

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ЗАСОБІВ СПІЛЬНОГО РЕДАГУВАННЯ JAVASCRIPT КОДУ
В РЕАЛЬНОМУ ЧАСІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF TOOLS FOR CO-
EDITING A JAVASCRIPT CODE
IN REAL TIME

Виконав: студент(ка) 2 курсу, групи КНТ-129м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Сергєєв О.В.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Сергєєва Олександра Володимировича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація засобів спільного редагування javascript коду в реальному часі. Research and Software Implementation of Tools for Co-Editing a Javascript Code in Real Time

керівник проєкту (роботи) Федорончак Тетяна Василівна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “3” листопада 2020 року
 № 301

2. Строк подання студентом проєкту (роботи) 5 грудня 2020 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання на роботу

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Аналіз та вибір засобів для розробки вебзастосунку. 3. Програмна реалізація вебзастосунку. 4. Експлуатація, тестування та експериментальне дослідження програми. 5. Економіко-організаційна частина. 6. Охорона праці та безпека в надзвичайних ситуаціях. 7. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Федорончак Т.В., доцент		
Економіко-організаційна частина	Пожуєва Т.О., професор		
Охорона праці та безпека в надзвичайних ситуаціях	Коробко О.В., ст. викладач		
Нормоконтролер	Андрєєв М.О., асистент		

7. Дата видачі завдання “ 4 ” вересня 2020 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Розробка економіко-організаційної частина	10 тиждень	Розділ 6
8	Розробка заходів з охорони праці та безпеки в надзвичайних ситуаціях	11 тиждень	Розділ 7
9	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
10	Нормоконтроль та рецензування.	14–15 тижні	
11	Захист роботи.	16 тиждень	

Студент(ка)

(підпис) Сергєєв О.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Федорончак Т.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
136 с., 16 табл., 51 рис., 2 дод., 20 джерел.

РЕДАКТОР КОДУ, JAVASCRIPT, СПІЛЬНЕ РЕДАГУВАННЯ,
ОПЕРАЦІЙНЕ ПЕРЕТВОРЕННЯ, ВЕБЗАСТОСУНОК.

Об'єкт дослідження – спільна розробка алгоритмів та методів на мові JS для вирішення конкретних задач та цілей.

Предмет дослідження – засіб для можливості спільної розробки алгоритмів та методів мовою JavaScript в реальному часі.

Мета роботи – створення вебзастосунку для спільного редагування програмного коду на мові програмування JS в реальному часі.

Актуальність роботи пояснюється тим, що попит на розробників вебзастосунків, що передбачає використання мови JS з плином часу тільки збільшується.

В процесі розробки було використано мову програмування JS для реалізації клієнтської частини вебзастосунку та мову JS з використанням програмної платформи NodeJS для написання серверної частини вебзастосунку, інтегровану середу розробки Visual Code та бібліотеку Code-Mirror в якості універсального текстового редактору.

Результати. Створено вебзастосунок для спільного написання та редагування програмного коду на мові програмування JS в реальному часі що дозволяє створювати швидкі прототипи та алгоритми при командному вирішенні задач, тестуванні та демонстрації логічних блоків застосунку; допомагати знаходити рішення проблем віддаленим товаришам та організовувати навчання.

Висновки. Досліджено засоби, що дозволяють організувати спільне редагування коду в реальному часі, розроблено вебзастосунок, що використовує ці засоби для надання можливості спільного редагування коду в реальному часі.

Галузь використання – спільна розробка алгоритмів та прототипів для подальшого використання в будь-якій галузі, навчальна практика; віддалена допомога.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 136 pages, 51 figures, 16 tables, 2 appendixes, 20 sources.

CODE EDITOR, JAVASCRIPT, COLLABORATIVE EDITING, OPERATIONAL TRANSFORMATION, WEB APPLICATION.

The object of research is the joint development of algorithms and methods in the JS language to solve specific problems and goals.

The subject of the research is a tool for the possibility of joint development of algorithms and methods in JavaScript in real time.

The purpose of the work is to create a web application for joint editing of program code in the JS programming language in real time.

The relevance of the work is due to the fact that the demand for web application developers, which involves the use of the JS language over time, only increases.

The development process used the JS programming language to implement the front end of the web application and the JS language using the NodeJS software platform to write the backend part of the web application, the Visual Code integrated development environment and the Code-Mirror library as a universal text editor.

Results. Created a web application for joint writing and editing of program code in the JS programming language in real time that allows you to create rapid prototypes and algorithms for team solving problems, testing and demonstration of logical blocks of the application; help remote teammates find solutions to problems and organize training.

Conclusions. The tools that allow you to organize real-time code editing have been explored, and a web application has been developed that uses these tools to enable real-time code editing.

Field of use - joint development of algorithms and prototypes for further use in any field; educational practice; remote assistance.

ЗМІСТ

	С.
ВСТУП.....	12
1 Аналіз проблеми та постановка завдань дослідження	14
1.1 Парне програмування як один із варіантів спільної розробки ПЗ	14
1.1.1 Переваги парного програмування	14
1.1.2 Виклики парного програмування	16
1.2 Моб-програмування як один із варіантів спільної розробки ПЗ.....	17
1.2.1 Рішення класичних проблем при моб програмуванні.....	18
1.3 Відхід від принципу використання одного комп'ютера при груповій розробці коду	19
1.3.1 Поняття редактора коду	19
1.3.2 Онлайн редактор коду	20
1.4 Огляд деяких редакторів коду	22
1.4.1 Codesandbox.....	23
1.4.2 Playcode	24
1.4.3 Codepen	24
1.4.4 Stackblitz.....	25
1.5 Висновок та постановка задачі	26
2 Аналіз та вибір засобів для розробки вебзастосунку	27
2.1 Редактор тексту	27
2.1.1 CodeMirror.....	28
2.1.2 Ace	29
2.1.3 Monaco Editor.....	30
2.1.4 Порівняння редакторів	31
2.2 Вибір NodeJS в якості основи для побудови серверної частини	35
2.2.1 ExpressJS	36
2.2.2 KoaJS	37
2.3 Вибір технології для створення клієнтської частини.....	38

2.3.1	React.....	38
2.3.2	VueJS	40
2.4	Вибір інструментів для створення вебзастосунку та висновки розділу .	42
3	Програмна реалізація вебзастосунку	43
3.1	Операційне перетворення.....	43
3.1.1	СС модель узгодженості	44
3.1.2	ССІ модель узгодженості	47
3.1.3	Модель ОП.....	48
3.1.4	Функції перетворення.....	49
3.1.5	Критика ОП	49
3.1.6	Вибір ShareDB в якості бази даних на основі ОП	50
3.2	Архітектура розробленого вебзастосунку	50
3.2.1	Серверний модуль.....	53
3.2.2	Клієнтський модуль	57
3.3	Розроблений графічний інтерфейс	61
3.3.1	Сторінки	61
3.3.2	Компонент редактора	62
3.3.3	Компонент відображення результату	64
3.3.4	Компонент верхньої панелі.....	64
3.3.5	Компонент повідомлень	65
3.4	Висновок	66
4	Експлуатація, тестування та експериментальне дослідження програми	68
4.1	Призначення й умови застосування програми.....	68
4.2	Характеристики вебзастосунку	68
4.3	Вхідні і вихідні дані	69
4.4	Виконання програми.....	70
5	Економіко-організаційна частина.....	78
5.1	Опис продукту що розробляється	78
5.2	Планування розробки	83

5.3	Визначення витрат на розробку.....	85
5.3.1	Розрахунок основної заробітної плати	85
5.3.2	Розрахунок додаткової заробітної плати.....	85
5.3.3	Відрахування на єдиний соціальний внесок	86
5.3.4	Визначення витрат на матеріали	87
5.3.5	Витрати на ПК.....	88
5.3.6	Витрати на сервер	91
5.3.7	Накладні витрати	93
5.3.8	Розрахунок експлуатаційних витрат.....	94
5.3.9	Розрахунок витрат на адміністратора.....	94
5.4	Розрахунок капітальних вкладень	95
5.5	Оцінка економічної ефективності	96
6	Охорона праці та безпека в надзвичайних ситуаціях.....	98
6.1	Аналіз потенційних небезпек.....	98
6.2	Заходи із забезпечення безпеки	100
6.3	Заходи з виробничої санітарії та гігієни праці.....	104
6.4	Заходи безпеки у надзвичайних ситуаціях	108
6.4.1	Заходи з пожежної безпеки.....	108
6.5	Заходи з цивільного захисту	109
	Висновки	1133
	Перелік джерел посилання	113
	Додаток А Текст програми.....	116
	Додаток Б Слайди презентації	129

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ОП – Операційне перетворення;

JS – Мова програмування JavaScript.

ВСТУП

На сьогоднішній день JS (JavaScript) входить в число найпопулярніших мов програмування; в першу чергу JS широко використовується у фронтенд-розробці. Ця мова разом з HTML і CSS входить в базовий набір інструментів фронтендера, а фронтенд-розробка вважається одним з найпростіших способів почати кар'єру в IT.

У липні 2020 року JS посідає сьоме місце в індексі TIOBE, який складається на основі статистики пошукових запитів - мова популярніша, ніж PHP, Swift, Ruby, Go. JS посів перше місце за кількістю пулреквестів на GitHub в другому кварталі 2020 року [1].

На рахунок актуальності також можна сказати, що JS використовується у всіх сучасних популярних браузерах та у найближчому майбутньому буде залишатися основною мовою програмування фронтенд-додатків, а ще, сфери застосування JS не обмежуються браузерами і вебзастосунками. За допомогою цієї мови вирішують такі завдання як: розробка нативних застосунків, серверна розробка, розробка десктопних додатків, програмування обладнання та побутової техніки, наприклад, платіжних терміналів, телевізійних приставок.

Це все приводить до щорічного збільшення JS громади та бажаючих вивчати мову і використовувати її в якості основної мови програмування в своїй роботі. Під час роботи програміст не рідко використовує не тільки свою налаштоване середовище розробки, а і популярні рішення для розробки програмного коду в реальному часі, використовуючи вебзастосунки.

У випадках спільної розробки функціональної частини вебзастосунку, чи якогось алгоритму або методу для вирішення існуючої проблеми, було би раціонально припустити, що цю саму розробку потрібно проводити сумісно в одному редакторі коду. На жаль, в реаліях сучасного життя, коли все більше робітників переходять до режиму роботи з дому, у зв'язку з розповсюдження вірусу, спільна розробка в одному редакторі виявляється неможливою. Саме тут

на допомогу приходить вебзастосунки що дозволяють дистанційно та спільно редагувати код в реальному часі. Переваги такого способу важко переоцінити, так як його вдала реалізація дозволяє зменшити час відклику в процесі спільної розробки, тим самим, в перспективі, прискорити цю саму розробку та її якість, порівняно зі способом, коли декілька розробників, у випадку необхідної спільної роботи, лише обмінюються готовим кодом через месенджери чи будь-які інші канали зв'язку.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Парне програмування як один із варіантів спільної розробки ПЗ

Багато людей, які сьогодні працюють над розробкою програмного забезпечення, чули про практику парного програмування, але вона все ще має лише неоднозначне прийняття в галузі. Однією з причин його різного прийняття є те, що його переваги не відразу очевидні, вони приносять більше результатів у середньо- та довгостроковій перспективі. І це також не так просто, як «двоє людей, що працюють за одним комп'ютером», тому багато хто швидко відмовляється від цього, коли стає некомфортно. Однак, на досвіді багатьох, парне програмування життєво важливо для спільної роботи в команді та створенні високоякісного програмного забезпечення [2].

Парне програмування по суті означає, що двоє людей пишуть код разом на одній машині. Це дуже контактний спосіб роботи, який передбачає багато спілкування. Хоча пара розробників працює над завданням разом, вони не лише пишуть код, а також планують і обговорюють свою роботу. Вони уточнюють ідеї на шляху, обговорюють підходи та приходять до кращих рішень.

1.1.1 Переваги парного програмування

Розглянемо, для чого потрібно парне програмування. Поінформованість про всі його потенційні переваги важлива, щоб вирішити, коли саме вам потрібно це використовувати, як робити це добре. Основними цілями, які може допомогти вам досягти парне програмування, є якість програмного забезпечення та налагоджений потік роботи в команді.

Найочевидніша та найменш спірна перевага - обмін знаннями. Наявність двох осіб, які працюють над алгоритмом, допомагає команді поширювати

знання про технології та предмету область на щоденній основі і запобігає розрізненню знань. Більш того, коли дві незалежні людини аналізують і обговорюють проблему, вони підвищують шанси знайти хороше рішення. Різний досвід і точки зору приведуть до розгляду більшої кількості альтернатив.

Парне програмування змушує розробників обговорювати підходи і рішення, а не тільки обмірковувати їх у своїй голові. Проговорення і пояснення речей вголос підштовхує до роздумів, чи дійсно у команди є правильне розуміння або гарне рішення. Це відноситься не тільки до коду і технічного дизайну, але також до користувацької історії і цінності, яку вона для нього приносить.

Набагато простіше мати структурований підхід, коли розробників двоє: кожен з них повинен чітко повідомляти, чому він щось робить і рухається. Працюючи поодиноці, набагато легше відволіктися, наприклад бажаючи швидко спробувати інший підхід, детально перед цим його не обдумав. Напарник може перешкодити потрапити в таку ситуацію і зосередитися на тому, що важливо для завершення завдання. Такий підхід дозволяє один одному не збитися зі шляху.

Ще однією перевагою парного програмування є перевірка коду. При об'єднанні в пари, у з'являються “додаткові очі”, що призводить до більшого виявлення помилок в процесі розробки, а не після того, як задачу було закінчено.

Рефакторинг коду завжди є частиною розробки і, отже, парного програмування. Коли поруч з розробником є хтось, йому гораздо легше поліпшити код, тому що він можете, наприклад, обговорити різні підходи в створенні або найменуванні речей.

Оскільки об'єднання в пари сприяє обміну знаннями, воно може допомогти в адаптації нових членів команди. Нові учасники можуть ознайомитися з проектом, бізнесом і організацією за допомогою свого партнера. Зміни в команді впливають на командний потік. Людям просто потрібен час,

щоб дізнатися один одного. Парне програмування може допомогти звести до мінімуму цей вплив, тому що воно змушує людей спілкуватися набагато більше, ніж їм потрібно при роботі поодиноці.

Об'єднання в пари дозволяє вам поглянути на код з різних точок зору, а саме: «тактичний» режим з обмислюванням конкретних деталей реалізації та «стратегічний» режим, що дозволяє розглядати загальну картину. Поєднання тактичного і стратегічного погляді дозволяє підвищити якість коду, адже це дозволить розробникам звертати увагу на деталі при цьому тримаючи в голові цільну картину того, що відбувається.

1.1.2 Виклики парного програмування

Хоч парне програмування і має багато переваг, воно також вимагає практики і може не працювати гладко з самого початку.

Працюючи на самоті, людина може робити перерви, коли захоче, і її розум може відволіктися або ненадовго відключитися, коли це необхідно. Розробка чогось спільно навпаки змушує зосереджуватися на потенційно більш тривалих проміжках часу роботи і знайти спільну мову з ритмом і способом мислення іншої людини. Підвищена концентрація уваги - одна з переваг, але це також може виявитись досить інтенсивним і виснажливим.

Працювати в тісному контакті з іншою людиною протягом тривалого часу - це напруження. Парі потрібно постійно спілкуватися, а це вимагає співчуття і навичок міжособистісного спілкування.

У розробників можуть бути різні сторони в техніці, знаннях, навичках, екстраверсії, особистості чи підходах до вирішення проблем. Деякі з цих пунктів можуть не збігатися і привести до досить важкого старту. В цьому випадку потрібно витратити якийсь час на те, щоб поліпшити співпрацю та зробити його взаємним навчанням, а не боротьбою.

Зустрічі необхідні для обговорення, планування і узгодження речей, які необхідно побудувати, але, з іншого боку, вони переривають потік самої роботи. Коли команда практикує парне програмування, ефект від занадто великої кількості зустрічей може навіть погіршитися. Якщо у кожного члена пари зустрічі в різний час, кількість перерв збільшується до безкінечності.

Коли двоє людей з різним рівнем досвіду об'єднуються по одній темі, це часто призводить до помилкових припущень про те, який внесок кожен з них може внести, або розчарування через різницю в темпі роботи.

Постійні розмови з однією людиною також можуть неабияк виснажити сили. Більшості людей протягом дня також потрібен час для самостійного заняття, що особливо актуально для інтровертів.

Працюючи поодиноці, люди цілком природно знаходять час, щоб заглибитися в тему або навчитися чомусь, коли це потрібно. Але це може здатися непотрібним перериванням процесу розробки в парі [3].

1.2 Моб-програмування як один із варіантів спільної розробки ПЗ

Моб-програмування або мобінг - це підхід до розробки програмного забезпечення, при якому вся команда цілком працює над одним і тим же, в один і той же час і в одному і тому ж місці за одним комп'ютером. Це схоже на парне програмування із попереднього розділу. При моб-програмуванні спільна робота поширюється на всіх в команді, при цьому все ще використовується тільки один комп'ютер для написання коду [4].

У мобінг є також як і в парному програмуванні дві основні ролі: драйвер і навігатор. Навігатор дає інструкції, а завдання драйвера не думати, а виконувати вказівки навігатора. Якщо навігатор не знає, що робити далі, він звертається за допомогою до мобу - інших учасників команди, які спостерігають за тим, що відбувається на екрані і завжди в контексті задачі.

Кожні "N" хвилин ролі змінюються і команда рухається за годинниковою стрілкою.

1.2.1 Рішення класичних проблем при моб програмуванні

Спільна природа моб програмування викликає безліч проблем, з якими зазвичай стикаються команди розробників програмного забезпечення. Ці проблеми включають наступні.

Наприклад такі як очікування відповідей на питання, час, що витрачається на обмін повідомленнями по електронній пошті, і ймовірність непорозуміння, що виникає при спілкуванні за допомогою документації. Всі ці проблеми зменшуються або усуваються спільним особистим спілкуванням в Mob Programming.

Для моб програмування характерні такі проблеми, як небажання приймати рішення і необхідність відстоювати прийняті рішення. Оскільки рішення приймає вся команда, ці рішення приймаються на основі найбільш актуальної інформації. Також команди, що використовують групове програмування, розглядають рішення як можливість вчитися і постійно йти до найкращого рішення.

Коли команда працює над чимось одним і прагне кожен день використовувати щось на практиці, команда прагне спростити завдання і не допускати додавання непотрібних речей. Залучення всієї команди в ці щоденні зусилля запобігають ненавмисному додаванню чого-небудь більшого, ніж дійсно необхідно.

Моб-програмування змушує всіх її членів пильно стежити за поточною роботою і різко знижує шанси появи багатьох типів важких в подальшій технічній підтримці рішень [5].

Ще одним недоліком такого підходу є погана політика різних рівнів розробників. Вона полягає в тому наскільки люди будуть допомагати один

одному, і може згубно позначитися на ефективності команд. Моб-програмування підсилює роботу команди і спонукає людей зрозуміти, як вони найкраще підходять для допомоги всій команді.

Завдяки використанню постійних «робочих зустрічей» моб-програмування забезпечує об'єднання роботи, накопичення знань і прийняття рішень, що призводить до набагато краще погодженням і більш ефективним діям.

1.3 Відхід від принципу використання одного комп'ютера при груповій розробці коду

Сучасні проблеми потребують сучасних рішень. В реаліях сучасного світу, коли все більше робітників переходять до режиму роботи з дому, у зв'язку з розповсюдження вірусу, спільна розробка на одному комп'ютері виявляється неможливою. Саме тут на допомогу і приходять вебзастосунки, більш відомі як редактори коду онлайн, що дозволяють дистанційно та спільно редагувати код в реальному часі.

Переваги такого способу важко переоцінити коли мова йде про невеликі за розміром задачі, так як його вдала реалізація дозволяє зменшити час відклику в процесі спільної розробки, тим самим, в перспективі, прискорити цю саму розробку та її якість, порівняно зі способом, коли декілька розробників, у випадку необхідної спільної роботи, лише обмінюються готовим кодом через git, месенджери чи будь-які інші канали зв'язку або проводять сесію програмування використовуючи демонстрацію свого екрану.

1.3.1 Поняття редактора коду

Редактор коду — текстовий редактор для створення, та редагування коду програм. Він може бути окремим додатком, або інтегрованим в середовище

розробки (IDE). Редактори коду мають деякі можливості, які спрощують та прискорюють написання та редагування коду, такі як підсвітка синтаксису, автодоповнення, перевірка правильності розстановки дужок, контекстна допомога по коду та багато інших. Такі редактори надають зручний спосіб для запуску компілятора, інтерпретатора, налагоджувача або інших програм необхідних у процесі розробки програмного забезпечення. Незважаючи на те, що багато текстових редакторів можуть бути використані для редагування тексту програм, якщо вони не мають розширених можливостей, що автоматизують або спрощують введення та модифікацію коду, то вони не можуть називатися «редакторами початкового коду», а просто є «текстовими редакторами», які також можуть бути використані для редагування програм.

До прикладів існуючих редакторів коду можна віднести наступні: AkelPad, Atom, Eclipse, Emacs, Geany, IntelliJ IDEA, Light Table, Visual Studio Code, Notepad++, PSPad, Embarcadero RAD Studio, RJ TextEd, Sublime Text, vi и vim, тощо.

Не будемо зупинятись на детальному розгляді представників із списку вище, так як в контексті дипломної роботи більший інтерес викликають їх онлайн аналоги.

1.3.2 Онлайн редактор коду

Онлайн редактор коду - це все такий же текстовий редактор для створення, та редагування коду програм, с той тільки різницею, що він знаходиться на віддаленому сервері, доступ до якого можливий за допомогою браузера. Онлайн редактор коду так само володіє можливостями для спрощення та прискорення написання та редагування коду, наприклад: підсвітка синтаксису, автодоповнення, перевірка правильності розстановки дужок, контекстна допомога по коду та багато інших. Деякі онлайн редактори коду

мають функціонал, схожий зі звичайними текстовими редакторами, а інші більше нагадують повноцінні IDE.

Конкурентний функціонал онлайн-редакторів коду:

- можливість ділитися кодом;
- настройки макета;
- настройки табуляції;
- коментування;
- перегляд результатів;
- базова файлова структура.

Можна виділити наступні переваги онлайн-редакторів коду:

- не потрібно нічого налаштовувати. Вам не доведеться завантажувати, встановлювати та налаштовувати IDE;

- можливість поділитися кодом і спільно працювати над ним.

Почніть писати код і відправте колезі посилання, щоб він міг продовжити розпочате або допоміг з налагодженням;

- редакторами можна користуватися безкоштовно або за невелику плату. Переважна більшість онлайн-редакторів мають безкоштовні версії, функціоналу яких буде достатньо для звичайних повсякденних завдань;

- ціна на основі передплати. Може бути зручно платити за користування редактором раз на місяць, а не купувати ліцензію на рік.

До переваги онлайн-редакторів коду можна віднести наступне:

- обмеженість функціоналу. У деяких редакторах коду відсутній навіть базовий функціонал;

- проблеми з продуктивністю;

- відсутність контролю версій. Переважна більшість онлайн-редакторів коду не мають ніяких інтеграцій з інструментами git;

- немає інтернету - немає розробки.

Розглянемо для чого використовуються онлайн редактори коду.

За допомогою онлайн-редакторів можна поділитися кодом з колегами, друзями, та й цілим світом. Можна розміщувати вбудовані уривки коду або посилання на «пісочницю» в своєму блозі, посібниках, документації.

Деякі редактори коду мають інтеграцію з популярними хостингами, завдяки чому можна зробити простий додаток і буквально за лічені хвилини розмістити його на хостингу, щоб протестувати.

За допомогою онлайн-редакторів коду можна проводити співбесіди. Таким чином інтерв'юер може побачити, як кандидат підходить до вирішення завдань і як він працює з кодом.

Ідеї потрібно тестувати якомога раніше і якомога частіше. Перевіряючи свої ідеї, з'являється можливість отримувати негайний фідбек від користувачів. Функціонал, призначений для демонстрації і спільної роботи, допоможе з цим.

У сучасній веброзробці потрібно постійно стежити за останніми новинами, а ще краще - відчувати новинки в реальних умовах. Онлайн-редактори коду дозволяють це робити швидко і без непотрібної метушні з настройками.

Налагодити спільну роботу з іншою людиною, використовуючи десктопні IDE, досить складно. А онлайн-редактори полегшують цей процес. Робота з ними нагадує роботу з Google Docs.

1.4 Огляд деяких редакторів коду

Саме спільна розробка і є об'єктом дослідження в кваліфікаційній роботі магістра. Тобто для створення чогось актуального та потрібного, необхідно розглянути існуючі приклади онлайн редакторів коду.

1.4.1 Codesandbox

Codesandbox позиціонує себе як платформу для розробки. Спільнота Codesandbox створила безліч шаблонів. Крім того, ви можете з легкістю створювати власні шаблони для своїх специфічних потреб, з файловою структурою і залежностями, як в IDE. Інтерфейс застосунку зображено на рис.1.1 [6].

Цей інструмент інтегрований з GitHub, так що не виникне проблем зі створенням коммітів і відкриттям пул-реквестов.

Для розробників клієнтської частини є підтримка NPM.

Цей онлайн-редактор має багато прекрасних функцій:

- консоль;
- вбудований термінал;
- автозапуск Jest-тестів;
- перевантаження модуля на зміну коду;
- організація проєкту;
- вбудований Линтер (Eslint).

Що стосується ціни, Про-версія коштує 9 доларів. Вона включає приватні GitHub-репозиторії та приватні кімнати.

