifying bachelor's thesis

Web-based application for flowchart and diagram creation and editing

Student of 409 group: Balinska Oleksandra

Supervisor: senior lecture Borovlova Svitlana

Relevance. The widespread use of visual modeling tools in software development, business analysis, and information systems design requires effective solutions for diagram creation. Flowcharts, UML diagrams, and ER models are essential tools for analysis and documentation. However, existing tools do not always combine usability, flexibility, and support for multiple diagram types within a single application.

The goal of the work is to develop a web application for creating, editing, and storing flowcharts and diagrams with support for both local user operations and server-side data storage.

The object of the work is the process of designing and developing a web application for creating and editing flowcharts and diagrams.

The subject of the work is software engineering methods and tools for implementing a client-server visual modeling system.

The thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction defines the relevance, purpose, objectives, object, and subject of the research. The first chapter analyzes existing solutions. The second describes technologies and system requirements. The third presents system design using diagrams. The fourth focuses on the development of core application modules. The conclusions summarize the results.

The qualification work is presented on 72 pages, it consists of 4 sections, 27 figures, 21 tables, a list of references with 22 titles.

Keywords: web application, diagram editor, flowchart, React, TypeScript, Node.js, Express, PostgreSQL, SVG.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 3
ВСТУП 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 5
1.1 Опис предметної області 5
1.2 Функціональні особливості вебзастосунку 6
1.3 Аналіз існуючих реалізацій аналогічних систем 8
Висновки до розділу 1 15
2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ 16
2.1 Аналіз сучасного стану інструментарію, моделей та методів 16
2.2 Моделювання предметної області 19
2.3 Специфікація вимог до програмного забезпечення 22
Висновки до розділу 2 26
3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ТА ОГЛЯД ТЕХНОЛОГІЙ 28
3.1 Розробка UML-діаграм 28
3.2 Варіанти використання системи 38
3.3 Проєктування архітектури та вибір технологічного стеку 44
3.4 Мокапи інтерфейсу 46
Висновки до розділу 3 50
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ 52
4.1 Структура коду та специфікації 52
4.2 Тестування програмного забезпечення 57
4.3 Результати рішення 60
4.4 Керівництво користувача 62
Висновки до розділу 4 70
ВИСНОВКИ 71
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 72

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ПК – персональний комп’ютер

ПЗ – програмне забезпечення

СКБД – система управління базами даних

API – інтерфейс програмування застосунків (Application Programming Interface)

BPMN – нотація моделювання бізнес-процесів (Business Process Model and Notation)

DOM – об’єктна модель документа (Document Object Model)

ER – модель «сутність-зв’язок» (Entity-Relationship)

HTML – мова гіпертекстової розмітки (HyperText Markup Language)

HTTP – протокол передавання гіпертексту (HyperText Transfer Protocol)

HTTPS – захищений протокол передавання гіпертексту (HyperText Transfer Protocol Secure)

JSON – текстовий формат обміну даними (JavaScript Object Notation)

LDA – латентне розміщення Діріхле (Latent Dirichlet Allocation)

REST – архітектурний стиль взаємодії вебсервісів (Representational State Transfer)

SPA – односторінковий вебзастосунок (Single Page Application)

SVG – масштабована векторна графіка (Scalable Vector Graphics)

UI – користувацький інтерфейс (User Interface)

UML – уніфікована мова моделювання (Unified Modeling Language)

UX – користувацький досвід (User Experience)

XML – розширювана мова розмітки (Extensible Markup Language)

ВСТУП

Актуальність теми обумовлена широким використанням засобів візуального моделювання у сфері розробки програмного забезпечення, бізнес-аналізу та проєктування інформаційних систем. Блок-схеми, UML-діаграми, ER-моделі та інші графічні представлення процесів є важливим інструментом аналізу та документування. Незважаючи на велику кількість існуючих рішень, вони не завжди поєднують зручність, гнучкість і підтримку різних типів діаграм в одному рішенні,що ускладнює роботу при створенні складних схем.

Метою роботи є розробка вебзастосунку для створення, редагування та збереження блок-схем і діаграм із можливістю локальної роботи користувача та серверного зберігання даних.

Для досягнення поставленої мети треба вирішити такі **завдання**:

1) аналіз сучасних програмних засобів для створення діаграм та визначення їх функціональних можливостей;
2) формування функціональних і структурних вимог до розроблюваної системи;
3) проєктування клієнт-серверної архітектури вебзастосунку;
4) розробка механізмів створення та редагування графічних елементів на полотні;
5) реалізація збереження, відновлення, та експорту діаграм;
6) визначення ролей користувачів і сценаріїв взаємодії із системою;
7) проведення тестування програмного забезпечення.

Об’єктом роботи є процес проєктування та розробки вебзастосунку для створення і редагування блок-схем та діаграм.

Предметом роботи є методи та засоби програмної інженерії, що використовуються для реалізації клієнт-серверного вебзастосунку візуального моделювання, зокрема архітектурні рішення, механізми збереження даних а також організація взаємодії користувача з графічним інтерфейсом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка вебзастосунків для візуального моделювання передбачає не лише реалізацію окремих функцій, а й розуміння загальних підходів до побудови подібних систем. На сучасному етапі існує велика кількість рішень, які відрізняються як за функціональністю, так і за архітектурними підходами. Тому перед безпосереднім проєктуванням доцільно провести аналіз предметної області, визначити основні принципи роботи таких систем, а також виявити їх сильні та слабкі сторони. Це дозволяє сформувати обґрунтовані вимоги до майбутнього вебзастосунку та уникнути типових помилок при його розробці.

1.1 Опис предметної області

Розробка вебзастосунків для візуалізації інформації у сучасних умовах є досить поширеною практикою [1; 2]. Особливо це стосується інструментів, які дозволяють створювати діаграми, блок-схеми та інші графічні моделі. Подібні засоби активно застосовуються у випадках, коли необхідно представити інформацію не лише текстово, а у вигляді структурованої схеми.

У межах даної роботи розглядається процес створення та редагування графічних моделей безпосередньо у браузері. У такому підході користувач взаємодіє з вебзастосунком замість десктопної програми, що з одного боку накладає певні обмеження, а з іншого – забезпечує гнучкість та доступність.

Діаграми широко використовуються у різних сферах, зокрема:

- у розробці ПЗ (UML-діаграми для опису систем);
- у бізнес-аналітиці (flowchart, BPMN для процесів);
- при роботі з БД (ER-діаграми);
- у навчанні (для пояснення складних тем).

Незалежно від області застосування, основна задача таких систем полягає у відображенні структури та взаємозв’язків між елементами. Водночас

способи реалізації цього можуть суттєво відрізнятись. Основна ідея таких систем – дати користувачу можливість створювати графічні об’єкти і пов’язувати їх між собою. На практиці це не є тривіальною задачею, оскільки необхідно забезпечити коректну поведінку елементів при зміні їх положення, розміру та взаємодії між собою.

Базові функції в таких системах стандартні: створення елементів, їх редагування, переміщення, зв’язування. Але проблема не в наявності функцій, а в тому як саме вони реалізовані. У багатьох існуючих рішеннях інтерфейс є перевантаженим, що ускладнює роботу користувача [3]. Крім того, виникають труднощі при роботі з великими діаграмами, коли кількість елементів значно зростає. Окрему проблему становить побудова зв’язків між об’єктами, яка у ряді випадків потребує додаткових дій і точного позиціонування. Тому актуальною залишається задача створення вебзастосунку, який забезпечує достатній рівень функціональності при збереженні зрозумілого та зручного інтерфейсу.

1.2 Функціональні особливості вебзастосунку

Вебзастосунок, що розроблюється, призначений для створення та редагування діаграм безпосередньо у браузері без необхідності встановлення додаткового ПЗ. Такий підхід спрощує доступ до системи та дозволяє працювати з нею у різних середовищах.

Основна мета застосунку полягає у наданні користувачу інструменту, який дозволяє швидко створювати графічні моделі та виконувати базові операції без зайвих дій. Робота користувача здійснюється у межах робочої області, де розміщуються графічні елементи. Кожен елемент має набір параметрів, зокрема координати, розмір, текстове наповнення та візуальні характеристики. Важливою умовою є стабільність цих параметрів при взаємодії між елементами.

Основний функціонал застосунку включає:

– створення базових графічних елементів (прямокутники, еліпси, ромби тощо);
– редагування тексту всередині фігур;
– зміни розмірів та переміщення об’єктів;
– налаштування стилів (колір, прозорість, контур);
– видалення елементів.

Цей набір виглядає базовим, але на практиці саме він використовується найчастіше, тому його реалізація повинна бути максимально простою, без зайвих кроків. Окремо варто звернути увагу на механізм побудови зв’язків між елементами. Зв’язки можуть створюватися як окремі об’єкти, або одразу прив’язуватись до фігур. При цьому їх положення автоматично оновлюється під час переміщення елементів. Це важливо, бо в багатьох системах саме тут виникають проблеми – стрілки можуть залишатись «в повітрі» або змішуватись не туди куди треба.

Інтерфейс організований у вигляді декількох панелей. Ліва панель містить список фігур, центральна частина відведена під робочу область, а права використовується для редагування властивостей. Користувач може змінювати розміри панелей або взагалі їх приховувати, що трохи спрощує роботу, особливо коли потрібно більше місця під саму діаграму.

З технічної точки зору застосунок побудован з використаням React та SVG [4; 5; 6]. Такий підхід дозволяє досить швидко відображати елементи і дає більше контролю над ними. Хоча SVG іноді може бути не таким продуктивним як Canvas. Але в цьому випадку гнучкість важливіша, ніж максимальна швидкість. Крім базового функціоналу, у системі передбачено можливість роботи з галереєю діаграм. Користувач може зберігати власні роботи, а також переглядати або використовувати чужі шаблони. Таким

способом спрощується процес створення нових схем. Особливо якщо не хочеться починати з нуля.

Також реалізовано елементи модерації, які дозволяють контролювати контент у галереї. Це потрібно для того, щоб уникнути появи випадкових або некоректних матеріалів в публікаціях. Без такого контролю система може швидко наповнитись некоректними, шкідливими або випадковими матеріалами.

1.3 Аналіз існуючих реалізацій аналогічних систем

Після опису основних функціональних можливостей доцільно провести аналіз існуючих аналогів, оскільки це дозволяє визначити сильні та слабкі сторони сучасних систем для побудови діаграм. На сьогодні існує велика кількість вебзастосунків, призначених для створення блок-схем, UML-діаграм, ER-моделей та інших графічних структур. Дослідники розглядають три аспекти створення інтерфейсу редактора: як зберігатимуться дані та як користувач взаємодіятиме з графічними елементами. Цей аналіз аналогів надає метод для виявлення функцій, які потрібні користувачам. Також включені обмеження існуючих рішень, які також необхідно враховувати під час створення власного вебзастосунку. Прикладами таких рішень є diagrams.net (draw.io), Lucidchart та Creately. Інструменти, які вони надають, їхні інтерфейси, формати, які вони підтримують, та спосіб, яким вони забезпечують співпрацю, – це те, що відрізняє ці три різні рішення.

Вебзастосунок diagrams.net (draw.io)

Серед доступних інструментів для побудови діаграм окремо варто виділити diagrams.net (рисунок 1.1) [7]. Даний сервіс відноситься до безкоштовних рішень і підтримує широкий спектр типів діаграм, зокрема блок-схеми, UML та ER-моделі.

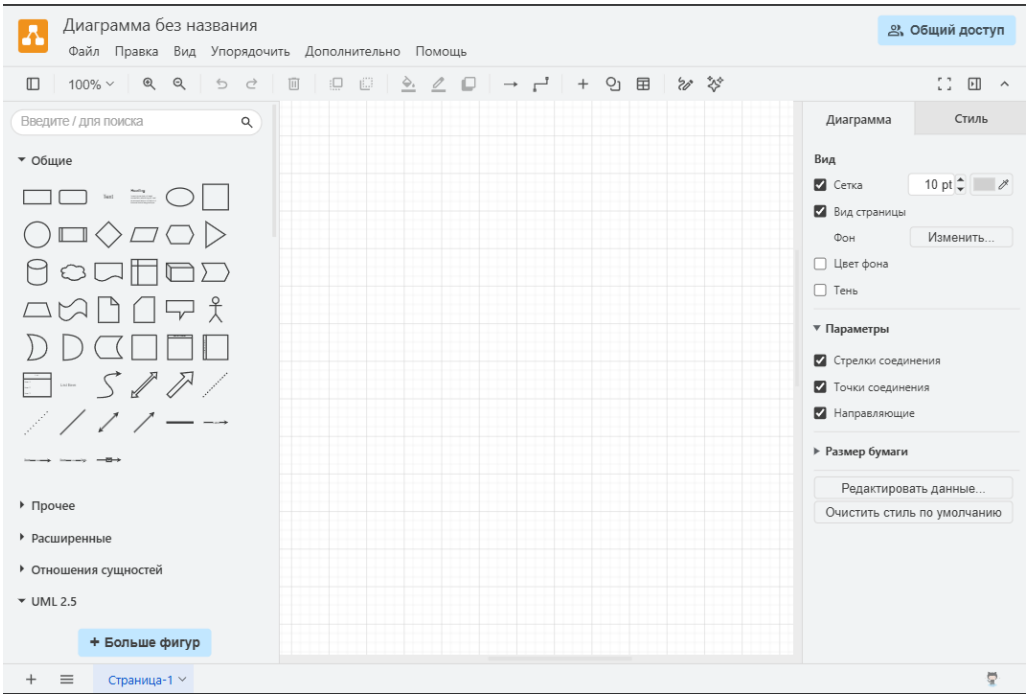

Рисунок 1.1 – Інтерфейс сервісу diagrams.net

З технічної точки зору значна частина обробки виконується безпосередньо на стороні клієнта. Це означає, що взаємодія з елементами відбувається без постійних запитів до серверу, що позитивно впливає на швидкість роботи, особливо при створенні складних або великих структур.

Переваги:

- великий набір інструментів і шаблонів;
- безкоштовне використання;
- підтримка різних форматів імпорту та експорту;
- можливість роботи без серверної частини.

Недоліки:

- складний інтерфейс для нових користувачів;
- перевантаженість функціоналом;
- обмежені можливості спільної роботи;
- не завжди зручна навігація.

Таблиця 1.1 – Технічні характеристики diagrams.net

Критерій	**diagrams.net**	**Проєктований вебзастосунок**
Фронтенд	JavaScript, HTML5, використання mxGraph (нині diagrams.net core)	React, TypeScript
Рендеринг	SVG, Canvas	SVG (з Canvas fallback)
Архітектура	SPA (client-heavy)	SPA
Backend	Відсутній (опціонально Node.js для інтеграцій)	Node.js, Express
API	REST (інтеграції Google Drive, OneDrive)	REST
Зберігання	LocalStorage, Google Drive / Dropbox	PostgreSQL, cloud storage
Формати діаграм	XML	JSON
Побудова зв’язків	Ручна через control points	Автоматична, snap до вузлів
Колаборація	Обмежена (через зовнішні сервіси)	Не основна
Масштаб полотна	Infinite canvas	Обмежений canvas
Продуктивність	Висока	Висока

У результаті аналізу визначено, що diagrams.net є потужним та гнучким інструментом. Але його складність може негативно впливати на зручність використання системи.

Вебзастосунок Lucidchart

Інший підхід реалізовано у системі Lucidchart (рисунок 1.2), яка більше орієнтована на спільну роботу користувачів [8]. У даному випадку основний акцент зроблено не стільки на локальну продуктивність, скільки на синхронізації дій між кількома користувачами.

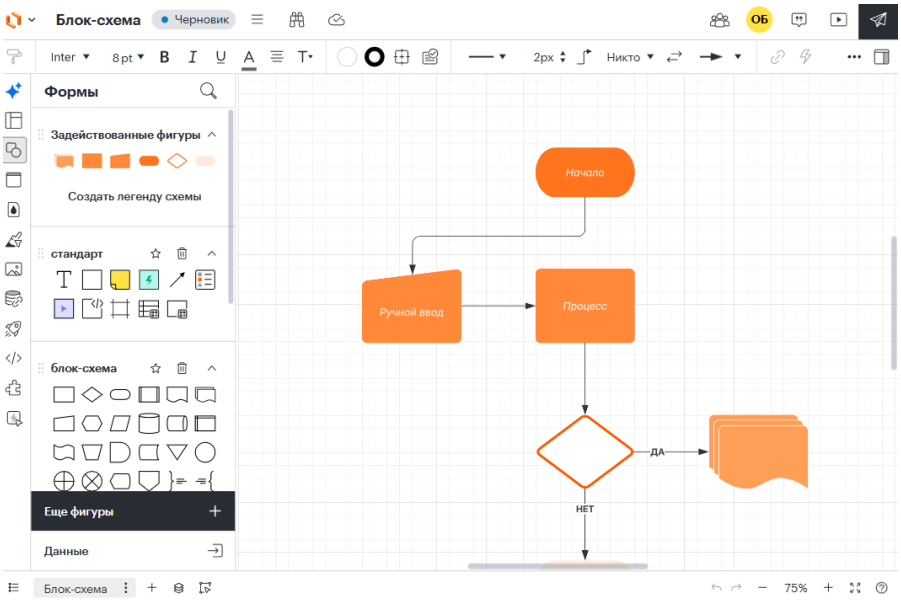

Рисунок 1.2 – Інтерфейс сервісу Lucidchart

Застосунок використовує клієнт-серверну модель. Всі зміни передаються через мережу у режимі реального часу. Кілька користувачів можуть редагувати подібний документ одночасно. Однак система залежить від стабільного з'єднання із сервером. Інтерфейс системи виглядає більш організоованим у порівнянні з diagrams.net, але частина функціональних можливостей доступна лише після оформлення підписки.

