БАКАЛАВРСЬКА РОБОТА

БР. ІП - 01.00.00.000 ПЗ

Група ІП-21-1

Гурак Іван

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Гурак Іван Андрійович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Проектування та розробка інтерактивного навчального середовища

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Гурак І.А.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Гураку Івану Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Проектування та розробка інтерактивного навчального середовища”

керівник проекту (роботи) Піх В.Я., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області застосування інтерактивних технологій навчання
2. Алгоритмічна реалізація інтерактивного навчального середовища
3. Проектування серверної частини і протоколів взаємодії
4. Програмна реалізація інтерактивного навчального середовища
5. Проектування дизайну інтерфейсу користувача

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Алгоритм (послідовність) виконання роботи (рис. 1.1)
2. Приклад навчального полотна із хронологічними мітками (рис. 1.2)
3. Схематичне представлення користувачів в системі (рис. 1.3)
4. Діаграма послідовності передачі повідомлень за допомогою протоколу TCP (рис. 2.1)
5. Діаграма послідовності передачі повідомлень за допомогою протоколу UDP (рис. 2.2)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області застосування інтерактивних технологій навчання	07.05.2025	виконано
2	Алгоритмічна реалізація інтерактивного навчального середовища	17.05.2025	виконано
3	Проектування серверної частини і протоколів взаємодії	27.05.2025	виконано
4	Програмна реалізація інтерактивного навчального середовища	01.06.2025	виконано
5	Проектування дизайну інтерфейсу користувача	07.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 75 сторінок, 22 рисунки, список використаних джерел із 32 найменуваннями, 1 додаток.

Мета роботи - розробити та реалізувати інтерактивне навчальне середовище, що забезпечує зручну, ефективну та гнучку взаємодію між користувачами за допомогою сучасних технологій передачі даних, інтерактивних інструментів та хмарної інфраструктури.

Об'єкт дослідження - процес організації інтерактивного навчання в цифровому середовищі.

Предмет дослідження - моделі, алгоритми та програмні засоби реалізації інтерактивного навчального середовища з використанням сучасних протоколів комунікації та веб-технологій.

У першому розділі проведено всебічний аналіз предметної області, що лежить в основі проєктування та впровадження інтерактивного навчального середовища.

У другому розділі було здійснено комплексний аналіз технологічних та алгоритмічних засад реалізації інтерактивного навчального середовища.

У третьому розділі детально описано архітектуру серверної частини, яка реалізує бізнес-логіку та відповідає за обробку запитів, автентифікацію користувачів, а також комунікацію з базою даних.

Висновок: кінцевим результатом роботи є розробка прототипу інтерактивного навчального середовища, що може бути використане як основа для створення сучасних освітніх платформ або впровадження в існуючі цифрові навчальні системи.

КЛЮЧОВІ СЛОВА: ІНТЕРАКТИВНЕ НАВЧАННЯ, АСИНХРОННЕ ПРОГРАМУВАННЯ, НАВЧАЛЬНЕ СЕРЕДОВИЩЕ, ПРОГРАМНА АРХІТЕКТУРА, БАЗА ДАНИХ, UX/UI-ДИЗАЙН, ПРОТОКОЛИ ПЕРЕДАЧІ ДАНИХ, СИНХРОНІЗАЦІЯ ДАНИХ

ANNOTATION

This Bachelor's thesis includes 75 pages, 22 figures, a list of 32 references, and 1 appendix.

The goal of this work is to develop and implement an interactive learning environment that provides convenient, efficient, and flexible user interaction through modern data transfer technologies, interactive tools, and cloud infrastructure.

The object of the research is the process of organizing interactive learning in a digital environment.

The subject of the research includes models, algorithms, and software tools for implementing an interactive learning environment using modern communication protocols and web technologies.

The first chapter provides a comprehensive analysis of the subject area underpinning the design and implementation of an interactive learning environment.

The second chapter conducts a comprehensive analysis of the technological and algorithmic principles for implementing an interactive learning environment.

The third chapter details the architecture of the server-side, which implements business logic and is responsible for request processing, user authentication, and communication with the database.

Conclusion: The final result of this work is the development of a prototype interactive learning environment that can serve as a foundation for creating modern educational platforms or be integrated into existing digital learning systems.

KEY WORDS: INTERACTIVE LEARNING, ASYNCHRONOUS PROGRAMMING, LEARNING ENVIRONMENT, SOFTWARE ARCHITECTURE, DATABASE, UX/UI DESIGN, DATA TRANSFER PROTOCOLS, DATA SYNCHRONIZATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІНС — Інтерактивне навчальне середовище

UI — User Interface (інтерфейс користувача)

UX — User Experience (досвід користувача)

JSON — JavaScript Object Notation (формат обміну даними)

XML — eXtensible Markup Language (розширювана мова розмітки)

WebSocket — протокол двосторонньої комунікації в реальному часі поверх TCP

DB — Database (база даних)

ВСТУП

Сучасний етап розвитку освіти характеризується активним впровадженням цифрових технологій, зокрема інтерактивних навчальних середовищ, які забезпечують нові можливості для ефективного засвоєння знань, гнучкої організації навчального процесу та підвищення рівня залучення студентів. В умовах дистанційного та змішаного навчання потреба у таких рішеннях є особливо актуальною.

Сучасна освітня система стрімко трансформується під впливом цифрових технологій, що зумовлює необхідність переходу від традиційних форм навчання до інтерактивних, адаптивних і більш ефективних моделей засвоєння знань. Особливої актуальності набувають програмні рішення, що дозволяють організувати повноцінний навчальний процес у цифровому середовищі з використанням інноваційних засобів — інтерактивних дошок, хмарних платформ, синхронної та асинхронної взаємодії. У даній роботі розглядається проєктування та реалізація інтерактивного навчального середовища, яке поєднує сучасні підходи до навчання з технічними рішеннями, що забезпечують гнучкість, масштабованість і зручність у використанні як для викладачів, так і для студентів. Основна увага приділяється архітектурі системи, алгоритмічному та програмному забезпеченню, а також способам обміну даними між користувачами.

Актуальність роботи

У зв’язку зі зростаючою потребою в дистанційному та гібридному навчанні, а також викликами, що постають перед традиційною освітою, інтерактивні навчальні середовища стають важливою складовою освітнього процесу. Пандемія COVID-19 лише пришвидшила цифровізацію освіти, виявивши обмеження існуючих платформ та необхідність у нових рішеннях, які б забезпечували якісну взаємодію в режимі реального часу, високий рівень візуалізації навчального матеріалу та простоту інтеграції з іншими

сервісами. Таким чином, розробка ефективного інтерактивного середовища навчання, яке ґрунтується на сучасних протоколах передачі даних, адаптивній структурі і можливостях масштабування, є вкрай актуальною на сучасному етапі розвитку освіти.

Інтерактивне навчальне середовище — це система, що поєднує інструменти управління навчанням, засоби комунікації та обміну інформацією між учасниками освітнього процесу. Вона має підтримувати адаптивність до потреб користувача, стабільність під час одночасної роботи багатьох користувачів, а також відповідати принципам сучасної методології навчання.

Мета роботи - розробити та реалізувати інтерактивне навчальне середовище, що забезпечує зручну, ефективну та гнучку взаємодію між користувачами за допомогою сучасних технологій передачі даних, інтерактивних інструментів та хмарної інфраструктури.

Завдання дослідження

1. Проаналізувати предметну область інтерактивного навчання та сучасні освітні методології.

2. Сформулювати вимоги до системи на основі обраної методології.

3. Обґрунтувати вибір платформи, протоколів та технологій для розробки.

4. Розробити архітектуру програмної частини системи.

5. Реалізувати базу даних, серверну логіку та інтерфейс користувача.

6. Забезпечити ефективний обмін даними між користувачами системи з використанням WebSocket.

7. Оцінити ефективність розробленого середовища в контексті реального навчального процесу.

Об’єкт дослідження - процес організації інтерактивного навчання в цифровому середовищі.

Предмет дослідження - моделі, алгоритми та програмні засоби реалізації інтерактивного навчального середовища з використанням сучасних протоколів комунікації та веб-технологій.

Методи дослідження:

- системний аналіз;
- порівняльний аналіз технологій та протоколів;
- моделювання програмної архітектури;
- проектування баз даних;
- алгоритмічний аналіз синхронізації даних;
- експериментальна перевірка ефективності взаємодії користувачів у середовищі.

Наукова новизна

Запропоновано архітектурну модель інтерактивного навчального середовища, яке базується на асинхронній комунікації з використанням WebSocket та JSON/XML для обміну даними в реальному часі, що дозволяє значно покращити швидкість реакції системи та зручність користування у порівнянні з традиційними рішеннями.

Практичне застосування

Розроблена система може бути використана у навчальних закладах для проведення інтерактивних занять, практичних робіт, семінарів, а також у корпоративному навчанні для підвищення кваліфікації співробітників.

Бакалаврська робота містить 75 сторінок, 22 рисунки, 3 розділи список використаних джерел із 32 найменуваннями, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ІНТЕРАКТИВНИХ ТЕХНОЛОГІЙ НАВЧАННЯ

1.1. Представлення структури роботи

На першому етапі ми розглянемо, як працює оригінальна навчальна методологія. З її принципів зробимо короткий аналіз вимог, а для задоволення цих вимог зробимо кілька технічних дизайнерських рішень. Потім ми розглянемо низку варіантів, перш ніж прийняти рішення. Нарешті, ми детально розглянемо кілька аспектів реалізації.

Рисунок 1.1 – Алгоритм (послідовність) виконання роботи

На рисунку 1.1 зображено лінійний алгоритм, що складається з чотирьох послідовних етапів. Кожен етап є кроком у процесі розробки або створення чогось, починаючи з ідеї та закінчуючи її втіленням.

Опис цього алгоритму:

- Original teaching methodology (Оригінальна методика навчання): Це початковий етап, який представляє собою існуючий підхід або концепцію, що є відправною точкою для подальшого розвитку або вдосконалення. У контексті розробки освітніх технологій, це може бути наявна методика викладання, яку планують перенести або адаптувати в нове середовище.

- Requirements analysis (Аналіз вимог): На цьому етапі відбувається вивчення та визначення потреб і специфікацій для майбутнього продукту або системи. Визначаються функціональні та нефункціональні вимоги, цілі

користувачів та обмеження. Цей етап є критично важливим для розуміння того, що саме потрібно створити.

- Design choices (Вибір проектування): Після того, як вимоги визначено, на цьому етапі приймаються рішення щодо архітектури, структури, технологій та інших аспектів майбутнього продукту або системи. Розробляються плани та моделі, які визначають, як буде реалізовано визначені вимоги.

- Implementation (Реалізація): Це заключний етап, на якому відбувається безпосереднє створення продукту або системи на основі прийнятих проектних рішень. Проводиться написання коду, розробка інтерфейсу, налаштування обладнання тощо. Результатом цього етапу є готовий до використання продукт або система.

Таким чином, алгоритм на рисунку відображає послідовний процес від вихідної ідеї через аналіз потреб та проектування до кінцевої реалізації.

1.2. Опис використовуваної методології навчання

У цьому розділі ми розглянемо методологію, яка може бути використана для підтримки викладання історії. Для того щоб розмірковувати про історію, обов'язково потрібно мати додаткове знання про період часу, який розглядається. Коли ми говоримо про вивчення історії, ми зазвичай маємо на увазі процес розгляду певного періоду часу, зосереджуючись на певному місці - країні або континенті, щоб краще зрозуміти зміни, які відбулися протягом цього часу. Звичайно, це включає детальний розгляд подій, які відбулися, та (політичних) рухів та ідеологій, які їх поєднують.

Хоча вивчення фактів може бути відносно простим, цей процес є більш складним, ніж здається на перший погляд. Часто не існує такого поняття, як істина при реконструкції історичного контексту. Це стає очевидним, коли ми розглядаємо те, що надані факти можуть бути селективними або навіть

неправдивими. Крім того, існує багато кутів зору на одну і ту ж проблему - зміни не відбуваються за одну ніч і є результатом багатьох подій, які відбулися раніше.

Процес історичної контекстуалізації є складним процесом; не існує єдиного "правильного" відповіді. Крім того, об’єкти навчання мають труднощі з поєднанням здавалося б окремих подій в безперервний хронологічний огляд.

Конструювання історичного контексту настільки складне, що керівництво через добре структуровану навчальну послідовність недостатнє. Вчитель повинен задавати питання, щоб ще більше оцінити та викликати (початкові) відповіді та аргументи об’єктів навчання. Для цього потрібно, щоб їх мислення було видимим для вчителя. Робота в групах та класні обговорення є корисними інструментами для того, щоб зробити мислення учнів видимим."

Викладачі можуть використовувати методологію навчання “Активне історичне мислення” для надання такого огляду історичних подій та рухів. Ідея полягає в тому, щоб дати об’єктам навчання уявлення про те, як розвивається історія: не лише як окремий ряд подій, а швидше як безперервний процес. Цей процес називається історичною контекстуалізацією.

Об’єкти навчання розміщуються в групах по дві або три особи, кожна з яких ділиться папером формату А3 з надрукованим на ньому шаблоном часової шкали, що містить кілька замінників, а також осіб, які цікавлять. Об’єкти навчання повинні заповнити цю часову шкалу за допомогою карт. Це робиться в кілька раундів.

На початку кожного раунду кожна група отримує конверт з наступним набором карт. Протягом обмеженого часу об’єкти навчання потім повинні розмістити карти з цього набору на часовій шкалі, крім карт, які вже розміщені на полотні. Суперечки можуть виникнути тут, і оскільки всі

об’єкти навчання мали однакові заняття до заповнення часової шкали, це може призвести до цікавих обговорень. Це стимулює критичне мислення щодо того, як події пов'язані між собою.

Коли час закінчується, раунд оцінюється. Це обговорення веде викладач, який виконує роль керівника та спостерігача в цій методології: чому певні карти були розміщені в певні моменти часу? Як вони пов'язані з наступними та активними рухами? Як люди, згадані на картах, грали в них роль? І так далі.

Для ілюстрації методу ми розглянемо хронологію, яка складається з чотирьох раундів, що показано на рисунку 1.2.

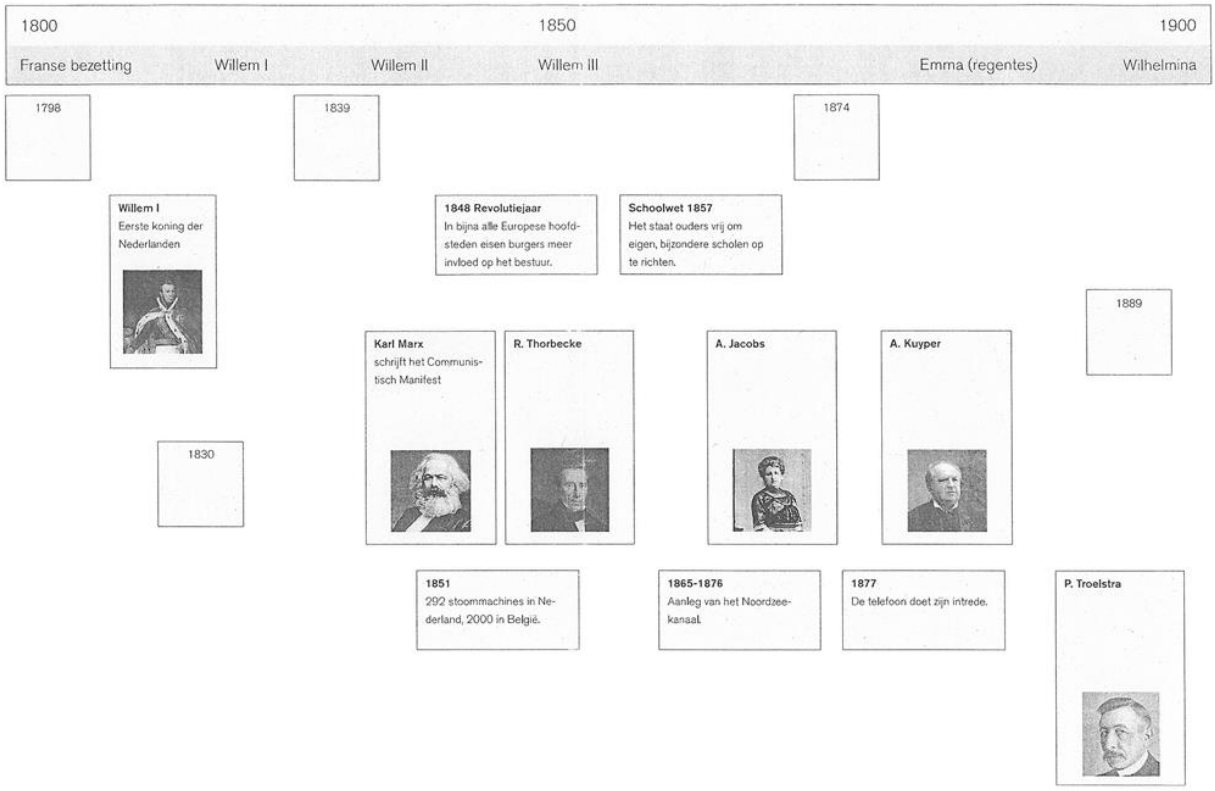

Рисунок 1.2 – Приклад навчального полотна із хронологічними мітками

В першому раунді об’єкти навчання повинні розмістити історичні події та описи ключових фігур на часовій шкалі. Кожна відповідна карта повинна бути розміщена в одному з замінників, безпосередньо відображаючи частини,

охоплені в лекціях. Очікується, що об’єкти навчання карти в правильних слотах протягом 10 хвилин. Після цього раунд коротко оцінюється: де групи розмістили певні карти, і чому? Особливо в разі дисонансу: це створює когнітивну невідповідність.