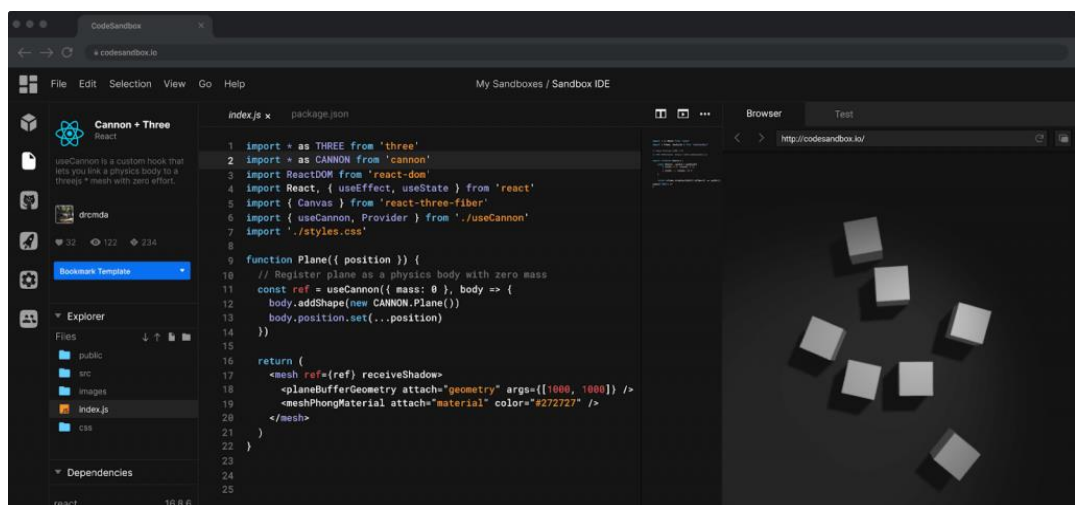


Рисунок 1.1 – Інтерфейс CodeSandbox

1.4.2 Playcode

Playcode це простий інструмент для швидкого прототипування і перегляду результатів написання коду. Він має класичний трьохвіконний макет: редактор коду, консоль і вікно перегляду результату. Продукт має базову файлову структуру, але в ньому немає контролю версій і інших функцій IDE.

Playcode підтримує тільки JavaScript, HTML і CSS. У ньому є можливість вибрати стиль редактора, розмір шрифту та інші речі, що стосуються редагування тексту. Інтерфейс застосунку наведений на рис. 1.2.

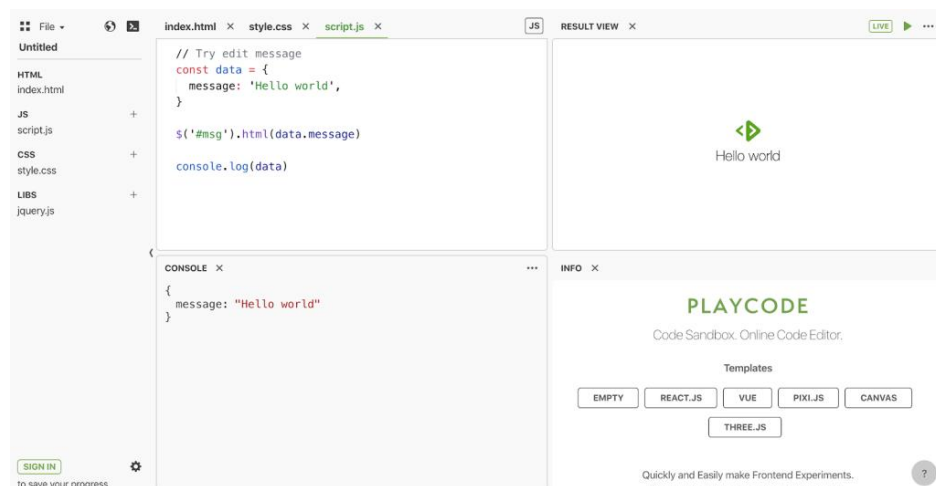


Рисунок 1.2 – Інтерфейс Playcode

1.4.3 Codepen

Codepen це, ймовірно, найпопулярніший інструмент на ринку онлайн-редакторів коду. Ця платформа почала деяким чином перероджуватися в соціальну мережу, подібну Pinterest. Розробники можуть ділитися своїми кращими практиками та отримувати схвалення від інших членів спільноти.

Усередині інструменту є продуманий функціонал пошуку. Ви можете шукати шаблони, проєкти, уривки коду і цікаві для вас теми.

Одна з важливих особливостей Codepen - орієнтація на вивчення нових технологій і кращих підходів.

Ще одна цікава деталь платформи - режим презентації коду. Це дуже корисно на конференціях, мітапах і зборах.

Варто відзначити і інші особливості цієї платформи:

- файлова система;
- автодоповнення коду;
- приватні пісочниці;
- деплоймент сайтів;
- режим спільної роботи, що доступний тільки по платній підписці.

Ціна варіюється від 8 до 26 доларів в місяць. Є також обмежена безкоштовна версія продукту.

1.4.4 Stackblitz

Stackblitz дуже схожий на повноцінну IDE. Продукт має широкий спектр функцій, що дозволяють почати, продовжити і завершити full-stack розробку програми.

Stackblitz автоматично, у міру того як ви вводите код, запускає установку залежностей, компіляцію, зборку і hot reloading.

У веброботці велике значення має імпорт бібліотек, тому StackBlitz включає вбудований в браузер прт-клієнт, що підтримує одночасну установку декількох пакетів і їх специфічних версій.

Головна особливість цього проєкту - можливість офлайн-роботи. Цей функціонал доступний завдяки вебсервера в браузері.

У Stackblitz попередній перегляд і редагування відбуваються в одному розділеному вікні, що є перевагою в порівнянні з iframe і маленькими віконцями інших проєктів.

Також варто відзначити підключення до Github для імпорту і експорту проєктів, інтеграцію з Google Firebase для створення full-stack проєктів і можливість скачування проєкту у вигляді ZIP-файлу.

1.5 Висновок та постановка задачі

Підбиваючи висновки для поточного розділу можна виділити наступне: в сучасному світі що зіткнувся з несподіваною перепорою для фізичного контакту людей у вигляді необхідної ізоляції та дистанціюванню, як ніколи потрібно мати можливість зберегти свою максимальну продуктивність в роботі. А так як розробники досить часто використовують інших розробників як джерело нових знань чи відповідей для рішення цікавлячих задач та прийняття рішень по реалізації спірних моментів, потрібно надати цим самим розробникам зручний у використанні інструмент.

Також в зону потенційних користувачів досить зрозумілим чином попадають різноманітні навчальні практики, будь то навчання інформатиці в школі чи індивідуальні заняття на курсах по програмуванню.

Задача даної роботи саме полягає в тому щоб реалізувати вебзастосунок для спільного написання та редагування програмного коду на мові програмування JS в реальному часі, що дозволить створювати швидкі прототипи та алгоритми при командному вирішенню задач і прийняттю зважених та завірених декількома людьми рішень, демонстрації та тестуванню логічних блоків застосунку; допомагати вирішувати несподівані проблеми віддаленим товаришам по роботі та організовувати інтерактивне навчання для своїх учнів.

2 АНАЛІЗ ТА ВИБІР ЗАСОБІВ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ

2.1 Редактор тексту

Так як головною задачею стоїть вивчення та саме створення можливості спільної розробки програмного коду в реальному часі, вирішено, що сили на те, щоб створити власне свій редактор тексту з нуля прикладати не раціонально.

Проаналізувавши існуючі варіанти засобів інтегрованих редакторів тексту було знайдено трійку найкращих із них. Для того щоб вибрати саме той редактор, що буде найбільш доцільно підходити до задачі цієї роботи, можна виділити дві основні категорії функціоналу, яким повинен володіти цей самий редактор:

- мати можливість з легкістю бути інтегрованим та використаним разом з алгоритмами “спільної розробки”;
- надавати функціонал для створення якомога кращого користувацького досвіду щоб реалізований вебзастосунок міг конкурувати з існуючими подібними рішеннями.

До першої категорії функціоналу треба віднести наступне:

- надавати можливість через програмний інтерфейс редактору зчитувати зміни, що створює користувач до того, як ці самі зміни відобразяться на дисплеї користувача (тобто надання можливості підписатись на `beforeChange` подію). Це потрібно для якомога найкращого та швидкого обміну інформацією з сервером, що контролює спільну розробку в реальному часі для поточної сесії;
- надавати інформацію про зміни в форматі придатному для використання з технологією операційного перетворення (ОП), а не у вигляді повного тексту редактору. Це також необхідно для створення максимально швидкого обміну даними про зміни без потреби попередньої обробки;

- надавати можливість повного та зручного контролю над змістом редактору у вигляді методів для часткової зміни чи вставки тексту в будь якій позиції редактору;

- як приємний бонус надавати вбудоване рішення для відображення дій віддаленого користувача у вигляді, наприклад, курсору та виділення тексту.

До другої категорії функціоналу необхідного для створення якомога кращого користувацького досвіду можна віднести наступний перелік:

- бути браузерно незалежним та підтримувати якомога більшу вибірку версій браузера, для того щоб створюваний вебзастосунок не обмежував користувача у використанні улюбленого чи тимчасово необхідного інструменту, хто знає з якого місця з'явиться потреба до спільної розробки;

- мати вбудоване розуміння програмного коду принаймні на мові програмування JS, тобто надавати можливість з мінімальним зусиллям створити цілковитий редактор коду; або мати можливість за допомогою додаткових, простих в написанні модулів, доповнити свій функціонал цією можливістю;

- мати можливість додавання нових синтаксисів для підсвічування коду для подальшого покращення вебзастосунку і можливості підтримки більшого переліку програмних мов;

- мати підтримку автодоповнення для мови програмування JS чи можливість додати цю підтримку за допомогою модулів чи адонів;

- мати функціонал для відображення підказок в редакторі.

2.1.1 CodeMirror

CodeMirror - це універсальний текстовий редактор, реалізований у JavaScript для браузера. Він спеціалізується на редагуванні коду та має ряд мовних режимів та доповнень, що реалізують розширені функції редагування.

Для налаштування CodeMirror відповідно до потреб створюваного вебзастосунку та його розширення за допомогою нових функцій доступний багатий та простий у використанні програмний інтерфейс.

CodeMirror можна вбудовувати у будь-які вебсторінки. Основна бібліотека забезпечує лише компонент редактора, без супровідних кнопок, автозавершення чи інших функцій IDE. Він надає багатий API, поверх якого така функціональність може бути реалізована без якихось проблем. Він також працює з мовними режимами. Режими - це програми JavaScript, які допомагають помічати кольором (і, за бажанням, відступами) текст, написаний певною мовою програмування.

Багато основних редакторів коду використовують CodeMirror у своїх проєктах. Adobe Brackets - один з основних прикладів, інші приклади включають BlueGriffon (редактор HTML), Cargo Collective (платформа для творчого видавництва), CodeBug, CodePen тощо. CodeMirror насправді є тим самим редактором, який використовується в інструментах розробників для Firefox, Chrome та Safari, у Light Table, Bitbucket та багатьох інших проєктах.

CodeMirror - це приватна ініціативна берлінця Марина Хавербеке (Marijn Haverbeke), якому був потрібен редактор коду для його інтернет-навчальної програми Eloquent JavaScript. Перша версія редактора випущена в 2007 році. Його відкритий код доступний на GitHub.

2.1.2 Ace

Ace - це самостійний редактор коду, написаний на JavaScript. Метою якого є створення вебредактора коду, що відповідає та розширює можливості, зручність використання та продуктивність існуючих власних редакторів, таких як TextMate, Vim або Eclipse. Його можна легко вбудувати в будь-яку вебсторінку та додаток JavaScript. Ace розроблений як основний редактор для Cloud9 IDE і як наступник проєкту Mozilla Skywriter.

Раніше відомий як Bepin або Skywriter, тепер відомий як Ace (Ajax.org Cloud9 Editor). Bepin та Ace розпочали роботу як два незалежних проєкти, обидва яких мали на меті створити безкомпромісний компонент редактора коду для Інтернету. Bepin розпочав роботу в рамках Mozilla Labs і базувався на тегу <canvas>, тоді як Ace є компонентом редактора ID9 Cloud9 і використовує DOM для візуалізації. Після випуску Ace на JSConf.eu 2010 у Берліні команда Skywriter вирішила об'єднати Ace зі спрощеною версією системи плагінів Skywriter та деякими точками розширюваності Skywriter. Усі ці зміни були об'єднані з Ace так щоб повністю замінити Skywriter. Як і Ajax.org, так і Mozilla активно розробляють і підтримують Ace [8].

Ace також використовується як редактор коду в таких відомих проєктах як Codecademy, Khan Academy, MODX.

Більша частина функціональності що підтримується в CodeMirror за допомогою аддонів, працює у Ace з коробки, а саме:

- підсвічування синтаксису програмної мови;
- автовідступ;
- автодоповнення коду;
- теми (можуть бути імпортовані теми TextMate);
- пошук і заміна з використанням регулярних виразів;
- виділення відповідних дужок;
- відображення прихованих символів;
- підсвічування виділеного слова.

2.1.3 Monaco Editor

Monaco Editor - це редактор коду, який керує VS Code. Він не підтримується у мобільних браузерах або мобільних вебфреймворках.

Опублікований Microsoft в наприкінці 2015-го, використовується такими відомими застосунками як IDE GitLab та CodeSandbox.

На даний момент має найбільш розвинені інструменти для взаємодії з редактором коду, має підказки та автокомпліт для більш ніж 20 найбільш популярних мов програмування. Має розумну навігація по коду в контекстному меню.

Має вбудований двохпанельний режим активних змін для простого та зручного інтегрування з системами керування версій. Написаний на TypeScript - ще одному творінню Microsoft. Досить активний процес розробки на момент написання роботи. Швидко правляться баги і зливаються пул-реквести з потрібними поправками.

На офіційному сайті редактора коду знаходиться досить незвичайна але зручна автозгенерована документація з перехресними посиланнями між інтерфейсами та методами, лаконічний API [9].

2.1.4 Порівняння редакторів

Нижче наведено порівняльну таблицю для редакторів тексту CodeMirror, Ace та Monaco Editor (табл. 2.1).

Таблиця 2.1 – Порівняльна таблиця

Порівняння	Ace	CodeMirror	Monaco
1	2	3	4
Розробники	Cloud9 IDE (Ajax.org), нині - частина AmazonMozilla	Marijn Haverbeke	Microsoft

Продовження таблиці 2.1.

1	2	3	4
Підтримка мов та підсвітка синтаксису	>120	>100	>20
Кількість символів у вихідного коду редактору	366 608 (v1.4.3)	394 269 (v5.44.0)	2 064 949 (v0.16.2)
Розмір останніх версій, gzip	2.147 KB	1.411 KB	10.898 KB
Рендеренг	DOM	DOM	DOM та частково <canvas> (для скроллу та minimap)

Продовження таблиці 2.1.

1	2	3	4
Документація	немає пошуку, не завжди зрозуміло, що повертають методи, є сумніви в повноті і актуальності (В документації працюють не всі посилання)	поєднана з керівництвом по початку роботи. Пошук по Ctrl+F, відсутня повна документація.	красива, з пошуком і перехресними посиланнями. Відсутність пояснень до деяких конфігів, застосування яких не цілком очевидно з назви
Керівництво для швидкого початку роботи, демо версії програм	How-to - текстові документи з прикладами коду, окремо є демки з прикладами коду але вони розкидані по різних сторінках сайту та не всі працюють, шукаються простіше через гугл пошук. є досить багато демо версій різноманітних налаштувань але керувати ними	Досить мало How-to в основному все розкидано по github у відповідних папках проєкту є демо версії прикладів функціоналу та коду для їх реалізації	Об'єднані в форматі плейграунда, тобто можна відразу спробувати і оцінити багато можливостей редактору коду. код з коментарями і поруч демо версія для закріплення матеріалу

Продовження таблиці 2.1.

1	2	3	4
Активність спільноти	Середня	Висока	Середня
Активність розробників	Середня	Середня	Висока

Розмір наведено цілковито для демонстрації різниці повного рішення, так як безглуздо вибирати засіб по розміру - все залежить від того, що і як підключати для конкретного проєкту. Досить важливо в якому обсязі використовується редактор: завантажуються чи всі стилі і теми, скільки і яких аддонів і плагінів використано. Наприклад, в CodeMirror велика частина функціональності, яка працює в Monaco і Ace з коробки, доступна тільки з аддонами.

У всіх варіантах реалізації приблизно однаковий набір базового функціоналу: автоформатування коду, згортання рядків, cut / copy / paste, гарячі клавіші, можливість додавання нових синтаксисів для підсвічування і теми. Перевірка синтаксису (в CodeMirror - тільки через аддони, в Ace - поки тільки для JavaScript / CoffeeScript / CSS / XQuery), підказки та автодоповнення (в CodeMirror - через аддони), функціональний пошук по коду (в CodeMirror - через аддони), методи для реалізації табів інструмент для простого інтегрування засобів для контролю версій з (в CodeMirror - в одному вікні, або двохпанельний інтерфейс через аддон, в Ace - окрема бібліотека). Для CodeMirror в силу його віку написано багато аддонів, але їх кількість буде впливати і на вагу, і на швидкість редактора [10].

Враховуючи перераховане вище та потреби для розробки вебзастосунку для спільного редагування коду в реальному часі було здійснено вибір на користь CodeMirror в зв'язку з його простим налаштуванням, багатою

функціональністю повністю задовольняючи потреби вебзастосунку та ще і маючи можливість бути розширеною за допомогою адонів, найбільшим розміром спільноти та великою кількістю прикладів по використанню основного потрібного для застосунку функціоналу, найбільшою підтримкою старих версій браузерів.

2.2 Вибір NodeJS в якості основи для побудови серверної частини

Для створення серверної частини застосунку, що буде відповідати за реалізацію технології ОП (операційного перетворення) підтримку сесії між декількома розробниками та іншого функціоналу, обрано платформу NodeJS, так як вона надає можливість використовувати JS для розробки серверу.

NodeJS - програмна платформа, заснована на двигуні V8 (трансліруючий JavaScript у машинний код), перетворююча JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js надає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API (написаний на C++), підключити інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи визови до нього з JavaScript-коду. Node.js застосовується переважно на серверах, виконуючи роль вебсервера, але також є можливість розробити на Node.js і десктопні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows та macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js и espruino). В основі Node.js лежить подійно-орієнтоване та асинхронне програмування з не блокуючим вводом/виводом.

Якихось особливих вимог до фреймворку для написання серверної частини немає, так як наш вебзастосунок потребуватиме мінімальну кількість створеного API для встановлення зв'язку між декількома користувачами та створенню сесії. Алгоритм для реалізації технології ОП може бути написаний як на чистому NodeJS так і використовувати модульні рішення, які однаково нескладно підключаються до любого фреймворку, що базується на NodeJS. В

зв'язку з цим є потреба розглянути лиш найбільш популярні рішення та обрати інструмент за своїм уподобанням [11].

2.2.1 ExpressJS

Express.js - це фреймворк вебзастосунків, який побудований поверх Node.js. Він забезпечує мінімальний інтерфейс з усіма інструментами, необхідними для створення вебдодатків. Express.js додає гнучкість додатку з величезним набором модулів, доступних на npm, які ви можете безпосередньо підключити до Express відповідно до вимог. Це допомагає легко керувати потоком даних між сервером і маршрутами в серверних додатках. Він був розроблений TJ Holowaychuk і випущений на ринок 22 травня 2010 року. Раніше ним керував IBM, але в даний час він знаходиться під керівництвом Node.js Foundation.

Express в основному відповідає за обробку внутрішньої частини в стеку MEAN. Mean Stack - це стек програмного забезпечення з відкритим кодом JavaScript, який активно використовується для створення динамічних вебсайтів та вебдодатків на ринку. MEAN розшифровується як MongoDB, Express.js, та Node.js.

Основні переваги використання Express.js при розробці вебзастосунків:

- надання підтримки для різних плагінів та розширень для підвищення його функціональності;
- безшовна інтеграція даних;
- легка крива навчання для розробників, які вже знайомі з Node.js;
- механізм маршрутизації на основі URL-адреси за допомогою HTTP;
- міцний, масштабований та повністю налаштовуваний;
- поставляється з MVC, який допомагає у створенні централізованих додатків;

- підтримує 14+ шаблонних механізмів та пропонує чудову продуктивність.

Express підходить для всіх типів розробок вебзастосунків та програм для мобільних пристроїв від малого до корпоративного об'єму. Оскільки цей фреймворк Node.js легко доступний для генерації API, фреймворк має надійний шаблон маршрутизації, функції безпеки та умови обробки помилок, тому розробники можуть легко використовувати його для створення застосунків на рівні підприємства або браузера.

2.2.2 KoaJS

Коа - це великий серверний фреймворк Node.js, який дозволяє розробляти застосунки з найбільшою ефективністю для обробки помилок. Цей фреймворк пропонує так звані генератори, які допомагають уникнути зворотних викликів та конструктивно нормалізують недоліки Node. За допомогою Коа можливо керувати різними типами вмісту за однією і тією ж URL-адресою, наприклад, перекладати сторінку, персоналізувати вміст на вебсайті електронної комерції, керувати різними форматами зображень та багато іншого.

Основні переваги використання KoaJS при розробці вебзастосунків:

- можливість високого рівня налаштування додатків
- наявність генератора ES6 дозволяє розробникам уникати зворотних викликів;
- легка структура, яка в підсумку підвищує продуктивність програми;
- забезпечення модульного підходу до розробки складних вебзастосунків простим способом.

Коа підходить в ситуаціях, коли є потреба у високопродуктивному вебзастосунку. Коа інтегрує з коробки такі компоненти, як узгодження вмісту,

нормалізація розбіжностей вузлів, перенаправлення тощо, тому обробка великих проєктів з Коа набагато простіша ніж з будь-яким іншим фреймворком. Будучи однією з молодих платформ, її спільнота все ще перебуває на зростаючій стадії, але за допомогою відомої компанії з розробки мобільних додатків та досвідченої команди розробників ви можете легко створювати великі програми.

2.3 Вибір технології для створення клієнтської частини

Для того щоб написати гарний в розумінні та подальшому розвитку і підтримці код, вважаю за необхідне використати готовий фреймворк чи бібліотеку для створення користувацьких інтерфейсів.

Як і до серверної частини, кількість вимог невелика та не включає в себе якісь обмеження, але розуміючи масштаб та необхідність в мінімальній кількості функціоналу, доречно було би використати технологію, що надає найбільшу гнучкість у виборі інструментів для розробки якогось функціоналу, а не мати готові вбудовані рішення, так як більшість із них не буде використана та лише займатиме лишнє місце в кінцевому застосунку.

Враховуючи це, доцільно порівняти React та VueJS як найбільш популярні рішення в поточний момент для створення гнучкої та структурованої системи користувацьких інтерфейсів.

2.3.1 React

React - відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови вебзастосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

Особливості React:

- одностороння передача даних – дані передаються в компоненту, як властивості html тегу. Компонент не може напряду змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору»;
- віртуальний DOM - React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити;
- JSX - компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP;
- не лише рендеринг HTML в браузері - React використовують не лише для рендерингу HTML в браузері. Наприклад, Facebook має динамічні

графіки які рендеряться в теги `<canvas>`, Netflix та PayPal використовують ізоморфне завантаження для рендерингу ідентичного HTML на сервері та клієнті;

- методи життєвого циклу - це різні методи, які вбудовуються за допомогою ReactJS. Вони дозволяють розробнику обробляти дані в різних точках життєвого циклу програми React;

- вкладені елементи - кілька елементів на одному рівні повинні бути загорнутими в один елемент контейнера, наприклад елемент `<div>`, або повернутий як масив;

- атрибути - JSX надає ряд атрибутів елементів, призначених для відображення тих, що надаються у форматі HTML. Користувацькі атрибути також можуть бути передані компоненту. Всі атрибути будуть отримані компонентом як реквізит [12].

2.3.2 VueJS

Vue.js - фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів. Легко інтегрується в проєкти з використанням інших JavaScript-бібліотек. Може функціонувати як вебфреймворк для розробки односторінкових додатків в реактивному стилі.

Розробники називають Vue.js прогресивним і поступово адаптуючимся фреймворком в порівнянні з іншими вебфреймворками.

Це дозволяє розробнику налаштувати структуру програми відповідно до власних вимог. Розробники вважають що Vue.js вважається більш простим в освоєнні, ніж AngularJS чи React, оскільки API побудований набагато простіше. Vue.js можна використовувати маючи тільки знання JavaScript, HTML та CSS. Vue.js реалізується шаблон MVVM, пропонує для javascript можливість прив'язки даних, так що висновки і введення даних сполучаються

безпосередньо з джерелом даних. Таким чином, режим ручного визначення даних (наприклад, через jQuery) з HTML-DOM не потрібен.

Особливості VueJS:

- шаблони - Vue використовує синтаксис шаблонів на основі HTML, що дозволяє декларативно зв'язувати рендеринг DOM з основними екземплярами даних в Vue. Всі Vue шаблони валідні HTML, і можуть бути розпарсені браузером та HTML парсерами. Всередині Vue компілює шаблони в рендерингові функції віртуального DOM. В поєднанні з реактивною системою, Vue здатний розумно обчислити кількість компонентів для ре-рендингу та застосувати мінімальну кількість маніпуляцій з DOM, коли стан застосунку зміниться. В Vue ви можете використовувати синтаксис шаблонів або напряду писати рендерингові функції використовуючи JSX. Для того, щоб це зробити просто замініть шаблон на рендерингову функцію.[9] Рендерингова функція відкриває можливості для потужних патернів базованих на компонентах - для прикладу, нова транзитна система тепер повністю базована на компонентах, що використовують рендерингові функції всередині;

- реактивність - одна із найвиразніших особливостей Vue, це ненав'язлива реактивна система. Моделі це просто плоскі JavaScript об'єкти. Це робить керування станами дуже простим та інтуїтивним. Vue надає оптимізований ре-рендеринг з коробки без потреби робити що-небудь додатково. Кожен компонент слідує за своїми реактивними залежностями під час рендерингу, тому система знає точно коли має відбуватись ре-рендеринг і які компоненти потрібно ре-рендерити;

- переходи - Vue надає різноманітні шляхи для застосування ефектів переходу, коли елемент додають, оновлюють або видаляють з DOM;

- роутинг - Vue сам по собі не включає роутингу, та є vue-router пакет, який вирішує це питання. Він підтримує зв'язування вкладених шляхів з вкладеними компонентами і пропонує деталізований контроль над переходами. Vue дозволяє створення додатків за допомогою компонентів. Якщо додати vue-

router до цього, все що потрібно зробити це зв'язати ваші компоненти з роутами і дозвольте vue-router вирішувати де їх рендерити [13].

