Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ІНТЕРАКТИВНОГО ВЕБДОДАТКУ ДЛЯ СПІЛЬНОГО МАЛЮВАННЯ З ВИКОРИСТАННЯМ SOCKET.IO

DEVELOPMENT OF AN INTERACTIVE WEB APPLICATION FOR COLLABORATIVE DRAWING USING SOCKET.IO

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Хмільовський Олександр Миколайович

Керівник:
Суринович Олена Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 10 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Хмільовському Олександру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка інтерактивного вебдодатку для спільного малювання з використанням Socket.IO»*
Керівник роботи: *к.т.н., доцент Суринович О. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *MongoDB, Express.js, React, Node.js, редактор Visual Studio Code, Tailwind CSS, DaisyUI, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз предметної області і існуючих програмних рішень, визначення вимог до розробки вебдодатка, вибір засобів для реалізації вебдодатка, розробка вебдодатка, тестування з обраними засобами*
5. Перелік графічного матеріалу: *15 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Суринович О. М.*		
Специфікація вимог до розробленої системи	*Суринович О. М.*		
Розробка об`єкта проектування	*Суринович О. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,81 %		
Академічна доброчесність	*Суринович О. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Олександр ХМІЛЬОВСЬКИЙ

Керівник кваліфікаційної роботи _____ Олена СУРИНОВИЧ

АНОТАЦІЯ

Хмільовський О. М. Розробка інтерактивного вебдодатку для спільного малювання з використанням Socket.IO. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області та існуючих аналогічних рішень, визначено основні завдання проекту. У другому розділі описано архітектуру системи, створення моделей даних та вибір технологій для реалізації вебдодатка. У третьому розділі представлено процес розробки клієнтської та серверної частин додатка, інтеграцію Socket.IO для підтримки роботи в реальному часі, а також проведено тестування функціональності. У висновках підсумовано результати розробки, визначено перспективи розвитку та можливості подальшого вдосконалення системи.

Ключові слова: MERN-стек, React, Node.js, MongoDB, Socket.IO, вебдодаток, реальний час, Canvas, малювання, багатокористувацький інтерфейс.

ABSTRACT

Khmilovsky O. Development of an Interactive Web Application for Collaborative Drawing Using Socket.IO. Manuscript. Bachelor's qualification work of the OP "Software Engineering" specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three sections, conclusions, a list of used sources and appendices.

The first section analyzes the subject area and existing similar solutions, identifies the main tasks of the project. The second section describes the system architecture, the creation of data models and the choice of technologies for implementing the web application. The third section presents the process of developing the client and server parts of the application, the integration of Socket.IO to support real-time work, and functionality testing. The conclusions summarize the development results, identify development prospects and opportunities for further improvement of the system.

Keywords: MERN stack, React, Node.js, MongoDB, Socket.IO, web application, real-time, Canvas, drawing, multi-user interface.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап розвитку вебтехнологій характеризується високими вимогами до інтерактивності, масштабованості та залучення користувачів у спільну взаємодію в реальному часі. Одним із актуальних напрямів є створення креативних платформ для колективної творчості, які забезпечують високу швидкодію та синхронізацію дій великої кількості користувачів. Реалізація подібної платформи на основі сучасного MERN-стеку (MongoDB, Express.js, React, Node.js) із використанням бібліотеки Socket.IO дозволяє забезпечити ефективну обробку подій у реальному часі та підтримку багатокористувацької взаємодії.

Мета роботи – розробити вебдодаток, який дає можливість користувачам у реальному часі змінювати пікселі на спільному полотні, а також створювати окремі кімнати (лобі) з індивідуальними полотнами для малювання, використовуючи стек MERN та Socket.IO.

Об’єкт роботи – технології створення інтерактивних вебзастосунків із функціональністю реального часу.

Предмет роботи – методи та засоби реалізації клієнт-серверної архітектури вебдодатків на основі MERN-стеку з інтеграцією Socket.IO для синхронізації змін на полотні між користувачами.

Для досягнення поставленої мети були визначені такі завдання:

– проаналізувати предметну область та аналогічні рішення;
– визначити вимоги до розроблюваної системи;
– спроєктувати архітектуру системи;
– обрати технології для розробки;
– провести розробку вебдодатку;
– провести тестування;
– розробити базу даних;
– реалізувати синхронізацію даних в реальному часі;
– розробити захисні механізми для вебдодатку.

Апробація результатів дослідження (додаток А): Хмільовський О. М. Суринович О.М. Транслювання інформації в реальному часі у вебзастосунках. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 403-406.

РОЗДІЛ 1
АНАЛІЗ ІНТЕРКАКТИВНИХ ВЕБДОДАТКІВ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Інформація є основою сучасного цифрового світу, але її оперативність відіграє не менш важливу роль, особливо в інтерактивних вебдодатках. Швидке отримання даних забезпечує зручність користування, підвищує довіру до платформи та дозволяє користувачам оперативно реагувати на зміни.

Технології реального часу стали основою функціональності багатьох популярних вебдодатків, якими щодня користуються мільйони людей. Наприклад, Discord пропонує широкий спектр функцій – від текстових чатів і голосових дзвінків до стрімінгу ігор у реальному часі, що забезпечує живу взаємодію між користувачами. Месенджери, такі як Viber, Telegram чи WhatsApp, використовують протоколи реального часу для синхронізації повідомлень, що дозволяє користувачам обмінюватися текстами, медіафайлами чи навіть проводити групові відеодзвінки без затримок. Соціальні мережі, такі як Facebook, Twitter чи Instagram, застосовують подібні технології для миттєвого оновлення стрічок новин, відображення лайків, коментарів чи нових публікацій. Ці платформи також інтегрують push-повідомлення, щоб користувачі отримували сповіщення про важливі події в момент їх виникнення. Наприклад, у Facebook оновлення коментарів у реальному часі дозволяє вести живі дискусії, що підвищує залученість користувачів. Крім того, такі платформи використовують складні алгоритми кешування та розподілені бази даних для забезпечення стабільної роботи навіть при пікових навантаженнях, наприклад, під час масових подій чи вірусних трендів. Це дозволяє мільйонам користувачів одночасно взаємодіяти з контентом без помітних затримок, що робить реальний час стандартом для сучасних цифрових сервісів.

Користувачі очікують миттєвого зворотного зв’язку, чи то в месенджерах, соціальних мережах, чи ігрових платформах. Без технологій реального часу

багато сервісів втратили б свою привабливість, адже затримки в обробці даних знижують якість взаємодії та можуть призвести до втрати аудиторії. Наприклад, у потокових сервісах, таких як Twitch, затримка в передачі відео чи чату може суттєво погіршити досвід глядачів і стримерів. Для забезпечення такої швидкості розробники використовують технології, як-от WebSocket, Server-Sent Events чи асинхронні API, які дозволяють оновлювати дані без перезавантаження сторінки. Крім того, важливу роль відіграє оптимізація серверної інфраструктури, включаючи балансування навантаження та використання хмарних рішень, таких як AWS чи Google Cloud, для обробки великих обсягів даних у реальному часі. Ці технології дозволяють створювати динамічний і інтерактивний досвід, який відповідає сучасним очікуванням користувачів.

Соціальний експеримент r/place, запущений Reddit у 2017 році, є унікальним прикладом використання технологій реального часу для масової взаємодії. На віртуальному полотні розміром 1000 на 1000 пікселів користувачі могли кожні 5 хвилин розміщувати один піксель, обираючи колір із доступної палітри. Ця проста механіка дозволила мільйонам людей спільно створювати зображення, відображаючи як творчу співпрацю, так і конкуренцію. Спільноти координували свої дії через форуми чи чати, створюючи складні малюнки, такі як логотипи, прапори чи меми, тоді як інші користувачі намагалися їх змінити чи знищити. Фінальний вигляд полотна зображено на рисунку 1.1.

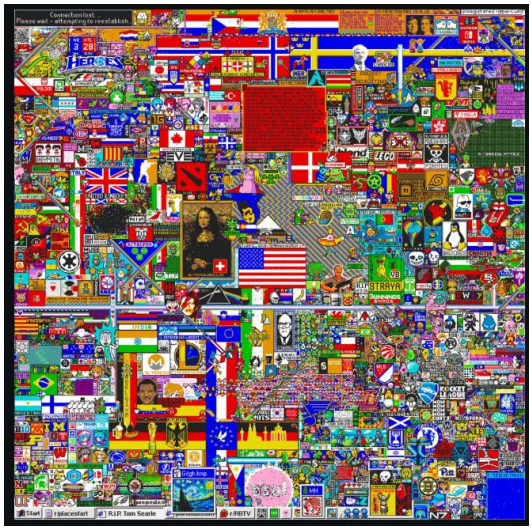

Рисунок 1.1 – Полотно r/place 2017 [12]

Технічна реалізація r/place вимагала складної синхронізації даних у реальному часі [15], щоб кожен піксель оновлювався миттєво для всіх учасників. У 2022 році експеримент повторили, і він залучив ще більше користувачів, що розмістили десятки мільйонів пікселів. Цей проєкт не лише підкреслив важливість реального часу для масової взаємодії, але й показав, як технології можуть об'єднувати людей для творчості, одночасно створюючи виклики, пов'язані з масштабуванням серверів і захистом від зловмисних дій. Наприклад, Reddit використовував розподілені системи та механізми захисту від ботів, щоб забезпечити стабільність платформи та справедливість взаємодії, що стало ключовим фактором успіху проєкту.

Успіх соціального експерименту спричинив появу численних аналогів, які намагалися відтворити або розвинути його концепцію колективного малювання, пропонуючи нові підходи до творчої взаємодії в реальному часі. Такі платформи, як Pixelcanvas.io, OurWorldOfPixels.com, Pixelplace.io, Gartic.io та Drawception, запозичили ідею спільного створення контенту, але кожна з них має унікальні особливості, переваги та недоліки.

Pixelcanvas.io – це вебплатформа, яка дозволяє користувачам малювати на спільному піксельному полотні без обмежень на частоту розміщення пікселів і без необхідності авторизації. Зовнішній вигляд головної сторінки додатку зображено на рисунку 1.2.

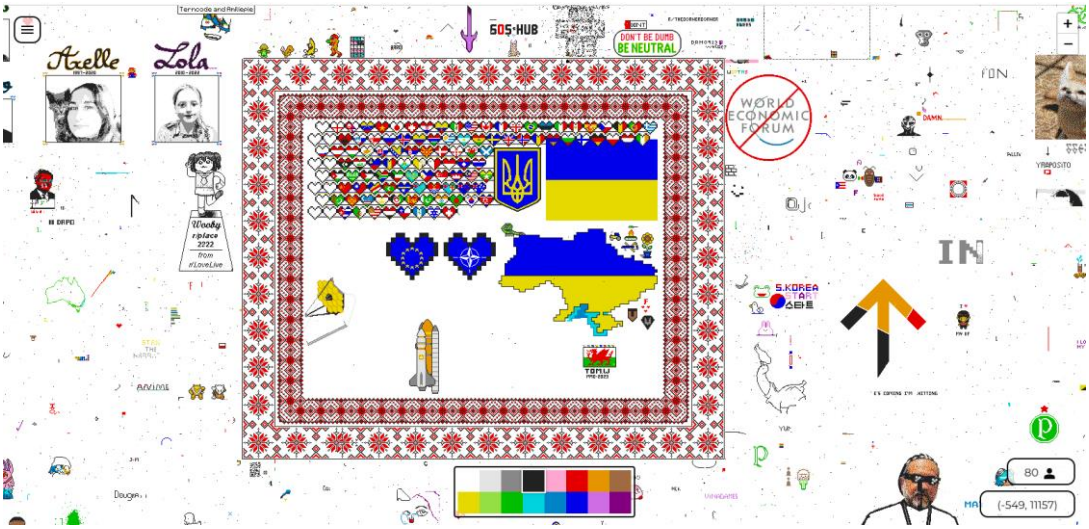

Рисунок 1.2 – Головна сторінка Pixelcanvas.io [10]

Полотно є розширюваним, автоматично збільшуючись залежно від активності користувачів, що дозволяє створювати масштабні зображення. Платформа пропонує обмежену палітру з 10 кольорів. Відсутність вимоги до реєстрації робить платформу надзвичайно доступною, залучаючи широку аудиторію – від випадкових користувачів до організованих спільнот, які координують свої дії через зовнішні платформи, як-от Reddit чи Discord. Однак Pixelcanvas.io стикається з серйозними проблемами вандалізму, оскільки відсутність ефективної модерації дозволяє зловмисникам чи ботам масово змінювати пікселі, знищуючи спільні малюнки. Наприклад, у 2023 році платформа зазнала атак, коли боти заповнювали полотно хаотичними пікселями, що демотивувало творчих учасників.

OurWorldOfPixels.com також пропонує спільне піксельне полотно без вимоги авторизації, дозволяє користувачам вільно обирати кольори з широкої палітри та малювати без часових обмежень. Велике полотно підтримує масштабування, що полегшує навігацію та дозволяє користувачам зосереджуватися на окремих ділянках. Простота інтерфейсу робить платформу зручною для новачків. Однак, як і Pixelcanvas.io, OurWorldOfPixels.com страждає від вандалізму через слабку модерацію. Боти та зловмисники можуть легко порушувати цілісність полотна, що знижує мотивацію учасників. Крім того, монотонність досвіду через відсутність ігрових чи змагальних елементів може зробити платформу менш привабливою для довготривалої участі.