Другий раунд, який також триває 10 хвилин, передбачає, що об’єкти навчання визначать початок рухів та ідеологій та розміщують їх карти над роками на часовій шкалі. Для цього вони повинні уважно розглянути карти на полотні та поєднати їх між собою. Під час оцінки викладач може спровокувати обговорення щодо того, де були розміщені карти. Наприклад, карта соціалізму пов'язана з появою радіо.

На основі ідеологій, розміщених у другому раунді, третій раунд передбачає, що об’єкти навчання зв'яжують ключові слова з цими політичними або соціальними рухами. Цей процес триває трохи довше, ніж перші два раунди: загалом 15 хвилин. Знову слідує оцінка, під час якої учні обговорюють, чому певні ключові слова асоціюються з певними ідеологіями.

Нарешті, четвертий раунд передбачає, що об’єкти навчання працюють з трохи більшими картами, на яких є цитовані джерела. Після розміщення їх у правильний час вони також повинні зв'язати їх принаймні з одним ключовим словом з попереднього раунду. Для цього дається близько 20 хвилин на це, після чого слідує обговорення, під час якого можна враховувати все полотно.

1.3. Принципи проектування на основі методології навчання

Одним з головних аспектів методу є співпраця між об’єктами навчання. Шляхом обговорення та дискусії вони створюють власну контекстуалізацію історичних подій. Як ми згадували раніше, це складніше, ніж здається: хоча вони можуть знати факти, але вони можуть не мати необхідних додаткових фонових знань, які завжди доступні.

Ми можемо абстрагуватися від історичного фону повністю та зосередитися виключно на полотні, над яким працюють об’єкти навчання. У цій роботі ми проектуємо систему таким чином, щоб викладачі могли створювати власні модулі, включаючи часову шкалу та карти, тим самим делегуючи відповідальність за вміст.

У процесі співпраці об’єктів навчання над їхніми полотнами викладач виконує роль керівника та наставника. Він не обов'язково виправляє об’єктів навчання, а швидше стимулює їхній процес мислення: чому карти розміщені там, де вони розміщені? Як події пов'язані між собою? Змусити об’єктів навчання запитати свої власні знання - це не легкий процес, але необхідний, щоб вони прогресували у своїх переконаннях.

Автори в [3] узагальнюють свою роботу наступними дизайнерськими принципами:

1. Викликати історичні знання: створити когнітивну невідповідність

- Створити історичну напругу (минуле як інше) шляхом надання об’єктам навчання інформації, яка суперечить їхнім попереднім знанням і є внутрішньо суперечливою.

- Викликати їхні знання та навички (знання та діяльність історії).

2. Стимулювати обґрунтовані роздуми

- Задати об’єктам навчання (оціночне) питання, щоб кілька хороших відповідей.

- Зосередити (оціночне) питання на історичній контекстуалізації шляхом уваги до часу/тривалості, причини/зміни та місцезнаходження.

- Спрямовувати розвиток їхніх епістемологічних переконань, задаючи їм сформулювати альтернативні відповіді.

- Спрямовувати розвиток їхньої епістемологічної позиції, оцінюючи їхні відповіді на те, як знання та діяльність історії поєднуються.

- Стимулювати використання попередніх знань шляхом задавання питань.

- Стимулювати мислення про взаємозв'язки між фактами та концепціями за допомогою колігаторних концепцій.

3. Скаффолдинг навчання

- Структурувати питання шляхом надання фактичної інформації та оціночного питання на початку навчальної послідовності;

- Надання відповідних колігаторних концепцій;

- Структурувати навчальну послідовність шляхом навчальних дій шляхом чіткого введення та надання чітких інструкцій;

- Чергування навчальних дій (робота в парах або трійках, обговорення та індивідуальні завдання);

- Сприяти навчанню шляхом постійного керівництва;

- Зробити історичне мислення об’єктів навчання видимим шляхом підсумування всієї навчальної послідовності, явно звертаючись до знання та діяльності історії та їхніх переконань.

1.4. Вимоги до інтерактивної системи навчання

Розглянувши, як працює метод як навчальна методологія, ми тепер спробуємо розбити його на менші компоненти з технічної точки зору.

Як ми обговорювали, в оригінальному методі об’єкти навчання співпрацюють, заповнюючи часову шкалу на спільному аркуші паперу формату А3, використовуючи паперові карти для заповнення пустих місць полотна. Під час цього процесу вони вербально обговорюють історичний контекст для визначення того, куди розмістити карти.

Під час переходу до цифрової інтерактивної системи ми перенесемо полотно на планшети клієнтів, використовуючи поширення інтерактивних особистих пристроїв. Однак спілкування між об’єктами навчання все ще буде здійснюватися вербально.

Три аспекти пропонованого дизайну потребують аналізу вимог:

- Інтерактивне місце об’єкту навчання;
- Програмне забезпечення інтерактивної дошки для викладача;
- Сервер, який використовується для обміну інформацією;

1.4.1. Інтерактивне місце об’єкту навчання

Об’єкти навчання будуть використовувати планшети для заповнення цифрової версії часової шкали. Модель планшетів буде варіюватися, але це буде або iPad, або пристрій Android, тому програмне забезпечення повинно бути сумісним з обома операційними системами iOS та Android.

Коли об’єкт навчання відкриває додаток, першим кроком буде процес входу в систему для аутентифікації. Це повинно бути зроблено шляхом простого поєднання адреси електронної пошти та пароля, пов'язаного з навчальним закладом і зберігатиметься на стороні сервера. Те, в якому приміщенні навчального закладу вони знаходяться, впливає на те, які сесії вони можуть бачити - таким чином, це діє як рівень доступу.

Після успішного входу в систему користувач переходить до списку поточних та минулих сесій, до яких він може отримати доступ. Кожна з цих сесій є посиланням між навчальним модулем та приміщенням навчального закладу (аудиторією). Один і той же модуль може викладатися в кількох аудиторіях, але оскільки об’єкти навчання повинні бути фізично присутніми для обговорень, це потрібно для такого інстанціювання.

Вибір однієї з цих сесій переведе користувача на екран вибору групи. Тут вони можуть зв'язати свій пристрій з іншими групами, які присутні. Більшість модулів передбачають роботу в парах або групах з трьох людей. Будь-який користувач може приєднатися до будь-якої групи, за умови, що квота ще не досягнута. Якщо ніхто з членів їхньої групи ще не створив групу, один з них повинен натиснути кнопку "Додати нову групу". Це відкриє нову

групу, а створювач стане її єдиним членом. Інші користувачі можуть приєднатися до щойно створеної групи.

1.4.2 Особливості екземпляру робочого середовища

Кожна з груп має спільний екземпляр полотна (робоче середовище). Цей екземпляр полотна - це колекція карт, які користувачі повинні розмістити - змінна частина, яка залежить від групи. Кожен такий екземпляр карти може бути або розміщений на полотні, або знаходитися в стопці карт. Оскільки кожен з членів групи ділиться полотном, відповідно, зміни одного користувача повинні бути видимі іншим членам групи. Ми розглянемо, як це можна забезпечити на технічному рівні в наступному розділі.

Екземпляри карт, які ще не розміщені на часовій шкалі, будуть збережені в стопці. Це буде згорнутий шар збоку екрану. Користувачі можуть перетягувати карту з цієї стопки на полотно. Коли це перетягування зупиняється, зміна повинна бути видимою на всіх планшетах, які беруть участь.

Цей спосіб співпраці піднімає цікаві питання синхронізації. Ми розглянемо їх у наступному розділі.

Також слід зазначити, що карти можуть продовжувати перетягуватися після їх початкового розміщення. Система не автоматично перевіряє, чи карти знаходяться в "правильних" позиціях. Шляхом обговорення та співпраці користувачі знайдуть "правильні" позиції самостійно.

Як вже згадувалося, кожен навчальний модуль складається з кількох раундів. Об’єкти навчання зможуть бачити лише карти, які беруть участь у послідовних раундах на даний момент, і залежатимуть від викладача, який роздає нові карти на стопки груп після того, як він задоволений результатами поточного раунду. Таким чином, викладач повинен мати можливість додавати карти до екземплярів полотен груп у своїх аудиторіях. Ми розглянемо це в наступному розділі.

1.4.3. Програмне забезпечення інтерактивної дошки

Як і клієнт об’єктів навчання, коли викладач відкриває додаток, першим кроком буде процес входу в систему для аутентифікації. Знову ж, це повинно бути зроблено шляхом простого поєднання адреси електронної пошти та пароля, пов'язуючи ці облікові дані з привілеями і в зберіганні на стороні сервера.

Основним для програмного забезпечення інтерактивної дошки є система агрегації, за допомогою якої викладач може отримати ілюстративний огляд полотен усіх груп. Це може бути використано не лише для виявлення прогалин у знаннях, а й для демонстраційних цілей: чому деякі об’єкти навчання вирішили розмістити певну особу в певний час, тоді як інші розміщують її на кілька років пізніше?

Ця система не повинна мати багато функцій, але повинна працювати інтуїтивно. Ідея полягає в тому, що після відкриття модуля, який цікавить, викладач може вибрати кілька карт зі стопки для вивчення. Ці карти потім отримують унікальний колір. Для кожної з груп карт, які цікавлять, потім будуть представлені кольоровою крапкою в позиціях, де групи їх розмістили. Якщо більше однієї групи розмістила ту саму карту в тому самому місці, крапка буде більшою і отримає кількість груп як підпис.

Крім інструменту, який використовується, програмне забезпечення клієнта викладача також повинно виконувати функцію управлінського інструменту. Викладачі повинні мати можливість створювати облікові записи для своїх користувачів та додавати їх до груп.

Крім того, викладачі повинні мати можливість створювати, імпортувати та експортувати модулі, включаючи додавання карт та міток часової шкали. Нарешті, вони повинні мати можливість починати та керувати вищезгаданими сесіями, а також переходити до наступного раунду в системі навчання.

1.4.4. Стек програмного забезпечення

Оскільки система розробляється в навчальних цілях то вона має бути спроектована як система, щоб є вільно доступною. З цієї причини ми будемо проектувати наш підхід з урахуванням технологій з відкритим вихідним кодом. Це означає, що ми будемо використовувати інструменти, фреймворки та мови програмування, які є вільно доступними - тобто вони не вимагають ліцензійної плати за використання.

Хоча це обмежує вибір не включати програмне забезпечення, яке доступне лише комерційно, ми сподіваємося, що це означає, що кінцеве програмне забезпечення буде легко розгорнути.

1.5. Графічне представлення розподілу користувачів в системі

Рисунок 1.3 показує схематичний огляд розподілу клієнтів у системі. Загалом два клієнти об'єднані разом, але все ще працюють та спілкуються автономно.

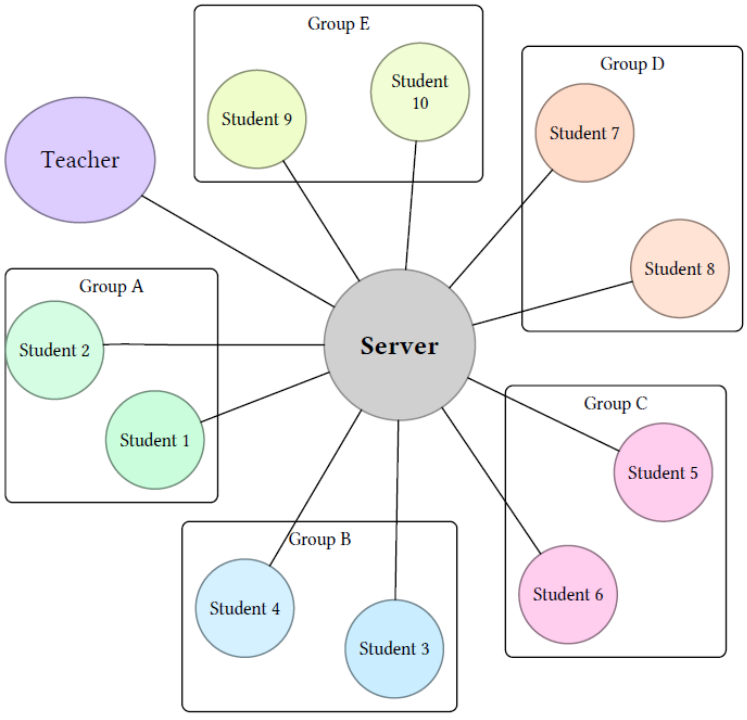

Рисунок 1.3 – Схематичне представлення користувачів в системі

Як показано, викладач не об'єднаний безпосередньо з жодним з об’єктів навчання, але отримує свою інформацію безпосередньо від сервера. Оскільки групи будуть ділитися станом своїх полотен з цим сервером, викладач зможе бачити їхній прогрес таким чином.

На рисунку 1.3 зображено схему користувачів навчальної системи, яка має централізовану архітектуру типу "зірка". У центрі знаходиться Сервер, до якого підключені всі інші елементи системи.

Основними користувачами та організаційними одиницями є:

- Teacher (Викладач). Представлений одним великим колом світло-фіолетового кольору, підключеним безпосередньо до сервера. Це вказує на те, що він має зв'язок із сервером для керування навчальним процесом.

- Groups (Групи). Представлені п'ятьма прямокутними блоками з позначеннями Group A, Group B, Group C, Group D та Group E. Кожна група об'єднує кількох студентів.

- Students (Студенти): Представлені меншими колами різних кольорів, розміщеними всередині відповідних груп. Кожен студент ідентифікується номером (Student 1, Student 2, ..., Student 10) та належить до певної групи. Кожен студент у групі також має прямий зв'язок із сервером.

Опишемо зв'язки представлені на рисунку 1.3:

- Викладач має прямий зв'язок із сервером, що дозволяє йому, ймовірно, керувати сесіями, завданнями, переглядати прогрес студентів тощо.

- Кожен студент також має прямий зв'язок із сервером. Це може означати, що студенти взаємодіють із навчальним матеріалом, виконують завдання та спілкуються через сервер.

- Студенти об'єднані в групи (Group A, B, C, D, E), що може вказувати на можливість групової роботи, спільного виконання завдань або організації навчального процесу за групами. Однак, прямих зв'язків між студентами

всередині групи на схемі не показано, що може означати, що вся комунікація та обмін даними відбувається через сервер.

Отже, схема відображає централізовану навчальну систему, де сервер виступає посередником у взаємодії між викладачем та студентами, а також між студентами (опосередковано через групову організацію). Викладач має центральну роль у керуванні, а студенти організовані в групи для навчальної діяльності, при цьому кожен індивідуально підключений до сервера.

Висновки до розділу

В даному розділі проведено опис методології навчання, що дав змогу окреслити підхід до організації освітнього процесу з урахуванням активної участі здобувачів освіти та використання сучасних цифрових технологій. На основі цієї методології сформульовано принципи, які визначають напрями проектування системи, зокрема гнучкість, адаптивність, інтуїтивність та підтримку зворотного зв’язку.

Окремо проаналізовано вимоги до інтерактивної системи навчання, серед яких: забезпечення індивідуалізованого робочого середовища, підтримка інтерактивного місця об’єкту навчання, інтеграція з програмним забезпеченням інтерактивних дощок та відповідність сучасному стеку технологій.

Графічне представлення розподілу користувачів у системі дозволило візуалізувати модель її функціонування та взаємодії різних ролей, що є важливим для подальшого етапу проєктування програмної архітектури.

Узагальнюючи, аналіз предметної області дав змогу сформувати чітке бачення функціональних і технічних вимог до інтерактивного навчального середовища, що створює основу для його алгоритмічної та програмної реалізації.

РОЗДІЛ 2. АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО НАВЧАЛЬНОГО СЕРЕДОВИЩА

2.1. Особливості вибору платформи для розробки

Одним з ключових рішень у процесі розробки є те, чи ми хочемо побудувати нативний додаток або веб-додаток. Будь-який вибір впливає на те, які технології зв'язку будуть доступні нам. Наприклад, веб-додаток не має безпосереднього доступу до Bluetooth або WiFi технологій, тому весь зв'язок має проходити через гіпертекстовий протокол.

Розглянемо основні переваги та недоліки кожного підходу. Відзначається, що веб-додатки значно дешевші завдяки тому, що всі платформи ділять один і той же код. Їхній найбільший недолік полягає в тому, що вони не ділять продуктивність своїх нативних аналогів, що є помітним при виконанні, наприклад, 3D маніпуляцій зображень.

Особливим недоліком створення нативного додатка є те, що він обмежений однією платформою. Хоча можливо поділитися логічним кодом в певній мірі, кожна платформа має свої власні інтерфейси користувача. Отже, код для інтерфейсу користувача, який зазвичай займає значну частину коду, повинен бути написаний спеціально для платформ, які цікавлять. На жаль, це зазвичай призводить до багатьох схожих, але все ж таки різних кодів.

Особливою причиною повільнішої реакції веб-додатків є продуктивність JavaScript та конструкція DOM (Document Object Model). Отже, оптимізація є предметом багатьох досліджень, але тепер ця продуктивність значно зросла. Зусилля великих виробників браузерів, включаючи Microsoft, Google, Mozilla та Apple, значно прискорили процес інтерпретації та виконання.

Ураховуючи їхні відмінності, не дивно, що вибір між нативним та веб-додатком має великий вплив на програмний стек, який ми будемо

використовувати. З точки зору мов програмування, нативні додатки зазвичай побудовані за допомогою мов програмування з жорсткою типізацією (наприклад, C++, Java, Objective-C), тоді як веб-додатки часто побудовані за допомогою мов скриптування з м'якою типізацією (наприклад, JavaScript, PHP, Python) на основі веб-сервера.

У випадку веб-додатків розробники також можуть вибрати побудову свого додатка в мові з жорсткою типізацією, який потім обслуговується через веб-сервер. Приклади таких мов - Java на TomCat або будь-яка компільована мова за допомогою CGI (Common Gateway Interface) для інтерфейсу з веб-сервером.

2.2.1. Протоколи передачі даних

В залежності від того, чи ми будемо будувати додаток, нам доведеться визначити, які апаратні можливості планшетів ми будемо використовувати для передачі та обміну даними. Всі планшети, які беруть участь, мають два засоби бездротового зв'язку короткої дії: стек Bluetooth та стек WiFi. Ми коротко обговоримо обидва стеки.

Зв'язок Bluetooth працює шляхом парного зв'язку пристроїв один з одним: один майстер-пристрій ініціює та керує зв'язками з дочірніми пристроями. Процес парного зв'язку здійснюється з явною згоди з обох сторін шляхом обміну ключем: ініціюючий пристрій надає його, після чого отримувач повинен ввести його для підтвердження парного зв'язку. Після цього пристрої можуть спілкуватися через безперервний, зашифрований зв'язок.

Як згадувалося, існує ієрархія пристроїв: в будь-який момент часу лише один з пристроїв може бути майстер-пристроєм. Лише майстер-пристрій може надсилати повідомлення; інші знаходяться лише на приймальній стороні. Шляхом протоколу пристрої можуть змінювати ролі - тобто інший пристрій в мережі стає майстер-пристроєм. В будь-який момент

часу лише один пристрій в мережі може бути майстер-пристроєм - всі інші призначаються роллю дочірнього пристрою.

Стек WiFi - це мережеве рішення на основі точки доступу. Це означає, що для створення мережі один з пристроїв повинен взяти на себе роль маршрутизатора або роутера. Хоча це може здатися трохи незручним, це має одну велику перевагу: мережа є повнодуплексною: вона дозволяє постійний зв'язок в обох напрямках.

Використання WiFi має ще одну перевагу: ми можемо використовувати існуючу інфраструктуру. Аудиторії оснащені маршрутизаторами WiFi для забезпечення когристувачів інтернет-зв'язком.

2.1.2. Опис мережевого протоколу обміну інформацією

Якщо ми вирішимо побудувати нативний додаток, нам доведеться визначити, який мережевий протокол використовувати для спілкування з сервером. Існують два поширених протоколи, які використовуються разом з Інтернет-протоколом (IP): протокол передачі керування (TCP) та протокол користувацьких датаграм (UDP). Обидва можуть використовуватися для передачі одних і тих самих даних, але мають різні властивості.

TCP - це протокол на основі зв'язку, що означає, що перед кожним корисним навантаженням відбувається взаємне погодження. Зв'язок відкривається шляхом надсилання SYN (синхронізація) пакета, який отримувач може прийняти шляхом SYNACK (синхронізація-підтвердження) пакета. Після цього зв'язок встановлено: відправник надсилає свої повідомлення в порядку, а доставка кожного повідомлення підтверджується отримувачем за допомогою ACK пакета, перш ніж наступне повідомлення буде надіслано. Рисунок 2.1 ілюструє це за допомогою діаграми послідовності.

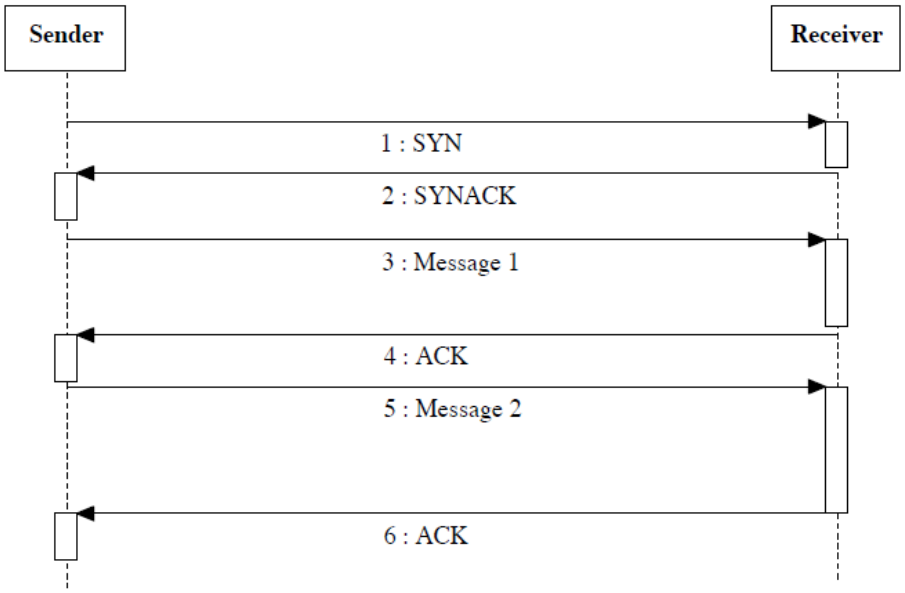

Рисунок 2.1 - Діаграма послідовності передачі повідомлень за допомогою протоколу TCP

Одним з важливих висновків для наших цілей є те, що протокол TCP гарантує, що корисне навантаження (дані) передається остаточно неушкодженим, а також у правильному порядку. Це може здатися очевидним, але якщо це обробляється на мережевому рівні за допомогою TCP, ми можемо абстрагуватися від цього повністю на рівні додатків.

Через ці переваги TCP має певні накладні витрати: він вимагає встановлення зв'язку, перш ніж будь-які дані можуть бути надіслані. Хоча це супроводжується певними гарантіями, які є приємними, деякі цілі можуть не потребувати гарантованої доставки та скоріше виграли б від швидкості статичного протоколу.

На відміну від TCP, UDP - це статистичний протокол, що означає, що пакети надсилаються безпосередньо до їхнього призначення без попереднього встановлення зв'язку. Як і TCP, кожен пакет містить контрольну суму для перевірки цілісності пакета. Однак немає підтвердження отримання з боку отримувача. Хоча це робить протокол легким, це також обмежує можливості після отримання пошкодженого повідомлення. Рисунок 2.2 ілюструє це за допомогою діаграми послідовності.

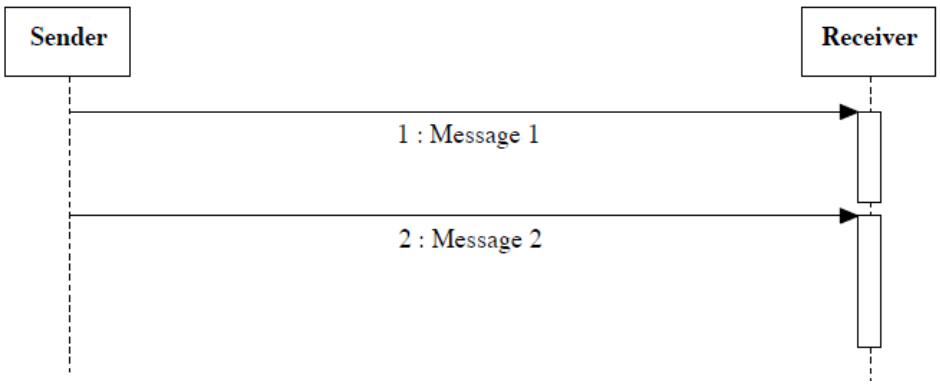

Рисунок 2.2 – Діаграма послідовності передачі повідомлень за допомогою протоколу UDP

Статистична природа UDP робить його ідеальним і швидким, однак він використовується для протоколів, які вважають швидкість найважливішою, як-от система доменних імен (DNS) та DHCP (протокол динамічної конфігурації хостів), які використовуються для пошуку мережі та регулювання.

Інше поширене використання UDP можна знайти в багатокористувацьких іграх. Тут стан гри надсилається кілька разів на секунду. Особливо в іграх, де низька затримка є найважливішою (наприклад, ігри, які вимагають швидкої реакції), переваги надсилання стану вчасно переважають недоліки - помилки можуть бути оброблені грою через наступні пакети.

Отже, UDP швидкий, але делегує відповідальність за обробку помилок на рівень додатків. Для наших цілей нам потрібно визначити, чи це найбільш корисно.

2.2. Опис процесу взаємодії та обміну даними в системі

2.2.1. Polling

Наївний спосіб підходу до обміну даними полягає в тому, щоб клієнти, які беруть участь, періодично опитували зміни. Щоб створити враження

реального часу, нам довелося б опитувати часто - принаймні один раз на секунду. Це відносно легко реалізувати, але досить неефективно: більшу частину часу немає змін для надсилання опитувальному клієнту ("push"), отже більша частина трафіку є накладними витратами.

Ситуація ускладнюється ще більше затримками трафіку: надто часте опитування призведе до того, що багато пакетів потрібно буде обмінювати через маршрутизатор і, оскільки ми використовуємо бездротові технології, по повітрю. Кожен з цих пакетів не йде безпосередньо до призначеного отримувача; всі пристрої, крім маршрутизатора та отримувача, проігнорують пакети, які не призначені для них. Звичайно, всі пристрої, крім маршрутизатора та отримувача, проігнорують пакети, які не призначені для них. Однак через свою природу це рішення важко масштабувати.

Для наших цілей опитування все ще є варіантом. Однак в залежності від реалізації це може спричинити проблеми на практиці, якщо кілька сусідніх класів використовують подібне програмне забезпечення одночасно.

2.2.2. WebSockets

Для досягнення обміну даними в реальному часі можуть бути використані WebSockets. Зв'язок WebSocket бере свій початок у взаємному погодженні HTTP (HyperText Transfer Protocol). Запит містить заголовки, які сервер, який підтримує оновлення, розуміє, після цього початкового взаємного погодження зв'язок продовжується через протокол WS (WebSocket). Ось приклад такого запиту HTTP з RFC6455.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGh1IHNhbXBsZSBub25j2Q==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Як і звичайний HTTP-зв'язок, можливо (і навіть рекомендовано) забезпечити зв'язок за допомогою TLS (Transport Layer Security). Для HTTP кінцевим протоколом є HTTPS. Відповідно, для WS протоколом є WSS.

З міркувань безпеки можливо використовувати WSS з HTTP-зв'язку, тим самим оновлюючи безпеку. Навпаки, з HTTPS-зв'язку ми не можемо використовувати небезпечний WS-зв'язок.

Простий WebSocket можна відкрити в JavaScript, викликавши конструктор WebSocket над URL-адресою, наприклад:

```
var socket = new WebSocket("https://websocket.url");
```

Як і більшість конструкцій у JavaScript, ми можемо призначити функції сокету для виклику, коли подія спрацьовує - ще один спосіб асинхронного програмування.

Згідно з доступною документацією в розробницькій мережі Mozilla (MDN), об'єкт WebSocket має наступні атрибути, пов'язані з подіями:

- onopen Функція-слухач, яка повинна бути викликана, коли стан зв'язку WebSocket змінюється на OPEN; це вказує на те, що зв'язок готовий до надсилання та отримання даних.

- onmessage Функція-слухач, яка повинна бути викликана, коли повідомлення отримано від сервера.

- onclose Функція-слухач, яка повинна бути викликана, коли стан зв'язку WebSocket змінюється на CLOSED.

- onerror Функція-слухач, яка повинна бути викликана, коли стається помилка.

Наступний приклад ілюструє, як ми можемо підключитися до цих подій, реєструючи кожну з них у консолі разом з їхніми повідомленнями:

```
var socket = new WebSocket("https://websocket.url");
socket.onopen = function() {
    console.log("WebSocket відкрито.");
};
socket.onmessage = function(message) {
    console.log("Нове повідомлення отримано: " + message);
};
socket.onerror = function(message) {
    console.log("Сталася помилка: " + message);
};
socket.onclose = function() {
    console.log("WebSocket було закрито.");
};
```

2.3. Особливості використання протоколів передачі даних

Оскільки наше середовище включатиме багато рядків, нам доведеться використовувати серіалізацію для форматування повідомлень. Цей розділ має на меті коротко пояснити дві поширені техніки серіалізації, зрозумілі для людини: JSON та XML. Обидві техніки добре працюють з міжнародним текстом: вони повністю підтримують кодування Unicode.

Ми не будемо розглядати бінарні техніки серіалізації, оскільки їх використання обмежене нативними додатками.

2.3.1 JSON

JSON (JavaScript Object Notation) - це поширена техніка серіалізації, яка походить від JavaScript. Через її походження від мови скриптування, однією з головних переваг JSON є її синтаксис, зрозумілий для людини, при цьому зберігаючи накладні витрати до мінімуму. Її стандартизація RFC7159 призвела до багатьох реалізацій як у мовах з м'якою, так і з жорсткою типізацією.

Хоча JavaScript - це мова з м'якою типізацією, JSON підтримує жорстку типізацію. Типи включають цілі числа, числа з плаваючою комою, рядки (заключені в подвійні лапки), булеві значення (літерали true та false), масиви

та об'єкти. Особливою перевагою цього є те, що документи JSON не вимагають складної структури даних для їхнього представлення в мові програмування. Це на відміну від структур DOM, які використовують похідні SGML (наприклад, XML, HTML).

Хоча спочатку це була жорстка підмножина JavaScript, JSON була розширена з тих пір, щоб дозволити повне кодування в Unicode, що дозволяє кодувати документи JSON в UTF-8, UTF-16 або UTF-32.

У JSON масиви - це послідовність елементів, розділених комами та заключених у квадратні дужки [і], з неявним індексом, що починається з нуля.

Об'єкти, з іншого боку, є відображеннями на основі ключових слів. Двокрапки : розділяють ключові слова від їхніх значень, а коми розділяють пари ключ-значення. Об'єкти заключені в фігурні дужки { і }. Хоча JavaScript не вимагає цього, ключові слова обов'язково кодуються як рядки в JSON:

```
{"name": "Douglas Crockford", "achievement": "First specified JSON"}
```

Об'єкти та масиви можуть бути вкладені один в одного в обох напрямках. Щоб проілюструвати це, переглянемо наступний приклад для закодованого об'єкта, рядків, цілих чисел та масиву.

```
{
    "name": "University",
    "location": "N",
    "age": 91,
    "num_students": 19200,
    "studies": [
        "Biology",
        "Chemistry",
        "Computer science",
        "Information science",
        "Mathematics",
        "Physics"
    ]
}
```

2.3.2. XML

Іншою поширеною технікою серіалізації є мова розмітки Extensible Markup Language (XML), найбільш відома своїм використанням для обміну даними через Інтернет. Як і HTML, основна мова розмітки для веб, вона має корені в мові Standard Generalized Markup Language (SGML). XML була спроектована як підмножина SGML, яка використовує схожий синтаксис, але менше функцій для полегшення реалізації відповідного парсера.

XML представляє дані як дерево вузлів, кожен з яких містить або тіло тексту, або інше дерево. Кожен вузол заключений між початковим та кінцевим тегами, обидва заключені між < і >. Крім того, початкові теги можуть містити атрибути. Наприклад:

```
<province name="G">
    <city>A</city>
    <city>N</city>
</province>
```

Тут A та N розуміються як екземпляри типу city, які належать до батьківської провінції на ім'я G.

Як і JSON, XML повністю відповідає Unicode. Як і SGML, її парсер непрощаний до порушень її синтаксичних правил: документ не буде продовжуватися парситися, якщо станеться помилка. Це на відміну від HTML5, який був спроектований як поблажливий у цьому відношенні та має чітко визначені специфікації для таких крайніх випадків.

Серіалізація дерева вузлів, подібного до об'єкта, використаного в прикладі JSON, призводить до наступного розміткового коду. Зверніть увагу, що документ XML не містить явної інформації про типізацію; це робиться в окремому документі XML Schema, якщо потрібно. Такий файл XSD може визначити схему документа з 19 примітивних типів даних, включаючи цілі числа, рядки, дати та URL.

На відміну від JSON, в XML немає примітиву для масивів. Натомість структури, подібні до масивів, можуть бути представлені шляхом робити всіх них дітьми одного і того ж батьківського елемента. У наступному прикладі це зроблено шляхом додавання елементів study до вузла studies.

Нарешті, елементи без тіла можуть бути використані для представлення необов'язкових булевих виразів. На відміну від використання кінцевого тега, початковий тег закривається додатковою косою рискою: />. Вузол is_public в наступному прикладі є прикладом такого порожнього тега.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<university>
    <name> University</name>
    <location>location</location>
    <age>91</age>
    <num_students>19200</num_students>
    <studies faculty="science">
        <study>Biology</study>
        <study>Chemistry</study>
        <study>Computer science</study>
        <study>Information science</study>
        <study>Mathematics</study>
        <study>Physics</study>
    </studies>
    <is_public />
</university>
```

При роботі з документами XML зазвичай використовується модель об'єкта документа (DOM) для представлення дерева вузлів та його елементів, забезпечуючи дійсність кінцевого документа. Звичайно, документ XML може бути створений шляхом запису рядків безпосередньо в файл. Однак оскільки мова не є регулярною, для обробки документа використання парсера DOM в формі є неминучим.

2.4. Проблеми синхронізації даних при роботі навчального середовища

Оскільки ми маємо справу з інтерактивним полотном, яке включає кількох користувачів, ми очікуємо, що можуть виникнути проблеми з

синхронізацією при спілкуванні з сервером. Хоча, взаємодія користувачів з полотном обмежена перетягуванням карт з однієї позиції в іншу, що зменшує проблему синхронізації до визначення того, яка є фактичною реальністю.

Існує кілька способів вирішення цієї проблеми. Ми коротко обговоримо три з них.

2.4.1. Запобіжні засоби вирішення проблем синхронізації

Один із способів вирішення проблем з синхронізацією полягає в тому, щоб спробувати запобігти їхньому виникненню. Наприклад, для наших цілей ми могли б реалізувати ексклюзивне блокування, що блокує карту, як тільки відбувається перша взаємодія користувача, запобігаючи іншій взаємодії з нею. Блокування потім знімається, як тільки інший користувач припиняє взаємодію з нею, що представлено на рисунку 2.3.

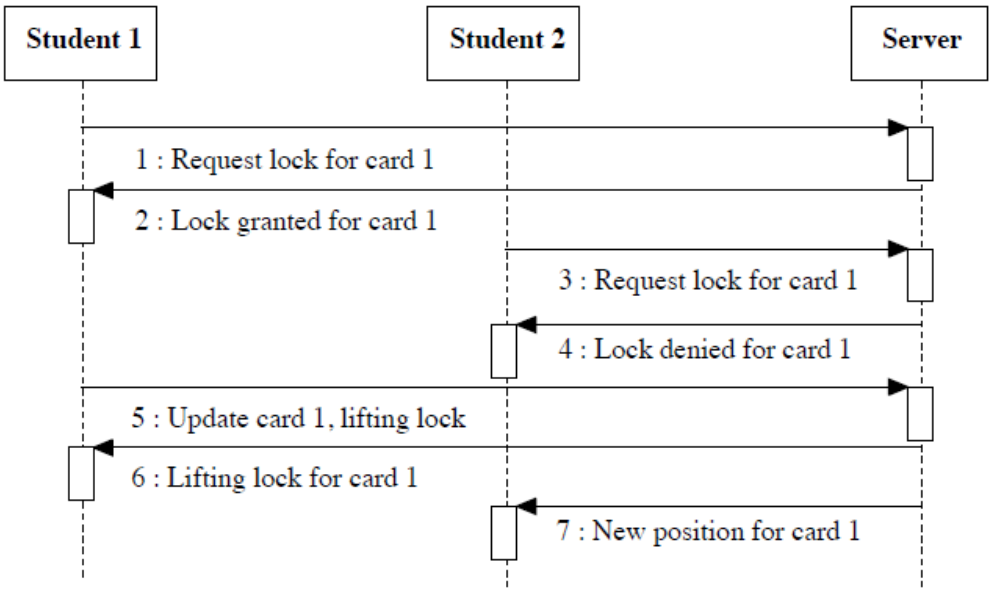

Рисунок 2.3 - Діаграма послідовності, що показує можливий протокол для двох клієнтів, які намагаються отримати ексклюзивне блокування

Хоча це вирішить симптом того, що користувач бачить свої зміни, перекритих конкурентною дією, це може спричинити іншу проблему: гонку умов. Якщо два користувачі взаємодіють з однією і тою ж картою одночасно,

хто отримає ексклюзивне блокування замок ? Технічно, один з них завжди буде першим на кілька мілісекунд, але в ідеалі ми не повинні чекати, поки сервер зреагує, перш ніж починати взаємодію.

2.4.2. Метод лінійного розбору повідомлень

Інший спосіб вирішення проблеми конкурентних повідомлень полягає в тому, щоб розглядати їх лінійно: будь-яке повідомлення, яке приходить першим, обробляється першим. Це відносно легко реалізувати: повідомлення чергуються в черзі першим прийшов, першим вийшов (FIFO). Після обробки повідомлення в горі черги результати надсилаються іншим клієнтам. Якщо інший клієнт взаємодіяв з тим самим елементом на полотні, їхній результат буде перекритий. Звичайно, це означає, що наступні повідомлення в черзі знову перекриють цей результат.

Ключовим тут є те, як користувач очікує механізму роботи. Якщо ми обробляємо повідомлення лінійно, в залежності від того, скільки конкурентних змін є, елементи можуть бути переміщені кілька разів на полотні користувача.

Ідеально, коли користувачі спочатку обговорюють, куди повинні бути розміщені карти, перш ніж вони переміщають карту. Однак ймовірно, що вони будуть переміщати карти для ілюстративних цілей: уявіть собі користувача, який пояснює контекст певної історичної події, наприклад. Отже, ймовірно, що два клієнти будуть перетягувати ту саму карту з її початкової позиції в іншу позицію на полотні.

Як тільки перший користувач відпускає карту, яку він перетягує, зміна буде передана на сервер і, отже, іншому користувачу - перекриваючи позицію їхнього локального екземпляра карти. Однак оскільки вони все ще взаємодіють з нею, подія зміни може бути поставлена в чергу, в залежності від реалізації. Потім, після того як другий користувач відпускає, зміна,

зроблена першим, можливо, ненавмисно буде перекрита, що показано на рисунку 2.4.

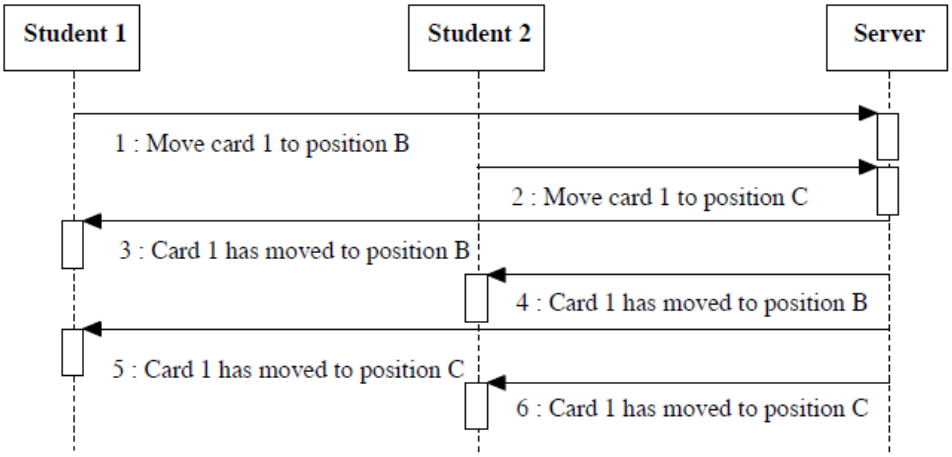

Рисунок 2.4 - Діаграма послідовності, що показує можливий протокол для двох клієнтів, які переміщують одну картку (лінійний розбір)

2.4.3. Метод призначення попередньої умови

Третім методом, який ми розглядаємо, є призначення попередньої умови кожному оновленню карти. У нашому випадку ця попередня умова буде позицією, в якій карта знаходилася, коли користувач почав взаємодіяти з нею. Ефективно це означає, що перша зміна, яка буде надіслана, вважатиметься фактичною реальністю - конкурентна зміна матиме застарілу попередню умову і, отже, буде відкинута.

Розглянемо ситуацію, яку ми розглянули в розділі 2.4.2. Якщо це відбувається при використанні попередніх умов, користувач, який першим надіслав зміну, не побачить, як карта переміщується в іншу позицію невдовзі після цього. Натомість другий користувач, який змінив карту, побачить, як його зміна ігнорується, оскільки попередня умова для його зміни більше не є дійсною. Даний метод представлено на рисунку 2.5.

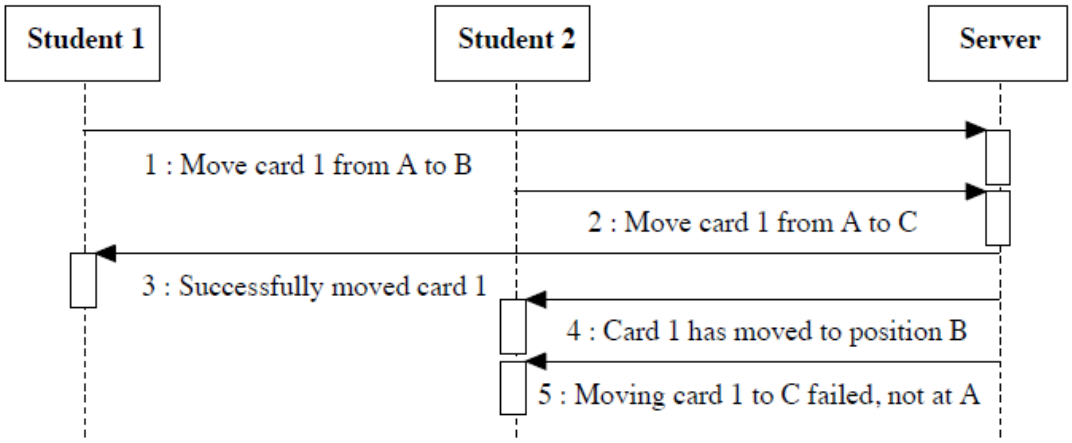

Рисунок 2.5 - Діаграма послідовності, що показує можливий протокол для двох клієнтів, які переміщують одну картку (попередня умова)

2.5. Використання концепції асинхронного програмування

Традиційно оператори в імперативній мові програмування зазвичай виконуються один за одним: тільки після того, як оператор завершив виконання, наступний оператор виконується. Чистий результат полягає в тому, що програма завжди "синхронізована" з тим, що програміст диктує. Отже, це називається синхронним програмуванням.

Розглянемо наступний синхронний приклад, який отримує набір записів з бази даних.

```
function getDataSynchronously(dbconn) {
    console.log("Запит набору елементів");
    var set = dbconn.query("SELECT * FROM items");
    while (var row = set.fetchRow()) {
        appendResult(row.title, row.value);
        console.log("Результат #" + row.id + " додано");
    }
    console.log("Кінець функції");
}
```

Його оператори виконуються послідовно, тому консольний вивід після виклику функції getDataSynchronously буде таким:

```
Запит набору елементів
Результат #1 додано
Результат #2 додано
...
Кінець функції
```

Синхронні програми відносно легко зрозуміти, оскільки їхній потік виконання досить прямолінійний: програма не продовжує, поки попередній оператор не буде повністю виконано. Недоліком цього є те, що програма ефективно блокується, якщо оператор виконується довше: немає конкуренції. Традиційно такі вузькі місця були введенням-виведенням диска та спілкуванням з іншими системами для спеціальних послуг, наприклад, сервером бази даних. Як прямий результат, такі операції могли робити інтерфейс користувача (UI) для програми нереагуючим на помітний час.

Коли програмісти стикаються з цими проблемами, вони часто використовують потоки, техніку конкуренції, яка використовується для розділення потоку виконання програми. Зазвичай обчислювально інтенсивна програма працюватиме в окремому потоці від основної програми.

Однак потоки не завжди є варіантом. Для цієї роботи ми використовуємо веб-середовище, яке зазвичай працює в однопотоковому режимі в браузері. В результаті будь-яке синхронне спілкування клієнт-сервер затримає або блокує будь-яку взаємодію, яку користувач намагається викликати на полотні. У реальних системах таке спілкування може відбуватися кілька разів на секунду, що помітно заважає користувачеві.

Але, ми можемо пом'якшити таке блокування, використовуючи техніку конкуренції, придатну для веб-браузерів: функції зворотного виклику. Ми передаємо додатковий аргумент до блокуючих функцій: вказівник на іншу функцію, яку потрібно виконати після завершення відповідної блокуючої функції.

```
function getDataAsynchronously(dbconn) {
    console.log("Запит набору елементів");
    dbconn.query("SELECT * FROM items", callback);
    console.log("Кінець функції");
}
function callback(set) {
    while (var row = set.fetchRow()) {
        appendResult(row.title, row.value);
        console.log("Результат #" + row.id + " додано");
    }
}
```

У нашому прикладі функція запиту бази даних тепер викликає функцію обробки тільки тоді, коли вона повністю отримала запитаний набір записів з бази даних. Ефективно це розкладає обробку кінцевого корисного навантаження на час, коли воно доступне. Тим часом програма продовжить свою звичайну роботу - тут, ймовірно, чекаючи на введення користувача. Це називається асинхронним програмуванням.

Оскільки спілкування більше не блокує потік виконання програми, наш консольний вивід зміниться.

```
Запит набору елементів
Кінець функції
Результат #1 додано
Результат #2 додано
...
```

Деякі мови програмування дозволяють програмісту передавати анонімну (лямбда) функцію як аргумент зворотного виклику, зазвичай призводячи до меншого коду. Недоліком цього є те, що це може заплутати потік виконання програми: на перший погляд код може здатися таким, що виконується послідовно, тоді як насправді він все ще виконується як зворотний виклик. Наступний приклад ілюструє це: хоча синтаксично він переважно нагадує приклад, що вже був розглянутий вище, оскільки він йому семантично еквівалентний.

```
function getDataAsynchronously(dbconn) {
    console.log("Запит набору елементів");
    dbconn.query("SELECT * FROM items", function (set) {
        while (var row = set.fetchRow()) {
            appendResult(row.title, row.value);
            console.log("Результат #" + row.id + " додано");
        }
    });
    console.log("Кінець функції");
}
```

2.6. Вибір платформи та протоколів передачі даних для реалізації інтерактивного середовища навчання

Цей розділ деталізує рішення, які ми прийняли для реалізації, враховуючи варіанти, які ми розглянули в попередніх розділах.

Вирішальним у нашому рішенні обрати веб-підхід є намір роботи на трьох платформах: планшети, які працюють або на iOS або на Android, а також система викладача для роботи на інтерактивній дошці ("digiboard"). Хоча остання, в принципі, є системою, яка працює під Windows, вона підпорядкована кільком обмеженням. На практиці це означає, що більшість її додатків працюють у веб-браузері. Отже, має сенс повторно використовувати цей кодовий базис для побудови клієнта для iOS та Android.

У попередньому розділі ми обговорювали деякі недоліки веб-додатків, зокрема продуктивність виконання JavaScript. Як ми згадували, ця область була предметом багатьох досліджень, і з тих пір продуктивність значно зросла.

Як згадувалося в попередньому розділі, TCP має значні переваги. По-перше, оскільки це протокол на основі зв'язку, кожен зв'язок має стан. Це означає, що нам доведеться аутентифікувати канал лише один раз, на початку зв'язку. Припускаючи, що ми забезпечимо зв'язок за допомогою TLS, це означає, що у нас не лише буде безпечний зв'язок, але ми також запобіжимо певним накладним витратам, які супроводжували б надання

аутентифікованого зв'язку на статистичному протоколі (наприклад, токени безпеки на UDP).

Крім того, гарантії порядку та цілісності, які супроводжують TCP, будуть надзвичайно важливими для нашого методу надання спілкування в реальному часі. Як ми побачимо пізніше в цьому розділі, такі техніки, як WebSockets, покладаються на роботу з одним зв'язком на клієнт, що ще більше підкреслює нашу потребу в становому протоколі.

У рішенні між Bluetooth та WiFi ми спочатку схилялися до першого, оскільки він зазвичай споживає менше енергії. Однак оскільки стек Bluetooth дозволяє спілкування лише між одним майстер-пристроєм та одним дочірнім пристроєм, це не є дійсно хорошим засобом комунікації в реальному часі між більш ніж одним пристроєм. Навіть якщо ми враховуємо лише два пристрої, їм все одно доведеться чергувати роль майстра (відправника) та слухача. Це, звичайно, призводить до багатьох накладних витрат, оскільки стек не призначений для цієї мети.

Це ускладнюється ще більше потребою викладача в тому, щоб його комп'ютер надсилав нові карти на дочірні пристрої після кожного раунду, що вимагає від нього бути в режимі майстра. Крім того, стек Bluetooth не дозволяє пристрій бути в парі з більш ніж одним майстром одночасно. Ми могли б, ймовірно, обійти це шляхом реалізації якоїсь форми часового поділу, але в цілому рішення на основі WiFi виглядає кращим і набагато більш масштабованішим.

Для полегшення процесу співпраці важливо, щоб клієнти бачили зміни один одного, як тільки вони відбуваються: ми хочемо, щоб полотна оновлювалися в реальному часі. Отже, важливо, щоб клієнти постійно обмінювалися інформацією. Хоча більшість груп складаються з двох осіб, у кожній сесії групи бере участь принаймні три клієнти: два студенти та один викладач, як це показано на рис. 1.3.

Отже, ми спроектуємо додаток таким чином, щоб він спілкувався своїми базовими сторінками через HTTPS, а після входу в інтерактивний компонент ми ініціюємо зв'язок WebSocket для обміну даними в реальному часі.

Обидві техніки, JSON та XML, є відмінними для серіалізації даних і повністю підтримуються в усіх сучасних браузерах. Коли ми працюємо з JavaScript, JSON має значну перевагу в тому, що він не вимагає моделі об'єкта документа (DOM). Це означає, що він має менші накладні витрати: об'єкти можуть бути доступні безпосередньо, а не через виклики функцій на DOM. Оскільки JavaScript - це (зазвичай) інтерпретована мова, це дає певні переваги в швидкодії порівняно з XML.

Оскільки ми розробляємо додаток, у якого інтерфейс користувача буде працювати з JavaScript, то було вирішено реалізувати протокол за допомогою серіалізації JSON.

Планується, що протокол буде слідувати шаблону передачі повідомлень. Це означає, що клієнт може запитати сервер, наприклад, знайти список активних сесій, залишаючи серверу виконувати фактичний запит. Подібним чином сервер може надіслати набір нових карт клієнту, залишаючи процес розміщення на стороні клієнта. Найбільш помітно, однак, це просте передавання дельт між сервером та клієнтами: переміщення карт з позиції A в позицію B. Знову ж передаються лише позиція, тобто контейнер та координати, а фактичне переміщення на стороні клієнта обчислюється на стороні клієнта.

Оскільки більше одного клієнта може одночасно редагувати один і той сам елемент на полотні, можуть виникнути проблеми з конкуренцією при спілкуванні з сервером. Ми могли б стверджувати, що, враховуючи обмежену кількість конкурентних клієнтів, які працюють над одним і тим самим полотном, найгірше, що може статися, - це гонка умов - невеликий незручний момент для одного з користувачів. Однак без протоколу ці гонки

можуть призвести до такого роду несподіваної поведінки, яка викликає плутанину.

Ми очікуємо, що обидва об’єкти навчання можуть перетягувати одну і ту саму карту, навіть для ілюстративних цілей, але не повинні випадково скасовувати один одного.

У попередньому розділі ми описували різні методи вирішення потенційних проблем з конкуренцією. У випадку коли об’єкти навчання співпрацюють, ми вважаємо, що призначення попередніх умов повідомленням має найбажаніші результати. Застосування цього методу відхилить наступні повідомлення, якщо їхні початкові умови більше не виконуються - наприклад, якщо інший студент вже перемістив ту саму карту.

Висновки до розділу

В даному розділі було проаналізовано основні протоколи передачі даних, зокрема HTTP, Polling та WebSockets, що дозволило оцінити їхню ефективність у контексті інтерактивного навчання. Показано, що WebSockets забезпечують двосторонню асинхронну комунікацію з низькою затримкою, що є критично важливим для динамічної взаємодії в реальному часі.

Дослідження форматів обміну даними, таких як JSON та XML, виявило переваги JSON завдяки його легкості, гнучкості та широкій підтримці у сучасних мовах програмування та фреймворках.

Особливу увагу приділено проблемам синхронізації даних, які можуть виникати у багатокористувацьких середовищах. Розглянуто методи їх вирішення, зокрема лінійний розбір повідомлень та метод попередніх умов, що сприяють підвищенню надійності системи.

Також було обґрунтовано доцільність використання асинхронного програмування як ключового підходу для реалізації високопродуктивної взаємодії з користувачем без блокування основного потоку виконання.

Узагальнюючи, вибір платформи, протоколів і методів реалізації повинен базуватися на вимогах до швидкодії, масштабованості, надійності синхронізації та зручності розробки. Запропонований підхід забезпечує основу для побудови ефективного, інтерактивного та масштабованого навчального середовища.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО НАВЧАЛЬНОГО СЕРЕДОВИЩА

3.1. Ідентифікація сутностей

Перш ніж почати роботу над реалізацією, ми повинні спочатку визначити, з якими сутностями ми будемо мати справу, а тому потрібно створити модель сутностей та їхніх зв'язків.

Ми визначаємо наступні сутності в Active Historical Thinking – підході до вивчення історії, який передбачає активну взаємодію з історичними джерелами, подіями та концепціями, а не пасивне запам’ятовування фактів. Графічно сутності подано на рисунку 3.1.

Рисунок 3.1 – Представлення сутностей розроблюваної інтерактивної навчальної системи

1. Користувачі - це викладачі та студенти, які використовують систему. Вони мають унікальну адресу електронної пошти для ідентифікації, а також ім'я, за яким їх можуть ідентифікувати.

2. Модулі - це загальні сутності для окремих занять: крім назви та опису, вони мають зв'язок з сутностями сесій, міток, типів, карт та екземплярів карт.

3. Сесії модулів - це інстанціації модулів, які зв'язують модуль з групою користувачів (студентів).

4. Групи - це сутності, з якими студенти зв'язані в сесії модуля. Для ідентифікації кожна з груп має автоматично згенеровану назву, яка використовує імена студентів, наприклад "Андрій і Марія".

5. Членство в групі - це зв'язок сутності між користувачем та групою.

6. Карти - це елементи полотна: прямокутні об'єкти з підписом або зображенням, різного розміру. Карти належать до певного раунду; ті, що належать до раунду 0, розміщуються викладачем, тоді як інші є змінними студентами.

7. Екземпляри - це інстанціації карт, які зв'язують карту з певною групою. Отже, вони також визначають позицію карти на полотні або в стопці для певної групи.

8. Мітки - це підписи, розміщені викладачем над полотном для вказівки року або періоду часу. Приклади: "1789", "Раннє Середньовіччя", "Богдан Хмельницький" тощо.

Типи визначають види карт. Вони визначаються модулем і визначають, наприклад, розмір та колір карт.

Поведінка типу визначає, що можна робити з певним типом карти. В залежності від раунду тип карти може мати різну поведінку, наприклад, карта може бути перетягувана в раунді 2, але раунд 3 може розширити діапазон карти до області мітки.

3.2. Розробка моделі бази даних

Для зберігання всіх даних, які використовуються в системі, ми вирішили використовувати реляційну базу даних. Кожна з сутностей, описаних у розділі 3.1 буде відображена як таблиця, як показано на рисунку 3.2.

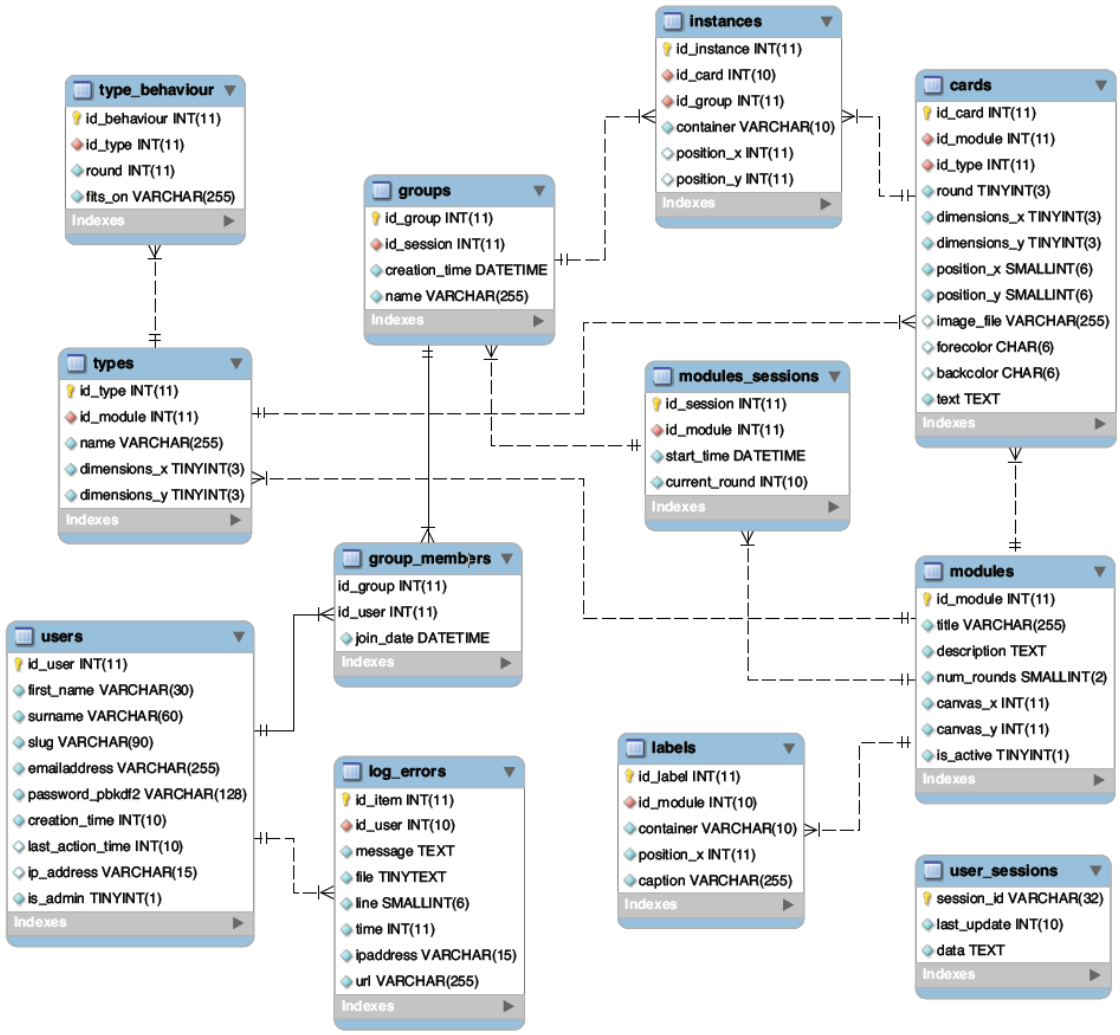

Рисунок 3.2 – Модель бази даних

Крім згаданих сутностей, ми включаємо дві додаткові таблиці: одна використовується для реєстрації помилок (log_errors), а інша для зберігання клієнт-серверних сесій, зберігання, наприклад, аутентифікації (user_sessions).

На рисунку 3.2 представлена модель бази даних інтерактивного навчального середовища. Це реляційна модель, що складається з кількох таблиць, пов'язаних між собою за допомогою зовнішніх ключів. Опишемо кожну таблицю та її зв'язки:

1. `type_behaviour`:

- `id_behaviour` (INT, первинний ключ)

- `id_type` (INT, зовнішній ключ, посилається на `types`)

- `its_on` (VARCHAR(255))

- Індекси: `id_type`

Зв'язок: Один `type` може мати багато записів у `type_behaviour`.

2. `types`:

- `id_type` (INT, первинний ключ)

- `id_module` (INT, зовнішній ключ, посилається на `modules`)

- `name` (VARCHAR(255))

- `dimensions_x` (TINYINT)

- `dimensions_y` (TINYINT)

- Індекси: `id_module`

Зв'язок: Один `module` може мати багато `types`.

3. `users`:

- `id_user` (INT, первинний ключ)

- `id_type` (INT, зовнішній ключ, посилається на таблицю ролей або типів користувачів)

- `first_name` (VARCHAR(30))

- `surname` (VARCHAR(60))

- `slug` (VARCHAR(80))

- `emailaddress` (VARCHAR(255), унікальний)

- `password_pbkdf2` (VARCHAR(128))

- `creation_time` (INT(10))

- `last_action_time` (INT(10))

- `ip_address` (VARCHAR(15))

- `is_admin` (TINYINT(1))

- Індекси: `emailaddress`

Зв'язок: Один `type` користувача (наприклад, "студент", "викладач") може мати багато записів у `users`.

4. `groups`:

- `id_group` (INT, первинний ключ)

- `id_session` (INT(11), зовнішній ключ, посилається на `modules_sessions`)

- `creation_time` (DATETIME)

- `name` (VARCHAR(255))

- Індекси: `id_session`

Зв'язок: Одна `modules_sessions` може мати багато `groups`.

5. `group_members`:

- `id_group` (INT(11), зовнішній ключ, посилається на `groups`, частина первинного ключа)

- `id_user` (INT(11), зовнішній ключ, посилається на `users`, частина первинного ключа)

- `join_date` (DATETIME)

- Первинний ключ: (`id_group`, `id_user`)

- Індекси: `id_user`

Зв'язок: Багато користувачів можуть бути членами багатьох груп (багато-до-багатьох через цю зв'язкову таблицю).

6. `modules_sessions`:

- `id_session` (INT(11), первинний ключ)

- `id_module` (INT(11), зовнішній ключ, посилається на `modules`)

- `start_time` (DATETIME)

- `current_round` (INT(10))

- Індекси: `id_module`

Зв'язок: Один `module` може мати багато `modules_sessions` (наприклад, різні запуски одного модуля).

7. `modules`:

- `id_module` (INT(11), первинний ключ)
- `title` (VARCHAR(255))
- `description` (TEXT)
- `num_rounds` (SMALLINT(2))
- `canvas_x` (INT(11))
- `canvas_y` (INT(11))
- `is_active` (TINYINT(1))
- Індекси: `title`

Зв'язок: Один `module` може мати багато `types` та `modules_sessions`.

8. `instances`:

- `id_instance` (INT, первинний ключ)
- `id_card` (INT(10), зовнішній ключ, посилається на `cards`)
- `id_group` (INT(11), зовнішній ключ, посилається на `groups`)
- `container` (VARCHAR(10))
- `position_x` (INT(11))
- `position_y` (INT(11))
- Індекси: `id_card`, `id_group`

Зв'язок: Одна `card` може мати багато `instances` у різних `groups`. Одна `group` може містити багато `instances` різних `cards`.

9. `cards`:

- `id_card` (INT(11), первинний ключ)
- `id_module` (INT(11), зовнішній ключ, посилається на `modules`)
- `id_type` (INT(11), зовнішній ключ, посилається на `types`)
- `round` (TINYINT(3))
- `dimensions_x` (TINYINT(3))
- `dimensions_y` (TINYINT(3))

- `position_x` (SMALLINT(6))

- `position_y` (SMALLINT(6))

- `image_file` (VARCHAR(255))

- `forecolor` (CHAR(6))

- `backcolor` (CHAR(6))

- `text` (TEXT)

- Індекси: `id_module`, `id_type`

Зв'язок: Один `module` може мати багато `cards`. Один `type` може мати багато `cards` у межах одного `module`.

10. `labels`:

- `id_label` (INT(11), первинний ключ)

- `id_module` (INT(11), зовнішній ключ, посилається на `modules`)

- `container` (VARCHAR(10))

- `position_x` (INT(11))

- `position_y` (INT(11))

- `caption` (VARCHAR(255))

- Індекси: `id_module`

Зв'язок: Один `module` може мати багато `labels`.

11. `log_errors`:

- `id_item` (INT(11), первинний ключ)

- `id_user` (INT(10), зовнішній ключ, посилається на `users`)

- `message` (TEXT)

- `file` (TINYTEXT)

- `line` (SMALLINT(6))

- `time` (INT(11))

- `ipaddress` (VARCHAR(15))

- `url` (VARCHAR(255))

- Індекси: `id_user`

Зв'язок: Один `user` може мати багато записів у `log_errors`. Ця таблиця використовується для зберігання інформації про помилки в системі.

12. `user_sessions`:

- `session_id` (VARCHAR(32), первинний ключ)

- `id_user` (INT(11), зовнішній ключ, посилається на `users`)

- `last_update` (INT(10))

- `data` (TEXT)

- Індекси: `id_user`

Зв'язок: Один `user` може мати багато записів у `user_sessions` (наприклад, для відстеження активних сесій).

Отже ця модель бази даних призначена для підтримки інтерактивного навчального середовища. Вона охоплює:

- Керування користувачами: Реєстрація, автентифікація, розмежування прав (адміністратор/звичайний користувач).

- Керування навчальними модулями: Зберігання інформації про навчальні курси, їх опис, кількість раундів тощо.

- Організація навчальних сесій: Відстеження запусків навчальних модулів.

- Групова робота: Організація студентів у групи для спільної діяльності.

- Інтерактивні елементи (`cards`, `labels`, `types`, `type_behaviour`, `instances`): Зберігання інформації про різні інтерактивні об'єкти, їх типи, поведінку, розташування та належність до модулів і груп.

- Збереження стану користувача (`user_sessions`): Відстеження сесій користувачів та збереження їхніх даних.

- Логування помилок (`log_errors`): Запис інформації про помилки для діагностики та налагодження.

Зв'язки між таблицями забезпечують цілісність даних та дозволяють ефективно отримувати інформацію про навчальний процес, користувачів та

інтерактивні елементи. Наявність таблиць `types` та `type_behaviour` свідчить про гнучку систему для визначення різних типів інтерактивних елементів та їхньої поведінки. Таблиця `instances` дозволяє використовувати одні й ті самі `cards` у різних групах та з різним розташуванням.

Ми вирішили реалізувати базу даних у MariaDB, системі баз даних з відкритим вихідним кодом на основі MySQL. Щоб зробити можливим повторне використання існуючих серверів баз даних, ми використовуватимемо шар абстракції бази даних, який також сприяє використанню інших популярних систем реляційних баз даних з відкритим вихідним кодом, наприклад PostgreSQL та SQLite.

3.3. Проектування серверної частини і протоколів взаємодії

Для полегшення взаємодії між клієнтами ми використовуватимемо централізовану систему, на відміну від розподіленої (децентралізованої). Хоча обидві мають свої переваги, централізований підхід має більше сенсу для наших цілей. Оскільки весь зв'язок проходитиме через центральний сервер, це робить зв'язування пристроїв з групами легшим та більш стійким. Крім того, викладач зможе легко агрегувати дані без перешкод для окремих пристроїв.

Використовуючи базу даних як бекенд, сервер стежить за станами полотен усіх груп, які беруть участь. Отже, якщо група сформована, сервер може швидко надіслати весь стан (включаючи карти, мітки, тощо) кожному з користувачів, які беруть участь. Подібним чином, якщо пристрій випадково від'єднається, вони можуть продовжити з того місця, де зупинилися, після взаємного погодження з сервером.

Ці стани можуть бути змінені користувачами: студенти можуть змінювати екземпляри карт для своїх власних груп, а викладачі можуть робити більше карт доступними. Як ми згадували в попередніх розділах, це

буде зроблено шляхом повідомлень: простих об'єктів, закодованих у JSON, які деталізують зміни, які були зроблені. Всі ці повідомлення пройдуть через сервер. Після того, як сервер застосує повідомлення до стану, як потрібно, вони надішлють результати всім зацікавленим сторонам.

Наприклад, якщо користувач A змінює позицію карти з позиції P на позицію Q, їхній клієнт надішле повідомлення, яке деталізує цю зміну, серверу. Припускаючи, що карта ще не була переміщена з її позиції P, сервер потім застосовує цю зміну і розповсюджує зміну пристрою для користувачів A та B. Це має додаткову перевагу - повідомлення A про те, що їхня зміна була успішною.

Щоб ефективно використовувати шаблон передачі повідомлень, нам потрібно визначити протокол для всіх клієнт-серверних комунікацій. Оскільки повідомлення, які передаються, обмежені в обсязі, ми можемо обмежитися складанням вичерпного списку всіх можливих повідомлень, залишивши нам визначити, як вони форматуються.

Щодо формату, кожне повідомлення буде об'єктом, серіалізованим у JSON, обмінюваним через стек TCP/IP. Запит повідомлень передають параметр "call", щоб ідентифікувати, як інші атрибути об'єкта повідомлення повинні оброблятися. Відповіді повідомлень не містять такого параметра; сервер обробляє повідомлення синхронно, тому відповіді гарантовано будуть в тому ж порядку, в якому запити надсилаються.

Нижче наведено список усіх запитів повідомлень, що обмінюються через протокол. Для кожного виклику прикладено приклад запиту (=>) та відповіді (<=) повідомлення. Для покращення читаності повідомлення тут відформатовані. В реальному додатку це не так.

authenticate {cookie}

Аутентифікує зв'язок сокета на основі session-id у cookie користувача.

