Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Методи створення колективного інформаційного середовища для організації процесів розробки програмного продукту

Виконав: студент ___4___ курсу
групи _1-ПІ-18б_____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Івасьов О.С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Коваленко О.О.___
(прізвище та ініціали)

Рецензент:.к.т.н.,доц.каф. КН Колесницький О.К.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Івасьову Олексію Станіславовичу

1. Тема роботи – «Методи створення колективного інформаційного середовища для організації процесів розробки програмного продукту»
Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66
2. Строк подання студентом роботи 13 травня 2022 р.
3. Вихідні дані до роботи: Середовище розробки – Visual Studio Code, Мова розробки – JavaScript, Операційна система – Windows 10.
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка структури та алгоритмів програмного продукту; розробка веб-додатку; тестування веб-додатку; висновки; список використаних джерел; додатки, графічна частина.
5. Перелік графічного матеріалу: демонстрація роботи аналогів, блок-схема функціонування програмного продукту, блок-схема взаємодії модулів програмного продукту, загальний алгоритм роботи додатку та тестування додатку.

6. Консультанти розділів бакалаврської дипломної роботи.

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О.О., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Термін виконання етапів проекту (роботи)	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.22 – 30.03.22	Вик.
2	Розробка архітектури та алгоритмів роботи системи	30.03.22 – 5.04.22	Вик.
3	Вибір середовища та мови розробки	5.04.22 – 10.04.22	Вик.
4	Розробка програмного продукту	11.04.22 – 21.04.22	Вик.
5	Тестування роботи системи	22.04.22 – 30.04.22	Вик.
6	Оформлення матеріалів до захисту БДР	1.05.22 – 10.06.22	Вик.

Студент _____ **Івасьов О.С.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Коваленко О.О.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 51 сторінок формату А4, на яких є 28 рисунків, 3 таблиці, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі проведено детальний аналіз існуючих колективних інформаційних середовищ. Проведено аналіз аналогів, визначено їх переваги та недоліки. На основі досліджень було прийняте рішення розробити власний застосунок з врахуванням недоліків аналогів Сформульовано мету досліджень – підвищення продуктивності працівників за рахунок модифікації існуючих технік програмування шляхом створення колективного інформаційного середовища для організації процесів розробки програмного продукту.

Розроблено методи створення колективного інформаційного середовища та програмна реалізація, яка дозволяє організувати процес розробки програмного продукту, що значно прискорює сам процес, робить його прозорим та зрозумілим, полегшує комунікацію між користувачами у процесі розробки.

Створений програмний продукт написаний на мові програмування JavaScript з використанням двох бібліотек з відкритим доступом: React та CodeMirror. Характеризується швидкістю та надійністю роботи, а також широким спектром можливих застосувань.

Отримані в бакалаврській дипломній роботі результати можна використати для створення колективного інформаційного середовища.

Ключові слова: колективне інформаційне середовище, метод віддаленого парного програмування.

ABSTRACT

The bachelor's thesis consists of 51 A4 pages, which have 28 figures, 3 tables, a list of sources used contains 21 titles.

A detailed analysis of the existing collective information environments was conducted in the bachelor's thesis. The analysis of analogues is carried out, their advantages and disadvantages are defined. Based on research, it was decided to develop its own application, taking into account the shortcomings of analogues. The purpose of research is to increase employee productivity by modifying existing programming techniques by creating a collective information environment for organizing software development processes.

Methods of creating a collective information environment and software implementation have been developed, which allows to organize the process of software product development, which significantly speeds up the process, makes it transparent and clear, facilitates communication between users in the development process.

The created software product is written in JavaScript programming language using two open access libraries: React and CodeMirror. It is characterized by speed and reliability, as well as a wide range of possible applications.

The results obtained in the bachelor's thesis can be used to create a collective information environment.

Key words: collective information environment, method of remote pair programming.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Не так давно браузери використовувалися лише для перегляду веб-сторінок, а сам Інтернет був лише поєднанням електронної пошти і новин. І ось сьогодні ми практично живемо в Інтернеті й оточені безліччю цифрових можливостей. Веб-технології просуваються та розвиваються, і з розвитком хмарних обчислень, ми спостерігаємо ще одну еру веб-додатків, які обладнані для виконання великої кількості завдань без обмежень обладнання.

Більшість веб-розробників і програмістів користуються лише автономними інструментами програмування. Однак розвиток сучасних веб-технологій робить процес програмування набагато гнучкішим, дає більше можливостей та забезпечує більш багатий досвід програмування [1]. Використання цих технологій для створення колективного інформаційного середовища відкрило б нові можливості для розробників, такі як праця у парі або командою. Така техніка розробки програмного забезпечення вже існує і називається парним програмуванням. Парне програмування [2] — це підхід до розробки програмного забезпечення, коли пари програмістів працюють разом над проектом, при чому один програміст займається безпосередньо написанням коду, а другий сидить поряд і спостерігає за результатом його праці по ходу даючи підказки та обговорюючи недоліки в коді, в тому числі помилки у реалізації, логіці, синтаксисі тощо. Через певний проміжок часу програмісти змінюють один одного і ролі змінюються.

Оскільки існуюча техніка парного програмування має певні недоліки, як наприклад, відсутність можливості дистанційної роботи, були створенні застосунки для вирішення цієї проблеми, але у більшості випадків вони мають платну підписку, створені у виді плагіну до певного редактору коду та потребують стороннього програмного забезпечення для роботи. Це робить їх досить вузькоспеціалізованими та не підходить для широкого кола людей, тому

актуальним є питання створення програмного забезпечення з метою вирішення виявлених недоліків.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета і завдання дослідження. Метою роботи є підвищення продуктивності працівників за рахунок модифікації існуючих технік програмування шляхом створення колективного інформаційного середовища для організації процесів розробки програмного продукту.

У відповідності до поставленої мети потрібно виконати наступні завдання:

- аналіз стану задачі;
- порівняльний аналіз аналогів;
- аналіз методів розв'язання поставленої задачі;
- розробка структури інтерфейсу програмного додатку;
- розробка алгоритмів роботи додатку;
-варіантний аналіз і обґрунтування вибору засобів для реалізації програмного застосунку;
- розробка модулю підключення користувачів;
- розробка модулю редактору коду;
- тестування програми та інструкції користувача

Об'єкт дослідження – процес колективної розробки програмного продукту.

Предмет дослідження – методи та програмні засоби колективної розробки програмного продукту.

Методи дослідження. У роботі використовувалися: теорія ймовірностей для створення унікального ідентифікатору кожної кімнати, методи побудови веб-додатків.

Наукова новизна отриманих результатів. Подальшого розвитку отримав метод створення колективного інформаційного середовища у вигляді веб-додатку, у якому, на відміну від існуючих реалізацій, які використовують обмін інформацією через технологію WebRTC та протокол HTTP, використовується

протокол WebSocket, що дозволило не тільки знизити навантаження на сервер, але й підвищити швидкість та оперативність.

Практичне значення одержаних результатів. Практична цінність одержаних результатів полягає у можливості практичного використання розробленого спеціалізованого програмного застосунку колективного інформаційного середовища.

Особистий внесок. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, автору належать: розробка веб-додатку редактору коду.

Апробація матеріалів роботи. Матеріали роботи доповідалися та обговорювалися на науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії ВНТУ (Вінниця 2022).

Публікації. Основні результати дослідження опубліковані в науковій роботі – тезах-доповіді на конференції факультету інформаційних технологій та комп’ютерної інженерії ВНТУ[1].

1 АНАЛІЗ ІСНУЮЧИХ КОЛЕКТИВНИХ ІНФОРМАЦІЙНИХ СЕРЕДОВИЩ ТА ЇХ ПРОГРАМНИХ РЕАЛІЗАЦІЙ

1.1 Аналіз стану проблеми

У більшості випадків співпраця над розробкою програмного забезпечення означає, що кожен програміст процює над своєю частиною проєкту, вкінці збираючи проєкт докупи. При такому підході часто використовують інструменти контролю версій як Git. Система контролю версій відстежує кожну зміну, внесену в проєкт. Кожен програміст працює над своїм власним фрагментом і завантажує свої зміни. Якщо внесені зміни суперечать іншим змінам, вони зобов’язані виправити це та зробити код працюючим, не порушуючи код, який був завантажений до них. Така техніка розробки програмного забезпечення найбільш вживана, проте є і інші техніки. Наприклад, парне програмування та моб-програмування.

Парне програмування передбачає роботу двох розробників за одним комп'ютером. Один розробник виконує роль водія, а інший – навігатора [3].

Водій використовує робочу станцію і пише код, а навігатор переглядає код в режимі реального часу. Час від часу обидва розробники міняються ролями, тож кожен має шанс керувати напрямком проекту та втілювати рішення в реальний код.

Загалом, робота навігатора полягає в тому, щоб визначити стратегічний напрямок коду та запропонувати, що він повинен робити. Водій, з іншого боку, відповідає за «тактичні» аспекти і зосереджується на написанні коду та слідкуванню інструкціям компанії щодо іменування змінних, коментування тощо.

Проєкт парного програмування може складатися з двох розробників із подібними навичками та досвідом, або він може об’єднати старшого розробника, який наставляє молодшого розробника під час проекту.

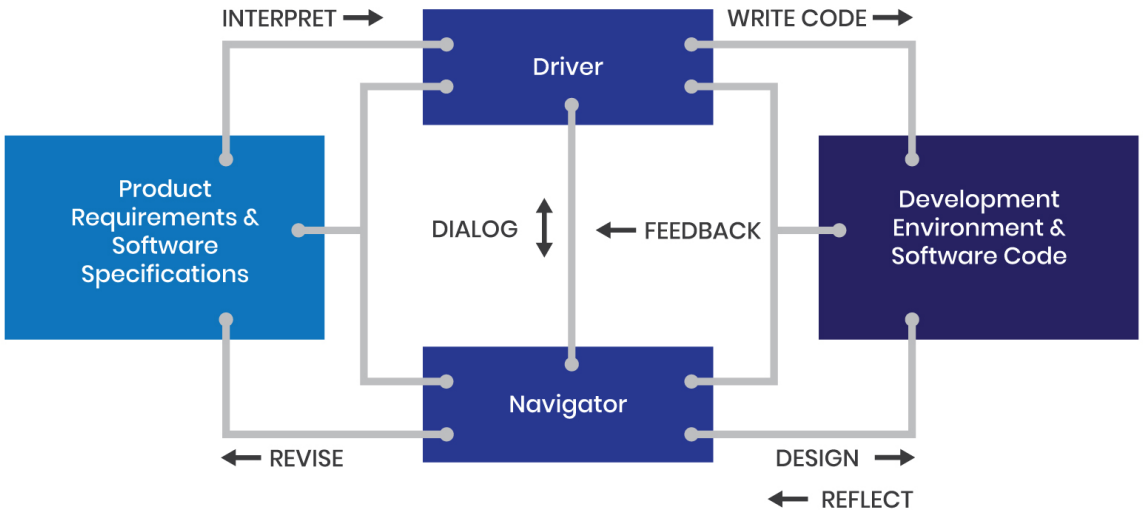


Рисунок 1.1 – Схема роботи згідно техніки парного програмування [3]

Ідея моб-програмування схожа на парне програмування, за винятком того, що в ньому беруть участь більше двох людей. У будь-який момент часу один розробник перебуває за робочою станцією, тоді як інші програмісти-навігатори направляють його [4].

Така техніка заснована на принципах бережливого виробництва, екстремального програмування та бережливого розробки програмного забезпечення. Ранній термін використання фрази «mob programming» був зроблений у Extreme Programming Perspectives [5].

Хоча це звучить дещо хаотично, це чудовий спосіб об’єднати команду розробників для створення кращого програмного продукту. Звісно якщо команда дотримується кількох правил:

По-перше, вони повинні дотримуватися принципу «водій-навігатор». Щоб ідея розробника потрапила на комп’ютер, вона повинна пройти через руки іншого розробника. Це означає, що навігатори повинні передавати свої ідеї достатньо чітко, щоб водій міг перетворити їх у код.

У той же час водіям пропонується задавати уточнюючі питання, включаючи логіку, яка лежить в основі певної ідеї. Але ні в якому разі водій не повинен починати писати код, який не відповідає вказівкам навігатора.

По-друге, моб-програмування працює краще, коли кожен має можливість

стати навігатором. Успішні проєкти моб-програмування завжди включають в себе зміну ролей, часто використовуючи сувору систему таймера.

Порівнюючи різні техніки розробки програмного забезпечення здається, що спільне кодування сповільнює роботу, але є чимало переваг, які роблять його вигідним підходом:

Краще спілкування. Коли кожен працює над власним кодом для проєкту розробки, погана комунікація може призвести до збою. З іншого боку, для спільного кодування потрібна відмінна комунікація. Під час проекту спільного програмування кожен чує, що говорять інші. І оскільки будь-які ідеї повинні бути втіленні у виді коду кимось іншим, кожен повинен чітко пояснити, як він хоче підійти до проєкту. Спільне кодування також змушує команду відкрито повідомляти про свої розбіжності та залагоджувати їх до того, як проєкт зможе рухатися далі. В результаті виходить кінцевий продукт, який задовільняє кожного.

Швидше налагодження. Коли всі дивляться на той самий код одночасно, виявити помилки набагато легше і швидше. Незалежно від того, чи то кома, чи то змінна з помилкою, відловлення помилок на етапі написання коду, заощаджує час усієї команди, коли потім потрібно буде просіяти набагато більше коду [6].

1.2 Порівняльний аналіз аналогів

Зазвичай колективні інформаційні середовища для розробки програмного продукту існують у вигляді веб-додатку, програми для ПК під відповідну операційну систему та розширення для існуючого редактору коду. Розглянемо найбільш поширені середовища у цій галузі.

Teletype for Atom (рисунок 1.2) [7] – розширення для редактору коду Atom, один з перших серед інструментів для спільної роботи з кодом у реальному часі, дозволяє користувачам Atom ділитися своїм робочим простором з членами команди. Teletype вводить концепцію «порталів» у режимі реального часу для спільного використання робочих просторів. Коли хост відкриває портал, його активна вкладка стає спільною робочою областю. Там запрошені співавтори

можуть приєднатися та вносити зміни в режимі реального часу. Коли хост переміщається між файлами, співавтори автоматично слідують за активною вкладкою.

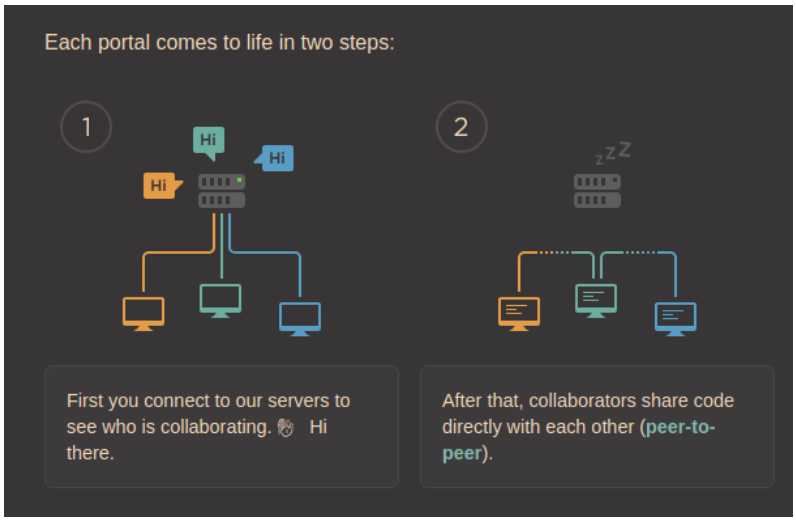


Рисунок 1.2 – Схема роботи додатку Teletype [7]

CodePen (рисунок 1.3) [8] — це соціальне середовище розробки для фронт-енд дизайнерів і розробників з редагуванням та зберіганням HTML, CSS та JavaScript з переглядом готового результату у браузері. Отриманим кодом можна розділитися і видозмінити його за необхідності. Вікно браузера ділиться на кілька робочих областей, які відображають результат, а також код HTML, CSS та JavaScript.

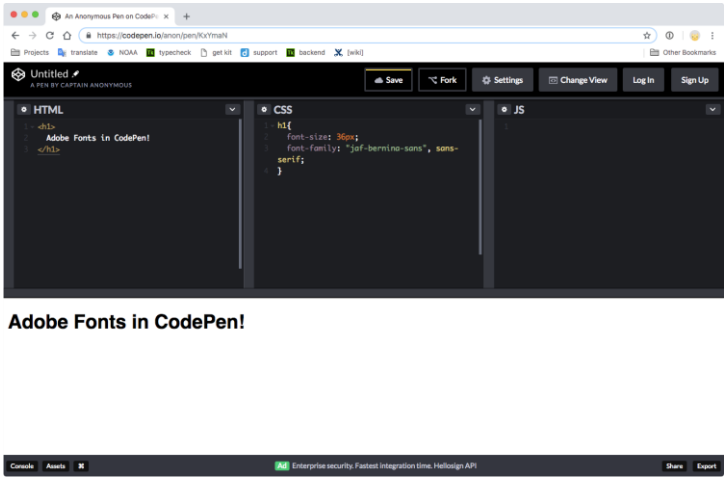


Рисунок 1.3 – Інтерфейс веб-додатку CodePen

AWS Cloud9 (рисунок 1.4) [9] — продукт стороннього розробника, придбаний Amazon, забезпечує середовище розробки у браузері, яке підтримує близько 40 мов програмування з різними рівнями інструментарію. Декілька розробників можуть співпрацювати над одним і тим же проектом, розміщеним у хмарі, в режимі реального часу, використовуючи спільне середовище. Користувач може спостерігати, як друкує його колега з візуальними підказками, які вказують, хто який рядок коду написав і спілкуватися на панелі в IDE. Адміністратори проекту можуть надавати співавторам привілеї читання/запису чи права лише на читання.

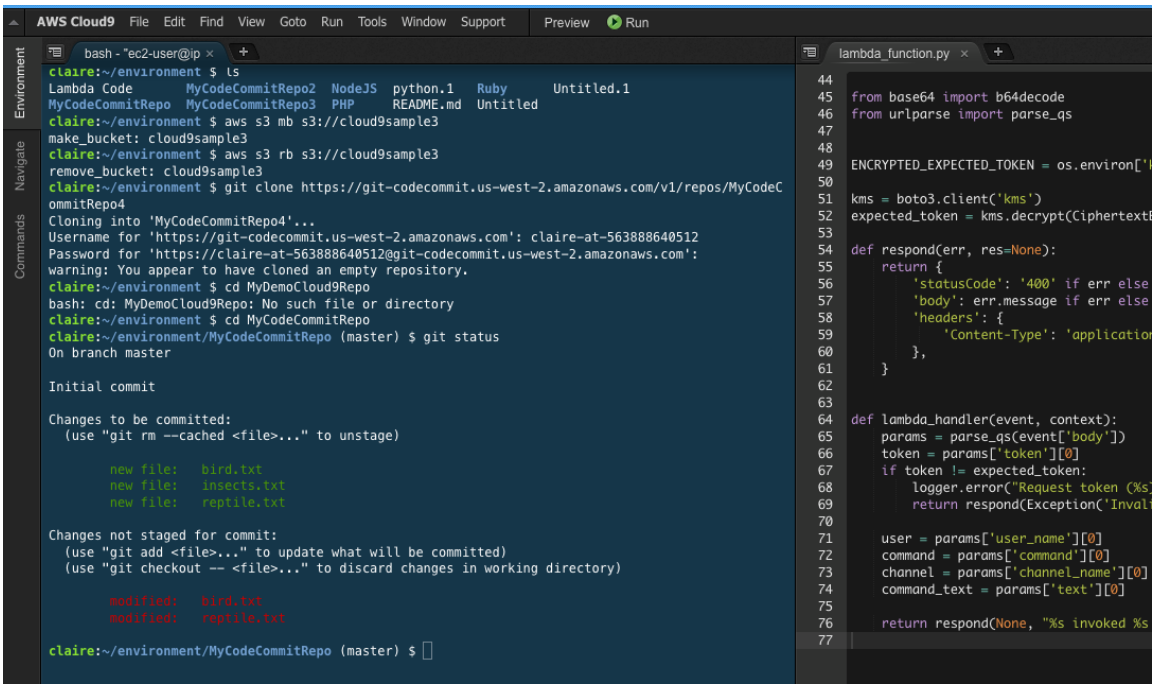


Рисунок 1.4 – Інтерфейс веб-додатку AWS Cloud9

Remote Collab for Sublime Text (рисунок 1.5) — додаток до редактора коду Sublime Text. Кожна сесія прив’язана до певного документа. Після того, як хост розпочав сеанс, співробітникам просто потрібна IP-адреса хоста, щоб приєднатися, і будь-які внесені зміни відображатимуться на всіх машинах.

Проаналізувавши усі аналоги, було визначено їх сильні та слабкі місця та створено порівняльну таблицю зі власним додатком (табл. 1.1).