Pixelplace.io вирізняється серед аналогів завдяки поєднанню класичного піксельного полотна з ігровими режимами, такими як командні змагання чи битви за контроль над ділянками. Платформа пропонує кілька тематичних полотен із різними правилами, що додає різноманітності та залучає користувачів, які шукають як творчі, так і змагальні елементи. Наприклад, у режимі «битви» спільноти можуть боротися за контроль над певними зонами полотна, що додає стратегічний аспект. Вбудований чат дозволяє учасникам координувати дії безпосередньо на платформі. Базові інструменти модерації, такі як обмеження на частоту розміщення пікселів для нових користувачів, допомагають зменшити

вандалізм, хоча й не усувають його повністю. Pixelplace.io також підтримує авторизацію, що дозволяє зберігати прогрес і брати участь у рейтингах, додаючи елемент гейміфікації. Складність ігрових режимів може бути бар'єром для новачків, які не знайомі з правилами чи механікою. Незважаючи на ці недоліки, Pixelplace.io залишається однією з найбільш інноваційних альтернатив r/place завдяки своїм змагальним елементам.

Gartic.io трансформує концепцію піксельного малювання в інтерактивну гру за принципом «зламаного телефону», де користувачі малюють за заданими темами, а інші учасники намагаються вгадати зміст малюнка. Платформа підтримує як публічні, так і приватні ігрові кімнати, що дозволяє користувачам організовувати заходи для друзів чи спільнот. Вбудований чат сприяє соціальній взаємодії, дозволяючи учасникам обговорювати гру чи координувати дії. Інтуїтивний інтерфейс і низький бар'єр входу роблять Gartic.io доступним для широкої аудиторії. Однак Gartic.io обмежує творчу свободу, оскільки зосереджена на вгадуванні, а не на створенні масштабних спільних проєктів, як у r/place. Задані теми та обмежений час на малювання стримують можливості для складних зображень. Крім того, слабка модерація дозволяє створювати некоректний контент, що може зіпсувати досвід для інших учасників. Наприклад, у 2024 році користувачі скаржилися на невідповідні малюнки в публічних кімнатах, що підкреслило потребу в кращих інструментах фільтрації.

Drawception також базується на концепції «зламаного телефону», де гравці чергуються між створенням малюнків і описів: один користувач малює за заданим описом, а наступний описує цей малюнок, і так далі. Платформа підтримує реальний час і дозволяє створювати як публічні, так і приватні ігри, що робить її привабливою для групового дозвілля. Присутній вбудований чат і командна гра, а простий інтерфейс забезпечує доступність на різних пристроях, включаючи мобільні. Унікальна механіка додає елемент несподіванки, оскільки результати гри часто стають кумедними через спотворення початкових ідей. Наприклад, початковий опис «кіт на дереві» може перетворитися на «монстр у лісі» через серію малюнків і описів. Однак, як і Gartic.io, Drawception обмежує

творчу свободу, оскільки зосереджена на послідовному малюванні, а не на створенні спільних масштабних зображень.

Усі ці платформи використовують технології реального часу, такі як WebSocket [19], для синхронізації дій користувачів, забезпечуючи миттєве відображення змін на полотні. Однак їхній успіх виявив низку проблем. Вандалізм і атаки ботів є серйозною загрозою: наприклад, Pixelcanvas.io та OurWorldOfPixels.com неодноразово зазнавали масових атак, коли боти заповнювали полотно хаотичними пікселями, знищуючи спільні зусилля. Це вимагає впровадження складних систем модерації, таких як CAPTCHA, обмеження частоти дій чи алгоритми машинного навчання для виявлення підозрілої активності.

Розробка платформ, подібних до r/place, пов’язана з низкою технічних викликів, які впливають на їхню продуктивність і користувацький досвід. Однією з основних проблем є високе навантаження на сервери через велику кількість одночасних підключень. Наприклад, під час піків активності, коли тисячі чи навіть мільйони користувачів одночасно змінюють пікселі, сервери повинні обробляти величезний обсяг запитів у реальному часі. Це вимагає використання розподілених систем, таких як хмарні сервери, і балансування навантаження для уникнення збоїв.

Ще одним викликом є синхронізація даних: платформи використовують WebSocket або Server-Sent Events для миттєвого оновлення полотна, але підтримка стабільного з’єднання для тисяч користувачів потребує оптимізації протоколів і захисту від втрати пакетів. Масштабування бази даних є ще одним викликом, оскільки кожен піксель і його зміни потрібно зберігати та швидко отримувати, що може перевантажувати традиційні реляційні бази даних; для цього часто використовують NoSQL-рішення, як MongoDB чи Redis.

Також кросплатформна сумісність ускладнює розробку, оскільки платформа має працювати однаково стабільно на настільних комп’ютерах, планшетах і смартфонах, що вимагає адаптивного дизайну та оптимізації для різних апаратних конфігурацій. Ці технічні виклики потребують значних

ресурсів і ретельного планування для забезпечення стабільності та якості взаємодії.

Незважаючи на різноманітність підходів та викликів пов’язаних з розробкою, усі ці проєкти мають спільний недолік – обмежену гнучкість у створенні персоналізованих просторів для малювання. Користувачі зазвичай змушені працювати на одному спільному полотні або обирати з обмеженого набору заздалегідь створених кімнат. Вони не можуть налаштувати розмір полотна, доступні кольори, кількість учасників чи частоту оновлення пікселів, що обмежує їхню творчу свободу та можливості для організації власних заходів. Наприклад, організатору складно створити приватне полотно для тематичного конкурсу чи обмежити доступ для певної групи користувачів. Ця проблема стримує розвиток спільнот, які могли б створювати унікальні проєкти чи проводити заходи, як-от освітні воркшопи чи командні змагання. Крім того, відсутність інструментів модерації ускладнює контроль за діями учасників, що може призводити до вандалізму чи конфліктів на полотні. Наприклад, без належної модерації зловмисники можуть знищувати спільні малюнки, що демотивує учасників і знижує привабливість платформи. Нове рішення могло б включати інструменти для призначення модераторів, фільтрації дій користувачів і захисту контенту, що підвищило б якість взаємодії.

Отже, виникає потреба у новій платформі, яка б поєднувала переваги існуючих сервісів і додавала гнучкість у створенні кастомних кімнат. Таке рішення могло б дозволити користувачам налаштовувати параметри полотна, такі як його розмір, палітру кольорів, частоту розміщення пікселів і рівень доступу (приватний чи публічний). Наприклад, організатори могли б створювати кімнати для спільного малювання на тему мистецтва, історії чи поп-культури, залучаючи учасників із різними інтересами. Додаткові функції, як-от чат чи голосові канали, подібні до Discord, полегшили б координацію між учасниками. Технічно платформа могла б використовувати WebSocket для миттєвої синхронізації даних і хмарні сервери для обробки великих обсягів запитів. Крім того, аналітика дій користувачів, як-от статистика внеску чи теплові карти активності, могла б

додати елемент гейміфікації, підвищуючи залученість. Таке рішення відкрило б нові можливості для творчості, соціальної взаємодії та організації подій у реальному часі, усуваючи обмеження сучасних платформ.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є розробка інноваційного вебдодатку, який забезпечує інтерактивну взаємодію користувачів із віртуальним полотном у реальному часі, надаючи можливості для створення кастомізованих кімнат, піксельного малювання, координації спільної творчості та соціальної взаємодії. Додаток має поєднувати гнучкість налаштувань, високу продуктивність і безпеку, щоб відповідати сучасним вимогам до масштабованості та користувацького досвіду. Платформа спрямована на подолання обмежень існуючих аналогів, таких як r/place, шляхом впровадження функцій для створення персоналізованих творчих просторів, інтеграції соціальних інструментів і забезпечення стабільної роботи при високих навантаженнях.

Для досягнення цієї мети необхідно виконати такі завдання:

- проаналізувати предметну область та аналогічні рішення;
- визначити вимоги до розроблюваної системи;
- спроєктувати архітектуру системи;
- обрати технології для розробки;
- провести розробку вебдодатку;
- провести тестування;
- розробити базу даних;
- реалізувати синхронізацію даних в реальному часі;
- розробити захисні механізми для вебдодатку.

Висновки до розділу 1

У цьому розділі було проаналізовано роль обробки та передачі інформації в реальному часі для інтерактивних вебдодатків, які є важливою частиною сучасного цифрового середовища. Розглянуто приклади платформ, таких як Discord, Facebook і Viber, що ефективно застосовують технології реального часу для забезпечення швидкої взаємодії з користувачами. Особливу увагу приділено проєкту r/place як прикладу масової онлайн-співпраці на основі простої, але потужної механіки піксельного малювання.

Аналіз аналогів r/place (Pixelcanvas.io, OurWorldOfPixels, Pixelplace.io, Gartic.io, Drawception) дозволив визначити їхні переваги – доступність, гейміфікацію, підтримку реального часу – та виявити обмеження: нестачу гнучкості, обмежені параметри налаштувань полотна, проблеми з модерацією і продуктивністю. Це засвідчує потребу у створенні нового рішення, що поєднує плюси попередників і усуває їхні недоліки.

На основі аналізу сформульовано завдання для створення сучасного вебдодатку для спільного піксельного малювання в реальному часі. Такий застосунок має передбачати підтримку кастомних кімнат із можливістю налаштування розміру полотна, палітри, частоти розміщення пікселів, доступу, а також вбудовані чати.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Перед початком розробки будь-якого проєкту надзвичайно важливо виконати проєктування програмної системи та визначити її функціональні й нефункціональні вимоги. Цей етап суттєво спрощує подальшу реалізацію, оскільки розробник отримує чітке уявлення про структуру системи, її компоненти та очікувану поведінку.

Для моделювання системи було обрано мову UML (Unified Modeling Language) – поширений і зрозумілий інструмент для візуалізації об’єктно-орієнтованих програмних систем.

Окрім цього, було застосовано RESTful API-модель [18], яка дозволяє описати архітектуру взаємодії між клієнтською та серверною частинами застосунку. REST підходить для побудови масштабованих, зрозумілих і ефективних систем передачі даних, де кожен запит визначається HTTP-методом і відповідним маршрутом.

Таким чином, комбінація UML-діаграм і REST API-моделі дозволяє забезпечити повне логічне охоплення внутрішньої структури системи та зовнішніх взаємодій, що робить ці підходи найбільш доцільними для розробки даного програмного продукту.

Хоча UML включає велику кількість типів діаграм, у межах цього проєкту було створено діаграму класів, яка відображає основні сутності системи та зв’язки між ними, а також діаграму варіантів використання (use case), що демонструє сценарії взаємодії користувачів з системою. Діаграму класів зображено на рисунку 2.1.

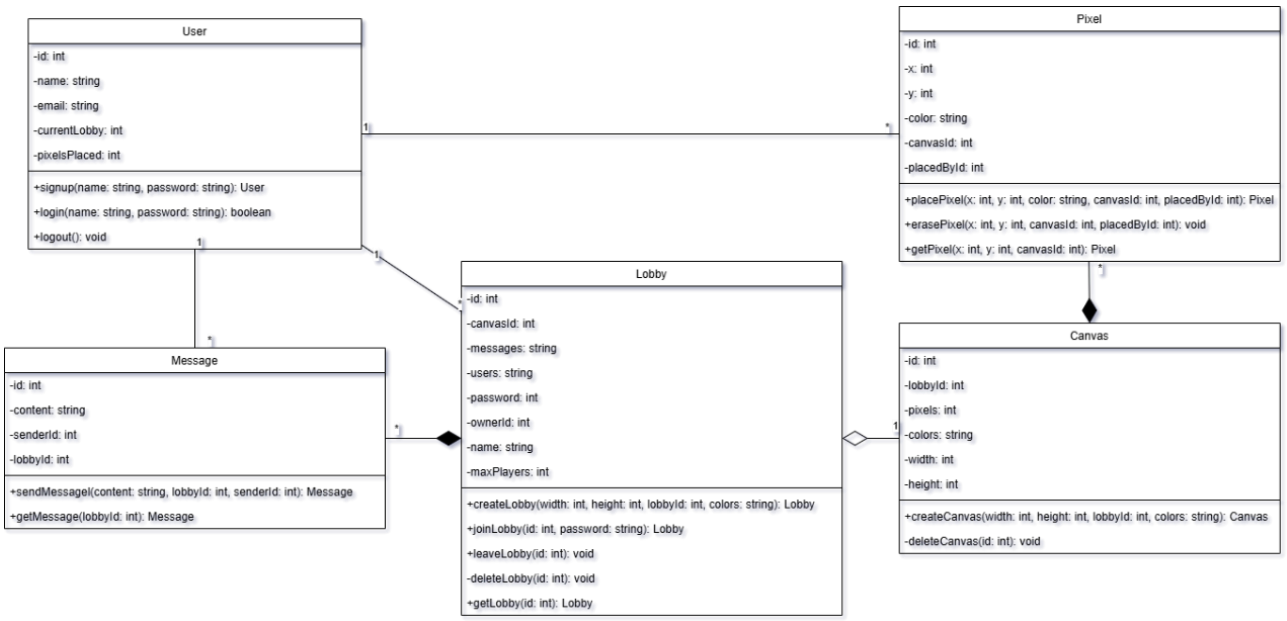

Рисунок 2.1 – Діаграма класів розроблюваної системи

На діаграмі можна побачити п’ять основних класів: User, Lobby, Canvas, Pixel та Message. Кожен з них містить в собі атрибути та методи, які характеризують клас, а також зв’язки з один з одним.

Клас користувача (User) має такі дані як: унікальний номер(id), ім’я(name), пошту(email) та пароль(password), а також актуальну кімнату в якій він знаходиться (currentLobby). Перші 4 атрибута відповідають за успішне проходження автентифікації користувача, останнє ж значення потрібне для навігації користувача. Наприклад якщо користувач вийде з додатку, а потім повернеться, ми будемо знати яку сторінку варто відобразити. Клас користувача має такі методи як реєстарція (signup), вхід (login) та вихід (logout).