```
// Клієнт до сервера
=> {"call":"authenticate", "cookie":"PHPSESSID=1234...ABCD"}
```

```
// Сервер підтверджує клієнту (empty = success)
<= {}
```

announcement {msg}

Сервер розповсюджує оголошення для відображення користувачеві клієнтами.

```
// Клієнт надсилає повідомлення про оголошення серверу
=> {"call":"announcement", "msg":"Hello world!"}
// Сервер підтверджує клієнту, якщо він викладач
<= {}
// Сервер розповсюджує оголошення всім іншим клієнтам
<= {"call":"announcement", "msg":"Hello world!"}
```

error {msg}

Сервер повідомляє про помилку у відповідь на дію, ініційовану клієнтом. Це повідомлення про помилку надсилається лише клієнту, який його стосується.

```
// Клієнт до сервера
=> {"call":"authenticate", "cookie":"PHPSESSID=1234...ABCD"}
// Якщо сесія клієнта не аутентифікована, повідомлення про помилку надсилається назад
<= {"call":"error", "msg":"Session is not authenticated."}
```

list_sessions {}

Сервер повертає всі відкриті сесії.

```
// Клієнт надсилає запит серверу
=> {"call":"list_sessions"}
// Сервер повертає список сесій
<= {"sessions": {"1": {"title": "Session title goes here", "description": "The description for the module goes here."}}}
```

reload_sessions {}

Змушує сервер перезавантажити свою сесію з бази даних, від'єднуючи всіх клієнтів. Може ініціюватися лише викладачем.

```
// Клієнт надсилає команду перезавантаження серверу
=> {"call":"reload_sessions"}
// Сервер підтверджує клієнту, якщо він викладач
<=
```

register_session {id}

Реєструє зв'язок клієнта з певною сесією.

```
// Клієнт висловлює намір приєднатися до сесії 1
=> {"call":"register_session","id":"1"}
// Сервер підтверджує
<=
```

unregister_session {}

Відреєстровує зв'язок клієнта з сесії.

```
// Клієнт висловлює намір залишити сесію 1
=> {"call":"unregister_session","id":"1"}
// Сервер підтверджує
<=
```

list_groups {}

Повертає всі доступні групи для поточної сесії як масив

```
// Користувач Аліса викликає 'list_groups'
=> {"call":"list_groups"}
// Сервер відповідає списком груп, відзначаючи, що вона є членом групи з id=1
<= {"groups": {"0": {"id": 1, "name": "Alice and Bob", "is_member": true}, "1": {"id": 2, "name": "Charlie and David", "is_member": false}}}
```

create_group

Створює нову групу для поточної сесії. Поточний зв'язок автоматично приєднується до групи і надає їй ім'я свого користувача. Цей виклик зазвичай слідує за запитом get_canvas.

```
// Користувач Аліса викликає "create_group"
=> {"call":"create_group"}
// Сервер створює нову групу для Аліси та додає її до неї
<= {}
```

join_group {id}

Поточний зв'язок приєднується до запитаної групи. Якщо ім'я користувача не присутнє в назві групи, воно додається до неї. Цей виклик зазвичай слідує за запитом get_canvas.

```
// Користувач Боб приєднується до групи, якою він ділиться з Алісою
=> {"call":"join_group", "id":"1"}
// Сервер додає Боба до поточної сесії
<= {}
```

get_canvas

Отримує серіалізовану версію даних полотна для поточної групи, до якої приєднано. Це включає не лише карти, розміщені на полотні, але й мітки часової шкали та карти в стопці.

```
// Клієнт запитує повний список поточної сесії
=> {"call":"get_canvas"}
// Сервер відповідає повною серіалізацією сесії
<= {
    "moduletitle": "Example module's title",
    "dimensions": {"width": "106", "height": "55"},
    "labels": {
        "year": {
            "0":   {"id_label":   "1",   "id_module":   "1",
"container": "year", "position_x": "15", "caption": "1800"},
            // [...]
```

```
            },
            "context": {
                "0":   {"id_label":   "4",   "id_module":   "1",
"container": "context", "position_x": "15", "caption": "French
occupation"},
                // [...]
            }
        },
        "cards": {
            "0": {
                "id_card": "10", "id_module": "1", "id_type":
"2", "round": "0", "dimensions_x": "12", "dimensions_y": "2",
"position_x":   "4",   "position_y":   "2",   "image_file":   "",
"forecolor": "000000", "backcolor": "ffffff", "text": "Willem
I<br>First king of The Netherlands"
            },
            // [...]
        },
        "instances": {
            "0": {
                "id_instance": "1", "id_card": "5", "id_module":
"1",  "id_type":  "1",  "round":  "1",  "container":  "stack",
"dimensions_x": "12", "dimensions_y": "2", "image_file": "",
"forecolor": "000000", "backcolor": "ffffff", "text": "First
constitution"
            },
            // [...]
        }
    }
```

update_instance {id, old_position, new_position}

Оновлює стан для певного екземпляра карти в поточній групі. Обидві позиції, стара і нова, надсилаються: у разі конфліктів стара позиція діє як попередня умова.

```
// Клієнт переміщує карту зі стопки на сітку...
=> {"call":"update_instance", "old_position":
{"container":"stack", "position_x":"0", "position_y":"0"},
"new_position": {"container":"grid", "position_x":"58",
"position_y":"28"}}
// Сервер підтверджує зміну ...
<= {}
// ... і передає її іншим клієнтам в тій самій групі
<= {"call":"update_instance", "old_position":
{"container":"stack", "position_x":"0", "position_y":"0"},
"new_position": {"container":"grid", "position_x":"58",
"position_y":"28"}}
```

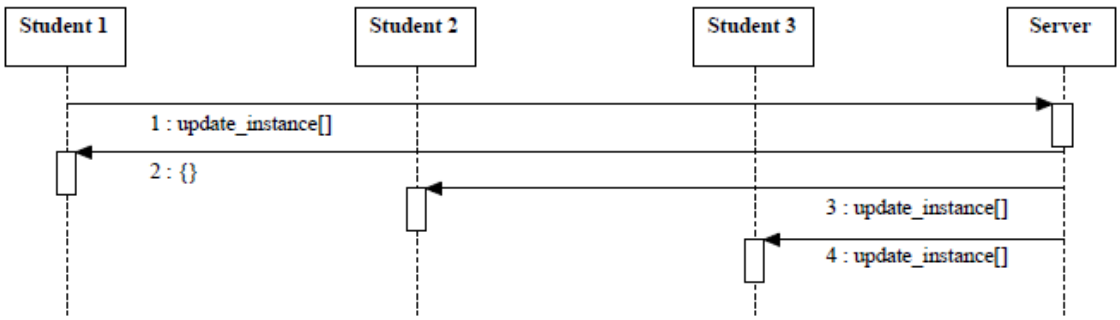

Рисунок 3.3 – Схематичне зображення передачі повідомлень для виклику update_instance

advance_round {new_round}

Переходить до наступного раунду. Параметр new_round передається для запобігання випадкового переходу на кілька раундів через взаємодію користувача.

```
// Викладач вказує серверу перейти до другого раунду
=> {"call":"advance_round","new_round":"2"}
// Сервер надсилає кожному клієнту список екземплярів карт,
які потрібно додати до стопки
// Зверніть увагу, що кожна група має свій власний набір
екземплярів карт
```

```
<= {
    "instances": {
        "0": {
            "id_instance": "25", "id_card": "14",
"id_module": "1", "id_type": "1", "round": "2", "container":
"stack", "dimensions_x": "12",
        }
    }
}
```


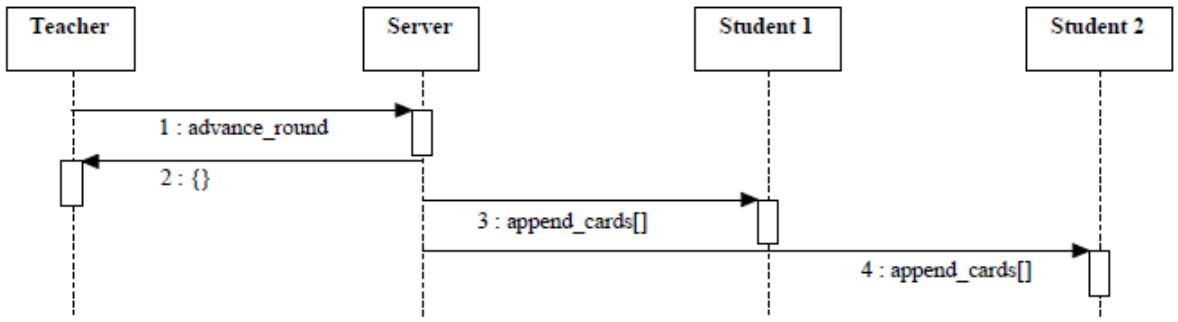

Рисунок 3.4 – Схематичне зображення передачі повідомлень для виклику advance_round call

Прототип системи реалізовано за допомогою PHP на стороні сервера та JavaScript на стороні клієнта. У відповідності з вимогами ми обрали мову з відкритим вихідним кодом для серверного програмного забезпечення. PHP є вільно розповсюджуваним і доступним на всіх платформах, тому серверне програмне забезпечення може бути розгорнуто в школах без проблем. Оскільки PHP є інтерпретованою мовою, програмне забезпечення не вимагає жодного платформо-специфічного перекомпілювання.

Вибір JavaScript на стороні клієнта є внутрішнім для розробки додатку. Хоча PHP-бекенд генерує деякий HTML, більшість інтерфейсу генерується за допомогою JavaScript та маніпуляції DOM.

3.4. Реалізація абстрактного рівня WebSocket

Як було описано раніше, протокол WebSocket сам по собі є досить грубим. Для наших цілей ми б хотіли абстрагуватися від базових подій WebSocket, додавши шар абстракції поверх них. Ми зробимо це, реалізувавши наші виклики передачі повідомлень поверх події onmessage, використовуючи JSON-закодований протокол, який ми описали раніше.

Згідно з протоколом, кожне вхідне повідомлення буде об'єктом, серіалізованим у JSON, з атрибутом "call". Ці повідомлення будуть розпарсерені нашим шаром абстракції, делегуючи об'єкт повідомлення відповідній функції на основі атрибута "call". Централізуючи цю делегацію, ми спрощуємо виклики до наших окремих функцій виклику: замість того, щоб усі функції "слухали" усі повідомлення, лише відповідні повідомлення будуть делеговані від центрального диспетчера.

WebSockets походять від JavaScript, тому нам не потрібен додатковий фреймворк для роботи з ними. Щоб проілюструвати реалізацію нашого шару абстракції на стороні клієнта, нижче наведені заголовки функцій:

```
function Socket(address, open, debug)
Socket.prototype.send = function (event, params, callback)
Socket.prototype.on = function (event, listener)
Socket.prototype.unbind = function (listener)
Socket.prototype.unbindAll = function ()
Socket.prototype.close = function ()
```

Після створення об'єкта Socket інші функції можуть додати слухача події до нього, викликавши метод on. Зазвичай такий виклик складається з двох аргументів: очікуваного виклику та функції, в яку потрібно передати повідомлення. Якщо не додано слухача для певного повідомлення, повідомлення буде відкинуто при його надходженні.

Щоб проілюструвати це, наступний фрагмент буде слухати два повідомлення про події: перший рядок викликає додавання карт після їх отримання, а другий рядок викликає оновлення карт при надходженні повідомлення про дельту:

```
socket.on('append_cards', this.drawer.push_cards);
socket.on('update_card', this.canvas.update_card);
```

Подібним чином надсилання повідомлення можна зробити через метод send, який використовує три аргументи: виклик, його параметри та необов'язкову функцію зворотного виклику для виконання після отримання відповіді. Наприклад, це може бути використано для відображення списку активних сесій користувачеві після запиту списку.

Внутрішньо надсилання повідомлення серіалізує перші два аргументи (як виклик, так і його параметри) в об'єкт повідомлення, серіалізований за допомогою JSON. Ми також додамо зворотний виклик до черги "очікувачів" - якщо програміст не надав його, буде передано порожній.

Переважно, кожне повідомлення, яке клієнт надсилає, буде підтверджено сервером. Це може бути просто підтвердженням або списком об'єктів, запитаних попереднім повідомленням - наприклад, списком доступних карт на полотні.

Щоб досягти взаємодії WebSocket у нашому PHP-серверному бекенді, ми вибрали фреймворк Ratchet (http://socketo.me/). Його модульний підхід означає, що ми можемо легко побудувати власний сервер на основі основ, закладених Ratchet. Це означає, що наші серверні класи повинні мати справу лише з фактичними вхідними та вихідними JSON-навантаженнями, тоді як фактичні зв'язки з клієнтами обробляються та абстраговані Ratchet.

Оскільки сервер зберігає стан усіх полотен через базу даних, ми оснастили його модельними класами, які безпосередньо відображають таблиці в базі даних.

3.5. Проектування дизайну інтерфейсу користувача

Інтерфейс користувача спроектований так, щоб бути аналогічним оригінальній навчальній методології. Розмітка полотна використовує сітку на основі квадратів для розміщення карт, але початковий набір карт (раунд 0) розміщується вручну викладачем при створенні модуля, щоб забезпечити їх розміщення в відповідному місці.

Центральним у нашому дизайні інтерфейсу користувача є полотно, на якому студенти обмінюються своїми ідеями: історична часова лінія.

Ми намагалися як можна точніше відтворити оригінальні елементи полотна, водночас використовуючи переваги інтерактивності:

- Карти тепер повністю відображаються в кольорі;
- Карти автоматично вирівнюються до сітки при розміщенні;
- Полотно можна збільшувати та зменшувати;
- Зміни коротко підсвічуються під час їхнього відбування за допомогою ефекту пульсування.

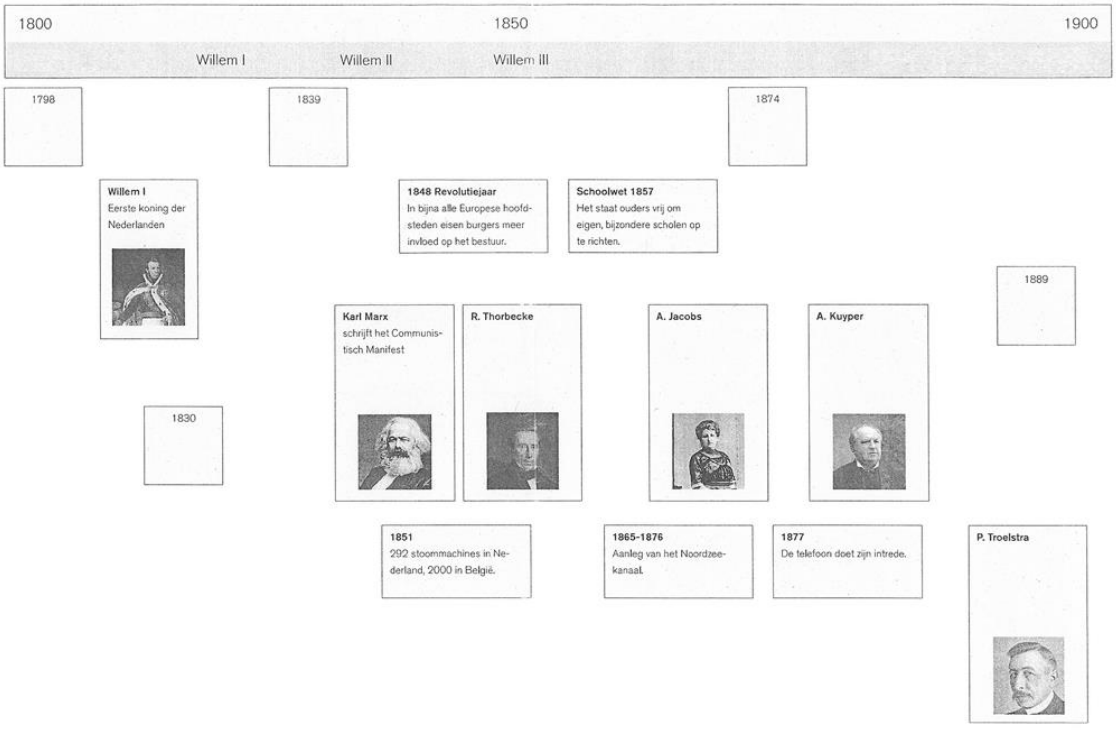

Рисунок 3.5 - Оригінальне паперове середовище для застосування методики активного історичного мислення

Порівняємо рисунки 3.5 і 3.6 на яких показано оригінальну (паперову) методику активного мислення та інтерактивне навчальне середовище. Ми дотримувалися конвенцій інтерфейсу користувача iOS, оскільки використано пілотні класи iPad. Для пристроїв Android може бути прийнято інший стиль аркуша для надання додатку вигляду нативного.

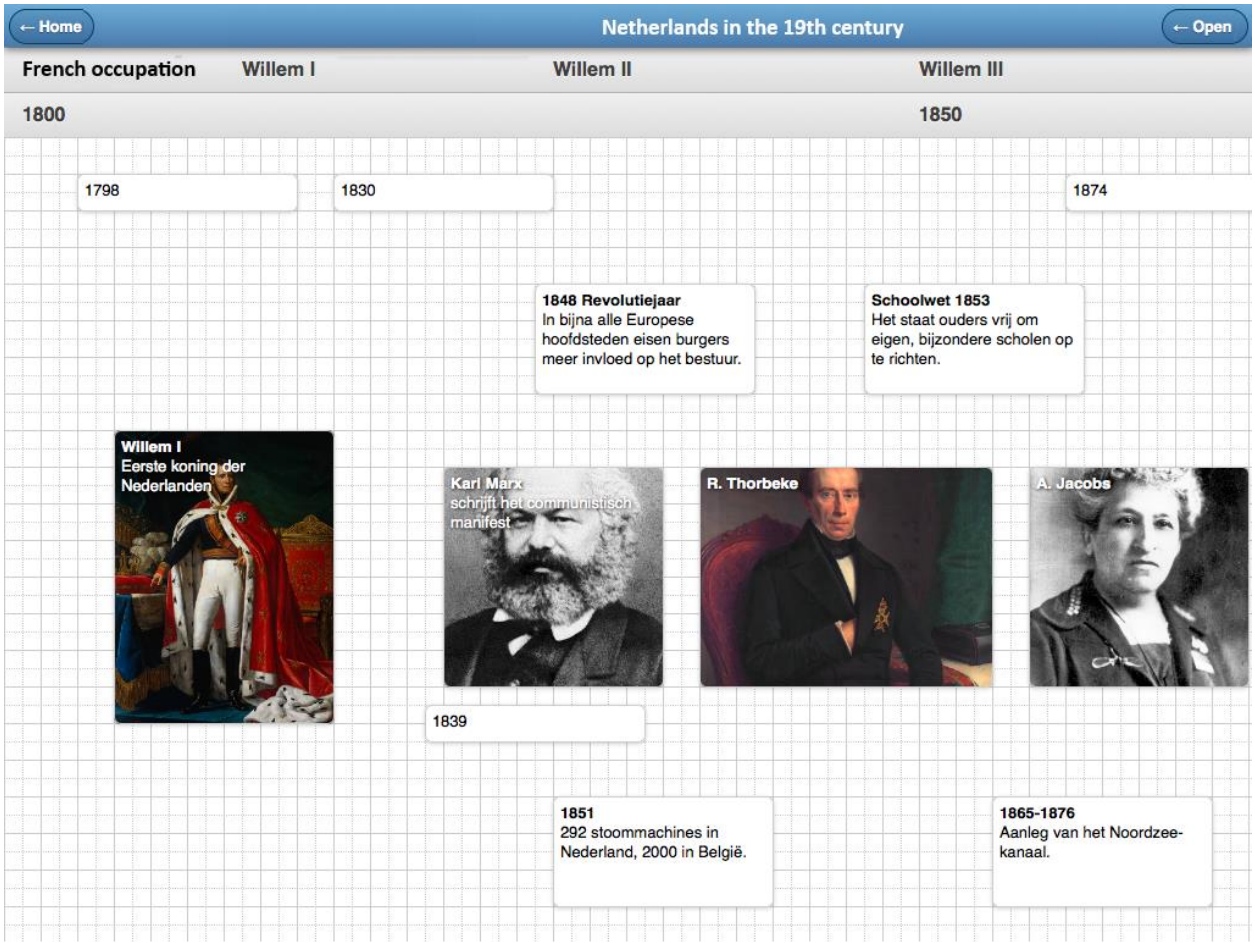

Рисунок 3.6 – Реалізоване інтерактивне середовище для співпраці згідно методики активного історичного мислення

Як ми обговорювали, система агрегації дозволяє викладачу отримати огляд знань у групі. Ми досягаємо цього, дозволяючи викладачу вибирати відповідні карти з панелі справа на екрані. Після вибору кожна з карт отримує унікальний і відчутно відрізняючий колір. Для кожної з карт, що цікавлять, сервер потім кластеризує позиції, в яких студенти розмістили карти.

У цих позиціях кожна з карт відображається як кольорова крапка. Карти однієї і тієї ж сторони в близькому оточенні групуються разом, показуючи їхню кардинальність як число зверху крапки. Рисунок 3.7 показує цей процес для чотирьох карт.

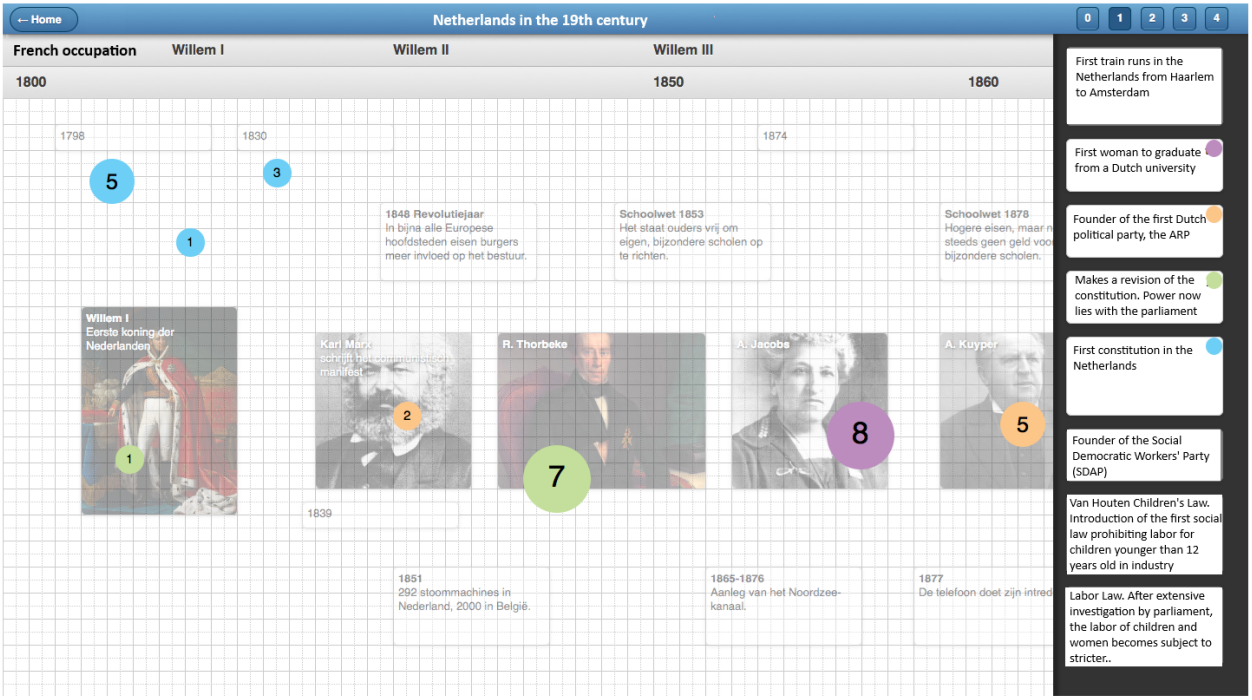

Рисунок 3.7 - Система агрегації для вчителя. Обрані картки відображатимуться зі своїми позиціями на полотні ліворуч

На рисунку 3.8 показано адміністративний інтерфейс для керування модулями. Нові картки можна додати, натиснувши кнопку [+] у нижньому лівому куті полотна. «Зафіксовані» картки на полотні можна перетягувати на потрібну позицію в цьому режимі редагування.

Картки, які мають бути розміщені студентами, відображаються у панелі, що згортається з правого боку екрана. Картки для певного раунду можна редагувати після натискання відповідного номера раунду на верхній панелі інструментів (рис. 3.9).

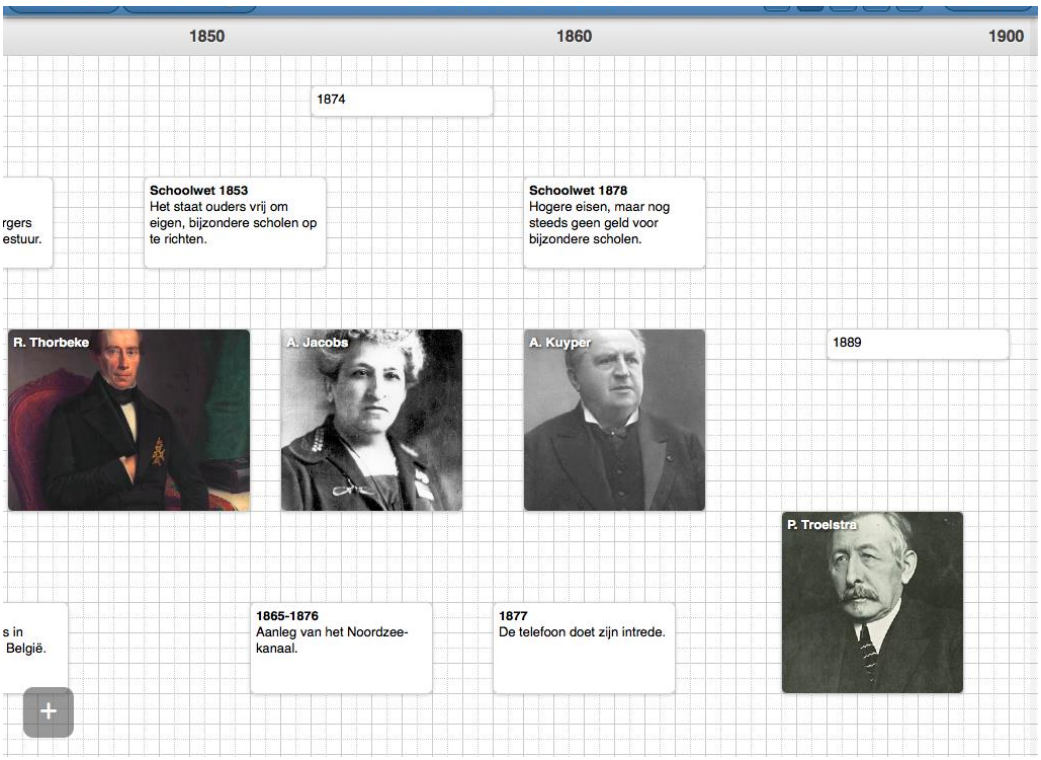

Рисунок 3.8 - Адміністративний інтерфейс для керування модулями

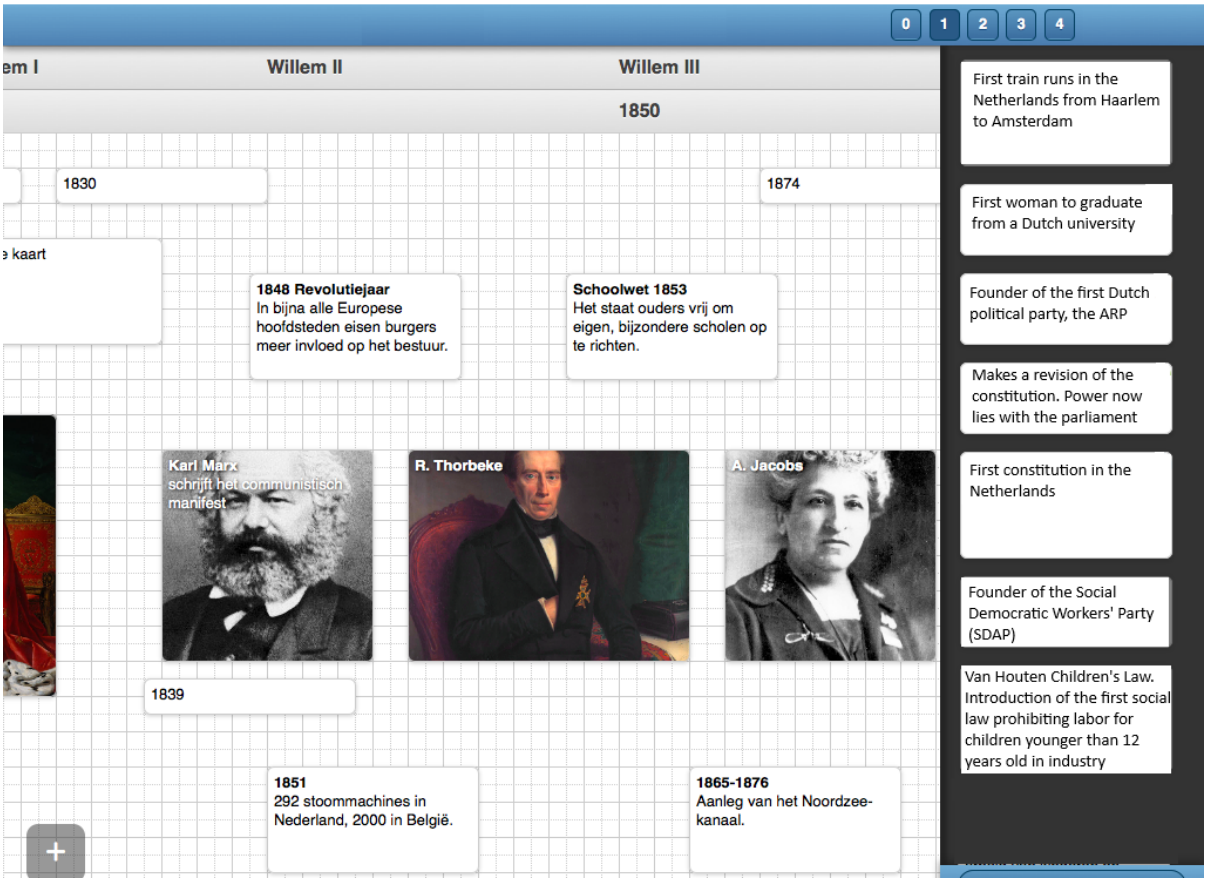

Рисунок 3.9 – Зміна та розміщення карток на полотні

Як ми бачили, адаптація існуючої, встановленої навчальної методології включає багато дизайнерських рішень щодо безпосереднього відображення або невеликої адаптації існуючих принципів до цифрової системи.

В роботі ми вирішили адаптувати навчальну методологію як можна безпосередньо, включивши планшети (телефони) лише як інструменти для розміщення карт на полотні. З точки зору комп'ютерних наук ми могли б вибрати переміщення обговорення в чат-функцію, але оскільки дослідження свідчать про те, що це може негативно вплинути на якість обговорення, ми вирішили не робити цього.

Для реалізації додатка ми вирішили не використовувати нативний інструментарій. Натомість ми вибрали веб-технології для того, щоб зробити програмне забезпечення крос-платформним, дозволяючи використовувати додаток на планшетах обох основних операційних систем, iOS та Android. Використовуючи стильові аркуші, ми імітуємо вигляд нативних додатків, роблячи додаток більш інтуїтивним.

Ми використовували технології WebSockets та JSON для забезпечення спілкування в реальному часі енергоефективним способом. Це також дозволило нам розширити методологію за допомогою підтримувальної інтерактивної дошки, яку можуть використовувати викладачі для ілюстрації та вирішення суперечок у обговореннях після кожного раунду.

Висновки до розділу

У третьому розділі розглянуто практичні аспекти реалізації інтерактивного навчального середовища з акцентом на програмні компоненти та архітектурні рішення.

Початковим етапом стала ідентифікація ключових сутностей, що дозволило формалізувати структуру системи та забезпечити чітке відображення реальних об'єктів у програмній моделі. На основі цього було

спроєктовано логічну модель бази даних, яка охоплює основні взаємозв’язки між сутностями та забезпечує збереження, цілісність і доступність навчальної інформації.

Запропонована реалізація абстрактного рівня WebSocket забезпечує ефективний двосторонній зв’язок у режимі реального часу, що критично важливо для підтримки синхронної взаємодії між учасниками навчального процесу.

Окремим етапом стало проектування користувацького інтерфейсу, орієнтованого на інтуїтивну взаємодію з користувачем, адаптивність та відповідність сучасним принципам UX/UI-дизайну.

Таким чином, реалізовані програмні компоненти формують цілісну систему, здатну забезпечити ефективне, стабільне й зручне навчальне середовище з інтерактивною підтримкою в реальному часі

ВИСНОВКИ

В дипломній роботі було здійснено повний цикл дослідження, проектування та реалізації інтерактивного навчального середовища, яке відповідає сучасним вимогам до цифрової освіти.

У першому розділі проведено аналіз предметної області застосування інтерактивних технологій у навчанні. Розглянуто методологічні основи освітнього процесу, принципи, на яких базується проектування системи, а також визначено технічні й функціональні вимоги до інтерактивного середовища. Здійснено класифікацію ключових компонентів: програмне забезпечення, стек технологій, архітектура користувацької взаємодії.

Другий розділ присвячено алгоритмічній реалізації середовища. Досліджено можливості сучасних протоколів передачі даних (Polling, WebSockets) та форматів обміну (JSON, XML), оцінено переваги асинхронного підходу до програмування у контексті навчальних систем. Детально описано способи забезпечення синхронізації даних і обґрунтовано вибір ефективних технічних рішень для реального часу.

У третьому розділі розроблено безпосередню програмну реалізацію. Визначено сутності системи, створено модель бази даних, реалізовано серверну частину з підтримкою WebSocket-з’єднання та розроблено інтерфейс користувача відповідно до принципів UX/UI-дизайну.

У межах даної роботи було проведено аналіз предметної області, досліджено освітні методики, на основі яких визначено принципи побудови інтерактивного середовища. Особливу увагу приділено алгоритмічним та програмним аспектам реалізації системи: обрано технологічний стек, спроєктовано архітектуру, базу даних, механізми синхронізації та інтерактивної взаємодії в режимі реального часу із використанням WebSocket-протоколу.

Отже, розроблена система є прикладом інтерактивного навчального середовища, здатного підтримувати ефективну комунікацію, адаптивність до потреб користувача та високу інтерактивність. Результати роботи можуть бути використані для подальшої розробки освітніх платформ або як основа для впровадження інтерактивних технологій у навчальні заклади.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bray, T. (2014, March). The JavaScript Object Notation (JSON) Data Interchange Format (RFC No. 7159). RFC Editor. Internet Requests for Comments. Retrieved from http://www.rfc-editor.org/rfc/rfc7159.txt
2. Ko, S., & Rossen, S. (2017). Teaching online: A practical guide. Routledge.
3. Laurillard, D. (2012). Teaching as a design science: Building pedagogical patterns for learning and technology. Routledge.
4. Mayer, R. E. (2009). Multimedia learning (2nd ed.). Cambridge University Press.
5. Charland, A., & Leroux, B. (2011). Mobile application development: Web vs. native. Communications of the ACM, 54(5), 49-53.
6. Eduapp. (2013). Retrieved 2014-01-20, from http://eduapp.nl/zoeken/apps
7. Fette, I., & Melnikov, A. (2011, December). The websocket protocol (RFC No. 6455). RFC Editor. Internet Requests for Comments. Retrieved from http://www.rfc-editor.org/rfc/rfc6455.txt
8. Dabbagh, N., & Bannan-Ritland, B. (2005). Online learning: Concepts, strategies, and application. Pearson.
9. Garrison, D. R. (2011). E-learning in the 21st century: A framework for research and practice. Routledge.
10. Herrington, J., Reeves, T. C., & Oliver, R. (2010). A guide to authentic e-learning. Routledge.
11. Achten, P., Koopman, P., & Plasmeijer, R. (2015). An introduction to task oriented programming. In V. Zsók, Z. Horváth, & L. Csató (Eds.), Central european functional programming school (Vol. 8606, p. 187-245). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-15940-9_5 doi: 10.1007/978-3-319-15940-9_5

12. Lipponen, L., Rahikainen, M., Lallimo, J., & Hakkarainen, K. (2003). Patterns of participation and discourse in elementary students' computer-supported collaborative learning. Learning and Instruction, 13, 487-509.
13. Martinsen, J. K., Grahn, H., & Isberg, A. (2013, March). Using speculation to enhance javascript performance in web applications. Internet Computing, IEEE, 17(2), 10-19. doi: 10.1109/MIC.2012.146
14. Bonk, C. J., & Graham, C. R. (2006). The handbook of blended learning: Global perspectives, local designs. Pfeiffer.
15. Clark, R. C., & Mayer, R. E. (2016). E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning. Wiley.
16. Qveflander, N. (2010). Pushing real time data using HTML5 Web Sockets (Unpublished master's thesis). Umeå University.
17. Resta, P., & Laferrière, T. (2007). Technology in support of collaborative learning. Educational Psychology Review, 19(1), 65-83. Retrieved from http://dx.doi.org/10.1007/s10648-007-9042-7 doi: 10.1007/s10648-007-9042-7
18. Swamy, R., & Mahadevan, G. (2011). Event Driven Architecture using HTML5 Web Sockets for Wireless Sensor Networks. White Papers, Planetary Scientific Research Center.
19. Ahn, W., Choi, J., Shull, T., Garzarán, M. J., & Torrellas, J. (2014). Improving javascript performance by deconstructing the type system. In Proceedings of the 35th acm sigplan conference on programming language design and implementation (pp. 496-507). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2594291.2594332 doi: 10.1145/2594291.2594332
20. Fjermestad, J. (2003). An analysis of communication mode in group support systems research. Decision Support Systems, 37, 239-263.

21. Havekes, H., Coppen, P. A., & Luttenberg, J. (2012). Knowing and doing history: A conceptual framework and pedagogy for teaching historical contextualisation. International Journal of Historical Learning, Teaching and Research, 11.1, 72-93.
22. Wessels, A., Purvis, M., Jackson, J., & Rahman, S. (2011). Remote data visualization through websockets. In Information technology: New generations (itng), 2011 eighth international conference on (pp. 1050-1051).
23. Anderson, T. (2008). The theory and practice of online learning. Athabasca University Press.
24. Bates, A. W. (2019). Teaching in a digital age: Guidelines for designing teaching and learning. Tony Bates Associates Ltd.
25. Beetham, H., & Sharpe, R. (Eds.). (2013). Rethinking pedagogy for a digital age: Designing for 21st century learning. Routledge.
26. Jonassen, D. H. (2000). Toward a design theory of problem solving. Educational Technology Research and Development, 48(4), 63–85.
27. Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. Teachers College Record, 108(6), 1017–1054.
28. Moore, M. G., & Kearsley, G. (2011). Distance education: A systems view of online learning. Wadsworth.
29. Palloff, R. M., & Pratt, K. (2007). Building online learning communities: Effective strategies for the virtual classroom. Jossey-Bass.
30. Salmon, G. (2011). E-moderating: The key to teaching and learning online (3rd ed.). Routledge.
31. Simonson, M., Smaldino, S., & Zvacek, S. (2014). Teaching and learning at a distance: Foundations of distance education. Information Age Publishing.
32. Spector, J. M., Merrill, M. D., Elen, J., & Bishop, M. J. (Eds.). (2014). Handbook of research on educational communications and technology. Springer.

ДОДАТКИ

Додаток А

Фрагменти програмних кодів

Authentication.php

