

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

«Веб-додаток для створення блок-схем UML»

Виконав студент групи ІІЗ-21-1
ШУМЛЯКІВСЬКИЙ Роман Володимирович

Керівник:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ШУМЛЯКІВСЬКИЙ Роман Володимирович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Веб-додаток для створення блок-схем UML»**, затверджена
наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с.**

Вихідні дані для роботи: Об'єктом дослідження є процес побудови веб-
додатків прикладного спрямування. Предметом дослідження є моделі, алгоритми,
технології та засоби розробки веб-додатку для створення блок-схем UML.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	02.03.2025р.	02.03.2025р.
2	Лобанчикова Н.М.	04.04.2025р.	04.04.2025р.
3	Лобанчикова Н.М.	02.05.2025р.	02.05.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного додатку для створення блок-схем UML та пояснювальної записки. Пояснювальна записка до роботи містить 82 сторінку, 49 ілюстрацій та 11 таблиць.

Метою роботи є розробка веб-додатку для створення блок-схем UML для підвищення зручності процесів моделювання програмного забезпечення, аналізу бізнес-процесів та проектування систем. Запропонований додаток спрямований на підвищення продуктивності розробників, аналітиків та дизайнерів, а також на полегшення процесу документування та представлення проєктних рішень.

Об'єктом дослідження є процес побудови веб-додатків прикладного спрямування.

Предметом дослідження є моделі, алгоритми, технології та засоби розробки веб-додатку для створення блок-схем UML.

У роботі було використано методи об'єктно-орієнтованого аналізу та проектування, веб-розробки, інтерфейсного дизайну та тестування програмного забезпечення. Особливу увагу приділено оптимізації взаємодії користувача з додатком та забезпеченню гнучкості інструменту для роботи з різними типами діаграм UML.

КЛЮЧОВІ СЛОВА: БЛОК-СХЕМИ UML, ДОДАТОК ДЛЯ МОДЕЛЮВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ-ТЕХНОЛОГІЇ, ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ.

					ІПЗ.КР.Б-121-25-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-додаток для створення блок-схем UML Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		Шумляківський Р.					3	81
Керівник		Лобанчикова Н.М.				ФІКТ, гр.ІПЗ-21-1		
Рецензент		Савіцький Р.С.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's final graduation qualification consists of a web application for creating UML diagrams and an explanatory note. The explanatory note contains 82 pages, 49 illustrations, and 11 tables.

The purpose of this work is the development of a web application for creating UML diagrams to enhance the convenience of software modeling processes, business process analysis, and system design. The proposed application aims to increase the productivity of developers, analysts, and designers, as well as to facilitate the process of documenting and presenting design solutions.

The object of the study is the process of building web applications for practical purposes.

The subject of the study is the models, algorithms, technologies, and tools for developing a web application for creating UML diagrams. In this work, methods of object-oriented analysis and design, web development, interface design, and software testing were used. Particular attention was paid to optimizing user interaction with the application and ensuring the flexibility of the tool for working with various types of UML diagrams.

KEYWORDS: UML DIAGRAMS, MODELING TOOL, SOFTWARE, WEB TECHNOLOGIES, OBJECT-ORIENTED ANALYSIS.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВЕБ-ДОДАТКУ ДЛЯ СТВОРЕННЯ БЛОК-СХЕМ UML.....	9
1.1 Постановка задачі	9
1.2. Аналіз існуючих рішень та програм-аналогів для створення блок-схем .	12
1.3. Вибір архітектури веб-додатку.....	18
1.4. Інструментальні засоби реалізації програмного продукту та вимоги до апаратних засобів.....	23
Висновки до першого розділу	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ СТВОРЕННЯ UML БЛОК- СХЕМ.....	27
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	27
2.2 Розробка бази даних системи	37
2.3 Проєктування та реалізація алгоритмів роботи системи.....	43
2.4 Реалізація функціоналу додатка	47
Висновок до другого розділу	52
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ.....	53
3.1. Інтерфейс програмного продукту	53
3.2. Порядок розгортання клієнт-серверного додатку	59
3.3. Тестування веб-додатку для побудови блок-схем UML.....	61
Висновок до третього розділу	65
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТКИ.....	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

JS – JavaScript

JSX – JavaScript XML – розширення синтаксису JavaScript

MVC – Model-View-Controller – схема розділення даних додатків і керуючої логіки на три окремі компоненти

REST API – архітектурний стиль взаємодії компонентів розподіленої програми у мережі

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. У сучасному світі, де цифрові технології активно інтегруються в усі сфери життя, зростає потреба у зручних інструментах для візуалізації та проектування процесів. Блок-схеми є одним із найпоширеніших способів представлення алгоритмів, бізнес-процесів, технічних рішень та інших структур даних. Вони допомагають спрощувати складні концепції, полегшуючи їх розуміння та аналіз. Проте багато існуючих інструментів для створення блок-схем мають обмежену функціональність, недостатньо зручний інтерфейс або високу вартість, що ускладнює їх використання як для професіоналів, так і для початківців.

Окремої уваги заслуговує той факт, що сучасні умови роботи вимагають інструментів, орієнтованих на командну взаємодію та гнучкість. Зростання популярності віддаленої роботи та гібридних форматів співпраці підвищує потребу в онлайн-рішеннях, які дозволяють одночасно працювати над діаграмами, обговорювати зміни та інтегрувати їх у загальний процес розробки. Крім того, важливим аспектом є сумісність з іншими інструментами, такими як системи управління проєктами, середовища розробки та хмарні сховища, що дозволяє автоматизувати робочий процес та зменшити навантаження на користувачів.

Метою випускної роботи є розробка веб-додатку для створення блок-схем UML для підвищення зручності процесів моделювання програмного забезпечення, аналізу бізнес-процесів та проектування систем. Додаток спрямований на підвищення ефективності роботи розробників, аналітиків і дизайнерів шляхом автоматизації процесу створення діаграм UML, а також на полегшення документування та представлення проєктних рішень.

Для досягнення поставленої мети необхідним є вирішення низки взаємозалежних інженерних задач, зокрема:

- провести аналіз існуючих інструментів для створення UML-діаграм;
- обґрунтувати вибір архітектури, технологій та засобів реалізації програмного продукту;

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

- здійснити проектування веб-додатку для створення блок-схем UML;
- провести програмну реалізацію основного функціоналу програми, який дозволить користувачам зручно проектувати, редагувати та експортувати UML-блок-схеми;
- провести тестування додатку, розробити рекомендації щодо встановлення та подальшого використання.

Реалізація цих завдань дозволить створити універсальний інструмент, який можна буде використовувати як у навчальному процесі, так і в професійній діяльності для моделювання, проектування та аналізу систем.

Об’єкт дослідження. Об’єктом дослідження є процес побудови веб-додатків прикладного спрямування.

Предмет дослідження. Предметом дослідження є моделі, алгоритми, технології та засоби розробки веб-додатку для створення блок-схем UML.

Практичне значення одержаних результатів. Розроблений веб-додаток для створення UML-діаграм має важливе прикладне значення, оскільки він дозволить суттєво спростити процес проектування програмного забезпечення, аналізу бізнес-процесів та системного дизайну. На відміну від існуючих рішень, цей інструмент буде орієнтований на зручність користувача, пропонуючи інтуїтивний інтерфейс, можливість спільної роботи в режимі реального часу та безпроблемний експорт діаграм у популярних форматах.

Завдяки автоматизації рутинних операцій – таких як вирівнювання елементів, перевірка коректності зв’язків, генерація коду на основі діаграм – додаток дозволить фахівцям зосередитися на творчих аспектах проектування, а не на технічних деталях візуалізації. Крім того, веб-орієнтований підхід забезпечить доступ до інструменту з будь-якого пристрою без необхідності встановлення додаткового ПЗ, що робить його зручним для навчальних закладів, стартапів і великих команд, які працюють у розподіленому форматі. Впровадження такого рішення дозволить не лише прискорити процес моделювання, а й підвищити якість документації за рахунок стандартизованого підходу до створення діаграм.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВЕБ-ДОДАТКУ ДЛЯ СТВОРЕННЯ БЛОК-СХЕМ UML

1.1 Постановка задачі

Метою випускної кваліфікаційної роботи бакалавра є розробка веб-додатку для створення блок-схем, який надасть користувачам інструмент для візуалізації алгоритмів, бізнес-процесів та інших структур даних. Такий додаток дозволить автоматизувати процес створення блок-схем, забезпечивши зручний інтерфейс для проектування, редагування та експорту результатів. Веб-додаток буде враховувати можливості, що зображені на діаграмі варіантів використання, і буде включати функціональність для створення схем різної складності, збереження проєктів у хмарі, експорту у популярні формати (наприклад, JPG) та спільної роботи над проєктами. Така система сприятиме підвищенню продуктивності користувачів та забезпеченню доступності інструменту для широкого кола користувачів.

Сьогоднішній етап розвитку технологій відкриває нові можливості для вдосконалення інструментів візуалізації даних. Розроблений веб-додаток не лише відповідає сучасним вимогам, але й має потенціал для подальшого розширення функціоналу, такого як інтеграція зі штучним інтелектом для автоматичного генерування схем на основі текстового опису, підтримка VR/AR-режимів для інтерактивної роботи з діаграмами або розширені аналітичні можливості для оцінки складності алгоритмів. Такі перспективи роблять даний проєкт актуальним не лише на сьогодні, а й у довгостроковій перспективі, забезпечуючи його конкурентоспроможність у міру розвитку цифрових технологій.

У сучасному світі, де цифрові технології активно інтегруються в усі сфери життя, роль веб-розробки набуває особливої значущості. Створення зручних, функціональних та привабливих інструментів є ключовим фактором успіху для багатьох галузей, зокрема для освіти, програмування, бізнес-аналітики та проектування. Блок-схеми є одним із найпоширеніших способів представлення інформації, і їх використання поширюється від навчальних закладів до великих

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

корпорацій. У цьому контексті розробка веб-додатку для створення блок-схем стає важливим етапом у забезпеченні користувачів інструментом, який відповідає їх потребам.

Опис системи та її функціональності. Запропонована система має широкий спектр можливостей для роботи з блок-схемами. Її архітектура передбачає реалізацію ключових модулів, які забезпечать повноцінний цикл роботи з діаграмами. Основним елементом системи є модуль авторизації, що дозволить користувачам реєструватися та отримувати доступ до своїх проектів, історії робіт і функцій спільної роботи. Для тих, хто лише ознайомлюється з системою, передбачений гостьовий доступ з можливістю перегляду базового функціоналу без необхідності реєстрації.

Центральним компонентом системи виступає модуль створення блок-схем, який надає користувачам інструменти для додавання різноманітних блоків (початок/кінець, процеси, рішення, введення/виведення) та їх з'єднання за допомогою стрілок. Додатково розроблятиметься модуль редагування та налаштувань, що дозволить персоналізувати зовнішній вигляд діаграм – змінювати кольори, розміри, шрифти блоків, налаштовувати макет схеми та додавати коментарі. Користувачі зможуть тонко налаштовувати візуальні параметри, включаючи положення елементів, колір фону, вирівнювання тексту та інші стильові характеристики.

Збереження та спільна робота. Система включатиме потужний модуль збереження та експорту, який забезпечить зберігання проектів у хмарному сховищі та експорт у популярні формати зображень і документів (PNG, PDF, SVG). Особливістю цього модуля стане можливість обміну проектами через посилання, а також імпорт/експорт у JSON форматі для подальшої обробки. Для організації колективної роботи передбачається реалізація спеціального модуля, що дозволить декільком користувачам одночасно редагувати схеми, обговорювати зміни в інтегрованому чаті та відстежувати історію змін.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Адміністрування та розробка. Адміністративний модуль надасть керівникам системи інструменти для управління користувачами, проектами та системними налаштуваннями. Адміністратори зможуть аналізувати статистику використання додатку, додавати нові шаблони та елементи, а також модерувати публічні блок-схеми. Процес розробки веб-додатка охоплює кілька ключових етапів, починаючи з проектування інтерфейсу, яке передбачає створення концептуального дизайну з урахуванням потреб користувачів та специфіки роботи з блок-схемами. Будуть розроблені всі необхідні UI-елементи: кнопки, панелі інструментів, палітри блоків та робоча область для редагування схем.

Технічна реалізація. Архітектура клієнтської частини передбачає вибір сучасних фреймворків (React, Vue.js або Angular) та створення інтерактивних компонентів для динамічної роботи зі сторінками. Особлива увага приділятиметься реалізації механізмів перетягування блоків, їх з'єднання та редагування тексту. Серверна частина буде розроблена на базі таких технологій як Node.js, Django або Express.js, що забезпечить надійне зберігання проектів, обробку запитів та функціонал спільної роботи. Система безпеки включатиме механізми авторизації та систему ролей для диференційованого доступу до функцій додатка.

Основні функції та тестування. Ядро системи складатиметься з інструментів створення та редагування блок-схем, включаючи всі необхідні операції з блоками, механізми збереження проектів та їх експорту, функції колаборативної роботи та публікації схем. Завершальним етапом розробки стане комплексне тестування, яке охопить перевірку всіх модулів, юзабіліті інтерфейсу та продуктивність системи при роботі з великими схемами. Буде проведена оптимізація роботи додатка для забезпечення стабільної роботи навіть зі складними проектами.

Результатом виконання плану дій є створення платформи, за допомогою якої користувачі матимуть змогу швидко та зручно створювати, редагувати та експортувати блок-схеми. Іншими словами, користувач зможе легко проектувати складні алгоритми, бізнес-процеси та інші структури даних, отримуючи при цьому доступ до інструментів для спільної роботи та збереження проектів.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

1.2. Аналіз існуючих рішень та програм-аналогів для створення блок-схем

Перед розробкою власного веб-додатку для створення блок-схем необхідно ретельно вивчити ринок існуючих рішень, щоб виявити їх сильні та слабкі сторони, а також визначити потенційні напрямки для вдосконалення. Серед сучасних інструментів особливої уваги заслуговують такі популярні платформи як Lucidchart [10], draw.io [7], Microsoft Visio [12] та Creately [6], кожна з яких має свої унікальні особливості.

Lucidchart є потужним онлайн-інструментом, який активно використовується у бізнес-середовищі, освітніх закладах та сфері програмування для створення різноманітних діаграм та блок-схем [10]. Його головною перевагою є інтуїтивно зрозумілий інтерфейс (див. рис.1.1), який дозволяє користувачам швидко освоїти основні функції. Система підтримує спільну роботу в реальному часі, що робить її ідеальним рішенням для командних проєктів. Доступ до широкої бібліотеки шаблонів та елементів діаграм значно спрощує процес створення професійно виглядаючих схем. Інтеграція з популярними сервісами, такими як Google Drive, Slack та Microsoft Teams, додатково розширює можливості платформи. Користувачі можуть експортувати свої роботи у різних форматах, включаючи PDF, PNG, SVG та VDX, що забезпечує гнучкість у подальшому використанні створених матеріалів.

Однак Lucidchart має і певні обмеження [10]. Безкоштовна версія сервісу накладає суттєві обмеження на кількість документів та елементів, що може бути недостатньо для активних користувачів. Повний доступ до функціоналу вимагає оформлення платної підписки, причому вартість для корпоративних клієнтів може бути досить високою. Це робить сервіс менш доступним для студентів або фрілансерів, які потребують професійного інструменту, але не мають можливості інвестувати значні кошти у програмне забезпечення.

Перспективи вдосконалення функціоналу. Проведений аналіз показує, що незважаючи на широкі можливості існуючих рішень, залишається простір для

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

вдосконалення. Особливу увагу варто приділити створенню більш доступного за ціною рішення, яке б зберегло основні переваги професійних інструментів. Важливим аспектом є розробка інтуїтивного інтерфейсу, який буде зручним як для професіоналів, так і для новачків. Додатковою конкурентною перевагою може стати глибша інтеграція з безкоштовними хмарними сервісами та відкритими платформами, що значно розширить аудиторію користувачів. Також варто врахувати потребу у локальних рішеннях, які б дозволяли працювати з конфіденційними даними без необхідності їх завантаження на сторонні сервери.

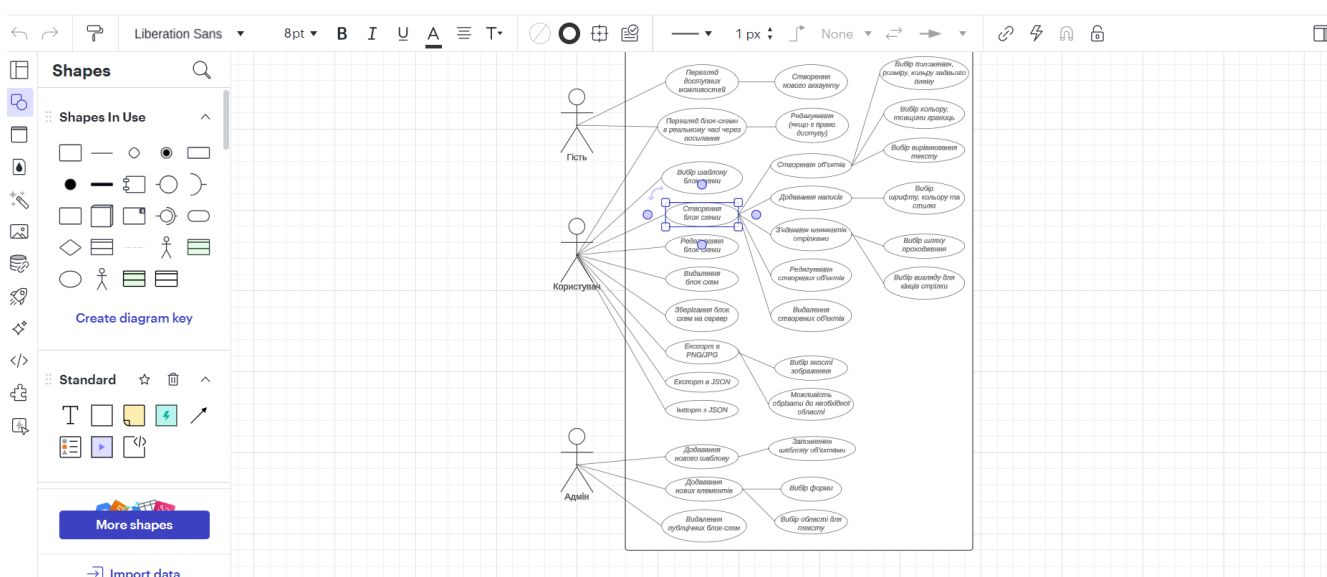


Рис. 1.1. Інтерфейс Lucidchart

Серед безкоштовних рішень для створення діаграм особливо виділяється draw.io, який нещодавно змінив назву на Diagrams.net [7]. Цей онлайн-інструмент став популярним завдяки своїй простоті та доступності – для початку роботи з ним навіть не потрібна реєстрація. Основною перевагою сервісу є його повна безкоштовність як для особистого, так і для комерційного використання, що робить його привабливим вибором для студентів, фрілансерів та малого бізнесу.

Draw.io пропонує мінімалістичний, але досить зручний інтерфейс (див. рис.1.2), який дозволяє швидко створювати базові блок-схеми. Важливою особливістю є можливість роботи офлайн завдяки наявності десктопної версії, що

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

є особливо цінним для користувачів з нестабільним інтернет-з'єднанням або тих, хто працює з конфіденційними даними. Сервіс підтримує експорт у різні формати, включаючи растрові (PNG), векторні (SVG) та редагувані (XML), а також інтегрується з популярними хмарними сховищами, такими як Google Drive та OneDrive [7].

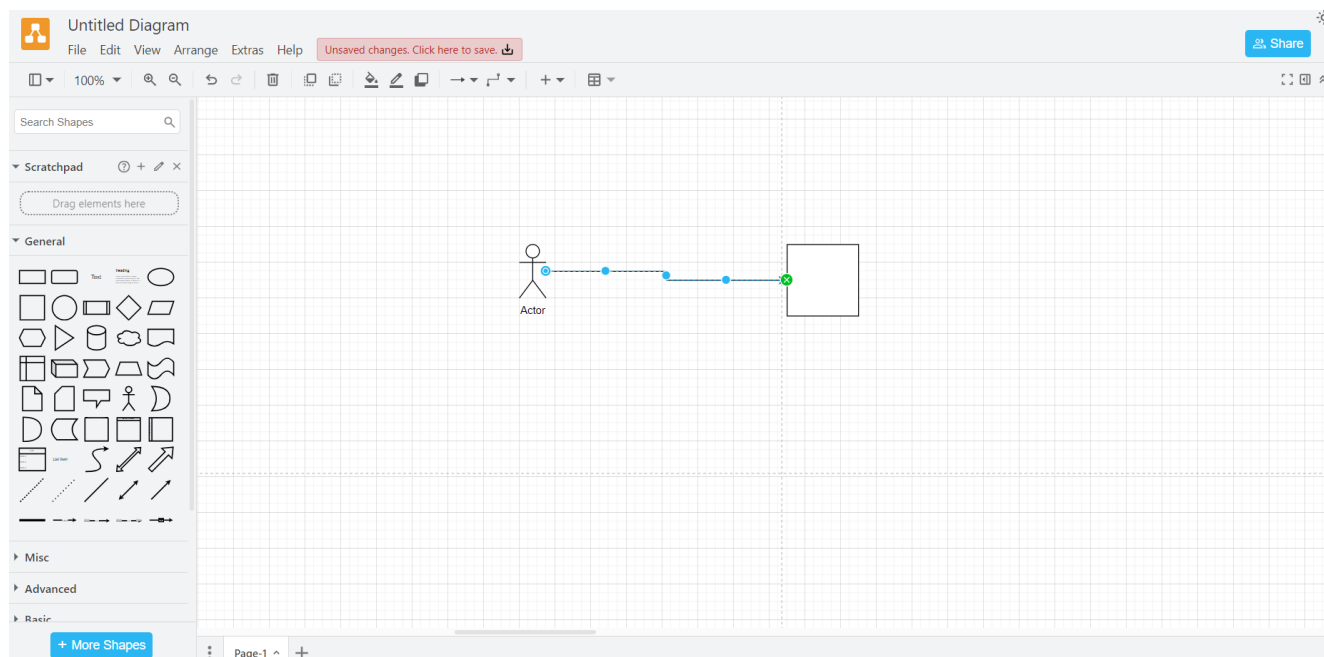


Рис. 1.2. Інтерфейс draw.io

Однак draw.io має і свої обмеження [7]. Найсуттєвішим недоліком є відсутність повноцінної підтримки спільної роботи в реальному часі, що суттєво обмежує його використання у великих командах. Бібліотека шаблонів досить обмежена, особливо коли мова йде про складні спеціалізовані діаграми. Деяким користувачам інтерфейс може здатися надто спрощеним і позбавленим сучасного дизайну. Хоча draw.io є чудовим вибором для індивідуальних користувачів, які потребують базового функціоналу, для професійних команд, які працюють над складними проектами, його можливостей може виявитися недостатньо.

Microsoft Visio займає особливе місце серед професійних інструментів для створення діаграм і блок-схем [12]. Як частина екосистеми Microsoft Office, цей програмний продукт став стандартом де-факто у багатьох корпоративних

середовищах, де використовується для моделювання бізнес-процесів, проектування мережевих інфраструктур та візуалізації організаційних структур.

Головною силою Visio є його глибока інтеграція з іншими продуктами Microsoft, зокрема Excel, Word та SharePoint. Ця взаємодія дозволяє легко імпортувати дані для автоматичного створення діаграм та експортувати результати в інші документи. Програма пропонує вражаючий набір професійних шаблонів (див. рис.1.3) і елементів, які покривають практично будь-які потреби у візуалізації – від технічних схем до стратегічних бізнес-моделей [12]. Можливість експорту в різні формати забезпечує гнучкість у подальшому використанні створених матеріалів.

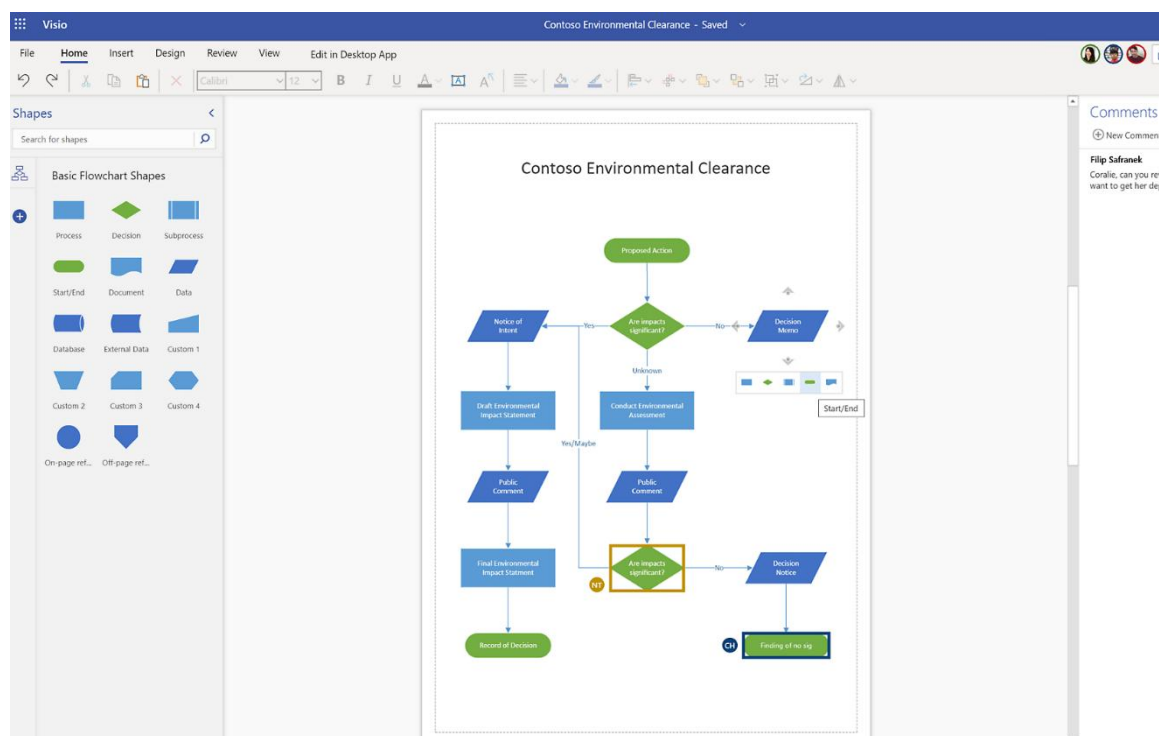


Рис. 1.3. Інтерфейс Microsoft Visio

Проте Visio має істотні обмеження, які можуть бути вирішальними для багатьох користувачів. Найбільш очевидним є висока вартість ліцензій, яка робить програму недоступною для індивідуальних користувачів та малого бізнесу. Іншим суттєвим недоліком є платформна обмеженість – настільна версія доступна виключно для Windows, що виключає користувачів macOS та Linux. Відсутність

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

будь-якої безкоштовної версії або навіть обмеженого безкоштовного тарифу додатково звужує коло потенційних користувачів.

Creately – це онлайн-платформа для створення діаграм, блок-схем, карт знань та інших візуальних матеріалів [6]. Вона орієнтована на команди, які працюють над проектами, і пропонує широкий набір інструментів для спільної роботи. Інтерфейс Creately інтуїтивно зрозумілий (див. рис.1.4), що дозволяє користувачам швидко освоїти основні функції. Платформа підтримує спільну роботу в реальному часі, що є важливим для команд, які працюють віддалено. Крім того, сервіс пропонує велику бібліотеку шаблонів та елементів, що спрощує створення професійних діаграм. Інтеграція з популярними сервісами, такими як Google Drive, Confluence та Trello, робить роботу ще зручнішою. Також є можливість експортувати матеріали у різні формати, включаючи PNG, PDF та SVG.

Однак у Creately є і недоліки. Безкоштовна версія має обмежений функціонал, що може бути недостатньо для серйозних проектів. Корпоративні тарифи є досить дорогими, що може стати перешкодою для малих команд або індивідуальних користувачів. Крім того, інтерфейс може здатися перенасиченим для новачків, особливо при першому знайомстві з платформою.

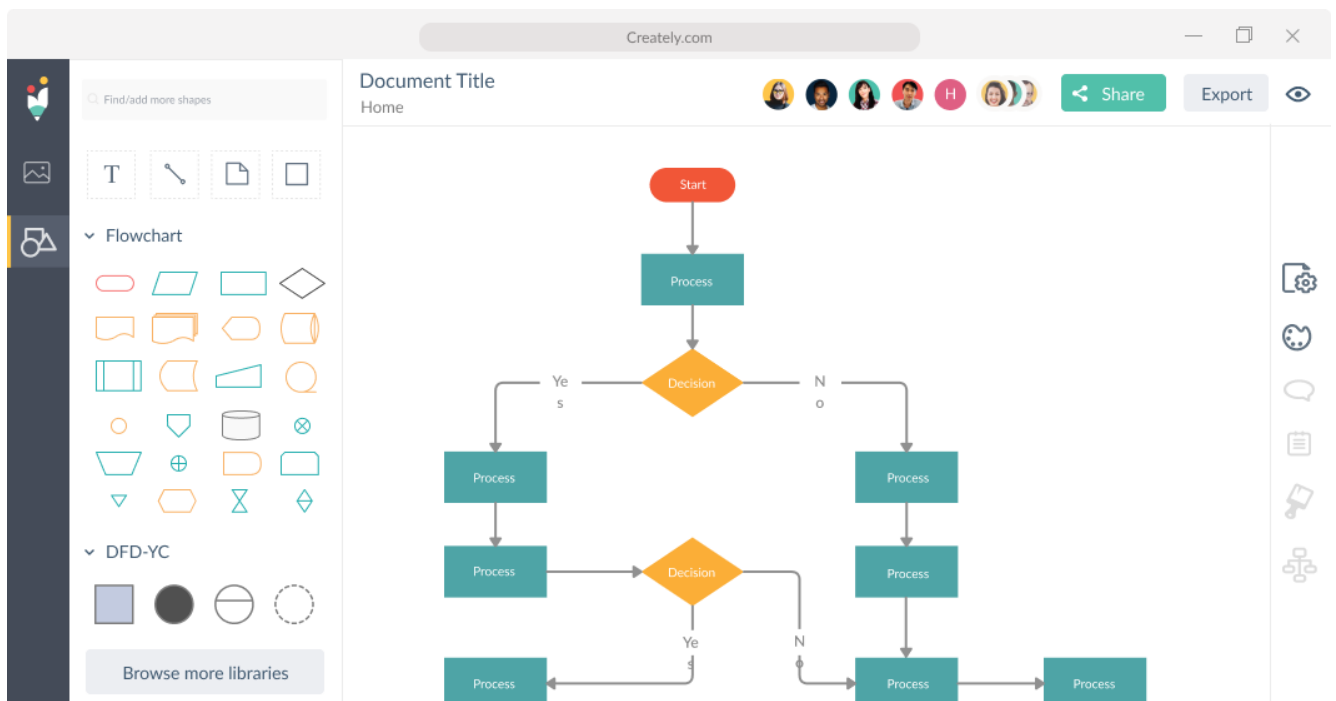


Рис. 1.4. Інтерфейс Creately

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Для кращого розуміння сильних та слабких сторін існуючих аналогів, нижче наведено порівняльну таблицю (див. табл.1.1), яка враховує ключові аспекти їх функціоналу.

Таблиця 1.1

Порівняльний аналіз інструментів для побудови блок-схем

Характеристика	Lucidchart	draw.io	Microsoft Visio	Creately
Наявність шаблонів	Широкий набір	Обмежений набір	Професійні шаблони	Різноманітні шаблони
Спільна робота	Так	Обмежено	Так	Так
Інтеграція з сервісами	Google Drive, Slack	Google Drive, OneDrive	Microsoft Office	Google Drive, Confluence
Експорт форматів	PDF, PNG, SVG	PDF, PNG, SVG, XML	PDF, PNG, SVG	PDF, PNG, SVG

Аналіз аналогічних інструментів показує, що кожен із них має свої сильні та слабкі сторони. При розробці майбутнього веб-додатку варто звернути увагу на кілька ключових аспектів [1]. По-перше, важливо забезпечити простий та інтуїтивно зрозумілий інтерфейс, щоб будь-хто міг легко користуватися додатком без тривалого навчання [26]. По-друге, наявність різноманітних шаблонів для створення блок-схем (наприклад, для алгоритмів, бізнес-процесів чи організаційних структур) значно прискорить роботу користувачів. Ще одним важливим фактором є можливість спільної роботи в реальному часі [31]. Крім того, додаток повинен підтримувати експорт у популярні формати (наприклад, PDF, PNG або SVG) та інтегруватися з хмарними сервісами [29].

На основі проведеного дослідження можна зробити висновок, що майбутній веб-додаток має всі шанси зайняти проміжну нішу між безкоштовними рішеннями (на кшталт draw.io) та професійними платформами (такими як Lucidchart чи Microsoft Visio).

1.3. Вибір архітектури веб-додатку

При розробці веб-додатку для створення блок-схем важливо обрати оптимальну архітектуру, яка забезпечить чітке розділення обов'язків між компонентами, спростить процес розробки та підтримки, а також забезпечить масштабованість та гнучкість системи. У цьому розділі буде проведений аналіз декількох популярних архітектурних підходів, таких як MVC (Model-View-Controller) [14-16], MVVM (Model-View-ViewModel), Flux [8] та Microservices, з подальшим обґрунтуванням вибору архітектури MVVM [17].

Архітектура MVC (Model-View-Controller) є однією з найпоширеніших моделей для розробки веб-додатків, рис. 1.5 [23]. Її основна ідея полягає у чіткому поділі програми на три компоненти. Model відповідає за роботу з даними, включаючи взаємодію з базою даних [14]. View забезпечує відображення інформації користувачеві через інтерфейс [15]. Controller обробляє дії користувача та керує взаємодією між Model і View [16].

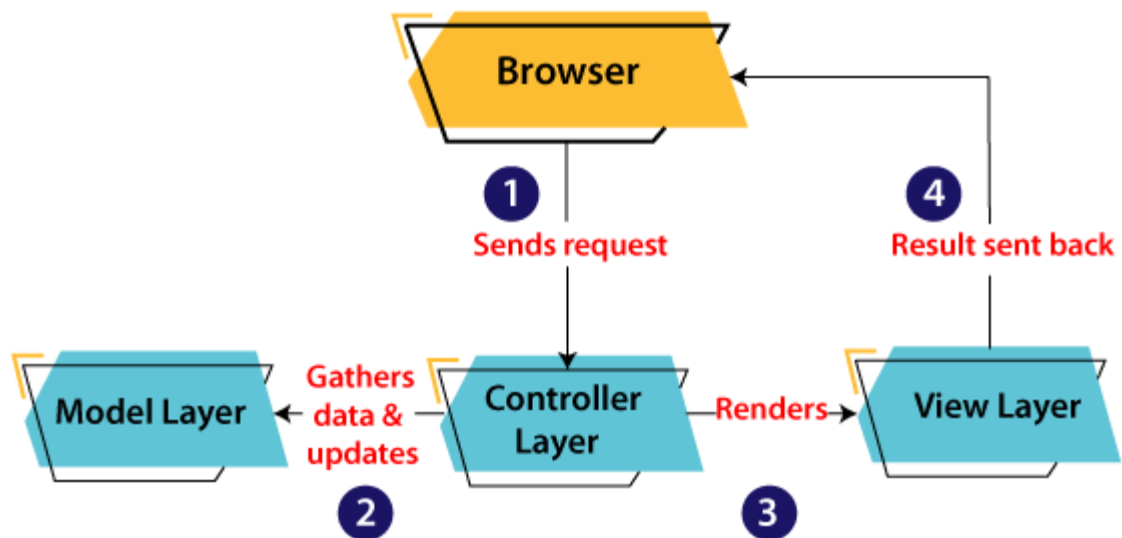


Рис. 1.5. Архітектура MVC

Основні переваги MVC – це простота розуміння, чітке розділення логіки, що спрощує тестування та підтримку, а також добра придатність для невеликих і середніх проектів [15]. Однак архітектура має й недоліки: при зростанні складності проекту Controller може стати надто перевантаженим, а відсутність механізму

двосторонньої синхронізації між View та Model ускладнює роботу зі складними інтерактивними елементами [3]. MVC добре підходить для простих додатків, але для веб-рішень з інтенсивною інтерактивністю (таких як перетягування блоків чи редагування тексту) ця архітектура може виявитися недостатньо гнучкою [16]. У таких випадках варто розглянути альтернативні підходи, такі як MVVM або компонентні архітектури [5].

MVVM (Model-View-ViewModel) – це сучасна архітектурна модель, яка набула широкого поширення завдяки фреймворкам Angular, Vue.js та React (у поєднанні з Redux) [2]. Вона організовує програму у три взаємопов'язані частини. Model відповідає за управління даними та бізнес-логікою, рис.1.6 [17].

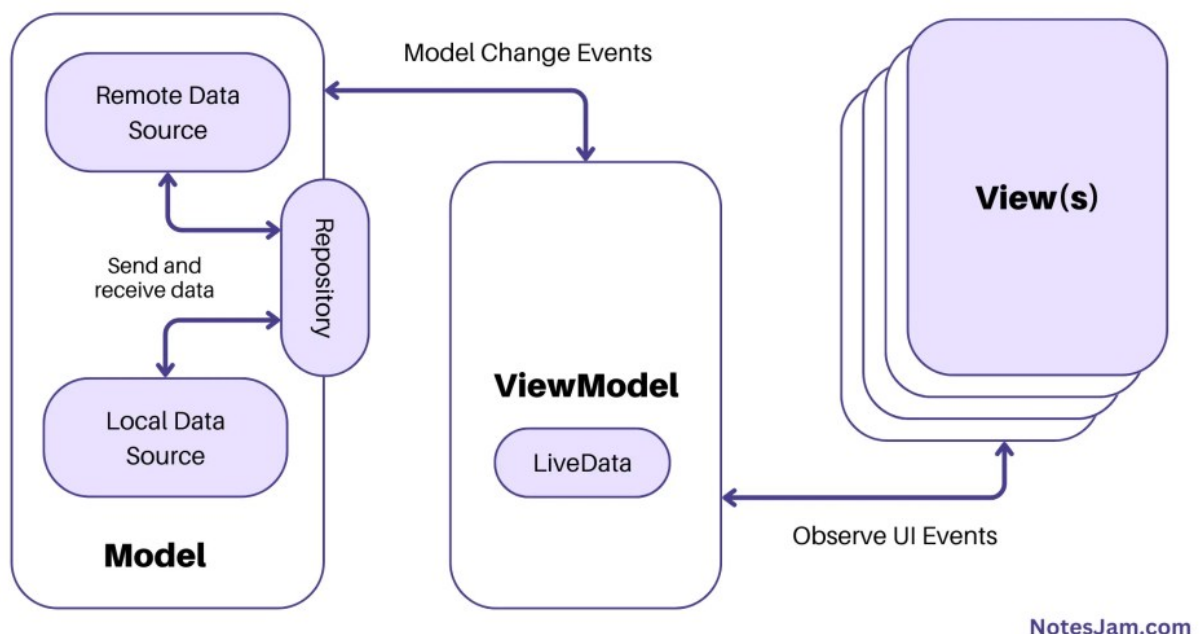


Рис. 1.6. Архітектура MVVM

View займається відображенням інтерфейсу користувача. ViewModel виступає проміжною ланкою, яка забезпечує двостороннє зв'язування даних між Model та View, автоматизуючи їх синхронізацію [20]. Головною перевагою MVVM є механізм двостороннього зв'язування даних, який суттєво спрощує розробку складних інтерактивних інтерфейсів. ViewModel дозволяє структуровано

розміщувати бізнес-логіку, не перевантажуючи при цьому шаблони відображення. Архітектура також сприяє легшому тестуванню завдяки чіткій ізоляції компонентів і підтримує принципи реактивного програмування, що критично важливо для динамічних веб-додатків [21]. Однак MVVM може виявитися надто складною для невеликих проектів через необхідність реалізації механізмів двостороннього зв'язування. Додаткова абстракція у вигляді ViewModel іноді створює надлишкову складність для простих завдань.

Для нашого проекту зі створення інтерактивних блок-схем MVVM є оптимальним вибором. Її здатність до автоматичної синхронізації даних між інтерфейсом і бізнес-логікою, а також підтримка реактивних оновлень ідеально відповідають вимогам до динамічного редагування візуальних елементів. Ця архітектура дозволить створити гнучкий та масштабований додаток із зручним інтерфейсом.

Flux – це архітектурний підхід, створений Facebook для керування потоком даних у веб-додатках [8]. Його основна особливість полягає в організації одностороннього потоку інформації, що забезпечує чітку структуру та передбачуваність роботи програми.

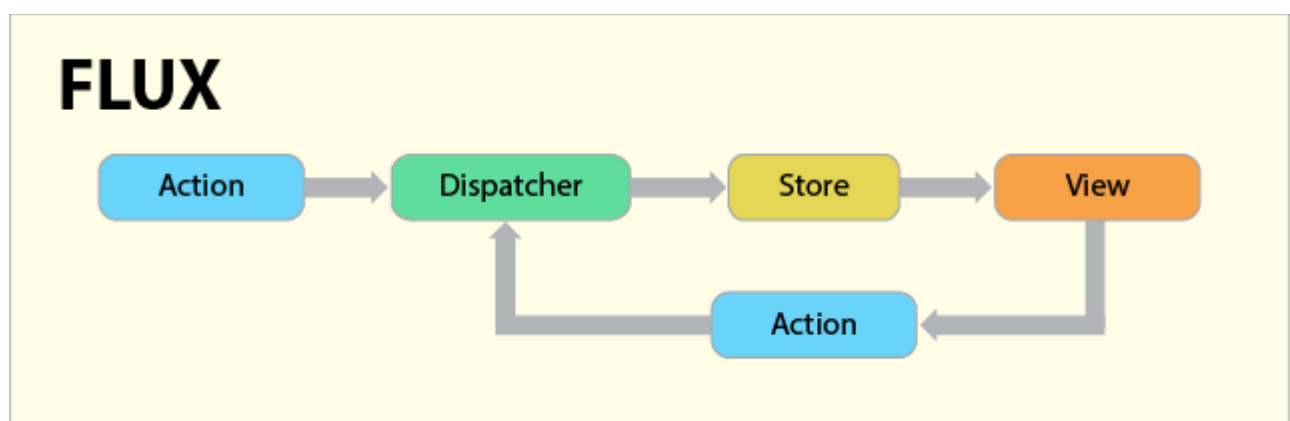


Рис. 1.7. Архітектура Flux

Архітектура складається з чотирьох ключових елементів, рис.1.7. Action представляє події, які виникають у результаті дій користувача. Dispatcher виступає

як центральний розподільник цих подій. Store відповідає за зберігання стану додатку та логіку його оновлення. View, у свою чергу, відображає дані з Store у вигляді інтерфейсу користувача [8]. Головною перевагою Flux є його односторонній потік даних, який значно підвищує передбачуваність поведінки додатку та спрощує відлагодження. Така архітектура добре підходить для складних програм з великою кількістю станів і компонентів, оскільки забезпечує чітку структуру і масштабованість.

Однак Flux має і свої недоліки. Він вимагає більше коду для реалізації порівняно з іншими архітектурами, що може бути надмірним для простих додатків. Додаткова складність у вигляді обов'язкового Dispatcher та строгих правил потоку даних не завжди виправдана для невеликих проектів.

Для нашого веб-додатку зі створення блок-схем Flux може виявитися занадто громіздким рішенням. Хоча він і забезпечує чітку структуру, його складність може перевищити необхідний рівень для нашого проекту, де важливі швидкість розробки та простота підтримки.

Мікросервісна архітектура організовує структуру додатку як набір незалежних сервісів, кожен з яких відповідає за певну функціональну частину системи, рис.1.8. Наприклад, окремі сервіси можуть обробляти авторизацію користувачів, зберігання даних або функціонал редагування схем [22]. Основна перевага такого підходу полягає у високій масштабованості та гнучкості системи [9]. Кожен сервіс може розвиватись і оновлюватись незалежно від інших, що дозволяє швидко адаптувати окремі компоненти до змінних вимог(див.). Крім того, така архітектура дозволяє оптимально розподіляти навантаження між різними серверами.

Однак мікросервісний підхід має і суттєві недоліки. Він значно ускладнює процес розробки та подальшої підтримки системи. Для його реалізації потрібні додаткові ресурси на організацію взаємодії між сервісами, моніторинг та управління розподіленою системою. Також зростають вимоги до інфраструктури та кваліфікації розробників.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

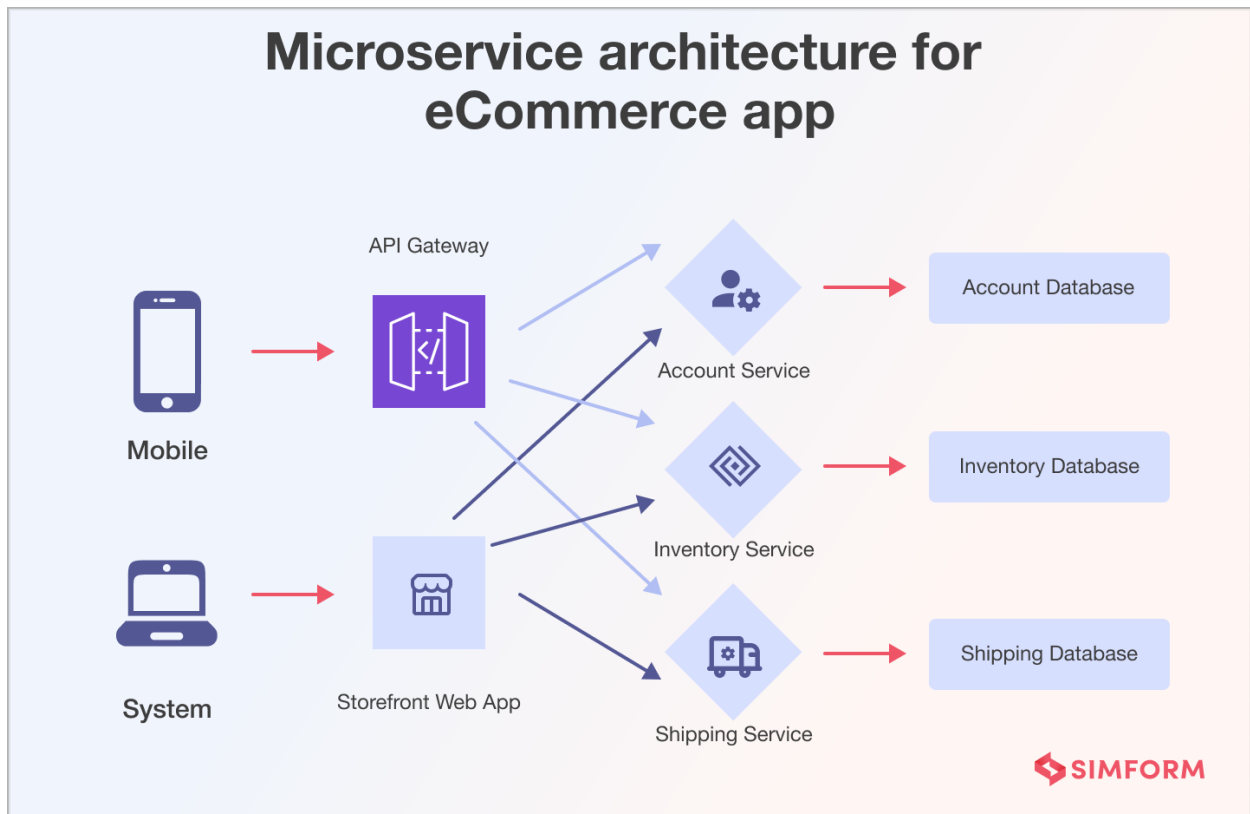


Рис. 1.8. Мікросервісна архітектура

Для нашого проекту зі створення веб-додатку для роботи з блок-схемами мікросервісна архітектура є надмірною. Оскільки наш додаток не передбачає складного розподіленого функціоналу та не потребує високого рівня масштабованості, використання монолітної або модульної архітектури буде більш доцільним рішенням.

