НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

«До захисту допущено»
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
“___”_____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Веб-застосунок для VoIP-комунікації на основі SIP-протоколу”

Виконав: студент 4 курсу, групи ТР-11

Боднар Орест Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н, доцент Володимир ТИХОХОД _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент доцент каф. ТАЕ , к.т.н, доцент Артур РАЧИНСЬКИЙ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

БОДНАРЮ Оресту Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи **Веб-застосунок для VoIP-комунікації на основі SIP-протоколу**

керівник роботи ***Тихоход Володимир Олександрович, к.т.н., доцент***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» червня 2025 р. № 1875-с

2. Термін подання роботи студентом 09.06.2025

3. Вихідні дані до роботи мова програмування Java, TypeScript, фреймворки Spring, React, СКБД MongoDB, сервер автризиції Keycloak, комунікаційна платформа Asterisk.

4. Перелік питань, які потрібно розробити:
1) охарактеризувати задачу розробки веб-застосунку для VoIP-комунікації;
2) виконати аналіз технологій голосового зв’язку через Інтернет та існуючих рішень у цій сфері;
3) обґрунтувати вибір інструментів, технологій та алгоритмів для реалізації

програмного забезпечення;
4) розробити веб-застосунок для VoIP-комунікації з використанням зазначених інструментів.
5) пердставити процес застосування веб-застосунку.

5. Орієнтований перелік ілюстративного матеріалу: аналіз ринку VoIP-комунікації, аналіз існуючих рішень, функціональні можливості системи, основні VoIP-протоколи, порівняння протоколів SIP і H.323, програмні засоби реалізації, архітектура системи, схема бази даних.

6. Дата видачі завдання 13.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	20.02.2025	Виконано
2.	Аналіз методів та засобів розв'язання задачі	17-20.04.25	Виконано
3.	Розробка архітектури та загальної структури системи	21-25.04.25	Виконано
4.	Розробка окремих підсистем	25.04-02.05.25	Виконано
5.	Програмна реалізація системи	03-07.05.25	Виконано
6.	Оформлення пояснювальної записки	08-11.05.25	Виконано
7.	Захист програмного забезпечення	12-16.05.25	Виконано
8.	Передзахист	26-28.05.25	Виконано
9.	Захист	16-20.06.2025	Виконано

Студент _____ **Орест БОДНАР**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Володимир ТИХОХОД**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 77 сторінках, містить 26 ілюстрацій, 6 таблиць, 1 додаток, 26 джерел в переліку посилань.

Мета роботи – розробка захищеного, автономного веб-застосунку для VoIP-комунікації з використанням протоколу SIP.

Методи та засоби: SIP-протокол для ініціації голосових викликів, серверна логіка на базі Java та фреймворку Spring Boot, клієнтська частина на основі React і TypeScript, база даних MongoDB, авторизація через Keycloak, маршрутизація викликів за допомогою Asterisk.

Результат – функціональний веб-застосунок, який дозволяє здійснювати захищені голосові виклики через браузер без передачі даних третім сторонам, із підтримкою автентифікації, керування користувачами та модульної архітектури.

Ключові слова: SIP, VOIP, WEBRTC, ВЕБ-ЗАСТОСУНОК, АВТОНОМНІСТЬ.

ABSTRACT

The thesis consists of 77 pages, includes 26 figures, 6 tables, 1 appendix, and 26 references.

Purpose: to develop a secure, autonomous web application for VoIP communication using the SIP protocol.

Methods and tools: SIP protocol for session initiation, server-side logic built with Java and the Spring Boot framework, client-side implemented with React and TypeScript, MongoDB as the database, Keycloak for user authentication and authorization, and Asterisk for call routing and SIP communication handling.

The result: A functional web application enabling secure voice calls via a browser without transferring user data to third parties, featuring user management, modular architecture, and support for real-time interaction.

Keywords: SIP, VOIP, WEBRTC, WEB APPLICATION, AUTONOMY.

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

VoIP (Voice over IP) - технологія, що дозволяє здійснювати голосові виклики через Інтернет.

SIP (Session Initiation Protocol) - протокол встановлення, модифікації та завершення сеансів зв'язку, включаючи голосові та відеовиклики, через мережу Інтернет.

NAT traversal - технології для встановлення мережевих з'єднань, коли один або обидва клієнти знаходяться за NAT (Network Address Translation).

STUN (Session Traversal Utilities for NAT) - протокол для виявлення зовнішньої IP-адреси та порту клієнта, що знаходиться за NAT.

TURN (Traversal Using Relays around NAT) - протокол-розширення STUN для ретрансляції трафіку у випадках, коли пряме з'єднання через NAT неможливе.

TLS (Transport Layer Security) - криптографічний протокол, що забезпечує безпечну передачу даних між клієнтом і сервером.

2FA (Two-Factor Authentication) - двофакторна аутентифікація, метод захисту облікового запису, що вимагає надання двох різних факторів підтвердження особи.

UDP (User Datagram Protocol) - протокол без встановлення з'єднання, що використовується для передачі даних в IP-мережах з мінімальними затримками, але без гарантії доставки.

ITU (International Telecommunication Union) - Міжнародний союз електрозв'язку, спеціалізована установа ООН, що займається стандартизацією в галузі інформаційно-комунікаційних технологій.

IP-АТС (IP Private Branch Exchange) - IP-телефонна станція, приватна телефонна станція, що використовує IP-мережі для передачі голосових даних.

ВСТУП

У зв’язку зі зростанням популярності мобільного Інтернету та гібридних форм роботи збільшується потреба в ефективних та захищених засобах цифрової комунікації. Розвиток веб-технологій, а також загальні тенденції децентралізації цифрових сервісів стимулюють організації шукати автономні рішення для голосового зв’язку [1]. Однією з таких технологій є VoIP (Voice over IP), що дозволяє передавати голосові дані через мережу Інтернет, не покладаючись на традиційну телефонну інфраструктуру.

Популярність VoIP-технологій зумовлена їхньою гнучкістю, можливістю побудови приватної внутрішньої інфраструктури зв’язку, а також зниженням залежності від сторонніх провайдерів [2]. Для реалізації таких систем широко використовується протокол SIP (Session Initiation Protocol), що слугує основою побудови гнучких і масштабованих голосових сервісів. Проте аналіз наявних ринкових рішень демонструє, що багато з них не відповідають сучасним вимогам щодо захисту даних, автономного розгортання або простоти інтеграції, особливо в контексті малих організацій чи волонтерських ініціатив.

Актуальність теми дипломної роботи обумовлена потребою у створенні відкритого, захищеного веб-застосунку для VoIP-комунікації, який може бути розгорнутий без залежності від сторонніх сервісів та забезпечує конфіденційність зв’язку в браузерному середовищі.

Метою роботи є розробка безпечного веб-застосунку для голосової SIP-комунікації, що функціонує через інтерфейс сучасних веб-браузерів без встановлення додаткового програмного забезпечення, підтримує автономне розгортання та просту інтеграцію в інфраструктуру малих організацій.

Для досягнення поставленої мети передбачено виконання таких завдань:

- проаналізувати існуючі підходи, методи та алгоритми організації VoIP-комунікації;
- здійснити огляд готових VoIP-рішень та визначити їхні обмеження щодо безпеки та автономності;

– обґрунтувати вибір інструментів для реалізації клієнтської та серверної частин системи;

– розробити інтерфейс веб-клієнта для ініціації та отримання голосових дзвінків у режимі реального часу;

– реалізувати серверну логіку обробки дзвінків, зберігання даних користувача та системи автентифікації;

– провести тестування програмного забезпечення, оцінити його продуктивність, надійність та зручність використання.

Структура дипломної роботи включає шість основних розділів:

– у першому розділі «Постановка задачі розробки веб-застосунку для VoIP-комунікації» окреслено мету, завдання роботи та формалізовано вимоги до програмного забезпечення;

– у другому розділі “Технології голосового зв’язку через Інтернет” розглянуто архітектуру та особливості роботи SIP-протоколу й VoIP-технологій загалом;

– у третьому розділі “Огляд існуючих рішень для VoIP-комунікації” проаналізовано сучасні VoIP-платформи, їхні функції та недоліки;

– у четвертому розділі “Засоби програмної реалізації” обґрунтовано вибір мов програмування, фреймворків та бібліотек;

– у п’ятому розділі “Опис програмної реалізації” описано клієнтську, серверну частини застосунку та особливості архітектури;

– шостий розділ “Робота користувача з програмною системою” присвячено прикладам сценаріїв використання та оцінці зручності інтерфейсу.

Обсяг дипломної роботи складає 77 сторінкок, містить 26 ілюстрацій, 6 таблиць і 1 додаток. Робота написана українською мовою.

1 ПОСТАНОВКА ЗАВДАЧІ РОЗРОБКИ ВЕБ ЗАСТОСУНКУ ДЛЯ VOIP КОМУНІКАЦІЇ

У розділі сформульовано постановку завдання на розробку веб-орієнтованої системи для VoIP-комунікацій з використанням протоколу SIP. Подається технічне завдання, що окреслює загальну мету проєкту, вимоги до архітектури, ключових компонентів та безпекових механізмів системи. Визначаються функціональні вимоги, які описують основні можливості й поведінку застосунку з точки зору кінцевих користувачів. Завершують розділ нефункціональні вимоги, що висвітлюють очікування щодо продуктивності, масштабованості, безпеки та підтримуваності системи.

1.1 Технічне завдання

У межах дипломного проєкту ставиться завдання розробити безпечний, модульний та автономний веб-застосунок для здійснення VoIP-комунікацій із використанням протоколу SIP. Застосунок орієнтовано передусім на малі організації, які потребують локального рішення для голосового зв’язку, що забезпечує високий рівень безпеки, контроль над інфраструктурою та простоту використання навіть за відсутності спеціальзованої IT-підготовки користувача.

Система має реалізовувати повноцінний цикл SIP-комунікації, що включає:

- реєстрацію користувачів на сервері;
- ініціацію та встановлення голосових сесій;
- передачу голосових даних у реальному часі;
- завершення викликів.

Архітектура системи передбачає наявність таких основних компонентів:

- веб-клієнт — забезпечує користувацький інтерфейс та взаємодію через браузер;

- серверна частина — обробляє запити, виконує бізнес-логіку та забезпечує безпеку;
- SIP-сервер — відповідає за встановлення, маршрутизацію та завершення VoIP-сесій;
- база даних — зберігає інформацію про користувачів, налаштування та історію викликів;
- система автентифікації — контролює доступ і розмежування прав користувачів.

Особливу увагу необхідно приділити модульності архітектури, яка дозволяє масштабувати застосунок, додавати нові функціональні блоки та оновлювати окремі компоненти без порушення цілісності системи.

Основним інтерфейсом користувача є сучасний веб-додаток, що працює у популярних браузерах (з підтримкою WebRTC) без потреби встановлення додаткового програмного забезпечення. Користувачі повинні мати змогу здійснювати та приймати SIP-виклики в реальному часі. Веб-клієнт також має надавати доступ до персональних налаштувань, історії дзвінків, контактної книги, функціоналу управління обліковим записом. Ключовою вимогою є інтуїтивно зрозумілий інтерфейс із мінімальною кількістю кроків для виконання основних дій.

Ядром голосової інфраструктури виступає VoIP-сервер як окремий компонент, що обробляє запити на встановлення, маршрутизацію та завершення SIP-сесій. Його конфігурація має відповідати сучасним вимогам до безпеки та бути сумісною з веб-клієнтом. Передбачена підтримка таких функцій, як реєстрація SIP-клієнтів, маршрутизація дзвінків, NAT traversal (STUN/TURN), а також можливість масштабування до кількох одночасних викликів.

Для зберігання інформації про користувачів, записи викликів, налаштування та профілів планується використання документно-орієнтованої бази даних, що забезпечує гнучкість у структурі збережених даних.

Передбачено централізовану систему ідентифікації користувачів із підтримкою безпечного зберігання облікових даних, використанням хешування паролів та можливістю розширення до двофакторної аутентифікації (2FA). Ролі

користувачів дозволять розділити повноваження між адміністраторами та звичайними користувачами.

Особлива увага приділяється захисту даних та комунікацій. SIP-сигналізація має бути захищена через TLS. Всі внутрішні запити мають бути захищені через HTTPS, а дані у базі мають бути захищені на рівні зберігання.

1.2 Функціональні вимоги

Безпечний, модульний, самостійно розгорнутий веб-застосунок для VoIP повинен задовольняти низку функціональних вимог, щоб ефективно забезпечувати комунікаційні потреби сучасних організацій. Ці вимоги охоплюють управління користувачами, управління дзвінками, функції безпеки та інші важливі функціональні характеристики.

Управління користувачами є важливим для контролю доступу до системи. Застосунок повинен надавати механізм для реєстрації користувачів і створення їхніх облікових записів. Цей процес має супроводжуватися надійною аутентифікацією користувачів, зокрема підтримкою сильних паролів і, за потреби, багатофакторної аутентифікації для посилення безпеки. Користувачі повинні мати можливість керувати своїми профілями, оновлюючи особисту інформацію та налаштування. Крім того, система повинна впроваджувати ролі користувачів та дозволи, щоб контролювати доступ до різних функцій і адміністративних функцій залежно від ролі користувача в організації.

Функції управління дзвінками є основою роботи застосунку. Користувачі повинні мати можливість ініціювати та отримувати VoIP дзвінки безперешкодно через веб-інтерфейс. Необхідні функції дзвінків, такі як утримання та відновлення дзвінка, а також перенаправлення дзвінка, повинні підтримуватися. Система також повинна забезпечувати можливість переадресації дзвінків, дозволяючи користувачам перенаправляти вхідні дзвінки в залежності від умов. Потрібна також підтримка прямого набору телефонних номерів та набору розширень всередині організації.

Безпека є критично важливою для цього застосунку, зважаючи на необхідність забезпечення безпечної комунікації користувачів. Система повинна гарантувати захист SIP сигналізації за допомогою шифрування через протокол TLS (Transport Layer Security).. Система також повинна реалізувати надійні механізми аутентифікації для користувачів та SIP точок доступу, щоб підтвердити їхні особи.

1.3 Нефункціональні вимоги

Окрім реалізації функціональних можливостей, веб-застосунок для VoIP-комунікацій повинен відповідати низці нефункціональних вимог, які визначають його якість, надійність, продуктивність і безпеку. Ці вимоги мають критичне значення для ефективного використання системи в умовах малих організацій з обмеженими IT-ресурсами (таб. 1.1).

Однією з основних категорій виступає безпека, яка включає захист переданих і збережених даних, а також механізми аутентифікації, контролю доступу та аудит подій. Це дозволяє запобігти несанкціонованому доступу та забезпечити довіру до системи.

Продуктивність оцінюється через ключові параметри голосового зв’язку в реальному часі: затримку, джиттер, втрату пакетів і час встановлення виклику. Всі вони повинні відповідати міжнародним рекомендаціям для забезпечення якісного аудіозв’язку.

Доступність і надійність вимагають стабільної роботи системи з мінімальними перервами навіть у випадках зростання навантаження, що критично для організацій, які використовують VoIP як основний канал комунікації.

Масштабованість передбачає здатність системи адаптуватися до зростаючої кількості користувачів без погіршення продуктивності.

Окрему увагу слід приділити зручності, зокрема простоті інтерфейсу та адаптивності застосунку до різних пристроїв, що підвищує його доступність для користувачів без технічної підготовки.

Нарешті, підтримуваність забезпечується за рахунок модульної архітектури та належного документування, що спрощує оновлення, обслуговування й розвиток системи у довгостроковій перспективі.

Таблиця 1.1 — Нефункціональні вимоги до веб-застосунку

Категорія	Вимога	Цільовий показник	Обґрунтування
Безпека	Конфіденційність	Використання TLS для SIP; HTTPS для клієнта	Захист даних користувачів і комунікацій від перехоплення
	Аутентифікація	Паролі з хешуванням і підтримка 2FA	Запобігання несанкціонованому доступу
	Контроль доступу	Модель RBAC (розмежування ролей)	Розмежування прав між адміністратором і користувачами
Продуктивність	Затримка (One-way delay)	≤ 150 мс	Природність розмов, відповідно до ITU-T G.114
	Джиттер (варіація затримки)	≤ 30 мс	Плавність звучання голосу
	Втрата пакетів	≤ 3%	Якість аудіо без спотворень і пропусків
	Час встановлення виклику	≤ 2 с	Швидка реакція при ініціації виклику
Масштабованість	Кількість користувачів/викликів	Підтримка зростання без деградації	Адаптація до організацій, які розширюються
Надійність	Стабільність при навантаженнях	Відсутність збоїв у типових сценаріях	Плавна робота без переривань
Зручність	Інтерфейс	Інтуїтивна логіка, простота навігації	Придатність для користувачів без технічного досвіду
	Адаптивність	Робота в мобільних і десктоп-браузерах	Гнучкість в умовах різних пристроїв
Підтримуваність	Архітектура	Модульність, розділення функцій	Спрощення оновлення, усунення помилок
	Документування	Код і API задокументовані	Забезпечення легкого супроводу та розвитку системи

У підсумку, нефункціональні вимоги до веб-застосунку VoIP забезпечують високий рівень безпеки, продуктивності, масштабованості та зручності використання. Завдяки модульному дизайну та хорошій підтримуваності система може ефективно адаптуватися до зростаючих потреб організацій, слугуючи надійною, безпечною та економічною альтернативою стороннім сервісам.

2 ТЕХНОЛОГІЇ ГОЛОСОВОГО ЗВ'ЯЗКУ ЧЕРЕЗ ІНТЕРНЕТ

У розділі оглянуто ключові технології, що лежать в основі розробки веб-орієнтованої системи IP-телефонії (VoIP — Voice over Internet Protocol). Розглядається концепція та принцип дії VoIP, зокрема те, як голосові сигнали передаються через IP-мережі на відміну від традиційних комутаційних систем. Далі наводяться основні протоколи зв'язку, які найчастіше використовуються в реалізаціях VoIP, з акцентом на їх функціональне призначення та порівняльні переваги. Окремий підрозділ присвячено протоколу SIP (Session Initiation Protocol), що виконує критично важливу роль у механізмах сигналізації VoIP-архітектури. На завершення подається огляд сучасних вебтехнологій, зокрема WebRTC, які забезпечують можливість комунікації в реальному часі безпосередньо у веббраузерах, створюючи підґрунтя для інтеграції SIP у вебзастосунки.

2.1 Механізм функціонування VoIP

Технологія голосового зв'язку через Інтернет (Voice over Internet Protocol, VoIP) є однією з найважливіших інновацій, що докорінно трансформувала сферу голосової комунікації. У своїй основі VoIP дозволяє як окремим користувачам, так і організаціям здійснювати голосові виклики за допомогою широкосмугового Інтернет-з'єднання, фактично усуваючи потребу у використанні традиційних аналогових телефонних ліній. Це забезпечується шляхом застосування Інтернет-протоколу (IP) — базового комунікаційного протоколу для передавання даних у глобальній мережі Інтернет. У результаті голосова комунікація переходить від традиційної телефонії, заснованої на комутації каналів, до пакетної комутації, що є фундаментом функціонування Інтернету [3]. Така парадигмальна зміна має

вагомі наслідки з огляду на економічну ефективність, гнучкість в експлуатації, а також розширення функціональних можливостей, доступних для користувачів.

Функціонування VoIP-системи розпочинається з процесу оцифрування голосового сигналу, що є ключовим етапом передачі мовлення цифровими каналами (рис. 2.1). Це передбачає перетворення аналогових сигналів — безперервних звукових коливань, що виникають під час мовлення, — у цифровий формат, придатний для обробки комп'ютерними пристроями та передавання мережею Інтернет. Така трансформація також відома як кодування мовлення. З метою підвищення ефективності передавання і зменшення використання смуги пропускання, застосовуються алгоритми стиснення аудіо, що дозволяють зменшити обсяг цифрових даних. Таким чином, процес оцифрування є базовим кроком, який забезпечує можливість передавання мовлення у цифровому середовищі, перетворюючи безперервний аналоговий сигнал у дискретні цифрові пакети.

