

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "магістр"

на тему: Системний аналіз процесу створення розподіленої інформаційної системи на прикладі сервісу спільного редагування документів

Виконав: студентка 6144 м групи

_____ ***Кардаш А.О.***
(підпис)

Керівник роботи:

к.т.н., доцент каф.ІУСТ,
(посада, науковий ступінь вчене звання)

_____ ***Книрік Н.Р.***
(підпис)

м. Миколаїв – 2021 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 126 "Інформаційні системи та технології"
Освітня програма "Інформаційні системи та технології"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

к.т.н, доц. Чернова Лб.С.

" ____ " _____ 2021 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"

Студентці Кардаш Анастасії Олександрівні

(прізвище, ім'я, по батькові)

1. Тема роботи: Системний аналіз процесу створення розподіленої інформаційної системи на прикладі сервісу спільного редагування документів

Керівник роботи к.т.н., доц. Кнirik Н.Р.

Затверджені наказом ректора № 1232-уч. від "12" жовтня 2021 року.

2. Термін подання роботи: 21 грудня 2021 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація для розробки web-сервісів, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів) _____

- Титульний аркуш, завдання на дипломну роботу, анотація (українською, російською, англійською), зміст, перелік умовних означень, символів, одиниць та термінів (при необхідності).
- Вступ.
- Дослідження технологій створення розподілених інформаційних систем.
- Формування вимог до інформаційної системи, розробка її концепції і постановка задачі.
- Розробка проектних рішень по інформаційній системі.
- Реалізація проекту інформаційної системи (технічна документація по проекту).
- Розділ з економіки.
- Розділ з охорони праці.
- Розділ з охорони навколишнього середовища.
- Висновки.
- Список використаної літератури.
- Додатки: текст програми.

5. Перелік презентаційних матеріалів

- 5.1 Мета, актуальність та завдання дослідження.
- 5.2 Загальна архітектура.
- 5.3 Фізична модель бази даних.
- 5.4 Діаграма використання.
- 5.5 Діаграма послідовності.
- 5.6 Діаграма діяльності
- 5.7 Діаграма послідовності.
- 5.8 Інтерфейс користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	К.т.н., доц. Кнirik Н.Р.		
2	К.т.н., доц. Кнirik Н.Р.		
3	К.т.н., доц. Кнirik Н.Р.		
4			
5	Ст. викл. Гурець Н.В.		
6	Ст. викл. Гурець Н.В.		

7. Дата видачі завдання 04 вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Вибір теми, визначення мети, завдань, об'єкта та предмета дослідження	01 жовтня 2020 р.	
2	Підбір та опрацювання літератури	01 лютого 2021 р.	
3	Дослідження технологій інформаційного забезпечення	01 травня 2021 р.	
4	Розробка концепції інформаційного забезпечення	01 вересня 2021 р.	
5	Розробка та реалізація проекту	01 жовтня 2021 р.	
6	Розробка додаткових розділів, оформлення роботи, підготовка презентаційних матеріалів	01 грудня 2021 р.	
7	Представлення до попереднього захисту	01 грудня 2021 р.	
8	Отримання відгуку керівника	13 грудня 2021 р.	
9	Отримання рецензії	15 грудня 2021 р.	
10	Захист кваліфікаційної роботи	21 грудня 2021 р.	

Студент

(підпис)

Кардаш А.О.

(ПІБ)

Керівник роботи

(підпис)

Кнirik Н.Р.

(ПІБ)

Анотація

Дана магістерська робота присвячена дослідженню засобів та технологій створення розподіленої інформаційної системи спільного редагування документів у режимі реального часу. У роботі досліджується сучасний стан технології та існуючі рішення.

У роботі також виконується розробка інформаційної системи. Для створення системи розробляється її концепція, приймаються загальні рішення, а також рішення з організаційного, інформаційного, технічного та програмного забезпечень. Розробляється відповідна документація. Також в роботі виконується розділи з техніко-економічної частини, охорони праці, охорони навколишнього середовища.

Робота виконана українською мовою на 106 сторінках друкованого тексту, містить 32 рисунки, 6 таблиць, 3 додатки та список література з 19 джерел.

Аннотация

Данная магистерская работа посвящена исследованию средств и технологий создания распределенной информационной системы взаимного редактирования документов в режиме реального времени. В работе исследуются современное состояние технологий и существующие решения.

В работе также выполняется разработка информационной системы. Для создания системы разрабатывается ее концепция, принимаются общие решения, а также решения по организационному, информационному, техническому и программному обеспечению. Разрабатывается соответствующая документация. Также в работе выполняются разделы с технико-экономической части, охраны труда, охраны окружающей среды.

Работа выполнена на украинском языке на 106 страницах печатного текста, содержит 32 рисунка, 6 таблиц, 3 приложений и список литературы из 19 источников.

Abstract

This master's thesis is devoted to the methods and technologies research of creating distributed information system of co-editorial documents in real-time. The thesis investigates the current state of technologies and existing solutions.

This work develops an information system. The concept of developing system is being developed, made basic decisions as well as organizational, informational, technical and software decisions. The relevant documentation is being developed. There are also technical and economic parts, labor protection and environment protection.

The work is done in Ukrainian on 106 pages of printed text, contains 32 pictures, 6 tables, 3 applications and a list of literature from 19 sources.

Зміст

Вступ	10
1. Загальна частина	
1.1. Загальні відомості про веб-додатки	11
1.2. Загальна архітектура	12
1.3. Мережа модель OSI	13
1.3.1. Транспортні протоколи передачі даних	14
1.4. Горизонтальне та вертикальне масштабування	14
1.4.1. Вертикальне масштабування	15
1.4.2. Горизонтальне масштабування	15
1.5. Серверна частина	15
1.5.1. Node.js	16
1.5.2. PHP	16
1.5.3. Порівняння	17
1.6. Протоколи прикладного рівня	18
1.6.1. HTTP та HTTPS	18
1.6.2. WebSocket and Secure WebSocket	18
1.7. Шифрування	20
1.7.1. Симетричні алгоритми	20
1.7.1.1. Advanced Encryption Standard (AES)	21
1.7.2. Асиметричні алгоритми	24
1.7.2.1. Rivest, Shamir и Adleman (RSA)	24
1.7.3. Змішане шифрування	25
1.7.4. TLS	26
1.8. Збереження даних	28
1.8.1. Реляційні бази даних	28
1.8.2. NoSQL бази даних	29
1.8.3. Порівняння	29
2. Формування вимог до спільного редагування документів у режимі реального часу у веб-додатку, розробка її концепції та постановка задачі	
2.1. Формування вимог до спільного редагування документів у режимі реального часу у веб-додатку	37

2.1.1. Характеристики об'єкту автоматизації	32
2.1.2. Існуючий стан автоматизації	32
2.1.3. Обґрунтування необхідності модернізації ІС	33
2.1.4. Цілі, критерії та обмеження впровадження модернізації ІС	33
2.1.5. Задачі і функції створюваної системи	34
2.1.6. Висновки і пропозиції	34
2.2. Розробка концепції інформаційної системи	35
2.2.1. Розробка концепції	35
2.2.2. Обґрунтування варіанту	35
2.2.3. Очікувані результати від впровадження системи	36
2.2.4. Умови приймання інформаційної системи	36
2.3. Постановка задач	37
3. Реалізація інформаційної схеми спільного редагування документів у режимі реального часу	
3.1. Загальна архітектура	38
3.2. Обмін даними	39
3.3. Шифрування даних	42
3.4. Збереження даних	43
3.5. Взаємодія клієнта з додатком	44
3.6. Формат збереження та передачі даних	45
3.6.1. JSON-формат	45
3.6.2. BSON-формат	46
4. Розрахунок економічної ефективності від розробки і використання проекрованої установки	
4.1. Вступ до проблем	47
4.2. Розрахунок вартості створення інформаційної системи спільного редагування документів у режимі реального часу	47
4.3. Розрахунок економічної ефективності	50
4.4. Підсумки	51
5. Охорона праці	
5.1. Вступ	53

5.2. Аналіз шкідливих та небезпечних факторів у відділі розробки програмного забезпечення	54
5.3. Розрахунок систем штучного освітлення приміщення відділу розробки програмного забезпечення	57
5.4. Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів	59
6. Охорона навколишнього середовища	
6.1. Вступ	63
6.2. Забруднення навколишнього середовища підприємством з розробки програмного забезпечення	66
6.3. Розробка заходів щодо зменшення забруднення	67
Висновок	69
Список використаних джерел	70
Додаток А. Технічне завдання на розробку системи	72
Додаток Б. Посібник користувача	75
Додаток В. Исходний код інформаційної системи	79

Вступ

Метою виконання дипломного проекту є проаналізувати та дослідити засоби та технології створення розподіленої інформаційної системи спільного редагування документів у режимі реального часу та створити власний веб-додаток.

Актуальність роботи полягає у тому, що зростає кількість технологій, які потенційно можна використовувати при розробці онлайн платформ, і тому стає дуже важливим питання вибору, які з них використовувати доцільніше. Також зростаючий попит на онлайн-курси збільшую необхідність у онлайн-платформах. Як результат для створювання таких платформ необхідно провести аналіз існуючих засобів та змоделювати можливе рішення, базуючись на отриманих результатах.

Всесвітня мережа зараз містить величезну кількість інформації та знань. Користувачі на різних умовах можуть переглядати всілякі документи, аудіо – та відеофайли.

За оцінкою фахівців Інтернет використовую більше 4,5 мільярдів людей по всьому світу, це маже 58% від населення Землі. Велика кількість людей так чи інакше обмінюються інформацією через Інтернет, тому компанії почали створювати платформи для взаємодії між людьми. На сьогоднішній день існують електрона пошта, месенджери, а також платформи для взаємного редагування документів, створені Google та Microsoft. Але ці платформи не дозволяють використовувати їх для проведення онлайн-курсів. Близько 20 мільйонів людей навчаються у коледжах та університетах. З цієї кількості 2/3 зареєстровано на онлайн-курси. Ці курси використовуються уніфіковані платформи з записаними попередньо відео або VoIP-телефонію, що є не завжди зручно.

В Україні с кожним роком росте кількість онлайн-курсів, деякі школи, коледже та університети пропонують дистанційні програми навчання. Тому є велика необхідність у платформах, які допомагали б при проведенні цих онлайн-курсів та дистанційних програм.

Дана робота спрямована на вивчення засобів та технологій створення розподіленої інформаційної системи спільного редагування документів у режимі реального часу. Підставою для виконання даної розробки є відповідне завдання на магістерську роботу.

1 ЗАГАЛЬНА ЧАСТИНА

1.1 Загальні відомості про веб-додатки

Веб-додаток - клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером - веб-сервер. Логіка веб-додатків розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Одним з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є кроссплатформенними сервісами. Веб-додатки стали широко популярними в кінці 1990-х - початку 2000-х років.

Суттєва перевага побудови веб-додатків для підтримки стандартних функцій браузера полягає в тому, що функції повинні виконуватися незалежно від операційної системи даного клієнта. Замість того, щоб писати різні версії для Microsoft Windows, Mac OS X, GNU / Linux та інших операційних систем, додаток створюється один раз для довільно обраної платформи і на ній розгортається. Однак різна реалізація HTML, CSS, DOM і інших специфікацій в браузерах може викликати проблеми при розробці веб-додатків і подальшій підтримці. Крім того, можливість користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольору, відключення підтримки сценаріїв) може перешкоджати коректній роботі програми.

Веб-додаток складається з клієнтської і серверної частин, тим самим реалізуючи технологію «клієнт-сервер». Клієнтська частина реалізує користувацький інтерфейс, формує запити до сервера і обробляє відповіді від нього. Серверна частина отримує запит від клієнта, виконує обчислення, після цього формує веб-сторінку і відправляє її клієнту через мережу з використанням протоколу HTTP. Останнім часом набирає велику популярність технологія WebSocket, яка не вимагає постійних запитів від клієнта до сервера, а створює двонаправлене з'єднання, при якому сервер може відправляти дані клієнта без запиту від останнього. Таким чином з'являється можливість динамічно управляти контентом в режимі реального часу.

Для створення веб-додатків на стороні сервера використовуються різноманітні технології, найбільш популярна для проектів, які обмінюються даними у

реальному часі є Node.js. На стороні сервера використовуються різноманітні back-end фреймворки, а на стороні клієнта використовуються різноманітні front-end фреймворки. Фреймворки надають базову інфраструктуру та структуру проекту, а також набір бібліотек, які полегшують розробку додатків.

1.2 Загальна архітектура

Трирівнева архітектура (рис. 1.1) - це тип архітектури програмного забезпечення, який складається з трьох "рівнів" або "шарів" логічних обчислень. Вони часто використовуються в додатках як для системи клієнт-сервер. Трирівнева архітектура надає багато переваг як для підтримки так і для розробки, розподіляючи на модулі: інтерфейс користувача, бізнес-логіку та рівень зберігання даних. Це дає більшу гнучкість командам розробників, дозволяючи їм оновлювати певну частину програми незалежно від інших частин. Ця додаткова гнучкість може покращити загальний час виходу на ринок та скоротити час циклу розробки, надаючи командам розробників можливість замінювати або модернізувати незалежні рівні, не впливаючи на інші частини системи.

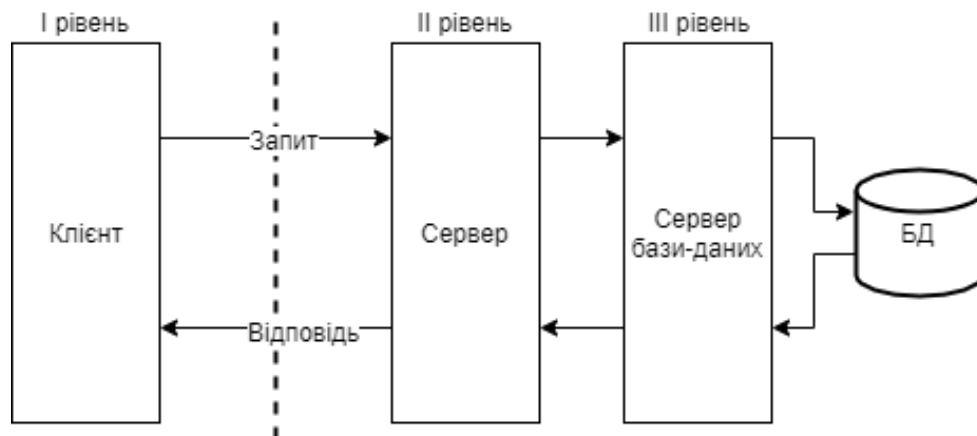


Рис. 1.1 – Трирівнева модель клієнт-сервер

Виділяють такі рівні моделі (рис. 1.2):

- Рівень представлення – це front-end у трирівневій системі та складається з інтерфейсу користувача. Інтерфейс користувача є графічним, доступним через веб-браузер або веб-додаток, який відображає вміст та інформацію. Цей рівень найчастіше побудований на веб-технології як HTML, JavaScript, CSS, тощо та спілкується з іншими рівнями за допомогою API.

- Рівень програми – містить функціональну бізнес-логіку, яка керує основними можливостями програми. Цей рівень пишеться на Java, .NET, Python, C++, тощо.
- Рівень даних – складається з бази даних, системи зберігання даних та рівня доступу до даних. Наприклад: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, тощо. Доступ до даних здійснюється на рівні програми через виклики API.

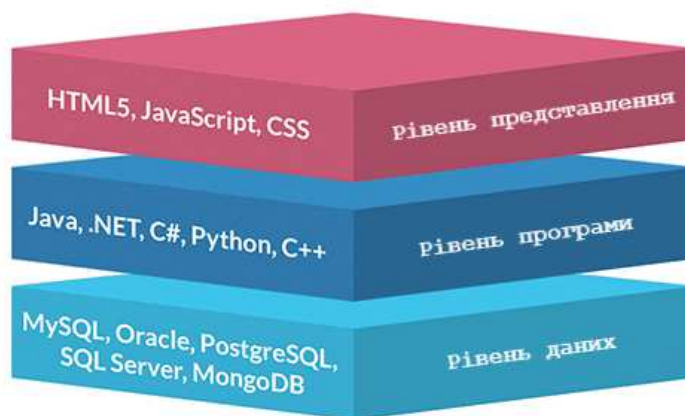


Рис. 1.2 – Рівні трирівневої архітектури

1.3 Мережа модель OSI

Мережева модель OSI (еталонна модель взаємодії відкритих систем — (англ. Open Systems Interconnection Reference MODEL-OSI) — абстрактна модель для мережових комунікацій і розробки мережових протоколів. Представляє собою рівневий підхід до мережі.

Кожен рівень обслуговує свою частину процесу взаємодії. Завдяки такій структурі спільна робота мережевого устаткування і програмного забезпечення набагато простіша і зрозуміліша.

Модель складається з 7-ми рівнів (таб. 1.1), розташованих «вертикально» один над одним. Кожен рівень може взаємодіяти лише зі своїми сусідами і виконувати відведені тільки йому функції.

Таблиця 1.1 – 7 рівнів моделі OSI

№	Рівень OSI	Протоколи
7	прикладний	HTTP, DNS, DHCP, SMTP, FTP, SSH, IMAP, POP3, WebSocket, BitTorrent, Telnet

6	представлення	XML, ASCII, Unicode
5	сеансовий	ADSP, SSL, TLS, PPTP
4	транспортний	TCP, UDP
3	мережевий	IPv4, IPv6, IPSec
2	канальний	Ethernet, ARP, PPPoE, PPTP, L2TP, WiFi
1	фізичний	RS-232, RS-422, ITU-T, RJ-11

Стандартом де-факто для глобальних мереж у цей час є стек протоколів TCP/IP.

1.3.1 Транспортні протоколи передачі даних

Транспортний рівень пов'язує більш високі рівні, які сильно залежать від додатків, з нижніми рівнями, які більше прив'язані до ліній зв'язку. На транспортному рівні відбувається розбиття потоку даних на сегменти при відправленні даних або збирання вихідного потоку даних із сегментів при прийманні. Сегментом називається блок даних транспортного рівня. Транспортний рівень призначений для доставки даних без помилок, втрат і дублювання в тій послідовності, у якій вони були передані. Він забезпечує передачу даних між двома додатками з необхідним рівнем надійності. Протоколи транспортного рівня, які гарантують надійну доставку даних, встановлюють перед обміном даними віртуальне з'єднання та у випадку втрати або пошкодження сегментів повторно їх відправляють (наприклад, TCP). Протоколи ненадійної доставки не ретранслюють дані (наприклад, UDP).

1.4 Масштабування інформаційних систем

Масштабованість (scalability) означає здатність системи збільшувати свою продуктивність при додаванні ресурсів (зазвичай апаратних). Масштабованість - важливий аспект електронних систем, програмних комплексів, баз даних, маршрутизаторів, мереж і т. п., якщо для них потрібна можливість працювати під великим навантаженням. Система називається масштабованою, якщо вона здатна збільшувати продуктивність пропорційно додатковим ресурсам. Масштабованість можна оцінити через відношення приросту продуктивності системи до приросту ресурсів, які вона використовує. Чим ближче цей показник

до одиниці, тим краще. Також під масштабованістю розуміється можливість нарощування додаткових ресурсів без структурних змін центрального вузла системи. Існує два види масштабованості: горизонтальна та вертикальна.

1.4.1 Вертикальне масштабування

Збільшення кількості доступних для ПЗ ресурсів за рахунок збільшення потужностей застосованих серверів (рис. 1.3).

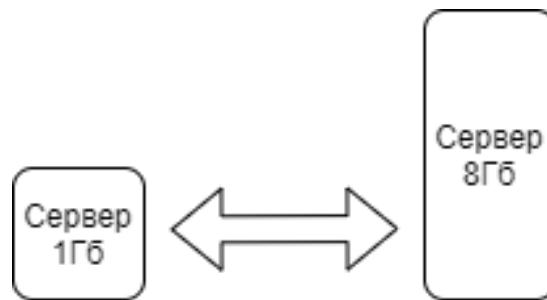


Рис 1.3 – Вертикальне масштабування

1.4.2 Горизонтальне масштабування

Збільшення кількості серверів, об'єднаних в кластери серверів при нехватці потужностей процесорів, пам'яті або дискового простору (рис. 1.4).



Рис 1.4 – Горизонтальне масштабування

1.5 Серверна частина

1.5.1 Node.js

Node.js - це середовище виконання JavaScript з відкритим кодом та міжплатформною. Node.js запускає двигун JavaScript V8, ядро Google Chrome, поза браузером. Це дозволяє Node.js бути дуже ефективним.

Додаток Node.js запускається в одному процесі, не створюючи нової потоки для кожного запиту. Node.js надає набір асинхронних примітивів вводу та виводу у своїй стандартній бібліотеці, які запобігають блокуванню коду JavaScript і, як правило, бібліотеки в Node.js записуються з використанням неблокуючих парадигм, що робить поведінку блокування винятком, а не нормою. Коли Node.js повинен виконати операцію вводу та виводу, як-от читання з мережі, доступ до бази даних або файлової системи, замість блокування потоку і витрачання очікування циклів процесора, Node.js відновить операції, коли відповідь повернеться. Це дозволяє Node.js обробляти тисячі одночасних з'єднань з одним сервером, не вводючи тягар управління паралельністю потоку, що може бути важливим джерелом помилок.

Як крос-платформне серверне середовище, Node.js полегшує розробку швидких та масштабованих веб-додатків з JavaScript. Також можна додатково прискорити розробку веб-додатків та прикладних програмних інтерфейсів (API), використовуючи ряд фреймворків Node.js.

1.5.2 PHP

PHP (рекурсивна аббревіатура для PHP: Hypertext Preprocessor) - широко використовувана скриптова мова загального призначення з відкритим ісходним кодом, яка особливо підходить для веб-розробки та може бути вбудована в HTML.

За допомогою HTML розробляються веб-сторінки. Цей код буде виконаний у браузері користувача (на стороні клієнта). Серверна скриптова мова PHP виконується на стороні сервера та результат відправляється з HTML-кодом веб-сторінки.

PHP може бути вбудований у HTML, і він добре підходить для веб-розробки та створення динамічних веб-сторінок для веб-додатків, електронної комерції та додатків баз даних. Мова PHP здатна легко з'єднуватися з MySQL, Oracle та іншими базами даних.

1.5.3 Порівняння

Архітектура сервера з використанням PHP працює як (рис. 1.5): веб-сервер отримавши запит від клієнта передає його в новопроініціалізовану середу PHP процес. Процес PHP оброблює один запит, після чого результат вертається у веб-сервер та процес перестає існувати. Веб-сервер отримавши відповідь відправляє результат клієнту та закриває з'єднання.

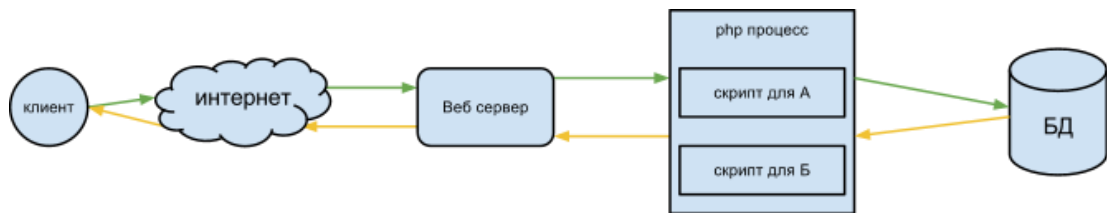


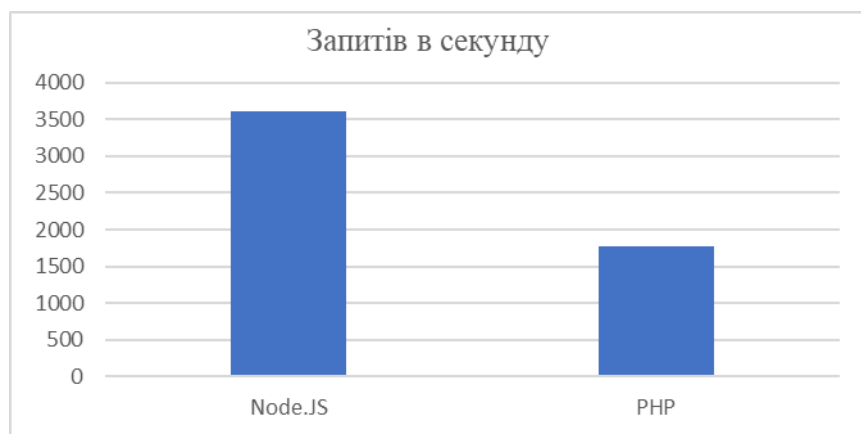
Рис 1.5 – Архітектура взаємодії з використанням PHP

Архітектура сервера Node.js працює як (рис. 1.6): веб-сервер отримавши запит від клієнта передає його в обробник для цього клієнта. Процес веб-сервера та обробників постійно висить у пам'яті. Обробники працюють асинхронно та стан веб-сервера шариться між обробниками.



Рис 1.6 – Архітектура взаємодії з використанням Node.js

На діаграмі 1.1 видно, як різниця в процесі обробки запитів на сервер призводить до більшої пропускнуєї спроможності при використанні асинхронного Node.js.



Діаграма 1.1 – Оброблення запитів в секунду

1.6 Протоколи прикладного рівня

Прикладний рівень моделі (Application Layer) забезпечує взаємодію мережі і користувача. Рівень дозволяє програмам користувача доступ до мережевих служб, таким як обробник запитів до баз даних, доступ до файлів, пересилки електронної пошти. Також цей рівень відповідає за передачу службової інформації, надає програмам інформацію про помилки і формує запити до рівня представлення. Одиниця даних, якою оперує прикладний рівень, називається повідомленням (message).

1.6.1 HTTP та HTTPS

HTTP - широко поширений протокол передачі даних, спочатку призначений для передачі гіпертекстових документів.

Протокол HTTP використовує клієнт-серверну структуру передачі даних. Клієнтська програма формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє цей запит, формує відповідь і передає його назад клієнтові. Після цього клієнтська програма може продовжити відправляти інші запити, які будуть оброблені аналогічним чином.

Передача даних по протоколу HTTP здійснюється через TCP/IP-з'єднання. Серверне програмне забезпечення при цьому зазвичай використовує TCP-порт 80.

API багатьох програмних продуктів також використовує HTTP для передачі даних - самі дані при цьому можуть мати будь-який формат, наприклад, XML або JSON.

Сам по собі протокол HTTP не передбачає використання шифрування при передачі інформації. Проте, для HTTP є поширене розширення, яке реалізує упаковку переданих даних в криптографічний протокол SSL або TLS.

Для HTTPS-з'єднань зазвичай використовується TCP-порт 443. HTTPS широко використовується для захисту інформації від перехоплення.

1.6.2 WebSocket and Secure WebSocket

WebSocket - це комп'ютерний протокол комунікацій, що забезпечує повно дуплексні канали зв'язку через одне з'єднання TCP.

Протокол WebSocket дозволяє взаємодія між веб-браузером (або іншим клієнтським додатком) та веб-сервером, полегшуючи передачу даних у реальному часі з сервера та на нього. Це стає можливим завдяки забезпеченню стандартизованого способу для сервера надсилання контенту клієнтові без попереднього запиту клієнта та дозволяють передавати повідомлення, зберігаючи з'єднання відкритим. Таким чином, між клієнтом і сервером може відбуватися двостороння бесіда. Зв'язок здійснюється через порт 80 TCP (або 443 у випадку зашифрованих TLS з'єднань), що є корисним для тих середовищ, які блокують немережеві підключення до Інтернету за допомогою брандмауера. Подібні двосторонні зв'язки між браузером та сервером були досягнуті нестандартними способами за допомогою технологій зупинки, таких як Comet. WebSocket відрізняється від HTTP. Обидва протоколи розміщені на рівні 7 в моделі OSI і залежать від TCP на рівні 4. WebSocket призначений для роботи над HTTP-портами 80 і 443, а також для підтримки HTTP-проксі, таким чином робить його сумісним з протоколом HTTP. Щоб досягти сумісності, для рукописання WebSocket використовується заголовок оновлення HTTP для того щоб перейти від протоколу HTTP до протоколу WebSocket (рис. 1.7). Також існують WebSocket Secure для шифрованого з'єднання використовуючи TLS, що примушує використовувати HTTPS протокол.

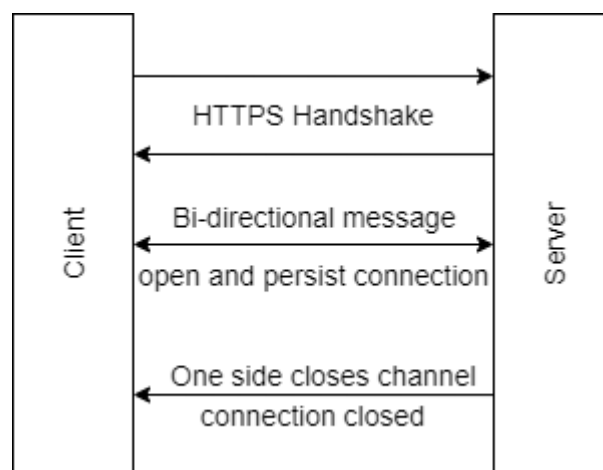


Рисунок 1.7 – З'єднання клієнта та сервера через WebSocket Secure протокол

Більшість сучасних браузерів підтримують протокол, включаючи Google Chrome, Microsoft Edge, Internet Explorer, Firefox, Safari та Opera.

1.7 Шифрування

Криптологія визначається як вивчення криптографії або криптоаналізу. Криптографія - це просто процес таємного спілкування за допомогою шифрів, а криптоаналіз - це процес злому або розшифрування таких таємних комунікацій. Використання криптографії у цивільному житті стає все більш популярним, оскільки в Інтернеті відбуваються більш критичні транзакції. Сніффінг мережі є настільки поширеним, що паролі, номери кредитних карток та інша захищена інформація можуть бути викрадені за допомогою незашифрованих протоколів. Зашифровані протоколи комунікацій дозволяють вирішити цей недолік конфіденційності та дозволяють функціонувати Інтернет-банкінгів. Без шару захищених сокетів (SSL) транзакції кредитними картками на популярних веб-сайтах були б дуже незручними або небезпечними.

Усі ці приватні дані захищені криптографічними алгоритмами, які, ймовірно, захищені. В даний час криптосистеми, для яких можна довести, що вони захищені, є занадто непростими для практичного використання. Тож замість математичного підтвердження безпеки використовуються практично захищені криптосистеми. Це значне, що, можливо, існують можливості для злому цього шифру, але їх ще ніхто не зміг здійснити. Звичайно, є також криптосистеми, які взагалі не захищені. Це може бути пов'язано з реалізацією, розміром ключа або просто криптоаналітичними недоліками у самому шифрі.

Дорогі мережеві системи виявлення вторгнень, розроблені для сніффінгу мережевого трафіку для виявлення атак, марні, якщо зловмисник використовує зашифрований канал зв'язку..

1.7.1 Симетричні алгоритми

Симетричні шифри - це криптосистеми, які використовують один і той же ключ для шифрування та дешифрування повідомлень. Процес шифрування та дешифрування, як правило, швидше, ніж при асиметричному шифруванні, але розподіл ключів може бути важким. Ці шифри, як правило, або блокові шифри, або потокові шифри. Блоковий шифр працює на блоки фіксованого розміру, як правило, 64 або 128 біт. Один і той же блок простого тексту завжди буде зашифрований до того ж блоку шифротексту, використовуючи той самий ключ.

DES, Blowfish та AES - блокові шифри. Поточкові шифри генерують потік псевдовипадкових бітів, як правило, або один біт або байт за раз. Це називається ключовим потоком, і це XORється з простим текстом. Це корисно для шифрування безперервних потоків даних. RC4 і LSFR - приклади популярних поточкових шифрів. DES та AES - обидва популярні блокові шифри. Багато зусиль іде на побудову блокових шифрів, щоб зробити їх стійкими до відомих криптоаналітичних атак.

1.7.1.1 Advanced Encryption Standard (AES)

Найпопулярнішим і широко прийнятим симетричним алгоритмом шифрування, з яким, можливо, зустрічаються нині, є Розширений стандарт шифрування AES. Він виявляється принаймні в шість разів швидше, ніж TDES.

Потрібно було замінити DES, оскільки його розмір ключа був занадто малим. Зі збільшенням обчислювальної потужності він вважався вразливим проти «брут-форсе» атак. TDES був розроблений для подолання цього недоліку, але він виявився повільним.

Особливості AES такі:

- Симетричний ключ симетричного блок-шифру
- 128-бітні дані, 128/192/256-бітні ключі
- Більш стійкий і швидше, ніж Triple-DES
- Існує повна специфікація

AES – ітеративний шифр. Він заснований на "мережі заміщення - перестановка". Він складається з серії пов'язаних операцій, деякі з яких передбачають заміну вхідних даних конкретними вихідними даними (підмінами), а інші передбачають переміщення бітів (перестановки).

AES виконує всі свої обчислення на байтах, а не на бітах. Отже, AES розглядає 128 біт блоку простого тексту як 16 байт. Ці 16 байт розташовані в чотири стовпці і чотири рядки для обробки.

На відміну від DES, кількість раундів в AES є змінною і залежить від довжини ключа. AES використовує 10 раундів для 128-бітних ключів, 12 раундів для 192-бітних ключів і 14 раундів для 256-бітних ключів. Кожна з цих ітерацій

використовує різний 128-бітний ключ для кожного раунда, який обчислюється з оригінального ключа AES (рис 1.8).

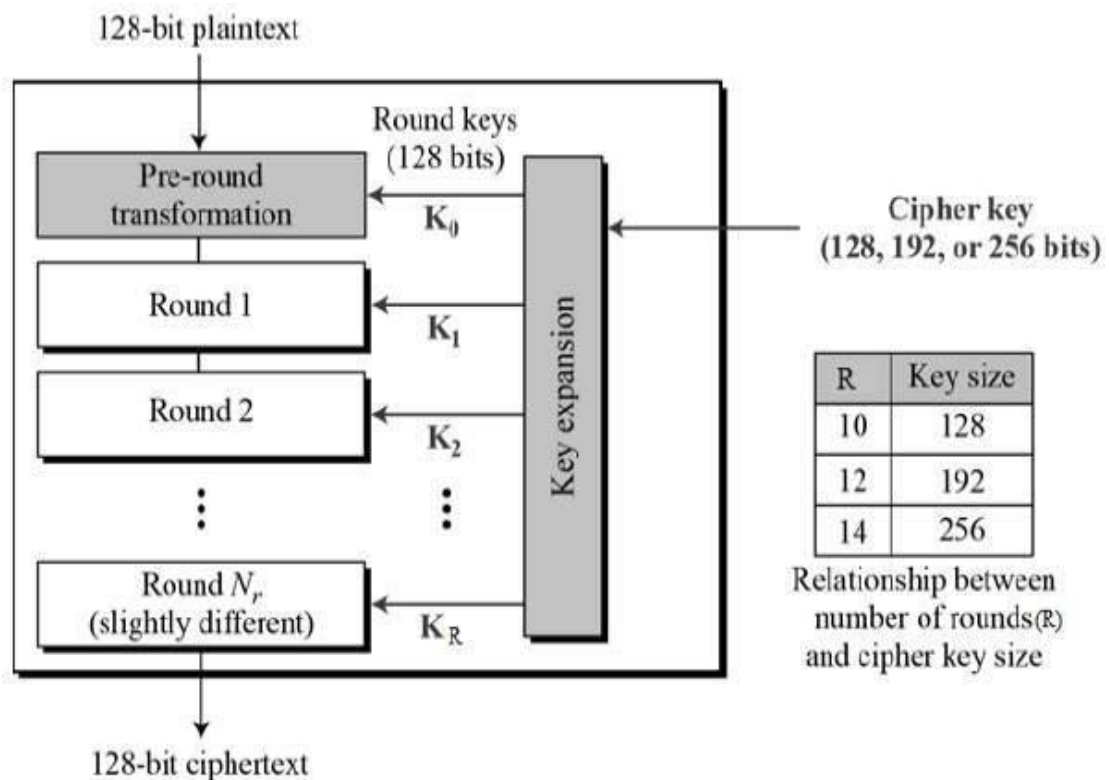


Рисунок 1.8 – Структура AES шифрування

Кожен раунд складається з чотирьох підпроцесів (рис. 1.9):

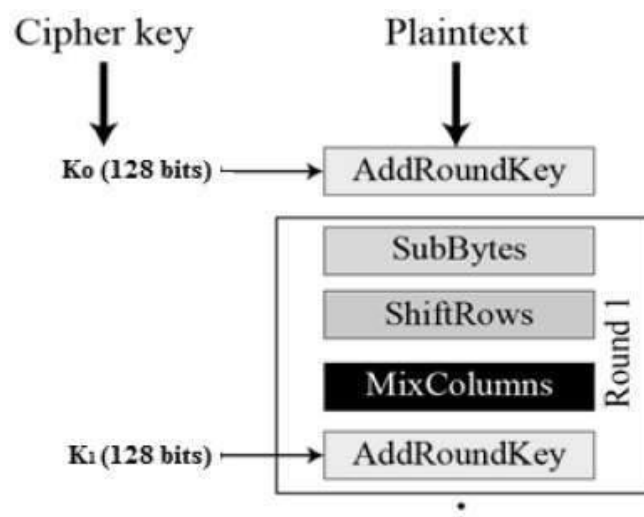


Рисунок 1.9 – Процес шифрування на кожному раунді

1. Byte substitution - 16 вхідних байтів нелінійно підміняються використовуючи таблицю заміни (S-box). Результат - в матриці з чотирьох рядків і чотирьох стовпців.

2. Shift rows - кожен з чотирьох рядків матриці зміщується вліво. Будь-які записи, які "відпадають", знову вставляються в правій частині рядка. Зсув здійснюється наступним чином -

- Перший ряд не зміщується.
- Другий ряд зміщений на одну (байту) позицію вліво.
- Третій ряд зміщений на дві позиції вліво.
- Четвертий ряд зміщується на три положення вліво.

В результаті виходить нова матриця, що складається з тих же 16 байт, але зміщених відносно один одного.

3. Mix columns - кожен стовпець із чотирьох байтів тепер перетворюється за допомогою спеціальної математичної функції. Ця функція приймає на вхід чотири байти одного стовпця і вертає чотири абсолютно нові байти, які замінюють вихідний стовпець. Результат - ще одна нова матриця, що складається з 16 нових байтів. Слід зазначити, що цей крок не виконується на останньому раунді.

4. Add round key - 16 байт матриці XORється з 128-бітним раундним ключем. На останньому раунді отримується шифротекст. В іншому випадку отримані 128 біт інтерпретуються як 16 байт і починається наступний раунд.

Процес дешифрування шифротексту AES аналогічний процесу шифрування у зворотному порядку. Кожен раунд складається з чотирьох процесів, що проводяться у зворотному порядку -

1. Add round key
2. Mix columns
3. Shift rows
4. Byte substitution

Оскільки підпроцеси в кожному раунді відбуваються у зворотному порядку, алгоритми шифрування та дешифрування повинні бути реалізовані окремо, хоча вони дуже тісно пов'язані.

1.7.2 Асиметричні алгоритми

Асиметричні шифри використовують два ключі: відкритий ключ та приватний ключ. Відкритий ключ оприлюднюється, тоді як приватний ключ зберігається приватним. Будь-яке повідомлення, зашифроване відкритим ключем, може бути розшифроване лише приватним ключем. Це усуває проблему розподілу ключів - відкриті ключі є загальнодоступними, і за допомогою відкритого ключа повідомлення може бути зашифровано для відповідного приватного ключа. Однак асиметричні шифри, як правило, повільніше, ніж симетричні шифри.

1.7.2.1 Rivest, Shamir и Adleman (RSA)

Криптосистема RSA - це найбільш широко використовуваний алгоритм криптографії з відкритим ключем у світі. Він може використовуватися для шифрування повідомлення без необхідності обмінюватися секретним ключем окремо.

Алгоритм RSA може використовуватися як для шифрування відкритим ключем, так і для цифрових підписів. Його безпека ґрунтується на складності факторизації великих цілих чисел.

Сторона А може надіслати зашифроване повідомлення стороні В без попереднього обміну секретними ключами. Просто використовується відкритий ключ В для шифрування повідомлення, а В розшифровує його за допомогою приватного ключа, який знає лише він. RSA також може використовуватися для підписання повідомлення, тому А може підписати повідомлення за допомогою свого приватного ключа, а В може підтвердити це за допомогою відкритого ключа А.

Оригінальний алгоритм:

1. Створіть два великих випадкових простих числа, p і q , приблизно однакового розміру, щоб їх добуток $n = pq$ був потрібної довжини біта, наприклад 1024 біт.
2. Обчисліть $n = pq$ і $\phi = (p - 1)(q - 1)$.
3. Виберіть ціле число e , $1 < e < \phi$, таке, що $\gcd(e, \phi) = 1$.
4. Обчисліть таємну експоненту d , $1 < d < \phi$, такий, що $ed \equiv 1 \pmod{\phi}$.
5. Відкритим ключем є (n, e) і приватним ключем (d, p, q) .

Практичний алгоритм генерації ключів:

$e \in 3, 5, 17, 257, 65537$

repeat

$p \leftarrow \text{genprime}(k / 2)$

until $(p \bmod e) \neq 1$

repeat

$q \leftarrow \text{genprime}(k - k / 2)$

until $(q \bmod e) \neq 1$

$N \leftarrow pq$

$L \leftarrow (p-1)(q-1)$

$d \leftarrow \text{modinv}(e, L)$

return (N, e, d)

Типова довжина бітів становить $k = 1024, 2048, 3072, 4096, \dots$, зі збільшенням обчислювальних витрат для більших значень.

Алгоритм шифрування:

1. Отримує відкритий ключ одержувача $B(n, e)$.
2. Представляє повідомлення як натуральне ціле m , де $1 < m < n$.
3. Обчислює зашифрований текст $c = m^e \bmod n$.
4. Відсилає зашифрований текст c до B .

Алгоритм дешифрування:

1. Використовуючи приватний ключ (n, d) для обчислення $m = c^d \bmod n$.
2. Витягує текст з m , який представляє повідомлення.

1.7.3 Змішане шифрування

Криптографічні системи використовують переваги двох основних криптосистем: симетричного і асиметричного алгоритмів. На цьому принципі побудовані такі протоколи, як PGP і TLS.

Основний недолік асиметричної криптографії полягає в низькій швидкості через складні обчислення, необхідних алгоритмам, в той час як симетричні алгоритми показує високу швидкість роботи. Однак симетричні алгоритми мають один істотний недолік - її використання передбачає наявність захищеного каналу для

передачі ключів. Для подолання цього недоліку вдаються до асиметричних алгоритмів, які використовують пару ключів: відкритий і закритий.

1.7.4 TLS

Transport Layer Security (TLS) - це один з найважливіших і широко використовуваних протоколів безпеки (рис 1.10). Він захищає значну частину даних, які передаються в Інтернеті. Він найбільш широко використовується для захисту даних, які пересуваються між веб-браузером та веб-сайтом через HTTPS, але він також може бути використаний для захисту електронної пошти та безлічі інших протоколів.

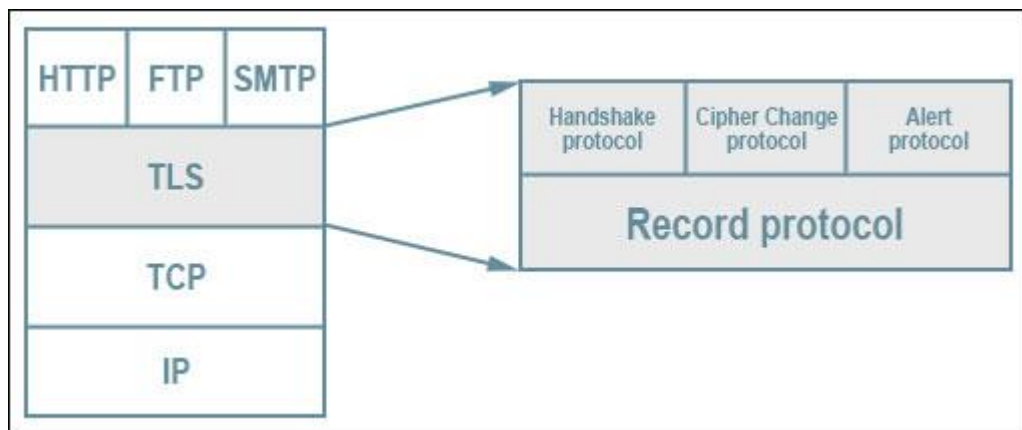


Рис. 1.10 – Стек TLS протокола

Процес встановлення захищеного зв'язку між клієнтом та сервером називається «рукоштовання» (рис. 1.11).

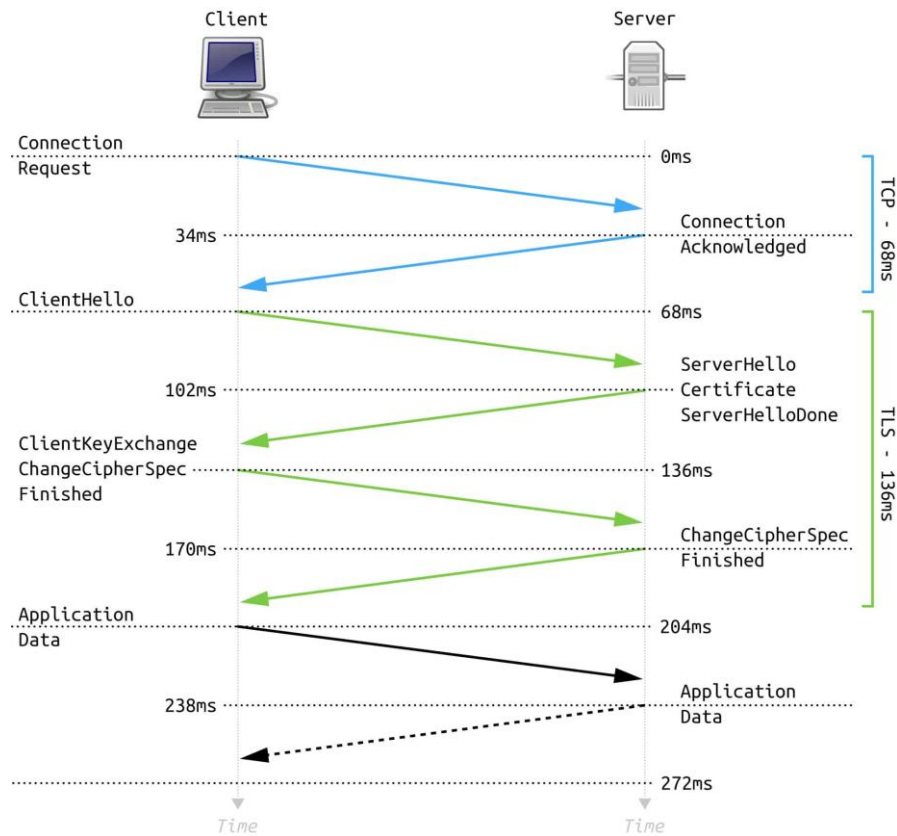


Рис. 1.11 —«рукоостискання» TLS

Після встановлення захищеного з'єднання рукоостисканням TLS протокол програми використовується для захисту передачі даних. Він може бути налаштований на використання широкого спектру алгоритмів для відповідності різним сценаріям.

Цілісність повідомлень можна перевірити за допомогою багатьох різних алгоритмів:

- HMAC-MD5
- HMAC-SHA1
- HMAC-SHA2
- AEAD

TLS використовує шифрування симетричного ключа для забезпечення конфіденційності даних, які він передає. На відміну від шифрування відкритим ключем, лише один ключ використовується як в процесах шифрування, так і в дешифруванні. Після того, як дані будуть зашифровані за допомогою алгоритму, вони будуть відображатися як шифротекст. Поки використовується відповідний

алгоритм, зломисники не зможуть отримати доступ до фактичних даних, навіть якщо вони перехоплять їх.

TLS може використовувати безліч різних алгоритмів, таких як Camellia або ARIA, хоча найпопулярнішим є AES.

Як і більшість протоколів, TLS зазнала низки вразливих і теоретичних атак проти різних його реалізацій. Незважаючи на це, останні версії вважаються безпечними для практичних цілей.

Минулі версії, такі як SSL 2.0 та 3.0 (і TLS 1.0, який по суті такий же, як і SSL 3.0), мають численні недоліки безпеки, але оскільки вони є старими та застарілими протоколами, ми не будемо вникати в деталі.

Нові версії TLS мають численні оновлення, які роблять його менш вразливим, ніж SSL. Незважаючи на це, у протоколі все ще були такі проблеми безпеки:

1. Renegotiation attacks
2. BEAST
3. Timing attacks
4. CRIME & BREACH
5. Downgrade attacks
6. Heartbleed
7. DROWN
8. Unholy PAC

Існує багато питань безпеки, але реальність полягає в тому, що більшість з них є досить незначними. По мірі прогресування технологій виявляються вразливості та розробляються нові атаки, тому у TLS буде виникати більше проблем із безпекою. Незважаючи на це, зараз і в досяжному майбутньому TLS буде одним з головних і найнадійніших інструментів, які використовуються для захисту інтернету.

1.8 Збереження даних

1.8.1 Реляційні бази даних

Реляційна база даних - це тип бази даних, який зберігає і забезпечує доступ до даних, які пов'язані один з одним. Реляційні бази даних базуються на реляційній

моделі, інтуїтивно зрозумілому, прямому способі подання даних у таблицях. У реляційній базі даних кожен рядок таблиці - це запис із унікальним ідентифікатором, який називається ключем. Стовпці таблиці містять атрибути даних, і кожен запис зазвичай має значення для кожного атрибуту, що полегшує встановлення зв'язків між точками даних.

Реляційна модель означає, що логічні структури даних - таблиці даних, представлення даних та індекси - є окремими від фізичних структур зберігання.

1.8.2 NoSQL бази даних

NoSQL - це база даних, яка забезпечує механізм зберігання та пошуку даних. Ці дані моделюються іншими засобами, ніж табличні співвідношення, що використовуються у реляційних базах даних. Такі бази даних почали існувати наприкінці 1960-х, але вони не отримали популярності до початку двадцятого століття. Бази даних NoSQL використовуються у веб-додатках у реальному часі та big data, а їх випадка використання з часом збільшується. Системи NoSQL також іноді називають «не тільки SQL», щоб підкреслити той факт, що вони можуть підтримувати SQL-подібні мови запитів.

База даних NoSQL включає простоту проектування, більш просте горизонтальне масштабування для кластерів машин і більш чіткий контроль за доступністю. Структури даних, що використовуються в базах даних NoSQL, відрізняються від тих, які використовуються за замовчуванням у реляційних базах даних, що робить деякі операції швидшими в NoSQL. Придатність бази даних NoSQL залежить від проблеми, яку вона повинна вирішити. Структури даних, що використовуються в базах даних NoSQL, іноді також розглядаються як більш гнучкі, ніж таблиці реляційних баз даних.

1.8.3 Порівняння

NoSQL бази даних мають ряд переваг, що робить цю технологію більш привабливою (таб. 1.2).

Таблиця 1.2 – Порівняння NoSQL з реляційними базами даних

Переваги	Опис
----------	------

Постійна доступність	Різні реляційні бази даних можуть демонструвати сучасну та високу доступність для транзакцій даних, хоча це набагато краще з базами даних NoSQL, які чудово демонструють постійну доступність, щоб впоратися з різними видами транзакцій даних у будь-який момент часу та у складних ситуаціях.
Низький показник затримки	Отримати дані з неструктурованих моделей даних у базі даних NoSQL легко та менш забирає багато часу. Час реакції досить швидкий і може працювати з великим потоком операцій в додатках.
Легка масштабованість	За допомогою NoSQL можна легко масштабувати базу даних для поточних та майбутніх вимог. Ці бази можуть легко обробляти розподіл даних на декількох серверах, щоб задовольнити вимоги до зростаючій кількості збережених даних. Цілком легко масштабувати бази даних NoSQL, в яких можна використовувати недороге обладнання та сервери, порівняно з реляційними базами даних, які потребують дорогих апаратних рішень для вирішення питань масштабування, що виникають із-за зростаючих кількості даних.

Можливість обробляти зміни	Структура баз даних NoSQL без схем допомагає легко впоратися зі змінами, що виникають з часом. Існує універсальний індекс для структури, значень та тексту, знайдених у даних, а отже, легко впоратися зі змінами швидко, використовуючи цю інформацію.
Підтримка декількох структур даних	Незалежно від структури даних, що підтримується програмою, база даних NoSQL може задовольнити свої вимоги до даних. Ці неструктуровані бази даних цілком можуть задовольнити всі види потреб у додатках. Прості бінарні значення, рядки, списки складних ієрархій parent-child, зберігання графів тощо, можуть бути оброблені базою даних NoSQL.

Реляційні бази працюють швидше на вибірку даних, але NoSQL швидше для оновлення та додавання нових даних (рис. 1.12).

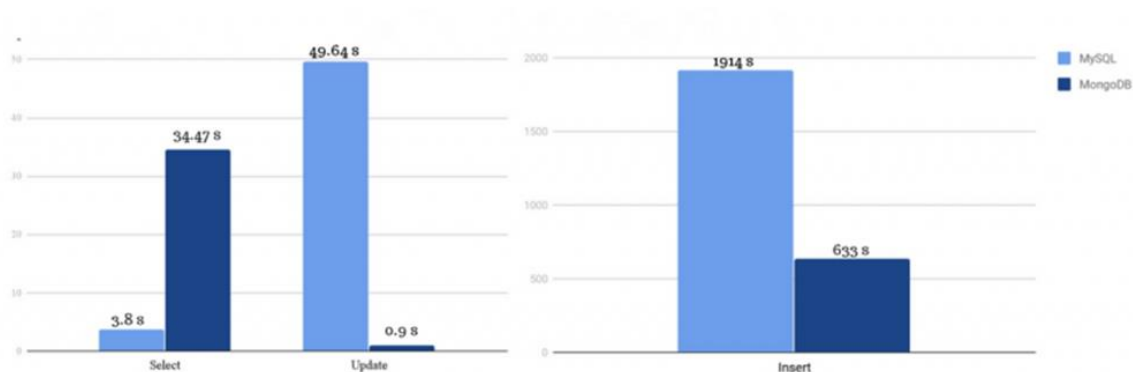


Рис. 1.12 – Порівняння швидкості вибірки, оновлення та вставлення для 1000000 записів

2 ФОРМУВАННЯ ВИМОГ ДО СПІЛЬНОГО РЕДАГУВАННЯ ДОКУМЕНТІВ У РЕЖИМІ РЕАЛЬНОГО ЧАСУ У ВЕБ-ДОДАТКУ, РОЗРОБКА ЇЇ КОНЦЕПЦІЇ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Формування вимог до спільного редагування документів у режимі реального часу у веб-додатку

2.1.1 Характеристики об'єкту автоматизації

Сучасні методи освіти вимагають нових підходів взаємодії студентів з викладачами. Вже з'явилась велика кількість онлайн-курсів по різних напрямкам. Взаємодія досягається багатьма засобами від VoIP-телефонії (найчастіше Skype), до власних платформ, які зазвичай сильно відрізняються одна від одної, але такі методи не дозволяють отримати динамічну взаємодію між студентом та викладачем. Буває легше намалювати нову схему, яка більш зрозуміла для цього конкретного студента, ніж використовувати уніфіковані малюнки. Також сучасні курси намагаються знайти індивідуальний підхід під кожного студента, а тому єдина платформа для цього вже не підходить. Викладач повинен мати змогу додавати окрім тексту також малюнки, мультимедійну інформацію (зображення, аудіо та відео) і таким чином підлаштовуватись під студента. На сьогоднішній день існує майже уся інформація в інтернеті, тому треба надати можливість вставляти її безпосередньо у сам документ. Вся інформація якою обмінюються студент з викладачем повинна бути надійно захищено, тому що є інтелектуальною власністю.

Створення єдиної платформи спільного редагування документів допоможе підлаштовуватися індивідуально під кожного студента та бути більш конкурентоспроможні аніж ті, які використовуються уніфіковану платформу.

2.1.2 Існуючий стан автоматизації

Існуючі системи використовують застарілі технології, такі як Flash. На сьогоднішній день ця технологія є незахищеною і по замовчуванню виключена в сучасних браузерів, що робить її ще й дуже не зручною. Системи побудовані на

цій технології все застарілі і не відповідають ні функціональним потребам, ні зручностію користування. Інші системи реалізовані не для потреб інтерактивної взаємодії між студентом та вчителем в онлайн-курсі. Тому сучасною задачею є створення такої інформаційної система, яка б вирішувала ці проблеми.

2.1.3 Обґрунтування необхідності модернізації ІС

Перелічені вище недоліки суттєво ускладнюють процес викладання та індивідуалізації процесу навчання, а також вимагають більше емоціональних зусиль від викладача та концентрації студента.

Зручність використання єдиної платформи для взаємного редагування документів полягає в тому, що є можливість повністю змінювати наповнення документу, що не дозволяють роботи звичайні платформи для онлайн-курсів. Є можливість додавати різноманітний матеріал, як вчителем, так і студентом за допомогою лише однієї платформи. Що значно полегшує порозуміння питань та потреб студента.

2.1.4 Цілі, критерії та обмеження впровадження модернізації ІС

Створення модернізованої системи має на меті вирішення наступних завдань:

- створення платформи для спільного редагування документів;
- використання новітніх технологій для забезпечення безпечного обміну та збереження даних;
- легке додавання нових інструментів для більшої інтерактивності;
- легке горизонтальне та вертикальне масштабування.

Критерії для якісної оцінки:

- дружність до користувача, інтуїтивно зрозумілий інтерфейс;
- оптимальна завантаженість комп'ютерної мережі та обчислювальних ресурсів;
- швидкий обмін змінами між клієнтами.

Обмеження модернізації:

- складні алгоритми вирішування проблем, пов'язаних з пересічними змінами.

2.1.5 Задачі і функції створюваної системи

Після створення розроблювана система повинна дозволяти функції, описані нижче.

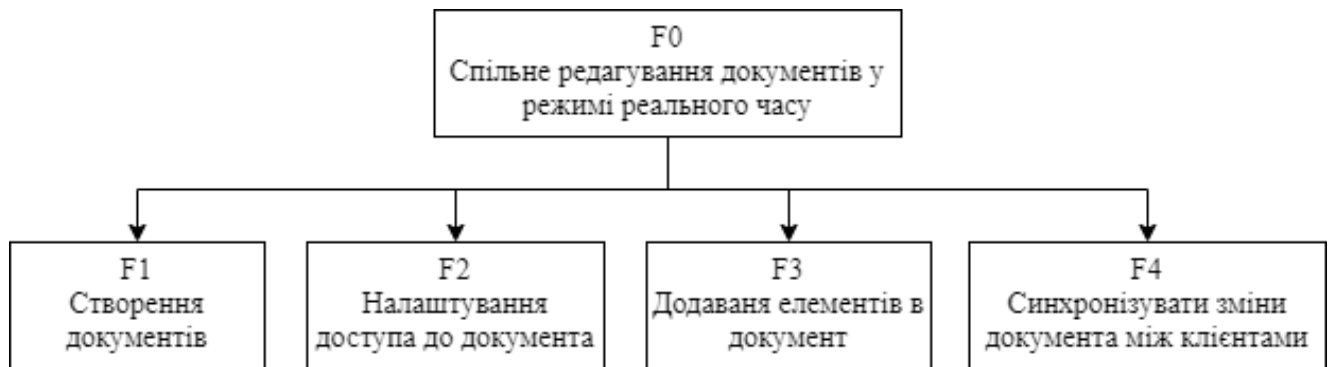


Рисунок 2.1 – Структура задачі 1 у реалізації ІС, що пропонується

Задача 1 – спільне редагування документів у режимі реального часу:

F1) Надавати користувачам змогу створювати документи.

F2) Встановлювати політику доступу до створеного документа.

F3) Додавати різноманітні інтерактивні елементи до документа.

F4) Синхронізувати в режимі реального часу усі зміни, які були зроблені користувачами.

2.1.6 Висновки і пропозиції

Під час виконання дослідження існуючих рішень та технологій, які використовуються в системах було знайдено знехтування захищеністю даних, а також малий функціонал систем.

Використання нових технологій вирішить проблеми з захищеністю даних та за допомогою нових підходів до написання програмного кода зробить систему більш модульну та легко масштабовану. А також використання сучасних клієнтських платформ, засобів розробки клієнтських програм та сучасних принципів побудови значно збільшить дружність інтерфейсу користувача, спросить навчання роботі системи.

2.2 Розробка концепції інформаційної системи

2.2.1 Розробка концепції

Розподілена інформаційна система являє собою комплекс апаратних та програмних засобів.

Концептуально розроблювана система має складатися з наступних компонентів:

- сервери для обробки даних;
- сервери для сховища даних;
- сервер для управління навантаженням у системі.

Усі апаратні компоненти системи повинні бути поєднані між собою високошвидкісним мережевим з'єднанням (більше за 100 Мбіт/с).

До програмних компонентів розроблюваної системи входять:

- операційні системи серверів
- серверне програмне забезпечення
- системи керування базами даних

Існує два варіанти розробки розподіленої інформаційної системи:

1. Створення системи на власних обчислювальних потужностях;
2. Створення системи на базі потужностей провайдерів хмарних обчислень.

Від вибору варіанта залежить легкість впровадження та вартість використання системи.

2.2.2 Обґрунтування варіанту

Основні критерії оцінки ефективності розподіленої інформаційної системи спільного редагування документів у режимі реального часу – є швидкість, відмовостійкість, пропускна здатність серверів та сховищ даних.

Основні переваги й недоліки двох вище приведених варіантів розгортання інформаційної системи наведені у таблиці 2.1.

Таблиця 2.1 – Порівняльна таблиця переваг та недоліків варіантів розгортання системи.

№ варіанту	Переваги варіанту	Недоліки варіанту
1	- гнучкість системи; - високий рівень безпеки та конфіденційності даних при правильному налаштуванні; - відсутність окремих витрат на використання системи.	- необхідність використання власних апаратних засобів; - порівняно низька швидкість впровадження системи; - дорогі кваліфіковані кадри.
2	- висока швидкість створення системи; - відсутність необхідності апаратного забезпечення та апаратної підтримки; - гнучкі тарифи на використання системи; - не треба наймати працівників для обслуговування та налаштування.	- невизначений рівень конфіденційності та безпеки даних.

Найкращий варіант розгортання системи в хмару, так як на сьогоднішній день існують дуже різноманітні платформи, дуже гнучкі тарифи на використання, легкий та гнучкий в налаштуваннях, моніторингу і по необхідності легко масштабується.

2.2.3 Очікувані результати від впровадження системи

Забезпечення швидкого і відмовостійкого спільного редагування документів – основна мета. Очікується, що система дасть наступні результати:

- високу модульність системи;
- високу захищеність даних;
- безперебійний обмін змінами між користувачами.

2.2.4 Умови приймання інформаційної системи

Випробування розроблюваної системи виконується у відповідності з програмою та методикою випробувань. За результатами випробувань складається акт прийому-здачі, затверджений керівником організації випробувань і замовником. У випадку виявлення при випробуванні помилок або невідповідності результатів

роботи підсистеми технічному завданню, проводяться виправлення у двотижневий срок.

2.3 Постановка задач

Відповідно до вимог концепції розробки розподіленої інформаційної системи спільного редагування документів у режимі реального часу уточнимо постановку задачі.

У даній роботі необхідно розробити систему спільного редагування документів. Призначення якої буде ефективне та безперебійне проведення персоналізованих онлайн-курсів.

Метою створення системи є покращення інформаційної структури онлайн-курсів та універсализація інтерфейсів обміну даними.

Вимоги до системи, склад і зміст робіт зі створення системи, порядок контролю й приймання системи тощо наведені у технічному завданні на створення інформаційної системи (додаток А).

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СХЕМИ СПІЛЬНОГО РЕДАГУВАННЯ ДОКУМЕНТІВ У РЕЖИМІ РЕАЛЬНОГО ЧАСУ

3.1 Загальна архітектура

Для розробки розподіленої інформаційної системи спільного редагування документів (рис. 3.1) була вибрана трирівнева архітектура клієнт-сервер.

Перший рівень архітектури розробляється за допомогою фреймворку Vue.js та обгортки над JavaScript – Babel.js. Використання Babel.js обумовлено використанням високорівневих структур мови JavaScript та їх сумісність з застарілими браузерами.

Другий рівень архітектури розробляється, використовуючи Node.js з допомогою фреймворку Express.js.

Обмін даними у режимі реального часу буде здійснюватися через протокол WebSocket Secure. Для використання даного протоколу в Node.js існують бібліотека Socket.IO, яка є обгортка над протоколом та реалізує подія-орієнтовну модель, що значно підвищує модульність взаємодії клієнта-сервер. Використовуючи цю бібліотеку є змога зберігати та контролювати підключених клієнтів, що дозволяє обмінюватися даними у реальному часі.

Для зберігання даних використовується NoSQL база-даних MongoDB.

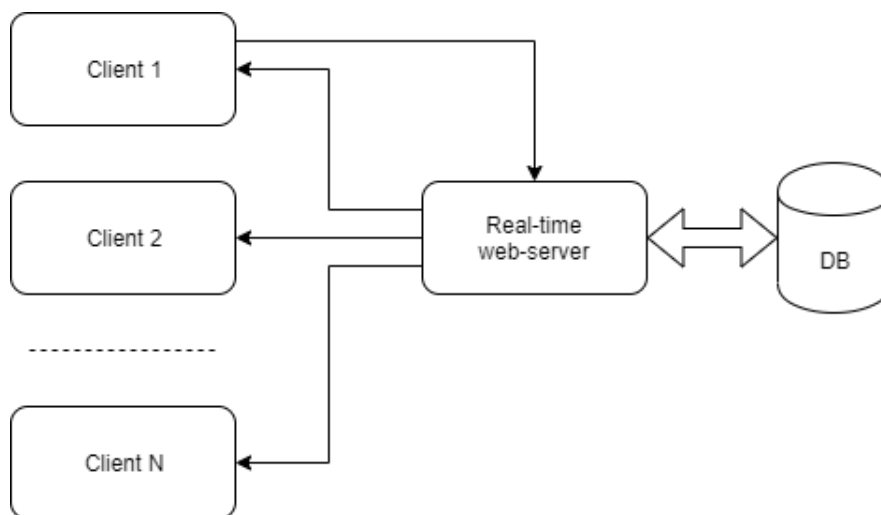


Рисунок 3.1 – Загальна схема

3.2 Обмін даними

Загальна модель взаємодії клієнта з сервером та з базою даних зображена на діаграмі послідовності (рис. 3.2).

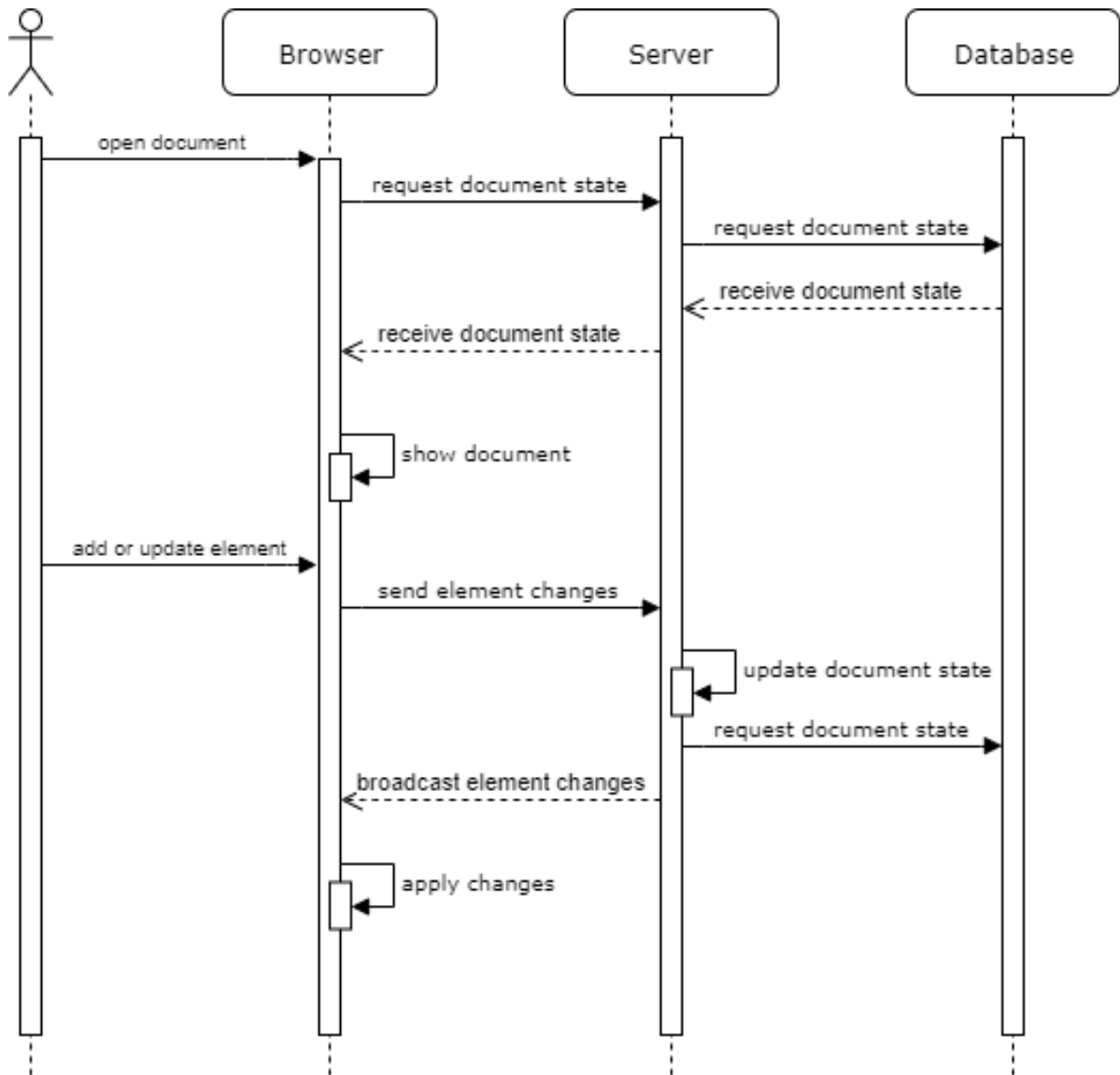


Рисунок 3.2 – Діаграма послідовності

Загальний процес взаємодії реалізують три варіанти обміну повідомленнями між клієнтом та сервером.

Перший та найбільш частіший варіант (рис. 3.3) – отримання останніх змін від сервера.

```

Sheet
{
  id: "1",
  objs:
  [
    {
      id: "e11",
      type: "DragableText",
      deltas:
      [
        {
          type: "Text",
          version: 2,
          text: "Hello, world!"
        }
      ]
    }
  ]
}

```

Рисунок 3.3 – Приклад останньої зміни

Другий варіант – отримання повного стану елемента (рис. 3.4). Він використовується якщо під час оновлення сталася помилка та неможливо відновити поточний стан елемента.

```

Sheet
{
  id: "1",
  objs:
  [
    {
      id: "e11",
      type: "DragableText",
      state:
      {
        xy:
        {
          x: 10.5,
          y: 2000
        },
        rotation: 0.0,
        version: 2,
        text: "Hello, world!"
      }
    }
  ]
}

```

Рисунок 3.4 – Приклад повного стану елемента

Третій варіант – отримання повного стану документа (рис. 3.5). Він використовується у випадку відкриття існуючого документа для завантаження збережених елементів.

```
Sheet
{
  id: "1",
  objs:
  [
    {
      id: "el1",
      type: "DragableText",
      state:
      {
        xy:
        {
          x: 10.5,
          y: 2000
        },
        rotation: 0.0,
        version: 2,
        text: "Hello, World!"
      }
    },
    {
      id: "el2",
      type: "DragableText",
      state:
      {
        xy:
        {
          x: 10.5,
          y: 2025
        },
        rotation: 0.0,
        version: 1,
        text: "Привет, мир!"
      }
    }
  ]
}
```

Рисунок 3.5 – Приклад повного стану документа

Підчас оновлення даних від клієнта на сервері відправляються запити на зберігання поточних змін (рис. 3.6) та оновлений стан всього документа. Використання змін, замість відправки усього стану документа значно пришвидшує обмін за рахунок відправлення меншого об'єму інформації.

```

Instrument
{
  id: "el1",
  type: "DragableText",
  deltas:
  [
    {
      type: "Position",
      version: 1,
      xy:
      {
        x: 10.5,
        y: 2000.0
      }
    },
    {
      type: "Text",
      version: 2,
      text: "Hello, world!"
    }
  ],
  state:
  {
    xy:
    {
      x: 10.5,
      y: 2000
    },
    rotation: 0.0,
    version: 2,
    text: "Hello, world!"
  }
}

```

Рисунок 3.6 – Приклад збережених змін та стану елемента

3.3 Шифрування даних

В протоколі WebSocket Secure(wss://) використовують TLS шифрування. Для обміну ключами шифрування – алгоритм RSA, а для подальшого обміну зашифрованою інформацією – алгоритм AES.

Шифрування даних виконується за допомогою бібліотеки Crypto, яка у свою чергу є обгортка над бібліотекою OpenSSL. Бібліотека OpenSSL широко використовується у сучасних криптографічних рішеннях.

Починаючи з 2008 року у сучасних процесорів для прискорення шифрування та дешифрування алгоритмом AES-NI з'явилися спеціальний набір команд процесора AES-NI. Використання даного набору команд активується на етапі компілювання бібліотек OpenSSL (рис. 3.7).


```

dmytro@debian: ~/Documents/Lsts
File Edit View Search Terminal Help
dmytro@debian:~/Documents/Lsts$ cat libcrypto-1_1-x64.lst | grep -w aesenc -c
450
dmytro@debian:~/Documents/Lsts$ cat libcrypto-1_1-x64.lst | grep -w aesenc
.text:000000018000D551      aesenc    xmm2, xmm1
.text:000000018000D562      aesenc    xmm3, xmm1
.text:000000018000D573      aesenc    xmm4, xmm1
.text:000000018000D578      aesenc    xmm5, xmm1
.text:000000018000D585      aesenc    xmm2, xmm0
.text:000000018000D58A      aesenc    xmm3, xmm0
.text:000000018000D58F      aesenc    xmm4, xmm0
.text:000000018000D59C      aesenc    xmm5, xmm0
.text:000000018000D5A9      aesenc    xmm2, xmm1
.text:000000018000D5AE      aesenc    xmm3, xmm1
.text:000000018000D5B3      aesenc    xmm4, xmm1
.text:000000018000D5C0      aesenc    xmm5, xmm1
.text:000000018000D5CD      aesenc    xmm2, xmm0
.text:000000018000D5D7      aesenc    xmm3, xmm0

```

Рисунок 3.7 – Використання набору команд AES-NI у бібліотеці OpenSSL

Набір команд AES-NI значно прискорює процес шифрування та дешифрування. Далі на рисунку 3.8 наведено бенчмарк з використанням та без використання AES-NI.

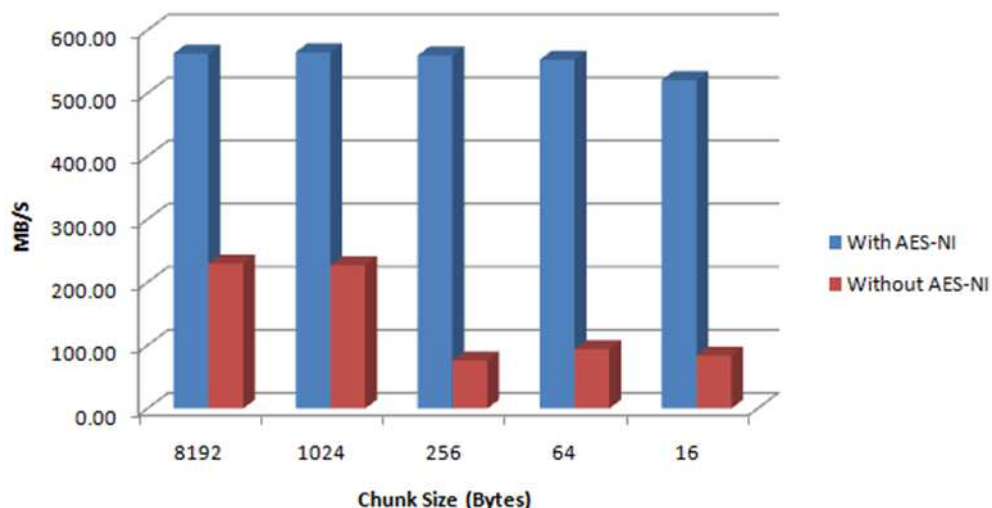


Рисунок 3.8 – Бенчмарк використання набору команд AES-NI

3.4 Збереження даних

Зберігання даних повинно бути легкомасштабоване та мати якомога менше обмежень, тому більш за все підходить MongoDB. MongoDB – це є NoSQL база даних, яка не є частиною сервера і може розташовуватись на іншій машині та взаємодіє працює за допомогою API.

Збереження усіх елементів документа здійснюється у колекцію Sheet (рис. 3.9). Кожен документа має унікальний ідентифікатор

```
Sheet
{
  id: string,
  objs: Instrument []
}
```

Рисунок 3.9 – Основна колекція Sheet

Кожен елемент має свій унікальний ідентифікатор, тип елемента(текст, графічна інформація, тощо), стан та набір змін у часі (рис. 3.10).

```
Instrument
{
  id: string,
  type: string,
  deltas: Delta [],
  state:
  {
    xy:
    {
      x: double,
      y: double
    },
    rotation: double,
    version: int,
    ...
  }
}
```

Рисунок 3.10 – Дані для кожного елемента в Instrument

Стан елемента має версію, яка змінюється кожен раз, коли ми отримуємо нові зміни з клієнта (рис 3.11). Ця версія зберігається також на клієнті і якщо нова версія зміни, яку отримав клієнт, відрізняється більш ніж на одиницю, то у такому випадку клієнт запитає повний стан елемента, щоб мати змогу відновити правильний порядок змін.

```
Delta
{
  type: string,
  version: int,
  ...
}
```

Рисунок 3.11 – Дані для кожної зміни

3.5 Взаємодії клієнта з додатком

Взаємодія клієнта з додатком відбувається за наступним алгоритмом та зображене за допомогою діаграми діяльності (рис. 3.12):

1. Вхід у систему
2. Відкриття потрібного документу по ідентифікатору документа
3. Отримання повного стану документа та його відображення
4. Робота з документом

- a. Редагування документа
- b. Отримання останніх змін
- c. Отримання стану елемента у разі помилки (описано вище)
5. Вихід з системи

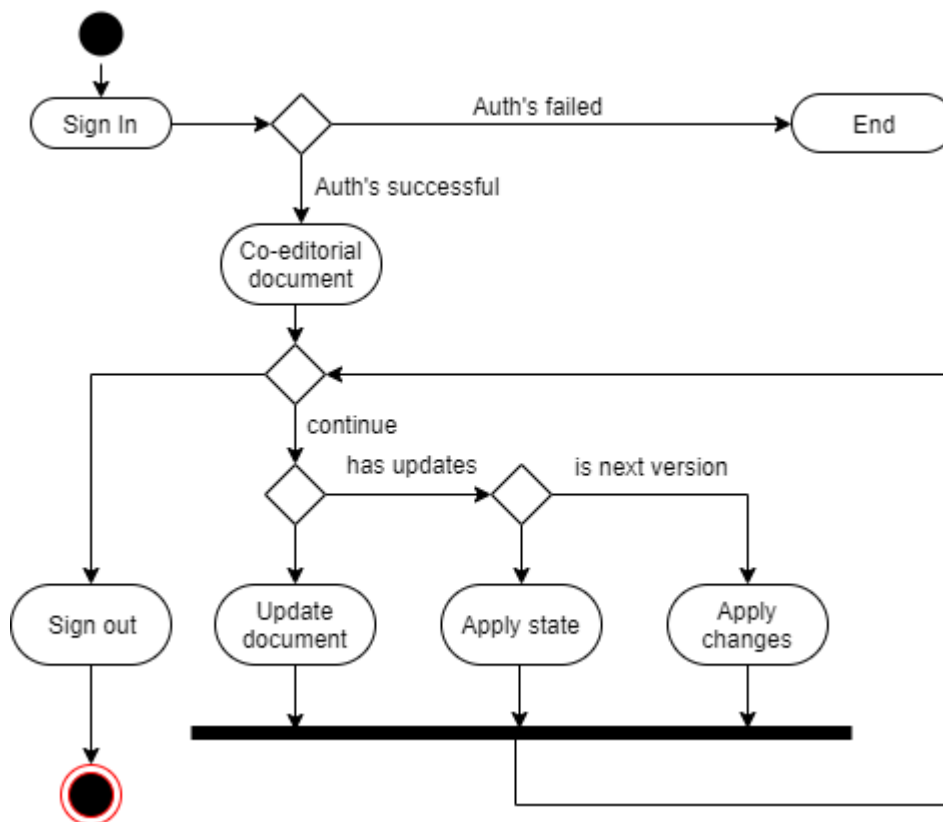


Рисунок 3.12 – Діаграма діяльності

3.6 Формат збереження та передачі даних

3.6.1 JSON

JSON (JavaScript Object Notation) формат використовується для обміну даними між клієнтом та сервером. Цей формат є асоціативний масив пара ключ-значення.

Для серіалізації у JSON, використовується команда `JSON.stringify()`, для десеріалізації та отримання JavaScript-об'єкту використовується `JSON.parse()`.

Наприклад:

```

JSON.stringify(
{
hello: "world"
}

```

```
) стae {"hello":"world"}
```

3.6.2 BSON

BSON (Binary JavaScript Object Notation) формат використовується для збереження даних у MongoDB базі даних. Цей формат є двійкова форма представлення асоціативного масива ключ-значення. BSON є ефективним з точки зору збереження даних та швидкості сканування (таб. 3.1) порівняно з використанням JSON.

Таблиця 3.1 – Серіалізація JSON-строки у BSON

JSON	BSON
{"hello":"world"}	\x16\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00

4. РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ І ВИКОРИСТАННЯ ПРОЕКТОВАНОЇ УСТАНОВКИ

4.1. Вступ до проблем

Інформаційні системи широко використовуються багатьма організаціями України різної форми власності. Ці системи розповсюджуються все більше і більше, замінюючись відповідно до вимог замовників та модернізуючись з плином часу.

На сьогоднішній день існує невелика кількість вже застарілих систем спільного редагування документів, які використовують старі технології, що робить їх незручними та незахищеними для впровадження. Безперечно, з розвитком технологій з'являються ефективніші альтернативи. Впровадження нової розподіленої інформаційної системи спільного редагування документів у режимі реального часу дозволяє користувачам цієї системи зменшити видатки на створення власної IT-інфраструктури.

Створення інформаційної системи вимагає одноразових витрат на її розробку та придбання необхідних технічних засобів, а також поточних витрат, пов'язаних з функціонуванням підсистеми. Економічний ефект від експлуатації системи визначається з урахуванням витрат на її експлуатацію. Відношення економічного ефекту до витрат на створення системи характеризує економічну ефективність капітальних вкладень. Економічні показники визначаються за діючими на момент розрахунку цінами, тарифами та ставками заробітної плати.

4.2. Розрахунок вартості створення інформаційної системи спільного редагування документів у режимі реального часу

Загальні витрати (на проектування, реалізацію та впровадження системи)

$$З = C_{п-р} + C_{тз} + C_{пз} + C_{впр},$$

де $C_{п-р}$ – витрати на проектування і реалізацію;

$C_{тз}$ – витрати на придбання або модернізацію обчислювальної техніки, периферійних пристроїв, засобів зв'язку, допоміжного устаткування та оргтехніки (з урахуванням витрат на транспортування, монтаж, налагодження і запуск), виробничо-господарського інвентарю;

$C_{пз}$ – витрати на придбання та інсталяцію програмного забезпечення (ПЗ), необхідного для розробки;

$C_{впр}$ – витрати на впровадження системи (навчання персоналу тощо). Витрати на проектування та реалізацію розраховуються за формулою

$$C_{п-р} = C_з + C_{від} + C_м + C_н,$$

де $C_з$ – заробітна плата розробників;

$C_{від}$ – відрахування на соціальні заходи (відповідно до чинного законодавства $C_{від}$ становить 47,5% від фонду оплати праці);

$C_м$ – експлуатація та амортизація, $C_м = Z_{експл} + Z_{амор}$;

$C_н$ – накладні витрати у розмірі 50-150% від витрат на оплату праці (охоплюють оплату праці управлінського персоналу з нарахуваннями; оплату службових відряджень; оплату консультаційно-інформаційних послуг; оплату ремонту і техобслуговування основних фондів, крім персональних комп'ютерів (ПК)).

1) Розробка підсистеми вимагає 5 місяців (експертна оцінка, яка базується на досвіді розробки систем такого класу). Заробітна платня розробників за місяць дорівнює 8000 грн (відповідно до сучасних розцінок та попиту на спеціалістів такого профілю). Кількість розробників – одна людина.

$$C_з = 1 * 8000 * 5 = 40000 \text{ грн.}$$

$$C_{від} = 40000 * 0,475 = 19000 \text{ грн.}$$

2) витрати на амортизацію ЕОМ, на якій виконується розробка:

$$Z_{амор} = (C_{еом} / T_{ел}) * (T_{раз} / 12),$$

де $C_{еом} = 8000$ грн – балансова вартість ЕОМ;

$T_{ел} = 5$ років – термін служби ЕОМ;

$T_{раз} = 0,417$ року – час, необхідний для розробки системи, виражений в місяцях, і тому ділиться на 12, щоб одержати число років.

$$Z_{амор} = (8000/5) * 0,417 = 666,67 \text{ (грн.)}$$

3) витрати на експлуатацію ЕОМ, на якій виконується розробка, полягають в оплаті споживаної нею електричної енергії:

$$Z_{укпл} = P_{евм} * T_{разр} * N_{рд} * N_{рч} * Э_{ст}$$

де $P_{евм} = 0,4$ кВт/год – потужність ЕОМ;

$T_{разр} = 5$ міс. – тривалість розробки;

$N_{рд} = 21$ день – число робочих днів у місяці;

$N_{рч} = 8$ годин – число годин у робочому дні;

$\mathcal{E}_{ст} = 1.68$ грн – вартість 1 кВт електроенергії.

$$Z_{експл} = 0,4 * 5 * 21 * 8 * 1,68 = 564.48 \text{ (грн)}$$

$$C_{м} = 564.48 + 666.67 = 1231.15 \text{ (грн)}$$

$$C_{н} = 40000 * 0,3 = 12000 \text{ (грн)}$$

$$C_{п-р} = 40000 + 19000 + 1231.15 + 12000 = 72231,15 \text{ (грн)}$$

Витрати на програмне забезпечення не закладаються у даний кошторис, оскільки в роботі системи планується використовувати безкоштовне ПЗ з відкритими вихідними кодами.

Витрати на апаратне забезпечення складають 23483 грн (придбання серверу та комутаційного обладнання і матеріалів). До витрат на впровадження системи можна віднести витрати вартість навчання кадрів, витрати на монтаж і налаштування мережі, вартість устаткування, вартість програмного забезпечення.

Витрати на впровадження системи приведені в таблиці 4.1.

Таблиця 4.1 – Витрати на впровадження системи

Найменування інвестицій	Сума, грн
Вартість устаткування	23483
Вартість програмного забезпечення	-
Вартість підготовки кадрів	750
Монтаж і налаштування локальної мережі	500
<i>Разом на впровадження системи</i>	<i>24733</i>

Таким чином, загальні витрати на проектування, реалізацію та впровадження системи складають:

$$Z = C_{п-р} + C_{тз} + C_{пз} + C_{впр} = 72231.15 + 24733 = 96964.15 \text{ (грн)}$$

Витрати на створення та експлуатацію системи в перший рік експлуатації

$$Z_{се} = Z + Z_{ер},$$

де $Z_{ер}$ – експлуатаційні витрати на технічне забезпечення за рік.

$$З_{ер} = A_{об} + З_{м} + З_{е},$$

де $A_{об}$ – амортизаційні відрахування на устаткування;

$З_{м}$ – витрати на основні і допоміжні матеріали;

$З_{е}$ – витрати на електроенергію.

Амортизаційні відрахування на устаткування складають 60% балансової вартості в рік (за чинним законодавством)

$$A_{об} = 24733 * 0,6 = 14839,8 \text{ грн.}$$

В масштабі підприємства річні витрати на основні та допоміжні матеріали (гнучкі диски, папір тощо) визначаються в розмірі 3% вартості основного устаткування

$$З_{м} = 24733 * 0,03 = 741,99 \text{ грн.}$$

Річний обсяг робіт ПК у нормо-годинах

$$\Phi_{м} = 253,3 * T_{с},$$

де $T_{с}$ – середнє місячне завантаження устаткування (близько 5 годин); 253,3 – середня кількість робочих днів у році.

$$\Phi_{м} = 253,3 * 5 = 1266,5 \text{ годин.}$$

Витрати на електроенергію під час 1266,5 годин роботи устаткування в рік при потужності ЕОМ 0,4 кВт/год та ціні 1,68 грн за 1 кВт електроенергії складуть

$$З_{е} = 1266,5 * 1,68 * 0,4 = 851,09 \text{ грн.}$$

Експлуатаційні витрати на технічне забезпечення за рік

$$З_{ер} = 14839,8 + 741,99 + 851,09 = 16432,88 \text{ грн.}$$

Отже, у перший рік витрати на створення та експлуатацію системи складуть

$$З_{се} = 96964,15 + 16432,88 = 113397,03 \text{ грн.}$$

4.3. Розрахунок економічної ефективності

Економічна ефективність використання розподіленої інформаційної системи спільного редагування документів у режимі реального часу:

- зменшення часу на обслуговування системи за рахунок її відмовостійкості та автоматичного масштабування;
- можливість надання користувачам обчислювальних потужностей в оренду

Це зменшення відбудеться за рахунок зменшення складності користування системою, збільшення надійності, стабільності та рівня безпеки. Визначимо, що це зменшення складе 50% від сучасного.

Однак, не піддається прямій грошовій оцінці зменшення кількості помилкових та необережних рішень, підвищення оперативності керування, поліпшення організації роботи та своєчасне отримання необхідної інформації про пріоритетність варіантів рішень тощо.

Використання даної системи дозволяє вивільнити 0,2 робочого часу. Так як з системою працює 10 робітників, то система умовно вивільнить 2 робітників.

Визначимо пряму економічну ефективність, ґрунтуючись на тому, що впровадження системи вивільнить 2 робітників.

Річна платня за рахунок вивільнення робочого часу складе:

$$\Delta C = 8000 * 2 * 12 - 16432,88 = 175567,12 \text{ (грн)}$$

Річний економічний ефект розраховується за наступною формулою:

$$\mathcal{E}_{\text{год}} = \Delta C - E_{\text{п}} * k$$

де $E_{\text{п}}$ – нормативний коефіцієнт економічної ефективності капітальних вкладень у галузь і для обчислювальної техніки приймається 0,5;

k – додаткові капітальні вкладення з урахуванням витрат на проектування, створення і функціонування системи.

$$\mathcal{E}_{\text{год}} = 175567,12 - 0,5 * 113397,03 = 118868,61 \text{ (грн)}.$$

Строк окупності підсистеми

$$T = k / \Delta C_n$$

тобто

$$T = 113397,03 / 175567,12 = 0,65 \text{ (року)}.$$

Таким чином, витрати на створення розподіленої інформаційної системи окупляться протягом 0,65 року.

4.4. Підсумки

У результаті розрахунків витрати на створення системи обліку робочого часу (проектування, реалізація і впровадження) складуть 96964,15 грн; експлуатаційні

витрати щорічно складатимуть 16432,88 грн. Загалом в перший рік витрати на модернізацію й експлуатацію складуть 113397,03 грн.

Вивільнюється 2 умовних фахівці.

Вивільнена річна зарплатня працівників складає 175567,12 грн.

Витрати на створення системи окупляться протягом 0,65 року.

За отриманими значеннями річного економічного ефекту та строку окупності можливо зробити висновок про економічну вигідність і обґрунтованість створення розподіленої інформаційної системи спільного редагування у режимі реального часу. Слід зазначити, що підвищити економічну ефективність цієї системи можна за рахунок запропонування стороннім користувачам обчислень як послуги. Таким чином можна буде скоротити строк, за який окупляться витрати на створення системи.

5. ОХОРОНА ПРАЦІ

5.1. Вступ

Охорона праці — це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Метою управління охороною праці є забезпечення безпеки, збереження здоров'я та працездатності людини в процесі праці. Ця мета досягається виконанням відповідних функцій управління, тобто комплексом взаємопов'язаних видів, що здійснюються суб'єктом управління цілеспрямовано на об'єкт управління.

Об'єктом управління в системі управління охороною праці є діяльність функціональних служб і структурних підрозділів конкретних керівників та інженерно-технічних робітників підприємств із метою забезпечення безпечних, нешкідливих та сприятливих умов праці на робочих місцях, виробничих ланках. У цехах і на підприємстві в цілому.

Мета впровадження системи управління охороною праці (УОП) — це всебічне сприяння виконанню вимог, які повністю ліквідують, нейтралізують або знижують до допустимих норм вплив на працюючих небезпечних та шкідливих факторів виробничого середовища, забезпечують усунення джерел небезпеки, ізолювання від них персоналу, використання засобів, що усувають небезпечні ситуації та підвищують технічну безпеку, створюють надійні санітарно-гігієнічні та ергономічні умови. УОП передбачає встановлення конкретних кількісних показників діяльності виробничих підрозділів, підтримування котрих в заданих межах забезпечує досягнення основної мети щодо організації безпечних та нешкідливих умов праці.

Власник зобов'язаний створити в кожному структурному підрозділі і на робочому місці умови праці відповідно до вимог нормативних актів, а також забезпечити дотримання прав працівників, гарантованих законодавством про охорону праці.

Вирішення задач управління охороною праці має бути забезпечене взаємодією усіх структурних підрозділів, служб і фахівців, які визначаються керівником

підприємства. Функції структурних підрозділів і служб, посадові обов'язки керівних та інженерно-технічних працівників щодо виконання задач управління охороною праці встановлюються на місцях на всіх рівнях, виходячи з структури, штатів і конкретних умов діяльності установи (підприємства).

5.2. Аналіз шкідливих та небезпечних факторів у відділі розробки програмного забезпечення

Під час роботи на виробництві на людину можуть впливати один, або низка небезпечних та шкідливих виробничих факторів. Безпека того чи іншого технологічного процесу може бути визначена за їх кількістю і за ступенем небезпеки кожного з них зокрема. Безпека праці на виробництві визначається ступенем безпеки окремих технологічних процесів.

Небезпечні й шкідливі виробничі фактори стандартом ГОСТ 12.0.003-74 поділяються на фізичні, хімічні, біологічні й психофізіологічні. Останні за характером впливу на людину підрозділяються на фізичні й нервово-психічні перевантаження, а інші - на конкретні небезпечні й шкідливі виробничі фактори. В процесі роботи на підприємстві на працівника можуть впливати такі небезпечні й шкідливі виробничі фактори:

- машини, що рухаються, автотранспорт і механізми;
- рухомі незахищені елементи механізмів, машин і виробничого обладнання;
- падаючі вироби техніки, інструмент і матеріали під час роботи;
- ударна хвиля (вибух посудини, що працює під тиском пари рідини);
- струмені газів і рідин, що стікають, із посудин і трубопроводів під тиском;
- підвищене ковзання (через зледеніння, зволоження й замаслювання поверхонь, по яких переміщується робочий персонал);
- підвищені запыленість й загазованість повітря;
- підвищена чи знижена температура поверхонь техніки, обладнання й матеріалів;
- підвищена чи знижена температура, вологість і рухомість повітря;
- підвищений рівень шуму, вібрації, ультра- та інфразвуку;

- підвищена напруга в електричному ланцюзі, замикання якого може відбутися через тіло людини;
- підвищений рівень статичної електрики;
- гострі кромки, задирки й шорсткість на поверхнях обладнання й інструментів;
- відсутність чи нестача природного світла;
- недостатня освітленість робочої зони;
- знижена контрастність об'єктів в порівнянні з фоном;
- пряма блискість (прожекторне освітлення територій виробництв, світло фар автотранспорту) і відбита блискість (від розлитої води й інших рідин на поверхні територій виробництв);
- підвищена пульсація світлового потоку;
- підвищений рівень ультрафіолетової й інфрачервоної радіації;
- хімічні речовини (токсичні, подразнюючі, сенсibiliзуючі, канцерогенні, мутагенні, що впливають на репродуктивну функцію людини);
- хімічні речовини , що проникають в організм через органи дихання, шлунково-кишковий тракт, шкірні покриви і слизові оболонки;
- патогенні мікроорганізми (бактерії, віруси, гриби, найпростіші) і продукти їхньої життєдіяльності;
- перевантаження (статичні й динамічні) і нервово-психічні чинники (емоційні перевантаження, перенапруга аналізаторів, розумова перенапруга, монотонність праці).

Використання персональних електронно-обчислювальних машин повинно здійснюватися у відповідності з санітарними нормами і правилами СанПіН 2.2.2/2.4.1340-03 «Гігієнічні вимоги до персональних електронно-обчислювальних машин та організації роботи».

Небезпечні і шкідливі виробничі чинники:

- підвищена температура поверхонь ПК; □ підвищена або знижена температура повітря робочої зони;
- виділення в повітря робочої зони ряду хімічних речовин;
- підвищена або знижена вологість повітря;
- підвищений або знижений рівень негативних і позитивних аероіонів;

- підвищене значення напруги в електричному ланцюзі, замикання;
- підвищений рівень статичної електрики;
- підвищений рівень електромагнітних випромінювань;
- підвищена напруженість електричного поля;
- відсутність або недолік природного світла;
- недостатня штучна освітленість робочої зони;
- підвищена яскравість світла;
- підвищена контрастність;
- пряма і відбита блескость;
- зорова напруга;
- монотонність трудового процесу;
- нервово-емоційні перевантаження

До фізичних шкідливих і небезпечних чинників відносяться: підвищені рівні електромагнітного, рентгенівського, ультрафіолетового і інфрачервоного випромінювання; підвищений рівень статичної електрики і запиленої повітря робочої зони; підвищений вміст позитивних аеронів і понижений вміст негативних аеройонів в повітрі робочої зони; підвищений рівень блескості і засліпленої; нерівномірність розподілу яскравості в полі зору; підвищене значення напруги в електричному ланцюзі, замикання якого може статися через тіло людини.

Хімічні шкідливі і небезпечні чинники наступні: підвищений вміст в повітрі робочої зони двоокису вуглецю, озону, аміаку, фенолу і формальдегіду.

Психофізіологічні шкідливі і небезпечні чинники: напруга зору і уваги; інтелектуальні, емоційні і тривалі статичні навантаження; монотонність праці; великий об'єм інформації, що обробляється в одиницю часу; нераціональна організація робочого місця.

Типовими відчуттями, які відчують до кінця робочого дня оператори ПЕОМ, є: головний біль, різь в очах, що тягнуть болі в м'язах шиї, рук і спини, сверблячка шкіри обличчя і т.д. Пережиті день за днем, ці нездужання призводять до мігрені, часткової втрати зору, сколіозу, шкірним запаленням та іншим небажаним явищам. Розвиваються нездужання не тільки знижують працездатність, але і підривають здоров'я людей. На стан здоров'я оператора ЕОМ можуть впливати й

такі шкідливі фактори, як тривалий незмінне положення тіла, що викликає м'язово-скелетні порушення; постійне напруження очей; вплив радіації; вплив електростатичних і електромагнітних полів та інше. Тривала і інтенсивна робота на комп'ютері може стати джерелом важких професійних захворювань, таких, як травма навантажень (ТПН), що повторюються, що є нездужаннями, що поступово накопичуються, перехідними в захворювання нервів, м'язів і сухожиль руки.

До професійних захворювань, пов'язаних з ТПН, відносяться:

- тендовагініт — запалення сухожиль кисті, зап'ястя, плеча;
- тендосиновіт — запалення синовіальної оболонки сухожильної підстави кисті і зап'ястя;
- синдром зап'ястного каналу (СЗК) — викликається утиском серединного нерва в зап'ястному каналі. Травма, що накопичується, викликає утворення продуктів розпаду в області зап'ястного каналу, внаслідок чого спочатку виникає набряк, а потім СЗК.

5.3. Розрахунок систем штучного освітлення приміщення відділу розробки програмного забезпечення

Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначення необхідної кількості освітлювачів, їх типу і розміщення. Необхідно виконати розрахунок загального освітлення виробничого приміщення. Довжина $A = 12$ м, ширина $B = 8$ м, висота $H = 3,2$ м. Висота робочої поверхні $h = 1$ м. Для освітлення використовується світильник типу ППД2-500. Мінімальна освітленість лампи розжарювання по нормах $E_{min}=100$ лк. Коефіцієнт відбиття стелі $S_n=50\%$, стін $S_c=30\%$, робочої поверхні $S_p=10\%$. Напруга мережі 220 В.

Для розрахунку використаємо метод світлового потоку.

Для визначення кількості світильників визначимо світловий потік однієї лампи, який падає на поверхню по формулі:

$$F = \frac{E_{min} * K_3 * S * z}{N * \eta} \quad (5.1)$$

де

F – світловий потік, що розраховується, лм;

$E_{\min}$ – нормована мінімальна освітленість, лк;

S – площа освітлюваного приміщення (у нашому випадку $S = 96 \text{ м}^2$);

z – відношення середньої освітленості до мінімального (звичайно приймається рівним 1,1...1,2, нехай $z = 1,1$);

K_z – коефіцієнт запасу, що враховує зменшення світлового потоку лампи результаті забруднення світильників у процесі експлуатації (його значення залежить від типу приміщення і характеру проведених у ньому робіт і в нашому випадку $K = 1,5$);

N – необхідна кількість світильників, яка розраховується по формулі 5.2:

$$N = \frac{S}{L^2} = \frac{S}{(1.5 * h)^2} = \frac{96}{(1.5 * 3)^2} \approx 5 \quad (5.2)$$

Кількість світильників будемо вважати рівною 6 (2 ряди по 3 світильники).

η – коефіцієнт використання, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в частках одиниці; залежить від характеристик світильника, розмірів приміщення, фарбування стін, стелі і робочої поверхні, які характеризуються відповідними коефіцієнтами відображення). Значення коефіцієнтів приведені вище. Значення η визначимо по таблиці коефіцієнтів використання різних світильників. Для цього обчислимо індекс приміщення по формулі:

$$i = \frac{S}{h * (A + B)} \quad (5.3)$$

де

S – площа приміщення, $S = 96 \text{ м}^2$;

h – розрахункова висота підвісу, $h = 3 \text{ м}$;

A – довжина приміщення, $A = 12 \text{ м}$;

B – ширина приміщення, $B = 8 \text{ м}$.

Підставивши у формулу 5.3 значення отримаємо: $i = \frac{96}{3 * (12 + 8)} = 1,6$

Знаючи індекс приміщення знаходимо $\eta = 0,59$

Підставимо всі значення у формулу 5.1 для визначення світового потоку F :

$$F = \frac{100 * 1.5 * 96 * 1.1}{6 * 0.59} \approx 4474 \text{ (лм)};$$

Для освітлення вибираємо лампу типу ЛТБ 80-4, потужністю 80 Вт, світловий потік якої $F = 4440$ лм, найбільш близький до розрахункового.

При цьому фактична освітленість:

$$F = E_{min} * \frac{F_n}{F_p} = 100 * \frac{4440}{4474} \approx 99 \text{ (лм)}$$

Загальна потужність $P_{\text{общ}} = P_{\text{л}} * N = 80 * 6 = 480$ (Вт).

5.4. Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів

З метою запобігання або зменшення впливу на працюючих шкідливих і небезпечних виробничих чинників застосовують засоби колективного та індивідуального захисту.

Чинне законодавство з охорони праці передбачає, що працівники певних професій на роботах зі шкідливими та небезпечними умовами праці, а також роботах, пов'язаних із забрудненням, або таких, що здійснюються у несприятливих метеорологічних умовах, повинні безоплатно (за встановленими нормами) забезпечуватися спеціальним одягом, спеціальним взуттям, іншими засобами індивідуального захисту (далі - ЗІЗ) та спецхарчуванням. Обов'язок щодо дотримання законодавства з охорони праці і забезпечення працівників ЗІЗ покладено на роботодавців.

Засоби захисту працюючих за призначенням поділяються на дві категорії: засоби колективного захисту і засоби індивідуального захисту.

Засоби колективного захисту призначені для:

- нормалізації повітряного середовища виробничих приміщень і робочих місць (вентиляція, кондиціонування, опалення, автоматичний контроль і сигналізація);
- нормалізації освітлення виробничих приміщень і робочих місць (джерела світла, освітлювальні прилади, світлозахисне обладнання, світлофільтри);
- захисту від іонізуючих, інфрачервоних, ультрафіолетових, електромагнітних, лазерних, магнітних та електричних полів (огороження, герметизація, знаки безпеки, автоматичний контроль і сигналізація, дистанційне керування тощо);

- захисту від шуму, вібрації (огородження, звукоізоляція, віброізоляція);
- захисту від ураження електричним струмом (різні види огородження, захисне заземлення, автоматичне відключення, дистанційне керування);
- захисту від дії механічних факторів (огородження, автоматичний контроль і сигналізація, знаки безпеки);
- захисту від хімічних факторів (огородження, герметизація, вентиляція та очищення повітря, дистанційне керування, знаки безпеки);
- захисту від високих і низьких температур навколишнього середовища (огородження, автоматичний контроль і сигналізація, термоізоляція, дистанційне керування).

Створення на робочому місці сприятливих і безпечних умов праці тісно пов'язано із забезпеченням робітників спецодягом, спецвзуттям та іншими засобами індивідуального захисту.

Засоби індивідуального захисту, залежно від призначення, поділяються на такі:

- ізолюючі костюми (пневмокостюми, скафандри);
- засоби захисту органів дихання (протигази, респіратори, пневмошоломи, пневмомаски);
- спеціальний одяг (комбінезони, куртки, штани, костюми, халати, плащі, кожухи, фартухи, жилети, наруківники);
- спеціальне взуття (чоботи, черевики, боти, бахіли);
- засоби захисту рук (рукавиці, рукавички);
- засоби захисту очей (захисні окуляри) ;
- засоби захисту обличчя (захисні маски, захисні щитки);
- засоби захисту голови (каски, шоломи, шапки, берети) ;
- засоби захисту від падіння з висоти тощо (запобіжні пояси, діелектричні килимки, ручні захвати, маніпулятори);
- засоби захисту органів слуху (протишумові шоломи, навушники, вкладиші);
- захисні дерматологічні засоби (різні миючі розчини, паста, креми, мазі).

При роботі з ЕОМ необхідно:

Робочі місця з візуальними дисплейними терміналами (відеотермінали, або ж ВДТ), персональними електронно-обчислювальними машинами (ПЕОМ),

електронно-обчислювальними машинами (ЕОМ) розміщують на відстані не менше 1 м від стін зі світловими прорізами. Робочі місця відносно світлових прорізів повинні розміщуватися так, щоб природне світло падало збоку, переважно зліва, та забезпечувало коефіцієнт природної освітленості не нижче 1,5%. Відстань між бічними поверхнями ВДТ має бути не меншою 1,2 м. Відстань між тильною поверхнею одного ВДТ і екраном другого – не менше 2,5 м. Площа одного обладнаного ПЕОМ робочого місця має бути не менше 6 м².

Робота на ПЕОМ належить до зорovo напружених. Це значить, що страждають передусім очі. Саме цим зумовлено більшість регламентних вимог до організації робіт з використанням ПЕОМ. Шкідливий вплив на очі виявляється через потребу їх постійно напружувати під час зчитування (споглядання) інформації з екрана. Багато користувачів, довго працюючи на комп'ютері, відчують різь в очах (відчуття «піску»), почервоніння очних яблук, біль у ділянці очної ямки і лоба, шийних хребців, а також під час руху очей. Це ознаки так званого комп'ютерного зорового синдрому. Крім того, є скарги й на появу симптомів серцево-судинних, нервових, шлунково-кишкових та інших захворювань. З'явилися повідомлення про «комп'ютерні» шийні радикуліти, захворювання суглобів кистей рук, дерматити шкіри обличчя. Тому для профілактики негативних наслідків треба дотримувати певного режиму праці та відпочинку, оптимальної організації робочого місця з комп'ютером.

Інша причина швидкої зорової втомлюваності – низький рівень освітлення робочого місця. Найчастіше це виявляється, коли одночасно працюють з паперовими матеріалами. Екран напряму випромінює світло, а людське око «звикло» читати у відбитому світлі. Тому, постійно читаючи інформацію з монітора, очі швидко втомлюються, особливо при надмірній яскравості. Також подразнення очей спричиняється миготінням зображення на моніторі через низьку частоту кадрової розгортки. Аби знизити миготіння екрана ЕПТ, рекомендується встановлювати частоту кадрів не менше 75 Гц. Для моніторів з рідкокристалічною індикацією достатня мінімальна частота кадрів – 60 Гц.

Ще однією небезпекою, що підстерігає під час роботи з комп'ютером, є статичність пози. Внаслідок цього під час тривалої роботи може настати запалення м'язів, зв'язок і сухожилів спини і ніг, захворювання хребта і суглобів

(остеохондроз, тендиніт тощо). Постійне напруження рук може спровокувати пошкодження зап'ястків та сухожилів (синдром променевоzap'ясткового суглоба, чи тунельний синдром). Ці захворювання спричиняються так званими травмами повторюваних навантажень – поступово накопичується нездужання, що згодом може перейти у хворобу нервів, м'язів, сухожилів. Для послаблення негативних наслідків дії цього шкідливого фактора годиться правильно, з індивідуальним підходом, організувати робоче місце: підібрати меблі, оптимально розташувати блоки комп'ютера.

Для того щоб зберегти здоров'я тих, хто працює на ПЕОМ, запобігти професійним захворюванням і підтримати працездатність, слід установити внутрішньозмінні регламентовані перерви для відпочинку. Для окремих груп користувачів ПЕОМ при 8-годинному режимі робочого дня (зміни) встановлюються такі перерви:

- для розробників програм – тривалістю 15 хв. через кожну годину роботи;
- для операторів – 15 хв. через кожні дві години роботи;
- для операторів комп'ютерного набору – 10 хв. через кожну годину роботи.

Якщо неможливо встановити вказані перерви, тривалість безперервної роботи з ВДТ не має перевищувати чотирьох годин. Обліковуючи робочий час працівників, які використовують ПЕОМ, слід урахувати, що вказані перерви для внутрішньозмінного відпочинку є складовою робочого часу (а час обідньої перерви не входить до нього). Цей час при відрядній формі оплати праці закладається в норми виробітку (норми праці), які знижують з урахуванням того, що протягом зміни (робочого дня) працівник не працює безперервно, а має право на виділений йому час для внутрішньозмінного відпочинку та особистих потреб.

6. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1. Вступ

Охорона навколишнього середовища являє собою форму відносин між суспільством і природою. Вона здійснюється різними засобами: економічними, правовими, науково-технічними, санітарно-гігієнічними, біологічними та іншими.

В загальному випадку проблема охорони навколишнього середовища зводиться до вирішення двох завдань:

- організації раціонального природокористування;
- забезпечення чистоти природних (екологічних) систем. При здійсненні різних видів економічної діяльності суб'єкти господарювання використовують різноманітні природні ресурси: землю, воду, корисні копалини тощо.

Проте ресурси ці обмежені. Обмеженість природних ресурсів була і залишається головною і дуже жорсткою умовою, що накладається на розвиток економіки і відповідно зростання суспільного добробуту.

Наслідком обмеженості природних ресурсів є конкуренція за їх застосування, тобто суперництво між альтернативними цілями використання ресурсів. Адже майже всі ресурси можуть використовуватися для задоволення найрізноманітніших потреб. Наприклад, нафта може служити сировиною для одержання палива, виробництва синтетичних волокон, пластмас, лакофарбових виробів, побутової хімії тощо. І всі ці альтернативні цілі конкурують за використання сировини нафти, обсяги якої, як відомо, обмежені.

Раціональне природокористування означає розробку та здійснення концепції і конкретних заходів щодо раціонального використання і відтворення природних ресурсів, гармонічну взаємодію суспільства і природи, людини і навколишнього природного середовища.

Завдання організації раціонального природокористування вирішується шляхом:

- оптимального розподілу ресурсів між різними господарськими цілями;
- використання технологій, що зберігають ресурси;
- проведення заходів щодо поповнення природних ресурсів.

Іншим, не менш важливим, завданням охорони навколишнього середовища є забезпечення чистоти природних екологічних систем, тобто водного середовища, повітряного басейну, ґрунтових покривів тощо, з тим, щоб забезпечити населення екологічно чистими продуктами харчування, водою, повітрям і, в остаточному підсумку, зберегти високий рівень здоров'я населення та його активного довголіття.

Економічна діяльність у всіх її проявах здійснює забруднення навколишнього середовища. У процесі цієї діяльності забруднюються і стають дефіцитними ресурси повітря, води, територій, що здавалися нескінченними. Нині рівень забруднення досяг загрозливих розмірів, набувши по суті кризового характеру.

Однією з причин забруднення навколишнього середовища є збільшення обсягу відходів та викидів. До них відносять: не використані у виробництві матеріали, що не підлягають подальшій переробці, або продукти, що відслужили свій термін споживання, різні пакувальні матеріали, всілякі відвали та терикони породи тощо.

Рациональне використання землі, лісу, атмосфери і водних ресурсів в Україні передбачено Конституцією. В даний час у сфері охорони навколишнього середовища діє цілий ряд нормативних актів: Закон України «Про охорону навколишньої природного середовища»; Постанова Уряду України «Про затвердження порядку визначення плати і її граничних розмірів за забруднення навколишньої природного середовища» і ін.

Під навколишнім середовищем розуміють цілісну систему взаємопов'язаних природних і антропогенних об'єктів і явищ, під впливом і при безпосередньому використанні яких відбувається праця, побутова діяльність, відпочинок людей. Поняття «навколишнє середовище» включає соціальні, природні і штучно створені фізичні, хімічні та біологічні фактори, тобто все те, що впливає на життя і діяльність людини. Складовою частиною навколишнього середовища є природне середовище. Перед сучасним суспільством стоїть завдання не тільки зберегти природу, а й запобігти негативним наслідкам господарської діяльності людини в майбутньому.

Охорона навколишнього середовища являє собою важку комплексну проблему, яка має відношення до всього суспільства в цілому і до кожного окремого громадянина.

Розмова йде про рішення життєво важливої проблеми – захисту і охорони здоров'я нинішнього і майбутнього покоління людей від шкідливих наслідків їх науково-технічної і промислової діяльності.

Завданням держави є розробка на основі наукових досліджень програми заходів у галузі охорони навколишнього середовища і формування достатніх фінансових коштів на екологічні потреби й покриття екологічних витрат. В Україні функції з охорони природи і контролю за дотриманням екологічних нормативів здійснює Міністерство екології, а також Державна екологічна інспекція.

У загальному випадку до екологічних витрат належать:

- економічна оцінка негативних наслідків забруднення і порушення компонентів цілісного природного середовища (економічні збитки, шкода);
- поточні і капітальні витрати на здійснення природоохоронних заходів;
- платежі за викиди забруднюючих речовин і розміщення відходів у навколишньому середовищі;
- платежі за використання природних ресурсів;
- штрафи за екологічні правопорушення і платежі на відшкодування заподіяних збитків;
- система компенсаційних виплат населенню, що мешкає в несприятливих екологічних умовах.

Усі екологічні витрати суспільства можна поділити на такі групи:

1. залежно від того, ким покриваються витрати, - витрати держави і витрати суб'єктів господарювання;
2. залежно від характеру здійснених витрат - витрати на запобігання негативного впливу на навколишнє середовище і витрати, викликані таким впливом.

Економічний збиток від забруднення навколишнього середовища - це втрати від впливу забрудненого середовища на здоров'я населення, на майно юридичних і фізичних осіб, на сільськогосподарські угіддя, лісові та водні ресурси. Санкції за

заподіяння шкоди пред'являються державними органами охорони навколишнього середовища юридичним і фізичним особам, що заподіяли збиток. Набагато ефективнішим є впровадження в економіку прогресивних безвідхідних технологій, удосконалення заходів щодо утилізації відходів, прийняття і дотримання міжнародних норм з охорони навколишнього середовища.

За порушення у сфері охорони природи, незаконне використання природних ресурсів законодавством України передбачена адміністративна (статті 52-92 Кодексу про адміністративні правопорушення) та кримінальна (статті 236-254 Кримінального кодексу) відповідальність.

6.2. Забруднення навколишнього середовища підприємством з розробки програмного забезпечення

Величезна кількість відходів є результатом значного збільшення обсягів виробництва. За підрахунками фахівців, за останнє століття світове промислове виробництво збільшилося більш як у 50 разів, причому 4/5 цього приросту припадає на кінець XX століття. Як слушно зазначають американські економісти, збільшення валового національного продукту є одночасно збільшенням "валового національного сміття". Проблема утилізації відходів стала сьогодні проблемою глобального масштабу.

Не менш значною причиною забруднення є широке використання забруднюючих технологій, які для багатьох підприємств є вигіднішими, ніж екологічно чисті, в силу більшої дешевизни виробництва продукції і менших витрат товарообігу.

Для підприємців, що прагнуть мінімізувати свої витрати, здійснювати природоохоронні заходи не вигідно. Так, набагато простіше скинути відходи або викиди, ніж будувати дорогі очисні споруди. Тому стати на захист суспільних інтересів з охорони навколишнього середовища покликана держава.

Підприємства хутрової промисловості для вичинки та фарбування хутра за рік потребують 9 млн. м³ води і витрачають різних хімічних речовин та сполук до 100 тис. тонн. Основний напрям інтенсифікації обробки шкіри полягає у застосуванні більш концентрованих технологічних розчинників та сухих реагентів, що розчиняються у капілярах сировини. Більш ефективним процесом

є вичинка та фарбування хутра в органічних розчинах, без води. Токсичність стічних вод у хутровому виробництві зумовлюється наявністю у них шестивалентного хрому барвників та формаліну.

6.3. Розробка заходів щодо зменшення забруднення

У переробній промисловості для очищення від органічних забрудників доцільно застосовувати біологічні методи очищення.

Всі відомі методи фізико-хімічного очищення стічних вод від барвників (які є наймасовішим забрудником стоків легкої промисловості) можна розділити на три основні групи.

Перша група методів забезпечує вилучення забруднень перетворенням їх в осад шляхом сорбції на частинках гідроксидів металів, які утворюються в процесі реагентної обробки стічних вод. Методи цієї групи – коагуляція, реагентна напірна флотація, електрокоагуляція.

Друга група об'єднує деструктивні методи, які ґрунтуються на глибоких перетвореннях органічних молекул в результаті редокспроцесів. З деструктивних методів найбільш широко використовують очищення стічних вод окиснювачами, реагентною відновно-окислюючою, електрохімічною та електрокаталітичною деструкцією. Вони володіють рядом переваг, а саме: висока ефективність та технологічність, компактність і простота автоматизації та управління. Але поряд з перевагами в очищенні стічних вод окиснювачами існує і ряд недоліків: невисока ступінь окиснення хімічно стійких органічних речовин, в результаті чого можливе утворення більш токсичних речовин; складність апаратурного оформлення вузла генерування окиснювача. За умови реалізації реагентної відновно-окислювальної деструкції потрібно проводити доочищення стічних вод, а за умови реалізації електрохімічної та електрокаталітичної деструкції потрібна велика затрата електроенергії та застосування обладнання великих розмірів.

Третя група включає сепаративні методи, такі, як сорбція на активованому вугіллі та макропористих іонітах, інших видах сорбентів (в тому числі на природних сорбентах) зворотній осмос, ультрафільтрація, пінна сепарація,

електрофлотація. Ці методи, виключаючи два останніх, забезпечують високу степінь очищення стічних вод.

Висновок

Під час виконання магістерської роботи було проаналізовано засоби та технології та була розроблена інформаційна система взаємного редагування документів для онлайн-курсів.

В роботі були виконані:

- вивчення предметної області та формування вимог до системи;
- розробка концепції системи;
- розробка технічного завдання на створення системи;
- розробка загальносистемних рішень, рішень з організаційного, інформаційного, технічного та програмного забезпечення системи;
- розробка супроводжувальної документації.

Під час прийняття загальносистемних рішень була визначення архітектура системи та її дерево функцій.

Було обрано технології для реалізації, а саме Node.js з використанням фреймворка Express.js для роботи на сервері, Socket.IO для обміну даними між клієнтами в режимі реального часу та NoSQL базу даних – MongoDB.

Результатами виконання роботи є:

- сервер для взаємодії з клієнтами та базою даних;
- веб-клієнт для редагування документів.

Впровадження розробленої системи може бути використано для проведення онлайн-курсів, а також на інших підприємствах.

Список використаних джерел

1. Jon Erickson. Hacking: The art of exploitation, 2nd edition. – No Starch Press, 2008. – 472 pages.
2. Dennis Yurichev. Reverse Engineering for Beginners. 2019. [https://beginners.re/RE4B-EN.pdf]
3. Roberto Sanchez. Comparing Node.js vs PHP Performance. 2018. [https://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/]
4. Weblab Technology. Encrypted client-server communication. 2017. [https://medium.com/@weblab_tech/encrypted-client-server-communication-protection-of-privacy-and-integrity-with-aes-and-rsa-in-c7b180fe614e/]
5. David Ireland. RSA Algorithm. 2018. [https://www.di-mgt.com.au/rsa_alg.html]
6. AES-NI: Hardware Encryption in your Processor. 2011. [https://datacenteroverlords.com/2011/09/07/aes-ni-pimp-your-aes/]
7. Josh Lake. What is TLS and how does it work. 2019. [https://www.comparitech.com/blog/information-security/tls-encryption/]
8. Mihir Shah. MongoDB vs MySQL – Dzone Database. 2017. [https://dzone.com/articles/comparing-mongodb-amp-mysql/]
9. JReport. 3-Tier Architecture: A Complete Overview. 2018. [https://www.jinfony.com/resources/bi-defined/3-tier-architecture-complete-overview/]
10. Binary JSON Specification. [http://bsonspec.org/spec.html]
11. Комп'ютерні мережі: підручник / А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник. – Львів: «Магнолія 2006», 2013. – 256 с.
12. Семирівнева модель OSI. 2015. [https://studfile.net/preview/3741336/]
13. Методичні вказівки до виконання дипломного проектування зі спеціальності 7.080401 «Інформаційні управляючі системи та технології». – К.В. Кошкін, С.Б. Приходько, С.В. Суслов, В.І. Тимофєєв. – Миколаїв: УДМТУ, 2003. – 52 с.
14. Безопасность жизнедеятельности. Под редакцией Белова Н.А. – М.: Знание, 2000. – 364 с.