На основі проведеного аналізу можна зробити висновок, що архітектура MVC (Model-View-Controller) є оптимальним рішенням для розробки додатку створення блок-схем UML [4]. Ця архітектура пропонує низку важливих переваг, які ідеально відповідають потребам нашого проекту. Головною перевагою MVC є чітке розділення обов'язків між трьома основними компонентами. Model відповідає за роботу з даними та бізнес-логікою, View займається відображенням інтерфейсу користувача, а Controller керує взаємодією між ними [23]. Така структура значно полегшує процес тестування, підтримки та подальшого розвитку програми.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Особливо важливим для нашого додатку є те, що Controller дозволяє централізовано керувати всіма користувацькими діями, такими як перетягування елементів, додавання нових блоків або збереження діаграм. Це суттєво спрощує розробку інтерактивного інтерфейсу та забезпечує логічну організацію коду. Архітектура MVC також сприяє ефективному повторному використанню коду. Наприклад, одна й та сама Model може працювати з різними типами інтерфейсів - веб-версією або мобільним додатком.

Для веб-додатків MVC є особливо зручною архітектурою, оскільки вона природним чином відповідає принципам роботи веб-застосунків [27]. Серверний Controller обробляє запити користувачів, Model забезпечує доступ до даних, а View генерує HTML-сторінки для відображення [1]. Використання сучасних фреймворків, таких як Django, Ruby on Rails або Express.js, дозволить максимально ефективно реалізувати архітектуру MVC у нашому проекті [11]. Ці інструменти надають готові рішення для організації коду за принципами MVC, що прискорює процес розробки. Таким чином, архітектура MVC є ідеальним вибором для нашого додатку зі створення блок-схем UML.

1.4. Інструментальні засоби реалізації програмного продукту та вимоги до апаратних засобів

Для успішної реалізації веб-додатку для створення блок-схем важливо обрати оптимальні інструментальні засоби, які забезпечать ефективну розробку, тестування та підтримку системи. У цьому розділі будуть описані обрані технології для клієнтської та серверної частин додатку, а також визначені вимоги до апаратних засобів. Вибір технологічного стеку ґрунтувався на таких критеріях: продуктивність, масштабованість, спільнота підтримки та відповідність специфічним вимогам проекту. Для клієнтської частини було обрано сучасні JavaScript-бібліотеки, тоді як серверна частина базується на Node.js екосистемі [21]. Окрему увагу приділено апаратним вимогам, оскільки вони безпосередньо

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

впливають на продуктивність як під час розробки, так і під час експлуатації готового рішення.

Для розробки клієнтської частини обрано сучасний технологічний стек. TypeScript, як основна мова програмування, додає статичну типізацію до JavaScript, що підвищує надійність коду та полегшує його підтримку [13].

В якості основної бібліотеки для побудови інтерфейсу використовується React. Його компонентний підхід дозволяє створювати модульні та повторно використовувані елементи інтерфейсу. Для здійснення HTTP-запитів до сервера застосовується бібліотека Axios, яка ідеально підходить для роботи з даними блок-схем UML, включаючи операції збереження, завантаження та синхронізації.

Специфічний функціонал для роботи з діаграмами реалізований за допомогою бібліотеки ReactFlow. Вона надає готові рішення для створення інтерактивних блок-схем, включаючи інструменти перетягування, масштабування та редагування елементів, а також підтримку з'єднань між вузлами та гнучке налаштування їх зовнішнього вигляду.

Серверна частина також розробляється на TypeScript, що забезпечує єдиність кодової бази [25]. Як основу сервера обрано легкий фреймворк Express для Node.js, який надає простий та ефективний спосіб обробки HTTP-запитів та маршрутизації [21]. Система безпеки реалізована за допомогою JWT для аутентифікації та авторизації користувачів. Для роботи з даними використовується зв'язка MongoDB (документоорієнтована NoSQL база даних) та Mongoose (ODM бібліотека), що забезпечує високу продуктивність, гнучкість та структурованість моделей даних [18].

Додаткові бібліотеки включають bcrypt для безпечного хешування паролів, zod для валідації вхідних даних та nodemailer для відправки електронних листів [19].

Апаратні вимоги. Для забезпечення оптимальної роботи додатку необхідно дотримуватись встановлених апаратних вимог. Для клієнтської частини рекомендовано використовувати комп'ютери з процесором від 1.5 GHz, мінімум 2

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

GB оперативної пам'яті та сучасні версії браузерів (Chrome, Firefox, Edge або Safari). Необхідна швидкість інтернет-з'єднання від 5 Mbps для комфортної роботи з інтерактивними елементами додатку. Ці вимоги враховують необхідність обробки графічних елементів блок-схем у реальному часі.

Серверні вимоги включають: процесор з 2 ядрами, 4 GB оперативної пам'яті, SSD диск від 20 GB, операційну систему Linux (Ubuntu 20.04 LTS або новішу) та PostgreSQL 13+ для роботи з даними [24]. Важливим є наявність стабільного інтернет-з'єднання для безперебійної взаємодії з клієнтами. Зазначені характеристики забезпечать ефективну обробку запитів, зберігання даних та підтримку одночасної роботи багатьох користувачів [9]. Для продуктивного середовища розробки рекомендується використовувати більш потужні конфігурації, що дозволить прискорити процес збірки та тестування.

Висновки до першого розділу

Проведено комплексний аналіз напрямків використання інформаційних технологій для розробки веб-додатку, призначеного для створення блок-схем. Детально визначено мету проєкту, його основні модулі та ключову функціональність, що включає інноваційні можливості для створення, редагування, збереження та спільної роботи над блок-схемами. Глибокий аналіз існуючих аналогів (Lucidchart, draw.io, Microsoft Visio, Creately) дозволив не лише виявити їх сильні та слабкі сторони, а й сформулювати унікальні конкурентні переваги нового продукту.

Особливу увагу приділено обґрунтуванню вибору архітектури MVC, яка забезпечує оптимальний баланс між простотою реалізації, чіткістю структури та можливістю подальшого масштабування. Технологічний стек (TypeScript, React, Express, MongoDB тощо) обрано з урахуванням сучасних тенденцій веб-розробки, продуктивності та специфічних вимог проєкту. Визначені апаратні вимоги гарантують стабільну роботу системи як на стороні клієнта, так і на сервері.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Важливим аспектом дослідження стало також визначення критеріїв зручності (UX) та безпеки користувачів під час роботи з додатком. Для цього проаналізовано сучасні стандарти інтуїтивного інтерфейсу, механізми автентифікації та шифрування даних, а також інтеграцію з хмарними сервісами для надійного зберігання проектів. Окремо розглянуто питання адаптивності системи до різних пристроїв (десктоп, планшети, смартфони), що є ключовим для забезпечення доступності та зручності.

Проведене дослідження створило міцну теоретичну базу для подальшої практичної реалізації проєкту, заклавши основи для створення інноваційного, конкурентоздатного продукту, що відповідає потребам сучасних користувачів та вимогам ринку. Отримані результати свідчать про доцільність обраного підходу та перспективність подальшого розвитку додатку.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ СТВОРЕННЯ UML БЛОК-СХЕМ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Для забезпечення ефективного функціонування веб-додатка для створення блок-схем наведено високорівневі вимоги, що враховують гнучкість, інтерактивність і підтримку співпраці між користувачами. *Бізнес-вимоги* [26]. Основні цілі веб-додатка включають надання зручного інструменту для створення блок-схем, адаптованого до потреб користувачів різного рівня, а також інтерактивні функції, такі як спільне редагування та коментування. Форма реалізації – це веб-додаток, що дозволяє створювати, редагувати, зберігати й експортувати блок-схеми з можливістю співпраці в реальному часі.

Вимоги користувачів. Реєстрація та авторизація передбачають можливість створення облікового запису, входу через email або соцмережі для збереження даних і спільної роботи. Створення блок-схем включає вибір шаблонів або створення з нуля, додавання різних типів елементів (процес, рішення, дані тощо), а також підтримку перетягування (drag-and-drop) для зручного розташування елементів. Редагування передбачає налаштування параметрів елементів (колір, текст, зв'язки), додавання коментарів до блок-схеми та спільне редагування з іншими користувачами. Функціонал експорту дозволяє зберігати блок-схеми у форматах PNG, JPG або JSON, а також отримувати посилання для поділу. Персоналізація включає вибір теми інтерфейсу (світла/темна), налаштування гарячих клавіш і панелі інструментів. Статистика передбачає відображення змін, внесених різними користувачами, історію змін із можливістю повернення до попередніх версій.

Вимоги адміністрації. Управління контентом охоплює створення та редагування шаблонів блок-схем, а також нових об'єктів. Модерація забезпечує дотримання правил спільної роботи, перегляд та видалення неприйнятних

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

контенту. Аналітика включає інструменти для оцінки активності користувачів. Сповіщення дозволяють надсилати користувачам повідомлення про оновлення.

Характеристика системи. Користувачі веб-додатка матимуть змогу гнучко налаштовувати блок-схеми під свої потреби, використовувати шаблони для швидкого старту, працювати в команді над однією блок-схемою, додавати коментарі та переглядати історію змін. Інтуїтивний інтерфейс із підтримкою перетягування та налаштування забезпечить зручність роботи.

Функціональні вимоги. Авторизація включає реєстрацію та вхід із можливістю вибору ролі (користувач, адміністратор) [28]. Створення блок-схем передбачає підтримку перетягування елементів, налаштування їх параметрів та додавання зв'язків між ними. Спільна робота реалізується через режим реального часу для командної роботи та синхронізацію змін між користувачами. Експорт дозволяє зберігати блок-схеми у популярних форматах. Історія зберігає зміни та надає можливість відновлення попередніх версій.

Нефункціональні вимоги. Сприйняття системи передбачає час завантаження блок-схеми – до 1 секунди, а також інтуїтивний багатомовний інтерфейс із підказками. Надійність забезпечується часом безвідмовної роботи – не менше 30 днів – та мінімальною кількістю помилок (максимум 1 на 10 тисяч рядків коду). Продуктивність додатка підтримує роботу щонайменше 500 одночасних користувачів. Масштабування дозволяє легко інтегрувати нові об'єкти і шаблони без впливу на продуктивність.

Враховуючи результати аналізу та сформовані вимоги до функціональності програмного продукту будуюмо діаграму використання веб-додатку для побудови блок-схем. Визначаємо, що у системі може бути три типи користувачів: гість, зареєстрований користувач та адміністратор, рис. 2.1-2.4.

Діаграми використання визначають дії користувачів у веб-додатку, що будуть реалізовані функціонально при програмній реалізації [30]. Розглянемо діаграми використання у розрізі кожного користувача, рис.2.2-рис.2.4.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

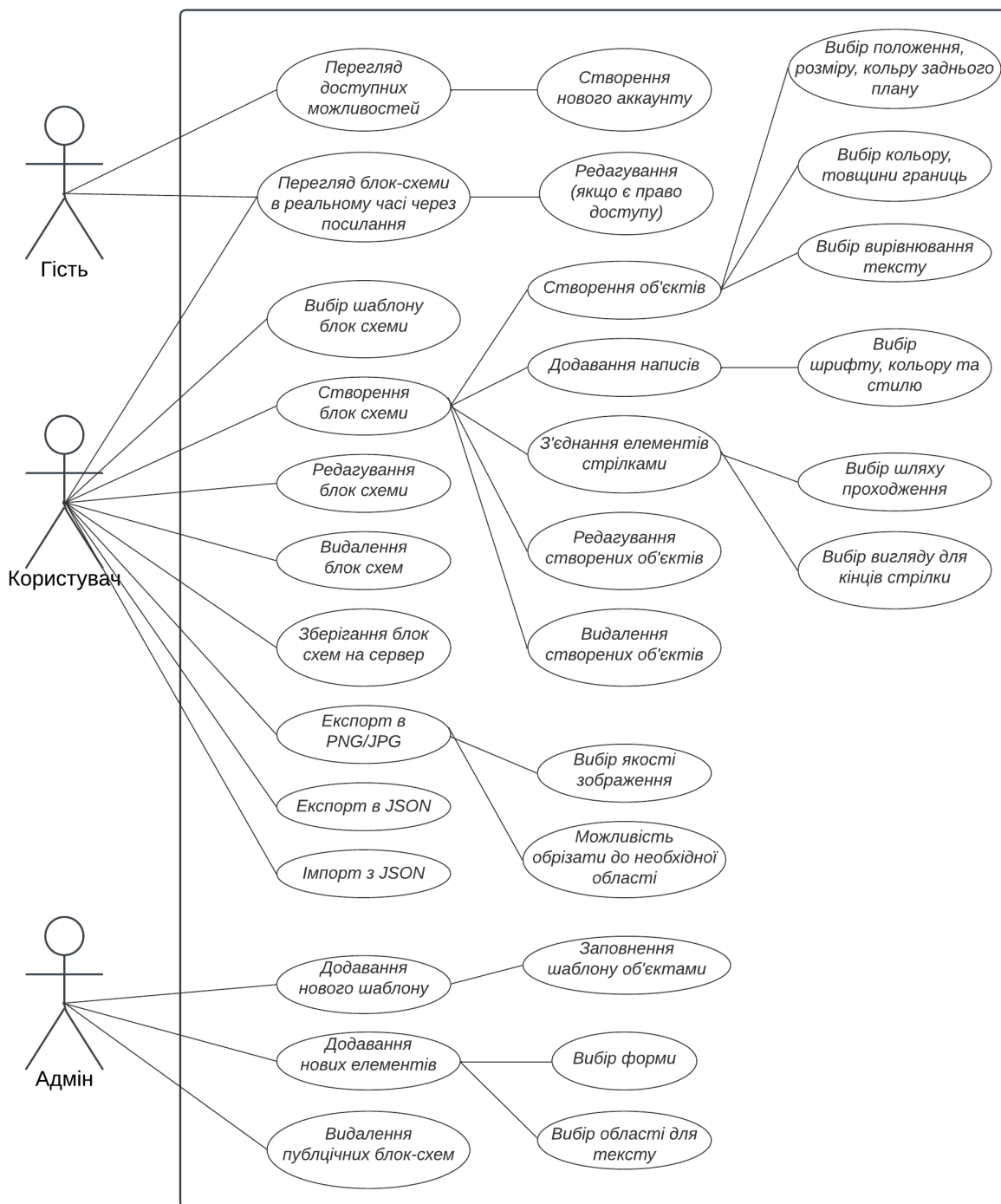


Рис.2.1. Узагальнена діаграма використання веб-додатку

Діаграма використання користувача «Гість», рис.2.2, відображає функціонал, доступний неавторизованим користувачам системи. Гість може переглядати загальні можливості сервісу, переглядати блок-схеми в реальному часі за

посиланням, створювати новий акаунт для отримання повного доступу до функцій системи.

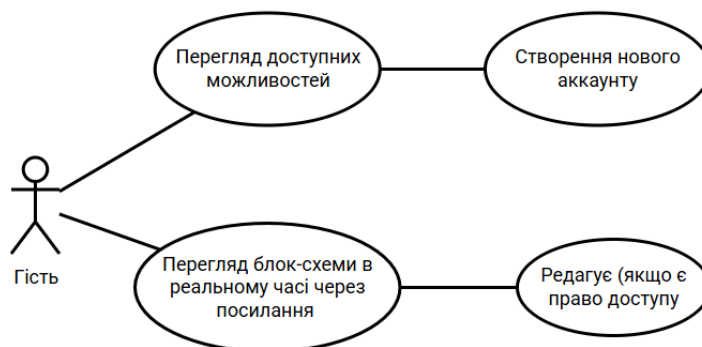


Рис. 2.2. Діаграма варіантів використання для користувача «Гість»

Ця схема відображає функціонал, доступний неавторизованим користувачам системи. Гість може переглядати загальні можливості сервісу, переглядати блок-схеми в реальному часі за посиланням та за наявності доступу, їх редагувати, а також створювати новий акаунт для отримання повного доступу до функцій системи.



Рис. 2.3. Діаграма варіантів використання користувача «Користувач»

Діаграма, рис.2.3, ілюструє основні можливості для зареєстрованих користувачів. Вона охоплює створення, редагування, збереження та видалення блок-схем, експорт та імпорт даних, а також гнучке налаштування вигляду елементів блок-схеми. Користувач має доступ до розширених інструментів для роботи з об'єктами, включаючи налаштування стилю, положення, кольору, тексту, стрілок та інших властивостей.

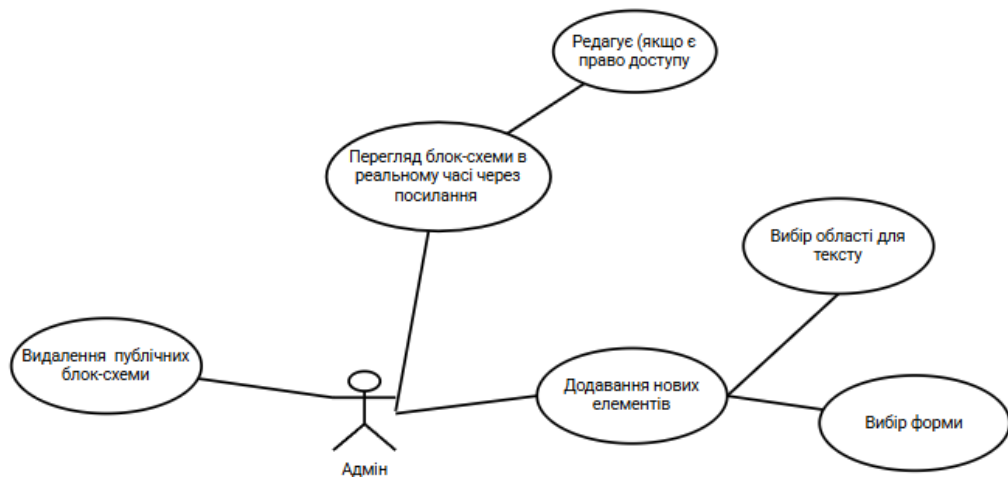


Рис. 2.4. Діаграма варіантів використання користувача «Адмін»

Діаграма варіантів використання користувача «Адмін» демонструє функції, доступні адміністраторам системи. «Адмін» може додавати нові шаблони та елементи для блок-схем, видаляти публічні блок-схеми, а також заповнювати шаблони об'єктами, обирати форму та область для тексту. Це дозволяє адміністратору підтримувати та розширювати функціонал сервісу для всіх користувачів.

Для розробки веб-додатку для створення блок-схем було спроектовано об'єктно-орієнтовану модель системи, рис.2.5, яка визначає залежності та взаємозв'язки між основними компонентами системи. У межах цієї моделі виділено ключові класи, що відображають логіку роботи додатку. Клас Guest представляє користувача без облікового запису, який має можливість переглядати публічно доступні діаграми. Зареєстрованим користувачам відповідає клас User, який надає функціонал для створення, редагування та видалення власних діаграм.

Адміністратор системи, представлений класом Admin, має розширені права, включаючи можливість створення шаблонів, додавання нових шаблонів та управління користувачами.

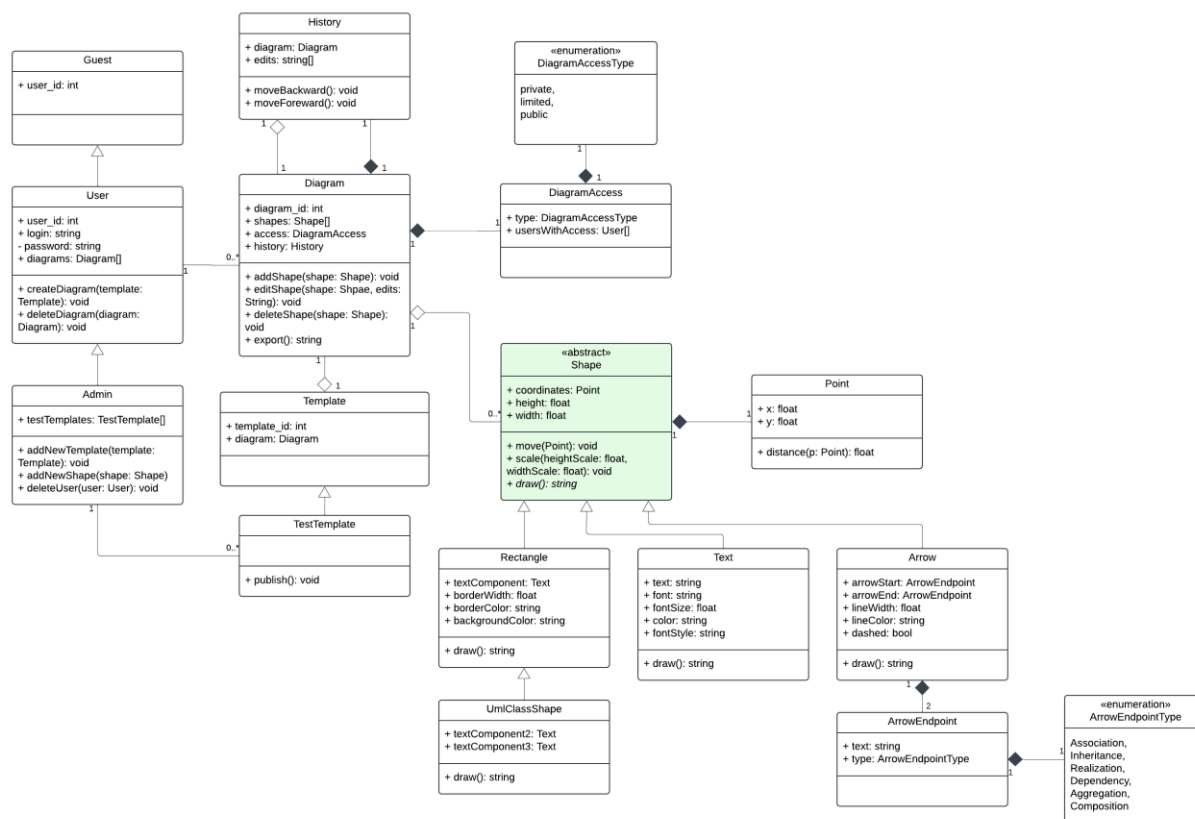


Рис. 2.5. Діаграма класів

Центральним елементом системи є клас Diagram, який інкапсулює основну логіку роботи з діаграмами. Він містить інформацію про шаблони, історію змін та параметри доступу до кожної конкретної діаграми. Абстрактний клас Shape визначає базові властивості всіх графічних елементів, такі як координати, розміри, а також функції масштабування та переміщення. Цей клас є батьківським для конкретних реалізацій графічних примітивів: Rectangle (прямокутник), Text (текстовий елемент), Arrow (стрілка) та UMLClassShape (елемент UML-діаграми). Клас Diagram зберігає унікальний ідентифікатор, набір фігур (shapes), доступ до діаграми, реалізований через об'єкт DiagramAccess, та історію змін, що

відслідковується за допомогою об'єкта History. Серед функцій, які реалізує цей клас, – додавання нової форми на діаграму, редагування вже існуючої, а також її видалення. Також є метод export, який дозволяє експортувати діаграму у відповідному форматі.

Система також включає клас Template, який відповідає за шаблони діаграм, що можуть використовуватись при створенні нових робочих областей. Для роботи з просторовими координатами призначений клас Point, який зберігає положення (x, y) усіх елементів на діаграмі. Він використовується для представлення координат на площині, а також дозволяє обчислювати відстань до іншої точки за допомогою методу distance. Окремий клас ArrowEndpoint відповідає за визначення типів кінцевих точок стрілок, використовується для визначення переліку типів взаємозв'язків (наприклад, асоціація, агрегація, композиція тощо). Клас TestTemplate успадковує Template і реалізує додаткову функцію publish.

У системі передбачено різні типи користувачів. Клас Guest репрезентує тимчасового або неавторизованого користувача, який має лише ідентифікатор user_id і не володіє жодним додатковим функціоналом. На відміну від нього, повноцінний користувач представлений класом User, який має унікальний ідентифікатор, логін, пароль, а також колекцію діаграм, які належать цьому користувачеві. Він може створювати нові діаграми на основі шаблонів за допомогою методу createDiagram, а також видаляти вже створені через метод deleteDiagram. Клас Admin є розширенням звичайного користувача. Окрім усього, що доступно класу User, адміністратор має змогу керувати шаблонами тестів, додавати нові шаблони або форми, а також видаляти інших користувачів із системи. Для цього клас надає методи addNewTemplate, addNewShape та deleteUser.

Клас History відповідає за зберігання історії редагувань. Він включає посилання на пов'язану діаграму та список редагувань. За допомогою методів moveBackward та moveForward користувач може перегортати попередні версії змін або повертатися до поточного стану, що дозволяє реалізувати функцію скасування та відновлення змін.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Доступ до діаграм організовано через клас `DiagramAccess`, який містить тип доступу (наприклад, приватний, публічний чи обмежений) та список користувачів, які мають право взаємодіяти з цією діаграмою. Тип доступу визначається через окрему перераховану сутність (enumeration) `DiagramAccessType`.

Форми або графічні елементи на діаграмі представлені абстрактним класом `Shape`, який включає координати, висоту, ширину та набір методів для переміщення, масштабування й візуалізації. Від цього класу успадковуються конкретні типи фігур, зокрема `Rectangle`, `Text`, `Arrow` та `UmlClassShape`.

Клас `Rectangle` відповідає за прямокутники з текстом і кольорами, клас `Text` – за окремі текстові елементи, а клас `Arrow` – за стрілки з початковими та кінцевими точками. Кожен з цих класів реалізує метод `draw`, що відповідає за відображення на екрані.

Всі описані класи та їх взаємозв'язки представлені на діаграмі класів (див. рис. 2.5), що дає повне уявлення про архітектуру та логіку взаємодії компонентів системи. Для кожного класу визначено публічні та приватні методи, що забезпечують взаємодію та модернізацію об'єктів.

Таким чином, дана діаграма класів демонструє повноцінну модель взаємодії користувачів із системою створення діаграм, включно з керуванням доступом, шаблонами, фігурами, редагуванням та історією змін. Усі класи пов'язані логічними відношеннями успадкування, асоціацій та агрегацій, що забезпечує гнучкість і масштабованість архітектури.

Для розробки проєкту «Веб-додаток для створення блок-схем UML» можна використати різні дизайн-патерни, щоб забезпечити зрозумілу архітектуру, спростити підтримку та розширення системи. Ось основні патерни, які підходять для цього проєкту: `Singleton`, `Factory Method`, `Proxy`, `Observer`.

Singleton. У веб-додатку необхідно забезпечити єдину точку доступу до бази даних, щоб уникнути створення зайвих з'єднань і забезпечити оптимальне використання ресурсів. Приклад діаграми класів зображено на рисунку 2.6.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

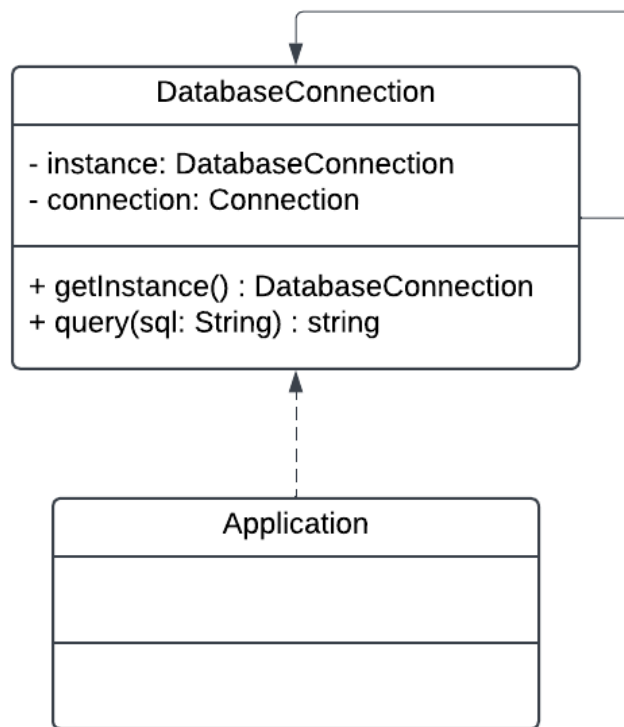


Рис. 2.6. Діаграма класів підключення до БД

У додатку різні типи елементів блок-схеми (наприклад, "Процес", "Рішення", "Дані") мають спільні базові риси, але відрізняються у реалізації. Factory Method дозволяє делегувати створення цих елементів окремим підкласам, спрощуючи масштабування та дотримуючись принципу відкритості/закритості (ОСР). Приклад діаграми класів різних типів елементів зображено на рисунку 2.7.

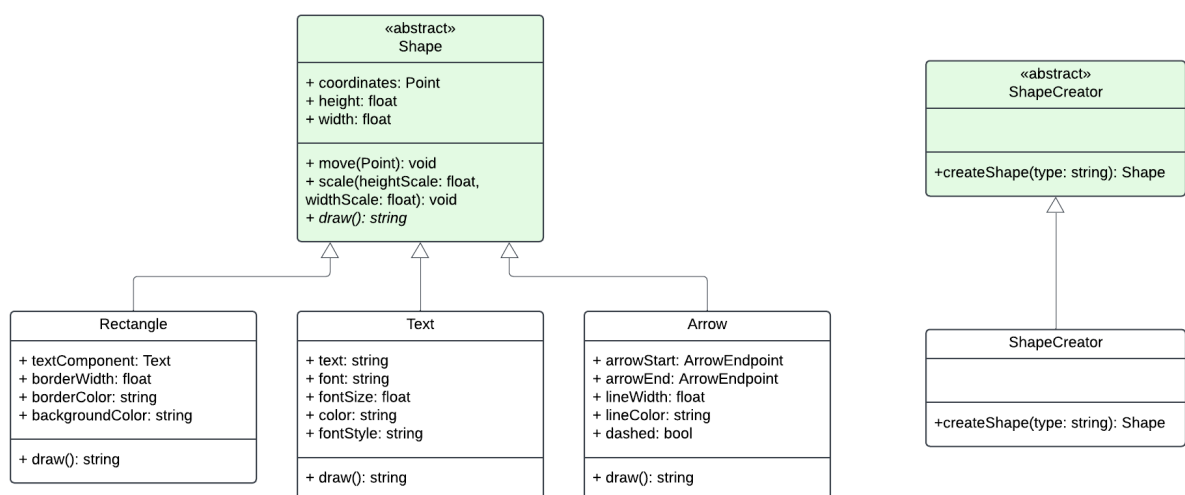


Рис. 2.7. Діаграма класів створення різних типів елементів

Щоб забезпечити ефективне збереження змін блок-схеми в історії, Proxy дозволяє контролювати доступ до об'єкта редагування. Він записує всі зміни в журнал, не порушуючи роботи основного об'єкта. Приклад діаграми класів для історії змін представлено на рисунку 2.8.

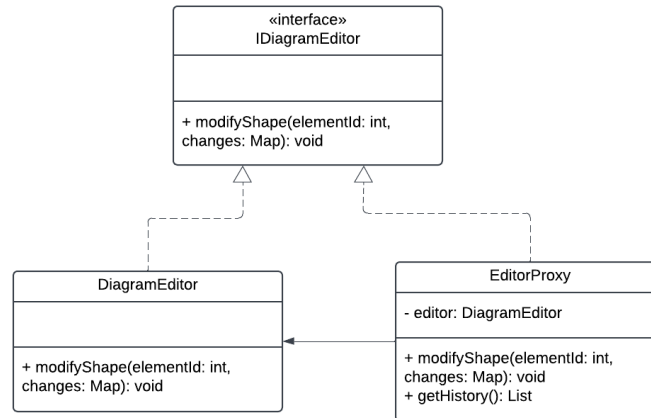


Рис. 2.8. Діаграма класів для історії змін блок-схеми

Приклад діаграми класів для синхронізації роботи за однією блок схемою зображено на рисунку 2.9:

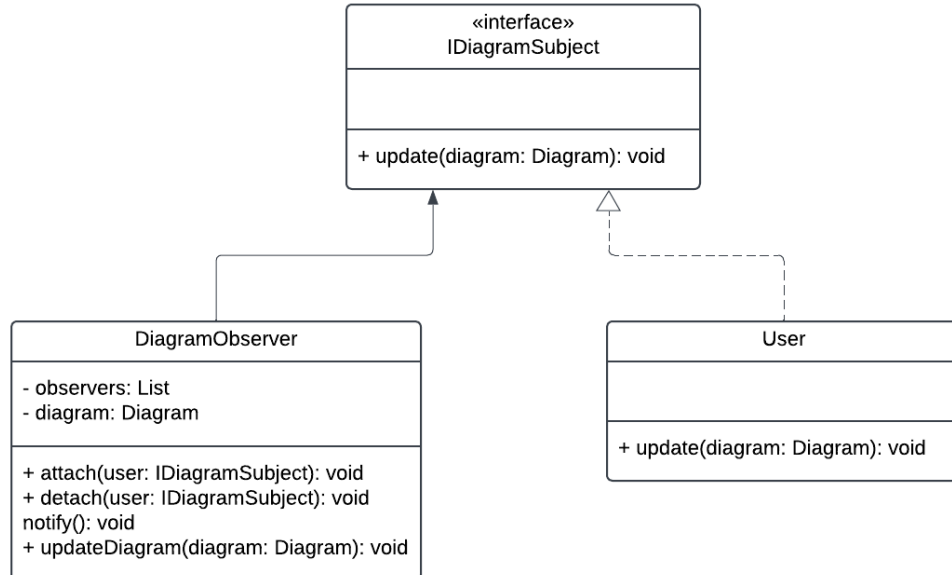


Рис. 2.9. Діаграма класів для синхронізації роботи з однією блок схемою

Для забезпечення синхронізації змін у блок-схемі між декількома користувачами (спільна робота), Observer дозволяє автоматично оновлювати клієнтські інтерфейси при внесенні змін на сервері.

- templates – таблиця шаблонів блок-схем;
- test_templates – таблиця тестових шаблонів блок-схем;
- history – таблиця історії змін блок-схем;
- diagram_access – таблиця доступу до блок-схем.

Далі наведено детальний опис полів кожної таблиці (див. табл. 2.1 – 2.10).

Опис полів таблиці «users» представлено в таблиці 2.1

Таблиця 2.1

Таблиця «users»

Назва	Тип даних	PK	FK	Опис
user_id	SERIAL	+		Ідентифікаційний номер користувача
login	VARCHAR(50)			Логін користувача
password	VARCHAR(255)			Хешований пароль користувача

Опис полів таблиці «diagrams» представлено в таблиці 2.2.

Таблиця 2.2

Таблиця «diagrams»

Назва	Тип даних	PK	FK	Опис
diagram_id	SERIAL	+		Ідентифікаційний номер блок-схеми
user_id	INT		+	Ідентифікаційний номер автора блок-схеми
name	VARCHAR(100)			Назва блок-схеми
access	ENUM('private', 'limited', 'public')			Тип доступу до блок-схеми

Опис полів таблиці «shapes» представлено в таблиці 2.3.

Таблиця «shapes»

Назва	Тип даних	PK	FK	Опис
shape_id	SERIAL	+		Ідентифікаційний номер фігури
diagram_id	INT		+	Ідентифікаційний номер блок-схеми
type	VARCHAR(50)			Тип фігури (прямокутник, текст, стрілка)
coordinates_x	FLOAT			Координата X верхнього лівого кута фігури
coordinates_y	FLOAT			Координата Y верхнього лівого кута фігури
height	FLOAT			Висота фігури
width	FLOAT			Ширина фігури

Опис полів таблиці «rectangles» представлено в таблиці 2.4.

Таблиця 2.4

Таблиця «rectangles»

Назва	Тип даних	PK	FK	Опис
shape_id	INT	+	+	Ідентифікаційний номер фігури
text_component	TEXT			Текст усередині прямокутника
border_width	FLOAT			Товщина межі
border_color	VARCHAR(50)			Колір межі
background_color	VARCHAR(50)			Колір фону

Опис полів таблиці «texts» представлено в таблиці 2.5.

Таблиця 2.5

Таблиця «texts»

Назва	Тип даних	PK	FK	Опис
shape_id	INT	+	+	Ідентифікаційний номер фігури
text_string	TEXT			Текстовий рядок
font	VARCHAR(50)			Шрифт тексту
font_size	FLOAT			Розмір шрифту
color	VARCHAR(50)			Колір тексту
font_style	VARCHAR(50)			Стиль шрифту (жирний, курсив тощо)

Опис полів таблиці «arrows» наведено у таблиці 2.6.

Таблиця 2.6

Таблиця «arrows»

Назва	Тип даних	PK	FK	Опис
shape_id	INT	+	+	Ідентифікаційний номер фігури
arrow_start_text	VARCHAR(50)			Текст на початку стрілки
arrow_end_text	VARCHAR(50)			Текст у кінці стрілки
line_width	FLOAT			Товщина лінії
line_color	VARCHAR(50)			Колір лінії
is_dashed	BOOLEAN			Показник пунктирної лінії (true/false)

Опис полів таблиці «templates» наведено у таблиці 2.7.

Таблиця «templates»

Назва	Тип даних	PK	FK	Опис
template_id	SERIAL	+		Ідентифікаційний номер шаблону
diagram_id	INT			Ідентифікаційний номер блок-схеми

Опис полів таблиці «test_templates» представлено у таблиці 2.8.

Таблиця 2.8

Таблиця «test_templates»

Назва	Тип даних	PK	FK	Опис
template_id	SERIAL		+	Ідентифікаційний номер шаблону

Опис полів таблиці «history» наведено у таблиці 2.9.

Таблиця 2.9

Таблиця «history»

Назва	Тип даних	PK	FK	Опис
history_id	SERIAL	+		Ідентифікаційний номер запису історії
diagram_id	INT		+	Ідентифікаційний номер блок-схеми
edits	TEXT			Опис змін у блок-схемі

Опис полів таблиці «diagram_access» представлено у таблиці 2.10. Розроблена база даних забезпечує зручне та ефективне зберігання всієї необхідної інформації для функціонування веб-додатку для створення блок-схем. Вона враховує взаємозв'язки між основними сутностями системи, такими як користувачі, блок-

схеми, фігури та шаблони. Такий підхід гарантує масштабованість, надійність та зручність підтримки проєкту у майбутньому.

Таблиця 2.10

Таблиця «diagram_access»

Назва	Тип даних	PK	FK	Опис
diagram_id	INT		+	Ідентифікаційний номер блок-схеми
user_id	INT		+	Ідентифікаційний номер користувача
access_type	ENUM('private', 'limited', 'public')			Тип доступу користувача до блок-схеми

У структурі бази даних передбачено окремі таблиці для кожного важливого об'єкта. Таблиця «users» зберігає інформацію про користувачів системи, включаючи унікальний ідентифікатор, логін та хешований пароль. Це дозволяє реалізувати механізм авторизації та розмежування прав доступу. Таблиця «diagrams» відповідає за зберігання метаданих кожної блок-схеми, зокрема її назви, типу доступу та зв'язку з автором. У свою чергу, таблиця «diagram_access» дозволяє визначати, які користувачі мають доступ до тієї чи іншої блок-схеми, і в якому режимі (приватний, обмежений або публічний). Це забезпечує гнучке налаштування спільної роботи над схемами.

Фігури, що входять до складу блок-схем, представлені в таблиці «shapes», яка містить координати, розміри, тип фігури та зв'язок із конкретною діаграмою. Усі фігури мають спільну базову структуру, а деталізація реалізується у відповідних підтаблицях: «rectangles», «texts» і «arrows». Таблиця «rectangles» розширює базову фігуру параметрами, що стосуються прямокутника: текстом, кольорами меж і фону, а також товщиною ліній. Таблиця «texts» включає інформацію про текстові елементи, такі як шрифт, розмір, колір і стиль. У свою чергу, таблиця «arrows»

описує стрілки між елементами схеми, вказуючи їхнє оформлення, товщину, колір, пунктирність та текстові мітки на початку й в кінці.

Для забезпечення повторного використання структур існує таблиця «templates», що зберігає шаблони діаграм, пов'язані з конкретними схемами. Таблиця «test_templates» дозволяє відокремити шаблони, які мають спеціальний статус для тестування або навчання. Такий розподіл шаблонів полегшує адміністрування та створення нових діаграм на основі існуючих рішень. Ще одним важливим компонентом є таблиця «history», яка дозволяє зберігати історію змін кожної блок-схеми. Вона містить опис редагувань і пов'язується з відповідною діаграмою. Це дає змогу реалізувати функціонал скасування змін та повернення до попередніх версій.

Завдяки логічному розділенню даних по таблицях, чітким зв'язкам між ними та використанню первинних і зовнішніх ключів, створена база даних є надійною основою для розробки стабільного та зручного у користуванні веб-додатку. Вона дозволяє ефективно організовувати роботу з блок-схемами, забезпечуючи при цьому контроль доступу, збереження структури елементів і можливість ведення історії змін.

2.3 Проєктування та реалізація алгоритмів роботи системи

У цьому пункті описуємо алгоритм функціонування програмної системи для створення блок-схем, її стани та взаємодія користувача з різними компонентами системи. Процес створення блок-схеми користувачем починається з відкриття сторінки блок-схем, рис. де доступний інтерфейс для взаємодії з уже створеними схемами або створення нових. Користувач натискає кнопку «Створити нову блок-схему», після чого відкривається інструментарій для конструювання власної схеми. На цьому етапі він може додавати нові об'єкти на робоче поле, обираючи їх розташування, розмір, колір фону, а також налаштовуючи товщину та колір меж. Крім того, користувач має змогу з'єднувати об'єкти за допомогою стрілок,

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

визначаючи шлях проходження стрілки та зовнішній вигляд її початкової та кінцевої точок. Діаграма активності для цього процесу зображена на рисунку 2.11.

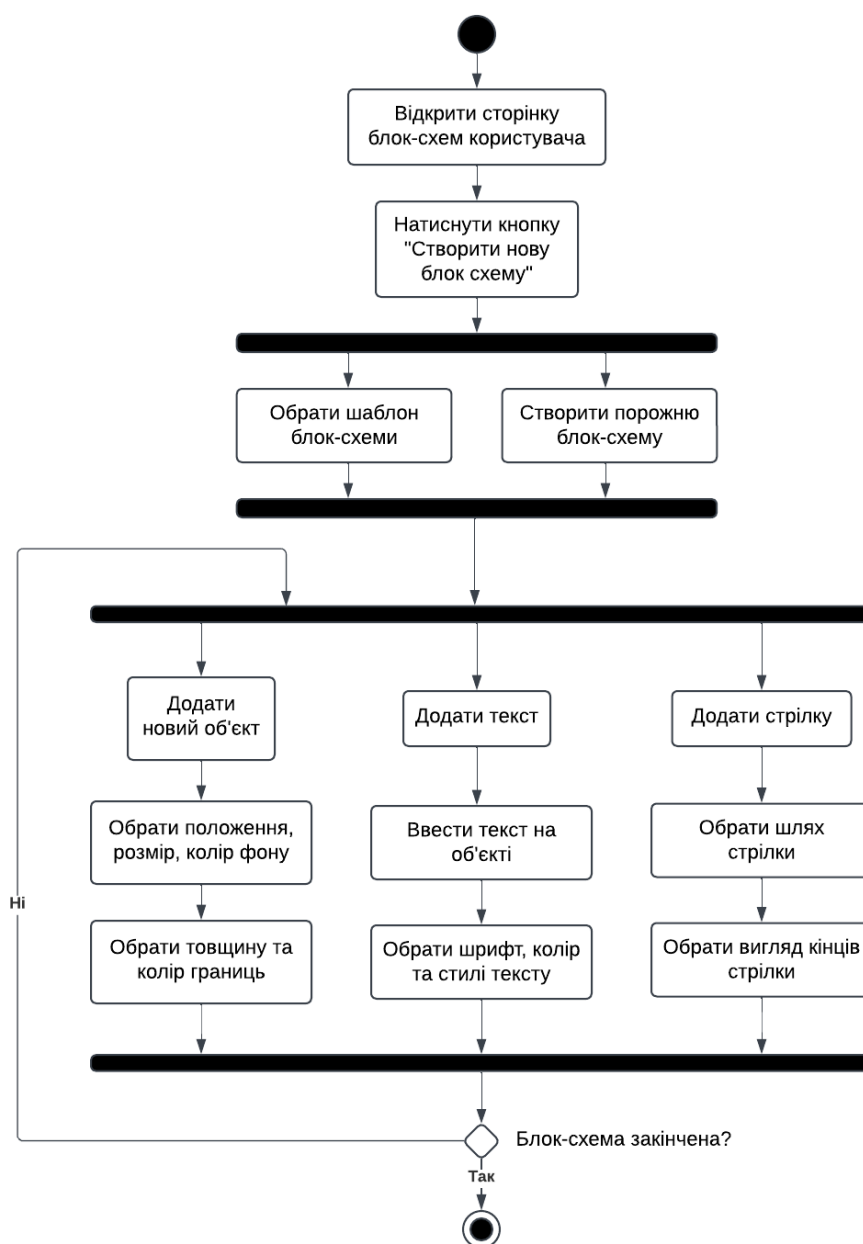


Рис. 2.11. Діаграма активності для створення користувачем блок-схеми

Для публікації нового об'єкта адміністратор спочатку проходить авторизацію у своєму обліковому записі та відкриває панель адміністратора, рис.2.12. Далі він переходить до створення нового об'єкта, що передбачає намалювання його форми з вибором стандартного розміру, кольору фону, товщини та кольору меж. Після цього адміністратор додає до об'єкта текстове поле, в якому задається стандартний

текст, а також обираються шрифт, колір і стиль тексту. Завершивши редагування, адміністратор має змогу опублікувати створений об'єкт. Діаграма активності для цього процесу зображена на рисунку 2.12.

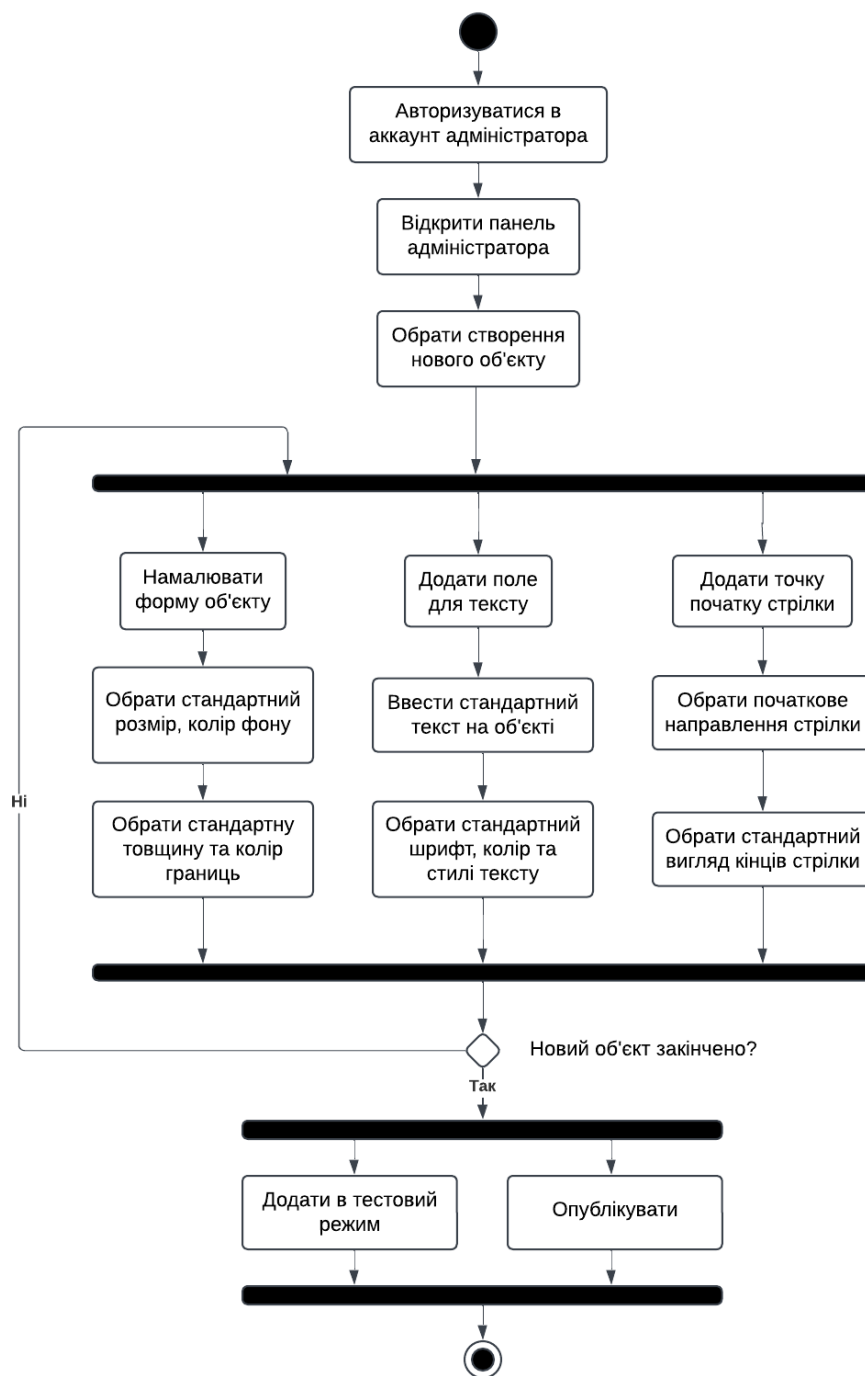


Рис. 2.12. Діаграма активності для публікації нового об'єкту

Експорт блок-схеми починається з того, що користувач обирає необхідну схему на відповідній сторінці (див. рис.2.13). Після цього він натискає кнопку «Експорт», і система пропонує варіанти доступних форматів, зокрема JPG або PNG.

Обравши бажаний формат і налаштувавши відповідні параметри, користувач отримує можливість завантажити файл на свій пристрій. Весь процес експорту відображено на рисунку 2.13.

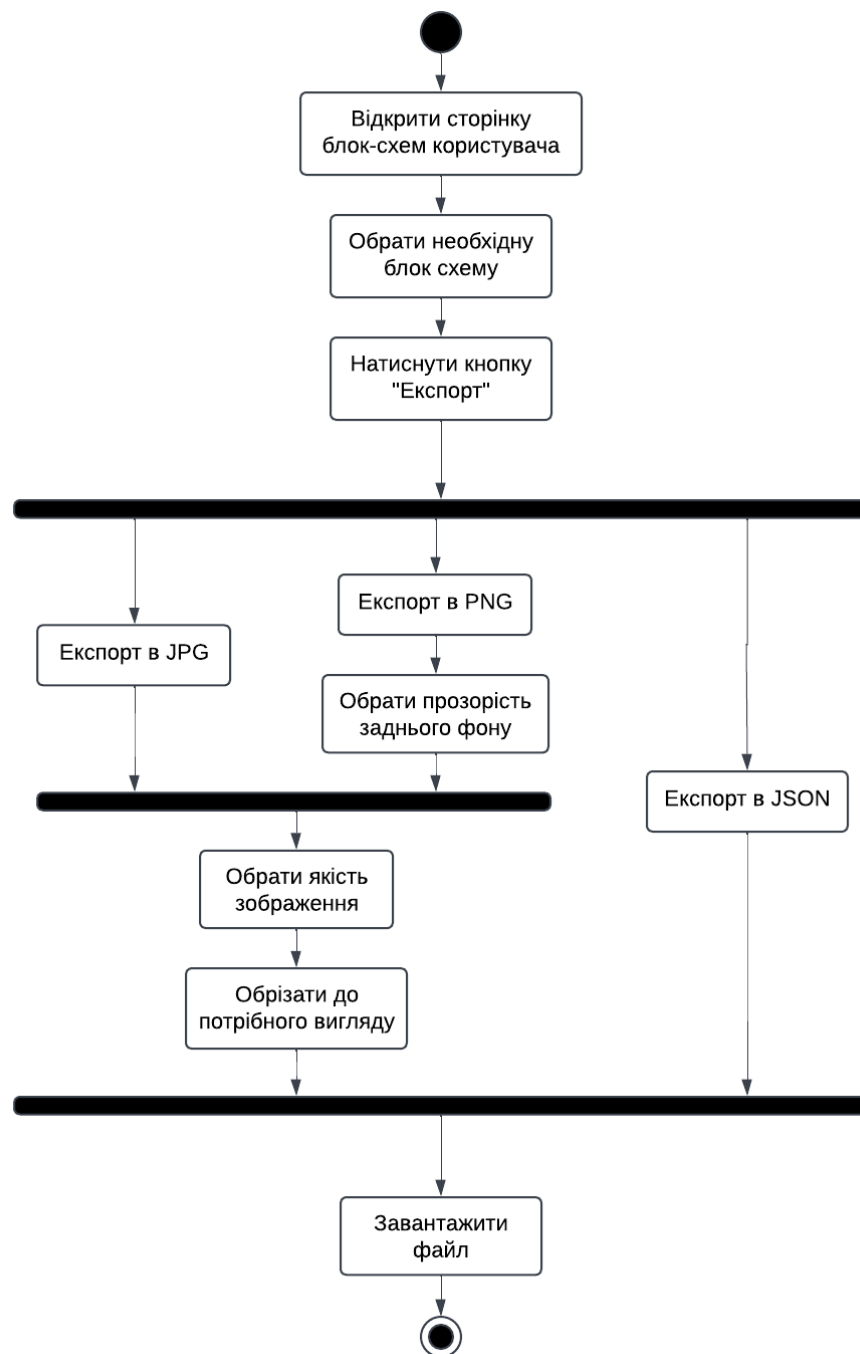


Рис. 2.13. Діаграма активності для експорту блок-схеми

Описані алгоритми відображають основні етапи роботи програмної системи та демонструють її можливості у створенні, публікації та експорті блок-схем.

2.4 Реалізація функціоналу додатка

У цьому розділі детально описується реалізація основних модулів та функціональності веб-додатку для створення блок-схем. Кожен модуль був розроблений з урахуванням архітектури MVC, що забезпечує чітке розділення обов'язків між компонентами. Основні етапи реалізації включають налаштування серверної частини, розробку клієнтської частини та інтеграцію ключових функцій, таких як авторизація, створення та редагування блок-схем, спільна робота та експорт даних.

Серверна частина додатку була розроблена з використанням TypeScript та Express.js. Основним завданням сервера є обробка HTTP-запитів від клієнта, зберігання даних у базі даних MongoDB та забезпечення безпечної взаємодії користувачів через JWT-токени. Код ініціалізації сервера представлено на рис. 2.14.

```
1 import dotenv from 'dotenv';
2 import app from './app';
3 import connectDB from './db';
4
5 dotenv.config();
6
7 const start = (port: number) : void => {
8   try {
9     app.listen(port, callback: () : void => {
10       console.log(`Server listening on ${port}`);
11     });
12     connectDB();
13   } catch (err) {
14     console.error(err);
15     process.exit();
16   }
17 };
18
19 start(parseInt( string: process.env.PORT || '5000', radix: 10));
```

Рис. 2.14. Код ініціалізації серверу

Підключення до бази даних відбувається за допомогою фрагменту коду, що представлено на рис.2.15.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

```

import mongoose, { ConnectOptions } from 'mongoose';

1+ usages
const connectDB = async () : Promise<void> => {
  try {
    await mongoose.connect( uri: process.env.MONGO_URI || '', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    } as ConnectOptions);

    console.log('MongoDB connected!');
  } catch (error) {
    console.error(error);
    process.exit( code: 1);
  }
};

1+ usages
export default connectDB;

```

Рис. 2.15. Код підключення до бази даних

Реалізація аутентифікації: у цьому блоці коду демонструється, як генеруються JWT-токени для авторизованих користувачів, а також як виконується перевірка токенів при кожному запиті. Повний код аутентифікації наведено у додатку А. Процедура обробки запитів продемонстровано на рис. 2.16.

```

74   }
75   });
76
77   router.put( path:('/:diagramId/privacy', withAuth, async (req : Request<ParamsDictionary, any,... , res : Response<any, Record<string,
78     try {
79       await setDiagramPrivacy(req.params.diagramId, req.body.isPublic);
80       res.status( code: 200).json( body: { message: 'OK' });
81     } catch (e) {
82       res.status( code: 404).json( body: { message: getErrorMessage(e) });
83       console.log(getErrorMessage(e));
84     }
85   });
86
87   router.delete( path:('/:diagramId', withAuth, async (req : Request<ParamsDictionary, any,... , res : Response<any, Record<string, a... )
88     try {
89       await deleteDiagram(req.params.diagramId);
90       res.status( code: 200).json( body: { message: 'OK' });
91     } catch (e) {
92       res.status( code: 404).json( body: { message: 'Could not delete diagram' });
93       console.log(getErrorMessage(e));
94     }

```

Рис. 2.16. Код додавання та видалення блок-схем

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Клієнтська частина додатку була реалізована з використанням TypeScript, React та бібліотеки ReactFlow для створення інтерактивних блок-схем. Основним завданням клієнта є надання користувачам зручного інтерфейсу для створення, редагування та перегляду блок-схем.

У фрагменті коду, (див. рис. 2.17) показано як ініціалізується React-додаток.

```

1  > import ...
18
    1+ usages
19  function MainApp() : JSX.Element {
20      axios.defaults.baseURL = import.meta.env.VITE_API_URL;
21
22      return (
23          <ThemeProvider theme={mainTheme}>
24              <AlertProvider>
25                  <ToastAlert />
26                  <Routes>
27                      <Route path="/" element={<LandingPage />} />
28                      <Route path="/login" element={<AuthLogin />} />
29                      <Route path="/signup" element={<AuthRegister />} />
30                      <Route path="/verify" element={<AuthVerification />} />
31                      <Route path="/send-reset-password" element={<AuthSendReset />} />
32                      <Route path="/reset-password" element={<AuthResetPassword />} />
33                      <Route element={<PrivateRoute />}>
34                          <Route path="/dashboard" element={<DashboardPage />} />
35                          <Route path="/:diagramId/edit" element={<DiagramEditor />} />
36                          <Route path="/:diagramId/view" element={<DiagramViewer />} />
37                      </Route>
38                      <Route path="*" element={<PageNotFound />} />
39                  </Routes>
40              </AlertProvider>
41          </ThemeProvider>
42      );
43  }

```

Рис. 2.17. Код ініціалізації React-додатку

Фрагмент інтеграції ReactFlow наведено на рис.2.18. Повний код інтеграції ReactFlow надано у додатку Б.

```

53   return (
54     <div className="diagram">
55       <ReactFlow
56         fitView
57         nodes={entities}
58         nodeTypes={nodeTypes}
59         onNodesChange={onNodesChange}
60         onNodeDragStop={onNodeDragStop}
61         onNodesDelete={onNodesDelete}
62         onConnect={onConnect}
63         edges={relationships}
64         edgeTypes={edgeTypes}
65         onEdgesChange={onEdgesChange}
66         onEdgeUpdateStart={onEdgeUpdateStart}
67         onEdgeUpdate={onEdgeUpdate}
68         onEdgeUpdateEnd={onEdgeUpdateEnd}
69         onEdgesDelete={onEdgeDelete}
70         connectionMode={ConnectionMode.Loose}
71       >
72         <Background color="#444" variant={'dots' as BackgroundVariant} />
73         <Controls style={{ flexDirection: 'column', gap: 0, margin: '3em' }}>
74           <ControlButton onClick={handleDownload} title="Download Diagram">

```

Рис. 2.18. Код ReactFlow елементу

У цьому блоці коду демонструється, як виконуються HTTP-запити до сервера за допомогою Axios для отримання даних про блок-схеми (див. рис. 2.19).

```

26   useEffect( effect: () : void => {
27     1+ usages
28     const getDiagrams = async () : Promise<void> => {
29       try {
30         const res : AxiosResponse<any, any> = await axios.get( url: '/api/diagram' );
31         res.data.sort(
32           (a: any, b: any) =>
33             new Date(b.modified).getTime() - new Date(a.modified).getTime()
34         );
35         setDiagrams(res.data);
36         setLoading( value: false );
37       } catch (err) {
38         showAlert( message: 'Could not get diagrams. Please reload.', AlertType.ERROR );
39       }
40     };
41     getDiagrams();
42     }, deps: [setAlert]);

```

Рис. 2.19. Код отримання списку блок-схем

Створення та редагування блок-схем відіграє ключову роль у роботі додатку. На рисунку 2.20 представлено фрагмент коду для створення нового блоку.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

1+ usages
57 function ClassNode({ id, data }: NodeProps<Klass>) {
58   const [editOpen : boolean , setEditOpen : React.Dispatch<React.SetStateAction<boolean>>] = useState( initialState: false);
59   const entitiesDispatch : React.Dispatch<EntityAction> = useEntitiesDispatch();
60
61   const { setAlert } = useAlert();
62   const { diagramId : string | undefined } = useParams();
63
64   1+ usages
65   const handleDelete = async () : Promise<void> => {
66     try {
67       await axios.delete( url: `/api/entity/${id}?diagramId=${diagramId}`);
68       entitiesDispatch( value: { type: 'DELETE_ENTITY', id });
69       setAlert( message: 'Class successfully deleted', AlertType.SUCCESS);
70     } catch (e) {
71       setAlert( message: 'Could not delete class. Try again', AlertType.ERROR);
72     }
73   };
74
75   return (
76     <>
77     <NodeToolBarCustom
78       setEditOpen={setEditOpen}
79       handleDelete={handleDelete}
80     />
81     <div onClick={() => setEditOpen( value: true)}>

```

Рис. 2.20. Код створення нового блоку

Фрагмент коду експорту діаграми представлено на рисунку 2.21.