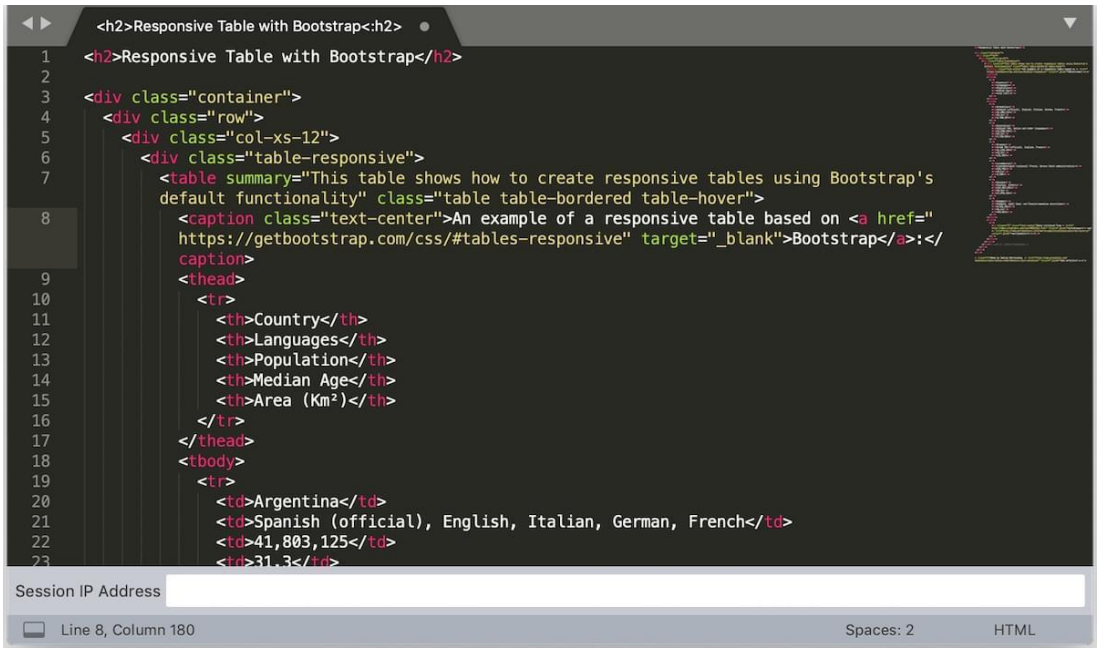


Рисунок 1.5 – Інтерфейс плагіну Remote Collab

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Teletype	Codepen	AWS Cloud9	Remote Collab	Власний додаток
Безкоштовне використання	+	-	-	+	+
Відсутня необхідність завантаження стороннього ПЗ	-	+	-	-	+
Необмежена кількість працівників над одним проєктом	-	+	-	-	+
Відсутність використання додаткових апаратних ресурсів	-	+	-	-	+
Відсутність реєєстрації	-	-	-	-	+
Підсумковий результат	1	3	0	1	5

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує кращу ефективність.

1.3 Методи реалізації програмних модулів колективного інформаційного середовища

Існує декілька варіантів розв’язання задачі з створення колективного інформаційного середовища для організації процесів створення програмного продукту: у вигляді розширення або плагіну для існуючого редактору коду, у вигляді самостійного додатку для ПК та у вигляді веб-додатку.

Варіант з плагіном для існуючого редактору коду досить обмежує потенціальне коло користувачів через те, що розробляється лише для одного конкретного редактору. Також від сильно залежить архітектури та коду редактору до якого прив’язаний, це робить складнощі с його підтримкою, адже з оновленням редактору необхідно оновлювати плагін.

Додаток для ПК є складнішим для розробки в порівнянні з іншими варіантами, адже необхідно враховувати операційну систему користувача та його апаратні ресурси.

Веб-додаток є простим в реалізації, доступним для більшості користувачів, та не змушує користувача завантажувати стороннє програмне забезпечення, тому для розробки додатку було обрано саме такий метод реалізації.

Метод реалізації з’єднання та передачі інформації між користувачами у веб-додатку можливо створити за допомогою декількох засобів: HTTP, WebRTC та WebSocket.

HTTP є протоколом запиту/відповіді в обчислювальній моделі клієнт-сервер і основним способом зв'язку у всесвітній мережі. Початкова версія, запропонована як протокол програми в 1989 році Тімом Бернерсом-Лі, була дуже обмеженою і швидко модифікована для підтримки ширшої функціональності браузера і сервера [10].

HTTP polling представляло собою крок уперед від класичного механізму запиту/відповіді – і хоча існують різні версії опитування, лише довге опитування (long polling) є доречним у застосуванні при розробці застосунку, що працює у реальному часі.

Наприклад, коротке опитування HTTP використовує таймер на основі AJAX, щоб гарантувати, що клієнтські пристрої надсилають запити серверу через фіксовані інтервали. Однак сервер все одно негайно відповідатиме на кожен запит, надаючи нові дані або надсилаючи «порожню» відповідь, якщо нових даних немає, перш ніж закрити з'єднання. Таким чином, це насправді не дуже корисно в програмах реального часу, коли клієнту потрібно знати про нові дані, як тільки вони з'являться.

Саме це обмеження призвело до розвитку тривалого опитування HTTP. Тривале опитування (long polling) — це техніка, коли сервер вирішує утримувати з'єднання клієнта відкритим якомога довше (зазвичай до 20 секунд), надаючи відповідь лише після того, як дані стають доступні або досягається поріг часу очікування [10]. Роботу техніки схематично зображено на рисунку 1.5.

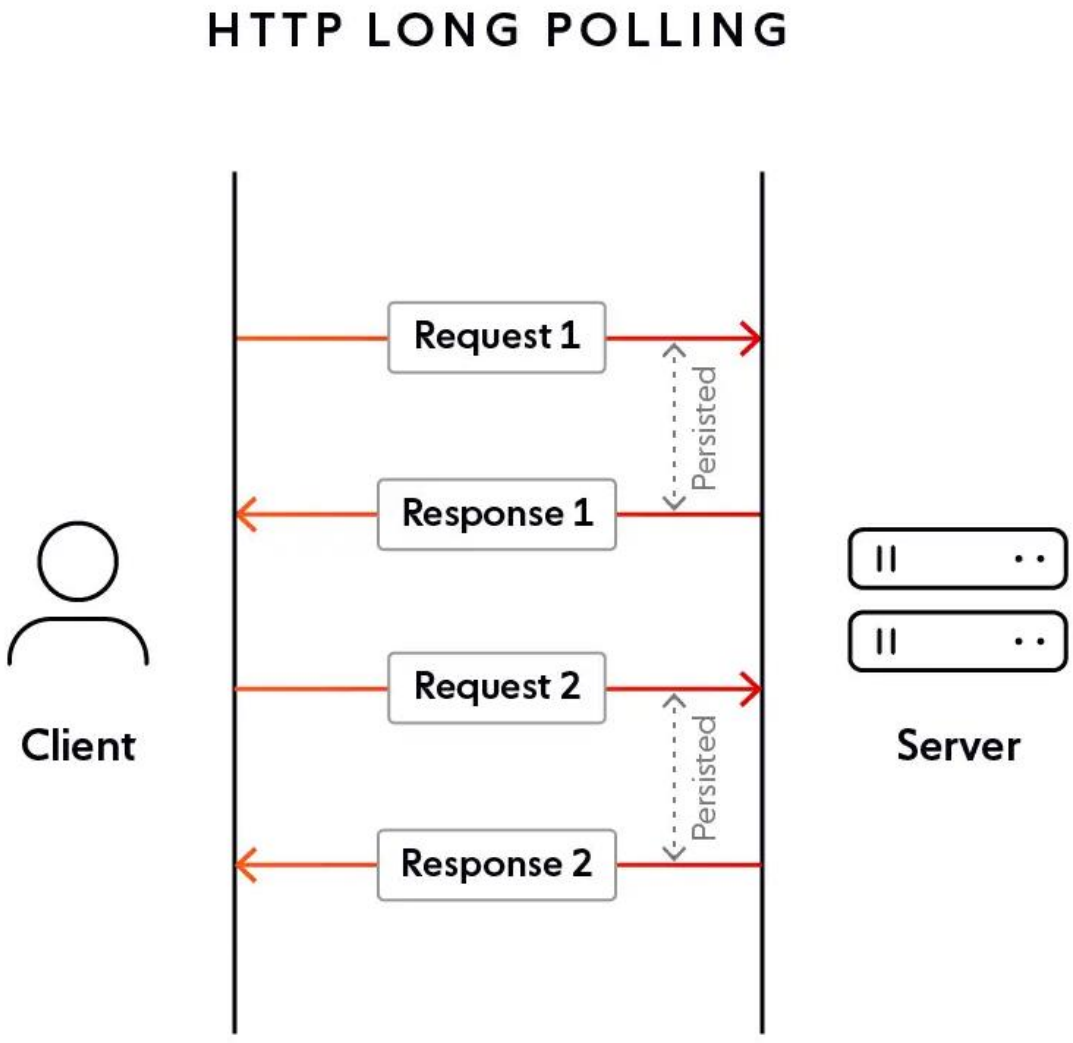


Рисунок 1.6 – Принцип роботи техніки long polling

Основна перевага тривалого опитування полягає в тому, що нова інформація, теоретично, надсилається клієнту, як тільки вона стає доступною. Недоліком, однак, є складнощі, пов'язані з обробкою HTTP-запитів, що може створити масу проблем у масштабі.

Тобто з HTTP ви повинні постійно запитувати оновлення (і отримувати відповідь), що дуже затратно по ресурсам: клієнт встановлює з'єднання, запитує оновлення, отримує відповідь від сервера, а потім закриває з'єднання. Уявіть, що цей процес повторюється нескінченно тисячами одночасних користувачів – це неймовірно важко для сервера в масштабі.

Ще однією технікою, яку в теорії можна використати для побудови застосунку, що працює в режимі реального часу це потокова передача HTTP.

Потокова передача HTTP – це метод передачі даних, який дозволяє веб-серверу безперервно надсилати дані клієнту через одне з'єднання HTTP, яке залишається відкритим на невизначений термін. По суті, клієнт робить HTTP-запит, а сервер надсилає відповідь невизначеної довжини.

Проте, хоча потокова передача HTTP є ефективною, легкою у використанні та може бути альтернативою WebSockets, вона має обмеження. Основна проблема з точки зору реального часу полягає в тому, що посередник може перервати з'єднання – чи то через тайм-аут, чи просто тому, що він обслуговує кілька запитів одночасно, тому не завжди можна гарантувати роботу у режиме реального часу.

Усі ці технології та підходи – мають одну спільну рису: вони прагнуть створити ілюзію обміну/спілкування даними у реальному часі (керованого подіями), щоб, коли сервер мав нові дані, він надсилав відповідь.

Незважаючи на те, що HTTP не є протоколом, який керується подіями, тож не є справді реальним часом, ці підходи насправді досить добре працюють у конкретних випадках використання, наприклад, у чаті Gmail. Однак проблеми виникають у програмах із низькою затримкою або в масштабі, головним чином через вимоги обробки, пов'язані з HTTP.

WebRTC — це специфікація HTML5 з відкритим вихідним кодом, випущена в 2011 році. Вона дозволяє веб-застосункам захоплювати та передавати аудіо та/або відео медіа, а також обмінюватися довільними даними між браузерами без потреби посередника. Проект з відкритим кодом під назвою Web Real-Time Communications передає голос, текст і відео в режимі реального часу в браузері. Це дозволяє P2P зв'язок між браузерами без будь-якого додаткового програмного забезпечення.

Традиційна веб-архітектура заснована на парадигмі клієнт-сервер, коли клієнт надсилає запит HTTP на сервер і отримує відповідь, що містить запитану інформацію. На відміну від цього, WebRTC дозволяє обмінюватися даними між N одноранговими мережами. У цьому обміні вони спілкуються один з одним без сервера посередині. [11]. Робота технології WebRTC зображена на рисунку 1.6

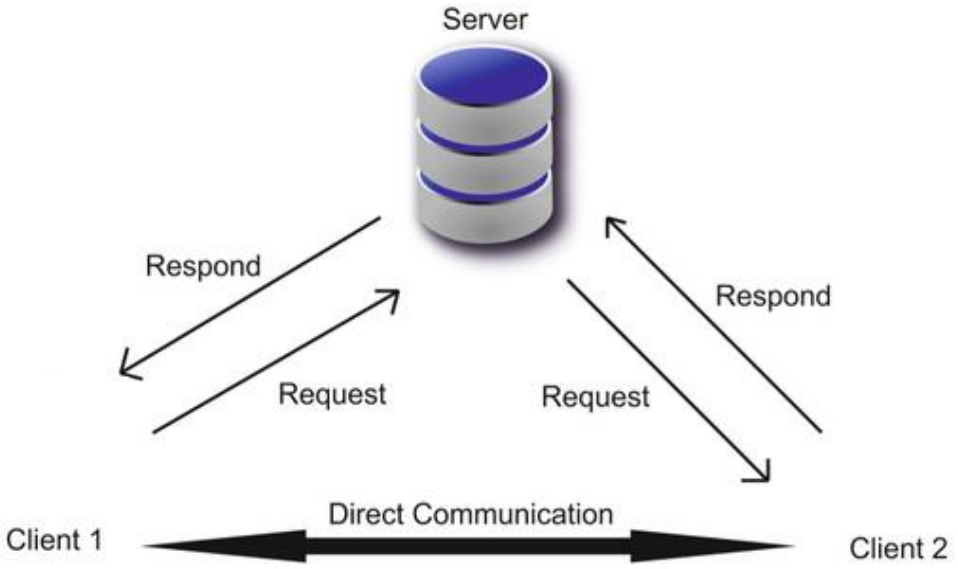


Рисунок 1.7 – Робота технології WebRTC

Веб-сокети це просунута технологія, що дозволяє відкрити постійне двонаправлене мережне з'єднання між браузером користувача та сервером. WebSocket — це протокол, керований подіями, що означає, що його можна використовувати для справжнього спілкування в реальному часі. На відміну від HTTP, де необхідно постійно запитувати оновлення, з веб-сокетами оновлення

надсилаються негайно, коли вони доступні [12]. WebSocket підтримує єдине постійне з'єднання відкритим, усуваючи проблеми із затримкою, які виникають із методами на основі запитів/відповідей HTTP. Робота протоколу WebSocket представлена на рисунку 1.7.

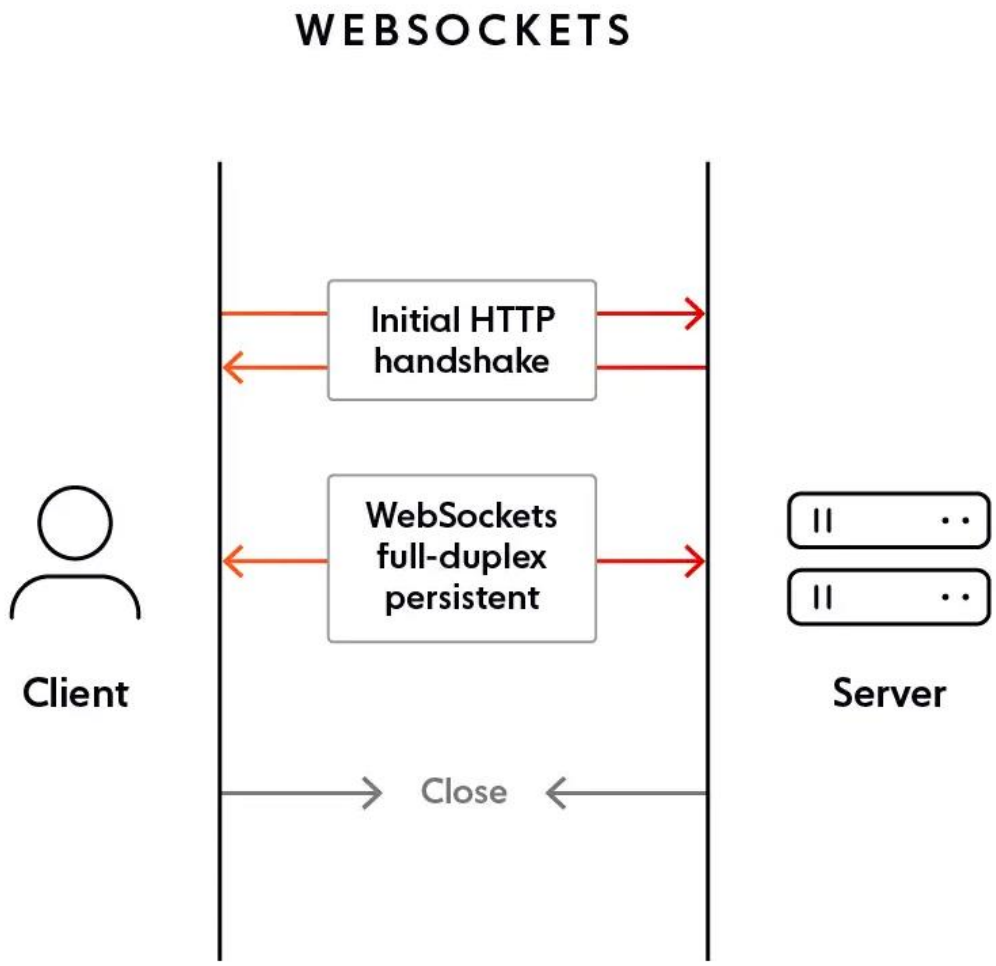


Рисунок 1.8 – Робота протоколу WebSocket

WebSocket зазвичай не використовує XMLHttpRequest, і тому заголовки не надсилаються щоразу, коли нам потрібно отримати більше інформації від сервера. Це, у свою чергу, зменшує навантаження даних, що надсилаються на сервер. Це протокол із збереженням стану, що означає, що з'єднання між клієнтом і сервером буде існувати, поки його не розірве будь-яка сторона (клієнт або сервер). Після закриття з’єднання клієнтом і сервером з’єднання припиняється з обох сторін [13]. WebSocket розроблено для втілення у веб-браузерах та веб-серверах, але він може бути використаний для будь-якої клієнтської або серверної програми.

В даний час W3C здійснюється стандартизація API Web Sockets. На даний момент WebRTC підтримується лише деякими браузерами, тоді як WebSocket сумісний майже з усіма існуючими браузерами. Також WebSocket має можливість асинхронної роботи, замість класичної роботи у форматі запит-відповідь. З огляду на це, для реалізації поставленого завдання був використан протокол WebSocket.

1.4 Постановка задач для програмної реалізації колективного інформаційного середовища

Проаналізувавши питання розробки інформаційного середовища для організації процесу створення програмного продукту, було визначено завдання, які необхідно виконати для розробки програмного додатку:

- аналіз існуючих рішень;
- виконати порівняльний аналіз аналогів;
- визначити основні завдання які необхідно вирішити для розробки колективного інформаційного середовища;
- виконати проектування колективного інформаційного середовища;
- вибір програмних засобів для вирішення поставлених завдань;
- розробка та тестування програмного продукту.

1.5 Висновки

У першому розділі було розглянуто стан питання створення колективного інформаційного середовища на теперішній час.

Проаналізувавши аналоги розроблюваного продукту, було створено порівняльну таблицю аналогів. У результаті порівняння було доведено доцільність розробки власного програмного додатку.

Проведено аналіз методів розв’язання поставленої задачі. У результаті аналізу було обрано метод реалізації з’днання користувачів за допомогою протоколу WebSocket.

Сформульовано основні завдання, які необхідно виконати для розробки програмного додатку.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1. Аналіз даних

Так як розроблюваний додаток є редактором коду, на вході він отримує велику кількість даних. Користувачу необхідно ввести унікальний ідентифікатор кімнату, до якої він збирається підключитись, який складається з чисел та строкових символів та разом зі ім’ям користувача вводиться у строковому вигляді при запуску додатку. Велику кількість даних користувач вводить коли займається написанням коду у додатку. Всього додаток має 3 текстових поля для коду, під HTML, CSS, та JavaScript відповідно. Технічно користувач вводить у текстові поля текст, що має тип string, але при цьому додаток аналізує текст коли вводиться символ у текстове полу та по різному інтерпретує його відповідно до коду, який користувач обрав. Інтерпретація полягає у різній підсвітці коду, фігурними дужками та HTML-тегами, що автоматично закриваються.

Результат роботи користувачі можуть побачити у фреймі, що знаходиться на головній сторінці додатку та який спеціальною кнопкою можна відкрити у новій вкладці браузера. Фрейм – це окремий HTML-документ, який «вбудовано» у інший HTML-документ, при чому кількість таких фреймів не обмежена. Коли користувач вводить будь-який символ в текстове поле, додаток аналізує символ та додає його в DOM-дерево HTML-документа у фреймі. Фрейм оновлюється з кожною зміною в коді, окрім JavaScript коду, який запускається коли користувач натискає відповідну кнопку, тому користувач може бачити зміни в розроблюваному додатку в режимі реального часу. За допомогою протоколу WebSocket додаток «прослуховує» спеціальні події, і коли один користувач щось змінює, WebSocket передає ці зміни його колегам та оновлює їх стан додатку.

2.2 Розробка структури інтерфейсу програмного додатку

Інтерфейс програмного додатку поділяється на дві сторінки: сторінку, де користувач обирає кімнату, та головну сторінку де можна писати код.

Користувача зустрічає форма, де є 2 поля для вводу інформації. Перше поле для вводу унікального ідентифікатору кімнати, а друге для імені користувача. Кнопка під полями робить перехід до наступної сторінки за умови, якщо користувач правильно ввів інформацію. У самому низу форми є кнопка, що має нетиповий вигляд, яка відповідає за генерацію ідентифікатору кімнати, це потрібно, якщо користувач не має ідентифікатору і хоче створити нову кімнату.

Інтерфейс першої сторінки додатку зображено на рисунку 2.1.

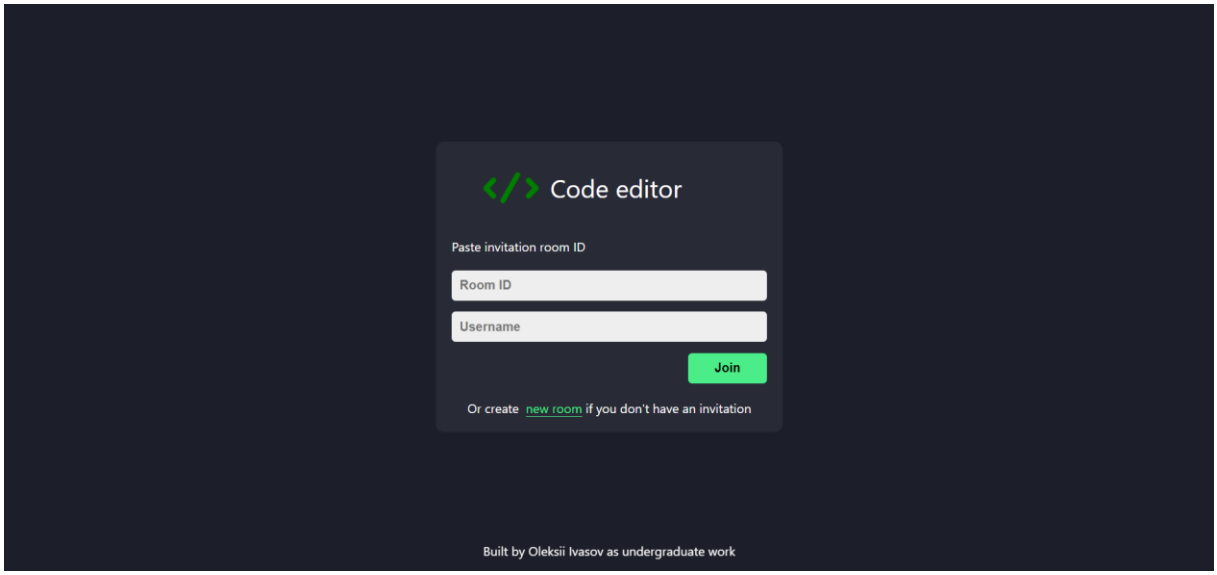


Рисунок 2.1 – Інтерфейс першої сторінки

Також була створена графічна схема вікна логіну (рисунок 2.2).

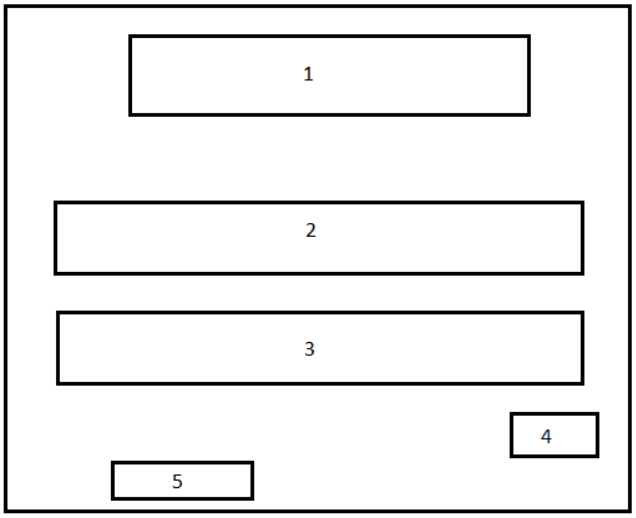


Рисунок 2.2 – Графічний схема інтерфейсу вікна логіну

Елементи вікна логіну:

1. Логотип додатку;
2. Поле для вводу ідентифікатору;
3. Поле для вводу імені користувача;
4. Кнопка входу до кімнати;
5. Кнопка генерації ідентифікатору.

Графічна схема головного вікна зображена на рисунку 2.3

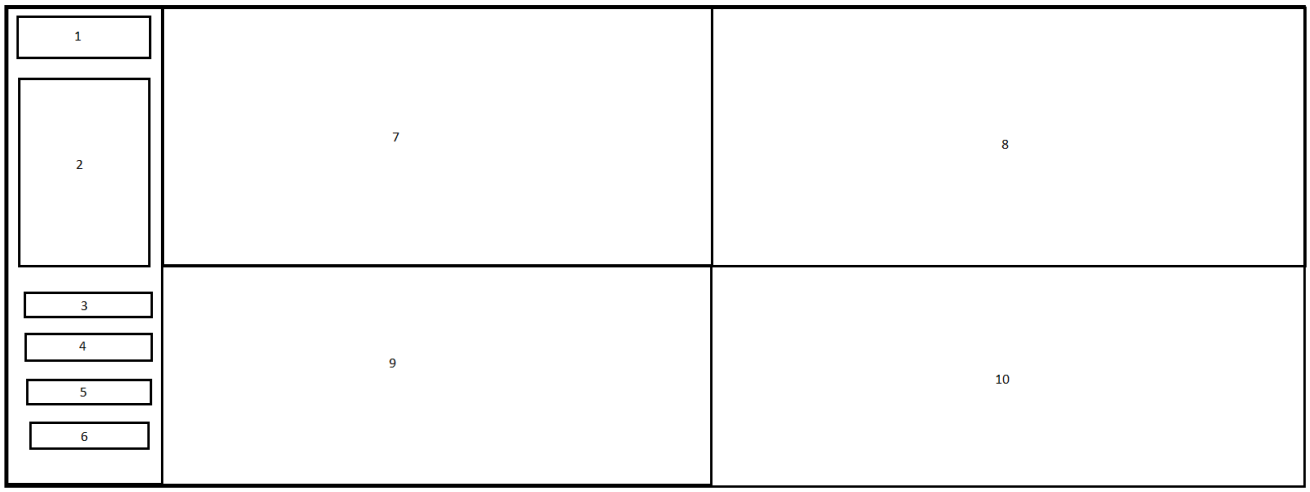


Рисунок 2.3 – Графічний схема головного вікна

Головне вікно умовно поділене на дві частини: бокове меню та текстові поля з фреймом. Бокове меню складається зі списку приєднаних користувачів та кнопок для керування додатком, а саме:

1. Логотип застосунку;
2. Список під’єднаних користувачів;
3. Open output in a new window – відкриває фрейм, у якому відображається розроблюваний продукт у новій вкладці браузера;
4. Run JS – запускає код, який користувач написав у полі для JavaScript;
5. Copy room ID – копіює ідентифікатор кімнати користувача та вставляє його у буфер обміну;
6. Leave – користувач покидає кімнату та повертається до першої сторінки;
7. Редактор JavaScript коду;

8. Редактор HTML;
9. Редактор CSS;
10. Фрейм у якому відображається результат роботи користувача.

Друга половина сторінки має в собі 3 текстових поля для HTML, CSS та JavaScript коду відповідно. Програма компілюється кожен час, коли користувач змінює код, окрім JavaScript коду щоб уникнути помилок, і виводиться у фрейм в правій нижній частині екрану.

Розробка графічного інтерфейсу додатку була проведена у середовищі розробки Microsoft Visual Code з використанням бібліотеки React та стилів таблиць CSS.

Інтерфейс другої сторінки додатку зображено на рисунку 2.2.

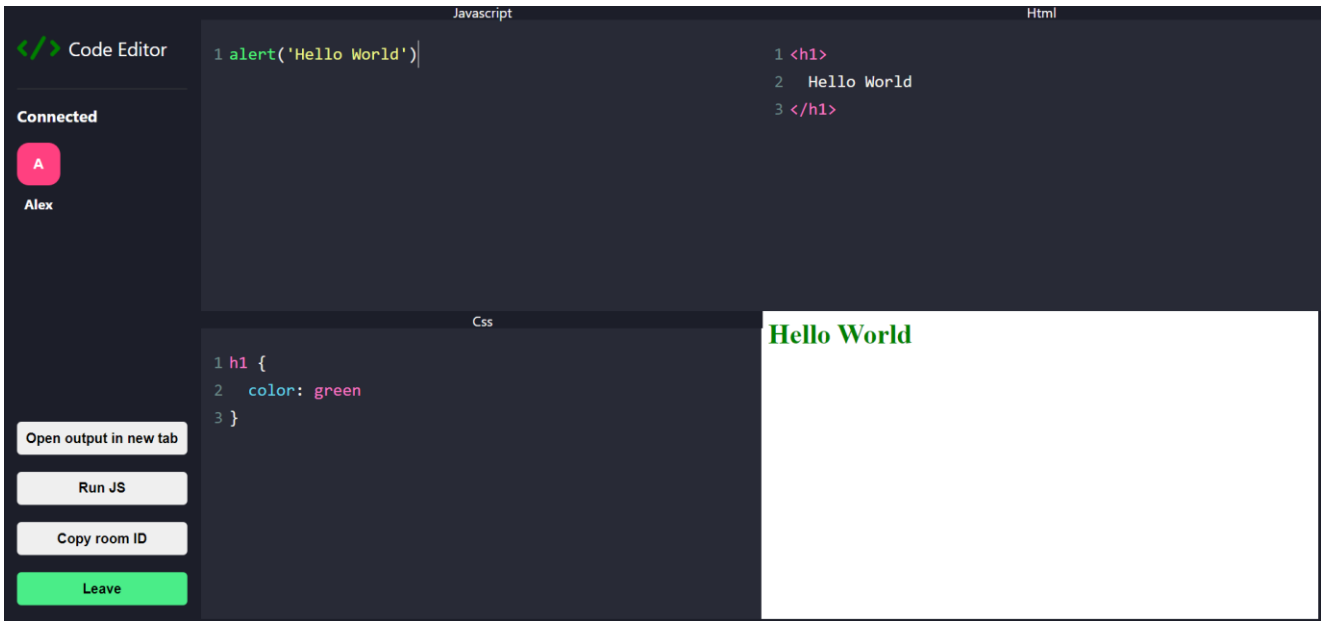


Рисунок 2.2 – Інтерфейс другої сторінки

Розроблено структурні схеми додатку. Структура схема застосунку представлена на рисунку 2.3

Рисунок 2.3 – Структурна схема застосунку

На рисунку 2.4 зображена структурна схема вікна логіна.

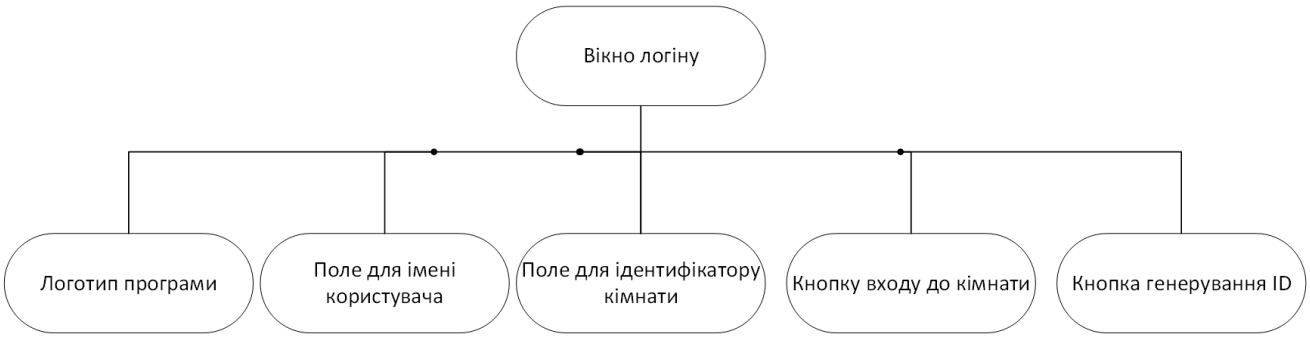


Рисунок 2.4 – Структурна схема вікна логіна

На рисунку 2.5 представлено структурну схему програми

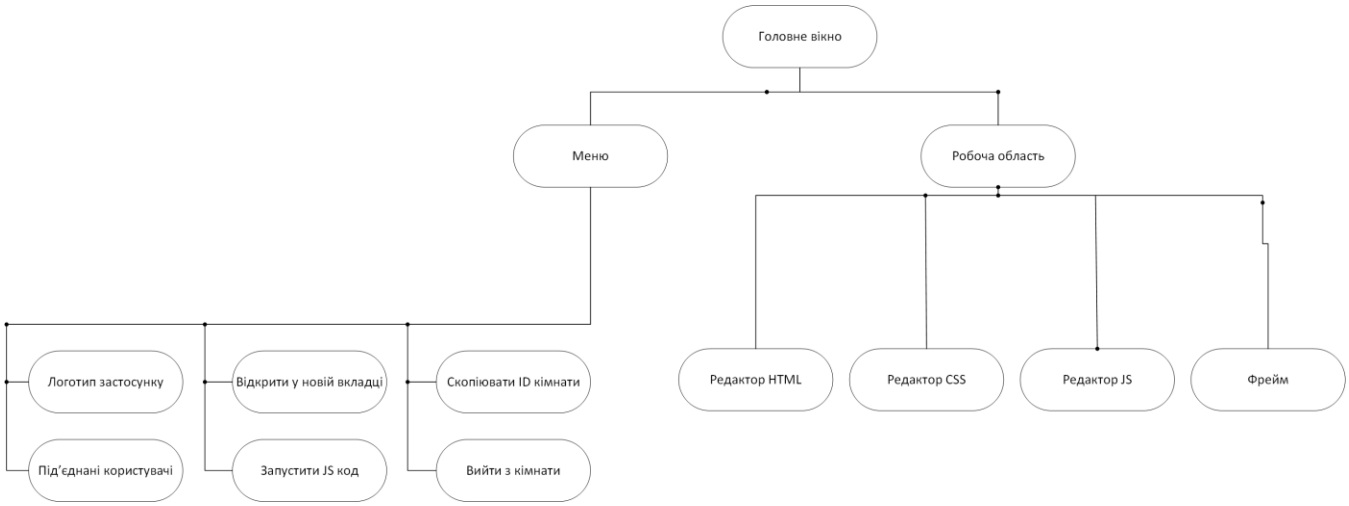


Рисунок 2.5 – Структурна схема програми

2.3 Розробка алгоритмів роботи додатку

Щоб розробити додаток колективної інформаційної системи необхідно розробити загальний алгоритм роботи додатку (рисунок 2.3), алгоритм логіну, алгоритми обробки вхідних та вихідних даних.

Загальний алгоритм роботи додатку має такі кроки: логін, отримання вхідних даних, обробка вхідних даних, обробка вихідних даних, вивід даних на екран. Блок-схема алгоритму зображена на рисунку 2.3.

Рисунок 2.3 – Блок-схема загального алгоритму роботи додатку

Для того, щоб увійти до кімнати та почати користуватись додатком користувачу необхідно ввести ідентифікатор, або за відсутності, згенерувати його. Програма робить це використовуючи теорію ймовірностей. Також необхідно ввести ім’я користувача. Обидва поля не повинні бути пусті. Блок-схема алгоритму зображена на рисунку 2.4.

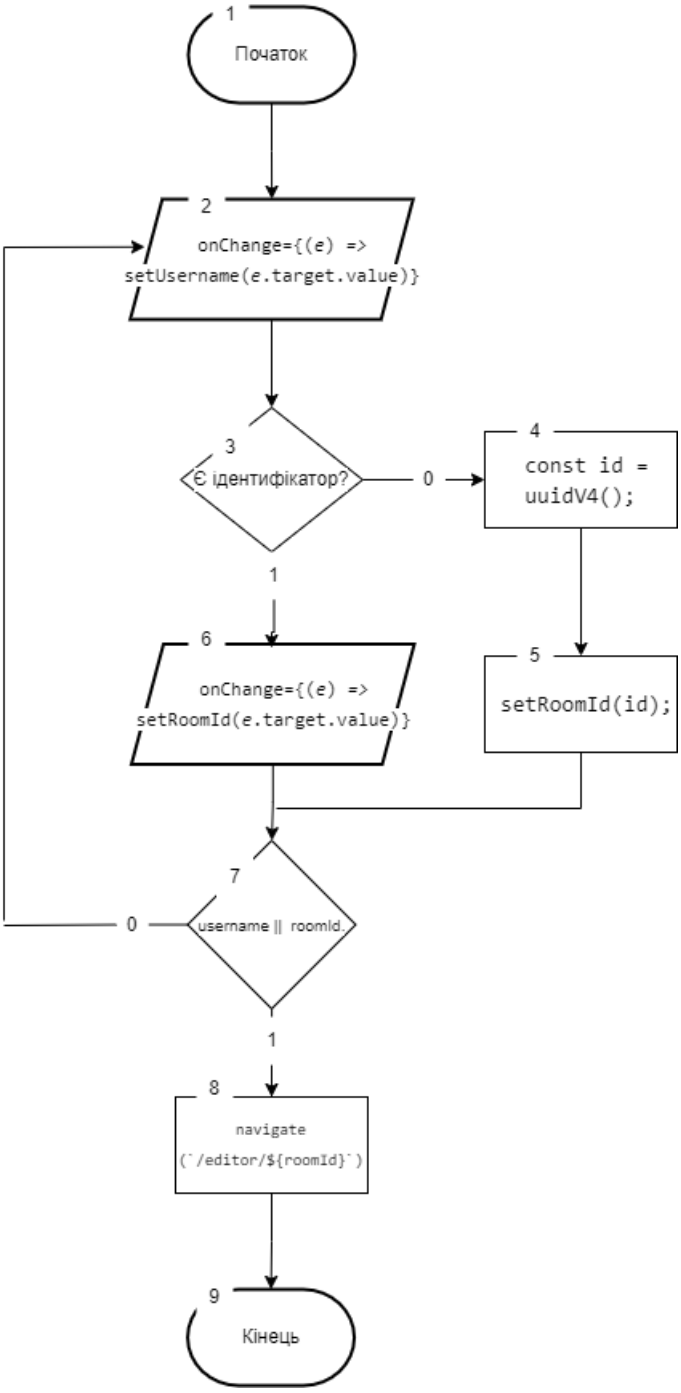


Рисунок 2.4 – Блок-схема алгоритму авторизації користувача

Дані, які користувач вводить у редактор коду, програмний застосунок розбиває на слова та аналізує їх зміст. Якщо слово ключове, воно отримує спеціальний клас, який змінює його зовнішній вигляд. Робиться це шляхом додавання спеціальних класів, які завчасно описані в таблицях стилів. Наприклад, ключове слово function програма розпізнає та підсвітить його іншим кольором, теж саме з CSS та HTML тегами. Блок-схема алгоритму представлена на рисунку 2.5

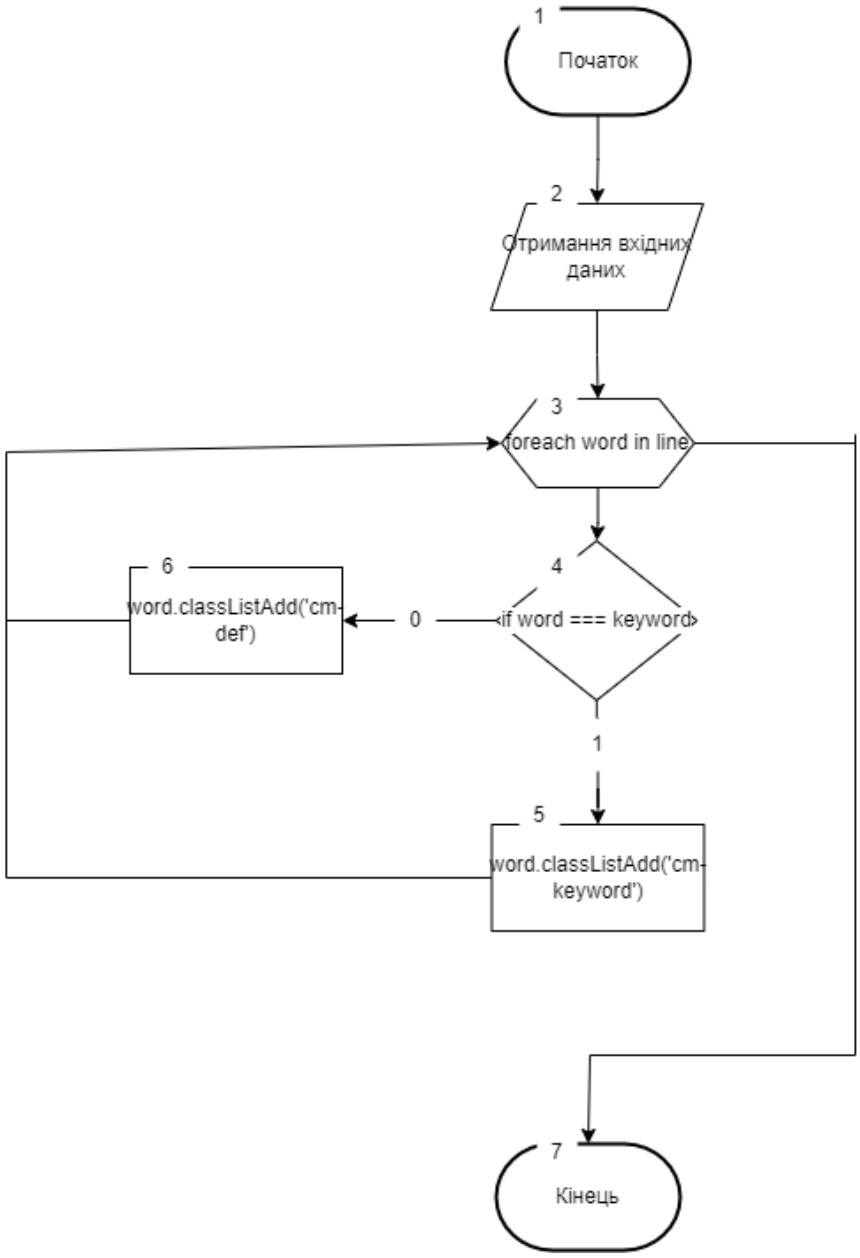


Рисунок 2.5 – Блок-схема алгоритму роботи з вхідними даними

Вивід даних здійснюється у фрейм. Дані, які були введені у програму, огортаються у спеціальні теги та вставляються в DOM-дерево HTML-документу фрейму. Блок-схема алгоритму зображена на рисунку 2.6.