15. Борьба с шумом на производстве: Справочник / Юдина Э.Я., Борисова Л.А. под редакцией Юдина Э.Я. – М.: Машиностроение, 1985. – 400 с.
16. Самгин Э.Б. Освещение рабочих мест. – М.: МИРЭА, 1989. – 186 с.
17. Новиков Ю.В. Экология, окружающая среда и человек: Учебное пособие для вузов. – М.: Агенство «ФАИР», 1998. – 295 с.
18. ГОСТ 34.310 – 95. Информационные технологии. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на базе ассиметричного криптографического алгоритма.
19. ГОСТ 34.602 – 89. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

Додаток А. Технічне завдання на розробку

1. Загальні відомості

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: Розподілена інформаційна система спільного редагування документів у режимі реального часу.

1.2 Перелік документів, на основі яких створюється система

Підставою для розробки є наступний документ:

- завдання на магістерську роботу зі спеціальності 122 «Інформаційні управляючі системи та технології».

1.3 Планові строки початку та закінчення робіт зі створення системи

Плановий строк початку робіт зі створення системи – 4 липня 2019 року.

Плановий строк закінчення робіт зі створення системи – 12 листопада 2019 року.

1.4 Порядок оформлення та пред'явлення замовнику результатів робіт зі створення системи

Система передається у вигляді комплексу на базі засобів обчислювальної техніки Замовника та Виконавця у встановлені строки. Приймання системи здійснюється комісією у складі вповноважених представників Замовника та Виконавця.

Спільно з пред'явленням системи здійснюється здача розробленого Виконавцем комплексу документації.

1.5 Перелік нормативно-технічних документів, методичних матеріалів, використаних при розробці ТЗ

При розробці системи та створенні проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ГОСТ 19.201-78. Техническое задание. Требования к содержанию и оформлению;

- ГОСТ 34.601-90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;
- ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплексность и обозначение документов при создании автоматизированных систем.

1.6 Визначення, позначення та скорочення

БД – база даних;

ІПС – інформаційна пошукова система;

ІС – інформаційна система;

ОС – операційна система;

СУБД – система управління базами даних;

ТЗ – технічне завдання.

2. Призначення та цілі створення системи

2.1 Призначення системи

Система призначена для ефективного та безперебійного спільного редагування документів у режимі реального часу.

2.2 Цілі створення системи

Створення модернізованої системи має на меті вирішення наступних завдань:

- побудова ефективного навчального процесу в онлайн;
- полегшення інтерактивної взаємодії між студентів та вчителем;
- збільшення кількості людей, які використовують онлайн навчання.

4. Необхідні засоби для створення і функціонування системи

Web-сервер – використовується для спілкування з клієнтом. Він повинен приймати повідомлення по протоколу HTTPS та відправляти запитувану інформацію.

Сервер обміну інформації – його архітектура оптимізована для роботи в реальному часі між клієнтами та сервером. Використовує локальне кеширування для оптимізації роботи загальної бази даних.

Сервер зберігання інформації – використовує NoSQL базу даних - MongoDB для прискореного зберігання необхідної інформації.

5. Загальні вимоги до приймання робіт

Здача-приймання робіт проводиться поетапно, у відповідності з робочою програмою та календарним планом. Здача-приймання здійснюється комісією, до складу якої входять представники Замовника і Виконавця. За результатами приймання підписується акт приймальної комісії.

Всі створювані в рамках цієї роботи програмні вироби (за винятком покупних) передаються Замовнику, як у вигляді готових модулів, так і у вигляді вихідних кодів, що подаються в електронній формі на стандартному машинному носії.

5.1 Статус приймальної комісії

Статус приймальної комісії визначається замовником до проведення випробувань.

6. Вимоги до документації

Разом з розробленими програмними виробами Виконавець повинен передати Замовникові наступний перелік відповідно оформлених документів:

- програма та методика випробувань, протокол випробувань системи;
- пояснювальна записка до проекту системи;
- опис застосування системи;
- керівництво з технічного обслуговування.

Додаток Б. Посібник користувача

Для відкриття онлайн-платформи перейдіть по посиланню <https://whiteboard.up2native.com.ua>. Відкриється головне вікно для вводу ідентифікатора документа (рис. Б.1).

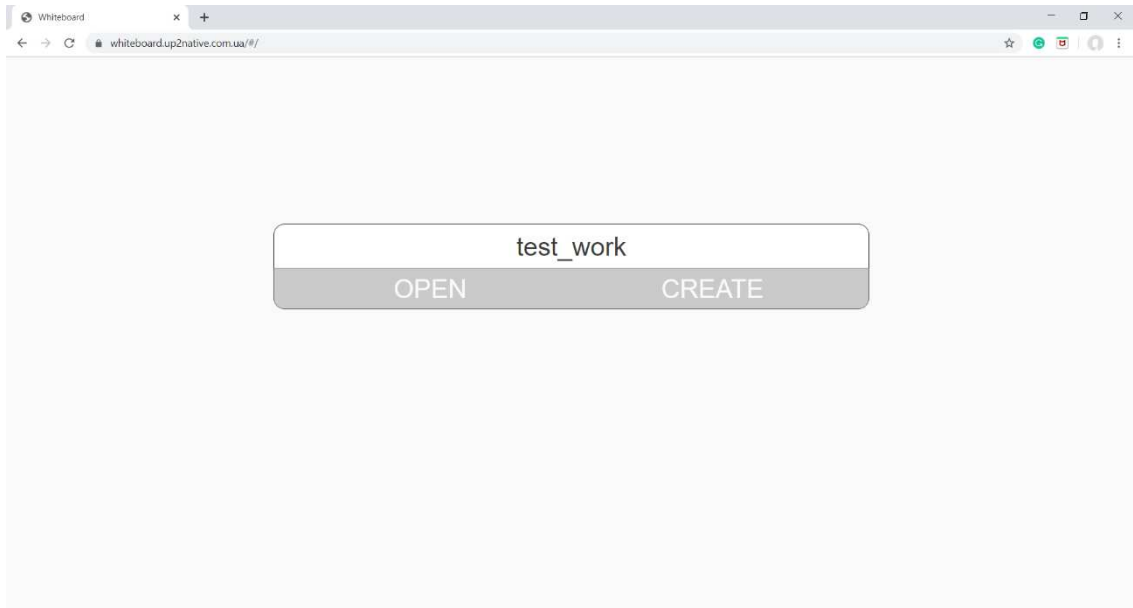


Рисунок Б.1 – Головне вікно для вводу ідентифікатора документа

Для відкриття документа натисніть «OPEN». Після цього відкриється новий (рис. Б.2) або існуючий документ.

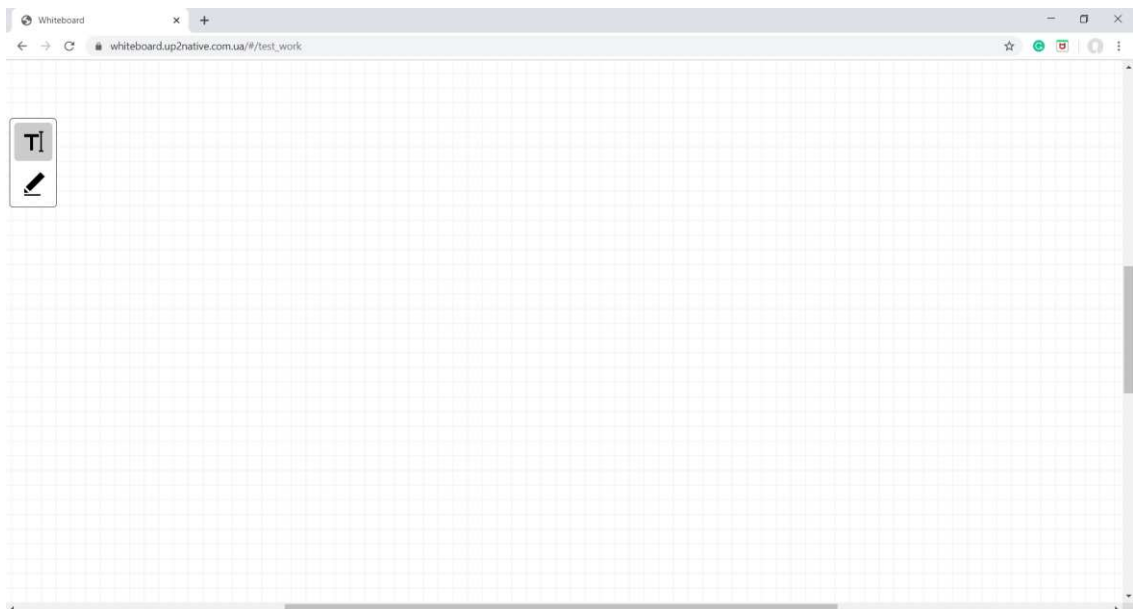


Рисунок Б.2. – Відкритий новий документ

У лівій частині є панель інструментів. Панель інструментів складається з інструмента «текст» та «перо» (рис. Б.3). Для вибору інструмента натисніть на нього.

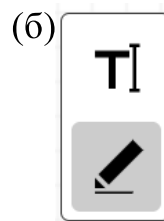
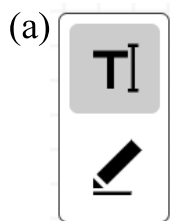


Рисунок Б.3 – (а) виділений інструмента «текст», (б) виділений інструмент «перо»

Після вибору інструмента «перо» натисніть на робочій області і новий текстовий елемент буде вставлено (рис. Б.4).

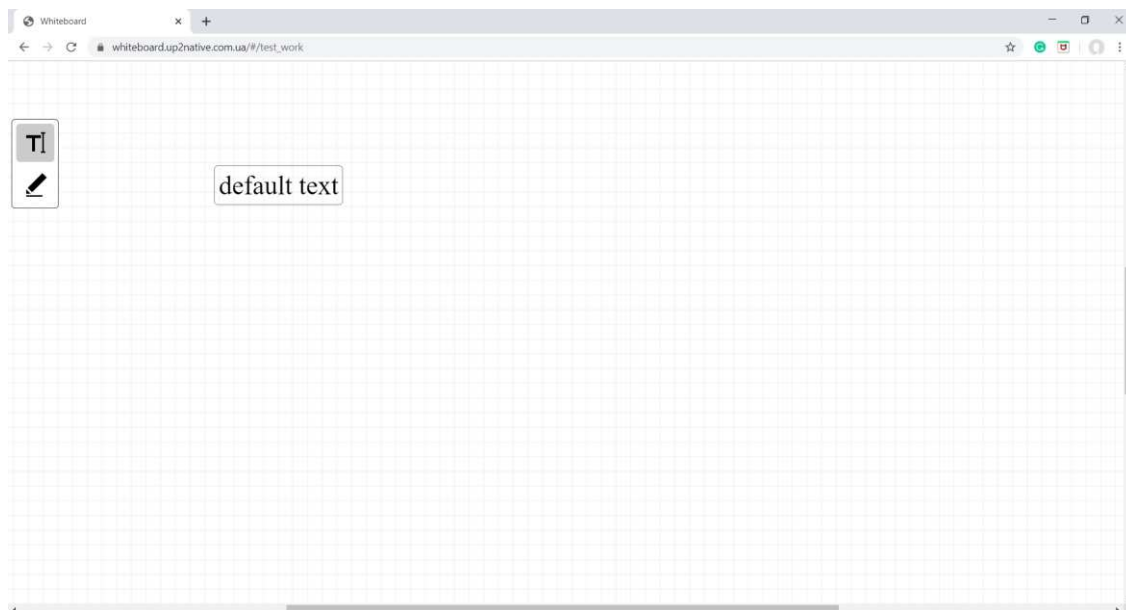


Рисунок Б.4 – Вставлений елемент «текст»

Для редагування тексту використайте двійне натискання. Після набрання нового тексту натисніть на вільній області і текст буде оновлено.

Для використання інструмента «перо» натисніть на нього у панелі інструментів та зажав ліву клавішу миші почніть водити по екрану, відпустивши клавішу, малюнок буде створено (рис. Б.5).

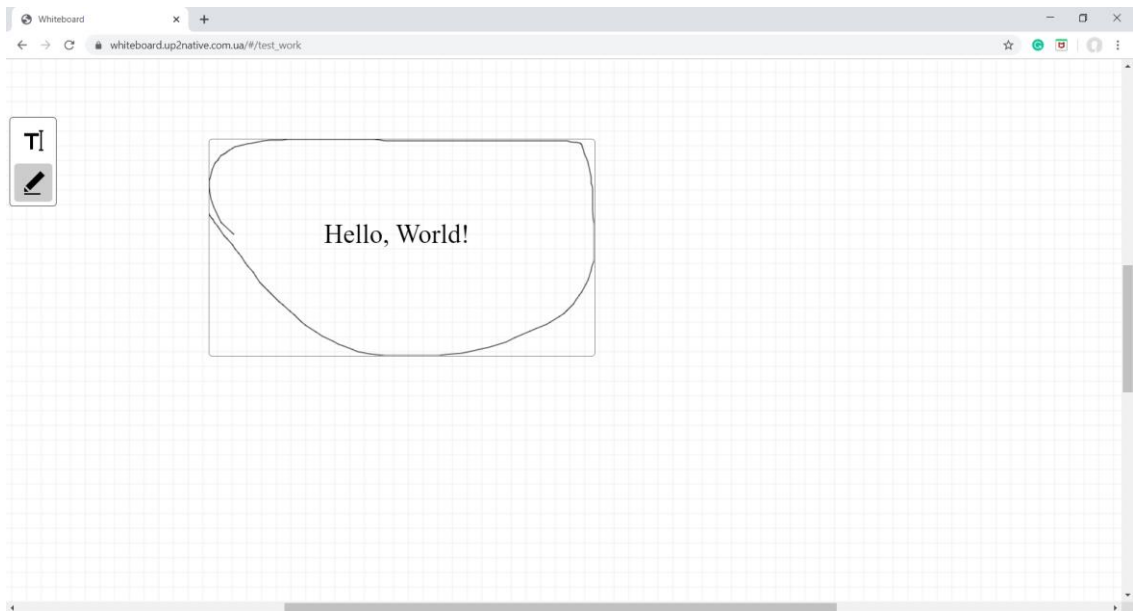


Рисунок Б.5 – Використання інструмента «перо»

Відкривши документ на іншому комп'ютері усі створені елементи будуть підгружені (рис. Б.6).

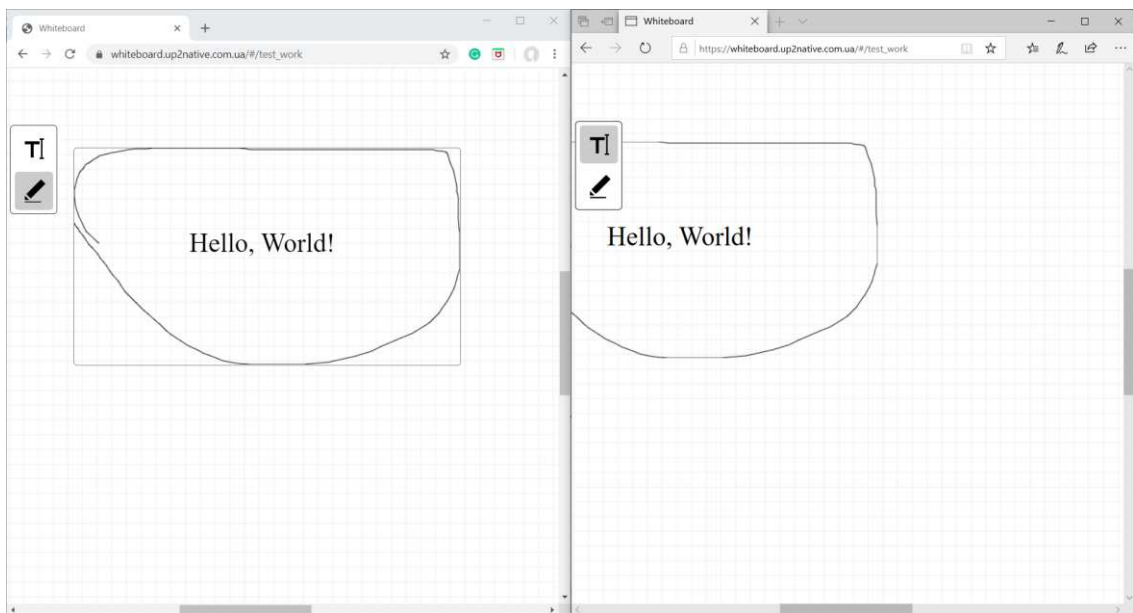


Рисунок Б.6 – Відкриття існуючого документа

Та можна буде продовжити редагування, де усі зміни будуть синхронізуватися. Додавши новий елемент типу «текст», він буде синхронізуватися з усіма відкритими вікнами (рис. Б.7).

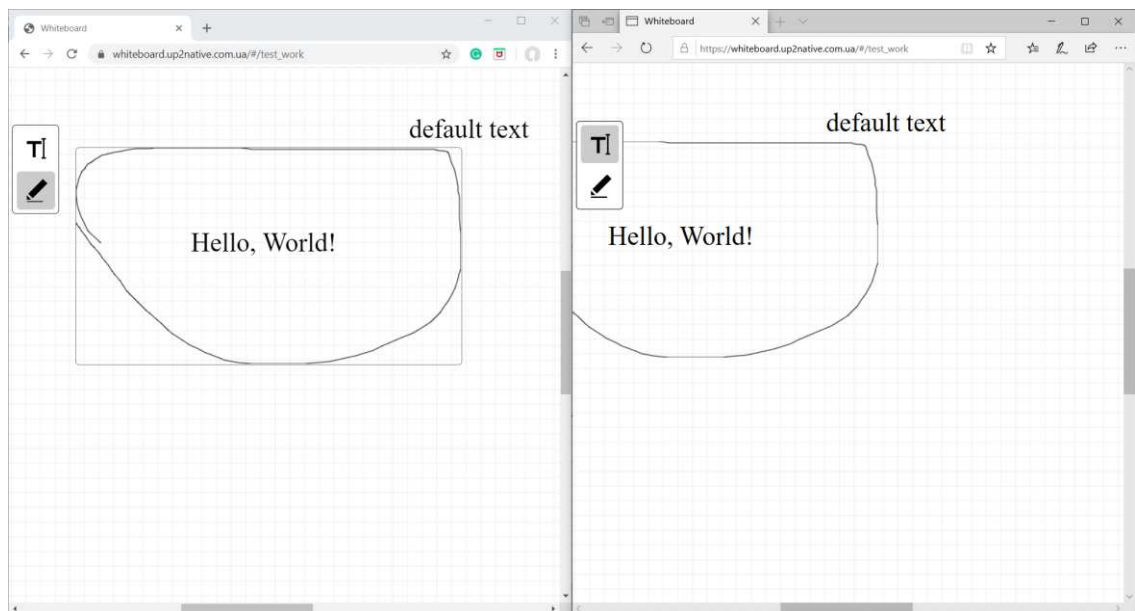


Рисунок Б.7 – Синхронізування нового елементу типу «текст»

Додаток В. Исходный код

1. Domain.js

```
class Document{
  constructor(){
    this.id = 0;
    this.objs = [];
  }
}

class InstrumentType{
  static DragableText = "DragableText";
  static Drawing = "Drawing";
}

class Instrument{
  constructor(){
    this.id = 0;
    this.type = 0;
    this.deltas = [];
  }
}

class UpdateType{
  static Position = "Position";
  static Size = "Size";
  static Text = "Text";
  static Dots = "Dots";
}

export { Document, InstrumentType, Instrument, UpdateType }
```