```

const viewport : HTMLInputElement = document.querySelector(
  selectors: '.react-flow__viewport'
) as HTMLInputElement;
if (viewport) {
  const dataUrl : string = await toPng(viewport, options: {
    backgroundColor: '#faf9f6',
    width: imageWidth,
    height: imageHeight,
    style: {
      width: String(imageWidth),
      height: String(imageHeight),
      transform: `translate(${transform[0]}px, ${transform[1]}px) scale(${transform[2]})`,
    },
  });
  const now : Date = new Date();
  const link : HTMLAnchorElement = document.createElement( tagName: 'a' );
  link.download = `diagram${diagramId}-${now.toLocaleDateString()}.png`;
  link.href = dataUrl;
  link.click();
  setAlert( message: 'Diagram successfully downloaded', AlertType.SUCCESS);
}
};

```

Рис. 2.21. Код експорту діаграми

Таким чином представлено програмну реалізацію ключових модулів додатку для реалізації спроектованого функціоналу.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Висновок до другого розділу

Було спроектовано архітектуру веб-додатку для створення UML блок-схем, включаючи варіанти використання, об'єктно-орієнтовану модель системи, базу даних і ключові алгоритми. Визначено функціональні та нефункціональні вимоги, класи користувачів і логіку взаємодії з елементами діаграм. Окрім цього, обґрунтовано вибір патернів проектування для забезпечення гнучкості, масштабованості та підтримки спільної роботи, що формує надійне підґрунтя для реалізації системи.

Побудовано базу даних, яка охоплює всі основні сутності системи – від користувачів і діаграм до фігур, шаблонів і історії редагувань. Завдяки чіткому структуруванню таблиць і налагодженим зв'язкам між ними вдалося забезпечити ефективне збереження даних, контроль доступу та можливість відстеження змін. Це дозволяє не лише надійно реалізувати поточний функціонал, а й легко розширювати його в майбутньому.

Розроблені основні сценарії використання, діаграми активності та діаграми класів, що наочно демонструють логіку роботи системи. Такий підхід дозволяє сформувати чітке бачення структури додатку ще до етапу реалізації та зменшує ризики проектних помилок. Усі ці елементи у сукупності створюють міцну основу для подальшої розробки, тестування та впровадження веб-додатку

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1. Інтерфейс програмного продукту

Враховуючи прийняті проєктні рішення та функціональні вимоги до програмної реалізації веб-додатку для створення блок-схем UML було проведено розробку прототипу додатку.

Враховуючи наявність трьох груп користувачів, форма входу в додаток має вигляд, що представлено на рис.3.1.

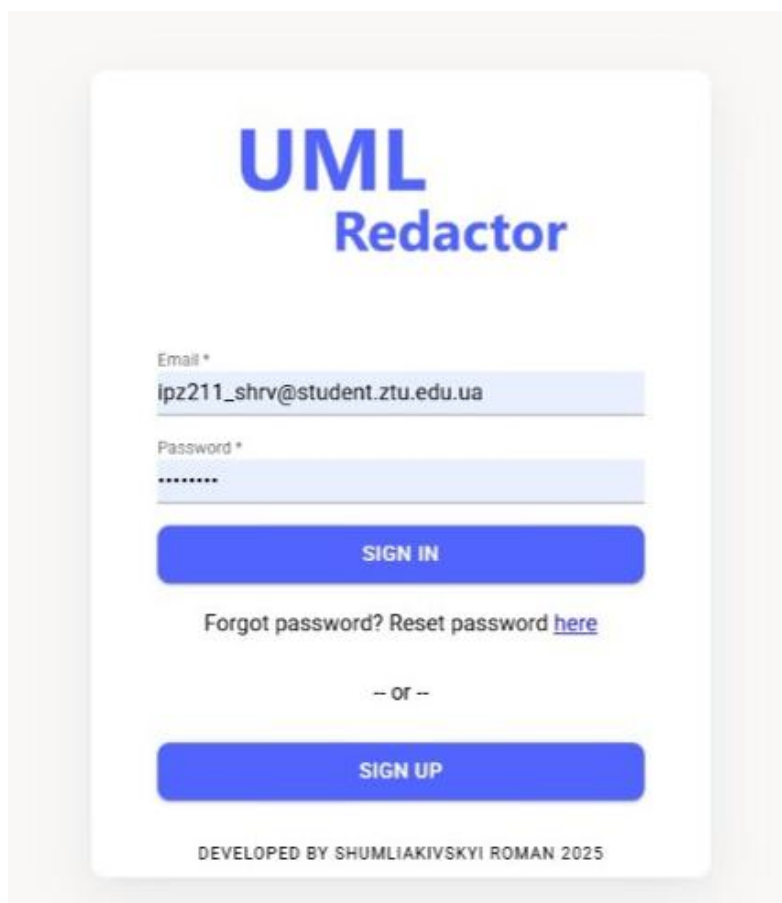
The image shows a web application interface for 'UML Redactor'. At the top, the title 'UML Redactor' is displayed in a large, blue, sans-serif font. Below the title, there are two input fields: 'Email *' containing the text 'ipz211_shrv@student.ztu.edu.ua' and 'Password *' containing a series of dots. Below these fields is a blue button labeled 'SIGN IN'. Under the button, there is a link that says 'Forgot password? Reset password [here](#)'. Below this link is a separator consisting of a horizontal line with 'or' in the middle. At the bottom of the form is another blue button labeled 'SIGN UP'. At the very bottom of the interface, in small capital letters, it says 'DEVELOPED BY SHUMLIKIVSKYI ROMAN 2025'.

Рис.3.1. Вигляд вікна авторизації користувачів додатку

Вхід в систему відбувається відповідно заповнених даних при реєстрації та здійснюється через електронну пошту та введення паролю. Доступна процедура відновлення паролю та перехід на реєстраційну форму.

Реєстрація користувача здійснюється за рахунок заповнення форми, а саме введення електронної адреси в поле «Email», імені користувача «Username»,

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

задання пароллю в полі «Password» та його повторення в полі «Re-enter Password»
Вигляд реєстраційної форми представлено на рисунку 3.2.

Рис.3.2. Вигляд форми для реєстрації користувача

Після успішної процедури реєстрації користувачу завантажується робочий інтерфейс додатку, де він може будувати блок-схеми та проводити інші дії відповідно до реалізованого функціоналу.

У додатку передбачене вікно «My Diagrams», де зазначені діаграми з якими користувач вже працював, відповідно він може їх переглянути та повернутися до роботи з будь якою, або ж створити нову, натиснувши на відповідну іконку «+New» (див. рис.3.3.). На даному вікні зазначається дата останньої зміни вказаної діаграми.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		54



Рис.3.3 Вигляд інтерфейсу вікна «My Diagrams»

Робоче вікно для створення та редагування блок-схем представлена на рис.3.4.

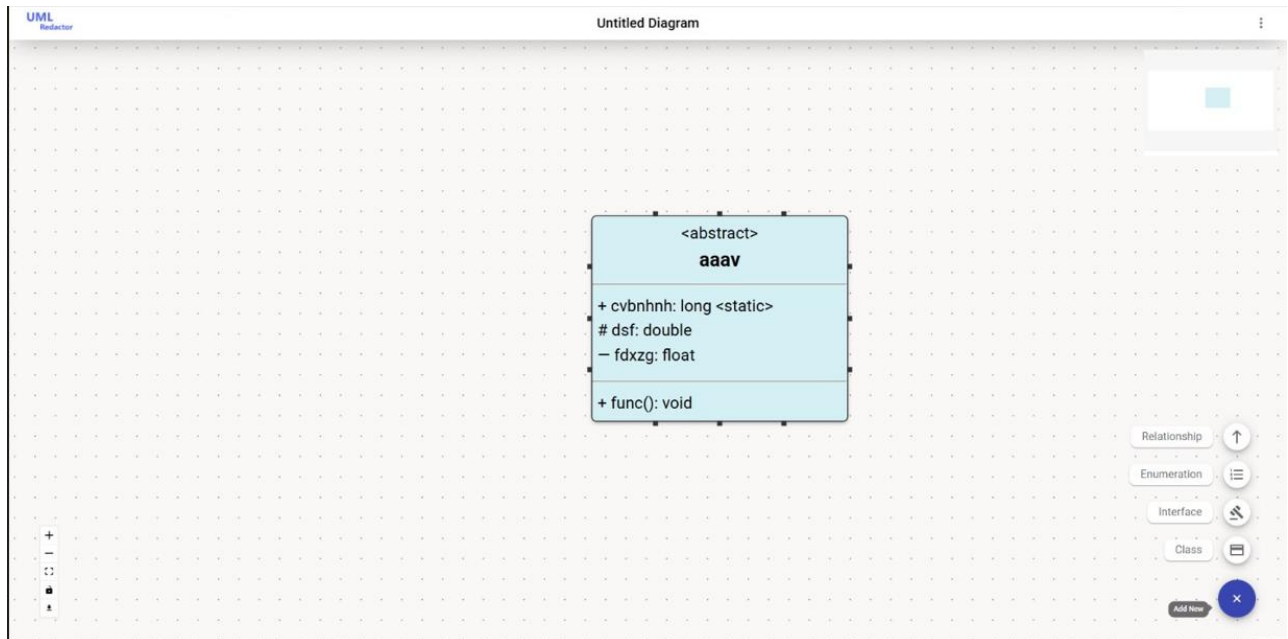


Рис.3.4. Вигляд робочого вікна зареєстрованого користувача

Для створення блоку ми можемо скористатися панеллю зліва, рис. 3.5. Зокрема вибрати тип створюваного блоку (клас, інтерфейс, енумерація), провести його налаштування відповідно до задач та потреб встановити зв'язки.

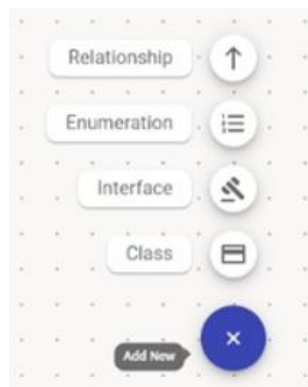


Рис.3.5. Вигляд панелі інструментів

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Праворуч у нас знаходиться навігаційна панель, рис.3.6. Використовуючи її ми можемо збільшувати/зменшувати формат відображення діаграми, вибирати потрібні об'єкти та копіювати їх, блокувати редагування або налаштовувати доступ для спільного користування, скачувати діаграму.



Рис. 3.6 Панель навігації та налаштування

При натисненні на «Class» (див. рис.3.5), відобразиться вікно для налаштування (див. рис.3.7) , зокрема поле введення імені «Class Name», позначення абстрактності класу («Abstract»), а також наповненість класу (константи, атрибути, методи).

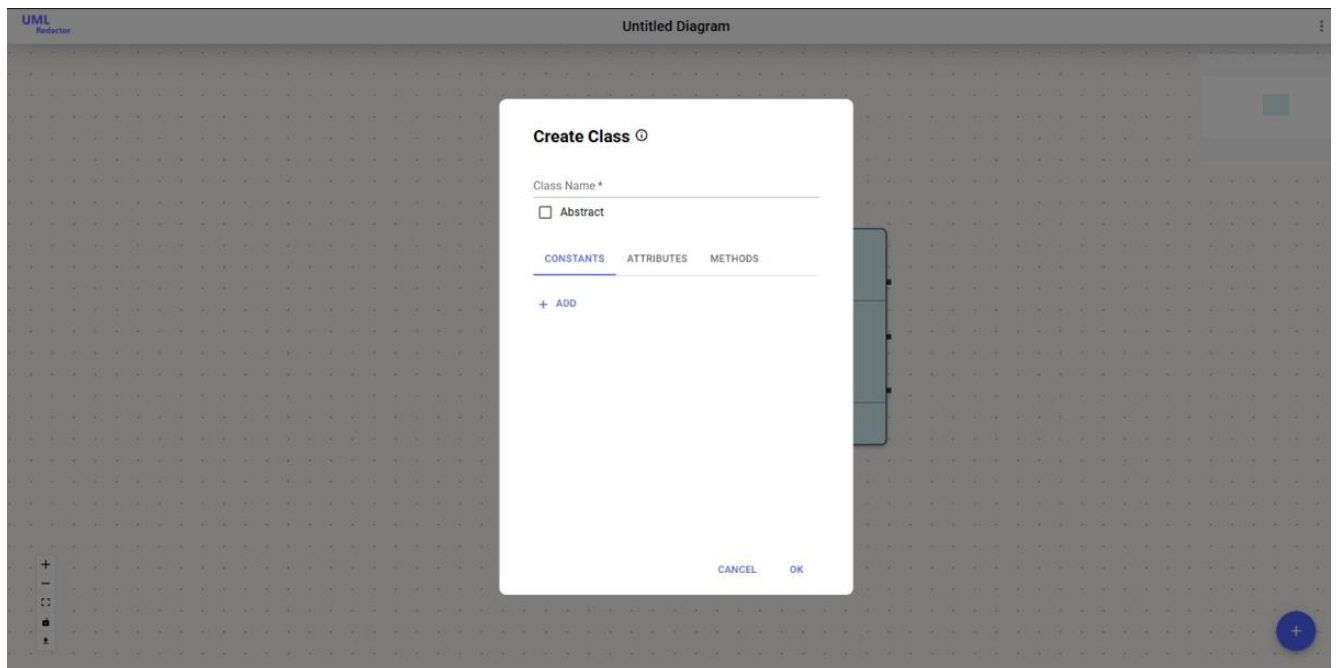


Рис. 3.7. Вигляд вікна при створенні класу

Додавання атрибутів класу представлено на рис. 3.8.

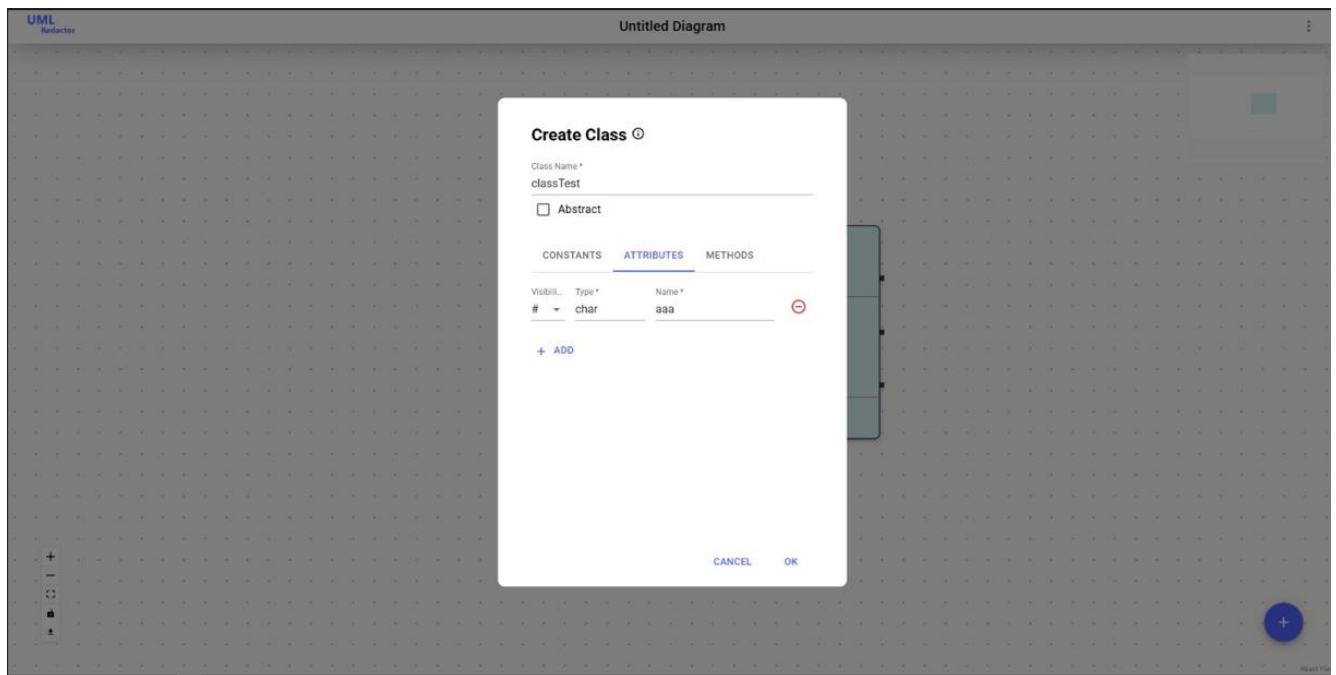


Рис.3.8. Вигляд вікна для додавання атрибутів класу

Вигляд вікна для додавання методів, що будуть використовуватися представлено на рисунку 3.9.

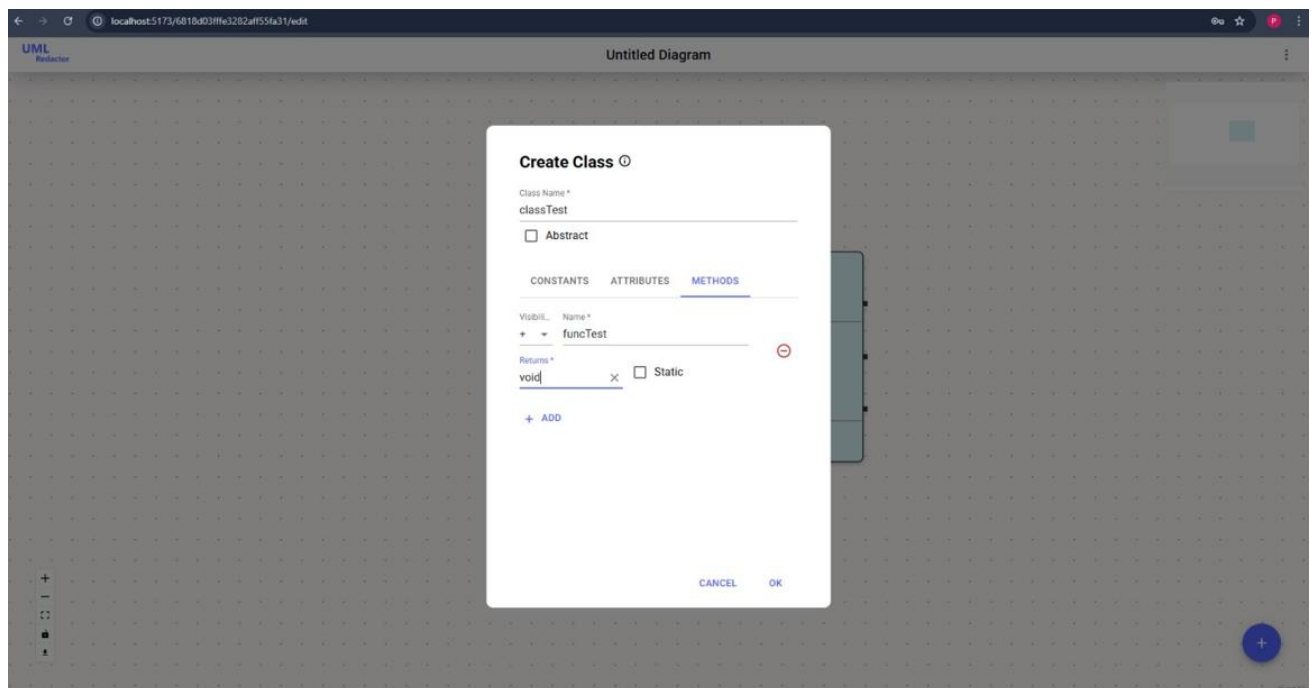


Рис. 3.9. Вигляд вікна для додавання методів класу

В результаті проведених дій було створено клас «class Test», що представлено на рис.3.10. Як видно з рисунку, при виборі блоку, зверху з'являється панель для редагування та видалення у разі необхідності.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

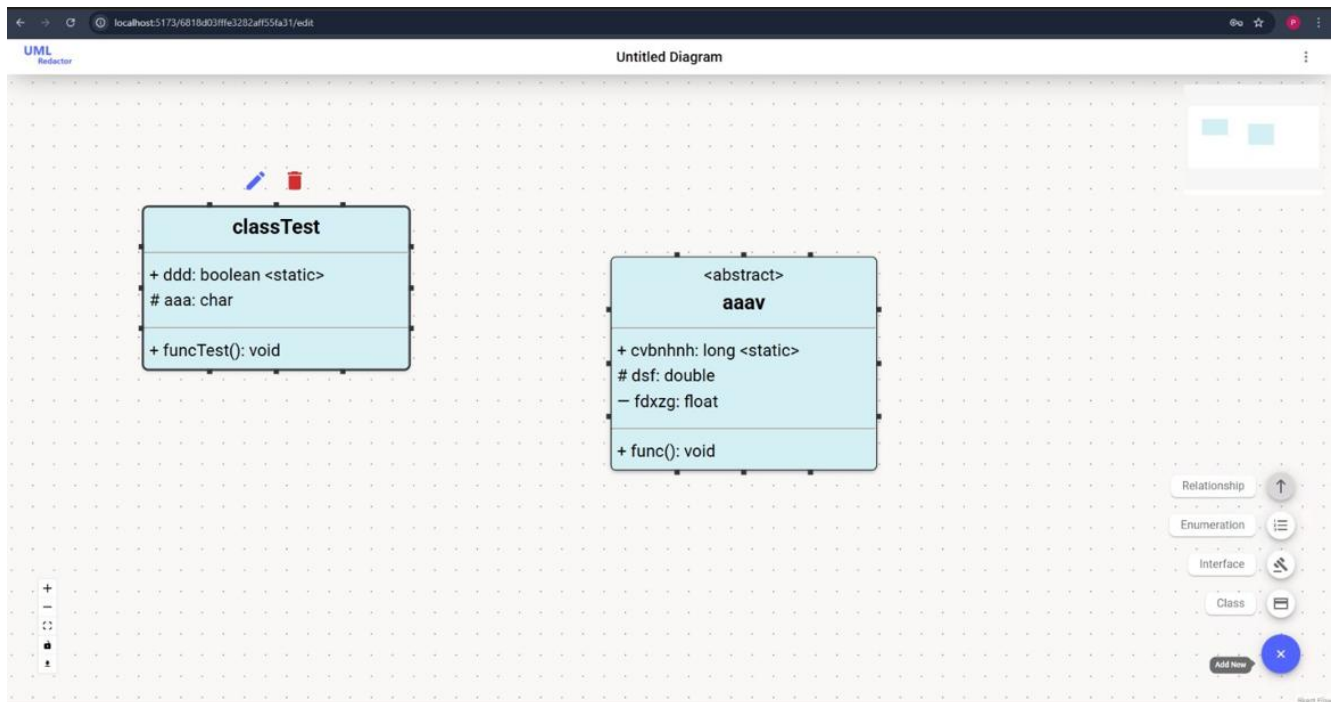


Рис. 3.10. Вигляд вікна зі створеним класом

Важливим елементом даного веб-додатку є можливість встановлення зв'язків з можливістю налаштування їх типу, що доволі важливо для подальшої програмної реалізації системи (додатку), що буде проєктуватися за допомогою даного додатку. Для встановлення зв'язків використовуємо «Relationship» з лівої панелі інструментів (див. рис. 3.4 та рис.3.5). Після вибору даного інструменту висвітиться вікно, що представлене на рис. 3.11.

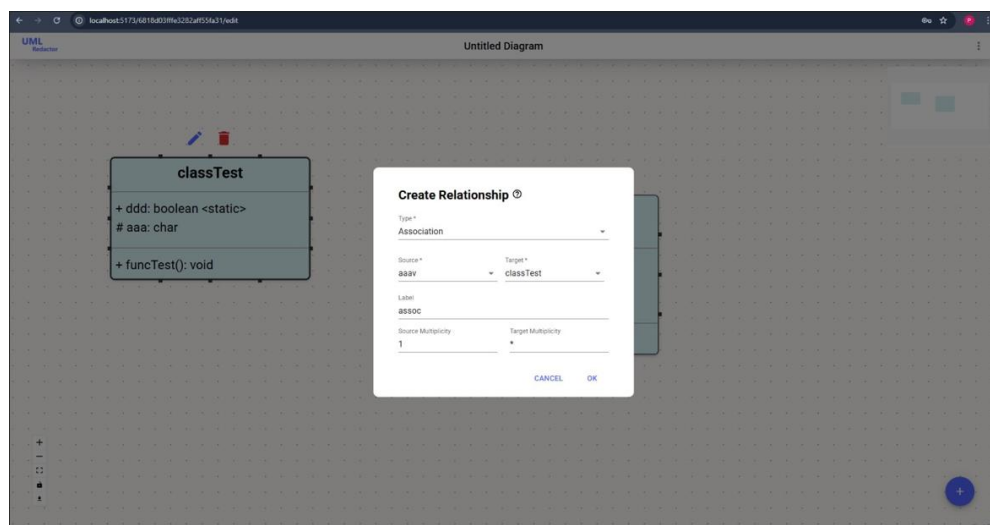


Рис.3.11. Вигляд вікна для встановлення типів зв'язку

Як видно з рисунку 3.11 доступний вибір блоків для з'єднання, зазначення назви зв'язку та встановлення відношення зв'язку. Після заповнення всіх полів та підтвердження операції внизу вікна, отримуємо відображення встановленого зв'язку на діаграмі (див. рис. 3.12).

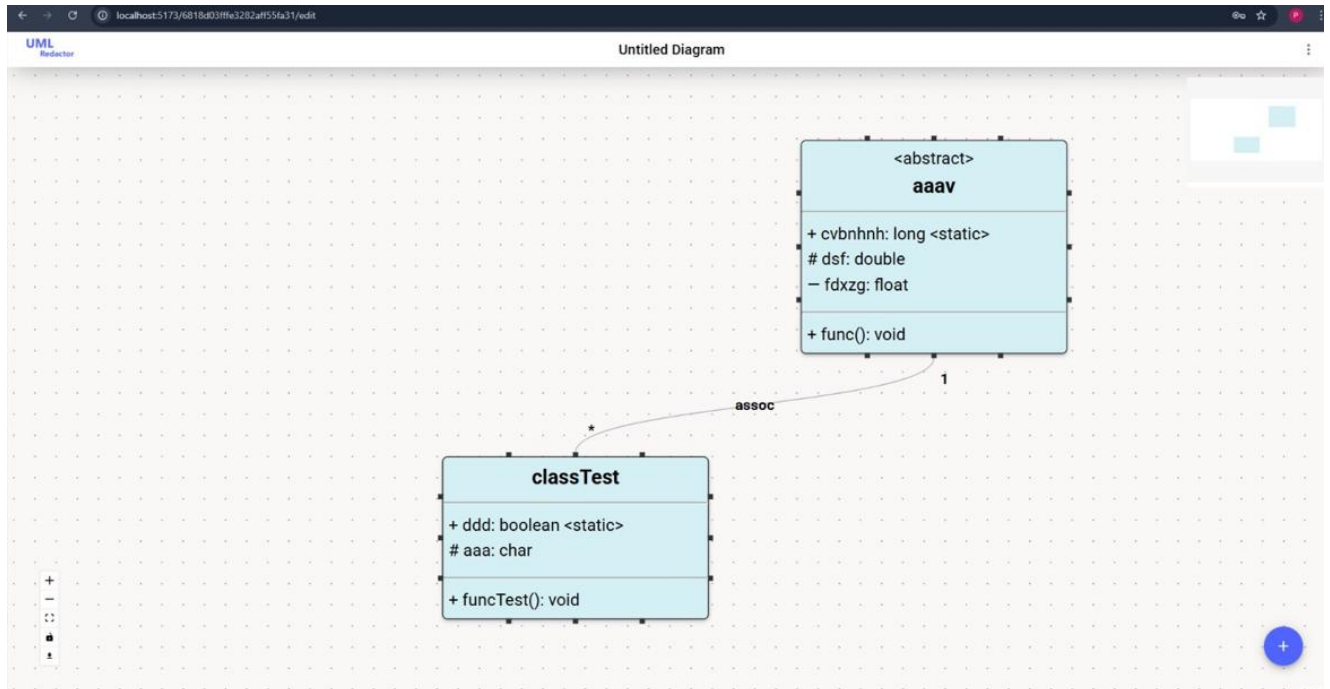


Рис.3.12. Вигляд блок-схеми зі встановленим зв'язком між блоками

Аналогічним чином проводиться робота із іншими наявними інструментами. Проведемо тестування розробленого прототипу веб-додатку та перевіримо його функціональність.

3.2. Порядок розгортання клієнт-серверного додатку

Процес налаштування та запуску веб-додатку передбачає послідовне виконання критичних етапів, починаючи з підготовки середовища виконання. На першому етапі необхідно встановити залежності для обох компонент системи: виконати команду `npm install` у кореневих директоріях серверної та клієнтської частин окремо, що забезпечить наявність усіх необхідних бібліотек та модулів Node.js.

Конфігурація середовища вимагає створення та заповнення файлу `.env` у корені проекту. Ключовим аспектом є налаштування параметрів для сервісу Nodemailer: для коректної роботи відправки електронних листів обов'язково потрібно активувати двофакторну аутентифікацію в обліковому записі електронної пошти (наприклад, Gmail), після чого згенерувати спеціальний «Пароль додатку» (App Password). Цей пароль разом з адресою електронної пошти мають бути додані до змінних середовища у файлі `.env` у форматі `EMAIL_USER` та `EMAIL_PASSWORD`.

Паралельно необхідно інтегрувати сховище даних MinIO: розгорнути інстанс сервісу (локально через Docker або на хмарній платформі), ініціалізувати кошик (bucket) з відповідною назвою (наприклад, `app-data`) та забезпечити доступ через криптографічні ключі. Параметри підключення до MinIO, включно з кінцевою точкою («`MINIO_ENDPOINT`»), публічним ключем («`MINIO_ACCESS_KEY`») та секретним ключем («`MINIO_SECRET_KEY`»), також додаються до файлу `.env`. Запуск системи здійснюється шляхом послідовної ініціалізації компонентів: у директорії сервера виконується команда `npm run start`, що активує серверний API, а в директорії клієнта – `npm run dev` для старту фронтенд-дodatка.

Після запуску необхідно провести валідацію роботи системи: проаналізувати логи на відсутність критичних помилок, протестувати ключові функції (зокрема, відправку листів через Nodemailer). Важливими технічними аспектами є забезпечення безпеки: усі конфіденційні параметри (ключа MinIO, паролі додатків) мають зберігатися виключно в `.env`, який виключається з системи контролю версій через `.gitignore`.

Для підвищення стабільності в продакшн-середовищі рекомендується використання процес-менеджерів (наприклад, PM2) та зворотного проксі (Nginx). Типові проблеми під час запуску часто пов'язані з недостатніми правами доступу до MinIO, невірними SMTP-налаштуваннями або відсутністю залежностей, що вимагає повторного виконання `npm install`. Зазначимо, що перший запуск може вимагати подвійної ініціалізації для активації всіх залежностей.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

3.3. Тестування веб-додатку для побудови блок-схем UML

Після виконання інструкцій щодо розгортання системи, проведемо тестування веб-додатку побудови блок-схем UML. В пункті 3.1 було частково описано інтерфейсну частину. Проведемо доповнення та перевіримо функціональність розроблених інструментів. Для створення блоку, що є інтерфейсом скористаємося лівою панеллю інструментів (див. рис.3.5) та виберемо «Interface». Відкриється вікно, вигляд якого представлено на рисунку 3.13.

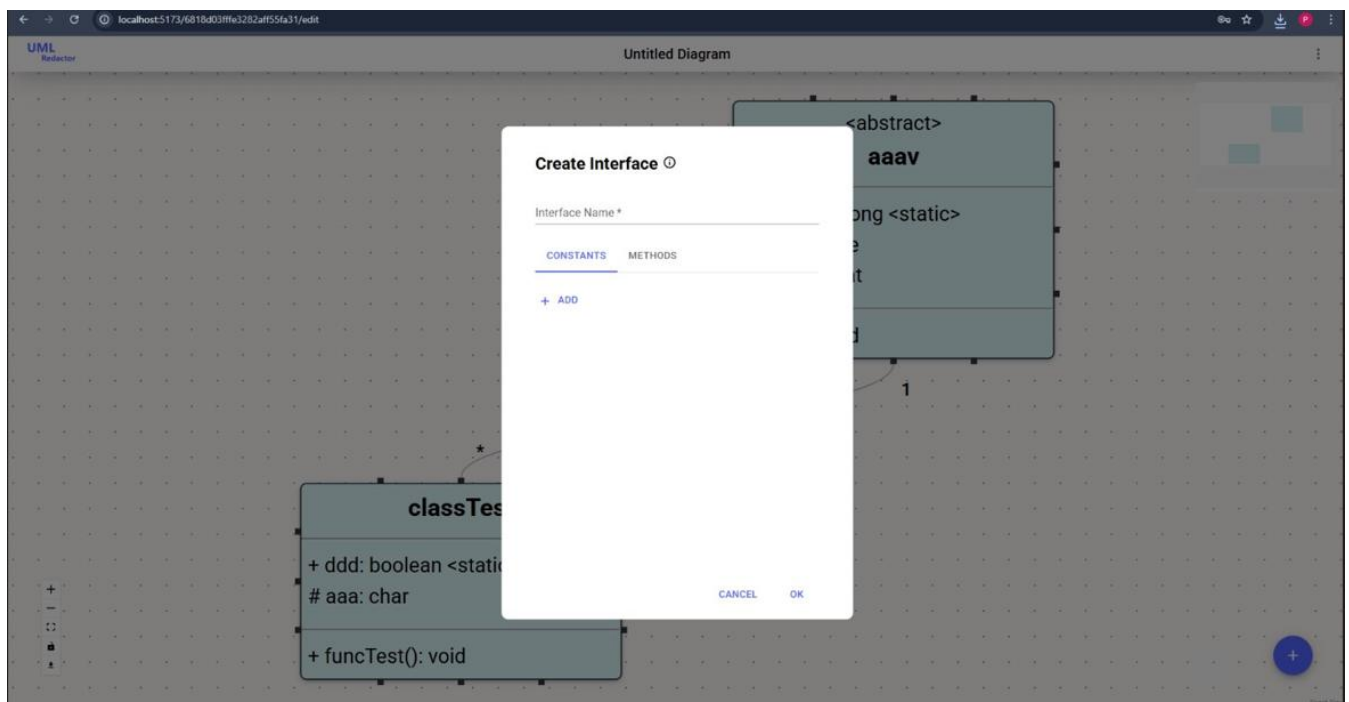


Рис.3.13 Вигляд вікна налаштування блоку «Interface»

При створенні блоку-інтерфейсу задається його ім'я та додаються константи та методи, що будуть йому притаманні. Проведемо заповнення полів як показано на рис. 3.14. Можна підтвердити або відмінити дію внизу вікна. Після створення, блок буде доступний для редагування або видалення, рис. 3.15.

Проведемо перевірку інструменту для створення блоку-енумерації. Скориставшись меню інструментів, запускаємо процедуру створення даного блоку. В результаті дій, висвітлюється вікно, що представлене на рис. 3.15.

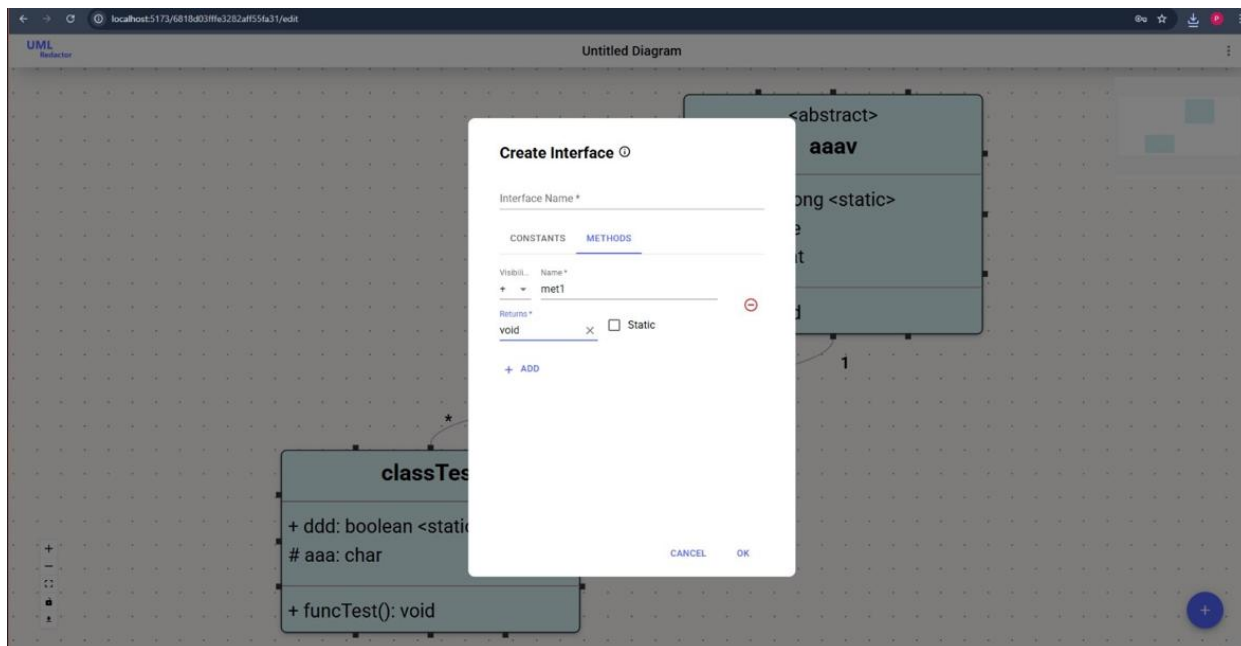


Рис.3.14. Вигляд заповнених даних для створення блоку-інтерфейсу

