МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

ГЕЙКО ІГОР ВІКТОРОВИЧ

ДОСЛІДЖЕННЯ ПРИНЦИПІВ ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКІВ НА ПРИКЛАДІ РОЗРОБКИ ЕЛЕКТРОННОЇ ДОШКИ ДЛЯ ДИСТАНЦІЙНОГО НАВЧАННЯ

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
БУЛАТЕЦЬКИЙ ВІТАЛІЙ ВІКТОРОВИЧ,
кандидат фізико-математичних наук, доцент
кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп’ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК – 2025

ЗМІСТ

ВСТУП

Актуальність теми. Розробка вебінтерфейсів є однією з ключових сфер сучасної ІТ-індустрії, де зручність, доступність та інтерактивність вебзастосунків відіграють критично важливу роль у забезпеченні ефективної взаємодії користувачів з цифровими сервісами. В умовах стрімкого розвитку дистанційного навчання, особливо під впливом глобальної пандемії, зростає потреба у створенні високоякісних інструментів, що підтримують співпрацю в реальному часі. У цьому контексті електронні дошки стають важливим інструментом для організації динамічного та зручного навчального процесу. Вони не лише забезпечують візуалізацію інформації, а й дозволяють здійснювати синхронну взаємодію між учасниками незалежно від їхнього місцезнаходження. Дослідження принципів створення таких інтерактивних вебзастосунків є вкрай актуальним, адже сприяє розробці функціональних, інтуїтивно зрозумілих і надійних рішень для освітнього й професійного середовища.

Мета роботи – дослідження принципів проектування вебзастосунків на прикладі розробки електронної дошки для дистанційного навчання.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- вивчити основні принципи проектування вебзастосунків для дистанційного навчання;
- проаналізувати існуючі рішення та визначити їх переваги і недоліки;
- розробити електронну дошку для дистанційного навчання з урахуванням вимог до зручності та доступності;
- забезпечити інтуїтивно зрозумілий дизайн та функціональність вебінтерфейсу;
- провести тестування розробленого продукту.

Об'єкт дослідження – вебзастосунок електронної дошки для дистанційного навчання.

Предмет дослідження – процес проектування та розробки вебзастосунку електронної дошки для дистанційного навчання.

РОЗДІЛ 1
АНАЛІЗ ВЕБЗАСТОСУНКІВ ДЛЯ ДИСТАНЦІЙНОГО НАВЧАННЯ ТА ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ЇХ РОЗРОБКИ

1.1. Історія розвитку програмного забезпечення для дистанційного навчання

На сьогоднішній день дистанційне навчання стало невід'ємною частиною сучасної освітньої системи, особливо під час пандемії COVID-19, коли значну частину навчальних процесів перевели в онлайн-формат.

Вебсервіси, що забезпечують доступ до онлайн-курсів, електронних матеріалів та інтерактивних інструментів, стали необхідними для підтримки навчального процесу.

Перші кроки в розвитку дистанційного навчання розпочалися ще в 1960-х роках, коли з'явилася перша система електронного навчання PLATO (Programmed Logic for Automated Teaching Operations). Ця система стала революційною, дозволяючи студентам використовувати комп'ютери для навчання та взаємодії. [1]

З того часу технології дистанційного навчання постійно удосконалювались, і з розвитком Інтернету з'явились платформи на зразок Moodle, Blackboard, а також сучасні інструменти для організації вебінарів, такі як Zoom, Google Meeting та інші.

Серед сучасних платформ для дистанційного навчання важливе місце займають електронні дошки, які використовуються для інтерактивного спілкування між викладачами та студентами в режимі реального часу. Ці дошки дають можливість записувати, редагувати та обмінюватися інформацією в реальному часі. Зокрема, такі інструменти, як Liveboard, Jamboard, Whiteboard.fi, дозволяють організувати онлайн-заняття з використанням інтерактивних дошок, що є важливою частиною процесу онлайн-навчання.

Дослідження принципів проектування вебзастосунків для таких платформ є важливим етапом у створенні ефективних інструментів для дистанційного навчання, що відповідають вимогам сучасних освітніх стандартів.

1.2. Архітектурні підходи до розробки вебзастосунків

Архітектура вебзастосунку визначає його структуру, взаємодію між компонентами та спосіб обробки запитів. Вибір архітектурного підходу впливає на продуктивність, масштабованість та гнучкість системи. У сучасній розробці вебзастосунків застосовують різні архітектурні підходи, і далі детальніше розглянемо їх особливості.

1.2.1. Монолітна архітектура

Монолітна архітектура є традиційним підходом до побудови вебзастосунків, де всі компоненти, такі як бізнес-логіка, інтерфейс користувача та база даних, інтегровані в один єдиний застосунок (рис. 1.1).

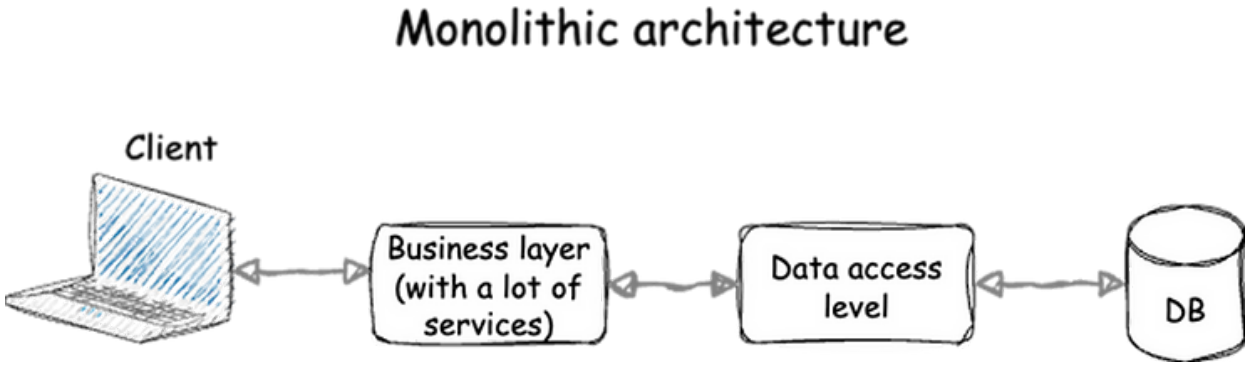

Рисунок 1.1 – Монолітна архітектура

Випадки використання включають малі та середні вебзастосунки, де простота може бути перевагою для проєктів з обмеженою складністю. Такий підхід також підходить для швидкого прототипування, оскільки дає змогу оперативно розробляти та розгортати початкові версії програмного

забезпечення. Крім того, він може використовуватися у застарілих системах, де монолітні застосунки залишаються в експлуатації протягом багатьох років. [2]

Водночас цей підхід має певні недоліки. Зі зростанням системи код стає громіздким і важким для підтримки. Будь-яка зміна в одній частині може вплинути на інші компоненти, що підвищує ризик виникнення помилок. [2]

1.2.2. Мікросервісна архітектура

Мікросервісна архітектура передбачає розподіл функціональності вебзастосунку на невеликі, незалежні сервіси, кожен із яких відповідає за окремий аспект системи (наприклад, управління користувачами, обробка платежів, управління контентом). Ці сервіси взаємодіють між собою через спеціальні протоколи (наприклад, через API) (рис. 1.2).

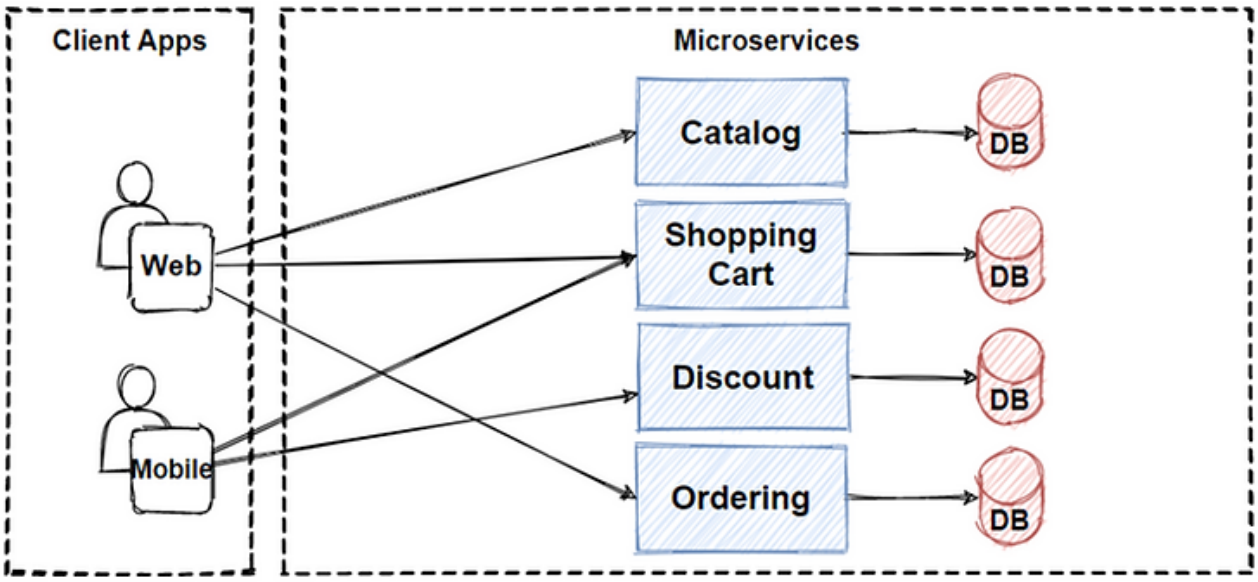

Рисунок 1.2 – Мікросервісна архітектура

Мікросервісна архітектура знаходить широке застосування в різних сферах. Наприклад, у маркетплейсах e-commerce кожен мікросервіс відповідає за окремі функції, такі як керування користувачами, каталог товарів, платежі та обробку замовлень. Подібний підхід використовується і в застосунках для райдшерингу, де окремі сервіси займаються аутентифікацією користувачів,

обробкою запитів на поїздки, відстеженням водіїв і проведенням платежів. Також мікросервіси активно застосовуються у платформах потокового мовлення, забезпечуючи доставку контенту, управління профілями користувачів, надання рекомендацій і виставлення рахунків. [2]

Попри численні переваги, ця архітектура має й певні недоліки. Одним із головних викликів є складність управління розподіленою системою, що потребує ретельного адміністрування. Додатково, комунікація між сервісами створює додаткове навантаження, що здатне негативно впливати на загальну продуктивність системи. [2]

1.2.3. Клієнт-серверна архітектура

Клієнт-серверна архітектура є основою більшості сучасних вебзастосунків. У цьому підході вебклієнт (наприклад, застосунок на основі React.js) взаємодіє із серверною частиною (наприклад, на базі NestJS) через HTTP-запити, WebSocket-з'єднання, тощо (рис. 1.3).

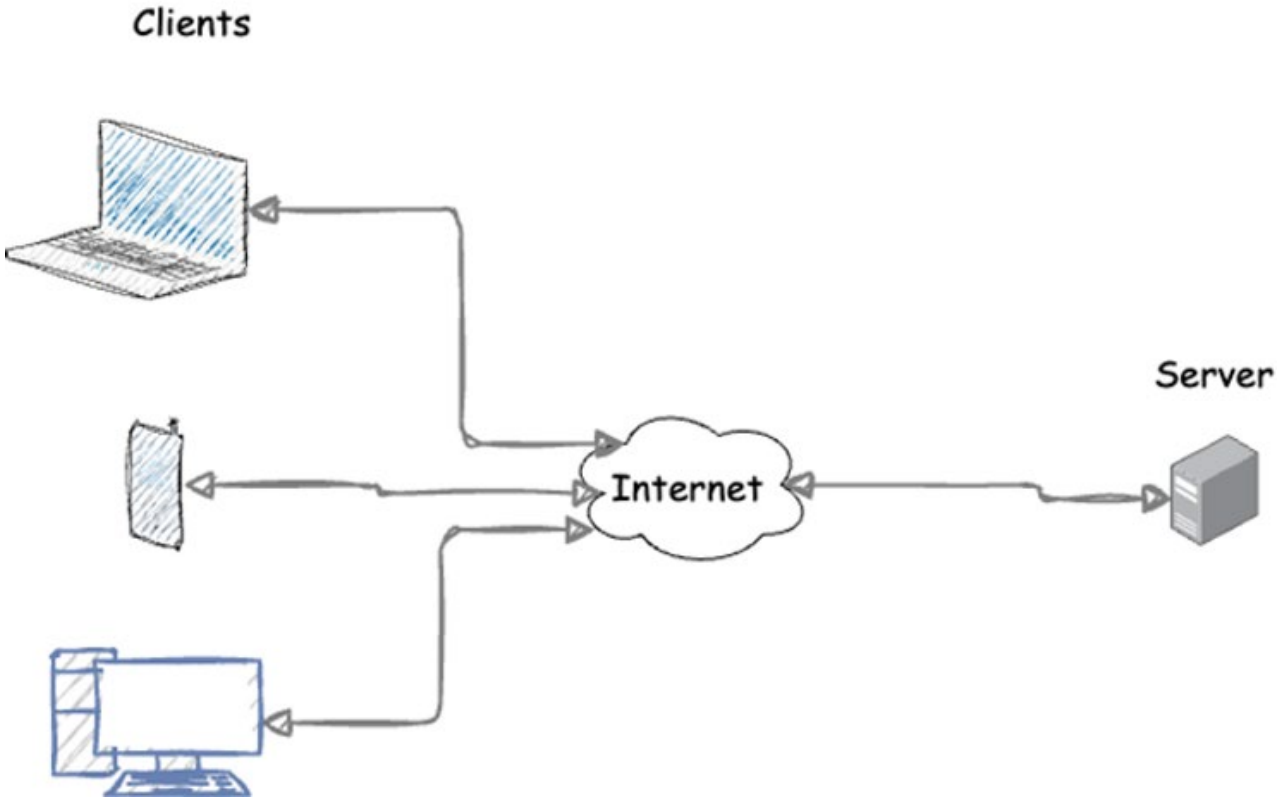

Рисунок 1.3 – Клієнт-серверна архітектура

Архітектура клієнт-сервер широко використовується в різних сферах. Наприклад, у системах електронної пошти клієнти надсилають запити на отримання або відправлення листів через центральний сервер. Подібний принцип застосовується в онлайн-іграх, де гравці взаємодіють із сервером у режимі реального часу, отримуючи оновлення гри та координуючи дії з іншими учасниками. [2]