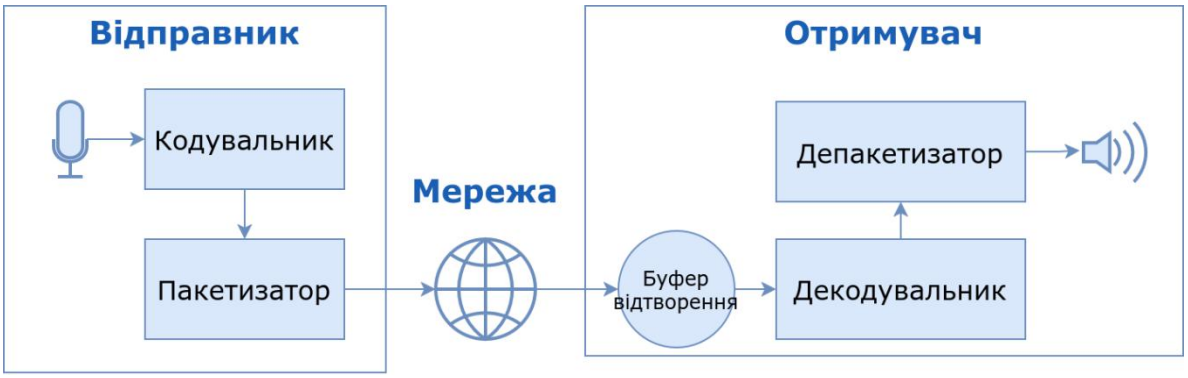

Рисунок 2.1 — Наскрізні компоненти VoIP

Наступним етапом є пакетування цифрових голосових даних. У рамках цього процесу оцифрована інформація розбивається на менші структурні одиниці — пакети. Кожен пакет доповнюється службовими заголовками, які містять критично важливу інформацію, зокрема IP-адреси джерела та одержувача, а також номери послідовності, що забезпечують правильне відновлення порядку даних на приймальному боці [4]. Пакетування є необхідною умовою ефективного передавання мовлення через мережі на основі Інтернет-протоколу, оскільки дає

змогу маршрутизувати дані незалежно один від одного, що підвищує стійкість до затримок і втрат у мережі.

Після пакетування пакети голосових даних передаються мережею Інтернет із використанням Інтернет-протоколу (IP). Під час цього вони проходять через складну систему маршрутизаторів і комутаторів, кожен з яких приймає рішення щодо найефективнішого шляху доставки до кінцевого пункту призначення.

У сучасних VoIP-системах використовується принцип пакетної комутації, згідно з яким мовленнєві сигнали обробляються як звичайні дані та маршрутизуються відповідно. Для забезпечення передачі мовлення в реальному часі найчастіше застосовується протокол User Datagram Protocol (UDP), який завдяки мінімальному службовому навантаженню забезпечує низьку затримку. На базі UDP додатково використовується протокол Real-time Transport Protocol (RTP), який виконує функції маркування пакетів номерами послідовності та часовими мітками, що критично важливо для збереження цілісності та синхронізації мовленнєвих даних [5].

На завершальному етапі пакети даних, що надійшли до місця призначення, проходять процес депакетизації та повторного збирання. Пакети агрегуються та впорядковуються згідно з початковою послідовністю, що забезпечує відновлення цілісного цифрового голосового сигналу. При цьому службова інформація, додана під час етапу пакетування, видаляється. Для компенсації варіацій у часі прибуття окремих пакетів (явище джитеру) застосовується буферизація, яка сприяє згладженню аудіопотоку. Процес повторної збірки пакету є критично важливим для коректного відтворення мовлення, а буферизація джитеру істотно впливає на стабільність та якість звуку, незважаючи на потенційні флуктуації мережевого трафіку.

Завершальним етапом у функціонуванні VoIP-системи є зворотне перетворення цифрових голосових даних у аналогові звукові хвилі. Цей процес, відомий як відтворення мовлення або декодування, забезпечує сприйняття інформації людиною за допомогою стандартних аудіопристроїв. У випадках, коли в інфраструктурі VoIP використовуються традиційні аналогові телефони, ключову роль відіграють аналогово-цифрові адаптери (Analog Telephone Adapters, ATA). Ці

пристрої виконують необхідне перетворення сигналів, слугуючи містком між цифровим середовищем VoIP та аналоговими технологіями класичної телефонії. Подібну функцію виконують і VoIP-шлюзи, які забезпечують взаємодію між різними типами мереж та здійснюють перетворення як з аналогового в цифровий формат (ADC), так і в зворотному напрямку (DAC) [3]. Цей завершальний етап гарантує, що цифрові голосові дані повертаються у форму, зрозумілу людині за допомогою звичайного аудіообладнання.

Запровадження VoIP-технологій забезпечує численні переваги:

– суттєве зниження витрат завдяки меншим тарифам на дзвінки та зменшенню залежності від інфраструктури традиційної телефонії;

– сучасні VoIP-системи пропонують широкий спектр функцій, які виходять за межі базових голосових викликів: відеоконференції, обмін миттєвими повідомленнями, інтегровані календарі нарад тощо. VoIP також легко інтегрується з програмним забезпеченням для управління взаємовідносинами з клієнтами (CRM) та іншими бізнес-додатками;

– високий рівень мобільності та гнучкості цих систем створює сприятливі умови для віддаленої роботи, дозволяючи співробітникам залишатися на зв’язку практично з будь-якого місця, де є доступ до Інтернету;

– завдяки масштабованості VoIP-рішення легко адаптуються до змін у потребах підприємств щодо кількості телефонних ліній;

– вилучення потреби в окремих телефонних лініях і дорогому обладнанні PBX, що характерне для традиційних систем, ще більше підсилює економічні вигоди VoIP. При цьому за наявності стабільного та швидкого Інтернет-з’єднання, VoIP здатен забезпечити якість зв’язку, що не поступається, а іноді й перевершує традиційну телефонію.

Водночас VoIP має низку обмежень:

– залежність від Інтернет-з’єднання означає, що у випадку його недоступності зв’язок буде перервано;

– якість аудіо може погіршуватись через затримки, фонові шуми, відлуння — усе це залежить від доступної смуги пропускання та загального стану мережі;

– питання безпеки також є актуальним, оскільки VoIP, як інтернет-технологія, може бути вразливим до загроз на кшталт крадіжки особистих даних, атак шкідливого ПЗ та перехоплення комунікацій;

– додаткові витрати можуть виникати при здійсненні дзвінків на номери, які не належать до тієї ж VoIP-мережі;

– можливі також проблеми сумісності з деяким устаткуванням, таким як старі охоронні системи або факсимільні апарати, які використовують аналогові сигнали [2].

Забезпечення функціонування VoIP стало можливим завдяки скоординованій роботі низки комунікаційних протоколів. Серед них центральне місце займає протокол ініціації сесій (Session Initiation Protocol, SIP), який є універсальним стандартом для встановлення, управління та завершення VoIP-сеансів між пристроями. Після встановлення з'єднання, за передавання мультимедійного вмісту, включно з мовленням і відео, відповідає протокол RTP (Real-time Transport Protocol), який гарантує своєчасну доставку пакетів для забезпечення плавності відтворення. Доповнює його протокол опису сесії (Session Description Protocol, SDP), що у текстовому форматі передає параметри мультимедійної сесії: тип носія, використовувані кодеки, мережеві адреси тощо. SDP часто вбудовується у SIP-повідомлення, спрощуючи узгодження параметрів зв'язку. Хоча SIP на сьогодні домінує у сфері VoIP, раніше активно використовувався стандарт H.323, розроблений Міжнародним союзом електрозв'язку (ITU) для передачі голосу та відео через IP-мережі [4].

2.2 Протокол ініціації сеансів

Протокол ініціації сеансів (Session Initiation Protocol, SIP) є базовим стандартом, який лежить в основі сучасних VoIP-систем, функціонуючи як сигнальний протокол прикладного рівня, що відповідає за встановлення, підтримку та завершення сеансів мультимедійного зв'язку в реальному часі через IP-мережі [5]. По суті, SIP визначає правила комунікації, що дозволяють

користувачам передавати голосові, відео- та інші форми цифрового мультимедіа через інтернет-з'єднання. Протокол охоплює весь життєвий цикл комунікаційної сесії: від ініціалізації до її можливих модифікацій під час розмови та остаточного завершення. Однією з ключових переваг SIP є його інтуїтивний дизайн, завдяки якому користувачі можуть без суттєвого навчання переходити від традиційної телефонії до VoIP-систем.

Протокол SIP функціонує у зв'язці з іншими протоколами, формуючи комплексну архітектуру VoIP-зв'язку (рис 2.2). Вона включає кілька критичних компонентів. Агенти користувача (User Agents), або кінцеві пристрої, — це телефони, комп'ютери або мобільні пристрої з SIP-додатками, що забезпечують VoIP-з'єднання. Ці агенти можуть виконувати функції клієнта (User Agent Client, UAC), який ініціює дзвінки, та сервера (User Agent Server, UAS), який відповідає на вхідні запити. Проксі-сервери (Proxy Servers) виконують роль посередників у SIP-комунікаціях, маршрутизуючи повідомлення між клієнтами та кінцевими точками, а також можуть реалізовувати мережеві політики, як-от перевірка прав користувача на ініціацію дзвінка. Деякі проксі, звані форкувальними (forking proxies), здатні надсилати один запит на кілька цільових адрес одночасно [6].

Реєстраційні сервери (Registrar Servers) виконують ключову роль в управлінні розташуванням користувачів. Коли агент надсилає запит REGISTER, сервер зчитує й обробляє ідентифікаційні дані пристрою, пов'язуючи його SIP-адресу з поточною IP-адресою. Ці дані використовуються для аутентифікації та правильного маршрутизування дзвінків. Сервери переадресації (Redirect Servers) отримують запити та повертають актуальну контактну інформацію цільового пристрою, спрямовуючи виклик до відповідної IP-адреси. Нарешті, контролери прикордонних сеансів (Session Border Controllers, SBCs) — це спеціалізовані пристрої, що розміщуються на межі мережі для регулювання потоків IP-комунікацій, вирішення питань безпеки, забезпечення сумісності протоколів та підтримки якості обслуговування.

Процес встановлення SIP-дзвінка включає певну послідовність обміну повідомленнями між абонентами. Пристрій ініціатора надсилає повідомлення INVITE, вказуючи на бажання встановити з'єднання. У відповідь пристрій

абонента зазвичай надсилає службове повідомлення 100 Trying, що свідчить про прийом запиту. Далі, під час сигналу виклику, надсилається відповідь 180 Ringing, яка інформує ініціатора, що виклик сповіщено. Коли абонент відповідає, система надсилає повідомлення 200 OK, що означає прийняття виклику. Після цього пристрій ініціатора надсилає підтвердження ACK, завершуючи процес встановлення сесії. Після обміну ACK-повідомленням розпочинається передача голосових даних у вигляді RTP-пакетів (Real-time Transport Protocol).

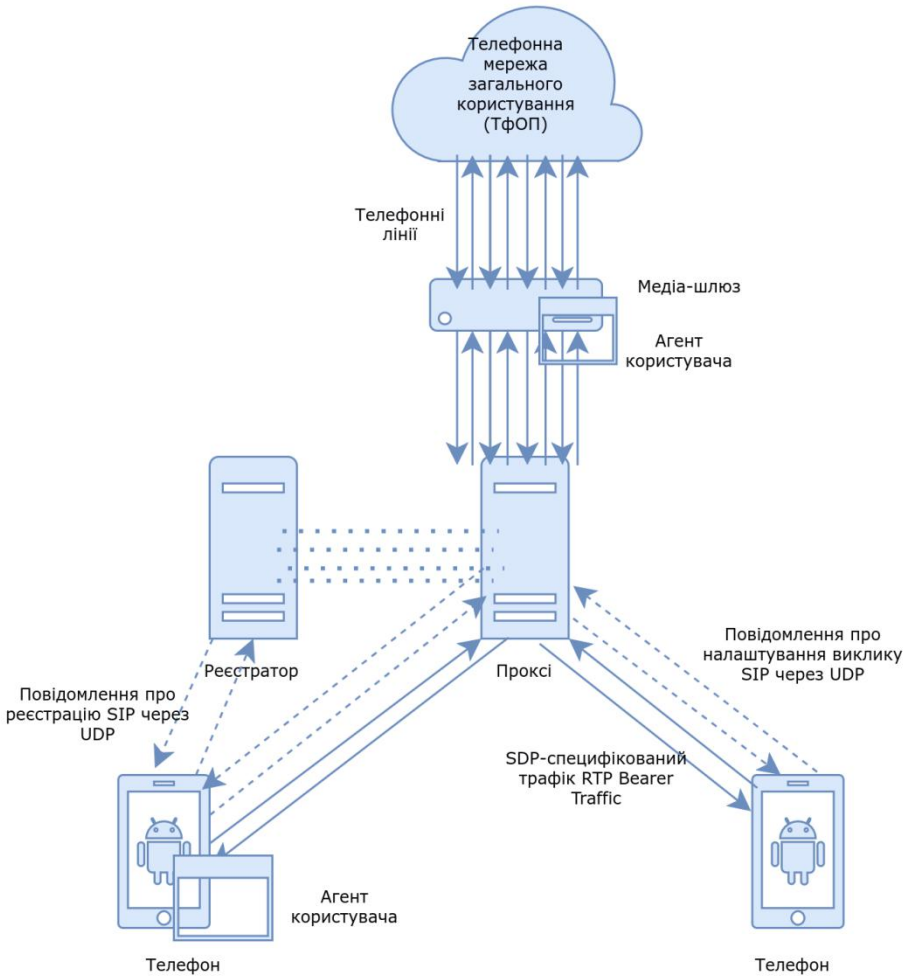

Рисунок 2.2 — Архітектура SIP

Окрім встановлення з’єднання, SIP відіграє важливу роль у його підтримці та завершенні. Під час активної розмови SIP може забезпечувати періодичний обмін сигналами для підтримки з’єднання, включно з оновленням параметрів мультимедійної сесії або зміною кількості учасників. Щоб уникнути зависання з’єднання у "мертвому" стані, SIP використовує таймери сесій (Session Timers),

які ініціюють регулярні повідомлення типу REINVITE або UPDATE з метою перевірки активності сторін. Для завершення виклику одна зі сторін надсилає запит BYE, а у відповідь отримує підтвердження у вигляді 200 OK, що фіксує завершення комунікації.

Під завершенням SIP-сесії також розуміється процес маршрутизації дзвінків від приватної телефонної мережі (PBX) або VoIP-системи до мережі загального користування (PSTN). Постачальники SIP-термінації забезпечують таку взаємодію, дозволяючи дзвінкам, ініційованим через IP-мережі, досягати кінцевих точок у традиційній телефонії або на інших SIP-сумісних пристроях.

Окрім базових повідомлень, необхідних для встановлення та завершення викликів, протокол SIP визначає низку додаткових типів повідомлень для обробки різних аспектів комунікаційної сесії. Повідомлення CANCEL використовується для припинення спроби встановлення сеансу до моменту з'єднання. Повідомлення REGISTER дозволяє агентам користувача повідомляти реєстраційні сервери про своє поточне розташування (ім’я хоста та IP-адресу). За допомогою повідомлення OPTIONS кінцеві пристрої можуть обмінюватися інформацією про свої технічні можливості, зокрема підтримувані кодеки. Разом з основними повідомленнями INVITE, ACK і BYE, ці типи формують ядро SIP-запитів, що забезпечують повний життєвий цикл VoIP-дзвінка [5].

Хоча SIP став домінантним протоколом сигналізації для VoIP, важливо розуміти його відмінності від інших протоколів (таб. 2.1), зокрема H.323 та MGCP. Протокол H.323, розроблений ITU-T до SIP, спочатку призначався для мультимедійного зв’язку через ISDN, а згодом був адаптований для IP-мереж. На відміну від нього, SIP був створений IETF як легкий і масштабований протокол. Архітектурно H.323 є монолітним і менш модульним. Він покладається на Gatekeeper для реалізації функцій контролю доступу та керування пропускною здатністю, тоді як SIP розподіляє інтелектуальні функції між різними серверами. Ще однією відмінністю є тип кодування: H.323 використовує бінарне кодування, що ускладнює діагностику проблем, тоді як SIP — текстове кодування у стилі HTTP, що спрощує налагодження. Завдяки цьому SIP вважається гнучкішим та зручнішим для впровадження у сучасних системах зв’язку.

Таблиця 2.1 — Порівняння VoIP-протоколів

Характеристика	SIP	H.323	MGCP
Орган стандартизації	IETF	ITU-T	IETF
Архітектура	Модульна, Peer-to-Peer (із серверами)	Монолітна, Peer-to-Peer	Клієнт/Сервер
Масштабованість	Висока	Обмежена	Масштабована для шлюзів
Тип кодування	Текстове (ASCII)	Бінарне	Текстове
Складність реалізації	Гнучкий, простий у впровадженні	Менш гнучкий, складніший	Просте налаштування шлюзів
Інтелект мережі	Сервери	Gatekeeper	Call Agent
Основне призначення	VoIP та мультимедійні сесії	Відеоконференції, застарілі VoIP	Контроль медіа-шлюзів для PSTN
Розмір повідомлень	Великий	Малий	Середній
Налагодження	Просте (текстові повідомлення)	Складне	Просте (через централізований контроль)
Сучасні VoIP-функції	Розширена підтримка	Обмежена підтримка	Базове керування викликами

Протокол MGCP (Media Gateway Control Protocol) реалізує іншу модель — клієнт/сервер, де центральний керуючий агент (Call Agent або Media Gateway Controller) здійснює контроль над медіа-шлюзами. SIP, навпаки, переважно ґрунтується на моделі «peer-to-peer», хоча деякі його компоненти також мають архітектуру клієнт/сервер. У MGCP самі шлюзи є менш інтелектуальними та залежать від Call Agent у питаннях логіки виклику. Цей підхід добре підходить для управління голосовими шлюзами, що забезпечують зв’язок із телефонною мережею загального користування (PSTN). У той час як SIP охоплює ширший спектр пристроїв, таких як IP-телефони, програмні клієнти та мультимедійні термінали.

Однією з переваг MGCP є спрощене налаштування шлюзів, оскільки логіка контролю централізована. Проте це створює потенційну точку відмови: при втраті зв’язку з Call Agent шлюзи можуть стати нефункціональними. У плані функціональності та гнучкості SIP пропонує значно ширший набір можливостей, ніж базові засоби керування викликами в MGCP [5].

2.3 Комунікація в режимі реального часу через Інтернет

Еволюція вебтехнологій істотно вплинула на сферу комунікацій у реальному часі, відкриваючи нові можливості для інтеграції голосових та відеофункцій у вебдодатки. Визначальною технологією у цій сфері є WebRTC (Web Real-Time Communication) — платформа, що дозволяє вебсайтам та вебзастосункам захоплювати та передавати аудіо й відео, а також обмінюватися довільними даними безпосередньо між браузерами без потреби у проміжних серверах або додаткових плагінах (рис. 2.3). Це досягається за допомогою набору взаємопов’язаних API та протоколів, які функціонують як єдина система.

