Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Тітова Юрія Ярославовича*
(ПІБ)

академічної групи *122-20-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка програмного забезпечення соціальної мережі на базі платформи AWS Cloud*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*доц. Ширін А.Л.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-1* *Тітова Юрія Ярославовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення соціальної мережі на базі платформи AWS Cloud*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *доц. Ширін А.Л.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Тітов Ю.Я.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *24.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 81 с., 31 рис., 8 табл., 3 дод., 28 джерел.

Об'єкт розробки: соціальна мережа, реалізована на основі хмарних технологій AWS Cloud.

Мета кваліфікаційної роботи: створення соціальної мережі з можливістю реєстрації та редагування свого профілю (додавання інтересів, тегів, картинки), знаходження користувачів за ім’ям, сортування за тегами, віком, місцем перебування та можливість почати з ними розмову.

У вступі розглядається аналіз та мета кваліфікаційної роботи, галузь її застосування, уточнюється постановка завдання та умови її успішного завершення.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні та вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні вебдодатку, що надає можливість дистанційного спілкування між зареєстрованими користувачами шляхом текстового чату, зберігання даних чатів, файлів поширених у чатах, ведення бази даних.

Актуальність вебдодатку визначається великим попитом на подібні розробки, що полегшують процес спілкування між людьми.

Список ключових слів: СОЦІАЛЬНА МