НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Індивідуальний дослідницький проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Система обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів»

Виконав:
студент IV курсу, групи ІА-81
Топольський Артем Володимирович _____

Керівник:
Асистент кафедри ІСТ
Шинкевич Микола Костянтинович _____

Засвідчую, що у цьому проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАВДАННЯ

на індивідуальний дослідницький проєкт студенту
Топольському Артем Володимировичу

1. Тема проєкту «Система обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів», керівник проєкту асистент кафедри ІСТ
Шинкевич Микола Костянтинович.
2. Термін подання студентом проєкту: 15 червня 2022 року
3. Вихідні дані до проєкту: серверний та клієнтський додатки, що дозволяють обмінюватися текстовими повідомленнями та відслідковувати місцезнаходження контактів.
4. Зміст пояснювальної записки: вступ, огляд існуючих рішень, аналіз обраної системи, обґрунтування вибору засобів розробки, розробка діаграм програмного застосунку, програмна реалізація застосунку, висновки.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): діаграма варіантів використання, діаграма послідовності, діаграма розгортання, схеми таблиць бази даних.
6. Дата видачі завдання 1 грудня 2021 року

Календарний план

№ з/п	Назва етапів виконання проєкту	Термін виконання етапів проєкту	Примітка
1	Огляд та аналіз існуючих рішень	09.05.2022	
2	Аналіз засобів розробки	16.05.2022	
3	Розробка діаграм програмного застосунку	23.05.2022	
4	Розробка програмного застосунку	06.06.2022	
5	Створення документації проєкту	13.06.2022	
6	Оформлення документації проєкту	20.06.2022	

Студент Артем ТОПОЛЬСЬКИЙ

Керівник Микола ШИНКЕВИЧ

АНОТАЦІЯ

Топольський А.В. Система обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів. КПІ ім. Ігоря Сікорського, Київ, 2022.

Проєкт містить 60 сторінок тексту, 29 рисунків, 16 посилань на літературні джерела, 4 конструкторскі документи.

Ключові слова: обмін повідомленнями, GPS-трекінг, браузерний та серверний застосунок.

Даний дипломний проєкт має за ціль розробку програмного забезпечення для обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів у реальному часі. Було розроблено серверний та клієнтський браузерний застосунок.

Застосунок використовує систему GPS для відстеження власного місцезнаходження. Для трансляції місцезнаходження, отримання місцезнаходження контактів та обміну текстовими повідомленнями використовується протокол WebSocket.

Програмна складова застосунку реалізована на високорівневій мові програмування JavaScript, розширеній використанням TypeScript для забезпечення надійності та простого розширення функціональності застосунку.

Для побудови API застосунку було використано мову GraphQL та супутні бібліотеки. Збереження даних було реалізовано з використанням об’єктно-реляційної бази даних PostgreSQL. Для реалізації клієнтської частини було використано сучасні бібліотеки та фреймворки, такі як React, Apollo Client, що спрощує реалізацію клієнтської частини на різних платформах (Web-застосунок, мобільний застосунок для Android, мобільний застосунок для IOS), дозволяючи перевикористовувати створені компоненти.

Крім програмного застосунку було також розроблено перелік необхідних діаграм: діаграми варіантів використання, послідовності, розгортання та схема таблиць у базі даних.

ANNOTATION

Topolskyi A.V. Text messaging system with contacts GPS-tracking. Igor Sikorsky KPI, Kyiv, 2022.

The project contains 60 pages of text, 29 pictures, 16 references to literary sources, 4 design documents.

Keywords: messaging, GPS tracking, browser and server application.

The project objective is to develop software for text messaging with the contact's GPS-tracking ability in real-time. Also, there have been developed the server and client browser application.

The application uses GPS to track your location. WebSocket is applied to broadcast location, retrieve contacts, and exchange text messages.

The software component of the application is implemented in a high-level JavaScript programming language, extended using TypeScript to ensure reliability and easy extension of application functionality.

GraphQL and related libraries were in use for building the application API. The implemented data storage using the PostgreSQL object-relational database. Implementing the client's part includes such modern libraries as Apollo Client for simplifying the realization processes on different platforms (Web application, mobile application for Android, mobile application for iOS). An approach like this allows the reusing of created components.

A list of required diagrams was developed as an addition: usage diagrams, sequences, deployments, and database tables chart.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА81.280БАК.003 ПЗ	Пояснювальна записка	60		
6	А3	ІА81.280БАК.003 Д1	Система обміну текстовими	1		
7			повідомленнями з			
8			можливістю GPS-трекінгу			
9			контактів.			
10			Діаграма варіантів			
11			використання			
12	А3	ІА81.280БАК.003 Д2	Система обміну текстовими	1		
13			повідомленнями з			
14			можливістю GPS-трекінгу			
15			контактів.			
16			Діаграма розгортання			
17	А3	ІА81.280БАК.003 Д3	Система обміну текстовими	1		
18			повідомленнями з			
19			можливістю GPS-трекінгу			
20			контактів.			
21			Діаграма послідовності			
22	А3	ІА81.280БАК.003 Д4	Система обміну текстовими	1		
23			повідомленнями з			
24			можливістю GPS-трекінгу			
25			контактів.			
26			Схема таблиць бази даних			
27						
28						

					ІА81.280БАК.003 ТП		
Зм.	Аркуш	№ докум.	Підпис	Дата			
Розроб.		Топольський А.В.			Система обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів. Відомість проєкту	Літ. Т	Аркуш 1 / Аркушів 1
Керівн.		Шинкевич М.К.				КПІ ім. Ігоря Сікорського	
Затв.							

Пояснювальна записка
до індивідуального дослідницького проєкту
на тему: «Система обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів»

Київ – 2022 року

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

ЦП – центральний процесор.

ООП – об’єктно-орієнтоване програмування.

СУБД – система управління базою даних.

API – Application Programming Interface.

ORM – Object-Relational Mapping.

GPS – Global Positioning System.

UML – Unified Modeling Language.

HTTP – HyperText Transfer Protocol.

HTTPS – HyperText Transfer Protocol Secure.

WS – WebSocket.

REST – Representational State Transfer.

CRUD – Create, Read, Update, Delete.

MVVC – Model-View-View-Controller.

SQL – Structured Query Language.

CSS – Cascading Style Sheets.

SCSS – Syntactically Awesome Stylesheets.

DOM – Document Object Model.

ВСТУП

Пошук та створення нових способів коммунікації та обміну різноманітною інформацією завжди були одними з найактуальніших задач, а в 21 сторіччі дане питання продовжує набирати обертів. Так, зі стрімким розвитком діджиталізації, месенджери дуже швидко заполонили інформаційний простір, витіснивши більш звичні для старшого покоління телефонні дзвінки та електронні листи - зараз ці способи використовуються лише в формальних випадках або як виключення (як правило, термінова або офіційна розмова). Месенджери надають можливість тримати постійний контакт з близькими, друзями та знайомими в абсолютно всіх сферах життя: побутовій, робочій, медійній тощо.

Застосунки для листування вже давно перестали виконувати лише фунцію обміну текстовими повідомленнями - зараз месенджери надають можливість спілкуватися за допомогою дзвінків і відеозв‘язку, а також аудіо та відеоповідомлень; обмінюватися файлами без форматних обмежень; проводити трансляції та безліч другорядних функцій.

Однак, на сьогоднішній день вищеописані можливості не можна назвати новизною, а скоріш переліком функціональних особливостей для розробки мінімально життєздатного застосунку. Тому багато компаній знаходяться в активному пошуку новітніх, а іноді й автентичних рішень в галузі месенджерів.

Одним з таких рішень є впровадження трансляції геопозиції користувача. Така можливість може бути надзвичайно корисною як і на побутовому рівні (наприклад, корегування маршруту відносно іншого користувача), так і в особливих випадках (відстеження девайсу у разі викрадення). В більш звичних месенджерах (таких як Telegram або Viber) дана особливість є другорядною, втім існують застосунки, в яких трекінг є основною функцією, а обмін повідомленнями відходить на інший план. Слід

зазначити, що можливість відстеження та транслювання позиції користувача наразі реалізована виключно на мобільних платформах.

Об’єктом дослідження є обмін текстовими повідомленнями та геопозицією у реальному часі. Предметом – система обміну.

Метою роботи є створення веб-застосунку для обміну повідомленнями та трекінгу геопозиції. Розроблення браузерної версії програми також перебачає створення адаптації для мобільних телефонів, однак в даній дипломній роботі задача головним чином зосереджена саме на створенні комп‘ютерної версії. Важливо зауважити, що з міркувань збереження приватності та особистих кордонів, кожному з користувачів буде доступна функція, що приховує геопозицію від інших юзерів.

Під час створення дипломної роботи були вирішені наступні задачі:

– огляд та аналіз існуючих систем для обміну повідомленнями та відстеження геопозиції;

– вибір засобів розробки;

– розробка діаграм програмного додатка;

– розробка програмного забезпечення.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

На сьогоднішній день існує досить велика кількість систем обміну повідомленнями, які надають можливість обмінюватися актуальною геолокацією. Однак, можна зазначити, що основною функцією переважної більшості систем – є саме обмін текстовими повідомленнями, аудіо та відеозаписами, файлами тощо. Функція трансляції та відстеження місцезнаходження контактів – є скоріше допоміжним додатковим рішенням. Існуючі рішення, як правило, є незалежними від платформи, можуть працювати у будь-якому оточенні. Однак функція обміну геолокацією є доступною переважно на мобільних пристроях.

Далі наведено приклади систем, які дозволяють обмінюватися повідомленнями та відстежувати геолокацію контактів. Для дослідження було обрано найбільш популярні та розвинені системи, а саме: «Telegram», «Viber», «Zenly».

1.1 Система «Telegram»

Telegram – це програмний комплекс, розроблений з метою забезпечити можливість безпечного спілкування. Telegram використовує провідні методики шифрування повідомлень, дозволяє передавати аудіо та відеоповідомлення, файли. Клієнтські додатки доступні на більшості платформ, а саме: Android, iOS, Windows Phone, Windows, macOS, Linux.

Розглянемо більш детально функцію трансляції геопозиції. Для зчитування геопозиції користувача Telegram використовує вбудовані API пристрою. Для відображення отриманих координат на мапі Telegram виокристовує сервіс Google Maps.

Для того, щоб поділитися своїм місцезнаходженням користувач повинен обрати пункт «Локація». Після цього доступними є дві опції: одноразово

відправити своє місцезнаходження або транслювати його з певною періодичністю: наприклад, один раз на хвилину.

Із переваг реалізації даної функції у програмному застосунку «Telegram» можна виділити простий та інтуїтивно зрозумілий для користувача інтерфейс.

Недоліками можна назвати неможливість використовувати дану функцію на усіх платформах (доступна лише на мобільних пристроях) та неможливість відслідковувати одночасно місцезнаходження декількох користувачів на одній мапі. Приклад використання геопозиції у застосунку «Telegram» представлено на рисунку 1.1.

