

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОМУНІКАЦІЇ
МІЖ КОРИСТУВАЧАМИ
SOFTWARE FOR USER-TO-USER COMMUNICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-121сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БЕЛИМЕНКО Д.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ОЛІЙНИК А.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БЕЛИМЕНКА Дмитра Романовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для комунікації між користувачами.
Software for User-to-User Communication

керівник проєкту (роботи) д.т.н., професор ОЛІЙНИК Андрій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми.
4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “15” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Дмитро БЕЛИМЕНКО

(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Андрій ОЛІЙНИК

(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
83 с., 4 табл., 39 рис., 3 дод., 15 джерел.

JAVASCRIPT, NEST.JS, VUE.JS, DOCKER, БАЗА ДАНИХ,
ВЕБЗАСТОСУНОК, КОМУНІКАЦІЇ, САЙТ.

Об'єкт дослідження – процес передачі потокового аудіо/відео між користувачами.

Предмет дослідження – програмні засоби для передачі потокового аудіо/відео між користувачами.

Метою роботи є розробка програмного забезпечення для передачі потокового аудіо/відео між користувачами з можливістю швидкого та легкого встановлення на власний сервер з використання платформи Docker.

Матеріали, методи та технічні засоби: мова програмування JavaScript, фреймворки Nest.js та Vue.js, платформа для розробки додатків Docker.

Результати. Розроблено програмне забезпечення для передачі потокового аудіо/відео між користувачами з можливістю швидкого та легкого встановлення на власний сервер за допомогою мови програмування JavaScript та її фреймворків Vue.js та Nest.js, що дозволяє забезпечити користувачам комунікувати між собою, а адміністраторам бачити статистику системи.

Практична цінність роботи полягає у розробці програмного забезпечення для комунікації між користувачами.

Висновки. Виконано проєктування програмного забезпечення для забезпечення для комунікації між користувачами. Розроблено програмне забезпечення для комунікації між користувачами. Здійснено тестування розробленого програмного забезпечення для комунікації між користувачами.

Галузь використання – програмні засоби для комунікації.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 83 pages, 4 tables, 39 figures, 3 appendixes, 15 sources.

JAVASCRIPT, NEST.JS, VUE.JS, DOCKER, DATABASE, WEB APPLICATION, COMMUNICATIONS, WEBSITE.

The object of research is the process of transfer of streaming audio/video between users.

Subject of research - software tools for streaming audio/video between users.

The aim of the work is to develop software for streaming audio/video between users with the ability to quickly and easily install on your own server using the Docker platform.

Materials, methods and techniques: JavaScript programming language, Nest.js and Vue.js frameworks, Docker application development platform.

Results. The software for streaming audio/video between users with the ability to quickly and easily install on your own server using the JavaScript programming language and its frameworks Vue.js and Nest.js, which allows users to communicate with each other and administrators to see system statistics.

The practical value of the work is the development of software for communication between users.

Conclusions. The design of software for communication between users has been completed. Software for communication between users has been developed. The developed software for communication between users was tested.

The field of application is software for communication.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Поняття про WebRTC та програмні засоби для комунікації	11
1.2 Дослідження програмних засобів для комунікації між користувачами .	11
1.3 Висновки за розділом 1	17
2 Матеріали і методи.....	18
2.1 Вибір мови програмування та інших технологій розробки	18
2.2 Вибір фреймворків та інструментів для створення програмного забезпечення для підтримки комунікації між користувачами	20
2.3 Висновки за розділом 2	23
3 Опис програми	24
3.1 Структурна схема програми для комунікації між користувачами.....	24
3.2 Функціонування програмного забезпечення підтримки комунікації	25
3.3 Проектування інтерфейсу взаємодії користувача з програмним забезпеченням для підтримки комунікації між користувачами.....	28
3.4 Висновки за розділом 3	32
4 Експлуатація, тестування та експериментальне дослідження програми.....	33
4.1 Призначення й умови застосування програми	33
4.2 Характеристики програми та розгортання вебсерверу.....	34
4.3 Інструкція по експлуатації програми.....	36
4.3.1 Звернення до програми.....	36
4.3.2 Вхідні й вихідні дані	36
4.3.3 Повідомлення	36
4.4 Виконання програмного забезпечення для підтримки комунікації між користувачами	36
4.5 Тестування програмного забезпечення для підтримки комунікації між користувачами	38

4.6 Висновки за розділом 4	39
Висновки	40
Перелік джерел посилань	42
Додаток А Текст програми.....	44
Додаток Б Фрагмент тексту програми	48
Додаток В Слайди програми.....	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

HTML – HyperText Markup Language;

JS – JavaScript;

БД – база даних;

ПЗ – програмне забезпечення;

ТЗ – технічне завдання.

ВСТУП

Розробка програм для комунікації в цифровому форматі є важливим аспектом сучасного інтернет-середовища. Багато існуючих месенджерів та платформ пропонують різноманітні можливості, включаючи , відео та аудіо-повідомлень, групові чати, віртуальні стікери, та інші функції. Однак, часто ці програми навантажені зайвими функціями, які можуть збентежити користувача та ускладнювати взаємодію. Крім того, у зв'язку із тим, що популярні платформи не дають розгорнутися на власному сервері у організацій не має можливості контролювати систему та продивлятися статистику.

Отже, актуальною стає розробка програмного забезпечення для комунікації, яке відповідає сучасним вимогам та уникне зайвих функцій. Важливо створити програму, яка буде спрощеною у використанні, ефективною та забезпечує можливість власникам більше контролю над системою та дасть змогу зібрати статистику.

Розроблене програмне забезпечення доцільно реалізувати у вигляді вебзастосунку, що є розподіленим вебдодатком, де браузер виконує роль клієнта, а вебсервер – роль сервера. Логіка розробленого вебзастосунку буде зосереджена на сервері, а відображення інформації, яка завантажена з сервера, та передача даних від користувача буде організована за допомогою веббраузера, що забезпечить міжплатформову взаємодію та, відповідно, незалежність від конкретної операційної системи користувача.

У дипломній кваліфікаційній роботі бакалавра розв'язується актуальне завдання розробки програмного забезпечення для підтримки комунікації між користувачами.

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для підтримки комунікації між користувачами;

- здійснити проєктування програмного забезпечення для підтримки комунікації між користувачами;
- створити програмне забезпечення для підтримки комунікації між користувачами;
- виконати тестування розробленого програмного забезпечення для підтримки комунікації між користувачами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття про WebRTC та програмні засоби для комунікації

WebRTC (Web Real Time Communications) – це стандарт, який забезпечує однорангову комунікацію в режимі реального часу та обмін медіа-даними в браузерях, усуваючи необхідність завантажувати та встановлювати додаткові програми або надбудови. Завдяки WebRTC будь-який браузер може виступати в ролі кінцевої точки відеоконференції: вам потрібно лише відкрити вебсторінку наради, щоб розпочати відеоконференцію [1].

Основні компоненти WebRTC:

- getUserMedia API: Дозволяє отримувати доступ до відео та аудіо з камери та мікрофона користувача;
- RTCPeerConnection: Відповідає за установлення та управління підключенням між двома або більше пристроями для обміну потоками відео, аудіо та даними;
- RTCDataChannel: Надає можливість обміну даними в реальному часі між підключеними пристроями;
- RTCSessionDescription і RTCIceCandidate: Використовуються для обміну інформацією про сесію та кандидатів для встановлення підключення.

1.2 Дослідження програмних засобів для комунікації між користувачами

Зараз існує значна кількість програм та вебсайтів для комунікації між користувачами. Проведемо огляд певних програм та вебсайтів цієї галузі.

Програмне забезпечення google meet [2] є вебсайтом для відеоконференції з участю різної кількості учасників. Зручний інтерфейс сприяє ефективному спілкуванню та співпраці в реальному часі. Один із сильних боків Google Meet – це його простий та зрозумілий інтерфейс, який робить комунікацію доступною для користувачів будь-якого рівня технічної

підготовки. Це важливий аспект, оскільки дозволяє швидко адаптуватися і використовувати інструмент без значних зусиль.

На рис. 1.1 наведено користувацький інтерфейс вебсайту Google meet.

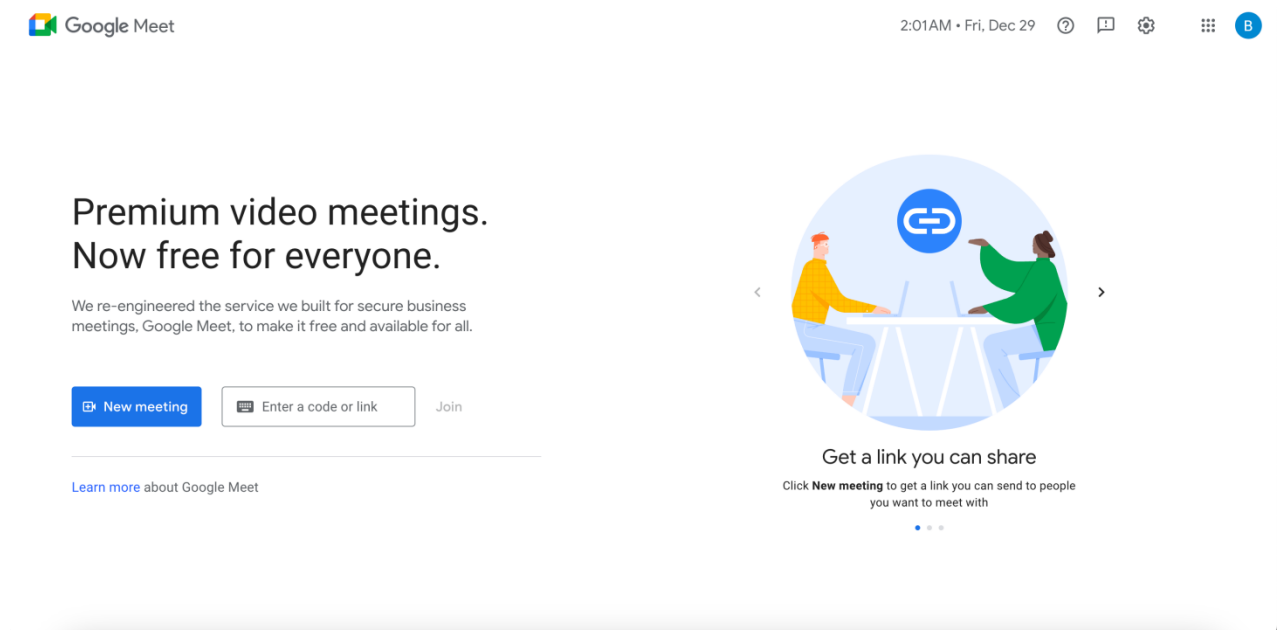


Рисунок 1.1 – Програмне забезпечення Google meet [2]

Серед особливостей даного сервісу, що надає можливості обміном зображеннями, можна виділити такі:

- обмін зображеннями та контентом екрана;
- використання віртуальних фонів та фільтрів;
- авторизація через акаунт google;
- наявність версій мобільного застосунку для різних платформ.

Серед недоліків програмного забезпечення Google meet виділяється Обмежена тривалість відеоконференцій: У безкоштовній версії Google Meet може бути обмежена максимальна тривалість відеоконференцій. Це може стати проблемою для користувачів, які ведуть довгі зустрічі.

Програмне забезпечення Skype [3] функціонує як відмінний інструмент для відеоконференцій та обміну повідомленнями, дозволяючи користувачам взаємодіяти в режимі реального часу. Skype дозволяє користувачам

обмінюватися не лише текстовими повідомленнями, а й зображеннями, документами.

На рис. 1.2 наведено користувацький інтерфейс вебсайту Skype.

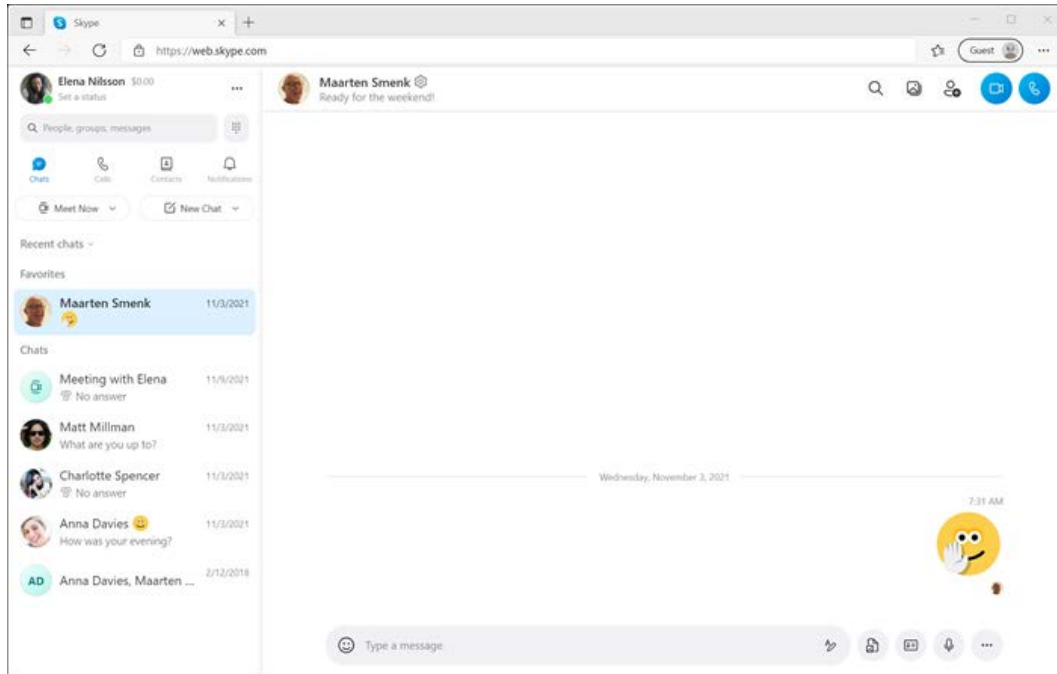


Рисунок 1.2 – Програмне забезпечення Skype [3]

Серед особливостей Skype можна виділити такі:

- обмін зображеннями та контекстом екрана;
- наявність версій мобільного застосунку для різних платформ;
- можливість додавання тексту на зображення;
- переклад чатів;

Серед недоліків програмного забезпечення skype виділяється те, що технічні параметри, обмежують і кількість користувачів. Інколи програма висне, голос відстає від зображення, а саме зображення часто втрачає чіткість.

Програмне забезпечення Slack [4] для обміну повідомленнями, інструментами та файлами. Slack можна розглядати, як платформу обміну миттєвими повідомленнями, але з інтегрованим доступом до файлів, відео- та телефонних дзвінків.

На рис. 1.3 наведено користувацький інтерфейс вебсайту Slack.

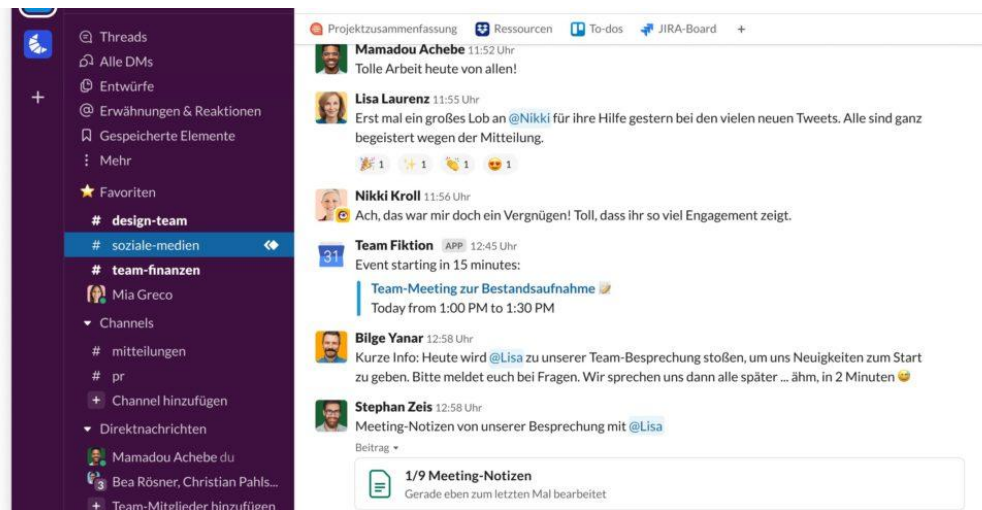


Рисунок 1.3 – Програмне забезпечення Slack [4]

Серед особливостей Slack можна виділити такі:

- обмін зображеннями та контекстом екрана;
- наявність версій мобільного застосунку для різних платформ;
- slack інтегрується з низкою програм, які ви вже використовуєте, таких як Google Doc, Jira та Salesforce.

Серед недоліків програмного забезпечення slack виділяється те, що у безкоштовній версії Slack є обмежені можливості щодо зберігання повідомлень та інтеграції з іншими сервісами. Деякі корисні функції доступні тільки у платних планах.

Програмне забезпечення WhatsApp [5] забезпечення для обміну миттєвими повідомленнями, яке також надає інструменти для обміну файлами, відео- та голосовими повідомленнями. WhatsApp визначається як мобільна платформа для комунікації, де користувачі можуть обмінюватися текстовими повідомленнями, а також зручно передавати файли та використовувати відео- і голосові виклики для спілкування з контактами. Ця платформа створена для швидкого та ефективного обміну інформацією, а також для зручного спілкування через віртуальні канали [11].

На рис. 1.4 наведено користувацький інтерфейс вебсайту WhatsApp.

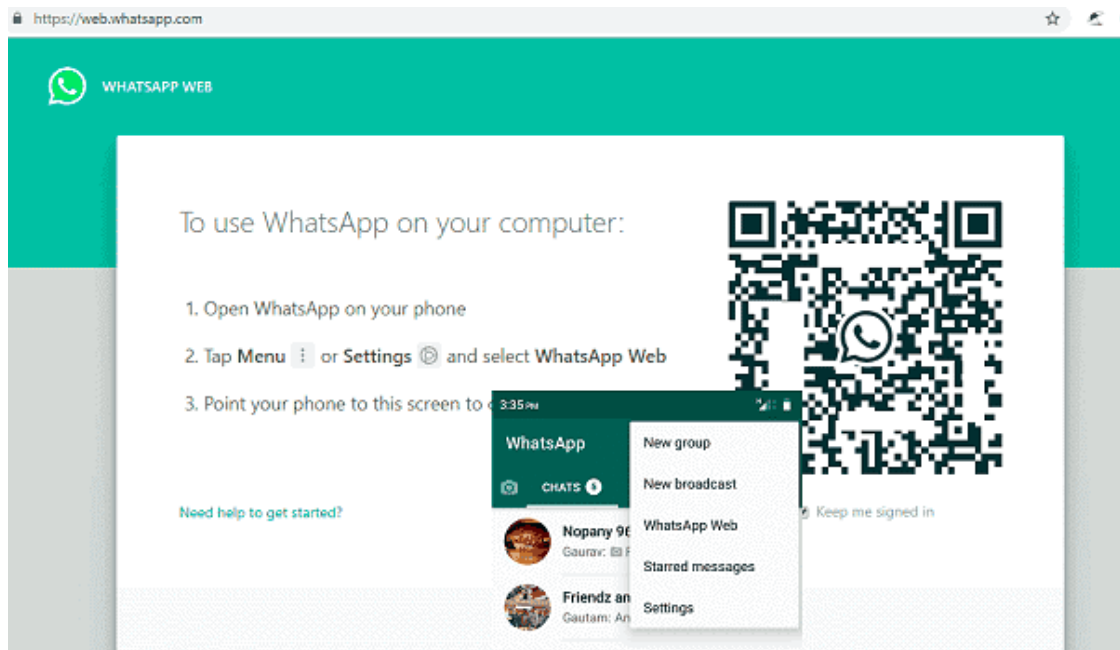


Рисунок 1.4 – Програмне забезпечення WhatsApp [5]

Серед особливостей WhatsApp можна виділити такі:

- наявність версій мобільного застосунку для різних платформ;
- WhatsApp використовує end-to-end шифрування для захисту конфіденційності повідомлень. Це означає, що тільки відправник та отримувач можуть переглядати вміст повідомлення, що забезпечує високий рівень безпеки;
- обмін зображеннями та контекстом екрана.

Серед недоліків програмного забезпечення WhatsApp виділяється те, що у безкоштовній версії відсутні деякі розширені функції, які можуть бути доступні у платних аналогах. Наприклад, відсутній функціонал для бізнес-користувачів, такий як інструменти для спільної роботи над проєктами та відсутність можливості створення розширених групових чатів у безкоштовній версії може обмежити функціональність для користувачів зі специфічними потребами. Також, для певних функцій, таких як видалення відправлених повідомлень, може бути обмеження часу на можливість виправлення помилок після їх відправлення.

Таблиця 1.1 – Порівняння програмного забезпечення для комунікації

Критерій порівняння	Google meet	Skype	Slack	WhatsApp
Максимальна кількість учасників у безкоштовній версії	100	250	2	256
Наявність безкоштовної версії	+	+	+	+
Наявність демонстрації екрану	+	+	+	—
Запис розмови	—	+	—	—

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних систем для підтримки комунікації між користувачами.

Проте деякі з них є досить складними у використанні, що пов'язано з великою кількістю зайвих функцій. Та не дають можливості розгорнутися на власному сервері щоб мати доступ до системи навіть без доступу до глобального інтернету.

При розробці програми для підтримки комунікації необхідно забезпечити такі функціональні вимоги:

- авторизація та реєстрація користувачів з можливістю додавання, редагування та видалення особистих даних (ім'я, електронна пошта);
- додавання організацій;
- видалення організацій;
- додавання зустрічей;
- видалення зустрічей;
- можливість розгорнути додаток на власному сервері;
- адміністратори можуть бачити статистику систему та додавати організації.

1.3 Висновки за розділом 1

Проаналізовано поняття про webRTC [1] та програмні засоби для комунікації та програмні засоби для підтримки комунікації між користувачами.

За результатами проведеного аналізу можна зроблено висновок про те, що деякі з програмних засобів для підтримки комунікації між користувачами є досить складними у використанні, що пов'язано з великою кількістю зайвих функцій, непотрібних у процесі комунікації між користувачами.

Інші програмні забезпечення не надають можливості розгорнутися на власному сервері. Тому актуальною є розробка програмного забезпечення для підтримки комунікації між користувачами.

Сформульовано функціональні вимоги до програмного забезпечення для підтримки комунікації між користувачами.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування та інших технологій розробки

При розробці програмного забезпечення для підтримки комунікації між користувачами було використано мову програмування JavaScript та TypeScript.

JavaScript (JS) — це мова сценаріїв, створена для розробки сучасних вебсайтів. Вона виникла для подолання обмежень застарілих статичних вебсайтів і введення динаміки в вебсторінки, а також для реалізації інших корисних функцій, таких як анімація та графіка. Хоча головне призначення - це розробка клієнтських програм, завдяки технологіям, наприклад, Node.js [6], вона може ефективно використовуватися і на стороні сервера [7].

Мова програмування JavaScript тепер є частиною багатьох вебсайтів, які використовують фреймворки JavaScript, такі як Vue [8] або Nest [9]. для одночасної розробки додатків на кількох платформах, включаючи Android та iOS. Також на основі JavaScript компанія microsoft випустила TypeScript [10] який вирішує деякі проблеми JavaScript.

TypeScript – це мова програмування, яка представляє розширення JavaScript, призначене для розробки великих та складних вебпроектів. Вона додає статичну типізацію до JavaScript, що робить код більш надійним та легшим у розумінні [10].

Основна мета TypeScript – це полегшити роботу розробникам при написанні та підтримці великих проектів. Завдяки статичній типізації, розробник може зазначити типи даних змінних, параметрів та інших об'єктів, що допомагає у виявленні помилок на етапі компіляції, що раніше могли виявитися лише під час виконання програми.

TypeScript також впроваджує концепції класів та модулів, що робить код більш структурованим та легшим у підтримці. Крім того, він включає сучасні функції ECMAScript [11] та дозволяє використовувати нові функціональності, навіть якщо середовище виконання не повністю підтримує їх.

Однією з основних переваг TypeScript є те, що він забезпечує велику міру сумісності з існуючим JavaScript кодом. Розробник може поступово впроваджувати TypeScript в проєкт, починаючи з окремих файлів чи модулів.

Отже, TypeScript дозволяє підвищити продуктивність розробки, зменшити кількість помилок та полегшити підтримку складних вебпроєктів.

Виконаємо порівняння JavaScript з іншими мовами, зокрема з Python та C#. Загальною рисою для всіх трьох мов програмування є широкий спектр застосування, наявність великої кількості ресурсів та підтримка ООП. Однак, JavaScript та C# використовуються більш для великих корпоративних проєктів та розробки ПЗ, тоді як Python частіше використовується для наукових досліджень, штучного інтелекту, інтернет-розробки. Складність синтаксису Python є суттєвою, ніж у JavaScript та C#. Швидкодія JavaScript більша порівняно з Python. Велика кількість бібліотек та фреймворків є наявною для всіх трьох мов програмування.

Обґрунтування вибору мови програмування для розробки програмного забезпечення для комунікації між користувачами наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для комунікації між користувачами

Критерій порівняння мов програмування	Мова програмування		
	JavaScript	Python	C#
Сфера застосування	Web-розробка	Web-розробка, штучний інтелект	Великі корпоративні проєкти, розробка ПЗ для Window
Зручність для створення програмного забезпечення для комунікації між користувачами	+	+	+
Підтримка функціонального підходу до програмування	+	+	+
Швидкодія	+	–	+
Простота синтаксису	+	+	–

Отже, для реалізації програмного забезпечення для підтримки комунікації обрано мову програмування JavaScript.

2.2 Вибір фреймворків та інструментів для створення програмного забезпечення для підтримки комунікації між користувачами

В якості фреймворку для серверної частини було обрано NestJS [9].

Nest JS – це фреймворк для розробки серверних додатків (backend) на мові програмування TypeScript, побудований на основі екосистеми Node.js. Він надає розробникам ефективні та масштабовані інструменти для створення сучасних та потужних серверних застосунків.

MongoDB – документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів [12].

Node.js – це виконавче середовище для виконання JavaScript коду на сервері. Розроблено на основі двигуна V8, який починає використовуватися у вебпереглядачі Google Chrome. Node.js дозволяє виконувати JavaScript на сервері, а не тільки на клієнтському боці, що відкриває нові можливості для розробки вебдодатків .

Основні особливості Nest JS:

- модульність: Nest JS створений з використанням модульної архітектури, яка сприяє організації коду в логічні блоки, що полегшує підтримку та розширення проєктів;

- Використання TypeScript: Основна мова розробки в NestJS - TypeScript, що дозволяє використовувати статичну типізацію та інші переваги цієї мови для підвищення надійності та швидкості розробки;

– Dependency Injection (DI): NestJS використовує вбудований механізм Dependency Injection для зручного та ефективного управління залежностями в додатках, що сприяє створенню легко тестованого коду;

– Express та Fastify: NestJS може працювати на основі популярних вебфреймворків Express або Fastify, надаючи гнучкість та швидкість вибору для розробників;

– Middleware та Pipes: Нест переважно використовує концепцію middleware та pipes для обробки запитів та відповідей, дозволяючи розробникам легко впроваджувати логіку на різних рівнях застосунку;

– WebSocket підтримка: NestJS надає зручні інструменти для роботи з WebSocket, що дозволяє розробникам побудовувати реальні часи та інтерактивні додатки;

– Масштабованість: Фреймворк добре підходить для розробки як невеликих проєктів, так і великих масштабованих систем, завдяки своїй модульності та структурі.

NestJS надає розробникам зручний і сучасний інструментарій для розробки серверної частини вебдодатків, що дозволяє створювати ефективні та легко збережені застосунки.

Обґрунтування вибору фреймворку для створення серверної частини програмного забезпечення для підтримки комунікації наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору фреймворку для створення серверної частини програмного забезпечення для підтримки комунікації

Критерій порівняння фреймворків	Фреймворки		
	Nest	Django	.Net
1	2	3	4
Швидкість	Висока	Середня	Висока
Мова програмування	JavaScript	Python	C#
Тип архітектури	Подієва	Модель-представлення-контролер	Модель-представлення-контролер

Продовження таблиці 2.2

1	2	3	4
Використання у високонавантажених додатках	+	+	+
Розширюваність	+	+	-
Однопоточність	+	—	—

В якості фреймворку для клієнтської частини було обрано Vue.

Vue.js – це прогресивний JavaScript фреймворк для створення користувацьких інтерфейсів (UI) та односторінкових додатків. Він часто використовується для розробки вебдодатків, де взаємодія користувача з інтерфейсом є ключовою частиною додатку [8].

Основні особливості Vue.js:

- легкість вивчення. Vue.js відомий своєю легкістю вивчення та інтеграції в існуючі проекти. Це частково через його простий синтаксис та документацію, що робить його доступним для новачків у веброзробці;

- прогресивність. Vue.js є прогресивним фреймворком, що дозволяє розробникам використовувати лише ту частину фреймворку, яка необхідна в конкретному проекті. Це робить його гнучким та підходящим для проектів будь-якої складності;

- двостороннє зв'язування даних. Vue.js надає простий та ефективний механізм двостороннього зв'язування даних між інтерфейсом користувача та моделлю, що полегшує синхронізацію даних;

- компонентна архітектура. Vue.js базується на концепції компонентів, що дозволяє розбити інтерфейс на невеликі та перевикористовувани блоки. Це полегшує роботу над проектами та сприяє підтримці коду;

- життєвий цикл компонента. У Vue.js кожен компонент має свій життєвий цикл, що дозволяє виконувати різні дії в різних етапах життєвого циклу компонента, такі як створення, оновлення, та знищення;

– велика активна спільнота. Vue.js має велику та активну спільноту розробників, що забезпечує швидший розвиток фреймворку та надає доступ до безлічі різноманітних розширень та додаткових ресурсів;

– підтримка та інтеграція. Vue.js легко інтегрується в існуючі проекти, що робить його зручним для використання в різних сценаріях веброзробки.

Обґрунтування вибору фреймворку для створення клієнтської частини програмного забезпечення для підтримки комунікації між користувачами наведено у таблиці 2.3.

Таблиця 2.3 – Обґрунтування вибору фреймворку для створення клієнту програмного забезпечення для підтримки комунікації між користувачами

Критерій порівняння фреймворків	Фреймворки		
	Vue	React	Angular
Підтримує однофайлові компоненти	+	+	–
Всі інструменти є з коробки за замовчуванням	+	–	+
Низький поріг входу	+	–	–
Підтримка virtual dom	+	+	–

2.3 Висновки за розділом 2

Для створення програмного забезпечення підтримки комунікації між користувачами обрано мову програмування JavaScript, що дозволяє програмам працювати на більшості веббраузерах без потреби інсталяції додаткового програмного забезпечення.