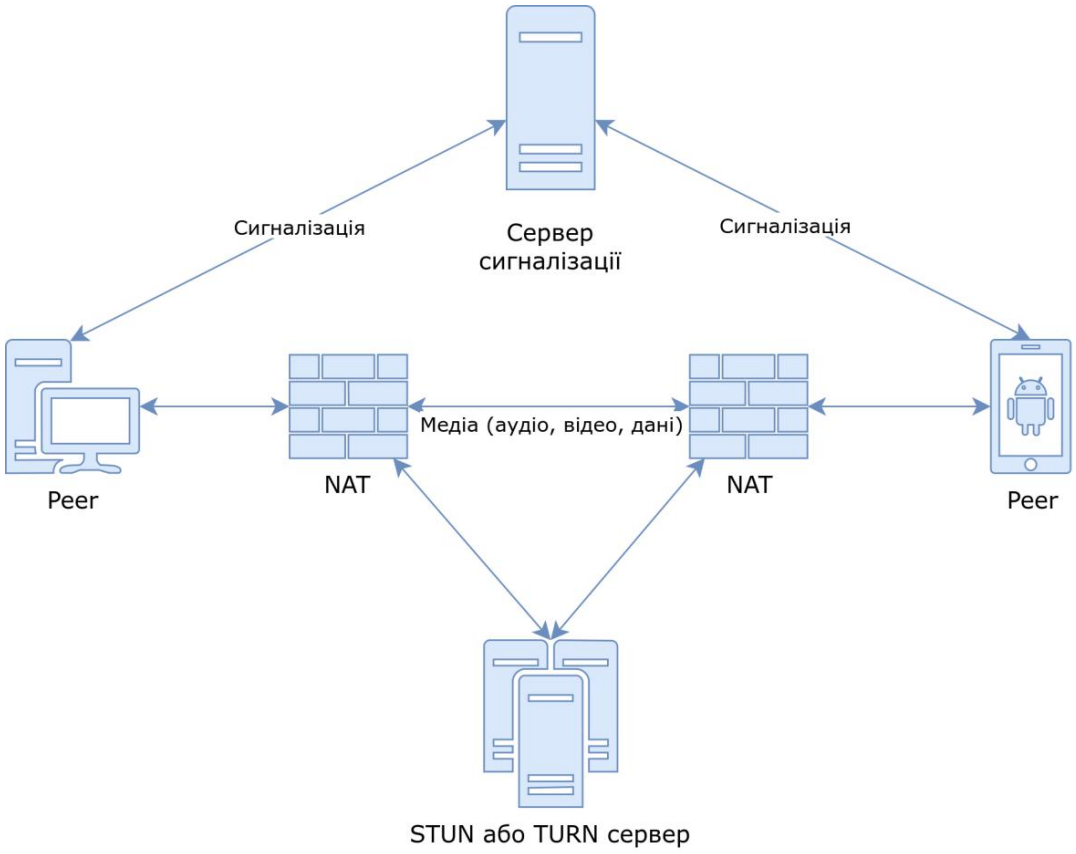

Рисунок 2.3 — Архітектура WebRTC

Основними компонентами WebRTC є:

- getUserMedia() — надає доступ до мікрофона та камери користувача;
- RTCPeerConnection — встановлює та підтримує аудіо- та відеоз’єднання між клієнтами;
- RTCDataChannel — забезпечує передачу даних і текстових повідомлень.

Завдяки потужній підтримці з боку сучасних веббраузерів WebRTC став основою для VoIP-викликів у браузері, усуваючи потребу у встановленні додаткового програмного забезпечення. Відомими прикладами застосунків, що використовують WebRTC, є Google Meet, Discord, а також численні телемедичні платформи, які потребують відео- та аудіозв'язку в реальному часі. Навіть корпоративні рішення, як-от ServiceNow, інтегрують WebRTC для покращення взаємодії в режимі реального часу — зокрема в функціях допомоги агенту, ескалації віртуального агента та віддаленої підтримки [8].

Ще однією критично важливою вебтехнологією для комунікації в реальному часі є WebSocket. На відміну від традиційної моделі HTTP, яка працює за принципом запит-відповідь, WebSocket забезпечує постійне, двонаправлене з'єднання через єдиний, тривалий канал. Після встановлення таке з'єднання дозволяє одночасну передачу та прийом даних, що робить його надзвичайно ефективним для реалізації чатів, онлайн-ігор, систем миттєвих сповіщень та інших застосунків з високими вимогами до швидкості обміну даними.

Процес починається з HTTP-handshake, який перетворюється у з'єднання WebSocket. Безпечні з'єднання використовують URI-схему wss://. У порівнянні з традиційними методами HTTP-polling, WebSocket має значно менші затримки та нижче навантаження на мережу, що забезпечує ефективність при тривалому обміні даними. Водночас, WebSocket-з'єднання є станозалежними, що може ускладнювати реалізацію, і не всі мережі чи старі браузери підтримують цю технологію через можливі конфлікти з проксі-серверами або міжмережевими екранами.

Крім основного призначення, WebSocket може також виконувати роль сигналізаційного каналу для застосунків WebRTC, забезпечуючи передачу необхідної мережевої та медіаінформації для встановлення прямого з'єднання між клієнтами в реальному часі [10].

Ще одним підходом до забезпечення комунікації в реальному часі у вебсередовищі є події, що надсилаються сервером (Server-Sent Events, SSE). Ця технологія дозволяє серверу надсилати клієнту оновлення в реальному часі через одне, постійне HTTP-з'єднання, яке працює в односторонньому режимі (від

сервера до клієнта). Клієнт ініціює з’єднання, створюючи об’єкт EventSource, що підключається до SSE-ендпоїнту на сервері. Після встановлення з’єднання сервер може постійно транслювати дані у вигляді подій, які обробляються браузером по мірі їх надходження.

Однією з ключових особливостей SSE є механізм автоматичного перепідключення: у випадку переривання з’єднання браузер автоматично намагається його відновити. Передача даних відбувається у певному форматі, де кожна подія починається з ключового слова data: і завершується подвійним перенесенням рядка \n\n. Завдяки своїй простоті SSE є ефективним рішенням для задач, де потрібні лише оновлення від сервера до клієнта, зокрема для відображення живих сповіщень, аналітичних панелей або потокової передачі результатів моделей штучного інтелекту. У таких випадках SSE часто розглядається як легковагова альтернатива WebSocket. Водночас, інтеграція SSE з існуючими системами безпеки може викликати певні труднощі. У контексті VoIP, SSE ефективно використовується для реалізації сповіщень у реальному часі та оновлень статусу користувачів [11].

Отже, розвиток технологій передачі даниї в реальному часу через інтернет створив широкі можливості для інтеграції голосового й відеозв’язку безпосередньо у вебінтерфейси. Серед проаналізованих підходів WebRTC вирізняється як найоптимальніше рішення для реалізації VoIP-застосунків у браузері, адже поєднує пряме передавання медіа, високу сумісність із сучасними браузерами та мінімальну потребу в додаткових компонентах. Використання SIP-протоколу поверх WebRTC дає змогу забезпечити стандартизовану сигналізацію, що особливо важливо для сумісності з існуючими VoIP-інфраструктурами. На фоні альтернатив — таких як SSE, що мають обмежену функціональність у сфері мультимедійної комунікації, — саме WebRTC SIP є найбільш збалансованим рішенням для побудови надійної вебплатформи для комунікації в реальному часі.

3 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ VOIP КОМУНІКАЦІЇ

У розділі розглянуто актуальні програмні рішення в сфері VoIP-комунікацій, що доступні на ринку. Проведено класифікацію рішень за принципом розгортання та типом ліцензування. Такий підхід дозволяє здійснити комплексну оцінку доступних інструментів з точки зору функціональності, рівня контролю, безпеки, вартості та придатності для впровадження в умовах обмежених ресурсів. Окрема увага приділяється порівнянню переваг і недоліків кожного класу рішень, що є підґрунтям для обґрунтування вибору архітектури власного VoIP-додатку, розробленого в межах дипломної роботи.

3.1 Огляд комерційних хмарних рішень

Сучасний ринок хмарних VoIP-послуг надзвичайно розгалужений: багато постачальників пропонують гнучкі тарифні плани та широкий набір функцій, орієнтуючись на зручність керування та швидке впровадження.

RingCentral — один із найвідоміших постачальників послуг у сфері уніфікованих комунікацій. Ця платформа пропонує комплексне рішення, яке включає голосові дзвінки, відеозв’язок і обмін повідомленнями, роблячи акцент на командну співпрацю та покращення взаємодії з клієнтами. До функціоналу входять інструменти на базі штучного інтелекту, зокрема транскрипція дзвінків та аналіз емоційного тону розмов (sentiment analysis). Крім того, RingCentral забезпечує широкі можливості інтеграції з іншими бізнес-додатками, що сприяє автоматизації робочих процесів і підвищенню продуктивності.

Цінова політика RingCentral зазвичай стартує від $20–30 на місяць за одного користувача, залежно від обраного пакету послуг. Як наслідок для дуже малих організацій або структур із жорстко обмеженим бюджетом регулярні витрати можуть бути суттєвим бар’єром. Крім того, використання стороннього сервісу

означає обмежений контроль над збереженням та обробкою комунікаційних даних, що може бути неприйнятним для організацій із підвищеними вимогами до конфіденційності [12].

Nextiva — постачальник VoIP-послуг, що орієнтується на простоту використання та надійність, передусім для малих і середніх підприємств, які шукають функціональне, але зручне у впровадженні рішення. Сервіс пропонує низку можливостей, зокрема вдосконалену систему інтерактивного голосового меню (IVR) для ефективного маршрутизації викликів, запис дзвінків для контролю якості та дотримання нормативних вимог, бізнес-обмін повідомленнями для розширення каналів комунікації, а також інтеграцію з популярними бізнес-додатками. Ціни на тарифи Nextiva зазвичай коливаються в межах $20–30 на одного користувача щомісяця. Завдяки орієнтації на простоту в користуванні та стабільну роботу, ця платформа є привабливим варіантом для малих організацій, які не мають окремого IT-відділу для адміністрування складних систем [13].

CloudTalk — VoIP-платформа, орієнтована насамперед на команди з продажу та обслуговування клієнтів. Серед ключових функцій — автодозвонювачі на основі ШІ, розширені аналітичні інструменти для оцінки ефективності дзвінків у режимі реального часу, а також глобальна доступність, що дозволяє ефективно вести комунікацію між офісами в різних країнах. CloudTalk також підтримує інтеграцію з CRM-системами для оптимізації робочих процесів, функції моніторингу дзвінків для тренування персоналу та контролю якості, а також маршрутизацію дзвінків на основі навичок операторів. Вартість використання CloudTalk становить від $25 до понад $50 на одного користувача щомісяця залежно від обраного тарифного плану. Попри широкий функціонал, підписна модель оплати може бути неприйнятною для організацій із обмеженим бюджетом, що стимулює інтерес до більш доступних і автономних рішень із можливістю самостійного розгортання [14].

Окрім вищезгаданих, ринок комерційних VoIP-рішень включає багато інших платформ, що орієнтуються на вузькі сегменти користувачів або пропонують унікальні можливості:

Grasshopper — рішення для індивідуальних підприємців та фрилансерів, яке дозволяє організувати віртуальну телефонну систему без необхідності придбання додаткового обладнання.

Vonage — платформа уніфікованих комунікацій, що добре підходить як для контакт-центрів, так і для невеликих бізнесів, яким потрібна інтеграція з іншими бізнес-інструментами.

Ooma — відома своєю швидкою інсталяцією та зручністю у використанні; часто включає функцію віртуального секретаря та необмежені дзвінки.

Dialpad — вирізняється розширеним використанням ШІ, зокрема функціями транскрипції дзвінків у режимі реального часу та створенням коротких підсумків розмов [15].

Попри широкий спектр функцій, більшість таких рішень побудовані за моделлю «програмне забезпечення як сервіс» (SaaS), що накладає додаткові обмеження на гнучкість і контроль.

Фундаментальне обмеження комерційних хмарних VoIP-рішень для організацій із підвищеними вимогами до безпеки полягає у відсутності прямого контролю над конфіденційністю даних та інфраструктурою комунікацій. У таких рішеннях вся інфраструктура — сервери, програмне забезпечення та передані дані — управляється та обслуговується стороннім постачальником послуг. Це породжує занепокоєння щодо конфіденційності комунікацій, ефективності реалізованих протоколів безпеки та обмежень у можливостях впливу на конфігурацію чи оновлення системи.

Хоча авторитетні провайдери зазвичай впроваджують відповідні заходи захисту, кінцевий користувач все ж залежить від політик безпеки сторонньої компанії, її технічної інфраструктури та відповідності правовим нормам. Для організацій, яким необхідна повна автономія над комунікаційними системами — зокрема, можливість самостійно налаштовувати безпекові параметри або контролювати, де й як обробляються дані — ці обмеження можуть стати критичними. Це, у свою чергу, стимулює інтерес до самостійно розгорнутих альтернатив.

3.2 Огляд відкритих систем IP-АТС

FreePBX — одна з найпоширеніших відкритих систем адміністрування IP-телефонії, яка надає веб-інтерфейс для управління сервером Asterisk. Система має модульну архітектуру, що дозволяє розширювати функціональність через додаткові модулі — як безкоштовні, так і комерційні. Важливою перевагою FreePBX є велика активна спільнота користувачів і розробників, що забезпечує наявність якісної документації, регулярних оновлень і підтримки.

FreePBX пропонує широкі можливості налаштування та адаптації, що робить її привабливою для організацій із достатніми технічними ресурсами. Водночас система може виявитися складною для первинного налаштування та адміністрування — знання у сфері VoIP, мережевих технологій і безпеки є обов’язковими для ефективного її використання. Таким чином, попри високу функціональність і підтримку спільноти, FreePBX може бути складною у впровадженні для організацій, які не мають власного IT-персоналу [16].

Asterisk є базовою технологією для багатьох VoIP-рішень, зокрема FreePBX, і водночас потужним open-source фреймворком, призначеним для створення широкого спектру комунікаційних застосунків — від IP-АТС до VoIP-шлюзів і серверів конференц-зв’язку. Asterisk підтримує велику кількість телекомунікаційних інтерфейсів та протоколів, включно з SIP, що робить його надзвичайно гнучкою платформою для розробки індивідуальних комунікаційних рішень.

Однак ефективне використання Asterisk вимагає глибоких технічних знань у таких сферах, як операційні системи Linux, мережеві конфігурації, скриптування та принципи телефонії. Через свою складність Asterisk часто розглядається не як готове до використання рішення для кінцевого користувача, а радше як інструментарій або платформа для розробки, орієнтована на досвідчених спеціалістів.

Попри майже необмежені можливості налаштування та контроль, які Asterisk надає кваліфікованим користувачам, крута крива навчання та високі

вимоги до технічної обізнаності роблять його менш придатним для невеликих організацій без власного IT-персоналу [17].

OpenSIPS — високооптимізований SIP-сервер, відомий своєю швидкодією та здатністю обробляти велику кількість викликів за секунду. Завдяки цим характеристикам він підходить для компаній, яким потрібна масштабована й надійна платформа, але без складних функцій, таких як відеоконференції.

Kamailio — ще один високопродуктивний SIP-сервер, здатний обслуговувати тисячі одночасних викликів. Kamailio часто використовується як основа для побудови об’єднаних комунікаційних платформ з підтримкою функцій статусу присутності, миттєвих повідомлень тощо.

SIP Foundry — хоча активність розробки цього проєкту останніми роками зменшилась, він свого часу позиціонувався як рішення класу «enterprise» з відкритим вихідним кодом для побудови IP-АТС [18].

Різноманіття таких проєктів свідчить про розвинену екосистему самостійно розгорнутих VoIP-рішень. Кожне з них орієнтоване на різні сценарії використання та рівні технічної підготовки, надаючи користувачам можливість обрати платформу відповідно до своїх потреб і ресурсів.

3.3 Аналіз комерційних самостійно розгорнутих VoIP-рішень

Окрім рішень з відкритим кодом, на ринку представлені також комерційні VoIP-системи, які можуть бути розгорнуті локально, поєднуючи переваги самостійного хостингу з зручністю використання та технічною підтримкою, притаманною комерційному програмному забезпеченню.

3CX є програмною IP-АТС (PBX), яка надає можливість самостійного розгортання як локально, на власному обладнанні організації, так і у хмарному середовищі. Система підтримує повноцінний набір функцій для сучасних комунікацій: голосовий зв’язок, відеоконференції, живий чат на вебсайті, а також інтеграцію з CRM-системами та іншими бізнес-додатками.

На відміну від багатьох хмарних провайдерів, які використовують модель оплати за кількістю користувачів, 3CX пропонує ліцензування на основі кількості одночасних дзвінків, що може бути більш економічно вигідним для організацій із великим обсягом дзвінків, але відносно невеликою кількістю користувачів. Інтерфейс управління 3CX реалізовано у вигляді веб-додатку, що спрощує конфігурацію та адміністрування системи навіть для користувачів без глибокої технічної підготовки.

3CX надає як безкоштовну версію з обмеженою кількістю одночасних викликів, так і комерційні редакції з розширеним функціоналом, що дозволяє адаптувати систему до потреб організацій різного масштабу. Завдяки поєднанню широких функціональних можливостей, помірної складності управління та моделі ліцензування, 3CX може бути привабливим рішенням для малих організацій, які прагнуть контролювати власну комунікаційну інфраструктуру, але водночас не мають ресурсу для повноцінного адміністрування складних open-source систем [19].

3.4 Порівняльний аналіз рішень для VoIP-комунікації

Малі організації опиняються перед вибором між хмарними комерційними VoIP-рішеннями та самостійно розгорнутими системами. Кожен з підходів має свої переваги й недоліки, що суттєво впливають на придатність рішення залежно від конкретних потреб та ресурсів організації. Наведено основні особливості кожного рішення у таблиці 3.1.

Малі організації часто надають перевагу простоті використання та доступності, що зумовлює початковий інтерес до хмарних VoIP-рішень із простим налаштуванням та передбачуваними щомісячними витратами. Такі сервіси зазвичай не потребують значних технічних знань для запуску, що є важливою перевагою для компаній без власного IT-відділу. Однак у довгостроковій перспективі накопичення абонентської плати, залежність від стороннього постачальника та обмеження в контролі над даними й інфраструктурою можуть

стати суттєвими недоліками, особливо в умовах зростання масштабів діяльності компанії.

Таблиця 3.1 — Порівняння хмарних і самостійно розгорнутих рішень

Параметр	Хмарні комерційні VoIP-рішення	Самостійно розгорнуті VoIP-рішення
Модульність	Функціональність визначається тарифним планом, обмежена можливість гнучкого налаштування.	Висока модульність, особливо у рішень з відкритим кодом (FreePBX, Asterisk).
Безпека	Забезпечується провайдером, зручно, але обмежений контроль.	Повна відповідальність за безпеку, але і максимальний контроль.
Контроль	Обмежений контроль над конфігурацією, оновленнями та даними.	Повний контроль над інфраструктурою, даними та оновленнями.
Вартість	Прогнозовані щомісячні витрати, які можуть накопичуватись.	Вищі початкові витрати, але нижча сукупна вартість у довгостроковій перспективі.
Простота адміністрування	Просте налаштування, відсутність потреби в глибоких знаннях.	Відносно просте (3CX) до складного (Asterisk), залежно від рішення.
Підтримка	Як правило, надається вендором у складі сервісу.	Спільнота (open-source) або офіційна підтримка (комерційні рішення)

Організації, що володіють певним рівнем внутрішньої технічної компетенції або акцентують увагу на приватності й автономії, можуть вважати самостійно розгорнуті VoIP-системи привабливішими. Відкриті рішення, зокрема FreePBX, вирізняються високою гнучкістю та мінімальними ліцензійними витратами, хоча їх використання передбачає відповідальність за налаштування, адміністрування та безпеку. Такі системи надають широкі можливості кастомізації, що дозволяє адаптувати функціональність відповідно до специфіки бізнес-процесів.

Комерційні самостійно розгорнуті рішення, такі як 3CX, пропонують своєрідний компроміс для малих організацій: вони забезпечують більший контроль порівняно з хмарними сервісами та водночас спрощують розгортання й обслуговування у порівнянні з повноцінними open-source платформами. Модель ліцензування на основі кількості одночасних викликів може бути економічно вигідною для організацій із відповідним профілем використання. Крім того, підтримка, яка часто надається постачальником, знижує навантаження на внутрішній IT-персонал.