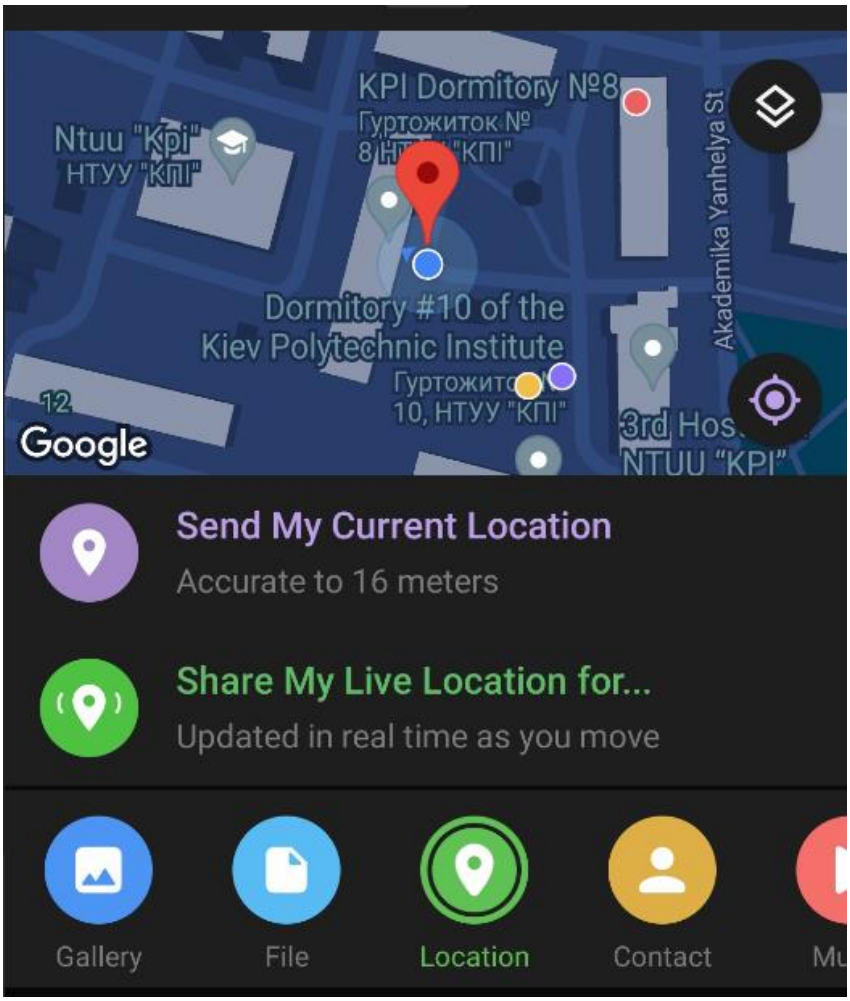

Рис. 1.1 – Функція трансляції геопозиції у системі «Telegram»

1.2 Система «Viber»

Viber - застосунок для дзвінків і обміну повідомленнями. Застосунок ідентифікує користувачів за номером телефона, але не використовує мобільну мережу. Месенжер працює на пристроях з операційною системою iOS, Android, macOS, Windows, Linux та Ubuntu.

Розглянемо функцію відправки місцезнаходження. Viber не дозволяє транслювати геопозицію у реальному часі, наявна лише функція одноразової відправки поточних координат контакта. Приклад наведено на рисунку 1.2.

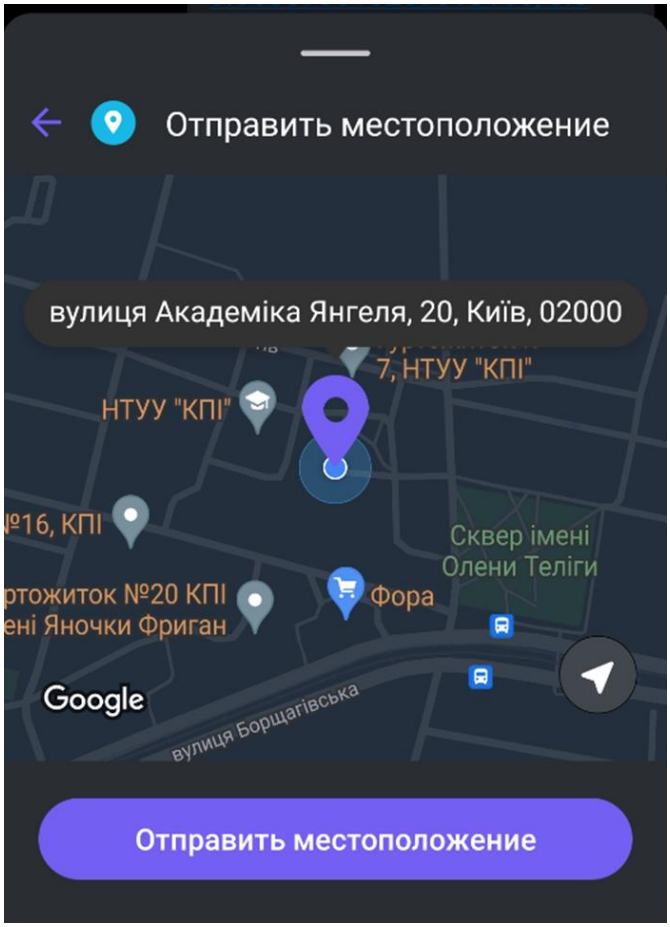

Рис. 1.2 – Функція трансляції геопозиції у системі «Viber»

Після цього у чаті будуть відображені координати та можливість перейти до застосунку з мапами для перегляду місцезнаходження. Відображення координат у застосунку демонструється на рисунку 1.3.

Із недоліків такого підходу можна виділити необхідність переходу до стороннього програмного застосунку Google Maps для перегляду безпосереднього місцезнаходження контакту на мапі.

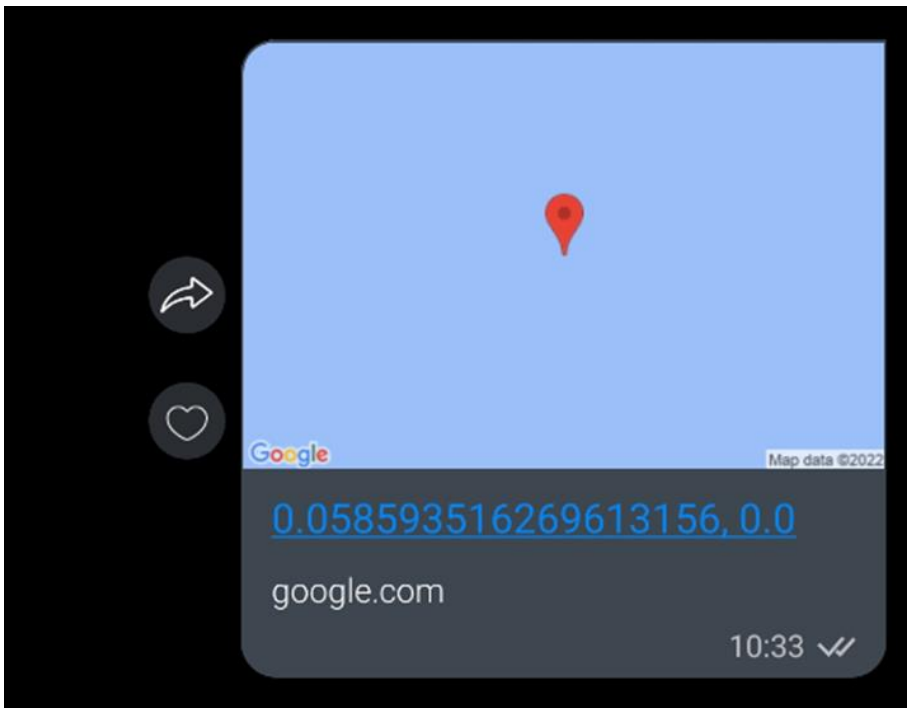

Рис. 1.3 – Відображення координат у системі «Viber»

1.3 Система «Zenly»

Zenly – це застосунок для обміну місцезнаходженням у режимі реального часу, створений у 2015 році. Основна функція Zenly – ділитися локаціями з друзями та відстежувати їх. Застосунок може ділитися не тільки місцезнаходженням у режимі реального часу, а й напрямком та швидкістю. Застосунок доступний для пристроїв на операційних системах Android та iOS.

На глобальній мапі, інтегрований за допомогою картографічної системи Google Maps можна побачити своє місцезнаходження та місцезнаходження

усіх контактів. Крім цього застосунок дозволяє обмінюватися повідомленнями з обраним контактом та показує на мапі актуальну інформацію про нього. Крім цього динамічно розраховується відстань між контактами та їх швидкість, що дозволяє відобразити імовірний час зустрічі. Застосунок має досить простий та інтуїтивно зрозумілий користувацький інтерфейс, дозволяє налаштовувати приватність при користуванні. Інтерфейс застосунку відображено на рисунку 1.4.

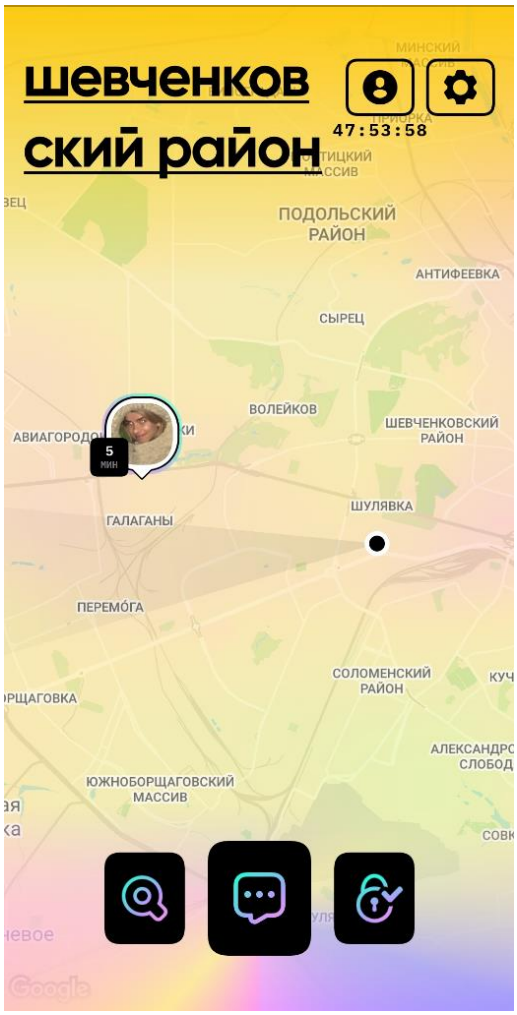

Рис. 1.4 – Функція трансляції геопозиції у системі «Zenly»

Висновки до розділу 1

Розглянемо принципи роботи розглянутих застосунків, основні переваги, які можна наслідувати, та недоліки, яких слід уникати. Усі розглянуті системи використовують Geolocation API пристрою для отримання актуальних координат користувача. Для відображення мапи усі застосунки використовують картографічний сервіс Google Maps, який надає змогу відображати високоякісні супутникові знімки.

Із відмінностей у реалізації варто виділити необхідність переходу до стороннього застосунку для перегляду мапи у клієнті «Viber». Така поведінка є не дуже зручною для користувача, доцільно було б реалізувати вбудоване відображення мап у застосунку.

Ще одним недоліком усіх розглянутих систем є неможливість використання функції відстежування місцезнаходження на усіх платформах, крім клієнтських додатків, реалізованих на мобільних пристроях. Доцільно було надати користувачам можливість користуватися браузерними версіями додатків.

Із переваг розглянутих додатків можна виділити досить простий та інтуїтивно зрозумілий користувацький інтерфейс.

2 АНАЛІЗ ТЕХНОЛОГІЙ ПРОЄКТУ

2.1 Дослідження GPS

GPS (Глобальна система позиціонування) – супутникова радіонавігаційна система. Це одна з глобальних навігаційних супутникових систем, яка надає інформацію про геопозицію та час GPS-приймачу в будь-якому місці на Землі або поблизу неї, де є безперешкодна пряма видимість для чотирьох або більше супутників GPS. Такі перешкоди, як гори чи будівлі, можуть блокувати слабкі GPS сигнали.

GPS не вимагає від користувача передачі будь-яких даних, і він працює незалежно від будь-якого телефонного або Інтернет-з’єднання, хоча ці технології можуть підвищити корисність інформації про позиціонування GPS.

2.1.1 Принцип роботи

GPS-приймач обчислює власне чотиривимірне положення в просторі-часі на основі даних, отриманих від кількох супутників GPS. Кожен супутник несе точний запис про своє положення та час і передає ці дані на приймач.

Супутники мають дуже стабільні атомні годинники, які синхронізовані один з одним і з наземними годинниками. Будь-який зсув від часу коригується щодня. Таким же чином місцеположення супутників відомі з великою точністю. Приймачі GPS також мають годинники, але вони менш стабільні та менш точні.

Оскільки швидкість радіохвиль є постійною і не залежить від швидкості супутника, час затримки між тим, коли супутник передає сигнал і приймає його, пропорційна відстані від супутника до приймача. Щонайменше, чотири супутники повинні бути в полі зору приймача, щоб він міг обчислити чотири невідомі величини (три координати положення і відхилення власного годинника від супутникового часу) [1].

2.1.2 Сфери застосування

– GPS ефективний в міському господарстві: при заміні каналізаційних, газо- та водопроводів, електро- та телефонних ліній. Автомобілі швидкої допомоги та ремонтні бригади можуть використовувати GPS для навігації безпосередньо до місця аварії зв’язку. Час їх прибуття та від'їзду точно фіксується разом з їхніми коментарями та планом обслуговування;

– у сільському господарстві системи картографування GPS можуть допомогти описати особливості територій, які інтенсивно використовуються в сільському господарстві. Можна точно визначити такі характеристики, як мікроклімат, тип ґрунту, посівні площі, пошкодження комахами або хворобами, обсяг виробництва;

– GPS допомагає збирати дані про типи ґрунтів, які в поєднанні з тривимірними моделями територій дозволяють виділити певні аспекти для прогнозування територій, які потребують спеціального управління. Крім того, GPS можна використовувати для картографування розташування колодязів та інших джерел води; фіксація розмірів озер та їх стану; реєстрація ареалів поширення риби та тваринного світу; зміни берегової лінії, полів і кліматичних поясів;

– військове використання сигналів GPS дозволяє покращити контроль над збройними силами шляхом точного наведення зброї або армії на ціль. На дні океану GPS необхідний для пошуку затонулих кораблів або виконання інших технічних операцій, на суші не менш важливо використання навігаційних пристроїв;

– система GPS дозволяє визначити місце розташування в будь-якій точці на суші, на морі та в навколоземному просторі. Залежно від галузей застосування, діапазон яких досить широкий, а також від вартості, яка може

коливатися від сотень до декількох тисяч доларів, види GPS-приймачів також різноманітні. Різновиди приймачів можна умовно поділити на чотири групи:

– персональні GPS-приймачі індивідуального застосування. Ці моделі є досить малими за розмірами та володіють широким набором сервісних функцій: від базових навігаційних, включаючи можливість формування й розрахунку маршрутів прямування, до функції прийому та передачі електронної пошти;

– автомобільні GPS-приймачі призначені для встановлення в будь-якому виді наземних транспортних засобів й мають можливість підключення зовнішнього приймально-передавального обладнання для автоматичної передачі параметрів руху на диспетчерські пункти;

– авіаційні GPS-приймачі використовують для визначення маршруту літальних апаратів, включаючи комерційну авіацію;

– морські GPS-приймачі, оснащені ультразвуковим ехолотом, а також додатковими змінними картриджами з картографічною й гідрографічною інформацією для конкретних берегових районів.

2.2 Дослідження протоколу WebSocket

Для обміну інформацією у реальному між клієнтом та сервером використовується протокол WebSocket. Протокол забезпечує двонаправлений повнодуплексний канал зв’язку через один TCP-сокет. WebSocket – може використовуватися у веб-браузерах, веб-серверах та будь-якими клієнт-серверними застосунками [2].

Дані передаються між клієнтом та сервером через постійне з’єднання. Передача відбувається у двох напрямках у вигляді пакетів, без розриву з’єднання та додаткових запитів.

2.2.1 Відкриття WebSocket

Під час відкриття веб-сокета клієнт запитує сервер за допомогою заголовків запиту про підтримку прокотолу WebSocket. Якщо сервер підтримує таке з'єднання – то воно відкривається і елементи системи можуть почати обмін пакетами за протоколом WebSocket. Схема обміну зображена на рисунку 2.1.

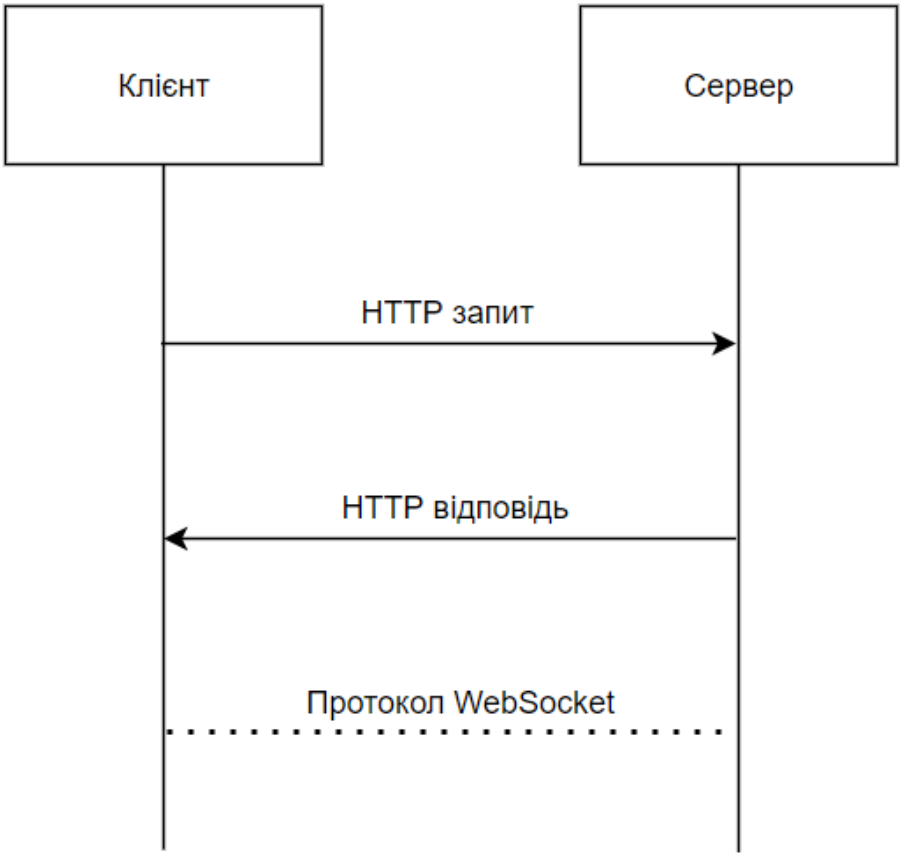

Рис. 2.1 – Схема встановлення з’єднання протоколом WebSocket

2.2.2 Передача даних протоколом WebSocket

Потік даних у WebSocket складається з «фреймів», фрагментів даних, які можуть бути надіслані будь-якою стороною і які можуть бути наступних типів:

– «текстові фрейми» - містять текстові дані, якими обмінюються сторони;

– «бінарні фрейми» - містять бінарні дані, якими обмінюються сторони;

– «пінг-понг фрейми» - використовуються для перевірки з’єднання; відправляються з сервера, браузер реагує на них автоматично;

– також існує «фрейм закриття з'єднання» та інші службові фрейми.

2.2.3 Закриття з'єднання

Зазвичай, коли сторона бажає закрити з’єднання (клієнт та сервер мають рівні права), вона надсилає «фрейм закриття з’єднання» з кодом закриття та вказує причину у вигляді тексту.

Найбільш розповсюджені значення кодів закриття з’єднання:

– 1000 – за замовчуванням, нормальне закриття;

– 1006 – з’єднання було втрачене;

– 1001 – сторона відключилась, наприклад сервер вимкнутий або користувач залишив сторінку;

– 1009 – повідомлення занадто велике для обробки;

– 1011 – непередбачена помилка на сервері.

Загалом коди WebSocket схожі на коди HTTP, але будь-які коди, менші за 1000 є зарезервованими. Спроба встановити такий код викличе помилку.

Висновки до розділу 2

У даному розділі було розглянуто технологію GPS та протокол обміну пакетами у реальному часу WebSocket.

Технологія GPS виділяється своєю надійністю та широко використовується у багатьох галузях. Завдяки своїй високій розповсюдженості

ця технологія є чудовим вибором для реалізації відстеження та подальшої трансляції місцезнаходження користувачів програмного застосунку у реальному часі.

Також було розглянуто протокол WebSocket. Основним застосуванням даного протоколу є передача динамічних даних у реальному часі. Даний протокол підтримується більшістю сучасних браузерів, є надійним та нескладним у використанні. Завдяки цим перевагам протокол WebSocket є чудовим рішенням для реалізації обміну повідомленнями у реальному часі та трансляції актуальної геопозиції.

3 ОБҐРУНТУВАННЯ І ВИБІР ЗАСОБІВ РОЗРОБКИ

3.1 Вибір засобів розробки серверної частини

3.1.1 Вибір мови програмування

Серверна частина застосунку – код, який виконується на віддаленому сервері. Серверна частина відповідальна за зв’язок з БД (операції запису, зчитування), формування відповідей на запити користувачів, виконання необхідних обчислень тощо. В даному випадку серверна частина буде відповідальною за такі функції:

– отримання нових повідомлень від користувачів та запис їх до БД;

– розсилання повідомлень від одного користувача усім учасникам чату;

– трансляція геопозиції користувача усім контактам.

Було розглянуто найпопулярніші сучасні мови програмування та обрано ту, що найбільше задовольняє вимоги до функціоналу:

– Java;

– C#;

– Python;

– JavaScript;

– Ruby.

3.1.1.1 Мова програмування Java

Java – об’єктно-орієнтована мова програмування, яка широко використовується для розробки веб-додатків корпоративного масштабу, Android застосунків, настільних додатків, наукових програм. Основною перевагою Java є принцип «Write Once Run Anywhere», тобто, скомпільований Java код може бути виконаним на будь-якій платформі, яка підтримує Java (це може бути персональний комп’ютер, мобільний телефон, розумний годинник,

або, навіть, пилосос). Спочатку Java код компілюється у байт-код, який не залежить від машини, а потім виконується за допомогою JVM.

Крім того, Java підтримує багатопоточність, що дозволяє одночасно використовувати декілька ядер ЦП. З інших переваг Java можна виділити дотримання концепцій ООП, автоматичне виділення пам’яті та збір сміття.

Переваги Java:

– безпечність – Java використовує концепції ООП, так як: інкапсуляція, наслідування, що сприяє підвищенню її безпечності;

– незалежність від платформи – завдяки своїй віртуальній машині Java може бути виконана майже на будь-якій платформі;

– підтримка багатопоточності – Java надає розробнику ряд засобів для роботи з потоками, що дозволяє розробляти більш оптимізовані застосунки.

Недоліки Java:

– низька швидкість – Java потребує більше пам’яті порівняно із іншими мовами програмування. Також частина ресурсів необхідна для функціонування віртуальної машини;

– об’ємність коду – хоча код Java і є надійним, однак в той же час він є досить об’ємним та громіздким.

3.1.1.2 Мова програмування C#

C# - мова загального призначення, яка спочатку була розроблена компанією Microsoft переважно для фреймворку .Net. Крім розробки веб-серверів, зараз C# широко використовується в багатьох областях: розробка додатків для Windows, розробка ігор тощо. Ця мова надає різноманітні функції, такі як швидша компіляція, сумісність, масштабованість та оновлення, орієнтованість на компоненти, структурованість. Крім того, C#

пропонує великий набір бібліотек, які допомагають здійснювати швидший та ефективний процес розробки.

Переваги C#:

– чудова інтеграція з системою Windows – немає необхідності налагоджувати додаткову конфігурацію, щоб виконати C#;

– користі засоби для роботи в команді – завдяки вбудованим утилітам Visual Studio робота над проєктом командою є досить простою.

Недоліки C#:

– залежність від Windows. Windows – це єдина система, на якій може працювати C# сервер, що значно обмежую розробників, адже сервери на Linux є значно дешевшими;

– низька швидкість роботи – у зв'язку з процесом компіляції C# має досить не низьку швидкість роботи.

3.1.1.3 Мова програмування Python

Python – мова, яка відома своєю сумісністю з передовими технологіями, такими як машинне навчання, Інтернет речей (IoT), наука про даны. Однак Python також чудово підходить для розробки веб-серверів, завдяки величезній колекції стандартних бібліотек, які значно спрощують процес розробки, роблять його ефективнішим.

Також помітними на унікальними перевагами Python є висока читабельність коду, відносно проста інтеграція з іншими мовами та портативність.

Переваги Python:

– швидка розробка – Python не вимагає від розробників написання великої кількості обслуговуючого коду. Просто речі можна писати простим чином, що значно пришвидшую розробку;

– гнучкість – Python дозволяє вести розробку у будь-якій популярній парадигмі програмування;

– розвинена спільнота – спільнота розробників на Python є досить великою, завдяки чому існую велика кількість бібліотек, фреймворків тощо.

Недоліки Python:

– низька швидкість виконання – Python є досить повільною мовою програмування;

– непристосованість до мобільних платформ – так як Python є відносно неоптимізованою мовою з точки зору використання пам'яті та потужностей пристроя, тому розробка на Python для мобільних платформ є досить поганою ідеєю.

3.1.1.4 Мова програмування JavaScript

JavaScript – одна з найпопулярніших сучасних мов програмування. Ця популярність зумовлена тим, що це єдина мова, яка може виконуватись в браузерах, отже усі клієнтські браузерні застосунки будуються на основі цієї мови.

Однак завдяки фреймворку Node.js JavaScript може використовуватись і як серверна мова програмування. Значною перевагою Node.js є те, що усі запити від користувачів оброблюються одним потоком за допомогою спеціальних workers. Це означає, що немає необхідності виділяти окремий потік під кожного користувача, що робить сервер незалежним від кількості ядер ЦП.

Крім того JavaScript має ряд переваг, а саме: висока читабельність коду, можливість писати у об’єктно-орієнтовану та функціональному стилях.

За замовчуванням JavaScript не має суворої типізації. Однак ця проблема вирішується використанням TypeScript – мовою-обгорткою, що дозволяє описувати типи та компілюється у JavaScript.

Переваги JavaScript:

– інтерпретованість – JavaScript є інтерпретованою мовою програмування, а отже не потребує етапу компіляції;

– висока популярність – мова є дуже популярною і має значну спільноту, завдяки чому розроблена велика кількість додаткових засобів, що пришвидшують та полегшують розробку;

– незалежність від платформи – завдяки розробленим бібліотекам та фреймворкам JavaScript можна використовувати на будь-якій платформі: на сервері, клієнті, мобільних пристроях тощо.

Недоліки:

– динамічна типізація – JavaScript має динамічну типізацію, що робить розробку великих систем досить важкою та ненадійною;

– безпека на стороні клієнта – так як JavaScript використовується на стороні клієнта, то можуть виникати ситуації, коли помилки у коді призводять до проблем із безпекою.

3.1.1.5 Розширення можливостей та вирішення проблем JavaScript за допомогою TypeScript

При створенні великих систем розробники очікують, що код завжди буде працювати передбачувано та надійно. При розробці на JavaScript існує необхідність у великій кількості перевірок для забезпечення надійності. Крім цього, при значному збільшенні об’ємів коду - процес підтримки JavaScript застосунків стає значно складнішим. Саме для вирішення таких проблем існує TypeScript.

TypeScript – це мова програмування, розроблена компанією Microsoft у 2012 році [3]. Загалом вона складається з трьох частин:

– синтаксис мови програмування;

– компілятор;

– сервіс для редактора коду.

За допомогою компілятора можна помітити більшість дефектів коду ще до його використання. Компілятор перетворює код, написаний мовою TypeScript у JavaScript, при цьому аналізуючи програму та шукаючи проблемні місця, про які повідомляється розробник. Загалом розробник має змогу налаштовувати правила для TypeScript, роблячи перевірку менш або більш суворою.

За останні роки TypeScript значно набирає популярність та стає нормою розробки.

3.1.1.6 Мова програмування Ruby

Ruby — це інтерпретована мова програмування загального призначення, яка підтримує різні парадигми програмування, такі як процедурне, функціональне та об’єктно-орієнтоване програмування. Ця мова широко використовується для веб-розробки в усьому світі. Ruby зосереджується на підвищенні продуктивності розробників, що в кінцевому підсумку прискорює процес веб-розробки. Ця мова підтримує майже всі основні платформи, такі як Windows, Mac і Linux. Ruby в значній мірі заснована на багатьох інших мовах програмування, таких як: Perl, Lisp, Eiffel, Ada тощо. Автоматичне збирання сміття, велика стандартна бібліотека, користувальницька поведінка диспетчеризації, гнучкість і масштабованість, централізоване керування пакетами за допомогою RubyGems тощо — це деякі з визначних функцій, які пропонує Ruby.

Переваги Ruby:

– простота використання – має інтуїтивно зрозумілий і зручний синтаксис, який більшості програмістів легко зрозуміти;

– усталені стандарти розробки – мова є дуже популярною і за час існування напрацювала певні стандарти розробки, що роблять програми якісними, надійними та легкорозширюваними.

Недоліки:

– низька гнучкість – налаштування за допомогою Ruby є досить нетривіальним. Міграція баз даних, налаштування маршрутизації та інші можуть подовжити процес розробки продукту;

– неправильне рішення може коштувати дорого – при допущені помилки на початкових етапах розробки, зміна архітектури на кінцевих етапах може бути значно складнішою у порівнянні з іншими мовами та фреймворками.

3.1.2 Розгляд платформи Node.js

Node.js – платформа, побудована на основі рушія Google V8 та написана мовою JavaScript. Основною ціллю платформи є можливість виконання JavaScript-скриптів не тільки у браузері, а й на сервері. Для забезпечення обробки значної кількості паралельних запитів у Node.js використовується асинхронна модель виконання коду.

Така реалізація асинхронності є значною перевагою Node.js, адже при побудові серверів на основі цієї платформи кількість запитів від користувачів, які можуть оброблюватися паралельно, може бути значно більшою, ніж використовуючи мови, у яких відбувається прив’язка до кількості потоків процесора.

Також значною перевагою Node.js є швидкість розробки цією мовою: розробка прототипа продукту не займає багато часу. При гарно продуманій архітектурі – майбутній розвиток та розширення застосунку не є проблемою.

Разом із Node.js також поставляється пакетний менеджер npm, який дозволяє зручним чином встановлювати, видаляти та оновлювати додаткові пакети.

3.1.3 Вибір бази даних та ORM фреймворка

В першу чергу необхідно визначитись між вибором реляційної або нереляційної бази даних.

Реляційна база даних – база даних, побудована на реляційній моделі даних. Іншими словами, це база даних, яка представляє собою набір нормалізованих відношень різного ступеня. Дані у таких БД є сукупністю елементів, організованих у вигляді формально описаних таблиць, у яких записи доступні для багаторазового зчитування багатьма різними способами без необхідності реорганізації таблиць.

Нереляційні БД, на відміну, є менш структурованими та обмеженими у форматі, тому забезпечують більшу гнучкість та адаптивність. Такі БД мають більшу перевагу, якщо необхідно працювати з великим неорганізованим та неструктурованими об’ємами даних.

Зважаючи на це, найбільш доцільним варіантом буде використання реляційної БД, адже основні сутності застосунку (Контакт, Чат, Повідомлення) органічно буде пов’язати певними відношеннями.

3.1.3.1 База даних PostgreSQL

Для практичної реалізації було обрано реляційну БД PostgreSQL. PostgreSQL – одна з найбільш професійних реляційних баз даних. Вона має відкритий вихідний код та ряд переваг, таких як: надійність, стабільність, висока безпечність системи та легка масштабованість [4]. Усі ці фактори сприяли вибору даної БД.

3.1.3.2 Sequelize

Для спрощення роботи та оптимізації запитів до БД у сучасних застосунках використовують ORM – об'єктно-реляційне відображення. ORM – це технологія програмування, яка пов'язує бази даних із концепціями об'єктно-орієнтованих мов програмування. Іншими словами, ORM дозволяють взаємодіяти із сутностями бази даних, як із звичайними об’єктами обраної мови програмування. Крім того, ORM значно спрощують та оптимізують побудову SQL запитів до БД завдяки спрощеному синтаксису.

Найбільш популярною та розвиненою ORM для обраної мови програмування JavaScript є фреймворк Sequelize.

Sequelize підтримує такі БД, як PostgreSQL, MySQL, MariaDB, SQLite та Microsoft SQL Server, дозволяє використовувати усі необхідні ORM операції, активно розвивається та підтримується спільнотою [5].

3.1.4 Вибір архітектури API

3.1.4.1 REST API

REST API – це архітектурний стиль, який відповідає набору обмежень при розробці веб-сервісів. Коли клієнт викликає REST API, сервер передає ресурси у стандартизованому представленні. Вони працюють, повертаючи інформацію про запитане джерело – і перекладається у формат, який можна інтерпретувати.

REST запит складається з Endpoint (кінцевої точки на сервері), методу HTTP, заголовка та тіла.

Кінцева точка містить URI (уніфікований ідентифікатор ресурсу), який допомагає ідентифікувати ресурс онлайн.

Метод HTTP описує тип запиту, який надсилається на сервер. Серед них:

– GET – читає представлення вказаного джерела;

– POST – створює новий ресурс вказаного джерела;

– PUT – оновлює/замінює кожен ресурс у колекції вказаного джерела;

– PATCH – змінює ресурс джерела;

– DELETE – видаляє ресурс джерела.

Під час роботи с даними REST API використовує методи HTTP для виконання CRUD операцій (Створення, читання, оновлення, видалення).

Заголовки надають клієнтам і серверам інформацію для таких цілей, як кешування, A/B-тестування, аутентифікація тощо.

Тіло містить інформацію, яку клієнт хоче надіслати на сервер, а саме корисне навантаження запиту.

3.1.4.2 GraphQL API

GraphQL – це мова запитів і керування даними на API, а також середовище для виконання запитів за наявними даними [6].

Серед основних переваг GraphQL можна виділити:

Отримання тільки необхідним даних – одне з основних обмежень REST API – це завантаження недостатніх або надлишкових даних. Це відбувається тому, що єдиний спосіб для завантаження даних клієнтом – звернення до вже описаних кінцевих точок API, які повертають фіксовані набори даних. Дуже важко розробити API таким чином, щоб він надавав клієнтам тільки необхідні дані, адже у різних ситуаціях використання кінцевої точки – вимоги до даних можуть відрізнятись.

GraphQL вирішує цю проблему, дозволяючи вам описати, які саме поля даних вам необхідні. Завдяки цьому можно перевикористовувати кінцеві точки, запитуючи саме ті дані, які необхідні клієнту у даний момент.

Типізованість – GraphQL використовує строго типізовану систему для визначення можливостей API. Усі типи, які доступні в API, записуються в схему з використанням мови визначення схем GraphQL (SDL).

Після цього у клієнтському коді можна описувати структуру запитів, використовуючи описані на API типи. Крім цього, можна використовувати популярні інструменти, такі як генератор коду GraphQL для автоматичної генерації коду запитів. Це значно прискорює розробку та запобігає помилкам.

Швидка розробка продукту – REST API описує кінцеві точки відповідно до ресурсів, доступних на API. Це зручно, оскільки дозволяє клієнту отримати всю необхідну інформацію для первого ресурсу, просто звернувшись до відповідної кінцевої точки.

Недоліком такого підходу є те, що він не допускає швидких ітерацій розробки. З кожною зміною, внесеною в користувацький інтерфейс – існує ймовірність, що необхідно буде змінювати і структуру ресурсів на API, наприклад для відображення необхідно буде більше або менше даних, ніж раніше.

GraphQL вирішує цю проблему завдяки своїй гнучкості: у клієнтському коді можна вносити зміни у запити без необхідності змінювати кінцеві точки на сервері.

Створення сервісу на базі GraphQL означає опис усіх типів, якими можуть обмінюватися API та клієнт, та створення спеціальних функцій-резолверів, які забезпечують передачу даних даного поля. Приклад функції-резолвера наведено на рисунку 3.2.

Для збереження усіх GraphQL типів використовується спеціальна схема. Крім типів, описаних розробником, схема може містити уже передбачені мовою типи, так як: Query, Mutation, Subscription. У цих типах описуються точки входу у GraphQL API. Тип Query дозволяє користувачам завантажувати дані з сервера, можна уявити його, як аналог GET запиту у REST API. Тип Mutation містить запити, які змінюють внутрішній стан серверу, записує дані,

є аналогом POST запису. Тип Subscription реалізовує двосторонній обмін даними, є аналогом WebSocket. Приклад опису типів у схемі наведено на рисунку 3.3.

```
function Query_me(request) {
  return request.auth.user;
}

function User_name(user) {
  return user.getName();
}
```

Рис. 3.2 – Приклад функцій-резолверів

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Рис. 3.3 – Приклад опису типів у схемі

При формуванні запиту користувач описує схему необхідних йому полей із відповідних типів, отримуючи у відповідь саме ті дані, що йому потрібні. Приклад користувацького запиту наведено на рисунку 3.4.

```
{
  hero {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

Рис. 3.4 – Приклад користувацького запиту та відповіді

Мова GraphQL має перелік базових примітивних типів:

– Int – 32-бітове ціле число зі знаком;

– Float – число з плаваючою комою;

– String – рядок тексту;

– Boolean – логічний тип, що може приймати два значення: True або False;

– ID – спеціальний тип, який представляє собою унікальний ідентифікатор, зазвичай використовується для повторного отримання об’єкта або як ключ для кешу, однак не є призначеним для читання людиною.

3.2 Вибір засобів розробки клієнтської частини

Невід’ємною частиною розробки сучасних клієнтських застосунків є JavaScript фреймворки, які надають розробникам перевірені та протестовані інструменти для створення масштабованих та інтерактивних веб-додатків. Серед основних переваг фреймворків можна виділити:

Компонентний підхід – фреймворки дозволяють розбивати користувацький інтерфейс на частинки, які можна перевикористовувати: компоненти. Це значно пришвидшує розробку та позитивно впливає на якість коду.

Декларативність – основна перевага фреймворків у розробці користувацьких інтерфейсів. Фреймворки дозволяють розробнику сфокусуватись на описі зовнішнього вигляду інтерфейсу у різних станах. Фреймворк же сам буде взаємодіяти з кодом сторінки, своєчасно змінюючи його, відповідно до актуального стану.

3.2.1 Порівняння Frontend фреймворків

Основними та найпопулярнішими фреймворками для розробки користувацьких інтерфейсів на даний момент є React, Angular та Vue. Розглянемо переваги та недоліки кожної із цих технологій:

React – бібліотека, розроблена компанією Facebook. Чудово підходить для реалізації сучасних односторінкових додатків різних розмірів і масштабів.

Переваги React:

– досить простий для освоєння, адже React використовує уже вбудовані в JavaScript конструкції як механізми створення інтерфейсів (умовні оператори, цикли, функції-колбеки);

– висока швидкість роботи завдяки технології віртуального DOM-дерева та численним механізмам оптимізації відображення змін на сторінці.

– чудова підтримка спільноти, велика кількість допоміжних інструментів для створення React-застосунків;

– React підтримує як функціольну, так і об’єктно-орієнтовану парадигми програмування, що дозволяє обрати більш комфортний підхід;

– код, написаний для веб версій застосунків, може бути перевикористаний для реалізації мобільних додатків на платформі React Native.

Недоліки React:

– React дозволяє різні підходи до проєктування застосунків, що може призвести до написання неоптимізованого та нерозширюваного коду недосвідченими розробниками;

– збереження логіки застосунку та користувацького інтерфейсу в одному компоненті може призводити низької читабельності коду та розростання компонентів до значних розмірів.

Angular – об’ємна середа для розробки інтерфейсів, яка реалізовує паттерн MVVM. Розроблена у компанії Google, чудово підходить для створення інтерактивних веб-додатків [7].

Переваги Angular:

– Angular використовується виключно у зв’язці з TypeScript, що забезпечує сурову типізацію та робить додатки значно надійнішими;

– Angular поставляється разом з пакетом Angular CLI, який містить велику кількість корисних засобів для генерації модулей та обслуговування Angular додатка;

– паттерн MVVM, який дозволяє розробникам працювати окремо над одним і тим же розділом застосунку, використовуючи один набір данних;

– структура і архітектура розроблені спеціально для підтримки великої масштабованості проєкту.

Недоліки Angular:

– Angular є досить складним для освоєння, адже містить велику кількість різноманітних структур, таких як: Injectable, Component, Pipe, Module тощо;

– Angular має досить повільну продуктивність, відносно інших фреймворків.

Vue – це JavaScript фреймворк, чудово підходить для створення високоадаптивних користувацьких інтерфейсів і складних односторінкових додатків [8].

Переваги Vue:

– адаптивніть – може бути здійснений швидкий перехід від інших фреймворків до Vue, завдяки подібній архітектурі та дизайну;

– чудова інтеграція - Vue можна використовувати як для створення односторінкових застосунків, так і для більш складних веб-інтерфейсів програм. Важливо, що невеликі інтерактивні елементи можна легко інтегрувати до існуючої інфраструктури без негативних наслідків;

– масштабування - Vue може допомогти в розробці досить великих шаблонів багаторазового використання, які можуть бути зроблені майже за той самий час, що і простіші;

– крихітний розмір - Vue має розмір близько 20 кБ, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти кращої продуктивності в порівнянні з іншими платформами.

Недоліки Vue:

– відносно невелика кількість ресурсів – Vue є менш популярним, ніж інші фреймворки, що призводить до значно меншої кількості допоміжних засобів та бібліотек. Так як спільнота фреймворку є не дуже великою, порівняно з конкурентами, то і обмін інформацією у середовищі знаходиться на початковому етапі;

– надмірна гнучкість – у Vue можуть виникати проблеми під час інтеграції у великі проєкти через відсутніть усталених практик і паттернів його використання.

3.2.2 Засоби керування станом застосунку

Для керування станом користувацької частини застосунку було обрано бібліотеку Apollo Client. Це JavaScript бібліотека, що дозволяє взаємодіяти з локальними та віддаленими даними за допомогою GraphQL [9].

Бібліотека дозволяє формувати GraphQL запити, зберігати отримані дані у кеш, який можна модифікувати, відповідно до потреб, автоматично оновлює інтерфейс користувача при оновленні стану застосунку.

Із переваг використання Apollo Client можна виділити:

– декларативний підхід у завантаженні даних – бібліотека забезпечує збереження завантажених даних у кеш застосунку, автоматично спричиняє оновлення користувацького застосунку при зміні даних;

– можливість звертатися до кешу застосунку з будь-якого рівня системи, дане сховище є «єдиним джерелом правди» застосунку, гарантує доступ до завжди актуального стану застосунку.

Крім того дана бібліотека має чудову інтеграцію з TypeScript та з найсучаснішими підходами розробки користувацьких застосунків з використанням популярних фреймворків.

3.2.3 Засоби стилізації застосунку

Мовою стилів у браузері є мова CSS, яка дозволяє у декларативному стилі описувати зовнішній вигляд веб-сторінок [10]. Однак можливості CSS значно розширюються при використані препроцесорів. Для стилізації користувацької частини застосунку було обрано препроцесор SCSS. SCSS – це потужна мова для написання стилів, яка компілюється у CSS. Препроцесор дозволяє використовувати додаткові функції, які пришвидшують та полегшують написання стилів, а саме: змінні, вкладеність, модулі, функції тощо [11].

За допомогою SCSS можна уникнути повторюваності коду та зробити його більш читабельним.

Переваги SCSS:

– відносно невеликий об’єм коду – SCSS допомагає уникати повторюваності, написаний код можна; перевикористовувати, що дозволяє зменшити його об’єм;

– швидка компіляція – SCSS швидко компілюється у CSS, не потребуючи при цьому значних ресурсів;

– велика спільнота розробників – SCSS має значну екосистему та велику кількість активних розробників, завдяки цьому препроцесор активно підтримується, існує велика кількість прикладів коду.

Висновки до розділу 3

В розділі було проаналізовано засоби розробки, які можуть бути використаними для створення застосунку.

Після детального ознайомлення з сучасними мовами програмування, їх перевагами та недоліками, було обрано мову JavaScript на платформі Node.js з використанням TypeScript для суворої типізації. Ця мова найбільше задовольняє потреби розробки серверної частини застосунку. У якості архітектури API було розглянуто REST і GraphQL. В результаті аналізу було обрано GraphQL, адже дана реалізація надає сурову типізації при передачі даних та дозволяє простим чином розширювати API. Для створення серверу було обрано Apollo Server. Apollo Server – бібліотека з відкритим кодом, яка дозволяє розгорнути GraphQL сервер, сумісний з будь-яким GraphQL клієнтом.

Для реалізації користувацької частини було проаналізовано популярні Frontend фреймворки. Після детального огляду переваг та недоліків цих технологій – було обрано React для реалізації користувацького інтерфейсу. React дозволить швидко розробити гнучкий інтерфейс та легко інтегрувати його на різні платформи.

4 РОЗРОБКА ДІАГРАМ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Діаграма варіантів використання мовою UML

Діаграма варіантів використання програмного застосунку представлена на кресленику ІА81.280БАК.003 Д1. Основними акторами є безпосередньо користувач застосунку та сервіс, який оброблює запити користувача. В першу чергу у користувача є такі варіанті, як реєстрація та авторизація, які дозволяють користувачу авторизуватися у системі та користуватися основною частиною її функціональності. Після цього для користувача стають доступними такі варіанти використання, як відправка текстових повідомлень, перегляд інформації про актуальне місцезнаходження контактів, трансляція власного місцезнаходження для своїх контактів. Для забезпечення функціонування основних варіантів використання користувача сервіс приймає місцезнаходження користувача та ретранслює його для усіх інших користувачів, які є контактами. Крім того сервіс виконує фунції збереження даних до бази даних.

Для забезпечення приватності та безпеки при користуванні застосунком користувач матиме можливість вимикати трансляцію свого актуального місцезнаходження за потреби. Використання застосунку користувачем завершується після того, як користувач виходить із свого акаунту.

4.2 Діаграма розгортання

На кресленику ІА81.280БАК.003 Д2 зображено діаграму розгортання розроблюваного застосунку. Основними вузлами системи є: клієнтська частина, сервер та сервери баз даних. Клієнтський застосунок складається з двох основних компонентів: безпосередньо користувацького інтерфейсу, який забезпечує взаємодію з користувачем, та набір серверів, які відповідають за відправку запитів на віддалений сервер за допомогою протоколів

HTTP/HTTPS або WS. Компонент клієнтського застосунку зображено на рисунку 4.1.

Роботу серверу забезпечують два основних компонента: шар функцій-контролерів, які займаються обробкою користувацьких запитів та шар сервісів, які забезпечують обмін даними із базами даних. Компонент серверного застосунку зображено на рисунку 4.2.

У системі використовується об’єктно-реляційна система керування базами даних PostgreSQL, основна ціль якої – збереження довгострокових записів, таких як дані про користувачів, повідомлення та чати.

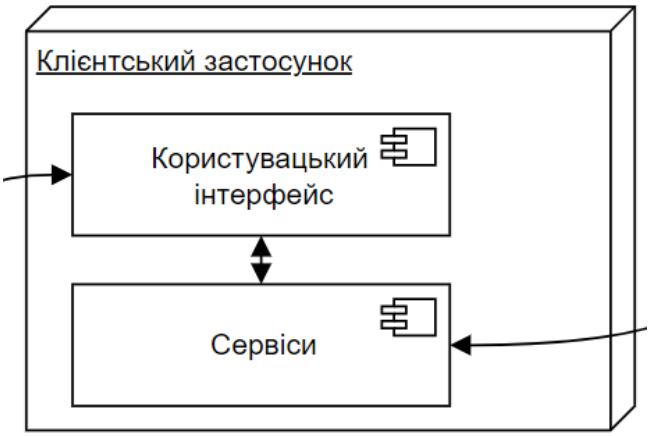

Рис. 4.1 – Компонент клієнтського застосунку

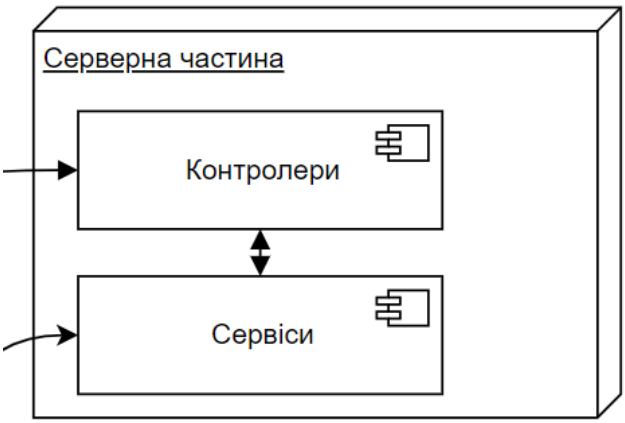

Рис. 4.2 – Компонент серверного застосунку

4.3 Діаграма послідовності

На кресленику ІА81.280БАК.003 Д3 зображено діаграму послідовності застосунку.

Основними об'єктами діаграми є користувач, який взаємодіє із системою, клієнтський застосунок та сервер.

В першу чергу користувачу необхідно авторизуватися у системі. Після введення даних для авторизації застосунок надсилає запит до серверної частини із користувацькими даними. На сервері відбувається перевірка коректності введених користувачем даних. У випадку існування користувача із надісланими обліковими даними – у відповідь на запит застосунку надсилаються дані користувача та токен сесії, який буде зберігатися у браузері користувача певний проміжок часу. Після цього користувач є авторизованим та може користуватися усіма можливостями застосунку. Після відкриття користувачем сторінки із геопозицією контактів із застосунку надсилається запит на підписку на геопозицію контактів користувача. Після встановлення з'єднання сервер починає надсилати користувачу геодані контактів із певною періодичністю, клієнтський застосунок же відображає місцезнаходження контактів на мапі. Крім цього встановлюється також з'єднання для трансляції серверу актуального місцезнаходження користувача для відображення його на мапі контактів.

Після відкриття користувачем сторінки із чатами – користувацький застосунок відправляє на сервер запит на завантаження актуальних чатів. У відповідь сервер надсилає перелік актуальних чатів. Аналогічним чином відбувається завантаження і повідомлень при відкритті чатів. На сторінці чатів також відправляється запит на підписку на оновлення у чатах. При створені нового повідомлення на сервері – дане повідомлення буде розіслане усім підписаним клієнтам даного чату.

Аналогічним чином користувач і сам може надіслати повідомлення, відправивши його у клієнтському застосунку. Після цього застосунок надсилає запит на сервер, на сервері нове повідомлення буде записане у БД та надіслано іншим користувачам чату.

4.4 Схема таблиць бази даних

На кресленику ІА81.280БАК.003 Д4 зображено схему бази даних даного застосунку. Для збереження інформації про користувачів використовується таблиця users. Дана таблиця зберігає username користувача, його пошту та хешований пароль для авторизації, мітки часу створення та останнього оновлення даних про користувача. Для збереження інших користувачів, як контактів, використовується таблиця user_contacts: поле owner_id посилається на користувача, який є власником контакту, поле contact_id посилається на іншого користувача, який буде збережений, як контакт. Загалом дана таблиця реалізовує відношення багато до багатьох для користувачів: кожен користувач може мати багато контактів, таким же чином, як і сам користувач може бути контактом для багатьох інших користувачів.

Так як користувач може бути членом багатьох різних чатів, а чати містять багато різних користувачів – то для цього відношення було створено таблицю chat_participants. Ця таблиця реалізовує відношення багато до багатьох та має поля chat_id, user_id, які посилаються на відповідні таблиці та часові мітки створення та видалення запису у таблиці (вступ та вихід користувача із чату). Реалізацію даного відношення відображено на рисунку 4.3.

Для збереження повідомлень у системі реалізовано таблицю messages. Ця таблиця має посилання на користувача, який надіслав дане повідомлення: sender_id, посилання на чат, у якому надіслане дане повідомлення, реалізоване

полем chat_id. Крім цього таблиця містить сам текст повідомлення та мітки часу створення, видалення та редагування повідомлення.

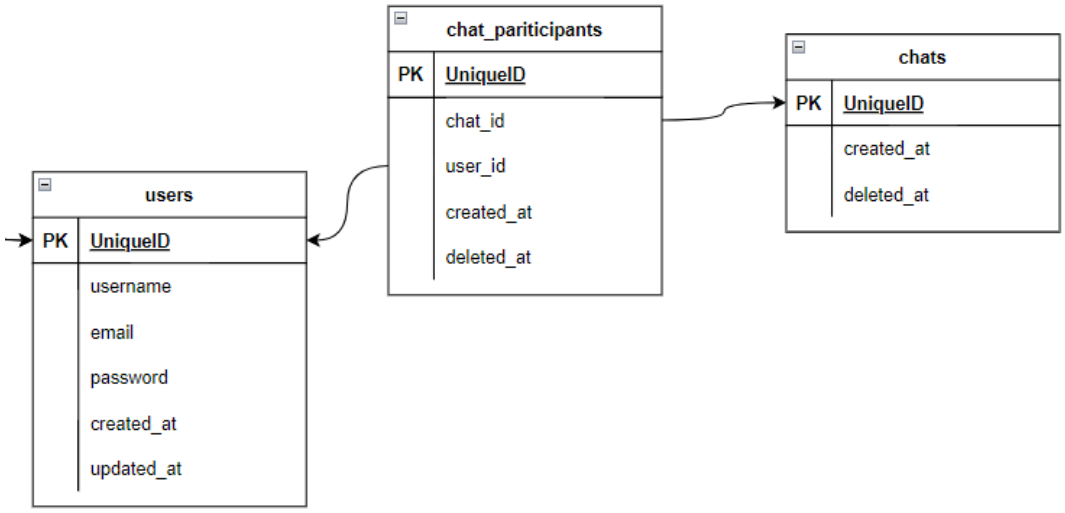

Рис. 4.3 – Відношення багато до багатьох між таблицями chats та users

5 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

5.1 Розробка серверного застосунку

Після закінчення аналізу предметної області, етапу проєктування та побудови необхідних діаграм – можна починати безпосередню розробку програмного застосунку.

В першу чергу потрібно створити необхідні таблиці у базі даних, описати їх поля, залежності, індекси тощо. Крім цього необхідно описати моделі об’єктно-реляційного відображення.

5.1.1 Створення міграцій бази даних

Для створення таблиць у базі даних було вирішено використати систему міграцій, яку надає Sequelize ORM. Міграції дозволяють керувати змінами у структурі бази даних у коді застосунку, надають можливість контролювати версії змін у структурі. Крім того міграції вирішують проблему перенесення баз даних у інші середовища: достатньо просто описати усі структурні зміни у вигляді коду. Після цього дані файли можуть бути зчитаними та форматованими у SQL запити.

Якщо розглядати більш детально – то міграції – це звичайні JavaScript файли, які експортують дві функції: up та down та описують, яким чином необхідно застосувати міграцію та відповідно відмінити її.

Для роботи із міграціями використовуємо утилітою, яка постачається разом із Sequelize: sequelize-cli. Це набір команд для роботи із міграціями та іншими структурами, які надає ORM. За допомогою команд можна створювати міграції, запускати їх, відміняти зміни.

В першу чергу було створено необхідні таблиці: users, chats, chat_participants, messages, user_contacts. Для кожної таблиці було описано поля, відповідно до побудованої діаграми таблиць. Також було додано

необхідні обмеження для таблиць (foreign key constaints) та індекси, що оптимізують виконання майбутніх запитів до бази даних. Міграцію, яка створює таблицю users зображено на рисунку 5.1, функцію, яка виконує протилежну дію і скасовує міграцію відображено на рисунку 5.2.

```
async up(queryInterface, Sequelize) {
  await queryInterface.createTable('users', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: Sequelize.INTEGER
    },
    username: {
      type: Sequelize.STRING,
      allowNull: false,
    },
    email: {
      type: Sequelize.STRING,
      allowNull: false,
    },
    password: {
      type: Sequelize.STRING,
      allowNull: false
    },
    createdAt: {
      allowNull: false,
      type: Sequelize.DATE,
      field: 'created_at',
    },
    updatedAt: {
      allowNull: false,
      type: Sequelize.DATE,
      field: 'updated_at',
    }
  });
},
```

Рис. 5.1 – Приклад міграції, яка створює таблицю users

```
async down(queryInterface) {
  await queryInterface.dropTable('users');
}
```

Рис. 5.2 – Приклад функції, яка відміняє міграцію

5.1.2 Створення класів моделей

У ході розробки для створення моделей було використано бібліотеку sequelize-typescript, яка дозволяє описувати моделі, як типізовані класи та використовувати для них декоратори [12].

Також використання класів дає змогу використати переваги наслідування: у застосунку було описано базовий клас ModelBase, який містить спільні для багатьох моделей поля. Після цього деякі моделі було наслідовано від ModelBase. Батьківську модель ModelBase зображено на рисунку 5.3.

```
export class ModelBase extends Model {
  @PrimaryKey
  @AutoIncrement
  @AllowNull(false)
  @Column
  id: number;

  @AllowNull(false)
  @CreatedAt
  @Default(DataType.NOW)
  @Column({
    field: 'created_at',
  })
  createdAt: Date;

  @AllowNull(false)
  @UpdatedAt
  @Default(DataType.NOW)
  @Column({
    field: 'updated_at',
  })
  updatedAt: Date;
}
```

Рис. 5.3 – Клас моделі ModelBase

Дочірню модель User зображено на рисунку 5.4.

```
class User extends ModelBase {
  @AllowNull(false)
  @Column
  username: string;

  @AllowNull(false)
  @Unique(true)
  @Column
  email: string;

  @AllowNull(false)
  @Column
  password: string;
}
```

Рис. 5.4 – Дочірня модель User

Важливою перевагою, яку надає Sequelize – можливість описувати асоціації між моделями. Загалом Sequelize надає змогу використовувати такі асоціації, як: HasOne, BelongsTo, HasMany, BelongsToMany. Комбінації цих асоціацій надають змогу реалізувати стандартні відношення між таблицями баз даних: один до одного, один до багатьох, багато до багатьох. Використання асоціацій наведено на рисунках 5.5 та 5.6.

```
class ChatParticipant extends Model {
  @BelongsTo(() => Chat)
  chat: Chat
```

Рис. 5.5 – Використання асоціації BelongsTo

```
class Chat extends Model {
  @HasMany(() => ChatParticipant)
  participants: ChatParticipant[]
```

Рис. 5.6 – Використання асоціації HasMany

5.1.3 Побудова GraphQL схеми

GraphQL схема – це опис даних, які може завантажувати користувач, використовуючи GraphQL API. Схема містить повний перелік типів та їх полей. Варто зазначити, що схема не має жодного відношення до типів, які описані у базі даних або у будь-якому іншому джерелі. Схема є абсолютно абстрагованим переліком типів, що надає розробникам гнучкості у побудові API.

GraphQL надає декілька примітивних типів, які існують за замовчуванням, крім того розробник може оголошувати власні типи, приклад яких наведено на рисунку 5.7.

```
type User {
  id: Int!
  username: String!
  email: String!
  authToken: String!
}

type Chat {
  id: Int!
  previewMessage: Message
  participants: [ChatParticipant!]!
}

type ChatParticipant {
  id: Int!
  username: String!
  userId: Int!
  chatId: Int!
}
```

Рис. 5.7 – Приклад оголошення типів у GraphQL схемі

Для опису операцій, які доступні для виконання на API, використовуються стандартні типи Query, Mutation, Subscription.

Тип Query описує операції читання даних з API.

Тип Mutation описує операції запису, оновлення, зміни даних на API.

Тип Subscription описує операції підписки на події. Користувач буде отримувати відповідні дані у реальному часі, коли API повідомить про них.

Операції різних типів зображено на рисунку 5.8.

```
type Query {
  me: String!
  login(fields: LoginFields!): User!
  chats: [Chat!]!
  chatMessages(chatId: Int!): [Message!]!
}

type Mutation {
  register(fields: RegisterUserFields!): User!
  createChat(userIds: [Int!]!): Chat!
  sendMessage(fields: SendMessageFields!): Message!
  updateMessage(fields: UpdateMessageFields!): Message!
  deleteMessage(messageId: Int!): Boolean!
}

type Subscription {
  messageCreated: Message!
  messageUpdated(chatId: Int!): Message!
  messageDeleted(chatId: Int!): Message!
}
```

Рис. 5.8 – Операції, доступні для виконання на API

Для опису вхідних параметрів запитів у GraphQL схемі може використовуватися спеціальний тип input, приклади використання даного типу зображено на рисунку 5.9.

```
input RegisterUserFields {
  username: String!
  email: String!
  password: String!
}

input LoginFields {
  email: String!
  password: String!
}
```

Рис. 5.9 – Приклад оголошених input типів

Окрім оголошених за замовчуванням примітивних типів можуть бути оголошені також власні примітивні типи. Під час розробки виникла необхідність використання типу Date, який і було додатково створено для використання у даній схемі, реалізація відображена на рисунку 5.10.

```
import { GraphQLScalarType, Kind } from 'graphql';

export const dateScalar = new GraphQLScalarType({
  name: 'Date',
  description: 'Date custom scalar type',
  serialize(value: any) {
    return value.getTime();
  },
  parseValue(value: any) {
    return new Date(value);
  },
  parseLiteral(ast) {
    if (ast.kind === Kind.INT) {
      return new Date(parseInt(ast.value, 10));
    }
    return null;
  },
});
```

Рис. 5.10 – Оголошення власного примітивного типу Date

5.1.4 Створення функцій-резолверів

Для того, щоб обробити користувацький запит та сформувати відповідь, GraphQL використовує спеціальні функції-резолвери. Такі функції описуються окремо для кожного поля типів схеми. Після того, як користувач робить запит, сервер формує відповідь, рекурсивно виконуючи усі функції-резолвери пов’язані із відповідними полями. Таке динамічне формування відповіді є одним із переваг GraphQL: користувач може запитувати усі необхідні йому поля (і тільки їх, ігноруючи надлишкові дані), а сервер здатен сформувати будь-яку структуру відповіді. Достатньо лише описати для цього відповідні функції-резолвери між різними графами даних, приклад такої функції наведено на рисунку 5.11.

```
const resolvers = {
  Query: {
    user(parent, args, context, info) {
      return users.find(user => user.id === args.id);
    }
  }
}
```

Рис. 5.11 – Приклад функції-резолвера

Під час роботи резолвери застосунку будуть відпрацьовувати за схожим шаблоном: отримувати запит, формувати відповідь, звертаючись до ORM, та повертати її. Приклад резолвера, який робить запит у БД і повертає усі чати, доступні користувачу наведено на рисунку 5.12.

```
export const chatsResolver = async (
  _: ApolloServer,
  __: Arguments,
  context: Context,
) => {
  const { authUser, models } = context;

  checkAuthUser(authUser);

  return models.Chat.findAll({
    include: {
      model: models.ChatParticipant,
      where: {
        userId: authUser?.id,
      },
    },
    raw: true,
    nest: true,
  });
};
```

Рис. 5.12 – Резолвер, який повертає усі чати, доступні користувачу

В першу чергу резолвер отримує дані про користувача та список моделей застосунку із спільного контексту застосунку. Після цього

відбувається перевірка на те, чи є користувач авторизованим. У випадку, коли користувач ще не авторизувався, він отримає помилку про необхідність авторизації та не зможе отримати відповідь на запит.

Після цього формується запит за чатами до бази даних. Застосунок поверне усі чати, в яких авторизований користувач є учасником чату. Завдяки Sequelize запит можна записати у зручному, читабельному та досить короткому вигляді. Після чого Sequelize сформує та виконає SQL запит, повернувши результат у вигляді JavaScript об’єктів. Даний запит представляє собою SELECT запит, який за певними фільтрами шукає та повертає дані із бази даних [13]. Запити наведені на рисунку 5.13.

Рис. 5.13 – SQL запити, сформовані Sequelize

5.1.5 Зв'язування функцій-резолверів на полей схеми

Для того, щоб GraphQL Server мав змогу виконати резолвери для відповідних полей – необхідно явним чином описати зв’язок між полем та його резолвером. Для цього описується спеціальний об’єкт resolvers, який згодом передається у конструктор сервера. Зв’язування полей схеми з резолверами виглядає наступним чином, представленим на рисунку 5.14.

```
export const resolvers = {
  Date: dateScalar,
  User: {
    authToken: authTokenResolver,
  },
  Chat: {
    participants: chatParticipantsResolver,
    previewMessage: chatPreviewMessageResolver,
  },
  ChatParticipant: {
    username: chatParticipantUsernameResolver,
  },
  Query: {
    me: meResolver,
    login: loginResolver,
    chats: chatsResolver,
    chatMessages: chatMessagesResolver,
  },
  Mutation: {
    register: registerResolver,
    createChat: createChatResolver,
    sendMessage: sendMessageResolver,
    updateMessage: updateMessageResolver,
    deleteMessage: deleteMessageResolver,
  },
```

Рис. 5.14 – Об’єкт резолверів

5.1.6 Створення API сервера

Для створення GraphQL API сервера використовуються пакети express та apollo-server-express. Під час виконання процесу запуску сервера також виконується аутентифікація інших сервісів, які виконує застосунок, першочергово – створюється сервіс Sequelize, який встановлює зв’язок із базою даних. Облікові дані, які використовує застосунок для доступу до бази даних та інших сервісів, зберігаються у файлі .env, який може бути окремим для усіх середовищ виконання коду застосунку та є чудовим рішенням для забезпечення безпеки облікових даних.

Після цього ключові об'єкти, які надають змогу звертатися до сервісів, передаються у контекст сервера. Це надає доступ до сервісів із будь-якого резолвера на API. Код, який створює сервіс, наведено на рисунку 5.15.