2.4 Вибір інструментів для створення вебзастосунку та висновки за розділом

Проаналізувавши три найбільш популярних текстових редактора для створюваного вебзастосунку, я зупинився на виборі CodeMirror як найбільш зрілому, протестованому та піддатливому до розширення текстовому редакторі, що можна вбудовувати у будь-які вебсторінки. Основна бібліотека забезпечує лише компонент редактора, без супровідних кнопок, автозавершення чи інших потрібних функцій IDE, але вона також надає і багатий API, поверх якого така функціональність може бути реалізована в найбільш зручній формі без будь-яких проблем.

Для вибору фреймворку для створення серверної частини вебзастосунку, було проаналізовано три найбільш популярних NodeJS фреймворків для створення серверної частини та прийнято рішення використовувати ExpressJS, так як він підходить для всіх типів розробок вебзастосунків. А також цей фреймворк легко доступний для генерації API, він має надійний шаблон маршрутизації, функції безпеки та умови обробки помилок, нічого більшого для створення серверної частини вебзастосунку не потрібно.

React був обраний як засіб для створення інтерфейсів користувача, так як він в порівнянні з VueJS надає найбільшу гнучкість у виборі інструментів для вирішення різноманітних задач та має чіткий, декларативний спосіб створення інтерфейсу, що робить код більш передбачуваним і спрощують його налагодження.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Операційне перетворення

Найголовнішою частиною розробленого додатку є технологія операційного перетворення [15], яку ми детально розглянемо далі. Її вдала реалізація найбільшим чином впливає на роботу всієї системи вебзастосунку.

Основною ідеєю ОП є перетворення (або зміна) параметрів операцій редагування в злагоді з ефектами виконання в попередніх одночасних операціях так, щоб перетворена операція могла спрацювати коректно і не порушувати узгодженість в документі.

Розглянемо ОП більше детально.

ОП - це алгоритм чи техніка для перетворення операцій таким чином, що їх можна застосовувати до документів, стани яких розходились, повертаючи їх до одного угодженого стану.

Описуючи коротко те, як працює ОП, можна сказати наступне:

- кожна зміна (вставка або видалення) представляється як операція. Операція застосовується до поточного документа, що приводить до оновлення стану документа;
- для обробки паралельних операцій використовується функція перетворення, що приймає на вхід дві операції, які були застосовані до одного і того ж стану документа (але для різних клієнтів), і обчислює нову операцію, яка може бути застосована після другої операції і яка зберігає заплановані зміни першої операції.

Схематичний приклад наведено на рис. 3.1.

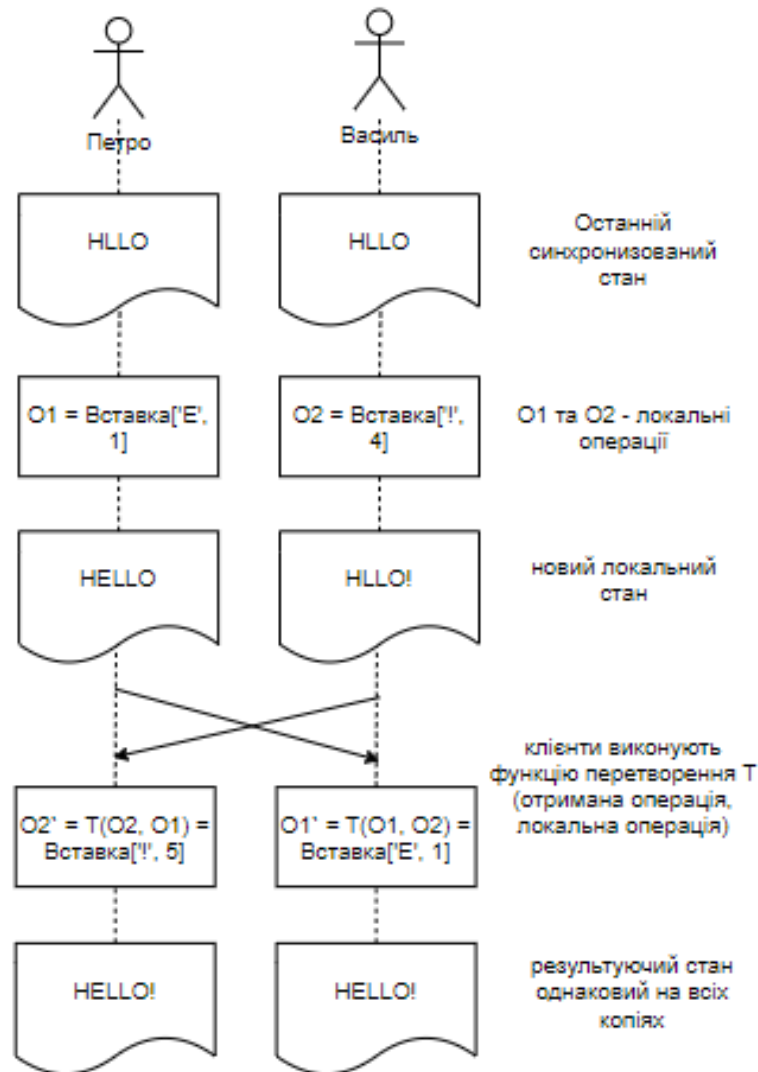


Рисунок 3.1 – Схема роботи ОП

Для забезпечення узгодженості результуючого стану в системах спільного редагування існує декілька моделей узгодженості, розглянемо дві найпопулярніші із них.

3.1.1 СС модель узгодженості

СС модель забезпечує виконання двох властивостей [16].

Перша властивість – збіжність: властивість, при якій копії загального документа повинні бути ідентичними після виконання всіх операцій до документа. Схематичне зображення на рис. 3.2.

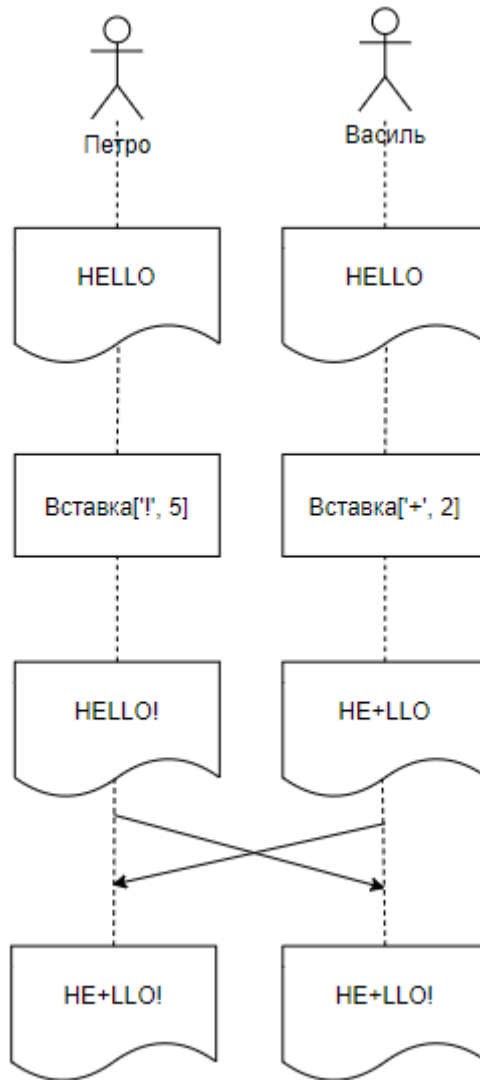


Рисунок 3.2 – Збіжність в ОП

В даному прикладі Петро та Василь починають з ідентичних копій. Потім кожен з них вирішує змінити документ і копії розходяться - так досягається мінімальний час відгуку. Після відправки локальних змін іншим клієнтам властивість збіжності вимагає, щоб підсумковий стан документа у всіх клієнтів був ідентичним.

Друга властивість - збереження причинності, це означає, що операції повинні бути виконані в причинно-наслідковому порядку (грунтуючись на відношенні відбулося-до). Схематичне зображення на рис 3.3.

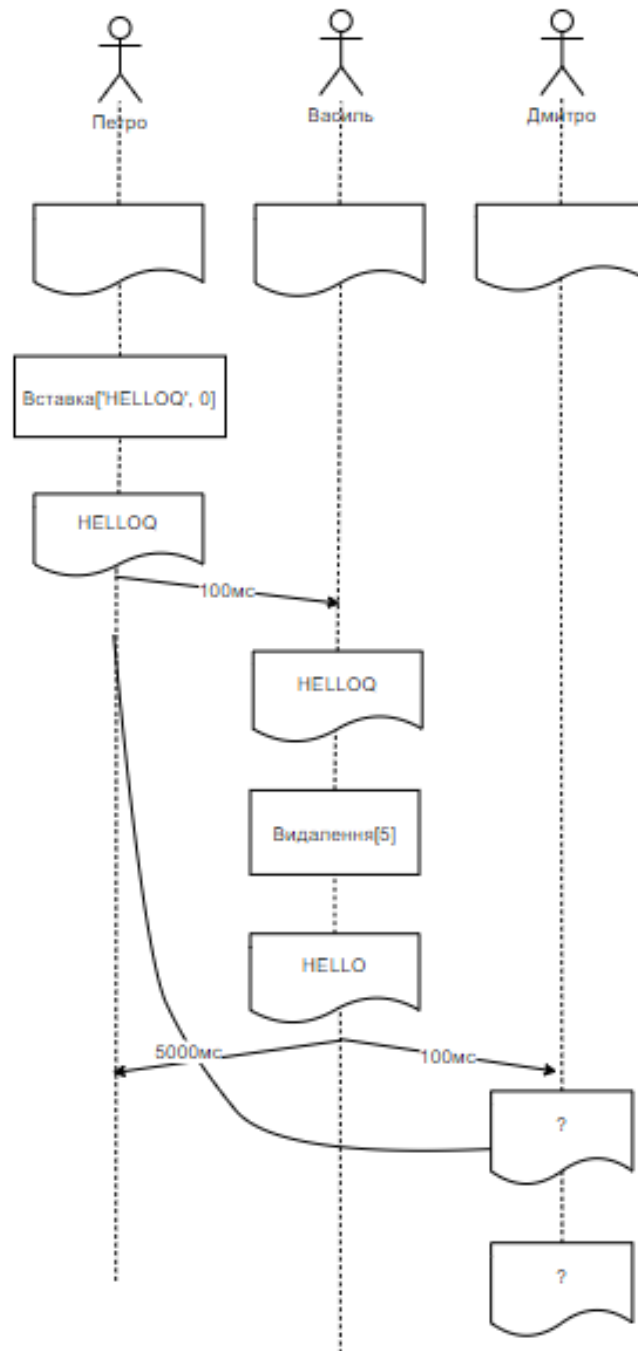


Рисунок 3.3 – Збереження причинності в ОП

Як ми бачимо, Петро відправив Василю та Дмитру свою зміну документа, Василь отримав його першим і виконав свою зміну - видалив останній символ. Через мережевий лаг Дмитро отримує першим операцію Василя на видалення символу, якого ще не існує, який повинен бути поточний стан - невідомо.

ОП не може забезпечити виконання властивості збереження причинності, тому для цих цілей використовують такий алгоритм, як векторний годинник - алгоритм, що використовується для визначення часткового порядку подій в розподіленій системі і виявлення порушень причинно-наслідкового зв'язку.

Так як паралельні операції можуть бути виконані в різному порядку і операції редагування зазвичай не комутативні, копії документа на різних сайтах можуть відрізнятися. Саме для вирішення цього питання було запропоновано модель CCI.

3.1.2 CCI модель узгодженості

CCI модель була запропонована в якості загального фреймворка для управління узгодженістю в системах спільного редагування [16]. У середині CCI моделі, згруповані разом три властивості узгодженості.

Перша властивість – така ж сама, що і в властивість збіжності в CC моделі (рис. 3.2).

Друга властивість - збереження причинності: те ж саме, що і властивість старшинства по причинно-наслідковому порядку в CC моделі (рис. 3.3).

Третя властивість – властивість, якою CCI модель розширює CC модель, збереження наміру, тобто намір операції (вставки чи видалення) повинен зберігатися на всіх копіях, в незалежності від накладення виконання декількох операцій одночасно. Набір визначається як ефект від виконання операції на локальній копії, де операція була створена.

Третю властивість схематично зображено на рис. 3.4.

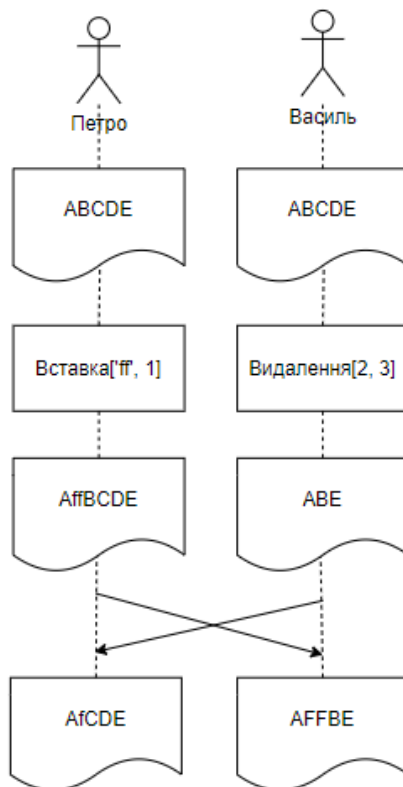


Рисунок 3.4 – Збереження наміру в ОП

Обидва клієнта починають з однакових копій, після цього кожен з них робить по одній зміні. Для Петра намір його операції - вставити "ff" на перший індекс, а для Василя - видалити символи між індексами 2 та 3. Після синхронізації у Петра намір Василя і у Василя намір Петра порушені. Копії розійшлися, але це не є вимогою для даної властивості [17].

3.1.3 Модель ОП

Систему ОП можна поділити на три частини, які схематично зображено на рис. 3.5.

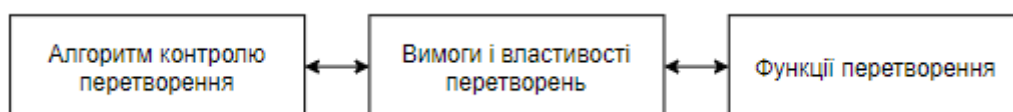


Рисунок 3.5 – Дизайн ОП

Опис наведених вище частин:

- алгоритми контролю перетворення визначають, коли цільова операція готова до перетворення, щодо яких операцій потрібно проводити перетворення, і в якому порядку їх виконувати відносно одна одної;
- функції перетворення виконують перетворення на цільовій операції з урахуванням впливу контрольних операцій;
- вимоги і властивості перетворень забезпечують зв'язок між цими компонентами і виконують перевірки на коректність.

3.1.4 Функції перетворення

Функції перетворення в ОП можна розділити на два типи:

- включення/пряме перетворення: перетворення операції *O1* перед операцією *O2* таким чином, що враховується ефект від *O2* (наприклад, дві операції вставки);
- виключення/зворотне перетворення: перетворення операції *O1* перед операцією *O2* таким чином, що ефект від *O2* виключається (наприклад, операція вставки після видалення).

Функції перетворення називаються по-різному в різних ОП-системах, а деякі складені функції перетворення можуть навіть поєднувати в собі декілька функцій [18].

3.1.5 Критика ОП

В адресу ОП можна виділити таку критику:

- реалізація ОП є дуже складним завданням з точки зору програмування. Цитуючи Джозефа Джентела, розробника бібліотеки ShareJS і колишнього інженера Google Wave: “На жаль, написати ОП дуже складно. Існує мільйон алгоритмів зі своїми плюсами і мінусами, але більшість з них - тільки

академічні дослідження. Алгоритми насправді дуже складні і вимагають багато часу для коректної їх реалізації. [...] Ми витратили 2 роки на написання Wave, і якби довелося сьогодні написати його заново - нам треба було б стільки ж часу” [19];

- необхідно зберігати абсолютно кожну зміну документу, що при збільшенні кількості одночасних користувачів зростає в геометричній прогресії.

3.1.6 Вибір ShareDB в якості бази даних на основі ОП

Так як створення власного варіанту, що реалізує ОП є дуже складним завданням, було вирішено використати готове рішення у вигляді npm пакету ShareDB.

ShareDB - це серверна база даних у реальному часі, що заснована на ОП документів JSON [20].

Особливості:

- синхронізація будь-якого документа JSON у реальному часі;
- паралельна багатокористувацька співпраця;
- API синхронного редагування з асинхронною послідовністю;
- підписки на запити в режимі реального часу;
- проста інтеграція з будь-якою базою даних - MongoDB, PostgreSQL;
- офлайн-синхронізація змін при повторному підключенні.

3.2 Архітектура розробленого вебзастосунку

Розроблений вебзастосунок поділяється на дві основні частини (модулі):

- серверний модуль;
- клієнтський модуль.

Схема структури розробленого вебзастосунку наведена на рис. 3.6.

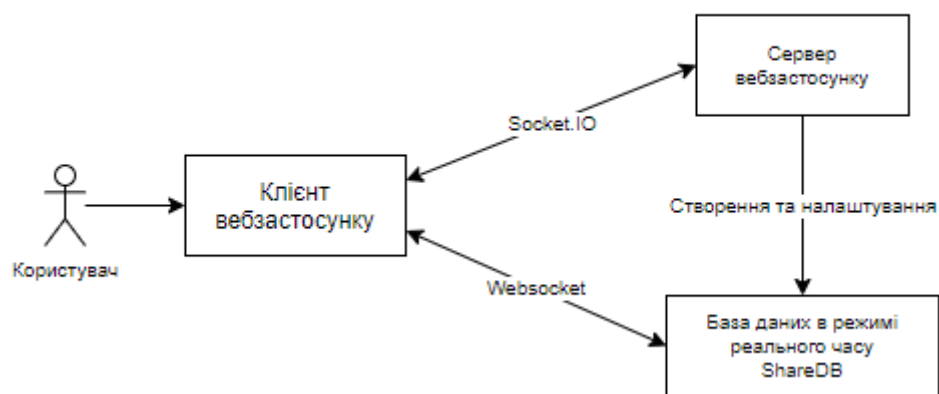


Рисунок 3.6 – Схема структури вебзастосунку

Файлова структура розробленого вебзастосунку наведена на рис. 3.7

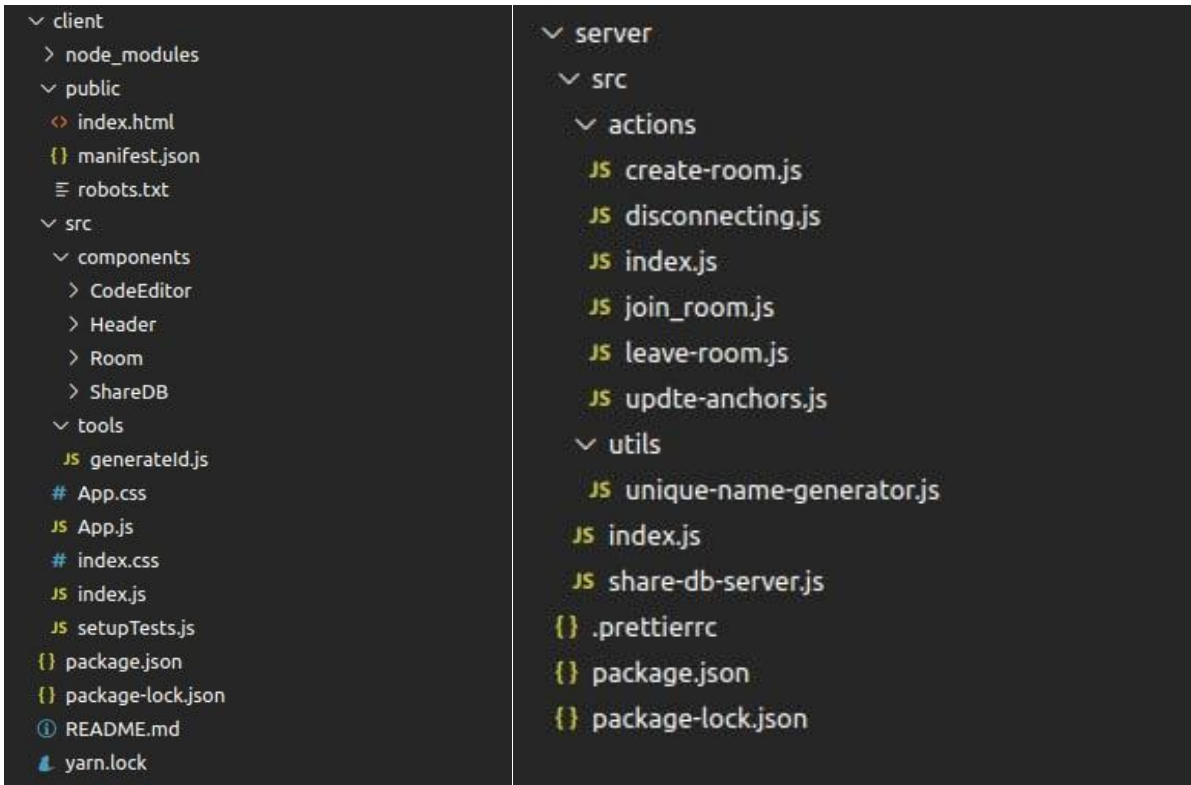


Рисунок 3.7 – Файлова структура розробленого вебзастосунку

Назва файлів та їх опис для серверного та клієнтського модулю вебзастосунку наведено в таблиці 3.1 та 3.2 відповідно.

Таблиця 3.1 – Файли серверної модулю вебзастосунку

Назва	Опис
index.js	Основний файл серверної частини вебзастосунку, в ньомі відбувається ініціалізація всіх необхідних інструментів для роботи застосунку (socket.io, express, sharedb).
share-db-server.js	Опис класу ShareDBServer, що далі використовується в index.js
utils/unique-name-generator.js	Допоміжна функція для створення унікального ідентифікатора користувача
actions/create-room.js	Функція для обробки події “create_room”
actions/disconnecting.js	Функція для обробки події “disconnecting”
actions/join_room.js	Функція для обробки події “join_room”
actions/leave_room.js	Функція для обробки події “leave_room”
actions/update-anchors.js	Функція для обробки події “update-anchors”
actions/index.js	Файл що включає всі обробники подій та експортує їх з одного місця
.prettierrc	Налаштування допоміжного розширення Prettier
package.json	Містить в собі інформацію про застосунок: назва, версія, залежно тощо
package-lock.json	Контроль версій залежностей застосунку

Таблиця 3.2 – Файли клієнтської модулю вебзастосунку

Назва	Опис
index.js	Файл що ініціалізує клієнтський застосунок в середовищі React
App.js	Компонент застосунку, використовується для ініціалізації в index.js.

Продовження таблиці 3.2

Назва	Опис
tools/generateid.js	Допоміжна функція для створення унікального ідентифікатора
components/CodeEditor	Компонент редактора коду та відображення результату компіляції коду
components/Header	Компонент верхньої панелі
components/Room	Компонент кімнати, що контролює спілкування між декількома користувачами
components/ShareDB	Компонент для налаштування ShareDB
public/index.html	Файл розмітки в якому відображається застосунок
public/manifest.json	Містить базові метадані про застосунок
public/robots.txt	Файл для пошукових роботів
App.css	Стили для зовнішнього вигляду компонентів застосунку
index.css	Загальні стилі для всього застосунку

3.2.1 Серверний модуль

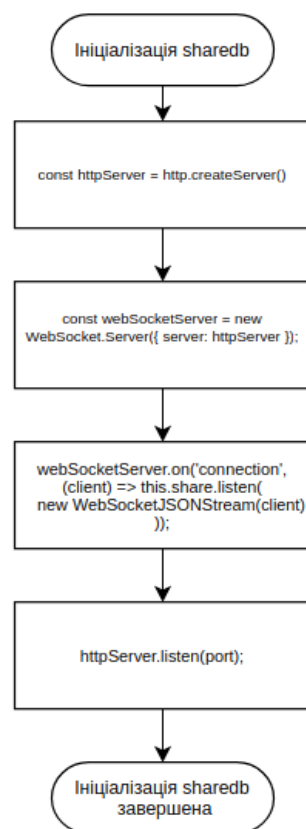
Серверний модуль в свою чергу поділяється на два незалежних між собою підмодуля.

Перший підмодуль серверного модулю реалізованого вебзастосунку це те місце, де саме реалізується технологія ОП та відбувається створення бази даних в реальному часі - sharedb, що налагоджує процес узгодження та трансформації операцій переданих від клієнта. Для того щоб забезпечити максимально швидкий відклик роботи, обмін операціями здійснюється за допомогою WebSocket, протоколу, що призначений для обміну інформацією між браузером і сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет.

Робота цього підмодуля реалізується наступною послідовністю дій:

- створення http серверу;
- створення websocket серверу на основі створеного http серверу та ініціалізація сховища кімнат, websocket сервер тут - це не що інше, як програма, яка прослуховує будь-який порт сервера TCP, який слідує певному протоколу, в нашому випадку http;
- створення екземпляру sharedb та налаштуванням прослуховування на створений раніше websocket сервер;
- починаємо прослуховувати, приймати та передавати всі повідомлення до sharedb через порт з номером 8000.

На даний момент серверна частина налаштована на прослуховування та передачу повідомлень, що приходять з клієнту до створеного sharedb. Далі до sharedb буде підключено клієнтську частину, що дозволить реалізувати обмін операціями. Алгоритм налаштування sharedb схематично зображена на рис. 3.8.



Ринок 3.8 – Блок-схема роботи алгоритму налаштування sharedb

Наступний підмодуль серверної частини реалізує можливість створення сесій між користувачами, іншими словами - кімнат. В цих кімнатах відбувається обмін повідомленнями між користувачами, що не залежать від змін, які вносяться в бд. Для реалізації цього обміну використовується SocketIO - бібліотека JS, заснована (написана поверх) на вебсокетах, що надає зручний інтерфейс для роботи як на сервері так і на клієнті.

Робота цього підмодуля реалізується наступною послідовністю дій:

- ініціалізація серверного додатку (express серверу) за допомогою фреймворку Express.js;
- створення http серверу на основі створеного express серверу;
- створення socket io серверу на основі створеного http серверу;
- налаштування обробника повідомлень до express серверу, в якому йде процес створення нового документу по кімнаті для sharedb сервісу, якщо він ще не був створений. Цей документ є тим самим місцем, що зберігає актуальний зміст тексту, що редагується учасниками кімнати;
- налаштування socket io серверу на підключення до нього клієнту.

На даний момент серверна частина налаштувала express сервер, та на його основі створила http сервер, який в свою чергу буде прослуховуватись створеним далі socket io сервером.

Описана послідовність схематично зображена на рис. 3.9.

Далі додаються обробники подій до створеного socket io серверу. Цей алгоритм реалізується наступними кроками:

- на подію підключення (connection) клієнту до серверу socket io створюється унікальний ідентифікатор клієнта що буде використовуватись в усіх обробниках подій;
- створений ідентифікатор передається підключеному клієнту через подію “i_am_connected”;

- відбувається підписка користувача на подію “create_room”, в якій створюється нова кімната та поміщається клієнт з ініціалізованими координатами курсору;
- відбувається підписка на подію “join_room”, в якій обробник приймає ідентифікатор кімнати та додає користувача до кімнати з ініціалізованими координатами курсору. Далі розсилає всім підписникам події “user_joined_room” юзера, що підключився до кімнати.
- відбувається підписка на подію “leave_room”, в якій обробник приймає ідентифікатор кімнати та видаляє користувача з кімнати. Далі розсилає всім підписникам події “user_left_room” юзера, що вийшов з кімнати.
- відбувається підписка на подію “update_anchors”, в якій обробник приймає ідентифікатор кімнати та нові координати. Далі розсилає всім підписникам події “user_updated_anchors” нові координати курсору користувача.
- відбувається підписка на подію socket.io “disconnecting”, що повідомляє всім групам, в яких знаходиться поточний користувач про його відключення.

Наведений вище алгоритм схематично представлено на рис. 3.10.

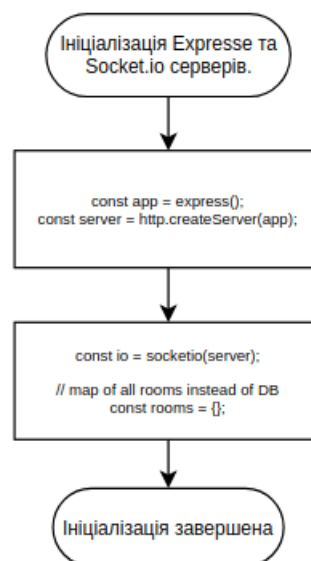


Рисунок 3.9 – Блок-схема алгоритму налаштування express та socket io серверів

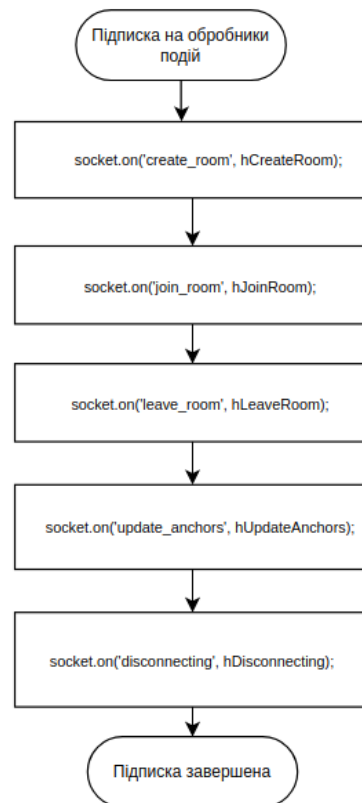


Рисунок 3.10 – Блок-схема алгоритму налаштування обробників подій для створеного socket іо серверу

З налаштуваннями серверної частини завершено, все необхідне для роботи зроблено та готове до використання у зв'язці з клієнтським модулем.

3.2.2 Клієнтський модуль

Клієнт - це саме та частина вебзастосунку з якою неперимінно працює користувач. Написана з використанням бібліотеки для побудови користувацьких інтерфейсів React та бібліотеки редактору тексту CodeMirror.

Клієнт надає користувачу весь необхідний для роботи інструментарій в якості редактору коду, панелі швидкого доступу, на якій розміщена кнопка для запрошення, та вікна відображення результату виконання скрипту з редактора.

Клієнт спілкується з сервером за допомогою socket іо та sharedb серверів. Перший потрібен для створення кімнати та обміну інформацією про

підключення та відключення інших користувачів, а також обмін інформації про місцезнаходження їх курсорів. Другий сервер виконує обмін актуальним станом спільного документу, що містить в собі код з редактору та операціями з внесеними змінами до документу. Це все дозволяє реалізувати спільне редагування коду в реальному часі.

Основний алгоритм роботи клієнтської частини вебзастосунку доцільно розділити на три частини:

- ініціалізація клієнтського екземпляру `socket io` та налаштування обробників подій;
- ініціалізація клієнтського екземпляру `sharedb` та налаштування генерації операцій для відправлення у `sharedb`;
- ініціалізація екземпляру редактора `CodeMirror` та налаштування його обробників подій.

Перший алгоритм реалізується наступними кроками:

- ініціалізується клієнтський `socket io` екземпляр через підключення до порту, на якому розміщений серверний `socket io`;
- додається обробник на подію `“i_am_connected”`, що інформує вебзастосунок про успішне підключення до сокету та передає сформований ідентифікатор поточного користувача;
- додається обробник на подію `“user_joined_room”`, що інформує вебзастосунок про те, що до поточної кімнати був успішно підключений новий користувач та передає інформацію про ідентифікатор та положення курсору;
- додається обробник на подію `“user_left_room”`, що інформує вебзастосунок про користувача, що покинув поточну кімнату. Далі вся локальна інформація про цього користувача видаляється;
- додається обробник на подію `“user_updated_anchors”`, що інформує вебзастосунок про те, що певний користувач змінив положення свого курсору в тексті та надсилає актуальні дані.

Алгоритм наведено у вигляді блок-схеми на рис. 3.11.

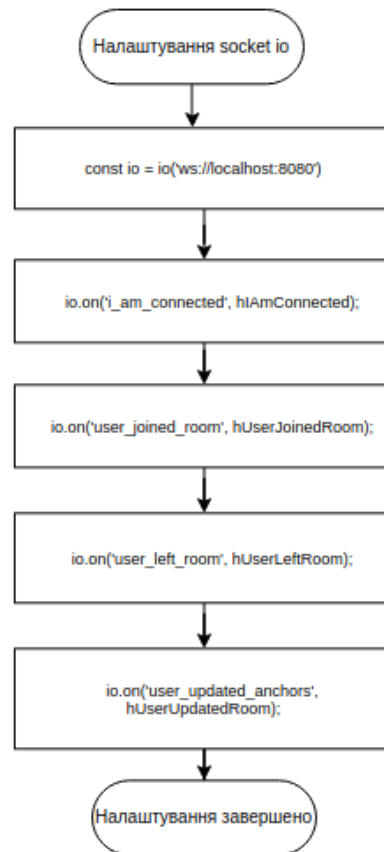


Рисунок 3.11 – Блок-схема алгоритму ініціалізації клієнтського екземпляру socket io та налаштуванню обробників подій

Алгоритм ініціалізації клієнтського екземпляру sharedb та налаштування генерації операцій для відправлення у sharedb реалізується наступними кроками:

- ініціалізація websocket-у по порту, на якому розміщено websocket з sharedb на стороні серверу;
- підключення до sharedb документу по ідентифікатору кімнати;
- підписка на зміни до документу sharedb. Отримані зміни приймаються до поточного стану тексту в редакторі;
- підписка на операції що приймаються до поточного стану іншими користувачами. Отримана операції детермінується між операцією по додаванні чи видаленню тексту.

Наведений алгоритм приведено у вигляді блок-схеми на рис. 3.12.

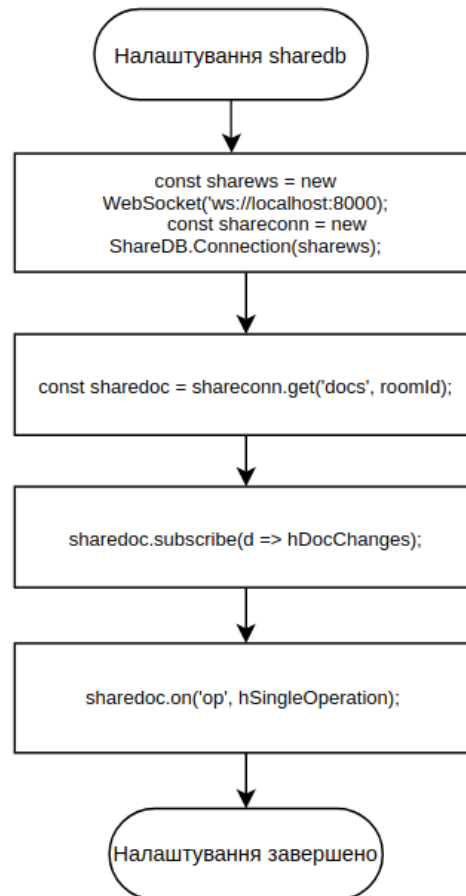


Рисунок 3.12 – Алгоритм ініціалізації клієнтського екземпляру sharedb та налаштування генерації операцій для відправлення у sharedb

Алгоритм ініціалізація екземпляру редактора CodeMirror та налаштування необхідних обробників подій що генерує редактор реалізується наступними шляхами:

- ініціалізація екземпляру редактора CodeMirror.
- підписка на подію “change” в обробнику якої зміна детермінується як додавання чи видалення тексту, створюється операція та відправляється до sharedb документу.
- підписка на подію “cursorActivity” в обробнику якої початкова позиція виділення, кінцева позиція виділення та поточна позиція курсору користувача віправляється всім іншим підключеним до кімнати користувачам.

Наведений алгоритм приведено у вигляді блок-схеми на рис. 3.13.

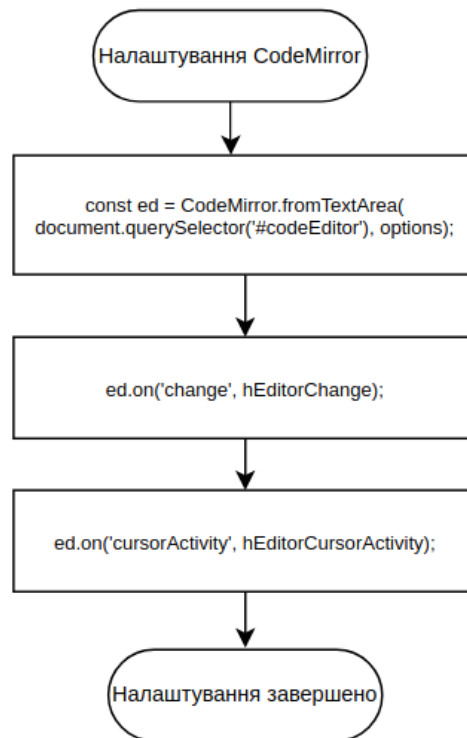


Рисунок 3.13 – Алгоритм ініціалізації екземпляру редактора CodeMirror та налаштування необхідних обробників подій що генерує редактор

3.3 Розроблений графічний інтерфейс

При розробці графічного інтерфейсу вебзастосунку використовувалася бібліотека React. Це дозволило швидко та структуровано реалізувати графічний інтерфейс у вигляді набору логічних і взаємопов'язаних компонентів.

3.3.1 Сторінки

Головною сторінкою вебзастосунку є сторінка з поточною кімнатою, в якій відбувається основний процес - редагування коду. Вигляд сторінки продемонстровано на рис. 3.14. Спроба навігації на неіснуючу URL адресу буде перенаправлятися на головну сторінку.

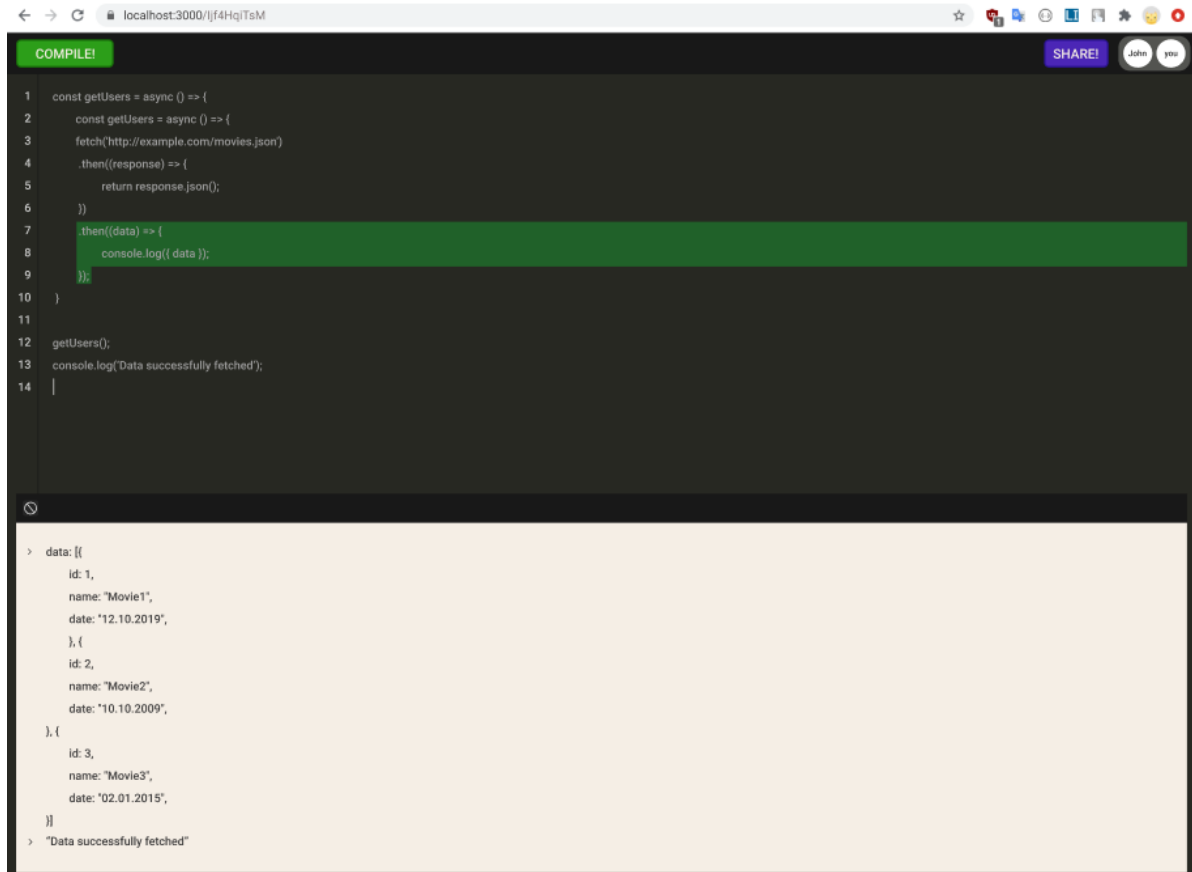


Рисунок 3.14 – Головна сторінка вебзастосунку

Сторінка візуально поділена на три головні частини - редактор, панель відображення результату виконання коду та верхню панель.

3.3.2 Компонент редактора

Редактор є найважливішою частиною вебзастосунку. Цей компонент має назву `CodeEditor` та являє собою обгортку над редактором `CodeMirror` в якому проходить розширення можливостей, кастомізація графічних і логічних частин `CodeMirror` та організація обробки всіх необхідних операцій. Компонент зображено на рис. 3.15.

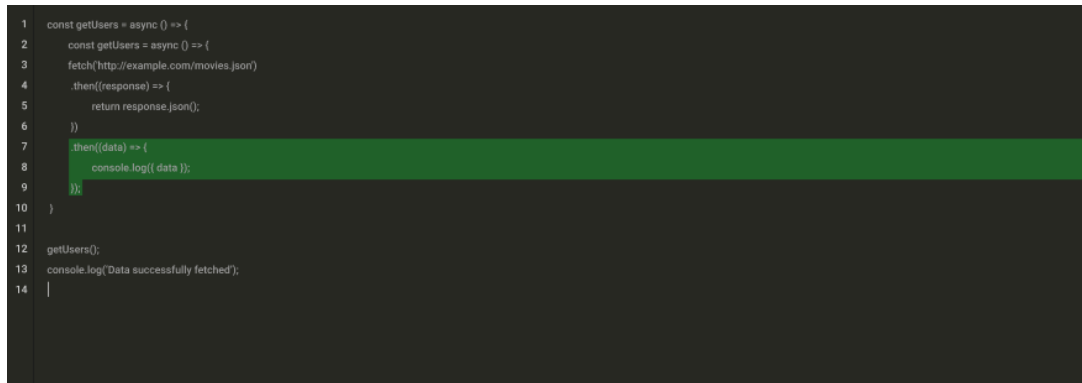


Рисунок 3.15 – Компонент редактора коду

Його візуальна роль надає користувачу:

- відображення коду;
- відображення каретки користувача;
- відображення виділень інших користувачів в кімнаті;
- відображення підказок по автодоповненню.

За кожним користувачем кімнати закріплюється особистий індивідуальний колір, який змінює вигляд виділення а в майбутніх доповненнях буде також коліров повідомлень від користувача.

Підказки по доповненню є дуже важливою частиною функціонала так як допомагає користувачеві економити велику кількість часу на те щоб не писати все по пам'яті.

Компонент підказки реалізован відповідним модулем, що надається бібліотекою CodeMirror. Компонент зображено на рис. 3.16.

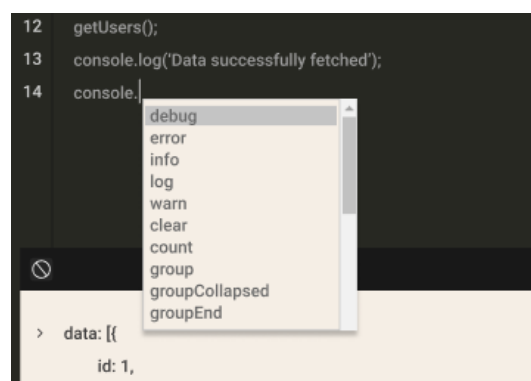


Рисунок 3.16 – Компонент підказок

3.3.3 Компонент відображення результату

Компонент відображення результатів виконання цього коду, реалізований у вебзастосунку під назвою `ResultOutput` та знаходиться в компоненті `CodeEditor`. Компонент зображено на рис 3.17. Для додаткової зручності додана можливість для очистки всього поточного виводу. Вигляд кнопки яка викликає відповідну функцію на рис. 3.18.

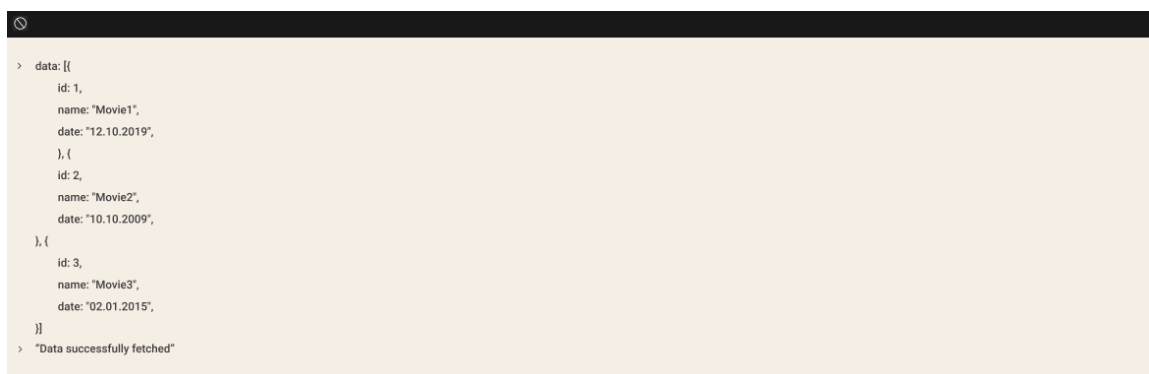


Рисунок 3.17 – Компонент відображення результату

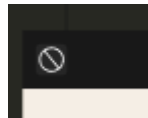


Рисунок 3.18 – Кнопка для очистки поточного виводу

3.3.4 Компонент верхньої панелі

Компонент верхньої панелі `Header` служить для швидкого та зручного доступу до функції компіляції коду та панелі з користувачами в кімнаті. `RoomHeader` зображено на рис. 3.19.

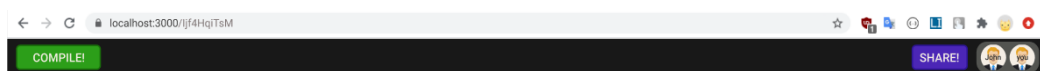


Рисунок 3.19 – Компонент верхньої панелі

На ньому розміщено кнопку для компіляції коду, що сигналізує вебзастосунку взяти текст з редактору та відобразити результат його виконання в компоненті ResultOutput.

В правій частині розміщено блок поточних користувачів в кімнаті та кнопка для запрошення (рис. 3.20).

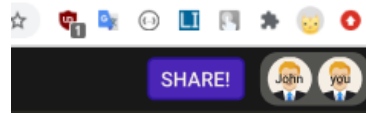


Рисунок 3.20 – Блок користувачів та кнопка для їх запрошення

3.3.5 Компонент повідомлень

Для гарного користувацького досвіду було реалізовано компонент повідомлень Notification. Каркас для побудови компоненту було взято в бібліотеці react-toastify та доповнено власними стилями відповідно до дизайнів. Цей компонент має два варіанта вигляду - “success” та “failure”, де success - відображає повідомлення про успішно виконані дії, наприклад результат кліку по кнопці Share. На рис. 3.21. зображено “success” повідомлення та його місцезнаходження на інтерфейсі. “Failure” відображає повідомлення у разі якоїсь помилки, наприклад при невдалій спробі скомпілювати код, зображено на рис. 3.22.

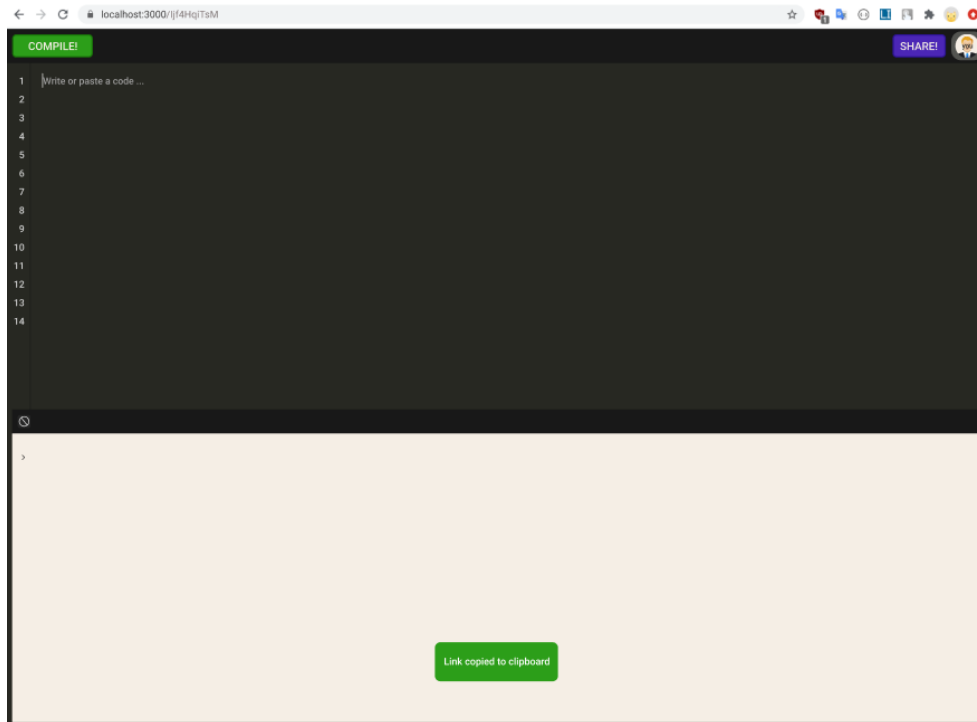


Рисунок 3.21 – Повідомлення про успішно виконану дію

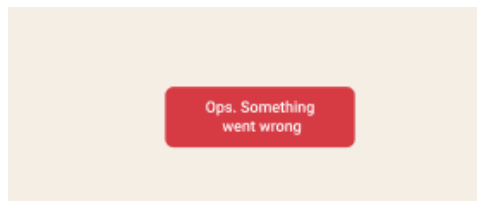


Рисунок 3.22 – Повідомлення про помилку

3.4 Висновок

Для створення вебзастосунку було проаналізовано технологію ОП, в контексті якої було розглянуто дві популярних моделі узгодження, обрана модель ССІ, що забезпечує виконання трьох властивостей: збіжність, збереження причинності, збереження наміру – необхідних для збереження узгодженості результуючого стану в системах спільного редагування. Проаналізовано модель ОП та наведено визначення функції перетворення, що є частиною моделі. На основі проаналізованого матеріалу було обрано використовувати ShareDB в якості бази даних, що реалізує технологію ОП.

Було створено клієнтську та серверну частину вебзастосунку, наведено алгоритми їх налаштування і роботи для забезпечення зв'язку між декількома користувачами та реалізації можливості спільного редагування коду в реальному часі.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Призначенням розробленого вебзастосунку є надання можливості спільного написання та редагування програмного коду на мові програмування JS в реальному часі, створювати швидкі прототипи та алгоритми при командному вирішенню задач, тестуванню та демонстрації логічних блоків застосунку, допомагати знаходити рішення проблем віддаленим товаришам а також організовувати інтерактивне навчання.

Характеристики комп'ютера для стабільної та швидкої роботи програми повинні задовольняти вимогам зазначеним в таблиці 4.1.

Таблиця 4.1 – Вимоги до характеристик комп'ютера

Компонент	Конфігурація
Операційна система	Microsoft Windows XP і вище, Linux, MacOS
Процесор	800 MHz
RAM	556 Мб і більше DDR3 RAM
Браузер	Chrome v57+, Opera v44+, IE10, 11, Firefox v52+, Edge v16+

4.2 Характеристики вебзастосунку

Вебзастосунок було розроблено в редакторі вихідного коду Microsoft Visual Studio Code на мові програмування JS, з використанням бібліотеки

користувацьких інтерфейсів React, редактора коду CodeMirror, бази даних в реальному часі що реалізує технологію ОП - sharedb, фреймворку express, технології вебсокетів та бібліотеки що її реалізує - socketio.

Вебзастосунок в інсталяції не має потреби. Для того щоб запустити, потрібно розмістити білд програми на будь-якому зручному хостингу. Також можливий варіант з запуском в локальній мережі чи з використанням транслуючих локальний порт до мережі інтернет, наприклад ngrok. Далі застосунок можна запустити з використанням браузеру що відповідає зазначеним вимогам в табл. 4.1 по адресі розташування білда.

Програма виконує наступні функції:

- створення кімнати з редактором коду для забезпечення приватності;
- надання можливості писати та редагувати код;
- спільне редагування в реальному часі;
- компіляція та відображення результату виконання коду;
- допоміжні (автодоповнення, підсвітка виділень коду іншими користувачами).

4.3 Вхідні і вихідні дані

Вхідними даними вебзастосунку виступає операція по зміні чи видаленні якоїсь частини можливого для спільного редагування тексту, який представляє із себе код на мові програмування JS; інформація про поточних користувачів кімнати, а саме їх унікальні ідентифікатори та актуальні координати курсору та виділення тексту в редакторі.

Вихідними даними вебзастосунку є організована та приведена до відповідного виду інформація, що враховує всі внесені до неї одночасні зміни та є ідентичною для всіх користувачів в кімнаті; візуальне відображення

інформації про місцезнаходження виділення коду всіх користувачів кімнати; результат та візуальне відображення виконання коду редактора.

4.4 Виконання програми

Для того щоб розпочати використовувати вебзастосунок, необхідно розмістити білд програми на будь-якому зручному хостингу. Також можливий варіант з запуском в локальній мережі чи з використанням трансляторів локального порту до мережі інтернет, наприклад ngrok. Далі застосунок потрібно запустити з використанням браузеру що відповідає зазначеним вимогам в табл. 4.1 по адресі хостингу білда.

Наведемо приклад однієї з послідовностей - запуск в локальній мережі. Запуск буде відбуватися з використанням операційної системи Linux. Перш за все, якщо замість білду присутній тільки вихідний код, треба збілдити вебзастосунок. Виконується за наступними кроками:

- перейти в папку з вихідним кодом вебзастосунку, наприклад за допомогою терміналу (рис. 4.1);
- виконати команду “`npm run build`” для того щоб збілдити вебзастосунок (рис. 4.2);
- розгорнути білд на статичному сервері, наприклад за допомогою npm пакету “`serve`” виконавши команду “`serve -s -p 3000 build`” (рис. 4.3);
- відкрити вебзастосунок в браузері, наприклад Google Chrome, по адресі “`localhost:3000/`” (рис. 4.4).

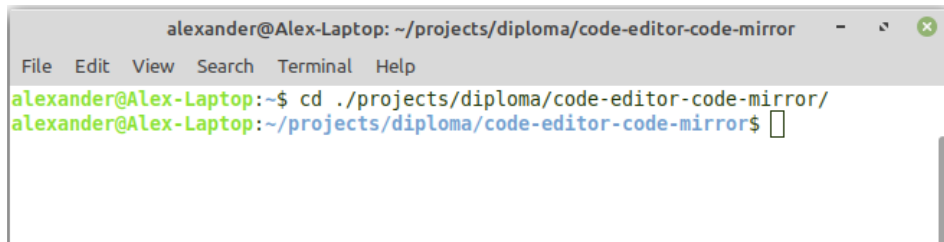


Рисунок 4.1 – Папка з вихідним кодом вебзастосунку

```
alexander@Alex-Laptop:~/projects/diploma/code-editor-code-mirror$ npm run build

> code-editor-code-mirror@0.1.0 build /home/alexander/projects/diploma/code-editor-code-mirror
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

 106.35 KB    build/static/js/2.710a8dc8.chunk.js
  2.03 KB     build/static/css/2.79738bb3.chunk.css
  1.03 KB (-1 B) build/static/js/main.84311daa.chunk.js
    784 B     build/static/js/runtime-main.3e40b256.js
    380 B     build/static/css/main.2e3ce07c.chunk.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  serve -s build

Find out more about deployment here:

  bit.ly/CRA-deploy

alexander@Alex-Laptop:~/projects/diploma/code-editor-code-mirror$
```

Рисунок 4.2 – Виконання скрипту “build”

```
alexander@Alex-Laptop:~/projects/diploma/code-editor-code-mirror$ serve -s -p 3000 build
UPDATE AVAILABLE The latest version of `serve` is 11.3.2
You can specify an explicit port and host to listen on which to listen
? port on hostname "localhost":
  Serving!
  - Local:      http://localhost:3000
  - On Your Network: http://192.168.0.109:3000
  You can now use Ctrl+C to stop the server
  Copied local address to clipboard!
```

Рисунок 4.3 – Виконання команди “serve -s -p 3000 build”

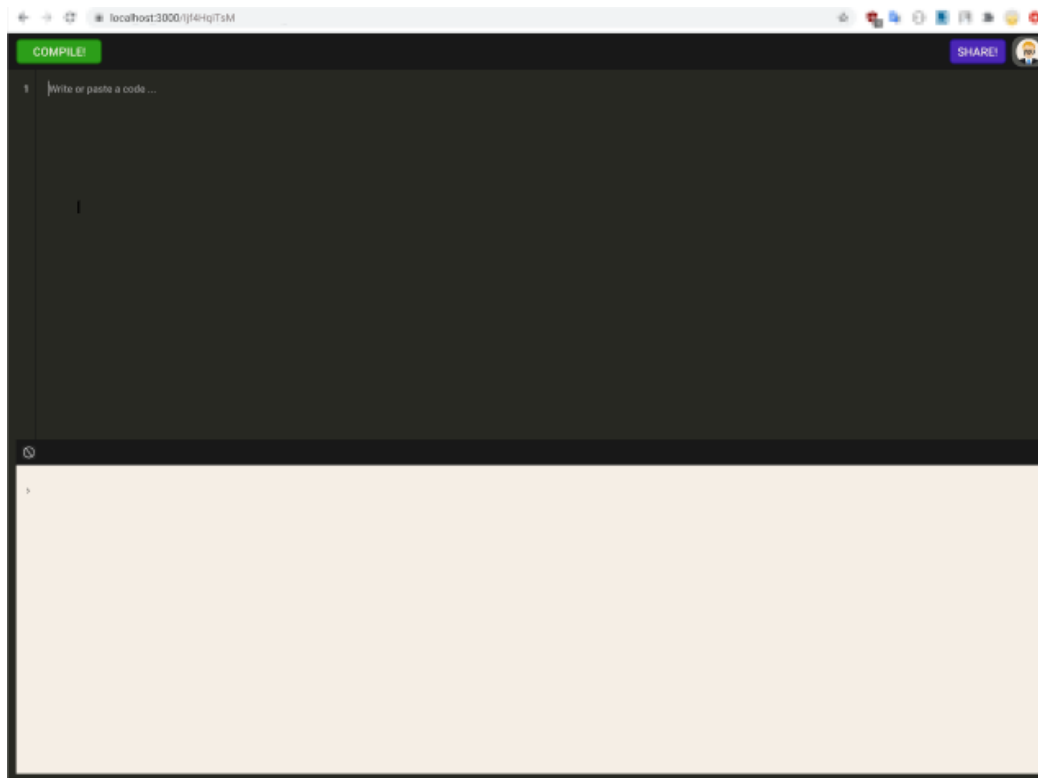


Рисунок 4.4 – Відкритий вебзастосунок по адресі localhost:/3000

Вебзастосунок готовий до безпосередньої роботи. Продемонструємо можливий порядок роботи з вебзастосунком.

Робота з редактором виконується за допомогою клавіатури та маніпулятора миші. Клік по області в якій знаходиться текст “Write or paste a code...” поміщає курсор на перший рядок редактору.

Напишемо невеликий алгоритм, наприклад алгоритм бульбашкової сортировки (рис. 4.5).

Добавимо в редактор створення масиву що необхідно сортувати та виведемо результат до консолі натиснувши на кнопку “Compile” в верхній лівій частині вікна (кнопка зображена на рис. 4.6). Отриманий результат відобразиться в блоці консолі (рис. 4.7).

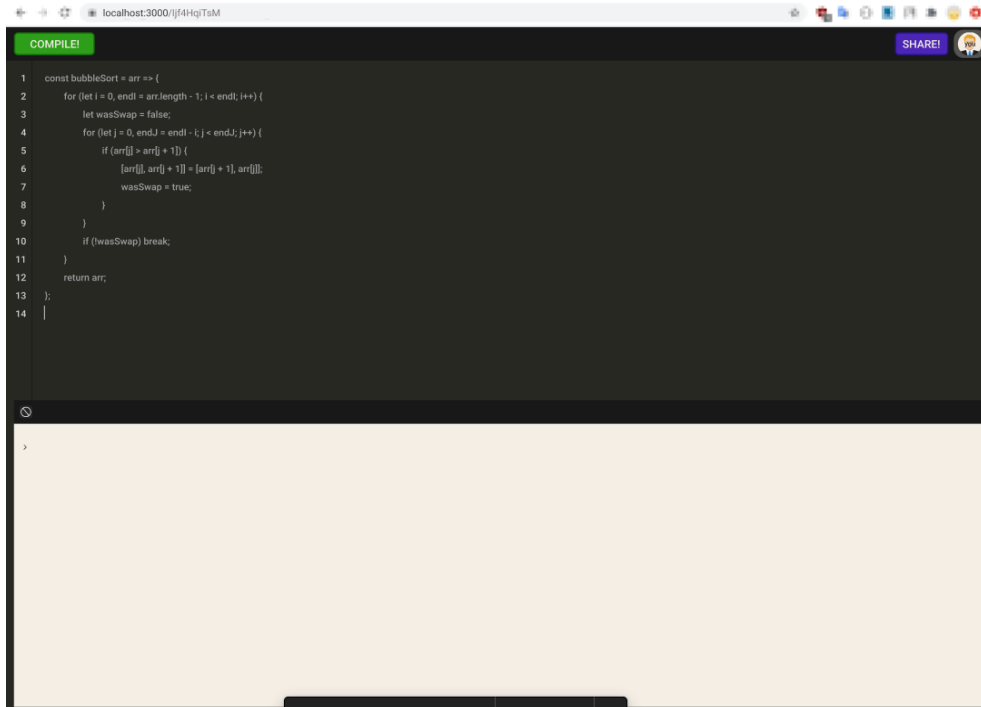


Рисунок 4.5 – Алгоритм бульбашкової сортировки

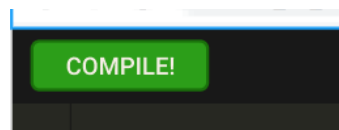


Рисунок 4.6 – Кнопка для компіляції коду

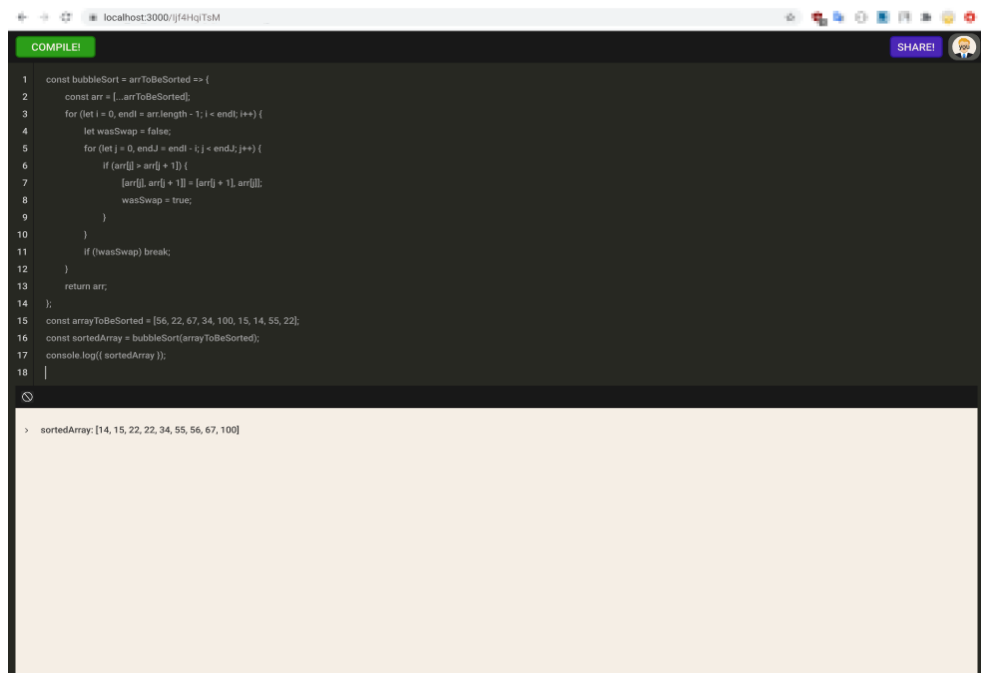


Рисунок 4.7 – Результат виконання алгоритму сортування

Наступним кроком буде використання вебзастосунку для спільного редагування коду з іншим користувачем.

Для цього потрібно клікнути по кнопці “Share” що знаходиться у верхній правій частині застосунку (рис. 4.8).

Після кліку по кнопці до буферу обміну було скопійовано URL адресу, що необхідно передати іншому користувачу для його здійснення входу. Успішне виконання копіювання супроводжується повідомленням з боку вебзастосунку (рис. 4.9). URL адресу також можливо дістати скопіювавши поточну URL адресу кімнати.

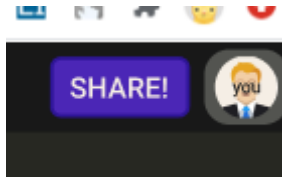


Рисунок 4.8 – Кнопка “Share”

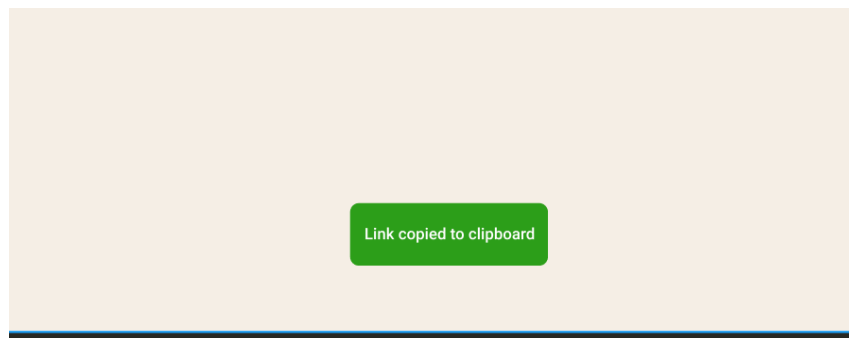


Рисунок 4.9 – Повідомлення про успішну дію

Для того щоб зайти до кімнати іншим користувачем необхідно помістити отриману URL адресу до панелі пошуку будь-якого браузеру або просто клікнувши на посилання. (рис. 4.10).

Після успішного входу новий користувач буде додано до відображення в панелі користувачів в кімнаті (рис. 4.11).

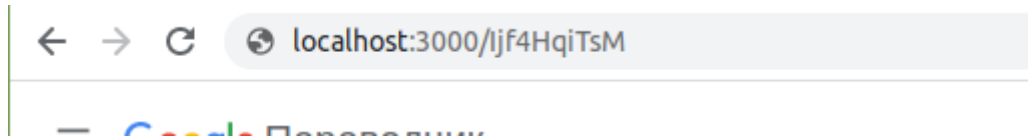


Рисунок 4.10 – URL адреса

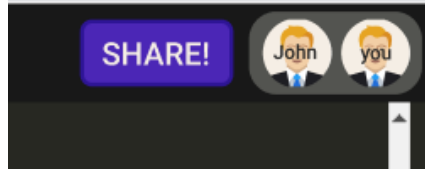


Рисунок 4.11 – Панель користувачів в кімнаті

Припустимо, що новий користувач вирішив доповнити поточний код ще одним алгоритмом сортування, наприклад вставками. З нашого боку ми можемо успішно спостерігати процес редагування (рис. 4.12).

Для комфорту також реалізовано обмін інформацією про координати виділеного тексту у кожного із користувачів кімнати (рис. 4.13).

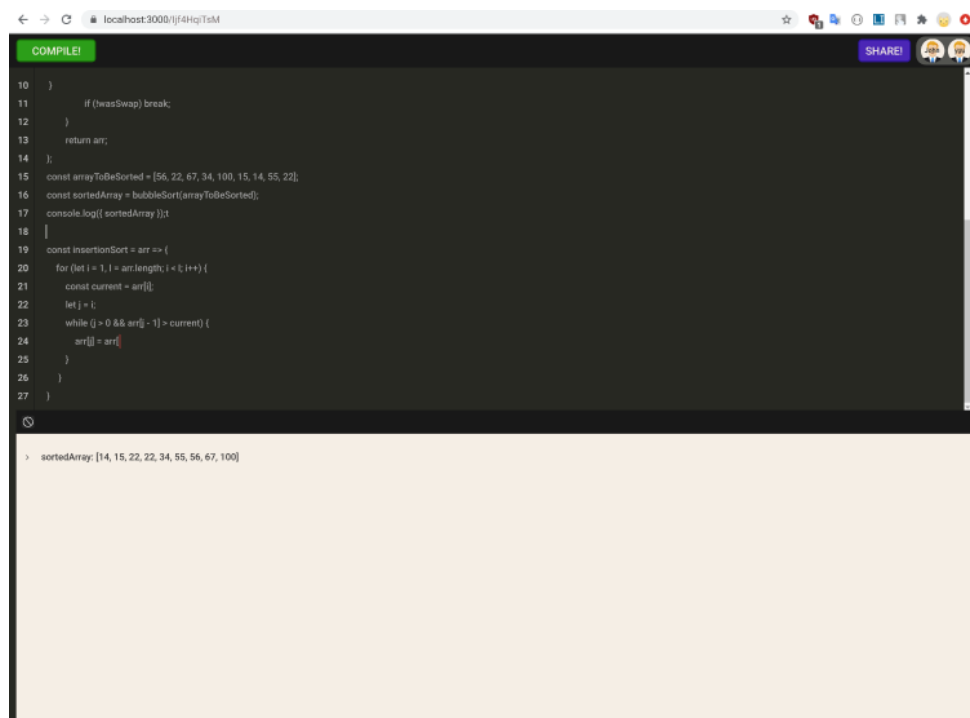


Рисунок 4.12 – Інший користувач редагує код

```

10   return arr;
11 }
12 const arrayToBeSorted = [56, 22, 67, 34, 100, 15, 14, 55, 22];
13 const sortedArray = bubbleSort(arrayToBeSorted);
14 console.log(sortedArray);
15
16 const InsertionSort = arr => {
17   for (let i = 1; i < arr.length; i++) {
18     const current = arr[i];
19     let j = i;
20     while (j > 0 && arr[j - 1] > current) {
21       arr[j] = arr[j - 1];
22       j--;
23     }
24     arr[j] = current;
25   }
26   return arr;
27 }
28
29 console.log(InsertionSort(arrayToBeSorted));
30

```

sortedArray: [14, 15, 22, 23, 34, 55, 56, 67, 100]

Рисунок 4.13 – Виділений текст іншим користувачем

Для зручності користувачів вебзастосунку додана можливість автодоповнення тексту. Для її виклику необхідно натиснути поєднання клавіш “Ctrl” + “Space”. Для вибору необхідного доповнення потрібно скористатися стрілками та натиснути “Enter” (рис. 4.14).



Рисунок 4.14 - Вікно підказки для автодоповнення

З ознайомленням по виконанню програми завершено, планується регулярне поповнення функціоналу вебзастосунку найпопулярнішими вимогами користувачів та впровадженням сучасних поглядів користувацький досвід.

5 ЕКОНОМІКО-ОРГАНІЗАЦІЙНА ЧАСТИНА

5.1 Опис продукту що розробляється

Розроблений продукт - це вебзастосунок для спільного редагування JS коду в реальному часі що дозволяє створювати швидкі прототипи та алгоритми при командному вирішенню задач і прийняттю взважених та завірених декількома людьми рішень, демонстрації та тестуванню логічних блоків застосунку; допомагати вирішувати несподівані проблеми віддаленим товаришам по роботі та організовувати інтерактивне навчання для своїх учнів.

Сучасний світі зіткнувся з несподіваною перепорою для фізичного контакту людей у вигляді необхідної ізоляції та дистанціюванню, як ніколи потрібно мати можливість зберегти свою максимальну продуктивність в роботі.

В зв'язку з чим, цей вебзастосунок являється досить актуальний продуктом для конкретного переліку користувачів. Його актуальність цілком логічно буде позитивно впливати на матеріальне становище власника продукту у зв'язку з невеликої кількістю можливих альтернативних рішень.

В таблиці 5.1 наведено опис ідеї продукту, що розробляється.

Таблиця 5.1 – Опис ідеї

Зміст ідеї	Напрямки застосування	Вигоди для споживачів (користувачів)
Вебзастосунок для спільного редагування JS коду в реальному часі	1. Використання програмістами для спільної розробки алгоритмів та прототипів для подальшого використання в будь-якій галузі	Можливість інтерактивного командного вирішення задач і прийняття зважених та завірених декількома людьми рішень. Можливість віддаленої співпраці в

Продовження таблиці 5.1

Зміст ідеї	Напрямки застосування	Вигоди для споживачів (користувачів)
		реальному часі. Спільна відповідальність за розроблений алгоритм.
	2. Навчальна практика	Інтерактивний вид навчання має підвищений рівень зацікавленості, що дозволить заволікати до предмету навчання більшу кількість учнів та зберегти їх увагу впродовж довгого часу.
	3. Віддалена допомога	Зручна, безпечна віддалена допомога товаришу по “цеху”. Інтерактивний вид допомоги дозволяє більш надійно зрозуміти та засвоїти рішення.

Надано цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Проаналізовано характеристику потенційного ринку та подано у вигляді таблиці 5.2.

Таблиця 5.2 – Попередня характеристика потенційного ринку

№	Показники стану ринку (найменування)	Характеристика
1	Головні конкуренти	3
2	Загальний обсяг продаж, грн/ум.од	-
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу	Частково технічного характеру. Продукти конкурентів мають велику кількість функціоналу але багато за що потрібно додатково платити.
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні.
6	Середня норма рентабельності в галузі (або по ринку) %	70

Враховуючи кількість головних гравців на ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент ринок для входження продукту є привабливим та навіть надає змогу реалізувати продукт на світовому ринку.

Визначена потенційна група клієнтів, їх характеристики, та сформувано орієнтовний перелік вимог до товару для кожної групи, що наведено в таблиці 5.3.

Таблиця 5.3 – Попередня характеристика потенційних клієнтів

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів (користувачів)
1	Потреба створювати алгоритми для власного продукту на мові JS з можливістю спільної співпраці та в режимі реального часу для найкращого результату та продуктивної роботи.	Програмісти	Продукт буде використовуватися в максимальному режимі спільної розробки, потрібно забезпечити найкраще можливий відклик системи.	Зручність у використанні. Ергономічний та гарний інтерфейс. Швидка робота продукту. Спроможність швидко освоїти як користуватися продуктом.
2	Потреба інтерактивного способу навчання та демонстрації можливостей мови JS для збільшення залученості учнів.	Вчителі та їх учні	Продукт буде використовуватися більш в режимі демонстрації з можливістю взаємодії, потрібно забезпечити користувачів гарним та не відволікаючим інтерфейсом.	Забезпечення неперервною підтримкою продукту.

Здійснено SWOT-аналіз середовища реалізації проєкту (табл. 5.4).

Таблиця 5.4 – SWOT-аналіз

Сильні сторони:	Слабкі сторони:
- актуальність; - націленість на досить широкий сектор користувачів; - простота у використанні; - наявність відкритого вихідного коду; - збереження приватності інформації користувача;	- недосконалість алгоритму що реалізує можливість спільної праці; - на старті невеликий набір функціоналу порівняно з конкурентами; - відкритий вихідний код як можливість
Можливості:	Загрози:
- вихід на міжнародний та світовий ринок - розширення базового функціоналу до повноцінної IDE система з можливістю обміну повідомленнями	- поява сильних конкурентів з таким же набором функціоналу по безоплатній підписці - збої в роботі алгоритму, що можуть призвести до поганого користувацького досвіду

Опис цільових груп потенційних споживачів наведено в таблиці 5.5.

Таблиця 5.5 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Програмісти	Присутня	Середний	Присутня	Легка
2	Викладачі та їх учні	Присутня	Середній	Майже відсутня	Середня
Які цільові групи обрано: 1 та 2.					

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є як програмісти, так і викладачі з їх учнями.

5.2 Планування розробки

Весь комплекс розробки можна розділити на етапи. Для кожного етапу вказується трудомісткість, кількість виконавців і тривалість робіт. У розробці беруть участь програміст в перебігу протягом двох з половиною місяців і консультант протягом пів місяця. Розробка розпочинається 3 вересня і повинна бути виконана до 15 листопада 2020 року. Етапи розробки продукту наведені в таблиці 5.6.

Таблиця 5.6 – Етапи розробки продукту

Назва роботи	Трудомісткість		Виконавці	Тривалість, днів
	люд.- дні	% до підсумку		
1. Аналіз предметної області	6	9,23	Програміст, консультант	3
2. Визначення вимог до програмного продукту	4	6,15	Програміст, консультант	2
3. Аналіз принципів побудови експертних систем	7	10,77	Програміст	7
4. Дослідження методів вирішення задач	7	10,77	Програміст	7
5. Побудова моделі системи	6	9,23	Програміст	6
6. Створення програмного коду	25	38,46	Програміст	25

Продовження таблиці 5.6.

Назва роботи	Трудомісткість		Виконавці	Тривалість, днів
7. Тестування і налагодження програмного продукту	6	9,23	Програміст, консультант	3
8. Складання програмної документації	4	6,15	Програміст, консультант	2
Разом	65	100		55

За даними таблиці 5.6 складено стрічковий графік планування розробки продукту наведений у таблиці 5.7 (сірий колір – консультант, чорний – програміст).

Таблиця 5.7 – Стрічковий графік

Дата / Найменування роботи	03.09.2020	09.09.2020	16.09.2020	23.09.2020	30.09.2020	07.10.2020	14.10.2020	21.10.2020	28.10.2020	04.11.2020	11.11.2020
1. Аналіз предметної області											
2. Визначення вимог до програмного продукту											
3. Аналіз принципів побудови експертних систем											
4. Дослідження методів вирішення задач											
5. Побудова моделі системи											
6. Створення програмного коду											
7. Тестування і налагодження програмного продукту											
8. Складання програмної документації											

5.3 Визначення витрат на розробку

Для визначення витрат на розробку програми складається калькуляція кошторисної вартості робіт, яка включає наступні статті:

- основна заробітна плата;
- додаткова заробітна плата;
- відрахування на єдиний соціальний внесок;
- матеріали і комплектуючі вироби;
- витрати на спеціальне обладнання;
- накладні витрати.

Для розрахунку заробітної плати маємо на увазі, що за час, витрачений на розробку системи кількість робочих днів дорівнює 55.

Тривалість робіт визначають за формулою (5.1):

$$T_{\text{ц}} = \frac{Q}{R}, \quad (5.1)$$

де $T_{\text{ц}}$ – кількість днів;

Q – трудомісткість;

R – кількість робітників (люд.).

5.3.1 Розрахунок основної заробітної плати

Проводиться розрахунок основної заробітної плати, який наведено у таблиці 5.8.

5.3.2 Розрахунок додаткової заробітної плати

Додаткова заробітна плата приймається рівною 10% від основної заробітної плати працівника і розраховується за формулою (5.2).

Таблиця 5.8 – Основна заробітна плата

Посада виконавця	Чисельність, чол.	Місячний оклад, грн	Кількість місяців	Сума ЗП, грн
Програміст	1	10 000	2,5	25 000
Консультант	1	8 000	0,5	4 000
Підсумок	2			29 000

$$ЗП_{\text{дод}} = ЗП_{\text{осн}} \cdot 0.1. \quad (5.2)$$

Після підстановки величини основної заробітної плати в дану формулу, отримується:

$$ЗП_{\text{дод}} = 29000 \cdot 0.1 = 2900 \text{ грн.}$$

5.3.3 Відрахування на єдиний соціальний внесок

Відрахування на єдиний соціальний внесок становлять 22% і беруться від основної та додаткової заробітної плати. Розраховуються за формулою (5.3):

$$ВІД = (ЗП_{\text{осн}} + ЗП_{\text{дод}}) \cdot 0.22, \quad (5.3)$$

де $ЗП_{\text{осн}}$ – основна заробітна плата (грн.);

$ЗП_{\text{дод}}$ – додаткова заробітна плата (грн.).

$$ВІД = (29000 + 2900) \cdot 0.22 = 7018 \text{ грн.}$$

Додаткові соціальні відрахування становлять 7018 грн.

5.3.4 Визначення витрат на матеріали

Витрати на матеріали розраховуються за формулою (5.4):

$$M = \sum_{i=1}^n (C_i N_i (1 + K_{т.з.}) - C_{io} N_{io}), \quad (5.4)$$

де M – витрати на матеріали, покупні напівфабрикати та комплектуючі (грн.);

$K_{т.з.}$ – коефіцієнт, що враховує транспортно–заготівельні витрати;

C_i – ціна i -го найменування чи комплектуючого (грн.);

N_i – потреба в i -му матеріалі, напівфабрикаті, комплектуючих;

C_{io} – ціна зворотних відходів i -го найменування матеріалу (грн.);

N_{io} – кількість зворотних відходів i -го найменування;

n – кількість найменувань матеріалів, напівфабрикатів та комплектуючих.

Витрати на матеріали для розробки наведені в табл. 5.9.

Таблиця 5.9 – Витрати на матеріали

Найменування, вид, тип, марка	Одиниця виміру	Ціна за одиницю вим., грн.	Кількість од. вим.	Сума витрат, грн.
Канцелярія	Набір	300	2	600
Картридж для принтера	шт.	650	1	650
Папір формату А4 для принтера	Листів	0,19	500	95
USB Flash накопичувач об'ємом 32Gb	шт.	250	2	500
Всього				1845

$$M = 1.05 \cdot 1\,845 = 1\,937.25 \text{ грн.}$$

З урахуванням можливих транспортних витрат (5% від вартості матеріалів), сумарна вартість матеріалів становить 1937.25 грн.

5.3.5 Витрати на ПК

У цю статтю входять витрати на придбання, транспортування, монтаж і налагодження нестандартного обладнання. Також необхідно розрахувати витрати на оплату машинного часу ЕОМ (амортизаційні витрати на обслуговування та вартість спожитої електроенергії).

Розраховуємо витрати на обслуговування ЕОМ. Амортизаційні відрахування розраховуються за формулою (5.5):

$$A = \Phi_B \frac{H_a}{100}, \quad (5.5)$$

де Φ_B – балансова вартість обчислювальної техніки (грн.);

H_a – норма амортизаційних відрахувань на повне відновлення обчислювальної техніки (%).

Підставивши у формулу (5.5) значення балансової вартості використовуваних ЕОМ Asus VivoBook 15 X540UA-DM3095R (16000 грн.), маніпулятору типу миша (420 грн.) та принтеру HP Neverstop Laser 1200n (8300 грн.), що разом коштує 24720 грн. отримаємо:

$$A = 24\,720 \cdot 0.25 = 6\,180 \text{ грн.}$$

Амортизаційні відрахування становлять 6180 грн.

Для розрахунку вартості експлуатації обладнання використаємо формулу (5.6):

$$C_{\text{э}} = N_{\text{н}} \cdot \Phi_{\text{еф}} \cdot K_{\text{зч}} \cdot K_{\text{зп}} \cdot C_{\text{е}}, \quad (5.6)$$

де $N_{\text{н}}$ – номінальна потужність ЕОМ, кВт;

$\Phi_{\text{еф}}$ – річний ефективний фонд часу роботи ЕОМ;

$K_{\text{зч}}$ – середні коефіцієнт завантаження за часом;

$K_{\text{зп}}$ – коефіцієнт завантаження за потужністю;

$C_{\text{е}}$ – ціна одного кВт · год. Електроенергії, грн./(кВт · год.).

Річний фонд часу роботи ЕОМ (персонального комп'ютера працівника) становить 1800 годин. Середні коефіцієнти завантаження за часом та за потужністю становлять 0.6 та 0.9. Ціна одного кВт · год = 1.68грн.

$$C_{\text{э1}} = 0.2 \cdot 1800 \cdot 0.9 \cdot 0.6 \cdot 1.68 = 326.6 \text{ грн.}$$

Заробітна плата обслуговуючого персоналу розраховується за формулою (5.7):

$$ЗП_{\text{обсл}} = \Phi ЗП_{\text{р}} (1 + K_{\text{від}}) \frac{t_{\text{обсл}}}{\Phi_{\text{еф.обсл}}}, \quad (5.7)$$

де $\Phi ЗП_{\text{р}}$ – річний фонд заробітної плати (основної та додаткової) обслуговуючих робіт (грн.),

$K_{\text{від}}$ – коефіцієнт відрахувань на соціальне страхування і до інших фондів (%);

$t_{\text{обсл}}$ – час протягом року, необхідний на технічне обслуговування ЕОМ, (ч/рік);

$\Phi_{\text{еф. обсл.}}$ – річний ефективний фонд часу обслуговуючого персоналу.

Місячна заробітна плата обслуговуючого персоналу становить 5100 грн., а річний фонд заробітної плати становить 61200 грн. Річний ефективний фонд робочого часу обслуговуючого ПК робітника становить 1800 годин/рік. На обслуговування одного ПК витрачається по 2 години на місяць, що на рік становить 24 години. З чого виходить, що:

$$ЗП_{\text{обсл}} = 61200 \cdot (1 + 0.22) \cdot \frac{24}{1800} = 995.52 \text{ грн.}$$

Стаття «Поточний ремонт обладнання» приймається рівною 3% від балансової вартості обладнання і складає $0.03 \cdot 24720 = 741$ грн.

Стаття «Інші витрати» приймається рівною п'яти відсоткам від суми всіх попередніх статей витрат на утримання та експлуатацію устаткування. Сума всіх попередніх статей дорівнює $6180 + 326.6 + 741 + 995.52 = 8243.2$ грн, 5% від суми становлять 412,16 грн.

Кошторис витрат на утримання та експлуатацію обладнання описано у таблиці 5.10.

Таблиця 5.10 – Кошторис витрат

Найменування статей витрат	Сума, грн.
Амортизація обладнання	6180
Експлуатація ПК (окрім витрат на поточний ремонт)	326.6
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	741
Поточний ремонт обладнання	995.52
Інші витрати	412.16
Разом	8655.28

Витрати на оплату машинного часу ЕОМ для написання й відлагодження програми вираховується за формулою (5.8):

$$C_{\text{мс}} = P_{\text{екс}} t_{\text{мч}}, \quad (5.8)$$

де $C_{\text{мс}}$ – витрати на сплату машинного часу (грн.);

$P_{\text{екс}}$ – експлуатаційні витрати на одиницю час машинного часу ЕОМ;

$t_{\text{мо}}$ – машинний час ЕОМ для написання та відлагодження програмного засобу.

$$P_{\text{екс}} = \frac{8655.28}{1800} = 4.80 \text{ грн./машино- час.}$$

ЕОМ експлуатується 55 днів в одну зміну, що становить у сумі 440 годин. Таким чином витрати на сплату машинного часу становлять:

$$C_{\text{мс}} = 4.8 \cdot 440 = 2112 \text{ грн.}$$

5.3.6 Витрати на сервер

У якості сервера використовується ПК Asus VivoBook 15 X540UA-DM3095R. Підставивши у формулу (5.5) значення балансової вартості використовуваних та маніпулятору типу миша (разом 16 420 грн.) отримаємо:

$$A = 16\,420 \cdot 0.25 = 4105 \text{ грн.}$$

Амортизаційні відрахування становлять 4 105 грн.

За формулою (5.6) річний фонд часу роботи ЕОМ (сервера) становить 8760 годин. Середні коефіцієнти завантаження за часом та за потужністю становлять 0.6 та 0.9.

$$C_{\Sigma 2} = 0.2 \cdot 8760 \cdot 0.9 \cdot 0.6 \cdot 1.68 = 1589.41 \text{ грн.}$$

Заробітна плата обслуговуючого персоналу розраховується за формулою (5.7).

Місячна заробітна плата обслуговуючого персоналу становить 5100 грн., а річний фонд заробітної плати становить 61200 грн. Річний ефективний фонд робочого часу обслуговуючого ПК робітника становить 1800 годин/рік. На обслуговування одного ПК витрачається по 2 години на місяць, що на рік становить 24 години. З чого виходить, що:

$$ЗП_{\text{обсл}} = 61200 \cdot (1 + 0.22) \cdot \frac{24}{1800} = 995.52 \text{ грн.}$$

Стаття «Поточний ремонт обладнання» приймається рівною 3% від балансової вартості обладнання і складає 741 грн.

Стаття «Інші витрати» приймається рівною п'яти відсоткам від суми всіх попередніх статей витрат на утримання та експлуатацію устаткування. Сума всіх попередніх статей дорівнює 7430.93 грн., 5% від суми становлять 371.55 грн. Кошторис витрат на утримання та експлуатацію серверу на рік описано у таблиці 5.11.

Для розробки ЕС також експлуатується сервер протягом 55 діб. Таким чином витрати на сплату машинного часу становлять:

$$C_{\text{мс}} = \frac{55}{365} \cdot 8173.48 = 1\,231.62 \text{ грн.}$$

Таблиця 5.11 – Кошторис витрат

Найменування статей витрат	Сума, грн.
Амортизація обладнання	4 105
Експлуатація (крім витрат на поточний ремонт)	1589.41
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	995.52
Поточний ремонт обладнання	741
Інші витрати	371.55
Разом	8173.48

5.3.7 Накладні витрати

Накладними витратами називають витрати на загальне управління, загальногосподарські потреби (заробітна плата апарату управління, канцелярські витрати і т.д.), утримання та експлуатацію будівель. Накладні витрати включають у вартість розробки програми непрямым шляхом – у відсотках до основної заробітної плати розробників. В даному випадку накладні витрати становлять 40% до основної заробітної плати розробників, що складає 11600 грн.

Результати визначення витрат на розробку програми приведені в таблиці 5.12.

Таблиця 5.12 – Вартість розробки програми

Стаття витрат	Сума (грн.)	Питома вага до підсумку (%)
Основна заробітна плата	29000	51.97
Додаткова заробітна плата	2900	5.2
ЄСВ	7018	12.57
Вартість матеріалів	1937.25	3.47
Витрати на ПК	2112	3.79
Витрати на сервер	1231.62	2.21

Продовження таблиці 5.12.

Стаття витрат	Сума (грн.)	Питома вага до підсумку (%)
Накладні витрати	11600	20.79
Разом	55798.86	100.00

5.3.8 Розрахунок експлуатаційних витрат

Витрати, пов'язані з експлуатацією програми протягом року, визначають за формулою (5.9):

$$И = P_{\text{екс.ктс}} \cdot T_{\text{ктс}} + \frac{S_{\text{рп}}}{n \cdot T_c}, \quad (5.9)$$

де $P_{\text{екс.ктс}}$ – експлуатаційні витрати, що припадають на 1 ч машинного часу КТС споживача програмного продукту, грн / машино–год;

$T_{\text{ктс}}$ – сумарний машинний час КТС протягом року, машино–год;

$S_{\text{рп}}$ – сумарні витрати на розробку програми, грн .;

n – кількість споживачів даної програми;

T_c – термін служби даної програми до її морального зносу, років.

$$И = 8173.48 + \frac{55798.86}{1 \cdot 10} = 13753.36 \text{ грн.}$$

5.3.9 Розрахунок витрат на адміністратора

Місячна заробітна плата адміністратора становить 5500 грн., а річний фонд заробітної плати становить:

$$ЗП_{\text{адм}} = 5500 \cdot 12 \cdot 1.22 = 80520 \text{ грн.}$$

5.4 Розрахунок капітальних вкладень

Обсяг капітальних вкладень споживача програми K , пов'язаних з її розробкою / купівлею / і впровадженням, визначають за формулою (5.10):

$$K = K_{\text{пп}} + K_{\text{пф}}, \quad (5.10)$$

де $K_{\text{пп}}$ – виробничі витрати, грн. ;

$K_{\text{пф}}$ – капітальні вкладення в виробничі фонди, необхідні для впровадження програми, грн.

Виробничі витрати включають витрати, пов'язані з розробкою і налагодженням програмного, математичного та інформаційного забезпечення, проектуванням системи. Виробничі витрати визначають за формулою (5.11):

$$K_{\text{пп}} = \frac{S_{pn}}{n} + K_{\text{пр.пп}}, \quad (5.11)$$

де S_{pn} – сумарні витрати на розробку або придбання програмного продукту, грн;

n – кількість споживачів даної програми;

$K_{\text{пр.пп}}$ – інші виробничі витрати, грн.

Капітальні вкладення в виробничі фонди включають витрати на придбання, установку, монтаж і налагодження комплексу технічних засобів (КТЗ) з урахуванням можливого використання КТЗ як для вирішення завдань для яких призначена розробляється програма, так і для завдань, що вирішуються за допомогою інших програм. Питомі капітальні вкладення в виробничі фонди, припадаючі на частку даного програмного продукту визначають за формулою (5.12):

$$K_{\text{пф}} = \frac{K_{\text{ктс}} \cdot T_{\text{ктс}}}{\Phi_{\text{еф.ктс}}}, \quad (5.12)$$

де $K_{\text{ктс}}$ – витрати на придбання, установку, монтаж і налагодження КТЗ, грн .;

$T_{\text{ктс}}$ – машинний час КТЗ, потрібне даному споживачеві програми для завдань, що вирішуються за допомогою даної програми, машино–год / рік;

$\Phi_{\text{еф.ктс}}$ – річний ефективний фонд часу роботи КТЗ, машино–год / рік.

$$K = \frac{55798.86}{1} + \frac{16\,420 \cdot 8\,760}{8\,760} = 72218.86 \text{ грн.}$$

5.5 Оцінка економічної ефективності

Створення даного застосунку направлено на отримання додаткового прибутку, залучення нових клієнтів і збільшення обсягу послуг.

За рахунок продажу підписки на розширений набір функцій продукту користувачам (очікується продати 1200 підписок за мінімальною вартістю 120 гривень за копію).

Прибутком будуть вважатися кошти, отримані від підписки користувачів за виключенням заробітної плати адміністратору та експлуатаційних витрат. Адміністратор буде контролювати правильність функціонування системи та за необхідності вносити зміни.

Прибуток розраховується за формулою (5.13):

$$\Pi = \text{Пост}_{\text{від продажу}} - 3\Pi_{\text{адм}} - \text{И}, \quad (5.13)$$

$$\Pi = 1200 \cdot 120 - 80\,520 - 13753.36 = 49726.64 \text{ грн.}$$

Річна економія розраховується за формулою (5.14):

$$\mathcal{E} = \Pi - E_n^a \cdot K, \quad (5.14)$$

де $\mathcal{E}$ – річна економія від впровадження системи, грн .;

Π – річний прибуток від впровадженої системи;

E_n^a – нормативний коефіцієнт ефективності капітальних вкладень в засоби автоматизації;

K – обсяг капітальних витрат, пов'язаних зі створенням ПП.

$$\mathcal{E} = 49726.64 - 0.15 \cdot 72218.86 = 38893.81 \text{ грн.}$$

Коефіцієнт ефективності вкладених коштів в програму і термін їх окупності розраховують по формулам (5.15) та (5.16):

$$E = \frac{\Pi}{K}, \quad (5.15)$$

$$T_{\text{ок}} = \frac{K}{\Pi}, \quad (5.16)$$

Таким чином за формулами (5.15) та (5.16):

$$E = 49726.64 / 72218.86 = 0.69,$$

$$T_{\text{ок}} = 72218.86 / 49726.64 = 1.45 \text{ року.}$$

Таким чином, зроблено висновок про економічну доцільність розробки продукту, так як окупність продукту відбувається за 1,45 року, що менше за допустиме значення часу на окупність продукту, яке дорівнює 6,67 року.

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Аналіз потенційних небезпек

Методологічною основою охорони праці є науковий аналіз умов праці, технологічних процесів, виробничого обладнання, робочих місць, трудових операцій, організації виробництва з метою виявлення шкідливих і небезпечних виробничих факторів, їх властивостей, особливостей впливу на організм людини.

Основні потенційні небезпеки при проведенні робіт в офісі:

- небезпека ураження електричним струмом;
- порушення роботи кістково-м'язового апарату внаслідок тривалих статичних навантажень при роботі з ПК, наприклад тунельний синдром;
- нервово-психічні перевантаження внаслідок постійного контакту з людьми, а саме з клієнтами, колегами по роботі, керівництвом,
- такі, що можуть носити конфліктний характер і призвести до емоційного дискомфорту, вигорання, внутрішнього роздратування, емоційної нестабільності та захворювань нервової системи;
- незадовільні ергономічні характеристики робочого місця внаслідок нераціонального планування робочого місця, що може призвести до механічних травм, уражень електричним струмом та іншого;
- негативний вплив недостатнього освітлення робочої зони на зір та продуктивність роботи працюючого, внаслідок несправності освітлювальних приладів або неправильного проєктування освітлювальної системи;
- негативний вплив незадовільних параметрів повітряного середовища робочої зони на здоров'я працюючого, внаслідок неправильного проєктування системи вентиляції або її несправності;

- негативний вплив підвищеного рівня шуму на психоемоційний стан працюючого, який пов'язаний з використанням застарілої периферійної техніки, кондиціонерів, копіювальної техніки, освітлювальних приладів;
- пожежна небезпека у зв'язку з несправністю електричного обладнання, недотриманням правил протипожежної безпеки обслуговуючим персоналом чи працівниками;
- неправильні дії персоналу у надзвичайних ситуаціях.

В роботі досліджується засоби спільного редагування JS коду в реальному часі. Команда складається з двох програмістів та одного дизайнера, кожен з яких відповідає за свою частину розробки - створення інтерфейсу, створення клієнтської частини, створення серверної частини. Передбачається, що вони розміщуються в офісному приміщенні. Розміри приміщення: довжина $A = 10\text{м}$, ширина $B = 5\text{м}$, висота $H = 3\text{м}$.

Площа приміщення становить 50 м^2 , об'єм – 150 м^3 .

Небезпечні та шкідливі виробничі фактори за природою виникнення поділяються на такі групи: фізичні (підвищена і знижена температура повітря, надмірна запиленість і загазованість повітря та ін.); хімічні (вміст у робочій зоні шкідливих речовин та ін.); психофізіологічні (нервово-емоційні перевантаження, розумове напруження та ін.); біологічні (підвищений вміст у повітрі мікроорганізмів та ін.).

У даному розділі було розглянуто наступні питання: мікроклімат та склад повітря робочої зони; освітлення робочого приміщення; шум та вібрації; виробниче випромінювання; безпека щодо організації робочих місць; безпечність технологічного обладнання та процесу; електробезпека; пожежна безпека.

6.2 Заходи із забезпечення безпеки

Приміщення офісу, в яких перебувають співробітники, відносяться до приміщень без підвищеної небезпеки ураження електричним струмом.

Обладнання, що використовується в цих приміщеннях є споживачем електроенергії, що живиться від змінного струму 220В від мережі з заземленою нейтраллю, та відноситься до електроустановок до 1000В закритого виконання. За способом захисту людини від ураження електричним струмом відповідає згідно з ГОСТ 12.2.007.0-75 (2001) «ССБТ. Изделия электротехнические. Общие требования безопасности» І (стаціонарні комп'ютери,) та ІІ (освітлювальні прилади, кондиціонери, опалювальні пристрої, ноутбуки, сканери) класу захисту.

Згідно «Правилам улаштування електроустановок» (далі «ПУЕ») виконані такі групи заходів з електробезпеки:

- конструктивні заходи забезпечують захист від випадкового дотику до струмопровідних частин за допомогою їх ізоляції та захисних оболонок;
- згідно з ГОСТ 12.1.009-76 (1999) «ССБТ. Электробезопасность. Термины и определения» у приладах ІІ класу захисту використовується подвійна ізоляція - електрична ізоляція, що складається з робочої і додаткової ізоляції;
- так як згідно з НПАОП 40.1-1.32-01 «Правила устройства электроустановок. Электрооборудование специальных установок» офісні приміщення у більшості своїй відносяться до класу пожежонебезпечної зони П-Па (приміщення, в яких містяться тверді горючі речовини), тому передбачений ступінь захисту ізоляції обладнання IP44.

Схемно-конструктивні заходи призначені для забезпечення захисту від ураження електричним струмом при дотику до металевих оболонок, які можуть опинитися під напругою в результаті аварій. Згідно з ГОСТ 12.1.030-81 (2001)

«ССБТ. Електробезпека. Захисне заземлення, занулення» у приміщеннях галузі управління персоналом влаштовується занулення.

Організаційні заходи, такі як експлуатація електроустановок і електроустаткування проводиться відповідно до НПАОП 40.1-1.01-97 «Правила безпечної експлуатації електроустановок» (далі «ПБЕЕ») та НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів» (далі «ПБЕЕС»).

Для запобігання статистичного навантаження при користуванні ПК рекомендовано використовувати перерви в роботі 10 хв. через кожні дві години. Синдром зап'ястного каналу, або тунельний синдром зап'ястя, який може бути наслідком хронічної травми, трапляється у людей внаслідок тривалої роботи з мишею: постійні напруга і здавлювання приводить до мікротравм, здавлювання нерва прилеглими оточуючими тканинами, через що виникає набряк.

Щоб тунельний синдром не турбував персонал, дотримано основні правила організації робочого місця:

- оптимальна висота клавіатури від підлоги – 65-75 см;
- наявність ергономічних і зручних особисто для вас миші і клавіатури;
- можливість регулювання висоти і нахилу клавіатури (відстань від поверхні стола до середини клавіатури – не більше 30 мм, кут підйому клавіатури – від 2 град до 15 град);
- наявність у клавіатури підставки для рук;
- наявність килимка для миші з захистом від тунельного синдрому (спеціальний виступ забезпечує правильне положення кисті);
- наявність стільця або крісла з підлокітниками.

При роботі з мишкою і клавіатурою також слід дотримуватися певних правил. Коли набирається текст, рука повинна бути зігнута в лікті під прямим кутом, а при роботі з мишкою стежте, щоб кисть була прямою і лежала на столі якнайдалі від краю.

Щоб попередити тунельний синдром працівникам було рекомендовано робити спеціальні вправи для зап'ясть – чим частіше, тим краще. Ці вправи допоможуть поліпшити кровообігу в м'язах і розтягнути їх. Комплекс вправ потрібно повторювати приблизно кожні 45 хвилин, тривалість однієї вправи – 1-2 хв.

Нервові напруження впливає на серцево-судинну систему, збільшуючи артеріальний тиск і частоту пульсу, а також на терморегуляцію організму та емоційний стан працівника. Особливу роль у запобіганні втоми працівників відіграють професійний відбір, організація робочого місця, правильне робоче положення, ритм роботи, раціоналізація трудового процесу, використання емоційних стимулів, впровадження раціональних режимів праці і відпочинку тощо. Боротьба зі втомою зводиться до покращення санітарно-гігієнічних умов виробничого середовища (ліквідація забруднення повітря, шуму, вібрації, нормалізація мікроклімату, раціональне освітлення тощо).

Крім того, для профілактики втоми працівників застосовуються специфічні методи, до яких можна віднести засоби відновлення функціонального стану зорового та опорно-рухового апарату, зменшення гіподинамії, підсилення мозкового кровообігу, оптимізацію розумової діяльності.

Загальні ергономічні вимоги встановлено ДСТУ ISO 9241-1:2003 «Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 1. Загальні положення». Організація робочого місця передбачає: правильне розміщення робочого місця у виробничому приміщенні; вибір ергономічно-обґрунтованого робочого положення, виробничих меблів з урахуванням антропометричних характеристик людини; раціональне компонування обладнання на робочих місцях; врахування характеру та особливостей трудової діяльності.

В результаті технічного переоснащення, що базується на впровадженні інформаційних технологій у приміщенні, що знаходиться на другому поверсі виробничого корпусу встановлено 3 робочих місця з комп'ютерами. Визначимо

скільки комп'ютеризованих робочих місць, оснащених відеодисплейними терміналами (ВДТ) насправді можна встановити у даному приміщенні і як їх розташувати відповідно до встановлених норм та правил з охорони праці.

Відповідно до НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин» є неприпустимим розташування приміщень, призначених для роботи з ВДТ у підвалах та цокольних поверхах. Також забороняється розташування вибухонебезпечних приміщень категорії А і Б (НАПБ Б.03.002-2007 «Нормы определения категорий помещений, зданий и наружных установок по взрывопожарной и пожарной безопасности») та виробництв з мокрими технологічними процесами поряд з приміщенням, де розташовуються ЕОМ (ПЕОМ), а також над такими приміщеннями, або під ними. Окрім того, виробничі приміщення для роботи з ВДТ не повинні межувати з приміщеннями, у яких рівень шуму і вібрації перевищує допустимі значення.

Вибране на початку розділу приміщення відповідає вищезазначеним вимогам, тому перейдемо до наступного етапу - визначення кількості комп'ютеризованих робочих місць, що можна розмістити в даному приміщенні. Оскільки площа приміщення становить $S_{пр} = 50 \text{ м}^2$, а площа, на якій розташовується одне робоче місце з ВДТ, відповідно до вимог ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» повинна становити не менше $6,0 \text{ м}^2$, в даному приміщенні можливе розміщення трьох комп'ютеризованих робочих місць. Це саме та необхідна кількість для двох програмістів та дизайнера.

Перевіримо, чи відповідає це число нормативу щодо мінімального об'єму приміщення на одне робоче місце з ВДТ, відповідно до вимог ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ($V_{P.M.Min} = 20 \text{ м}^3$). Об'єм приміщення становить 150 м^3 , а об'єм, що припадає на одне

комп'ютеризоване робоче місце - $V_{р.м.} = 50 \text{ м}^3$. Таким чином норматив щодо об'єму приміщення на одне робоче місце з ВДТ виконується.

Планування розміщення комп'ютеризованих робочих місць у приміщенні було проведено із врахуванням наступних вимог:

- робочі місця з ВДТ розміщуються на відстані не менше 1 м від стіни зі світловими прорізами;
- відстань між бічними поверхнями ВДТ має бути не менше за 1,2 м;
- відстань між тильною поверхнею одного ВДТ та екраном іншого не повинна бути меншою за 2,5 м;
- прохід між рядами робочих місць має бути не меншим 1 м.

Враховано розміри меблів на комп'ютеризованих робочих місцях, зокрема розміри робочих столів. Відповідно до НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин» рекомендовані розміри столу для робочого місця з ВДТ становлять: висота – 725 мм, ширина – 600-1400 мм, глибина – 800-1000 мм. В офісному приміщенні встановлено столи з наступним розміром: висота - 725 мм, ширина – 1200 мм, глибина – 800 мм.

Комп'ютеризовані робочі місця розміщені вздовж стіни з вікнами. Це дає змогу унеможливити дзеркальне відбиття на екрані ВДТ джерел природного світла (вікон) та потрапляння останніх у поле зору операторів, що погіршує умови їх зорової роботи.

6.3 Заходи з виробничої санітарії та гігієни праці

Згідно ДБН В.2.5-28-2018 «Природне і штучне освітлення» в офісних приміщеннях використовується природне та штучне освітлення. Природне освітлення здійснено через світлові прорізи, які забезпечують коефіцієнт природної освітленості (КПО) не нижче 1,5%. Для захисту від прямих сонячних променів, які створюють прямі та відбиті відблиски на поверхні екранів і

клавіатури, використовуються сонцезахисні пристрої, на вікнах встановлені жалюзі або штори.

Найбільш прийнятними для приміщень, де працюють ПК, є люмінесцентні лампи ЛБ (білого світла) і ЛТБ (тепло-білого світла) потужністю 20, 40 або 80 Вт.

Розраховуємо кількість ламп необхідних для освітлення приміщення розміром 10х5х3м освітленістю $E_p = 400 \text{лк}$. Коефіцієнт відбиття стелі 70% і стін 50%. Для освітлення використовуються люмінесцентні лампи типу ЛБ в світильниках ЛПО.

Індекс приміщення знаходиться за формулою (6.1):

$$i = A \cdot B \cdot h \cdot (A + B), \quad (6.1)$$

де A та B довжина та ширина приміщення;

h - висота установки світильників над розрахунковою.

$$i = 10 \cdot 52.28 \cdot (10 + 5) = 1.46.$$

Приймаємо коефіцієнт запасу $k = 1.6$ і коефіцієнт нерівномірності освітлення $z = 1.1$.

При індексі приміщення рівному 1.46 та коефіцієнтах відбиття світла від стін та стелі рівними 70% та 50% отримуємо, що коефіцієнт використання світлового потоку $\eta = 55\%$.

Світильники розміщуємо в два ряди ($N_p = 2$).

Визначаємо необхідний світловий потік ламп за формулою (6.2):

$$\Phi_p = E_n \cdot S \cdot k \cdot z \cdot N_p \cdot \gamma, \quad (6.2)$$

де E_n – нормоване значення освітленості;

S – площа освітленої поверхні;

k – коефіцієнт запасу;

z – коефіцієнт нерівномірності (мінімальної) освітленості;

γ – коефіцієнт використання світлового потоку.

$$\Phi_p = 400 \cdot 50 \cdot 1.1 \cdot 1.62 \cdot 0.55 = 32000 \text{ лм.}$$

Якщо в світильнику встановити по дві лампи ЛБ потужністю 80Вт і світловим потоком 5200лм, то необхідне число світильників розраховується за формулою (6.3):

$$N_p = \Phi_p \div (n \cdot \Phi_{\text{л}}), \quad (6.3)$$

де n – кількість ламп;

Φ_p – необхідний світловий потік;

$\Phi_{\text{л}}$ – світловий потік лампи.

$$N_p = 32000 \div (2 \cdot 5200) = 3.$$

Показники мікроклімату в офісних приміщеннях відповідають встановленим санітарно-гігієнічним вимогам ДСН 3.3.6-042-99 «Санітарні норми мікроклімату виробничих приміщень», ГОСТ 12.1.005-88 (1991) «ССБТ. Общие санитарно-гигиенические требования к воздуху рабочей зоны» і ГН 2152-80 «Санітарно-гігієнічні норми допустимих рівнів іонізації повітря виробничих та громадських приміщень».

Тип роботи програмістів в обчислювальному центрі належить до операторської роботи, даний тип роботи відносить до категорії 1а, тобто до неї належать роботи, що виконуються сидячи і не потребують фізичного напруження, робоче місце є постійним, робота проводиться у холодний та

теплий періоди року, тому були здійснені налаштування для підтримання оптимальних показників мікроклімату, а саме:

- у холодний період року: температура 22-24°C; відносна вологість: 40-60%; швидкість переміщення повітря: 0,1 м/с;
- у теплий період року: температура 23-25°C; відносна вологість: 40-60%; швидкість переміщення повітря: 0,2 м/с.

Оптимальні рівні позитивних і негативних іонів у повітрі приміщення з ПК нормовані згідно ГН 2152-80 «Санітарно-гігієнічні норми допустимих рівнів іонізації повітря виробничих та громадських приміщень» і становлять: 1500-30000шт. на 1 кубічний сантиметр для позитивних та 3000-5000шт. для негативних. Підтримка оптимального рівня легких позитивних і негативних аероіонів у повітрі на робочих місцях забезпечено за допомогою біполярних коронних аеройонізаторів.

Дотримання вимог цих документів досягається оснащенням приміщень транспортних засобів пристроями кондиціонування і вентиляції, дезодорації повітря, опалення згідно вимог ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування». Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях у приміщення нормуються згідно ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».

Зниження рівня шуму в приміщенні здійснено за допомогою:

- використання більш сучасного обладнання;
- розташування принтерів та різноманітного устаткування колективного користування на значній відстані від більшості робочих місць працівників;
- переведення персонального комп'ютера в режим сну (Standby), якщо комп'ютер не працює протягом визначеного часу;

– використання блоків живлення ПК з вентиляторами на гумових підвісках.

6.4 Заходи безпеки у надзвичайних ситуаціях

6.4.1 Заходи з пожежної безпеки

Закон України «Про пожежну безпеку» визначає загальні правові, економічні та соціальні основи забезпечення пожежної безпеки на території України, регулює відносини державних органів, юридичних і фізичних осіб у цій галузі незалежно від виду їх діяльності та форм власності.

Пожежна безпека – стан об’єкта, при якому з регламентованою імовірністю виключається можливість виникнення та розвиток пожежі і впливу на людей її небезпечних факторів, а також забезпечується захист матеріальних цінностей.

Для забезпечення пожежної безпеки було проведено пожежну профілактику, яка включає в себе комплекс організаційних і технічних заходів, спрямованих на забезпечення безпеки людей, на запобігання пожежі, обмеження її поширення, а також на створення умов для успішного гасіння пожежі.

Для ліквідації пожежі у початковій стадії її розвитку силами персоналу приміщення було обладнано наступними первинними засобами пожежогасіння, а саме - вогнегасниками, пожежним інвентарем (покривала з негорючого теплоізоляційного полотна, ящики з піском, пожежні відра, совкові лопати, ломи, сокири тощо), системою автоматичного пожежогасіння.

Залежно від агрегатного стану й особливостей горіння різних горючих речовин і матеріалів пожежі за ДБН В.1.1.7-2002 «Пожежна безпека об’єктів будівництва» поділяються на відповідні класи: офісному приміщенні знаходиться дерев'яна мебель, електронна апаратура, бумажні носії інформації.

Клас пожежі у офісному приміщенні (згідно з ДБН В.1.1.7-2002 «Захист від пожежі. Пожежна безпека об'єктів будівництва») – пожежі твердих речовин, переважно органічного походження, горіння яких супроводжується тлінням (деревина, пластмаси, папір) – визначається як клас А. Категорія приміщення (згідно із НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою») – визначається як категорії Д.

Було визначено тип та розраховано кількість первинних засобів пожежогасіння (згідно із ДБН В.1.1.7-2002 «Захист від пожежі. Пожежна безпека об'єктів будівництва») – для офісного приміщення площею 50 м² застосовано два порошкових вогнегасники типу ВП–5 (НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою»). ДБН В.1.1.7-2002 «Пожежна безпека об'єктів будівництва»). Крім цього офісне приміщення обладнане автоматичними пожежними сповіщувачами, що реагують на підвищення температури, дим, полум'я.

6.5 Заходи з цивільного захисту.

Захисні споруди цивільної оборони призначаються для захисту в мирний час персоналу, який переховується від наслідків аварій, катастроф та стихійного лиха, які загрожують масовому ураженню людей, а також у воєнний час - від сучасної зброї масового ураження. В мирний час захисні споруди використовуються для господарчих потреб.

За захисними властивостями споруди поділяються найпростіші укриття (відкриті та перекриті щілини), сховища і протирадіаційні укриття (ПРУ);

Сховища відповідно діляться на класи, а протирадіаційні укриття (ПРУ) на групи.

За призначенням захисні споруди будують для захисту населення та для розміщення органів управління (пунктів управління, вузлів зв'язку) і медичних установ.

За місцем розташування споруди бувають:

- вбудовані (за техніко-економічними, експлуатаційними показниками мають перевагу);
- що окремо розташовані (допускаються в тих випадках, коли за об'єктивними причинами будівництво вбудованих унеможлиблюється).

За термінами будівництває такі, що побудовані завчасно, а також такі, що швидко зводяться (з/б плити з висотою ґрунту не менше 1,2 м зверху).

Захисні споруди за місткістю умовно поділяють на споруди малої місткості (до 150 чол), середньої місткості (151 – 600 чол) і великої місткості (600 чол. та більше).

Найпростіші укриття – це споруди, що зменшують радіуси ураження людей світловим випромінюванням, проникаючою радіацією, ударною хвилею та послаблюють дію радіоактивних випромінювань.

У системі захисту населення особливе значення мають найпростіші укриття типу щілин. Це найбільш масові захисні споруди, що можуть бути збудовані населенням у найкоротший термін. Щілини будують відкритими і перекритими. Відкрита щілина зменшує ймовірність ураження ударною хвилею (в 1,2 – 2 рази), світловим випромінюванням і проникаючою радіацією. Перекрита щілина захищає: від світлового випромінювання – повністю, від ударної хвилі у 1,5 – 3 рази, від проникаючої радіації і радіоактивного випромінювання – у 200-300 разів, а також надійно захищає від осколкових і кулькових бомб, від запалювальних засобів.

Відкрита щілина - зигзагоподібна траншея з кількох прямолінійних ділянок довжиною до 15м глибина –1,8–2,0м., шириною: зверху – 1,1 – 1,3м., на дні – 0,8м. Будівництво щілини починається з розмітки і трасування, тобто визначення її плану на місцевості.

В міру поглиблення поступово порівнюють крутизну, доводячи до потрібних параметрів. Стінки (крутизну) щілини укріплюють дошками, жердинами, очеретом, іншими наявними матеріалами. Для вентиляції встановлюють витяжну коробку. Нормальна місткість щілини - 10-15 чоловік.

Сховища є найбільш надійним захистом від усіх уражальних чинників: високих температур і шкідливих газів у зонах пожеж, вибухонебезпечних, радіоактивних і сильнодіючих отруйних речовин, обвалів і уламків зруйнованих будинків і споруд тощо, а також засобів масового ураження і звичайних засобів ураження. Воно обладнане комплексом інженерних споруд, що забезпечують необхідні умови життєдіяльності протягом певного часу.

За місцем знаходження сховища бувають збудованими (у підвалах будинків) і відокремленими (поза будинками). Їх споруджують заздалегідь, у мирний час, але можуть будувати і в період загрози нападу або під час воєнних дій (швидко зведені).

За місткістю розрізняють малі сховища (150-300 чол.), середні (300-600 чол.) і великі (понад 600 чол.).

Система водопостачання забезпечує людей водою для життя і гігієнічних потреб від зовнішньої водопровідної мережі. На той випадок, коли водопровід перестане діяти, передбачено аварійний запас води або її джерело. Кожна захисна споруда має систему каналізації для відводу фекальних стоків. Санвузол влаштовують у приміщенні, ізольованому перегородками від секцій сховища, обов'язково з витяжкою. Система опалення сховища працює від опалювальної мережі будинку, під яким воно знаходиться. Освітлюється сховище від міської (об'єктної) електромережі, в аварійних випадках – від автономної електростанції, а якщо її немає – від акумуляторів або ліхтарями.

Запас продуктів харчування робиться з розрахунку не менше ніж на дві доби для кожної людини у сховищі. Медичне обслуговування здійснюють санітарні пости і медпункти об'єктів народного господарства.

За призначенням у сховищах передбачається основні та допоміжні приміщення.

До основних приміщень належать: приміщення для осіб, що укриваються; пункти управління; медичний пункт; операційно-перев'язочні, передопераційно-стерилізаційні (тільки у лікувальних установах).

До допоміжних приміщень належать: тамбур – шлюзи (тамбури); фільтровентиляційні приміщення; санітарні вузли; захищені ДЕС. електрощитові; приміщення для зберігання продовольства та ємностей запасу питної води; приміщення артезіанської свердловини; дренажні станції перекачування; балонні приміщення; приміщення для дозиметричного контролю, роздягальні, приміщення для брудного одягу, душові (тільки для сховищ АЕС).

У пристосованих під укриття приміщеннях встановлюють бачки з водою з розрахунку 3-4 л. на одну людину на добу, а в туалеті виносну тару або влаштовують люфт-клозет з вигрібною ямою. Крім того, в укритті встановлюють нари (лавки) для відпочинку, стелажі для продуктів харчування. освітлення – від електромережі або переносними електричними ліхтарями.

ВИСНОВКИ

В результаті виконання роботи було досліджено засоби, що дозволяють організувати спільне редагування коду в реальному часі та розроблено вебзастосунок для спільного написання та редагування програмного коду на мові програмування JS в реальному часі, що дозволяє створювати швидкі прототипи та алгоритми при командному вирішенню задач, тестуванню та демонстрації логічних блоків застосунку; допомагати знаходити рішення проблем віддаленим товаришам та організовувати навчання.

Розроблений вебзастосунок поділяється на дві частини – клієнтська частина, що реалізована за допомогою бібліотеки React та з використанням CodeMirror у якості текстового редактору, серверна частина, що реалізована за допомогою фреймворку ExpressJS, бібліотеки для обміну даними в реальному часі – Socket.IO та бази даних ShareDB, що забезпечує можливим конкурентний запис та оновлення інформації в реальному часі завдяки тому, що самостійно реалізовує технологію ОП.

Створений вебзастосунок має базовий функціонал, що включає в себе найбільш затребувані можливості. Обмежень по кількості одночасних користувачів в кімнаті немає, але рекомендовано не виконувати редагування одночасно більше ніж трьом особам для забезпечення збереження наміру (виконання конкретної операції над конкретними даними) кожного із редакторів.

У роботі була проаналізована предметна область, були розглянуті різні підходи до вирішення поставленого завдання, вже існуючі аналоги.

Для розробки вебзастосунку було використано редактор вихідного коду Visual Studio Code, мову програмування JS та програмну платформу NodeJS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Стоит ли учить JavaScript: перспективы, ситуация на рынке труда, мнения экспертов [Электронный ресурс]. – Режим доступа : <https://ru.hexlet.io/blog/posts/stoit-li-uchit-javascript-perspektivy-situatsiya-na-rynke-truda-mneniya-ekspertov>.
2. On Pair Programming [Electronic resource]. – Access mode : <https://martinfowler.com/articles/on-pair-programming.html#Challenges>.
3. Эксперимент — от парного программирования к программированию всей командой [Электронный ресурс]. – Режим доступа : <https://www.pvsm.ru/cat/mob-programmirovanie>.
4. Mob Programming [Electronic resource]. – Access mode : <https://www.agilealliance.org/glossary/mob-programming>.
5. Мы писали код вместе за одним компьютером пять месяцев. Вот что я узнал [Электронный ресурс]. – Режим доступа : <https://habr.com/ru/post/350588/>.
6. 12 лучших онлайн-редакторов кода для вебразработчиков [Электронный ресурс]. – Режим доступа : <https://techrocks.ru/2020/03/05/12-best-online-code-editors>.
7. CodeMirror [Electronic resource]. – Access mode : <https://codemirror.net/index.html>.
8. Ace (editor) [Electronic resource]. – Access mode : [https://en.wikipedia.org/wiki/Ace_\(editor\)](https://en.wikipedia.org/wiki/Ace_(editor)).
9. Monaco Editor. About [Electronic resource]. – Access mode : <https://microsoft.github.io/monaco-editor/>.
10. IDE нормального человека или почему мы выбрали Monaco [Электронный ресурс]. – Режим доступа : <https://habr.com/ru/company/Voximplant/blog/445390/>.
11. Top 10 Node.js Frameworks For Web App Development in 2020-21

[Electronic resource]. – Access mode : <https://medium.com/javascript-in-plain-english/top-10-node-js-frameworks-for-web-app-development-in-2020-21-a2a57e5>.

12. React [Электронный ресурс]. – Режим доступа : <https://uk.wikipedia.org/wiki/React>.

13. Vue.js [Электронный ресурс]. – Режим доступа : <https://uk.wikipedia.org/wiki/Vue.js>.

14. Angular vs React vs Vue: Which Framework to Choose in 2020 [Electronic resource]. – Access mode : <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>.

15. Operational transformation [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Operational_transformation.

16. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems [Electronic resource]. – Access mode : <https://dl.acm.org/doi/10.1145/274444.274447>.

17. Operational Transformations as an algorithm for automatic conflict resolution [Electronic resource]. – Access mode : <https://medium.com/coinmonks-operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447>.

18. Operational Transformation, the real-time collaborative editing algorithm [Electronic resource]. – Access mode : <https://medium.com/srijancse/operational-transformation-the-real-time-collaborative-editing-algorithm-bf8756683f66>.

19. Is implementing operational transform hard? [Electronic resource]. – Access mode : <https://digitalfreepen.com/2018/01/04/operational-transform-hard.html>.

20. ShareDB [Electronic resource]. – Access mode : <https://github.com/share/sharedb>.

ДОДАТОК А
Текст програми

Серверна частина вебзастосунку

A.1 Файл share-db-server.js

```
const ShareDB = require('sharedb');
const otText = require('ot-text');
const http = require('http');
const WebSocket = require('ws');
const WebSocketJSONStream = require('websocket-json-stream');

ShareDB.types.register(otText.type);

class ShareDBServer {
  constructor() {
    this.share = new ShareDB();
  }

  getConnection() {
    if (!this.connection) {
      this.connection = this.share.connect();
    }
    return this.connection;
  }

  listen(port = 8080) {
    const httpServer = http.createServer();
    const websocketServer = new WebSocket.Server({ server: httpServer });
    websocketServer.on('connection', (client) => {
      this.share.listen(new WebSocketJSONStream(client)),
    });
    httpServer.listen(port);
    console.log(ShareDBServer is listening on port ${port});
  }
}

module.exports = {
  ShareDBServer,
};
```

A.2 Файл index.js

```
const { generateName } = require('./utils/unique-name-generator.js');
const http = require('http');
const express = require('express');
const socketio = require('socket.io');
const { ShareDBServer } = require('./share-db-server');
const {
  disconnecting,
  createRoom,
```

```

    joinRoom,
    leaveRoom,
    updateAnchors,
  } = require('./actions');

const shareDbServer = new ShareDBServer();
const shareDBConnection = shareDbServer.getConnection();
shareDbServer.listen();

const app = express();
const server = http.createServer(app);
const io = socketio(server);

// map of all rooms instead of DB
const rooms = {};

io.on('connection', (socket) => {
  const userName = generateName();
  socket.emit('i_am_connected', { userName });

  // subscribe on actions
  socket.on('disconnecting', () =>
    disconnecting(io, socket, userName, rooms),
  );
  socket.on('create_room', (args) =>
    createRoom(io, socket, userName, rooms, {
      ...args,
      shareDBConnection,
    }),
  );
  socket.on('join_room', (args) =>
    joinRoom(io, socket, userName, rooms, args),
  );
  socket.on('leave_room', (args) =>
    leaveRoom(io, socket, userName, rooms, args),
  );
  socket.on('update_anchors', (args) =>
    updateAnchors(io, socket, userName, rooms, args),
  );
});

const port = 3000;
server.listen(port);
console.log(Server is listening on port ${port});

```

A.3 Файл utils/unique-name-generator.js

```

const {
  uniqueNamesGenerator,

```

```

    adjectives,
    colors,
    animals,
  } = require('unique-names-generator');

  module.exports.generateName = () => {
    return uniqueNamesGenerator({
      dictionaries: [adjectives, colors, animals],
      separator: ' ',
      style: 'upperCase',
    });
  };
};

```

A.4 Файл actions/create-room.js

```

const { v4: uuidv4 } = require('uuid');

module.exports.createRoom = (
  io,
  socket,
  userName,
  rooms,
  { roomName, shareDBConnection },
) => {
  const roomId = uuidv4();

  // create an empty document
  const document = shareDBConnection.get('docs', roomId);
  if (!document.data) {
    document.create('', 'text');
  }

  const room = {
    roomId,
    roomName,
    users: [{ userId: socket.id, userName, anchors: [0, 0] }],
  };
  rooms[roomId] = room;
  socket.join(roomId);
  socket.emit(i_joined_room, room);
};

```

A.5 Файл actions/disconnecting.js

```

module.exports.disconnecting = (io, socket, userName, rooms) => {
  Object.values(socket.rooms).map((roomId) => {
    if (roomId === socket.id) {
      return
    }
  })
};

```

```

    }
    const room = rooms[roomId];
    room.users = room.users.filter((user) => user.id !== socket.id);
    io.to(roomId).emit('user_left_room', {
      userId: socket.id,
      userName,
      roomId,
    });
  });
};

```

A.6 Файл actions/join-room.js

```

module.exports.joinRoom = (io, socket, userName, rooms, { roomId }) => {
  const room = rooms[roomId];
  if (!room) {
    socket.emit(error, { message: 'Room is not found.' });
    return;
  }
  const anchors = [0, 0];
  room.users.push({ userId: socket.id, userName, anchors });
  socket.join(roomId);
  socket.emit(i_joined_room, room);
  io.to(roomId).emit('user_joined_room', {
    roomId,
    userId: socket.id,
    userName,
    newUserAnchors: anchors,
  });
};

```

A.7 Файл actions/leave-room.js

```

module.exports.leaveRoom = (io, socket, userName, rooms, { roomId }) => {
  const room = rooms[roomId];
  if (!room) {
    socket.emit(error, { message: 'Room is not found.' });
    return;
  }
  room.users = room.users.filter((user) => user.userId !== socket.id);
  socket.leave(roomId);
  socket.emit(i_left_room, room);
  io.to(roomId).emit('user_left_room', {
    roomId,
    userId: socket.id,
    userName,
  });
};

```

A.8 Файл actions/update-anchors.js

```
module.exports.updateAnchors = (
  io,
  socket,
  userName,
  rooms,
  { anchors, roomId },
) => {
  const room = rooms[roomId];
  if (!room) {
    socket.emit(error, { message: 'Room is not found.' });
    return;
  }
  const user = room.users.find((user) => user.userId === socket.id);
  user.anchors = anchors;
  io.to(roomId).emit('user_updated_anchors', {
    userId: socket.id,
    anchors,
  });
};
```

Клієнтська частина вебзастосунку

A.9 Файл app.js

```
import './App.css';
import { CodeEditor } from './components/CodeEditor';

function App() {
  return (
    <div className="App">
      <Header />
      <CodeEditor />
    </div>
  );
}

export default App;
```

A.10 Файл index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
```

```
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

A.11 Файл tools/generateid.js

```
export const generateId = function () {
  // Math.random should be unique because of its seeding algorithm.
  // Convert it to base 36 (numbers + letters), and grab the first 9 characters
  // after the decimal.
  return '_' + Math.random().toString(36).substr(2, 9);
};
```