Переваги:

– підтримка спільної роботи в режимі реального часу;
– грані можливості синхронізації змін;
– інтеграція з іншими програмами;
– історія редагування.

Недоліки:

- більшість функцій доступні лише за умови оплати;
- обмеження безкоштовної версії;
- залежність від інтернет-з’єднання;
- відносно висока складність для нових користувачів.

Таблиця 1.2 – Технічні характеристики Lucidchart

Критерій	**Lucidchart**	**Проєктований вебзастосунок**
Фронтенд	React/Angular	TypeScript, React
Архітектура	Client-server	SPA, backend API
Backend	Cloud-based	Node.js
API	REST, WebSocket	REST
Синхронізація	WebSocket, real-time updates	Відсутня (або базова)
Зберігання	Хмарне	PostgreSQL
Формат даних	JSON	JSON
Побудова зв’язків	Напівавтоматична	Автоматична, прив’язка
Колаборація	Real-time (multi-user editing)	Не передбачено
Масштаб	Великий, server-managed	Обмежений
Продуктивність	Залежить від мережі	Незалежна від мережі

Lucidchart є ефективним інструментом для командної роботи, однак його використання обмежується платною моделлю доступу.

Вебзастосунок Creately

Ще один варіант реалізації представлено у Creately (рисунок 1.3), де акцент зроблено на спрощення взаємодії з користувачем [9]. У порівнянні з

попередніми системами тут менше складних налаштувань, але більше автоматизації.

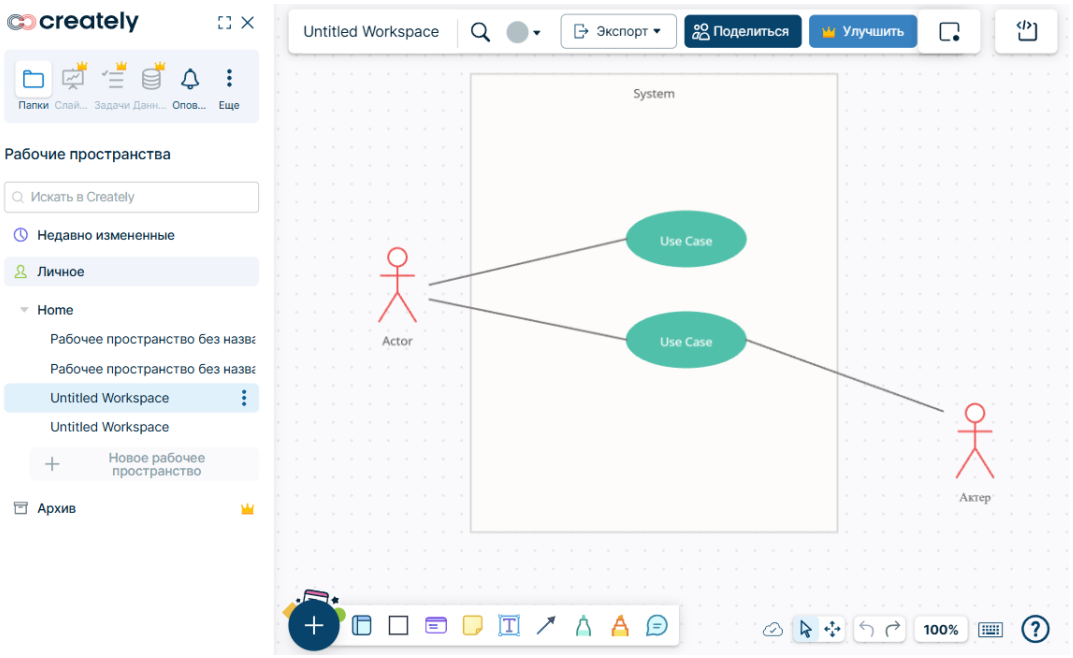

Рисунок 1.3 – Інтерфейс сервісу Creatly

Побудова зв'язків між частинами щойно виконувалася автоматично одразу. Це дозволяє скоротити ручну роботу та пришвидшити побудову схем. Однак цей метод обмежує можливість налаштування дрібних деталей, занадто багато деталей може ускладнити роботу зі структурою.

Переваги включають:

– простий та зрозумілий інтерфейс;
– легке створення діаграм;
– автоматичні зв’язки між елементами;
– сучасний дизайн.

Недоліки включають:

– обмежені можливості;
– деякі елементи потрібно купувати додатково;
– менша гнучкість налаштувань;
– обмежене використання складних схем.

Таблиця 1.3 – Технічні характеристики Creately

Критерій	**Creately**	**Проєктований вебзастосунок**
Фронтенд	JavaScript SPA	TypeScript, React
Рендеринг	SVG	SVG
Архітектура	SPA, cloud backend	SPA
Backend	Cloud (AWS / Firebase-подібна модель)	Node.js
API	REST	REST
Зберігання	Хмарне	PostgreSQL
Формат	JSON	JSON
Побудова зв’язків	Автоматична (snap, smart connectors)	Автоматична, anchor points
Колаборація	Часткова	Відсутня
Масштаб	Обмежений	Контрольований
UX	Простий	Мінімалістичний

Creately є зручним для швидкого створення діаграм, але має обмеження у функціональності.

Порівняльний аналіз систем

Таблиця 1.4 – Порівняльна характеристика систем-аналогів

Критерій	**diagrams.net**	**Lucidchart**	**Creately**	**Розроблюваний застосунок**
Архітектура	SPA	Клієнт-сервер	SPA,сервер	SPA
Зберігання даних	Локальне	Хмара	Хмара	БД

Кінець таблиці 1.4

Побудова зв’язків	Ручна	Напівавто	Авто	Авто, прив’язка
Спільна робота	Обмежена	Повна	Часткова	Не основна
Складність інтерфейсу	Висока	Середня	Низька	Низька
Продуктивність	Висока	Середня	Висока	Висока

Висновки до розділу 1

Аналіз існуючих вебзастосунків показує використання різних підходів до реалізації клієнтської частини, зокрема SPA-архітектури із застосуванням JavaScript-фреймворків та рендерингом через SVG або Canvas. Обробку даних у diagrams.net виконує переважно сторона клієнта, тоді як Lucidchart та Creately використовують клієнтсерверну модель з синхронізацією через мережу. Встановлено, що підходи до побудови зв’язків між елементами відрізняються. Від повністю ручного керування до автоматичних механізмів прив’язки. Це безпосередньо впливає на швидкість роботи користувача та складність інтерфейсу. Порівняння технічних рішень показує доцільність використання клієнтської архітектури з рендерингом на базі SVG.

Фактори, що забезпечують ефективність системи, включають простоту використання інтерфейсу, швидкість взаємодії з елементами та стабільну роботу системи зі збільшенням кількості об'єктів. Також виявлено, що надмірна функціональність та складна навігація негативно впливають на враження користувача, незважаючи на всі можливості, які пропонує система. Завдяки проведеному аналізу визначено основні вимоги до розробки вебзастосунку. Вони включають розробку простої у використанні системи, забезпечення належної функціональності та ефективну систему обробки графіки, які ми врахуємо на наступному етапі проектування.

2 МОДЕЛЮВАННЯ ОБ’ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

Розробка вебзастосунку для створення та редагування діаграм потребує розуміння того, як саме подібні системи будуються, та які підходи при цьому використовуються. Сучасні рішення відрізняються архітектурою, способом обробки даних та організацією взаємодій з користувачем. Аналіз доступних інструментів і технологій дозволяє визначити, які з них доцільно використовувати у проєкті, а також уникнути типових проблем, що виникають при реалізації подібних систем.

2.1 Аналіз сучасного стану інструментарію, моделей та методів

У сучасних наукових публікаціях, присвячених розробці вебзастосунків для візуального моделювання, основна увага зосереджується на ефективності побудови графічних інтерфейсів, швидкості обробки взаємодії користувача і гнучкості представлених даних. Для проведення аналізу використано матеріали з відкритих наукових джерел та технічної документації сучасних технологій (зокрема React, SVG, PostgreSQL, Web APIs) [4; 5; 6; 10], що дозволить оцінити актуальний стан інструментарію. У результаті узагальнення матеріалів можна виділити три основні напрями:

- клієнтські технології для побудови інтерфейсів;
- спосіб рендерингу графічних об’єктів;
- методи зберігання і обробки даних.

Для систематизації отриманих результатів застосовано LDA, що дозволяє виділити ключові кластери термінів, які найчастіше зустрічаються у дослідженнях.

Ключові спостереження

Аналіз джерел показує, що у більшості сучасних рішень:

– JavaScript-фреймворки (передусім React) згадуються як основний інструмент побудови SPA-застосунків, приблизно у 70% розглянутих матеріалів;

– рендеринг графіки у вебзастосунках найчастіше реалізується через SVG (близько 60%), рідше використовується Canvas, переважно у випадках високих вимог до продуктивності;

– архітектура SPA зустрічається більш ніж у половині прикладів, що пояснюється потребою у швидкій взаємодії без перезавантаження сторінки;

– для представлення даних використовується формат JSON, що дозволяє легко зберігати структуру діаграми у вигляді вузлів та зв’язків;

– механізми побудови зв’язків між елементами все частіше базуються на автоматичних алгоритмах (snap, anchor points), що зменшує навантаження на користувача.

Таблиця 2.1 – Розподіл тем у дослідженнях

Напрям (кластер LDA)	**Частка публікацій, %**	**Типові ключові слова**
Клієнтські інтерфейси	високий	react, spa, typescript, state, ui
Рендеринг графіки	високий	svg, canvas, rendering, vector
Архітектура застосунків	середній	spa, client-side, modular
Зберігання даних	середній	json, schema, serialization
UX та взаємодія	середній	interaction, drag-drop, usability

Отримані результати дозволяють зробити висновок, що розвиток вебзастосунків для роботи з графічними моделями орієнтований на перенесення логіки на клієнтську сторону та підвищення інтерактивності.

Текстовий виклад результатів

У більшості сучасних вебзастосунків використовується SPA-підхід, що дозволяє уникнути перезавантаження сторінок та забезпечити швидку

взаємодію з інтерфейсом. Дослідження багаторівневих вебархітектур показують доцільність розділення клієнтської та серверної частин системи для підвищення гнучкості та спрощення підтримки програмного забезпечення [14]. Окремі дослідження у сфері інтерактивних візуалізацій також підтверджують важливість швидкої обробки дій користувача та ефективного відображення графічних об'єктів у вебінтерфейсі [15]. React є дуже поширеним інструментом, оскільки підтримує компонентну модель та ефективне управління станом [4]. Для типізації даних і зменшення кількості помилок під час розробки часто використовується TypeScript [10]. У сучасних SPA-застосунках також застосовуються lightweight state-менеджери, зокрема Zustand [16], який дозволяє централізовано керувати станом інтерфейсу без складної конфігурації. Для швидкого запуску та збирання frontend-застосунків використовується Vite [17], який забезпечує швидку компіляцію та оптимізацію проєкту.

Для відображення графічних елементів найчастіше використовується SVG [18]. Це пов’язано з тим, що SVG дозволяє працювати з кожним елементом як окремим об’єктом, що є критичним для задач редагування діаграм. Canvas, хоча і забезпечує кращу продуктивність у деяких сценаріях, ускладнює взаємодію з окремими елементами.

У сфері зберігання даних переважає використання JSON-структур [19]. Діаграма описується як набір вузлів та зв’язків, що дозволяє легко передавати та зберігати її стан. Для серверної частини часто застосовуються реляційні СКБД, зокрема PostgreSQL, що підтримує роботу з JSON-полями.

Особливу увагу приділяють механізмам взаємодії користувача, таким як drag-and-drop, масштабування та побудова зв’язків. У нових підходах все частіше використовуються автоматизовані алгоритми прив’язки, що зменшують кількість помилок при побудові діаграм.

Таблиця 2.2 – Співставлення тенденцій та технологічного вибору

Галузевий тренд	**Використання у проєкті**	**Очікуваний ефект**
SPA-архітектура	React, TypeScript	Швидка взаємодія без reload
Клієнтська обробка	Логіка на фронтенді	Зменшення затримок
SVG-рердеринг	SVG як основа	Точність позиціонування
JSON-модель	Структура diagram JSON	Простота збереження
Автозв’язки	Snap, anchor points	Зменшення ручних дій

Проведений аналіз показує, що сучасний стан інструментарію у сфері вебзастосунків для візуального моделювання орієнтований на інтерактивність, гнучкість та клієнтську обробку даних. Обраний технологічний стек (React, SVG, PostgreSQL) відповідає цим тенденціям та забезпечує необхідний баланс між продуктивністю та зручністю роботи користувача.

2.2 Моделювання предметної області

Розроблюваний вебзастосунок орієнтований на створення та редагування діаграм у браузері, тому модель предметної області базується на представленні графічних об’єктів та зв’язків між ними. У центрі системи знаходиться не просто набір фігур, а структура, яка описує взаємозв’язки між елементами.

На відміну від класичних документів, діаграма розглядається як динамічна модель, що змінюється в реальному часі. Користувач взаємодіє з нею безпосередньо через інтерфейс, виконуючи операції створення, редагування та видалення елементів. При цьому система повинна забезпечити узгодженість стану – тобто щоб після будь-якої дії структура не «ламалась».

Основними сутностями предметної області є:

- графічний елемент (фігура);
- зв’язок між елементами;
- полотно (робоча область);
- діаграма як сукупність елементів;
- користувач (у контексті збереження та роботи з даними).

Графічний елемент представляє собою об’єкт із набором параметрів: координати, розмір, текст, стиль. Зв’язок між елементами описує логічне відношення між ними і має власні атрибути, зокрема тип лінії, напрямок та точки прив’язки.

Таблиця 2.3 – Основні сутності предметної області

Сутність	**Атрибути**	**Опис**
Елемент (Node)	id, position, size, text, style	базова одиниця діаграми
Зв’язок (Edge)	id, source, target, type	відображає відношення між елементами
Діаграма	id, name, nodes, edges	структура всієї моделі
Полотно	width, height, zoom	робоча область
Користувач	id, name	використовується для збереження

Взаємодія між сутностями відбувається у вигляді змін стану діаграми. Наприклад, при переміщенні елемента змінюються його координати, а також автоматично оновлюються прив’язані до нього зв’язки. Ручне оновлення таких залежностей значно ускладнює роботу. Також варто звернути увагу на модель збереження. Діаграма зберігається у вигляді JSON-структури, що містить масив елементів та зв’язків. Це дозволяє легко передавати дані між клієнтом і сервером та відновлювати стан після перезавантаження сторінки.

Схема взаємодії компонентів

Узагальнено процес роботи системи можна описати так:

– клієнтська частина (React) відображає інтерфейс і обробляє дії користувача [4; 10];

– при зміні стану (наприклад, створення елемента) оновлюється внутрішній state;

– за необхідності дані передаються на сервер через HTTP-запити;

– сервер (Node.js) виконує обробку і збереження у базі даних [18; 11];

– при повторному відкритті діаграми дані завантажуються і відновлюються у клієнті.

Таблиця 2.4 – Відповідність компонентів системи та їх функцій

Компонент	**Реалізація**	**Функція**
Клієнт	React , TypeScript	відображення інтерфейсу
Рендеринг	SVG	побудова графіки
Стан	Zustand	управління станом редактора
Frontend build tool	Vite	запуск і збирання SPA
API	REST	обмін даними
Сервер	Node.js	обробка запитів
БД	PostgreSQL	збереження діаграм

У процесі моделювання також враховується поведінка користувача. Основні сценарії включають створення нової діаграми, редагування елементів, побудову зв’язків та збереження результату. Важливо, що більшість дій виконується без затримок, оскільки логіка знаходиться на клієнті

2.3 Специфікація вимог до програмного забезпечення

На основі проведеного аналізу визначаються вимоги до ПЗ, які описують призначення системи, її функціональні можливості та обмеження. Так легше чітко зрозуміти, що саме має реалізовувати вебзастосунок і в яких межах він буде працювати.

1. Призначення та межі проєкту

1.1 Призначення системи

Вебзастосунок призначений для створення, редагування та збереження діаграм у середовищі браузера. користувач отримує доступ до робочої області, де може створювати графічні елементи, встановлювати між ними зв’язки та формувати структуровані схеми різного типу. Система дозволяє працювати без встановлення додаткового ПЗ, що спрощує доступ до функціоналу.

1.2. Погодження, що ухвалені в програмній документації

У межах проєкту прийнято використання клієнт-серверної моделі з SPA-архітектурою. Клієнтська частина реалізується на React та TypeScript, що дозволяє забезпечити структурованість коду та зменшити кількість помилок при розробці. Серверна частина реалізується на Node.js з використанням Express, що забезпечує обробку HTTP-запитів та взаємодію з базою даних. Для зберігання інформації використовується PostgreSQL, що підтримує роботу зі структурованими та напівструктурованими даними. Передбачено адаптивний інтерфейс із орієнтацією на використання на персональних комп’ютерах.

1.3 Межі проєкту

У межах поточної реалізації не передбачається: повноцінна багатокористувацька синхронізація в реальному часі, складні інтеграції із зовнішніми сервісами, розробка окремих мобільних застосунків. Також система не орієнтована на обробку надвеликих діаграм (тисячі елементів), хоча базова масштабованість передбачена.

2. Загальний опис

2.1 Сфера застосування

Застосунок використовується у навчанні, розробці ПЗ, бізнес-аналітиці та інших сферах, де необхідно виконувати візуальне представлення процесів або структур. Основна задача системи полягає у спрощенні створення діаграм та підвищенні наочності представлення інформації.

2.2 Характеристика користувачів

Користувачі без технічного досвіду (просте створення схем), користувачі з базовими знаннями (UML, логічні моделі), користувачі з досвідом, які працюють зі складнішими структурами.

2.3 Загальна структура системи

Клієнтської частини (React), що відповідає за інтерфейс і логіку взаємодії, серверної частини (Node.js), яка забезпечує обробку запитів і збереження даних, бази даних (PostgreSQL), де зберігаються діаграми.

2.4 Загальні обмеження

Для повноцінної роботи бажане стабільне інтернет-з’єднання, продуктивність залежить від кількості елементів, при великій кількості зв’язків можливі незначні затримки.

3. Функції системи

3.1 Функції роботи з користувачем

Опис функції забезпечує базову взаємодію користувача із системою, включаючи відкриття, створення та редагування діаграм.

Вхідна інформація включає дії користувача (введення тексту, кліки, переміщення об’єктів).

Вихідною інформацією є оновлений стан інтерфейсу та діаграми;

Функціональні вимоги - інтерфейс повинен реагувати без затримок, зміни повинні відображатись одразу, не допускається «втрата» стану при взаємодії.

3.2 Функції створення та редагування елементів

Опис функції: дозволяє створювати графічні елементи (фігури), змінювати їх параметри та переміщувати по робочій області.

Вхідна інформація – тип елемента, координати, текст.

Вихідна інформація – оновлений список елементів діаграми.

Функціональні вимоги - елемент має зберігати свої параметри при зміні, переміщення не повинно впливати на інші елементи, редагування тексту має бути стабільним.

3.3 Функції побудови зв’язків:

Опис функції - забезпечує створення зв’язків між елементами з автоматичним оновленням їх положення.

Вхідна інформація - початковий та кінцевий елемент.

Вихідна інформація - створений зв’язок.

Функціональні вимоги - зв’язок повинен бути прив’язаний до елементів, при переміщенні елементів зв’язок оновлюється, не повинно бути ситуацій, коли стрілка «йде під фігуру» або губиться.

3.4 Функції збереження та відновлення:

Опис функції - забезпечує збереження діаграми та її подальше відновлення.

Вхідна інформація - структура діаграми у форматі JSON.

Вихідна інформація - збережений запис або файл.

Функціональні вимоги - дані не повинні втрачатись, структура має повністю відновлюватись, формат повинен бути розширюваним.

3.5 Функції гелереї:

Опис функції - дозволяє переглядати та використовувати діаграми як шаблони.

Вхідна інформація - запит користувача.

Вихідна інформація - список доступних діаграм.

Функціональні вимоги - швидке завантаження, можливість відкриття, базова фільтрація.

3.6. Функції модерації:

Опис функції - контроль за контентом у галереї.

Функціональні вимоги - можливість видалення некоректних записів, обмеження небажаного контенту, не впливає критично на швидкість системи.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і вхідна інформація

Вхідна інформація формується на основі дій користувача, включаючи створення та редагування елементів, введення текстових даних, зміну параметрів об’єктів.

4.2 Організація даних

Основне сховищем, яке використовується для зберігання даних, є реляційна база даних (PostgreSQL). Діаграми зберігаються як JSON-структури, що представляють елементи та зв'язки. Такий підхід спрощує роботу та робить гнучкою зміну структури даних.

4.3 Обробка даних

Обробка даних відбувається і на стороні клієнта (React), і на стороні сервера, де сервер обробляє запити, перевіряє дані та зберігає їх у базі даних. Наявність певної обробки на стороні клієнта також допомагає зробити користувацький інтерфейс дуже швидким.

5. Вимоги до технічного забезпечення

Сучасний браузер із підтримкою SVG та JavaScript.

Персональний комп’ютер або ноутбук.

Стабільне підключення до мережі Інтернет для доступу до серверної частини.

6. Вимоги до програмного забезпечення

Архітектура - система побудована методом односторінкового застосунку (SPA), що включає серверну та клієнтську частини.

Системне ПЗ - Node.js використовується для виконання серверної логіки.

Мережева взаємодія - передача даних здійснюється через HTTPS із використанням REST API.

База даних - PostgreSQL для підтримки всієї структурованої інформації.

Мова і технології – ReactJs, TypeScript, NodeJs.

7. Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача - інтерфейс складається з робочої області, панелей інструментів та панелі властивостей.

Апаратний інтерфейс - система працює на стандартних пристроях без необхідності додаткового обладнання.

Програмний інтерфейс - передбачено REST API для взаємодії між клієнтом і сервером.

8. Властивості програмного забезпечення

Доступність - система доступна через браузер без встановлення додаткового ПЗ.

Продуктивність - інтерфейс повинен забезпечувати швидку реакцію на дії користувача.

Масштабованність - система повинна мати можливість функціонального розширення без зміни архітектури.

Безпека: передбачено базові механізми захисту даних та обмеження доступу.

Висновки до розділу 2

У другому розділі проведено аналіз сучасних підходів до розробки вебзастосунків для візуального моделювання та визначено основні

технологічні рішення, що використовуються у системах створення діаграм. На основі аналізу наукових джерел, технічної документації та існуючих аналогів встановлено, що сучасні вебзастосунки орієнтовані на SPA-архітектуру, клієнтську обробку даних та інтерактивну взаємодію користувача з графічними елементами у режимі реального часу. Також визначено, що для задач побудови та редагування діаграм найбільш доцільним є використання SVG-рендерингу, оскільки він забезпечує масштабованість графіки, точне позиціонування елементів та можливість окремої роботи з кожним об’єктом діаграми.

У процесі моделювання предметної області встановлено структуру домену вебзастосунку та визначено основні сутності системи: графічні елементи, зв’язки між елементами, полотно, діаграм та користувач. Для кожної сутності описано її ключові атрибути та роль у загальній структурі застосунку. Окрему увагу приділено взаємодії між елементами системи та механізмам оновлення стану під час виконання дій користувача. Визначено, що зміни координат, параметрів або структури елементів повинні автоматично відображатися у зв’язаних компонентах діаграми без порушення цілісності моделі.

Також у розділі сформовано основні функціональні вимоги до вебзастосунку. До них належать створення, редагування та видалення графічних елементів, побудова зв’язків між фігурами, зміна стилів і параметрів об’єктів, підтримка імпорту та експорту діаграм, а також збереження та відновлення структури проєкту. Додатково визначено функції роботи з галереєю діаграм, шаблонами та механізмами модерації публічного контенту. Описано основні сценарії взаємодії користувача із системою та уточнено логіку роботи з графічними об’єктами і зв’язками у межах редактора.

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ТА ОГЛЯД ТЕХНОЛОГІЙ

Проєктування ПЗ є одним із ключових етапів розробки вебзастосунку, оскільки саме на цьому етапі формується загальна структура системи, визначаються взаємозв’язки між її компонентами, моделюються сценарії взаємодії користувача та обираються технологічні рішення для реалізації функціоналу. Для вебзастосунків графічного редагування особливу роль відіграє правильна організація архітектури, адже система повинна забезпечувати одночасно швидку роботу інтерфейсу, збереження великої кількості об’єктів діаграми та підтримку багаторівневої взаємодії між клієнтською та серверною частинами.

3.1 Розробка UML-діаграм

Для вебзастосунку створення та редагування діаграм UML-діаграми використовуються як єдина візуальна модель, що описує структуру системи, ролі користувачів та взаємодію між основними компонентами. Вони дозволяють формально зафіксувати функціональні сценарії, архітектуру клієнт-серверної взаємодії, структуру бази даних та внутрішню організацію модулів редактора. Використання UML забезпечує послідовне проєктування системи – від загальних сценаріїв роботи користувача до фізичної структури збереження даних.

Таблиця 3.1 – Перелік UML-діаграм вебзастосунку

Тип діаграми	**Мета використання**	**Нотація / інструмент**	**Охоплення в системі**
Діаграма прецедентів (Use Case)	Опис ролей користувачів та сценаріїв взаємодії	UML Use Case / StarUML	Guest, User, Moderator, Administrator

Кінець таблиці 3.1

Діаграма компонентів	Відображення структури модулів системи	UML Component / StarUML	Frontend App, Backend API, Database
Діаграма класів	Опис доменних сутностей та зв’язків	UML Class / StarUML	User, Diagram, Template, DiagramElement
ER-діаграма	Модель таблиць бази даних	ER Diagram / pgAdmin	users, diagrams, templates, connections
Діаграма розгортання	Фізичне розміщення компонентів	UML Deployment / StarUML	Web Browser, Web Server, Database Server

Наведений набір діаграм охоплює основні рівні проєктування системи – від користувацьких сценаріїв до структури серверної частини та бази даних. Такий підхід дозволяє підтримувати узгодженість між функціональними вимогами, програмною логікою та моделлю даних у процесі подальшої реалізації застосунку.

3.1.1 Діаграма використання

Діаграма використання для вебзастосунку редагування діаграм відображає чотири основні типи користувачів: Guest, User, Moderator та Administrator. Кожен користувач має власний набір функцій, які визначають його можливості у системі.

Коли користувач входить у систему, він може зберігати власні проекти, використовувати різні шаблони та публікувати діаграми. Модератор контролює та затверджує матеріали, подані для публікації. Завданям адміністратора є керування користувачами, ролями та шаблонами в цій

системі. Взаємодія між різними ролями та їхніми відповідними робочими сценаріями, як саме вони можуть використовувати систему, зображена на діаграмі варіантів використання (рисунок 3.1).

Рисунок 3.1 – Діаграма використання вебзастосунку

Діаграма варіантів використання відображає ієрархію ролей за допомогою узагальнюючих зв'язків. Адміністратор успадковує можливості Модератора, як і Модератор Користувача, а Користувач надає розширені функціональні можливості, понад ті, що доступні Гостю. Для кожного з окремих сценаріїв діаграма варіантів використання використовує зв'язок «включити» або «розширити», який можна використовувати для відображення як обов'язкових, так і необов'язкових дій у межах заданого сценарію.

3.1.2 Діаграма взаємодії

Діаграми взаємодії використовуються для відображення послідовності обміну повідомленнями між компонентами системи під час виконання певного сценарію роботи. Вони дозволяють простежити порядок викликів методів, передачу даних між модулями та взаємодію користувацького інтерфейсу з внутрішньою логікою застосунку. Для вебзастосунку створення та редагування блок-схем такі діаграми використовуються для моделювання процесів додавання елементів, створення зв’язків між ними, автоматичного збереження та оновлення полотна редактора.

На діаграмі взаємодії відображено процес створення нового елемента блок-схеми та формування зв’язку між вузлами редактора (рисунок 3.2).

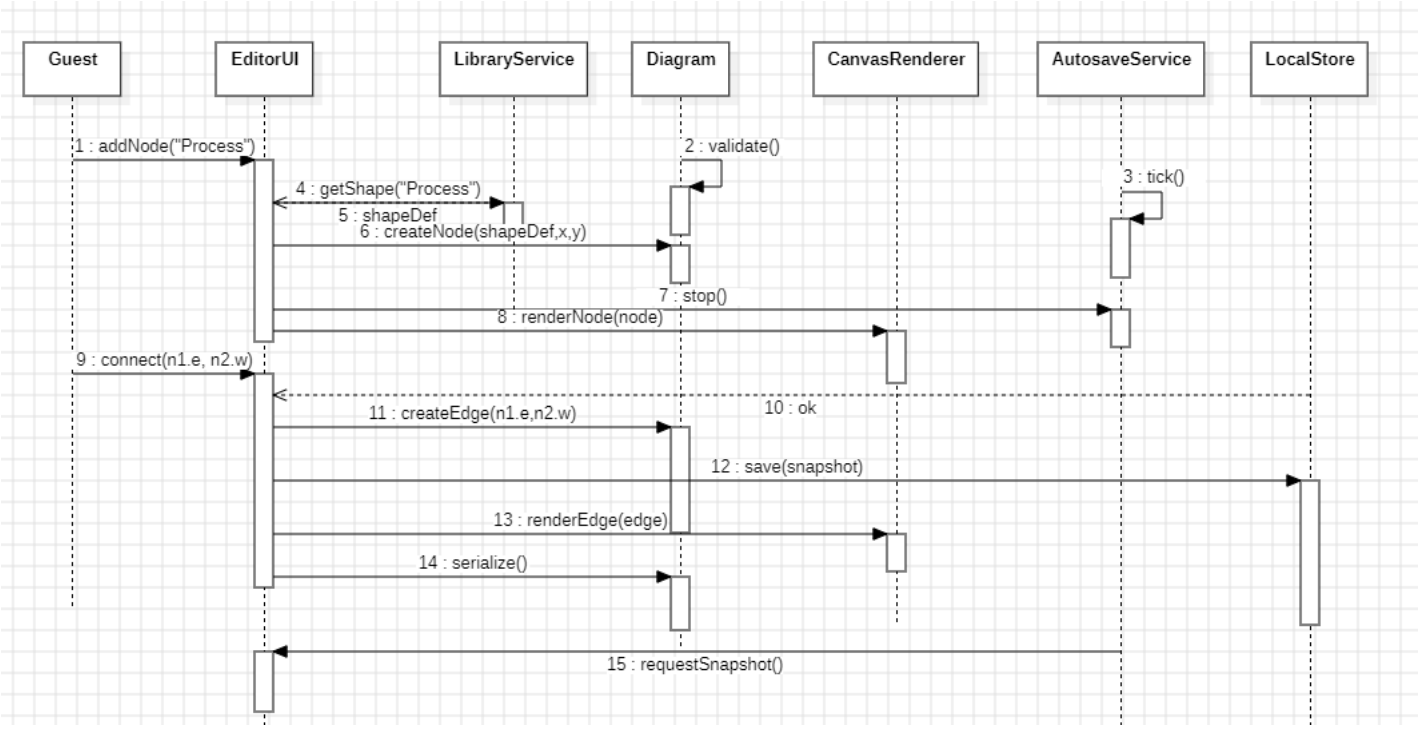

Рисунок 3.2 – Діаграма взаємодії компонентів редактора

Діаграма ілюструє взаємодію користувача з інтерфейсом редактора, сервісом бібліотеки елементів, модулем обробки діаграм, компонентом рендерингу та локальним сховищем. Весь процес починається, коли

користувач створює абсолютно новий елемент. Система отримує правильний шаблон фігури з групи шаблонів для створення нового вузла, який малюється на полотні редактора. Це перевіряється, а потім відображаються на CanvasRenderer. Після встановлення зв’язку між двома вузлами до графіка додається ребро. Відповідно система оновлює візуалізацію діаграми, щоб включити нове ребро. Оновлені дані серіалізуються для подальшого використання службою AutosaveService. Проєкти також можна зберігати за допомогою LocalStore, коли редактор відкрито. Нема необхідності зберігати свою роботу під час фактичної роботи над проєктом.

3.1.3 Діаграма класів

Основною сутністю системи є клас Diagram, який містить інформацію про діаграму, її назву, стиль та дату створення. Для збереження окремих графічних об'єктів використовується клас DiagramElement, а для опису зв'язків між ними – клас Connection. Користувачі системи представлені класом User, який містить облікові дані та роль доступу. Для підтримки шаблонів передбачено клас Template, а для реалізації процесу публікації та перевірки діаграм використовується клас ModerationRequest. Також у системі реалізовано механізм збереження версій діаграм за допомогою класу DiagramVersion. Діаграма класів (рисунок 3.3) зображує статичну структуру вебзастосунку разом з основними сутностями, їхніми атрибутами, функціями та елементами, що пов'язують різні компоненти системи. Є можливість формалізувати внутрішню структуру модулів, що складають застосунок, та описати, як об'єкти у вашому застосунку будуть взаємодіяти один з одним, перш ніж створювати базу даних та логіку на стороні сервера. У випадку вебзастосунку, який використовується для створення та редагування блок-схем, діаграма класів використовується для моделювання структури редактора, зберігання діаграм, шаблонів та процесів модерації.

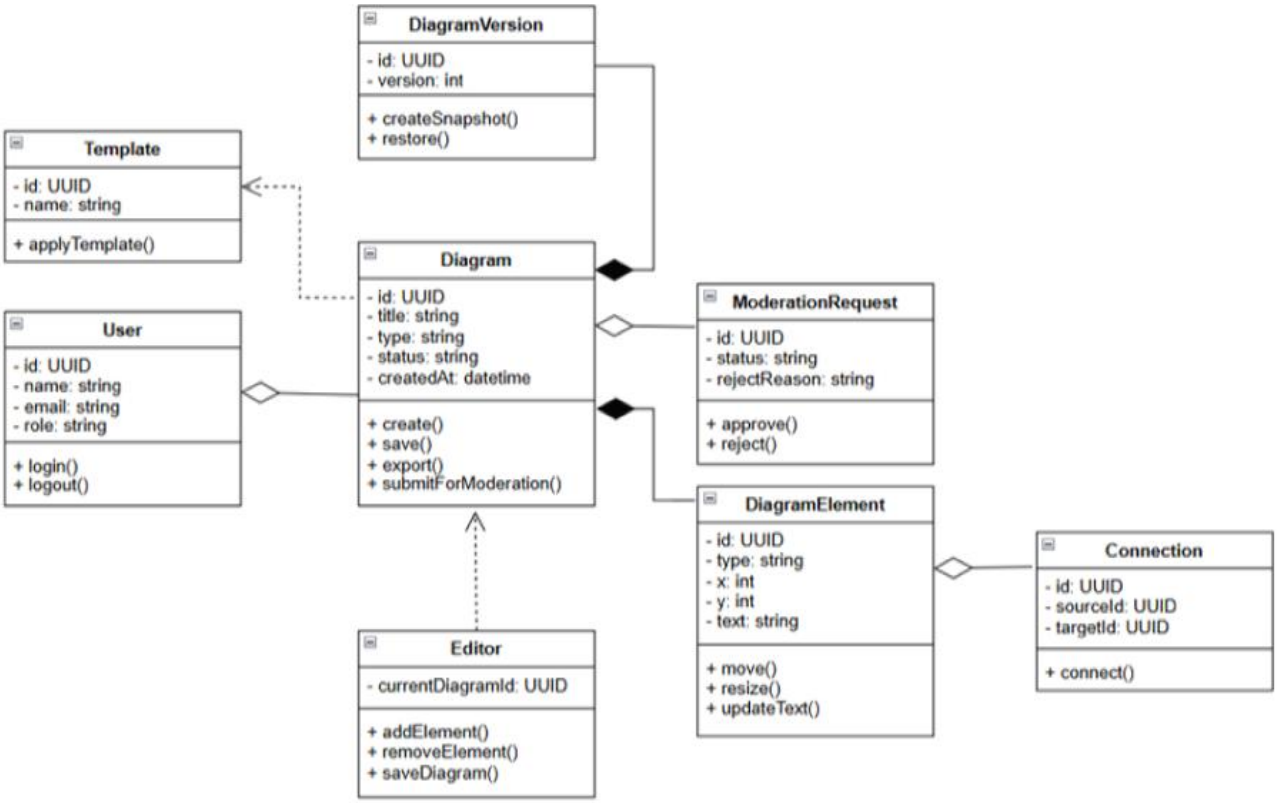

Рисунок 3.3 – Діаграма класів вебзастосунку

Діаграма відображає основні класи, що складають систему: User, Diagram, DiagramElement, Connection, Template, DiagramVersion та ModerationRequest. Основним класом у структурі є клас Diagram, який обробляє створення, збереження, експорт та публікацію діаграм. Клас DiagramElement представляє один елемент блок-схеми з його координатами, типом та текстовим вмістом. Connection визначає зв'язки між різними елементами на діаграмі.