```
const server = new ApolloServer({
  schema,
  csrfPrevention: true,
  context: async ({ req, res }) => {
    const authToken = req.headers.authorization || '';
    const authUser = await getAuthUser(authToken, JWT_SECRET_KEY, models);

    return {
      models,
      sequelize,
      req,
      res,
      secret: JWT_SECRET_KEY,
      authUser,
      pubsub,
    };
  },
```

Рис. 5.15 – Створення сервера та формування контексту застосунку

5.1.7 Реалізація підписок

Підписки – довготривалі операції зчитування, які оновлюють свій результат, коли на сервері відбувається відповідна подія. Після цього сервер надсилає клієнту нові дані. Підписки є чудовим варіантом для отримання нових повідомлень з серверу.

Для реалізації підписок серверу не підходить протокол HTTP/HTTPS, тому необхідно використати протокол WebSocket. Для цього на API було створено окремий сервер зі своїм контекстом. Код створення WebSocket сервера наведено на рисунку 5.16.

Для сповіщення GraphQL про необхідність відправки оновлених даних підпискою було використано паттерн EventEmitter, реалізований бібліотекою PubSub. Даний паттерн дозволяє з будь-якого місця у програмі сповістити про подію, яка відбулася.

```
const wsServer = new WebSocketServer({
  server: httpServer,
  path: '/graphql',
});

const serverCleanup = useServer({
  schema,
  context: async (ctx) => {
    const authToken = String(ctx.connectionParams?.authentication);
    const authUser = await getAuthUser(authToken, JWT_SECRET_KEY, models);

    return {
      pubsub,
      models,
      authUser,
    };
  },
}, wsServer);
```

Рис. 5.16 – Створення WebSocket сервера для реалізації підписок

Зазвичай відправляти клієнту необхідно лише ті дані, які відповідають певному критерію, наприклад, відправляти користувачу нові повідомлення лише з тих чатів, учасником яких він є. Або транслювати користувачу місцезнаходження тільки тих користувачів, які є його контактами.

Для цього у підписках реалізовуються фільтри. Фільтри представляють собою функції, які виконують перевірку і повинні повернути true або false. Якщо фільтр повертає негативне значення – то передача даних підпискою не буде виконана. Приклад підписки з фільтром наведено на рисунку 5.17.

```
export const messageDeletedResolver = {
  subscribe: withFilter(
    (_, __, { pubsub }: Context) => pubsub.asyncIterator(Subscription.MessageDeleted),
    async (payload: Payload, args: Arguments) => {
      const { chatId } = args;
      const  { messageDeleted } = payload;

      return chatId === messageDeleted.chatId;
    },
  ),
};
```

Рис. 5.17 – Підписка з фільтром видалених повідомлень

5.1.8 Реалізація авторизації користувача

Для реалізації авторизації користувача використовується JWT. Даний токен створюється під час реєстрації або авторизації користувача у застосунку. Після цього він відправляється у клієнтський застосунок. Клієнтський застосунок може зберігати свій токен у local storage браузера. При формуванні запиту до сервера – клієнт може додавати даний токен у заголовки запита, таким чином сервер знатиме про користувача, який робить відповідний запит. Приклади використання JWT токена на серверній частині для ідентифікації користувача зображено на рисунках 5.18 та 5.19 відповідно.

```
const authLink = setContext((_, { headers }) => {
  // get the authentication token from local storage if it exists
  const token = localStorage.getItem('token');
  // return the headers to the context so httpLink can read them
  return {
    headers: {
      ...headers,
      authorization: token ? `Bearer ${token}` : "",
    }
  }
});
```

Рис. 5.18 – Приклад додавання заголовку із JWT токеном до запиту

```
const authToken = req.headers.authorization || '';
const authUser = await getAuthUser(authToken, JWT_SECRET_KEY, models);
```

Рис. 5.19 – Використання JWT токена для ідентифікації користувача

5.2 Розробка клієнтського застосунку

5.2.1 Налагодження засобів розробки

Для створення початкового шаблону клієнтського застосунку було використано пакет create-react-app. Пакет значним чином спрощує процес встановлення усіх необхідних залежностей проєкту, налагодження конфігураційних файлів, додає скрипти, які оптимізують процес розробки.

Крім цього створює початковий шаблон проєкт [14]. Структура файлів, створених за допомогою create-react-app, зображена на рисунку 5.20.

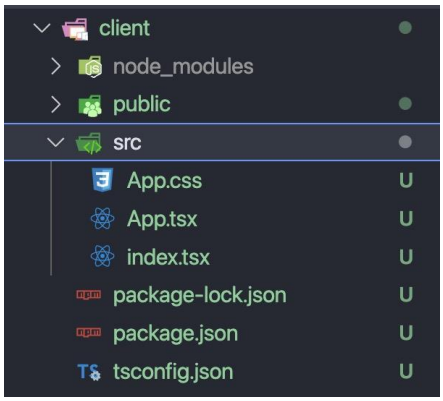

Рис. 5.20 – Початковий шаблон застосунку

Для використання переваг пакету Apollo Client, який був обраний засобом керування станом застосунку, необхідно ініціалізувати клієнт. Код, ініціалізуючий Apollo Client у застосунку зображено на рисунку 5.21.

```
const apolloClient = new ApolloClient({
  uri: 'https://localhost:4000/graphql',
  cache: new InMemoryCache(),
});
```

Рис. 5.21 – Ініціалізація клієнту Apollo

5.2.2 Створення клієнтського інтерфейсу

Під час розробки клієнтських застосунків з використанням бібліотеки React використовується компонентний підхід. Код розбивається на невеликі логічні частини, які є елементами інтерфейсу, та можуть бути перевикористаними у ході розробки. Дані компоненти вкладаються у батьківські компоненти, таким чином формуючи дерево тегів, з якого React

формує користувацький інтерфейс, вносячи зміни у DOM-дерево браузера. Приклад React-розмітки наведено на рисунку 5.22.

```
<ul
  onScroll={onScroll}
  className={styles.messageList}
>
  {messages.map((message) => (
    <li key={message.id} className={styles.message}>
      <Message
        message={message}
        isEditing={message.id === messageToUpdate?.id}
        parentMessage={parentMessage}
      />
    </li>
  ))}

  <div ref={listEndRef} />
</ul>
```

Рис. 5.22 – Розмітка, яка відображає повідомлення у чаті

Значною перевагою React також є його реактивність, концепт, який дозволяє писати код у декларативному стилі. Таким чином розробка спрощується, адже розробнику не потрібно детально описувати те, яким чином зміни будуть відображені на інтерфейсі, достатньо лише описати що і як зміниться, бібліотека самостійно вносить усі зміни у DOM-дерево браузера [15].

5.2.3 Виконання запитів до API

При виконанні запитів до API використовуються переваги, які надає вибір GraphQL для архітектури API. Запити створюються з використанням систаксису GraphQL, описуються усі поля об’єктів, які необхідно отримати. Так як серверна частина має відвовідні функції-резолвери, то в одному запиті можна завантажувати одразу декілька типів даних. На рисунку 5.23 продемонстровано створення запиту та опис відповідних полей. Клієнт у

даному випатку запитує на масив повідомлень у чаті, передаючи унікальний ідентифікатор чату, як параметр. Крім того, запит містить поле sender, яке поверне також відправника даного повідомлення, а саме поле fullname, описане у запиті.

```
export const CHAT_MESSAGES_QUERY = gql`
  query chatMessages($chatId: Int!) {
    chatMessages(chatId: $chatId) {
      id
      senderId
      sender {
        fullname
      }
      text
      updatedAt
      createdAt
    }
  }
`;
```

Рис. 5.23 – GraphQL запит

Після виконання запиту бібліотека Apollo Client поверне об’єкт, що містить усю необхідно інформацію про даний запит. А саме: статус завантаження, наявність помилок та безпосередньо отримані дані, які можна використовувати для побудови інтерфейсу.

5.2.4 Використання JWT токена для аутентифікації

Для аутентифікації користувача сервером необхідно додавати у хедери усіх запитів JWT токен. Даний токен отримується при логіні або авторизації із даними, які передаються з сервера. Для збереження сесії користувача у браузері використовується local storage браузера. Взаємодія React та local storage браузера реалізована за допомогою хука useLocalStorage [16]. Після

цього при відправці запиту клієнт додаватиме даний токен у хедер authentication.

Висновки до розділу 5

В розділі було розроблено програмний застосунок, який складається з двох частин: серверної та клієнтської. Під час розробки було використано сучасні підходи та технології.

Серверний застосунок працює на базі платформи Node.js. Основним фактором під час розробки було забезпечення легкого розширення фунціоналу та підтримки.

Для розробки серверної частини було використано провідну бібліотеку React, що забезпечує оптимальну роботу браузерного застосунку. Крім того використання даної бібліотеки полегшує можливість перенесення застосунку на іншу платформи, дозволяючи перевикористовувати готовий програмний код.

ВИСНОВКИ

У результаті виконання індивідуального проєкту було досліджено та реалізавно систему систему обміну текстовими повідомленнями з можливістю GPS-трекінгу контактів.

Для вирішення поставленого завдання було розглянуто існуючі аналоги, проаналізовано переваги та недоліки систем, виділено загальні особливості, які доречно використати при розробці застосунку. Було розглянуто технологію GPS, яка дозволяє відстежувати власне місцезнаходження для трансляції її контактам, та протокол WebSocket, який дозволяє передавати дані між клієнтом та сервером у реальному часі.

Було проаналізовано перелік програмних засобів, необхідних для розробки, та обрано найбільш підходящі.

Під час виконання технічного проєктування було створено необхідні діаграми, а саме: діаграми варіантів використання, послідовностей, розгортання та схема таблиць бази даних.

Було розроблено програмний застосунок, який складається з двох частин: серверної, яку побудовано з використанням архітектури GraphQL API та клієнтської. Програмний інтерфейс застосунку містить перелік методів для обміну повідомленнями, пошуку нових контактів, підписки на трансляції геопозиції. Для збереження даних використано реляційну базу даних PostgreSQL. Для реалізації браузерного клієнтського застосунку було використано бібліотеку React.

Загалом поставлена задача та мета були досягнуті в ході виконання дипломної роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia GPS. URL: https://uk.wikipedia.org/wiki/GPS
2. JavaScript Info. URL: https://uk.javascript.info/websocket
3. TypeScript Documentation. URL: www.typescriptlang.org/docs/handbook
4. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/
5. W3School SQL Tutorial. URL: https://www.w3schools.com/sql/
6. GraphQL Docs. URL: https://graphql.org/learn/
7. Angular Docs. URL: https://angular.io/docs
8. Vue Docs. URL: vuejs.org/guide
9. Apollo Docs. URL: https://www.apollographql.com/docs
10. Create-react-app Guideline. URL: https://create-react-app.dev/docs/getting-started
11. MDN WebDocs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS
12. React Документація. URL: https://uk.reactjs.org
13. Sequelize-Typescript Github Homepage. URL: https://github.com/RobinBuschmann/sequelize-typescript#readme
14. UseHooks-ts Docs. URL: https://usehooks-ts.com/react-hook/use-local-storage
15. SCSS Docs. URL: https://sass-lang.com
16. UseHooks-ts Docs. URL: https://usehooks-ts.com/react-hook/use-local-storage