Клас Lobby (кімната) відповідає за організацію окремого середовища, у якому користувачі можуть взаємодіяти між собою та спільно змінювати полотно. Кожне лобі має унікальний ідентифікатор (id), назву (name), пароль, що одночасно слугує як прапорець приватності (password, а також посилання на користувача-власника (owner). Такий зв’язок дозволяє реалізувати додаткові функції адміністрування кімнати, наприклад, видалення, запрошення або зміна налаштувань. Методи, пов’язані з лобі, включають створення кімнати (createLobby), приєднання до неї (joinLobby) та видалення або вихід

(leaveLobby), а також функції отримання кімнати (getLobby) та видалення (deleteLobby).

Клас Canvas (полотно) – центральна частина лобі, візуальний простір, який користувачі можуть змінювати. Полотно містить ідентифікатор (id), прив’язку до відповідного лобі (lobbyId), ширину та висоту (width, height), набір дозволених кольорів (colors).

Клас Pixel (піксель) відображає найменший елемент полотна. Кожен піксель має координати (x, y), колір (color), автора зміни (placedBy) та посилання на існуюче полотно (canvasId). Зв’язок з класом Canvas дає змогу чітко ідентифікувати розташування пікселя. Серед функцій класу пікселя є його розміщення (placePixel), видалення (erasePixel) та отримання (getPixel).

Клас Message (повідомлення) реалізує текстову комунікацію між користувачами в межах одного лобі. Повідомлення містить автора (senderId), кімнату (lobbyId) та зміст (content). Це дає змогу створити повноцінний чат, синхронізований із поточними діями користувачів на полотні. Основні методи – надсилання (sendMessage) та отримання (getMessage).

Діаграма класів відображає основну структуру системи, її ключові сутності та взаємозв’язки між ними, ілюструє об’єктно-орієнтований підхід до проєктування системи, який забезпечує логічну структуру, розмежування відповідальностей та можливість подальшого розширення функціональності. Така модель є основою для подальшої реалізації архітектури та розробки коду, а також спрощує аналіз поведінки системи в різних сценаріях.

Для повного розуміння поведінки системи з точки зору кінцевого користувача доцільно застосувати діаграму варіантів використання (use case diagram). Вона дозволяє візуально представити основні сценарії взаємодії користувача з програмним продуктом, визначити ролі користувачів (акторів) та відповідні функції, які їм доступні. Такий підхід допомагає виявити критичні точки взаємодії, уточнити вимоги до інтерфейсу та логіки системи.

Діаграма зображена на рисунку 2.2, включає як базові дії, пов’язані з автентифікацією, так і взаємодію з лоббі, полотном, обміном повідомленнями та малюванням у реальному часі.

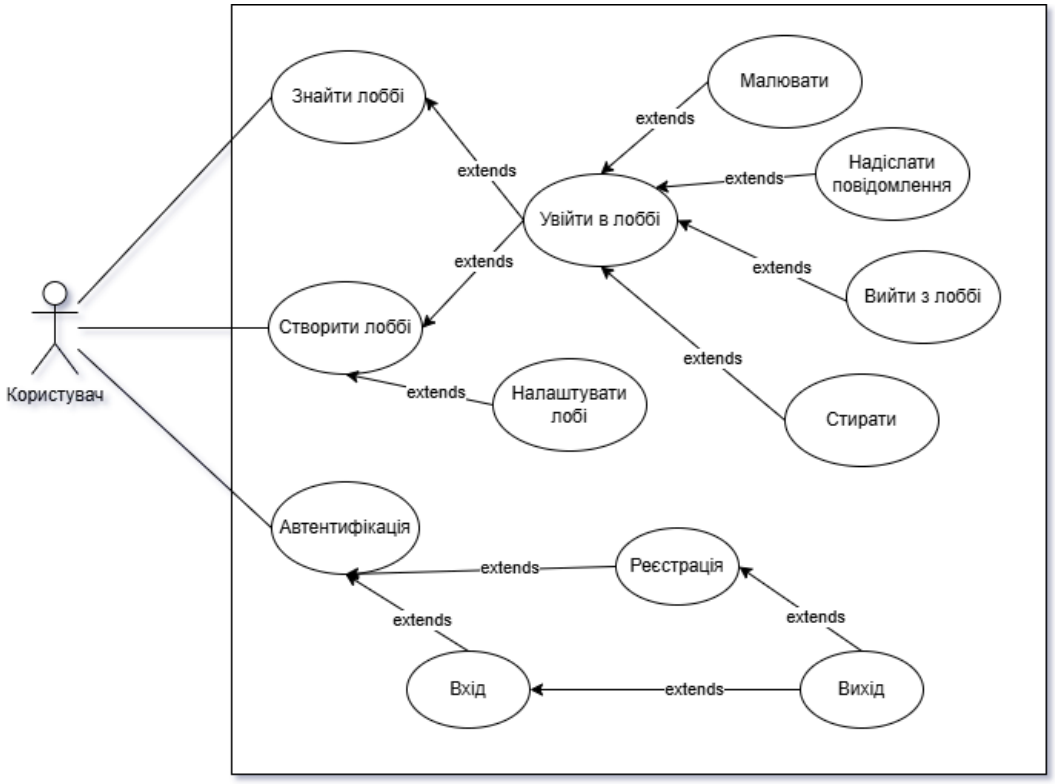

Рисунок 2.2 – Діаграма варіантів використання

Взаємодія між клієнтською (frontend) та серверною (backend) частинами додатку реалізується за допомогою архітектурного підходу REST API, який забезпечує структурований та стандартизований обмін даними. Для цього визначено набір кінцевих точок (endpoint-ів), що охоплюють основні функціональні можливості системи, зокрема:

– /api/auth/signup – реєстрація нового користувача;
– /api/auth/login – авторизація користувача;
– /api/auth/logout – вихід із системи;
– /api/lobby/create – створення нової кімнати;
– /api/lobby/join – приєднання до наявної кімнати;
– /api/pixel/place – розміщення пікселя на полотні;

– /api/message/send – надсилання повідомлення до чату кімнати;

– /api/canvas/{id} – отримання інформації про конкретне полотно.

Під час взаємодії користувача з інтерфейсом, особливо у процесі малювання та творчої діяльності, надзвичайно важливо забезпечити максимальну зосередженість та мінімізувати сторонні відволікання користувача. Саме тому при розробці дизайну вебдодатку був обраний мінімалістичний та інтуїтивно зрозумілий підхід. Макет головної сторінки зображено на рисунку 2.3.

Рисунок 2.3 – Макет головної сторінки.

Застосування стриманої, нейтральної кольорової палітри для елементів керування дозволяє зберегти візуальну легкість інтерфейсу та не відволікати користувача від основного – процесу малювання. Такий підхід допомагає створити мінімалістичний, але водночас функціональний дизайн, у якому кожен елемент має своє логічне місце.

Крім того, активне використання іконок замість текстових підписів дозволяє зменшити візуальне навантаження та зробити інтерфейс більш

універсальним. Це значно спрощує адаптацію дизайну для пристроїв з малими екранами, таких як смартфони чи планшети, зберігаючи при цьому зручність користування та доступ до всіх функцій додатку.

Втім, варто враховувати, що не всі користувачі однаково позитивно сприймають стримані кольори інтерфейсу – деякі можуть віддати перевагу яскравішим тонам або мати потребу в більш індивідуалізованому візуальному оформленні. Крім того, популярність темної теми серед користувачів зумовлює необхідність реалізації перемикання між світлою та темною версіями інтерфейсу.

У зв’язку з цим доцільно передбачити окрему сторінку налаштувань зовнішнього вигляду, де користувач зможе обрати бажану тему, змінити кольорову палітру інтерфейсу або навіть налаштувати вигляд полотна для малювання відповідно до власних уподобань. Такий функціонал покращує користувацький досвід і робить додаток більш доступним для ширшої аудиторії. Саме таку функцію буде відігравати кнопка Profile, де буде знаходитись інформація про користувача, його статистика та можливість змінювати кольорову тему.

Кнопка Lobby приведе користувача до сторінки з існуючими кімнатами в якій він зможе приєднатися або ж створити власну. Макет сторінки кімнат зображено на рисунку 2.4.

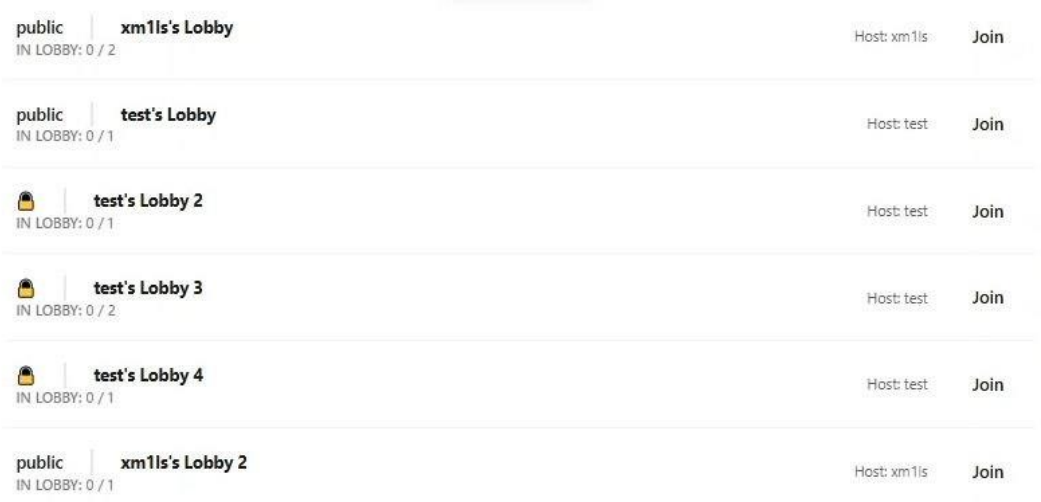

Рисунок 2.4 – Макет сторінки з кімнатами

Під час створення власної кімнати (лобі) користувачеві необхідно заповнити спеціальну форму, яка слугує для налаштування основних параметрів сесії. Зокрема, користувач вказує назву кімнати, що слугує її ідентифікатором серед інших, а також має можливість задати пароль у випадку, якщо бажає зробити кімнату приватною та обмежити доступ стороннім користувачам.

Додатково у формі передбачено вибір розміру полотна – це дозволяє адаптувати середовище під заплановану активність (наприклад, малювання у великій групі або індивідуальні сесії). Ще одним параметром є максимальна кількість учасників, що дає змогу регулювати навантаження на систему та зберегти якість взаємодії. Макет форми для створення лобі представлено на рисунку 2.5.

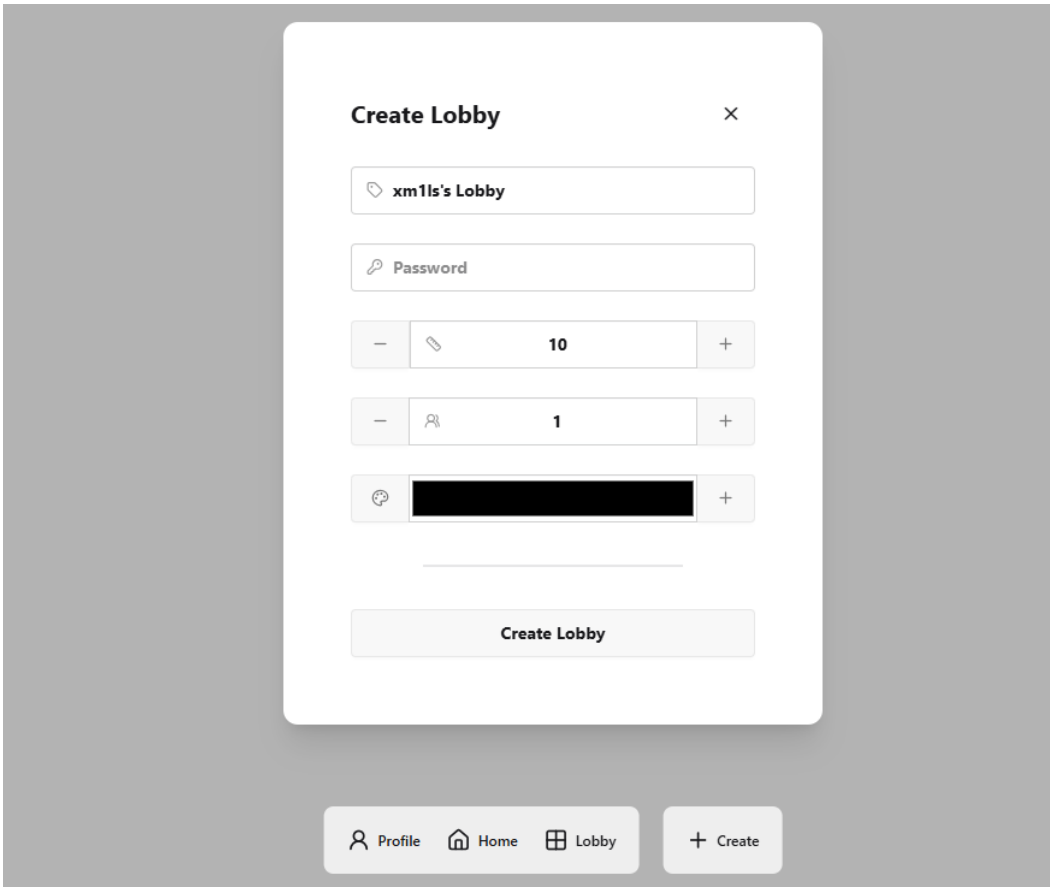

Рисунок 2.5 – Макет форми для створення кімнати

Форма також передбачає можливість створення власної палітри кольорів. Це дозволяє користувачеві індивідуалізувати візуальний стиль сесії, обравши

набір кольорів, які будуть доступні всім учасникам під час малювання. Такий підхід розширює творчі можливості та сприяє створенню унікального стилю для кожної окремої кімнати.