В якості фреймворку для створення програмного забезпечення підтримки комунікації між користувачами обрано фреймворки NestJS та VueJs, що мають набір вбудованих модулів, які дозволяють ефективно розробляти вебзастосунки.

3 ОПИС ПРОГРАМИ

3.1 Структурна схема програми для комунікації між користувачами

Структуру програмного забезпечення для підтримки комунікації між користувачами подамо у вигляді діаграм в нотації UML (рис. 3.1 – 3.3).

Загальна діаграма класів програмного забезпечення для підтримки комунікації між користувачами на рис. 3.1.

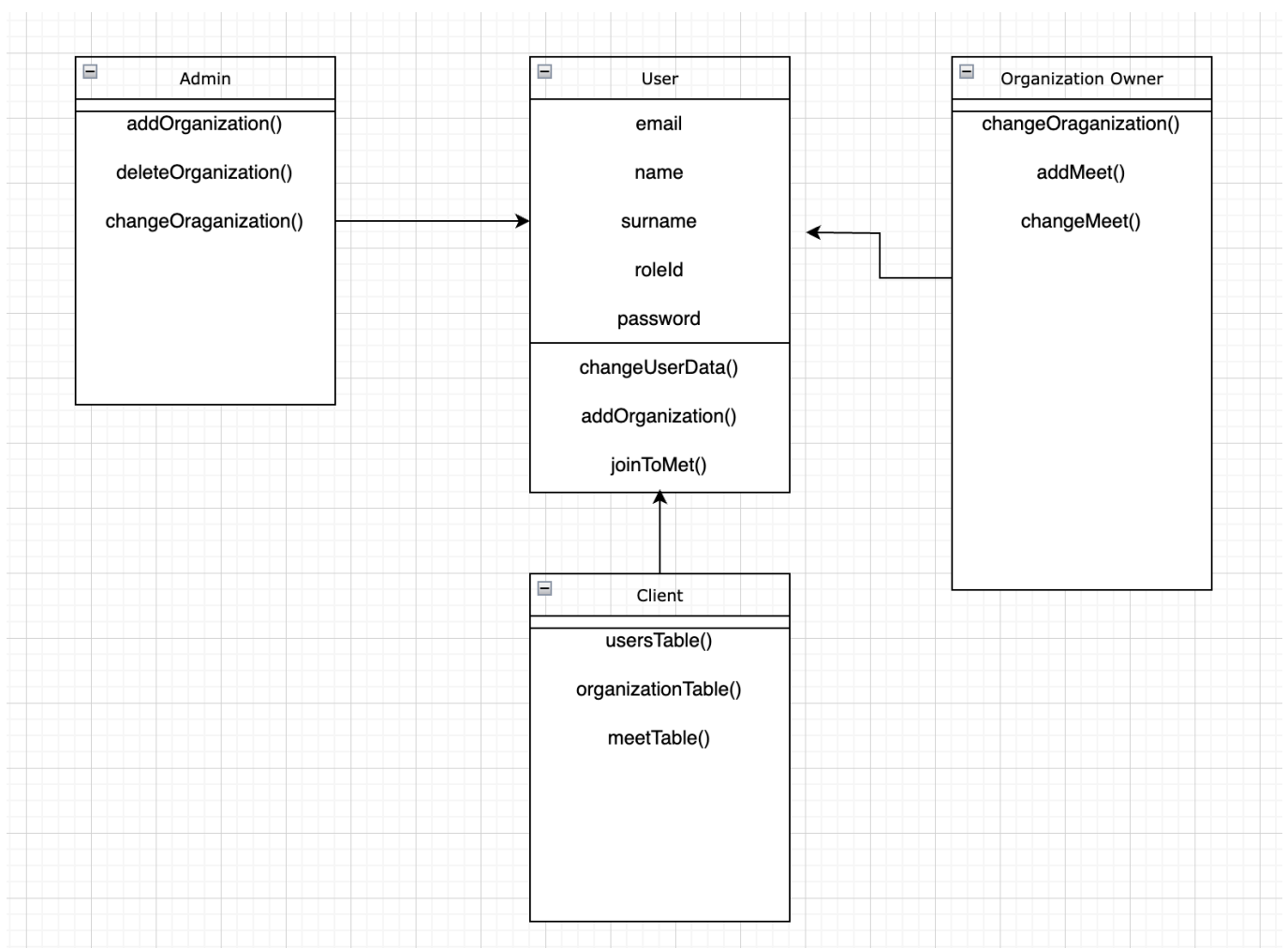


Рисунок 3.1 – Діаграма класів програмного забезпечення для підтримки комунікації між користувачами

Програмне забезпечення для підтримки комунікації між користувачами складається з пакетів, пов'язаних з клієнтами та API.

Діаграма компонентів програмного забезпечення для підтримки комунікації між користувачами. Програмне забезпечення було побудовано за архітектурою Rest api[7] зображена на рис. 3.2.

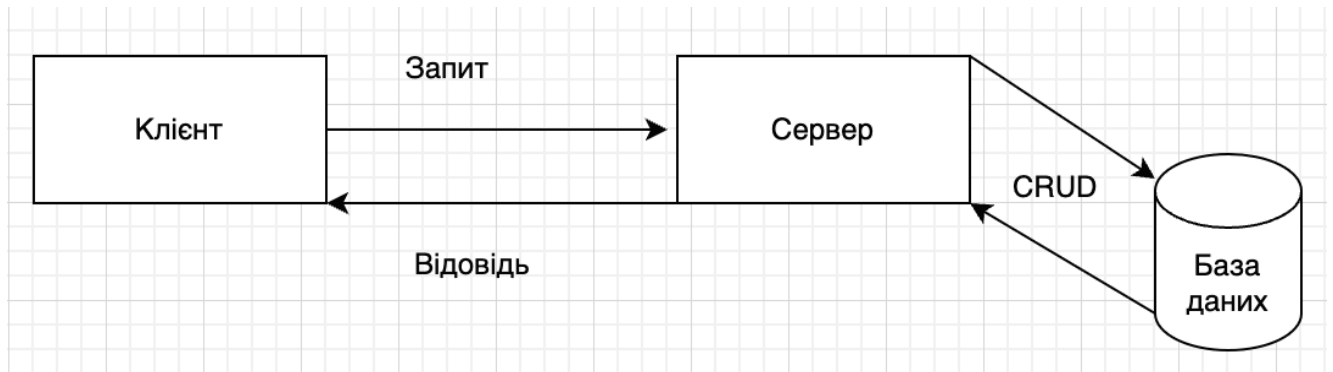


Рисунок 3.2 – Діаграма компонентів програмного забезпечення для підтримки комунікації між користувачами

Як видно з рисунку, основними компонентами є база даних, сервер та клієнт які спілкуються між собою.

3.2 Функціонування програмного забезпечення підтримки комунікації

Загальну схему функціонування програмного забезпечення для комунікації між користувачами подано у вигляді діаграмі прецедентів, зображеної на рис. 3.3.

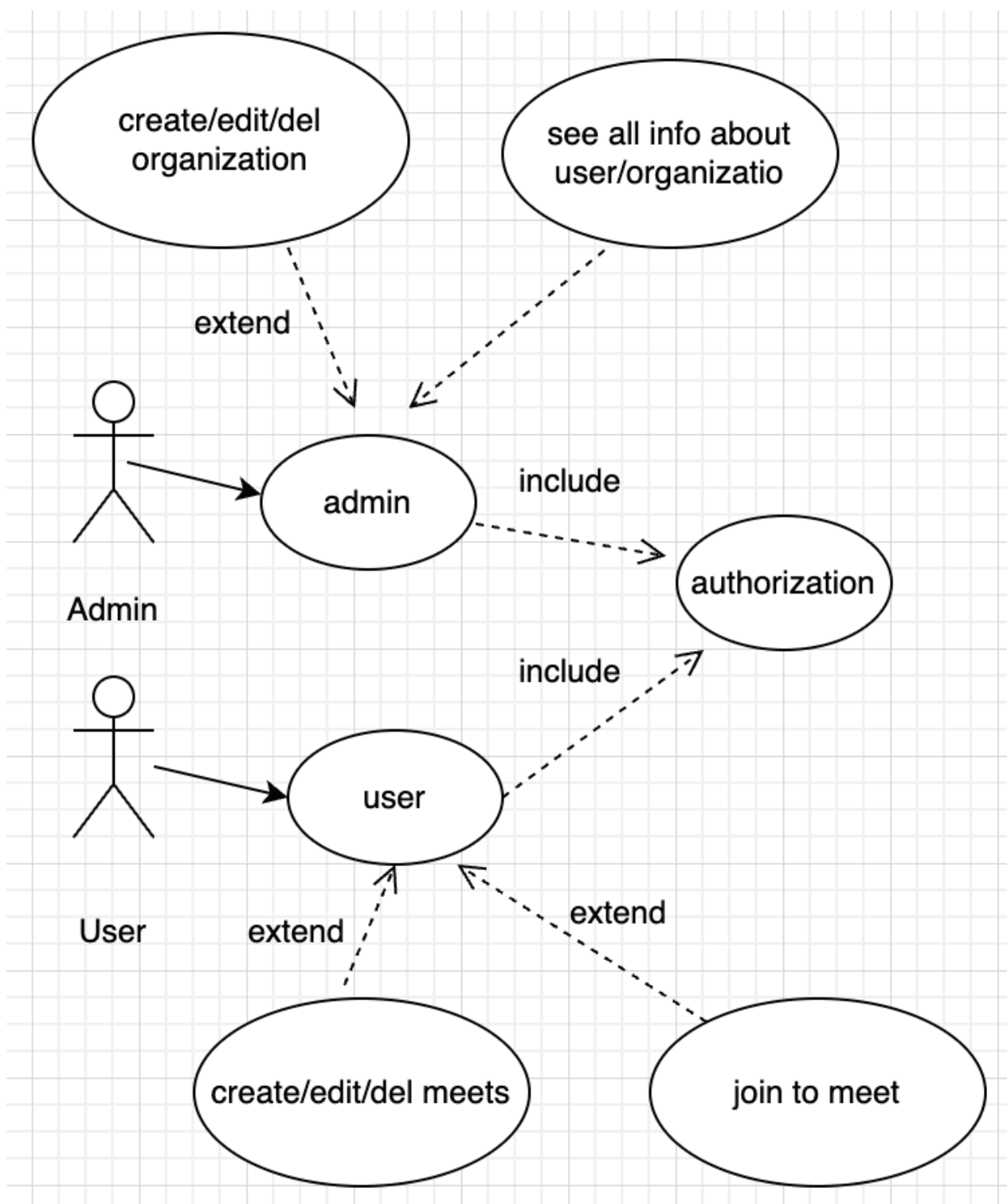


Рисунок 3.3 – Загальна схема функціонування програмного забезпечення для комунікації між користувачами у вигляді діаграми прецедентів

Діаграма діяльності програмного забезпечення комунікації між користувачами при пошуку інформації про користувача зображена на рис. 3.4.

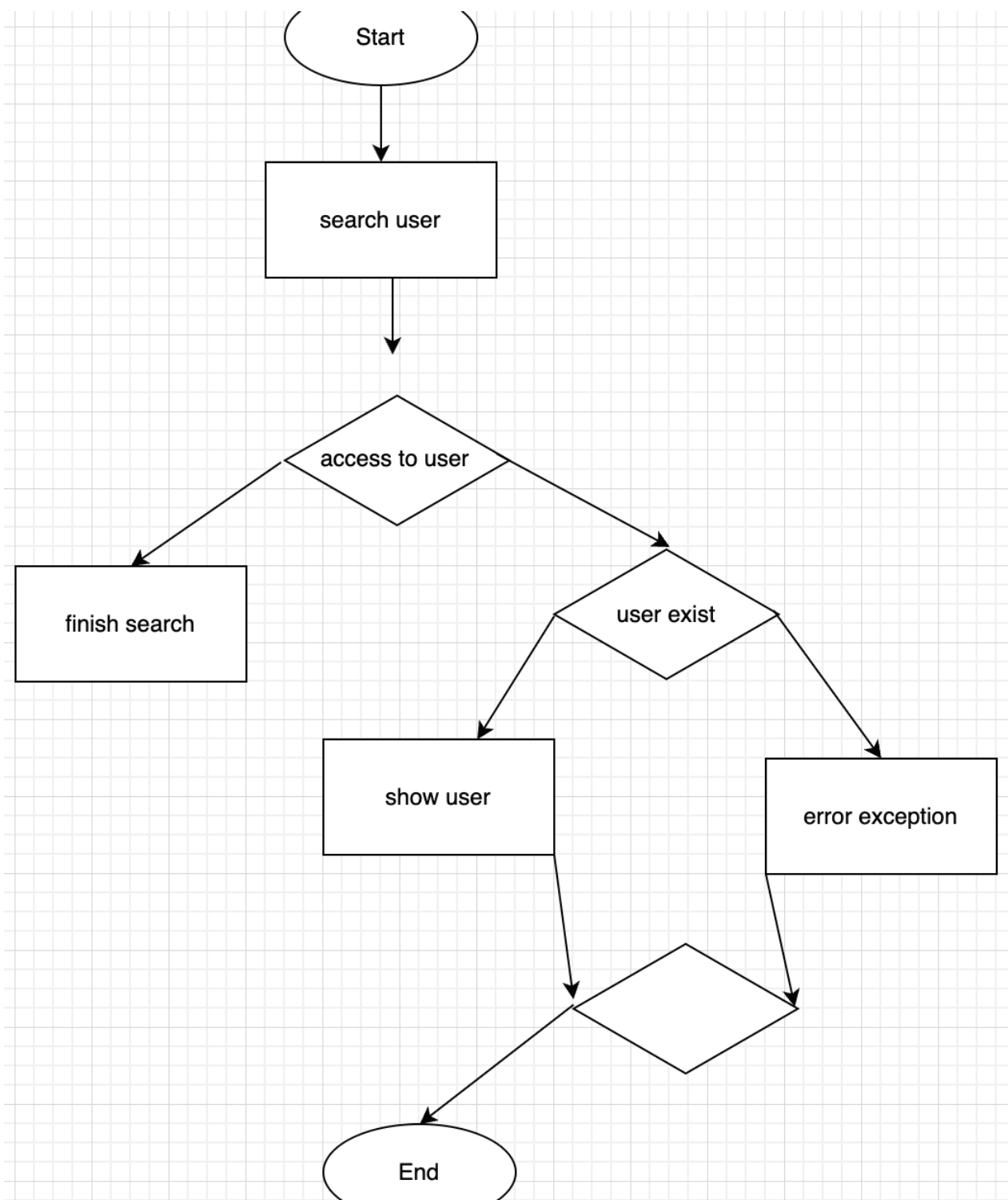


Рисунок 3.4 – Діаграма діяльності програмного забезпечення для комунікації між користувачами

Діаграма послідовності програмного забезпечення для комунікації між користувачами у нотації UML зображена на рис. 3.5.

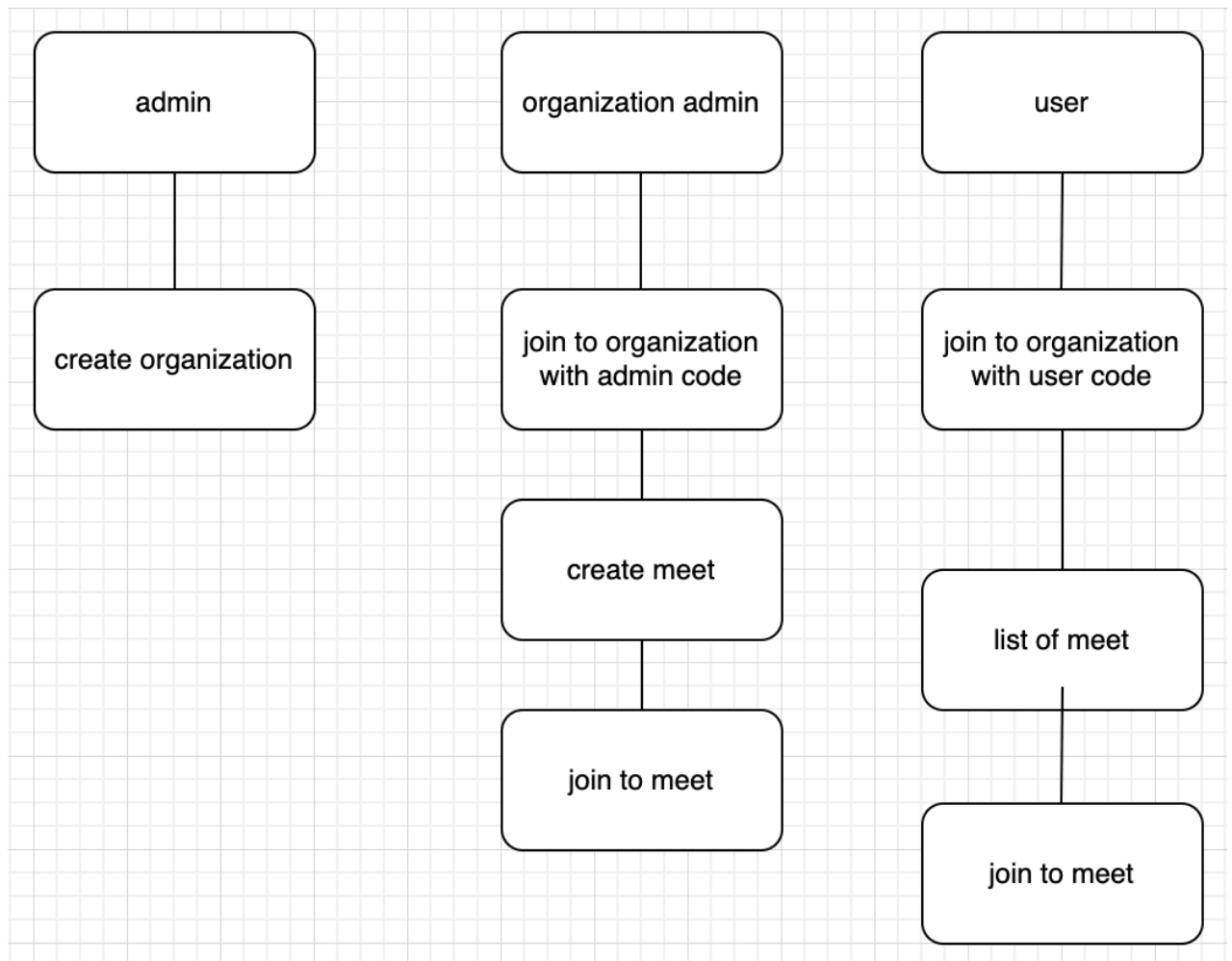


Рисунок 3.5 – Діаграма послідовностей програмного забезпечення для комунікації між користувачами

3.3 Проектування інтерфейсу взаємодії користувача з програмним забезпеченням для підтримки комунікації між користувачами

При розробленні програмного забезпечення для комунікації між користувачами враховані функціональні вимоги до програми:

- авторизація та реєстрація користувачів з можливістю додавання, редагування та видалення особистих даних (ім'я, електронна пошта);
- додавання організацій;
- видалення організацій;
- додавання зустрічей;
- видалення зустрічей;

- можливість розгорнути додаток на власному сервері;
- адміністратори можуть бачити статистику систему та додавати організації.

Головну сторінку вебсайту програмного забезпечення для підтримки комунікації між користувачами зображено на рис. 3.6. Тут користувачі можуть спілкуватися між собою.

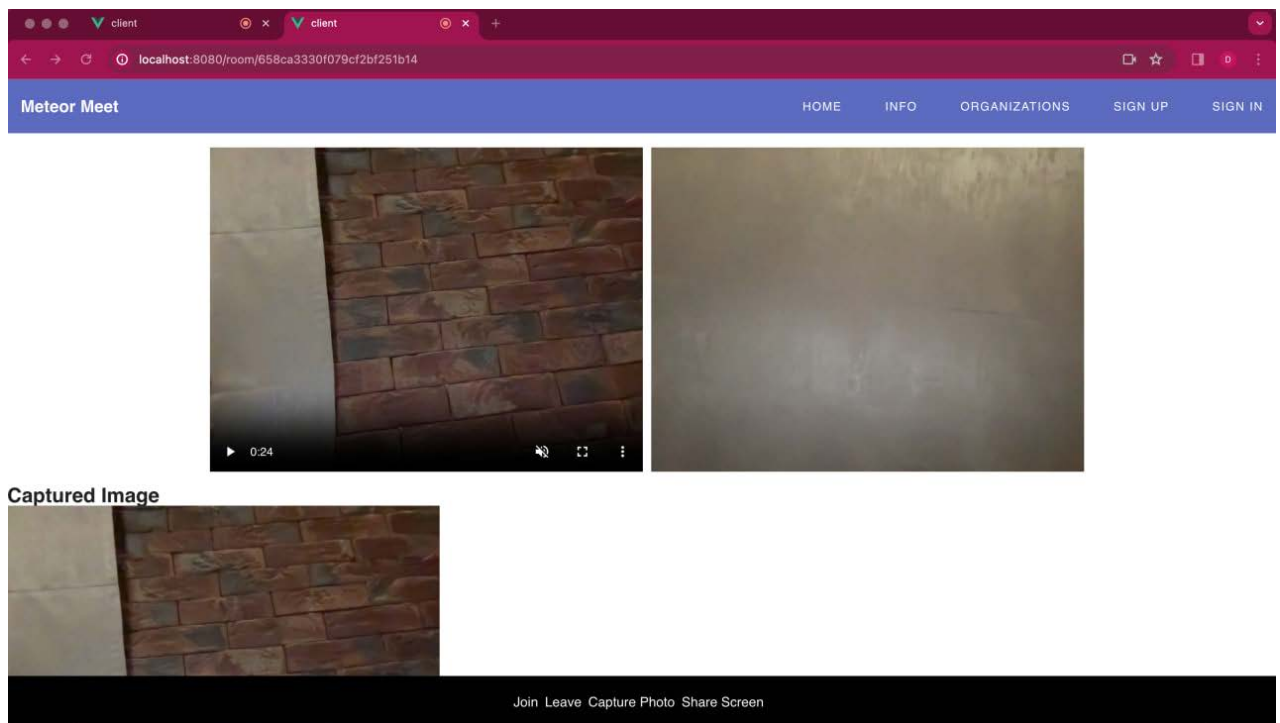


Рисунок 3.6 – Головна сторінка програми

Форма авторизації користувачів наведено на рис. 3.7.

The image shows the login form of the Meteor Meet application. It has a blue header with 'Meteor Meet' and navigation links 'HOME', 'INFO', 'ORGANIZATIONS', 'SIGN UP', and 'SIGN IN'. Below the header, there are two input fields: 'Email' and 'Password'. Below these fields is a blue button labeled 'Login'.

Рисунок 3.7 – Форма авторизації користувачів

Після реєстрації чи авторизації відкривається сторінка користувача (рис. 3.8), на якій відображаються інформація про користувача.



Рисунок 3.8 – Сторінка користувача програмного забезпечення для комунікації між користувачами

Сторінка з інформацією про користувачів зображено на рис. 3.9.

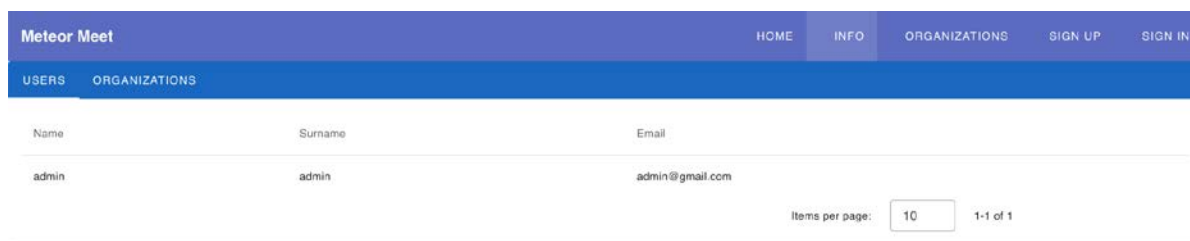


Рисунок 3.9 – Сторінка з інформацією про користувачів

На сторінці організацій адмін може подивитися всі організації зображено на рис. 3.10 та додати нову організацію зображено на рис. 3.11.

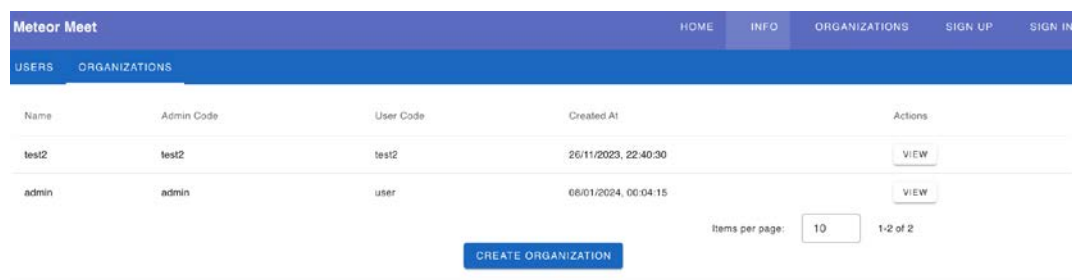
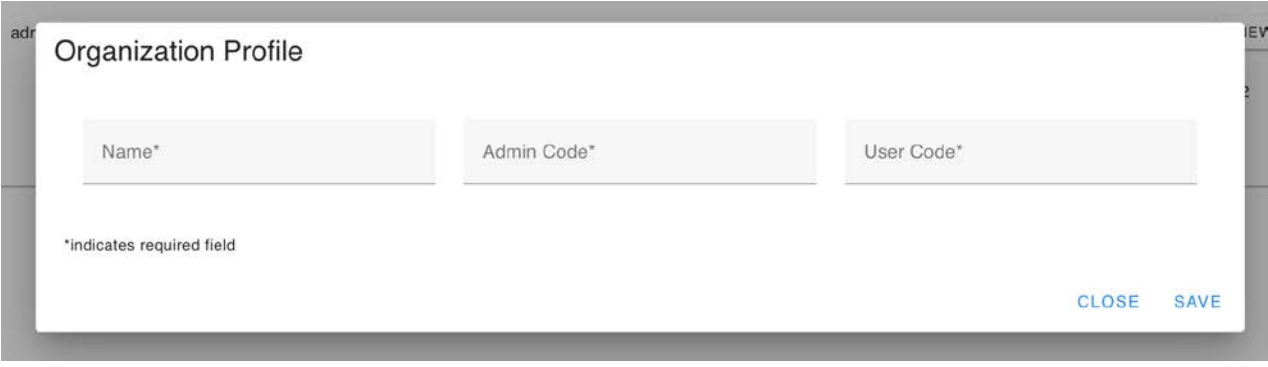


Рисунок 3.10 – Сторінка з інформацією про організації

A modal form titled "Organization Profile" with three input fields: "Name*", "Admin Code*", and "User Code*". A note below the fields states "*indicates required field". At the bottom right are "CLOSE" and "SAVE" buttons.

adr

Organization Profile

Name*

Admin Code*

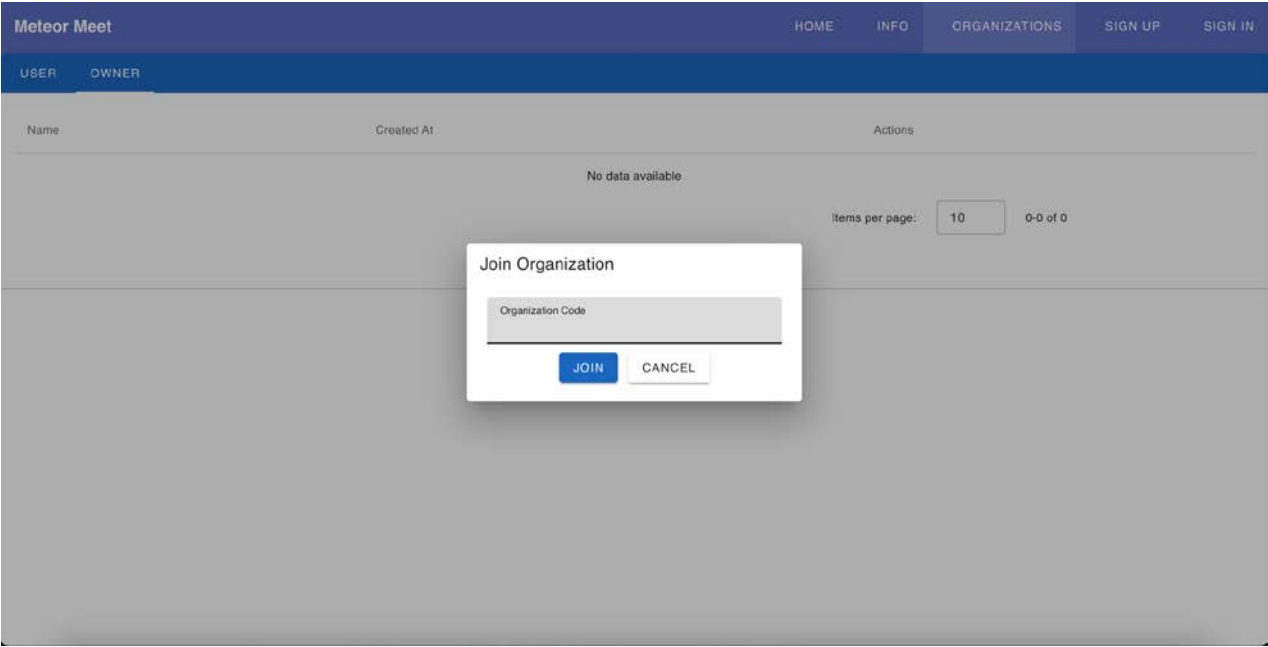
User Code*

*indicates required field

CLOSE SAVE

Рисунок 3.11 – Форма для додавання нової організації

Форма для приєднання до організації зображено на рис. 3.12.

A screenshot of the "Meteor Meet" application interface. The top navigation bar includes "HOME", "INFO", "ORGANIZATIONS", "SIGN UP", and "SIGN IN". Below it, a sub-navigation bar has "USER" and "OWNER". The main content area shows a table with columns "Name", "Created At", and "Actions", but it is empty with the message "No data available". A "Join Organization" modal is open, featuring an "Organization Code" input field and "JOIN" and "CANCEL" buttons. On the right, there are pagination controls showing "Items per page: 10" and "0-0 of 0".

Meteor Meet

HOME INFO ORGANIZATIONS SIGN UP SIGN IN

USER OWNER

Name Created At Actions

No data available

Items per page: 10 0-0 of 0

Join Organization

Organization Code

JOIN CANCEL

Рисунок 3.12 – Форма для приєднання до організації

На сторінці зустрічей можливо подивитися зустрічі та приєднатися до потрібної зустрічі зображено на рис. 3.13.

Meteor Meet				HOME	INFO	ORGANIZATIONS	SIGN UP	SIGN IN
Name	Owner name	Created At	Actions					
test	admin	28/12/2023, 00:20:35	EDIT DELETE JOIN					
			CREATE MEET					
			Items per page: 10 1-1 of 1					

Рисунок 3.13 – Сторінка зустрічей

3.4 Висновки за розділом 3

Розроблено програмне забезпечення для підтримки комунікації між користувачами. Запропоновано структуру та описано основні модулі програмного забезпечення. Визначено, що основними класами програми для комунікації між користувачами є: Зустрічі (Meets), Користувач (User), Адміністратор (Admin), Таблиці (Tables) та ін. Основними компонентами є база даних та набір програмних модулів для забезпечення комунікації.

Описано функціонування програмного забезпечення для підтримки комунікації між користувачами.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для підтримки комунікації між користувачами.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для підтримки функцій, пов'язаних з комунікацією. Програма написана на мові JavaScript, в якості фреймворку для створення обрано Nest Js та Vue JS, що має набір вбудованих модулів, які дозволяють ефективно розробляти вебзастосунки.

Програмне забезпечення для підтримки комунікації між користувачами забезпечує виконання таких функцій:

- авторизація та реєстрація користувачів з можливістю додавання, редагування та видалення особистих даних (ім'я, електронна пошта);
- додавання організацій;
- видалення організацій;
- додавання зустрічей;
- видалення зустрічей
- можливість розгорнути додаток на власному сервері;
- адміністратори можуть бачити статистику систему та додавати організації.

Для експлуатації програмного забезпечення для комунікації між користувачами необхідно таке апаратне забезпечення:

- процесор Intel I3/ amd ryzen 3 чи більш пізня модель;
- оперативна пам'ять – 2 Гб;
- встановлене на сервері програмне забезпечення: вебсервер, Docker [6];
- вільне місце на диску – 25 Гб.

Функціональні характеристики розробленого програмного забезпечення наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для підтримки комунікації між користувачами наведено у додатку Б.

4.2 Характеристики програми та розгортання вебсерверу

Програмне забезпечення для підтримки комунікації між користувачами працює безперервно, доступ до інформації бази даних надається тільки авторизованим користувачам з відповідними правами доступу.

Фізичну структуру програмного забезпечення наведено на рис. 4.1.

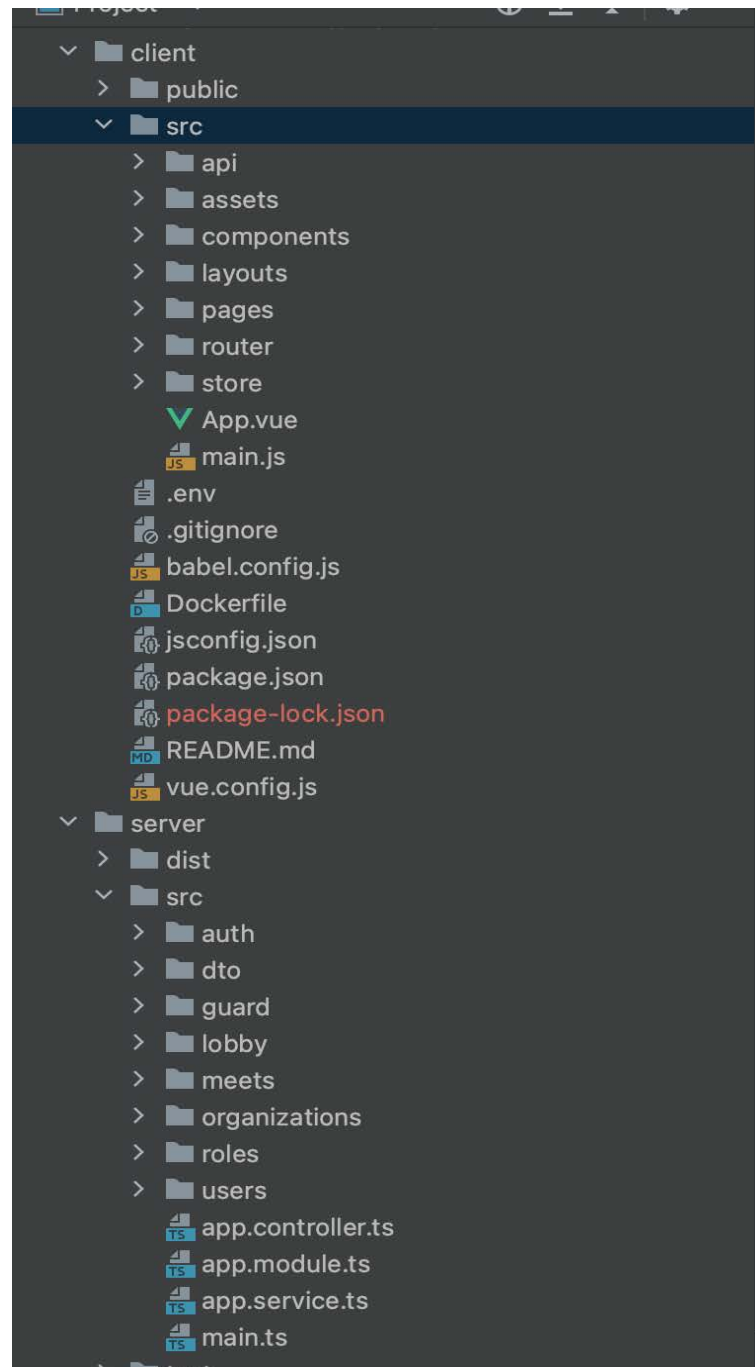


Рисунок 4.1 – Фізична програмного забезпечення для підтримки комунікації між користувачами

Для розгортання програмного забезпечення використовують Docker завдяки якому можна легко і швидко розгорнути клієнт та сервер.

Розгортання програмного забезпечення наведено на рис. 4.2.

```
[+] Building 22.0s (4/10)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 305B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18
=> [1/6] FROM docker.io/library/node:18@sha256:d7885552c11adb0f251db87d96a9c964990235569c8291bc61c276261947d6de
=> => resolve docker.io/library/node:18@sha256:d7885552c11adb0f251db87d96a9c964990235569c8291bc61c276261947d6de
=> => sha256:d7885552c11adb0f251db87d96a9c964990235569c8291bc61c276261947d6de 1.21kB / 1.21kB
=> => sha256:520b7d73a16c82100e4c52f39a3256b9ecfac75e9bc42902922c98822acf4f40 7.54kB / 7.54kB
=> => sha256:5665c1f9a9e17acd68ae05b2839df402eac34afdd095f8c115f09886d757840c 49.59MB / 49.59MB
=> => sha256:cc80d43376ca4eb47882ddd55128f05de44f2543602f4b15f7eb15ac299f02a9 2.00kB / 2.00kB
=> => sha256:f419b1a62fc83850ab3cb43274970bb20a18ae6e674535478a48f5bee11559b6 23.58MB / 23.58MB
=> => sha256:069a35492a4c5b1727f36b1184c413a96ce816d65578adaf3bce2023a1765c0a 63.99MB / 63.99MB
=> => sha256:599ff0dd2e553187212611c843bb09b42ae90ff2d37c73e897d9e2e86c995a9 17.83MB / 202.50MB
=> => sha256:1514d1c4225e6201fdca71cb7a1136f4e9a818be3c9ecf45c88a7d818986e5b9 3.37kB / 3.37kB
=> => sha256:68160b2a93005243c2463dc7d2f3d0ffc020d367da076fc410a69970678a0251 46.12MB / 46.12MB
=> => extracting sha256:5665c1f9a9e17acd68ae05b2839df402eac34afdd095f8c115f09886d757840c
=> => sha256:38ad88fcae0f2eab4952dba568d3820d7119e437607c58e8e923a05c7c001ee4 2.21MB / 2.21MB
=> => sha256:0fa5a8c933df2b22a829448bdea5061ce3fed5baa0ab6563800548af56718191 452B / 452B
=> => extracting sha256:f419b1a62fc83850ab3cb43274970bb20a18ae6e674535478a48f5bee11559b6
=> => extracting sha256:069a35492a4c5b1727f36b1184c413a96ce816d65578adaf3bce2023a1765c0a
=> [internal] load build context
=> => transferring context: 765.25kB
```

Рисунок 4.2 – Розгортання програмного забезпечення

Розгорнені застосунки у Docker Desktop наведено на рис. 4.3.

Container CPU usage ⓘ

0.37% / 1000% (10 cores available)

Container memory usage ⓘ

622.2MB / 7.48GB

Show charts ▾

🔍

Search

☰

Only show running containers

☐	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
☐	 eloquent_albattani df43a9a26387 	meteor-meet-vue	Running	0.19%	8080:8080 	3 minutes ago	  
☐	 hungry_yonath ca4398c9fd57 	meteor-back	Running	0.18%	3000:3000 	31 seconds ago	  

Рисунок 4.3 – Розгорнені застосунки у Docker Desktop

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Звертання користувачів до програми відбувається за допомогою інтернет-браузера шляхом переходу на відповідну адресу.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для підтримки комунікації між користувачами є інформація про користувачів та інша інформація, що зберігається у базі даних програми.

Вихідними даними програмного забезпечення є вебсторінки, що відображаються у веббраузері користувача, інформація користувача та дані про організації та зустрічі.

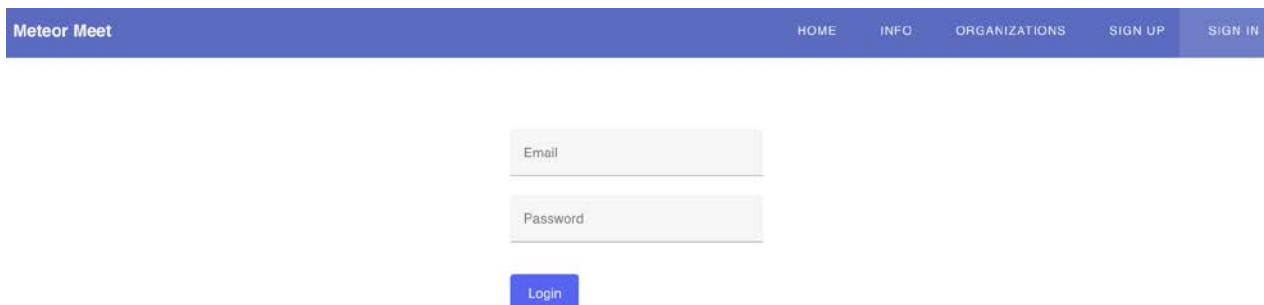
4.3.3 Повідомлення

Користувач програмного забезпечення для підтримки комунікації між користувачами може отримати такі повідомлення: повідомлення про успішну реєстрацію, повідомлення про невірне введення логіну або пароллю при спробі авторизації.

4.4 Виконання програмного забезпечення для підтримки комунікації між користувачами

Спочатку власнику програми для підтримки комунікації між користувачами необхідно внести у `.env` дані для адмін акаунтов які створюються автоматично при старті програми. Далі потрібно авторизуватися в акаунті та створити організації.

При запуску програмного забезпечення для комунікації між користувачами відкривається сторінка логіна, яка зображена на рис. 4.4.



Meteor Meet

HOME INFO ORGANIZATIONS SIGN UP SIGN IN

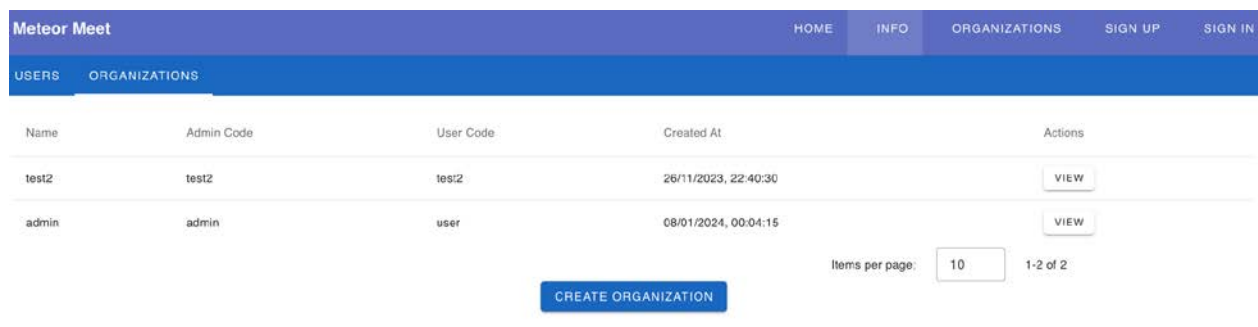
Email

Password

Login

Рисунок 4.4 – Сторінка логін

На сторінки організації можна побачити всі організації та створити на організацію зображеннями наведено на рис. 4.5.



Meteor Meet

HOME INFO ORGANIZATIONS SIGN UP SIGN IN

USERS ORGANIZATIONS

Name	Admin Code	User Code	Created At	Actions
test2	test2	test2	26/11/2023, 22:40:30	VIEW
admin	admin	user	08/01/2024, 00:04:15	VIEW

Items per page: 10 1-2 of 2

CREATE ORGANIZATION

Рисунок 4.5 – Сторінка організації

На сторінці “meet” можна побачити всіх учасників приклад наведено на рис. 4.6.

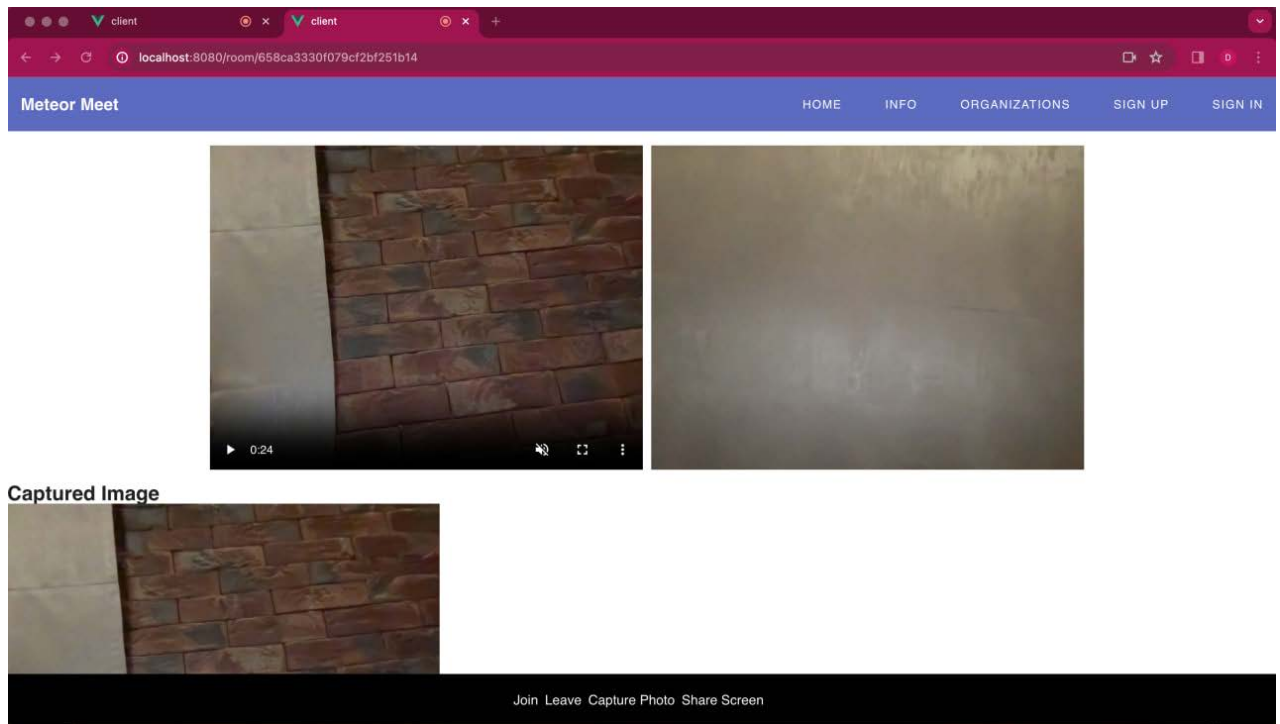


Рисунок 4.6 – Сторінка “meet”

4.5 Тестування програмного забезпечення для підтримки комунікації між користувачами

Під час тестування програмного забезпечення для підтримки комунікації між користувачами було перевірено його функціональність на різних операційних системах та веббраузерах. У процесі тестування було відтворено послідовність рекомендованих дій, щоб перевірити коректність роботи різних функцій програмного забезпечення. Під час перевірки функціоналу програмного забезпечення було перевірено коректність роботи таких функцій: авторизація та реєстрація користувачів з можливістю додавання, редагування та видалення особистих даних, додавання зустрічей та організацій

Оскільки під час тестування програмного забезпечення не виявлено помилок, збоїв або інших серйозних проблем, можна зробити висновок, що тестування було успішним, а програмне забезпечення готове до використання.

4.6 Висновки за розділом 4

Описано програмне забезпечення підтримки комунікації між користувачами. Виконано тестування розробленого програмного забезпечення. За результатами тестування програмного забезпечення не виявлено помилок, збоїв або інших серйозних проблем, тому можна зробити висновок, що тестування було успішним, а програмне забезпечення готове до використання.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процеси комунікації між користувачами.

Проаналізовано поняття про webRTC та програмні засоби для комунікації. Показано, що у наш час існує багато програм для комунікації між користувачами. Ці програми, як правило, представляють собою соціальні мережі та дозволяють користувачам комунікувати між собою.

За результатами проведеного аналізу можна зроблено висновок про те, що деякі з програмних засобів комунікації є досить складними у використанні, що пов'язано з великою кількістю зайвих функцій, непотрібних у процесі комунікації. Тому актуальною є розробка програмного забезпечення для підтримки комунікації між користувачами.

Сформульовано функціональні вимоги до програмного забезпечення для підтримки комунікації між користувачами.

Для створення програмного забезпечення для підтримки комунікації між користувачами обрано мову програмування JavaScript, що дозволяє програмам працювати на більшості веббраузерів без потреби інсталяції додаткового програмного забезпечення.

В якості фреймворків для створення програмного забезпечення обрано Nest.js та Vue Js, що мають набір вбудованих модулів, які дозволяють ефективно розробляти вебзастосунки.

Розроблено програмне забезпечення для підтримки комунікації між користувачами. Запропоновано структуру та описано основні модулі програмного забезпечення. Визначено, що основними класами програми для комунікації є: Зустрічі (Meets), Користувач (User), Адміністратор (Admin), Таблиці (Tables) та ін. Основними компонентами є база даних та набір програмних модулів для забезпечення комунікації.

Описано функціонування програмного забезпечення для підтримки комунікації між користувачами.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для підтримки комунікації між користувачами.

Описано функціонування програмного забезпечення для підтримки комунікації між користувачами.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для підтримки комунікації між користувачами.

Виконано тестування розробленого програмного забезпечення для підтримки комунікації між користувачами. За результатами тестування програмного забезпечення не виявлено помилок, збоїв або інших серйозних проблем, тому можна зробити висновок, що тестування було успішним, а програмне забезпечення готове до використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. WebRTC (Web Real-Time Communication): An open project that provides technologies to enable real-time audio, video, and data exchange between web and mobile applications through browsers [Electronic resource]. – Access mode: <https://webrtc.org/?hl=en>.
2. Google Meet [Electronic resource]. – Access mode: <https://www.sherweb.com/blog/g-suite/google-meet/>.
3. Skype [Electronic resource]. – Access mode: <https://www.skype.com/en/about/>.
4. Slack [Electronic resource]. – Access mode: <https://slack.com/about#:~:text=Slack%20is%20the%20productivity%20platform,a nd%20people%20love%20to%20use.>
5. WhatsApp Web [Electronic resource]. – Access mode: <https://softwarekeep.com/help-center/whats-app-web>.
6. Node.js API Documentation: The official API documentation for Node.js, which describes in detail all available modules and their functions [Electronic resource]. – Access mode: <https://nodejs.org/docs/latest/api/>.
7. RESTful API (Amazon Web Services) [Electronic resource]. – Access mode: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls
8. Vue.js: The Progressive JavaScript Framework [Electronic resource]. – Access mode: <https://vuejs.org>.
9. NestJS Documentation [Electronic resource]. – Access mode: <https://docs.nestjs.com/modules>.
10. TypeScript Documentation [Electronic resource]. – Access mode: <https://www.typescriptlang.org/docs/>.
11. ECMAScript [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/ECMAScript#:~:text=ECMAScript%20\(%2F%CB%88%C9%9Bkm,in%20the%20document%20ECMA%2D262.](https://en.wikipedia.org/wiki/ECMAScript#:~:text=ECMAScript%20(%2F%CB%88%C9%9Bkm,in%20the%20document%20ECMA%2D262.)

12. Вступ до JavaScript [Електрон. ресурс]. – Режим доступу:
<https://uk.javascript.info/intro>.

13. Інтернет-комунікація [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BA%D0%BE%D0%BC%D1%83%D0%BD%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F>.

14. Docker Command Line Reference [Electronic resource]. – Access mode:
<https://docs.docker.com/get-started/overview/>

15. MongoDB [Electronic resource]. – Access mode:
<https://www.mongodb.com/docs/>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення для комунікації між користувачами може використовуватися для підтримки функцій, пов'язаних з онлайн комунікацією

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для комунікації між користувачами», затверджене наказом Національного університету «Запорізька політехніка» №168 від 24 квітня 2024 р.

A.2 Призначення розробки

Програмний продукт призначений для комунікації між користувачами програми.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для комунікації виконання таких функцій:

- авторизація та реєстрація користувачів
- додавання та редагування організацій ;
- додавання та видалення зустрічей;
- управління камерой та мікрофоном.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програми для комунікації повинен бути зручним для користувачів. Взаємодія між програмою та користувачем повинна бути організована через веббраузер.

A.3.3 Вимоги до надійності

Програмне забезпечення для комунікації повинно забезпечити надійне функціонування та передбачити обробку помилок.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для комунікації між користувачами необхідна наявність пристрою у користувача зі встановленим веббраузером.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного забезпечення для комунікації між користувачами необхідно таке апаратне забезпечення:

- процесор Intel I3/ amd ryzen 3 чи більш пізня модель;
- оперативна пам'ять – 2 Гб;
- встановлене на сервері програмне забезпечення: вебсервер, Docker [6];
- вільне місце на диску – 25 Гб.

A.3.6 Вимоги до маркування і пакування

Програма для комунікації може бути записана на будь-якому носії інформації. На пакуванні повинна бути назва програми – «Програмне забезпечення для комунікації між користувачами».

A.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для комунікації між користувачами є інформація про користувачів, організації, зустрічі, користувачів та інша інформація, що зберігається у базі даних програми.

Вихідними даними програмного забезпечення для комунікації між користувачами є вебсторінки, що відображаються у веббраузері користувача, зустрічі користувача та інших користувачів, повідомлення про успішне створення зустрічі.

ДОДАТОК Б
Фрагмент тексту програми

```

import { Logger } from '@nestjs/common';
import { OnGatewayInit, WebSocketGateway, WebSocketServer } from '@nestjs/websockets';
import { Namespace } from 'socket.io';
import { LobbyService } from './lobby.service';
import { Server } from 'http';

@WebSocketGateway({ cors: { origin: '*', methods: ['GET', 'POST', 'PUT', 'PATCH'] } })
export class LobbyGateway implements OnGatewayInit {
  @WebSocketServer()
  server: Server;

  private logger: Logger = new Logger('LobbyGateway');

  constructor(private readonly lobbyService: LobbyService) {}

  afterInit(server: Namespace) {
    this.logger.log('WebSocket Gateway initialized');

    const signalServer = require('simple-signal-server')(server)
    signalServer.on('discover', (request) => {
      this.logger.log('discover');
      const memberId = request.socket.id;
      const roomId = request.discoveryData;
      const members = this.lobbyService.joinRoom(roomId, memberId);
      request.socket.roomId = roomId;
      request.discover({
        peers: Array.from(members),
      });
      this.logger.log('joined ' + roomId + ' ' + memberId);
    });

    signalServer.on('disconnect', (socket) => {
      const memberId = socket.id;
      const roomId = socket.roomId;
      const members = this.lobbyService.leaveRoom(roomId, memberId);
      this.logger.log('left ' + roomId + ' ' + memberId);
    });

    signalServer.on('request', (request) => {
      request.forward();
      this.logger.log('requested');
    });
  }
}

```

```

}

// src/lobby/lobby.controller.ts
import { Controller, Get } from '@nestjs/common';
import { LobbyService } from './lobby.service';
@Controller('lobby')
export class LobbyController {
  constructor(private readonly lobbyService: LobbyService) {}

  @Get()
  getLobbyInfo() {
    const rooms = this.lobbyService.getRooms();
    let sum = 0;
    rooms.forEach((members, roomId) => (sum = sum + members.size));
    return {
      rooms: rooms.size,
      members: sum,
    };
  }
}

// src/lobby/lobby.module.ts
import { Module } from '@nestjs/common';
import { LobbyService } from './lobby.service';
import { LobbyController } from './lobby.controller';
import { LobbyGateway } from './lobby.gateway';
@Module({
  providers: [LobbyService, LobbyGateway],
  controllers: [LobbyController],
})
export class LobbyModule {}

// src/lobby/lobby.service.ts
import { Injectable } from '@nestjs/common';
@Injectable()
export class LobbyService {
  private rooms = new Map();

  getRooms() {
    return this.rooms;
  }
}

```

```

joinRoom(roomId: string, memberId: string) {
  let members = this.rooms.get(roomId);
  if (!members) {
    members = new Set();
    this.rooms.set(roomId, members);
  }
  members.add(memberId);
  return Array.from(members);
}

leaveRoom(roomId: string, memberId: string) {
  const members = this.rooms.get(roomId);
  if (members) {
    members.delete(memberId);
  }
  return Array.from(members);
}
}

<template>
<v-app>
  <v-navigation-drawer color="red" v-model="sidebar" app>
    <v-list>
      <v-list-tile
        v-for="item in menuItems"
        :key="item.title"
        :to="item.path"
      >
        <v-list-tile-action>
          <v-icon>{{ item.icon }}</v-icon>
        </v-list-tile-action>
        <v-list-tile-content>{{ item.title }}</v-list-tile-content>
      </v-list-tile>
    </v-list>
  </v-navigation-drawer>
  <v-toolbar class="bg-indigo-lighten-1" app>
    <span class="hidden-sm-and-up">
      <v-toolbar-side-icon @click="sidebar = !sidebar"> </v-toolbar-side-icon>
    
```

```

</span>
<v-toolbar-title>
  <strong>{{ appTitle }}</strong>
</v-toolbar-title>
<v-spacer> </v-spacer>
<v-toolbar-items class="hidden-xs-only">
  <v-btn
    flat=""
    v-for="item in menuItems"
    :key="item.title"
    :to="item.path"
  >
    <v-icon left dark>{{ item.icon }}</v-icon>
    {{ item.title }}
  </v-btn>
</v-toolbar-items>
</v-toolbar>
<v-content style="margin-bottom: 60px">
  <router-view> </router-view>
</v-content>
</v-app>
</template>
<script>
import { ref, onMounted } from 'vue';
import { useUserStore } from "@store/user";
export default {
  name: "App",
  setup() {
    const appTitle = "Meteor Meet";
    const sidebar = ref(false);
    const menuItems = [
      { title: "Home", path: "/home", icon: "home" },
      { title: "Info", path: "/info", icon: "face" },
      { title: "Organizations", path: "/organizations", icon: "face" },
      { title: "Sign Up", path: "/signup", icon: "face" },
      { title: "Sign In", path: "/login", icon: "lock_open" }
    ]
  }
}

```

```

];
const store = useUserStore();
onMounted(async () => {
  try {
    await store.fetchUserData();
  } catch (error) {
    console.error(error);
  }
});
return {
  appTitle,
  sidebar,
  menuItems,
};
},
};
</script>
<style>
</style>
import { defineStore } from 'pinia'
import axiosInstance from "@/api/api";
export const useOrganizationStore = defineStore("organizations", {
  state: () => ({
  }),
  getters: {},
  actions: {
    async getAllOrganizations(page,limit) {
      try {
        return await axiosInstance.get(`/organizations/all?page=${page}&limit=${limit}`)
      }
      catch (error) {
        console.log(error)
      }
    },
    async getAllOrganizationsByRole(page,limit, role) {
      try {

```

```

        return
    },
    async createOrganization(data) {
        try {
            return await axiosInstance.post(`/organizations/create`, data)
        }
        catch (error) {
            console.log(error)
        }
    },
    async deleteOrganization(id) {
        try {
            return await axiosInstance.delete(`/organizations/delete/${id}`, )
        }
        catch (error) {
            console.log(error)
        }
    },
    async updateOrganization(data) {
        const { _id, name, adminCode, userCode } = data
        try {
            return await axiosInstance.put(`/organizations/update?organizationId=${_id}`, {name,
adminCode, userCode})
        }
        catch (error) {
            console.log(error)
        }
    }
},
))
<template>

```

```

<div class="container">
  <div class="row">
    <div id="room" class="col-md-12 mt-3 ml-3">
      <vue-webrtc ref="webrtc"
        socketURL="http://localhost:3000"
        width="100%"
        :roomId="$route.params.id"
        :enableLogs="true"
        v-on:joined-room="logEvent"
        v-on:left-room="logEvent"
        v-on:opened-room="logEvent"
        v-on:share-started="logEvent"
        v-on:share-stopped="logEvent"
        @error="onError" />
      </div>
    </div>
    <div class="row">
      <div class="col-md-12" style="padding-top: 5px;">
        <h2>Captured Image</h2>
        <figure class="figure">
          
        </figure>
      </div>
    </div>
  </div>
  <div class="bottom-bar">
    <button type="button" class="btn btn-primary text-white" @click="onJoin">Join</button>
    <button type="button" class="btn btn-primary text-white" @click="onLeave">Leave</button>
    <button type="button" class="btn btn-primary text-white" @click="onCapture">Capture
    Photo</button>
    <button type="button" class="btn btn-primary text-white" @click="onShareScreen">Share
    Screen</button>
  </div>
</template>
<script>
import { VueWebRTC } from 'vue-webrtc';

```

```

export default {
  name: 'room-component',
  components: {
    'vue-webrtc': VueWebRTC
  },
  data() {
    return {
      img: null,
      roomId: "public-room-v3"
    };
  },
  mounted: function () {
    this.$refs.webrtc.join();
  },
  computed: {
  },
  watch: {
  },
  methods: {
    onCapture() {
      this.img = this.$refs.webrtc.capture();
    },
    onLeave() {
      this.$refs.webrtc.leave();
    },
    onShareScreen() {
      this.img = this.$refs.webrtc.shareScreen();
    },
    onError(error, stream) {
      console.log('On Error Event', error, stream);
    },
    logEvent(event) {
      console.log('Event : ', event);
    },
  }
};