Однак ця архітектура має певні недоліки. Однією з основних проблем є масштабованість сервера, оскільки під час високого навантаження його продуктивність може знижуватися. Крім того, ще одним ризиком є те, що сервер виступає єдиною точкою відмови, і його вихід з ладу може зробити систему недоступною для користувачів. [2]

1.2.4. Подійно-орієнтована архітектура (EDA)

Подійно-орієнтована архітектура наголошує на взаємодію між компонентами через асинхронні події, що викликаються діями користувача або змінами даних (рис. 1.4). [2]

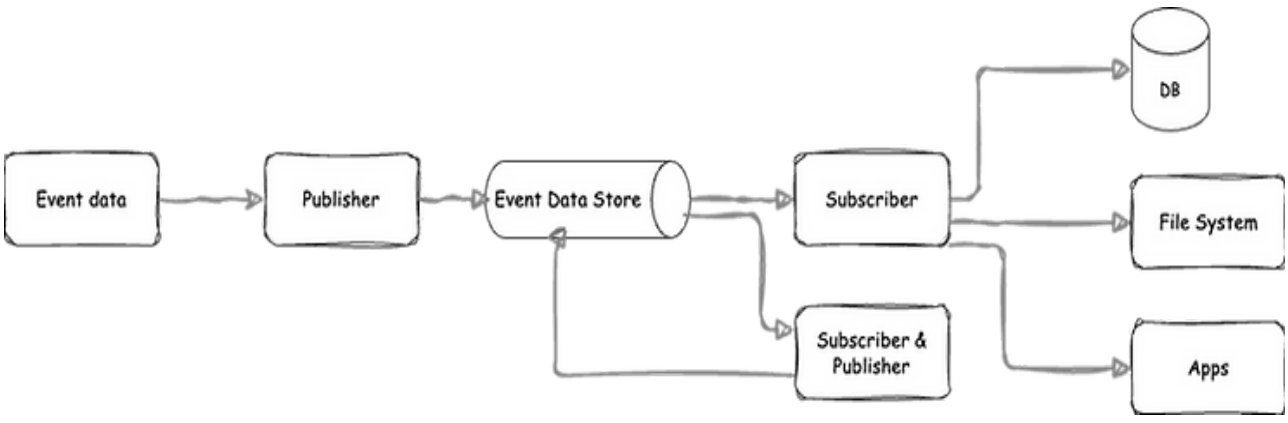

Рисунок 1.4 – Подійно-орієнтована архітектура (Event-driven architecture)

Архітектура обробки подій використовується в системах реального часу та графічних користувацьких інтерфейсах. Наприклад, у соціальних мережах дії користувачів, такі як публікації, лайки чи коментарі, викликають оновлення подій на платформі, що дозволяє відображати зміни в реальному часі. Також у

платформах для торгівлі акціями така архітектура дає змогу швидко реагувати на зміни ринку, виконуючи ордери на купівлю чи продаж в реальному часі, коли відбуваються події на ринку. [2]

Дана модель також має певні недоліки. Налагодження може бути складним через нелінійність потоків подій, що ускладнює виявлення причин помилок. Також порядок подій та їх таймінг можуть призводити до непередбачуваної поведінки, що важко прогнозувати. [2]

1.2.5. Сервіс-орієнтована архітектура (SOA)

Сервіс-орієнтована архітектура подібна до мікросервісної, але базується на взаємодії між великими модулями (сервісами), які можуть повторно використовуватися в різних частинах бізнес-системи. Вона використовує стандартизовані API (наприклад, SOAP або REST) для комунікації між сервісами (рис. 1.5).

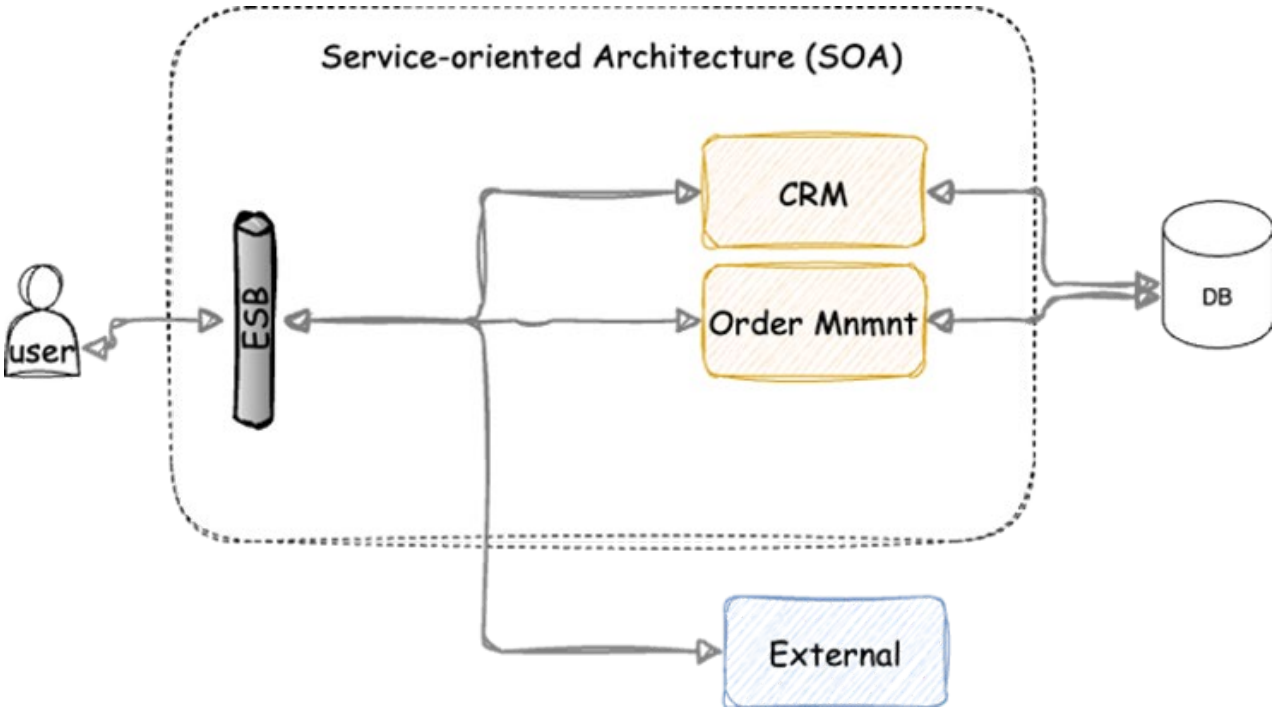

Рисунок 1.5 – Сервіс-орієнтована архітектура (SOA)

Архітектура орієнтована на сервіси (SOA) використовується в різних сферах для інтеграції та оптимізації процесів. У великих організаціях SOA

застосовується для інтеграції різних систем відділів, таких як HR, фінанси та продажі, що дозволяє забезпечити більш ефективну взаємодію між ними. В електронній комерції SOA дозволяє поєднувати послуги від різних постачальників, створюючи єдиний онлайн-шопінг досвід для користувачів. Крім того, ця архітектура може бути використана для інтеграції застарілих систем із новими технологіями, що дає змогу оновити інфраструктуру без необхідності переписування старих систем. [2]

Однак у SOA є деякі недоліки. Проектування та управління послугами може бути складним, оскільки вимагає налаштування великої кількості компонентів. Також існують додаткові витрати, пов’язані з мережею, оскільки послуги часто обмінюються даними через мережу, що може впливати на ефективність. [2]

Вибір архітектури вебзастосунку залежить від його функціональних та нефункціональних вимог. Для простих застосунків може бути достатньо монолітної архітектури, тоді як для масштабованих сервісів краще використовувати мікросервісний або сервіс-орієнтований підхід. Якщо система працює з подіями в реальному часі, доцільно впроваджувати подійно-орієнтовану архітектуру. У свою чергу, клієнт-серверна модель є основою більшості сучасних вебзастосунків, незалежно від обраної архітектурної парадигми.

1.3. Технології створення вебзастосунку електронної дошки для дистанційного навчання

Вебзастосунки в режимі реальному часу є сучасними інструментами, які забезпечують миттєву взаємодію між користувачами через інтернет. Вони знаходять застосування у різних сферах, таких як спільна робота, мультимедіа, онлайн-ігри та інші сервіси, де швидкість передачі даних є критичною.

Наведемо визначення, що таке вебзастосунок. Вебзастосунок – це розподілений застосунок, в якому клієнтом виступає браузер, а сервером –

вебсервер. Браузер може бути реалізацією так званих тонких клієнтів – логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у зображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. [3]

Прикладом таких вебзастосунків є інтерактивні системи для дистанційного навчання, зокрема, вебінтерфейси електронних дошок, що дозволяють користувачам працювати разом у реальному часі. Ці системи мають забезпечувати інтерактивність, низьку затримку, синхронізацію змін і зручність використання, що вимагає ретельно продуманої архітектури та вибору відповідних технологій. Розглянемо детальніше, як взагалі створюються такі вебзастосунки.

1.3.1. Frontend-технології

Фронтенд або клієнтська частина вебзастосунку – це інтерфейс, з яким безпосередньо взаємодіє користувач. Вона визначає не лише зовнішній вигляд програми, але й забезпечує динамічну поведінку елементів, обробку взаємодії, а також передачу даних до сервера. Розробка клієнтської частини сучасного вебзастосунку, зокрема електронної дошки для дистанційного навчання, ґрунтується на використанні сучасних технологій і фреймворків.

JavaScript, як одна з найпоширеніших мов програмування для веброзробки, має розвинену екосистему інструментів для створення інтерактивних інтерфейсів. Серед найпопулярніших JavaScript-інструментів для побудови інтерфейсів вирізняються React, Angular і Vue.js. React – бібліотека з відкритим кодом, створена компанією Meta, забезпечує компонентний підхід до побудови інтерфейсу та ефективну роботу з віртуальним DOM. Angular, повнофункціональний фреймворк від Google, дозволяє розробляти масштабовані односторінкові додатки з підтримкою TypeScript, ін’єкції залежностей та модульної архітектури. Vue.js – прогресивний фреймворк, що поєднує простоту використання з можливостями масштабування для великих проєктів. Вибір

конкретного фреймворку суттєво впливає на архітектуру застосунку, його масштабованість, зручність підтримки й швидкість розробки.

У контексті створення електронної дошки важливою складовою є реалізація графічного рендерингу. Для цього зазвичай використовується Canvas API – інтерфейс, що дозволяє малювати 2D-графіку безпосередньо в браузері за допомогою JavaScript. Елемент <canvas> у HTML5 створює «полотно», на якому можна реалізувати функції ручного малювання (через стилус або мишу), відображення анімацій, побудову графіків чи візуалізацію навчального матеріалу, а також обробку зображень у реальному часі. Canvas API не зберігає стан об’єктів після їх виведення, що забезпечує високий рівень контролю, але вимагає уважного керування станом сцени через імперативне програмування. [4]

Для потреб тривимірної візуалізації застосовують WebGL (Web Graphics Library) – низькорівневий API, що базується на OpenGL ES та дозволяє виводити 2D-графіку та 3D-графіку з апаратним прискоренням. WebGL інтегрується з елементом <canvas> і дає змогу створювати об’ємні сцени, працювати з текстурами та моделювати складні об'єкти. [5]

Альтернативним підходом до рендерингу є використання SVG (Scalable Vector Graphics) – мови розмітки, яка описує двовимірну векторну графіку у форматі XML. На відміну від Canvas, SVG оперує об’єктами, які мають DOM-представлення. Це дозволяє зручно анімувати окремі елементи, змінювати їхні властивості через CSS або JavaScript, а також обробляти події. SVG забезпечує масштабованість без втрати якості та особливо добре підходить для відображення схем, діаграм або інтерфейсних елементів, що потребують точної геометрії. [6]

Canvas і SVG можуть застосовуватися як окремо, так і спільно залежно від задачі. Canvas краще підходить для вільного малювання й високодинамічних сцен, тоді як SVG – для точних і керованих елементів інтерфейсу.

Окрім інструментів візуалізації, у сучасній фронтенд-розробці активно використовуються й інші технології. TypeScript як надмножина JavaScript забезпечує статичну типізацію, підвищуючи надійність і зрозумілість коду.

Бібліотеки для централізованого керування станом, такі як Redux, Zustand або Recoil, дозволяють зручно організовувати архітектуру складних застосунків. Для стилізації інтерфейсу застосовують CSS-препроцесори (SASS/SCSS) та утилітарні CSS-фреймворки, наприклад Tailwind CSS, які полегшують підтримку стилів і забезпечують гнучку кастомізацію.

1.3.2. Backend-технології

Backend або серверна частина вебзастосунку є фундаментом для забезпечення його функціональності та зберігання даних. Вона відповідає за обробку запитів від клієнтської частини, виконання бізнес-логіки, взаємодію з базами даних та забезпечення безпеки застосунку. Розробка серверної частини сучасного вебзастосунку, такого як електронна дошка для дистанційного навчання, спирається на широкий спектр технологій, вибір яких є важливим для продуктивності, масштабованості та надійності кінцевого продукту.

Для створення серверної частини вебзастосунків, зокрема електронної дошки, використовуються різні мови програмування, платформи та фреймворки. Вони забезпечують обробку даних, синхронізацію в реальному часі та взаємодію з клієнтською частиною. Серед поширених рішень: C# із фреймворком ASP.NET Core, Python із Django, JavaScript із платформою Node.js та фреймворками Express.js і NestJS. Однією з найпопулярніших платформ для розробки вебзастосунків є Node.js, тому розглянемо її детальніше.