Таким чином, вибір оптимального VoIP-рішення для малої організації має ґрунтуватися на ретельному аналізі її специфічних потреб, доступних технічних

ресурсів та бюджетних обмежень. Запропонована в межах цієї дипломної роботи розробка модульного, захищеного, самостійно розгорнутого VoIP-додатку на основі протоколу SIP орієнтована на конкретний сегмент організацій, що надають пріоритет приватності, контролю та доступності, прагнучи подолати наявні обмеження існуючих рішень.

Незважаючи на значну кількість наявних VoIP-продуктів, на ринку все ще залишаються незадоволені потреби — зокрема з боку невеликих організацій із суворими вимогами до захисту даних та автономності комунікаційної інфраструктури. Хмарні комерційні рішення, хоч і зручні у використанні, часто не дозволяють отримати повний контроль над системою. Водночас open-source платформи надають широкі можливості налаштування, але потребують значних зусиль для забезпечення їх стабільності й безпеки. Це створює потребу в новому рішенні, що поєднуватиме модульність, надійний захист, відносну простоту використання та економічну доцільність — і буде спеціально адаптоване до вимог малих організацій, орієнтованих на конфіденційність і контроль.

4 ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У розділі розглянуто програмні засоби, обрані для реалізації ключових компонентів системи: серверної логіки, клієнтського інтерфейсу, бази даних, механізмів автентифікації та телефонної інфраструктури. Наведено обґрунтування вибору кожного інструменту з урахуванням його переваг, сучасних тенденцій у сфері розробки вебзастосунків та специфіки предметної області — систем VoIP-комунікацій.

4.1 Засоби розробки серверної частини

В основі веб-застосунку VoIP лежить його серверна частина, яка відповідає за основну бізнес-логіку, управління даними та забезпечення безпеки. У межах даного проєкту серверну частину реалізовано з використанням мови програмування Java та фреймворку Spring Boot. Вибір цієї технологічної зв’язки обґрунтований її доведеною стабільністю, потужними корпоративними можливостями, активними зусиллями з модернізації екосистеми та здатністю ефективно підтримувати великомасштабні, модульні та захищені системи з високими вимогами до паралельності.

Тривалий час Java сприймалася як застаріла технологія, яку характеризує надмірна багатослівність та недостатня продуктивність — особливо у порівнянні з більш новими конкурентами, такими як Go чи Node.js. Проте таке сприйняття вже не є актуальним. Станом на 2025 рік Java зазнала суттєвих трансформацій, що дозволяють позиціонувати її як сучасну, гнучку та високопродуктивну мову для задоволення актуальних потреб корпоративного програмного забезпечення.

Ряд давно відомих недоліків Java — таких як повільний час запуску, велике споживання пам’яті та низька ефективність у паралельних обчисленнях — було системно вирішено завдяки впровадженню низки інноваційних проєктів:

GraalVM впроваджує попередню компіляцію (AOT — Ahead-of-Time Compilation), що дозволяє Java-застосункам запускатися швидше та споживати

менше пам’яті — критичні вимоги для хмарних середовищ та мікросервісної архітектури.

Проєкти Leyden та CraC (Coordinated Restore at Checkpoint) додатково скорочують час запуску Java-додатків, запроваджуючи механізми створення статичних образів та функціональність збереження/відновлення з контрольної точки.

Проєкт Loom вирішує обмеження Java у сфері паралелізму, запроваджуючи легковагові віртуальні потоки, що дозволяють Java обробляти мільйони одночасних задач з мінімальними накладними витратами [20].

Завдяки цим досягненням Java стала високо конкурентоспроможною у сферах, чутливих до продуктивності, таких як системи реального часу, мікросервіси та безсерверні обчислення, що робить її природним вибором для побудови VoIP-систем на основі SIP-протоколу, які висувають високі вимоги до паралельності, безпеки та оперативності.

Для побудови серверної частини системи обрано фреймворк Spring Boot, що відзначається модульністю, простотою використання та розвиненою екосистемою. Spring Boot є частиною більш широкого Spring Framework, який наразі став де-факто стандартом для розробки корпоративних застосунків на платформі Java. Основною причиною вибору Spring фреймворку стала його широка компонентна база. 4.1.

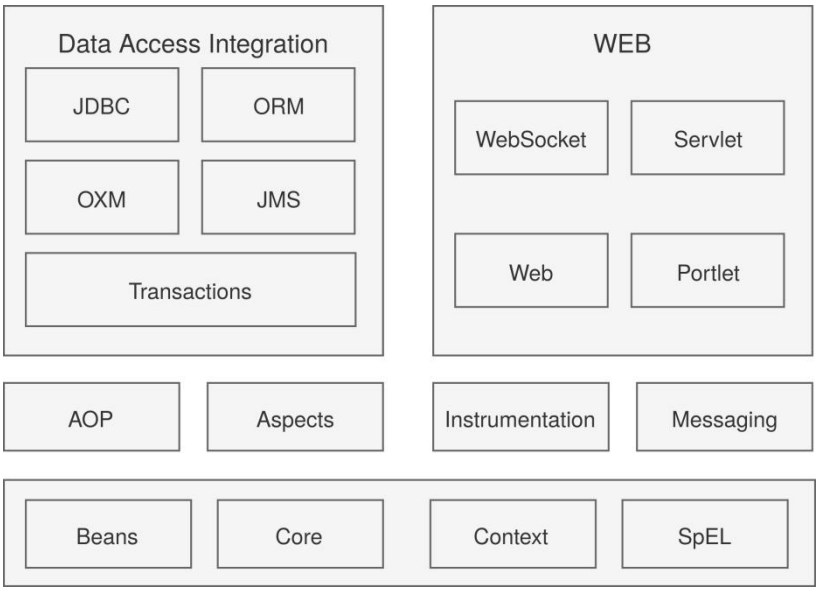

Рисунок 4.1 — Компоненти Spring фреймворку

В основі архітектури Spring лежить принцип впровадження залежностей (Dependency Injection) — патерн проєктування, що дозволяє послабити зв’язки між компонентами. У традиційних системах об'єкти часто самостійно створюють та управляють своїми залежностями, що призводить до жорсткого зв’язування та ускладнює супровід. Spring централізовано управляє залежностями, впроваджуючи їх у потрібні місця. Такий підхід підвищує тестованість, модульність та підтримуваність, що є критичними для масштабованої комунікаційної системи.

Крім того, Spring підтримує використання аспектно-орієнтованого програмування (AOP — Aspect-Oriented Programming), що дозволяє відокремити перехресні функції, такі як логування, аутентифікація чи управління транзакціями. У контексті VoIP-застосунку це означає, що аудит SIP-повідомлень, застосування політик дзвінків або запис журналів безпеки можуть виконуватися прозоро, без засмічення основної бізнес-логіки.

Для побудови серверної частини було використано компонент Spring MVC — веб-фреймворк, що реалізує архітектурну модель Model-View-Controller (MVC). Цей патерн дозволяє розділити внутрішнє представлення даних (Model) від користувацького інтерфейсу (View), а Controller управляє взаємодією між ними. Такий підхід забезпечує чисту архітектуру, покращує підтримуваність та спрощує розширення функціоналу системи [21].

У застосунку використовується Spring Security для забезпечення безпечного входу в систему та контролю доступу відповідно до ролей користувачів. Це гарантує, що такі ключові компоненти системи, як реєстрація SIP-ендпойнтів, отримання голосової пошти та управління користувачами, перебувають під суворим контролем доступу та автентифікацією, що повністю відповідає нефункціональним вимогам до надійної безпеки.

Для взаємодії з базою даних MongoDB використано Spring Data MongoDB, що є абстракційним шаром, який спрощує роботу з базою даних. Це дозволяє легко зберігати напівструктуровані дані, такі як контактна інформація, метадані викликів та посилання на голосові повідомлення. Розробники отримують

переваги у вигляді автоматичної генерації запитів, використання репозиторних інтерфейсів та послідовного доступу до даних.

4.2 Засоби розробки клієнтської частини

Еволюція веб-технологій суттєво трансформувала підходи до створення інтерактивних і застосунків реального часу. Традиційно вебсторінки мали статичний характер і вимагали повного перезавантаження при взаємодії користувача, що обмежувало можливості побудови складних та динамічних систем, подібних до комунікаційних платформ. З появою Single Page Applications (SPA) технології на кшталт React та TypeScript стали провідними інструментами для розробки надійних, масштабованих і високопродуктивних інтерфейсів, здатних ефективно обробляти складні користувацькі інтерфейси та події в реальному часі.

React, розроблений компанією Facebook, був представлений у 2013 році як відкрита бібліотека JavaScript для вирішення завдань побудови динамічних та високопродуктивних користувацьких інтерфейсів. Спочатку відомий під назвою FaxJS, React впровадив концепцію Virtual DOM, що дозволило суттєво підвищити ефективність рендерингу UI. Сьогодні React є однією з найпопулярніших технологій для клієнт-розробки, що використовується такими компаніями, як Facebook, Instagram, Airbnb, Netflix та Twitter [22].

Разом із React широко використовується TypeScript, розроблений компанією Microsoft. TypeScript є надмножиною JavaScript з підтримкою статичної типізації, що дозволяє вирішити низку обмежень JavaScript при створенні масштабних застосунків. Статична типізація, розширені можливості інструментів та підвищена продуктивність розробників роблять TypeScript популярним вибором для підвищення надійності React-застосунків.

React є декларативною, компонентною бібліотекою JavaScript, розробленою для створення ефективних, гнучких та масштабованих користувацьких інтерфейсів. Вона зосереджена на шарі View (V) в архітектурній моделі Model-

View-Controller (MVC), дозволяючи розробникам створювати багаторазові UI-компоненти, які можуть самостійно управляти своїм станом та складатися у більш складні інтерфейси.

React вважається доступним для розробників, знайомих з JavaScript та HTML. Його кривизна навчання є порівняно пологою, дозволяючи сконцентруватися виключно на розробці UI без необхідності опанування складних архітектурних патернів на початкових етапах. Використання JSX (JavaScript XML) дозволяє застосовувати HTML-подібний синтаксис безпосередньо в JavaScript-коді, що підвищує читабельність і зручність використання, а також зменшує ризики поширених вразливостей, таких як Cross-Site Scripting (XSS).

Архітектура React побудована навколо концепції компонентів — незалежних, багаторазових та інкапсульованих одиниць UI-логіки. Кожен компонент отримує дані через props та підтримує власний внутрішній стан, що забезпечує динамічне відображення залежно від взаємодії користувача (рис. 4.2). React підтримує як класові, так і функціональні компоненти, при цьому сучасною практикою є використання функціональних компонентів у поєднанні з React Hooks для управління побічними ефектами, станом та контекстом.

Рисунок 4.2 — Потік виконання в React

Така модульна структура спрощує розробку, тестування та супровід, сприяючи повторному використанню коду та паралельній роботі кількох команд розробників.

Однією з найвизначніших інновацій React є концепція Virtual DOM — легковагового внутрішнього представлення реального DOM у пам’яті. Під час зміни стану компонента React ефективно генерує новий Virtual DOM, порівнює його з попередньою версією за допомогою алгоритму диференціювання та оновлює лише необхідні частини фактичного DOM. Це суттєво знижує витрати на операції з DOM, покращуючи продуктивність та чутливість користувацького інтерфейсу [22].

Мова TypeScript — це типізована надмножина JavaScript, створена для усунення обмежень останнього при розробці великих та складних застосунків. TypeScript впроваджує опціональну статичну типізацію, розширену інференцію типів, дженерики та сучасні функції ECMAScript, пропонуючи розробникам безпечніше та надійніше середовище розробки.

Забезпечуючи статичну перевірку типів на етапі компіляції, TypeScript зменшує кількість типових помилок під час виконання, що особливо важливо в застосунках, де UI-компоненти взаємодіють з численними API, сервісами WebRTC та SIP-сигналізацією, де невідповідність типів може призвести до прихованих помилок.

Статична типізація також підвищує можливості інструментів, зокрема автодоповнення коду, підтримку рефакторингу та інтелектуальну навігацію кодом, що сприяє підвищенню продуктивності розробників та полегшує підтримку коду.

Мова TypeScript є повністю сумісною з існуючими JavaScript-кодами, що дозволяє поступово впроваджувати його у проєкти. Він компілюється у стандартний JavaScript, гарантуючи сумісність з усіма браузерами та середовищами виконання JavaScript [23].

У контексті складних UI-застосунків TypeScript забезпечує типобезпеку компонентів, сервісів та логіки управління станом, що знижує когнітивне навантаження на розробників та забезпечує безпечніше рефакторування у міру зростання застосунку.

У поєднанні React та TypeScript формують потужний стек технологій для розробки сучасних вебзастосунків, що вимагають високої інтерактивності, чутливості в реальному часі та масштабованості. Компонентно-орієнтований та декларативний підхід React у поєднанні з його Virtual DOM та уніфікованим потоком даних дозволяє створювати продуктивні та зручні для супроводу інтерфейси, а TypeScript підсилює ці можливості шляхом запровадження типобезпеки, підвищення якості коду та зменшення кількості помилок під час виконання.

4.3 База даних

У сучасному програмному забезпеченні зростає потреба у гнучких, масштабованих та високопродуктивних рішеннях для зберігання даних. Традиційні системи управління реляційними базами даних (RDBMS) протягом десятиліть залишалися стандартом, однак з появою неструктурованих та напівструктурованих даних популярності набули NoSQL-бази даних. MongoDB, як провідна відкрита NoSQL-база, пропонує документно-орієнтовану модель з підтримкою динамічних схем та горизонтального масштабування. Її можливості дозволяють вирішувати виклики, пов’язані з сучасними застосунками, включаючи Big Data, мобільні платформи та розподілені системи.

MongoDB є широко використовуваною, відкритою, документно-орієнтованою NoSQL-базою даних, створеною для забезпечення високої продуктивності, доступності та масштабованості. Реалізована на мові C++, MongoDB спроєктована для обробки різноманітних типів даних, включно зі складними та ієрархічними структурами [24].

На відміну від реляційних баз даних, що базуються на таблицях та рядках, MongoDB зберігає дані у вигляді колекцій документів. Ці документи представлені у форматі BSON (Binary JSON) та підтримують вкладені структури даних і масиви, що дозволяє розробникам більш природньо та гнучко моделювати об'єкти реального світу. Відсутність попередньо визначеної схеми дає змогу MongoDB

швидко адаптуватися до змін моделей даних, що робить її особливо придатною для агільних та ітеративних підходів до розробки.

MongoDB оптимізована для операцій з високою пропускною здатністю та низькою затримкою доступу до даних. Підтримка вбудованих документів і масивів зменшує потребу у ресурсомістких операціях з'єднання (join), характерних для традиційних реляційних баз даних. Крім того, використання індексів, зокрема для вкладених полів, підвищує продуктивність запитів та забезпечує ефективне отримання даних.

База даних надає потужну та виразну мову запитів, що дозволяє виконувати складні запити, агрегації та текстовий пошук. MongoDB підтримує стандартні операції CRUD (створення, читання, оновлення, видалення), а також розширені функції, такі як геопросторові запити, фасетний пошук та конвеєри агрегації, що дозволяє розробникам виконувати складні маніпуляції з даними безпосередньо на рівні бази даних.

Архітектура MongoDB ґрунтується на трьох основних поняттях: бази даних, колекції та документів (рис. 4.3). Гнучка архітектура підтримує динамічні схеми, дозволяючи розробникам зберігати різні структури даних в межах однієї колекції без потреби у визначених наперед схемах або дороговартісних міграціях [5].

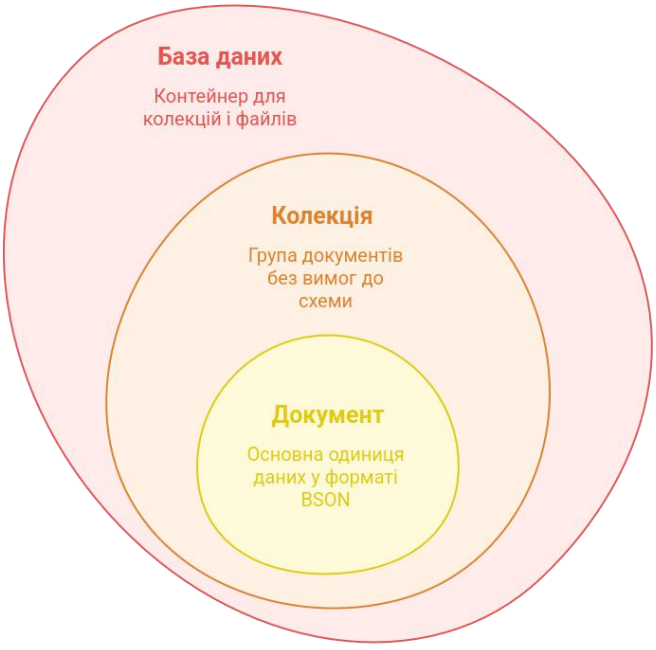

Рисунок 4.3 — Архітектура MongoDB

MongoDB є яскравим прикладом документно-орієнтованої NoSQL-бази даних, що забезпечує гнучкість, масштабованість та продуктивність, необхідні сучасним застосункам, орієнтованим на роботу з великими обсягами даних. Її динамічна схема, розвинена мова запитів та розподілена архітектура дають змогу організаціям ефективно управляти складними та постійно змінюваними наборами даних.

4.4 Сервер авторизації

Сучасні вебзастосунки часто потребують безпечних та масштабованих механізмів для управління ідентичностями користувачів, контролю доступу до ресурсів та забезпечення безперервного користувацького досвіду. Це особливо важливо у сферах, таких як вебзастосунки для Voice over IP (VoIP), де критичними є безпека даних, обробка реального часу та доступність сервісів. Системи управління ідентичностями та доступом (Identity and Access Management, IAM), зокрема сервери авторизації, вирішують ці завдання, надаючи централізовані сервіси автентифікації, авторизації та управління користувачами.

У традиційних вебзастосунках функції IAM часто реалізовувалися всередині самих застосунків. Проте такий підхід спричиняє складність, проблеми з масштабуванням та ризики безпеки, особливо у багатокористувацьких та розподілених системах.

Для вирішення цих викликів сучасні застосунки часто делегують автентифікацію та авторизацію спеціалізованим серверам авторизації, які централізовано керують ідентичностями та забезпечують контроль доступу до різних застосунків та сервісів.

Keycloak — це відкрите рішення для IAM, розроблене для захисту застосунків та сервісів шляхом використання стандартних протоколів та орієнтованих на користувача функцій. Вперше випущений у 2014 році під ліцензією Apache 2, Keycloak розроблений на мові Java та працює на сервері застосунків WildFly.

Основна мета Keycloak — спростити інтеграцію функцій безпеки, таких як єдиний вхід (SSO), федерація користувачів, соціальний вхід та багатофакторна автентифікація, у сучасні застосунки з мінімальними зусиллями з боку розробників. Завдяки автономному серверу авторизації Keycloak дає змогу застосункам делегувати процеси автентифікації та авторизації, дозволяючи розробникам зосередитись на бізнес-логіці, а не на реалізації безпеки "з нуля" [25].

У контексті вебзастосунків VoIP безпечна та зручна автентифікація користувачів є ключовою для захисту конфіденційних даних комунікації та контролю доступу до функцій та сервісів VoIP. Keycloak пропонує низку переваг, які є актуальними для таких систем:

- централізована автентифікація та авторизація;
- підтримка SSO для декількох компонентів;
- інтеграція з існуючими провайдерами ідентичностей;
- підтримка безпечних API та мобільних клієнтів;
- адаптація до багатокористувацьких середовищ.