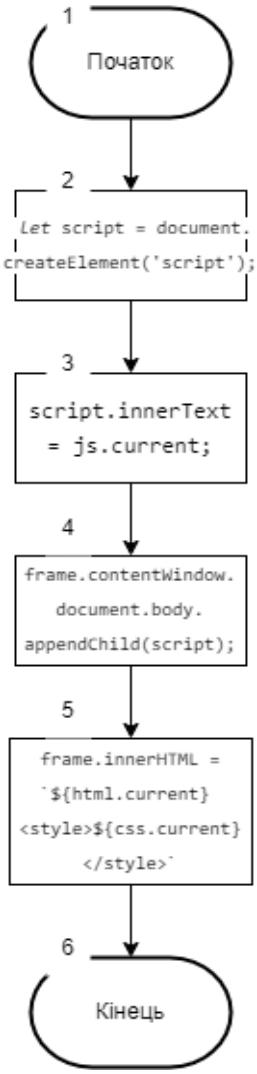


Рисунок 2.6 – Блок-схема алгоритму виводу вихідних даних

2.4 Розробка моделей активності додатку

Для розробки програмної реалізації колективної інформаційної системи необхідно розробити загальну модель активності застосунку (рисунок 2.7), та модель активності логіну, також було створено трансакційну моделі для групової комунікації.

У загальній моделі представлено дії користувача при роботі з програмним застосунком. Після логіну усіх користувачів до віртуальної кімнати, один

користувач робить зміни в одному з редакторів та бачить результат своїх змін, далі його зміни синхронізуються з іншими користувачами і вони теж можуть бачити його зміни разом з результатом цих змін.

Рисунок 2.7 – Загальна модель активності застосунку

Модель логіну представлена на рисунку 2.8. Вона відображає можливі дії користувача під час логіну в застосунок. Головними кроками є наявність у користувача ідентифікатору кімнати та правильність введених даних.

Ідентифікатор слугує унікальним номером віртуальної кімнати, яка об’єднує користувачів і без нього приєднатись до колег не вийде.

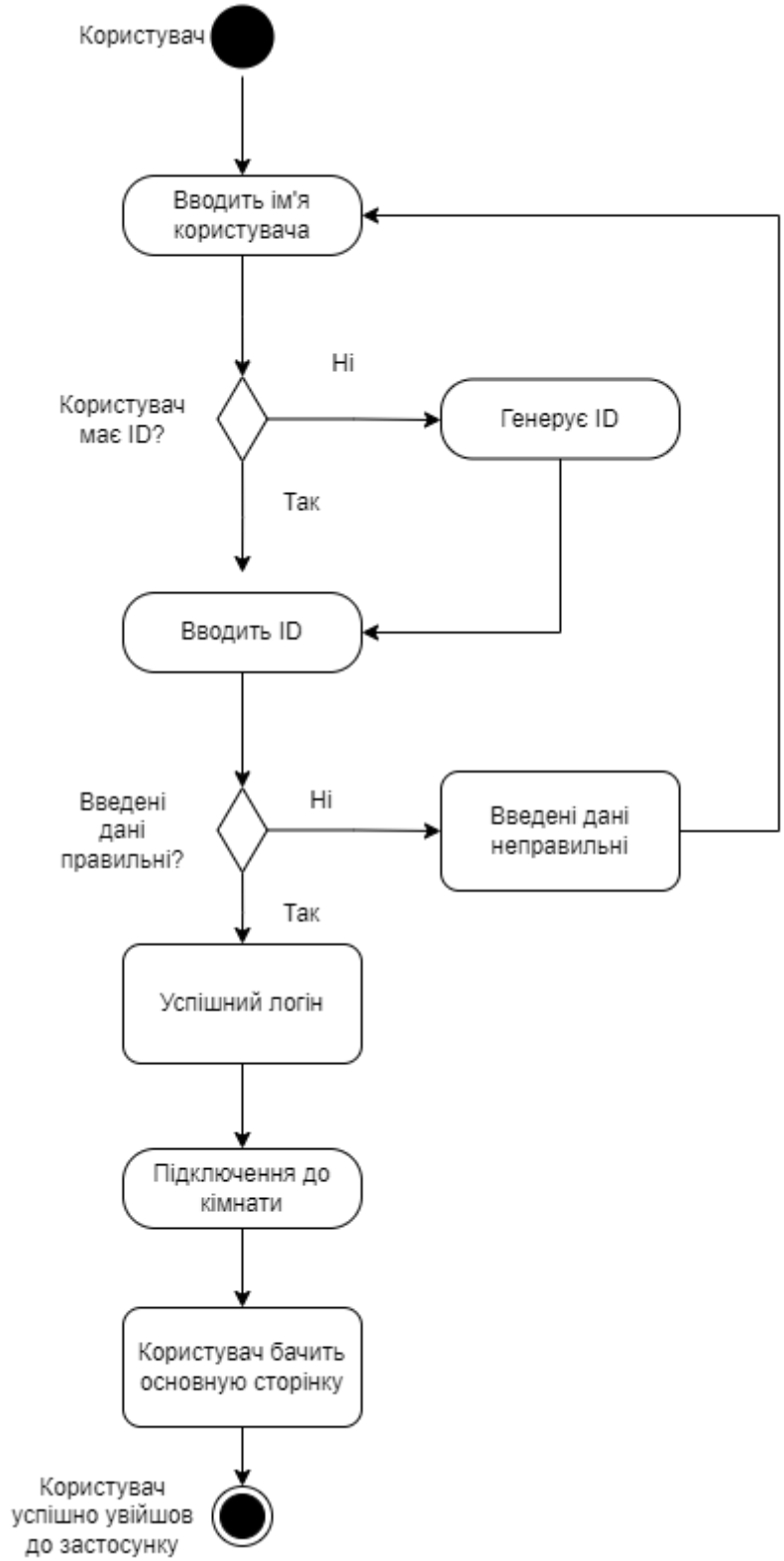


Рисунок 2.8 –Модель активності авторизації

Також для розуміння того, як саме комунікують користувачі між собою в процесі використання програми, була створена схема моделі комунікацій у даному застосунку, яка базується на трансакційній моделі комунікації (рисунок 2.9).

Серед моделей комунікації уснують декілька основних: лінійна, інтерактивна та трансакційна.

Лінійна модель передбачає односторонню спрямованість інформації без зворотного зв'язку. Вона використовується при передачі наказів та розпоряджень, що вимагають лише виконання, і не допускає інтерпретації отриманої інформації з боку реципієнта [14].

Інтерактивна модель комунікації відрізняється від лінійної наявністю зворотного зв'язку, що дозволяє учасникам такого спілкування по черзі ставати джерелом інформації, то її приймачем.

Трансакційна модель представляє комунікацію як процес одночасного відправлення та отримання повідомлень комунікаторами. Ця модель комунікації відрізняється від інтерактивної моделі як кількістю людей, одночасно діючих у процесі комунікації, а й постійної зміною ролей, виконуваних учасниками .Вони поперемінно виступають то ролі джерела інформації, то ролі приймаючого її, при тому, зміст кожного наступного повідомлення щоразу враховує результати зворотнього зв'язку (і залежить від нього), отриманого під час комунікації від різних її учасників, які у попередні моменти взаємодії виступали як отримувачі. Таким чином, акт комунікації неможливо відокремити від подій, які йому передують і йдуть за ним [15].

Саме ця модель найбільш повно відображає ситуацію та тип комунікації, які мають місце під час спільної роботи над одним програмним продуктом. При реалізації трансакційної моделі всі учасники подібної публічної комунікації однаково зацікавлені як у отриманні інформації, і у її обговоренні та прийнятті рішення, якщо це виявляється необхідним у цій ситуації.

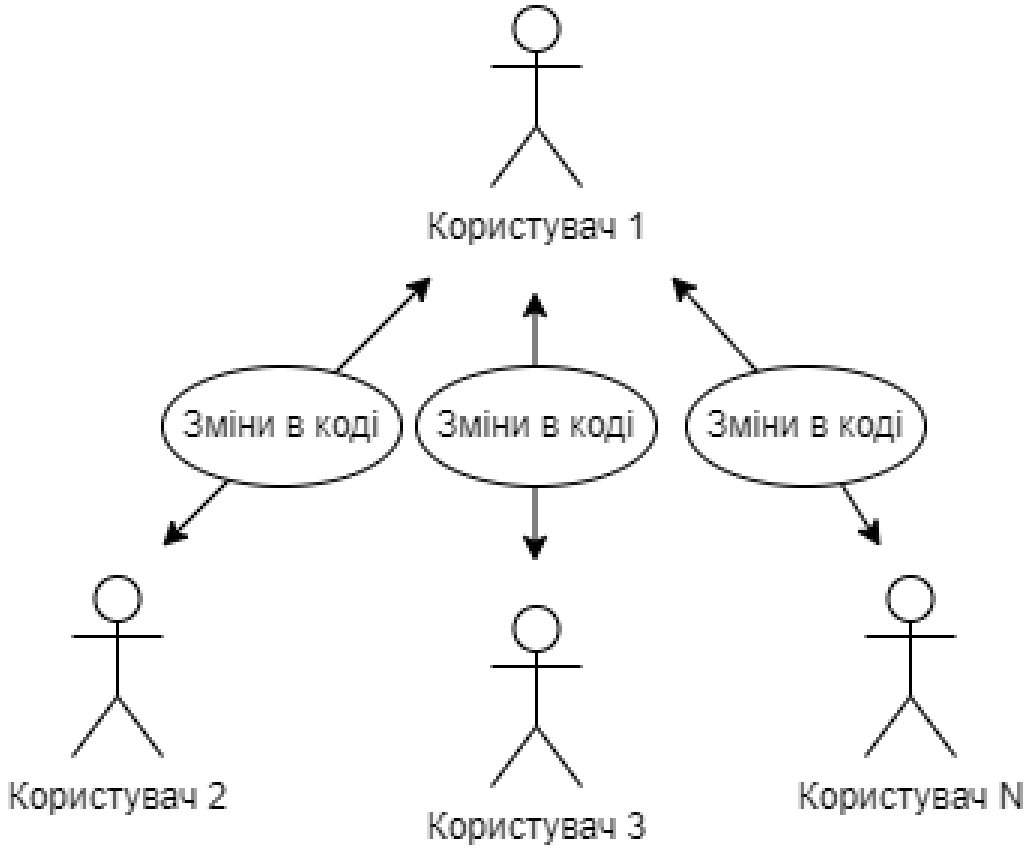


Рисунок 2.9 – Схема трансакційної моделі для групової комунікації

Даний варіант моделі реалізовується у вигляді колективного інформаційного середовища в веб-застосунку. В даній моделі номер користувача не важливий, адже коли будь-який з них робить якісь зміни, всі інші отримують їх.

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних додатку. У результаті чого було вирішено розбивати вхідні дані на слова та додавати їм спеціальні класи в залежності від змісту.

Проведено розробку графічного інтерфейсу користувача.

Розроблено алгоритми логіну, обробки вхідних та вихідних даних, загальний алгоритм роботи додатку, загальну модель додатку, модель логіну, а також схему трансакційної моделі для групової комунікації.

3 РОЗРОБКА ПРОГРАМИ КОЛЕКТИВНОГО ІНФОРМАЦІЙНОГО СЕРЕДОВИЩА

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Для розробки серверної частини додатку розглядались наступні найбільш вживані для цього мови програмування: NodeJS, Python, Ruby. Для розробки клієнтської частини було обрано мову програмування JavaScript вкупі з мовою розмітки HTML та таблицями стилів CSS. Мову було обрано через наявність бібліотеки CodeMirror, яка допомагає створити текстовий редактор, купи веб-фреймворків для полегшення розробки клієнтської частини серед яких було обрано React, та через те, що ця мова була створена саме для розробки веб-застосунків.

JavaScript — це мова високого рівня програмування, яка відповідає стандарту ECMAScript. Серед її особливостей динамічна типізація, об’єктно-орієнтовність, об’єкти базуються на прототипах та присутні функції вищого порядку. Ця мова складається з декількох парадигм, підтримує керування подіями, функціональні та імперативний стилі програмування. Вона має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних і об’єктною моделлю документа (DOM).

Механізми JavaScript спочатку використовувалися лише у веб-браузерах, але тепер вони є основними компонентами деяких серверів і різноманітних програм. Найпопулярнішою системою виконання для такого використання є Node.js.

Скрипти, написані на JS, вбудовуються в документ HTML і взаємодіють з DOM. Усі основні веб-браузери мають вбудований механізм JavaScript, який виконує код на пристрої користувача.

Приклади скриптів:

- Завантаження нового вмісту веб-сторінки без перезавантаження сторінки. Наприклад, користувачі соціальних мереж можуть надсилати та отримувати повідомлення, не залишаючи поточної сторінки.

- Анімації веб-сторінок, такі як поява та зникнення об’єктів, зміна розміру та їх переміщення.

- Гра в браузерні ігри.

- Керування відтворенням потокового медіа.

- Створення спливаючих оголошень.

- Перевірка введених значень веб-форми перед відправкою даних на веб-сервер.

- Реєстрація даних про поведінку користувача та надсилання їх на сервер. Власник веб-сайту може використовувати ці дані для аналітики, відстеження реклами та персоналізації.

- Перенаправлення користувача на іншу сторінку.

Більшість веб-сайтів використовують сторонню бібліотеку JavaScript або веб-фреймворк для написання сценаріїв на стороні клієнта. jQuery є найпопулярнішою бібліотекою для JavaScript, Facebook створив бібліотеку React для своїх потреб, а пізніше випустив її у відритому доступі, інші сайти, включаючи Twitter, тепер використовують його. Аналогічно, фреймворк Angular, створений Google для своїх веб-сайтів, включаючи YouTube і Gmail, тепер є проектом з відкритим кодом, яким користуються інші.

На відміну від цього, термін «Vanilla JS» був придуманий для веб-сайтів, які не використовують жодних бібліотек чи фреймворків, а натомість повністю покладаються на стандартну функціональність JavaScript.

Основні особливості мови JavaScript:

- Імперативний і структурований. JavaScript підтримує більшу частину синтаксису структурованого програмування з C (наприклад, оператори if, цикли while, оператори switch, цикли do while тощо). Одним частковим винятком є визначення області видимості: спочатку JavaScript мав лише область визначення

функції за допомогою ключового слова var; потім у ECMAScript 2015 було додано область визначення блоку з ключовими словами let і const.

- Слабо типізована JavaScript є слабко типізованою мовою програмування, що означає, що певні типи неявно приводяться в залежності від використовуваної операції.

- Динамічна типізація. JavaScript має динамічну типізацію, як і більшість інших мов сценаріїв. Тип асоціюється зі значенням, а не з виразом. Наприклад, змінну, спочатку зв'язану з числом, можна перепризначити рядку.

Node.js — це кросплатформне середовище виконання з відкритим вихідним кодом для розробки серверних і мережевих додатків. Програми Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js в OS X, Microsoft Windows і Linux. Node.js також надає багату бібліотеку різноманітних модулів JavaScript, що значною мірою спрощує розробку веб-додатків за допомогою Node.js [17].

Python – це мова програмування, яка часто використовується для створення веб-сайтів і програмного забезпечення, автоматизації завдань і аналізу даних. Python — це мова загального призначення, що означає, що його можна використовувати для створення різноманітних програм і не спеціалізується на будь-яких конкретних проблемах. Ця універсальність разом із зручністю для початківців зробили її однією з найбільш використовуваних мов програмування сьогодні. Python має велику стандартну бібліотеку, доступну будь-кому. Є бібліотеки для маніпулювання зображеннями, бази даних, модульне тестування, вирази та багато інших функцій. На додаток до стандартної бібліотеки існує також зростаюча колекція з тисяч компонентів, які всі доступні в індексі пакетів Python.

Коли мова програмування інтерпретується, це означає, що вихідний код виконується рядок за рядком, а не весь відразу. Мови програмування, такі як C++ або Java, не інтерпретуються, і, отже, їх потрібно спершу компілювати для їх запуску. Немає необхідності компілювати Python, оскільки він обробляється під час виконання інтерпретатором.

Однією з важливих функцій Python є те, що він підтримує як об'єктно-орієнтоване, так і процедурно-орієнтоване програмування. [18].

Ruby – це об’єктно-орієнтована мова сценаріїв з відкритим вихідним кодом, винайдена в середині 90-х Юкіхіро Мацумото. На відміну від таких мов, як C і C++, мова сценаріїв не спілкується безпосередньо з обладнанням. Він записується в текстовий файл, а потім аналізується інтерпретатором і перетворюється на код. Ці програми, як правило, мають процедурний характер. Ruby чудово підходить для створення настільних додатків, статичних веб-сайтів, служб обробки даних і навіть інструментів автоматизації. Він використовується для веб-серверів, DevOps, а також для скрейпінгу й сканування веб-сайтів [19].

У табл. 3.1 зведено результати порівняльного аналізу мов програмування.

Таблиця 3.1 – Порівняння мов програмування

Характеристики	Мова програмування		
	NodeJS	Python	Ruby
Асинхроність	+	+/-	-
Кількість біоліотек	+	+	-
Широка спеціалізація	+	+	+
Створення анонімних функцій	+	+	+
Створення динамічних масивів	+	+	+
Розробка серверної частини	+	+	+

Як видно з таблиці NodeJS має перевагу над Python і Ruby, тому для реалізації програмного додатку було обрано саме це середовище, адже воно задовольняє всі необхідні вимоги для створення модуля підключення користувачів та має великий набір інструментів, що грає значну роль в розробці програмних продуктів.

3.2 Розробка модулю підключення користувачів

Модуль підключення користувачів розміщується в файді server.js і використовується для того, щоб створити з'єднання між двома і більше користувачами, а також синхронізувати стан додатку та передавати зміни іншим користувачами. Для створення модулю було використано середовище NodeJS та протокол WebSocket. Фрагмент коду модулю наведено нижче.

```
function getAllConnectedClients(roomId) {
  return Array.from(io.sockets.adapter.rooms.get(roomId) || []).map(
    (socketId) => {
      return {
        socketId,
        username: userSocketMap[socketId],
      };
    }
  );
}
```

Описана вище функція повертає об'єкт, що містить в собі ідентифікатор користувача та його ім'я.

```
io.on('connection', (socket) => {
  console.log('socket connected', socket.id);

  socket.on(ACTIONS.JOIN, ({ roomId, username }) => {
    userSocketMap[socket.id] = username;
    socket.join(roomId);
```

```
    const clients = getAllConnectedClients(roomId);
        clients.forEach(({ socketId }) => {
            io.to(socketId).emit(ACTIONS.JOINED, {
                clients,
                username,
                socketId: socket.id,
            });
        });
    });

    socket.on(ACTIONS.CODE_CHANGE, ({ roomId, code, editor }) => {
        socket.in(roomId).emit(ACTIONS.CODE_CHANGE, { code, editor });
    });

    socket.on(ACTIONS.SYNC_CODE, ({ socketId, code, editor }) => {
        io.to(socketId).emit(ACTIONS.CODE_CHANGE, { code, editor });
    });

    socket.on('disconnecting', () => {
        const rooms = [...socket.rooms];
        rooms.forEach((roomId) => {
            socket.in(roomId).emit(ACTIONS.DISCONNECTED, {
                socketId: socket.id,
                username: userSocketMap[socket.id],
            });
        });
        delete userSocketMap[socket.id];
        socket.leave();
    });
});
```

Фрагмент коду представлений вище складається з функцій обробників подій. В коді представлені функції для під’єднання користувачів, їх відключення, зміни та синхронізації коду.

```
useEffect(() => {
  const init = async () => {
    socketRef.current = await initSocket();
    socketRef.current.on('connect_error', (err) => handleErrors(err));
    socketRef.current.on('connect_failed', (err) => handleErrors(err));

    function handleErrors(e) {
      console.log('socket error', e);
      toast.error('Socket connection failed, try again later.');
      reactNavigator('/');
    }
```

Реакт-хук представлений вище відповідає за обробку помилок при роботі з застосунком. Коли виникає якась помилка, об’єкт помилки виводиться до консолі, а користувач переходить до початкової сторінки.