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. В платформі використовується розроблений компанією Google рушій V8, що забезпечує швидке виконання JavaScript. Платформа Node.js призначена для виконання високопродуктивних мережевих застосунків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм. [7]

Основна частина Node.js написана на C і C++. Node.js базується на однопотоковій архітектурі циклу подій, що дозволяє Node.js обробляти численні запити клієнтів. Node.js використовує концепцію асинхронної моделі та неблокуючого введення-виведення, тобто цикл подій (event loop) (рис. 1.6). [8]

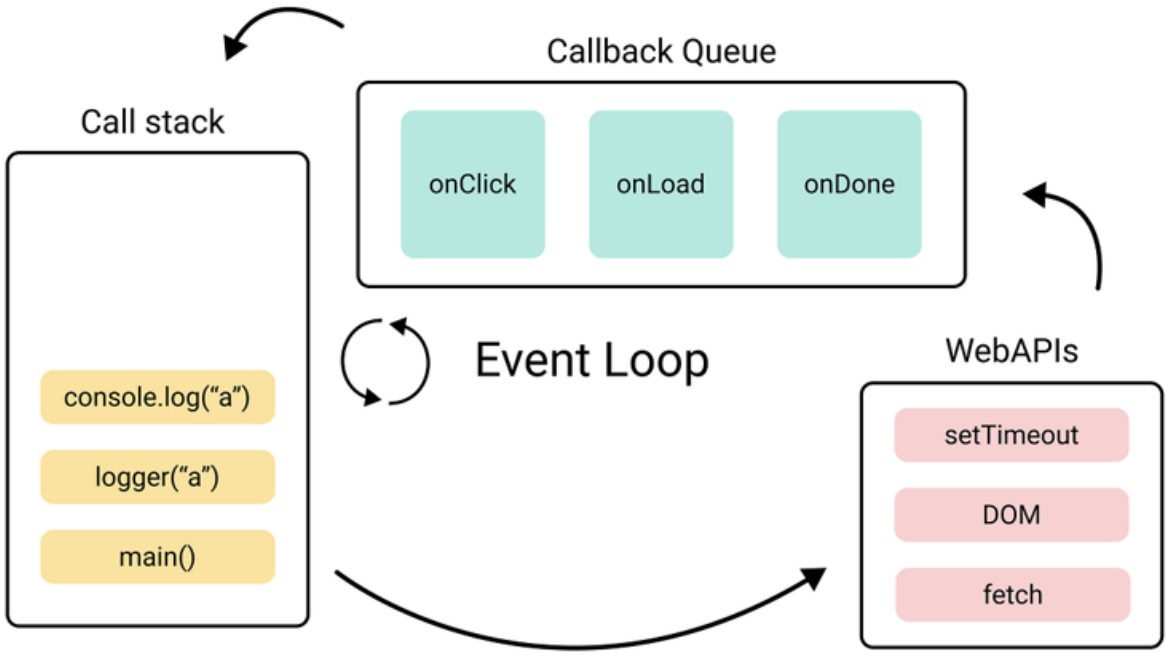

Рисунок 1.6 – Візуальне представлення «Event Loop»

У сучасній розробці серверних сервісів на Node.js використовуються різні фреймворки для оптимізації процесу створення вебсервісів. Серед популярних рішень – Express.js, Koa та Fastify, які є мінімалістичними та легковаговими. Проте для створення масштабованих і структурованих застосунків зручно використовувати NestJS.

Для організованого зберігання, обробки та керування даними використовують базу даних. База даних є основним компонентом більшості вебзастосунків. Існує два основних типи баз даних: реляційні (SQL) та нереляційні (NoSQL). Реляційні бази даних, такі як PostgreSQL, MySQL та Microsoft SQL Server, організовують дані у вигляді таблиць зі структурованими схемами та забезпечують ACID-транзакції, що гарантує цілісність даних.

Нереляційні бази даних, такі як MongoDB, Couchbase та Redis, пропонують більш гнучкі моделі даних (наприклад документи, ключ-значення, графові) та можуть краще підходити для певних типів даних та високонавантажених застосунків. Одним із найпотужніших рішень є PostgreSQL, що забезпечує надійність і гнучкість у роботі з великими обсягами даних.

У роботі з базами даних розробники часто використовують ORM (Object-Relational Mapping) для спрощення взаємодії з ними. ORM надає абстрактний шар, що дозволяє розробникам працювати з даними як з об'єктами мови програмування, замість написання SQL-запитів вручну. Популярними ORM є Django ORM для Python, Hibernate для Java, Eloquent для Laravel, а також Prisma та Sequelize для Node.js, які надають різні підходи до маніпуляції даними.

Забезпечення безпеки вебзастосунку є надзвичайно важливим, і однією з ключових її складових є авторизація та аутентифікація користувачів. Аутентифікація – це процес підтвердження особи користувача, а авторизація – це визначення прав доступу користувача до різних ресурсів застосунку. Існують різні підходи до реалізації авторизації та аутентифікації.

Розглянемо поширені методи аутентифікації. Базова аутентифікація (Basic Authentication) є одним з найпростіших методів аутентифікації. Вона передбачає надсилання імені користувача та пароля з кожним запитом. Зазвичай облікові дані кодуються за допомогою Base64 та включаються до заголовка HTTP. Попри свою простоту, базова аутентифікація має обмеження, такі як вразливість до перехоплення, якщо не використовується протокол HTTPS. [9]

Аутентифікація на основі токенів (Token-based Authentication) широко використовується завдяки своїй гнучкості та безпеці. Замість надсилання облікових даних з кожним запитом, після успішної аутентифікації генерується унікальний токен і надсилається клієнту. Потім цей токен включається до заголовка Authorization наступних запитів. Токени можуть бути короткочасними, що зменшує ризик, пов'язаний з довготривалими обліковими даними. [9]

OAuth (Open Authorization) це промисловий стандартний протокол аутентифікації, який використовується для надання стороннім програмам обмеженого доступу до ресурсів користувача без розкриття облікових даних. OAuth вводить поняття токенів доступу та токенів оновлення. Токени доступу надають конкретні дозволи на обмежений час, тоді як токени оновлення можуть використовуватися для отримання нових токенів доступу. [9]

Аутентифікація за допомогою ключів API (API Key Authentication) передбачає генерацію унікального ключа API для кожного клієнта, який включається до заголовка запиту. Цей метод часто використовується для аутентифікації та обмеження швидкості запитів. Ключами API легше керувати, ніж обліковими даними, і їх використання можна відстежувати з метою безпеки. [9]

Перейдемо до розгляду моделей авторизації. Однією з найпоширеніших моделей є контроль доступу на основі ролей (RBAC). Цей метод авторизує та обмежує доступ користувачів до системи на основі їхніх ролей в організації. Цей метод дозволяє користувачам отримувати доступ до даних і застосунків, необхідних для виконання їхніх посадових обов'язків, і мінімізує ризик несанкціонованого доступу співробітників до конфіденційної інформації або виконання неавторизованих завдань. [10]

Інша модель ABAC потрібна для досягнення більш точного контролю доступу. На відміну від RBAC, який надає доступ відповідно до попередньо визначених ролей, ABAC є політикою безпеки, що базується на комбінації атрибутів для зіставлення користувачів з ресурсами, необхідними їм для виконання роботи. [10]

Контроль доступу на основі політик (PBAC) є ще однією стратегією керування доступом, яка зосереджується на авторизації. У той час як RBAC обмежує доступ користувачів на основі статичних ролей, PBAC визначає права доступу динамічно на основі правил і політик. Хоча PBAC досить схожий на ABAC, ABAC вимагає більше розробницьких ресурсів (наприклад, кодування XML) зі збільшенням кількості необхідних атрибутів. [10]

Списки контролю доступу (ACL) контролюють або обмежують потік трафіку в цифровому середовищі. Організації використовують ACL у поєднанні з VPN для керування трафіком. Це може покращити продуктивність мережі, підвищити безпеку та забезпечити більш детальний моніторинг у точках входу та виходу. Це робить ACL придатними для захисту окремих користувачів і даних низького рівня, але це може бути не найкращим підходом до керування доступом у більшості бізнес-додатків. [10]

У той час як RBAC, також відомий як невибірковий контроль доступу, застосовує більш орієнтований на людину підхід до контролю доступу, вибірковий контроль доступу (DAC) використовує ACL для обмеження доступу до ресурсів, таких як файли та таблиці баз даних, і визначає, які права має користувач для цього ресурсу. За допомогою DAC кожен користувач контролює доступ до власних даних. Цей метод децентралізує рішення щодо безпеки, дозволяючи власникам ресурсів надавати або обмежувати доступ на свій розсуд. [10]

Кожна з розглянутих моделей має свої переваги та обмеження, і часто ефективне управління доступом передбачає поєднання кількох підходів для досягнення оптимального балансу між безпекою та зручністю використання.

У випадку розробки електронної дошки для дистанційного навчання найдоцільніше застосовувати саме модель RBAC. Вона дозволяє легко впровадити чіткий розподіл ролей між основними типами користувачів, а саме викладачами, студентами та адміністраторами. Такий підхід значно спрощує адміністрування доступу, забезпечує передбачувану логіку взаємодії з системою та масштабованість, що особливо важливо у середовищах з великою кількістю користувачів.

1.3.3. Мережеві протоколи

Для взаємодії між клієнтом і сервером застосовуються різні протоколи передачі даних. Протоколи HTTP/HTTPS чудово підходять для завантаження

сторінок і передачі даних, які не потребують частого оновлення. Такий підхід не дозволяє ефективно працювати в умовах, коли необхідно підтримувати постійний обмін даними, як у реальному часі. [11]

Для вирішення цієї проблеми було розроблено кілька підходів, які доповнюють можливості HTTP. Один із таких підходів називається Long Polling, який дозволяє клієнту надсилати запит до сервера, а сервер тримає це з'єднання відкритим до появи нових даних. Хоча це значно покращує взаємодію, Long Polling може створювати додаткове навантаження на сервер через велику кількість відкритих з’єднань. [12]

Інший підхід Server-Sent Events (SSE) дає можливість серверу надсилати оновлення клієнту в режимі реального часу, використовуючи HTTP-з’єднання. SSE добре підходить для передачі односторонніх потоків даних, наприклад, повідомлень чи оновлень статусу, але не підтримує двосторонньої комунікації. [13]

Для більш складних і вимогливих сценаріїв, які потребують миттєвого двостороннього обміну даними, використовуються спеціалізовані протоколи, такі як WebSocket. WebSocket дозволяє встановлювати постійне з’єднання між клієнтом і сервером, що дає змогу обмінюватися даними без затримки. Це ідеальне рішення для інтерактивних застосунків, таких як дошки для спільної роботи або онлайн-чати. [14]

Для мультимедіа-даних і прямої комунікації між клієнтами, наприклад, відеоконференцій, застосовується WebRTC. Цей протокол дозволяє передавати аудіо, відео або інші дані з низькою затримкою, використовуючи пряме з'єднання між клієнтами (P2P). [15]

Усі ці технології забезпечують створення високоякісних вебзастосунків, які працюють у реальному часі та задовольняють потреби користувачів у зручності, швидкодії та інтерактивності.

1.4. Огляд та аналіз аналогічних програмних розробок

Розглянемо подібні продукти. Нижче наведено аналіз таких програмних розробок.

Аналіз вебзастосунку «Liveboard» показує, що цей продукт, розроблений компанією LiveBoard EdTech Solutions, являє собою вебзастосунок, що використовує JavaScript та PHP як мови реалізації. Однією з ключових особливостей є інтерактивна дошка, яка надає користувачам можливість малювати, додавати нотатки, вставляти різноманітні зображення, графіки та файли. Важливим аспектом функціональності є підтримка спільної роботи та редагування контенту в реальному часі кількома користувачами одночасно. Крім того, застосунок забезпечує можливість запису занять для подальшого їх перегляду. Для зручності користувачів передбачена інтеграція з популярними платформами для відеоконференцій, такими як Zoom, Microsoft Teams або Google Meet. Варто відзначити, що «Liveboard» є кросплатформним рішенням, оскільки він доступний як у вигляді мобільних додатків для iOS та Android, так і через веб-браузер. Додатково, система має функції для відстеження активності студентів або інших учасників.

Серед переваг вебзастосунку «Liveboard» можна виділити його інтуїтивно зрозумілий інтерфейс та кросплатформність, що забезпечує гнучкість у використанні на різних пристроях. Користувачі також оцінять можливість налаштування інструментів відповідно до їхніх індивідуальних потреб. Застосунок пропонує широкий набір інструментів для створення різноманітних візуалізацій, включаючи схеми, креслення та діаграми. Крім того, передбачена можливість експорту даних у різних популярних форматах, таких як PDF або PNG.

Однак, аналіз виявляє і певні недоліки. Зокрема, безкоштовна версія застосунку має певні обмеження у функціональності. Також, для повноцінної та стабільної роботи з «Liveboard» необхідне надійне інтернет-з’єднання.

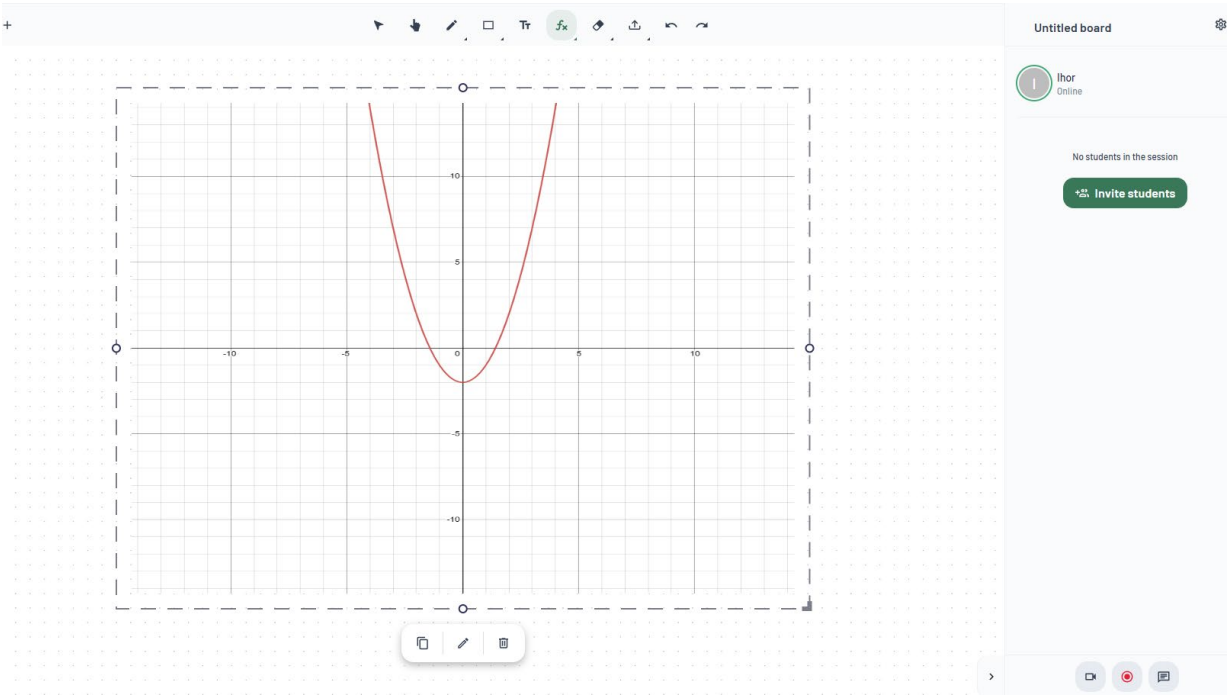

Рисунок 1.7 - Вебзастосунок «Liveboard»

Аналіз застосунку «Microsoft Whiteboard» показує, що цей продукт, розроблений Microsoft, використовує клієнт-серверну архітектуру та базується на мовах програмування JavaScript і C#. Основні можливості застосунку включають малювання, створення нотаток, а також розробку діаграм і схем. Важливою перевагою є глибока інтеграція з екосистемою Microsoft 365, що охоплює такі інструменти, як Teams, OneNote та Outlook, забезпечуючи зручну взаємодію в рамках звичного робочого середовища. Застосунок підтримує синхронізацію в реальному часі, що робить його ефективним інструментом для командної роботи. Усі дані автоматично зберігаються та синхронізуються в хмарі Microsoft Azure, гарантуючи їхню безпеку та доступність. Для оптимізації робочого процесу користувачам доступні різноманітні шаблони, які полегшують виконання стандартних завдань, таких як брейнштормінг або проведення ретроспектив. Крім того, «Microsoft Whiteboard» розпізнає рукописний текст і жести, розширюючи способи взаємодії з дошкою.

Серед ключових переваг застосунку «Microsoft Whiteboard» слід відзначити його безперешкодну інтеграцію в середовище Microsoft 365, що спрощує його використання для користувачів цієї екосистеми. Застосунок є

кросплатформним, що дозволяє працювати з ним на різних пристроях. Підтримка стилусів забезпечує зручне рукописне введення, що є важливим для багатьох користувачів. Хмарне зберігання даних гарантує постійний доступ до створених дошок з будь-якого підключеного пристрою.

Однак, аналіз виявляє і певні недоліки. Функціональність «Microsoft Whiteboard» може здаватися обмеженою порівняно з деякими конкурентними продуктами, такими як «Explain Everything» або «Mural», які пропонують ширший набір інструментів. Застосунку також бракує розширених інструментів, які могли б бути корисними для організації великих і складних проектів. Крім того, деякі функції стають доступними лише за наявності платної підписки на Microsoft 365.

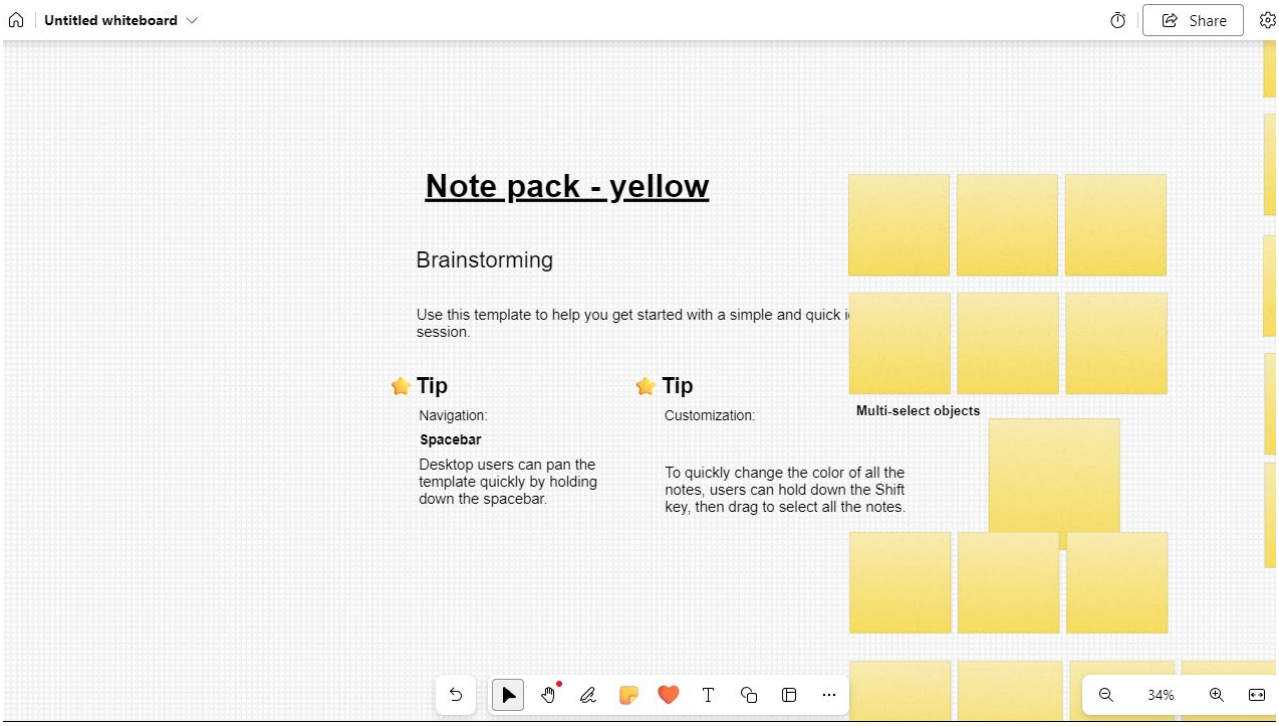

Рисунок 1.8 – Застосунок «Microsoft Whiteboard»

РОЗДІЛ 2
РОЗРОБКА ВЕБЗАСТОСУНКУ ЕЛЕКТРОННОЇ ДОШКИ ДЛЯ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1. Постановка задачі, призначення та вимоги до вебзастосунку «Classboard»

Завданням кваліфікаційної роботи є розробка вебзастосунку електронної дошки для дистанційного навчання «Classboard», використовуючи сучасні вебтехнології. Основна мета полягає у створенні інструменту, який забезпечить зручність та ефективність у проведенні онлайн-занять, інтерактивних лекцій і спільній роботі учнів та викладачів.

Для забезпечення зручності користування застосунок має мати простий та інтуїтивно зрозумілий інтерфейс. У вебзастосунку «Classboard» необхідно реалізувати такі функції:

- аутентифікація та авторизація користувачів;
- створення інтерактивних навчальних дошок;
- можливість роботи кількох користувачів одночасно та синхронізація дій користувачів на дошці у реальному часі;
- підтримка текстових, графічних та мультимедійних матеріалів;
- наявність інструментів для редагування, видалення та форматування елементів;
- збереження та завантаження матеріалів.

2.2. Вибір моделі розробки програмного засобу «Classboard»

При розробці програмного забезпечення важливим є вибір відповідної моделі розробки, яка відповідатиме потребам проекту, враховуючи обмеження і можливості команди, часові рамки та вимоги замовника.

Інкрементна модель є популярним підходом, у якому програмне забезпечення розробляється з лінійною послідовністю стадій, але в кілька

інкрементів (версій). Таким чином поліпшення продукту проходить заплановано весь час, поки життєвий цикл розробки ПЗ не завершиться (рис. 2.1).

Вимоги до системи визначаються на самому початку роботи, після чого процес розробки проводиться у вигляді послідовності версій, кожна з яких є закінченим і працездатним продуктом. [16]

Для проекту було обрано інкрементну модель розробки через її значні переваги. По-перше, на кожному етапі створення продукту команда отримує функціональну версію, готову до тестування і використання, що дає можливість виявляти і усувати помилки на ранніх етапах, мінімізуючи ризики серйозних проблем у фіналі. Ця модель дозволяє адаптувати розробку до змін: вимоги можуть коригуватися на основі результатів реалізації окремих частин системи. Раннє надання функціональних модулів продукту також є значним плюсом, оскільки це дає змогу користувачам раніше почати їх використання, що може позитивно вплинути на бізнес-процеси. Покроковий підхід до реалізації сприяє зменшенню ризиків завдяки можливості аналізувати кожний завершений етап роботи і вчасно вирішувати можливі проблеми.

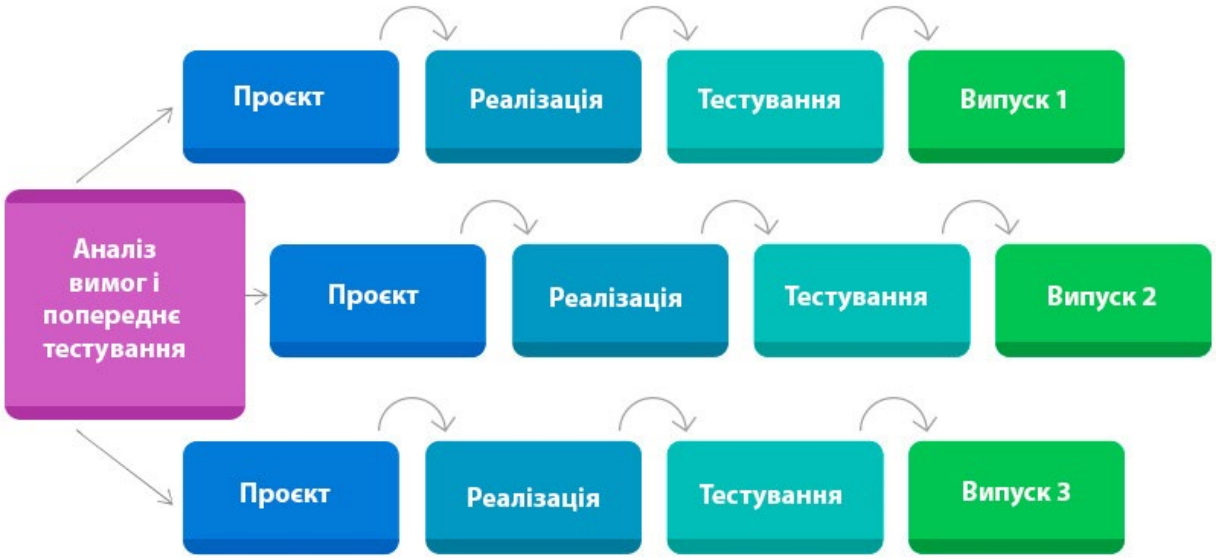

Рисунок 2.1 – Інкрементна модель розробки

2.3. Загальний опис проєкту «Classboard»

Архітектура вебзастосунку будується за принципом клієнт-серверної моделі з використанням вебсокетів для забезпечення інтерактивності в реальному часі та HTTP для обміну статичними даними й іншими запитами. Дані зберігаються в базі даних, що слугує центральним сховищем інформації та забезпечує ефективне управління даними, їхню цілісність і швидкий доступ (рис. 2.4).

Розглянемо інтерфейс вебзастосунку. Вебзастосунок містить такі сторінки:

- сторінка аутентифікації користувача;
- сторінка налаштування користувача;
- сторінка керування дошками: інтерфейс для перегляду, редагування та видалення всіх створених користувачем дошок;
- інтерактивна дошка.

Основними складовими інтерфейсу інтерактивної дошки є (рис. 2.2):

- панель інструментів для малювання, додавання тексту та інших елементів;
- бічна панель для налаштування стилів елементів;
- робоча область для створення та редагування контенту;
- меню користувача з можливістю експорту, зміни фона полотна (робочої області) та переходу на сторінку керування дошками;
- модальне вікно налаштування доступу для дошки.

Інтерактивна дошка призначена для створення графічних елементів, малювання, написання тексту та інших дій в реальному часі. Всі зміни, які виконуються на дошці, синхронізуються між підключеними користувачами.

Панель інструментів для малювання використовується для вибору основних інструментів таких як пензель, створення фігур, тексту тощо.

Бічна панель управління стилями об’єктів призначена для налаштування параметрів об'єктів, таких як колір межі, колір фону, ширина ліній і також

управління об’єктами. Користувач може застосовувати ці стилі до окремих об'єктів на дошці для створення бажаного дизайну.

Модальне вікно налаштування доступу призначене для керування правами доступу до інтерактивної дошки. У ньому власник дошки може скопіювати посилання на дошку, переглянути список осіб, які мають доступ, та змінити їхні ролі (наприклад, лише перегляд або редагування). Крім того, доступна функція запрошення нових учасників за електронною поштою.

Рисунок 2.2 - Макет структури інтерактивної дошки

Сервер побудований на платформі Node.js із використанням фреймворку NestJS для обробки HTTP-запитів Для реалізації WebSocket-з’єднань використовується вбудований модуль «@nestjs/websockets», що забезпечує інтеграцію з бібліотекою Socket.IO.

Для ініціалізація по протоколу WebSocket клієнт відправляє HTTP-запит із заголовком Upgrade: websocket, щоб перейти з HTTP-протоколу на WebSocket. Сервер в свою чергу відповідає кодом статусу 101 Switching Protocols і встановлює WebSocket-з’єднання. Цей процес називається «handshake» (рис. 2.3).

Після встановлення з’єднання всі повідомлення між клієнтом і сервером передаються через WebSocket-протокол. HTTP більше не використовується для передачі даних у рамках цього з'єднання.