Отже, Keycloak пропонує надійне та гнучке рішення для управління користувацькими ідентичностями, контролю доступу до сервісів та гарантує безпечний та зручний користувацький досвід на всіх платформах.

4.5 Сервер PBX

У телекомунікаційних системах приватна телефонна станція (Private Branch Exchange, PBX) відіграє центральну роль, здійснюючи маршрутизацію голосових викликів в межах організації та забезпечуючи доступ до зовнішніх телефонних мереж. З еволюцією технологій Voice over IP (VoIP) традиційні системи PBX були замінені або доповнені програмними рішеннями, які використовують IP-мережі для надання сучасних комунікаційних послуг.

PBX — це телефонна комутаційна система, яка полегшує внутрішню комунікацію в межах організації та з’єднує користувачів з зовнішніми телефонними службами [26].

Програмні системи PBX, часто звані IP PBX, дозволяють організаціям консолідувати комунікаційну інфраструктуру, знижувати операційні витрати, а також покращувати масштабованість та гнучкість. Вони також дозволяють інтеграцію з вебзастосунками, забезпечуючи сервіси, такі як click-to-call, відеоконференції та уніфіковані комунікації.

Asterisk — це відкрита програмна платформа PBX, призначена для створення додатків реального часу комунікацій. Розповсюджується під ліцензією GNU General Public License (GPL) та надає потужну і гнучку основу для розробки телефонних систем, що підтримують голос, відео, обмін повідомленнями та інші види комунікацій.

Створений у 1999 році, Asterisk з того часу став однією з найпопулярніших і найуніверсальніших платформ для VoIP-систем у світі. Його гнучкість дозволяє використовувати Asterisk як для простих офісних телефонних систем, так і як основу для складних, високонастроюваних комунікаційних застосунків [26].

Однією з визначальних рис Asterisk є його модульна та розширювана архітектура, яка дозволяє адміністраторам систем і розробникам адаптувати впровадження Asterisk відповідно до конкретних вимог, шляхом активації або деактивації обраних компонентів.

У сучасних вебзастосунках VoIP Asterisk виконує функцію центрального телефонного серверу, що забезпечує основні комунікаційні сервіси та дає змогу інтегруватися з вебклієнтами, мобільними застосунками та зовнішніми системами.

Його актуальність у таких контекстах включає:

- підтримку WebRTC та PJSIP;
- складну маршрутизацію викликів;
- інтеграцію з REST API та зовнішніми базами даних;
- міст між вебклієнтами та традиційними телефонними мережами;

Отже, Asterisk є потужною та гнучкою відкритою платформою PBX, здатною надавати широкий спектр сервісів комунікацій у реальному часі. Його модульна архітектура, розширюваність та підтримка відкритих стандартів роблять його відповідним вибором для сучасних VoIP-систем, зокрема тих, що інтегруються з вебклієнтами та сервісами.

5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У розділі представлено особливості реалізації розробленої системи з акцентом на ключові технічні та архітектурні аспекти. Окреслено процес побудови системи, її структуру, складові компоненти та використані технології.

Основною метою етапу реалізації було трансформування проєктної моделі системи у працездатний прототип, що відповідає як функціональним, так і нефункціональним вимогам. До них, зокрема, належать забезпечення безпечної SIP-комунікації, модульності та зручності використання, що робить систему придатною для малих організацій, які потребують самостійно розгорнутої, приватної інфраструктури зв’язку.

На загальному рівні система складається з веб-клієнта, розробленого з використанням React, серверної частини у вигляді REST API на базі Spring Boot, сервера, Asterisk PBX для обробки викликів та сервера Keycloak, що виконує функції аутентифікації. Система підтримує типові користувацькі сценарії, такі як керування SIP-акаунтами, здійснення та прийом викликів, управління контактами та перегляд історії дзвінків. У наступних підрозділах детально розглянуто архітектуру та особливості реалізації кожного з компонентів системи.

5.1 Опис функціональності

У підрозділі описуються основні функції та можливості веб-стільникового додатка, а також те, як користувачі взаємодіють із системою для виконання ключових завдань, таких як здійснення дзвінків, управління контактами та перегляд історії викликів.

Система підтримує як звичайних користувачів, так і адміністраторів, з чітким розподілом ролей та відповідальностей. Діаграма прецедентів (рис. 5.1) та супутні описи надають чітке уявлення про різні дії, доступні користувачам у додатку, а також необхідні взаємодії з зовнішніми компонентами, такими як сервер Asterisk та сервіс автентифікації Keycloak.

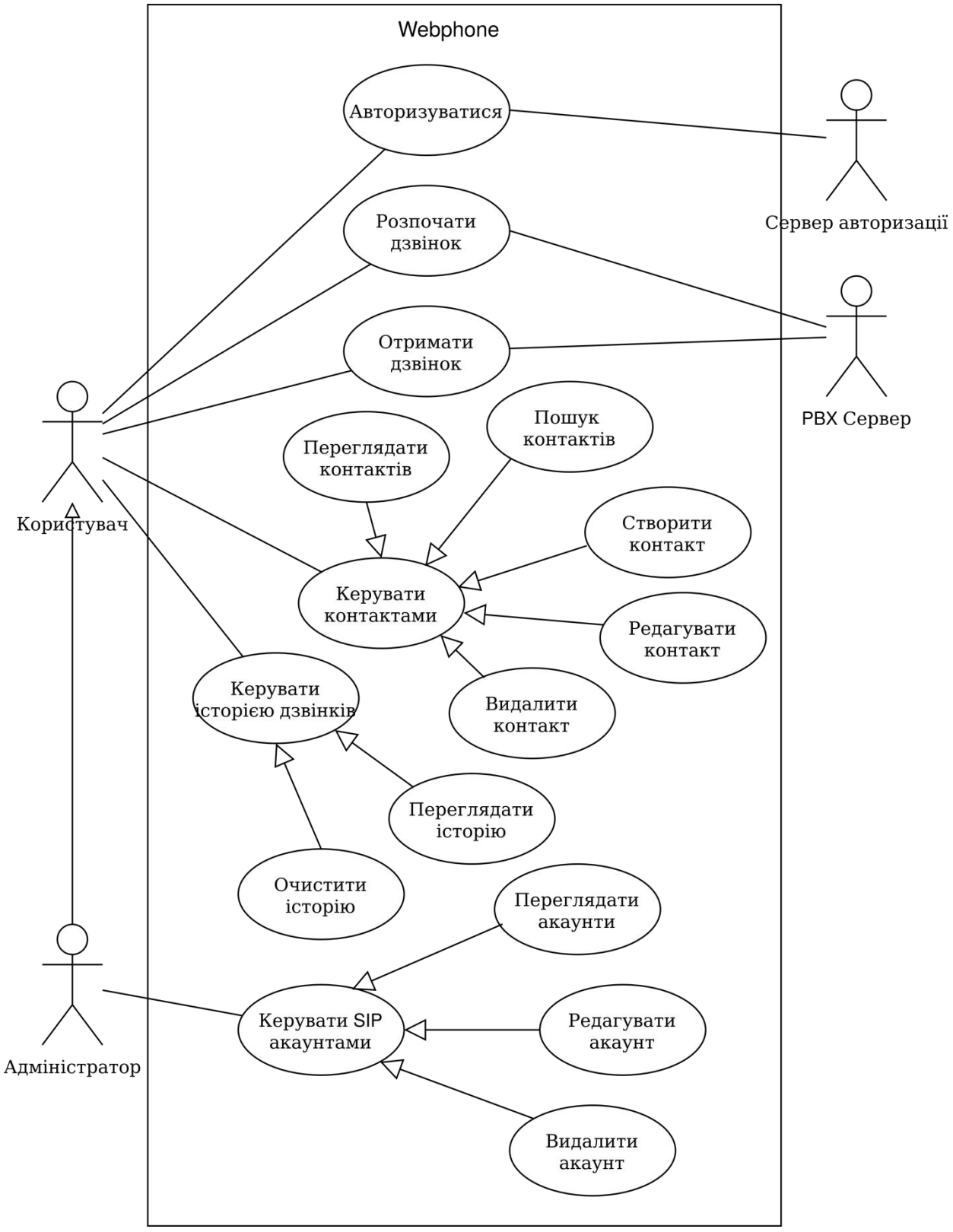

Рисунок 5.1 — Діаграма прецедентів

Система підтримує чотири основні ролі: Користувач, Адміністратор, Сервер Asterisk та служба автентифікації Keycloak. Кожна роль взаємодіє з системою в межах визначених меж і функцій, як це узагальнено в таблиці 5.1.

Таблиця 5.1 – Ролі та їх взаємодія з системою

Роль	Короткий опис
Користувач	Кінцевий користувач, який застосовує систему для SIP-комунікації, керування контактами, перегляду історії дзвінків та персоналізації налаштувань.
Адміністратор	Привілейований користувач із доступом до функцій адміністрування системи, зокрема керування обліковими даними SIP інших користувачів.
Сервер Asterisk	SIP-сервер, що обробляє сигналізацію та медіапотоки дзвінків, здійснених та отриманих користувачем через систему.
Keycloak	Зовнішній провайдер автентифікації, відповідальний за вхід користувача в систему, реєстрацію та управління сесіями.

Ґрунтуючись на аналізі вимог до системи, було ідентифіковано прецеденти наведені у таблиці 5.2 для опису очікуваних функціональних можливостей користувача.

Таблиця 5.2 – Прецеденти та їх короткі описи

Випадок використання	Короткий опис
Здійснити виклик	Користувач ініціює SIP-виклик шляхом введення номера телефону за допомогою номеронабирача та взаємодії з SIP-сервером.
Отримати виклик	Користувач отримує вхідний SIP-виклик від SIP-сервера та має можливість прийняти, відхилити або вимкнути звук виклику.
Перегляд адресної книги	Користувач переглядає адресну книгу, що містить перелік контактів, та може здійснювати навігацію до детальної інформації про контакт.
Створення контакту	Користувач створює новий контакт шляхом введення такої інформації, як ім'я, номери телефонів, біографічні дані, та завантаження фотографії.
Читання контакту	Користувач переглядає детальну інформацію про конкретний контакт, включаючи пов'язану історію дзвінків.
Оновлення контакту	Користувач редагує інформацію про контакт, забезпечуючи унікальність та цілісність даних.
Видалення контакту	Користувач видаляє контакт після підтвердження.
Перегляд історії дзвінків	Користувач переглядає записи історії дзвінків, згруповані та відсортовані за датою.
Очищення історії дзвінків	Користувач видаляє всі або вибрані записи історії дзвінків.
Керування контактами	Користувач виконує операції з керування контактами (створення, читання, оновлення, видалення) в межах адресної книги.
Керування історією дзвінків	Користувач виконує операції з керування історією дзвінків (перегляд, очищення, повторний набір номерів з історії).
Керування обліковими даними SIP	Адміністратор здійснює керування обліковими даними SIP користувачів, включаючи увімкнення, вимкнення або редагування SIP-інформації.

Таким чином, функціональні можливості системи охоплюють повний спектр дій, необхідних для забезпечення ефективної SIP-комунікації, керування

контактами та обліковими даними, а також взаємодії з серверною інфраструктурою. Чітке визначення ролей, прецедентів використання та взаємодій дозволяє сформувати узгоджену архітектуру застосунку, що задовольняє вимоги користувачів і забезпечує масштабованість та безпеку системи.

5.2 Архітектура застосунку

Розроблена система реалізована за принципами архітектури модульного моноліту: усі клієнтські та серверні компоненти об’єднано в єдиний блок розгортання з внутрішньо розділеною модульною структурою. Такий підхід було свідомо обрано для досягнення балансу між простотою розгортання та супроводу з одного боку, та гнучкістю для подальшої можливої модуляризації або розділення — з іншого. Архітектура модульного моноліту забезпечує тісну інтеграцію клієтської та серверної частини, дозволяючи ефективно використовувати спільні моделі даних, внутрішні API та єдиний механізм забезпечення безпеки, водночас зберігаючи чіткий внутрішній поділ відповідальностей.

Водночас зазначене архітектурне рішення свідомо обмежується межами ядра застосунку. Допоміжні сервіси, такі як сервер авторизації Keycloak, сервер телефонії Asterisk PBX та база даних MongoDB, розглядаються як зовнішні, слабо пов'язані компоненти. Зазначені сервіси є зрілими та спеціалізованими платформами, що надають перевірену часом функціональність — зокрема, автентифікацію та управління ідентичністю, обробку SIP-сигналізації та дзвінків, а також збереження даних, — повторна реалізація яких у складі ядра була б нераціональною та неефективною.

У результаті, загальна архітектура системи не є прикладом суворої мікросервісної архітектури, де кожна функціональна область реалізується у вигляді повністю автономного сервісу. Натомість, система являє собою гібридну модель, що поєднує модульне монолітне ядро з інтеграцією зовнішніх сервісів за принципами сервіс-орієнтованої архітектури. Такий підхід забезпечує операційну

ефективність, зниження складності інфраструктури та спрощене розгортання, одночасно зберігаючи можливість інтеграції з найкращими у своєму класі технологіями та забезпечуючи гнучкість для майбутнього розвитку системи.

Розроблена система включає низку основних компонентів, взаємодія між якими формує базову модель поведінки застосунку у середовищі виконання (рис. 5.2).

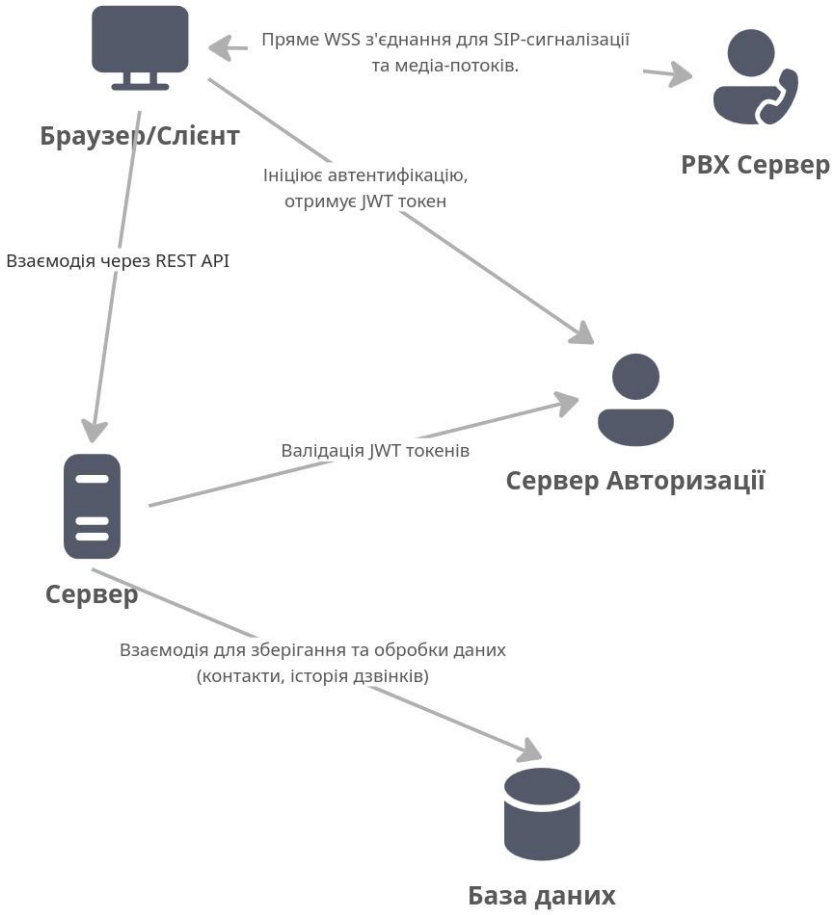

Рисунок 5.2 — Загальна архітектура системи

Фронтенд-компонент реалізовано у вигляді веб-орієнтованого клієнта softphone, що забезпечує користувацький інтерфейс та обробку SIP-сигналізації за допомогою WebRTC через бібліотеку JsSIP. Окрім того, клієнтська частина здійснює взаємодію з сервером через REST API для управління користувацькими даними, зокрема адресною книгою та історією дзвінків.

Бекенд виконує подвійну функцію — виступає як сервер API та сервер статичних файлів, необхідних для роботи клієнтської частини. Він надає захищені REST API для операцій з управління даними, здійснює валідацію токенів JWT, виданих сервером авторизації Keycloak, для забезпечення контролю доступу, та гарантує строгий поділ користувацьких даних. Варто підкреслити, що на поточному етапі серверна частина не взаємодіє безпосередньо з сервером Asterisk PBX; усі процеси SIP-сигналізації виконуються виключно на рівні клієнта.

Сервер Keycloak використовується для виконання процесів автентифікації та авторизації, генеруючи токени доступу (JWT) відповідно до стандартних протоколів OAuth2/OpenID Connect. Отримані токени використовуються клієнтом та валідуються сервером для забезпечення безпечного контролю доступу до ресурсів системи.

Сервер Asterisk, сконфігурований з підтримкою WebRTC через стек PJSIP, виконує обробку SIP-сигналізації та маршрутизацію медіа-потоків для VoIP-дзвінків. Клієнтська частина безпосередньо встановлює захищені WebSocket (WSS) з’єднання з сервером Asterisk, що забезпечує можливість здійснення дзвінків у режимі реального часу без участі серверу у сигнальному обміні.

MongoDB використовується як основне сховище даних системи, де зберігаються контакти, записи історії дзвінків та завантажені користувачами зображення. Доступ до бази даних обмежено виключно для серверу, який здійснює управління доступом на основі контексту автентифікованого користувача.

Отже, архітектура системи обирає прагматичний підхід модульного моноліту для ядра застосунку, поєднуючи клієнтську та серверні частини у єдиний артефакт для спрощення розгортання та обслуговування. При цьому критичні допоміжні сервіси, такі як Keycloak, Asterisk та MongoDB, свідомо винесені як незалежні компоненти. Це архітектурне рішення дозволяє скористатися перевіреними спеціалізованими технологіями для автентифікації, SIP-комунікацій та збереження даних, що забезпечує гнучкість, підтримуваність та сумісність системи.

5.3 Клієнтська частина

Фронтенд-застосунок виконує роль основного користувацького інтерфейсу системи, виступаючи у вигляді веб-клієнта. Він повністю реалізований з використанням технологій React та TypeScript, застосовуючи сучасні підходи до розробки клієнтської частини та багаторівневу внутрішню архітектуру.

Фронтенд несе відповідальність за обробку всіх взаємодій користувача на стороні клієнта, зокрема за автентифікацію, управління викликами, роботу з контактами та медіастрімінгом. При цьому він забезпечує захищену комунікацію з сервер-сервісами та безпосередньо взаємодіє з сервером Asterisk для здійснення SIP-сигналізації та переговорів медіа сесій через WebRTC.

На концептуальному рівні клієнт-компонент взаємодіє з трьома ключовими зовнішніми системами: сервером автентифікації Keycloak для управління користувачами, сервером для доступу до ресурсів, а також Asterisk PBX для забезпечення комунікацій у режимі реального часу. Ці взаємодії реалізовані через спеціалізовані контексти та хуки, що інкапсулюють складність інтеграцій та забезпечують чистий інтерфейс для UI-компонентів.

Фронтенд використовує набір спеціалізованих бібліотек та інструментів, що доповнюють основні технології React та TypeScript:

– React Router використовується для управління маршрутизацією на стороні клієнта;

– React Query забезпечує асинхронне отримання даних, кешування та механізми повторних запитів, значно зменшуючи обсяг шаблонного коду при роботі з REST API;

– Keycloak-JS забезпечує безшовну інтеграцію з сервером авторизації Keycloak;

– JsSIP відповідає за SIP-сигналізацію через WebSocket (WSS) та керування сесіями WebRTC.

Зазначені бібліотеки інтегровані в рамках багаторівневої та модульної структури, яка слідує принципам розділення відповідальностей та слабкого зв’язування компонентів (рис. 5.3).