A.12 Файл CodeEditor.js

```
import React, { useState, useRef } from 'react';
import { Controlled as CodeMirror } from 'react-codemirror2';
import { io, roomSocket } from '../Room';

const updateConsole = () => {
  return `(function () {
    var logger = window.document.getElementById('content');
    console.log = function () {
      for (var i = 0; i < arguments.length; i++) {
        if (typeof arguments[i] == 'object') {
          logger.innerHTML += (JSON && JSON.stringify ? JSON.stringify(arguments[i], undefined, 2) :
arguments[i]) + '<br />';
        } else {
          logger.innerHTML += arguments[i] + '<br />';
        }
      }
    }
  })();`;
}

const editorOptions = {
  mode: { name: 'javascript', statementIndent: 2 },
  theme: 'material',
  lineNumbers: true,
  extraKeys: {
```

```

    'Ctrl-Space': 'autocomplete',
    'Ctrl-Enter': compile,
  }
};

export const CodeEditor = ({
  sharedoc,
  addUserToList,
}) => {
  // Indicates whether or not we should not watch changes emitted by the editor due to change made via
  operations
  const [stopWatch, setStopWatch] = useState(false);
  const [editorValue, setEditorValue] = useState('');
  const iframeRef = useRef();

  const compile = useCallback(() => {
    const iframeJs = iframeRef.current.contentDocument;

    iframeJs.open();
    iframeJs.write(`
      <body>
        <div id="content"></div>
        <script>
          ${ updateConsole() }
        </script>
        <script id="result_script">
          (function(){${editorValue}})()
        </script>
      </body>
    `);

    iframeJs.close();
  }, [editorValue, iframeRef]);

  // Create operations for sharedb
  const onChangeHandler = (editor, change) => {
    if (stopWatch) {
      return;
    }

    const startindex = editor.indexFromPos(change.from);
    const delta = change.removed.join('\n').length;
    const addedText = change.text.join('\n');

    if (delta) {
      sharedoc.submitOp([startindex, { d: delta }]);
    }
    if (addedText) {
      sharedoc.submitOp([startindex, addedText]);
    }
  }
};

```

```

};

// Update the range in other editors when the selection changes
const onCursorActivityHandler = editor => {
  const startPos = editor.getCursor('start');
  const endPos = editor.getCursor('end');
  const headPos = editor.getCursor('head');

  const stindex = editor.indexFromPos(startPos);
  const edindex = editor.indexFromPos(endPos);
  const hdindex = editor.indexFromPos(headPos);
  const prefixed = hdindex === stindex && stindex !== edindex;

  roomSocket.emit('anchor-update', { stindex, edindex, prefixed })
};

roomSocket.handleConnected(({ e }) => {
  for (let id in e.anchors) {
    io.id !== id && setAnchor(id, e.anchors[id]);
  }

  for (let id in e.names) {
    io.id !== id && addName(id, e.names[id]);
  }
  console.log('Your id is ', e.user_name);
});

roomSocket.handleUserJoinRoom(({ e }) => {
  addUserToList(e.new_user);
});

io.on('connect', () => {
  io.on('disconnect', () => clearAll())

  io.on('i_am_connected',

  io.on('anchor-update', e => {
    if (io.id === e.id) return

    setAnchor(e.id, e.anchor)
  })
  io.on('id-join', e => {
    if (io.id === e.id) return

    addName(e.id, e.name)
    setAnchor(e.id, e.anchor)
  })
  io.on('id-left', e => {
    if (io.id === e.id) return

```

```

        removeId(e.id)
    })
});

const anchorMap = {};
const setAnchor = (id, anchor) => {
    if (id in anchorMap) {
        anchorMap[id].forEach(m => m.clear());
        delete anchorMap[id];
    }

    const edindex = anchor.edindex;
    let stindex = anchor.stindex;
    let emptyClass = '';

    // Add selection
    let stPos, edPos, range;
    anchorMap[id] = [];

    if(stindex !== edindex) {
        stPos = ed.posFromIndex(stindex);
        edPos = ed.posFromIndex(edindex);

        anchorMap[id].push(ed.markText(stPos, edPos, { className: u-`${id} }));
    }

    if (stindex === edindex) {
        stindex = Math.max(0, stindex - 1)
        emptyClass = 'empty'
    }
    // Add cursor
    const index = anchor.prefixed ? stindex : edindex
    stPos = ed.posFromIndex(index + (anchor.prefixed ? 0 : -1))
    edPos = ed.posFromIndex(index + (anchor.prefixed ? 1 : 0))

    anchorMap[id].push(ed.markText(stPos, edPos, {
        className: `u-`${id} ${emptyClass} cursor ${
            anchor.prefixed ? "left" : "right"
        }`
    }));
});

const removeId = id => {
    document.querySelector(#users li.u-`${id}`).remove()
    document.querySelector(#style-`${id}`).remove()
    if (id in anchorMap) {
        anchorMap[id].forEach(m => m.clear())
        delete anchorMap[id]
    }
}

```

```

}

const idToColor = id => {
  let total = 0
  for (let c of id) total += c.charCodeAt(0)

  let hex = total.toString(16)
  while(hex.length < 3) hex += hex[hex.length - 1]
  hex = hex.substr(0, 3)

  let color = '#'
  for (let c of hex) color += `${c}0`

  return color
}

const hexToRgbStyle = (hex, opacity) => {
  hex = hex.replace('#', '')
  let r, g, b, den
  if (hex.length === 3) {
    r = hex[0] + hex[0]
    g = hex[1] + hex[1]
    b = hex[2] + hex[2]
  } else {
    r = hex.substr(0, 2)
    g = hex.substr(2, 2)
    b = hex.substr(4, 2)
  }

  r = parseInt(r, 16)
  g = parseInt(g, 16)
  b = parseInt(b, 16)

  return `rgba(${r},${g},${b},${opacity})`
}

return (
  <div className="container">
    <CodeMirror
      value={editorValue}
      options={editorOptions}
      autoCursor={true}
      onChange={onChangeHandler}
      onCursorActivity={onCursorActivityHandler}
    />
    <div className="console-output">
      <iframe ref={iframeRef} title="result" id="result" />
    </div>
  </div>
)

```

A.13 Файл Header.js

```
import React from 'react'

export const Header = ({
  compileHandler,
  users,
}) => {
  const handleShare = () => {
    const url = window.location.href;
    navigator.clipboard.writeText(url);

    alert('Link copied to clipboard');
  }

  return (
    <div className="header">
      <button onClick={compileHandler} className="cmpl-button" >COMPILE!</button>
      <button onClick={handleShare} className="share-button">SHARE!</button>
      <div className="header-users">
        { users.map(u => <div className="user">{ u.name }</div>) }
      </div>
    </div>
  )
};
```

A.14 Файл Room.js

```
import io from 'socket.io-client';
import { generateId } from '../tools/generateId';

class RoomSocket {
  io = null;
  me = '';
  roomId = '';

  constructor() {
    if (!this.io) {
      this.io = io();
    }
  }

  handleConnected = (cb) => {
    this.io.on('i_am_connected', ({ e }) => {
      this.me = e.user_name;
      cb(e);
    });
  };
}
```

```

    }

    handleUserJoinRoom = (cb) => {
      this.io.on('user_joined_room', cb);
    }

    handleUserLeftRoom = (cb) => {
      this.io.on('user_left_room', cb);
    }

    handleUserUpdateAnchors = (cb) => {
      this.io.on('user_updated_anchors', cb);
    }

    createRoom = () => {
      const roomId = generateId();
      this.roomId = roomId;

      this.io.emit('create_room', { room_id: roomId });
    }
  }

  export const roomSocket = new RoomSocket();

```

A.15 Файл Sharedb.js

```

import sharedb from 'sharedb';

// ShareDB init //
const sharews = new WebSocket(ws://${window.location.host}:8080);
const shareconn = new sharedb.Connection(sharews);
// const sharedoc = shareconn.get('docs', 'default');

export {
  shareconn
};

```

ДОДАТОК Б
Слайди презентації

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЗАПОРІЗЬКА
ПОЛІТЕХНІКА
ПРЕЗЕНТАЦІЯ АТЕСТАЦІЙНОЇ РОБОТИ
МАГІСТРА НА ТЕМУ:**

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ЗАСОБІВ СПІЛЬНОГО РЕДАГУВАННЯ JAVASCRIPT КОДУ
В РЕАЛЬНОМУ ЧАСІ**

ВИКОНАВ СТУДЕНТ ГРУПИ КНТ-129М

СЕРГЄЄВ О.В.

КЕРІВНИК К.Т.Н., ДОЦЕНТ

ФЕДОРОНЧАК Т.В.

Рисунок Б.1 – Слайд 1

Мета роботи

2

створення вебзастосунку для спільного редагування програмного коду на мові програмування JS в реальному часі.

Задачі

- Аналіз предметної області та огляд існуючих аналогів
- Огляд технології операційного перетворення
- Проектування структури вебзастосунку
- Розробка серверної частини застосунку
- Розробка клієнтської частини застосунку та користувацького інтерфейсу
- Тестування вебзастосунку

Рисунок Б.2 – Слайд 2

Аналіз предметної області

3

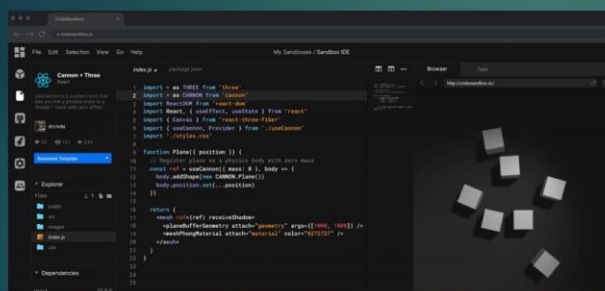
- **Спільна розробка** - це підхід до розробки програмного забезпечення, при якому двоє чи більше розробників об'єднують зусилля і працюють над одним і тим же плануючи та детально обговорюючи свою роботу для прийняття найкращих рішень
- **Редактор коду** - текстовий редактор для створення, та редагування коду програм. Він може бути окремим додатком, або інтегрованим в середовище розробки (IDE). Редактори коду мають деякі можливості, які спрощують та прискорюють написання та редагування коду, такі як підсвітка синтаксису, автодоповнення, перевірка правильності розстановки дужок, контекстна допомога по коду та багато інших.

Рисунок Б.3 – Слайд 3

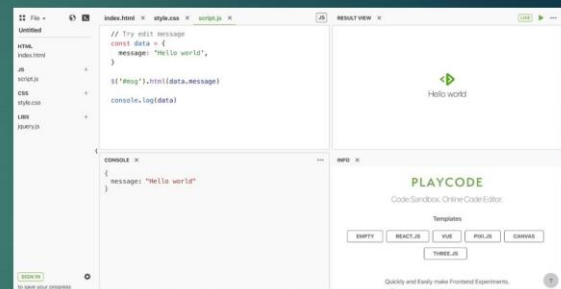
Огляд існуючих аналогів

4

Досліджено та виявлено два найкращих на даний момент рішення в якості редактора коду онлайн



Codesandbox



Playcode

Рисунок Б.4 – Слайд 4

Операційне перетворення

5

Основною ідеєю ОП є перетворення (або зміна) параметрів операцій редагування в злагоді з ефектами виконання в попередніх одночасних операціях так, щоб перетворена операція могла спрацювати коректно і не порушувати узгодженість в документі.

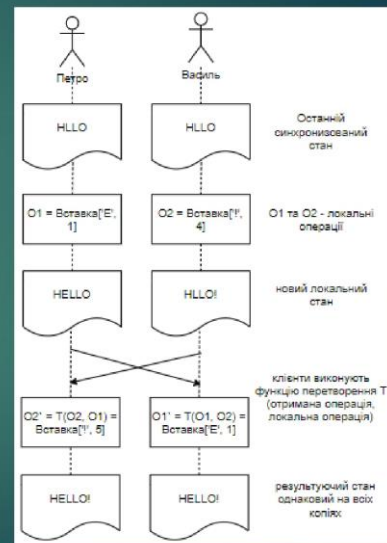


Рисунок Б.5 – Слайд 5

Операція редагування

6

Операція – одиниця інформації щодо зміни в документі.

Поділяється на дві операції: вставка та видалення.

Операція Вставка[«A», 100] інформує про те, що користувач вставив символ «A» на позицію 100 символів від початку.

Операція Видалення[10] інформує про те, що користувач видалив 10 символ від початку.

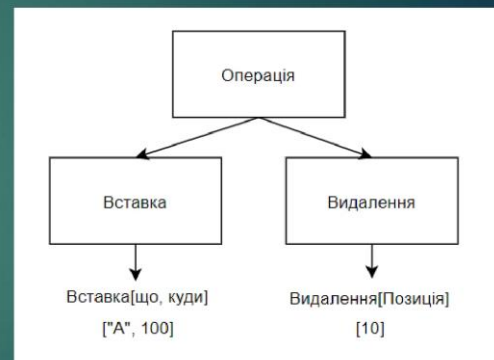
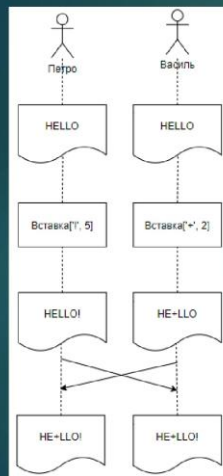


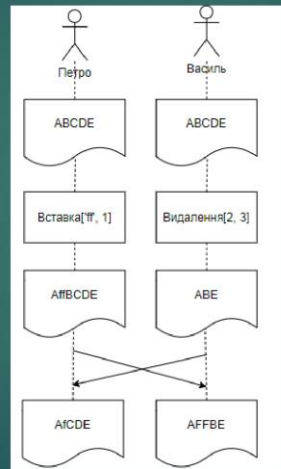
Рисунок Б.6 – Слайд 6

Модель узгодженості CCI

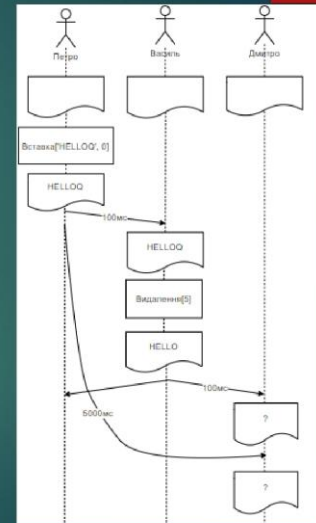
7



Збіжність



Збереження наміру



Збереження причинності

Рисунок Б.7 – Слайд 7

Схема структури вебзастосунку

8

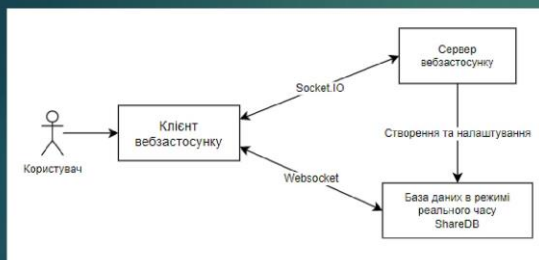


Схема вебзастосунку

- Користувач – реальний користувач створеного вебзастосунку, що використовує браузер
- Сервер – частина вебзастосунку, що відповідає за ініціалізацію ShareDB та організацію користувацьких сесій (кімнат) за допомогою Socket.IO
- Клієнт – частина вебзастосунку, безпосередньо з якою працює користувач
- ShareDB – база даних, що реалізує технологію ОП та надає можливість паралельного запису до документу в реальному часі

Рисунок Б.8 – Слайд 8

Вибір редактора тексту

9

Бібліотеки			
	Ace	CodeMirror	Monaco
Розробники	Cloud9 IDE	Marijn Haverbeke	Microsoft
Підтримка мов та підсвітка синтаксису	>120	>100	>20
Рендеренг	DOM	DOM	DOM та частково <canvas>
Розмір останніх версій	2.147 KB	1.411 KB	10.898 KB
Активність спільноти	Середня	Висока	Середня
Активність розробників	Середня	Середня	Висока

Рисунок Б.9 – Слайд 9

Функціонування вебзастосунку. Серверний модуль

10

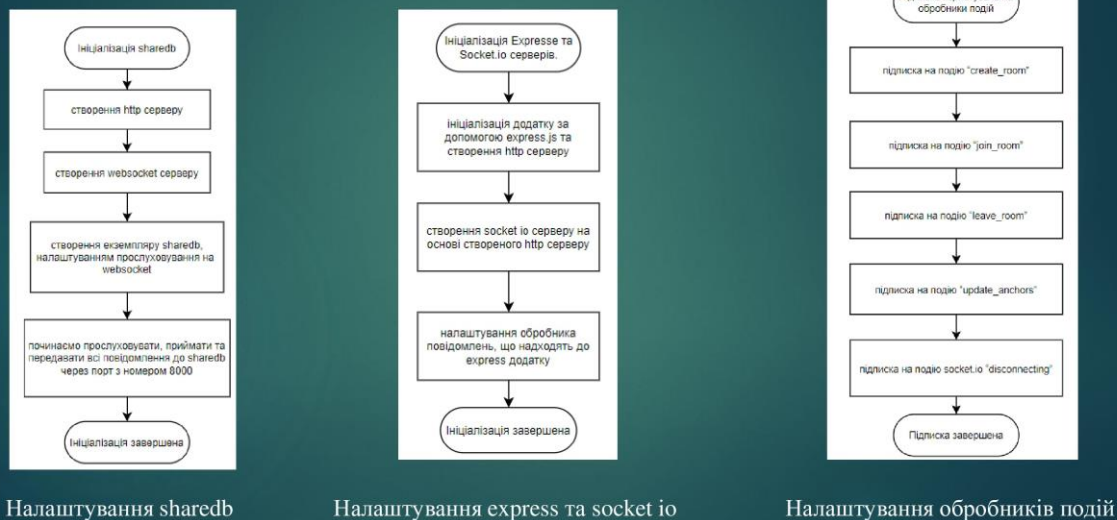
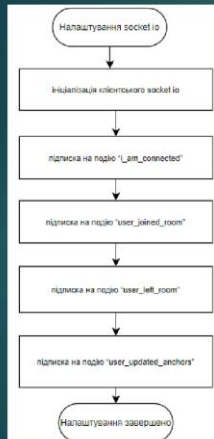


Рисунок Б.10 – Слайд 10

Функціонування вебзастосунку. Клієнтський модуль

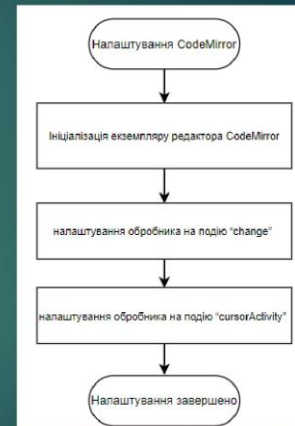
11



Ініціалізація клієнтського екземпляру socket.io



Ініціалізація клієнтського екземпляру sharedb

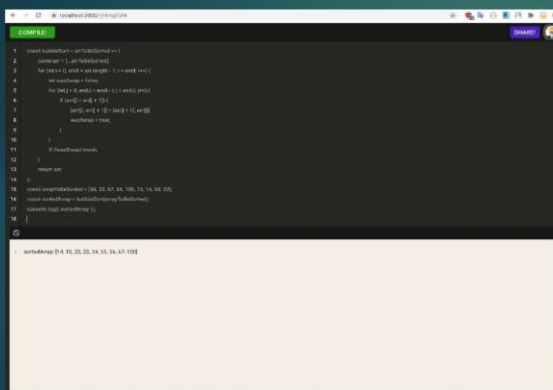


Ініціалізація та налаштування екземпляру CodeMirror

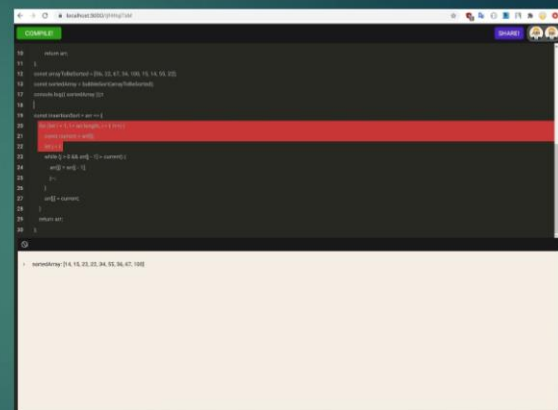
Рисунок Б.11 – Слайд 11

Розроблений графічний інтерфейс

12



Головна сторінка



Спільна робота

Рисунок Б.12 – Слайд 12

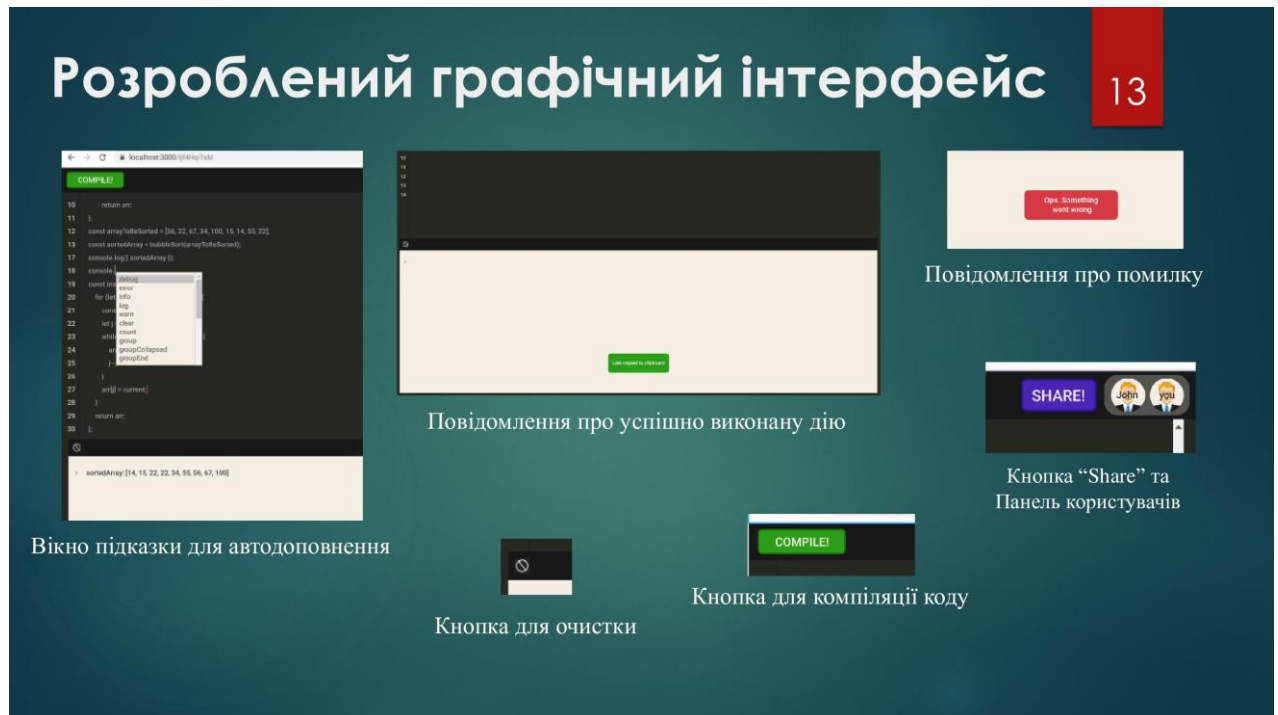


Рисунок Б.13 – Слайд 13

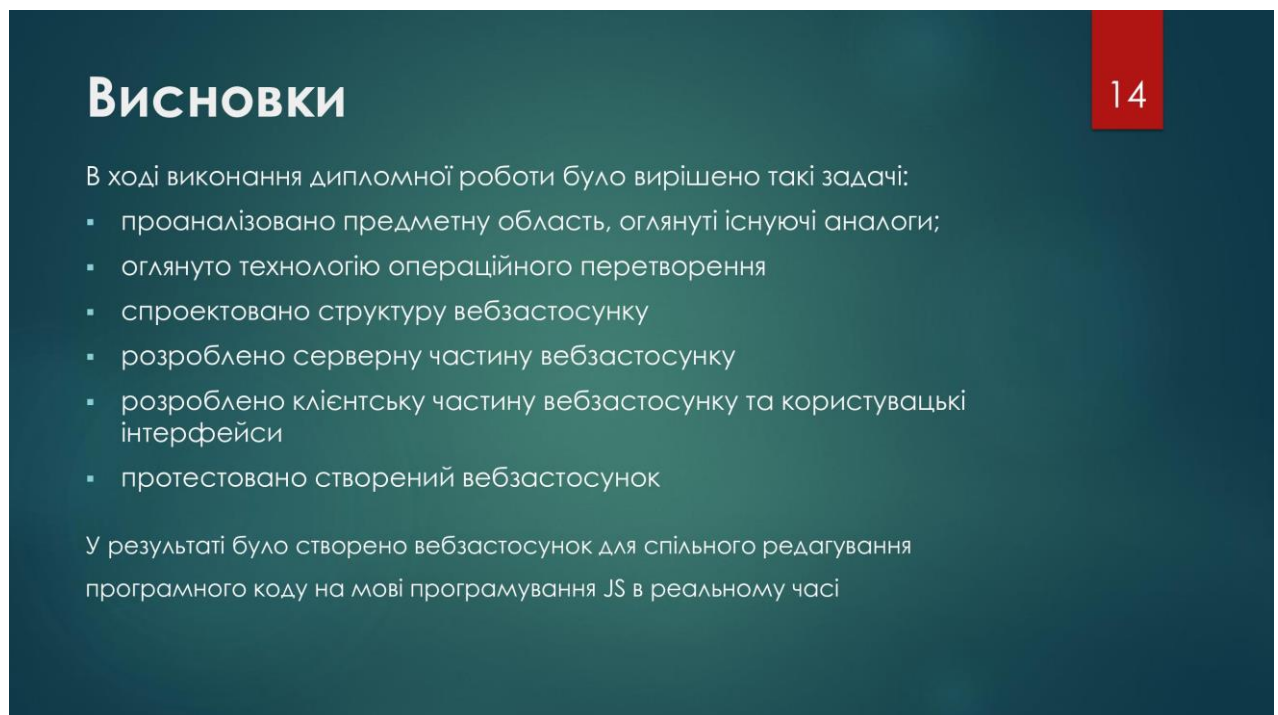


Рисунок Б.14 – Слайд 14