Але HTTP-запити можуть продовжувати використовуватися для виконання REST-запитів, які не потребують постійного з’єднання (наприклад, авторизація, завантаження файлів, отримання попередніх сесій). [17]

```
// Відправляєм запит до сервера за посиланням example.com/connect-to-ws
// Запит клієнта до сервера:
GET /connect-to-ws HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13

// Відповідь сервера:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Рисунок 2.3 - Процес «handshake»

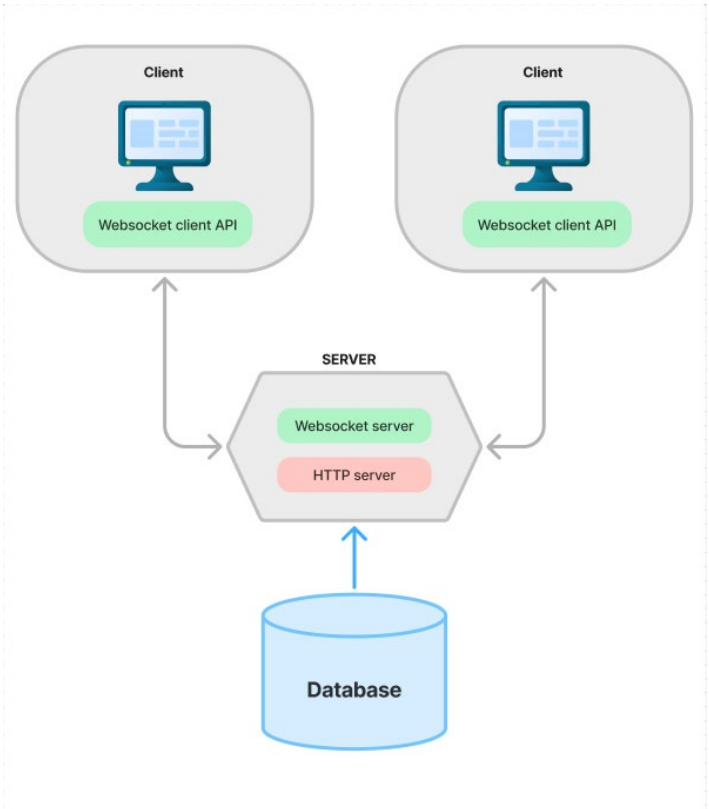

Рисунок 2.4 - Схема взаємодії клієнтської і серверної частин

2.4. Обґрунтування вибору інструментальних засобів розробки вебзастосунку «Classboard»

В якості програмного середовища розробки обрано Visual Studio Code. VS Code – це безкоштовний текстовий редактор, який має багато корисних функцій для розробки вебзастосунків. Він має вбудовану підтримку для JavaScript, TypeScript та інших мов програмування. [18]

Мовою програмування розробки обрано TypeScript, яка є розширенням JavaScript. Вона дозволяє створювати більш безпечний та структурований код. А саме дозволяє використовувати типи даних, що допомагає зменшити кількість помилок в коді.

Сама мова JavaScript має велику кількість бібліотек та фреймворків, які дозволяють розробникам створювати вебзастосунки швидко та ефективно. Наприклад, React – це одна з найпопулярніших бібліотек для розробки клієнтської частини вебзастосунків на JavaScript. Вона дозволяє розробникам створювати складні вебзастосунки з високою швидкістю та ефективністю.

Для полегшення та пришвидшення розробки проекту обрали фреймворк Next.js, який базується на React та надає деякі додаткові функції та переваги.

Наведемо деякі з переваг Next.js над React, які нам допоможуть у розробці вебзастосунку електронної дошки.

SSR (server side rendering). Серверний рендеринг (SSR) у Next.js передбачає, що вебсервер генерує повний вміст HTML вебсторінки перед тим, як надсилати його в браузер користувача. Це дозволяє доставити користувачу повноцінну HTML-сторінку без необхідності виконувати JavaScript на клієнті. [19]

SSG (static site generation). Генерація статичного сайту (SSG) передбачає рендеринг сторінок під час компіляції, тобто під час розгортання вебзастосунку на хостинг. Тобто HTML і вміст для кожної вебсторінки попередньо генеруються, тому вони готові та статичні (незмінні). Коли користувач відвідує

сторінку сайту, попередньо згенерований статичний HTML негайно надсилається в браузер без додаткової обробки під час запиту користувача. [19]

Маршрутизація: Next.js надає вбудовану систему маршрутизації, яка автоматично створює маршрути на основі вашої файлової структури. Вам не потрібно встановлювати або налаштовувати будь–які зовнішні бібліотеки, такі як React Router, щоб керувати навігацією між сторінками. [20]

Для розробки серверної частини використано платформу Node.js, фреймворк NestJS, базу даних PostgreSQL, та бібліотеку Socket.IO.

Node.js є середовищем виконання JavaScript на сервері, яке забезпечує високу продуктивність при обробці асинхронних запитів, що важливо для масштабованих систем з високою кількістю одночасних з'єднань. [21]

NestJS – це сучасний фреймворк для створення ефективних і масштабованих серверних додатків на Node.js. Він поєднує в собі принципи об’єктно-орієнтованого та функціонального програмування, надаючи розробникам інструменти для створення високопродуктивних додатків. [22]

Основні переваги NestJS:

- модульна архітектура: забезпечує зручну організацію коду та його розширюваність;

- підтримка TypeScript: підвищує безпеку коду та полегшує його підтримку;

- вбудована підтримка Dependency Injection (DI): спрощує керування залежностями;

- гнучкість: підтримка різних підходів до роботи з базами даних (ORM, SQL, NoSQL) та взаємодії з API (REST, GraphQL, WebSockets).

NestJS дозволяє легко реалізувати багаторівневу архітектуру серверної частини, що є ключовим аспектом у розробці складних вебзастосунків.

PostgreSQL – це об'єктно-реляційна база даних з відкритим кодом, яка має міцну репутацію завдяки своїй надійності, гнучкості та підтримці відкритих технічних стандартів. На відміну від інших RDMBS (систем керування реляційними базами даних), PostgreSQL підтримує як нереляційні, так і

реляційні типи даних. Це робить її однією з найбільш сумісних, стабільних і зрілих реляційних баз даних, доступних сьогодні. [23]

Основні переваги PostgreSQL:

- продуктивність і масштабованість: PostgreSQL підтримує низку оптимізацій продуктивності, таких як геопросторова підтримка та необмежений паралелізм;
- підтримка паралелізму;
- глибока підтримка мов;

Prisma ORM – це інструмент наступного покоління, призначений для спрощення взаємодії з базою даних для програм Node.js і TypeScript. Він пропонує сучасний підхід до керування базами даних, що робить його привабливим вибором для розробників, яким потрібна ефективність і надійність. [24]

Основні можливості Prisma:

- проста схема моделювання: Prisma використовує декларативний підхід до опису структури бази даних;
- типізація: клієнт Prisma генерує повністю типізовані запити, виявляючи помилки на етапі компіляції, а не під час виконання;
- підтримка міграцій: спрощує управління змінами у структурі бази даних;
- сумісність із TypeScript: забезпечує типізацію на рівні бази даних.

Бібліотека Socket.IO дозволяє створювати двостороннє з'єднання в реальному часі між клієнтом і сервером через WebSocket або інші протоколи (з автоматичним переходом на найкращий доступний протокол). [25]

Також при розробці були використані:

1. Tailwind – це CSS-фреймворк, який дозволяє швидко та легко створювати користувацькі дизайни для вебзастосунків. Він має багато вбудованих класів, які дозволяють швидко створювати різні елементи інтерфейсу. [26]

2. perfect-freehand – це бібліотека створена для створення плавних ліній, які імітують природний рух намальованих від руки ліній. Це дозволяє розробникам брати набір точок (зазвичай із введених користувачем, наприклад рухів миші чи стилуса) і генерувати плавний контур, що представляє штрих від руки. [27]

3. recoil – це бібліотека керування станом, розроблена для додатків React, яка пропонує більш гнучкий і масштабований підхід, ніж вбудовані React useState або useContext. Він вводить концепцію «atoms», які є одиницями стану, які можуть бути спільними між компонентами, і «selectors», які дозволяють обчислювати похідні стани. [28]

2.5. Особливості програмної реалізації «Classboard»

2.5.1. Розробка серверної частини

Для серверної частини вебзастосунку використали фреймворк NestJs, який базується на платформі NodeJs.

Щоб ініціювати та запустити застосунок NestJS, використовуємо функцію bootstrap, яка створює екземпляр програми на основі AppModule. Налаштовуємо конфігурацію механізму CORS для міждоменних запитів із підтримкою обміну cookie та запускаємо HTTP-сервер для очікування вхідних з'єднань на визначеному порту.

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.setGlobalPrefix('api')
  app.use(cookieParser())
  app.enableCors({
    origin: process.env.ORIGIN,
    credentials: true,
    exposedHeaders: 'set-cookie'
  })
  await app.listen(process.env.PORT || 5200);
}
bootstrap();
```

Механізм аутентифікації в застосунку базується на JWT-токенах (JSON Web Tokens). Основу складає модуль AuthModule. Він об'єднує сервіси, контролери, стратегії та конфігурації, які пов’язані з аутентифікацією.

```
@Module({
  imports: [UserModule, ConfigModule, JwtModule.registerAsync({
    imports: [ConfigModule],
    inject: [ConfigService],
    useFactory: getJwtConfig
  })],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
})
export class AuthModule {}
```

Клас JwtStrategy є частиною механізму бібліотеки PassportJS, що реалізує стратегію авторизації через JWT. Стратегія зчитує токен з заголовка Authorization HTTP-запиту. Після декодування вона викликає метод validate(), який повертає дані користувача, отримані через UserService.

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    private configService: ConfigService,
    private userService: UserService
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: configService.get('JWT_SECRET'),
    })
  }

  async validate({ id }: { id: string }) {
    return this.userService.getByIdWithoutRelations(id)
  }
}
```

Спершу, щоб використати цей захист на практиці, ми створюємо клас JwtAuthGuard, який наслідує стандартний AuthGuard, який надає бібліотека PassportJS, із використанням стратегії «jwt». Цей клас відповідає за перевірку наявності та дійсності accessToken, переданого в заголовку HTTP-запиту.

Після цього, щоб спростити застосування цього механізму до окремих маршрутів, ми створюємо декоратор @Auth(). Він є зручним синтаксичним скороченням (синтаксичним цукром), що фактично застосовує UseGuards(JwtAuthGuard). Такий підхід дозволяє легко та наочно захищати будь-який метод контролера, просто додавши до нього декоратор @Auth().

```
export class JwtAuthGuard extends AuthGuard('jwt') {}
export const Auth = () => UseGuards(JwtAuthGuard)
```

Спочатку, щоб обробляти WebSocket-підключення, створюємо клас BoardGateway з декоратором @WebSocketGateway(). Він відповідає за зв’язок між клієнтом і сервером. Для аутентифікації використовуємо JWT-токени: у методі afterInit, що викликається після ініціалізації WebSocket-сервера, додаємо middleware AuthWsMiddleware. Він перевіряє токен кожного клієнта перед підключенням до WebSocket-сервера, що забезпечує безпечну роботу з'єднання.

```
@WebSocketGateway()
export class BoardGateway {
  constructor(private readonly boardService
  @WebSocketServer() io: Server<ClientToSer

  afterInit(server: Server) {
    server.use(
      AuthWsMiddleware(
        this.jwtService,
        this.configService,
        this.userService
      ) as any
    )
  }
```

Дошка підтримує два режими доступу: приватний – коли доступ мають лише запрошені користувачі, і публічний – коли доступ надається всім, хто має посилання. Крім цього, як дошка, так і користувачі мають певні ролі, які визначають рівень доступу. Виділяють чотири рівні доступу: ADMIN – адміністратор дошки, який має повний контроль над нею; EDIT – надає можливість редагування вмісту дошки; VIEW – дозволяє лише перегляд дошки без можливості змін; NO_ACCESS – означає повну відсутність доступу.

Для реалізації цього підходу в застосунку використовується механізм авторизації на основі ролей – RBAC (Role-Based Access Control). На відміну від аутентифікації через JWT-токени, RBAC дає змогу точно визначати права доступу користувачів відповідно до їхньої ролі. Зокрема, у застосунку цей підхід застосовується для захисту WebSocket-обробників, обмежуючи або дозволяючи доступ до певних дій залежно від ролі користувача.

Щоб уникнути постійних звернень до бази даних під час роботи з WebSocket, створено кеш у вигляді змінної rooms, у якій дублюється конфігурація доступу з бази. Це структура типу Map, де ключем є ID кімнати (дошки), а значенням – об'єкт типу Room. Об’єкт Room створюється та зберігається в цій структурі лише тоді, коли хоча б один користувач перебуває на дошці з відповідним ідентифікатором.

```
private rooms = new Map<string, Room>()
export type Room = {
  access: {
    isPublic: boolean
    accessLevel: AccessLevel
  },
  users: Map<string, RoomUser>;
};
export type RoomUser = {
  username: string;
  email: string;
  userId: string;
  accessLevel: AccessLevel;
}
```

Для налаштування доступу до дошки та керування правами користувачів необхідно реалізувати відповідні механізми. У застосунку для цього передбачено три основні функції: setBoardAccess – для зміни конфігурації доступу самої дошки, setUserAccess – для керування доступом окремих користувачів, а також deleteUserAccess – для скасування доступу користувача. Усі ці функції зберігають інформацію про доступ у базі даних за допомогою Prisma ORM. Аналогічно реалізовано функції setRoomAccess та setUserAccessInRoom. На відміну від попередніх, вони зберігають конфігурацію доступу у змінну rooms, щоб уникнути частих звернень до бази даних.

```
async setBoardAccess(
  boardId: string,
  userId: string,
  isPublic: boolean,
  accessLevel: AccessLevel
) {
  // Отримуємо дошку за ідентифікатором
  const board = await this.prisma.board.findUnique({
    where: { id: boardId, userId }
  })
  if (!board) throw new NotFoundException('Board not found')

  // Оновлення конфігурації доступу дошки
  await this.prisma.board.update({
    where: { id: boardId },
    data: {isPublic, accessLevel}
  });
}
```

```
setRoomAccess(
  roomId: string,
  isPublic: boolean,
  accessLevel: AccessLevel
  ) {
  const room = this.rooms.get(roomId)
  if (!room) return

  room.access.isPublic = isPublic
  room.access.accessLevel = accessLevel
}
```

Для перевірки прав доступу використовуються дві функції: hasRequiredAccess та hasRequiredAccessFromRoom, які виконують одну й ту саму задачу. Різниця полягає в джерелі даних: hasRequiredAccess отримує інформацію безпосередньо з бази даних, тоді як hasRequiredAccessFromRoom – із змінної rooms, де тимчасово зберігається конфігурація доступу до кімнат. Ці функції викликаються у WebSocket-обробниках перед виконанням логіки, пов'язаної з певною дошкою, що дозволяє заздалегідь перевірити права доступу користувача.