Для створення блоку-енумерації необхідно задати ім'я блоку, та змінні, які будуть використовуватися, задавши їх ім'я і підтверджуємо введені дані, натиснувши «ок» (див. рис.3.15). На задньому плані даного рисунка ми бачимо створений блок-інтерфейс із заданими параметрами на попередньому етапі, що підтверджує функціональність даного інструменту.

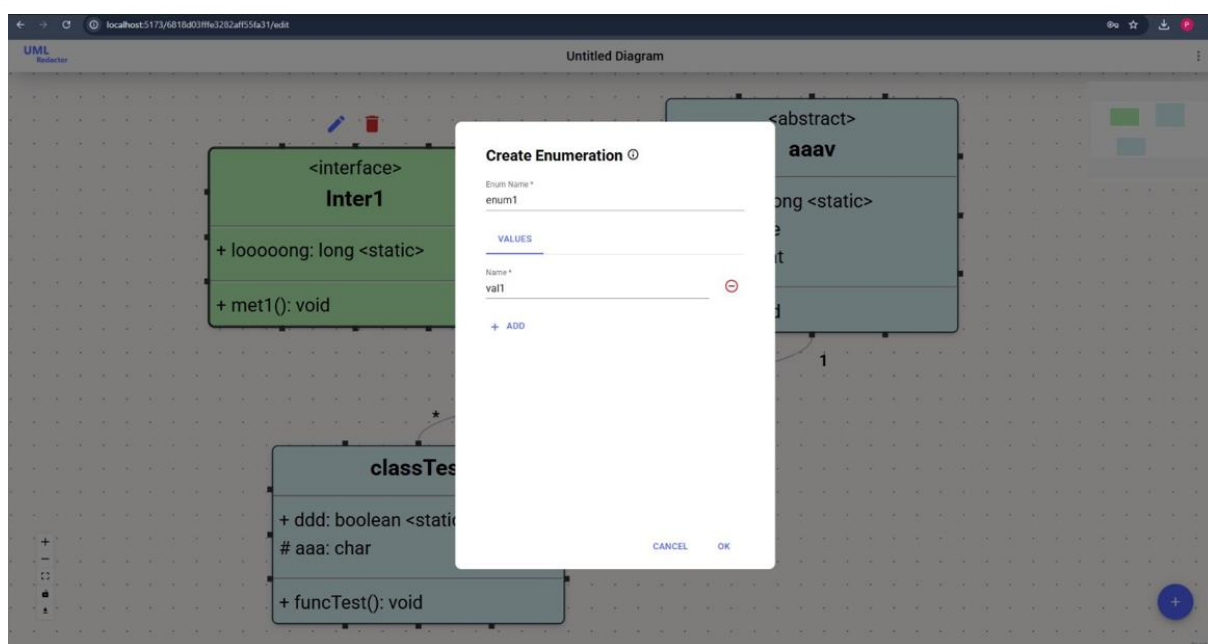


Рис.3.15 Вигляд вікна для створення блоку-енумерації

Перевіримо встановлення зв'язків між створеними блоками. Для цього скориставшись меню інструментів, запускаємо інструмент встановлення зв'язків, що представлено на рисунку 3.16 і заповнюємо поля.

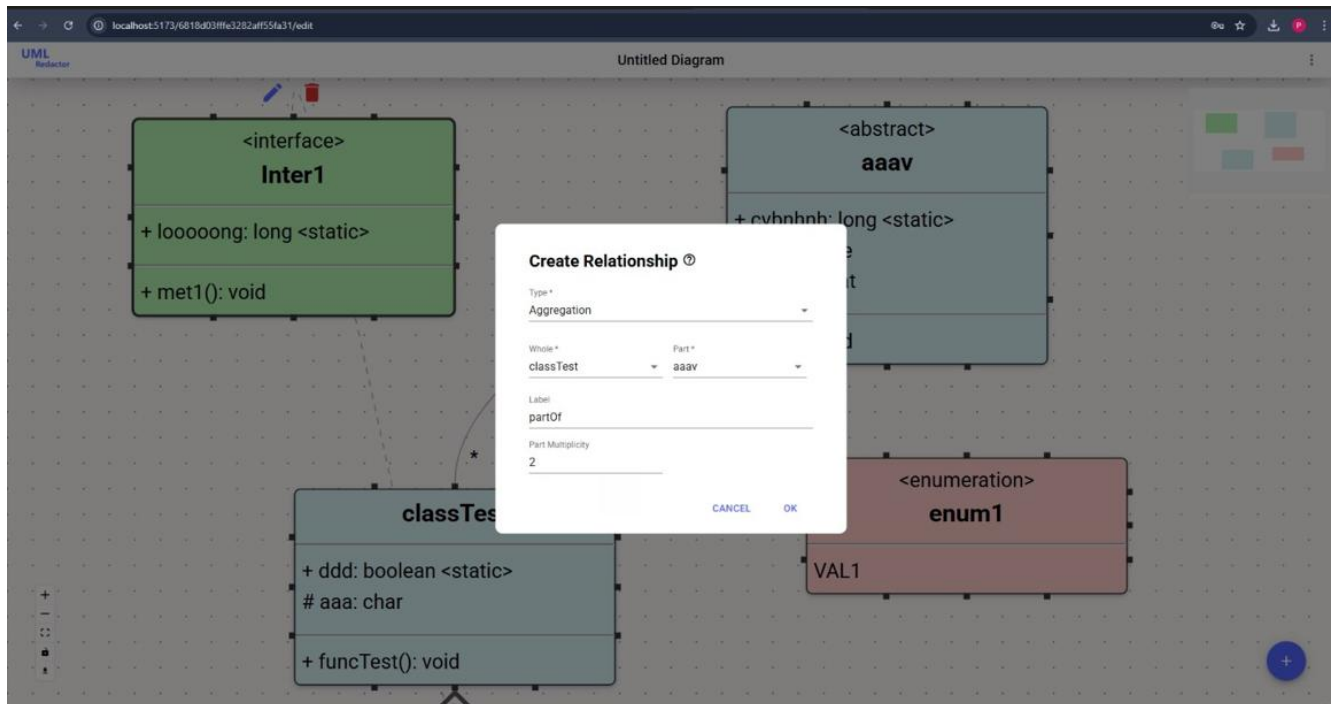


Рис.3.16. Вигляд вікна для встановлення зв'язків

Далі підтверджуємо свої дії і отримуємо блок-схему, що зображена на рис. 3.17.

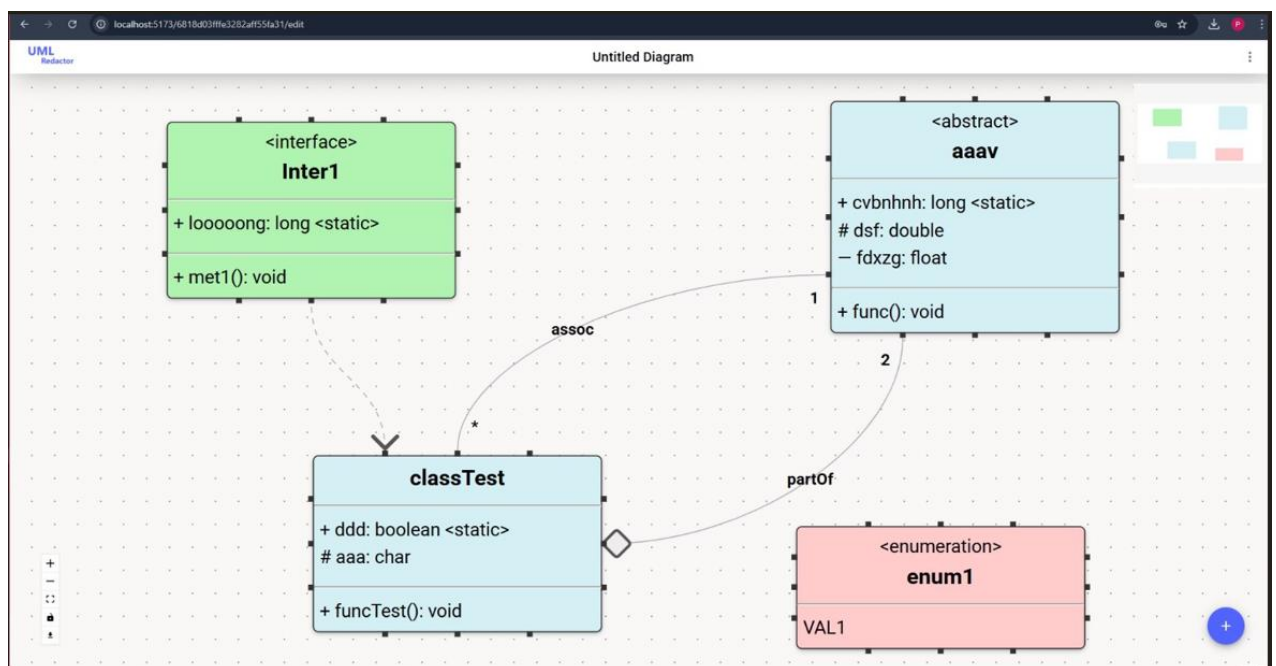


Рис. 3.17. Вигляд побудованої блок-схеми

Проведемо перевірку працездатності процедури перейменування діаграм. Для цього скористаємося випадним меню, що активується за допомогою трьох вертикальних крапок в правому верхньому кутку робочого вікна. В результаті таких дій відкриється вікно для перейменування діаграми, що зображене на рис.3.18. Вводимо бажане ім'я і підтверджуємо свої дії.

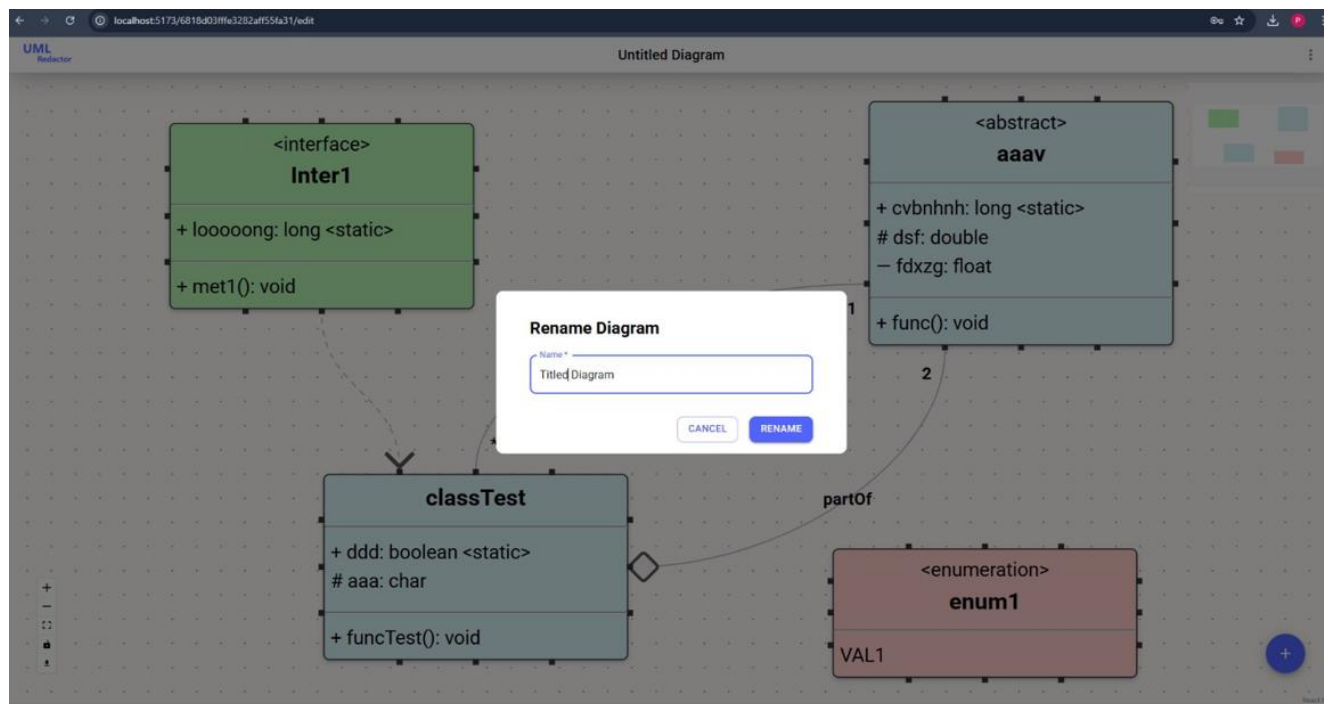


Рис. 3.18. Вигляд вікна для перейменування діаграми

В результаті таких дій створено діаграму, що відображена у переліку створених діаграм (My Diagrams) та додатково відбувається інформування користувача про останню подію, що сталася з діаграмою у вигляді повідомлення у верхній частині вікна, рис.3.19.

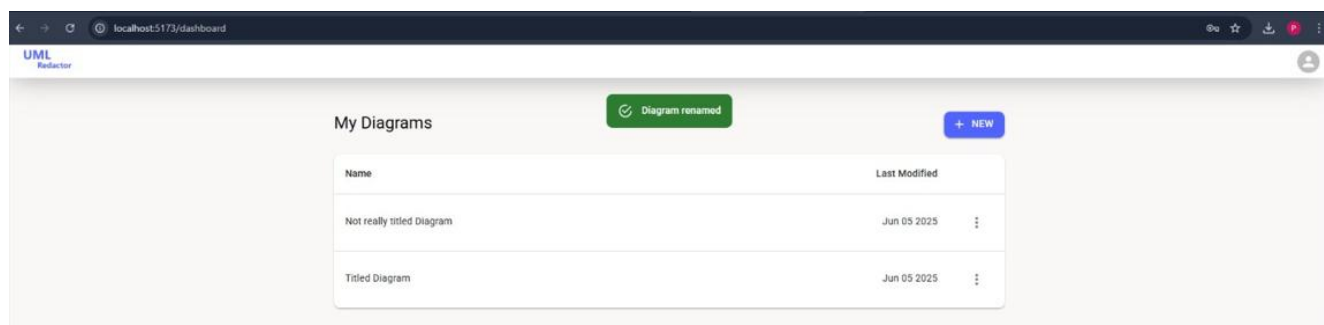


Рис.3.19 Вигляд вікна «My Diagrams» після проведених дій

На наступному етапі проведемо перевірку скачування діаграм. Для цього нам необхідно скористатися меню ліворуч. Активувати процедуру скачування. Як видно з рисунку 3.20, відбулося скачування діаграми та додатково проведено інформування користувача про дану подію.

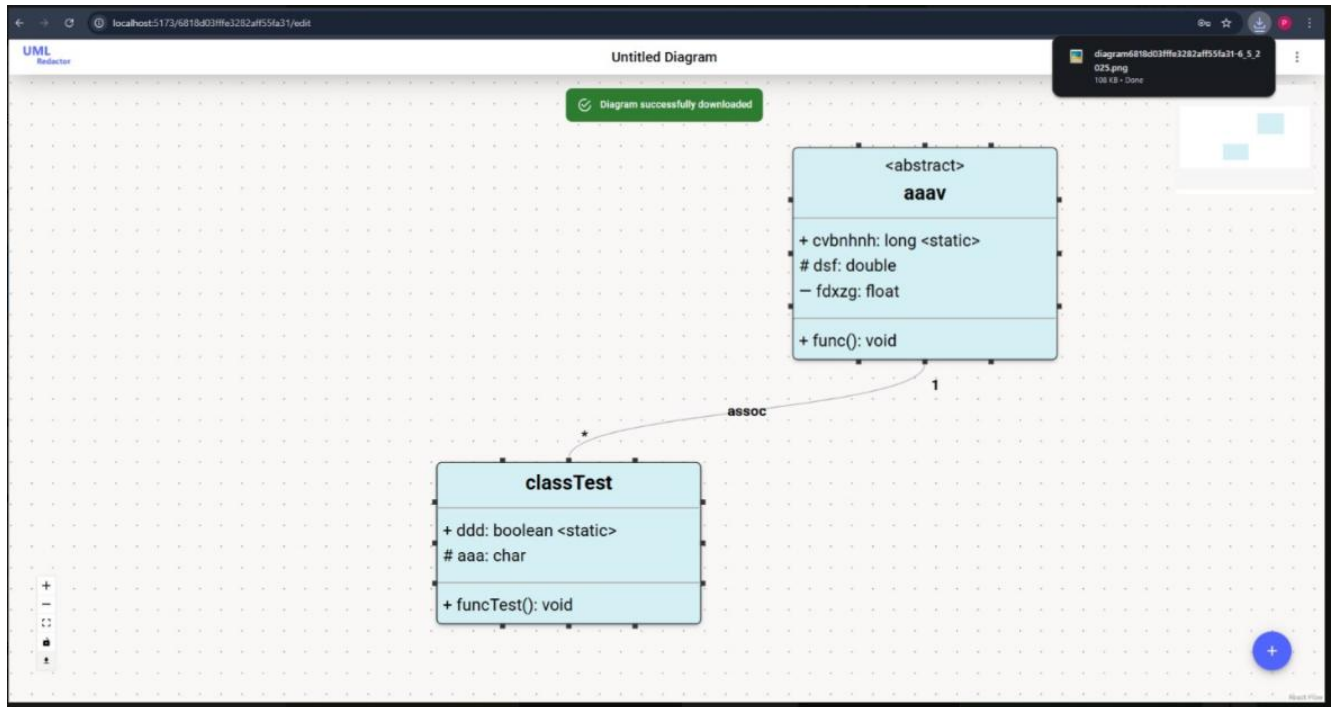


Рис.3.20. Вигляд вікна додатку при скачуванні діаграми

У результаті виконаних дій було перевірено функціональність інструментів веб-додатку для створення блок-схем UML, зокрема створення блоків, встановлення зв'язків між блоками, перейменування діаграм, а також їх скачування. Використання відповідних меню та інструментів підтвердило коректність і зручність реалізації цих функцій у реалізованому веб-додатку. Користувач отримує візуальний зворотний зв'язок у вигляді оновлених вікон та повідомлень про останні події, що сприяє ефективності роботи з додатком.

Висновок до третього розділу

Було проведено опис інтерфейсної частини веб-додатку для створення блок-схем UML. Описані панелі інструментів та меню. Показано як взаємодіяти зі системою.

Сформовано порядок розгортання клієнт-серверного додатку, де вказано послідовність дій, особливості розгортання клієнтської та серверної частини.

Проведене дослідження функціональності веб-додатку для створення блок-схем UML. В результаті проведеного тестування вдалося підтвердити ефективність реалізованих функцій, таких як створення блоків, встановлення зв'язків між ними, перейменування та скачування діаграм. Використання відповідних меню та інструментів забезпечує інтуїтивно зрозумілий інтерфейс для користувача, а візуальний зворотний зв'язок сприяє легкості освоєння додатку.

Таким чином, функціональність веб-додатку відповідає поставленим цілям і задачам дослідження, забезпечуючи зручний і ефективний інструментарій для створення UML-діаграм. Протестовані функції демонструють високу коректність та точність виконання дій, що робить цей додаток цінним для використання в практичній діяльності.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

ВИСНОВКИ

На основі проведених досліджень, було досягнуто ключових результатів, які підтверджують ефективність розробленого веб-додатку для створення блок-схем UML. Основними висновками є:

1. Вибір архітектури MVC та технологічного стеку (TypeScript, React, Express, MongoDB) був ретельно обґрунтований, враховуючи продуктивність, сучасні тенденції веб-розробки та можливість масштабування системи. Такий підхід забезпечує оптимальне поєднання простоти реалізації та стійкості до змін.
2. Проаналізовано та впроваджено ключові критерії зручності користування (UX), включаючи інтуїтивний інтерфейс, ефективну автентифікацію, шифрування даних і адаптивність системи до різних типів пристроїв (десктопи, планшети, смартфони). Це дозволяє створити безпечну й доступну систему, яка відповідає потребам користувачів.
3. Спроектовано веб-додаток для побудови блок-схем, визначено функціональні та нефункціональні вимоги, класи користувачів і логіку взаємодії елементів системи. База даних включає всі ключові сутності, такі як користувачі, діаграми, фігури, шаблони та історія редагувань, що забезпечує високу ефективність збереження даних і контроль доступу.
4. Розроблено сценарії використання, діаграми активності та класи системи, які демонструють логіку роботи додатку. Це сприяє формуванню чіткої структури і дозволяє мінімізувати ризики, пов'язані з реалізацією проєкту.
5. Описано порядок розгортання клієнт-серверного додатку, включаючи всі ключові етапи, що сприяє простому та зрозумілому налаштуванню системи для забезпечення її стабільної роботи.
6. Проведене тестування підтвердило ефективність основних функцій додатку, таких як створення блоків UML, встановлення зв'язків між елементами, перейменування та скачування діаграм. Використання інтуїтивних меню та інструментів гарантує зручність для користувачів.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

Функціональність веб-додатку повністю відповідає поставленим цілям дослідження, демонструючи високу точність виконання операцій. Це робить додаток корисним і практичним інструментом для створення UML-діаграм у професійній діяльності.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A basic study and review of UML versions and UML tools [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://medium.com/@okworthukwuebube/a-basic-study-and-review-of-uml-versions-and-uml-tools-af2c85e606> (дата звернення 14.03.2025).
2. Building a Full-Stack Application with React and Node.js – Best Practices and Tips [Електронний ресурс]. – Режим доступу до ресурсу: <https://medium.com/@nagarjunmallesh/building-a-full-stack-application-with-react-and-node-js-best-practices-and-tips-f9877494fab2> (дата звернення 18.03.2025).
3. Carson Gross. Model/View/Controller (MVC). htmx. 16.01.2024. [Електронний ресурс] – Режим доступу: [MVC Architecture in Java - Tpoint Tech](#) (дата звернення 13.05.2025).
4. Class Diagram | Unified Modeling Language (UML). [Електронний ресурс] – Режим доступу: <https://www.geeksforgeeks.org/system-design/unified-modeling-language-uml-class-diagrams/> (дата звернення 30.04.2025)
5. Costa, L.F. Testing JavaScript Applications / Lucas Fernandes da Costa – Manning Publications, 2021. – 384 с. – ISBN: 9781617297910 (дата звернення 04.04.2025).
6. Creately [Електронний ресурс] – Режим доступу: <https://creately.com/> (дата звернення 24.03.2025).
7. draw.io (Diagrams.net) [Електронний ресурс] – Режим доступу: <https://app.diagrams.net/> (дата звернення 14.03.2025).
8. Flux Architecture [Електронний ресурс] – Режим доступу: <https://facebookarchive.github.io/flux/docs/in-depth-overview/> (дата звернення 18.03.2025).
9. Konva.js Documentation [Електронний ресурс] – Режим доступу: <https://konvajs.org/> (дата звернення 18.04.2025).

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

10. Lucidchart [Електронний ресурс] – Режим доступу: <https://www.lucidchart.com/> (дата звернення 14.03.2025).
11. MDN Web Docs [Електронний ресурс] – Режим доступу: <https://developer.mozilla.org/uk> (дата звернення 14.03.2025)/.
12. Microsoft Visio [Електронний ресурс] – Режим доступу: <https://www.microsoft.com/uk-ua/microsoft-365/visio/flowchart-software> (дата звернення 14.03.2025).
13. Miell, I. Docker in Practice, Second Edition [Електронний ресурс] – Режим доступу до ресурсу: <https://www.manning.com/books/docker-in-practicessecond-edition> (дата звернення 05.05.2025).
14. MVC Architecture in Java. [Електронний ресурс] – Режим доступу: [MVC Architecture in Java - Tpoint Tech](#) (дата звернення 13.05.2025).
15. MVC Design Pattern. [Електронний ресурс] – Режим доступу: <https://www.geeksforgeeks.org/system-design/mvc-design-pattern/> (дата звернення 10.04.2025).
16. MVC Framework Introduction. [Електронний ресурс] – Режим доступу: [MVC Framework Introduction - GeeksforGeeks](#) (дата звернення 30.04.2025)
17. MVVM Architecture [Електронний ресурс] – Режим доступу: <https://en.wikipedia.org/wiki/Model-view-viewmodel> (дата звернення 02.05.2025).
18. MySQL for Developers – PlanetScale [Електронний ресурс]. – Режим доступу до ресурсу: <https://planetscale.com/learn/courses/mysql-for-developers> (дата звернення 30.04.2025).
19. MySQL Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/mysql-tutorial-excerpt/5.7/en/> (дата звернення 30.04.2025).
20. NestJS Documentation [Електронний ресурс] – Режим доступу: <https://nestjs.com/> (дата звернення 24.05.2025).
21. Node.js v24.0.2 documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/docs/latest/api/> (дата звернення 21.04.2025).

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

22. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: <https://www.postgresql.org/docs/> (дата звернення 29.04.2025).

23. Rafael D. Hernandez. The Model View Controller Pattern – MVC Architecture and Frameworks Explained. [Електронний ресурс] – Режим доступу: [The Model View Controller Pattern – MVC Architecture and Frameworks Explained \(freecodecamp.org\)](https://www.freecodecamp.org/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/) (дата звернення 30.04.2025).

24. Swagger Documentation [Електронний ресурс] – Режим доступу: <https://swagger.io/> (дата звернення 4.05.2025).

25. TypeORM Documentation [Електронний ресурс] – Режим доступу: <https://typeorm.io/> (дата звернення 6.04.2025).

26. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] – Режим доступу до ресурсу: <https://evergreens.com.ua/ua/articles/uml-diagrams.html> (дата звернення 6.04.2025).

27. Vue.js Documentation [Електронний ресурс] – Режим доступу: <https://vuejs.org/> (дата звернення 18.03.2025).

28. W3Schools [Електронний ресурс] – Режим доступу: <https://www.w3schools.com/> (дата звернення 18.03.2025).

29. Zosym Махум. Уніфікована мова моделювання (Unified Modeling Language - UML). [Електронний ресурс] – Режим доступу: [Уніфікована мова моделювання \(Unified Modeling Language - UML\) \(maxzosim.com\)](https://maxzosim.com/unified-modeling-language-uml/) ((дата звернення 13.05.2025).

30. Застосування техніки UML «Діаграма варіантів використання» (Use Case Diagram) [Електронний ресурс] – Режим доступу: <https://www.ba.in.ua/2024/02/23/zastosuvannya-tehniky-uml-diagrama-variantivvykorystannya-use-case-diagram/> (дата звернення 07.04.2025)

31. Самчук Л., Повстяна Ю., Ліщина Н., Кліменко А. Використання UML діаграм для технологічного процесу на виробництві. Вісник Хмельницького національного університету. Номер: №2, 2023 (319). С.268-275. DOI: <https://www.doi.org/10.31891/2307-5732-2023-319-1-268-275>.

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

ДОДАТКИ

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