```php
class Authentication
    // Checks whether a user exists in the database.
    public static function checkExists($id_user)
    // Finds the user id belonging to a certain emailaddress.
    public static function getUserId($emailaddress)
    // Verifies whether the user is currently logged in.
    public static function isLoggedIn()
    // Checks a password for a given username against the database.
    public static function checkPassword($emailaddress, $password)
    // Resets a password for a certain user.
    public static function updatePassword($id_user, $hash)
    public static function create_hash($password)
    public static function slow_equals($a, $b)
    /*
    * PBKDF2 key derivation function as defined by RSA's PKCS #5: https://www.ietf.org/rfc/rfc2
    * $algorithm - The hash algorithm to use. Recommended: SHA256
    * $password - The password.
    * $salt - A salt that is unique to the password.
    * $count - Iteration count. Higher is better, but slower. Recommended: At least 1000.
    * $key_length - The length of the derived key in bytes.
    * $raw_output - If true, the key is returned in raw binary format. Hex encoded otherwise.
    * Returns: A $key_length-byte key derived from the password and salt.
    *
    * Test vectors can be found here: https://www.ietf.org/rfc/rfc6070.txt
    */
    public static function pbkdf2($algorithm, $password, $salt, $count, $key_length, $raw_outpu
```

CardBehaviour.php

```php
class CardBehaviour
    public $id_behaviour;
    public $id_type;
    public $round;
    public $fits_on;
    private function __construct(array $data)
    public static function fromId($id_behaviour, $return_format = 'array')
    public static function getByType($id_type, $return_format = 'array')
    public static function getByTypes(array $id_types, $return_format = 'array')
    public static function createNew(array $data, $return_format = 'object')
    public function save()
    public function delete()
```

Card.php

```php
class Card
    public $id_card;
    public $id_module;
    public $id_type;
    public $round;
    public $dimensions_x;
    public $dimensions_y;
    public $position_x;
    public $position_y;
    public $image_file;
    public $forecolor;
    public $backcolor;
    public $text;
    private function __construct(array $data)
    public static function fromId($id_card, $return_format = 'object')
    public static function getByModule($id_module, $round = 'all', $return_format = 'object')
    public static function createNew(array $data, $return_format = 'object')
    public function save()
    public function delete()
```

CardType.php

```php
class CardType
    public $id_type;
    public $id_module;
    public $name;
    public $dimensions_x;
    public $dimensions_y;
    private function __construct(array $data)
    public static function fromId($id_type, $return_format = 'array')
    public static function getByModule($id_module, $return_format = 'array')
    public static function createNew(array $data, $return_format = 'object')
    public function save()
    public function delete()
```

Group.php

```php
class Group
    public $id_group;
    public $id_session;
    public $creation_time;
    public $name;
    private function __construct(array $data)
    public static function fromId($id_group, $return_format = 'object')
    public static function getAll($return_format = 'object')
    public static function createNew(array $data, $return_format = 'object')
```

Database.php

```php
class Database
    private $connection;
    private $query_count = 0;
    /**
    * Initialises a new database connection.
    * @param server: server to connect to.
    * @param user: username to use for authentication.
    * @param password: password to use for authentication.
    * @param name: database to select.
    */
    public function __construct($server, $user, $password, $name)
    // Fetches a row from a given recordset, using field names as keys.
    public function fetch_assoc($resource)
    // Fetches a row from a given recordset, using numeric keys.
    public function fetch_row($resource)
    // Destroys a given recordset.
    public function free_result($resource)
    // Moves internal pointer in given recordset.
    public function data_seek($result, $row_num)
    // Returns the amount of rows in a given recordset.
    public function num_rows($resource)
    // Returns the amount of fields in a given recordset.
    public function num_fields($resource)
    // Escapes a string.
    public function escape_string($string)
    // Unescapes a string.
    public function unescape_string($string)
    // Returns the last MySQL error.
    public function error()
    // Returns server info.
    public function server_info()
    // Selects a database on a given connection.
    public function select_db($database)
    // Returns the amount of rows affected by the previous query.
    public function affected_rows()
    // Returns the id of the row created by a previous query.
    public function insert_id()
    // Do a MySQL transaction.
    public function transaction($operation = 'commit')
    // Function used as a callback for the preg_match function that parses variables into datab
    private function replacement_callback($matches)
    // Escapes and quotes a string using values passed, and executes the query.
    public function query($db_string, $db_values = array())
    /**
    * Escapes and quotes a string just like db_query, but does not execute the query.
    * Useful for debugging purposes.
    */
    public function quote($db_string, $db_values = array())
    // Executes a query, returning an array of all the rows it returns.
    public function queryRow($db_string, $db_values = array())
```

```
    // Executes a query, returning an array of all the rows it returns.
    public function queryRows($db_string, $db_values = array())
    // Executes a query, returning an array of all the rows it returns.
    public function queryPair($db_string, $db_values = array())
    // Executes a query, returning an array of all the rows it returns.
    public function queryPairs($db_string, $db_values = array())
    // Executes a query, returning an associative array of all the rows it returns.
    public function queryAssoc($db_string, $db_values = array())
    // Executes a query, returning an associative array of all the rows it returns.
    public function queryAssocs($db_string, $db_values = array(), $connection = null)
    // Executes a query, returning the first value of the first row.
    public function queryValue($db_string, $db_values = array())
    // Executes a query, returning an array of the first value of each row.
    public function queryValues($db_string, $db_values = array())
    // This function can be used to insert data into the database in a secure way.
    public function insert($method = 'replace', $table, $columns, $data)
    // This function tries to work out additional error information from a back trace.
    private function error_backtrace($error_message, $log_message = '', $error_type = false, $f
```

Error.php

```
class Error
    public $id_entry;
    public $message;
    public $file;
    public $line;
    public $time;
    public $ipaddress;
    public $url;
    public $id_user;
    private function __construct($data)
    public static function fromId($id_entry)
    public static function log(array $data, $return_format = 'bool')
    public static function flushLog()
    public static function getCount()
```

ErrorHandler.php

```
class ErrorHandler
    private static $error_count = 0;
    // Triggers a fatal error, presenting the message in $text to the user, unless it contains
    public static function triggerFatalError($text, $contains_debug_info = false, $needs_loggin
    public static function trigger400()
    public static function trigger403()
    public static function trigger404()
    // Kicks a guest to a login form, redirecting them back to this page upon login
    public static function kickGuest($message, $url = '')
    private static function logError($error_message = '', $file = '', $line = 0)
    public static function handleError($error_level, $error_string, $file, $line)
```

GroupMembership.php

```
class GroupMembership
    public $id_group;
    public $id_user;
    public $join_date;
    private function __construct(array $data)
    public static function getMembershipBySession()
    public static function joinGroup($id_user, $id_group, $new_name = ' ')
```

Instance.php

```
class Instance
    public $id_instance;
    public $id_card;
    public $id_group;
    public $container;
```

```
public $position_x;
    public $position_y;
    private function __construct(array $data)
    public static function fromId($id_instance, $return_format = 'object')
    public static function getByGroup($id_group, $container = 'all', $return_format
        = 'array')
    public static function getByGroups($container = 'all', $return_format = 'array'
        )
    public static function createNew($id_group, $return_format = 'array',
        $from_round = 1, $to_round = 1)
    public function save()
    public static function update($data)
    public function delete()
```

Label.php

```
class Label
    public $id_label;
    public $id_module;
    public $container;
    public $position_x;
    public $caption;
    private function __construct(array $data)
    public static function fromId($id_label, $return_format = 'array')
    public static function getByModule($id_module, $container = 'all',
        $return_format = 'array')
    public static function createNew(array $data, $return_format = 'object')
    public function save()
    public function delete()
```

Member.php

```
class Member extends User
    private function __construct($data)
    public static function fromId($id_user)
    public static function fromSlug($slug)
    // Створює нового учасника на основі наданих даних.
    public static function createNew(array $data)
    // Оновлює учасника за допомогою наданих даних.
    public function update(array $new_data)
    // Видаляє учасника.
    public function delete()
    /**
    * Перевіряє, чи адреса електронної пошти вже пов'язана з обліковим записом.
    * @param emailaddress для перевірки
    * @return false якщо обліковий запис не існує
    * @return user id якщо користувач існує
    */
    public static function exists($emailaddress)
    public function updateAccessTime()
    public static function getCount()
    public function getProps()
```

Module.php

```
class Module
    public $id_module;
    public $title;
    public $description;
    public $num_rounds;
    public $canvas_x;
    public $canvas_y;
    public $is_active;
    private function __construct(array $data)
    public static function fromId($id_module, $return_format = 'object')
    public static function getAll($return_format = 'object')
    public static function getPairs()
    public static function createNew(array $data, $return_format = 'object')
    public function save()
    public function delete()
    public function getCards($round = 'all', $return_format = 'array')
    public function getTypes($return_format = 'array')
    public function getLabels($container = 'all', $return_format = 'array')
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “ Проектування та розробка інтерактивного навчального середовища ”

Обсяг пояснювальної записки: 75 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____