Рисунок 5.3 — Структура клієнтської частини

На практиці така структура реалізується через низку спеціалізованих контекстів і хуків, які відповідають за різні аспекти поведінки застосунку. Зокрема, AuthContext, SipContext та CallContext є ключовими вузлами, які формують основу керування станом і забезпечують інтеграцію з зовнішніми сервісами.

Компонент AuthContext керує станом автентифікації користувача та надає методи для входу та виходу з системи із використанням Keycloak-JS. Цей контекст гарантує наявність JWT-токенів для API-запитів та обробляє життєвий цикл сесії користувача.

Компонент SipContext забезпечує повний життєвий цикл SIP-сесії. Він ініціалізує користувацький агент JsSIP (UA) з використанням налаштувань облікового запису SIP, здійснює реєстрацію, моніторинг стану з'єднання та надає методи для управління дзвінками (ініціація, прийняття, завершення, DTMF, утримання, вимкнення звуку). Взаємодія з JsSIP інкапсульована у хуці useSip, а

SipProvider гарантує належну ініціалізацію агенту користувача та його доступність у межах застосунку.

Компонент CallContext є вищим рівнем абстракції над SipContext, що зосереджується на управлінні окремою сесією виклику на відповідних сторінках. Це спрощує логіку керування активним дзвінком, тоді як SipContext продовжує відповідати за багатосесійну логіку на рівні сигналізації.

Цей багаторівневий підхід з використанням контекстів та хуків забезпечує чітке розділення відповідальностей між SIP-сигналізацією та автентифікацією, надаючи UI-компонентам спрощені API для взаємодії.

Життєвий цикл SIP-сесії інтегровано до клієнту через комбінацію сервісного модуля sip.ts, хука useSip та контексту SipContext. Сервісний шар ініціалізує агент JsSIP, використовуючи вказані облікові дані SIP, встановлюючи WSS-з’єднання з сервером Asterisk. Хук useSip інкапсулює складне управління SIP-сесією, включаючи:

- реєстрацію SIP та моніторинг з’єднання;
- обробку вхідних викликів;
- ініціацію, прийом та завершення викликів;
- підтримку функцій сесії, таких як утримання виклику, вимкнення мікрофона та передача DTMF-сигналів.

Використання React Context гарантує, що всі дочірні компоненти мають доступ до функціоналу SIP без необхідності безпосередньої роботи з екземпляром JsSIP, що сприяє підвищенню підтримуваності та тестованості застосунку.

Управління життєвим циклом SIP-з’єднання включає базові механізми відновлення з’єднання та обробки помилок через SipContext та useSip. Хоча клієнт підтримує кілька одночасних викликів на рівні сигналізації, інтерфейс користувача наразі обмежений підтримкою одного активного дзвінка, а розширення функціональності для підтримки мультидзвінків заплановано на подальші ітерації.

Архітектура застосунку орієнтована на модульність, використовуючи React Context та власні хуки для управління станом та взаємодії з зовнішніми сервісами, такими як Keycloak та Asterisk. Незважаючи на дотримання принципів чистої

архітектури та багаторівневої організації, рішення залишається гнучким для подальшого масштабування та можливої декомпозиції сервісів.

5.4 Серверна частина

Бекенд системи побудований за допомогою Java Spring Boot, використовуючи потужність Spring Data MongoDB для збереження даних та MongoDB як базу даних для обробки масштабованого та гнучкого зберігання даних. Цей дизайн слідує принципу чистої, модульної архітектури, заснованої на патерні трьохшарової архітектури, яка розділяє систему на три чітко визначені шари: Контролер, Сервіс і Репозиторій (рис. 5.4). Такий підхід покращує розподіл обов'язків, що робить код більш підтримуваним і масштабованим.

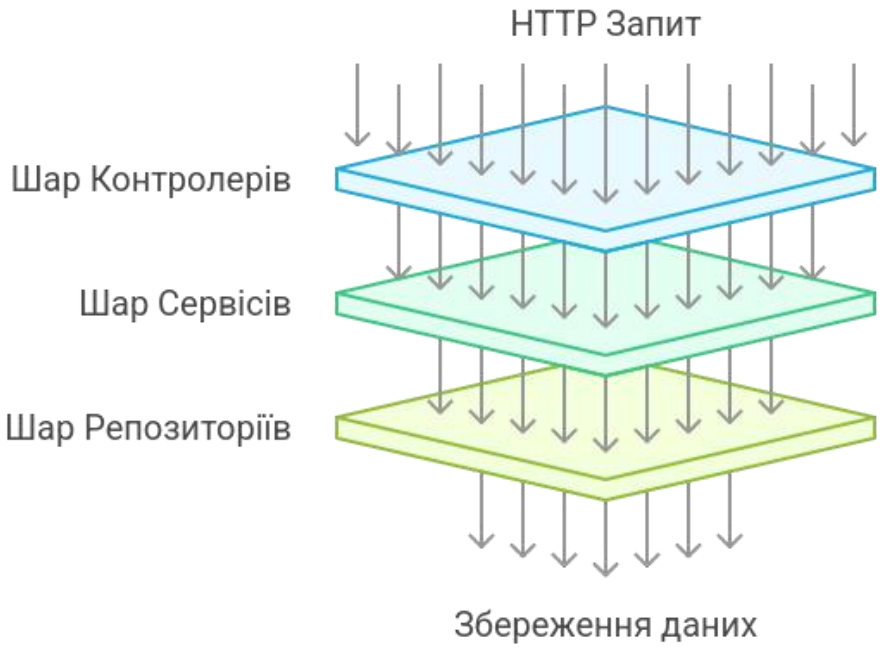

Рисунок 5.4 — Трьохшарова архітектура

Шар контролера служить точкою входу для вхідних HTTP-запитів. Він визначає REST API, з якими взаємодіють клієнти (фронтенд або інші системи). Контролери відповідають за обробку HTTP-запитів та їх мапування на відповідні методи сервісів. Наприклад, ContactController обробляє всі кінцеві точки, пов'язані

з контактами користувачів, а HistoryController відповідає за дії, що стосуються історії дзвінків. Обробка помилок також здійснюється централізовано через GlobalExceptionHandler.

Шар сервісів містить бізнес-логіку додатку. Він виступає посередником між шарами контролера та репозиторію, забезпечуючи правильне впровадження правил та робочих процесів додатку. Наприклад, ContactService керує операціями, такими як створення, оновлення та видалення контактів, а HistoryService займається операціями, пов'язаними з історією дзвінків. Шар сервісів також відповідає за перетворення сирих даних між доменними моделями та об'єктами передачі даних (DTO), зазвичай за допомогою маперів.

Шар репозиторію відповідає за збереження даних. Він визначає інтерфейс, через який додаток взаємодіє з базою даних, використовуючи Spring Data MongoDB для безшовних операцій CRUD. Репозиторії, такі як ContactRepository і HistoryRepository, дозволяють сервісам виконувати необхідні операції з отримання або збереження даних. Користувацькі запити та операції обробляються в кастомних імплементаціях репозиторіїв, таких як CustomContactRepositoryImpl.

Ця архітектура забезпечує чітке та зосереджене призначення для кожного шару. Вона ефективно розділяє обов'язки, що полегшує зміну, розширення та підтримку різних частин додатку без впливу на інші.

5.4.1 Інтерфейс REST API

У сучасних веб-застосунках використання RESTful API є стандартним підходом для забезпечення взаємодії між клієнтським- та серверним-компонентами. У контексті даної дипломної роботи сервер REST API виконує роль проміжної ланки між інтерфейсом користувача та базою даних. API надає набір HTTP-ендпоінтів, які дозволяють клієнт-частині керування контактами та історією дзвінків користувачів.

У розділі наведено детальний огляд REST API (таб 5.3), реалізованого у серверній частині розробленої системи. Для кожного ендпоінта описано його призначення, використовуваний HTTP-метод, очікувані вхідні та вихідні параметри, а також відповідну функціональність. Реалізоване API дотримується

принципів REST-архітектури, що забезпечує безстанову, безпечну та ефективну комунікацію.

Таблиця 5.3 – Ендпоїнти REST API

Ендпоінт	Опис	Відповіді при успіху
GET /api/contacts/user/{userId}	Отримання сторінки контактів користувача з можливістю пошуку.	200 OK (Сторінка контактів)
GET /api/contacts/{id}	Отримання детальної інформації про контакт за ID.	200 OK (Дані контакту)
POST /api/contacts/user/{userId}	Створення нового контакту.	201 OK (Дані створеного контакту)
PUT /api/contacts/{id}	Оновлення даних контакту (ім’я, біо, фото, номери). Якщо фото змінено на null — старе видаляється.	200 OK (Оновлені дані контакту)
DELETE /api/contacts/{id}	Видалення контакту та пов’язаного фото (якщо існує). Якщо контакт відсутній — операція не виконується.	204 OK
GET /api/photos/{photoId}	Отримання фото у форматі JPEG за ID.	200 OK (Зображення JPEG)
POST /api/photos	Завантаження фото. Виконується асинхронна оптимізація за ID. У випадку невдачі — генерується ImageOptimizationException.	201 OK (ID фото)
GET /api/history/user/{userId}	Отримання сторінки історії дзвінків користувача.	200 OK (Сторінка записів)
POST /api/history/user/{userId}	Створення запису історії дзвінка для користувача.	201 OK (Створений запис)
DELETE /api/history/user/{userId}	Видалення всієї історії дзвінків користувача.	204 OK
GET /api/history/user/{userId}/contact/{contactId}	Отримання сторінки історії дзвінків користувача з певним контактом.	200 OK (Сторінка записів)
DELETE /api/history/{recordId}	Видалення конкретного запису історії за ID.	204 OK
GET /api/accounts	Отримання сторінки акаунтів користувача з можливістю пошуку.	200 OK (Сторінка акаунтів)
GET /api/accounts/user/{userId}	Отримання інформації акаунт.	200 OK (Дані акаунту)
POST/api/accounts	Створення нового акаунту.	201 OK (Дані створеного контакту)
PUT /api/accounts/{accountId}	Оновлення даних акаунту.	200 OK (Оновлені дані акаунту)
DELETE /api/accounts/{accountId}	Видалення акаунту користувача.	204 OK

Розроблене REST API побудовано з дотриманням принципів RESTful-архітектури, що сприяє масштабованості, підтримуваності та простоті інтеграції з клієнт-застосунком. Забезпечуючи чітко визначений та захищений набір ендпоінтів, система надає кінцевим користувачам можливість зручно та безпечно взаємодіяти із базою даних через веб-інтерфейс.

5.4.2 Безпека серверної частини

Безпека є критичним аспектом будь-якого веб-застосунку, особливо у системах, що обробляють чутливі дані комунікацій. Серверна частина системи повинна гарантувати належний захист усіх API-ендпоінтів від несанкціонованого доступу, витоку даних та зловмисної активності. У цьому розділі описано механізми безпеки, реалізовані у сервері розробленої системи. Розглянуто способи впровадження автентифікації, авторизації, захищеної комунікації, а також керування запитами з різних доменів (CORS).

У розробленій системі безпека серверної частини реалізована з використанням Spring Security — потужного та гнучкого фреймворку для автентифікації та контролю доступу в Java-застосунках. Конфігурація безпеки централізована у класі WebSecurityConfig, де визначено основні налаштування, такі як автентифікація через OAuth2, валідація JWT-токенів, політика CORS та авторизація на рівні методів.

У системі реалізовано використання можливостей OAuth2 ресурс-сервера. Це забезпечує, що всі запити до захищених ресурсів повинні містити дійсний JWT (JSON Web Token), виданий довіреним сервером автентифікації. Валідація токенів здійснюється автоматично засобами Spring Security.

Використання JWT дозволяє впровадити безстатусний, масштабований та загальноприйнятий у галузі механізм захисту REST API. Токен містить claims, які ідентифікують користувача (authentication.name), що пізніше використовується для детальної авторизації доступу.

Основні налаштування безпеки HTTP визначені у bean-компоненті securityFilterChain. Дозволено запити лише з довіреного домену, вказаного у spring.security.cors-origin. Всі HTTP-методи та заголовки дозволені лише в межах

шляхів /api/**. Це гарантує, що лише клієнт-застосунки, що працюють із довіреного домену, можуть взаємодіяти з сервер-API.

Для реалізації детального контролю доступу використано анотації @PreAuthorize, що дозволяють обмежити доступ до ресурсів лише для користувачів з відповідними правами.

Деякі ендпоінти обмежені до контексту автентифікованого користувача. Це забезпечується порівнянням змінної шляху (userId) з іменем автентифікованого користувача (authentication.name), отриманого з JWT: @PreAuthorize("#userId == authentication.name"). Такий підхід використовується, наприклад, для: отримання та керування контактами користувача, створення та видалення записів історії дзвінків.

Більш складні перевірки доступу делеговано до сервісу AuthService, який реалізує логіку перевірки права власності користувача на певний контакт або запис історії дзвінків. Приклади:

– перевірка права отримання або оновлення контакту: @PreAuthorize("@authService.canGetContact(authentication.name, #id)");

– перевірка права видалення запису історії: @PreAuthorize("@authService.canDeleteRecord(authentication.name, #recordId)").

Усередині AuthService перевірка здійснюється шляхом запиту до бази даних та порівняння ID власника ресурсу з ID автентифікованого користувача.

У цьому розділі описано механізми безпеки, реалізовані у серверній частині. Система використовує перевірені практики безпеки, включаючи автентифікацію через OAuth2, що забезпечує безпечну та безстатусну ідентифікацію користувачів. Запити з інших доменів контролюються суворою політикою CORS, що обмежує доступ лише довіреним джерелам. Контроль доступу реалізовано на рівні методів із використанням анотацій, що гарантує доступ до ресурсів лише авторизованим користувачам. Додаткові перевірки прав доступу здійснюються через спеціалізовані методи сервісу, які перевіряють право власності користувача на ресурс перед виконанням дій. Сукупність цих механізмів сприяє захисту API від несанкціонованого доступу та забезпечує конфіденційність даних.

5.5 База даних

У цьому розділі розглядається структура та використання MongoDB бази даних. MongoDB була обрана через свою гнучкість, масштабованість та підтримку швидкої розробки за допомогою документно-орієнтованої моделі NoSQL.

На відміну від традиційних реляційних баз даних, які використовують таблиці та рядки, MongoDB використовує модель зберігання даних на основі документів. Це дозволяє досягти більшої гнучкості в представленні даних. Кожен документ може містити вкладені структури, що полегшує представлення складних ієрархічних даних. У нашому випадку це корисно для управління даними користувачів, такими як контакти, історія дзвінків та фотографії.

Гнучка схема MongoDB дозволяє швидше впроваджувати зміни та модифікації структур даних без необхідності коригувати складну реляційну схему. Це ідеально відповідає меті швидкої розробки та частих змін в схемі бази даних.

MongoDB безшовно інтегрується з Spring Boot, надаючи розробникам зручність роботи з знайомими репозиторіями Spring Data MongoDB. Це знижує витрати на розробку, необхідні для доступу до бази даних, та надає підтримку для типових операцій, таких як запити, збереження та оновлення документів.

MongoDB має вбудовані можливості для горизонтального масштабування та розроблена для обробки великих обсягів неструктурованих даних, що робить її ідеальним вибором для застосунків із зростаючими наборами даних.

Застосунок використовує три основні колекції в MongoDB: contacts, history та photos. Ці колекції призначені для зберігання ключових даних, пов'язаних із користувачами, їх історією дзвінків та фотографіями користувачів (рис 5.5).

Колекція contacts зберігає інформацію про контакти користувачів. Кожен документ у цій колекції представляє окремий контакт із наступними полями:

- _id: унікальний ідентифікатор, автоматично генерований MongoDB;
- user: ідентифікатор користувача, якому належить цей контакт. Зазвичай це унікальний ідентифікатор користувача;
- name: ім'я контакту;

– photo: ідентифікатор фотографії контакту;

– bio: коротка біографія або опис контакту;

– numbers: масив телефонних номерів, пов'язаних з контактом. Кожен елемент масиву містить тип (наприклад, "мобільний", "робочий") та сам номер телефону.

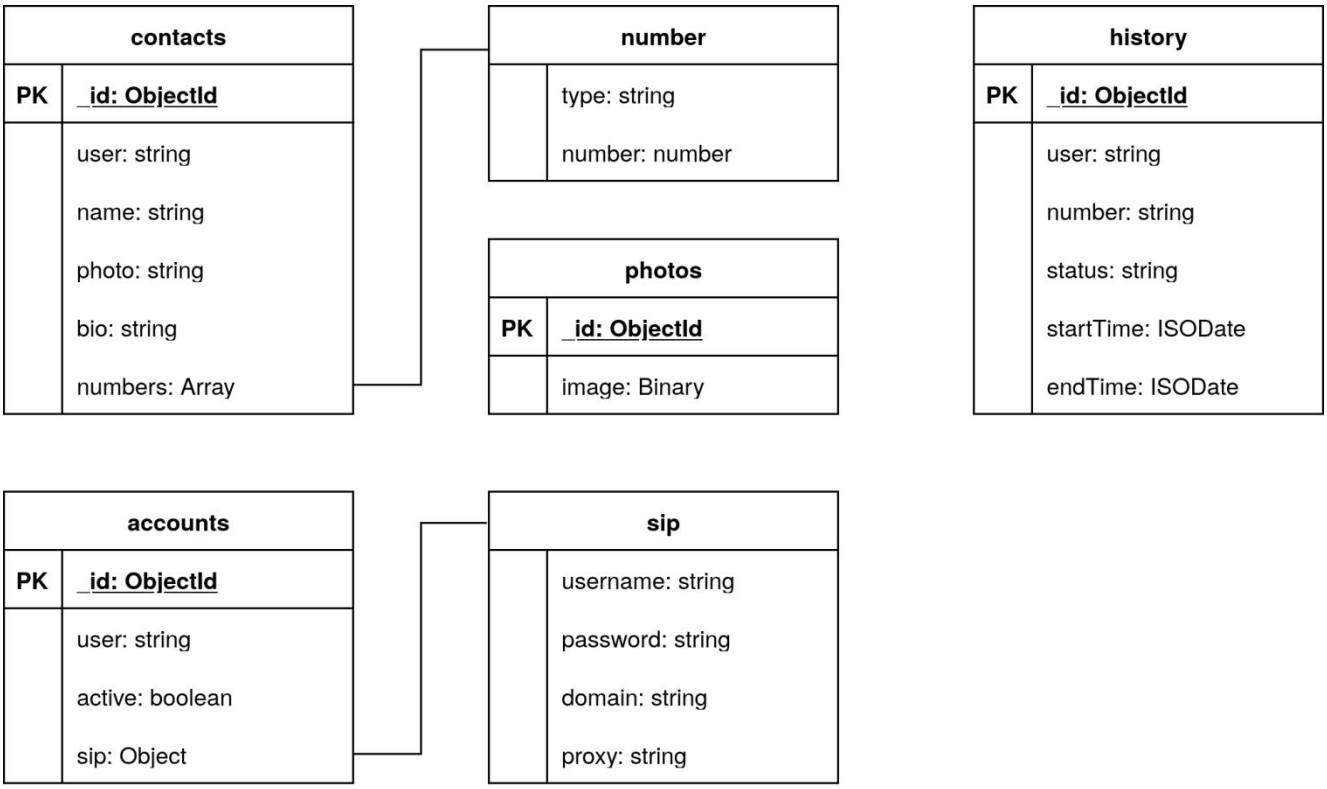

Рисунок 5.5 — Схема бази даних

Колекція history зберігає історію дзвінків кожного користувача. Це включає інформацію про кожен дзвінок, здійснений або отриманий користувачем, таку як номер, статус дзвінка та тривалість дзвінка. Документ у цій колекції має таку структуру:

– _id: унікальний ідентифікатор для кожного запису дзвінка;

– user: ідентифікатор користувача, пов'язаного з дзвінком;

– number: номер телефону, який був набраний або отриманий;

– status: статус дзвінка (наприклад, "завершено", "пропущено", "голосова пошта");

– startTime: час початку дзвінка;

– endTime: час завершення дзвінка.