Після успішного приєднання до кімнати користувач отримує доступ до основної функціональності – малювання на спільному полотні. Для забезпечення зручності та ефективності творчого процесу інтерфейс надає інтуїтивно зрозумілі інструменти. Зокрема, користувач може скористатися кистю, яка дозволяє вибрати доступний колір з палітри та наносити пікселі на полотно; гумкою, призначеною для видалення помилкових або зайвих пікселів; а також піпеткою, за допомогою якої можна швидко вибрати колір безпосередньо з полотна для подальшого використання. Такий набір інструментів робить процес взаємодії з додатком зручним навіть для новачків. Використання інструменту кисті для вибору доступного кольору зображено на рисунку 2.6.

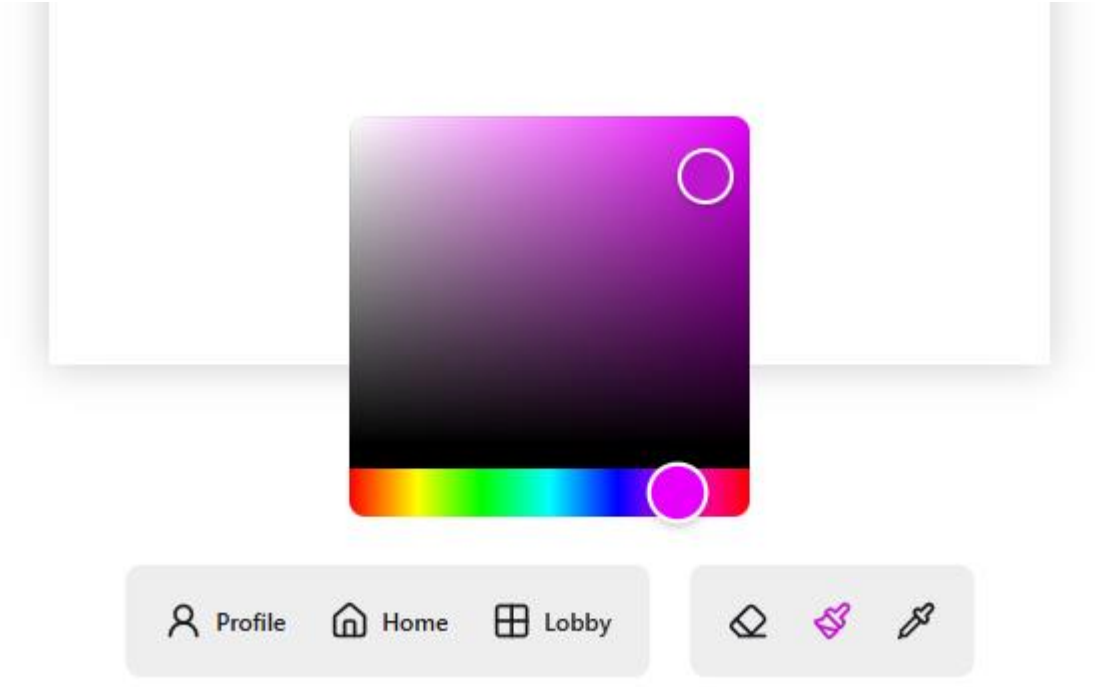

Рисунок 2.6 – Використання інструменту Кисть

Після того, як користувач обрав колір і визначив координати для розміщення пікселя на віртуальному полотні, критично важливо надати йому зворотний зв’язок про успішність виконаної дії. Це підтвердження відіграє

ключову роль у забезпеченні позитивного користувацького досвіду, особливо в ситуаціях, коли затримка між діями (наприклад, через обмеження частоти розміщення пікселів) може досягати значних значень.

Сповіщення допомагає користувачу переконатися, що його дія була зареєстрована системою, знижує відчуття невизначеності та підвищує довіру до платформи. Це реалізується через використання візуальних і текстових сповіщень, відомих як «тости», які призначені для інформування користувача про певні події. У цьому випадку вони повідомляють про статус дії, зокрема про те, чи був піксель успішно розміщений, чи виникла помилка. На рисунку 2.7 представлено макет сповіщення, що підтверджує успішне виконання дії.

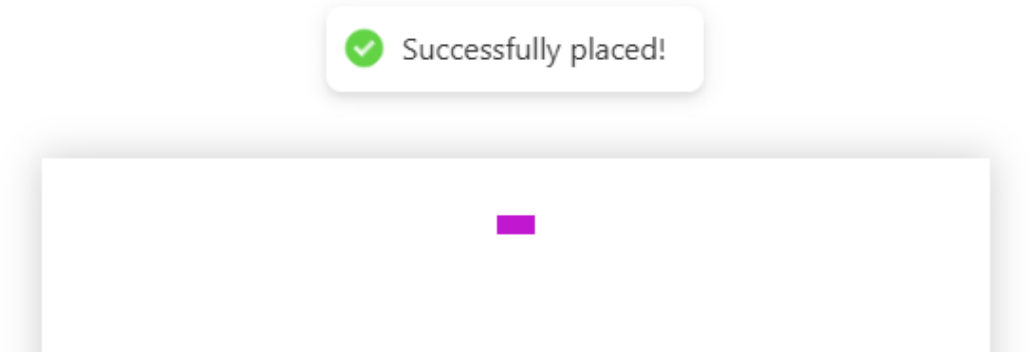

Рисунок 2.7 – Сповіщення користувача про статус дії

Окрім необхідності створення моделі даних і розробки зручного інтерфейсу вебдодатку, ключовим аспектом є забезпечення безпеки інформації користувачів. Недостатній захист може спричинити витік конфіденційних даних, що спричинить шкоду як користувачам, так і репутації проєкту, підірвавши довіру до платформи.

Тому, впровадження надійного механізму автентифікації є вкрай важливим. Використання JSON Web Tokens (JWT) [6] забезпечує безпечне та ефективне керування сесіями, дозволяючи швидко перевіряти права доступу користувачів без збереження їхнього стану на сервері. Завдяки компактності та універсальності, JWT-токени ідеально підходять для обміну даними між клієнтом і сервером у цій системі.

Також надзвичайно важливим є шифрування паролів для захисту від несанкціонованого доступу. Зберігання паролів у незашифрованому вигляді є неприпустимим. Для цього застосовуються сучасні алгоритми хешування, такі як bcrypt [2], із додаванням унікальної «солі», що значно підвищує стійкість до атак навіть у разі взлому бази даних.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для виконання завдань, визначених на етапі формування вимог до вебдодатку, що є клоном r/place із підтримкою кастомних лобі, необхідно ретельно оцінити доступні інструменти розробки, їхні сильні та слабкі сторони. Основною вимогою системи є забезпечення малювання в реальному часі з можливістю створення користувацьких кімнат, що передбачає стабільну двосторонню комунікацію між сервером і клієнтами вебдодатку, а також сервер із підтримкою базових CRUD-операцій [13], зокрема створення даних. З огляду на це, для реалізації серверної частини обрано Node.js [2] як середовище виконання JavaScript-коду та Express як фреймворк для побудови API, що забезпечують гнучку обробку запитів і ефективне створення серверної логіки.

Вибір Node.js обґрунтований його асинхронною моделлю, яка дозволяє обробляти велику кількість одночасних підключень із мінімальними затримками, що є критично важливим для синхронного малювання та чату. Express [1] доповнює Node.js, надаючи зручний інструментарій для створення RESTful API, обробки маршрутів і middleware для автентифікації, валідації та логування. Однак, щоб повною мірою реалізувати вимоги, необхідно також обрати інструменти для фронтенду, комунікації та управління даними в реальному часі.

Для фронтенду використовується React [11], бібліотека JavaScript, яка завдяки компонентному підходу та віртуальному DOM забезпечує швидке створення динамічних і адаптивних інтерфейсів. React дозволяє ефективно реалізувати полотно для малювання, чат і модальні вікна для налаштування лобі,

а досвід роботи з ним сприяє швидкій розробці. Для стилізації застосовується Tailwind CSS [16], утилітарний фреймворк, що прискорює створення адаптивного дизайну через класи, і DaisyUI [4], бібліотека компонентів на основі Tailwind, яка додає готові елементи, такі як кнопки та алерти, забезпечуючи консистентний вигляд. Zustand, легковагова бібліотека для управління станом, обрана для зберігання даних про вибраний колір, активне лобі та координати пікселя, завдяки простому API . Axios використовується для HTTP-запитів до сервера, забезпечуючи зручну роботу з API, наприклад, для створення лобі чи автентифікації. React Hot Toast додає сповіщення (тости) для інформування користувачів про успішні дії чи помилки, такі як розміщення пікселя чи порушення затримки.

Для синхронізованої комунікації обрано Socket.IO [5], бібліотеку на основі WebSocket, яка забезпечує двосторонній зв’язок із підтримкою кімнат для ізоляції лобі і автоматичним перепідключенням [14]. Socket.IO ідеально інтегрується з Express і React. Порівняно з альтернативами, такими як чистий WebSocket, який вимагає ручної реалізації багатьох функцій, Firebase або Pusher, що є платними хмарними сервісами з обмеженою кастомізацією, SignalR, прив’язаний до .NET, MQTT, складний для вебдодатків, чи Server-Sent Events, які підтримують лише однонаправлений зв’язок, Socket.IO пропонує оптимальний баланс продуктивності, гнучкості та економічності.

Для управління даними використовується MongoDB [17], NoSQL-база даних, яка завдяки гнучкості JSON-подібних документів ідеально підходить для зберігання пікселів, лобі та чатів. Mongoose [8], ODM-бібліотека, забезпечує схеми та валідацію, спрощуючи роботу з MongoDB. Joi застосовується для валідації вхідних даних, наприклад, координат пікселя чи параметрів лобі, зменшуючи ризик помилок. Безпека даних реалізується через bcryptjs [2], бібліотеку для хешування паролів із сіллю, що захищає облікові дані користувачів.

Алгоритм розміщення пікселя включає валідацію автентифікації, перевірку затримки між діями, координат і кольору через Joi [7], оновлення даних у

MongoDB і трансляцію події через Socket.IO. Полотно моделюється як двовимірна матриця пікселів із метаданими (колір, автор, час), що зберігається як документи в MongoDB. CRUD-операції для лобі реалізуються через Express, дозволяючи створювати, читати, оновлювати та видаляти дані. Діаграми класів і компонентів, створені в UML, описують сутності (User, Lobby, Canvas, Pixel) та їх взаємодію, забезпечуючи чітке бачення архітектури.

Крім того, важливим аспектом є вибір середовища розробки, і в цьому проєкті використовується Visual Studio Code (VS Code), яке відіграє ключову роль у забезпеченні продуктивності та зручності роботи розробника.

Обраний стек (MERN, Tailwind, DaisyUI, Axios, Zustand, React Hot Toast, Socket.IO, Mongoose, Joi, bcryptjs) відповідає вимогам малювання в реальному часі та кастомних лобі, враховує досвід розробника та забезпечує продуктивність, безпеку й масштабованість. Socket.IO виділяється як ключовий інструмент для двосторонньої комунікації, а Node.js і Express створюють надійний сервер для CRUD-операцій, що робить цей набір оптимальним для проєкту.

Висновки до розділу 2

У другому розділі проаналізовано вимоги до вебдодатку – покращеного аналога r/place із підтримкою кастомних лобі. Визначено функціональні та нефункціональні вимоги, змодельовано архітектуру системи та обґрунтовано вибір технологій для синхронізованого малювання, створення кімнат, двосторонньої комунікації та CRUD-операцій.

Проєктування з використанням UML дозволило візуалізувати структуру системи (діаграма класів із сутностями User, Lobby, Canvas, Pixel, Message) та сценарії взаємодії (use case-діаграма). REST API забезпечує ефективний обмін даними між клієнтом і сервером.

Інтерфейс побудовано за принципами мінімалізму з адаптивним дизайном, інтуїтивними інструментами малювання та можливістю налаштувань (теми, палітра, параметри лобі). Безпека реалізована через JWT і bcrypt.

Для реалізації обрано MERN-стек з додатковими бібліотеками (Tailwind CSS, DaisyUI, Axios, Zustand, Socket.IO тощо). Socket.IO забезпечує стабільну роботу в реальному часі та перевершує альтернативи за гнучкістю й продуктивністю. Visual Studio Code обрано як ефективне середовище розробки.

Таким чином, сформовано надійну, безпечну та гнучку архітектуру вебдодатку для спільного піксельного малювання в реальному часі.

РОЗДІЛ 3
РОЗРОБКА ІНТЕРАКТИВНОГО ВЕБДОДАТКУ ДЛЯ СПІЛЬНОГО МАЛЮВАННЯ З ВИКОРИСТАННЯМ SOCKET.IO

3.1 Практична реалізація об’єкта проектування

Для розробки програмного забезпечення обрано технологічний стек MERN, який охоплює MongoDB як базу даних для зберігання інформації, Express.js як фреймворк для серверної частини, React для побудови інтерфейсу користувача та Node.js як платформу для виконання JavaScript на сервері. Для забезпечення взаємодії в реальному часі застосовано Socket.IO, що дозволяє здійснювати двосторонній обмін даними між клієнтом і сервером.

На рисунку 3.1 зображено структуру готового проєкту з використанням стеку MERN та Socket.IO для досягнення функцій в реальному часі.

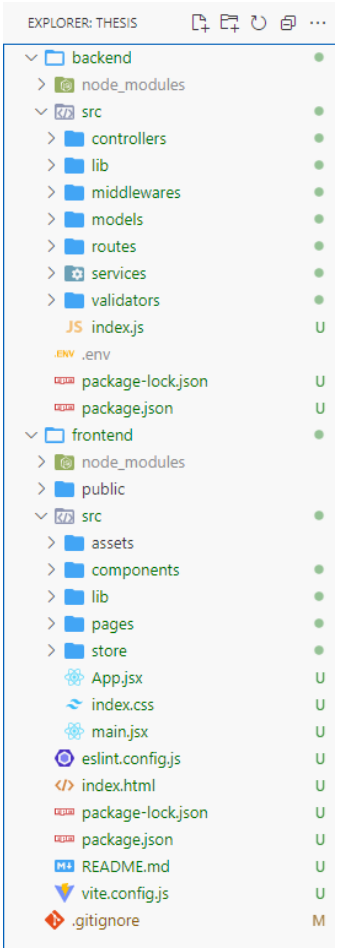

Рисунок 3.1 – Структура розроблюваного проєкту

З самого початку розробки було створено дві окремі папки які логічно розділяють проект на дві частини – частину бекенду та фронтенду. Таким чином досягається розділення відповідальностей, зручність та ефективність розробки.