```
async hasRequiredAccess(
  boardId: string,
  userId: string,
  allowedRoles: AccessLevel[] = []
): Promise<boolean> {
  // Отримуємо конфігурацію доступу дошки
  const { isPublic, boardAccessLevel } =
    await this.getBoardAccess(boardId)
  // Отримуємо рівень доступу користувача до цієї дошки
  const { userAccessLevel } =
    await this.getUserAccess(boardId, userId)

  // Перевіряємо, чи користувач є адміністратором дошки
  const userIdAdmin = userAccessLevel === AccessLevel.ADMIN
  // Перевіряємо, чи дошка має один із дозволених рівнів доступу
  const boardHasRequiredLevel = isPublic &&
   allowedRoles.includes(boardAccessLevel)
  // Перевіряємо, чи користувач має один із дозволених рівнів доступу
  const userHasRequiredLevel = allowedRoles.includes(userAccessLevel)

  const hasRequiredAccess =
    userIdAdmin || userHasRequiredLevel || boardHasRequiredLevel

  return hasRequiredAccess
```

Функція hasRequiredAccess працює наступним чином. Вона приймає три параметри: boardId – ідентифікатор дошки, userId – ідентифікатор користувача, та масив allowedRoles, у якому задаються допустимі рівні доступу. Якщо allowedRoles не передано, функція перевіряє лише, чи є користувач адміністратором дошки.

У ході виконання функція по черзі звертається до бази даних, спочатку для отримання загальної конфігурації доступу до дошки за допомогою методу getBoardAccess, а потім – для отримання індивідуального рівня доступу користувача через метод getUserAccess.

У результаті доступ надається, якщо виконується хоча б одна з таких умов:

- користувач є адміністратором;
- користувач має відповідну роль;
- дошка є відкритою та її доступ відповідає дозволеним рівням.

Таким чином, алгоритм дозволяє гнучко регулювати доступ дошки, з урахуванням ролей користувачів.

Для повноцінного керування правами доступу до дошки в реальному часі у застосунку передбачено окремі обробники WebSocket-подій. Вони дозволяють змінювати рівні доступу як до самої дошки, так і для окремих користувачів. Основними є: «set_room_access» – зміна конфігурації доступу дошки, «set_user_access» – оновлення прав доступу конкретного користувача, та «remove_user_access» – скасування його доступу. Ці події дають змогу адміністратору оперативно керувати доступом без перезавантаження сторінки.

Розглянемо обробник події «set_room_access». Перш за все перевіряється, чи має користувач роль адміністратора, використовуючи hasRequiredAccessFromRoom. Без відповідних прав зміна параметрів доступу заборонена. Після успішної перевірки змінюється конфігурація доступу: метод setBoardAccess оновлює дані в базі даних, а setRoomAccess – у кеші, щоб система працювала з актуальними налаштуваннями.

У випадку, якщо дошка була зроблена приватною, необхідно переконатися, що всі підключені до неї користувачі мають право залишатися. Тому за допомогою fetchSockets обробник отримує список усіх сокетів у кімнаті. Кожен із них перевіряється на відповідність новим умовам доступу. Якщо у користувача явно відсутній доступ, тобто його рівень доступу встановлено як NO_ACCESS, він автоматично виключається з кімнати за допомогою leaveRoom та socket.leave, отримує повідомлення «user_access_denied», а решта учасників –

повідомлення «user_disconnected» про його відключення. І на завершення, коли всі дії виконано, дошка надсилає всім учасникам повідомлення «room_access_updated», яке інформує про зміну налаштувань доступу.

```
@SubscribeMessage('set_room_access')
@UsePipes(new ValidationPipe({ whitelist: true }))
async handleSetBoardAccess(
  @ConnectedSocket() socket: AuthenticatedSocket,
  @MessageBody()
  { roomId, isPublic, accessLevel }: SetRoomAccessDto
) {
  const isClientConnectedToRoom = this.boardService.checkRoom(roomId)
  if (!isClientConnectedToRoom) return

  // Перевіряємо, чи користувач є адміністратором дошки
  const hasAdminRole = this.boardService.hasRequiredAccessFromRoom(
    roomId,
    socket.id
  )
  if (!hasAdminRole) return

  // Оновлюємо конфігурацію доступу дошки
  await this.boardService.setBoardAccess(
    roomId, socket.user.id, isPublic, accessLevel
  )
  this.boardService.setRoomAccess(roomId, isPublic, accessLevel)

  if (!isPublic) {
    const roomSockets = await this.io.in(roomId).fetchSockets()
    roomSockets.forEach(roomSocket => {
      const socketAccessLevel = this.boardService.getUserAccessFromRoom(
        roomId,
        roomSocket.id
      )
      // Перевіряємо, чи це сокет потрібного користувача
      if (socketAccessLevel === AccessLevel.NO_ACCESS) {
        this.boardService.leaveRoom(roomId, roomSocket.id)
        roomSocket.leave(roomId)

        roomSocket.emit('user_access_denied')
        this.io.to(roomId).emit('user_disconnected', roomSocket.id)
      }
    })
  }

  this.io.to(roomId).emit('room_access_updated')
}
```

2.5.2. Розробка клієнтської частини

Для створення двостороннього з'єднання в реальному часі між клієнтом і сервером через WebSocket потрібно ініціалізувати змінну socket, підключившись до головного сервера. Додатково додаємо HTTP-заголовок Authorization, щоб сервер міг аутентифікувати клієнта через JWT-токен.

```
const accessToken = getAccessToken()

export const socket: Socket<
  ServerToClientEvents, ClientToServerEvents
> = io(
  process.env.NEXT_PUBLIC_SERVER_URL,
  {
    extraHeaders: {
      Authorization: `Bearer ${accessToken}`
    }
  }
)
```

Тепер перейдемо до розгляду реалізації сторінки кімнати, тобто інтерактивної дошки разом з іншими допоміжними компонентами.

Перед початком роботи з дошкою необхідно отримати від сервера дані кімнати: користувачів, елементи полотна та ідентифікатори самих елементів. Спочатку компонент <canvas> повинен повністю завантажиться і відправити подію «joined_room», на яку в свою чергу сервер відреагує і відправить нам дані кімнати.

```
<canvas
  ref={canvasRef}
  onPointerDown={e => {…
  }}
  onPointerMove={e => {…
  }}
  onPointerUp={e => {…
  }}
```

```
useEffect(() => {
  if (ctx) socket.emit('joined_room')
}, [ctx])
```

Для отримання цих даних використовується спостерігач «room». Оскільки дані в форматі JSON, їх потрібно розпарсити, обробити та зберегти в глобальне сховище room, створене за допомогою бібліотеки recoil.

```
useEffect(() => {
  socket.on('room', (usersMovesIds, usersMoves, usersToParse) => {
    const usersParsed = new Map<string, RoomUser>(
      JSON.parse(usersToParse)
    )

    /*Додаємо до кожного користувача колір
    та зберігаємо до локальної змінної newUsers*/
    const newUsers = new Map<string, RoomUser>()
    usersParsed.forEach((user, id) => {
      const color = getColorByUsername(user.username)
      newUsers.set(id, {
        username: user.username,
        email: user.email,
        color,
        userId: user.userId,
        accessLevel: user.accessLevel
      })
    })

    /*Перетворюємо масив елементів дошки на нову структуру
    даних Map та зберігаємо до локальної змінної newMoves*/
    const newMoves = new Map<string, Move>()
    for (const move of usersMoves) {
      newMoves.set(move.id, move)
    }

    //Призначаємо нашому сховищу Room наші локальні змінні
    setRoom(prev => ({
      ...prev,
      movesIds: usersMovesIds,
      moves: newMoves,
      users: newUsers
    }))
  })
```

Для реалізації інтерактивності дошки, присвоюємо атрибутам onPointerDown, onPointerMove, onPointerUp компонента <canvas> відповідні функції, які будуть обробляти події миші. Розглянемо хук useDraw, який предоставляє ці функції. Функції onPointerDown, onPointerMove, onPointerUp викликаються при натисканні, переміщенні та відпусканні. Залежно від активного інструменту виконується одна з 4 дій: вибір та виділення елементів, переміщення елементів, введення тексту або малювання.

```
const onPointerDown = (x: number, y: number) => {
  if (options.mode === 'select') {
    // Режим вибору елементів
    handleStartSelect(x, y)
  } else if (options.mode === 'draw' && options.shape === 'text') {
    // Режим введення тексту
    tempMoves[0] = [x, y]
    setOptions(prev => ({
      ...prev,
      mode: 'writing'
    }))
  } else if (options.mode === 'draw') {
    // Режим малювання
    handleStartDrawing(x, y)
  }
}
```

```
const onPointerMove = (x: number, y: number, shift?: boolean) =>
  if (options.mode === 'translating') {
    // Режим переміщення елементів
    handleTranslate(x, y)
  } else if (options.mode === 'selectionNet') {
    // Режим виділення елементів
    handleSelectionNet(x, y)
  } else if (options.mode === 'draw' && options.shape !== 'text')
    // Режим малювання
    handleDraw(x, y, shift)
  }
}
```

```
const onPointerUp = () => {
  if (options.mode === 'translating') {
    // Режим переміщення елементів
    handleEndTranslate()
  } else if (options.mode === 'selectionNet') {
    // Режим виділення елементів
    clearOnYourMove()
    setOptions(prev => ({
      ...prev,
      mode: 'select'
    }))
  } else if (options.mode === 'draw') {
    // Режим малювання
    handleEndDraw()
  } else if (options.mode === 'writing') {
    // Режим введення тексту
    return
```

Розглянемо реалізацію малювання на полотні. Для малювання різних фігур і ліній використовуємо методи, які надає Canvas 2D API. На їх основі створюємо функції для малювання: drawRect – для прямокутників, drawEllipse – для еліпсів, drawLine – для ліній.

```
export const drawRect = (ctx: CanvasRenderingContext2D, from: [n
) => {
  ctx.beginPath() // Починаємо новий шлях для малювання

  // Отримуємо ширину та висоту на основі координат
  const { width, height } = getWidthAndHeight(x, y, from, shift)

  //Малюємо прямокутник
  if (fill) ctx.fillRect(from[0], from[1], width, height)
  else ctx.rect(from[0], from[1], width, height)

  ctx.stroke() // Наносимо контур прямокутника
  ctx.fill() // Заповнюємо прямокутник кольором
  ctx.closePath() // Завершуємо шлях

  return { width, height }
}
```

```
export const drawLine = (ctx: CanvasRenderingContext2D,
  points: number[][]
) => {
  if (!ctx || points == null || points.length <= 0) return
  // Отримуємо дані обведення (stroke)
  // для лінії на основі заданих точок
  const stroke = getStroke(points, {
    size: ctx.lineWidth,
    thinning: 0.5,
    smoothing: 0.5,
    streamline: 0.5
  })
  // Генеруємо SVG-подібний шлях із даних обведення
  const pathData = getSvgPathFromStroke(stroke)
  // Створюємо об'єкт шляху Path2D для використання в canvas
  const myPath = new Path2D(pathData)
  // Малюємо отриману лінію на полотні
  ctx.fill(myPath)
}
```

```
export const drawEllipse = (
  ctx: CanvasRenderingContext2D,
  from: [number, number],
  x: number,
  y: number,
  shift?: boolean
) => {
  ctx.beginPath() // Починаємо новий шлях для малювання
  // Отримуємо ширину та висоту на основі координат
  const { width, height } = getWidthAndHeight(x, y, from, shift)
  const cX = from[0] + width / 2 // центр еліпса по осі X
  const cY = from[1] + height / 2 // центр еліпса по осі Y
  const radiusX = Math.abs(width / 2)
  const radiusY = Math.abs(height / 2)

  // Малюємо еліпс із заданими параметрами
  ctx.ellipse(cX, cY, radiusX, radiusY, 0, 0, 2 * Math.PI)
  ctx.stroke() // Наносимо контур еліпса
  ctx.fill() // Заповнюємо еліпс кольором
  ctx.closePath() // Завершуємо шлях
  return { cX, cY, radiusX, radiusY }
```

Використовуючи функції малювання, реалізовуємо основний метод handleDraw, який буде викликатися при переміщенні курсора на полотні. Фігури та лінії будуть відображатися лише локально під час малювання.

Щоб намальований елемент відображався на полотні як у себе, так і в інших користувачів, спочатку необхідно створити об’єкт елемента move. В цьому об’єкті додаємо координати, та додаткові опції такі як: колір фону, колір межі, ширина лінії або межі. І потім методом emit надсилаємо на сервер подію «draw» та об’єкт move в якості даних. Цю логіку розміщуємо у функцію handleEndDraw, яка викликається коли відпускаємо кнопку миші.

Коли сервер обробив нашу подію «draw», він надішле автору доданого елементу подію «your_move», та решту учасникам - «user_draw». Спостерігач «your_move» зберігає отриманий від сервера елемент до map структури moves, яка знаходиться в глобальному сховищі room.

```
socket.on('your_move', move ⇒ {
  // Відновлюємо стан полотна та очищуємо тимчасове полотно
  clearOnYourMove()
  // Додаємо новий елемент до сховища
  handleAddMove(move)

  const isAfterUndo = history.movesToUndo.find(
    moveToUndo ⇒
      moveToUndo.mode === 'draw' && move.id === moveToUndo.move.id
  )
  // Якщо новий елемент не є результатом скасування дії видалення
  if (!isAfterUndo) {
    // Зберігаємо доданий елемент в історію скасування дій
    addMoveToUndo({ mode: 'draw', move })
  }
})
```

В свою чергу спостерігач «user_draw», який викликається лише в решта учасників сесії, виконує ту ж саму функцію, але при умові якщо користувач не виконує активних операцій з полотном. У разі активної взаємодії з полотном

новий елемент тимчасово зберігається у масиві movesToDrawLater, з якого потім дістаємо елементи та переносимо до сховища після завершення малювання.

```
const movesToDrawLater: Move[] | undefined = []
socket.on('user_draw', move => {
  // Якщо ми не малюємо на полотні,
  // то додаємо отриманий елемент до сховища
  if (!drawing) {
    handleAddMove(move)
  } else {
    movesToDrawLater.push(move)
  }
})
return () => {
  socket.off('user_draw')
  // Коли припинили малювати
  // додаємо отримані елементи до сховища
  if (movesToDrawLater.length) {
    for (const userMove of movesToDrawLater) {
      handleAddMove(userMove)
    }
  }
```

Щоразу, коли в сховище додається новий елемент, викликається хук useEffect. У ньому виконується функція drawAllMoves для відображення всіх елементів на дошці або drawMove, яка малює лише останній доданий елемент, забезпечуючи продуктивність і коректність рендерингу.