Клас Template використовується для створення та підтримки шаблонів діаграм. Клас DiagramVersion використовується для створення знімка поточного стану діаграми та відновлення її до будь-якої попередньої версії. Клас ModerationRequest надає функціональність для перегляду опублікованого контенту, включаючи можливість схвалення або відхилення запиту на публікацію.

Існують різні типи зв'язків між класами: асоціація, агрегація та композиція. Клас diagram має композиційний зв'язок з DiagramElement та diagramVersion, оскільки ці два об'єкти можуть існувати лише в межах певної діаграми. Користувачі взаємодіють зі своїми проектами та шаблонами системи через асоціативні зв'язки. Використання цієї структури забезпечує модульність вебзастосунку, тому дозволяє просте подальше масштабування системи та включення нових функціональних компонентів.

3.1.4 Діаграма діяльності

Діаграма діяльності відображає алгоритмічний потік операцій під час експорту та публікації діаграми у вебзастосунку. Вона дозволяє формалізувати послідовність дій користувача і системи, а також описати можливі варіанти переходів між окремими етапами процесу. За допомогою activity diagram можна виявити точки перевірки даних, сценарії помилок та повторного виконання операцій ще на етапі проєктування.

У межах даного проєкту процес експорту має декілька можливих сценаріїв виконання залежно від вибраного формату та подальших дій користувача. Після формування файлу система перевіряє результат операції та надає можливість або зберегти діаграму локально, або перейти до процедури публікації. Якщо користувач обирає публікацію, додатково виконується введення метаданих діаграми та перевірка коректності заповнених полів перед передачею інформації на сервер. У вебзастосунку для створення та редагування блок-схем і діаграм діаграма діяльності використовується для моделювання процесу експорту готової діаграми та її подальшої публікації в системі (рисунок 3.4).

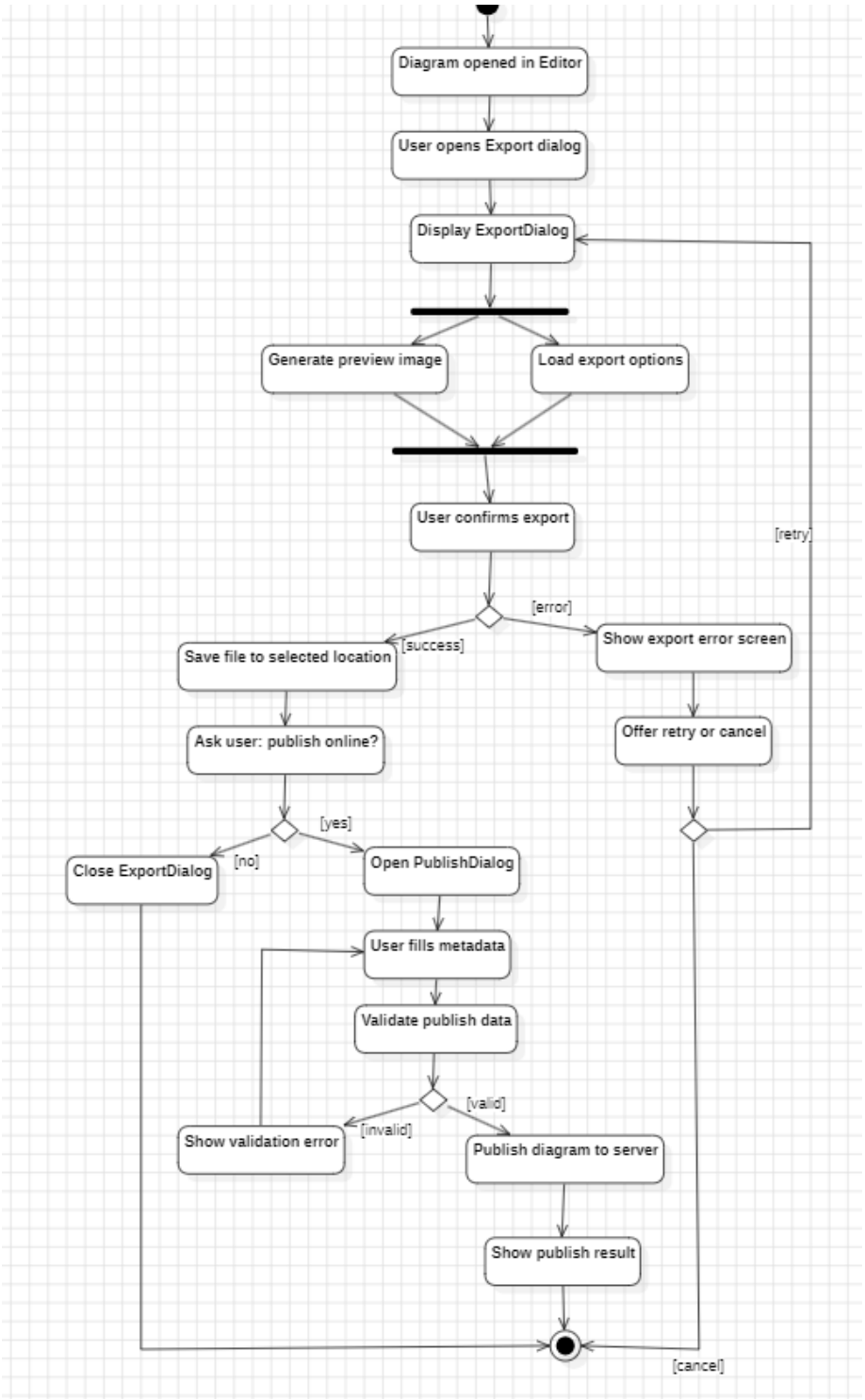

Рисунок 3.4 – Діаграма діяльності процесу експорту та публікації

Діаграма демонструє послідовність дій після відкриття користувачем вікна експорту. Система формує попередній перегляд зображення та одночасно завантажує параметри експорту. Після підтвердження експорту користувачем відбувається перевірка результату операції: у разі успішного

завершення файл зберігається у вибране місце, а при виникненні помилки система відображає екран помилки з можливістю повторної спроби.

Після успішного експорту користувачу пропонується опублікувати діаграму онлайн. Якщо користувач погоджується, відкривається форма публікації, де необхідно заповнити метадані діаграми. Перед надсиланням інформації виконується перевірка введених даних. У випадку некоректного введення система відображає повідомлення про помилку валідації, а після успішної перевірки діаграма публікується на сервері та користувачу відображається результат операції.

На діаграмі використано decision-вузли для перевірки результатів експорту та валідації даних, а також fork/join-конструкції для паралельного виконання окремих операцій. Такий підхід дозволяє більш детально відобразити поведінку системи та логіку взаємодії між користувачем і вебзастосунком.

3.1.5 Діаграма компонентів та розгортання

Діаграма компонентів описує модульну структуру вебзастосунку та показує, які частини системи відповідають за інтерфейс, логіку редактора, експорт, публікацію і локальне збереження даних. Для редактора блок-схем така модель важлива, оскільки основні дії користувача виконуються у клієнтській частині, а збереження та подальша робота з проєктами потребують окремого шару даних. У компонентній моделі вебзастосунок поділено на UIComponent, EditorUIComponent, EditorLogicComponent, ExportAndPublishComponent, LocalStorageComponent та FileStorageComponent (рисунок 3.5).

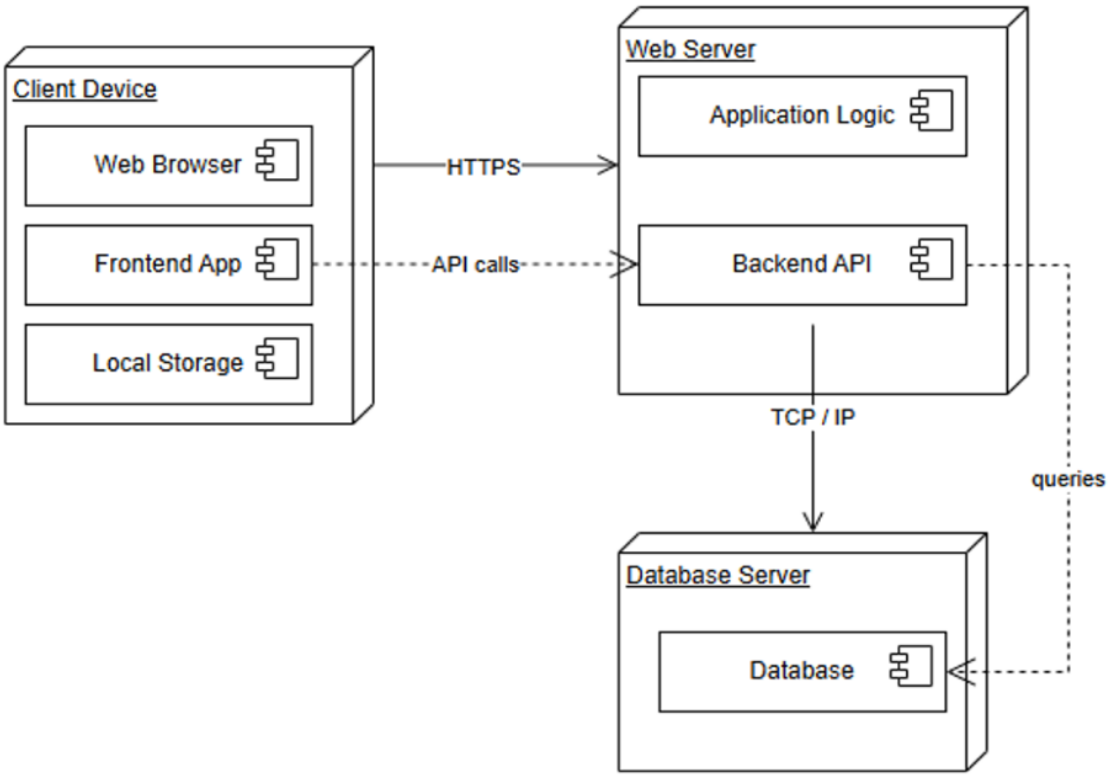

Рисунок 3.5 – Діаграма компонентів вебзастосунку

UIComponent відповідає за загальні сторінки застосунку: головну сторінку, галерею шаблонів, профіль користувача та екран помилки. EditorUIComponent містить частини інтерфейсу редактора: робочу область, панель інструментів і панель шарів. Основна логіка створення, редагування та перевірки діаграми зосереджена в EditorLogicComponent.

Компонент ExportAndPublishComponent використовується для експорту діаграми, відкриття вікна публікації та перетворення файлів у потрібний формат. LocalStorageComponent забезпечує роботу з локальним сховищем браузера, автозбереженням і тимчасовими даними сесії, що дозволяє відновити стан редактора після оновлення сторінки або випадкового закриття браузера. Компонент FileStorageComponent працює для збереження файлів, та для резервного копіювання діаграм. Такий поділ компонентів поділяє логіку на три різні області: редагування, збереження та експорт, що спрощує

підтримку коду та його розширення в майбутньому. Діаграми розгортання візуально відображають фізичне існування програми (рисунок 3.6).

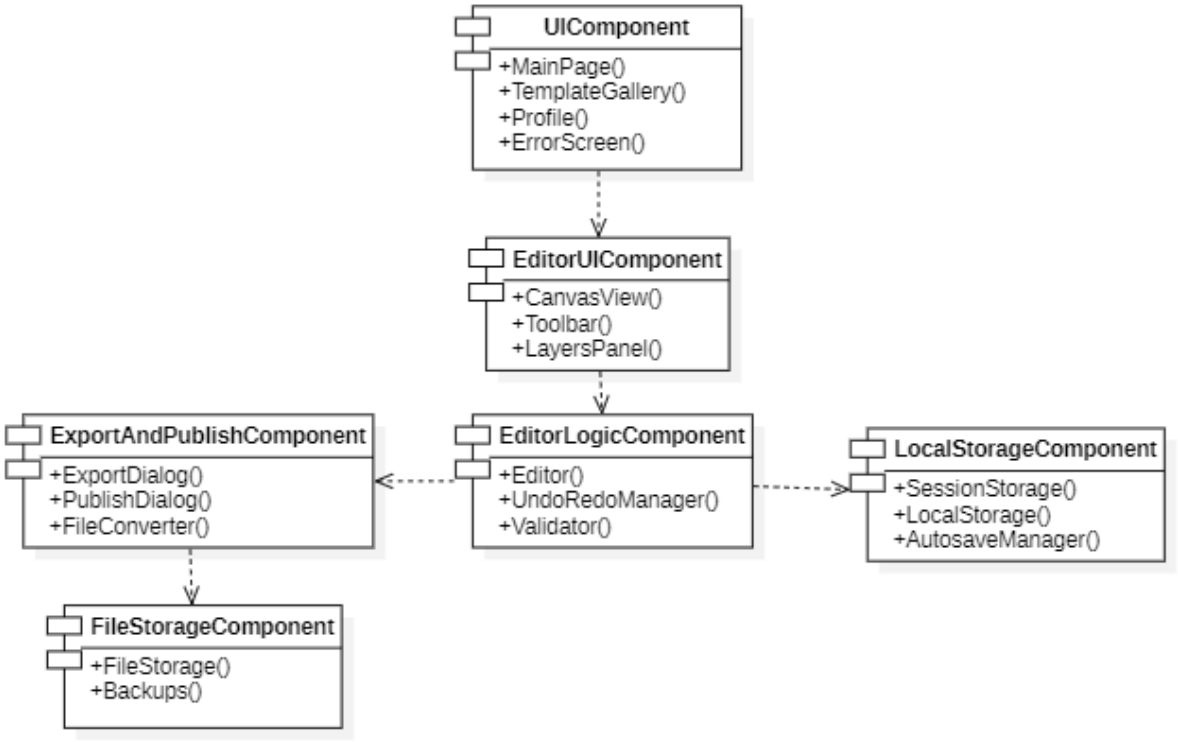

Рисунок 3.6 – Діаграма розгортання вебзастосунку

Клієнтська сторона взаємодіє із сервером за допомогою HTTPS/API. API серверної частини надсилає дані з фронтенду до БД через TCP/IP. Розділивши систему на ці секції (інтерфейс, сервер і дані), можна легко підтримувати та розширювати програму в майбутньому.

3.2 Варіанти використання системи

Розділ варіантів використання описує взаємодію зовнішніх акторів із вебзастосунком у вигляді окремих сценаріїв. Кожен сценарій визначає очікувану поведінку системи без деталізації внутрішньої реалізації коду. Для редактора діаграм основними акторами є гість, авторизований користувач, модератор та адміністратор, а ключові сценарії пов’язані зі створенням, редагуванням, збереженням, експортом і публікацією діаграм.

Таблиця 3.2 – Створення нової діаграми

Usecase section	**Comment**
Use Case Name	Створення нової діаграми
Scope	System
Level	User-goal
Primary Actor	Гість / користувач
Stakeholders and interests	Гість або користувач хоче створити нову діаграму для подальшого редагування.
Preconditions	Користувач відкрив вебзастосунок у браузері.
Success guarantee	Нова діаграма створена та відкрита в редакторі.
Main Success Scenario	1. Користувач переходить до створення нової діаграми. 2. Система відкриває галерею шаблонів або порожнє полотно. 3. Користувач вибирає шаблон або створює порожню діаграму. 4. Система ініціалізує робочу область редактора. 5. Користувач отримує доступ до панелі фігур і полотна.
Extensions	Якщо користувач закриває галерею шаблонів, система повертає його на попередній екран. Якщо шаблон не завантажився, система відкриває порожнє полотно.
Special Requirements	Робоча область має відкриватися без помітної затримки.
Technology and Data Variations List	Дані шаблону можуть завантажуватись із локального сховища або бази даних.
Frequency of Occurrence	30–40%
Miscellaneous	Сценарій є базовим для початку роботи з редактором.

Після створення нової діаграми користувач переходить до редактора, де може додавати графічні елементи, змінювати їх параметри та будувати зв’язки між ними. Якщо користувач не обирає шаблон, система створює порожній проєкт із базовими параметрами полотна.