Основним файлом у папці бекенду є index.js, який відіграє центральну роль у серверній логіці. Він забезпечує належне функціонування бекенду, ініціалізує ключові об’єкти, такі як початкове полотно, і дозволяє клієнтському додатку взаємодіяти з основними модулями системи через REST API. Файл відповідає за підключення до бази даних, запуск сервера та журналювання всіх дій у консолі.

Для реалізації REST API було створено три основні директорії: controller, middleware та service. Така структура забезпечує логічне розділення відповідальностей у коді, що відповідає принципам архітектурного патерну MVC (Model-View-Controller), адаптованого до серверного середовища.

Контролери (controller) відповідають за обробку HTTP-запитів. Вони отримують запит від клієнта, передають його далі до відповідного сервісу та повертають результат у вигляді відповіді. Контролери фактично є "посередниками" між зовнішнім світом і внутрішньою логікою застосунку.

Сервіси (service) інкапсулюють бізнес-логіку. Вони виконують основні дії, як-от взаємодія з моделями, перевірка даних, збереження у базу даних тощо. Це дозволяє зробити код більш модульним і придатним для тестування, а також уникнути дублювання логіки.

Проміжні обробники (middleware) використовуються для попередньої обробки запитів – перевірки автентифікації, авторизації, логування або валідації вхідних даних. Вони виконуються до того, як запит потрапляє до контролера.

Ось для прикладу, коли користувач вперше реєструється на платформі, він заповнює спеціальну форму та надсилає POST-запит на відповідну кінцеву точку бекенду /api/auth/signup. Цей запит містить основні дані, необхідні для створення облікового запису: ім’я користувача, електронну пошту та пароль.

На рисунку 3.2 зображено розроблену форму для реєстрування.

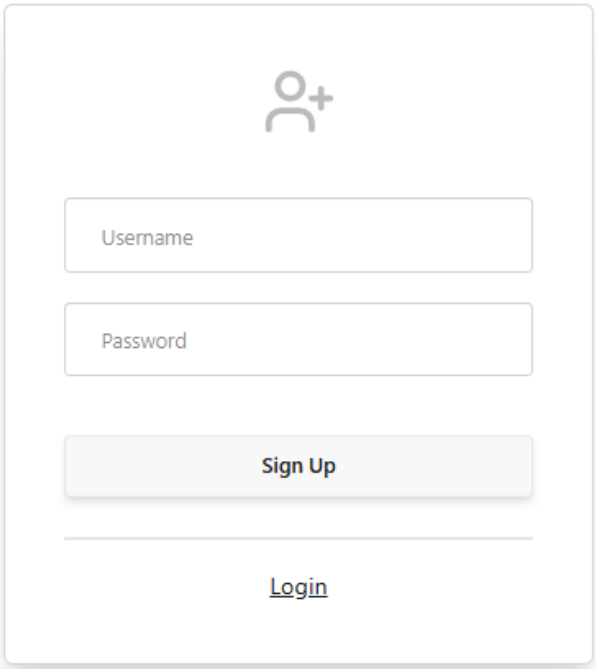

Рисунок 3.2 – Форма для реєстрації користувача

Далі запит проходить кілька етапів обробки, кожен з яких реалізований у відповідному шарі структури проєкту. Middleware – перший етап обробки. Тут виконується валідація вхідних даних: перевірка заповнення обов’язкових полів, довжини пароля тощо. Якщо дані не відповідають вимогам, користувач отримує відповідь з помилкою 400 (Bad Request). Service – на цьому рівні реалізована бізнес-логіка. Якщо валідація пройдена, відбувається перевірка, чи не існує вже користувача з таким email. Далі, за допомогою моделі User, створюється новий запис у базі даних. Пароль шифрується, а сам користувач зберігається. І нарешті controller – відповідає за формування остаточної відповіді. У разі успішного створення користувача, повертається код 201 (Created). Якщо під час створення виникає помилка (наприклад, на рівні бази даних), вона логується, а клієнту надсилається повідомлення з відповідним кодом помилки (500 Internal Server Error). На рисунку 3.3 зображено послідовність дій під час того коли користувач реєструється.

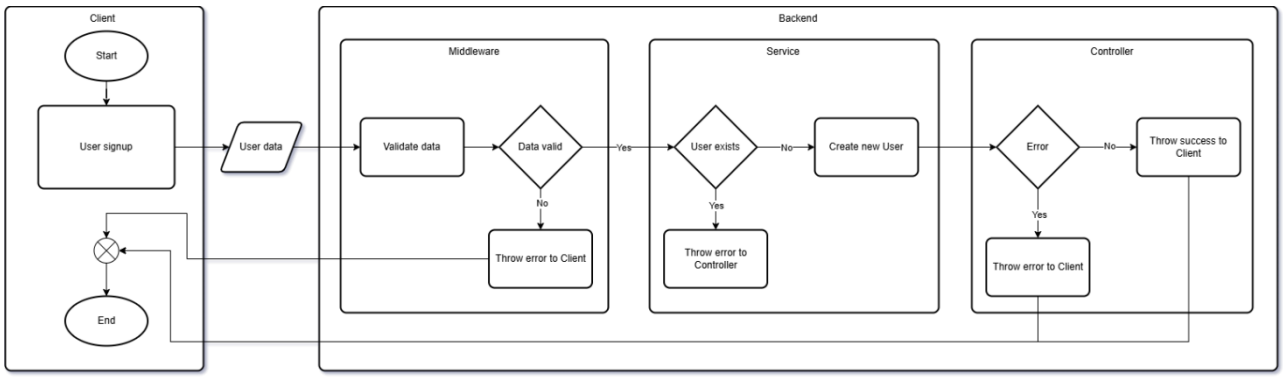

Рисунок 3.3 – Діаграма послідовності дій при реєстрації

Для того щоб визначити сутності з якими працює backend та валідації даних на етапі middleware було створено папки models та validators, які містять файли з описом основних об’єктів бази даних, а тобто користувача, кімнату, полотно, піксель та повідомлення. Кожна з моделей відповідає окремій колекції в базі даних MongoDB та реалізована з використанням Mongoose – бібліотеки, що забезпечує об’єктно-документне моделювання для Node.js.

Хоча Mongoose сам по собі виконує базову валідацію типів, для додаткової перевірки на етапі отримання запиту використовується бібліотека Joi. Це дозволяє виявляти помилки ще до передачі даних у базу, покращуючи стабільність та безпеку системи. Наприклад, перед реєстрацією нового користувача Joi-схема перевіряє, щоб поле name не було порожнім, email мав правильний формат, а пароль відповідав мінімальним вимогам (довжина, складність тощо) [9]. Таке розділення дозволяє відокремити логіку перевірки запитів (у middleware) від логіки збереження даних (у моделях), дотримуючись принципу single responsibility.

Дотримуючись того ж принципу у дерикторії під назвою routes знаходяться логічно розділені шляхи (ендпоїнти) REST API. Реалізовані вони з використанням функціоналу express.Router(). Кожен маршрут відповідає окремому модулю функціоналу, так наприклад файл з назвою auth.route.js – маршрути для реєстрації, входу та виходу користувачів. Саме у цьому місці поєднуються раніше створенні middleware, controller та service. Приклад такого маршруту для надсилання повідомлення у чат кімнати зображено на лістингу 3.1

Лістинг 3.1 – Код маршруту для надсилання повідомлення

```
const router = express.Router();
const sendMessageMiddlewares = [
    protectRoute,
    validateMessageMiddleware
];
router.post('/send', sendMessageMiddlewares, sendMessageController);
export default router;
```

кінець лістингу 3.1

Таким чином, коли користувач посилається на ендпоїнт для надсилання повідомлення, його запит проходить через послідовність middleware-функцій, визначених у масиві sendMessageMiddlewares. Першою виконується protectRoute, яка перевіряє автентифікацію користувача, дозволяючи доступ лише авторизованим. Далі викликається validateMessageMiddleware, яка відповідає за перевірку правильності структури та змісту повідомлення (наприклад, наявність тексту, допустима довжина тощо).

Лише після успішного проходження обох middleware запит передається у sendMessageController, який реалізує основну логіку – збереження повідомлення до бази даних, обробку можливих помилок і формування відповіді клієнту. Реалізація бізнес-логіки відбувається за рахунок делегування даних контролером до відповідного сервісу, який створює та зберігає нове повідомлення у базі даних та сповіщає про це контролер.

Такий підхід забезпечує чітке розділення відповідальностей, підвищує надійність API та дозволяє масштабувати логіку в майбутньому, додаючи нові перевірки або функціональність без зміни основного контролера.

Разом з цим усі файли у проєкті мають чітку, узгоджену схему іменування, яка полегшує навігацію та підтримку коду. Зокрема, файли, що знаходяться у директорії routes, названі з префіксом .route (наприклад, auth.route.js, message.route.js), що одразу дає зрозуміти, що вміст файлу стосується налаштування маршрутів REST API. Аналогічно, у папці models усі файли мають суфікс .model (наприклад, user.model.js, lobby.model.js), що вказує на те, що файл містить схему моделі бази даних, створену за допомогою Mongoose. Така

конвенція іменування підвищує читабельність структури проєкту, дозволяє швидко знаходити потрібні файли за функціональністю та сприяє злагодженій командній роботі, особливо у великих проєктах.

Ще однією важливою директорією у теці backend є папка lib, у якій зберігаються допоміжні файли, що не належать безпосередньо до контролерів, моделей або маршрутів, але відіграють ключову роль у функціонуванні застосунку. Зокрема, у цій директорії міститься модуль для підключення до бази даних MongoDB – він ініціалізує з'єднання з базою, відслідковує помилки підключення та забезпечує повторне підключення у разі збоїв. Також тут зосереджено весь функціонал, пов'язаний із WebSocket-з'єднанням за допомогою Socket.IO – обробка подій підключення, надсилання пікселів у реальному часі, робота з кімнатами тощо. Окрім цього, lib містить утиліти – допоміжні функції, які використовуються в різних частинах проєкту.

Такий розподіл дозволяє зберігати логічну чистоту основних частин програми, ізолюючи повторно використовуваний або технічний код у окремій структурованій директорії.

Окрім папок, у директорії backend також розташовані важливі конфігураційні файли, зокрема package.json, .env та package-lock.json.

Файл package.json є основним описом проєкту – у ньому вказані всі залежності, які використовуються в бекенді, а також версії цих бібліотек, назва проєкту, скрипти для запуску, розробки чи збірки (наприклад, start, dev, lint) та інші метадані. Він дозволяє швидко відновити середовище проєкту за допомогою команди npm install. Приклад коду скрипту запуску у режимі розробки зображено на лістингу 3.2.

Лістинг 3.2 – Скрипт для запуску проєкту в режимі розробки

```
"scripts": {
    "dev": "nodemon src/index.js"
  }
```

кінець лістингу 3.2

Файл .env використовується для зберігання конфіденційної або змінної інформації, яка залежить від середовища – таких як URI для бази даних, секретні ключі для токенів, порти для запуску серверу тощо. Його зміст не завантажується в репозиторій, що дозволяє зберігати приватність даних та конфігурацій.

Файл package-lock.json автоматично генерується після встановлення пакетів через npm. Він фіксує точні версії встановлених залежностей, включаючи вкладені пакети, що забезпечує стабільність та передбачуваність при запуску проєкту на інших пристроях або у продакшн-середовищі.

Разом ці файли забезпечують правильну роботу, масштабованість і безпечне налаштування бекенд-частини додатку.

Основним осередком вихідного коду клієнтської частини є директорія src, яка містить усю логіку, компоненти, сторінки та конфігурації React-додатку, забезпечуючи його модульність і масштабованість. У ній розташовано кілька ключових піддиректорій і файлів, що відповідають за чітке розділення функціоналу.

Директорія components зберігає повторно використовувані візуальні елементи інтерфейсу, такі як кнопки, поля введення чи модальні вікна. Ця структура сприяє чистоті коду та спрощує його повторне використання. У lib зібрано допоміжні функції та утиліти, зокрема логіку взаємодії з API через бібліотеку Axios. Сторінки додатку, логічно прив’язані до маршрутів, містяться в pages/, де кожна сторінка, наприклад, головна, кімната чи сторінка реєстрації, об’єднує необхідні компоненти для відображення. Глобальний стан додатку керується через бібліотеку Zustand у директорії store, що забезпечує ефективне управління станом без зайвої складності.

Головний компонент додатку, app.jsx, відповідає за маршрутизацію, підключення провайдерів і застосування глобальних стилів. Точка входу, main.jsx, використовує ReactDOM.createRoot() для монтування app.jsx у DOM, а також підключає Tailwind CSS, глобальні стилі та провайдери, такі як Zustand. Файл index.css об’єднує глобальні стилі та директиви Tailwind.

Файл index.html слугує основним HTML-шаблоном Vite-проєкту, містить елемент для монтування React-інтерфейсу та дозволяє налаштовувати мета-теги чи favicon. Конфігурація проєкту визначається в package.json, де вказані залежності, зокрема react і react-dom для створення інтерфейсу, axios для HTTP-запитів, zustand для управління станом, tailwindcss і daisyUI для стилювання, а також скрипти й мета-інформація. Файл package-lock.json фіксує точні версії залежностей, забезпечуючи стабільність збірки.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Наступним важливим етапом розробки є тестування інформаційної системи. На цьому етапі перевіряється працездатність окремих компонентів, взаємодія між ними, стійкість до помилок, а також зручність використання для кінцевого користувача.

