Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка системи "Віртуальна дошка" для системи дистанційного навчання»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

	_____ (підпис студента)	/Артур САЄНКО/ (Ім'я ПРІЗВИЩЕ)
Керівник:	_____ (підпис)	/ст. викл. Сергій САМОЙЛОВ/ (посада, Ім'я ПРІЗВИЩЕ)
Нормоконтролер:	_____ (підпис)	/доц. Олена КУРОП'ЯТНИК/ (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____ ____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Developing of the "Virtual Board" system for the distance learning system»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group ПІ31811: /Artur SAIENKO/

Scientific Supervisor: /Serhii SAMOILOV/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
 /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Саєнку Артуру Андрійовичу
1. Тема роботи: «Розробка системи "Віртуальна дошка" для системи дистанційного навчання»
Керівник роботи: Самойлов Сергій Петрович, старший викладач
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.__.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Аналітична частина, збір необхідної інформації щодо теми роботи для складання найбільш очікуваного потенційними користувачами програмного забезпечення. Питання вибору парадигми програмування системи, питання вибору фреймворків та мови програмування. Опис розробки системи, включаючи опис модулів, використаних технологій та алгоритмів. Тестування програмного забезпечення для виявлення помилок та багів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація – 10 слайдів

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	15.01.2022 – 24.01.2022	
2	Огляд літератури та аналіз аналогів	25.01.2022 – 31.01.2022	
3	Визначення вимог до програми. Вибір та обґрунтування мови програмування	01.02.2022 – 08.02.2022	
4	Узгодження та затвердження ТЗ	09.02.2022 – 10.02.2022	
5	Розробка та програмування логіки програми	11.02.2022 – 28.02.2022	
6	Розробка і реалізація інтерфейсу користувача	01.03.2022 – 25.03.2022	
7	Тестування та відлагодження програми	28.03.2022 – 12.04.2022	
8	Розробка, узгодження та затвердження програмної документації	13.04.2022 – 05.05.2022	
9	Подання кваліфікаційної роботи до кафедри	06.05.2022	
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.2022	

Студент _____ (підпис) Артур САЄНКО (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) ст. викл. Сергій САМОЙЛОВ (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

- вступ – в даному розділі описується сутність розробки, її актуальність, мета роботи, та задачі, які треба виконати. Перший розділ складається з 1 сторінки;
- збір вимог до програмного забезпечення – у цьому розділі проводиться опитування зацікавлених сторін для формування найбільш повних та точних вимог до програмного забезпечення, описуються аналоги програми, а також визначаються функціональні та нефункціональні вимоги. Другий розділ складається з 12 сторінок;
- зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд вхідних і вихідних даних, розробка фізичного проекту, приводиться опис об'єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи, вибір мови програмування. Складається з 17 сторінок;
- тестування та налагодження – включає в себе вибір стратегії тестування, опис тест-кейсів. Складається з 2 сторінок;
- висновки з виконаної роботи. Складається з 1 сторінки;
- список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;
- додатки – містить текст програми, який включає до себе усі файли проекту. Кількість таблиць: 2 штуки; Кількість рисунків: 41 штука.

Ключові слова: дистанційне навчання, віртуальна дошка, відображення у реальному часі, графіка, навчальний матеріал, домашнє навчання, парадигма програмування, автентифікація користувача.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення;

CMS (Content Management System) – Система керування змістом сайту;

СДН – Система дистанційного навчання;

HTML (HyperText Markup Language) – мова розмітки;

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

JWT (JSON Web Token) - це JSON об'єкт, який вважається одним з безпечних способів передачі інформації між двома учасниками.

ВСТУП

Актуальність

В даний час у зв'язку з важкою епідеміологічною обстановкою у світі, та в нашому випадку, за умови воєнного стану, повсюдно впроваджується новий формат взаємодії в різних сферах із застосуванням сучасних інтернет технологій. Технології дійшли до рівня, коли можна провести багато форматів заходів онлайн не гірше ніж якщо б вони були очно. Це можуть бути різні конференції, ділові зустрічі, консультації чи просто групова розмова. Сучасні технології дозволяють бачити і чути ваших співрозмовників, для цього вам потрібний тільки вихід в інтернет і пристрій, це може бути комп'ютер, ноутбук, планшет, телефон. Також потрібно мати гарнітуру, від якої залежить якість вашого звуку та якість зображення.

У зв'язку з цим набирає популярності таке поняття як дистанційне навчання, тому що з приходом якісного обладнання та появи зручного ПЗ, навчання з дому стало комфортним та ефективним. Слід зазначити також такі важливі функції для дистанційного навчання як: групові дзвінки, демонстрація екрану, віртуальні дошки, онлайн тести для підтвердження знань, курси з навчальними матеріалами та багато інших.

Зараз є безліч онлайн сервісів, де навчальний матеріал надається у вигляді курсів, всередині яких все грамотно структуровано, вони містять всю потрібну інформацію для підвищення кваліфікації, тому фахівцем у будь-якій сфері діяльності можливо стати прямо з дому, але треба приділити цьому достатньо свого часу.

Мета роботи

Метою випускної дипломної кваліфікаційної роботи є розробка віртуальної дошки для системи дистанційного навчання, що надає можливість викладачам та іншим працівникам сфери освіти, демонструвати у графічному форматі навчальний матеріал студентам, які перебувають на дистанційному навчанні. Студенти мають можливість швидко отримати доступ до потрібної інформації, проходити навчання, і спостерігати за віртуальною дошкою в реальному часі, що повинно підвищити ефективність навчання.

Щоб досягти поставленої мети, потрібно вирішити такі задачі:

1. Аналіз предметної галузі.

2. Проектування архітектури системи.

3. Проектування та розробку бази даних для підтримки роботи системи.

4. Реалізація системи.

5. Тестування віртуальної дошки.

2. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метою збору вимог є отримання інформації від незалежних або потенційних користувачів про те, які очікування про функціонал програми вони мають. На основі отриманих даних складається технічне завдання, цілі проекту, функціональне призначення та нефункціональне призначення програми. Існують різні способи збору вимог: інтерв'ю, розмова, анкетування та інші. Для збіру вимог був обраний метод дистанційного анкетування, що забезпечив збір інформації від різних потенційних користувачів такого класу систем.

2.1. Збір вимог від користувачів методом анкетування

Анкетування є найбільш формалізованою формою опитування. Це метод отримання інформації за допомогою письмових відповідей на систему заздалегідь підготовлених і стандартизованих питань з точно зазначеним способом відповідей. Основним інструментом цього виду опитування є анкета.

Для проведення анкетування була створена анкета за допомогою сервісу Google Forms, яка містить у собі питання різного характеру, починаючи з відомостей про опитуваного, про його досвід у використанні подібних сервісів віртуальних дошок, також питання про функціонал віртуальних дошок та інші питання, на основі яких можна зробити висновок про те, як повинна виглядати ефективна система дистанційного навчання з віртуальною дошкою. Результати анкетування представлені на рисунках 2.1-2.6 нижче.

Рисунок 2.1 – Відповідь на питання №1

Рисунок 2.2 – Відповідь на питання №2

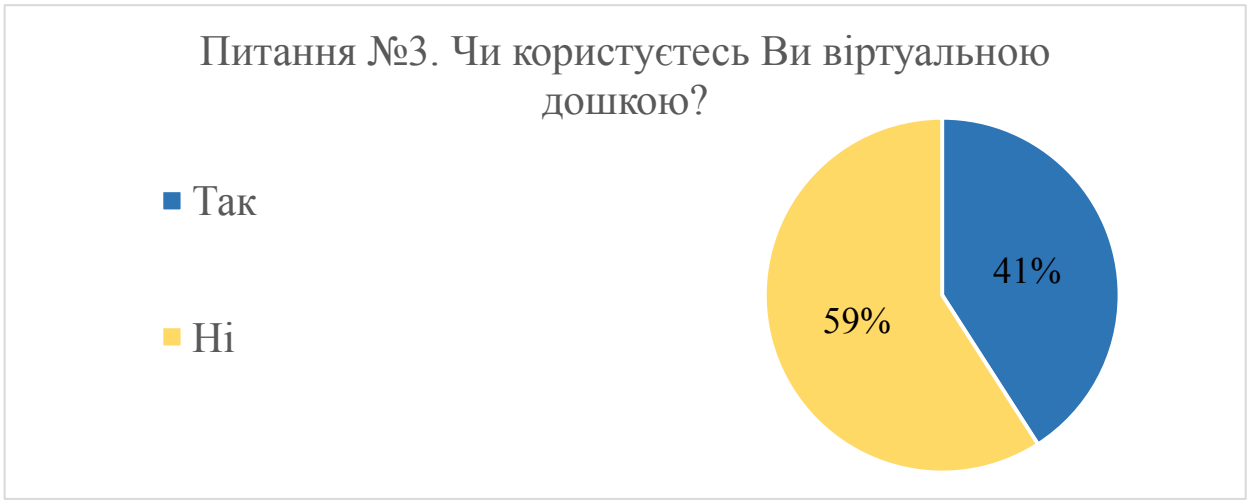

Рисунок 2.3 – Відповідь на питання №3

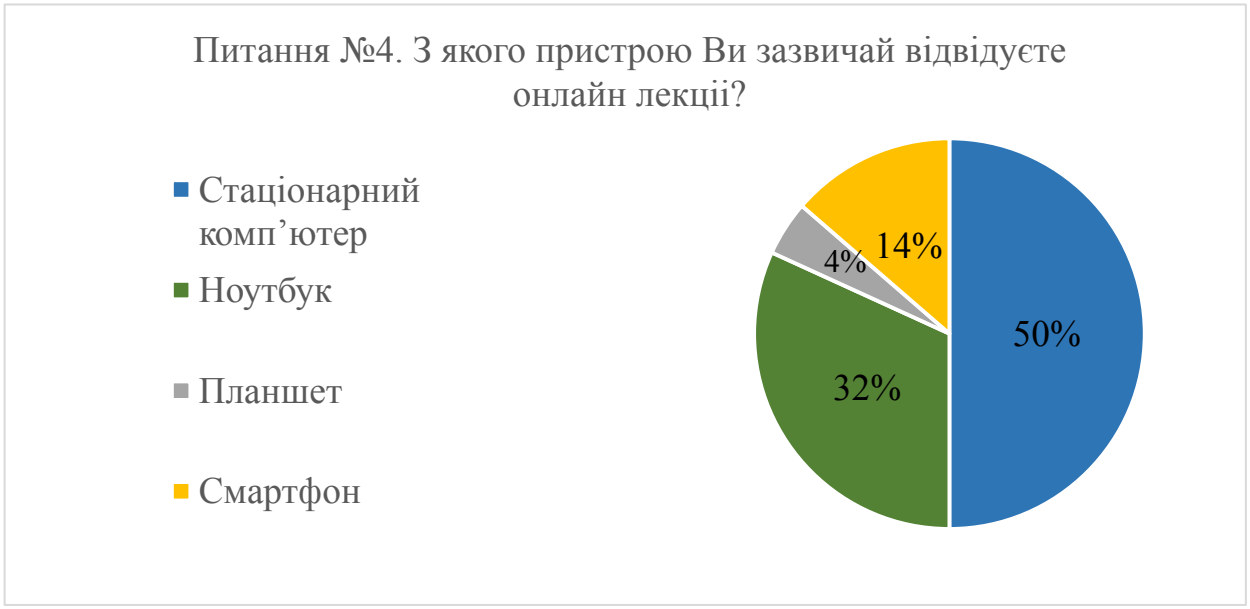

Рисунок 2.4 – Відповідь на питання №4

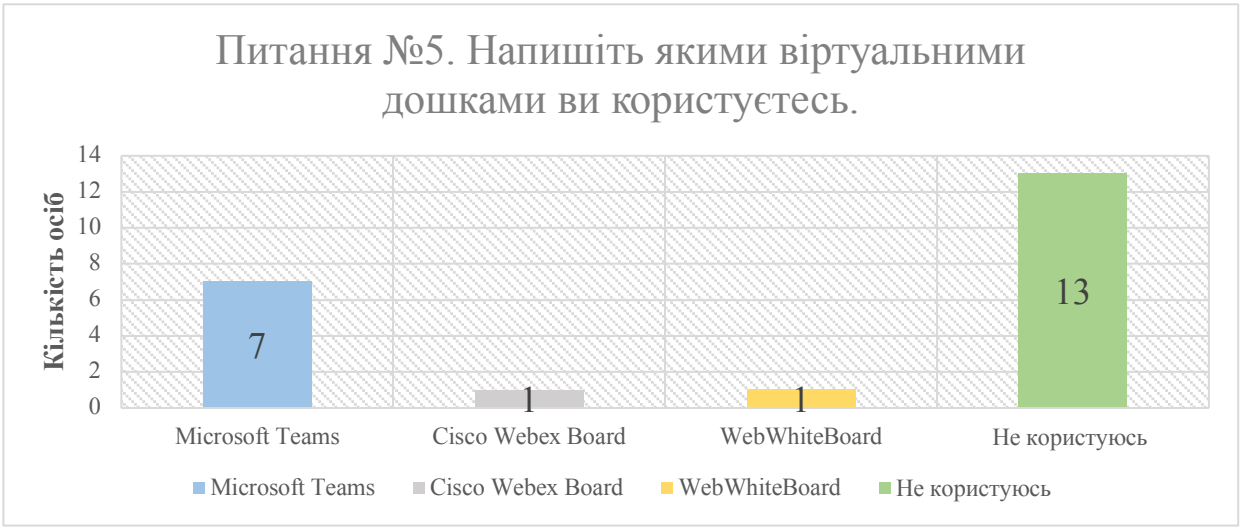

Рисунок 2.5 – Відповідь на питання №5

Рисунок 2.6 – Відповідь на питання №6

Анкета для збору інформації була надіслана можливим потенційним користувачам системи, в опитуванні брало участь 22 особи з різних сфер діяльності. Після проходження опитування всіма учасниками було отримано результати опитування. Згідно з результатами опитування були зроблені висновки щодо вимог, які стосуються програмного продукту.

Найпопулярнішою віртуальною дошкою опитуваних стала дошка, розроблена компанією Microsoft, яка пропонується в середовищі робочого простору системи Microsoft Teams.

Що стосується функціоналу віртуальної дошки, головними бажаними функціями опитуваних були: додавання зображення на віртуальну дошку, завантаження віртуальної дошки у форматі картинки до себе на пристрій, додавання фігур, стрілок, запис екрана під час роботи з віртуальною дошкою, малювання різними кольорами.

2.2. Огляд програмних аналогів

Для створення конкурентоспроможного і привабливого веб-сервісу, необхідно проаналізувати той функціонал, який реалізовано у вже існуючих подібних сервісах.

У ході виконання роботи було розглянуто кілька аналогічних сервісів з функцією віртуальна дошка, серед яких найбільш схожий функціонал мають такі веб-сервіси:

2.2.1 Microsoft Whiteboard

Дошка від компанії Microsoft, має хороший інтерфейс, і багатофункціональну панель приладів, дозволяє працювати над дошкою одночасно кількома користувачам. Інтерфейс зображений на рис. 2.7.

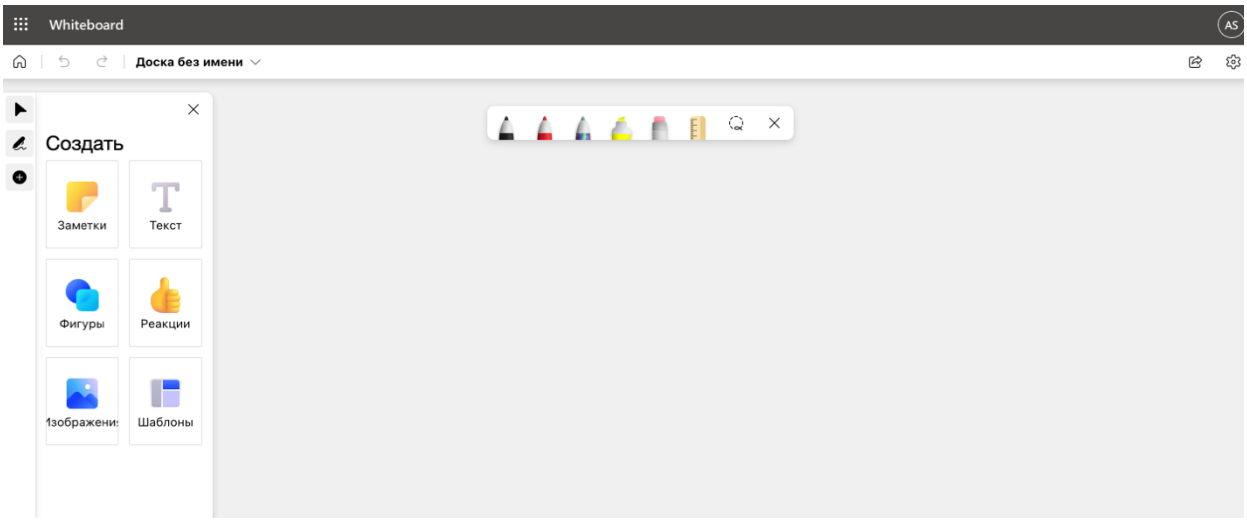

Рисунок 2.7 – Інтерфейс дошки Microsoft Whiteboard

Основні функції грамотно виконані - дуже легко створити нову дошку, і немає жодної складності щодо того, що роблять доступні інструменти на панелі інструментів.

Слід зазначити, що на перший погляд здається, що є три різні «ручки», насправді це всього три копії одного і того ж пера, всі з яких можна налаштувати лише мінімально (ширина, від "1" до "6" і ваш вибір з 15 кольорів).

Інструмент гумки занадто не стандартний; одне помилкове клацання миші і гігантські елементи вашої дошки зникають, тому що гумка не може стерти частину елемента, при контакті гумки з будь-яким елементом, лінією або текстом, елемент зникає з дошки.

У текстовому інструменті є один шрифт типу "рукописний", який не можна змінити. Існує також інструмент «нотатки», який імітує нотатки Post-It, але не відомо, яка ціль цих нотаток.

Приклад малювання пером, вид рукописного тексту та функцію нотатки можна побачити на рис. 2.8.

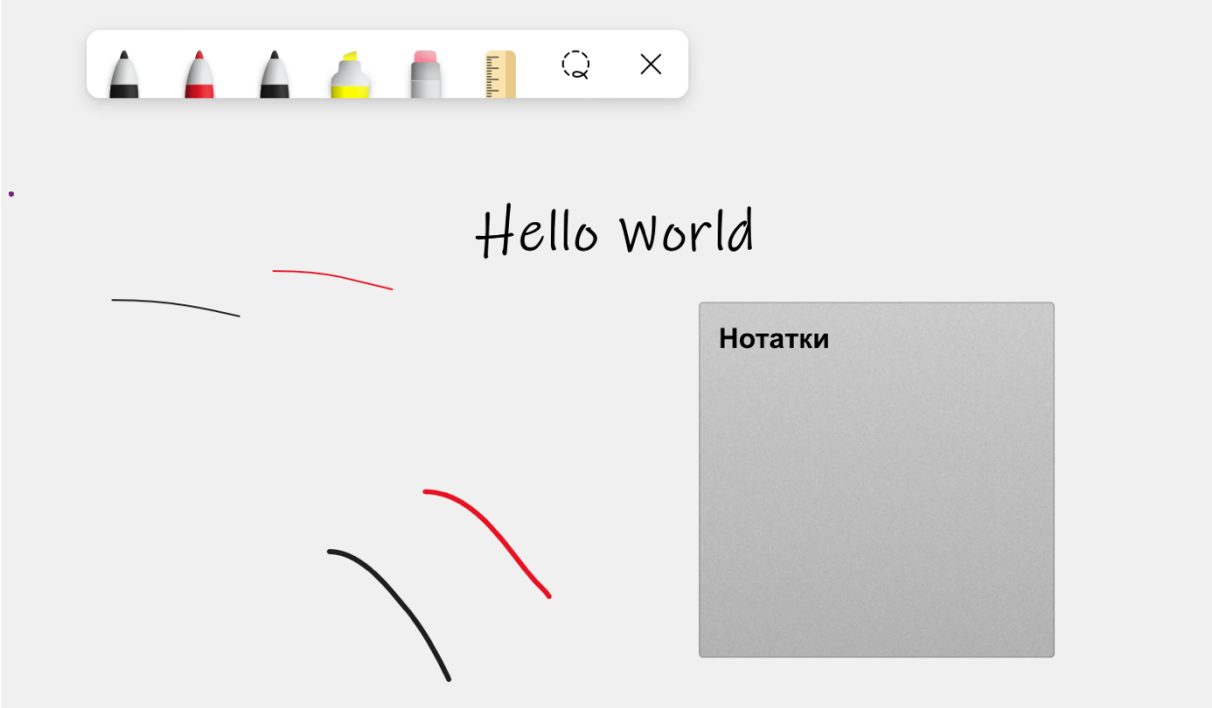

Рисунок 2.8 - Огляд інструментів Microsoft Whiteboard

Microsoft Whiteboard є гарна дошка для спільної роботи онлайн, має багатий функціонал для відображення різного матеріалу на полотні. Все ж таки для нових користувачів інтерфейс буде трохи не звичний і вимагатиме деякого часу щоб звикнути і розібратися. Також хочеться відмітити, щоб отримати доступ до дошки потрібна авторизація в обліковий запис Microsoft.

2.2.2. Web Whiteboard

Віртуальна дошка від компанії Miro користується популярністю по всьому світу. Має не дуже “дружній” інтерфейс, але значний функціонал. Для використання всіх функцій дошки, включаючи збереження даних на дошці, потрібна авторизація.

Якщо користувач не авторизований, дошка видаляється через 24 години після останньої зміни. Інтерфейс дошки відображено на рис. 2.9.

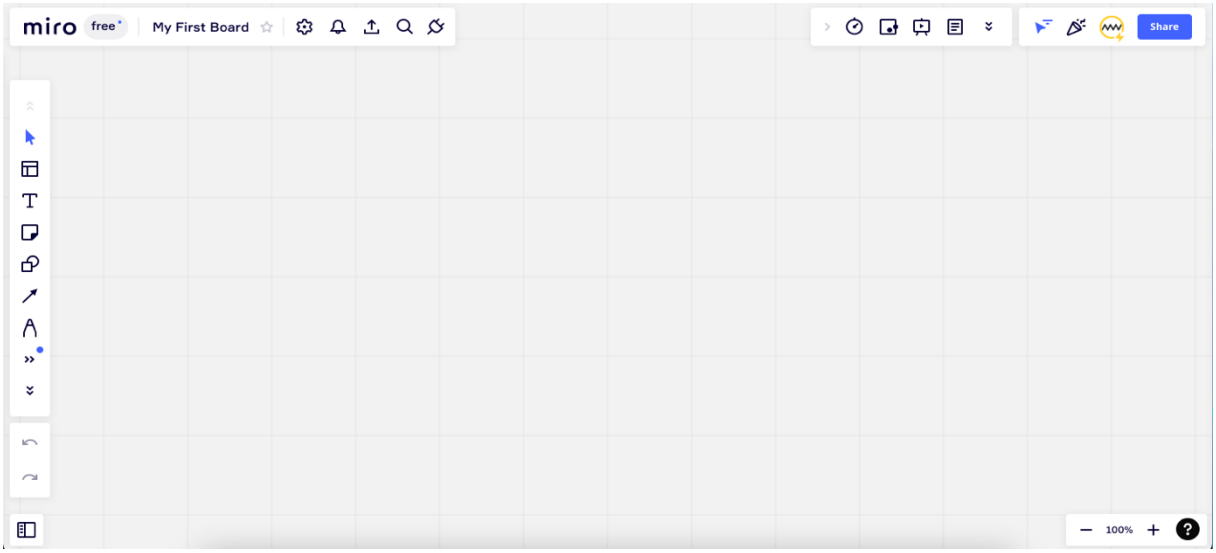

Рисунок 2.9 – Інтерфейс дошки Web Whiteboard

Головною особливістю цієї віртуальної дошки є великий набір шаблонів, які можна застосувати і вони відображатимуться на дошці. Панель вибору шаблонів відкривається в окремому вікні і всі шаблони зібрані у різних категоріях, відображені на рис. 2.10.

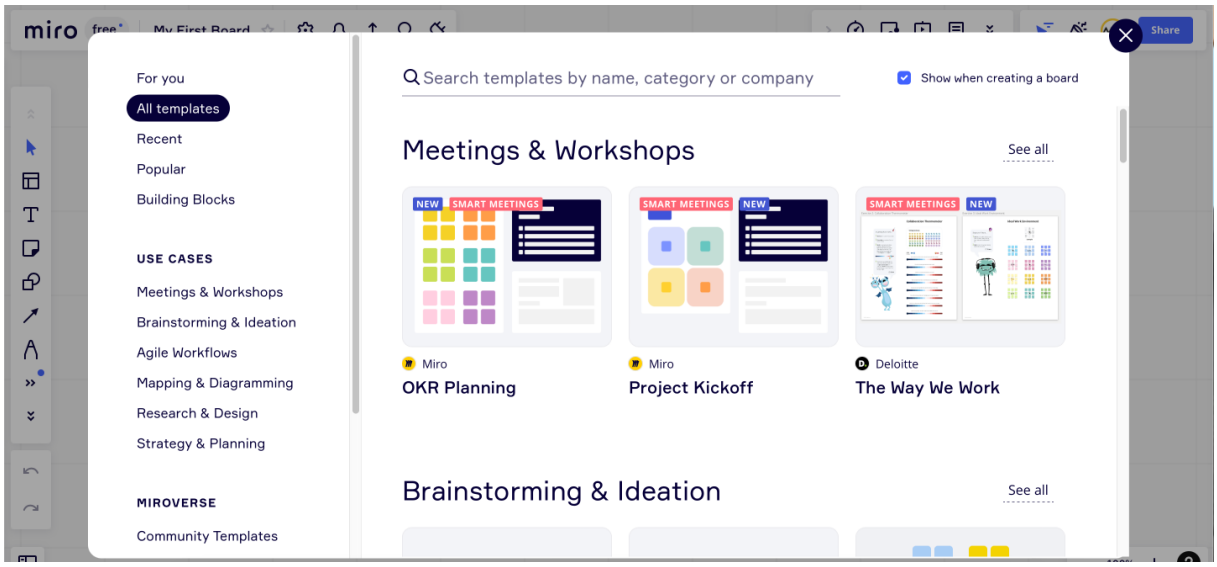

Рисунок 2.10 – Вікно вибору шаблону у системі Web Whiteboard

Шаблони дають можливість швидко створювати блок-схеми, графіки, діаграми, наприклад UML, швидко відображати планування задач, стратегію або алгоритм дій.

У цьому сервісі є різні види платних підписок, які дають доступ ще до трьох цікавих функцій: відео-чат, голосування і таймер, показано на рис. 2.11.

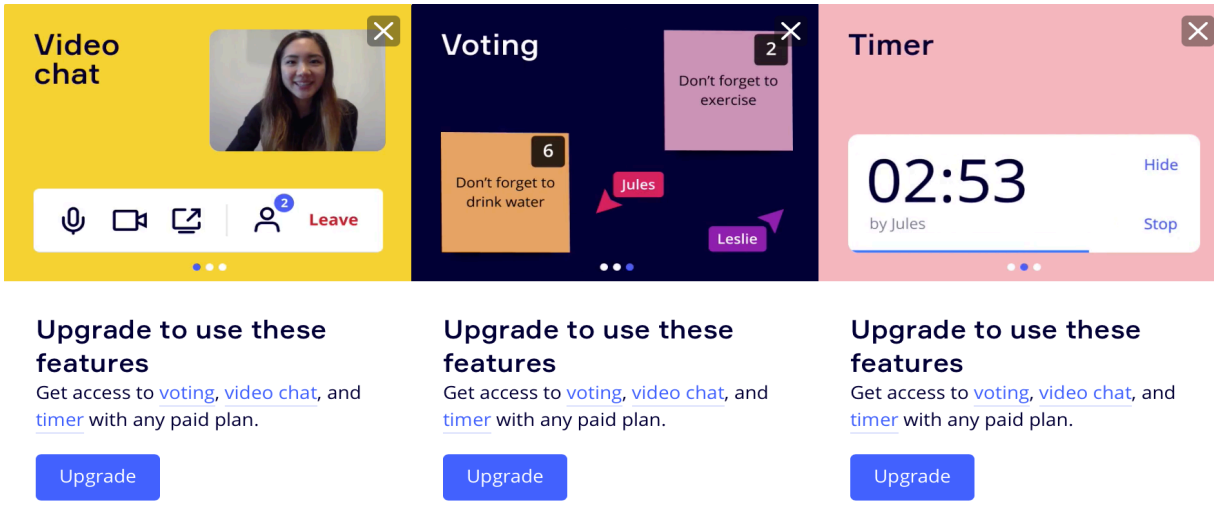

Рисунок 2.11 – Платний функціонал дошки Web Whiteboard

Існує функція додавання на дошку різних наклейок і смайлів, показаних на рис. 2.12.

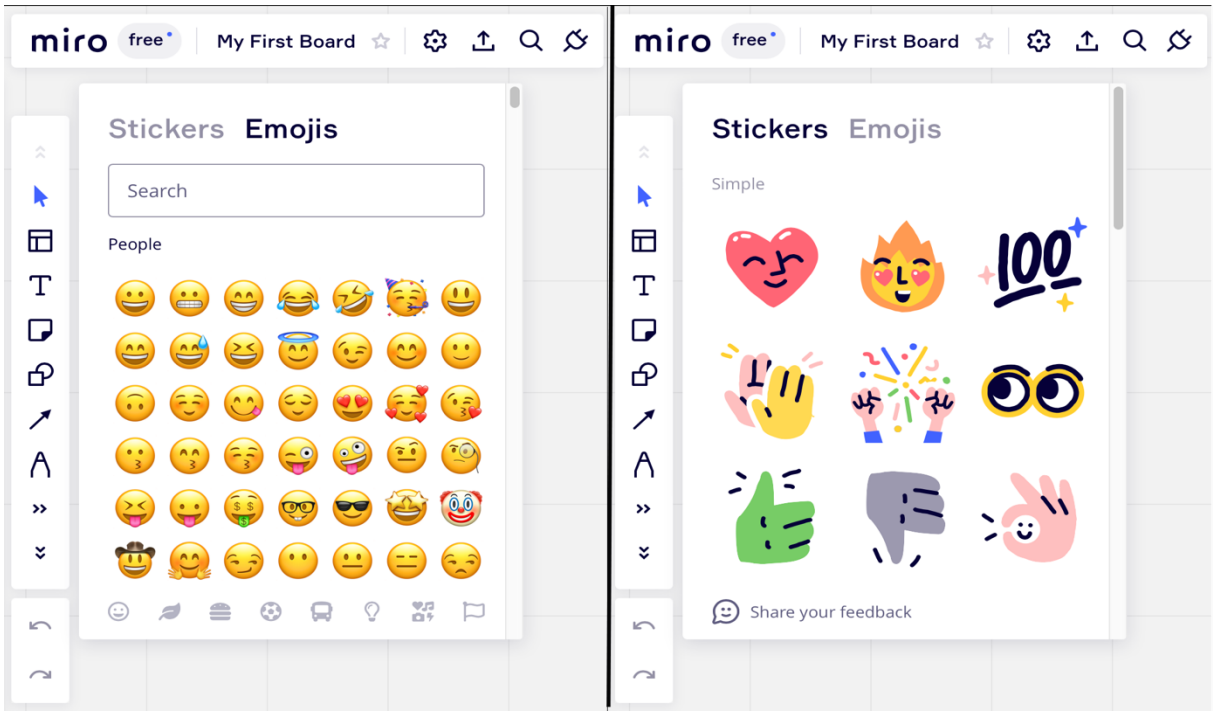

Рисунок 2.12 – Панель додавання стікерів та смайлів у системі Web Whiteboard

З дійсно незвичайних і гідних переваг цієї дошки, є вставка на дошку будь-яких форматів файлів, це може бути не тільки картинка, а й pdf файл або навіть презентація Power Point. Але з недоліків слід відзначити повільну роботу дошки за великої кількості елементів на ні, а також зазначити наявність платних функцій, що може не подобатись певним категоріям користувачів.

2.2.3. Twiddla

Twiddla – інтерактивна дошка для роботи та навчання. Безкоштовна версія обмежує кількість учасників до 10 осіб та тривалість роботи до 20 хвилин. Щоб зняти ці обмеження, необхідно оформляти підписку за 15$/місяць. Інтерфейс зображено на рис. 2.13.

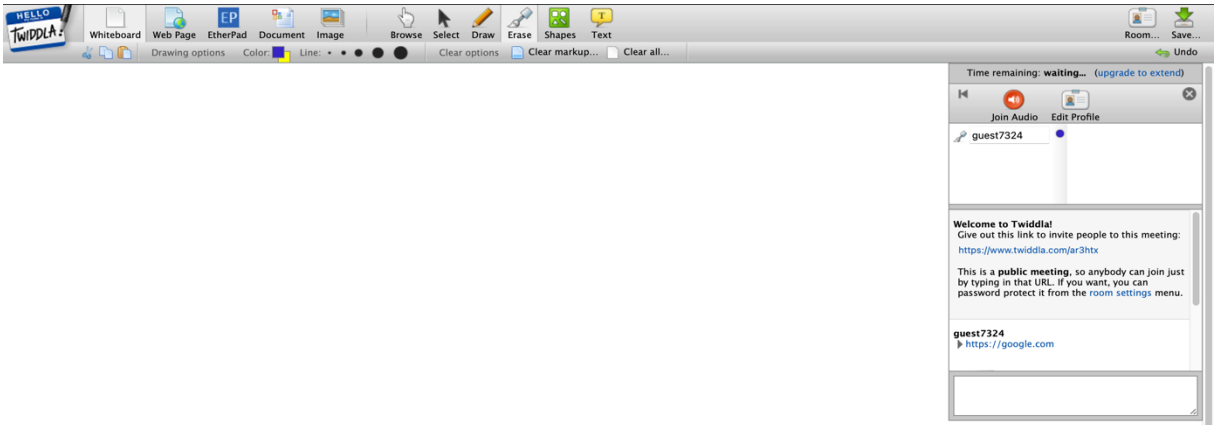

Рисунок 2.13 – Інтерфейс системи віртуальна дошка Twiddla

Інтерфейс цієї системи віртуальної дошки запозичено з елементів інтерфейсу операційної системи Windows XP , але має свої особливості. Наприклад, у неї є кілька робочих областей: Whiteboard, Web Page, Document, Image. У Whiteboard просто звичний білий аркуш, у Web Page потрібно вказати URL-адресу сторінки, вона відкривається на віртуальній дошці і користувач має можливість малювати та додавати елементи, рис. 2.14. Document та Image мають той же принцип, користувач може вибрати документ pdf або картинку, яка відобразиться на дошці на весь екран і вже працювати на ній. Слід зазначити, що додати картинку в маленькій мініатюрі на робочу область Whiteboard неможливо. Є також перевага у наявності голосового та текстового чату між учасниками.

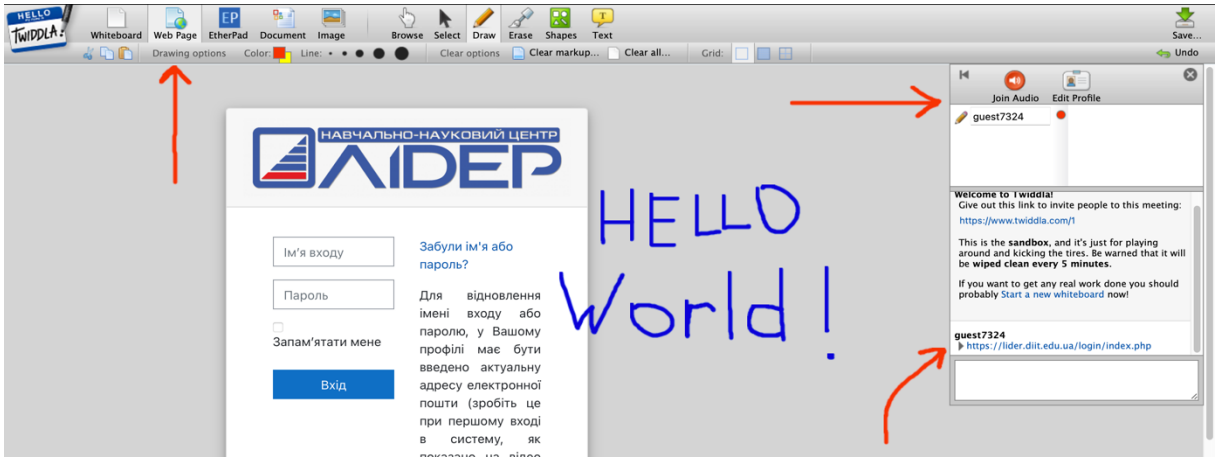

Рисунок 2.14 – Модуль Web Page у віртуальній дошці Twiddla

2.3. Функціональні вимоги

Щоб визначити функціональні вимоги, спочатку були побудовані діаграми використання для двох ролей користувачів: студента та викладача, вони відображені на малюнках 2.15 та 2.16. У таблиці 2.1 наведені функціональні вимоги та їх пріоритети.

Рисунок 2.15 – Діаграма варіантів використання для ролі “Студент”

Рисунок 2.16 – Діаграма варіантів використання для ролі “Викладач”

Таблиця 2.1 – Функціональні вимоги

Роль користувача	Варіант використання	Функціональна вимога	Пріоритет
Студент	Ознайомитися з навчальним матеріалом з курсу	Програма має відображати начальний матеріал з вибраного студентом курсу на веб-сторінці.	Високий
	Користуватись віртуальною дошкою	Програма має надати студенту можливість переглядати віртуальну дошку і відображати зміни на ній у реальному часі та надати функцію збереження віртуальної дошки у форматі png на пристрій студента.	Високий
	Зв’язатися з адміністратором сайту	Повинна бути можливість зв’язатися з адміністрацією сайту за допомогою форми.	Середній
Викладач	Маніпуляції з навчальним матеріалом	Програма має надавати викладачу можливість додавати, редагувати, видаляти навчальний матеріал з веб-сайту.	Високий
	Користуватись віртуальною дошкою	Програма має надавати можливість авторизуватись викладачу для активування функціоналу віртуальної дошки, такого як: додавання тексту, фігур, зображення. При використанні функціоналу - додавати елемент на робочу область (полотно). Також функціонал очищення робочої області та завантаження дошки у форматі png на пристрій викладача.	Високий
	Зв’язатися з адміністратором сайту	Повинна бути можливість зв’язатися з адміністрацією сайту за допомогою форми.	Високий

2.4. Нефункціональні вимоги

Були виділені наступні нефункціональні вимоги:

- інтерфейс має бути “дружнім” та зручним у використанні, читабельний шрифт та великі кнопки на панелі інструментів;
- веб-сайт не повинен містити яскравих кольорів;
- зображення віртуальної дошки, яке завантажується має бути у доступному усім форматі .png;
- зміни на віртуальній дошці повинні відображатися миттєво без оновлення сторінки;
- система повинна містить декілька окремих кімнат для використання віртуальної дошки одночасно різними групами користувачів;
- функціонал редагування сторінок з навчальним матеріалом повинен виключати можливість втручання у програмний код сторінки, але забезпечувати цю можливість за допомогою конструктора для зручності викладача (викладач не обов’язково повинен мати навики програмування);
- згідно з аналізом вимог, було виявлено, що користувачі використовують різні пристрої для дистанційного навчання, тому продукт повинен бути кросс-платформний, доступ до нього буде через веб-браузер.

Висновки до розділу

В результаті аналізу вимог до розробки програмного забезпечення було прийнято рішення створити кросс-платформний програмний продукт, який користувачі можуть використовувати з різних типів пристроїв. Оскільки система віртуальної дошки має на меті моментальну зміну інформації на веб-сторінці при будь-якій зміні в базі даних, було обрано можливість зберігати дані на хмарній платформі під назвою Firebase від Google. За рахунок специфіки роботи бази даних, ця платформа дозволяє отримати бажаного результату в миттєвому відображенні змін на веб-сторінці без її оновлення користувачем.

Сама система дистанційного навчання, де зберігатимуться веб-сторінки з навчальними матеріалами та посилання на кімнати з віртуальною дошкою розроблена на CMS WordPress, яка є системою керування веб-сайтом, завдяки цьому викладач отримує можливість легко змінювати зміст веб-сторінок з навчальним матеріалом,

додавати свої сторінки та редагувати їх при необхідності, без вторгнення до програмного коду сторінки.

Vue.js

Зараз існує багато різних фрейморків для розробки інтерфесів користувача, і кожен має свої переваги і недоліки. Було обрано фреймворк Vue 3 для розробки зовнішнього інтерфейсу відповідно до вимог, Vue має велику кількість можливостей та функцій. Це швидко прогресуючий фреймвок, його розвиток останнім часом справді дуже суттєвий. Це сучасний фреймворк JavaScript, який надає корисні засоби для поступового впровадження — на відміну від багатьох інших фреймворків, ви можете використовувати Vue для покращення існуючого HTML. Інтерфейс написаний за допомогою Vue складається з одного або кількох компонентів. Коли запускається екземпляр глобальної програми, спочатку є кореневий екземпляр. Цей кореневий екземпляр складається з дерева компонентів, які мають власний кореневий екземпляр [2].

Firebase

Щоб зробити можливим бачити оновлення на віртуальній дошці без оновлення сторінки, було розглянуто варіант зберігання даних проекту на хмарній платформі Firebase від компанії Google. У ній є функціонал використання бази даних Firebase Realtime Database, вона розміщена у хмарі. Дані зберігаються у форматі JSON та синхронізуються в реальному часі для кожного підключеного клієнта. Тобто замість типових запитів HTTP база даних Firebase Realtime використовує синхронізацію даних — кожен раз, коли дані змінюються, будь-який підключений пристрій отримує це оновлення протягом мілісекунд.

CMS WordPress

Зважаючи на майбутнє використання системи дистанційного навчання викладачами різного типу навчальних курсів, для полегшення маніпуляції з даними навчальних матеріалів, було спроектовано та розроблено систему, яка дозволяє додавати, редагувати та видаляти сторінки курсів без втручання в програмний код сторінки. Для цього була використана система керування змістом сайту CMS(Control Managment System) WordPress, та інтегрован перехід на віртуальну дошку за допомогою цього самого веб-сайту. Також для використання віртуальної дошки одночасно кількома різними групами користувачів було створено кілька робочих областей, які називаються кімнатами, що надають змогу викладачам та студентам обирати у якій кімнаті вони будуть працювати у даний момент часу [3].

3. ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

3.1. Зовнішнє проектування

Призначення цього розділу – розробка основних компонентів ПЗ, що створюється у рамках дипломного проектування. Вирішуються задачі проектування математичного, алгоритмічного, інформаційного, технічного, програмного, організаційного видів забезпечення, а також завдання, пов'язані із взаємодією між продуктом та користувачем ПЗ.

На рис. 3.1 відображено загальну схему складових компонентів розроблюваного програмного забезпечення. Призначення сайту відображає у собі відомості як навчального так і інформаційного характеру. Студенти можуть використовувати його для прочитання матеріалу та статей, які публікує викладач. Користувачі мають можливість ознайомитися з текстовою та відео-інструкцією щодо роботи з системою віртуальна дошка, а також можливість зв'язатися з адміністрацією у разі виникнення помилок чи питань пов'язаних із роботою системи.

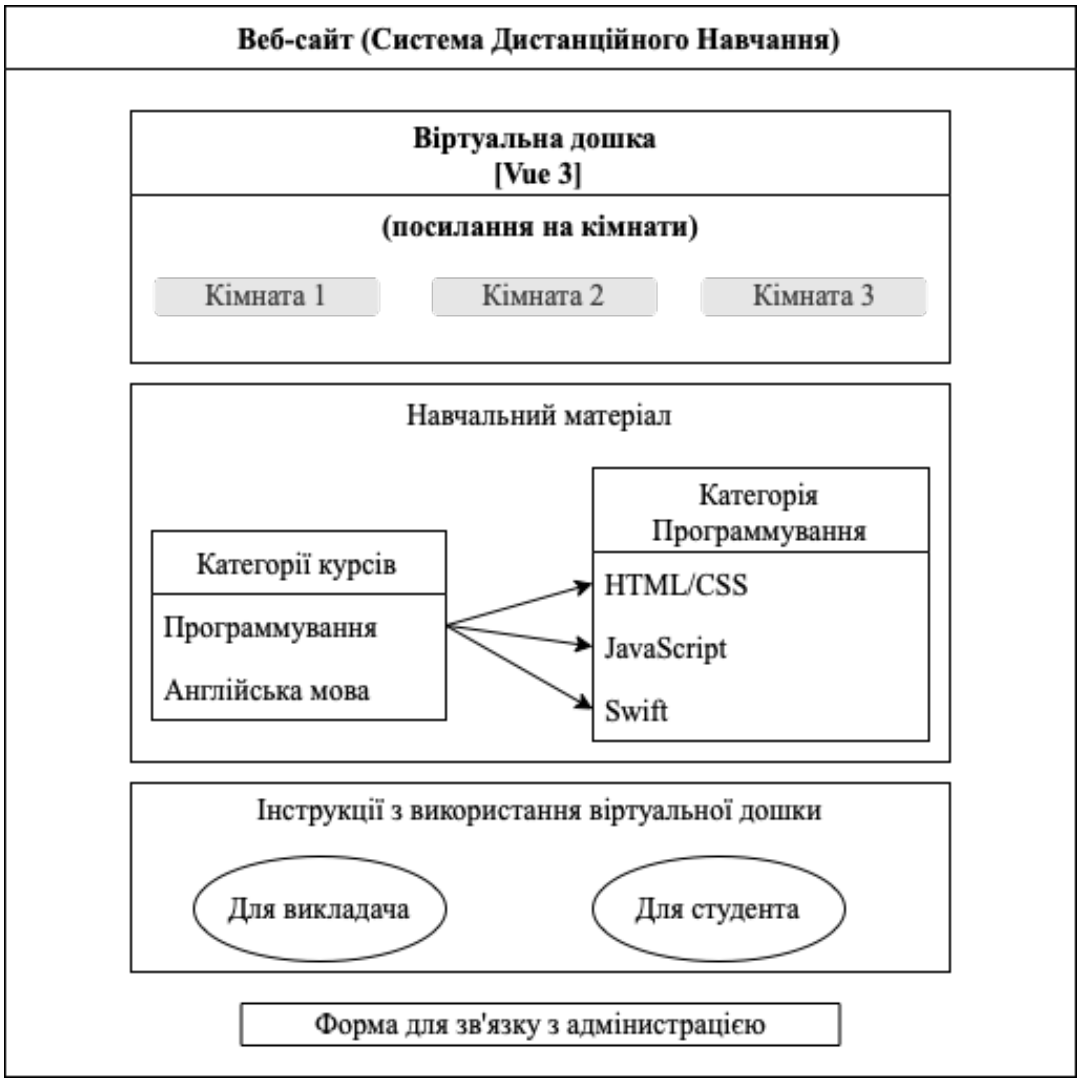

Рисунок 3.1 – Зовнішня структура розроблюваного ПЗ

Функціональне призначення полягає в тому, щоб надавати можливість викладачам відображати у реальному часі інформацію на екрані, яка буде використана студентом для навчання з певної навчальної дисципліни.

Експлуатаційне призначення полягає в тому, щоб спростити процес навчання студентам, графічне представлення навчального матеріалу може швидше та ефективніше пояснити складні речі простим способом – наочним. При цьому виграють обидва суб'єкти, які задіяні у цьому навчальному процесі: викладач та студент, обидва економлять свій час.

На UML діаграмі прецидентів, рисунок 3.2, відображено можливі дії, які можуть бути виконані при користуванні системою віртуальна дошка. У системі є два рівні доступу: гість (студенти, які будуть переглядати навчальний матеріал на віртуальній дошці) та зареєстрований користувач (викладачі, які будуть повністю взаємодіяти з віртуальною дошкою, використовуючи увесь її функціонал).

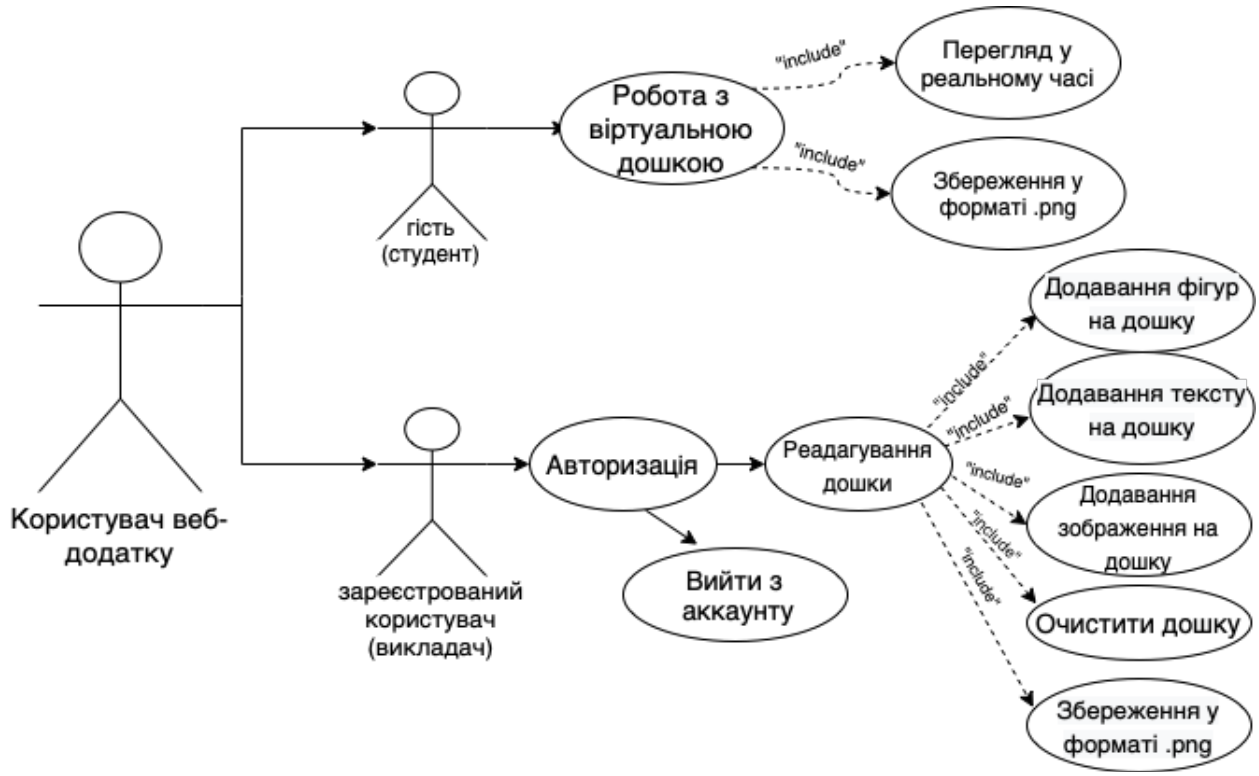

Рисунок 3.2 – UML Діаграма прецидентів

Функціональні вимоги, які були реалізовані при розробці системи віртуальної дошки, можна поділити на: функціонал для викладача, функціонал для студента, тому що частина функціоналу, яка доступна викладачеві не буде доступна студенту, тому це потрібно врахувати при розробці та правильно встановити параметри доступу різних ролей користувачів до функціоналу.

3.1.1. Проектування інтерфейсу користувача

Перед розробкою інтерфейсу програми був спроектований початковий дизайн , сам інтерфейс віртуальної дошки складається з таких компонентів: Header (шапка), DrawingTools (панель інструментів), Canvas (робоча область малювання), EditorBox (вікно додавання елемента), ModalForm (вікно авторизації користувача). На рисунку 3.4 відображено спроектований дизайн інтерфейсу перших трьох описаних вище компонентів.

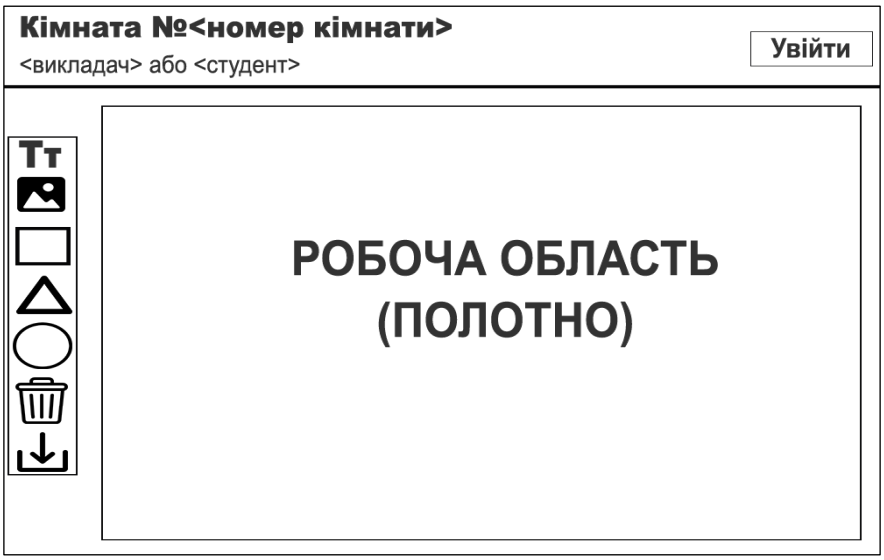

Рисунок 3.4 – Дизайн інтерфейсу користувача віртуальної дошки

Також було спроектовано дизайн основного сайту системи дистанційного навчання, з якого користувачі переходитимуть на віртуальну дошку за допомогою кнопок зі посиланням, дизайн інтерфейсу відображено на малюнку 3.5.

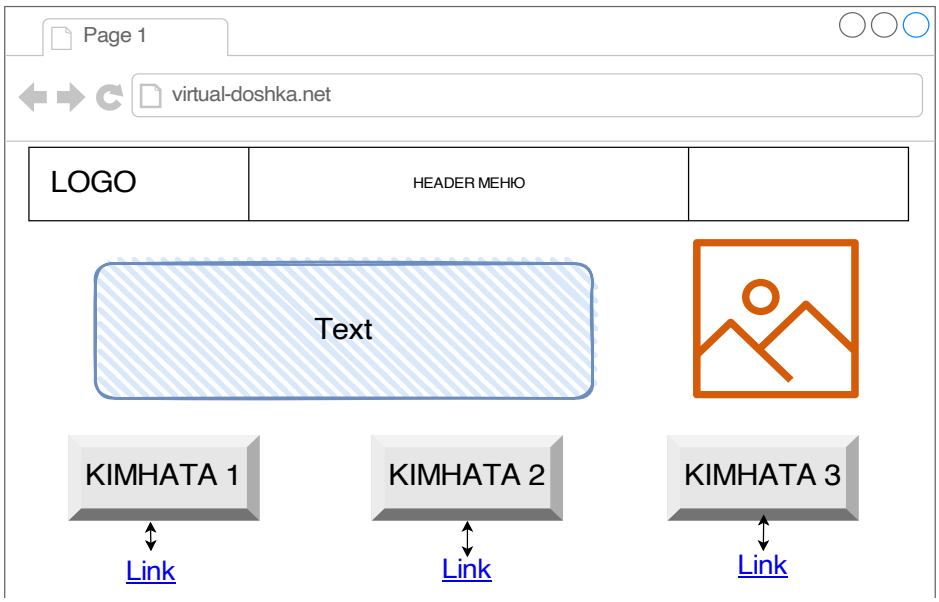

Рисунок 3.5 – Дизайн інтерфейсу користувача веб-сайту

3.1.2. Вхідні дані

У всіх працюючих програмних продуктів є вхідні та вихідні дані. Без вхідних даних программа завжди виконуватиме ту саму процедуру при запуску. При зміні значення вхідного параметра програма видаватиме різні результати. Вихідні параметри дозволяють повернути результат[1]. Існує кілька типів вхідних даних, що використовуються в веб-додатку, їх можна поділити на різнорівневі вхідні дані: на рівні інтерфейсу користувача, на рівні компонентів, і на рівні логіки. Кожен рівень представлення вхідних даних має власну специфіку обробки.

Вхідні дані на рівні користувача: email і пароль.

Являють собою два текстові значення типу String, які користувач вводить при авторизації в систему в однорядкових текстових полях введення, які відображені на рисунку 3.6, у момент натискання на “Send” дані надсилаються на сервер аутентифікації.

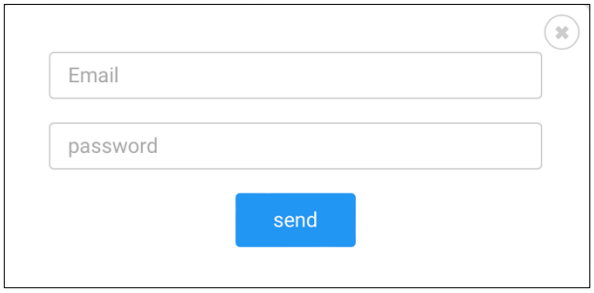

Рисунок 3.6 – Форма входу

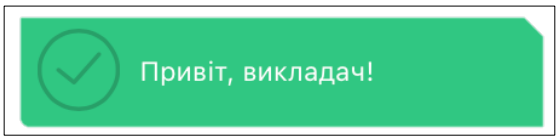

Рисунок 3.7 – Алерт “Привіт, викладач”

Потім сервер аутентифікації створює JSON Web Token (JWT) та надсилає його клієнту. Коли клієнт робить запит до сервера веб-додатку, він додає до нього отриманий раніше від сервера аутентифікації JWT. У схемі, зображеній на рисунку 3.8, сервер додатку налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації, у разі успішної перевірки користувач успішно авторизується і бачить повідомлення зображене на рисунку 3.7.

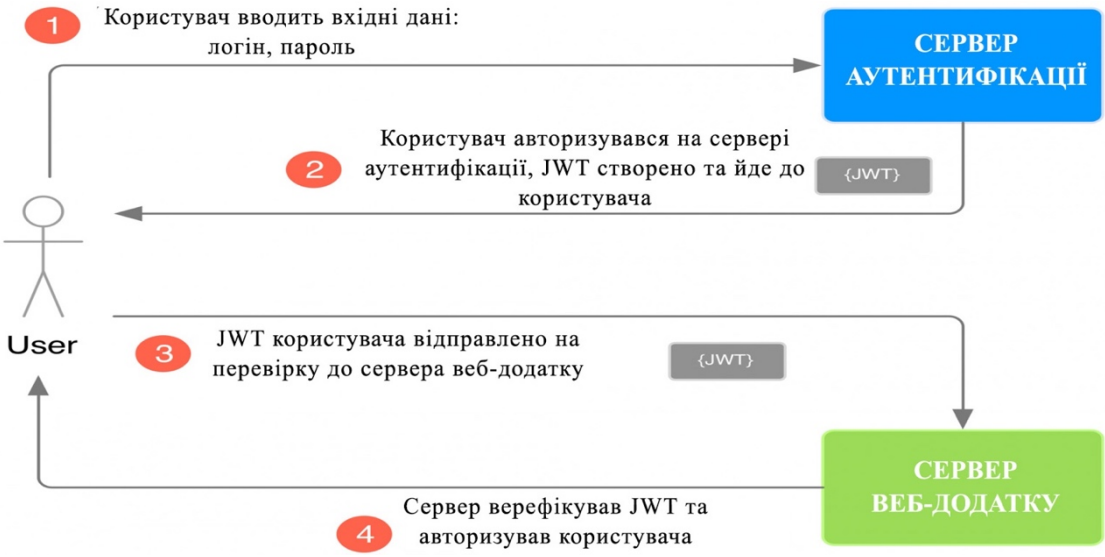

Рисунок 3.8 – Схема аутентифікації за допомогою JWT

На рисунку 3.3 зображено сторінку налаштування аутентифікації на платформі Firebase. Тут зберігається дані зареєстрованих користувачів: email, дату реєстрації та унікальний ідентифікатор UID. На цій сторінці є можливість додати нового користувача.

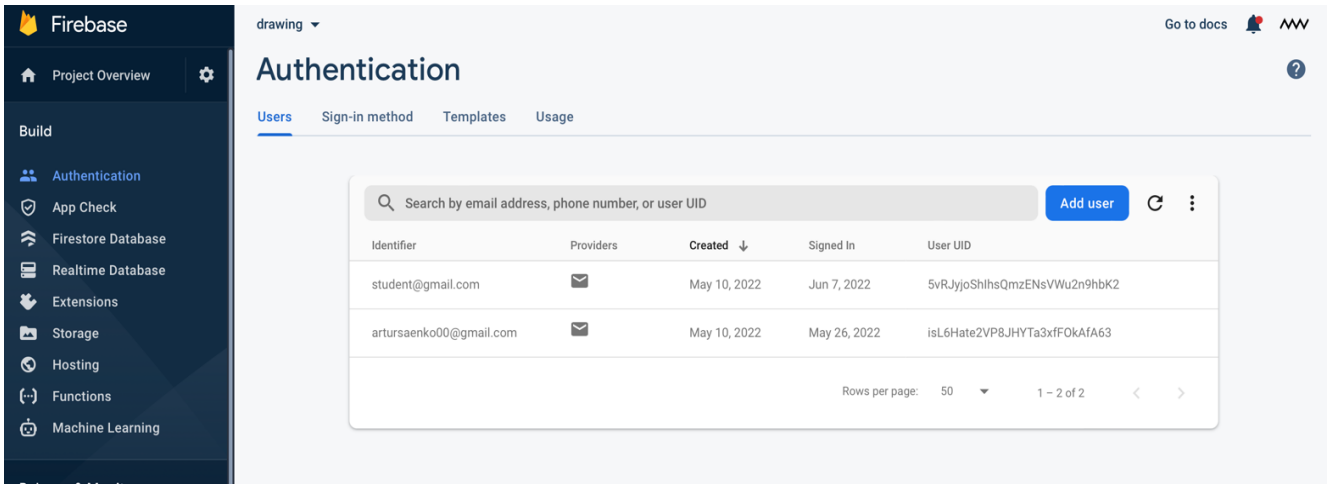

Рисунок 3.3 – Firebase Authentication

Вхідні дані на рівні компонентів Vue - параметри компонентів.

Батьківський компонент передає дані компоненту нащадку через вхідні параметри, компонент нащадок надсилає повідомлення батька за допомогою подій.

```
export default {
  name: "EditorBox",
  props: {
    posX: Number,
    posY: Number,
    updatePos: Function,
    onResetClass: Function,
    addTextToCanvas: Function,
    addImageToCanvas: Function,
  },
```

Рисунок 3.9 – Вхідні параметри компоненту EditorBox

Екземпляр компонента має свою ізольовану область бачення. Звертатися безпосередньо до даних батьківського компонента із шаблону компонента-нащадка - не можна. Дані передаються вниз по ієрхії за допомогою вхідних параметрів.

Для прикладу на рисунку 3.9 зображені вхідні параметри компонента EditorBox, який приймає своє ім'я для реєстрації екземпляра, два значення координат(x,y), оновлені координати після переміщення компонента EditorBox по Canvas, функція скасування додавання елемента, функції додавання тексту та додавання картинки.

Вхідні дані на рівні логіки: положення і тип фігури.

Відобразити елемент на віртуальній дошці дозволяє допоміжний компонент EditorBox, який представляє собою список елементів: кнопок і поля для введення. Кнопки мають події миші, які описуються в розділі внутрішнього проектування, вони надають можливість переміщати об’єкт, змінюючи цим дефолтні координати. Поле для введення може прийняти в собі вхідні параметри у вигляді типу елемента, що передається, це може бути фігура, зображення або текст. На рисунку 3.10 зображений блок коду, де прописана логіка передачі даних до компоненту EditorBox.

```
const figure = ["square", "triangle", "circle"];
const elements = [...figure, "text"];

const position = computed(() => {
  return {
    x: props.posX,
    y: props.posY,
  };
});

const type = computed(() => store.getters.GET_EDITOR_SETTINGS.type);
const typeFigure = computed(() => figure.includes(type.value));

const showEditor = computed(
  () => elements.includes(type.value) || userImg.value
);
```

Рисунок 3.10 – Передача параметрів у EditorBox

3.1.3. Вихідні дані

Основними вихідними даними програми є зображення, оскільки мова йдеться про систему віртуальна дошка, де користувачі взаємодіють із графічними елементами. Зображення, яке бачить користувач на полотні вважається вихідними даними, зображення генерується авторизованим користувачем (викладачем) за допомогою додавання на Canvas різних елементів.

Зображення віртуальної дошки може бути завантажене на пристрій користувача у форматі .png. На рисунку 3.11 проілюстровано блок коду, який відповідає за цей функціонал. Формат .png гарний вибір за рахунок своєї компактності, картинка займає дуже мало місця на диску, і ще цей формат зберігає прозорість, що може бути корисно в деяких випадках, наприклад для дизайнерів.

```
const downloadImage = () => {
  const link = document.createElement("a");
  link.href = dataURL();
  link.download = "my-image.png";
  link.click();
};
```

Рисунок 3.11 – Завантаження картинки у форматі .png

Для зберігання даних на віртуальній дошці передається картинка, яка містить у собі всі елементи на дошці (компонент Canvas), в змінну dataURL: за допомогою методу toDataURL(), згенерований на полотні PNG-файл переводиться у формат кодування Base64, який на виході з програми відправляється на сервер для зберігання. Наступний рядок коду присвоює значення змінній DataURL.

```
const dataURL = () => canvas.value.toDataURL("image/png");
```

Base64 — це двійкова схема кодування тексту, яка представляє двійкові дані у форматі String ASCII. Наприклад, закодоване зображення віртуальної дошки у форматі Base64 містить трохи більше 40 тисяч символів.

На рис. 3.12 проілюстровано блок коду, який відповідає за надсилання цих даних на сервер бази даних Firebase.

```
const sendImage = () => {
  sendData(urlParams, {
    image: dataURL(),
    id: Date.now(),
    width: canvas.value.width,
    height: canvas.value.height,
  });
};
```

Рисунок 3.12 – Відправка даних на сервер бази даних

Вже на рисунку 3.13 ми можемо бачити як відображаються дані, що посилаються програмою, на платформі Firebase, як було зазначено вище, вони зберігаються у файлах, схожих на формат JSON (ключ-значення) [11].

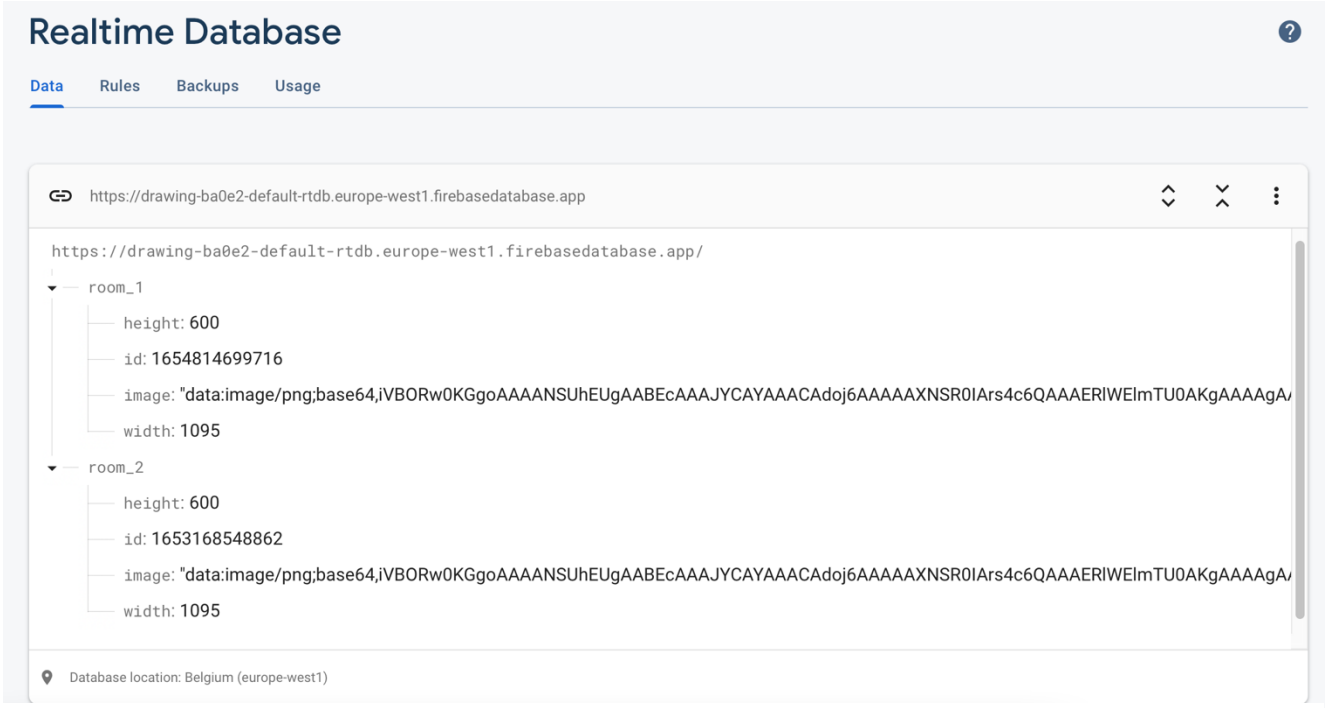

Рисунок 3.13 – База даних Realtime Database на платформі Firebase

3.2. Внутрішнє проектування

У цьому розділі розробляється сама програма, її модулі, її компоненти, логіка, методи, використовуючи підходи, парадигми, патерни проектування ПО. Ми вивчаємо поведінку програми з точки зору різних дій з боку користувача. Проектуємо архітектуру програмного забезпечення, підключаємо базу даних для зберігання та маніпулювання даними. Результатом проектування є детальна модель ПЗ, що розробляється, разом зі специфікаціями його компонентів усіх рівнів. Тип моделі залежить від вибраного чи заданого підходу (структурний, об'єктно-орієнтований чи компонентний) та конкретної технології проектування. Однак у будь-якому випадку процес проектування охоплює як проектування обробних програм (підпрограм) та визначення взаємозв'язків між ними, так і проектування даних, з якими взаємодіють ці програми або підпрограми.

3.2.1. Компоненти Vue

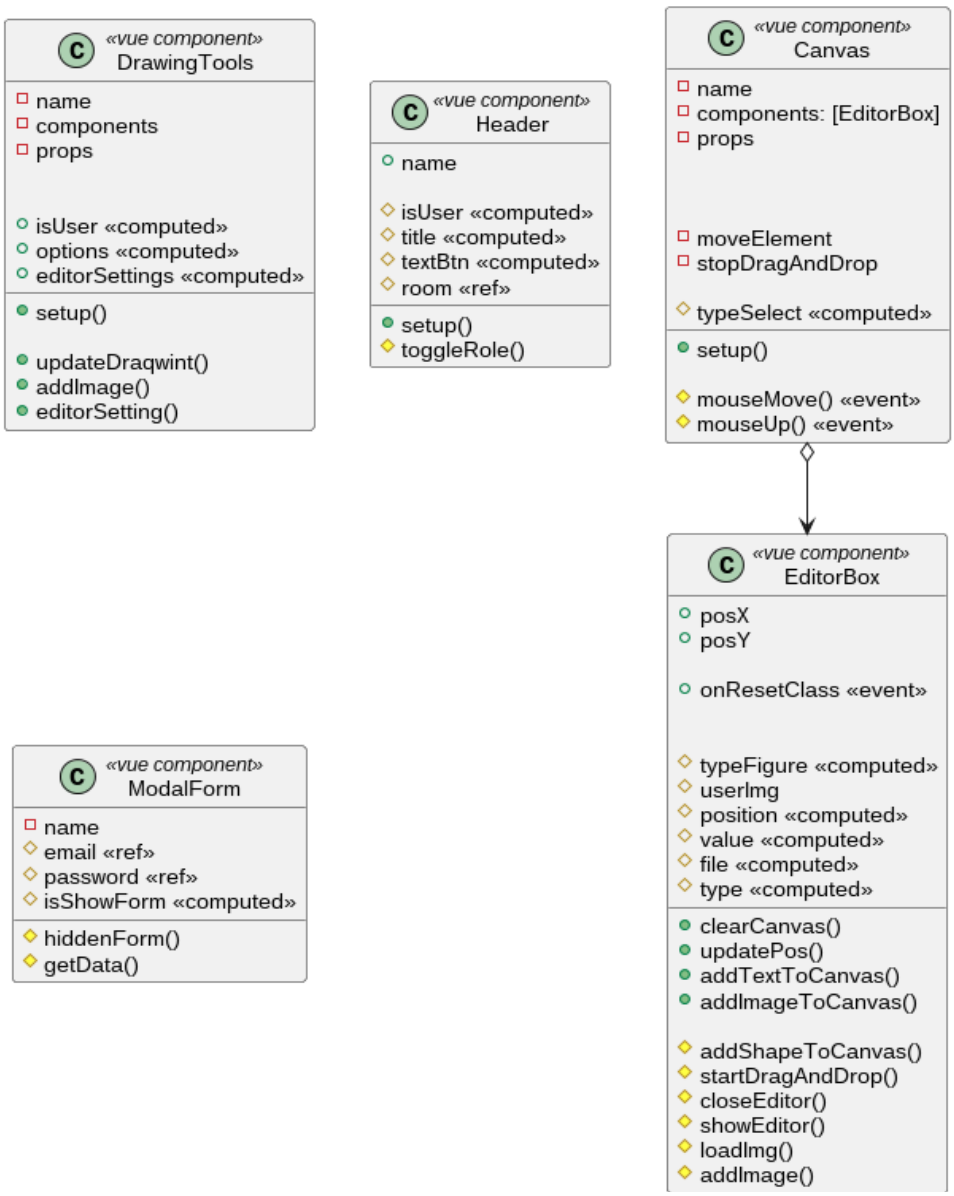

Рисунок 3.14 – Діаграма компонентів Vue для додатку “Віртуальна дошка”

Так як при розробці інтерфейсу віртуальної дошки використовується фреймвок Vue, ми використовуємо компонентний підхід. Компонент — це автономна, багаторазова частина логіки інтерфейсу користувача. Кожен компонент може мати свій власний стан, розмітку та стиль. Варто відзначити, що ви можете повторно використовувати компонент скільки завгодно разів. Потім це дозволяє розробникам створювати додаток, організований у дерево вкладених компонентів. Вже існують сотні існуючих компонентів, які ви можете швидко імпортувати у свої проекти для виконання простих або складніших завдань. На попередній сторінці, на рисунку 3.14 представлено діаграму компонентів для веб-додатку “Віртуальна дошка”, яка розробляється в рамках дипломного проекту

Кожен компонент Vue має свій шаблон, при створенні екземпляра компонента у себе в програмі, розробник задає внутрішні властивості, функції, події, які описують цей екземпляр і потім використовує цей екземпляр або перевикористовує якщо необхідно відобразити кілька подібних елементів у інтерфейсі користувача, компоненти можна перевикористовувати будь-яку кількість разів. Більшість розробки у фреймворку Vue.js відбувається у файлах компонентах типу Vue, ці файли складаються з трьох частин: template, scripts і style. Блок template є шаблонизатором, у цьому блоці пишеться Html розмітка, також там застосовуються JavaScript скрипти. Блок scripts містить усі скрипти поточного Vue компонентв. За допомогою JavaScript реалізується реактивність даних у даному фреймворку. Блок style містить у собі стилі CSS що відноситься до поточного описуваного компонента [12].

1. Компонент EditorBox складається зі списку елементів, ці елементи включають три кнопки і одне поле для введення. Кнопки мають свої події миші, які реагують на натискання, зажимання, переміщення, відпускання клавіші миші. У верхньому лівому куті компонента кнопка відповідає за переміщення вибраного елемента на віртуальній дошці, вона має подію @mousedown="startDragAndDrop" , затиснувши цю кнопку, користувач має можливість пересувати компонент разом з елементом на екрані. У правому верхньому куті компонента розміщені ще дві кнопки, вони мають у собі тільки подію кліку миші по ним, клік по лівій кнопці виконує процедуру додавання елемента на дошку (@click="addShapeToCanvas"), клік по правій кнопці виконує процедуру закриття компонента EditorBox, якщо Ви передумали додавати поточний елемент на дошку(@click="closeEditor"). Сам компонент відображено на рисунку 3.15.

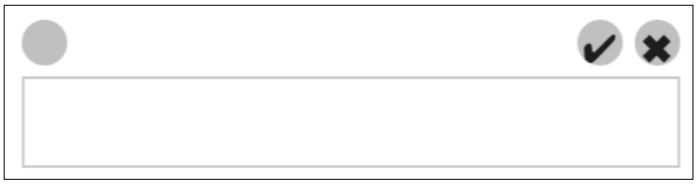

Рисунок 3.15 – Компонент EditorBox

Компонент EidtorBox приймає різні параметри, такі як тип елемента, що надсилається в поле введення, позиція розташування. У свою чергу ці параметри можуть тримати в собі структуру тексту, фігури, зображення або координати x, y.

2. Компонент Canvas має зв'язок з іншим компонентом описаним вище - EditorBox, тому що в блоці шаблонізатора Canvas компонент EditorBox вписаний усередину, тобто він вкладений усередину. Компонент Canvas має всередині процедури, які дозволяють малювати елементи на формі, включаючи їхнє розташування, яке походить від EditorBox передаючи координати (x, y). Компонент Canvas бере в себе елемент, якщо ми вирішимо його додати за допомогою EditorBox, цей функціонал реалізований за допомогою патерна Vuex, про який мова піде нижче, коротко кажучи ми зберігаємо наші дані в сховищі, і при зміні стану ці дані перезаписуються з допомогою геттерів, мутацій (для синхронних методів), дій (для асинхронних методів).

3. Компонент DrawingTools являє собою список, всередину якого вкладені кнопки вибору інструмента, кнопки мають іконки інструмента, за який вона відповідає. За замовчуванням, кнопки які мають у собі опцію редагувати дошку деактивовані, тому що всередині компонента прописано умову isUser, яке перевіряє чи авторизований користувач. У випадку, якщо користувач не авторизований, йому доступне лише завантаження дошки у форматі зображення до себе на пристрій. В іншому випадку, якщо користувач увійшов до системи, йому відкривається доступ до всіх кнопок для взаємодії з віртуальною дошкою.

Також у цей компонент включений масив, який несе за собою опції, пов'язані з кнопками. Ми отримуємо опцію залежно від кнопки, яку натисне користувач, таким чином програма розуміє, які дії необхідно виконати при натисканні на кожну кнопку. На рисунку 3.16 відображено компонент DrawingTools.

Рисунок 3.16 – Компонент DrawingTools

4. Компонент Header являє собою звичайний контейнер, що має в собі title "Кімната" + параметр room, що приймається з URL адреси сторінки, блок коду відповідальний за отримання цього параметра відображений на рисунку 3.17. У випадку, якщо ніякого параметра не було отримано, функція по дефолту повертає 1, що прирівнюється до адреси першої кімнати.

```
export const getParams = () => {
  //?room=1
  const url = new URL(location.href);
  let room = 1;
  if (url.search) {
    room = url.search.split("=")[1];
  }
  return room;
};
```

Рисунок 3.17 – Функція отримання параметру номеру кімнати

Також крім цього є ще один title, що має обчислювану властивість computed, тобто змінюється, цей title перевіряється умовою isUser і відображає роль користувача. Якщо користувач авторизований title відображає "Викладач", інакше відображається "Студент". З правої частини компонента розташована кнопка, яка викликає компонент ModalForm, title кнопки має той же принцип, перевіряється умова isUser, якщо користувач авторизований title відображає "Вийти", якщо користувач не авторизований – "Увійти". Компонент Header зображений на рисунку 3.18. На рисунку представлено Header авторизованого користувача у першій кімнаті.

Рисунок 3.18 – Компонент Header

5. Компонент ModalForm має два поля для введення email і password. Компонент відповідає за прийом даних від користувача, які потім будуть надіслані на сервер аутентифікації. Використовує сховище даних vuex та викликає звідти action, який в свою чергу запускає асинхронну функцію, яка повертає від сервера аутентифікації відповідь про успішну або не успішну авторизацію користувача.

3.2.2. Сховище даних Vuex

Vuex сховище - це паттерн або бібліотека, розроблена спеціально для програм Vue.js. Цей паттерн організовує використання централізованого сховища. За допомогою цього паттерна є можливість керування статусом усіх компонентів програми, та використовування відповідних правил, щоб гарантувати, що статус змінюється передбачуваним чином. Щоб полегшити маніпуляцію даних у програмі, за допомогою цього патерну ми можемо зберігати всі дані, які нам коли-небудь знадобляться у додатку, в одному централізованому сховищі. Кожен компонент усередині нашої програми буде запитувати дані зі стану Vuex і передавати змінені дані назад до стану. Схема роботи Vuex сховища зображена на рисунку 3.19.

Щоб компоненти могли отримати доступ до даних, що зберігаються в нашому стані всередині нашого сховища Vuex, необхідно визначити геттери, які повертатимуть дані зі сховища нашим компонентам.

Щоб встановити дані в сховищі ми визначимо сеттери, але Vuex-сеттери називаються інакше - Мутаціями (Mutation) застосовуються для установки даних у стан Vuex.

Але варто звернути увагу, що мутації є синхронними, і ми не можемо запускати асинхронні операції, всередині мутації. У цьому випадку необхідно створювати дії (Actions). Дії подібні до мутацій, але замість того, щоб безпосередньо змінювати стан, вони здійснюють мутацію яка отримає вже оновлені дані.[7]

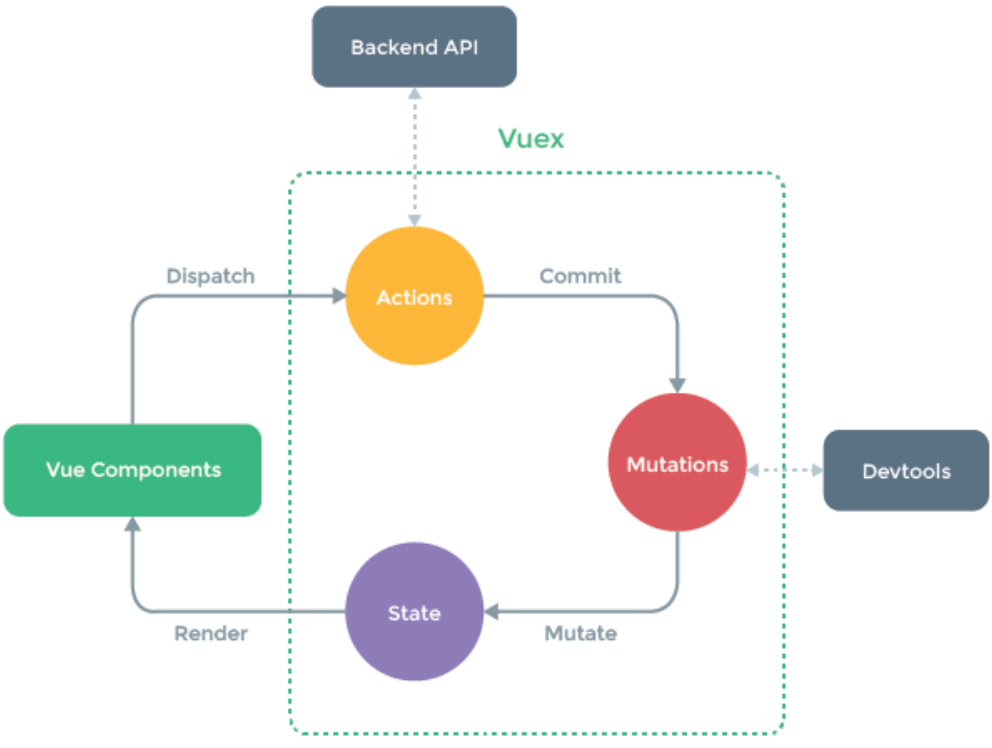

Рисунок 3.19 – Схема роботи сховища Vuex

3.2.3. Firebase Realtime Database

Google Firebase — це програмне забезпечення для розробки програм, яке підтримує Google, яке дає змогу розробникам розробляти iOS, Android та веб-програми. Firebase надає інструменти для відстеження аналітики, звітування та виправлення збоїв додатків, створення маркетингу та експерименту з продуктом і багато інших В рамках цього проекту ми використовуємо два інструменти Firebase: Authentication та Realtime Database. Інструмент Authentification дозволяє нам керувати користувачами веб-додатоку, нам надається віддалена служба автентифікації користувачів. Firebase пропонує різні методи автентифікації, які відображаються на рисунку 3.20 . Серед них є аутентифікація по телефону, яка відправляє SMS на номер телефона, також є авторизація за допомогою сторонніх сайтів, таких як Google, Microsoft, Github та інші. Але ми використовуємо у нашому проекті авторизацію за допомогою електронної адреси та паролю.

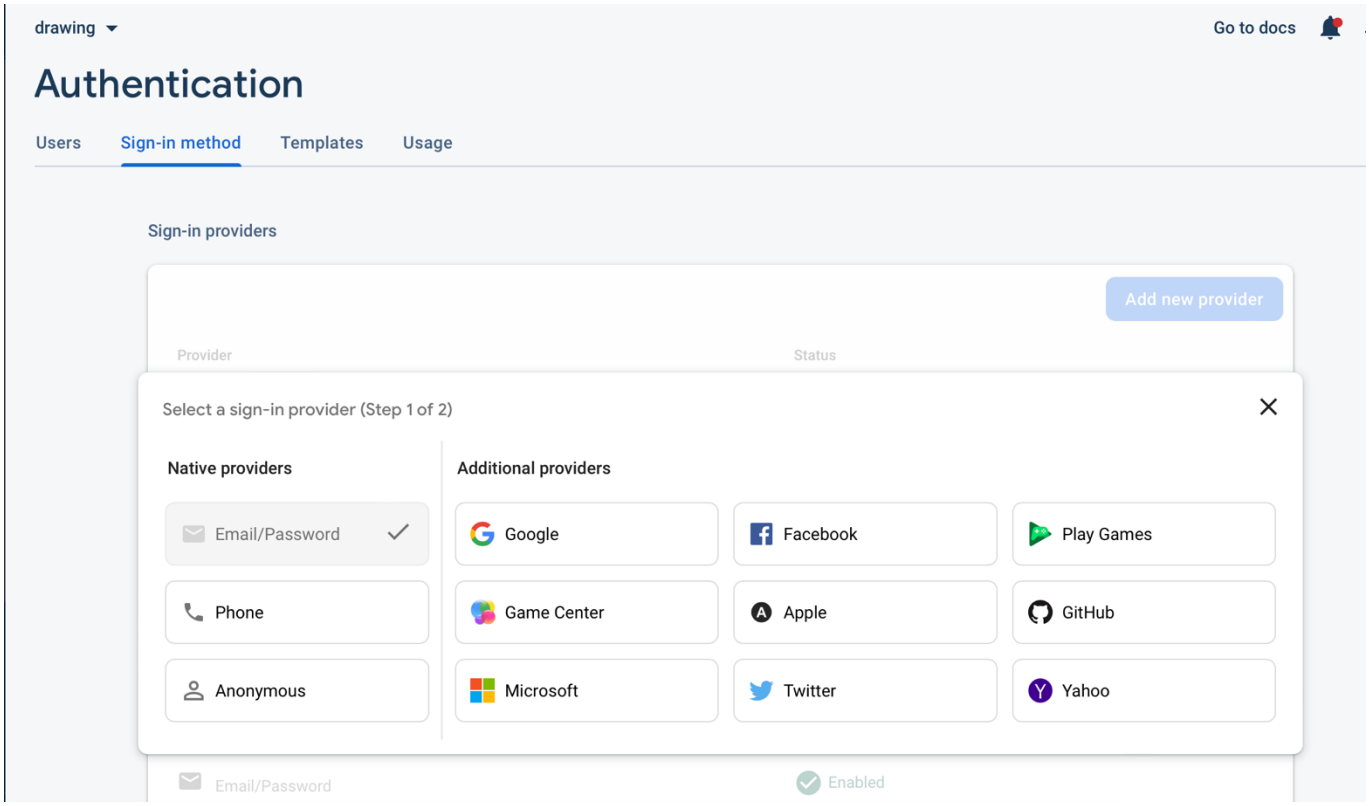

Рисунок 3.20 – Способи автентифікації у сервісі Firebase Authentification

Щоб підключити Firebase до проекту, спочатку необхідно завантажити пакет розробника Firebase SDK, це можна зробити, прописавши команду "npm install firebase" в середовищі розробки. При роботі робиться import з цієї SDK.

Тепер створюється проект на самій платформі Firebase, після створення проекту будуть надані дані, які необхідно прописати у файлах проекту, це називається config - конфігураційний файл платформи, в якому описані налаштування самої платформи Firebase. На рисунку 3.21. відображено config проекту “Віртуальна дошка”, завдяки цьому проект зв’язано з платформою Firebase. [9]

```
export const firebaseConfig = {
  apiKey: "AIzaSyCLmFPwuGDKDD2I_Yhi7A4CIqZdlRHKDpY",
  authDomain: "drawing-ba0e2.firebaseapp.com",
  databaseURL:
    "https://drawing-ba0e2-default-rtdb.europe-west1.firebasedatabase.app",
  projectId: "drawing-ba0e2",
  storageBucket: "drawing-ba0e2.appspot.com",
  messagingSenderId: "784399442719",
  appId: "1:784399442719:web:d146fa4158bb1dd7a4c62a",
};
```

Рисунок 3.21 – Налаштування файлу Firebase config

Також необхідно написати логіку в програмі, яка прийматиме від користувачів email і пароль і надсилатиме дані на сервер аутентифікації, а потім повертатиме результат залежно від коректності введених користувачем даних. Шаблон блоку коду, який потрібен можна знайти в документації Firebase SDK.[10] На малюнку 3.22 відображено блок коду, що відповідає за перевірку успішності авторизації.

```
1  import { initializeApp } from "firebase/app";
2  import { getAuth, signInWithEmailAndPassword, signOut } from "firebase/auth";
3
4  import { firebaseConfig } from "@/config/firebase-config";
5
6  initializeApp(firebaseConfig);
7  const auth = getAuth();
8
9  export const checkUserDetails = ({ email, password } = {}) => {
10   return signInWithEmailAndPassword(auth, email, password)
11     .then((userCredential) => {
12       const user = userCredential.user;
13       console.log("user :>> ", user);
14       return user;
15     })
16     .catch((error) => {
17       throw new Error(error);
18     });
19 };
20
21 export const logOff = () => {
22   signOut(auth)
23     .then(() => {})
24     .catch(() => {});
25 };
```

Рисунок 3.22 – Перевірка успішності авторизації користувача

Другий інструмент сервісу Firebase, який був використаний у дипломному проекті – це Realtime Database. База даних, яка має модель noSQL, яка зберігає дані у форматі ключ-значення (key-value). База даних – це великий об'єкт JSON, яким розробники можуть управляти в режимі реального часу. [8] На рисунку 3.23 показано схему як працює база даних.

Рисунок 3.23 – Схема роботи Realtime Database

Використовуючи один API, база даних Firebase надає програмі поточне значення даних та оновлює їх. Синхронізація в реальному часі дозволяє користувачам легко отримати доступ до даних з будь-якого пристрою. База даних Firebase в реальному часі використовує синхронізацію даних замість HTTP-запитів. Будь-який підключений пристрій отримує оновлення протягом мілісекунд.

В рамках управління проектами Firebase є можливість переглянути використання ресурсів бази даних (наприклад, скільки користувачів підключено, скільки місця займає ваша база даних), це показано на рисунку 3.24.

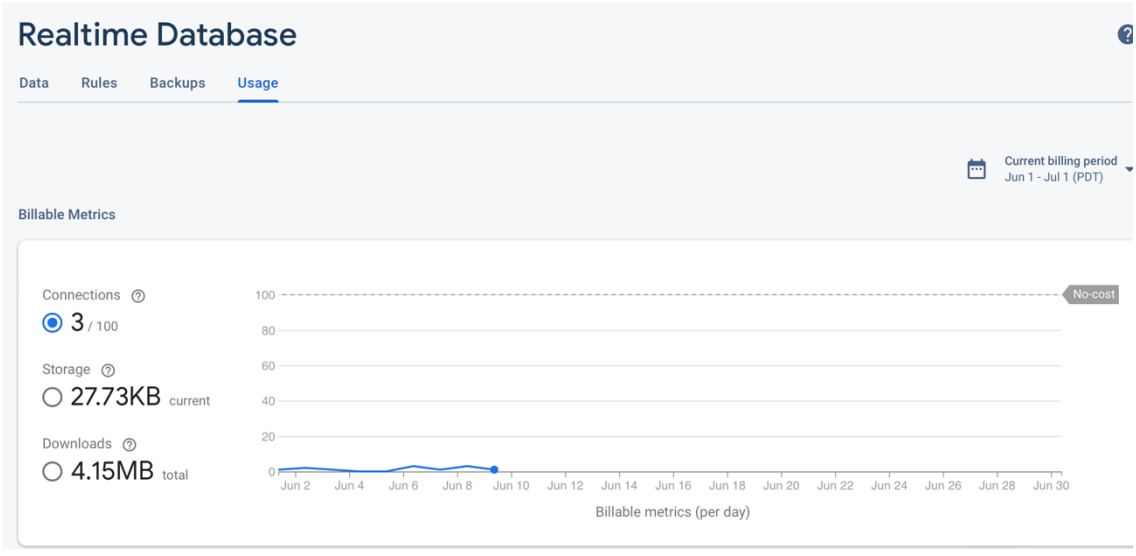

Рисунок 3.24 – Графік навантаження бази даних

3.2.4. Вибір парадигми програмування

Парадигма програмування — це система думок і понять, які визначають стиль написання комп'ютерних програм, а також спосіб мислення програміста [5].
Множину всіх створених підходів до програмування прийнято поділяти на 2 основні типи:

1) імперативний (англ. imperative — наказ) — підхід, що наказує послідовність кроків, що має зробити програма в процессі своєї роботи;
2) декларативний (англ. declaration — заява, декларація) — підхід, де розробник описує проблему та очікуваний результат, без жодних інструкцій щодо виконання самої програми;

Основною концепцією імперативної парадигми є покрокове виконання программ, які можуть мати змінні, розгалуження, умови та цикли. Також відмінною рисою служить поняття стану компьютера або программи.
Стан - це сукупність всіх даних у певний момент часу: змінних, масивів, лічильників тощо.
Імперативний стиль, у порівнянні з декларативним, де розробник описує лише проблему та очікуваний результат, але не пише жодних інструкцій, можна вважати більш низькорівневим, адже програмісту
потрібно знати як саме працює програма.

Станом на сьогодні, більшість коду пишеться в імперативному стилі програмування. Його найпопулярнішими підвидами є процедурна парадигма та ООП(об'єктно орієнтоване програмування чи об'єктно орієнтована парадигма).
У процедурному програмуванні алгоритм виконання програми також представлений як послідовність інструкцій, де набори однотипних інструкцій організовані в спеціальні блоки коду - процедури, які можна
викликати багато разів з будь якої точки програми та навіть усередині самих себе. Процедурна парадигма припускає що вся програма - це набір процедур, які можуть містити виклики один одного та повертати одна одній значення. [4]

Таким чином, процедурна парадигма програмування дозволяє повторно використати код, сприяє легшому розумінню логіки роботи програми, що в сукупності дозволяє зручно масштабувати проекти та є великим плюсом, тому що дозволяє
значно знизити час необхідний програмісту на написання коду, а сам код при цьому лише виграє у простоті. Ще одним важливим плюсом такого підходу є легкість у зміні вихідного коду, що на практиці найчастіше виявляється критичною
необхідністю, коли виникає вимога додавання або зміни функціоналу додатка. Оскільки замість необхідності багаторазово робити ті самі зміни в коді з ризиком пропустити і не помітити частини функціонал яких потрібно змінити - у нас з'являється можливість змінити логіку скрізь, де це потрібно шляхом одиничного внесення змін до самої процедури. [6]

У багатьох випадках цей стиль дуже зручний. Однак така велика кількість абстрікцій має свою ціну: зокрема в процесі трансляції об'єктно-орієнтованих програм у код центрального процесора, виникає ряд
неоптимальностей з використання пам'яті та обчислювального часу процесорних ядер. Доречно буде навести цитату Джо Армстронга, твореця мови програмування Erlang: "Проблема з об'єктно-орієнтованими мовами
полягає в тому, що у них є все це неявне середовище, яке вони носять із собою. Ви хотіли банан, але отримали горилу, що тримає банан і всі джунглі".

Декларативний стиль програмування не є таким розповсюдженним у промисловій розробці, як імперативний. Основною причиною є те, що декларативний стиль використовується у вузькоспеціалізованих областях.
До декларативної парадигми належать функціональне та логічне програмування, що широко застосовуються в галузях штучного інтелекту, інших областях науки, таких як біоінформатика, а також в платіжних системах, телекоммунікаціях.
З огляду на вузьку спеціалізованість функціонального та логічного програмування і недоречність їх застосування в предметній області, подальший розгляд цієї парадигми не є доречним.

Можна зробити висновок, що оскільки при розробці та підтримці програми в предметній області виникатиме необхідність багаторазово використовувати одні й ті самі частини коду(процедури), використання функціональної
парадигми є недоречним, а необхідності рівня абстракцій об'єктно-орієнтованого підходу немає, то найкращою моделью для предметної області є саме процедурна парадигма програмування, яка і буде використовуватися для розробки.

3.3. Вибір мови програмування

Для реалізації клієнтської сторони інтерфейсу користувача був використаний фреймворк Vue.js, який є одним з найпопулярніших фреймворків на сьогоднішній день. Компонентний підхід фреймворку Vue.js є реалізацією процедурної парадигми програмування, суть якої полягає у можливості об'єднання коду в блоки, призначені та готові для майбутнього використання шляхом їхнього виклику там, де вони необхідні. Це дозволяє не дублювати той самий фрагмент коду по кілька разів. Компоненти дозволяють з легкістю та простотою створювати веб-додатки, що складаються зі складових частин, які можна використовувати повторно. На додаток одні компоненти можуть у собі утримувати інші, що є великою перевагою, оскільки дозволяє значно знизити час необхідне програмісту на написання коду, а сам код при цьому лише виграє у простоті. Ще одним важливим плюсом такого підходу є легкість у зміні коду, що на практиці найчастіше виявляється критичною необхідністю коли виникає потреба додавання або зміни функціоналу програми. Оскільки замість необхідності багаторазово робити ті самі зміни в коді з ризиком пропустити і не помітити частини, функціонал яких потрібно змінити - у нас з'являється можливість змінити логіку скрізь, де це потрібно шляхом одиничного внесення змін до самої

процедури. Одним з ключових фактором у виборі даного JavaScript-фреймворку є його здатність підтримувати асинхронні методи програмування, що дозволяють позбавитися необхідності в оновлення всієї веб-сторінки при кожному запиті. Що дає нам зміну зображення на дошці у режимі реального часу. Даний фреймворк повністю підходить для написання простого, надійного та відповідного всім вимогам системи дистанційного навчання "Віртуальна дошка".

Висновки до розділу

Веб-програма "Віртуальна дошка" розроблена за допомогою мови програмування JavaScript, задіяний фреймворк Vue.js, використаний компонентний підхід. Завдяки компонентному підходу ми можемо розбити файли проекту на окремі менші файли, які будуть взаємодіяти між собою. Це перевага для розробника, так як функціонал кожного компонента буде описаний у відповідному компоненті файлі.

Фреймворк Vue.js є ефективним фреймворком для відображення елементів у веб-браузері, швидкість відображення елементів дуже висока, порівняно з іншими фреймворками, вибір саме Vue обґрунтований у цій роботі тим, що йдеться про віртуальну дошку, завдання якої відображати графічні зміни в реальному. часу, де швидкість відображення елементів грає важливу роль.

Вирішено використовувати базу даних моделі NoSQL, а саме хмарне зберігання даних на сервісі Firebase. Ця база даних дозволяє оновлювати дані реальному часі рахунок синхронізації даних. Також на цьому сервісі розташована база даних користувачів, яка дозволяє нам взаємодіяти з авторизацією користувача у програмі.

4. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Для виключення помилок у додатку, на етапі розробки ПЗ було проведено тестування, в якому додаток перевіряється на його функціональність, зручність використання та сумісність. Тестування веб-додатку включало написання тест-кейсів по розробленому функціоналу сервісу. Результати проведених тест-кейсів відображені у таблиці 4.1.

Таблиця 4.1 – Тест-кейси

Тест-кейс №1 Авторизація	
Кроки:	1. Натиснути на кнопку «Увійти»
	2. Заповнити поле «Email»
	3. Заповнити поле «Password»
	4. Натиснути кнопку «Send»
Очікуваний результат	Авторизація успішна Відображається алерт «Привіт, викладач»
Тест-кейс №2 Вихід з облікового запису	
Передумови:	Авторизуватись
Кроки:	1. Натиснути кнопку «Вийти»
Очікуваний результат	З облікового запису розлогінено
Тест-кейс №3 Додавання тексту на дошку	
Передумови:	Авторизуватись
Кроки:	1. Натиснути на іконку тексту
	2. Заповнити поле текстової інформації
	3. Натиснути іконку «Галочку»
Очікуваний результат	Текст відображається на дошці
Тест-кейс №4 Видалення тексту з дошки	
Передумови:	Авторизуватись
Кроки	1. Натиснути на іконку тексту
	2. Заповнити поле текстової інформації 3. Натиснути іконку «Хрестик»
Очікуваний результат	Написаний текст видалений з дошки
Тест-кейс №5 Додавання зображення на дошку	
Передумови:	Авторизуватись
Кроки:	1. Натиснути на іконку зображення
	2. Додати зображення
	3. Натиснути іконку «Галочку»
Очікуваний результат	Картинка відображається на дошці
Тест-кейс №6 Видалення зображення з дошки	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку зображення

Продовження табл. 4.1.

	2. Додати зображення
	3. Натиснути іконку «Хрестик»
Очікуваний результат	Додане зображення видалено
Текс-кейс №7 Додавання квадрата на дошку	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку квадрат
	2. Натиснути іконку «Галочку»
Очікуваний результат	Квадрат відображається на дошці
Тест-кейс №8 Видалення квадрату з дошки	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку квадрат
	2. Натиснути іконку «Хрестик»
Очікуваний результат	Квадрат видалено з дошки
Тест- кейс №9 Додавання трикутника на дошку	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку трикутник
	2. Натиснути іконку «Галочку»
Очікуваний результат	Трикутник відображається на дошці
Текс-кейс №10 Видалення трикутника з дошки	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку трикутник
	2. Натиснути іконку «Хрестик»
Очікуваний результат	Трикутник видалено з дошки
Тест-кейс №11 Додання кола на дошку	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку коло
	2. Натиснути іконку «Галочку»
Очікуваний результат	Коло відображається на дошці
Тест-кейс №12 Видалення кола з дошки	
Передумови	Авторизуватись
Кроки	1. Натиснути на іконку коло
	2. Натиснути іконку «Хрестик»
Очікуваний результат	Коло видалене з дошки
Тест-кейс №13 Очищення дошки	
Передумови	Авторизуватись
Кроки	1. Додати один або декілька елементів на дошку
	2. Натиснути іконку корзина
Очікуваний результат	Доданий елемент видалений

Продовження табл. 4.1.

Передумови	Авторизуватись
Кроки	1. Додати один або декілька елементів на дошку
	2. Натиснути іконку збереження
Очікуваний результат	Доданий елемент збережений
Тест-кейс №15 Перенесення елемента на дошці	
Передумови	Авторизуватись
Кроки	1. Вибрати один із елементів
	2. Перенести елемент на потрібне місце
Очікуваний результат	Перенесений елемент відображається на потрібному місці

Висновки до розділу

Під час проведення тест-кейсів на перевірку функціоналу програми були перевірені всі можливі варіанти використання програмного забезпечення, при проведенні тест-кейсів помилок або багів не виявлено.

ВИСНОВКИ

У результаті проведеної роботи була створена веб-платформа, яка повністю відповідає постановці задачі і виконує такі функції:

1. Робота з віртуальною дошкою у реальному часі, додавання елементів на дошку, видалення та відображення їх на дошці для інших користувачів без оновлення сторінки.
2. Завантаження віртуальної дошки до себе на пристрій у форматі зображення.
3. Надання платформи для зберігання наукових статей, навчальних матеріалів, інформаційних блоків для перегляду студентами.

Огляд головної сторінки розробленої платформи дистанційного навчання.

Рисунок 6.1 – Головна сторінка веб-сайту “Віртуальна дошка”

На головній сторінці веб-сайту (рис.6.1) розміщена вступна інформація для користувача, з визначенням терміна віртуальна дошка. Нижче три кнопки, які мають посилання на три кімнати веб-додатку "Віртуальна дошка", перехід здійснюється з відкриттям нової вкладки. Також на веб-сайті, в розділі меню є сторінка "Категорія курсів", на якій розташовані сторінки з навчальним матеріалом для студента, ці сторінки може редагувати викладач або додавати свої. На веб-сайті є розділ "Як це працює?", в якому можна знайти текстові та відео-інструкції по роботі з веб-додатком "Віртуальна дошка".

Проект був розроблений та реалізований за допомогою фреймворку Vue 3, на HTML, CSS, JavaScript у середовищі Visual Studio Code. Для бази даних використовувався Google Firebase.

ЛІТЕРАТУРА

1. Коваленко, О.С. Проектування інформаційних систем / Добровська, Л.М. Київ, 2020. – 190 с.
2. Ганчетт, Е. Vue.js в действии / Ганчетт, Е. , Листуон Б. Санкт-Петербург, 2020. – 304 с.
3. Бартлетт, Д. WordPress для начинающих / Д. Бартлетт [пер. с англ. М.А. Райтман]. – Москва : Вид-во «Э», 2017. – 208 с.
4. Павловская, Т. А. C/C++. Процедурное и объектно-ориентировочное программирование / Павловская Т.А. – Санкт-Петербург: Питер, 2015. – 496 с
5. Парадигма програмування [Електронний ресурс] // Вікіпедії. – 2022. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Парадигма_програмування.
6. Парадигмы программирования — парадигмы жизни [Електронний ресурс] // Хабр. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/sandbox/143062/.
7. Vue Documentation [Електронний ресурс] // Vue. – 2022. – Режим доступу до ресурсу: https://vuejs.org.
8. Oracle NoSQL Database: Real-Time Big Data Management for the Enterprise / A.Maqsood, M. Aalok, K. Chaitanya, J. Ashok. – New York: McGraw-Hill Education, 2013. – 256 с.
9. Firebase Documentation [Електронний ресурс] // Google Developers. – 2022. – Режим доступу до ресурсу: https://firebase.google.com/docs.
10. Wieruch, R. The Road to Firebase: Your journey to master Firebase in JavaScript / Robin Wieruch. – Berlin: Independently published, 2018. – 199 с.
11. JSON [Електронний ресурс] // Wikipedia. – 2022. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JSON.
12. Damiano Fusco. Large Scale Apps with Vue 3 and TypeScript 2020.– 80 с.

ДОДАТКИ

Текст програми

App.vue

```
<template>
  <!-- <div id="nav"><router-link
to="/">Home</router-link> |</div> -->
  <Header />
  <router-view />
  <ModalForm />
</template>

<script>
import Header from
"./components/Header.vue";
import ModalForm from
"./components/ModalForm.vue";

export default {
  components: { Header, ModalForm },
  setup() {},
};
</script>

<style lang="scss">
@import "@/assets/scss/reset.scss";
@import "@/assets/scss/common.scss";
</style>
```

Main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import store from "./store";

createApp(App).use(store).use(router).mo
unt("#app");
```

…/components/Canvas.vue

```
<template>
  <div
    class="wrapper-canvas"
    @mousemove="moveElement"
    @mouseup="stopDragAndDrop"
  >
    <EditorBox
      :posX="posX"
      :posY="posY"
      :clearCanvas="clearCanvas"
      :updatePos="updatePos"
      :addTextToCanvas="addTextToCanvas"

:addImageToCanvas="addImageToCanvas"
      :onResetClass="onResetClass"
    />
    <canvas class="canvas" width="1095"
height="600" ref="canvas"></canvas>
  </div>
</template>

<script>
import { ref, onMounted, watch, computed
} from "vue";
import { useStore } from "vuex";

import EditorBox from
"@/components/EditorBox.vue";

import { getData, sendData, getValue }
from "@/service/database";
import { getParams } from "@/utils";

export default {
  name: "Canvas",
  components: { EditorBox },
  props: {
    resetClass: Function,
  },
  setup(props) {
    const store = useStore();
    let posX = ref(0);
    let posY = ref(0);
    let canvas = ref(null);
    let ctx = ref(null);
    let urlParams = ref(1);
    urlParams = getParams();

    const updatePos = (x = 0, y = 0) =>
{
      posX.value = x;
      posY.value = y;
    };

    const typeSelect = computed(() =>
store.getters.GET_EDITOR_SETTINGS.type);

    const moveElement = (e) => {
      if
(!store.getters.GET_DRAG_AND_DROP)
return;
      const { layerX, layerY } = e;
      const { x, y } =
store.getters.GET_OFFSET_ELEMENT;
      updatePos(layerX - x, layerY - y);
    };

    const stopDragAndDrop = () => {

store.dispatch("SET_DRAG_AND_DROP",
false);
    };

    const addTextToCanvas = (text, x =
0, y = 0) => {
      ctx.value.font = "18px Roboto";
```

```
      ctx.value.fillText(text, x, y);
      sendImage();
    };

    getData(urlParams).then((data) => {
      if (data) {
        const { image, width, height } =
data;
        addImageToCanvas(image, 0, 0,
width, height, false);
      }
    });

    const sendImage = () => {
      sendData(urlParams, {
        image: dataURL(),
        id: Date.now(),
        width: canvas.value.width,
        height: canvas.value.height,
      });
    };

    const addImageToCanvas = (
      url,
      x = 0,
      y = 0,
      w = 300,
      h = 300,
      flag = true
    ) => {
      let img = new Image();
      img.src = url;
      img.onload = function () {
        ctx.value.drawImage(img, x, y,
w, h);
        if (flag) sendImage();
      };
    };

    const clearCanvas = () => {
      ctx.value.clearRect(0, 0,
canvas.value.width,
canvas.value.height);
      sendData(urlParams, null);
    };

    const onResetClass = () => {
      props.resetClass();
    };

    const dataURL = () =>
canvas.value.toDataURL("image/png");

    const downloadImage = () => {
      const link =
document.createElement("a");
      link.href = dataURL();
      link.download = "my-image.png";
      link.click();
    };

    watch(typeSelect, (currentValue) =>
{
      if (currentValue === "delete") {
        onResetClass();
        return clearCanvas();
      }
      if (currentValue === "download") {
        onResetClass();
        return downloadImage();
      }
    });

    onMounted(() => {
      ctx.value =
canvas.value.getContext("2d");
      getValue();
    });

    return {
      addTextToCanvas,
      clearCanvas,
      onResetClass,
      addImageToCanvas,
      moveElement,
      stopDragAndDrop,
      updatePos,
      posX,
      posY,
      canvas,
    };
  },
};
</script>

<style lang="scss">
.wrapper-canvas {
  position: relative;
  overflow: hidden;
  flex-grow: 2;
}
.canvas {
  border: 1px solid #ccc;
}
</style>
```

…/components/DrawingTools.vue

```
<template>
  <ul class="list-icon">
    <li class="list-item" v-for="option
in options" :key="option.icon">
      <button
        type="button"
        class="list-btn"
        @click="updateDrawing(option)"
        :disabled="option.disabled &&
!isUser"
        :class="{ current:
option.current && editorSettings.type }"
      >
        <img
:src="addImage(option.icon)"
alt="images" width="40" />
      </button>
    </li>
  </ul>
</template>

<script>
```

```
import { computed } from "vue";
import { useStore } from "vuex";

export default {
  name: "DrawingTools",
  props: {
    options: Array,
    resetClass: Function,
  },
  setup(props) {
    const store = useStore();

    const isUser = computed(() =>
store.getters.GET_USER);
    const option = computed(() =>
props.options);

    const editorSettings = computed(()
=> store.getters.GET_EDITOR_SETTINGS);

    const addImage = (img) =>
require("@/assets/images/" + img);

    const updateDrawing = (data) => {
      props.resetClass(data.icon);
      data.current = !data.current;
      const type = data.current ?
data.type : "";

store.dispatch("SET_EDITOR_SETTINGS",
type);
    };

    return {
      updateDrawing,
      addImage,
      editorSettings,
      option,
      isUser,
    };
  },
};
</script>

<style lang="scss">
.list-item:not(:last-child) {
  margin-bottom: 10px;
}
.list-icon {
  padding: 0 15px;
}
.list-btn {
  border: 1px solid #ccc;
  padding: 10px;
  display: block;
  &.current {
    background: #2195f33a;
    border: 1px solid #2196f3;
  }
  &:disabled {
    opacity: 0.5;
    background: #ebebeb;
  }
}
</style>
```

…/components/EditorBox.vue