Таблиця 3.3 – Редагування діаграми

Usecase section	**Comment**
Use Case Name	Редагування діаграми
Scope	System
Level	User-goal
Primary Actor	Гість / користувач
Stakeholders and interests	Користувач хоче змінити структуру діаграми, додати елементи, налаштувати їх вигляд і побудувати зв’язки.
Preconditions	Діаграма відкрита в редакторі.
Success guarantee	Зміни відображені на полотні та збережені у поточному стані діаграми.
Main Success Scenario	1. Користувач вибирає фігуру на панелі елементів. 2. Система додає елемент на полотно. 3. Користувач змінює текст, розмір або стиль елемента. 4. Користувач створює зв’язок між елементами. 5. Система оновлює відображення діаграми. 6. Поточний стан передається до механізму автозбереження.
Extensions	Якщо дані елемента некоректні, система не застосовує зміну та залишає попередній стан. Якщо користувач видаляє елемент, пов’язані з ним зв’язки також мають бути оновлені або видалені.
Special Requirements	Зміни на полотні повинні відображатися одразу після дії користувача.

Кінець таблиці 3.3

Technology and Data Variations List	Дані елементів зберігаються у JSON-структурі діаграми.
Frequency of Occurrence	60–80%
Miscellaneous	Сценарій є основним для роботи вебзастосунку.

Успішне редагування діаграми передбачає коректне оновлення не лише самого елемента, а й усіх пов’язаних із ним зв’язків. Це дозволяє уникнути ситуацій, коли структура діаграми візуально або логічно порушується після переміщення чи видалення об’єкта.

Таблиця 3.4 – Збереження діаграми

Usecase section	**Comment**
Use Case Name	Збереження діаграми
Scope	System
Level	User-goal
Primary Actor	Користувач
Stakeholders and interests	Користувач хоче зберегти створену діаграму та мати можливість відкрити її пізніше.
Preconditions	Користувач авторизований, діаграма відкрита в редакторі.
Success guarantee	Діаграма збережена у сховищі та прив’язана до профілю користувача.

Кінець таблиці 3.4

Main Success Scenario	1. Користувач натискає кнопку Save. 2. Система серіалізує поточний стан діаграми. 3. Дані перевіряються на коректність. 4. Система передає дані до сховища. 5. Користувач отримує повідомлення про успішне збереження.
Extensions	Якщо користувач не авторизований, система пропонує увійти до профілю. Якщо збереження неможливе, система залишає локальну копію діаграми.
Special Requirements	Не допускається втрата даних під час збереження.
Technology and Data Variations List	Основна структура діаграми зберігається у форматі JSON.
Frequency of Occurrence	40–60%
Miscellaneous	Локальне автозбереження використовується як додатковий механізм захисту від втрати змін.

Після збереження діаграма стає доступною у профілі користувача або бібліотеці проєктів. Якщо під час збереження виникає помилка, система повинна зберегти тимчасовий стан у локальному сховищі браузера.

Таблиця 3.5 – Експорт діаграми

Usecase section	**Comment**
Use Case Name	Експорт діаграми
Scope	System
Level	User-goal
Primary Actor	Гість / користувач

Кінець таблиці 3.5

Stakeholders and interests	Користувач хоче отримати діаграму у вигляді файлу для подальшого використання поза системою.
Preconditions	Діаграма відкрита в редакторі.
Success guarantee	Файл діаграми сформований і збережений на пристрої користувача.
Main Success Scenario	1. Користувач відкриває вікно експорту. 2. Система формує попередній перегляд діаграми. 3. Користувач вибирає формат експорту. 4. Система перетворює діаграму у вибраний формат. 5. Користувач завантажує файл.
Extensions	Якщо під час експорту виникає помилка, система показує повідомлення та пропонує повторити операцію.
Special Requirements	Експортований файл має відповідати поточному стану діаграми.
Technology and Data Variations List	Можливі формати експорту: SVG, PNG або JSON.
Frequency of Occurrence	20–30%
Miscellaneous	Експорт доступний як для гостя, так і для авторизованого користувача.

Сценарій експорту дозволяє використовувати створену діаграму поза межами вебзастосунку. Це важливо для навчальних, технічних і документаційних задач, де результат роботи потрібно вставити у звіт, презентацію або інший документ.

3.3 Проєктування архітектури та вибір технологічного стеку

Архітектура вебзастосунку для створення та редагування блок-схем і діаграм розроблена з поділом на клієнтську частину, серверний рівень та систему збереження даних. Це дозволяє ізолювати логіку інтерфейсу від backend-обробки та спрощує подальше розширення функціоналу системи. Клієнтська частина реалізована за допомогою React та TypeScript. React використовується для побудови інтерфейсу редактора, сторінок бібліотеки, авторизації, профілю користувача та адміністративної панелі. TypeScript забезпечує типізацію даних і дозволяє зменшити кількість помилок під час роботи зі станом редактора, шаблонами та елементами діаграм.

Для відображення та редагування діаграм використовується SVG. Такий підхід дає можливість масштабувати схеми без втрати якості, переміщувати елементи у реальному часі та працювати зі зв’язками між блоками. Частина логіки редактора реалізована у компонентах Canvas, Toolbar, ShapesPanel та PropertiesPanel, які відповідають за взаємодію користувача з робочою областю.

Серверна частина побудована на Node.js з використанням Express і TypeScript. Backend обробляє REST API-запити, виконує авторизацію користувачів, збереження діаграм, роботу з шаблонами та модерацію публікацій. Для взаємодії між frontend і backend використовується REST API, через який передаються дані про структуру діаграм, профілі користувачів і параметри шаблонів. Для збереження інформації використовується PostgreSQL. У базі даних зберігаються користувачі, діаграми, елементи схем, шаблони та зв’язки між блоками. Робота з базою даних виконується через ORM, що дозволяє працювати з даними через об’єктну модель без прямого написання SQL-запитів у більшості сценаріїв.

Таблиця 3.7 – Стек технологій

Категорія	**Обраний компонент**	**Роль у проєкті**
Frontend framework	React	Побудова інтерфейсу
Мова програмування	TypeScript	Типізація даних
Збірка frontend	Vite	Запуск і збирання клієнтської частини
Стан застосунку	Zustand	Збереження стану редактора
UI-бібліотека	Bootstrap / React-Bootstrap	Базові елементи інтерфейсу
Графічний рендеринг	SVG	Відображення та масштабування діаграм
Backend runtime	Node.js	Виконання серверної частини
Backend framework	Express	REST API та серверна логіка
API	REST API	Обмін даними між frontend і backend
БД	PostgreSQL	Збереження користувачів, діаграм і шаблонів
Робота з БД	pg	Виконання SQL-запитів до PostgreSQL
Локальне сховище	LocalStorage	Автозбереження тимчасових даних

Взаємодія основних компонентів системи та технологій, які використовуються у вебзастосунку, наведена у схемі архітектури (рисунок 3.7).

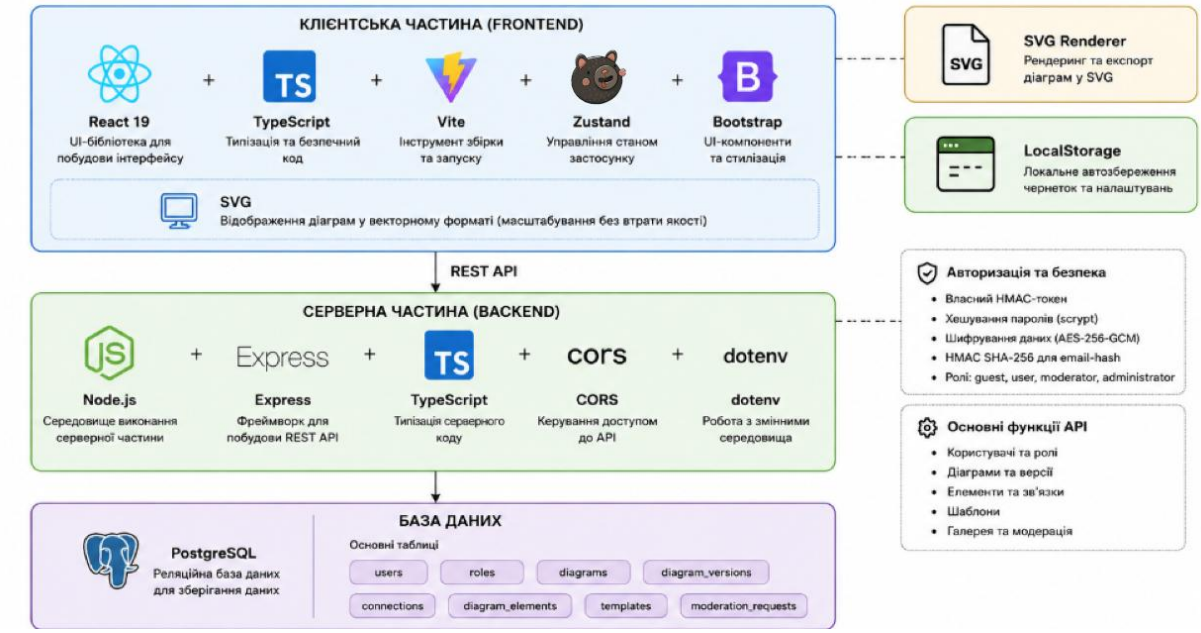

Рисунок 3.7 – Обрані технології вебзастосунку

Користувач працює із редактором через браузер, у якому запускається React-застосунок. Після виконання дій frontend надсилає REST API-запити до Node.js / Express backend. Сервер обробляє отримані дані та взаємодіє з PostgreSQL для збереження діаграм, шаблонів і користувацьких даних. Частина інформації додатково кешується у LocalStorage для реалізації автозбереження та відновлення змін після перезавантаження сторінки.

3.4 Мокапи інтерфейсу

Було розроблено серію макетів інтерфейсу користувача вебзастосунку для ранньої візуалізації основних сторінок. За допомогою макетів можна визначити розмітку та структуру сторінок та функціональні елементи перед повним впровадженням. Крім того, макети служать засобом забезпечення візуальної узгодженості в усій системі, а також оцінки та затвердження взаємодії користувача з редактором діаграм. Під час розробки макетів основну увагу було зосереджено на першочерговому створенні простого у використанні інтерфейсу зі швидким доступом до часто використовуваних функцій та логічно згрупованих елементів. Усі макети інтерфейсів були

розроблені на основі майбутнього фреймворку React, який сприятиме розробці компонентів застосунку та налаштуванню кінцевого дизайну застосунку для всіх розмірів екрану.

Мокап головної сторінки містить кнопку створення нової діаграми, список останніх проєктів та набір шаблонів для швидкого старту роботи (рисунок 3.8). Користувач може швидко перейти до створення порожньої діаграми або скористатися готовим шаблоном.

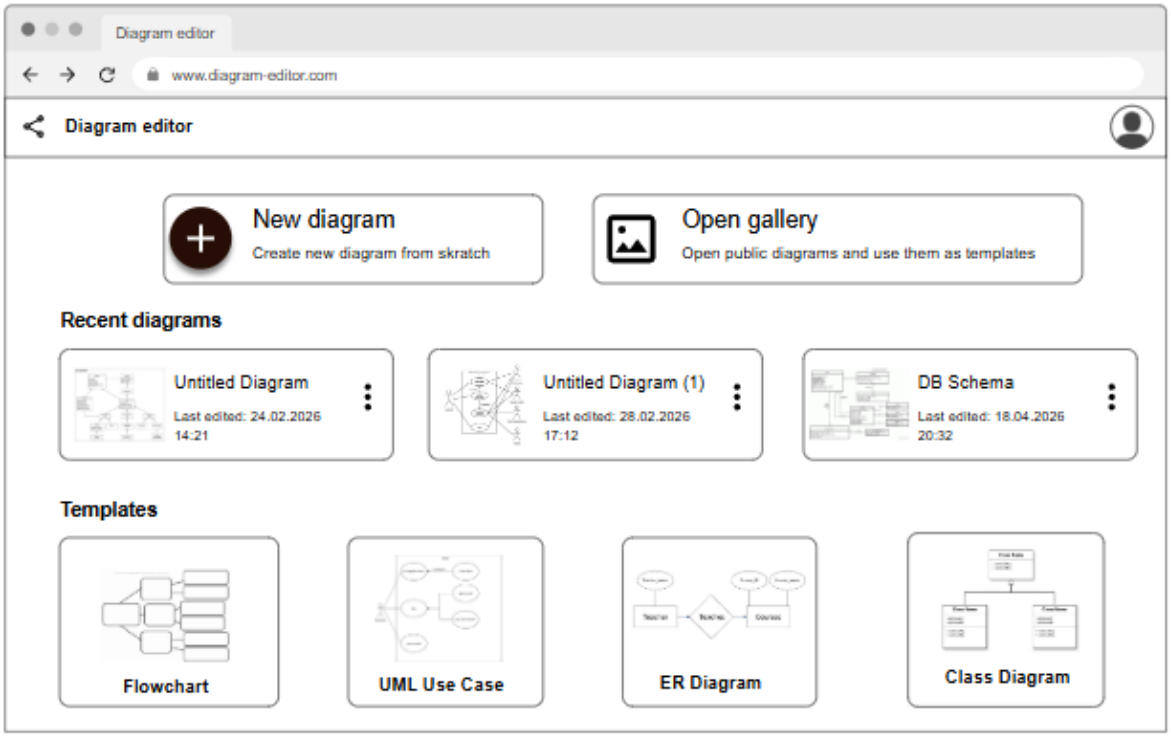

Рисунок 3.8 – Мокап головної сторінки редактора діаграм

Функціональні елементи данрї сторінки - це кнопка “New diagram”, блок “Recent diagrams”, секція шаблонів, картки останніх діаграм та кнопка відкриття галереї шаблонів. Сторінка галереї шаблонів призначена для перегляду публічних діаграм та їх використання як основи для нових проєктів (рисунок 3.9). Інтерфейс містить поле пошуку, фільтрацію за типами діаграм та картки шаблонів із короткою інформацією.

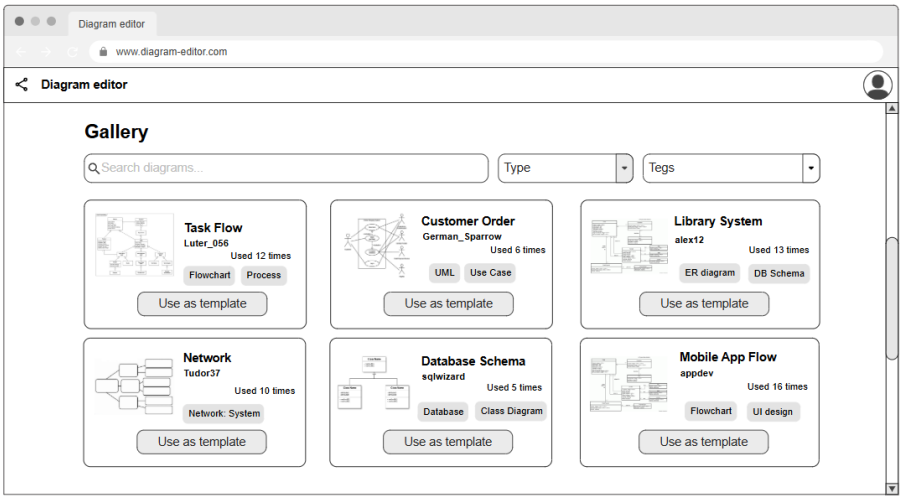

Рисунок 3.9 – Мокап сторінки галереї шаблонів

Функціональні елементи сторінки галереї включають поле пошуку шаблонів, фільтрацію за типами діаграм, список публічних шаблонів, кнопку “Use as template”, а також теги категорій діаграм. Основним елементом системи є сторінка редактора діаграм, на якій користувач створює та редагує схеми (рисунок 3.10). Саме редактор реалізує ключовий функціонал вебзастосунку та забезпечує взаємодію користувача з графічними елементами у реальному часі. Інтерфейс поділено на декілька функціональних областей: панель категорій, бібліотеку фігур, робочу область та панель властивостей обраного елемента.

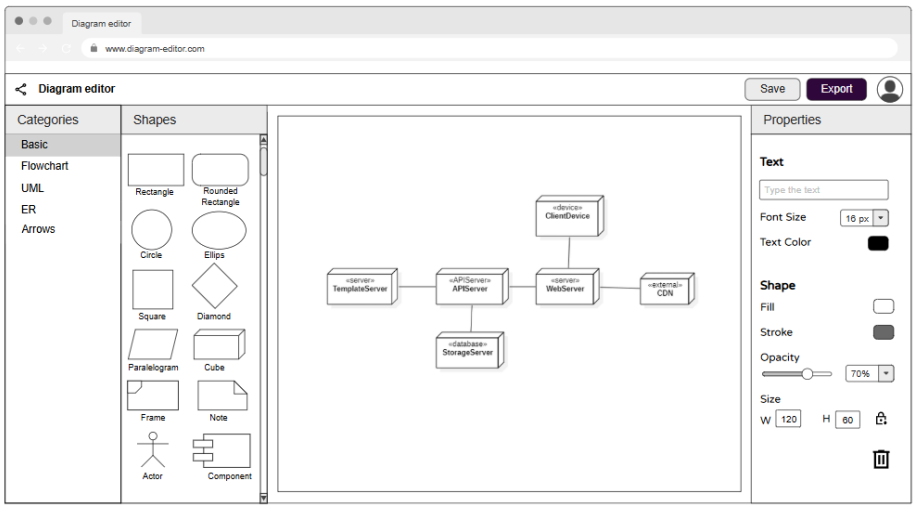

Рисунок 3.10 – Мокап сторінки редактора діаграм

Функціональні елементи: панель категорій елементів, бібліотека фігур, робоче полотно редактора, панель властивостей, кнопки “Save” та “Export”.