3.3 Розробка модулю редактору коду

Для створення редактору коду використовувалась бібліотеки CodeMirror та React. Стан редакторів керується за допомогою реакт-хуків. Фрагмент коду модулю представлений нижче.

```
const Editor = ({ socketRef, roomId, onCodeChange }) => {
  let html = useRef('');
  let css = useRef('');
  let js = useRef('');
  let srcDoc = useRef('');

  function runMarkup() {
    if (
      !document.getElementsByTagName('iframe')[0]?.contentWindow.document ||
      !document.getElementsByTagName('iframe')[0]?.contentDocument
    ) {
      return;
```

```
  } else if (
    document.getElementsByTagName('iframe')[0].contentWindow.document
      .readyState == 'complete'
  ) {
    srcDoc.current = `
    <html>
    <body>${html.current}</body>
    <style>${css.current}</style>
    </html>`;
    document.getElementsByTagName(
      'iframe'
    )[0].contentWindow.document.body.innerHTML = srcDoc.current;
  }
}

useEffect(() => {
  if (socketRef.current) {
    socketRef.current.on(ACTIONS.CODE_CHANGE, ({ code, editor }) => {
      if (code !== null) {
        if (editor === 'js') {
          editorRef.current.setValue(code);

        } else if (editor === 'html') {
          htmlEditorRef.current.setValue(code);
          runMarkup();

        } else if (editor === 'css') {
          cssEditorRef.current.setValue(code);
          runMarkup();

        } else if (!editor) {
          if (localStorage.getItem('js'))
            editorRef.current.setValue(localStorage.getItem('js'));
          if (localStorage.getItem('html'))
            htmlEditorRef.current.setValue(localStorage.getItem('html'));
```

```
if
(localStorage.getItem('css'))
          cssEditorRef.current.setValue(localStorage.getItem('css'));
        runMarkup();
       }
     }
    });
   }

   return () => {
    socketRef.current.off(ACTIONS.CODE_CHANGE);
   };
  }, [socketRef.current]);
```

Модуль переналаштовує текстові поля у коді та робить з них редактори коду. Також модуль відповідає за вивід результату роботи користувачів у фрейм.

3.4 Висновки

У третьому розділі було обґрунтовано вибір мов програмування та технологій, які будуть використовуватися при розробці програмного продукту та виконано порівняння аналогів.

У результаті порівняння для серверної частини було обрано середовище NodeJS з фреймворком Express та протоколом WebSocket, а для розробки графічного інтерфейсу – мову програмування JavaScript з бібліотеками React та CodeMirror.

Було проведено розробку модулів підключення користувачів та редактору коду.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Тестування програми

Тестування програмного забезпечення — це метод перевірки, чи відповідає фактичний програмний продукт очікуваним вимогам, і спосіб переконатися, що програмний продукт не містить дефектів. Він включає виконання програмних/системних компонентів з використанням ручних або автоматизованих інструментів для оцінки однієї або кількох властивостей застосунку, що тестується. Тестування програмного забезпечення традиційно відокремлювалося від решти етапів розробки програмного забезпечення. Віно часто проводиться пізніше в життєвому циклі розробки програмного забезпечення після етапу розробки. Метою тестування програмного забезпечення є виявлення помилок, багів та перевірка функціоналу відповідно до вимог. Правильно перевірений програмний продукт забезпечує надійність, безпеку та високу продуктивність, що в подальшому призводить до економії часу, економічної ефективності та задоволеності клієнтів. [20].

Сьогодні існує багато підходів до тестування програмного забезпечення, але найпопулярнішими в цій галузі є методи тестування чорного ящика, білого ящика та сірого ящика.

Особливість методу тестування чорного ящика полягає в тому, що тестувальники, які його виконують, не знають внутрішньої структури та вихідного коду програмного забезпечення, яке вони тестують. І вони їм не потрібні, як і глибокі знання мов програмування чи видатні навички програмування для виконання тестів. Це пов’язано головним чином з тим, що мета цього методу тестування полягає не в тому, щоб глибоко копатися в коді, проходячи через середини програмного забезпечення, а у взаємодії з його інтерфейсом користувача, тестування його функціональності та переконання, що робота застосунку відповідає визначеним вимогам. Тому тестування чорного

ящика можна також назвати функціональним тестуванням або тестуванням на основі специфікації.

Такі тести проводяться з точки зору кінцевих користувачів незалежною групою тестування в ході STLC (Software Testing Life Cycle). Тестер надає дійсні або недійсні вхідні дані та перевіряє вихідні дані щодо очікуваних результатів. Кожен неочікуваний результат і відхилення документуються та повідомляються команді розробників, що допомагає їм на ранній стадії знаходити та усувати функціональні помилки та невідповідності.

Цей метод застосовний практично на кожному рівні тестування програмного забезпечення: одиничному, інтеграційному, системному та прийнятному. У модульному тестуванні метод чорного ящика використовується для перевірки інтерфейсу на відповідність специфікаціям, наданим клієнтом.

При інтеграційному тестуванні метою є пошук та усунення помилок у взаємодії між інтегрованими компонентами інтерфейсу. Метод чорного ящика також може бути ефективно застосований під час тестування системи для аналізу відповідності системи вимогам, а також при приймальному тестуванні, де він може допомогти підтвердити прийнятність програмного продукту, тестуючи його в різних несподіваних ситуаціях та обставинах.

Деякі з найпоширеніших методів проектування тестів чорної скриньки включають:

- тестування таблицею рішень стає в нагоді під час налагодження програмного забезпечення на основі вбудованих операторів if-then-else та switch-case у таблицях рішень. Це ефективний спосіб знайти помилки в тому, які дії відповідають яким умовам;

- вгадування помилок означає, що тестувальник розробляє тестові випадки на основі своєї інтуїції та досвіду попереднього тестування. Вони використовують його, щоб визначити, що може призвести до збою програмного забезпечення або появи помилок;

- тестування парами – це методика, яка використовується для перевірки всіх можливих дискретних комбінацій кожної пари вхідних параметрів. Це дійсно

допомагає знайти поширені помилки, які зазвичай приховані у взаємодії між парами параметрів;

- техніка еквівалентного розподілу передбачає поділ вхідних даних на різні менші розділи, класи еквівалентних даних, з яких можна отримати тестові випадки. Ця техніка використовується для створення тестового випадку, який охоплює кожен розділ відразу, скорочуючи час, необхідний для тестування.

Тестування чорного ящика дійсно може допомогти виявити будь-яку двозначність, нечіткість і суперечності у функціональних специфікаціях. Це дає можливість тестувальникам оцінити та підвищити якість реалізації функціональних можливостей, не втручаючись безпосередньо у великі сегменти коду програмного забезпечення.

Тестування «чорного ящика» є абсолютно неупередженим, оскільки тестування проводиться незалежною командою, яка відокремлює погляди кінцевих користувачів від точки зору розробників. Серед трьох методів тестування «чорного ящика» має найшвидшу розробку тестового прикладу, оскільки воно не вимагає знань з програмування і може бути легко виконане тестувальниками без технічної підготовки.

Тим не менш, метод можна ефективно застосувати лише для тестування невеликих частин програмного забезпечення. Ретельне тестування великого складного програмного забезпечення за допомогою цього методу виявиться досить неефективним, а також займе дуже багато часу. Крім того, цей метод вимагає чітких і вичерпних специфікацій, щоб бути ефективним. Інакше буде надзвичайно важко розробити тестові випадки, а сценарії забезпечуватимуть дуже обмежене покриття.

На відміну від тестування чорного ящика, яке зосереджується на функціональності, мета методу тестування білого ящика полягає в тому, щоб виконати аналіз внутрішньої структури програмного забезпечення та логіки, що стоїть за нею. Тому тестування білого ящика іноді називають структурним тестуванням або тестуванням на основі логіки. Цей метод дуже трудомісткий і вимагає від тестувальників володіти сильними навичками програмування,

повного знання програмного забезпечення, яке вони тестують, і доступу до всіх документів з вихідним кодом і архітектурою, інакше тестувальники не зможуть розробити належні тестові випадки.

Отже, тестування «білого ящика» зазвичай виконується професійними розробниками, які застосовують свій досвід, щоб отримати внутрішній погляд на структуру, з’ясувати, що саме відбувається у вихідному коді, і виправити те, що не працює належним чином. На додаток до глибоких знань, метод також вимагає спеціалізованих інструментів для аналізу вихідного коду та налагодження.

Тестувальники ретельно вивчають код та інші внутрішні аспекти даного програмного забезпечення, визначають усі дійсні та недійсні введення, а потім звіряють вихідні дані з очікуваними результатами. Вони перевіряють твердження та умови, шляхи коду та потоки даних, щоб переконатися, що немає прихованих помилок або елементів, схильних до дефектів.

Тестування білого ящика можна застосувати на рівні модульного тестування, але сьогодні воно в основному використовується для інтерграційного та регресійного тестування. Цей метод дозволяє тестувальникам перевіряти шляхи всередині блоків на наявність дефектів коду та інших проблем, які заважають програмному забезпеченню працювати належним чином.

Під час інтеграційного тестування метод допомагає аналізувати взаємодії між різними інтерфейсами та підсистемами. Під час регресійного тестування метод білого ящика може бути дуже ефективно застосований завдяки використанню тестових випадків білого ящика, перероблених на рівні одиничного та інтеграційного тестування.

Деякі з найпоширеніших методів проектування тестів білого ящика включають:

- тестування потоку керування — це стратегія структурного тестування, яка використовує потоки керування програмним забезпеченням для перевірки логіки коду, виконуючи вхідні значення та перевіряючи, чи відповідають вони потрібним результатам;

- тестування потоку даних виявляє неправильне використання значень даних і аномалії потоку даних, створені помилками у коді. Техніка зосереджена на ловлі склданих ділянок коду, щоб можна було провести більше тестування для виправлення або усунення цих помилок.

Не завжди існує один безперервний потік коду. Іноді код розгалужується для виконання певних функцій, що охоплюють різні умови істини/неправди. Техніка тестування гілок зосереджена на перевірці цих гілок та усуненні відхилень.

У порівнянні з тестуванням чорної скриньки, метод білого ящика схожий на точність, яка виявляє помилки в прихованому коді, видаляючи зайві рядки. Таке глибоке знання вихідного коду полегшує боротьбу з побічними ефектами, що дуже корисно. Це також забезпечує відстеження кожного тесту на рівні вихідного коду, де кожну майбутню зміну можна легко відобразити в нещодавно доданих або змінених тестах.

Він виявляє кожне непомітне вузьке місце в коді, забезпечує команді розробників максимальне охоплення та чіткий, стислий відгук. Це полегшує команді розробників скорочення технічної заборгованості шляхом оптимізації та підтримки якості свого коду, тим більше, що тестування білого ящика також можна автоматизувати.

Проте, автоматизоване чи ні, тестування «білого ящика» зазвичай займає дуже багато часу та складне. Цей підхід вимагає від тестувальників першокласних навичок програмування та глибокого розуміння програмного забезпечення, яке вони тестують, на рівні коду. Це передбачає наймати першокласних інженерів, щоб тести були ефективними, що також робить метод дорожчим.

Результати тесту також суворо прив’язані до способу написання коду. Якщо код, прив’язаний до тієї ж функціональності, змінено, це робить попередні припущення недійсними, що може призвести до невдалого тестового випадку з помилковими результатами. Крім того, в той час як метод чорної скриньки блискучий у функціональному тестуванні, тестування білого ящика не зможе забезпечити результат, оскільки воно зосереджується лише на існуючому стані

програмного забезпечення. Це означає, що він не зможе надати зворотній зв’язок щодо відсутньої функціональності, залишаючи багато шляхів неперевіреними.

Як вже було описано, тестування чорної скриньки та тестування білої скриньки мають різну спрямованість, показуючи значні переваги в одному, а неефективність або серйозні недоліки в іншому. Тестування сірої скриньки, у свою чергу, дає переваги як тесту чорної скриньки, так і методу тестування білої скриньки, одночасно нейтралізуючи більшість недоліків завдяки ефективному, збалансованому поєднанню цих двох.

Метод сірої скриньки збільшує охоплення методів тестування, зосереджуючи увагу на всіх рівнях тестованого програмного забезпечення незалежно від його складності. У той час як тестувальники «чорної скриньки» перевіряють, чи все добре з інтерфейсами та функціональністю, а тестувальники «білих скриньок» занурюються у внутрішню структуру та виправляють код програмного забезпечення, тестувальник сірої скриньки займається тим і іншим одночасно.

Метод сірої скриньки націлений на складні системи з простим підходом до чорної скриньки, що дає змогу виконувати тести практично будь-кому, від розробників до тестувальників і кінцевих користувачів. Однак для розробки тестових випадків інженеру потрібно часткове знання внутрішньої структури, включаючи документацію щодо структур даних, архітектури, а також функціональних специфікацій програмного забезпечення. Сформовані тестові приклади спрямовані на пошук та усунення дефектів у структурі та ліквідацію будь-яких прогалин, які б уможливили неправильне використання програмного забезпечення.

Тестування сірого ящика виявляється найбільш корисним на рівні інтеграційного тестування. Він добре підходить для тестування веб-додатків, оскільки вони не мають вихідного коду або двійкових файлів, що унеможливлює їх тестування за допомогою методу білого ящика. Тестування сірої скрині також можна застосувати до тестування бізнес-домену, щоб підтвердити, що програмне забезпечення відповідає вимогам.

Деякі з найпоширеніших методів проектування тестів сірого ящика включають:

- матричне тестування відстежує та відображає вимоги користувачів, щоб переконатися, що все охоплено в тестових випадках. Це дозволяє легко визначити будь-які відсутні функції. Це як звіт про статус, який підтверджує, що тестове покриття повне;

- регресійне тестування – це в основному аналіз впливу змін програмного забезпечення. Це включає в себе перевірку, чи правильно працює програмне забезпечення після змін. Ця техніка використовується, щоб переконатися, що немає нових помилок і ніщо не заважає існуючій функціональності;

- техніка тестування шаблонів аналізує раніше виявлені дефекти в збірці, дизайні та архітектурі програмного забезпечення, що тестується. Цей аналіз застосовується, щоб знайти першопричину, конкретну причину даного дефекту та запобігти його повторенню.

Метод тестування сірої скрині пропонує комбіновані переваги тестування чорної скриньки та тестування білої скриньки, усуваючи більшість їхніх недоліків. Метод дуже ненав’язливий, оскільки він заснований на функціональних специфікаціях, інтерфейсах та документації, що дає тестувальникам зазирнути в архітектуру програмного забезпечення, а не повний доступ до вихідного коду або двійкових файлів.

Це також означає, що існує чітка межа між тестувальниками та розробниками, що робить цей метод тестування абсолютно неупередженим. Крім того, тестування сірого ящика дає можливість інтелектуального створення тестів – виняткових тестових сценаріїв для аналізу типів даних, протоколів зв’язку, винятків тощо.

Однак метод вимагає відмінного управління проектом, оскільки тестування може бути зайвим, якщо розробник уже запустив відповідні тестові випадки. Через обмежене знання внутрішньої структури програмного забезпечення та відсутність доступу до його коду тестування сірої скині забезпечує лише часткове

покриття тестом з багатьма неперевіреними шляхами коду, що також робить його непридатним для тестування алгоритму. І останнє, але не менш важливо, дуже важко пов’язати ідентифікацію дефектів у розподілених системах за допомогою методу сірої скрині.

Було описано та порівняно три основні методи тестування програмного забезпечення, які компанії використовують для забезпечення якості коду своїх продуктів і суворого дотримання специфікації вимог до програмного забезпечення. Ці методи тестування програмного забезпечення допомагають усунути багато проблем, які в довгостроковій перспективі можуть обернутися величезними технічними боргами. [21].

Отже, у якості методики тестування програмного додатку було обрано тестування методом сірого ящика, тому що були частково описані алгоритми роботи додатку, та моделі активності.

Протестуємо функцію логіну користувачів та підключення до однієї кімнати. Для цього необхідно запустити застосунок. Вікно логіну зображено на рисунку 4.1.

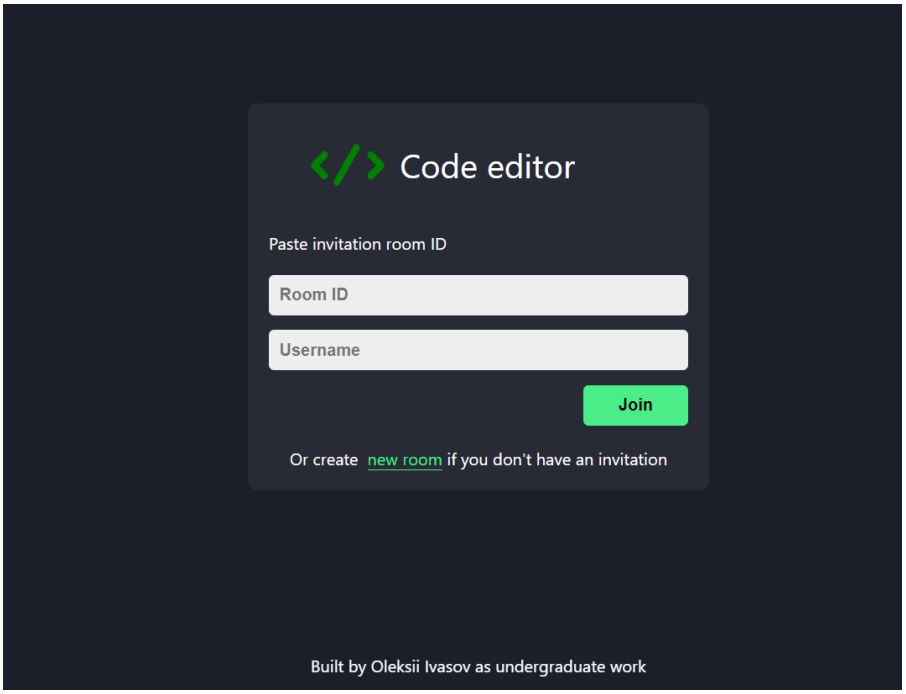


Рисунок 4.1 – Вікно логіну

Для логіну необхідно вписати ID кімнати та ім’я користувача. Після цього необхідно натиснути кнопку «Join». Якщо користувач не має ID кімнати, він може створити її за допомогою кнопки «new room» внизу вікна. ID кімнати може складатись з букв і цифр, головна вимога щоб він був унікальним. Приклад заповненої форми на рисунку 4.2.

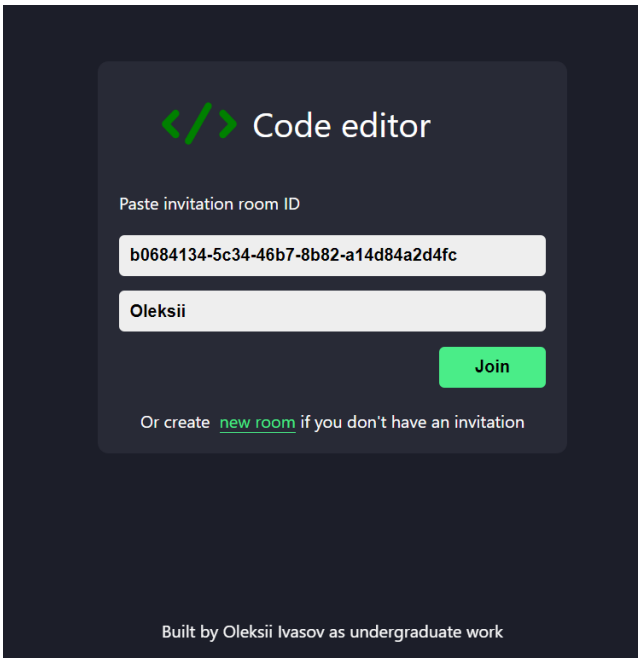


Рисунок 4.2 – Заповнене вікно логіну

Якщо користувач не ввів одне або обидва поля, в верхньому правому куті він побачить помилку, якщо ж все вірно, то в результаті отримаємо головне вікно застосунку (рисунок 4.3).

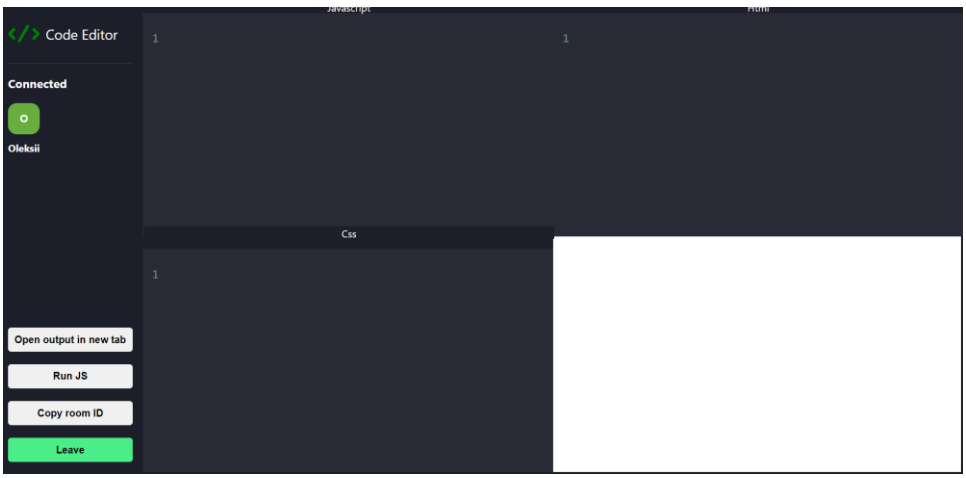


Рисунок 4.3 – Головне вікно застосунку

Для того, щоб інший користувач міг приєднатись, йому необхідно ввести такий самий ідентифікатор кімнати та своє ім’я користувача. Для того, щоб передати ID кімнати, користувач може натиснути кнопку «Copy room ID» в лівому меню та скопіювати ID таким чином. Під’єднаних користувачів можна побачити в лівій частині застосунку під надписом «Connected» (рисунок 4.4).

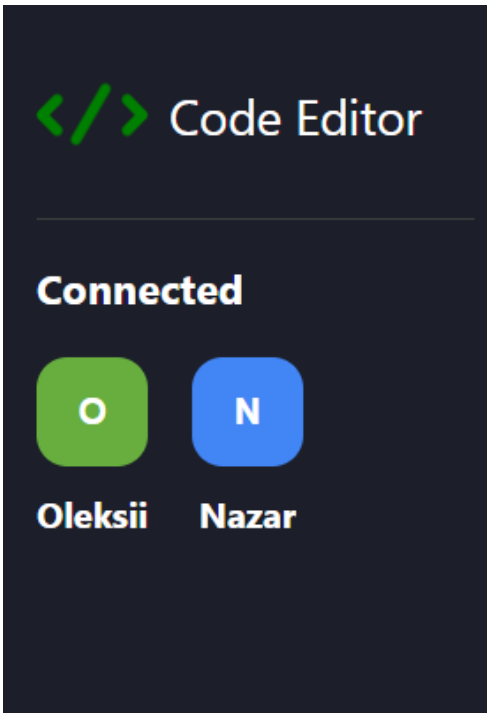


Рисунок 4.4 – Місце де можна побачити підключених користувачів

Отже, функція логіну та підключення користувачів до однієї кімнати працює правильно. Далі необхідно правильність роботи редакторів коду та їх синхронізації між користувачами. Для цього потрібно написати будь-який код в будь-який з редакторів. Результат роботи коду можна буде побачити у фреймі в правому нижньому куті застосунку. Код, який пише один користувач, автоматично буде з’являтись в інших користувачів в одній кімнаті (рисунок 4.5).

Рисунок 4.5 – Робота редакторів коду

Синхронізація коду також працює, наводити скріншот немає сенсу, адже він виглядає точно як рисунок 4.5.

Для перевірки функції відкрити результат роботи у новій вкладці необхідно натиснути на кнопку «Open output in new tab», після чого вміст фрейму відкриється у новій вкладці, де його можна зручніше переглянути (рисунок 4.6).

Рисунок 4.6 – Вміст фрейму у новій вкладці

Для підтвердження результатів роботи програмного застосунку очікуваним було здійснення таких функціональних можливостей: відображення елементів вікна логіну; генерація ID кімнати; функція логіну у застосунок; функція підключення користувачів; функція синхронізації коду між користувачами; функція відкриття результату роботи у новій вкладці.

Таблиця 4.1 Тестові випадки

Ідентифікатор	Назва	Методика проведення тестування	Очікуваний результат	Результат
ТВ-1	Перевірка відображення елементів вікна логіну	1. Завантажити додаток, відкривши сайт	1. Відображається логотип та назва додатку. 2. Відображається меню. 3. Відображається вікно логіну.	Виконано
ТВ-2	Перевірка Генерації ID	1. Завантажити додаток, відкривши сайт 2. Натиснути «new room»	1. Генерується ідентифікатор та вставляється у відповідне поле.	Виконано
ТВ-3	Перевірка функції логіну у застосунок	1. Завантажити додаток, відкрити сайт. 2. Ввести ім’я користувача та ідентифікатор. 3. Натиснути кнопку «Join».	1. Відкривається головне вікно застосунку.	Виконано
ТВ-4	Перевірка фунції підключення користувачів.	1. Завантажити додаток, відкривши сайт. 2. Повторити кроки вказані в ТВ-3. 3. Залогінитись іншим користувачем, по тому ж ID.	1. У меню з лівої сторони екрану видно 2 підключених користувача.	Виконано

Продовження таблиці 4.1

Ідентифікатор	Назва	Методика проведення тестування	Очікуваний результат	Результат
ТВ-5	Перевірка функції сихнронізації коду між користувачами	1. Завантажити додаток, відкривши сайт. 2. Перший користувач входить до кімнати. 3. Другий користувач входить до тієї ж кімнати. 4. Перший користувач вводить код у редактор.	1. Другий користувач бачить зміни першого. 2. Обидва користувачі бачуть результат роботи коду у фреймі.	Виконано
ТВ-6	Перевірка функції відкриття результату роботи у новій вкладці	1. Повторити кроки вказані в ТВ-3. 2. Ввести код у редактор. 3. Натиснути кнопку «Open output in new tab»	1. Відкриється нова вкладка браузера. 2. Нова вкладка браузера містить в собі те саме, що і фрейм в основній сторінці застосунку.	Виконано

Отже, за допомогою вищенаведених тестових випадків було проведено тестування графічного інтерфейсу та функціоналу програмного продукту, та підтверджено його працездатність і коректність виконання дій.

4.2 Розробка інструкції користувача

Так як це застосунок, який працює в інтернеті, в веб-браузері, то для роботи з ним все що потрібно це будь який веб-браузер: Google Chrome, Opera, Mozilla, Internet Explorer, Safari. А також потрібно підключення до інтернету.

Так як застосунок розгорнуто за допомогою сервісу Heroku та системи контролю версій Git, користувачу не потрібно нічого встановлювати, а лише перейти по посиланню https://ws-online-code-editor.herokuapp.com/.

На сайті користувач зіткнеться з вікном логіну, де необхідно буде ввести унікальний ідентифікатор кімнати та ім'я користувача, після чого він переходить до основого вікна програми. В основному вікні він може писати код в один з трьох редакторів коду, кооперувати свої дії з колегами, та переглядати результати роботи у фреймі в правому нижньому куті. Код компілюється та виконується без дій користувача, окрім JavaScript, який можна запустити клацнувши по кнопці «Run JS» в лівому меню. Також в тому ж меню є кнопки для копіювання ID кімнати, відкриттю результату роботи у новій вкладці та кнопка для виходу з кімнати.

4.3 Висновки

У четвертому розділі було проведено тестування додатку, у результаті якого було доведено його повну працездатність та відповідність поставленому технічному завданню.

Розглянуто методи тестування ПЗ та, проаналізувавши їх, було обрано метод «сірої скриньки» для тестування інтерфейсу і функціоналу додатку.

Створено таблицю тестових випадків для перевірки застосунку.

Розроблено інструкцію користувача, яка описує дії необхідні для керуванням програмним продуктом.

ВИСНОВКИ

Під час виконання бакалаврської дипломної роботи було розроблено програмний застосунок колективної інформаційної системи для організації процесу створення програмного продукту.

Було проаналізовано стан даної проблеми на сьогоднішній день. Розглянуто основні аналоги, визначено їх особливості та недоліки і розроблено порівняння з власним програмним продуктом.

У результаті аналізу для створення графічного інтерфейсу застосунку було обрано мову програмування JavaScript з бібліотекою React та CodeMirror, а для серверної середовище NodeJS. Для розробки було використано середовище програмування Visual Studio Code .

У результаті роботи було зроблено наступне:

- проаналізовано стан задачі;

- порівняно аналоги програмного застосунку, зроблено порівняльну таблицю;

- визначено найбільш ефективний підхід для створення колективного інформаційного середовища;

- розроблено графічний інтерфейс користувача для взаємодії із колективною інформаційною системою;

- розроблено алгоритми роботи програмного застосунку та його модулів;

- розроблено програмний модуль підключення користувачів та редактору коду;

- проведено тестування програмного модуля

Перед створенням програмного продукту було розроблено схеми загального алгоритму роботи додатку, логіну, обробки вхідних та вихідних даних.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О.О. Коваленко, О.С. Івасьов Розробка веб-додатку редактору коду. Матеріал науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ). Вінниця, 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15543/13109

2. Williams L., Kessler R., Pair Programming Illuminated. Бостон: Addison-Wesley Professional, 2002. 288 с.

3. Bolboaca A., Practical Remote Pair Programming: Best practices, tips, and techniques for collaborating productively with distributed development teams. Бірмінгем: Packt Publishing, 2021. 240 с.

4. What is collaborative coding? *Codecademy.:* веб-сайт. URL: https://www.codecademy.com/resources/blog/what-is-collaborative-coding/ (дата звернення 05.05.2022).

5. Beck K., Andres C., Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) 2nd Edition. Бостон: Addison-Wesley, 2004. 224 с.

6. Harrer S., Christ J., Huber M., Remote Mob Programming: At home, but not alone. Монхейм: innoQ Deutschland GmbH, 2020. 41 с.

7. Teletype for Atom. *Teletype.atom*; веб-сайт URL: https://teletype.atom.io/ (дата звернення 07.05.2022).

8. CodePen: Online Code Editor. *Codepen*: веб-сайт. URL: https://codepen.io/ (дата звернення 07.05.2022).

9. AWS Cloud9. *Amazon*: веб-сайт. URL: https://aws.amazon.com/ru/cloud9/ (дата звернення 08.05.2022).

10. WebSockets vs HTTP | Ably Realtime. *Ably*: веб-сайт. URL: https://ably.com/topic/websockets-vs-http#:~:text=Unlike%20HTTP%2C%20where%20you%20have,request%2Fresponse%2Dbased%20methods. (дата звернення 12.05.2022).

11. What is WebRTC and how is it used? TechTarget: веб-сайт. URL: https://www.techtarget.com/searchunifiedcommunications/definition/WebRTC-Web-Real-Time-Communications (дата звернення: 13.05.2022).

12. Lombardi A. WebSocket: Lightweight Client-Server Communications**,** Себастопол: O'Reilly Media, 2015. 144 с.

13. What is websocket and how is it used? *Geeksforgeeks*: веб-сайт. URL: https://www.geeksforgeeks.org/ (дата звернення: 15.05.2022).

14. Narula U. Communication Models, Нью-Делі: Atlantic Publishers & Distributors, 2006. 136 с.

15. Transaction model of communication. *Pressbooks.library*: веб-сайт. URL: https://pressbooks.library.ryerson.ca/communicationnursing/chapter/transaction-model-of-communication/ (дата звернення: 18.05.2022).

16. What is JavaScript used for? *Hackreator*: веб-сайт URL: https://www.hackreactor.com/blog/what-is-javascript-used-for. (дата звернення : 20.05.2022).

17. Node JS – Introduction. *Tuturialspoint*: веб-сайт. URL: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm. (дата звернення : 21.05.2022).

18. What is Python? Executive Summary. *Python*: веб-сайт. URL: https://www.python.org/doc/essays/blurb/ (дата звернення : 22.05.2022).

19. What is the Ruby programming language? *Acloud Guru*: веб-сайт. URL: https://acloudguru.com/blog/engineering/what-is-the-ruby-programming-language (дата звернення : 23.05.2022).

20. What is Software Testing? Definition, Basics & Types in Software Engineering. *Guru*: веб-сайт. URL: https://www.guru99.com/software-testing-introduction-importance.html (дата звернення: 25.05.2022).

21. Difference Between Black-Box, White-Box, and Grey-Box Testing. *Dzone*: веб-сайт. URL: https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo (дата звернення: 25.05.2022).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

" 25 " березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Методи створення колективного інформаційного середовища для організації процесів розробки програмного продукту»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доц. О.О. Коваленко

"____"_____2022 р.

Виконав:

_____ ст. гр. 1ПІ-18б О.С. Івасьов

"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Методи створення колективного інформаційного середовища для організації процесів розробки програмного продукту».

Галузь застосування – колективні інформаційні середовища.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 13 від « 07 » лютого 2022 р.

3. Мета та призначення розробки.

Метою роботи є підвищення продуктивності працівників за рахунок модифікації існуючих технік за рахунок створення колективного інформаційного середовища для організації процесів розробки програмного продукту.

Призначення роботи – практичне використання розробленого спеціалізованого програмного застосунку колективного інформаційного середовища.

4. Вихідні дані для проведення НДР

1. Williams L., Kessler R., Pair Programming Illuminated. Бостон: Addison-Wesley Professional, 2002. 288 с.
2. Bolboaca A., Practical Remote Pair Programming: Best practices, tips, and techniques for collaborating productively with distributed development teams. Бірмінгем: Packt Publishing, 2021. 240 с.
3. Harrer S., Christ J., Huber M., Remote Mob Programming: At home, but not alone. Монхейм: innoQ Deutschland GmbH, 2020. 41 с.
4. Transaction model of communication. Pressbooks.library: веб-сайт. URL: https://pressbooks.library.ryerson.ca/communicationnursing/chapter/transaction-model-of-communication/ (дата звернення: 18.05.2022).

5. Технічні вимоги

Вхідні дані – HTML, CSS або JavaScript код; вихідні дані – результат компіляції введеного коду.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Методи створення колективного інформаційного середовища для організації процесів розробки програмного продукту.

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Коваленко О.О.

Оригінальність	
Схожість	

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Івасьов Олексій Станіславович

Керівник роботи _____ Коваленко Олена Олексіївна

Додаток В – Лістинг програми

Server.js

```
const express = require('express');
const app = express();
const http = require('http');
const path = require('path');
const { Server } = require('socket.io');
const ACTIONS = require('./src/Actions');

const server = http.createServer(app);
const io = new Server(server);

app.use(express.static('build'));
app.use((req, res, next) => {
    res.sendFile(path.join(__dirname, 'build', 'index.html'));
});

const userSocketMap = {};
function getAllConnectedClients(roomId) {
    // Map
    return Array.from(io.sockets.adapter.rooms.get(roomId) || []).map(
        (socketId) => {
            return {
                socketId,
                username: userSocketMap[socketId],
            };
        }
    );
}

io.on('connection', (socket) => {
    console.log('socket connected', socket.id);

    socket.on(ACTIONS.JOIN, ({ roomId, username }) => {
        userSocketMap[socket.id] = username;
```

```
        socket.join(roomId);
        const clients = getAllConnectedClients(roomId);
        clients.forEach(({ socketId }) => {
            io.to(socketId).emit(ACTIONS.JOINED, {
                clients,
                username,
                socketId: socket.id,
            });
        });
    });

    socket.on(ACTIONS.CODE_CHANGE, ({ roomId, code, editor }) => {
        socket.in(roomId).emit(ACTIONS.CODE_CHANGE, { code, editor });
    });

    socket.on(ACTIONS.SYNC_CODE, ({ socketId, code, editor }) => {
        io.to(socketId).emit(ACTIONS.CODE_CHANGE, { code, editor });
    });

    socket.on('disconnecting', () => {
        const rooms = [...socket.rooms];
        rooms.forEach((roomId) => {
            socket.in(roomId).emit(ACTIONS.DISCONNECTED, {
                socketId: socket.id,
                username: userSocketMap[socket.id],
            });
        });
        delete userSocketMap[socket.id];
        socket.leave();
    });
});

const PORT = process.env.PORT || 5000;
server.listen(PORT, () => console.log(`Listening on port ${PORT}`));
```

Editor.js

```
import React, { useEffect, useRef } from 'react';
import Codemirror from 'codemirror';
import 'codemirror/lib/codemirror.css';
import 'codemirror/theme/dracula.css';
import 'codemirror/mode/javascript/javascript';
import 'codemirror/mode/htmlmixed/htmlmixed';
import 'codemirror/addon/edit/closetag';
import 'codemirror/addon/edit/closebrackets';
import ACTIONS from '../Actions';
import toast from 'react-hot-toast';

const Editor = ({ socketRef, roomId, onCodeChange }) => {
  let html = useRef('');
  let css = useRef('');
  let js = useRef('');
  let srcDoc = useRef('');
  function runMarkup() {
    if (
      !document.getElementsByTagName('iframe')[0]?.contentWindow.document ||
      !document.getElementsByTagName('iframe')[0]?.contentDocument
    ) {
      return;
    } else if (
      document.getElementsByTagName('iframe')[0].contentWindow.document
        .readyState == 'complete'
    ) {
      srcDoc.current = `
      <html>
      <body>${html.current}</body>
      <style>${css.current}</style>
      </html>`;
      document.getElementsByTagName(
        'iframe'
```

```
    )[0].contentWindow.document.body.innerHTML = srcDoc.current;
  }
}

document.addEventListener('keyup', runMarkup);

function runScript() {
  let script = document.createElement('script');
  script.innerText = js.current;
  document
    .getElementsByTagName('iframe')[0]
    .contentWindow.document.body.appendChild(script);
  toast.success('Script executed successfully');
}
function createNewDocument() {
  let newWin = window.open('/output.html');
  newWin.onload = function () {
    newWin.document.body.innerHTML = srcDoc.current;
    let script = document.createElement('script');
    script.innerText = js.current;
    newWin.document.body.appendChild(script);
  };
}

const editorRef = useRef(null);
const htmlEditorRef = useRef(null);
const cssEditorRef = useRef(null);
useEffect(() => {
  async function init() {
    editorRef.current = Codemirror.fromTextArea(
      document.getElementById('editor'),
      {
        mode: { name: 'javascript', json: true },
        theme: 'dracula',
        autoCloseTags: true,
```

```
    autoCloseBrackets: true,
    lineNumbers: true,
  }
);
htmlEditorRef.current = Codemirror.fromTextArea(
  document.getElementById('htmlEditor'),
  {
    mode: { name: 'htmlmixed' },
    theme: 'dracula',
    autoCloseTags: true,
    autoCloseBrackets: true,
    lineNumbers: true,
  }
);
cssEditorRef.current = Codemirror.fromTextArea(
  document.getElementById('cssEditor'),
  {
    mode: { name: 'css' },
    theme: 'dracula',
    autoCloseBrackets: true,
    lineNumbers: true,
  }
);

editorRef.current.on('change', (instance, changes) => {
  const { origin } = changes;
  const code = instance.getValue();
  onCodeChange(code);
  localStorage.setItem('js', code);
  js.current = code;
  if (origin !== 'setValue') {
    socketRef.current.emit(ACTIONS.CODE_CHANGE, {
      roomId,
      code,
      editor: 'js',
```

```
        });
      }
    });
    htmlEditorRef.current.on('change', (instance, changes) => {
      const { origin } = changes;
      const code = instance.getValue();
      onCodeChange(code);
      localStorage.setItem('html', code);
      html.current = code;
      if (origin !== 'setValue') {
        socketRef.current.emit(ACTIONS.CODE_CHANGE, {
          roomId,
          code,
          editor: 'html',
        });
      }
    });
    cssEditorRef.current.on('change', (instance, changes) => {
      const { origin } = changes;
      const code = instance.getValue();
      onCodeChange(code);
      localStorage.setItem('css', code);
      css.current = code;
      if (origin !== 'setValue') {
        socketRef.current.emit(ACTIONS.CODE_CHANGE, {
          roomId,
          code,
          editor: 'css',
        });
      }
    });
  }
  init();
}, []);
```

```
useEffect(() => {
  if (socketRef.current) {
    socketRef.current.on(ACTIONS.CODE_CHANGE, ({ code, editor }) => {
      if (code !== null) {
        if (editor === 'js') {
          editorRef.current.setValue(code);
        } else if (editor === 'html') {
          htmlEditorRef.current.setValue(code);
          runMarkup();
        } else if (editor === 'css') {
          cssEditorRef.current.setValue(code);
          runMarkup();
        } else if (!editor) {
          if (localStorage.getItem('js'))
            editorRef.current.setValue(localStorage.getItem('js'));
          if (localStorage.getItem('html'))
            htmlEditorRef.current.setValue(localStorage.getItem('html'));
          if (localStorage.getItem('css'))
            cssEditorRef.current.setValue(localStorage.getItem('css'));
          runMarkup();
        }
      }
    });
  }

  return () => {
    socketRef.current.off(ACTIONS.CODE_CHANGE);
  };
}, [socketRef.current]);

return (
  <>
    <button className="btn newTabBtn" onClick={createNewDocument}>
      Open output in new tab
    </button>
```

```
      <button className="btn runBtn" onClick={runScript}>
        Run JS
      </button>
      <div className="editorWrap">
        <textarea id="editor"></textarea>
        <textarea id="htmlEditor"></textarea>
        <textarea id="cssEditor"></textarea>
        <iframe frameBorder="0" title="code" id="frame"></iframe>
      </div>
    </>
  );
};

export default Editor;
```

Client.js

```
import React from 'react';
import Avatar from 'react-avatar';

const Client = ({ username }) => {
    return (
        <div className="client">
            <Avatar name={username} size={50} round="14px" />
            <span className="userName">{username}</span>
        </div>
    );
};

export default Client;
```