У ході реалізації було застосовано декілька підходів до тестування, що дозволило забезпечити стабільну та надійну роботу системи. Зокрема, активно використовувалося ручне тестування інтерфейсу користувача. Було перевірено реакцію системи на основні дії: кліки, наведення, введення тексту, перемикання темної та світлої теми, використання інструментів малювання (кисть, гумка, піпетка), а також зміну параметрів лобі. Це дозволило виявити та усунути візуальні та логічні помилки ще на ранніх етапах розробки.

Крім того, для перевірки роботи серверної частини активно використовувався інструмент Postman. З його допомогою здійснювалося тестування REST API: перевірялася коректність обробки HTTP-запитів, відповідність даних специфікації, адекватність відповідей сервера та правильність кодів статусу. Наприклад, при спробі реєстрації з некоректними або відсутніми даними сервер повертає відповідь із кодом 400 (Bad Request), а у випадку успішної реєстрації – код 201 (Created) з відповідним повідомленням. Також було протестовано обробку авторизації, створення та підключення до лобі, надсилання повідомлень, малювання пікселів і вихід із системи.

Особливу увагу приділено тестуванню граничних випадків: наприклад, повторна реєстрація з уже існуючим email, спроба доступу до захищеного ресурсу без токена, або надсилання пікселя в недозволений колір. Це дозволило переконатися, що система поводиться очікувано у виняткових ситуаціях. Приклад тестування ендпоінтів за допомогою Postman зображено на рисунку 3.4.

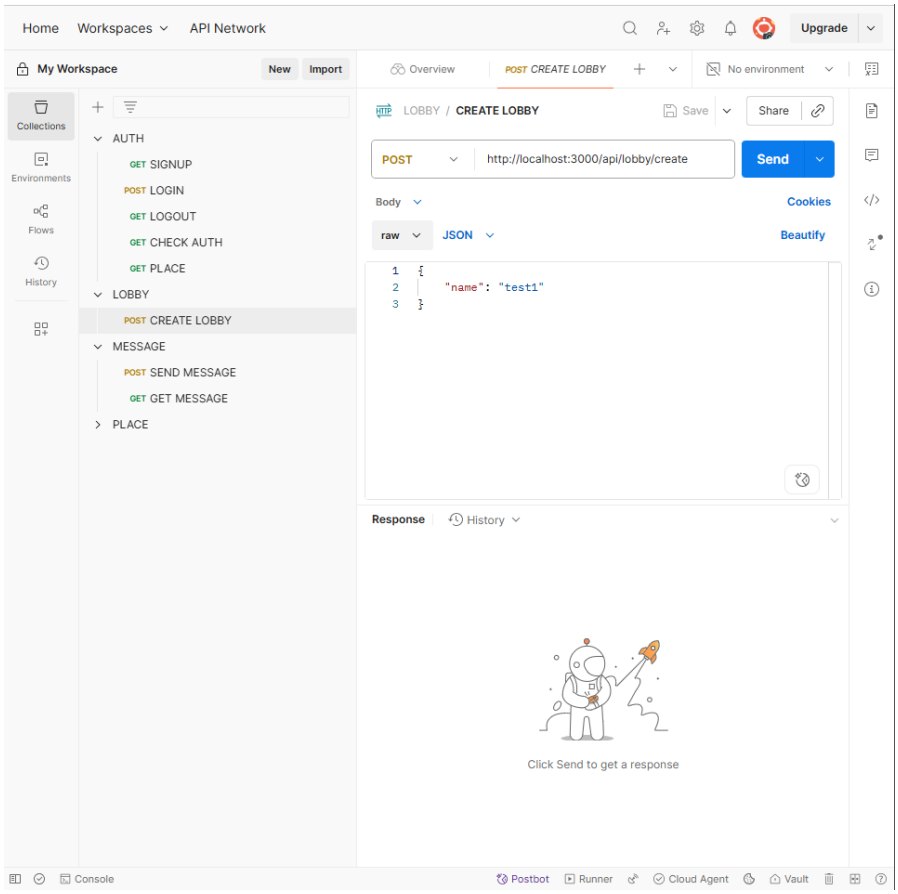

Рисунок 3.4 – тестування API за допомогою Postman

Окрім перевірки працездатності REST API, важливою складовою тестування було забезпечення коректної роботи WebSocket-з’єднань. Для цього проводилась перевірка стабільності підключень, обміну повідомленнями в реальному часі, а також реакції системи на відключення чи повторне підключення клієнтів. Тестування здійснювалось шляхом детального логування подій сокета – таких як підключення, відправлення та отримання даних, а також обробка можливих помилок. Це дозволило своєчасно виявити й усунути

критичні помилки у логіці взаємодії між клієнтами та сервером. Приклад коду логування подій можна побачити у лістингу 3.3.

Лістинг 3.3 – Логування сокетів при з'єднанні користувача

```
io.on("connection", (socket) => {

    console.log(`User connected. Total users:
${io.engine.clientsCount}`);

console.log("A user connected", socket.id);
}
```

кінець лістингу 3.3

Також варто згадати про такий інструмент як Nodemon, який відіграв важливу роль у тестуванні. Це утиліта для середовища Node.js, яка автоматично слідкує за змінами в коді проєкту та перезапускає сервер при виявленні змін у файлах. Такий підхід значно оптимізував процес розробки, оскільки дозволяв розробнику зосередитись на логіці додатку, не витрачаючи час на постійний ручний перезапуск сервера після кожної правки. Nodemon спрацьовував миттєво після збереження змін у будь-якому з файлів бекенду – будь то зміни в контролерах, сервісах або маршрутах. Це дозволяло швидко побачити результат внесених змін у браузері або в інструментах тестування. Особливо корисним Nodemon був під час роботи з API, де потрібно було багаторазово тестувати запити та відповідь сервера на різні сценарії. Крім того, у випадках, коли система мала справу з авторизацією, перевіркою токенів, доступом до захищених маршрутів або обробкою виняткових ситуацій, Nodemon давав змогу одразу бачити результати без затримки.

Під час тестування клієнтської частини активно використовувалися інструменти розробника у браузері, а також спеціалізовані розширення, зокрема React Developer Tools та Pesticide. React Developer Tools дозволив детально аналізувати структуру React-компонентів, їхні стани та властивості, що було особливо корисним при налагодженні логіки інтерфейсу та відстеженні змін у глобальному сховищі стану (zustand). Розширення Pesticide накладає контури на

всі HTML-елементи, що допомагає швидко виявляти проблеми з розміткою, відступами або накладанням елементів один на одного. Це значно полегшило перевірку адаптивності верстки та точність візуального розміщення компонентів на сторінці.

У процесі тестування було виявлено та усунено низку недоліків. Зокрема, при передачі даних від клієнта до сервера важливою виявилася попередня перевірка введеної інформації ще до взаємодії з базою даних. Для цього використовувалась бібліотека Joi, яка дозволяє здійснювати гнучку валідацію структури даних. Такий підхід значно знижує навантаження на базу, оскільки відсікає некоректні або потенційно шкідливі запити ще на етапі обробки валідаційного middleware. Наприклад, якщо користувач вводить неправильний формат електронної пошти або надсилає порожні поля, система одразу повертає відповідну помилку, не звертаючись до бази даних. Це покращує загальну стабільність системи, пришвидшує роботу сервера та підвищує безпеку.

Також були сформовані мінімальні системні вимоги до запуску та використання системи. Для серверної частини потрібно мати встановлений Node.js (версія 18 або вище), MongoDB, 4 ГБ оперативної пам’яті та будь-яку сучасну ОС. Клієнтська частина працює стабільно в останніх версіях браузерів (Chrome, Firefox, Edge) при мінімальному дозволі 1280×720 пікселів.

3.3 Розробка бази даних

Одним із важливих етапів створення інформаційно-комп’ютерної системи є проєктування та реалізація бази даних. У рамках розробки було обрано MongoDB – гнучку документно-орієнтовану NoSQL базу даних, яка чудово підходить для вебзастосунків із динамічною структурою даних. Для взаємодії з базою на стороні коду використовувалась бібліотека Mongoose, що є обгорткою для MongoDB та забезпечує декларативне створення моделей, а також зручний механізм валідації та маніпуляції документами.

Початковим кроком було створення нового кластера на офіційному сайті MongoDB за допомогою платформи MongoDB Atlas. Було обрано безкоштовний тарифний план, який дозволяє створити кластер із обмеженими, але цілком придатними для розробки ресурсами. Після цього створено окрему базу даних і додано користувача з відповідними правами доступу. Вигляд бази даних можна побачити на рисунку 3.5.

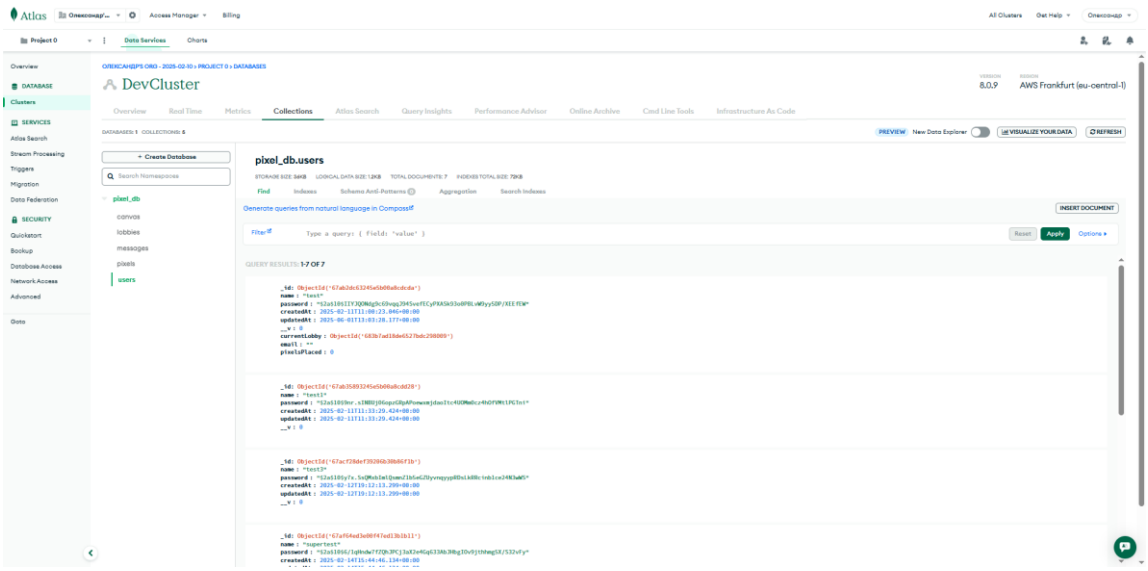

Рисунок 3.5 – Вигляд створеної бази даних

Для безпечного підключення до кластеру з локального середовища розробки було додано IP-адресу розробника до білого списку.

Після створення кластера та налаштування доступу необхідно організувати підключення до бази даних із застосунку. Код підключення до бази даних через додаток можна побачити у лістингу 3.4.

Лістинг 3.4 – Підключення до бази даних

```
import mongoose from 'mongoose'

export const connectDB = async () => {
    try {
        const conn = await mongoose.connect(process.env.MONGODB_URI);

        console.log("Connected successfuly:", conn.connection.host)
    } catch(error) {
```

```
        console.log("DB failed to connect:", error)
    }
}
```

кінець лістингу 3.4

Використовується бібліотека mongoose, яка асинхронно встановлює з’єднання з MongoDB на основі рядка підключення (connection string), що зберігається у змінній оточення MONGODB_URI. Такий підхід дозволяє не зберігати конфіденційну інформацію, таку як ім’я користувача, пароль або адреса кластера, безпосередньо в коді. У випадку успішного з’єднання виводиться повідомлення про успішне підключення з зазначенням хоста, до якого встановлено зв’язок. Якщо ж під час підключення виникає помилка (наприклад, через неправильний рядок з’єднання чи відсутність мережевого доступу), вона логуються в консоль

Наступним етапом стала розробка структури бази даних на рівні застосунку. За допомогою Mongoose було створено окремі схеми (schemas) для кожної основної сутності системи: користувача (User), кімнати (Lobby), полотна (Canvas), пікселя (Pixel) та повідомлення (Message). Кожна схема визначає обов’язкові поля, типи даних, значення за замовчуванням та зв’язки між сутностями, якщо це потрібно. Наприклад, Pixel посилається на Canvas, Message – на User та Lobby, а Lobby містить масив ідентифікаторів учасників. Приклад схеми для повідомлення наведено у лістингу 3.5.

Лістинг 3.5 – Схема сутності Message

```
import mongoose from "mongoose";
const messageSchema = mongoose.Schema(
    {
        content: {
            type: String,
            required: true
        },
        sender: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true
        },
```

```
        lobby: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Lobby",
            default: null
        },
        canvas: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Canvas",
            default: null
        },
    },
    { timestamps: true }
)
const Message = mongoose.model("Message", messageSchema);
export default Message;
```

кінець лістингу 3.5

На основі цих схем створювались моделі, які безпосередньо відповідають колекціям у MongoDB. Через ці моделі здійснюються всі CRUD-операції з базою: створення документів, оновлення, пошук та видалення.

Щоб уникнути зайвих запитів до бази даних при обробці некоректних вхідних даних, у системі реалізовано попередню валідацію запитів за допомогою бібліотеки Joi. Цей підхід дозволяє ще до етапу звернення до моделі Mongoose перевірити структуру, типи даних, допустимі значення, мінімальну та максимальну довжину, а також наявність обов’язкових полів у тілі запиту.

Joi-схеми створюються на основі існуючих моделей і використовуються як частина middleware. У лістингу 3.6 наведено приклад схеми повідомлення Joi на основі схеми mongoose.