Сторінка “My diagrams” використовується для перегляду збережених користувацьких проєктів (рисунок 3.11). Вона дозволяє швидко знаходити потрібні діаграми, сортувати їх та виконувати базові дії без відкриття редактора.

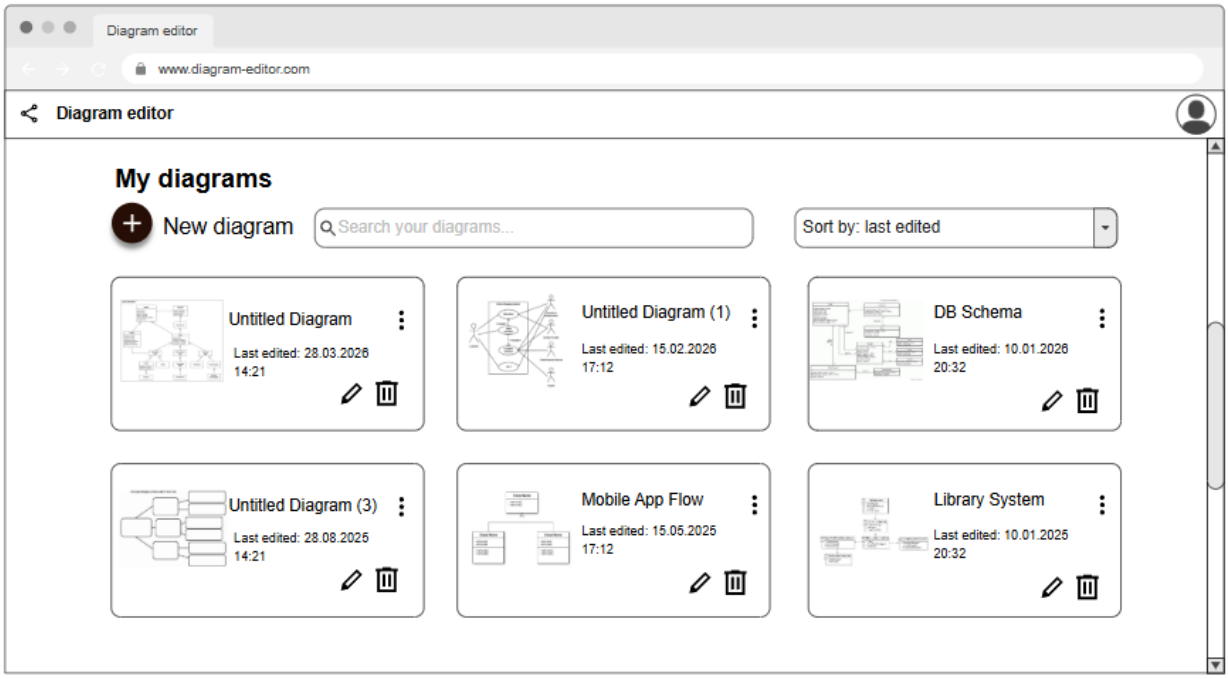

Рисунок 3.11 – Мокап сторінки “My diagrams”

Функціональні елементи: список збережених діаграм, поле пошуку, сортування проєктів, кнопка створення нової діаграми, кнопки редагування та видалення. Для модерації опублікованих матеріалів передбачено окремий інтерфейс перевірки діаграм (рисунок 3.12). Модератор може переглядати схему, підтверджувати її публікацію або відхиляти із зазначенням причини.

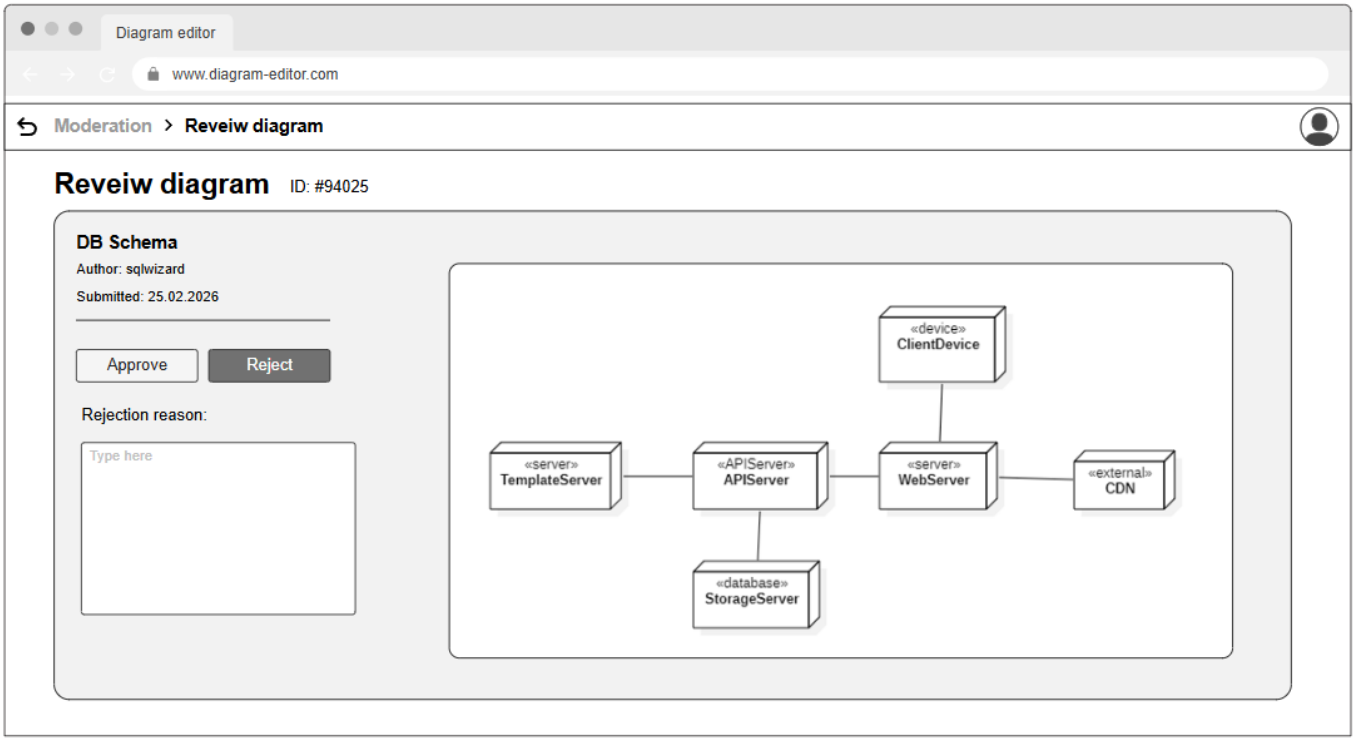

Рисунок 3.12 – Мокап сторінки модерації діаграм

Функціональні елементи: область перегляду діаграми, кнопка “Approve”, кнопка “Reject”, поле введення причини відхилення, інформація про автора та дату публікації.

Висновки до розділу 3

У третьому розділі сформовано загальне уявлення про структуру вебзастосунку для створення та редагування блок-схем і діаграм, а також визначено основні підходи до його проєктування. У процесі аналізу побудовані UML-діаграми, що описують ролі користувачів, взаємодію компонентів системи, структуру даних та організацію серверної і клієнтської частин застосунку.

Для моделювання різних функцій системи використовувалися діаграми варіантів використання, взаємодії, класів, дій, компонентів та розгортання. Вони були застосовані для формалізації різних основних сценаріїв

користувача, структури редактора діаграм, а також механізмів збереження та обміну даними між двома частинами системи (інтерфейсом та серверною частиною), включаючи створення діаграм, їх редагування, а потім збереження або експорт із системи.

У цьому розділі описано технологічний стек для вебдодатку. Користувацький інтерфейс розроблено за допомогою React та TypeScript. Тим часом серверна частина була розроблена на Node.js разом з Express. Дані додатку зберігаються в реляційній базі даних – PostgreSQL. Використано технологію SVG для створення графічних компонентівю, отже легко масштабувати та змінювати графіку в браузері. REST API забезпечують зв’язок між клієнтом та сервером системи.

Особливу увагу зосереджено для створення мокапів інтерфейсу користувача, які забезпечили основу для розробки головних сторінок вебзастосунку перед початком фактичної фази розробки. Мокапи передбачають кілька важливих сторінок, включаючи головну сторінку редактора, галерею шаблонів, керування діаграмами, інтерфейс редактора та сторінку модерації для публікації. Результат, отриманий в результаті роботи над приблизним дизайном, був використаний як орієнтир під час створення застосунку для забезпечення єдності дизайну всієї системи, БД та необхідної функціональності.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

У четвертому розділі обговорюється фактичне виконання застосунку, призначеного для створення та зміни блок-схем та діаграм в Інтернеті. Найголовнішу увагу зосереджено на організації фронтенд та бекенд розділів, організації коду, взаємодії компонентів редактора та методах, обраних для збереження створених даних. Крім того, представлено як створено способи входу в систему, призначено спосіб автоматичного збереження змін і як оброблюються графічні компоненти. Клієнтська частина застосунку реалізована за допомогою TypeScript та React. Серверна частина побудована на Node.js з використанням Express і REST API для обміну даними між frontend та backend. Для відображення діаграм використовується SVG, тому кожен елемент на діаграмі може оброблятися в режимі реального часу та змінювати його розмір до будь-якого, без втрати якості.

4.1 Структура коду та специфікації

Вихідні файли застосунку використовуються для створення вебзастосунку. Вихідний код класифікується наступним чином: frontend, backend, компоненти, сховище, типи та дані. Розділення логіки (інтерфейс користувача) та логіки на стороні сервера спрощує підтримку або розширення майбутніх функціональних можливостей системи.

Таблиця 4.1 – Основні сутності та структури даних

Сутність	**Атрибути - тип**	**Призначення**
Shape	id - string, text – string, type - ShapeType, x - number, y - number, width -number, height - number	елемент діаграми

Кінець таблиці 4.1

Edge	id - string, from - EdgeEndpoint, to – EdgeEndpoint, type - ConnectorType	зв’язок між елементами
ShapeStyle	stroke - string, fontSize – number, fill - string	стилізація фігур
Template	title - string, category - string, snapshot - JSON	шаблон діаграми
User	email - string, role - string, passwordHash - string	користувач системи

Файл diagram.ts містить основні типи даних, які можна використовувати в цьому редакторі (рисунок 4.1). У файлі diagram.ts описано типи фігур, які можна використовувати в діаграмі, доступні типи зв'язків між фігурами та структуру об'єктів, які будуть використовуватися для побудови діаграми. На етапі розробки система контролює тип даних за допомогою TypeScript, тим самим зменшуючи кількість помилок, які виникатимуть під час роботи редактора.

За допомогою TypeScript можна керувати всією структурою даних у редакторі як на фронтенді, так і на серверній частині. Тому дуже легко серіалізувати діаграми, зберігати та завантажувати їх у форматі JSON. Крім того, явне введення в TypeScript запобігає помилкам під час передачі даних між компонентами програми та спрощує підтримку коду під час додавання нових функцій до редактора. Наявність структурованої та розділеної моделі даних полегшує ізоляцію різних типів логіки в системі, наприклад, відокремлення зміни стилю від логіки, що використовується для з'єднання об'єктів, або створення структури шаблону без необхідності змінювати процес авторизації. В результаті, програму можна легше масштабувати, а нові функції можна легше впроваджувати.

```
export interface Shape {
  id: string
  type: ShapeType
  x: number
  y: number
  width: number
  height: number
  text: string
  style: ShapeStyle
  layout?: {
    classHeaderHeight?: number
    classAttributesHeight?: number
    tableHeaderHeight?: number
    laneHeaderWidth?: number
    componentTabWidth?: number
    packageTabWidth?: number
    serverSectionHeight?: number
  }
}

export interface EdgeStyle {
  stroke: string
  strokeWidth: number
}

export type EdgeEndpoint =
  | { kind: 'attached'; shapeId: string }
  | { kind: 'free'; x: number; y: number }

export interface Edge {
  id: string
  type: ConnectorType
  from: EdgeEndpoint
  to: EdgeEndpoint
  style?: EdgeStyle
  bendPoints?: { x: number; y: number }[]
}
```

Рисунок 4.1 – Типи Shape та Edge

Інтерфейс Shape в основному використовується для створення взаємозв’язків/відношень між графічними елементами, визначеними інтерфейсом форми в діаграмі. Це допомагає забезпечити єдиний формат даних як для фронтенду, так і для серверної частини системи. Стан редактора також керується з централізованого місця. Структура фігури містить параметри координат і розміру (ширина/висота), інформацію про стиль, текст та інші характеристики для складних об'єктів UML. Ця функція дозволяє створювати різні типи діаграм без потреби в нових структурах даних для кожного графічного елемента, визначеного в діаграмі. Оскільки в розробці

використовується типізація TypeScript, стає легше виконувати перевірку даних, а також легшн запобігти багатьом помилкам даних під час передачі стану редактора до різних частин програми.

Серверна частина застосунку побудована на Node.js та Express. Основні API-маршрути описані у файлі index.ts, де виконується реєстрація endpoint-ів авторизації, роботи з діаграмами та адміністративною панеллю (рисунок 4.2).

```
50  app.post('/api/auth/register', registerUser);
51  app.post('/api/auth/login', loginUser);
52  app.post('/api/auth/verify-admin-code', verifyAdminLoginCode);
53  app.get('/api/auth/me', requireAuth, getCurrentUser);
54  app.patch('/api/auth/me', requireAuth, updateCurrentUser);
55  app.patch('/api/auth/me/password', requireAuth, updateCurrentUserPassword);
56  app.patch('/api/admin/users/role', requireAuth, requireRole(['administrator', 'admin']), assignUserRole);
57  app.get('/api/admin/users', requireAuth, requireRole(['administrator', 'admin']), listUsers);
58  app.patch('/api/admin/users/:id/block', requireAuth, requireRole(['administrator', 'admin']), updateUserBlock);
59  app.delete('/api/admin/users/:id', requireAuth, requireRole(['administrator', 'admin']), deleteUser);
60
61  app.get('/api/diagrams', requireAuth, listDiagrams);
62  app.post('/api/diagrams', requireAuth, createDiagram);
63  app.get('/api/diagrams/:id', requireAuth, getDiagram);
64  app.put('/api/diagrams/:id', requireAuth, updateDiagram);
65  app.post('/api/diagrams/:id/submit', requireAuth, submitDiagram);
66  app.get('/api/gallery', listGallery);
67  app.get('/api/admin/diagrams', requireAuth, requireRole(['administrator', 'admin']), listAllDiagrams);
68  app.delete('/api/admin/diagrams/:id', requireAuth, requireRole(['administrator', 'admin']), deleteDiagram);
69  app.get('/api/admin/stats', requireAuth, requireRole(['administrator', 'admin']), systemStats);
70  app.get('/api/moderation/diagrams', requireAuth, requireRole(['moderator', 'administrator', 'admin']), listModerationQueue);
71  app.patch('/api/moderation/diagrams/:id', requireAuth, requireRole(['moderator', 'administrator', 'admin']), moderateDiagram);
72
73  app.get('/api/templates', listTemplates);
74  app.post('/api/admin/templates', requireAuth, requireRole(['administrator', 'admin']), createTemplate);
75  app.delete('/api/admin/templates/:id', requireAuth, requireRole(['administrator', 'admin']), deleteTemplate);
76
```

Рисунок 4.2 – Реєстрація REST API маршрутів

Backend використовує REST API для обміну даними між клієнтською та серверною частинами. Для захисту маршрутів застосовуються middleware requireAuth та requireRole, які перевіряють авторизацію користувача та його роль у системі.

Для захисту даних користувачів у системі реалізовано декілька рівнів безпеки. Паролі не зберігаються у відкритому вигляді та обробляються за допомогою алгоритму crypto.scrypt, після чого до бази даних записується лише хеш пароля (рисунок 4.4). Персональні дані користувача, зокрема ім’я та email, додатково шифруються алгоритмом AES-256-GCM перед збереженням у PostgreSQL. Для пошуку користувачів за email використовується окремий

HMAC SHA-256 hash, що дозволяє виконувати перевірку унікалності без зберігання email у відкритому вигляді. Авторизація користувача реалізована на основі власного HMAC-токена, який перевіряється middleware-компонентами backend частини застосунку.

```
function encryptText(value: string) {
  const iv = crypto.randomBytes(12);
  const cipher = crypto.createCipheriv('aes-256-gcm', encryptionKey, iv);
  const encrypted = Buffer.concat([cipher.update(value, 'utf8'), cipher.final()]);
  const tag = cipher.getAuthTag();

  return `${base64Url(iv)}.${base64Url(tag)}.${base64Url(encrypted)}`;
}

function hashEmail(email: string) {
  return crypto
    .createHmac('sha256', emailHashSecret)
    .update(email.trim().toLowerCase())
    .digest('hex');
}

async function hashPassword(password: string) {
  const salt = crypto.randomBytes(16);
  const derived = (await scryptAsync(password, salt, 64)) as Buffer;

  return `${base64Url(salt)}.${base64Url(derived)}`;
}
```

Рисунок 4.4 – Фрагмент реалізації шифрування персональних даних і хешування пароля

Для роботи з інтерфейсом редактора використовуються React-компоненти. Панель інструментів реалізована у компоненті Toolbar.tsx (рисунок 4.3).

```
<ButtonGroup>
  <Button
    variant={tool === 'select' ? 'dark' : 'outline-dark'}
    onClick={() => setTool('select')}
  >
    Select
  </Button>

  <Button
    variant={tool === 'connect' ? 'success' : 'outline-success'}
    onClick={() => setTool(tool === 'connect' ? 'select' : 'connect')}
  >
    Connect: {tool === 'connect' ? 'ON' : 'OFF'}
  </Button>
</ButtonGroup>
```

Рисунок 4.3 – Компонент Toolbar

Компонент Toolbar забезпечує взаємодію користувача з редактором. Через store викликаються функції додавання фігур та перемикання режимів роботи редактора.