Колекція photos зберігає зображення, пов'язані з користувачами, такі як фотографії профілю або інші медіафайли, пов'язані з контактами чи діяльністю користувачів. Кожен документ у колекції має таку структуру:

- _id: унікальний ідентифікатор для кожного документу з фотографією;
- image: бінарні дані, що представляють саме фото. Це зазвичай зберігається як тип даних Binary у MongoDB.

Колекція accounts зберігає інформацію про облікові записи користувачів системи. Кожен документ у цій колекції відповідає одному обліковому запису та містить такі поля:

- _id: унікальний ідентифікатор документа, автоматично створюється системою керування базами даних MongoDB;
- user: рядок, що представляє унікальний ідентифікатор користувача, якому належить обліковий запис;
- active: логічне або строкове значення, що вказує на стан облікового запису — активний або неактивний;
- sip: вкладений об’єкт, що містить налаштування SIP-профілю користувача і має наступні поля:
- username: рядок, що містить SIP-ім’я користувача, яке використовується для автентифікації на сигнальному сервері;
- password: рядок, що містить SIP-пароль користувача;
- domain: рядок, що визначає домен сигнального сервера SIP;
- proxy: рядок, який вказує адресу, за якою клієнт встановлює з’єднання із SIP-сервером (проксі).

Отже, MongoDB використовується в застосунку для управління даними, такими як облікові записи користувачів, контакти, історія дзвінків, фотографії. Підхід на основі документів та NoSQL забезпечує гнучкість та масштабованість, в той час як безшовна інтеграція з Spring Boot спрощує розробку та управління базою даних.

5.6 Налаштування Asterisk сервера

Для реалізації голосового зв’язку через вебзастосунок на основі SIP необхідна надійна та сумісна серверна інфраструктура. Asterisk, як відкритий сервер комунікацій, забезпечує потужну підтримку як протоколу SIP, так і технології WebRTC.

Перед налаштуванням підтримки WebRTC-клієнтів необхідно мати повнофункціональну інсталяцію Asterisk, версії не нижче 15.5. Рекомендовано здійснювати компіляцію з вихідних кодів, що дозволяє обрати необхідні модулі. До обов’язкових модулів належать:

- res_crypto — для криптографічної підтримки;
- res_http_websocket — для підключення через HTTP WebSocket;
- res_pjsip_transport_websocket — забезпечує транспорт WebSocket для PJSIP;
- codec_opus — необов’язковий, але рекомендований для високоякісної передачі звуку.

Сучасні веббраузери вимагають захищеного з’єднання WebSocket (WSS), тому HTTP-сервер Asterisk має бути налаштований на підтримку TLS (Transport Layer Security). У розробці допустимо використовувати самопідписані сертифікати, однак для продакшн-середовища рекомендовано застосовувати сертифікати від довірених центрів сертифікації, таких як Let's Encrypt.

У каталозі contrib/scripts міститься скрипт ast_tls_cert, що дозволяє згенерувати власний сертифікаційний центр (CA) та серверний сертифікат. Результатом виконання є набір ключових файлів, зокрема asterisk.crt та asterisk.key, які потрібні для TLS-конфігурації.

Вбудований HTTP-сервер Asterisk необхідно активувати та налаштувати для підтримки захищених з’єднань. Для цього редагується файл /etc/asterisk/http.conf (рис 5.6).

Після внесення змін сервер потрібно перезапустити. Перевірку можна здійснити за допомогою CLI-команди http show status, яка має показати, що HTTPS-сервер активний та доступний WebSocket-ендпоінт /ws.

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.crt
tlsprivatekey=/etc/asterisk/keys/asterisk.key
```

Рисунок 5.6 — Конфігурація захищеного з’єднання

Для підтримки протоколу WSS (WebSocket Secure) в стеку PJSIP необхідно додати відповідний блок у файл pjsip.conf (рис. 5.7):

```
[transport-wss]
type=transport
protocol=wss
bind=0.0.0.0
```

Рисунок 5.7 — Налаштування підтримки WSS

Для забезпечення реєстрації клієнтів та узгодження медіа необхідно визначити три об’єкти PJSIP: auth — автентифікація, aor — адреса реєстрації та endpoint — кінцева точка (рис. 5.8).

Налаштування webrtc=yes активує критично важливі функції, зокрема SRTP через DTLS, ICE-неготіацію для NAT-траверсу, та мультиплексування RTP. Кодеки opus та ulaw забезпечують широку сумісність з браузерами та SIP-клієнтами.

Для забезпечення з’єднання клієнтів необхідно переконатися, що TCP-порт 8089 відкритий у фаєрволі, оскільки саме на ньому Asterisk очікує WSS-з’єднань.

Після завершення конфігурації сервер слід перезапустити. Перевірка працездатності виконується за допомогою сумісного SIP-клієнта (наприклад, JsSIP або SIP.js у браузері), який підключається до порту 8089 по WSS, автентифікується за заданими параметрами та встановлює аудіозв’язок з використанням дозволених кодеків.

```
[webrtc_client]
type=aor
max_contacts=5
remove_existing=yes

[webrtc_client]
type=auth
auth_type=userpass
username=webrtc_client
password=webrtc_client

[webrtc_client]
type=endpoint
aors=webrtc_client
auth=webrtc_client
dtls_auto_generate_cert=yes
webrtc=yes
context=default
disallow=all
allow=opus,ulaw
```

Рисунок 5.8 — Реєстрація SIP клієнта

Таким чином, описана процедура дозволяє налаштувати сервер Asterisk як сервер для WebRTC-сумісної SIP-комунікації, що є основою для розробки веборієнтованої VoIP-системи. Така конфігурація забезпечує захищений та сумісний обмін медіаданими й сигналізацією з SIP-клієнтами через Інтернет.

6 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Клієнтська частина системи забезпечує інтегрований та орієнтований на користувача інтерфейс для управління голосовими комунікаціями на основі протоколу VoIP та супутніми користувацькими даними. Застосунок реалізовано як односторінковий веб-застосунок (SPA), побудований з використанням React та TypeScript, що дозволяє забезпечити безшовний користувацький досвід, охоплюючи управління SIP-викликами, роботу з телефонною книгою, перегляд історії викликів та налаштування облікових записів користувача.

Інтерфейс клієнтської частини структуровано таким чином, щоб логічно проводити користувача через етапи автентифікації, реєстрації SIP та подальшого використання телекомунікаційних функцій, забезпечуючи інтуїтивний та ефективний сценарій взаємодії.

При першому доступі до застосунку користувачі, які не пройшли автентифікацію, автоматично перенаправляються на сторінку автентифікації Keycloak (рис. 6.1).

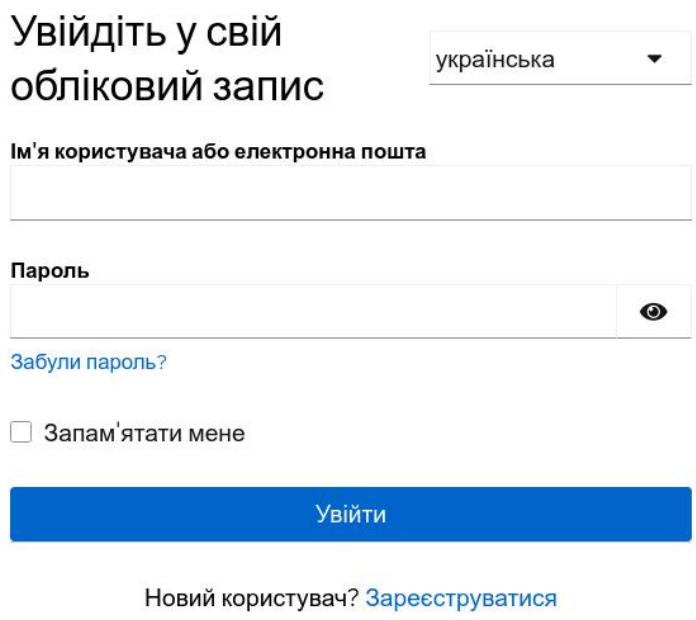

Рисунок 6.1 — Сторінка входу у обліковий запис

Процес автентифікації реалізується відповідно до протоколу OAuth2/OIDC, що гарантує безпечне управління користувацькими сесіями згідно з галузевими стандартами. Keycloak виконує роль провайдера ідентифікації (IdP) та, за потреби, забезпечує функціональність самостійної реєстрації користувачів. Після успішної автентифікації користувач повертається до застосунку, де йому необхідно налаштувати параметри SIP-акаунта.

На етапі конфігурації SIP-акаунта користувачеві пропонується ввести три обов'язкових параметри: SIP ім'я користувача, пароль та адресу домену SIP-сервера, яким в поточній реалізації є сервер Asterisk PBX (рис. 6.2). Ці параметри можуть бути введені вручну або завантажені з попередньо збереженого конфігураційного файлу. Після введення даних ініціюється встановлення SIP-з'єднання через WebSocket (WSS) за допомогою бібліотеки JsSIP, яка забезпечує передачу SIP-сигналізації через WebRTC. У разі успішного встановлення з'єднання користувач автоматично перенаправляється на сторінку викликів. У випадку помилки користувач залишається на сторінці конфігурації з відображенням відповідних повідомлень про відмову з'єднання.

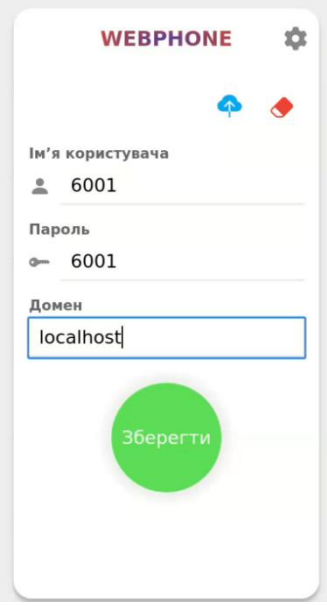

Рисунок 6.2 — Сторінка конфігурації SIP-акаунта

Сторінка викликів виконує роль основного інтерфейсу для ініціації SIP-викликів (рис. 6.3). Тут користувач може вводити телефонні номери вручну та

ініціювати дзвінки через встановлене з’єднання WebRTC. Під час активної сесії виклику інтерфейс надає користувачу додаткові функції, зокрема вимкнення мікрофона, регулювання гучності динаміка та відправлення DTMF-сигналів, що дає змогу взаємодіяти з автоматизованими телефонними системами або IVR (рис. 6.4).

Рисунок 6.3 — Сторінка викликів

Рисунок 6.4 — Сторінка активної сесії виклику

Після завершення виклику деталі сесії — зокрема час початку та завершення, тривалість виклику та його статус — автоматично зберігаються в історії викликів користувача (рис. 6.5).

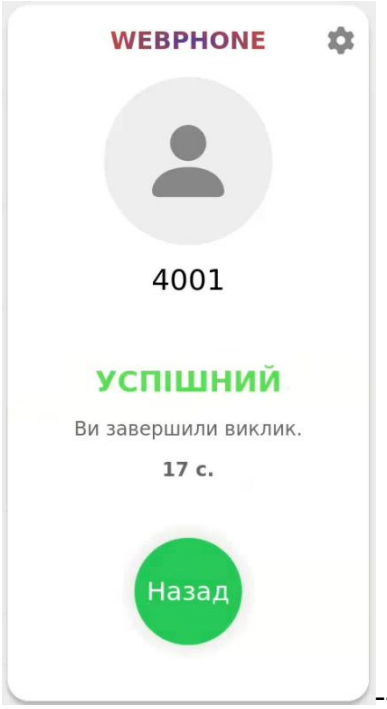

Рисунок 6.5 — Сторінка завершення виклику

Застосунок підтримує класифікацію викликів за статусами: вхідний, вихідний, пропущений та невдалий, які відображаються з використанням візуального розрізнення для покращення зручності перегляду.

Сторінка історії викликів надає користувачам хронологічно впорядкований список усіх минулих дзвінків, згрупований за датами (рис. 6.6). Для оптимізації користувацького досвіду та зменшення початкового часу завантаження реалізовано механізм безкінечного прокручування (infinite scrolling), за яким додаткові записи історії завантажуються з серверу по мірі прокручування сторінки вниз. Кожен запис містить детальну інформацію, зокрема напрямок виклику, тривалість та часові позначки. У разі, якщо виклик було здійснено або прийнято від контакту, що збережений у телефонній книзі користувача, запис доповнюється ім’ям контакту, фотографією та активним посиланням на сторінку детальної інформації про контакт. Користувачі мають можливість здійснювати

повторні дзвінки безпосередньо з інтерфейсу історії викликів, а також вибірково видаляти окремі записи або повністю очищувати історію.

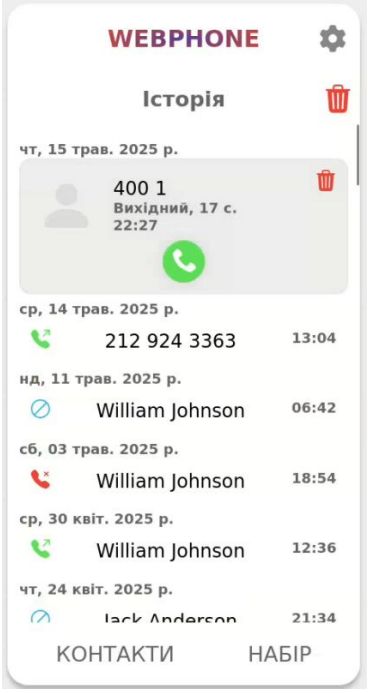

Рисунок 6.6 — Сторінка завершення виклику

Сторінка контактів відображає пошуковий список збережених користувачем контактів (рис. 6.7).

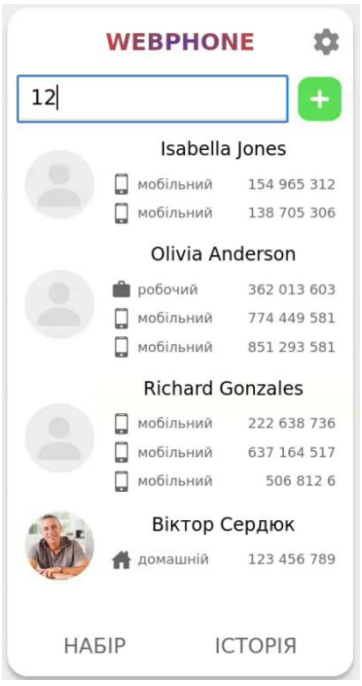

Рисунок 6.7 — Сторінка контактів

Пошук здійснюється за іменем, номерами телефонів або біографією контакту. Кожен запис супроводжується фотографією, що підвищує зручність візуальної ідентифікації. Для підвищення ефективності також застосовано механізм безкінечного прокручування.

Перехід до конкретного контакту відкриває сторінку з повною інформацією про контакт, що включає до десяти телефонних номерів, біографію та історію всіх викликів, пов’язаних з номерами цього контакту (рис. 6.8). З цієї сторінки користувач може ініціювати виклик на будь-який із вказаних номерів.

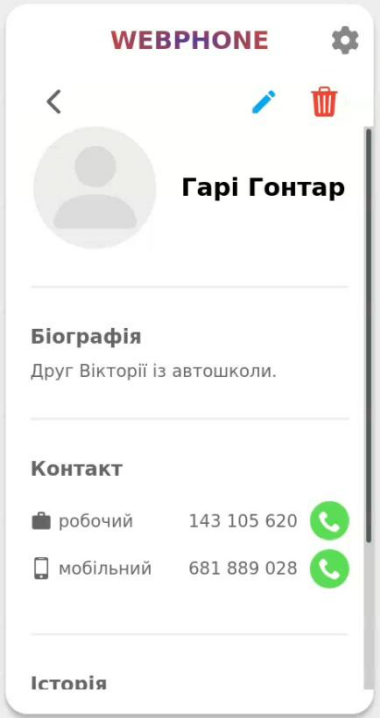

Рисунок 6.8 — Сторінка повної інформації про контакт

Крім того, на сторінці детальної інформації реалізовані функції редагування та видалення контакту. Видалення супроводжується діалогом підтвердження для запобігання випадкової втрати даних (рис. 6.9). Редагування дозволяє змінювати всі поля інформації, включаючи завантаження фотографії (розміром до 8 МБ, з подальшою оптимізацією на сервері) (рис. 6.10). Застосунок гарантує цілісність даних, забезпечуючи унікальність імен та телефонних номерів у межах телефонної книги користувача.

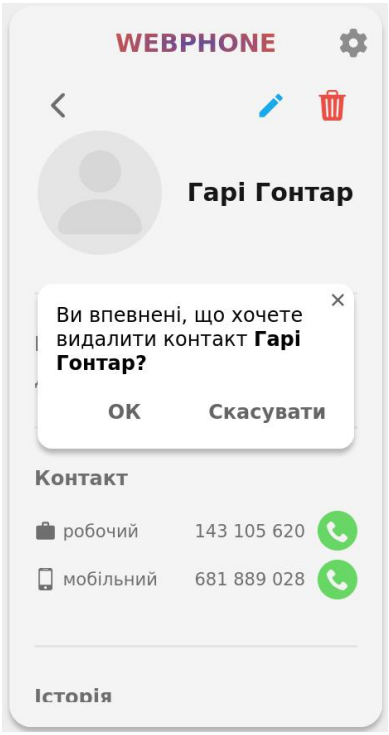

Рисунок 6.9 — Вікно підтвердження видалення контакту

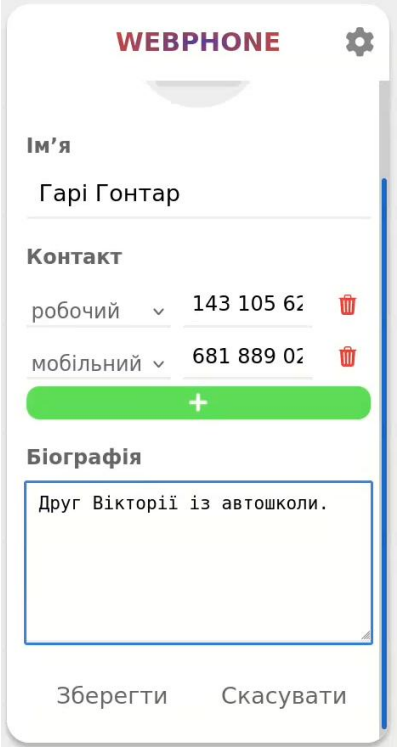

Рисунок 6.10 — Сторінка редагування контакту

Створення нового контакту здійснюється через окрему сторінку, доступну з телефонної книги (рис. 6.11). Інтерфейс створення є аналогічним до інтерфейсу редагування, проте працює виключно в режимі створення.

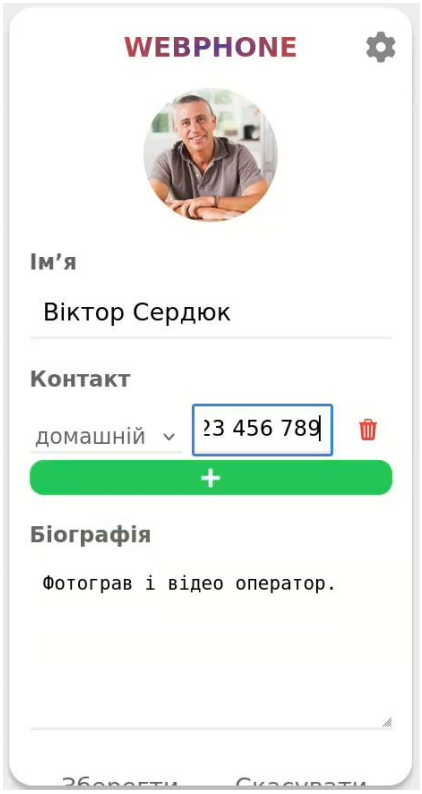

Рисунок 6.11 — Сторінка створення контакту

Сторінка налаштувань надає користувачам можливість конфігурувати певні аспекти застосунку (рис. 6.12).

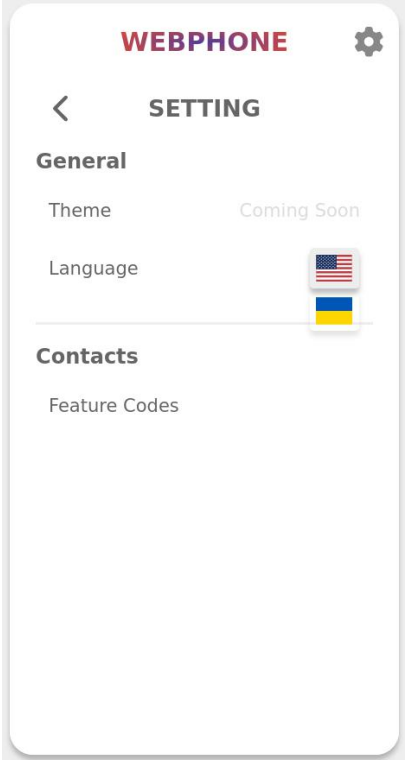

Рисунок 6.12 — Сторінка налаштувань

На поточному етапі доступні зміна мови інтерфейсу тоді як інші опції, перебувають у стадії розробки: налаштування теми оформлення та імпорт стандартних кодів функцій. Незважаючи на поки що обмежену функціональність, модульна архітектура клієнтської частини забезпечує гнучкість для майбутнього розширення і впровадження додаткових опцій персоналізації.

Поточна версія застосунку орієнтована виключно на кінцевих користувачів, у майбутніх ітераціях планується впровадження адміністративних функцій, доступних лише користувачам із підвищеними привілеями. Адміністративна панель дозволить уповноваженим користувачам керувати обліковими даними SIP-акаунтів, активувати або деактивувати користувацькі акаунти, а також здійснювати системні налаштування безпосередньо з інтерфейсу застосунку. Розподіл ролей і прав доступу забезпечуватиметься за допомогою механізмів контролю доступу на основі ролей, які реалізовані в сервері авторизації Keycloak.

ВИСНОВКИ

1. На основі аналізу сучасних підходів, методів і алгоритмів реалізації VoIP-зв’язку обґрунтовано доцільність використання протоколу SIP як основи для забезпечення захищеної та масштабованої комунікації через Інтернет. SIP був обраний завдяки своїй гнучкості, відкритості, модульності та активній підтримці спільноти.

2. Проведений огляд існуючих рішень для VoIP-комунікації — як комерційних хмарних сервісів, так і самостійно розгорнутих open-source платформ — дозволив ідентифікувати їх ключові обмеження щодо автономності, безпеки та гнучкості налаштування. Це дало підстави для обґрунтованого вибору відкритих технологій і програмних засобів, таких як Asterisk, Spring Boot, React, TypeScript, MongoDB і Keycloak, які у поєднанні забезпечують відповідність функціональним і нефункціональним вимогам проєкту.

3. Розроблено веб-застосунок для VoIP-комунікації, який забезпечує захищену SIP-телефонію через браузер, не потребуючи додаткового встановлення клієнтського ПЗ. Система поєднує модульну архітектуру, сучасний інтерфейс та можливість автономного розгортання, орієнтована на потреби малих організацій з підвищеними вимогами до приватності та контролю над інфраструктурою.

4. Архітектура розробленої системи передбачає використання концепції модульного моноліту з інтеграцією окремих незалежних сервісів, що відповідають за SIP-комунікацію, автентифікацію користувачів, обробку запитів та зберігання даних. Такий підхід забезпечує баланс між спрощеним розгортанням та гнучкістю підтримки і масштабування.

5. Розроблений застосунок надає користувачам функціональність для реєстрації та автентифікації, ініціації та прийому SIP-дзвінків, керування контактами та перегляду статусів у режимі реального часу. Інтерфейс побудований з використанням React та WebRTC для забезпечення плавної роботи з медіа-даними без посередників.

6. Хоча формальне тестування системи не було окремо задокументовано, під час розробки здійснено інтеграційні та модульні перевірки основних

функціональних компонентів, зокрема SIP-викликів, авторизації, та взаємодії з сервером. Попередня експлуатація підтвердила працездатність і відповідність заявленим вимогам до продуктивності та безпеки.

7. У подальшому можливим напрямком удосконалення програмної системи є розширення функціоналу шляхом інтеграції відеозв’язку, створення мобільного застосунку, впровадження наскрізного шифрування дзвінків, а також розробка адміністративної панелі для моніторингу та керування сеансами в реальному часі.

Таким чином, у результаті виконаної дипломної роботи розроблено захищений, автономний веб-застосунок для VoIP-комунікації з використанням протоколу SIP, орієнтований на потреби малих організацій. Система забезпечує безпечне голосове спілкування через стандартні веб-браузери без залежності від сторонніх хмарних сервісів. Запропоноване рішення поєднує модульність, простоту розгортання, підтримку реального часу та контроль над обробкою персональних даних, що робить його ефективним інструментом для організацій, які прагнуть цифрового суверенітету у сфері внутрішньої комунікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kemp S. Internet use in 2024. *Datareportal*. 31.01.2024. URL: https://datareportal.com/reports/digital-2024-deep-dive-the-state-of-internet-adoption.

2. Mordor Intelligence. Mobile VoIP Market Size & Share Analysis - Growth Trends & Forecasts (2025-2030). *Mordor Intelligence*. URL: https://www.mordorintelligence.com/industry-reports/mobile-voip-market (дата звернення: 16.05.2025).

3. Tuleun W. Design of an asterisk-based VoIP system and the implementation of security solution across the VoIP network. *World Journal of Advanced Research and Reviews*. 2024. С. 875–906. ISSN 2581-9615. URL: https://doi.org/10.30574/wjarr.2024.23.1.2048.

4. Ahson S., Ilyas M. VoIP Handbook: Applications, Technologies, Reliability, and Security. United States of America : CRC Press, 2009.

5. Sadiwala R. Analysis of Security Threats of VoIP Systems. *SHODH SANGAM – A RKDF University Journal of Science and Engineering*. 2018. С. 34–46. ISSN 2581-5806.

6. Sudi Lema H., Simba F., Cosmas Mushi J. Security Enhancement of SIP Protocol in VoIP Communication. *Journal of ICT Systems*. 2023. С. 71–92. URL: https://doi.org/10.56279/jicts.v1i2.32.

7. Ahson S., Ilyas M. SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol. United States of America : CRC Press, 2009.

8. WebRTC Testing: Challenges and Practical Solutions / B. García та ін. *Universidad Rey Juan Carlos*.

9. Understanding and Estimating Quality of Experience in WebRTC Applications / Garcı́a B. та ін. *Universidad Rey Juan Carlos*. 2018. URL: https://doi.org/10.1007/s00607-018-0669-7.

10. Emmanuel E., Dirting B. A Peer-To-Peer Architecture For Real-Time Communication Using Webrtc. *Journal of Multidisciplinary Engineering Science Studies*. 2017. С. 1671–1683.

11. Sassu A., Saenz-Cogollo J., Agelli M. Deep-Framework: A Distributed, Scalable, and Edge-Oriented Framework for Real-Time Analysis of Video Streams. *Center for Advanced Studies, Research and Development in Sardinia (CRS4)*. 2021. URL: https://doi.org/10.3390/s21124045.

12. What is RingCentral?. *RingCentral*. URL: https://www.ringcentral.com/us/en/blog/what-is-ringcentral/ (дата звернення: 16.05.2025).

13. The small business VoIP phone system for modern work. *Nextiva*. URL: https://www.nextiva.com/products/small-business-voip (дата звернення: 16.05.2025).

14. Business Phone System. *Cloudtalk*. URL: https://www.cloudtalk.io/business-phone-system/ (дата звернення: 16.05.2025).

15. Webb T. 10 Best VoIP Services Of 2025. *Forbes*. 07.03.2025. URL: https://www.forbes.com/advisor/business/software/best-voip-service/.

16. Ready for FreePBX Now? *FreePBX*. URL: https://www.freepbx.org/get-started/ (дата звернення: 16.05.2025).

17. Getting Started with Asteris. *Asterisk*. URL: https://www.asterisk.org/get-started/ (дата звернення: 16.05.2025).

18. Drew R. The Best Free and Open-Source PBX Software. *GetVoip*. 22.01.2025. URL: https://getvoip.com/blog/open-source-pbx/ (дата звернення: 16.05.2025).

19. Enterprise Grade Phone System. *3CX*. URL: https://www.3cx.com/phone-system/ (дата звернення: 16.05.2025).

20. Skowronski A. What is the future of Java in today’s enterprise?. *Virtuslab*. 08.09.2023. URL: https://virtuslab.com/blog/business-insights/what-is-the-future-of-java/ (дата звернення: 16.05.2025).

21. Introduction to Spring Framework. *GeeksforGeeks*. 02.05.2025. URL: https://www.geeksforgeeks.org/introduction-to-spring-framework/ (дата звернення: 16.05.2025).

22. Chen S., Thaduri U., Ballamudi V. Front-End Development in React: An Overview. *Engineering International,*. 2019. С. 117–126. ISSN 2409-3629.

23. Beattie-Hood B. Modern TypeScipt. Apress, Berkeley, CA, 2023. ISBN 978-1-4842-9722-3. URL: https://doi.org/10.1007/978-1-4842-9723-0_1.

24. Chauhan A. A Review on Various Aspects of MongoDb Databases. *International Journal of Engineering Research & Technolog*. 2019. С. 90–92. ISSN 2278-0181.

25. Divyabharathi D., Nagaraj G. A Review on Identity and Access Management Server (KeyCloak). *International Journal of Security and Privacy in Pervasive Computing*. 2020.

26. Critelli A. An introduction to Asterisk. *RedHat*. 09.01.2020. URL: https://www.redhat.com/en/blog/introduction-asterisk (дата звернення: 17.05.2025).

ДАДАТОК А

Веб-застосунок для VoIP-комунікації на основі SIP протоколу

Текст програми

НТУУ “КПІ ім. Ігоря Сікорського”

Аркушів 7

Київ - 2025

Додаток А

Програмні засоби:

- мова програмування — TypeScript
- веб-фреймворк — React
- бібліотека для роботи зі SIP — jssip

useSip.ts