Лістинг 3.6 – Схема сутності Message з використанням Joi

```
const messageValidationSchema = Joi.object({
    content:
Joi.string().min(1).max(60).required().pattern(messageContentPattern).mes
sages({
        "string.min": "Message content must be at least 1 characters
long",
        "string.max": "Message content should not exceed 60 characters
long",
        "any.required": "Message content is required",
        "string.base": "Message content must be a string",
```

```
        "string.pattern.base": "Invalid content format for message"
    }),
    sender: Joi.string().required().pattern(objectIdPattern).messages({
        "string.pattern.base": "Invalid ObjectId format for placedBy",
        "any.required": "User is required",
    }),
    lobby: Joi.string().pattern(objectIdPattern).optional().allow(null,
"").messages({
        "string.pattern.base": "Invalid ObjectId format for lobby",
    }),
    canvas: Joi.string().pattern(objectIdPattern).optional().allow(null,
"").messages({
        "string.pattern.base": "Invalid ObjectId format for canvas",
    })
})
```

кінець лістингу 3.6

У разі, якщо валідація не проходить, система не здійснює жодних подальших дій (не викликає сервісний шар і не звертається до бази), а натомість повертає читабельне повідомлення про помилку з поясненням, що саме пішло не так. Такий підхід дозволяє значно знизити навантаження на базу даних, підвищити безпеку системи (захист від шкідливих або неочікуваних даних), а також забезпечити кращий користувацький досвід за рахунок миттєвого зворотного зв’язку. Крім того, оскільки Joi має гнучкий API для створення схем, його легко масштабувати та підтримувати в міру розвитку проєкту.

3.4 Синхронізація даних в реальному часі за допомогою Socket.IO

Однією з ключових особливостей створеної інформаційної системи є підтримка функціоналу в реальному часі, що забезпечується за допомогою бібліотеки Socket.IO. Цей інструмент дозволяє встановлювати постійне двостороннє з’єднання між клієнтом і сервером через WebSocket-протокол, що значно підвищує інтерактивність та швидкодію системи, особливо для спільного малювання, що є ключовим функціоналом системи.

Процес малювання починається на клієнтській стороні. Коли користувач клікає по полотну, формується об’єкт пікселя – координати, колір, розмір, а

також ідентифікатор лобі (або полотна), до якого він належить. Цей об’єкт передається на сервер через HTTP-запит (REST API), а не безпосередньо через сокети. Приклад коду який реалізує запит до серверу можна побачити у лістингу 3.7.

Лістинг 3.7 – Запит до серверу для малювання на полотні

```
placePixel: async (x, y) => {
        try {
            const { name, canvasId } = get();

            const userId = useAuthStore.getState().authUser._id

            const pixelPromise =
axiosInstance.post(`/canvas/${name}/place`, {
                x,
                y,
                color: get().color,
                placedBy: userId,
                canvas: canvasId
            })

            toast.promise(
                pixelPromise,
                {
                    loading: "Placing...",
                    success: (res) => res?.data?.message || "Successfully
placed!",
                    error: (err) => err?.response?.data?.message ||
"Failed to place"
                }
            )

            const { data: { pixel } } = await pixelPromise;

            if (pixel) {
                const socket = useAuthStore.getState().socket
                const lobbyId = lobbyStore.getState().currentLobby?._id
?? "main";
                socket.emit("place", { pixel, lobbyId });
            }
        } catch (error) {
            console.error(error.message)
        }
    }
```

кінець лістингу 3.7

Це зроблено з міркувань безпеки та контролю: вся валідація та збереження змін має проходити через сервер, перш ніж інформація буде поширена іншим учасникам. На сервері дані про новий піксель проходять валідацію через Joi. У разі успіху вони зберігаються в колекції MongoDB за допомогою моделі Pixel. Збереження здійснюється у відповідній колекції, зв'язаній з конкретним Canvas. Цей крок є обов'язковим – синхронізація через сокети виконується лише після успішного збереження пікселя. Це гарантує узгодженість даних між усіма користувачами й базою. Код створення нового пікселя у базі даних продемонстровано у лістингу 3.8.

Лістинг 3.8 – Створення нового пікселя у базі даних

```
export const placePixelService = async (pixelData, userId, canvasId) => {

    const pixel = new Pixel({
        ...pixelData,
        placedBy: userId,
        canvas: canvasId
    })

    return await pixel.save();
}
```

кінець лістингу 3.8

Після успішного збереження пікселя на сервері, ініціюється надсилання події place за допомогою бібліотеки Socket.IO. Ця подія містить усі необхідні дані про новий піксель: координати, колір, ідентифікатор полотна та інформацію про користувача. Подія транслюється лише в межах відповідної кімнати (room), до якої приєднані користувачі певного лобі. У результаті кожен учасник обраного лобі миттєво бачить зміну – новий піксель з'являється на полотні в режимі реального часу, без потреби в ручному оновленні сторінки або періодичному опитуванні сервера (polling). Діаграму послідовності створення нового пікселя показано на рисунку 3.6.

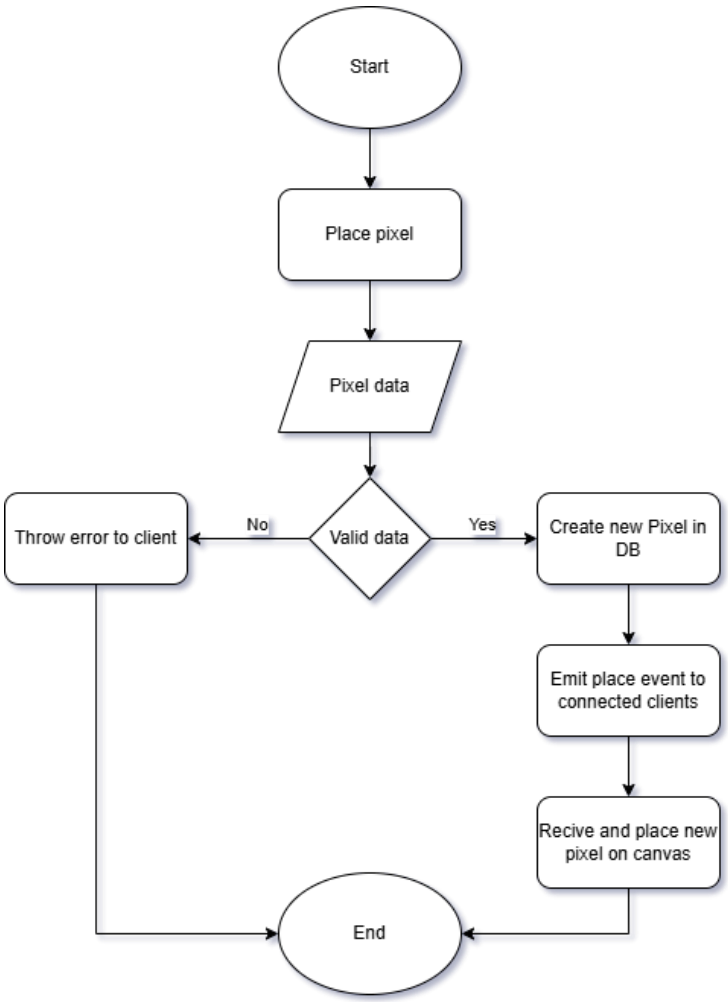

Рисунок 3.6 – Діаграма послідовності для події place

Також Socket.IO дозволяє логічно організовувати користувачів у кімнати. Коли користувач заходить у певне лобі, він надсилає сокет-подію join-lobby, у відповідь на яку сервер додає сокет до відповідної кімнати. У лістингу 3.9 реалізовано подію для підключення користувача до кімнати.

Лістинг 3.9 – Підключення до кімнати через Socket.IO

```
socket.on('join-lobby', (lobbyId) => {
    socket.join(lobbyId);
});}
```

кінець лістингу 3.9

Це дозволяє ізолювати події – пікселі, що малюються в одному лобі, не передаються іншим користувачам поза межами цієї кімнати.

Аналогічно реалізовано функціонал стирання (erase). Подія надсилається на сервер, в базі знаходиться відповідний піксель та видаляється. Після цього відбувається трансляція erase події до учасників відповідного лобі.

3.5 Захист інформаційно-комп'ютерної системи

Забезпечення захисту інформаційно-комп'ютерної системи є критично важливим етапом при розробці сучасних вебдодатків, особливо коли йдеться про інтерактивну систему з багатьма користувачами, доступом до бази даних та обміном подіями в реальному часі. У межах реалізованого проєкту було впроваджено низку заходів, спрямованих на захист персональних даних користувачів, обмеження доступу до функціоналу, валідацію запитів та запобігання зловживанням.

Під час реєстрації нового користувача пароль не зберігається у відкритому вигляді. Замість цього, перед записом у базу даних пароль шифрується за допомогою бібліотеки bcrypt. Алгоритм використовує сіль (salt), яка додається до пароля для унеможливлення атак типу rainbow table. У разі спроби входу користувача, пароль, введений при логіні, порівнюється з зашифрованим значенням у базі за допомогою методу bcrypt.compare.

Після успішної реєстрації або входу, користувач отримує JWT (JSON Web Token), який зберігається на клієнтській стороні (наприклад, у cookie) і додається до заголовків кожного наступного запиту до захищених ресурсів. На сервері реалізовано middleware, яке перевіряє валідність токена, а також отримує ідентифікатор користувача із його payload для подальшої роботи. Приклад коду для реєстрації користувача, хешування пароля та додавання JWT токена зображено у лістингу 3.10.

Лістинг 3.10 – Створення нового користувача

```
export const signup = async (req, res) => {
    const { name, password } = req.body
    try {
```

```
        if (password.length < 8) {
            return res.status(400).json({
                message: "Password minimum length is 8 characters"
            })
        }
        const user = await User.findOne({ name })
        if (user) return res.status(400).json({
            message: "User with this name already exists"
        })
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);
        const newUser = new User({
            name,
            password: hashedPassword
        })
        if (newUser) {
            generateToken(newUser._id, res)
            await newUser.save();
            res.status(200).json({
                _id: newUser._id,
                name: newUser.name,
                message: "User created"
            })
        }
        else {
            res.status(400).json({
                message: "Invalid user data"
            })
        }
    } catch (error) {
        console.log(error)
        res.status(500).json({
            message: "Something went wrong"
        })
    }
}
```

кінець лістингу 3.10

Це дозволяє реалізувати захищений доступ до маршруту, зберігати інформацію про сесію, а також уникати зайвих запитів до бази при кожному зверненні.

Окрім базової автентифікації, система має додаткові логіки обмеження, що підвищують захищеність додатку: Один користувач – одне лобі, тобто користувач не може приєднатись одразу до кількох лобі. Це реалізовано через зберігання поточного lobbyId у профілі користувача й перевірку при новій спробі

входу. Перевірка дозволених кольорів: при малюванні пікселя колір перевіряється на відповідність дозволеним значенням, щоб запобігти обхідним атакам або зміні DOM на клієнті. Лобі з паролями: система дозволяє створювати приватні лобі, які можуть мати пароль. Пароль хешується аналогічно до користувацьких паролів і зберігається у полі password моделі Lobby. При спробі входу до такого лобі користувач має ввести правильний пароль, інакше запит відхиляється. Захист від неавторизованого доступу: всі запити, пов’язані з малюванням пікселів, надсиланням повідомлень або створенням/видаленням лобі, проходять через middleware protectRoute, який перевіряє JWT і отримує дані користувача. Перевірка JWT токену зображено у лістингу 3.11.

Лістинг 3.11 – Валідація JWT токена

```
export const protectRoute = async (req, res, next) => {
  try {
    const token = req.cookies.jwt;

    if (!token) {

      return res.status(401).json({ message: "Unauthorized - No Token
Provided" });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    if (!decoded) {

      return res.status(401).json({ message: "Unauthorized - Invalid
Token" });

    }

    const user = await User.findById(decoded.userId).select("-password");

    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    req.user = user;

    next();

  } catch (error) {
```

```
    console.log("Error in protectRoute middleware: ", error.message);
    res.status(500).json({ message: "Internal server error" });
  }
};
```

кінець лістингу 3.11

Висновки до розділу 3

У розділі описано повний цикл реалізації системи для спільного редагування піксельного полотна в реальному часі. Сервер створено на основі Node.js і Express.js з чіткою архітектурою: маршрути, контролери, middleware. Клієнт реалізовано за допомогою React, Tailwind CSS і DaisyUI, що забезпечило сучасний адаптивний інтерфейс.

MongoDB із бібліотекою Mongoose використано для зберігання даних, структурованих з урахуванням зв’язків між користувачами, лобі, пікселями та повідомленнями. Валідація даних через Joi підвищила стабільність і надійність системи.

Socket.IO реалізує синхронізацію в реальному часі, дозволяючи ефективну взаємодію користувачів у межах окремих лобі через socket-кімнати. Це забезпечує масштабованість і точкову передачу подій.

У розробці активно застосовувалися інструменти для тестування та дебагу (React DevTools, Pesticide, nodemon), що прискорило виявлення помилок. Реалізована аутентифікація з JWT і bcrypt гарантує безпечне зберігання облікових даних. Додаткові механізми (захист лобі, контроль кольорів, обмеження активності) підвищують безпеку й контроль доступу.

Отже, створено функціональну, безпечну та зручну для користувача систему, готову до розгортання та подальшого розвитку.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено глибокий аналіз предметної області, пов’язаної зі створенням інтерактивних вебдодатків, зокрема – аналогів популярного сервісу r/place. Були розглянуті наявні реалізації подібних проєктів, визначено їхні сильні та слабкі сторони. Це дозволило виявити ключові функціональні можливості, які є очікуваними для кінцевих користувачів, та сформулювати загальні вимоги до функціоналу, зручності та ефективності системи.