```

```

</script>
<style>
.btn {
  margin-right: 8px;
}
.bottom-bar {
  position: fixed;
  bottom: 0;
  width: 100%;
  height: 8%;
  background-color: black;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 10px;
  box-sizing: border-box;
}
.video-list {
  background: white !important;
}
.video-item video {
  width: auto;
  height: 50vh;
  margin: 5px;
}
.video-item {
  background: white !important;
}
</style>

import { Logger } from '@nestjs/common';
import { OnGatewayInit, WebSocketGateway, WebSocketServer } from '@nestjs/websockets';
import { Namespace } from 'socket.io';
import { LobbyService } from './lobby.service';
import { Server } from 'http';

@WebSocketGateway({ cors: { origin: '*', methods: ['GET', 'POST', 'PUT', 'PATCH'] } })
export class LobbyGateway implements OnGatewayInit {

```

```

@WebSocketServer()
server: Server;
private logger: Logger = new Logger('LobbyGateway');
constructor(private readonly lobbyService: LobbyService) {}
afterInit(server: Namespace) {
  this.logger.log('WebSocket Gateway initialized');
  const signalServer = require('simple-signal-server')(server)
  signalServer.on('discover', (request) => {
    this.logger.log('discover');
    const memberId = request.socket.id;
    const roomId = request.discoveryData;
    const members = this.lobbyService.joinRoom(roomId, memberId);
    request.socket.roomId = roomId;
    request.discover({
      peers: Array.from(members),
    });
    this.logger.log('joined ' + roomId + ' ' + memberId);
  });
  signalServer.on('disconnect', (socket) => {
    const memberId = socket.id;
    const roomId = socket.roomId;
    const members = this.lobbyService.leaveRoom(roomId, memberId);
    this.logger.log('left ' + roomId + ' ' + memberId);
  });
  signalServer.on('request', (request) => {
    request.forward();
    this.logger.log('requested');
  });
}
}

// src/lobby/lobby.controller.ts
import { Controller, Get } from '@nestjs/common';
import { LobbyService } from './lobby.service';
@Controller('lobby')
export class LobbyController {
  constructor(private readonly lobbyService: LobbyService) {}

```

```

@Get()
getLobbyInfo() {
  const rooms = this.lobbyService.getRooms();
  let sum = 0;
  rooms.forEach((members, roomId) => (sum = sum + members.size));
  return {
    rooms: rooms.size,
    members: sum,
  };
}
}

// src/lobby/lobby.module.ts
import { Module } from '@nestjs/common';
import { LobbyService } from './lobby.service';
import { LobbyController } from './lobby.controller';
import { LobbyGateway } from './lobby.gateway';
@Module({
  providers: [LobbyService, LobbyGateway],
  controllers: [LobbyController],
})
export class LobbyModule {}

// src/lobby/lobby.service.ts
import { Injectable } from '@nestjs/common';
@Injectable()
export class LobbyService {
  private rooms = new Map();
  getRooms() {
    return this.rooms;
  }
  joinRoom(roomId: string, memberId: string) {
    let members = this.rooms.get(roomId);
    if (!members) {
      members = new Set();
      this.rooms.set(roomId, members);
    }
    members.add(memberId);
  }
}

```

```

    return Array.from(members);
  }
  leaveRoom(roomId: string, memberId: string) {
    const members = this.rooms.get(roomId);
    if (members) {
      members.delete(memberId);
    }
    return Array.from(members);
  }
}

<template>
<v-app>
  <v-navigation-drawer color="red" v-model="sidebar" app>
    <v-list>
      <v-list-tile
        v-for="item in menuItems"
        :key="item.title"
        :to="item.path"
      >
        <v-list-tile-action>
          <v-icon>{{ item.icon }}</v-icon>
        </v-list-tile-action>
        <v-list-tile-content>{{ item.title }}</v-list-tile-content>
      </v-list-tile>
    </v-list>
  </v-navigation-drawer>
  <v-toolbar class="bg-indigo-lighten-1" app>
    <span class="hidden-sm-and-up">
      <v-toolbar-side-icon @click="sidebar = !sidebar"> </v-toolbar-side-icon>
    </span>
    <v-toolbar-title>
      <strong>{{ appTitle }}</strong>
    </v-toolbar-title>
    <v-spacer> </v-spacer>
    <v-toolbar-items class="hidden-xs-only">
      <v-btn

```

```

        flat=""
        v-for="item in menuItems"
        :key="item.title"
        :to="item.path"
    >
        <v-icon left dark>{{ item.icon }}</v-icon>
        {{ item.title }}
    </v-btn>
</v-toolbar-items>
</v-toolbar>
<v-content style="margin-bottom: 60px">
    <router-view></router-view>
</v-content>
</v-app>
</template>
<script>
import { ref, onMounted } from 'vue';
import { useUserStore } from "@store/user";
export default {
  name: "App",
  setup() {
    const appTitle = "Meteor Meet";
    const sidebar = ref(false);
    const menuItems = [
      { title: "Home", path: "/home", icon: "home" },
      { title: "Info", path: "/info", icon: "face" },
      { title: "Organizations", path: "/organizations", icon: "face" },
      { title: "Sign Up", path: "/signup", icon: "face" },
      { title: "Sign In", path: "/login", icon: "lock_open" }
    ];
    const store = useUserStore();
    onMounted(async () => {
      try {
        await store.fetchUserData();
      } catch (error) {
        console.error(error);
      }
    });
  }
};

```

```

    }
  });
  return {
    appTitle,
    sidebar,
    menuItems,
  };
},
};
</script>
<style>
</style>
import { defineStore } from 'pinia'
import axiosInstance from "@api/api";
export const useOrganizationStore = defineStore("organizations", {
  state: () => ({
  }),
  getters: {},
  actions: {
    async getAllOrganizations(page,limit) {
      try {
        return await axiosInstance.get(`/organizations/all?page=${page}&limit=${limit}`)
      }
      catch (error) {
        console.log(error)
      }
    },
    async getAllOrganizationsByRole(page,limit, role) {
      try {
        return
        axiosInstance.get(`/organizations/byRole?page=${page}&limit=${limit}&role=${role}`)
      }
      catch (error) {
        console.log(error)
      }
    },
  },

```

```

    async createOrganization(data) {
      try {
        return await axiosInstance.post(`/organizations/create`, data)
      }
      catch (error) {
        console.log(error)
      }
    },
    async deleteOrganization(id) {
      try {
        return await axiosInstance.delete(`/organizations/delete/${id}`, )
      }
      catch (error) {
        console.log(error)
      }
    },
    async updateOrganization(data) {
      const { _id, name, adminCode, userCode } = data
      try {
        return await axiosInstance.put(`/organizations/update?organizationId=${_id}`, {name,
adminCode, userCode})
      }
      catch (error) {
        console.log(error)
      }
    }
  },
})
<template>
<div class="container">
  <div class="row">
    <div id="room" class="col-md-12 mt-3 ml-3">
      <vue-webrtc ref="webrtc"
        socketURL="http://localhost:3000"
        width="100%"
        :roomId="$route.params.id"

```

```

      :enableLogs="true"
      v-on:joined-room="logEvent"
      v-on:left-room="logEvent"
      v-on:opened-room="logEvent"
      v-on:share-started="logEvent"
      v-on:share-stopped="logEvent"
      @error="onError" />
    </div>
  </div>
  <div class="row">
    <div class="col-md-12" style="padding-top: 5px;">
      <h2>Captured Image</h2>
      <figure class="figure">
        
      </figure>
    </div>
  </div>
</div>
<div class="bottom-bar">
  <button type="button" class="btn btn-primary text-white" @click="onJoin">Join</button>
  <button type="button" class="btn btn-primary text-white" @click="onLeave">Leave</button>
  <button type="button" class="btn btn-primary text-white" @click="onCapture">Capture
Photo</button>
  <button type="button" class="btn btn-primary text-white" @click="onShareScreen">Share
Screen</button>
</div>
</template>
<script>
import { VueWebRTC } from 'vue-webrtc';
export default {
  name: 'room-component',
  components: {
    'vue-webrtc': VueWebRTC
  },
  data() {
    return {

```

```

    img: null,
    roomId: "public-room-v3"
  };
},
mounted: function () {
  this.$refs.webrtc.join();
},
computed: {
},
watch: {
},
methods: {
  onCapture() {
    this.img = this.$refs.webrtc.capture();
  },
  onLeave() {
    this.$refs.webrtc.leave();
  },
  onShareScreen() {
    this.img = this.$refs.webrtc.shareScreen();
  },
  onError(error, stream) {
    console.log('On Error Event', error, stream);
  },
  logEvent(event) {
    console.log('Event : ', event);
  },
}
};
</script>
<style>
.btn {
  margin-right: 8px;
}
.bottom-bar {
  position: fixed;

```

```

    bottom: 0;
    width: 100%;
    height: 8%;
    background-color: black;
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 10px;
    box-sizing: border-box;
}

.video-list {
    background: white !important;
}

.video-item video {
    width: auto;
    height: 50vh;
    margin: 5px;
}

.video-item {
    background: white !important;
}

</style>

import { Logger } from '@nestjs/common';
import { OnGatewayInit, WebSocketGateway, WebSocketServer } from '@nestjs/websockets';
import { Namespace } from 'socket.io';
import { LobbyService } from './lobby.service';
import { Server } from 'http';

@WebSocketGateway({ cors: { origin: '*', methods: ['GET', 'POST', 'PUT', 'PATCH'] } })
export class LobbyGateway implements OnGatewayInit {
    @WebSocketServer()
    server: Server;

    private logger: Logger = new Logger('LobbyGateway');
    constructor(private readonly lobbyService: LobbyService) {}
    afterInit(server: Namespace) {
        this.logger.log('WebSocket Gateway initialized');
        const signalServer = require('simple-signal-server')(server)
    }
}

```

```

signalServer.on('discover', (request) => {
  this.logger.log('discover');
  const memberId = request.socket.id;
  const roomId = request.discoveryData;
  const members = this.lobbyService.joinRoom(roomId, memberId);
  request.socket.roomId = roomId;
  request.discover({
    peers: Array.from(members),
  });
  this.logger.log('joined ' + roomId + ' ' + memberId);
});
signalServer.on('disconnect', (socket) => {
  const memberId = socket.id;
  const roomId = socket.roomId;
  const members = this.lobbyService.leaveRoom(roomId, memberId);
  this.logger.log('left ' + roomId + ' ' + memberId);
});
signalServer.on('request', (request) => {
  request.forward();
  this.logger.log('requested');
});
}
}
// src/lobby/lobby.controller.ts
import { Controller, Get } from '@nestjs/common';
import { LobbyService } from './lobby.service';
@Controller('lobby')
export class LobbyController {
  constructor(private readonly lobbyService: LobbyService) {}
  @Get()
  getLobbyInfo() {
    const rooms = this.lobbyService.getRooms();
    let sum = 0;
    rooms.forEach((members, roomId) => (sum = sum + members.size));
    return {
      rooms: rooms.size,
    }
  }
}

```

```

        members: sum,
    };
}
}
// src/lobby/lobby.module.ts
import { Module } from '@nestjs/common';
import { LobbyService } from './lobby.service';
import { LobbyController } from './lobby.controller';
import { LobbyGateway } from './lobby.gateway';
@Module({
  providers: [LobbyService, LobbyGateway],
  controllers: [LobbyController],
})
export class LobbyModule {}
// src/lobby/lobby.service.ts
import { Injectable } from '@nestjs/common';
@Injectable()
export class LobbyService {
  private rooms = new Map();
  getRooms() {
    return this.rooms;
  }
  joinRoom(roomId: string, memberId: string) {
    let members = this.rooms.get(roomId);
    if (!members) {
      members = new Set();
      this.rooms.set(roomId, members);
    }
    members.add(memberId);
    return Array.from(members);
  }
  leaveRoom(roomId: string, memberId: string) {
    const members = this.rooms.get(roomId);
    if (members) {
      members.delete(memberId);
    }
  }
}

```

```

    return Array.from(members);
  }
}
<template>
<v-app>
  <v-navigation-drawer color="red" v-model="sidebar" app>
    <v-list>
      <v-list-tile
        v-for="item in menuItems"
        :key="item.title"
        :to="item.path"
      >
        <v-list-tile-action>
          <v-icon>{{ item.icon }}</v-icon>
        </v-list-tile-action>
        <v-list-tile-content>{{ item.title }}</v-list-tile-content>
      </v-list-tile>
    </v-list>
  </v-navigation-drawer>
  <v-toolbar class="bg-indigo-lighten-1" app>
    <span class="hidden-sm-and-up">
      <v-toolbar-side-icon @click="sidebar = !sidebar"> </v-toolbar-side-icon>
    </span>
    <v-toolbar-title>
      <strong>{{ appTitle }}</strong>
    </v-toolbar-title>
    <v-spacer> </v-spacer>
    <v-toolbar-items class="hidden-xs-only">
      <v-btn
        flat=""
        v-for="item in menuItems"
        :key="item.title"
        :to="item.path"
      >
        <v-icon left dark>{{ item.icon }}</v-icon>
        {{ item.title }}
      </v-btn>
    </v-toolbar-items>
  </v-toolbar>
</v-app>
</template>

```

```

    </v-btn>
  </v-toolbar-items>
</v-toolbar>
<v-content style="margin-bottom: 60px">
  <router-view></router-view>
</v-content>
</v-app>
</template>
<script>
import { ref, onMounted } from 'vue';
import { useUserStore } from "@store/user";
export default {
  name: "App",
  setup() {
    const appTitle = "Meteor Meet";
    const sidebar = ref(false);
    const menuItems = [
      { title: "Home", path: "/home", icon: "home" },
      { title: "Info", path: "/info", icon: "face" },
      { title: "Organizations", path: "/organizations", icon: "face" },
      { title: "Sign Up", path: "/signup", icon: "face" },
      { title: "Sign In", path: "/login", icon: "lock_open" }
    ];
    const store = useUserStore();
    onMounted(async () => {
      try {
        await store.fetchUserData();
      } catch (error) {
        console.error(error);
      }
    });
    return {
      appTitle,
      sidebar,
      menuItems,
    };
  }
};

```

```

    },
  };
</script>
<style>
</style>
import { defineStore } from 'pinia'
import axiosInstance from "@/api/api";
export const useOrganizationStore = defineStore("organizations", {
  state: () => ({
  })),
  getters: {},
  actions: {
    async getAllOrganizations(page,limit) {
      try {
        return await axiosInstance.get(`/organizations/all?page=${page}&limit=${limit}`)
      }
      catch (error) {
        console.log(error)
      }
    },
    async getAllOrganizationsByRole(page,limit, role) {
      try {
        return
        axiosInstance.get(`/organizations/byRole?page=${page}&limit=${limit}&role=${role}`)
      }
      catch (error) {
        console.log(error)
      }
    },
    async createOrganization(data) {
      try {
        return await axiosInstance.post(`/organizations/create`, data)
      }
      catch (error) {
        console.log(error)
      }
    }
  }
});