2. server.js

```
const http = require("http");
const express = require("express");
const app = express();
const webpack = require("webpack");
const webpackDevMiddleware = require("webpack-dev-middleware");
const webpackHotMiddleware = require("webpack-hot-middleware");
const config = require('./webpack.config.js');
const compiler = webpack(config);
const mongo = require("mongodb").MongoClient;

var server = http.createServer(app);
var io = require("socket.io").listen(server);
app.use(webpackDevMiddleware(compiler, {
  noInfo: true,
  publicPath: config.output.publicPath
}));
app.use(webpackHotMiddleware(compiler));

server.listen(3000, "0.0.0.0", function(){ });
const url = "mongodb://127.0.0.1:27017/up2natedb";
const params = { useUnifiedTopology: true, useNewUrlParser: true };
mongo.connect(url, params, function(err, client){
  var db = client.db("up2natedb");
  var sheets = db.collection("sheets");
  io.on("connection", function(socket){
    socket.on("update", function(sheet){
```

```

    const sid = sheet.id;
    const sobjs = sheet.objs;
    sheets.updateOne({
      id: sid
    }, {
      $addToSet: {
        objs: {
          id: sobjs[0].id,
          type: sobjs[0].type
        }
      }
    },
    { upsert: true }
  ).then(() => {
    sheets.updateOne({
      id: sid,
      objs: { $elemMatch: { id: sobjs[0].id } }
    }, {
      $addToSet: {
        "objs.$.deltas": { $each: sobjs[0].deltas }
      }
    });
  });
  io.emit("update", sheet);
});
socket.on("initialize", function(sheet){
  sheets.findOne({
    id: sheet.id
  }).then((document) => {
    const sobjs = document.objs;
    io.emit("initialize", { id: sheet.id, objs: sobjs });
  });
});
socket.on("disconnect", function(){
});
});
});

```

3. app.js

```

import "reset-css"
import Vue from "vue"
import VueRouter from "vue-router"
import App from "./App.vue"
import Launch from "./components/Launch.vue"
import Sheet from "./components/Sheet.vue"

```

```

Vue.use(VueRouter)
const routes = [
  { path: "/", component: Launch },
  { path: "/:id", component: Sheet }
]
const router = new VueRouter({
  routes
})
new Vue({
  el: "#app",

```

```

    router: router,
    render: h => h(App)
  })

```

4. App.vue

```

<template>
  <div id="app">
    <router-view></router-view>
  </div>
</template>

```

```

<script>
export default{
  name: "App"
}
</script>

```

5. Sheet.vue

```

<template>
  <div
    id="sheet"
    v-on:mousedown="handleMouseDown"
    v-on:mouseup="handleMouseUp"
    v-on:mousemove="handleMouseMove"
    v-bind:style="sheetStyle">
    <canvas id="background" v-bind:width="maxWidth" v-
bind:height="maxHeight"></canvas>
    <div id="menu">
      
      
    </div>
    <component
      v-if="nobj != null"
      v-bind:is="nobj.type"
      v-bind:id="nobj.id"
      v-bind:top="nobj.y"
      v-bind:left="nobj.x"
      v-bind:maxWidth="nobj.maxWidth"
      v-bind:maxHeight="nobj.maxHeight"
      v-bind:isUpdated="nobj.isUpdated"
      v-on:handleObjClick="handleObjClick"
      v-on:handleObjDrag="handleObjDrag"
    >

```

```

        v-on:handleUpdateFromClient="handleUpdateFromClient">
    </component>
    <component
        ref="objs"
        v-for="(obj, index) in sheet.objs"
        v-bind:key="index"
        v-bind:is="obj.type"
        v-bind:id="obj.id"
        v-bind:top="obj.y"
        v-bind:left="obj.x"
        v-bind:maxWidth="obj.maxWidth"
        v-bind:maxHeight="obj.maxHeight"
        v-bind:isUpdated="obj.isUpdated"
        v-on:handleObjClick="handleObjClick"
        v-on:handleObjDrag="handleObjDrag"
        v-on:handleUpdateFromClient="handleUpdateFromClient">
    </component>
</div>
</template>

<script>
import io from "socket.io-client";
import {Document, InstrumentType, Instrument, UpdateType} from "../data/Domain.js"
import { debounce } from "../objects/Helper.js";
import DragableText from "./DragableText.vue"
import Drawing from "./Drawing.vue"

export default {
    data() {
        return {
            els: 0,
            width: 0,
            height: 0,
            resetFocus: null,
            dragging: null,
            selectedInstrument: InstrumentType.DragableText,
            maxWidth: 3000,
            maxHeight: 3000,
            sheet: new Document(),
            nobj: null,
            socket: io()
        }
    },
    created() {
        const params = this.$route.params;
        this.sheet.id = params.id;
        this.socket.on("update", this.handleBroadcastUpdate);
        this.socket.on("initialize", this.initializeSheet);
    },
    mounted() {
        this.handleResize();
        this.handleBackground();
        let initializeSheet = new Document();
        initializeSheet.id = this.sheet.id;
        this.socket.emit("initialize", initializeSheet);
    },

```



```

components: {
  DragableText,
  Drawing
},
props: {
  is_created:{
    type: Boolean,
    default: false
  }
},
methods: {
  initializeSheet(sheet){
    if(sheet == null || this.sheet.id !== sheet.id){
      return;
    }
    const sobjs = sheet.objs;
    for(let item = 0, els = 0; item < sobjs.length; ++item){
      if(sobjs[item].deltas !== undefined){
        let foundIndex = this.sheet.objs.findIndex((obj) => obj.id
== sobjs[item].id);
        if(foundIndex > -1){
          continue;
        }
        sobjs[item].deltas.sort(function(a, b){
          return a.datetime > b.datetime;
        });
        let data = sobjs[item];
        this.sheet.objs.push({
          type: data.type,
          x: data.deltas[0].xy.x,
          y: data.deltas[0].xy.y,
          id: data.id,
          maxWidth: this.maxWidth,
          maxHeight: this.maxHeight,
          isUpdated: true });
      }
    }

    this.$nextTick().then(() => {
      for(let item = 0, els = 0; item < sobjs.length; ++item){
        if(sobjs[item].deltas !== undefined){
          this.els = this.els + 1;
          let data = sobjs[item];
          this.$refs.objs[this.els - 1].receiveUpdate({ deltas:
data.deltas });
        }
      }
    });
  },
  handleBroadcastUpdate(sheet){
    if(this.sheet.id !== sheet.id){
      return;
    }
    let data = sheet.objs[0];

```

```

        let foundIndex = this.sheet.objs.findIndex((obj) => obj.id ==
data.id);
        if(foundIndex > -1){
            this.$refs.objs[foundIndex].receiveUpdate(data);
            return;
        }
        this.sheet.objs.push({
            type: data.type,
            x: data.deltas[0].xy.x,
            y: data.deltas[0].xy.y,
            id: data.id,
            maxWidth: this.maxWidth,
            maxHeight: this.maxHeight,
            isUpdated: true });
        this.$nextTick().then(() => {
            let foundIndex = this.sheet.objs.findIndex((obj) => obj.id ==
data.id);
            this.$refs.objs[foundIndex].receiveUpdate({ deltas: data.deltas
});
        });
    },
    handleUpdateFromClient(data){
        for(let i = 0; i < data.deltas.length; ++i){
            data.deltas[i].datetime = Date.now() + i;
        }
        let updateSheet = new Document();
        updateSheet.id = this.sheet.id;
        updateSheet.objs = [ data ];
        this.socket.emit("update", updateSheet);
        this.nobj = null;
    },
    handleResize(){
        const centerX = (this.maxWidth - window.innerWidth) / 2;
        const centerY = (this.maxHeight - window.innerHeight) / 2;
        window.scroll(centerX, centerY);
    },
    handleBackground(){
        let context =
document.getElementById("background").getContext('2d');
        context.fillStyle = "#ffffff";
        context.strokeStyle = "#efefef";
        context.fillRect(0, 0, this.maxWidth, this.maxHeight);
        this.drawBackgroundLines(context, this.maxWidth, this.maxHeight, 20,
1, 0);
        this.drawBackgroundLines(context, this.maxHeight, this.maxWidth, 20,
0, 1);
    },
    drawBackgroundLines(context, size, length, step, isVertical,
isHorizontal){
        for(let i = step; i < size; i += step){
            context.beginPath();
            context.moveTo(i * isVertical, i * isHorizontal);
            context.lineTo(i * isVertical + length * isHorizontal, i *
isHorizontal + length * isVertical);
            context.stroke();
        }
    }
}

```

```

    },
    handleMenuClick(item){
        this.selectedInstrument = item;
    },
    handleMouseDown(event){
        if(this.resetFocus != null){
            this.resetFocus();
            this.resetFocus = null;
            return;
        }
        this.els = this.els + 1;
        const uid = "el" + this.els;
        this.nobj = {
            type: this.selectedInstrument,
            x: event.pageX,
            y: event.pageY,
            id: uid,
            maxWidth: this.maxWidth,
            maxHeight: this.maxHeight,
            isUpdated: false };
    },
    handleMouseMove(event){
        if(this.dragging != null){
            this.dragging.move(event);
        }
    },
    handleMouseUp(event){
        if(this.dragging != null){
            this.dragging.up(event);
        }
    },
    handleObjClick(value){
        if(this.resetFocus != null){
            this.resetFocus();
        }

        value.focus();
        this.resetFocus = value.unfocus;
    },
    handleObjDrag(value){
        this.dragging = value;
    }
},
computed: {
    sheetStyle(){
        return{
            width: this.maxWidth + "px",
            height: this.maxHeight + "px"
        }
    },
    draggableTextOn(){
        return {
            background: this.selectedInstrument ==
InstrumentType.DragableText ? "#ccc" : "#fff"
        }
    },

```

```

        drawingOn() {
            return {
                background: this.selectedInstrument == InstrumentType.Drawing ?
"#ccc" : "#fff"
            }
        }
    };
</script>

```

```

<style lang="sass">
#sheet
    background: #FAFAFA
    font-size: 18px
    font-family: "Roboto", sans-serif

```

```

#sheet canvas
    position: absolute

```

```

#menu
    position: fixed
    left: 0
    top: 10%
    margin: 5px
    z-index: 1000

    border-width: 1px
    border-style: solid
    border-color: #333333
    border-radius: 5px

```

```

    background: #FFFFFF

```

```

    img
        display: block
        width: 32px
        height: 32px
        margin: 5px
        padding: 10px
        border: none
        border-radius: 5px
        background: #fff

```

```

    img:hover
        background: #ccc

```

```

</style>

```

6. DragableText.vue

```

<template>
    <div
        v-on:mousedown.stop="handleMouseDown"
        v-on:mouseover="handleHover(true)"
        v-on:mouseout="handleHover(false)"
        v-on:dblclick="handleDblClick"
        v-bind:id="id"
        v-bind:style="style"
    >

```

```

        class="dragabletext">
        <textarea wrap="off" ref="textareaText" v-on:mousedown.stop="{}" v-
if="editable" v-model="text" v-bind:style="textareaStyle"></textarea>
        <div ref="divText" v-else>{{ text }}</div>
    </div>
</template>

<script>
import { InstrumentType, Instrument, UpdateType } from "../../data/Domain.js"

export default {
  name: InstrumentType.DragableText,
  data() {
    return {
      defaultText: "default text",
      color: "transparent",
      isActive: false,
      moving: { isActive: false, x: 0, y: 0 },
      x: 0,
      y: 0,
      textWidth: 0,
      textHeight: 0,
      offsetX: 5,
      offsetY: 5,
      editable: false,
      instrument: new Instrument()
    }
  },
  created() {
    this.instrument.id = this.id;
    this.instrument.type = InstrumentType.DragableText;
    if(!this.isUpdated){
      this.sendUpdate([{ type: UpdateType.Position, xy: { x: this.left, y:
this.top } }]);
    }
    this.y = this.top - this.offsetX;
    this.x = this.left - this.offsetY;
  },
  methods: {
    receiveUpdate(data){
      for(let delta of data.deltas){
        this.instrument.deltas.push(delta);
        switch(delta.type){
          case UpdateType.Position:
            this.x = delta.xy.x;
            this.y = delta.xy.y;
            break;
          case UpdateType.Text:
            this.defaultText = delta.text;
            break;
        }
      }
    },
    sendUpdate(deltas){
      let updateInstrument = new Instrument();
      updateInstrument.id = this.instrument.id;

```

```

        updateInstrument.type = InstrumentType.DragableText;
        updateInstrument.deltas = deltas;
        this.$emit("handleUpdateFromClient", updateInstrument);
    },
    handleHover(isHover){
        if(!this.isActive){
            this.color = isHover ? "#BBBBBB" : "transparent";
        }
    },
    handleMouseDown(event){
        this.moving.isActive = true;
        this.moving.x = event.offsetX;
        this.moving.y = event.offsetY;
        this.$emit("handleObjClick", { focus: this.callbackFocus, unfocus:
this.callbackUnfocus });
        this.$emit("handleObjDrag", { move: this.handleMouseMove, up:
this.handleMouseUp });
    },
    handleMouseUp(event){
        if(this.moving.isActive){
            this.sendUpdate([{ type: UpdateType.Position, xy: { x: this.x,
y: this.y } }]);
            this.moving.isActive = false;
        }
    },
    handleMouseMove(event){
        if(this.moving.isActive){
            this.y = event.pageY - this.moving.y;
            this.x = event.pageX - this.moving.x;
        }
    },
    handleDbClick(event){
        if(this.editable){
            return;
        }
        const el = this.$refs.divText;
        this.textWidth = el.clientWidth;
        this.textHeight = el.clientHeight;
        this.editable = true;
        this.$nextTick(() => this.$refs.textareaText.focus());
    },
    callbackFocus(text){
        this.isActive = true;
        this.color = "#777777";
    },
    callbackUnfocus(text){
        this.isActive = false;
        this.color = "transparent";
        this.defaultText = this.defaultText.trim();
        if(this.editable){
            this.sendUpdate([{ type: UpdateType.Text, text: this.defaultText
}]);
            this.editable = false;
        }
    },
    adjustTextarea(){

```

```

        if(!this.editable){
            return;
        }
        const textareaEl = this.$refs.textareaText;
        textareaEl.style.width = 0;
        textareaEl.style.width = textareaEl.scrollWidth + "px";
        textareaEl.style.height = 0;
        textareaEl.style.height = textareaEl.scrollHeight + "px";
    }
},
computed: {
    text: {
        set: function(value){
            this.defaultText = value;
            this.adjustTextarea();
        },
        get: function(){
            return this.defaultText;
        }
    },
    textareaStyle(){
        return {
            width: this.textWidth + "px",
            height: this.textHeight + "px"
        }
    },
    style(){
        return {
            top: this.y + "px",
            left: this.x + "px",
            border: "1px solid " + this.color
        }
    }
},
props: ["top", "left", "id", "isUpdated"]
};
</script>

<style lang="sass">
.dragabletext
    position: absolute
    border: none
    font: 27pt serif
    box-sizing: border-box
    cursor: default
    border-radius: 5px

    -webkit-user-select: none
    -khtml-user-select: none
    -moz-user-select: -moz-none
    -o-user-select: none
    user-select: none

.dragabletext div
    margin: 5px;
    overflow-wrap: nowrap

```

```

    white-space: pre

.dragabletext:hover
  cursor: pointer

.dragabletext textarea
  margin: 3px;
  font: 27pt serif
  border: none
  background: transparent
  resize: none
  overflow: hidden
  outline: none
  vertical-align: top
</style>

```

7. Drawing.vue

```

<template>
  <canvas
    v-on:mousedown.stop="handleMouseDown"
    v-on:mouseover="handleHover(true)"
    v-on:mouseout="handleHover(false)"
    v-bind:id="id"
    v-bind:width="width"
    v-bind:height="height"
    v-bind:style="style"
  >
  </canvas>
</template>

<script>
import { InstrumentType, Instrument, UpdateType } from "../../data/Domain.js"
import { debounce } from "../../objects/Helper.js";

export default {
  name: "drawing",
  data() {
    return {
      color: "transparent",
      isActive: false,
      moving: { isActive: false, isFocus: false },
      x: 0,
      y: 0,
      width: 0,
      height: 0,
      editable: false,
      canvas: null,
      context: null,
      dots: [],
      instrument: new Instrument(),
      redrawDrawings: null
    }
  },
  created() {
    console.log("drawing instrument");
    this.instrument.id = this.id;
  }
}

```



```

    this.instrument.type = this.selectedInstrument;
    this.instrument.deltas.push({ xy: { x: this.left, y: this.top } });

    if(!this.isUpdated){
        this.width = this.maxWidth;
        this.height = this.maxHeight;
        this.dots.push({ x: this.left, y: this.top });
    }
    this.redrawDrawings = debounce(this.handleDebouncing, 10);
},
mounted(){
    this.canvas = document.getElementById(this.id);
    this.context = this.canvas.getContext("2d");
    if(!this.isUpdated){
        this.moving.isActive = true;
        this.context.clearRect(0, 0, this.width, this.height);
        this.context.beginPath();
        this.context.moveTo(this.dots[0].x, this.dots[0].y);
    }
    this.$emit("handleObjDrag", { move: this.handleDrawingMove, up:
this.handleDrawingEnd });
},
methods: {
    receiveUpdate(data){
        for(let delta of data.deltas){
            this.instrument.deltas.push(delta);
            switch(delta.type){
                case UpdateType.Position:
                    this.x = delta.xy.x;
                    this.y = delta.xy.y;
                    break;
                case UpdateType.Size:
                    this.width = delta.xy.x;
                    this.height = delta.xy.y;
                    break;
                case UpdateType.Dots:
                    this.dots = delta.dots;
                    this.redrawDrawings();
                    break;
            }
        }
    },
    sendUpdate(deltas){
        let updateInstrument = new Instrument();
        updateInstrument.id = this.instrument.id;
        updateInstrument.type = InstrumentType.Drawing;
        updateInstrument.deltas = deltas;
        this.$emit("handleUpdateFromClient", updateInstrument);
    },
    resizeDrawings(){
        let aabb = this.getBoundingBox();
        this.x = aabb.LeftUp.x;
        this.y = aabb.LeftUp.y;
        this.width = aabb.RightDown.x - aabb.LeftUp.x;
        this.height = aabb.RightDown.y - aabb.LeftUp.y;
    },

```

```

handleHover(isHover){
    if(!this.isActive){
        this.color = isHover ? "#BBBBBB" : "transparent";
    }
},
handleDebouncing(event){
    window.removeEventListener("resize", this.resizeWindow);
    this.context.clearRect(0, 0, this.width, this.height);
    this.context.beginPath();
    this.context.moveTo(this.dots[0].x, this.dots[0].y);
    for(let dot of this.dots){
        this.context.lineTo(dot.x, dot.y);
    }
    this.context.stroke();
},
handleMouseDown(event){
    this.moving.isFocus = true;
    this.moving.x = event.offsetX;
    this.moving.y = event.offsetY;
    this.$emit("handleObjClick", { focus: this.callbackFocus, unfocus:
this.callbackUnfocus });
    this.$emit("handleObjDrag", { move: this.handleMouseMove, up:
this.handleMouseUp });
},
handleMouseUp(event){
    if(this.moving.isFocus){
        this.moving.isFocus = false;
        this.sendUpdate([{ type: UpdateType.Position, xy: { x: this.x,
y: this.y } }]);
    }
},
handleDrawingEnd(event){
    if(this.moving.isActive){
        this.moving.isActive = false;
        this.resizeDrawings();
        for(let i = 0; i < this.dots.length; ++i){
            this.dots[i].x -= this.x;
            this.dots[i].y -= this.y;
        }
        this.redrawDrawings();
        this.sendUpdate([
            {type:UpdateType.Position, xy:{x: this.x,y: this.y}},
            {type:UpdateType.Size, xy:{x: this.width,y: this.height}},
            {type: UpdateType.Dots, dots: this.dots}
        ]);
    }
},
handleDrawingMove(event){
    if(this.moving.isActive){
        this.context.lineTo(event.pageX, event.pageY);
        this.context.stroke();
        this.dots.push({ x: event.pageX, y: event.pageY });
    }
},
handleMouseMove(event){
    if(this.moving.isFocus){

```

```

        this.y = event.pageY - this.moving.y;
        this.x = event.pageX - this.moving.x;
    }
},
handleDbClick(event){
    this.editable = true;
},
callbackFocus(text){
    this.isActive = true;
    this.color = "#777777";
},
callbackUnfocus(text){
    this.isActive = false;
    this.color = "transparent";
    this.editable = false;
},
getBoundingBox(){
    let leftUp = { x: this.width, y: this.height };
    let rightDown = { x: 0, y: 0 };
    for(let dot of this.dots){
        if(dot.x < leftUp.x){
            leftUp.x = dot.x;
        }
        if(dot.y < leftUp.y){
            leftUp.y = dot.y;
        }
        if(dot.x > rightDown.x){
            rightDown.x = dot.x;
        }
        if(dot.y > rightDown.y){
            rightDown.y = dot.y;
        }
    }
    return { LeftUp: leftUp, RightDown: rightDown };
}
},
computed: {
    style(){
        return{
            position: "absolute",
            width: this.width + "px",
            height: this.height + "px",
            top: this.y + "px",
            left: this.x + "px",
            border: "1px solid " + this.color,
            borderRadius: "5px"
        }
    }
},
props: ["top", "left", "id", "maxWidth", "maxHeight", "isUpdated"]
};
</script>

```