На основі проведеного аналізу були визначені чіткі вимоги до розроблюваної системи, серед яких – підтримка багатокористувацької взаємодії в реальному часі, створення окремих лобі, можливість малювання на спільному полотні, синхронізація змін між клієнтами, безпечна автентифікація та управління правами доступу. Особливу увагу було приділено ергономіці інтерфейсу, мінімізації затримок при взаємодії та зручності використання системи.

Проєктування архітектури системи відбувалося з урахуванням модульності, масштабованості та можливості подальшого розширення функціоналу. Для моделювання структури використовувались діаграми UML, що дозволило чітко визначити взаємозв’язки між сутностями, зокрема користувачами, лобі, повідомленнями, пікселями та полотнами. Вибрана архітектура MVC допомогла структурувати додаток і забезпечити зручне розділення відповідальностей.

У ході роботи було обґрунтовано вибір технологій для розробки. Стек MERN (MongoDB, Express.js, React.js, Node.js) забезпечив високу продуктивність, зручність розробки та можливість ефективної побудови SPA-додатка. Socket.IO став оптимальним рішенням для реалізації комунікації в реальному часі. Додатково використовувалися Redis для кешування та зберігання тимчасових даних, Tailwind CSS для швидкої верстки та JWT для безпечної автентифікації.

Розробка вебдодатку здійснювалася відповідно до поставлених вимог. Було реалізовано функціонал реєстрації, входу в акаунт, створення лобі, малювання та стирання пікселів у реальному часі, ведення історії подій, відображення кількості учасників у лобі тощо. Особливу увагу було приділено ергономічності та адаптивності інтерфейсу, що зробило систему зручною для користувачів.

Тестування системи проводилося на різних рівнях: перевірка роботи REST API, валідація сокет-подій, ручне тестування UI/UX. Завдяки цьому було виявлено та усунено ряд логічних помилок і забезпечено стабільну роботу системи при підключенні великої кількості клієнтів. Для діагностики використовувалися стандартні інструменти логування та дебагу.

База даних була розроблена з урахуванням вимог до ефективного зберігання та швидкого доступу до інформації. Структура колекцій у MongoDB була оптимізована для зберігання пікселів, повідомлень, користувачів та інших сутностей. Забезпечено зв’язки між документами, а також реалізовано механізми індексації та агрегації для підвищення швидкодії.

Синхронізація даних у реальному часі була реалізована за допомогою бібліотеки Socket.IO. Кожна дія користувача (наприклад, малювання пікселя) ініціюється на клієнті, надсилається на сервер, де проходить валідація та обробка. Після цього подія транслюється до всіх учасників відповідного лобі. Таким чином забезпечено узгодженість стану полотна між усіма клієнтами.

Захист вебдодатку був реалізований на кількох рівнях. Паролі користувачів зберігаються у вигляді хешів за допомогою bcrypt. Для автентифікації використовується JWT, що дозволяє здійснювати захищений обмін даними між клієнтом і сервером. Додатково реалізовано валідацію кольорів, перевірку доступу до лобі, обмеження на входження до кількох лобі одночасно та інші логічні обмеження, що підвищують безпечність і цілісність системи.

Таким чином, поставлені цілі та завдання кваліфікаційної роботи були повністю досягнуті, а результатом став сучасний, стабільний та масштабований вебдодаток для спільного малювання в реальному часі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Express – Node.js web application framework. URL: https://expressjs.com/ (date of access: 23.01.2025).

2. GitHub – dcodeIO/bcrypt.js. Optimized bcrypt in JavaScript with zero dependencies, with TypeScript support. GitHub. URL: https://github.com/dcodeIO/bcrypt.js (date of access: 15.02.2025).

3. Index – Node.js v24.2.0 Documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/docs (date of access: 28.02.2025).

4. Install daisyUI as a Tailwind plugin – Tailwind CSS Components (version 5 update is here). DaisyUI. URL: https://daisyui.com/docs/install/ (date of access: 28.03.2025).

5. Introduction – Socket.IO. URL: https://socket.io/docs/v4/ (date of access: 15.02.2025).

6. JSON Web Token Introduction – jwt.io. URL: https://jwt.io/introduction (date of access: 15.02.2025).

7. Joi.dev. URL: https://joi.dev/ (date of access: 15.03.2025).

8. Mongoose ODM v8.15.1. URL: https://mongoosejs.com/ (date of access: 28.02.2025).

9. OWASP Top Ten – OWASP Foundation. OWASP Foundation, the Open Source Foundation for Application Security. URL: https://owasp.org/www-project-top-ten/ (date of access: 27.01.2025).

10. PixelCanvas.io. GitHub. URL: https://github.com/pixelcanvasio (date of access: 13.01.2025).

11. React. URL: https://react.dev/ (date of access: 28.02.2025).

12. Rytz R. Place Atlas. The 2017 r/place. URL: https://2017.place-atlas.stefanocoding.me/ (date of access: 13.01.2025).

13. Same-origin policy – Security – MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (date of access: 27.01.2025).

14. Socket.IO. GitHub. URL: https://github.com/socketio (date of access: 15.02.2025).

15. Staff. How We Built r/Place – Upvoted. Reddit Inc Homepage. URL: https://redditinc.com/blog/how-we-built-rplace (date of access: 13.01.2025).

16. Tailwind CSS – Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/ (date of access: 28.03.2025).

17. Team M. D. MongoDB Documentation – Homepage. MongoDB: The World's Leading Modern Database. URL: https://www.mongodb.com/docs/ (date of access: 20.01.2025).

18. Web API Design Best Practices – Azure Architecture Center. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (date of access: 27.01.2025).

19. WebSocket – Web APIs – MDN. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (date of access: 27.01.2025).

ДОДАТКИ

Додаток А – Підтвердження апробації результатів кваліфікаційної роботи

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ІТОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

СЕКЦІЯ 6. УПРАВЛІННЯ ПРОЕКТАМИ В ГАЛУЗІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.41

ТРАНСЛЮВАННЯ ІНФОРМАЦІЇ В РЕАЛЬНОМУ ЧАСІ У ВЕБ-ЗАСТОСУНКАХ

Хмільовський Олександр Миколайович
Луцький національний технічний університет, студент, бакалавр інженерії програмного забезпечення, hmilovskijoleksandr@gmail.com
Суринович Олена Миколаївна
Луцький національний технічний університет, к.т.н., доцент кафедри інженерії програмного забезпечення, sivom@ukr.net

AUTHORIZATION THROUGH THIRD-PARTY SERVICES IN SOCIAL NETWORKS

Khmilovsky Oleksandr Mykolayovych
Lutsk National Technical University, Student, Bachelor of Software Engineering, hmilovskijoleksandr@gmail.com
Surynovych Olena Mykolayivna
Lutsk National Technical University, PhD in Technical Sciences, Associate Professor of Software Engineering Department, sivom@ukr.net

Real-time collaborative platforms, such as online drawing applications, are becoming an essential part of modern digital interaction. These systems rely on instant communication between users and the server, making real-time data exchange and privacy protection critical. To minimize the risk of data leaks and ensure a secure user experience, it is important to implement efficient real-time communication protocols and enforce secure handling of user sessions. A well-structured architecture with real-time features not only improves interactivity but also strengthens trust and usability.

***Keywords**: real-time, data privacy, WebSocket, user interaction, security, collaborative systems, usability*

У сучасному цифровому середовищі користувачі очікують миттєвої реакції від веб-застосунків. Затримки навіть у кілька секунд можуть суттєво знизити задоволеність користувачів і створити враження нестабільності. Це особливо актуально для онлайн-чатів, багатокористувацьких ігор, платформ для спільної роботи, онлайн-дошок, інтерактивних інструментів для навчання, а також веб-додатків, орієнтованих на творчість у реальному часі. У таких системах важливо не лише швидко передавати дані, а й забезпечити їхню узгодженість між усіма учасниками сеансу. З огляду на ці вимоги, транслювання інформації в реальному часі стало одним із

ключових напрямів розвитку сучасних веб-технологій, адже дозволяє створювати інтерактивні продукти, що відповідають очікуванням користувачів.

Метою даної роботи була розробка багатокористувацького веб-застосунку для спільного малювання в реальному часі. Суть такого застосунку полягає в тому, що велика кількість користувачів може одночасно взаємодіяти з одним спільним полотном, змінюючи його пікселі у режимі реального часу. Це вимагає миттєвої синхронізації дій усіх учасників та узгодженого оновлення інтерфейсу. Для досягнення цієї мети використано технологію Socket.IO, яка забезпечує двосторонній обмін даними між клієнтом і сервером на основі WebSocket. Реалізація механізмів обмеження частоти змін (rate limiting) та обробки одночасного доступу до ресурсів, що дозволить підтримувати стабільну й безпечну роботу системи навіть за умов високого навантаження.

З технічної точки зору, транслювання інформації в реальному часі у веб-застосунках передбачає використання технологій, які забезпечують постійне з'єднання між клієнтом і сервером. Найпоширенішим підходом є застосування WebSocket – протоколу, що дозволяє обмінюватися даними в обох напрямках без необхідності повторного відкриття з'єднання. Для спрощення роботи з WebSocket у JavaScript-орієнтованих застосунках часто використовують спеціалізовані бібліотеки, зокрема Socket.IO, яка додатково забезпечує зворотну сумісність із HTTP, автоматичне повторне з'єднання та підтримку кластеризації. Такі інструменти дозволяють миттєво передавати дані між користувачами – від повідомлень у чаті до оновлень інтерфейсу в багатокористувацьких системах.

У рамках власного проєкту для реалізації передачі інформації в реальному часі я використав бібліотеку Socket.IO [2], яка базується на технології WebSocket [1] і дозволяє організувати постійне двостороннє з'єднання між клієнтом і сервером. Основне завдання полягало в тому, щоб забезпечити миттєву синхронізацію змін, які користувачі вносили на канвасі – інтерактивному полі для малювання. На сервері було налаштовано Socket.IO-сервер, який приймав вхідні події від

клієнтів, зокрема події зміни пікселів. Кожна подія містила точні координати пікселя, новий вибраний колір, а також ідентифікатор відповідного канвасу або лоббі, у якому перебував користувач. Після обробки цієї події сервер одразу ж транслював її всім іншим користувачам, підключеним до відповідного каналу (namespace або room у Socket.IO), що забезпечувало миттєве відображення змін без перезавантаження сторінки або запитів до API. Така логіка дозволила реалізувати ефект колективного малювання, коли кілька користувачів бачать зміни в режимі реального часу. Для захисту системи було реалізовано перевірку автентичності кожного користувача перед обробкою подій, а також обмеження на частоту надсилання змін, щоб уникнути перевантаження сервера.

Водночас з усіма перевагами, впровадження технологій реального часу має і свої виклики. Найперше – це збільшене навантаження на сервер і мережу при великій кількості одночасних підключень, особливо якщо система погано оптимізована. Також виникає потреба в ретельному контролі доступу та автентифікації, адже транслювання даних напряму до клієнтів у реальному часі може створювати нові вектори для атак або витоку інформації. Ще один недолік – складність у тестуванні та відлагодженні таких систем, оскільки баги можуть проявлятись лише при взаємодії кількох клієнтів одночасно або при певних часових збігах.

Підсумовуючи, трансляція даних у реальному часі є надзвичайно ефективним інструментом для створення сучасних, інтерактивних веб-застосунків, які відповідають очікуванням користувачів. Її впровадження дозволяє значно покращити користувацький досвід за рахунок швидкої взаємодії, безперервної синхронізації та ефекту "живої" присутності в сервісі. Це особливо актуально в умовах високої конкуренції, коли швидкість реакції системи прямо впливає на залученість користувача, кількість активних сесій. Хоча технології реального часу вимагають додаткових ресурсів і складніші у впровадженні порівняно зі стандартними REST-застосунками,

вони відкривають новий рівень можливостей – від спільного редагування документів до інтерактивного командного управління або колективної творчості. У результаті можна стверджувати, що транслювання інформації в реальному часі – це не лише тренд, а й обґрунтована інвестиція в якість, швидкодію та конкурентоспроможність цифрових рішень.

Список використаних джерел

1. RFC 6455: The WebSocket Protocol. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc6455 (date of access: 12.05.2025).
2. Introduction | Socket.IO. Socket.IO. URL: https://socket.io/docs/v4/ (date of access: 12.05.2025).

УДК 004.9

ПОБУДОВА МАСШТАБОВАНОЇ ІНФРАСТРУКТУРИ TELEGRAM-БОТА ЗА ДОПОМОГОЮ AWS-СЕРВІСІВ

Шведа Валентин Олександрович
Луцький національний технічний університет, магістр спеціальності інженерія програмного забезпечення, valentine.shveda@gmail.com
Повстяна Юлія Славомирівна
Луцький національний технічний університет, к.т.н., доцент кафедри інженерії програмного забезпечення, yuliapovstyana@ukr.net

BUILDING A SCALABLE TELEGRAM BOT INFRASTRUCTURE USING AWS SERVICES

Shveda Valentyn Oleksandrovich
Lutsk National Technical University, Master's degree in software engineering, valentine.shveda@gmail.com
Povstiana Yullia Slavomirivna
Lutsk National Technical University, Ph.D., Associate Professor, Department of Software Engineering, yuliapovstyana@ukr.net

This paper explores the development of a scalable infrastructure for a Telegram bot designed for booking services using Amazon Web Services (AWS).

СЕРТИФІКАТ

Даний сертифікат засвідчує, що

Хмільовський Олександр Миколайович

брав(-ла) участь у

X Міжнародній науково-практичній конференції
з проблем вищої освіти і науки
"Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"
загальним обсягом 15 годин (0,5 кредитів ECTS),

яка проводилася 23-24 травня 2025 року у м.Луцьк
на базі Луцького національного технічного університету

Ректор ЛНТУ **Ірина ВАХОВИЧ**

Кафедра цифрових освітніх технологій

Кафедра інженерії програмного забезпечення

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

№ ІТОНВ-2025-164

Програма конференції: https://itonv.lntu.edu.ua/files/2025/program-itonv-2025.pdf