```
import { useEffect, useState, useRef } from "react";
import { UA } from "jssip";
import { RTCSessionEvent } from "jssip/lib/UA";
import { ConnectionState, IncomingCallHandler, CallInfo, SipAccount, init, CallState, CallDirection,
Call, ConnectionStateWrapper, Session, CallStateWrapper, getCallOriginator } from "../lib/sip";
import { log, warn } from "../util/log";
import { IncomingAckEvent, IncomingEvent, OutgoingAckEvent, OutgoingEvent, RTCSession }
from "jssip/lib/RTCSession";
import { IncomingRequest, OutgoingRequest } from "jssip/lib/SIPMessage";

interface Return {
  account: SipAccount | null;
  setAccount: (a: SipAccount) => void;
  connection: ConnectionStateWrapper;
  connectionError?: string;
  calls: Call[];
  makeCall: (number: string) => void;
  onIncomingCall: (handler: IncomingCallHandler) => void;
  answerCall: (id: string) => void;
  hangupCall: (id: string) => void;
  sendDtmf: (id: string, tone: string | number) => void;
  toggleHold: (id: string) => void;
  toggleMute: (id: string) => void;
}

export function useSip(): Return {

  const [account, setAccount] = useState<SipAccount | null>(null);
  const [connection, setConnection] = useState(ConnectionState.DISCONNECTED);
  const [connectionError, setConnectionError] = useState<string>();
  const [calls, setCalls] = useState(new Map<string, Session>());
  const handleIncomingCallRef = useRef<IncomingCallHandler>();
  const uaRef = useRef<UA>();

  useEffect(() => {
    if (!account) return;

    setConnectionError("");

    uaRef.current = init(account);

    const ua = uaRef.current;
```

```
    ua.on("connecting", (e) => {
      log("CONNECTING", e);
      setConnection(ConnectionState.CONNECTING);
    });

    ua.on("connected", (e) => {
      log("CONNECTED", e);
      setConnection(ConnectionState.CONNECTING);
    });

    ua.on("disconnected", (e) => {
      log("DISCONNECTED", e);
      setConnection(ConnectionState.DISCONNECTED);
      setConnectionError(e.reason || "Cannot establish connection to the server, please check entered
domain");
      if (e.error) ua.stop();
    });

    ua.on("registered", (e) => {
      log("REGISTERED", e);
      setConnection(ConnectionState.CONNECTED);
    });

    ua.on("unregistered", (e) => {
      log("UNREGISTRED", e);
      setConnection(ConnectionState.DISCONNECTED);
    });

    ua.on("registrationFailed", (e) => {
      log("REGISTRATION_FAILED", e);
      setConnection(ConnectionState.DISCONNECTED);

      if (e.response.status_code === 403) {
        setConnectionError("Registration failed: invalid username or password");
      } else {
        setConnectionError(e.response.reason_phrase);
      }
    });

    ua.on("newRTCSession", (e: RTCSessionEvent) => {
      log("NEW_RTC_SESSION", e);
      const session = e.session;
      const callId =  addCall(session, e.request, e.originator);

      if (e.originator === "remote" && handleIncomingCallRef.current) {
        handleIncomingCallRef.current(callId);
      }

      if (session.connection) {
        session.connection.ontrack = (e) => handleTracks(callId, e);
      }

      session.on("peerconnection", (e) => {
```

```
  log("peerconneciton", e);
  e.peerconnection.ontrack = (event) => handleTracks(callId, event);
});

session.on("connecting", (e) => {
  log("connecting", e);
});

session.on("sending", (e) => {
  log("sending", e);
});

session.on("progress", (e: IncomingEvent | OutgoingEvent) => {
  log("progress", e);
});

session.on("accepted", (e: IncomingEvent | OutgoingEvent) => {
  log("accepted", e);
});

session.on("confirmed", (e: IncomingAckEvent | OutgoingAckEvent) => {
  log("comfirmed", e);
  updateCall(callId, call => ({ ...call, state: CallState.ESTABLISHED }));
});

session.on("hold", (e) => {
  log("hold", e);
  updateCall(callId, call => ({ ...call, isOnHold: true }));
});

session.on("unhold", (e) => {
  log("unhold", e);
  updateCall(callId, call => ({ ...call, isOnHold: false }));
});

session.on("muted", (e) => {
  log("muted", e);
  updateCall(callId, call => ({ ...call, isMuted: true }));
});

session.on("unmuted", (e) => {
  log("unmuted", e);
  updateCall(callId, call => ({ ...call, isMuted: false }));
});

session.on("ended", (e) => {
  log("ended", e);
  const endTime = new Date();
  const endedBy = getCallOriginator(e.originator);
  const state = CallState.ENDED;
  const remoteStream = undefined;
  updateCall(callId, call => ({ ...call, state, remoteStream, endTime, endedBy }));
  scheduleCallRemoval(callId);
});
```

```
      session.on("failed", (e) => {
        log("failed", e);
        const endTime = new Date();
        const endedBy = getCallOriginator(e.originator);
        const state = CallState.ENDED;
        const remoteStream = undefined;
        const error = e.cause;
        updateCall(callId, call => ({ ...call, state, remoteStream, endTime, endedBy, error }))
        scheduleCallRemoval(callId);
      });

    });

    ua.start();

    return () => {
      ua.stop();
      ua.removeAllListeners();
    };

  }, [account]);

  const handleTracks = (callId: string, e: RTCTrackEvent) => {
    updateCall(callId, call => {
      e.streams[0]?.getTracks()
        .filter(track => call.remoteStream && !call.remoteStream.getTrackById(track.id))
        .forEach(track => call.remoteStream?.addTrack(track));
      return call;
    });
  }

  const addCall = (session: RTCSession, request: IncomingRequest | OutgoingRequest, originator:
string): string => {
    const number = originator == "local" ?
      (request?.ruri?.user || "unknown") :
      (request?.from?.uri?.user || "unknown");
    const info = {
      id: session.id,
      number: number,
      state: CallState.PROGRESS,
      direction: originator == "local" ? CallDirection.OUTGOING : CallDirection.INCOMING,
      startTime: session.start_time || new Date(),
      isOnHold: session.isOnHold().local,
      isMuted: session.isMuted().audio,
      startedBy: getCallOriginator(originator)!,
      remoteStream: new MediaStream(),
    }
    setCalls((calls) => {
      if (calls.get(info.id)) { warn(`Cannot add call with dublicate id: ${info.id}`); return calls; }
      const newCalls = new Map(calls);
      newCalls.set(info.id, { session, info });
      return newCalls;
    });
```

```
    return info.id;
  }

  const updateCall = (id: string, updateFunc: (c: CallInfo) => CallInfo) => {
    setCalls(calls => {
      const call = calls.get(id)
      if (!call) { warn("Cannot update unexisting call:", id); return calls; }
      const newCalls = new Map(calls);
      newCalls.set(id, { ...call, info: updateFunc(call.info) });
      return newCalls;
    });
  }

  const removeCall = (id: string) => {
    setCalls(calls => {
      const call = calls.get(id)
      if (!call) { warn(`Cannot remove unexisting call: ${id}`); return calls; }
      if (call.info.state !== CallState.ENDED) { warn(`Cannot remove active call: ${id} -
${call.info.state}`); return calls; }
      call.session.removeAllListeners();
      const newCalls = new Map(calls);
      newCalls.delete(id);
      return newCalls;
    });
  }

  const scheduleCallRemoval = (id: string) => {
    setTimeout(() => removeCall(id), 3600 * 1000);
  }

  const makeCall = (number: string): void => {
    if (!uaRef.current) { warn("Cannot make call as UA is undefined"); return; }
    uaRef.current.call(number, { "mediaConstraints": { "audio": true } });
  }

  const onIncomingCall = (handler: IncomingCallHandler): void => {
    handleIncomingCallRef.current = handler;
  }

  const answerCall = (id: string): void => {
    const call = calls.get(id);
    if (!call) { warn(`Cannot answer unexisting call: ${id}`); return; }
    call.session.answer();
  }

  const hangupCall = (id: string): void => {
    const call = calls.get(id);
    if (!call) { warn(`Cannot hangup unexisting call: ${id}`); return; }
    call.session.terminate();
  }

  const sendDtmf = (id: string, tone: string | number): void => {
    const call = calls.get(id);
    if (!call) { warn(`Cannot sendDtmf to unexisting call: ${id}`); return; }
```

```
    call.session.sendDTMF(tone);
  }

  const toggleHold = (id: string) => {
    const call = calls.get(id);
    if (!call) { warn(`Cannot toggle hold on unexisting call: ${id}`); return; }
    if (call.session.isOnHold().local) {
      call.session.unhold();
    } else {
      call.session.hold();
    }
  }

  const toggleMute = (id: string) => {
    const call = calls.get(id);
    if (!call) { warn(`Cannot toggle mute on unexisting call: ${id}`); return; }
    if (call.session.isMuted().audio) {
      call.session.unmute({ audio: true });
    } else {
      call.session.mute({ audio: true });
    }
  }

  return {
    account,
    setAccount,
    connection: new ConnectionStateWrapper(connection),
    connectionError,
    calls: mapCalls(calls),
    makeCall,
    onIncomingCall,
    answerCall,
    hangupCall,
    sendDtmf,
    toggleHold,
    toggleMute,
  };
};


function mapCalls(calls: Map<string, Session>): Call[] {
  return Array.from(calls.values()).map(call => call.info).map(mapCall);
}

function mapCall(callInfo: CallInfo): Call {
  return { ...callInfo, state: new CallStateWrapper(callInfo.state) }
}
```