EditorPage.js

```
import React, { useState, useRef, useEffect } from 'react';
```

```
import toast from 'react-hot-toast';
import ACTIONS from '../Actions';
import Client from '../components/Client';
import Editor from '../components/Editor';
import { initSocket } from '../socket';
import {
  useLocation,
  useNavigate,
  Navigate,
  useParams,
} from 'react-router-dom';

const EditorPage = () => {
  const socketRef = useRef(null);
  const codeRef = useRef(null);
  const location = useLocation();
  const { roomId } = useParams();
  const reactNavigator = useNavigate();
  const [clients, setClients] = useState([]);

  useEffect(() => {
    const init = async () => {
      socketRef.current = await initSocket();
      socketRef.current.on('connect_error', (err) => handleErrors(err));
      socketRef.current.on('connect_failed', (err) => handleErrors(err));

      function handleErrors(e) {
        console.log('socket error', e);
        toast.error('Socket connection failed, try again later.');
        reactNavigator('/');
      }

      socketRef.current.emit(ACTIONS.JOIN, {
        roomId,
        username: location.state?.username,
```

```
    });

    // Listening for joined event
    socketRef.current.on(
      ACTIONS.JOINED,
      ({ clients, username, socketId }) => {
        if (username !== location.state?.username) {
          toast.success(`${username} joined the room.`);
          console.log(`${username} joined`);
        }
        setClients(clients);
        socketRef.current.emit(ACTIONS.SYNC_CODE, {
          code: codeRef.current,
          socketId,
        });
      }
    );

    // Listening for disconnected
    socketRef.current.on(ACTIONS.DISCONNECTED, ({ socketId, username }) => {
      toast.success(`${username} left the room.`);
      setClients((prev) => {
        return prev.filter((client) => client.socketId !== socketId);
      });
    });
  };
  init();
  return () => {
    socketRef.current.disconnect();
    localStorage.clear();
    socketRef.current.off(ACTIONS.JOINED);
    socketRef.current.off(ACTIONS.DISCONNECTED);
  };
}, []);
```

```
async function copyRoomId() {
  try {
    await navigator.clipboard.writeText(roomId);
    toast.success('Room ID has been copied to your clipboard');
  } catch (err) {
    toast.error('Could not copy the Room ID');
    console.error(err);
  }
}

function leaveRoom() {
  reactNavigator('/');
}

if (!location.state) {
  return <Navigate to="/" />;
}

return (
  <div className="mainWrap">
    <div className="aside">
      <div className="asideInner">
        <div className="logo">
          <img className="logoImage" src="/logo.png" alt="logo" />
          <p className='logo__header'>Code Editor</p>
        </div>
        <h3>Connected</h3>
        <div className="clientsList">
          {clients.map((client) => (
            <Client key={client.socketId} username={client.username} />
          ))}
        </div>
      </div>
      <button className="btn copyBtn" onClick={copyRoomId}>
        Copy room ID
```

```
        </button>
        <button className="btn leaveBtn" onClick={leaveRoom}>
          Leave
        </button>
      </div>
      <div className="editor-name editor-name-js">Javascript</div>
      <Editor
        socketRef={socketRef}
        roomId={roomId}
        onCodeChange={(code) => {
          codeRef.current = code;
        }}
      />
      <div className="editor-name editor-name-html">Html</div>

      <div className="editor-name editor-name-css">Css</div>
    </div>
  );
};

export default EditorPage;
```

App.js

```
import './App.css';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { Toaster } from 'react-hot-toast';
import Home from './pages/Home';
import EditorPage from './pages/EditorPage';

function App() {
    return (
        <>
            <div>
                <Toaster
```

```
                position="top-right"
                toastOptions={{
                    success: {
                        theme: {
                            primary: '#4aed88',
                        },
                    },
                }}
            ></Toaster>
        </div>
        <BrowserRouter>
            <Routes>
                <Route path="/" element={<Home />}></Route>
                <Route
                    path="/editor/:roomId"
                    element={<EditorPage />}
                ></Route>
            </Routes>
        </BrowserRouter>
    </>
  );
}

export default App;
```

socket.js

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

export const initSocket = async () => {
    const options = {
        'force new connection': true,
        reconnectionAttempt: 'Infinity',
        timeout: 10000,
        transports: ['websocket'],
```

```
    };
    return io(process.env.REACT_APP_BACKEND_URL, options);
};
```

EditorPage.js

```
import React, { useState, useRef, useEffect } from 'react';
import toast from 'react-hot-toast';
import ACTIONS from '../Actions';
import Client from '../components/Client';
import Editor from '../components/Editor';
import { initSocket } from '../socket';
import {
  useLocation,
  useNavigate,
  Navigate,
  useParams,
} from 'react-router-dom';

const EditorPage = () => {
  const socketRef = useRef(null);
  const codeRef = useRef(null);
  const location = useLocation();
  const { roomId } = useParams();
  const reactNavigator = useNavigate();
  const [clients, setClients] = useState([]);

  useEffect(() => {
    const init = async () => {
      socketRef.current = await initSocket();
      socketRef.current.on('connect_error', (err) => handleErrors(err));
      socketRef.current.on('connect_failed', (err) => handleErrors(err));

      function handleErrors(e) {
        console.log('socket error', e);
```

```
      toast.error('Socket connection failed, try again later.');
      reactNavigator('/');
    }

    socketRef.current.emit(ACTIONS.JOIN, {
      roomId,
      username: location.state?.username,
    });

    // Listening for joined event
    socketRef.current.on(
      ACTIONS.JOINED,
      ({ clients, username, socketId }) => {
        if (username !== location.state?.username) {
          toast.success(`${username} joined the room.`);
          console.log(`${username} joined`);
        }
        setClients(clients);
        socketRef.current.emit(ACTIONS.SYNC_CODE, {
          code: codeRef.current,
          socketId,
        });
      }
    );

    // Listening for disconnected
    socketRef.current.on(ACTIONS.DISCONNECTED, ({ socketId, username }) => {
      toast.success(`${username} left the room.`);
      setClients((prev) => {
        return prev.filter((client) => client.socketId !== socketId);
      });
    });
  };
  init();
  return () => {
```

```
    socketRef.current.disconnect();
    localStorage.clear();
    socketRef.current.off(ACTIONS.JOINED);
    socketRef.current.off(ACTIONS.DISCONNECTED);
  };
}, []);

async function copyRoomId() {
  try {
    await navigator.clipboard.writeText(roomId);
    toast.success('Room ID has been copied to your clipboard');
  } catch (err) {
    toast.error('Could not copy the Room ID');
    console.error(err);
  }
}

function leaveRoom() {
  reactNavigator('/');
}

if (!location.state) {
  return <Navigate to="/" />;
}

return (
  <div className="mainWrap">
    <div className="aside">
      <div className="asideInner">
        <div className="logo">
          <img className="logoImage" src="/logo.png" alt="logo" />
          <p className='logo__header'>Code Editor</p>
        </div>
        <h3>Connected</h3>
        <div className="clientsList">
```

```
            {clients.map((client) => (
              <Client key={client.socketId} username={client.username} />
            ))}
          </div>
        </div>
        <button className="btn copyBtn" onClick={copyRoomId}>
          Copy room ID
        </button>
        <button className="btn leaveBtn" onClick={leaveRoom}>
          Leave
        </button>
      </div>
      <div className="editor-name editor-name-js">Javascript</div>
      <Editor
        socketRef={socketRef}
        roomId={roomId}
        onCodeChange={(code) => {
          codeRef.current = code;
        }}
      />
      <div className="editor-name editor-name-html">Html</div>

      <div className="editor-name editor-name-css">Css</div>
    </div>
  );
};

export default EditorPage;
```

Client.js

```
import React from 'react';
import Avatar from 'react-avatar';

const Client = ({ username }) => {
```

```
    return (
        <div className="client">
            <Avatar name={username} size={50} round="14px" />
            <span className="userName">{username}</span>
        </div>
    );
};

export default Client;
```

App.js

```
import './App.css';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { Toaster } from 'react-hot-toast';
import Home from './pages/Home';
import EditorPage from './pages/EditorPage';

function App() {
    return (
        <>
            <div>
                <Toaster
                    position="top-right"
                    toastOptions={{
                        success: {
                            theme: {
                                primary: '#4aed88',
                            },
                        },
                    }}
                ></Toaster>
            </div>
            <BrowserRouter>
                <Routes>
```

```
                <Route path="/" element={<Home />}></Route>
                <Route
                  path="/editor/:roomId"
                  element={<EditorPage />}
                ></Route>
              </Routes>
            </BrowserRouter>
          </>
    );
}

export default App;
```

Home.js

```
import React, { useState } from 'react';
import { v4 as uuidV4 } from 'uuid';
import toast from 'react-hot-toast';
import { useNavigate } from 'react-router-dom';

const Home = () => {
  const navigate = useNavigate();

  const [roomId, setRoomId] = useState('');
  const [username, setUsername] = useState('');
  const createNewRoom = (e) => {
    e.preventDefault();
    const id = uuidV4();
    setRoomId(id);
    toast.success('Created a new room');
  };

  const joinRoom = () => {
    if (!roomId || !username) {
```

```
    toast.error('ROOM ID & username is required');
    return;
  }

  // Redirect
  navigate(`/editor/${roomId}`, {
    state: {
      username,
    },
  });
};

const handleInputEnter = (e) => {
  if (e.code === 'Enter') {
    joinRoom();
  }
};
return (
  <div className="homePageWrapper">
    <div className="formWrapper">
      <div className="logoWrapper">
        <img className="homePageLogo" src="/logo.png" alt="logo" />
        <p className='home__header'>Code editor</p>
      </div>
      <h4 className="mainLabel">Paste invitation room ID</h4>
      <div className="inputGroup">
        <input
          type="text"
          className="inputBox"
          placeholder="Room ID"
          onChange={(e) => setRoomId(e.target.value)}
          value={roomId}
          onKeyUp={handleInputEnter}
        />
        <input
```

```
          type="text"
          className="inputBox"
          placeholder="Username"
          onChange={(e) => setUsername(e.target.value)}
          value={username}
          onKeyUp={handleInputEnter}
        />
        <button className="btn joinBtn" onClick={joinRoom}>
          Join
        </button>
        <span className="createInfo">
          Or create  
          <a onClick={createNewRoom} href="" className="createNewBtn">
            new room
          </a>{' '}
          if you don't have an invitation
        </span>
      </div>
    </div>
    <footer>
      <p>Built by Oleksii Ivasov as undergraduate work </p>
    </footer>
  </div>
 );
};

export default Home;
```

App.css

```
.homePageWrapper {
 display: flex;
 align-items: center;
 justify-content: center;
```

```
  color: #fff;
  height: 100vh;
}

.formWrapper {
  background: #282a36;
  padding: 20px;
  padding-top: 10px;
  border-radius: 10px;
  width: 400px;
  max-width: 90%;
}

footer {
  position: fixed;
  bottom: 0;
}

footer a {
  color: #4aee88;
}

.inputGroup {
  display: flex;
  flex-direction: column;
}

.mainLabel {
  margin-bottom: 20px;
  margin-top: 0;
  font-weight: 400;
}

.logoWrapper {
  display: flex;
```

```
  align-items: center;
  margin-bottom: 15px;
}

.homePageLogo {
  height: 70px;
  margin-right: 15px;
  margin-left: 40px;
}

.inputBox {
  padding: 10px;
  border-radius: 5px;
  outline: none;
  border: none;
  margin-bottom: 14px;
  background: #eee;
  font-size: 16px;
  font-weight: bold;
}

.btn {
  border: none;
  padding: 10px;
  border-radius: 5px;
  font-size: 16px;
  font-weight: bold;
  cursor: pointer;
  transition: all 0.3s ease-in-out;
}

.joinBtn,
.leaveBtn {
  background: #4aed88;
  width: 100px;
```

```
  margin-left: auto;
}

.joinBtn:hover,
.leaveBtn:hover {
  background: #2b824c;
}

.createInfo {
  margin: 0 auto;
  margin-top: 20px;
}

.createNewBtn {
  color: #4aed88;
  text-decoration: none;
  border-bottom: 1px solid #4aed88;
  transition: all 0.3s ease-in-out;
}

.createNewBtn:hover,
footer a:hover {
  color: #368654;
  border-color: #368654;
}

.mainWrap {
  display: grid;
  grid-template-columns: 230px 1fr;
}

.aside {
  background: #1c1e29;
  padding: 16px;
  color: #fff;
```

```
  display: flex;
  flex-direction: column;
}
.asideInner {
  flex: 1;
}

.clientsList {
  display: flex;
  align-items: center;
  flex-wrap: wrap;
  gap: 20px;
}

.client {
  display: flex;
  align-items: center;
  flex-direction: column;
  font-weight: bold;
}
.userName {
  margin-top: 10px;
}

.logo {
  border-bottom: 1px solid #424242;
  padding-bottom: 10px;
  display: flex;
  align-items: center;
}

.home__header {
  line-height: 1;
  font-size: 32px;
}
```

```
.logo__header {
  font-size:22px;
}

.logoImage {
  height: 50px;
  margin-right: 10px;
}

.leaveBtn {
  width: 100%;
  margin-top: 20px;
}
.CodeMirror {
  min-height: calc(50vh - 42px);
  font-size: 20px;
  line-height: 1.6;
  padding-top: 20px;
  margin-top: 20px;
}

.editorWrap {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 1fr 1fr;
}

#frame {
    width: 99.4%;
    height: 100%;
    background-color: #fff;
    color:#000;
}

#editor {
```

```
    display: none;
}

.editor-name {
    position: absolute;
    top:0;
    z-index: 1;
    background-color: #1c1e29;
    width:42.5%;
    height: 20px;
    text-align: center;
    color: #fff;
}

.editor-name-js {
    left: 232px;
}

.editor-name-html {
    right: 0;
}

.editor-name-css {
    left: 232px;
    top: 359px;
}

.runBtn,.newTabBtn {
  position: absolute;
  z-index: 1;
  left: 16px;
  bottom:137px;
  width:198px;
}
```

```
.newTabBtn {
  bottom: 195px;
}

.copyBtn:hover,
.runBtn:hover,
.newTabBtn:hover {
  background-color: rgb(148, 148, 148);
}
```

Додаток Г. – Графічна частина

ГРАФІЧНА ЧАСТИНА
МЕТОДИ СТВОРЕННЯ КОЛЕКТИВНОГО ІНФОРМАЦІЙНОГО СЕРЕДОВИЩА ДЛЯ ОРГАНІЗАЦІЇ ПРОЦЕСІВ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

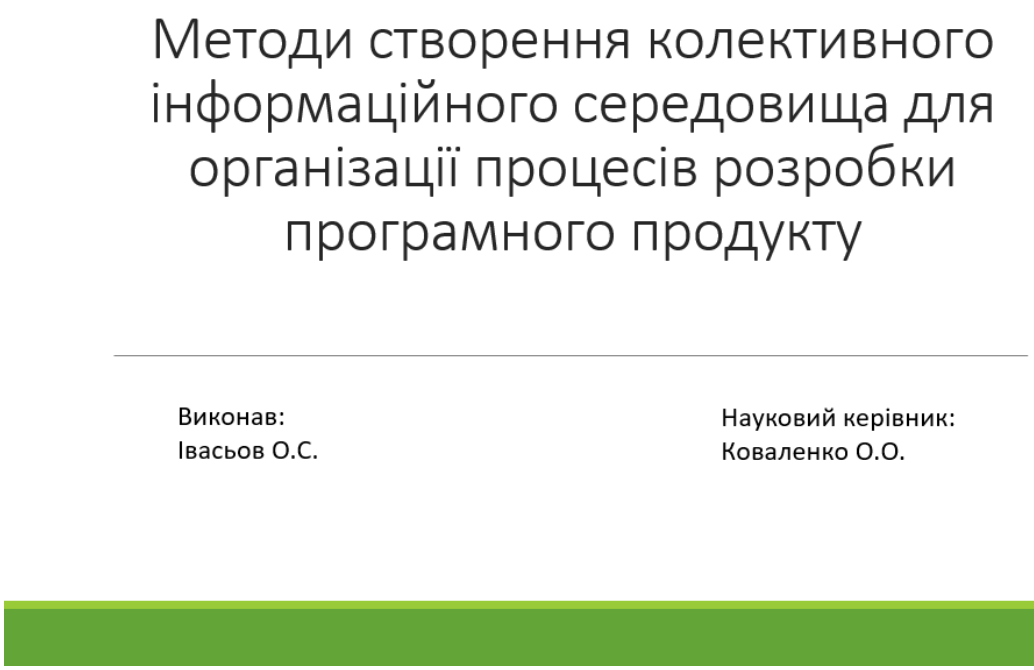


Рисунок Г.1 – Назва роботи

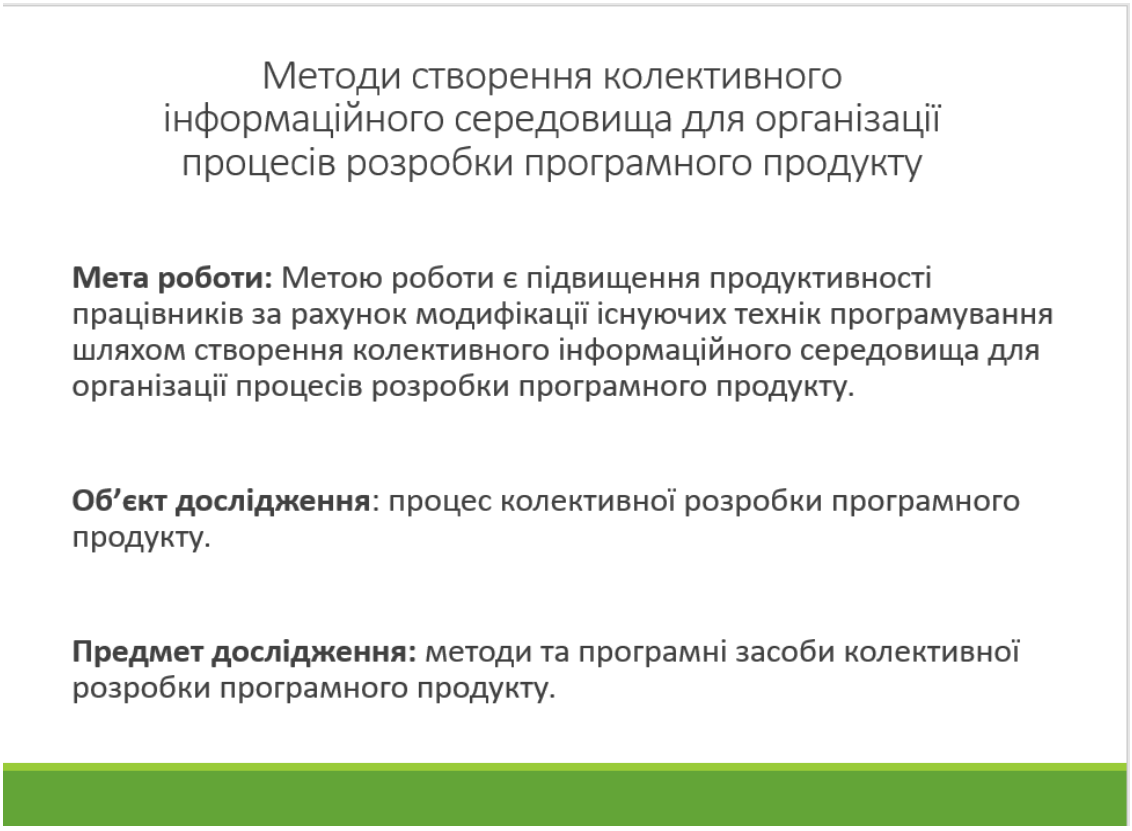


Рисунок Г.2 – Мета, об'єкт і предмет дослідження

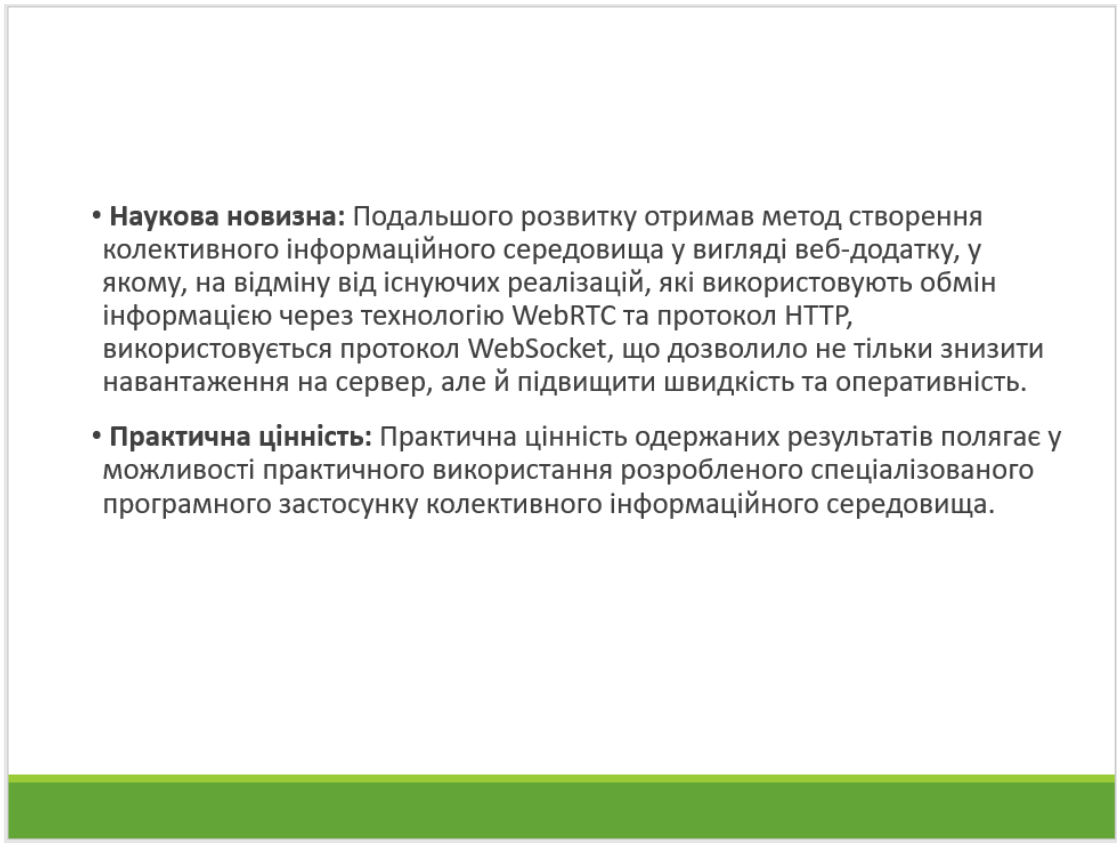


Рисунок Г.3 – Наукова новизна та практична цінність

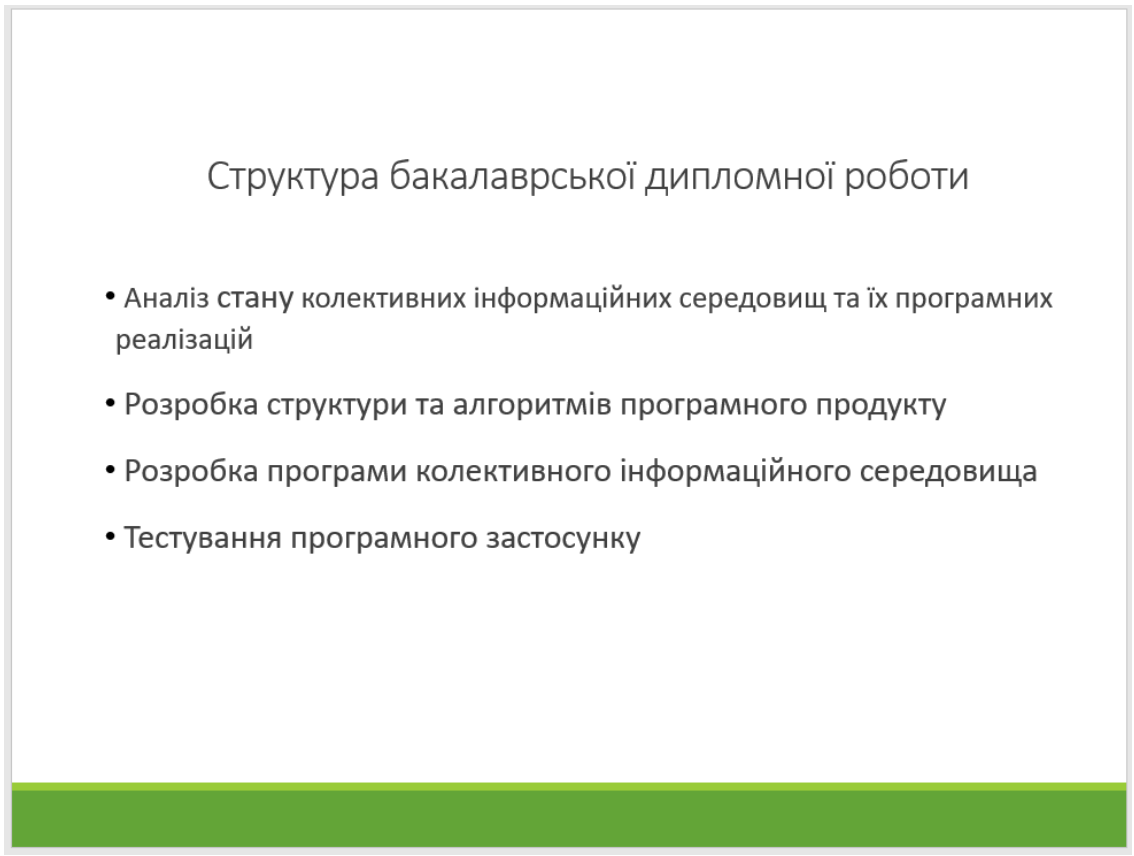
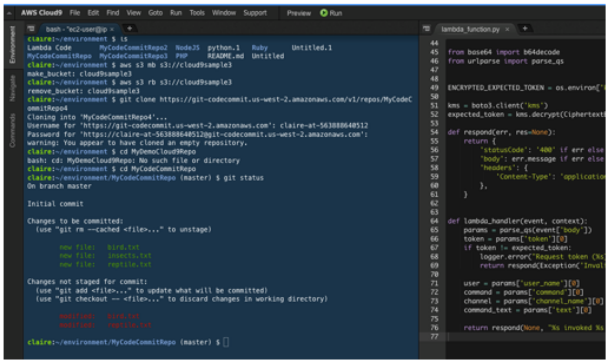


Рисунок Г.4 – Структура бакалаврської дипломної роботи

Аналоги

- Teletype for Atom
- Codepen
- AWS Cloud9
- Remote Collab for Sublime Text



AWS Cloud 9

Рисунок Г.5 – Аналоги

Порівняння аналогів

Критерій	Teletype	Codepen	AWS Cloud9	Remote Collab	Власний додаток
Безкоштовне використання	+	-	-	+	+
Відсутня необхідність завантаження стороннього ПЗ	-	+	-	-	+
Необмежена кількість працівників над одним проєктом	-	+	-	-	+
Відсутність використання додаткових апаратних ресурсів	-	+	-	-	+
Відсутність реєєстрації	-	-	-	-	+
Підсумковий результат	1	3	0	1	5

Рисунок Г.6 – Таблиця порівняння аналогів

Рисунок Г.7 – Алгоритм та модель авторизації користувача в системі

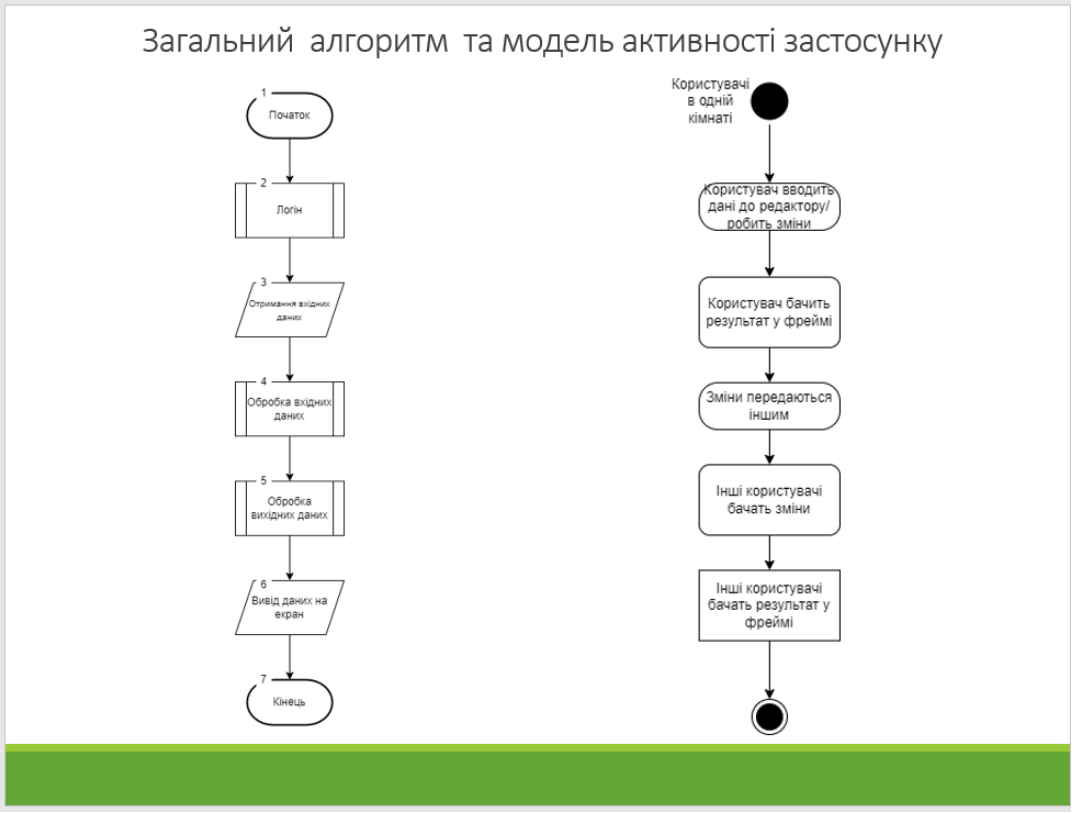


Рисунок Г.8– Загальний алгоритм та модель активності застосунку

Рисунок Г.9– Алгоритми роботи з вхідними та вихідними даними

Рисунок Г.10 – Схема трансакційної моделі для групової комунікації

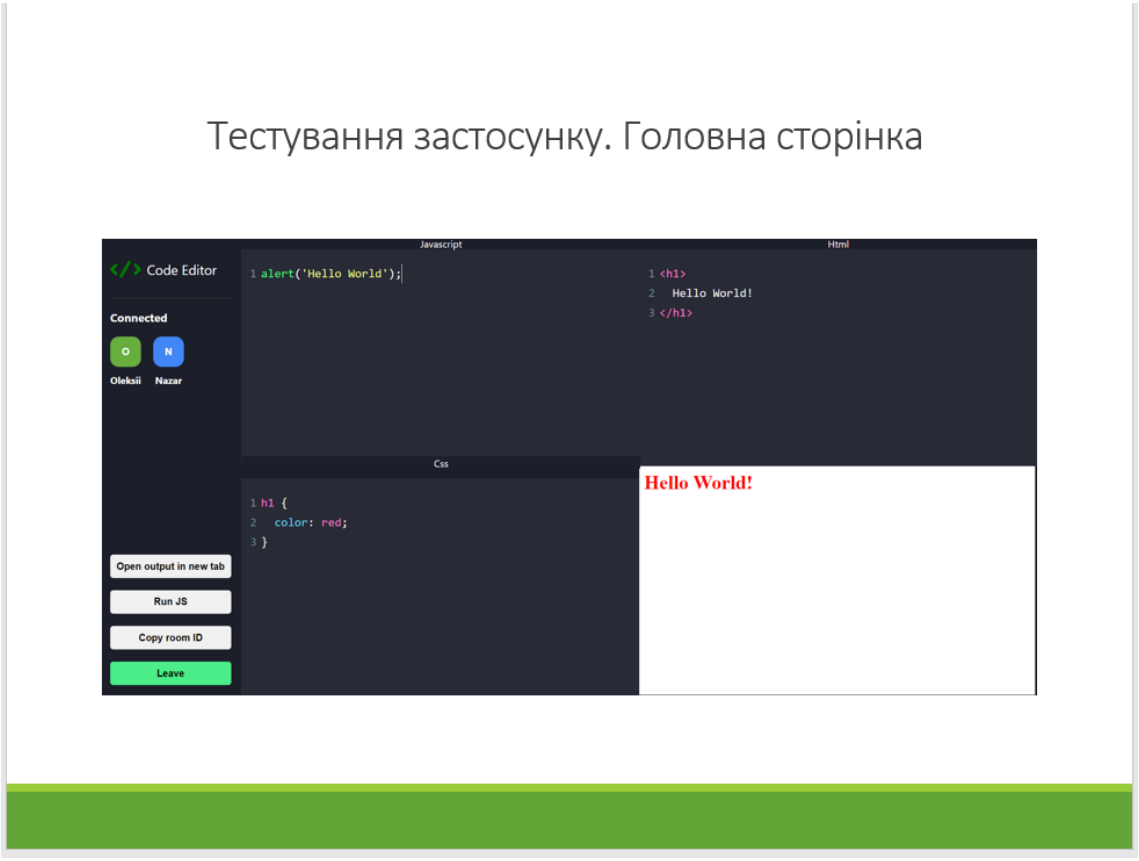


Рисунок Г.11 – Тестування головної сторінки застосунку

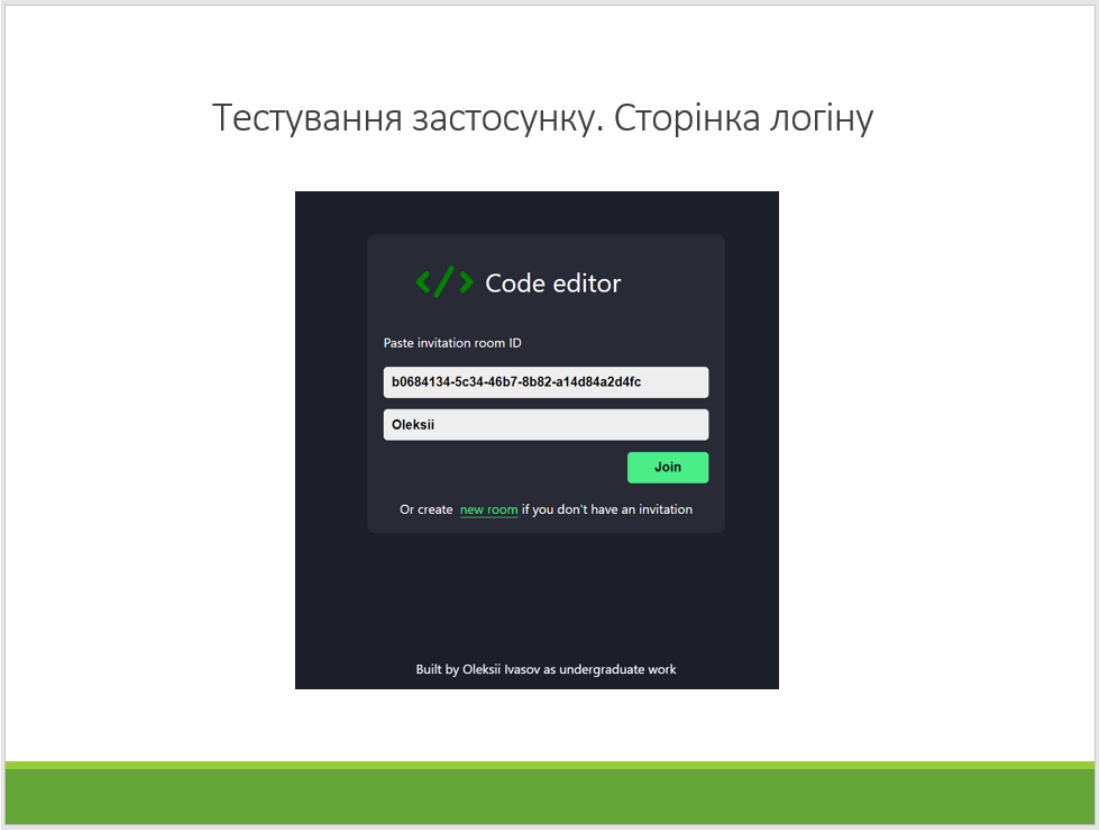


Рисунок Г.12 – Тестування сторінки авторизації застосунку

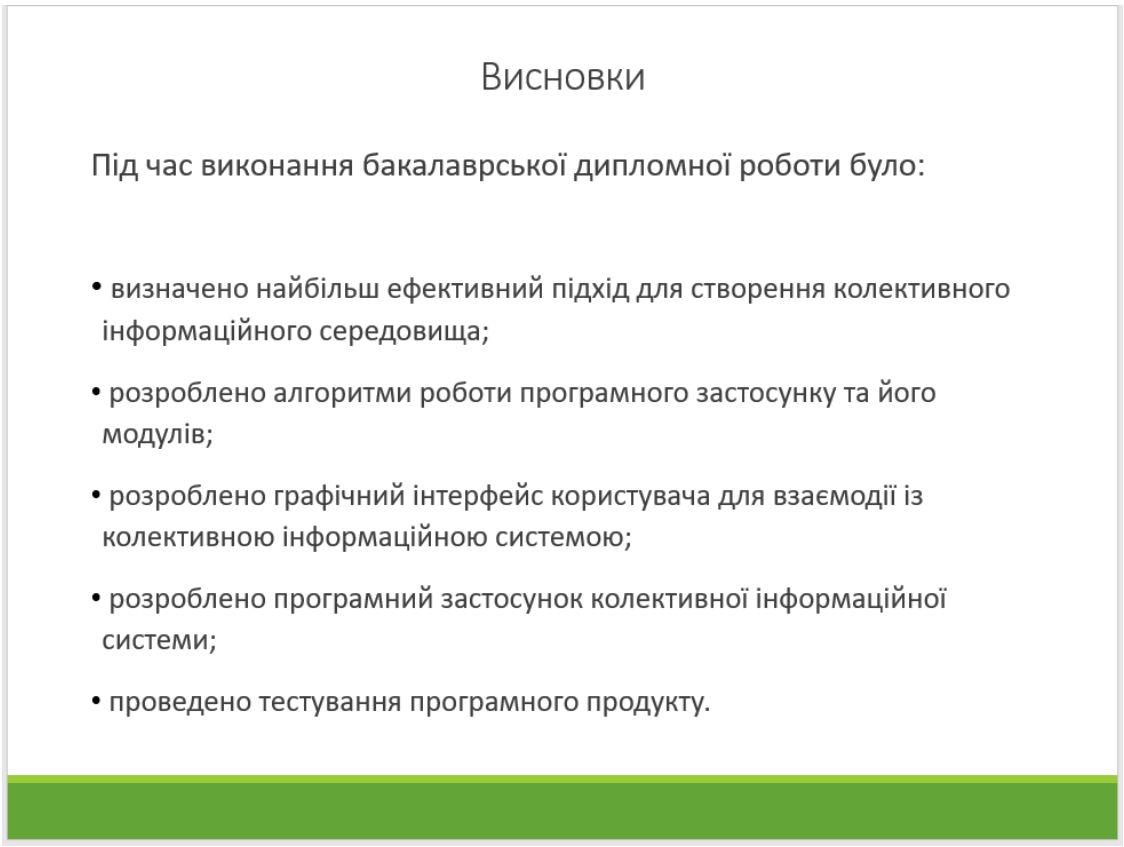


Рисунок Г.13 – Висновки

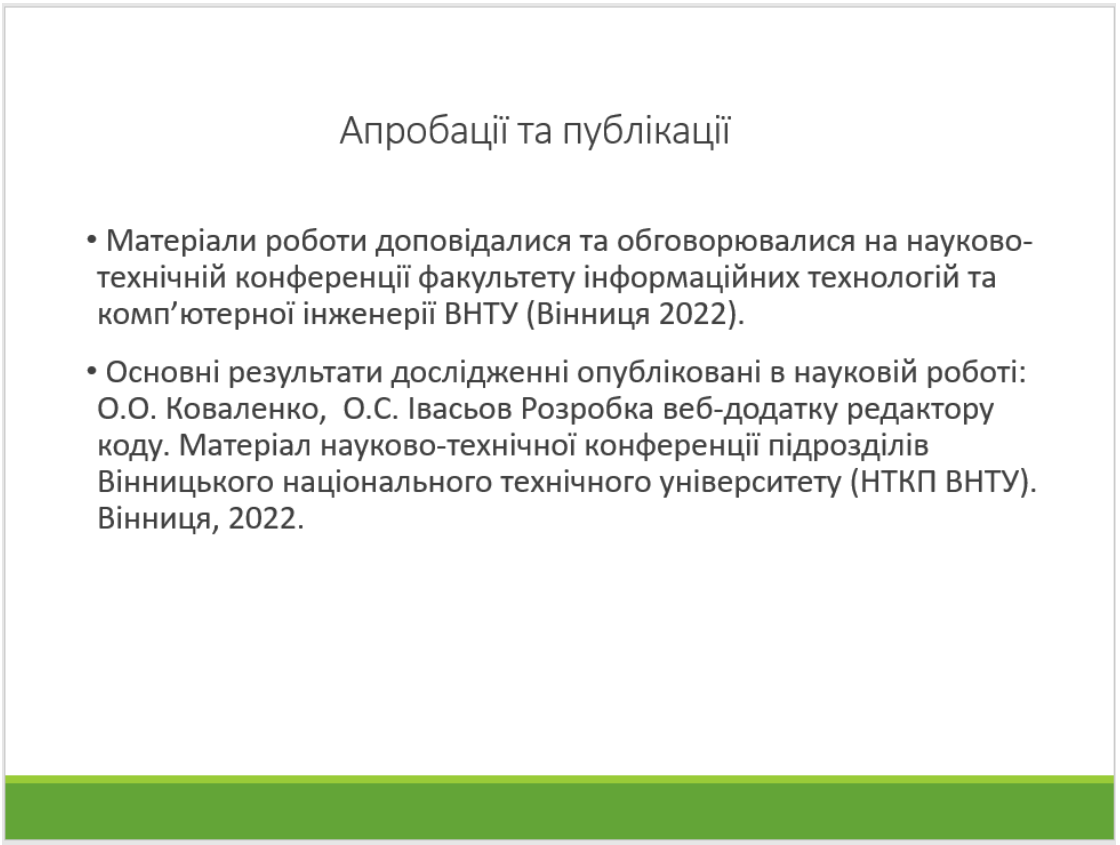


Рисунок Г.14 – Апробації та публікації