```

```

    },
    async deleteOrganization(id) {
      try {
        return await axiosInstance.delete(`/organizations/delete/${id}`, )
      }
      catch (error) {
        console.log(error)
      }
    }
  },
})
<template>
<div class="container">
  <div class="row">
    <div id="room" class="col-md-12 mt-3 ml-3">
      <vue-webrtc ref="webrtc"
        socketURL="http://localhost:3000"
        width="100%"
        :roomId="$route.params.id"
        :enableLogs="true"
        v-on:joined-room="logEvent"
        v-on:left-room="logEvent"
        v-on:opened-room="logEvent"
        v-on:share-started="logEvent"
        v-on:share-stopped="logEvent"
        @error="onError" />
    </div>
  </div>
  <div class="row">
    <div class="col-md-12" style="padding-top: 5px;">
      <h2>Captured Image</h2>
      <figure class="figure">
        
      </figure>
    </div>
  </div>
</div>

```

```

</div>
<div class="bottom-bar">
  <button type="button" class="btn btn-primary text-white" @click="onJoin">Join</button>
  <button type="button" class="btn btn-primary text-white" @click="onLeave">Leave</button>
  <button type="button" class="btn btn-primary text-white" @click="onCapture">Capture
Photo</button>
  <button type="button" class="btn btn-primary text-white" @click="onShareScreen">Share
Screen</button>
</div>
</template>
<script>
import { VueWebRTC } from 'vue-webrtc';
export default {
  name: 'room-component',
  components: {
    'vue-webrtc': VueWebRTC
  },
  data() {
    return {
      img: null,
      roomId: "public-room-v3"
    };
  },
  mounted: function () {
    this.$refs.webrtc.join();
  },
  computed: {
  },
  watch: {
  },
  methods: {
    onCapture() {
      this.img = this.$refs.webrtc.capture();
    },
    onLeave() {
      this.$refs.webrtc.leave();
    }
  }
}

```

```

    },
    onShareScreen() {
        this.img = this.$refs.webrtc.shareScreen();
    },
    onError(error, stream) {
        console.log('On Error Event', error, stream);
    },
    logEvent(event) {
        console.log('Event : ', event);
    },
}
};
</script>
<style>
.btn {
    margin-right: 8px;
}
.bottom-bar {
    position: fixed;
    bottom: 0;
    width: 100%;
    height: 8%;
    background-color: black;
    display: flex;

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для комунікації між користувачами

Виконав:

ст. гр. КНТ-121сп

Дмитро БЕЛИМЕНКО

Керівник:

д.т.н., професор

Андрій ОЛІЙНИК

Рисунок В.1 – Слайд 1

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес комунікації між користувачами.

Предмет дослідження – програмні засоби для комунікації.

Мета роботи – розробка програмного забезпечення для комунікації між користувачами.

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети необхідно було виконати такі завдання:

- виконати аналіз предметної області та програмних засобів для комунікації;
- здійснити проєктування програмного забезпечення для комунікації;
- створити програмне забезпечення для комунікації між користувачами;
- виконати тестування розробленого програмного забезпечення для комунікації між користувачами.

3

Рисунок В.3 – Слайд 3

Програмне забезпечення для комунікації

Критерій порівняння	Google meet	Skype	Slack	WhatsApp
Максимальна кількість учасників у безкоштовній версії	100	250	2	256
Наявність безкоштовної версії	+	+	+	+
Наявність демонстрації екрану	+	+	+	–
Запис розмови	–	+	–	–

Рисунок В.4 – Слайд 4

Обґрунтування вибору мови програмування

Критерій порівняння мов програмування	Мова програмування		
	JavaScript	Python	C#
Сфера застосування	Web-розробка	Web-розробка, штучний інтелект	Великі корпоративні проекти, розробка ПЗ для Window
Зручність для створення програмного забезпечення для комунікації між користувачами	+	+	+
Підтримка функціонального підходу до програмування	+	+	+
Швидкодія	+	–	+
Простота синтаксису	+	+	–

5

Рисунок В.5 – Слайд 5

Обґрунтування вибору фреймворку для серверної частини

Критерій порівняння фреймворків	Фреймворки		
	Nest	Django	.Net
Швидкість	Висока	Середня	Висока
Мова програмування	JavaScript	Python	C#
Тип архітектури	Подієва	Модель-представлення-контролер	Модель-представлення-контролер
Використання у високонавантажених додатках	+	+	+
Розширюваність	+	+	–
Однопоточність	+	–	–

6

Рисунок В.6 – Слайд 6

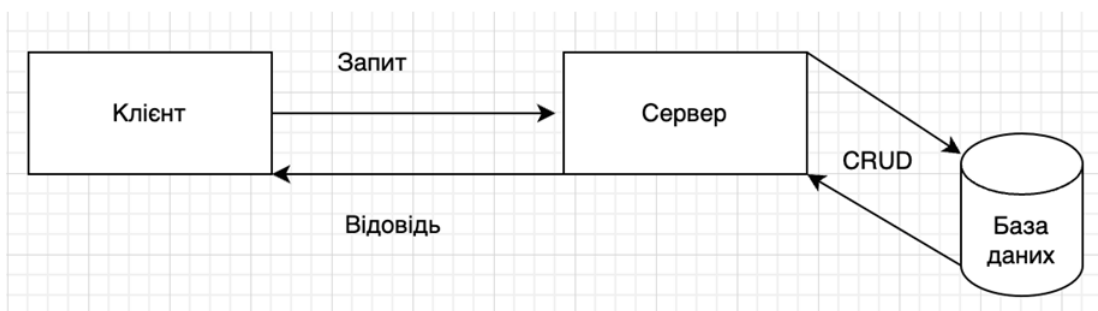
Обґрунтування вибору фреймворку для створення клієнту

Критерій порівняння фреймворків	Фреймворки		
	Vue	React	Angular
Підтримує однофайлові компоненти	+	+	–
Всі інструменти є з коробки за замовчуванням	+	–	+
низький поріг входу	+	–	–
Підтримка Virtual DOM	+	+	–

7

Рисунок В.7 – Слайд 7

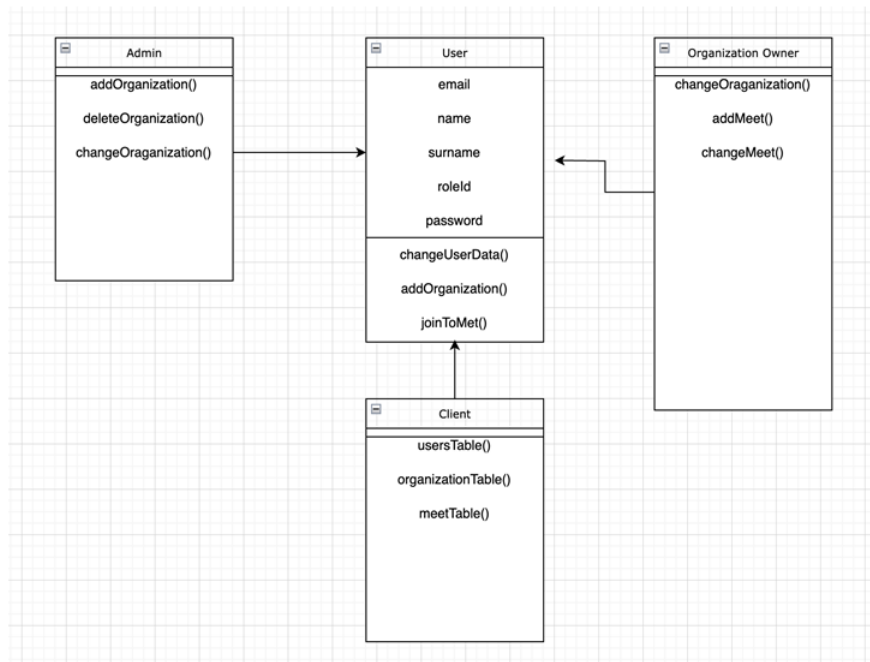
Архітектура програмного забезпечення для комунікації між користувачами



8

Рисунок В.8 – Слайд 8

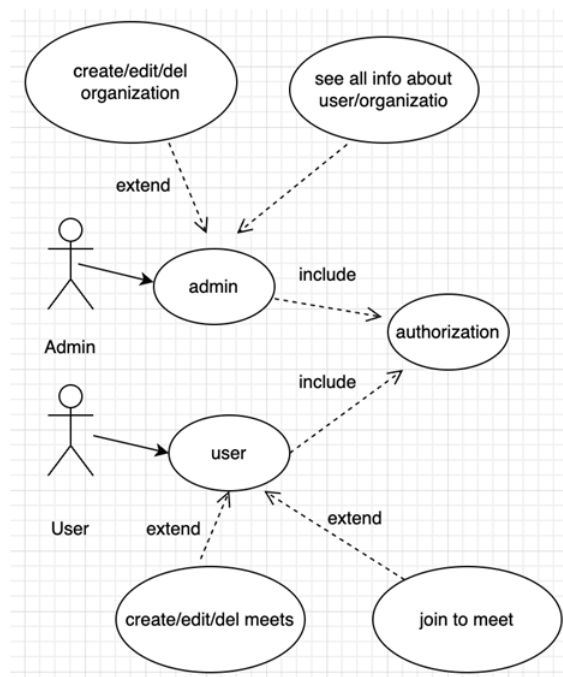
Структура програмного забезпечення для комунікації



9

Рисунок В.9 – Слайд 9

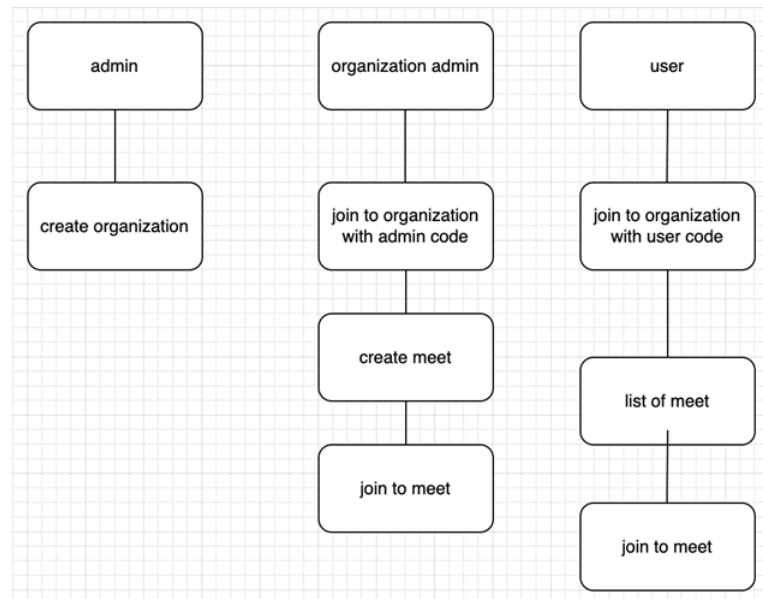
Схема функціонування програмного забезпечення для комунікації у вигляді діаграми прецедентів



10

Рисунок В.10– Слайд 10

Діаграма послідовностей програмного забезпечення обміну зображеннями



11

Рисунок В.11– Слайд 11

Сторінка логіну та інформації про акаунт

The screenshot shows the Meteor Meet application interface. At the top, there is a navigation bar with the logo 'Meteor Meet' and links for HOME, INFO, ORGANIZATIONS, SIGN UP, and SIGN IN. Below the navigation bar, there is a login form with fields for Email and Password, and a Login button. Below the login form, there is a section titled 'Account Information' displaying the following details:

- Email: admin@gmail.com
- Name: admin
- Surname: admin
- Role: admin

12

Рисунок В.12– Слайд 12

Сторінка організацій та додавання нової організації

The screenshot shows the 'Meteor Meet' interface with the 'ORGANIZATIONS' tab selected. A table lists existing organizations:

Name	Admin Code	User Code	Created At	Actions
test2	test2	test2	26/11/2023, 22:40:30	VIEW
admin	admin	user	08/01/2024, 00:04:15	VIEW

Below the table is a 'CREATE ORGANIZATION' button. To the right, there is a pagination control showing 'Items per page: 10' and '1-2 of 2'.

An 'Organization Profile' modal is open, containing three input fields: 'Name*', 'Admin Code*', and 'User Code*'. A note below the fields states '*Indicates required field'. At the bottom right of the modal are 'CLOSE' and 'SAVE' buttons.

13

Рисунок В.13– Слайд 13

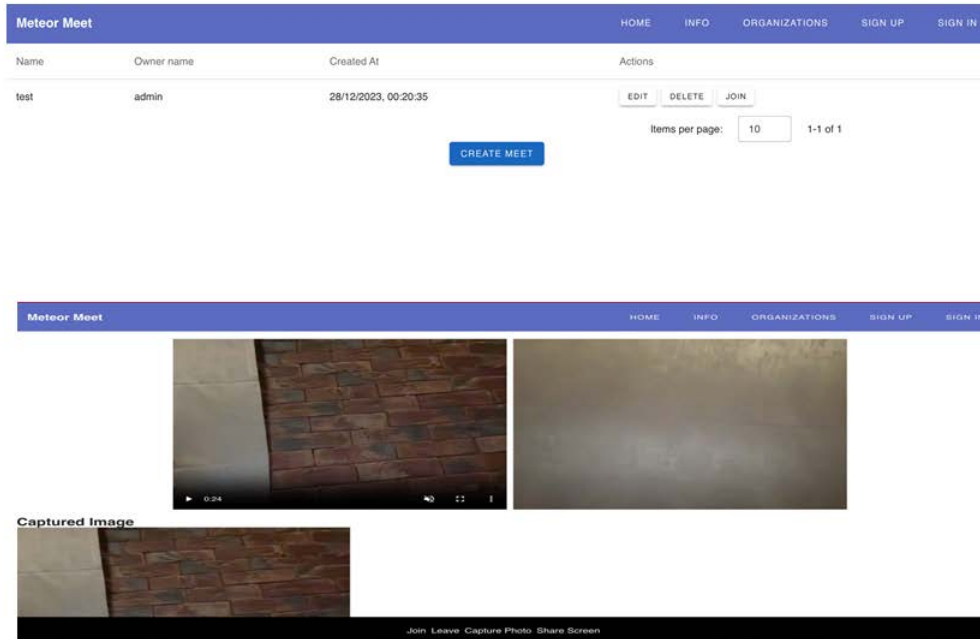
Модальне вікно для приєднання до організації

The screenshot shows the 'Meteor Meet' interface with the 'ORGANIZATIONS' tab selected. The table area displays 'No data available'. A 'Join Organization' modal is open, featuring an 'Organization Code' input field and 'JOIN' and 'CANCEL' buttons.

14

Рисунок В.14– Слайд 14

Сторінка зі списком зустрічей та сторінка зустрічі



15

Рисунок В.15– Слайд 15

ВИСНОВКИ

В результаті виконаної роботи було проаналізовано та досліджено додатки для комунікації та способи комунікації. Сформульовано функціональні вимоги до програмного забезпечення комунікації між користувачами. Для створення програмного забезпечення обміну зображеннями обрано мову програмування JavaScript. В якості фреймворку для створення програмного обміну зображеннями обрано фреймворк NestJS та VueJS.

Розроблено програмне забезпечення обміну зображеннями. Запропоновано структуру та описано основні модулі програмного забезпечення для комунікації між користувачами. Визначено, що основними класами програми є: Зустрічі (Meets), Користувач (User), Адміністратор (Admin), Таблиці (Tables) та ін. Основними компонентами є база даних та набір програмних модулів для забезпечення комунікації.

Виконано тестування розробленого програмного забезпечення для комунікації між користувачами. За результатами тестування програмного забезпечення не виявлено помилок, збоїв або інших серйозних проблем, тому можна зробити висновок, що тестування було успішним, а програмне забезпечення готове до використання.

16

Рисунок В.16– Слайд 16