```

import bcrypt from 'bcrypt';
import jwt, { JwtPayload } from 'jsonwebtoken';
import { User, UserModel } from '../models/user.model';
import { sendEmail } from './emailService';

const VERIFY = 'verify';
const RESET = 'reset';
const AUTH = 'auth';

const getUserByEmail = async (email: string) => {
  const user = await UserModel.findOne({ email });
  if (!user) throw new Error('User not found');
  return user;
};

const login = async (email: string, password: string) => {
  if (!email || typeof email !== 'string') throw new Error('Invalid email');
  if (!password || typeof password !== 'string')
    throw new Error('Invalid password');

  const user = await UserModel.findOne({ email });
  if (!user) throw new Error('User not found');

  const match = await bcrypt.compare(password, user.password);
  if (!match) throw new Error('Invalid password');

  if (user.verified === false) throw new Error('User not verified');

  const token = jwt.sign(
    { type: AUTH, userId: user._id },
    process.env.JWT_SECRET as string,
    {
      expiresIn: '7d',
    }
  );
  return { userId: user._id, username: user.username, authToken: token };
};

const signup = async (username: string, email: string, password: string) => {
  if (!username || typeof username !== 'string')
    throw new Error('Invalid username');
  if (!email || typeof email !== 'string') throw new Error('Invalid email');
  if (!password || typeof password !== 'string')
    throw new Error('Invalid password');

  const duplicateEmail = await UserModel.findOne({ email });
  if (duplicateEmail)
    throw new Error('An account already exists with this email');

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

const duplicateUsername = await UserModel.findOne({ username });
if (duplicateUsername) throw new Error('Username taken');

const hashedPassword = await bcrypt.hash(password, 10);
const newUser = await UserModel.create({
  username,
  email,
  password: hashedPassword,
});
return newUser;
};

const verifyAccount = async (token: string) => {
  if (!token || typeof token !== 'string') throw new Error('Invalid token');
  let decoded: JwtPayload;
  try {
    decoded = jwt.verify(token, process.env.JWT_SECRET as string) as JwtPayload;
  } catch (err) {
    throw new Error('Invalid or expired token');
  }
  if (!decoded || decoded.action !== VERIFY) throw new Error('Invalid token');

  const user = await UserModel.findById(decoded.userId);
  if (!user) throw new Error('User not found');
  if (user.verified === true) throw new Error('User already verified');

  user.verified = true;
  await user.save();
};

const sendVerificationEmail = async (user: User) => {
  // create a token that expires in 2 days
  const token = jwt.sign(
    { action: VERIFY, userId: user._id },
    process.env.JWT_SECRET as string,
    {
      expiresIn: '2d',
    }
  );

  const link = `${process.env.CLIENT_URL}/verify?token=${token}`;
  const templateParams = {
    to: user.email,
    subject: 'Email Confirmation',
    text: `
      <h2>Welcome to UML-Redactor!</h2>
      <p>Thank you for registering with us. Please confirm your email address to activate your
      account.</p>

      <p>Click the link below to confirm your email address:</p>
      <p><a href="${link}" target="_blank">Confirm your email address</a></p>
    `
  };

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    <p>If you did not create an account with us, please disregard this email.</p>

    <p>Best regards,</p>
    <p>The UML-Redactor Team</p>
  `,
  /* username: user.username,
  htmlLink: `<a href="\${link}">Verify your email</a>`,
  link, */
  });

  try {
    await sendEmail(
      templateParams.to,
      templateParams.subject,
      templateParams.text
    );
  } catch (err) {
    console.log(err);
    throw new Error('Error sending email');
  }
};

const sendPasswordResetEmail = async (user: User) => {
  // create a token that expires in 2 days
  const token = jwt.sign(
    { action: RESET, userId: user._id },
    process.env.JWT_SECRET as string,
    {
      expiresIn: '1h',
    }
  );

  const link = `${process.env.CLIENT_URL}/reset-password?token=${token}`;
  const templateParams = {
    to: user.email,
    subject: 'Reset password',
    text: `<h2>Password Reset Request</h2>
    <p>Hello,</p>
    <p>We received a request to reset your password. If you didn't make this request, please ignore
this email.</p>
    <p>To reset your password, click the link below:</p>
    <p>
    <a href="\${link}">Reset Your Password</a>
    </p>
    <p>If you have any questions, feel free to contact our support team.</p>
    <p>Best regards,<br/>The UML-Redactor Team</p>
  `,
  };

  try {

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

    await sendEmail(
      templateParams.to,
      templateParams.subject,
      templateParams.text
    );
  } catch (err) {
    console.log(err);
    throw new Error('Error sending email');
  }
};

const resetPassword = async (token: string, password: string) => {
  if (!token || typeof token !== 'string') throw new Error('Invalid token');
  if (!password || typeof password !== 'string')
    throw new Error('Invalid password');

  const decoded = jwt.verify(
    token,
    process.env.JWT_SECRET as string
  ) as JwpPayload;
  if (!decoded || decoded.action !== RESET) throw new Error('Invalid token');

  const user = await UserModel.findById(decoded.userId);
  if (!user) throw new Error('User not found');

  const hashedPassword = await bcrypt.hash(password, 10);
  user.password = hashedPassword;
  await user.save();
};

export {
  getUserByEmail,
  login,
  resetPassword,
  sendPasswordResetEmail,
  sendVerificationEmail,
  signup,
  verifyAccount,
};

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

import DownloadIcon from '@mui/icons-material/Download';
import axios from 'axios';
import { toPng } from 'html-to-image';
import { MouseEvent, useCallback, useEffect, useRef } from 'react';
import { useParams } from 'react-router-dom';
import ReactFlow, {
  Background,
  BackgroundVariant,
  Connection,
  ConnectionMode,
  ControlButton,
  Controls,
  Edge,
  EdgeChange,
  MiniMap,
  NodeChange,
  applyEdgeChanges,
  applyNodeChanges,
  getRectOfNodes,
  getTransformForBounds,
} from 'reactflow';
import 'reactflow/dist/base.css';
import {
  useEntities,
  useEntitiesDispatch,
} from '../context/EntitiesContext';
import {
  useRelationships,
  useRelationshipsDispatch,
} from '../context/RelationshipsContext';
import '../styles/Editor.css';
import { Entity, Relationship } from '../modelTypes';
import { AlertType } from '../alert/alertContext';
import useAlert from '../alert/useAlert';
import AggregationEdge from './edges/AggregationEdge';
import AssociationEdge from './edges/AssociationEdge';
import CompositionEdge from './edges/CompositionEdge';
import DependencyEdge from './edges/DependencyEdge';
import ImplementationEdge from './edges/ImplementationEdge';
import InheritanceEdge from './edges/InheritanceEdge';
import ClassNode from './nodes/ClassNode';
import EnumNode from './nodes/EnumNode';
import InterfaceNode from './nodes/InterfaceNode';

const nodeTypes = {
  class: ClassNode,
  interface: InterfaceNode,
  enum: EnumNode,
};

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export const nodeColor = (node: Entity) => {
  switch (node.type) {
    case 'class':
      return '#D4F1F4';
    case 'interface':
      return '#b1f3b1';
    case 'enum':
      return '#ffcccb';
    default:
      return '#ffffff';
  }
};

const edgeTypes = {
  Inheritance: InheritanceEdge,
  Association: AssociationEdge,
  Dependency: DependencyEdge,
  Implementation: ImplementationEdge,
  Aggregation: AggregationEdge,
  Composition: CompositionEdge,
};

type Props = {
  ent: Entity[];
  rel: Relationship[];
};

function DiagramEditor({ ent, rel }: Props) {
  const entities = useEntities();
  const entitiesDispatch = useEntitiesDispatch();
  const relationships = useRelationships();
  const relationshipsDispatch = useRelationshipsDispatch();
  const edgeUpdateSuccessful = useRef(true);

  const { diagramId } = useParams();
  const { setAlert } = useAlert();

  useEffect(() => {
    entitiesDispatch({ type: 'SET_ENTITIES', payload: ent });
    relationshipsDispatch({ type: 'SET_RELATIONSHIPS', payload: rel });
  }, [entitiesDispatch, relationshipsDispatch, ent, rel]);

  const onNodesChange = useCallback(
    async (changes: NodeChange[]) =>
      entitiesDispatch({
        type: 'SET_ENTITIES',
        payload: applyNodeChanges(changes, entities),
      }),
    [entities, entitiesDispatch]
  );
};

```

```

const onNodeDragStop = useCallback(
  async (event: MouseEvent, node: Entity) => {
    event.stopPropagation();
    try {
      const newPos = { x: node.position.x, y: node.position.y };
      await axios.put(
        `/api/entity/${node.id}/position?diagramId=${diagramId}`,
        newPos
      );
    } catch (e) {
      setAlert(
        'Could not update position. Please try again',
        AlertType.ERROR
      );
    }
  },
  [diagramId, setAlert]
);

const onNodesDelete = useCallback(
  async (nodes: Entity[]) => {
    try {
      const nodeToDelete = nodes[0];
      await axios.delete(
        `/api/entity/${nodeToDelete.id}?diagramId=${diagramId}`
      );
      entitiesDispatch({ type: 'DELETE_ENTITY', id: nodeToDelete.id });
      setAlert('Entity successfully deleted', AlertType.SUCCESS);
    } catch (e) {
      setAlert('Could not delete entity. Try again', AlertType.ERROR);
    }
  },
  [diagramId, entitiesDispatch, setAlert]
);

const onConnect = useCallback(() => {
  setAlert(
    'Unsupported Action - Please use the relationship form',
    AlertType.INFO
  );
}, [setAlert]);

// handle select and remove of edges
const onEdgesChange = useCallback(
  (changes: EdgeChange[]) =>
    relationshipsDispatch({
      type: 'SET_RELATIONSHIPS',
      payload: applyEdgeChanges(changes, relationships),
    }),
  [relationships, relationshipsDispatch]
);

```

```

);

// handle edge position updates, three separate handlers used to better alert the user of errors
const onEdgeUpdateStart = useCallback(() => {
  edgeUpdateSuccessful.current = false;
}, []);

const onEdgeUpdate = useCallback(
  async (oldEdge: Edge, newConnection: Connection) => {
    edgeUpdateSuccessful.current = true;
    const newEdge = {
      ...oldEdge,
      ...newConnection,
      id: oldEdge.id,
    } as Edge;
    try {
      relationshipsDispatch({
        type: 'UPDATE_RELATIONSHIP',
        payload: newEdge,
      });
      await axios.put(
        `/api/relationship/${oldEdge.id}/handle?diagramId=${diagramId}`,
        {
          type: oldEdge.type,
          ...newConnection,
        }
      );
    } catch (e) {
      showAlert('Cannot update relationship to that entity', AlertType.ERROR);
      relationshipsDispatch({
        type: 'UPDATE_RELATIONSHIP',
        payload: oldEdge,
      });
    }
  },
  [diagramId, relationshipsDispatch, showAlert]
);

const onEdgeUpdateEnd = useCallback(() => {
  if (!edgeUpdateSuccessful.current) {
    showAlert('Could not detect a port', AlertType.WARNING);
  }
  edgeUpdateSuccessful.current = true;
}, [setAlert]);

const onEdgeDelete = useCallback(
  async (edges: Edge[]) => {
    try {
      const edgeToDelete = edges[0];
      await axios.delete(
        `/api/relationship/${edgeToDelete.id}?diagramId=${diagramId}`

```

```

);
relationshipsDispatch({
  type: 'DELETE_RELATIONSHIP',
  id: edgeToDelete.id,
});
setAlert('Relationship successfully deleted', AlertType.SUCCESS);
} catch (e) {
  setAlert('Could not delete relationship. Try again', AlertType.ERROR);
}
},
[diagramId, relationshipsDispatch, setAlert]
);

const handleDownload = async () => {
  setAlert('Downloading diagram...', AlertType.INFO);
  const nodesBounds = getRectOfNodes(entities);
  const imageWidth = 1800;
  const imageHeight = 1200;
  const transform = getTransformForBounds(
    nodesBounds,
    imageWidth,
    imageHeight,
    0.5,
    2
  );

  const viewport = document.querySelector(
    '.react-flow__viewport'
  ) as HTMLElement;
  if (viewport) {
    const dataUrl = await toPng(viewport, {
      backgroundColor: '#faf9f6',
      width: imageWidth,
      height: imageHeight,
      style: {
        width: String(imageWidth),
        height: String(imageHeight),
        transform: `translate(${transform[0]}px, ${transform[1]}px) scale(${transform[2]})`,
      },
    });
  }

  const now = new Date();
  const link = document.createElement('a');
  link.download = `diagram${diagramId}-${now.toLocaleDateString()}.png`;
  link.href = dataUrl;
  link.click();
  setAlert('Diagram successfully downloaded', AlertType.SUCCESS);
}

return (

```

		Шумляківський Р.В.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

```

<div className="diagram">
  <ReactFlow
    fitView
    nodes={entities}
    nodeTypes={nodeTypes}
    onNodesChange={onNodesChange}
    onNodeDragStop={onNodeDragStop}
    onNodesDelete={onNodesDelete}
    onConnect={onConnect}
    edges={relationships}
    edgeTypes={edgeTypes}
    onEdgesChange={onEdgesChange}
    onEdgeUpdateStart={onEdgeUpdateStart}
    onEdgeUpdate={onEdgeUpdate}
    onEdgeUpdateEnd={onEdgeUpdateEnd}
    onEdgesDelete={onEdgeDelete}
    connectionMode={ConnectionMode.Loose}
  >
    <Background color="#444" variant={'dots' as BackgroundVariant} />
    <Controls style={{ flexDirection: 'column', gap: 0, margin: '3em' }}>
      <ControlButton onClick={handleDownload} title="Download Diagram">
        <DownloadIcon />
      </ControlButton>
    </Controls>
    <MiniMap pannable zoomable position="top-right" nodeColor={nodeColor} />
  </ReactFlow>
</div>
);
}

export default DiagramEditor;

```