```
useEffect(() => {
  if (prevMovesLength >= sortedMoves.length || !prevMovesLength) {
    drawAllMoves()
  } else {
    const lastMove = sortedMoves[sortedMoves.length - 1]
    drawMove(lastMove)
  }

  return () => {
    prevMovesLength = sortedMoves.length
  }
```

2.6. Організація тестування та налагодження програмного засобу «Classboard»

Організація тестування та налагодження програмного засобу є важливим етапом в розробці програмного забезпечення. Для вебзастосунків, таких як електронна дошка, тестування може включати перевірку функціональності, продуктивності, безпеки, сумісності та інших аспектів.

Було проведене ручне тестування на продуктивність, функціональність і сумісність. Тест на продуктивність показав хороші результати, тобто інтерфейс управління дошкою працював без підвисань. Застосунок в тесті на сумісність пройшов успішно в таких браузерах: Chrome, Firefox, Microsoft Edge. В тесті на функціональність успішно перевірили інтерфейс роботи з електронною дошкою для дистанційного навчання на дотримання вимог функціональності.

Отже, вебзастосунок пройшов успішно 3 види тестувань.

2.7. Рекомендації по використанню та впровадженню програмного засобу «Classboard»

Вебзастосунок «Classboard» – це інструмент, який забезпечує ефективну співпрацю між учасниками онлайн-занять. Основною функцією цього програмного забезпечення є проведення інтерактивних лекцій, та можливість співпраці спільній роботі студентів та викладачів.

Головною перевагою вебзастосунку є можливість співпраці в режимі реального часу: на одній дошці можуть працювати кілька користувачів одночасно, а їхні дії синхронізуються за допомогою технології websocket. Це дозволяє викладачам і студентам ефективно взаємодіяти, навіть коли вони знаходяться в різних частинах світу.

Для впровадження Classboard у навчальний процес достатньо мати пристрій із доступом до мережі Інтернет, який підтримує сучасний веббраузер. Це можуть бути персональні комп’ютери, ноутбуки, планшети або смартфони.

Програмний засіб розроблений із використанням сучасних вебтехнологій, таких як мова програмування TypeScript та фреймворки Next.js, Nest.js, що забезпечують його високу продуктивність і масштабованість. Це спрощує подальше розширення функціональності, наприклад, додавання нових інструментів взаємодії, підтримки голосового чи відеозв’язку, інтеграції з хмарними сховищами, внутрішньою інфраструктурою навчального закладу, тощо.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досліджено теоретичні основи створення вебзастосунків, інтерактивних інтерфейсів і технологій для розробки багатокористувацьких систем у реальному часі. Результатом роботи став вебзастосунок «Classboard» для дистанційного навчання, розроблений із використанням сучасних вебтехнологій, таких як TypeScript, клієнтський фреймворк Next.js, серверний фреймворк Nest.js, бібліотека Socket.IO та інші.

Вебзастосунок реалізовано з урахуванням вимог до зручності використання та функціональності. Забезпечено інтуїтивно зрозумілий та зручний інтерфейс, який покращує ефективну взаємодію користувачів із системою. Реалізовано аутентифікацію та авторизацію користувачів у реальному часі, створення інтерактивних навчальних дошок, можливість одночасної роботи кількох користувачів із синхронізацією дій у реальному часі. Підтримуються текстові, графічні та мультимедійні матеріали, доступні інструменти для форматування та видалення елементів. Також забезпечено функції збереження та завантаження матеріалів. Розроблений вебзастосунок відповідає сучасним потребам у сфері дистанційного навчання та забезпечує ефективність і зручність у проведенні онлайн-занять і спільній роботі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How e-learning has evolved over the past few years - online education blog of touro university. *Online Education Blog of Touro University*. URL: https://blogs.onlineeducation.touro.edu/how-e-learning-has-evolved-over-the-past-few-years/ (дата звернення: 29.04.2025).
2. Software architecture patterns: what are the types and which is the best one for your project | turing. *Turn AGI Research into Real-World Impact | Turing*. URL: https://www.turing.com/blog/software-architecture-patterns-types (дата звернення: 29.04.2025).
3. Учасники проектів Вікімедіа. Вебзастосунок – вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Вебзастосунок (дата звернення: 29.04.2025).
4. Canvas API - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (дата звернення: 29.04.2025).
5. WebGL: 2D and 3D graphics for the web - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (дата звернення: 29.04.2025).
6. SVG: scalable vector graphics | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/SVG (дата звернення: 29.04.2025).
7. Учасники проектів Вікімедіа. Node.js – вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення: 29.04.2025).
8. Adedimeji I. L. Node.js architecture: understanding node.js architecture. *Medium*. URL: https://medium.com/@ibrahimlanre1890/node-js-architecture-understanding-node-js-architecture-5fb32879b994 (дата звернення: 29.04.2025).
9. Oli N. Exploring popular authentication methods for REST apis: a comprehensive guide. *Medium*. URL: https://medium.com/@namanoli/exploring-popular-

authentication-methods-for-rest-apis-a-comprehensive-guide-9c68002210e2 (дата звернення: 29.04.2025).

10. McCarthy M. Difference between RBAC vs. ABAC vs. ACL vs. PBAC vs. DAC. *StrongDM: Your Partner in Zero Trust Privileged Access*. URL: https://www.strongdm.com/blog/rbac-vs-abac (дата звернення: 29.04.2025).
11. An overview of HTTP - HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата звернення: 29.04.2025).
12. Тривале опитування. *Сучасний підручник з JavaScript*. URL: https://uk.javascript.info/long-polling (дата звернення: 29.04.2025).
13. Using server-sent events - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events (дата звернення: 29.04.2025).
14. The WebSocket API (WebSockets) - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 29.04.2025).
15. WebRTC. *WebRTC*. URL: https://webrtc.org/ (дата звернення: 29.04.2025).
16. Життєві цикли розробки ПЗ. *Онлайн-курси від компанії QATestLab | Головна сторінка*. URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/ (дата звернення: 29.04.2025).
17. Writing WebSocket servers - Web APIs | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers (дата звернення: 29.04.2025).
18. Microsoft. Visual studio code - code editing. redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (дата звернення: 29.04.2025).
19. Roland C. E. Static site generation (SSG) vs server-side rendering in next.js. *Medium*. URL: https://medium.com/@chrisebuberoland/static-site-

generation-ssg-vs-server-side-rendering-in-next-js-debf43f4bb7f (дата звернення: 29.04.2025).

20. Routing: defining routes | next.js. *Next.js by Vercel - The React Framework*. URL: https://nextjs.org/docs/app/building-your-application/routing/defining-routes (дата звернення: 29.04.2025).
21. Учасники проектів Вікімедіа. Node.js – вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення: 29.04.2025).
22. Nest js: основні характеристики та переваги фреймворку. *FoxmindEd*. URL: https://foxminded.ua/nest-js/ (дата звернення: 29.04.2025).
23. IBM. What Is PostgreSQL? | IBM. *IBM - United States*. URL: https://www.ibm.com/think/topics/postgresql (дата звернення: 29.04.2025).
24. Understanding prisma ORM. *TiDB*. URL: https://www.pingcap.com/article/understanding-prisma-orm/ (дата звернення: 29.04.2025).
25. Учасники проектів Вікімедіа. Socket.IO – вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Socket.IO (дата звернення: 29.04.2025).
26. Tailwind CSS - rapidly build modern websites without ever leaving your HTML. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. URL: https://tailwindcss.com/ (дата звернення: 29.04.2025).
27. Perfect-freehand. *npm*. URL: https://www.npmjs.com/package/perfect-freehand (дата звернення: 29.04.2025).
28. freeCodeCamp. How to use recoil for state management in your react projects. *freeCodeCamp.org*. URL: https://www.freecodecamp.org/news/how-to-use-recoil-for-state-management-in-your-react-projects/ (дата звернення: 29.04.2025).

ДОДАТКИ

Додаток А

ТЕХНІЧНЕ ЗАВДАННЯ

НА РОЗРОБКУ ЕЛЕКТРОННОЇ ДОШКИ ДЛЯ ДИСТАНЦІЙНОГО НАВЧАННЯ

ВСТУП

Програмний продукт «Classboard» – це вебзастосунок електронної дошки, призначений для інтерактивного дистанційного навчання. Його розробка спрямована на створення інтуїтивно зрозумілого інструменту для викладачів і студентів, що забезпечує зручність у спільному використанні навчальних матеріалів у режимі реального часу.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка проводиться на основі завдання, поставленого керівником бакалаврської роботи для спеціальності 122 “Комп’ютерні науки”. Документ затверджено Волинським національним університетом імені Лесі Українки