```
<template>
  <div
    v-show="showEditor"
    class="input-box"
    :style="{
      transform: 'translate(' +
position.x + 'px,' + position.y + 'px)',
    }"
  >
    <ul class="d-flex edit-list">
      <li>
        <button
          type="button"
          class="circle-btn move-btn"
          @mousedown="startDragAndDrop"
        ></button>
      </li>
      <li class="save-item">
        <button type="button"
class="circle-btn"
@click="addShapeToCanvas">
          &#10004;
        </button>
      </li>
      <li class="close-item">
        <button type="button"
class="circle-btn" @click="closeEditor">
          &#10006;
        </button>
      </li>
    </ul>
    <input
      type="text"
      class="input-elem"
      v-show="type === 'text'"
      v-model="value"
    />
    <div class="box-figure" v-
show="typeFigure || userImg">
      <img :src="addImage" alt="figure"
/>
    </div>
    <input type="file" class="file"
ref="file" @change="loadImg" />
  </div>
</template>

<script>
import { computed, ref, watch } from
"vue";
import { useStore } from "vuex";
import { renderImage } from "@/utils";

export default {
  name: "EditorBox",
  props: {
    posX: Number,
    posY: Number,
    updatePos: Function,
```

```
    onResetClass: Function,
    addTextToCanvas: Function,
    addImageToCanvas: Function,
  },
  setup(props) {
    const store = useStore();
    const value = ref("");
    let file = ref(null);
    let userImg = ref(null);

    const figure = ["square",
"triangle", "circle"];
    const elements = [...figure,
"text"];

    const position = computed(() => {
      return {
        x: props.posX,
        y: props.posY,
      };
    });

    const type = computed(() =>
store.getters.GET_EDITOR_SETTINGS.type);
    const typeFigure = computed(() =>
figure.includes(type.value));

    const showEditor = computed(
      () =>
elements.includes(type.value) ||
userImg.value
    );

    const startDragAndDrop = (e) => {
      const { offsetX, offsetY } = e;

store.dispatch("SET_DRAG_AND_DROP",
true);

store.dispatch("SET_OFFSET_ELEMENT", {
x: offsetX, y: offsetY });
    };

    const closeEditor = () => {

store.dispatch("SET_EDITOR_SETTINGS",
"");
      props.onResetClass();
      props.updatePos();
      userImg.value = null;
    };

    const addImage = computed(() => {
      if (typeFigure.value) {
        return
require("@/assets/images/" + type.value
+ ".png");
      }
      return userImg.value;
    });

    const addShapeToCanvas = () => {
      const x = props.posX + 5;
      const y = props.posY + 30;
      if (typeFigure.value) {

props.addImageToCanvas(addImage.value,
x, y);
      }

      if (type.value === "text") {
        if (value.value)
props.addTextToCanvas(value.value, x +
15, y + 25);
        value.value = "";
      }
      if (type.value === "img") {

props.addImageToCanvas(userImg.value, x,
y);
      }
      closeEditor();
    };

    const loadImg = () => {

renderImage(file.value.files[0]).then((i
mg) => {
        userImg.value = img;
      });
    };

    watch(type, (currentValue) => {
      if (currentValue === "img") {
        file.value.click();
      }
    });

    return {
      addShapeToCanvas,
      startDragAndDrop,
      closeEditor,
      typeFigure,
      showEditor,
      userImg,
      loadImg,
      addImage,
      position,
      value,
      file,
      type,
    };
  },
};
</script>

<style lang="scss">
.input-box {
  background: transparent;
  position: absolute;
  width: 300px;
  padding: 5px;
  left: 0;
  top: 0;
  transition: transform 0.05s ease;
}
.input-elem {
  width: 100%;
  height: 40px;
  margin: 0;
  background: transparent;
```

```
  border: 1px solid #ccc;
  padding: 0 15px;
  &:focus {
    border-color: #2196f3;
  }
}

.edit-list {
  padding-bottom: 5px;
}

.circle-btn {
  width: 20px;
  height: 20px;
  border: none;
  justify-content: center;
  align-items: center;
  border-radius: 50%;
  display: block;
}
.save-item {
  margin-left: auto;
  margin-right: 5px;
  & button:hover,
  & button:focus {
    color: rgb(0, 255, 17);
  }
}
.close-item button:hover,
.close-item button:focus {
  color: rgb(206, 37, 37);
}
.move-btn {
  cursor: move;
}
.box-figure {
  width: 300px;
  height: 300px;
  border: 1px solid #ccc;
}
.file {
  width: 1px;
  height: 1px;
  overflow: hidden;
  opacity: 0;
  margin-top: -1px;
}
</style>
```

…/components/Header.vue

```
<template>
  <header class="header">
    <div class="container">
      <div>
        <p>Кімната {{ room }}</p>
        <h1 class="title">{{ title
}}</h1>
      </div>

      <button type="button" class="btn-
auth" @click="toggleRole">
        {{ textBtn }}
      </button>
    </div>
  </header>
</template>

<script>
import { computed, ref } from "vue";
import { useStore } from "vuex";

import { logOff } from "@/service/auth";
import { getParams } from "@/utils";

export default {
  name: "Header",
  setup() {
    const store = useStore();
    let room = ref(null);
    room = getParams();

    const isUser = computed(() =>
store.getters.GET_USER);
    const title = computed(() =>
(isUser.value ? "Викладач" :
"Студент"));
    const textBtn = computed(() =>
(isUser.value ? "Вийти" : "Увійти"));

    const toggleRole = () => {
      if (isUser.value) {
        logOff();
      } else {

store.dispatch("TOGGLE_SHOW_FORM",
false);
      }
    };
    store.dispatch("GET_USER");
    return { toggleRole, title, textBtn,
room };
  },
};
</script>

<style lang="scss">
.header {
  padding: 20px 0;
  border-bottom: 1px solid #ececec;
  .container {
    display: flex;
    justify-content: space-between;
    align-items: center;
  }
}

.btn-auth {
  cursor: pointer;
  padding: 10px 32px;
  border: none;
  color: #fff;
  font-size: 16px;
  font-weight: 600;
  border-radius: 4px;
  background: #2196f3;
  &:hover,
  &:focus {
```

```
    background: #188ce8;
  }
}
</style>
```

…/components/ModalForm.vue

```
<template>
  <div class="wrapper" :class="{ 'is-
hidden': isShowForm }">
    <form class="form"
@submit.prevent="getData">
      <button type="button"
class="close-btn" @click="hiddenForm">
        &#10006;
      </button>
      <div>
        <input
          type="email"
          required
          name="email"
          placeholder="Email"
          class="input-elem"
          v-model.trim="email"
        />
      </div>
      <div>
        <input
          type="password"
          required
          name="password"
          placeholder="password"
          class="input-elem"
          v-model.trim="password"
        />
      </div>

      <button type="submit" class="send-
btn">send</button>
    </form>
  </div>
</template>

<script>
import { ref, computed } from "vue";
import { useStore } from "vuex";
import Notiflix from "notiflix";

export default {
  name: "ModalForm",
  setup() {
    const store = useStore();

    let email = ref("");
    let password = ref("");

    const hiddenForm = () => {
      store.dispatch("TOGGLE_SHOW_FORM",
true);
      password.value = "";
      email.value = "";
    };

    const getData = () => {
      store
        .dispatch("VALIDATE_USER", {
          password: password.value,
          email: email.value,
        })
        .then(() => {

Notiflix.Notify.success("Привіт,
викладач!");
        })
        .catch(() => {
          Notiflix.Notify.warning("Упс!
Щось пішло не так...");
        })
        .finally(() => {
          hiddenForm();
        });
    };

    const isShowForm = computed(() =>
store.state.showForm);

    return { hiddenForm, getData, email,
password, isShowForm };
  },
};
</script>

<style lang="scss">
.wrapper {
  width: 100vw;
  height: 100vh;
  position: fixed;
  z-index: 100;
  left: 0;
  top: 0;
  background: #21212129;
  display: flex;
  justify-content: center;
  align-items: center;
  transition: opacity 0.5s ease;
  &.is-hidden {
    opacity: 0;
    pointer-events: none;
  }
}
.form {
  background: #fff;
  border-radius: 7px;
  padding: 40px;
  position: relative;
  max-width: 500px;
  width: calc(100% - 30px);
  .input-elem {
    padding: 0 15px;
    width: 100%;
    border: 1px solid #ccc;
    height: 40px;
    border-radius: 4px;
    margin-bottom: 20px;
    outline: none;
    &:hover,
    &:focus {
```

```
      border: 1px solid #188ce8;
    }
  }
}
.close-btn {
  width: 30px;
  height: 30px;
  border-radius: 50%;
  border: 1px solid #ccc;
  position: absolute;
  right: 8px;
  top: 8px;
  font-size: 16px;
  color: #ccc;
  line-height: 30px;
  text-align: center;
  background: transparent;
  &:hover,
  &:focus {
    color: #188ce8;
  }
}

.send-btn {
  padding: 10px 32px;
  border-radius: 4px;
  border: none;
  margin: 0 auto;
  color: #fff;
  display: block;
  background: #2196f3;
  &:hover,
  &:focus {
    background: #188ce8;
  }
}
</style>
```

…/views/Home.vue

```
<template>
  <main>
    <section class="section">
      <div class="container">
        <div class="row">
          <DrawingTools
:options="options"
:resetClass="resetClass" />
          <Canvas
:resetClass="resetClass" />
        </div>
      </div>
    </section>
  </main>
</template>

<script>
import { reactive } from "vue";
import Canvas from
"@/components/Canvas.vue";
import DrawingTools from
"@/components/DrawingTools.vue";

export default {
  name: "Home",
  components: { Canvas, DrawingTools },
  setup() {
    let options = reactive([
      { icon: "font.png", type: "text",
current: false, disabled: true },
      { icon: "picture-icon.png", type:
"img", current: false, disabled: true },
      { icon: "square.png", type:
"square", current: false, disabled: true
},
      {
        icon: "triangle.png",
        type: "triangle",
        current: false,
        disabled: true,
      },
      { icon: "circle.png", type:
"circle", current: false, disabled: true
},
      { icon: "bin.png", type: "delete",
current: false, disabled: true },
      {
        icon: "download.png",
        type: "download",
        current: false,
        disabled: false,
      },
    ]);

    const resetClass = (icon = "") => {
      options = options.map((item) => {
        if (item.icon !== icon)
item.current = false;
        return item;
      });
    };

    return { options, resetClass };
  },
};
</script>

<style lang="scss">
.section {
  padding: 20px 0;
}
</style>
```

…/store/index.js

```
import { createStore } from "vuex";

import { initializeApp } from
"firebase/app";
import { getAuth, onAuthStateChanged }
from "firebase/auth";
import { firebaseConfig } from
"@/config/firebase-config";
import { checkUserDetails } from
"@/service/auth";

initializeApp(firebaseConfig);
const auth = getAuth();
```

```
export default createStore({
  state: {
    offsetX: 0,
    offsetY: 0,
    showForm: true,
    user: null,
    dragAndDropElement: false,
    editorSettings: {
      type: "",
    },
  },
  getters: {
    GET_USER: (state) => {
      return state.user;
    },
    GET_OFFSET_ELEMENT: (state) => {
      return {
        x: state.offsetX,
        y: state.offsetY,
      };
    },
    GET_DRAG_AND_DROP: (state) => {
      return state.dragAndDropElement;
    },
    GET_EDITOR_SETTINGS: (state) => {
      return state.editorSettings;
    },
  },
  mutations: {
    TOGGLE_SHOW_FORM: (state, payload)
=> {
      state.showForm = payload;
    },
    SET_USER: (state, payload) => {
      state.user = payload;
    },
    SET_DRAG_AND_DROP: (state, payload)
=> {
      state.dragAndDropElement =
payload;
    },
    SET_OFFSET_ELEMENT: (state, payload)
=> {
      state.offsetX = payload.x;
      state.offsetY = payload.y;
    },
    SET_EDITOR_SETTINGS: (state, type)
=> {
      state.editorSettings = {
...state.editorSettings, type };
    },
  },
  actions: {
    SET_EDITOR_SETTINGS: ({ commit },
payload) => {
      commit("SET_EDITOR_SETTINGS",
payload);
    },
    TOGGLE_SHOW_FORM: ({ commit },
payload) => {
      commit("TOGGLE_SHOW_FORM",
payload);
    },
    VALIDATE_USER: (_, payload) => {
      return checkUserDetails(payload)
        .then((response) => response)
        .catch((error) => {
          throw new Error(error);
        });
    },
    GET_USER: ({ commit }) => {
      onAuthStateChanged(auth, (user) =>
{
        if (user) {
          commit("SET_USER", user);
        } else {
          commit("SET_USER", null);
        }
      });
    },
    SET_DRAG_AND_DROP: ({ commit },
payload) => {
      commit("SET_DRAG_AND_DROP",
payload);
    },
    SET_OFFSET_ELEMENT: ({ commit },
payload) => {
      commit("SET_OFFSET_ELEMENT",
payload);
    },
  },
});
```

…/utils/index.js

```
export const renderImage = (file) => {
  return new Promise((res) => {
    const reader = new FileReader();
    reader.onload = function (e) {
      res(e.target.result);
    };
    reader.readAsDataURL(file);
  });
};

export const getParams = () => {
  //?room=1
  const url = new URL(location.href);
  let room = 1;
  if (url.search) {
    room = url.search.split("=")[1];
  }
  return room;
};

let canvas = null;
let ctx = null;

export const addImageToCanvas = (url, w
= 300, h = 300) => {
  if (!canvas) canvas =
document.querySelector(".canvas");
  if (canvas && !ctx) ctx =
canvas.getContext("2d");
  let img = new Image();
  img.src = url;
  img.onload = function () {
    ctx.drawImage(img, 0, 0, w, h);
  };
```

```
};

export const clearCanvas = () => {
  ctx.clearRect(0, 0, canvas.width,
canvas.height);
};
```

…/router/index.js

```
import { createRouter, createWebHistory
} from "vue-router";
import Home from "../views/Home.vue";

const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
  },
];

const router = createRouter({
  history:
createWebHistory(process.env.BASE_URL),
  routes,
});

export default router;
```

…/service/auth.js

```
import { initializeApp } from
"firebase/app";
import { getAuth,
signInWithEmailAndPassword, signOut }
from "firebase/auth";

import { firebaseConfig } from
"@/config/firebase-config";

initializeApp(firebaseConfig);
const auth = getAuth();

export const checkUserDetails = ({
email, password } = {}) => {
  return
signInWithEmailAndPassword(auth, email,
password)
    .then((userCredential) => {
      const user = userCredential.user;
      console.log("user :>> ", user);
      return user;
    })
    .catch((error) => {
      throw new Error(error);
    });
};

export const logOff = () => {
  signOut(auth)
    .then(() => {})
    .catch(() => {});
};

//admin@gmail.com
//admin123456
```

…/service/database.js

```
import { initializeApp } from
"firebase/app";
import { firebaseConfig } from
"@/config/firebase-config";
import { getDatabase, set, ref, get,
onValue } from "firebase/database";

import { getParams, addImageToCanvas,
clearCanvas } from "../utils";

initializeApp(firebaseConfig);
const db = getDatabase();

export const getData = (payload) => {
  return get(ref(db, "room_" + payload))
    .then((snapshot) => {
      if (snapshot.exists()) {
        return snapshot.val();
      } else {
        return null;
      }
    })
    .catch((error) => {
      console.error(error);
    });
};

export const sendData = (payload, data =
{}) => {
  try {
    set(ref(db, "room_" + payload),
data);
  } catch (error) {
    throw new Error(error);
  }
};

const urlParams = getParams();
export const getValue = () => {
  onValue(ref(db, "room_" + urlParams),
(snapshot) => {
    const data = snapshot.val();
    if (data) {
      const { image, width, height } =
data;
      addImageToCanvas(image, width,
height);
    } else {
      clearCanvas();
    }
  });
};
```

…/config/firebase-config.js

```
export const firebaseConfig = {
```

```
  apiKey:
"AIzaSyCLmFPwuGDKDD2I_Yhi7A4CIqZdlRHKDpY
",
  authDomain: "drawing-
ba0e2.firebaseapp.com",
  databaseURL:
    "https://drawing-ba0e2-default-
rtdb.europe-west1.firebasedatabase.app",
  projectId: "drawing-ba0e2",
  storageBucket: "drawing-
ba0e2.appspot.com",
  messagingSenderId: "784399442719",
  appId:
"1:784399442719:web:d146fa4158bb1dd7a4c6
2a",
};
```