4.2 Тестування програмного забезпечення

Мета тестування Diagram Editor – переконатися, що основні функції редактора працюють коректно, механізми імпорту та експорту не призводять до втрати даних, а серверна частина стабільно обробляє API-запити та витримує базове навантаження. Система перевірялася за кількома рівнями тестування: модульним, інтеграційним та навантажувальним.

Таблиця 4.2 – Класифікація тестів

Рівень	**Інструмент**	**Об'єкт перевірки**	**Цільове покриття**
Unit	Node.js custom runner	Імпорт/експорт діаграм, CSV, Mermaid, SVG, JSON	Перевірка критичних сценаріїв перетворення даних
Integration	Fetch API smoke tests	REST API: health, templates, gallery, auth guard	Перевірка основних серверних маршрутів
Load	Node.js load script	REST API при 100-1000 RPS	Перевірка стабільності відповіді сервера

Модульні тести перевіряють правильність роботи механізмів імпорту та експорту діаграм. Зокрема, тестуються збереження структури елементів після CSV-експорту (рисунок 4.5), підтримка багаторядкового тексту, коректна обробка лапок і ком у CSV, створення вузлів і зв’язків із Mermaid, а також генерація SVG-представлення діаграми. Окремо перевіряється коректність

серіалізації JSON-структури діаграми та відновлення даних після повторного відкриття проєкту. Такий підхід дозволяє виявляти помилки перетворення даних ще до інтеграції з інтерфейсом користувача та зменшує ризик втрати структури diagrams, shapes і edges під час збереження або експорту.

```
test('exports and imports CSV while preserving shape data', () => {
    const snapshot = {
      shapes: [
        {
          id: 'shape-1',
          type: 'rect',
          x: 100,
          y: 120,
          width: 160,
          height: 80,
          text: 'User\nProfile',
          style: baseShapeStyle,
        },
      ],
      edges: [],
    }

    const imported = csvToSnapshot(snapshotToCsv(snapshot))

    assert.equal(imported.shapes.length, 1)
    assert.equal(imported.shapes[0].type, 'rect')
    assert.equal(imported.shapes[0].text, 'User\nProfile')
    assert.equal(imported.shapes[0].x, 100)
    assert.equal(imported.shapes[0].width, 160)
})

test('imports Mermaid nodes and relations as shapes and edges', () => {
    const imported = mermaidToSnapshot(`
flowchart TD
  A["Login"]
  B["Editor"]
  A --> B
`)

    assert.equal(imported.shapes.length, 2)
    assert.equal(imported.edges.length, 1)
    assert.equal(imported.shapes[0].text, 'Login')
```

Рисунок 4.5 – Unit-тести імпорту та експорту діаграм

Наведені тести демонструють перевірку ключових сценаріїв перетворення даних редактора. Такий підхід дозволяє виявляти помилки імпорту й експорту ще до інтеграції з інтерфейсом користувача.

Таблиця 4.3 - Результати автоматизованого тестування

Група тестів	**Кількість перевірок**	**Успішно виконано**	**Результат**
Unit	12	12	100 %
Integration	5	5	100 %
Load smoke	100 запитів	100	0 % помилок

Інтеграційне тестування перевіряло роботу REST API. Підтверджено, що endpoint /api/health повертає статус працездатності сервісу, /api/templates – список шаблонів, /api/gallery – список публічних діаграм, приватний маршрут /api/diagrams без авторизації повертає помилку доступу зі статусом 401. Також перевірено відхилення входу з неправильними обліковими даними.

Таблиця 4.4 – Продуктивність API залежно від навантаження

Навантаження	**p50 latency, мс**	**p95 latency, мс**	**Error rate, %**	**Фактичний RPS**
100 RPS	3	5	0,00	100,0
300 RPS	2	5	0,00	299,6
500 RPS	3	9	0,00	499,1
1000 RPS	3	10	0,00	999,5

Аналіз результатів:

– функціональна коректність - усі модульні тести імпорту та експорту діаграм завершилися успішно, структура shapes і edges після перетворення даних зберігається коректно;

– інтеграційна перевірка - основні REST API-маршрути /api/health, /api/templates, /api/gallery працюють стабільно, приватний маршрут /api/diagrams без авторизації правильно блокується зі статусом 401;

– продуктивність - при збільшенні навантаження до 1000 RPS сервер залишився в межах робочих показників, p95 latency не перевищила 10 мс, а частка помилок склала 0 %;

– надійність - під час тестування не зафіксовано втрати даних при імпорті, експорті та серіалізації діаграм.

Результати свідчать, що серверна частина зберігає стабільний час відповіді навіть при збільшенні навантаження до 1000 RPS. Під час тестування також не зафіксовано втрати даних при імпорті, експорті та повторному відкритті діаграм, що підтверджує надійність механізмів збереження та відновлення стану редактора.

4.3 Результати рішення

Для оцінки готовності Diagram Editor до дослідної експлуатації проведено серію контрольних запусків із різними вхідними даними та рівнями навантаження. Нижче подано підсумки вимірювань продуктивності, коректності роботи API та стабільності виконання основних операцій системи.

Таблиця 4.5 – Продуктивність API-шлюзу залежно від навантаження

Конкурентних запитів	**p50 latency, мс**	**p95 latency, мс**	**Error rate, %**	**Факт. RPS**
100 RPS	3	5	0,00	100,0
300 RPS	2	5	0,00	299,6
500 RPS	3	9	0,00	499,1
1 000 RPS	3	10	0,00	999,5

Показник p95 залишається нижче за 10 мс навіть при навантаженні 1 000 RPS. Частка помилок у всіх сцнаріях становить 0 %, що свідчить про стабільну роботу API в локальному тестовому середовищі. Фактичний RPS майже

відповідає заданому навантаженню, значить сервер встигає успішно обробляти запити без помітного накопичення черги.

Таблиця 4.6 – Час виконання основних операцій при 500 RPS

Операція	**Середнє, мс**	**Діапазон min-max, мс**
Перевірка доступності API	3	0-12
Завантаження шаблонів	4	0-20
Завантаження галереї	2	1-12
Перевірка помилкового входу	3	0-14

Операції отримання шаблонів, галереї та перевірки стану API виконуються менш ніж за 5 мс у середньому та не потребують додаткового кешування. Навіть при навантаженні 500 RPS сервер зберігає стабільний час відповіді без накопичення черги запитів.

Оцінка якості та коректності:

1) Функціональна повнота. Реалізовано основні сценарії MVP: створення та редагування діаграм, імпорт і експорт CSV/Mermaid/SVG, роботу з шаблонами, авторизацію користувачів і систему модерації.

2) Показники доступності. Під час контрольних запусків endpoint /api/health стабільно повертав статус працездатності сервісу, а API залишався доступним у всіх сценаріях навантаження.

3) Надійність даних. Модульні тести підтвердили коректність серіалізації та відновлення структури diagrams, shapes і edges без втрати інформації.

4) Масштабованість. REST API побудовано окремо від frontend-частини, що дозволяє незалежно масштабувати серверну логіку та клієнтський застосунок.

Результати тестування:

– результати навантажувальних запусків демонструють стабільне зростання фактичного RPS без різкого збільшення latency до рівня 1 000 запитів за секунду;

– розподіл часу відповіді показує, що більшість API-запитів завершується менш ніж за 10 мс, а відсутність error rate свідчить про стабільність серверної частини.

Тестові вимірювання доводять, що розроблений вебзастосунок відповідає основним нефункціональним вимогам. API стабільно обробляє навантаження, механізми імпорту та експорту працюють коректно, а архітектура системи забезпечує можливість подальшого розширення функціоналу редактора діаграм.

4.4 Керівництво користувача

Даний підрозділ містить послідовну інструкцію роботи із застосунком Diagram editor для основних ролей системи: гостя, зареєстрованого користувача, модератора та адміністратора. Опис подано у формі практичних сценаріїв роботи із системою та доповнено узагальнювальними таблицями елементів інтерфейсу.

Запуск застосунку

Вебзастосунок Diagram editor відкривається у браузері та не потребує встановлення додаткового ПЗ на пристрій користувача. Після переходу на адресу застосунку користувач потрапляє на головну сторінку, де розміщено основні дії: створення нової діаграми, перехід до редактора, перегляд шаблонів, відкриття галереї та вхід до профілю.

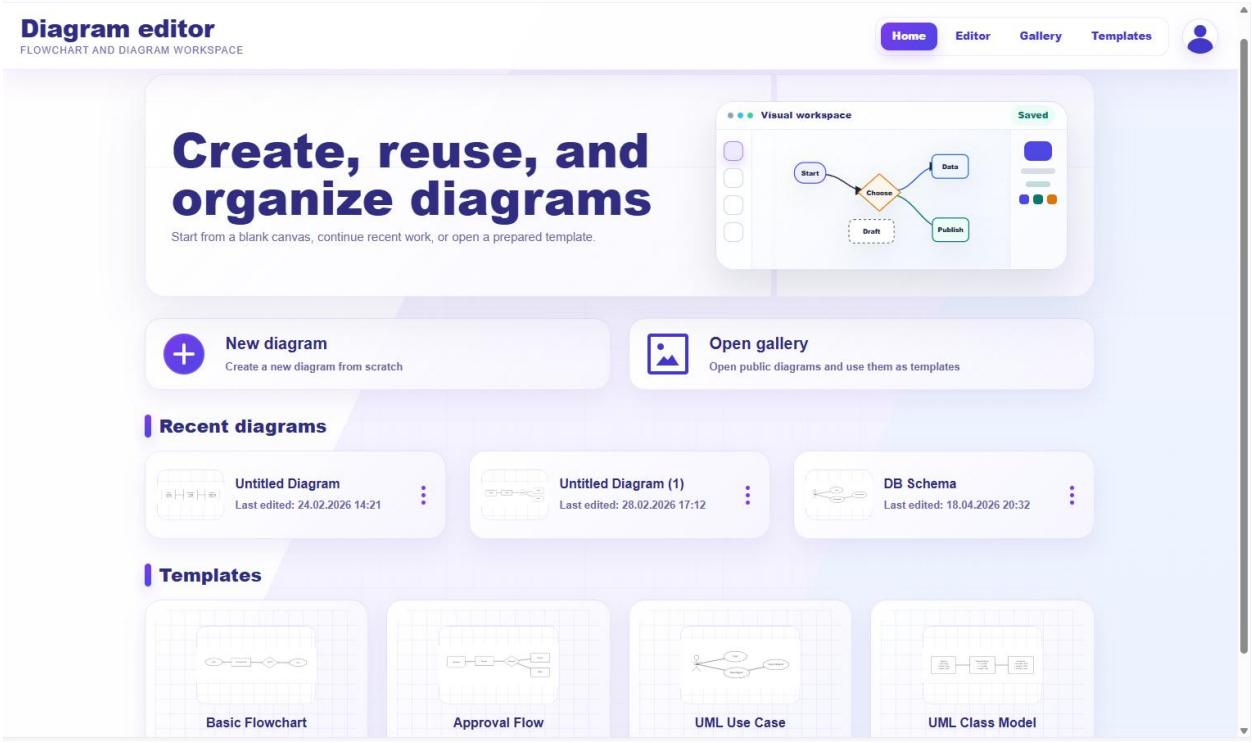

Рисунок 4.6 – Головна сторінка застосунку Diagram editor

Головна сторінка виконує роль навігаційного центру. На ній відображаються останні діаграми користувача, якщо він авторизований, а також доступні шаблони для швидкого створення нових схем. Для неавторизованого користувача доступне створення діаграми у режимі гостя, при цьому зміни тимчасово зберігаються у локальному сховищі браузера.

Сторінка реєстрації та авторизації

Для збереження робіт у базі даних користувач повинен створити обліковий запис або увійти в уже наявний. Сторінка авторизації містить дві вкладки: вхід та реєстрація. Під час реєстрації користувач вводить ім’я, адресу електронної пошти та пароль. Після успішної перевірки даних система створює новий обліковий запис і надає користувачу стандартну роль User.

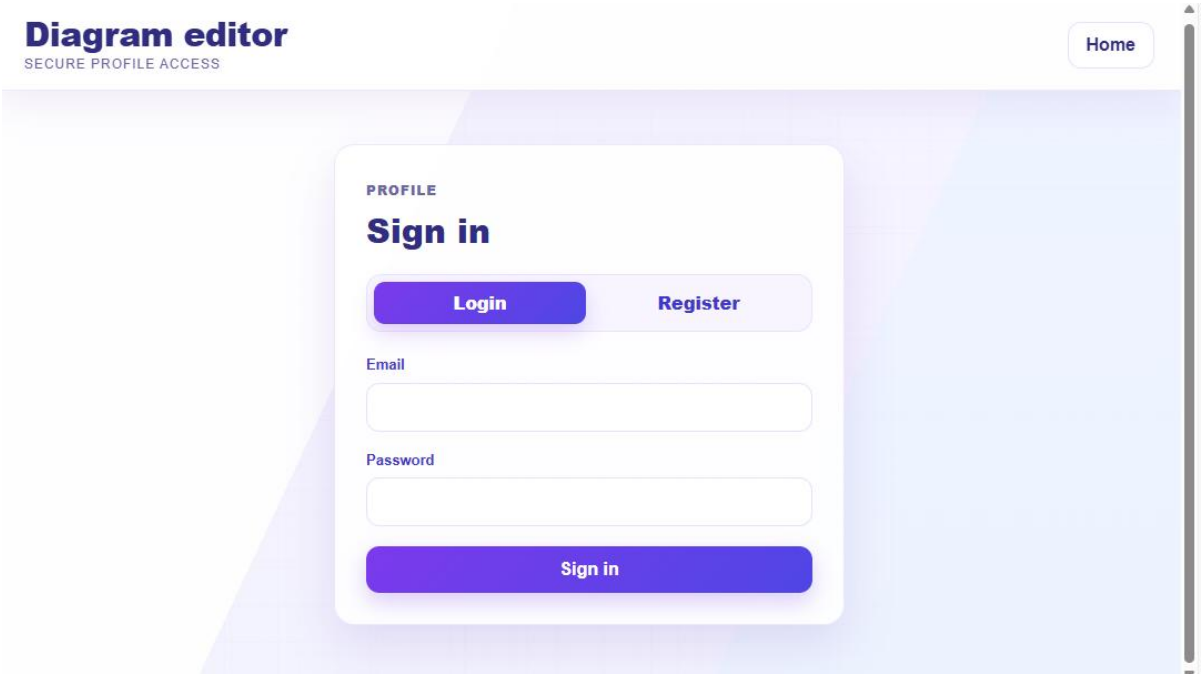

Рисунок 4.7 – Форма авторизації та реєстрації

Під час входу система перевіряє email та пароль. Паролі не зберігаються у відкритому вигляді: у базі даних зберігається лише захищений хеш пароля. Персональні поля користувача, зокрема ім’я та email, перед збереженням обробляються на сервері. У разі введення некоректних даних система виводить повідомлення про помилку без перезавантаження сторінки.

Компонент автентифікації створюється як SPA, сторінка взагалі не перезавантажується, а процеси перевірки, помилки та повідомлення відбувається в режимі «живого» процесу з використанням REST API. Це потрібно для перевірки введених користувачем даних за допомогою серверної частини. Якщо введено недійсне ім'я користувача або пароль, система одразу після введення інформації відобразить на екрані повідомлення про це, не перезавантажуючи сторінку. Це пришвидшить вхід для користувача в систему або реєстрацію облікового запису.

Існує дві вкладки сторінки входу: вхід та реєстрація. Під час реєстрації користувач надає своє ім'я, електронну пошту, пароль, і система створює обліковий запис користувача з роллю «user». Права доступу розрізняються за

окремими групами користувачів і складають основу для подальшого застосування модерації та адміністративної діяльності. У наведеній нище таблиці 4.8 описано огляд основних елементів інтерфейсу користувача редактора діаграм, а також показано спосіб використання окремих сторінок програми користувачем.

Таблиця 4.8 – Огляд інтерфейсу застосунку

Елемент	**Призначення**	**Коротка дія**
Navbar	Перехід між сторінками Home, Editor, Gallery, Templates, Profile	Натискання на пункт меню відкриває відповідну сторінку
Home	Початкова сторінка застосунку	Створення нової діаграми або відкриття останніх робіт
Editor	Робоча область редагування діаграм	Додавання, переміщення, редагування та з’єднання елементів
Templates	Список готових шаблонів	Вибір шаблону для швидкого створення діаграми
Gallery	Публічні діаграми користувачів	Перегляд схвалених діаграм і використання їх як шаблонів
Profile	Особистий кабінет користувача	Зміна імені, пароля, перегляд і відкриття власних діаграм
Moderation	Панель модератора	Перевірка діаграм, схвалення або відхилення публікації
Admin	Адміністративна панель	Управління користувачами, ролями, шаблонами та діаграмами

Сценарій користувача

Основний сценарій роботи користувача починається зі створення нової діаграми або вибору шаблону. У редакторі користувач може додавати елементи з бічної панелі, змінювати їх розмір, колір, заливку, товщину контуру, стиль тексту, а також створювати зв’язки між блоками. Для зручності підтримуються комбінації клавіш копіювання, вставлення, вирізання, скасування та повторення дій.