Найменування теми розробки – «Дослідження принципів проектування вебзастосунків на прикладі розробки електронної дошки для дистанційного навчання».

ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: програмний продукт «Classboard» призначений для забезпечення ефективної взаємодії між викладачем і студентами у дистанційному форматі. Основні функції включають візуалізацію навчальних матеріалів, спільну роботу з графічними елементами, керування правами доступу до інтерактивної дошки, а також збереження навчальних матеріалів для подальшого використання.

Експлуатаційне призначення: система може використовуватися в освітніх установах, корпоративних тренінгах та інших навчальних середовищах для

організації інтерактивних занять, підтримки зворотного зв’язку між викладачем та студентами.

ВИМОГИ ДО ПРОГРАМИ ЧИ ПРОГРАМНОГО ПРОДУКТУ

Вимоги до функціональних характеристик:

- аутентифікація та авторизація користувачів;
- створення інтерактивних навчальних дошок;
- можливість роботи кількох користувачів одночасно та синхронізація дій користувачів на дошці у реальному часі;
- підтримка текстових, графічних та мультимедійних матеріалів;
- наявність інструментів для редагування, видалення та форматування елементів;
- збереження та завантаження матеріалів.

Вимоги до надійності:

- забезпечення стійкої роботи навіть за умов підвищеного навантаження;
- механізми контролю вхідних даних для запобігання введенню некоректної інформаці;
- повинна бути реалізована система обмеження доступу для запобігання несанкціонованому доступу до конфіденційної інформації;
- всі паролі користувачів повинні зберігатися в зашифрованому вигляді з використанням надійних алгоритмів хешування, таких як bcrypt або argon2.

Умови експлуатації: наявність стабільного інтернет з’єднання та використання сучасного браузера для доступу до програмного забезпечення.

Вимоги до складу і параметрів технічних засобів: будь-який комп’ютер, смартфон або планшет із доступом до мережі інтернет та зі встановленим браузером.

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Етапи розробки:

1. Аналіз вимог та проєктування: формування технічного завдання та схеми роботи програмної розробки.

2. Розробка архітектури та базових компонентів: розробка структури бази даних, макетів інтерфейсу.

3. Реалізація основних функцій: реалізація серверної та клієнтської частини, бази даних, інтерфейсу користувача.

4. Тестування та налагодження: перевірка на помилки, навантаження та стабільність.

5. Розгортання: розміщення програмної розробки на сервері та запуск у роботу.

ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

По закінченню розробки, необхідно провести наступні види випробувань:

- функціональне тестування (перевірка виконання всіх заявлених функцій);
- тестування витривалості системи (робота при великій кількості запитів).

Загальні вимоги до приймання роботи:

- повне виконання функціональних вимог;
- надання програмної документації;
- успішне проходження тестувань.

Додаток Б

КЕРІВНИЦТВО КОРИСТУВАЧУ ПРОГРАМНОГО ЗАСОБУ «Classboard»

1. Загальні відомості

Вебзастосунок «Classboard» – це інструмент для дистанційного навчання, який забезпечує можливість проведення інтерактивних занять із використанням сучасних вебтехнологій.

2. Функціональне призначення

Застосунок призначений для організації спільної роботи викладачів та учнів на інтерактивній дошці. Користувачі можуть малювати, писати текст, керувати графічними елементами, а також працювати у реальному часі із синхронізацією дій. Основні функції включають візуалізацію навчальних матеріалів, спільну роботу з графічними елементами, керування правами доступу до інтерактивної дошки, а також збереження навчальних матеріалів для подальшого використання.

3. Умови застосування програми

Програмний засіб використовується на будь-якому комп’ютері, смартфоні або планшеті із доступом до мережі інтернет та зі встановленим сучасним браузером з підтримкою Javascript.

4. Опис роботи програми

Для початку роботи користувачу необхідно пройти аутентифікацію. Для цього застосунок передбачає дві окремі сторінки – логіну (входу) та реєстрації. Сторінка логіну призначена для користувачів, які вже мають обліковий запис (рис. Б.1). Після введення електронної пошти та пароля слід натиснути кнопку "Увійти". Для нових користувачів передбачено сторінку реєстрації (рис. Б.2), перехід на яку можливий через посилання під формою входу або кнопку у правому верхньому куті. Для реєстрації потрібно ввести ім’я користувача, електронну пошту, пароль та повторення паролю, після чого потрібно натиснути кнопку «Зареєструватися».

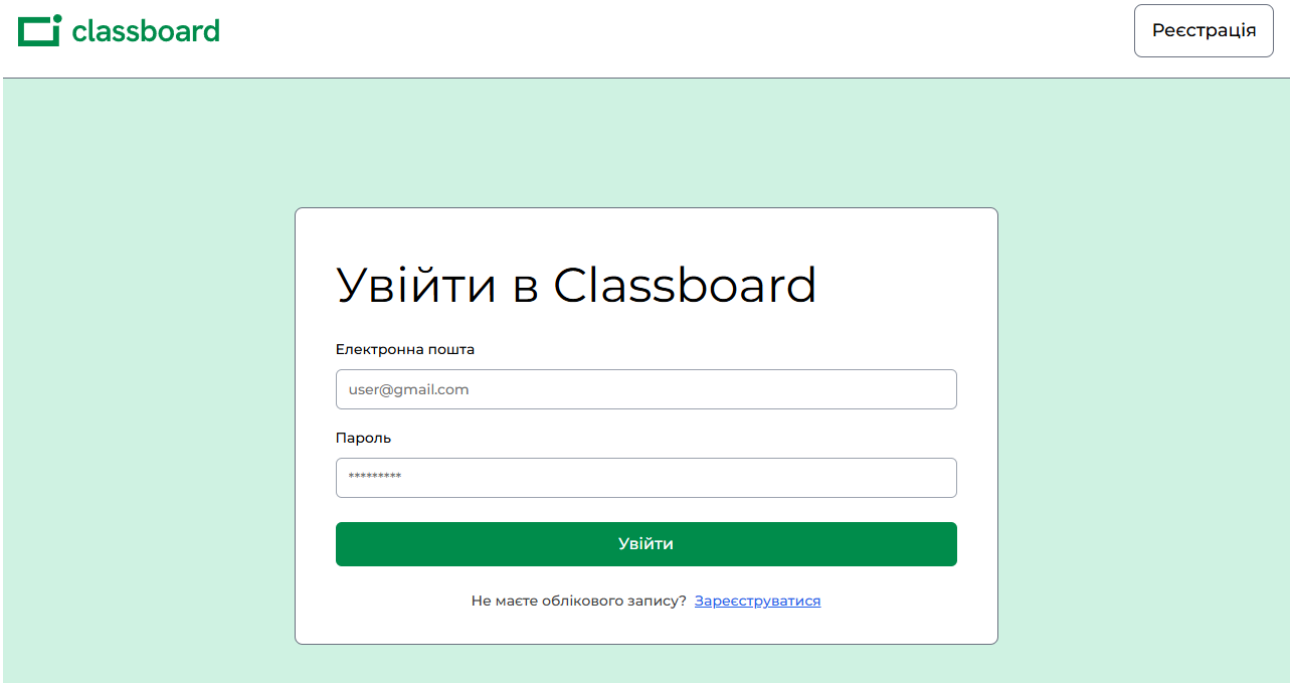

Рисунок Б.1 – Сторінка логіну (входу)

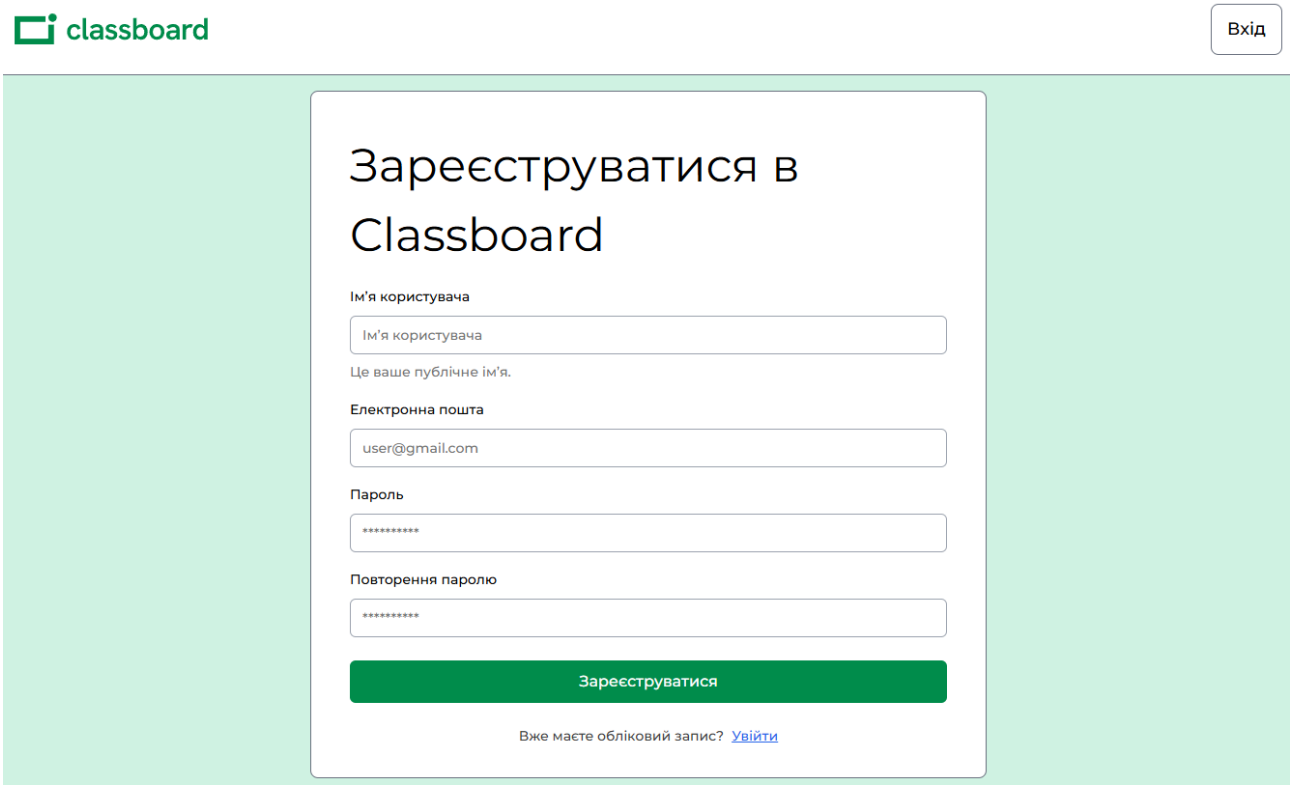

Рисунок Б.2 – Сторінка реєстрації

Після успішного входу користувач потрапляє на головну сторінку, яка виконує роль панелі керування всіма доступними дошками (рис. Б.3).

На цій сторінці знаходяться дошки, які відображаються у вигляді карток. Вони поділяються на дві категорії:

- всі дошки, які створив користувач;
- обрані дошки, які користувач обрав.

Кожна картка містить назву дошки, дату її створення та ім’я автора. Окрім цього, на картці розміщено меню дій: дошку можна додати до категорії «Обрані дошки», перейменувати, видалити або скопіювати посилання для спільного доступу.

У верхньому правому кутку інтерфейсу знаходиться меню профілю користувача, де можна змінити особисті дані або вийти із системи.

Щоб перейти на сторінку відповідної інтерактивної дошки, користувачеві достатньо натиснути на неї.

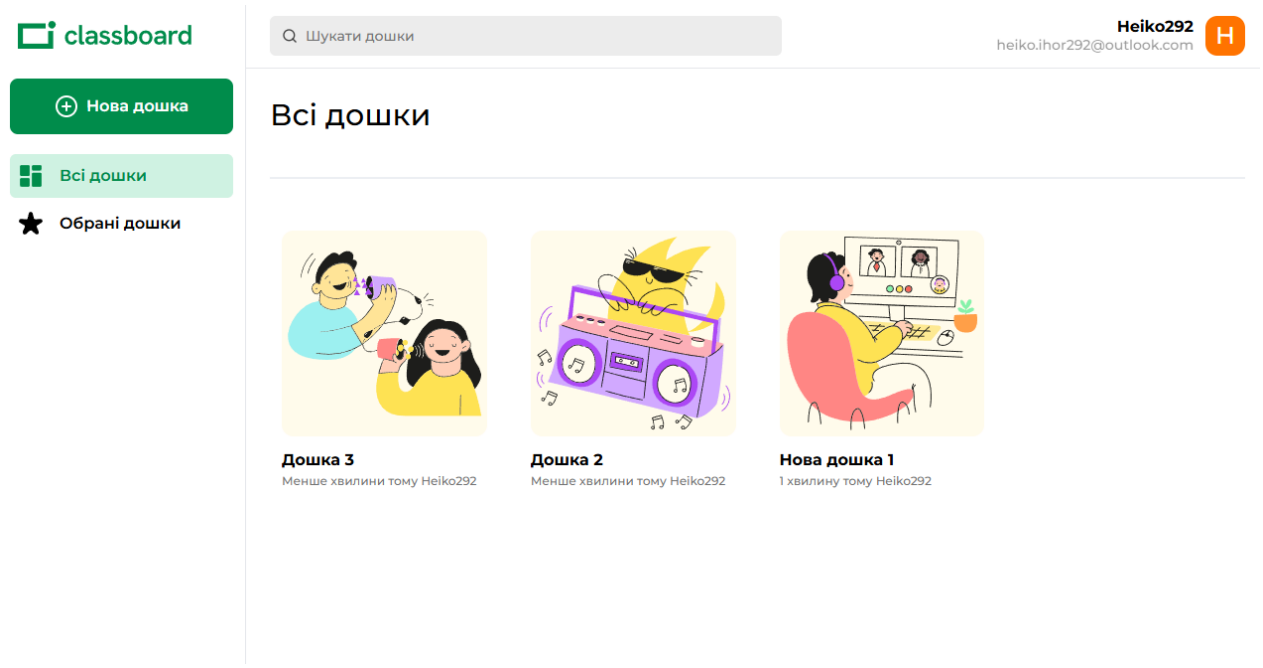

Рисунок Б.3 – Сторінка керування дошками

Інтерфейс інтерактивної дошки організований таким чином, щоб забезпечити зручність роботи користувачів (рис. Б.4).

Центральне місце займає робоча область, де можна створювати малюнки, додавати текст і графічні елементи. В лівому верхньому кутку розташоване меню

користувача, яке надає можливість експорту, зміни фона полотна (робочої області) та переходу на сторінку керування дошками. Внизу екрана розташована панель, яка дозволяє обирати інструменти, такі як переміщення, олівець, текстові блоки та геометричні фігури. Додатково, передбачені кнопки для скасування або повторення останніх дій, що значно полегшує процес роботи. У верхній лівій частині екрана знаходиться бічна панель, що надає можливість змінювати колір, розмір ліній, розмір шрифту, упорядковувати шари та видаляти їх.

Щоб запросити інших користувачів до кімнати, користувач повинен натиснути кнопку «Поділитися», після чого з’явиться модальне вікно. Модальне вікно налаштування доступу призначене для керування правами доступу до інтерактивної дошки. У ньому власник дошки може скопіювати посилання на дошку, переглянути список осіб, які мають доступ, та змінити їхні ролі (наприклад, лише перегляд або редагування). Крім того, доступна функція запрошення нових учасників за електронною поштою.

Щоб здійснити перехід на головну сторінку достатньо натиснути кнопку «Вийти» або логотип вебзастосунку в меню користувача.

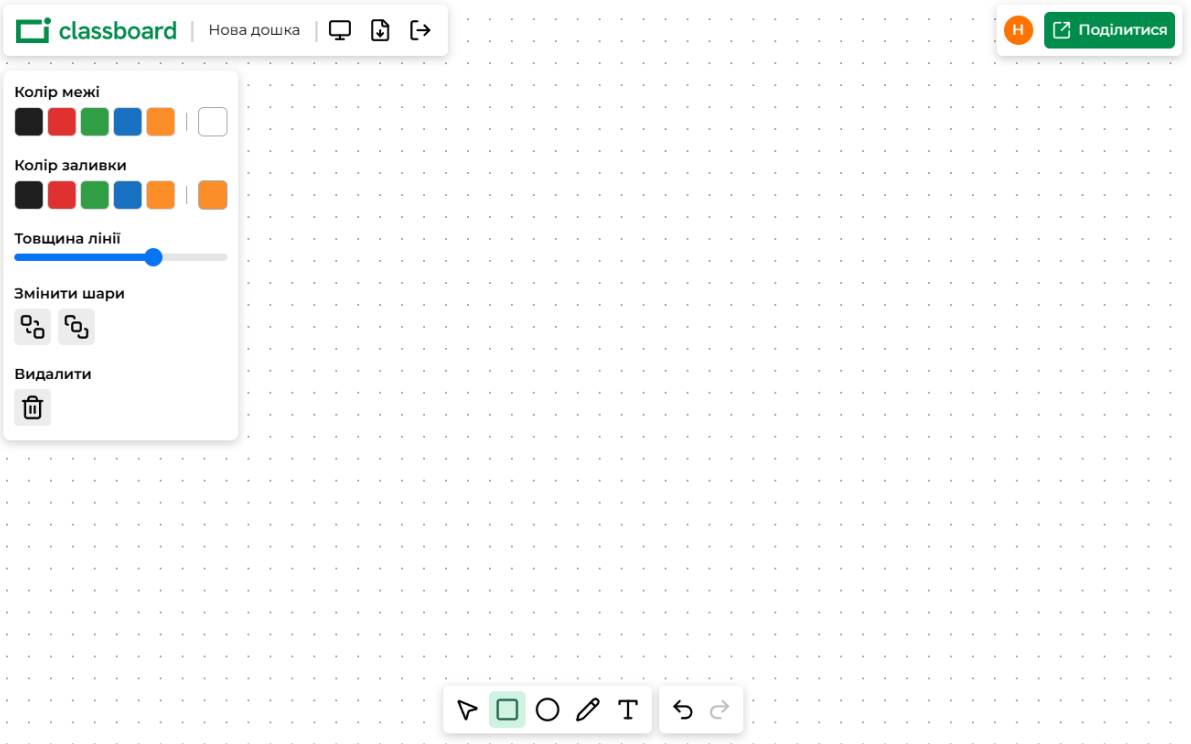

Рисунок Б.4 – Сторінка інтерактивної дошки

АНОТАЦІЯ

Гейко І. В. Дослідження принципів проектування вебзастосунків на прикладі розробки електронної дошки для дистанційного навчання. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025 р.

Робота присвячена дослідженню принципів створення сучасних вебзастосунків на прикладі розробки електронної інтерактивної дошки для дистанційного навчання. В процесі дослідження було проаналізовано актуальні підходи до побудови масштабованих, безпечних і продуктивних систем.

Одним з ключових компонентів реалізованого застосунку є система обробки даних у реальному часі. В основі функціонування дошки лежить технологія WebSocket, яка забезпечує миттєвий обмін інформацією між усіма підключеними користувачами, що є критично важливим для досягнення високої інтерактивності під час спільної роботи.

Система також включає компонент, що відповідає за безпеку та управління доступом. Впроваджено систему аутентифікації на базі JWT та рольову модель авторизації (RBAC). Особливістю реалізації є можливість динамічної зміни ролей користувачів адміністратором дошки в реальному часі, що дозволяє власнику миттєво керувати правами доступу, запрошувати учасників та змінювати їхні ролі (наприклад, лише перегляд або редагування).

У результаті реалізовано прототип електронної дошки для спільної роботи в освітньому середовищі, який відображає специфіку засобів дистанційного навчання та забезпечує виконання практичних завдань зі створення інтерактивних вебзастосунків.

Ключові слова: дистанційне навчання, електронна дошка, проектування вебзастосунків, взаємодія в реальному часі, WebSocket, JWT, RBAC.