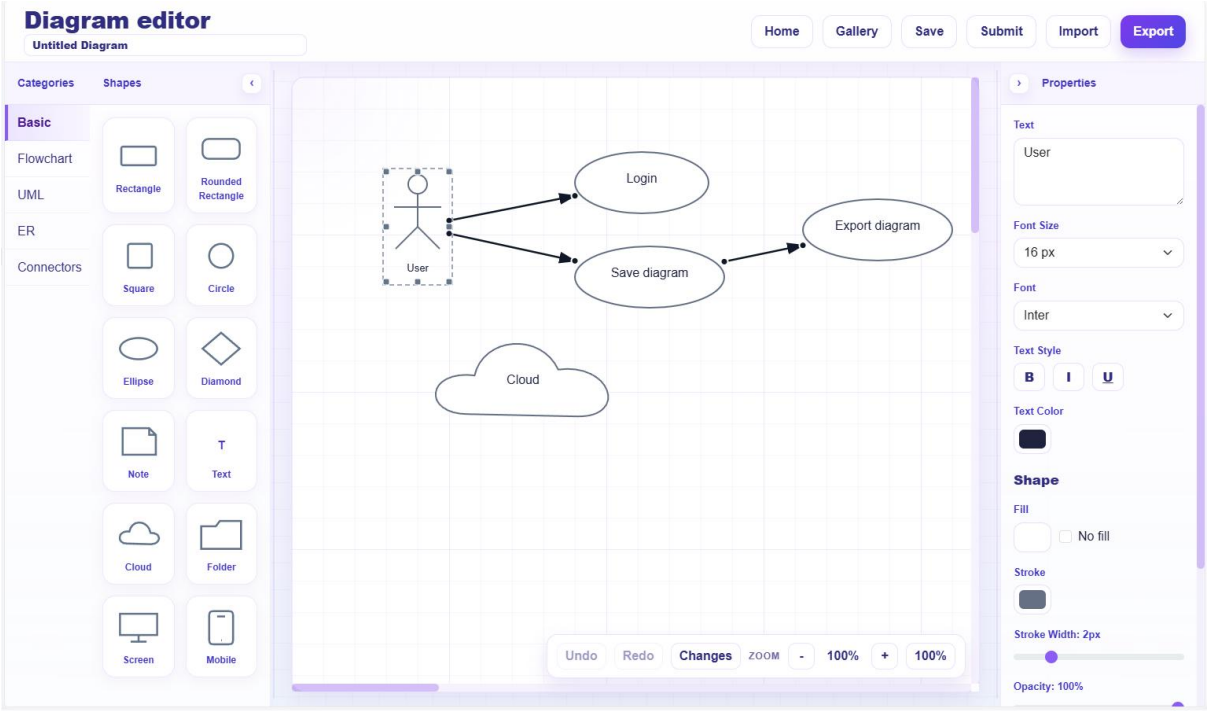

Рисунок 4.8 – Робоча область редактора діаграм

На панелі властивостей відображаються параметри вибраного об’єкта. Для фігур доступні налаштування тексту, шрифту, кольору, прозорості, заливки, контуру та розміру. Для ліній користувач може змінювати тип, колір, товщину та стиль з’єднання. Зміни застосовуються одразу, без необхідності додаткового підтвердження.

Після завершення редагування користувач може зберегти діаграму. Під час збереження відкривається окреме вікно, де вводиться назва діаграми, опис і теги. Для зареєстрованих користувачів робота зберігається у базі даних і

надалі відображається на головній сторінці у блоці останніх діаграм, а також у профілі користувача.

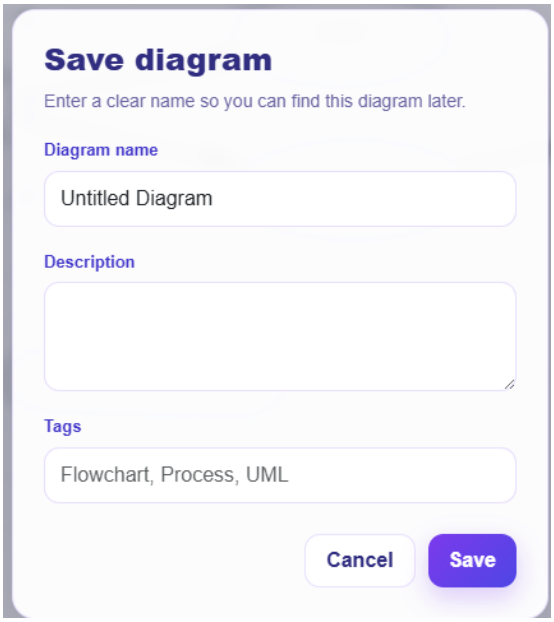

Рисунок 4.9 – Вікно збереження діаграми

Користувач також може експортувати діаграму у кількох форматах, зокрема JSON, SVG, PNG, PDF, CSV та Mermaid. Формат JSON призначений для подальшого редагування проєкту, SVG і PNG – для використання як зображення, PDF – для друку або додавання до документації, CSV – для табличного представлення елементів, а Mermaid – для текстового опису блок-схеми. Під час збереження користувач вводить назву діаграми, опис і теги, що спрощує подальший пошук та організацію робіт у системі. Збережені діаграми можуть повторно відкриватися для редагування, експортуватися або надсилатися на публікацію в галерею шаблонів.

Сценарій роботи з галереєю

Галерея містить лише ті діаграми, які були відправлені користувачами на модерацію та схвалені модератором. Для кожної діаграми відображається назва, автор, теги, опис за наявності, а також кнопка використання діаграми як шаблону. Це дає змогу повторно використовувати готові структури без ручного створення схеми з нуля.

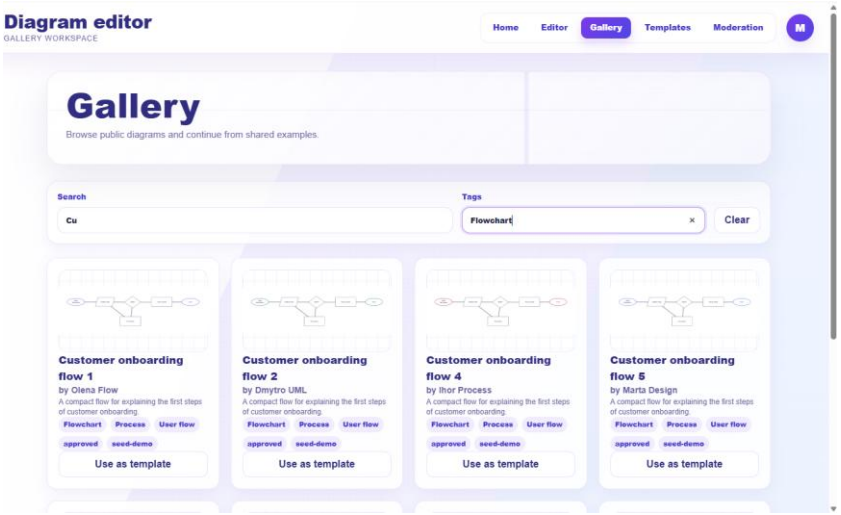

Рисунок 4.10 – Сторінка галереї діаграм

Якщо користувач хоче опублікувати власну діаграму в галереї, він надсилає її на модерацію. Після цього створюється заявка, яка потрапляє до черги модератора. До моменту схвалення діаграма залишається приватною і не відображається у публічній галереї.

Сценарій модератора

Модератор має доступ до окремої сторінки Moderation. На цій сторінці відображається список діаграм, надісланих користувачами для публікації. Модератор може відкрити діаграму для перегляду, перевірити її зміст, теги та опис, після чого прийняти рішення про схвалення або відхилення.

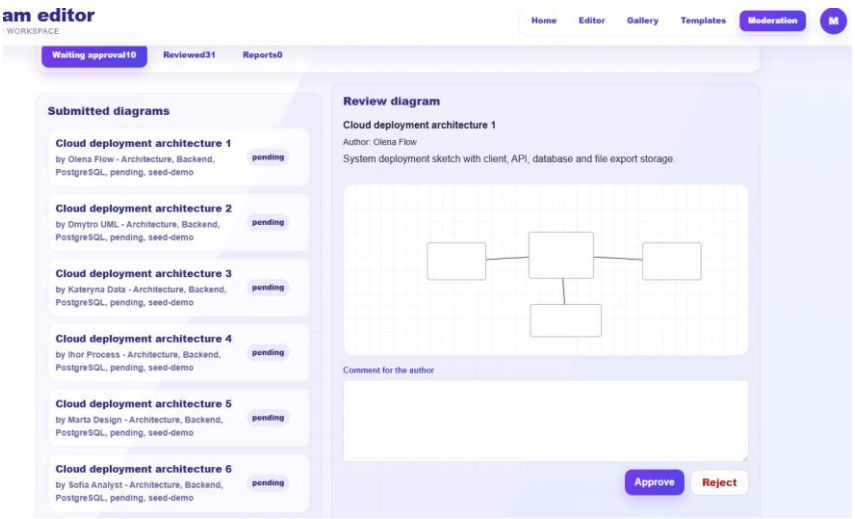

Рисунок 4.11 – Черга модерації діаграм

У разі схвалення діаграма отримує статус Published і стає доступною у галереї. Якщо діаграма не відповідає вимогам, модератор може її відхилити та вказати причину. Це повідомлення зберігається у системі й може бути використане користувачем для виправлення роботи.

Сценарій адміністратора

Адміністратор має доступ до адміністративної панелі, яка складається з кількох розділів: користувачі, діаграми, шаблони та статистика. У розділі користувачів адміністратор може переглядати зареєстровані облікові записи, змінювати ролі, блокувати або видаляти користувачів. Роль модератора або адміністратора може призначати лише адміністратор.

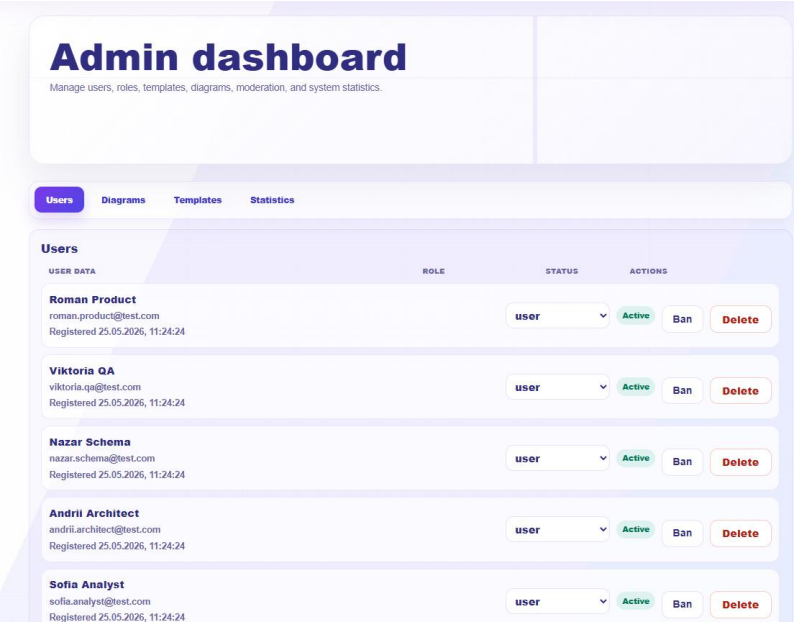

Рисунок 4.12 – Адміністративна панель користувачів

У розділі діаграм адміністратор може переглядати всі створені діаграми, змінювати їхній статус, публікувати, приховувати або видаляти матеріали. Розділ шаблонів використовується для створення та видалення готових шаблонів, які відображаються на сторінці Templates. Розділ статистики показує загальну кількість користувачів, діаграм, шаблонів та заявок на модерацію.

Таким чином, інтерфейс Diagram editor забезпечує повний цикл роботи з діаграмами: створення, редагування, локальне та серверне збереження, експорт, публікацію через модерацію і подальше повторне використання у вигляді шаблонів. Розподіл функцій між ролями користувача, модератора та адміністратора дозволяє контролювати доступ до критичних дій і підтримувати якість публічного контенту.

Висновки до розділу 4

У четвертому розділі розглянуто практичну реалізацію вебзастосунку Diagram Editor для створення та редагування блок-схем і діаграм. Описано структуру frontend та backend частин системи, організацію типів даних, REST API та основних React-компонентів редактора.

У межах підрозділу специфікації коду наведено основні структури даних Shape, Edge, Template та User, що використовуються для зберігання графічних об’єктів, зв’язків між ними та інформації користувачів. Показано приклади реалізації TypeScript-типів, API-маршрутів Express та компонентів інтерфейсу редактора.

Під час тестування ПЗ перевірено модульні, інтеграційні та навантажувальні сценарії. Модульні тести підтвердили коректність імпорту та експорту діаграм у форматах CSV, Mermaid, SVG і JSON. Інтеграційні перевірки засвідчили стабільність REST API та правильну роботу механізмів авторизації. Навантажувальні запуски показали, що серверна частина здатна стабільно обробляти до 1000 запитів за секунду без втрати даних і без збільшення error rate. У розділі наведено керівництво користувача для основних ролей системи: гостя, зареєстрованого користувача, модератора та адміністратора. Описано сценарії створення, редагування, експорту, публікації та модерації діаграм, а також принцип роботи адміністративної панелі.

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебзастосунок Diagram Editor для створення, редагування та збереження блок-схем і діаграм. Реалізована система забезпечує роботу з графічними елементами у браузері, підтримує побудову зв’язків між об’єктами, імпорт та експорт діаграм, а також серверне збереження даних користувачів і проєктів. У процесі виконання роботи було вирішено поставлені завдання:

1) проведено аналіз сучасних програмних засобів для створення діаграм та визначено їхні основні функціональні можливості;

2) сформовано функціональні та структурні вимоги до вебзастосунку;

3) спроєктовано клієнт-серверну архітектуру системи з використанням React, TypeScript, Node.js, Express та PostgreSQL;

4) реалізовано механізми створення, редагування та взаємодії графічних елементів на полотні редактора;

5) реалізовано збереження, відновлення та експорт діаграм у форматах CSV, Mermaid, SVG та JSON;

6) визначено ролі користувачів і реалізовано сценарії взаємодії для користувача, модератора та адміністратора;

7) проведено модульне, інтеграційне та навантажувальне тестування ПЗ.

У результаті тестування підтверджено коректність роботи основних функцій системи. Модульні тести засвідчили правильність імпорту та експорту діаграм, інтеграційні перевірки підтвердили стабільність REST API та механізмів авторизації, а навантажувальні сценарії показали стабільну роботу серверної частини навіть при навантаженні до 1000 запитів за секунду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Двірничук К. В., Вацек Д. О. Веб-програмування та веб-дизайн : навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2022. 472 с.

2. Говоров І. Аналіз переваг та недоліків статичних та динамічних сайтів. Матеріали науково-практичної конференції за підсумками виробничої практики здобувачів вищої освіти спеціальності 126 «Інформаційні системи та технології». Полтава : ПДАУ, 2022. С. 75–77.

3. Nielsen Norman Group. User Experience (UX) : website. URL: https://www.nngroup.com/articles/definition-user-experience/ (Accessed: 30.04.2026).

4. React Team. React Documentation : website. URL: https://react.dev (Accessed: 30.04.2026).

5. MDN Web Docs. SVG: Scalable Vector Graphics : website. URL: https://developer.mozilla.org/en-US/docs/Web/SVG (Accessed: 30.04.2026).

6. W3C. Scalable Vector Graphics (SVG) Specification : website. URL: https://www.w3.org/TR/SVG2/ (Accessed: 30.04.2026).

7. diagrams.net : website. URL: https://www.diagrams.net (Accessed: 30.04.2026).

8. Lucidchart : website. URL: https://www.lucidchart.com (Accessed: 30.04.2026).

9. Creately : website. URL: https://creately.com (date of access: 30.04.2026).

10. TypeScript Team. TypeScript Documentation : website. URL: https://www.typescriptlang.org/docs/ (Accessed: 30.04.2026).

11. PostgreSQL Global Development Group. PostgreSQL Documentation : website. URL: https://www.postgresql.org/docs/ (Accessed: 30.04.2026).

12. MDN Web Docs. HTML Drag and Drop API : website. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API (Accessed: 30.04.2026).

13. Google. Web Performance Fundamentals : website. URL: https://web.dev/learn/performance/ (Accessed: 30.04.2026).

14. Tesoriero R., Rueda A., Gallud J. A., Lozano M. D., Fernando A. Transformation architecture for multi-layered Web application source code generation. IEEE Access. 2022. Vol. 10. pp. 5223–5237.

15. Chen W., Haque A., Sedig K. Design of Interactive Visualizations for Next-Generation Ultra-Large Communication Networks. IEEE Access. 2021. Vol. 9. pp. 26968–26982.

16. Zustand Documentation : website. URL: https://zustand.docs.pmnd.rs/ (Accessed: 30.04.2026).

17. Vite Team. Vite Documentation : website. URL: https://vite.dev/guide/ (Accessed: 25.04.2026).

18. Cobilean V., Mavikumbure H. S., McBride B. J. A Review of Visualization Methods for Cyber-Physical Security: Smart Grid Case Study. IEEE Access. 2023. Vol. 11. pp. 1–15.

19. Lokumarambage M. U., Gowrisetty V. S. S., Rezaei H. Wireless End-to-End Image Transmission System Using Semantic Communications. IEEE Access. 2023. Vol. 11. pp. 37149–37163.

20. Node.js. Node.js Documentation : website. URL: https://nodejs.org/docs/latest/api/ (Accessed: 25.04.2026).

21. Express.js. Express Documentation : website. URL: https://expressjs.com/ (Accessed: 05.05.2026).

22. Bootstrap Team. Bootstrap Documentation : website. URL: https://getbootstrap.com/docs/ (Accessed: 26.04.2026).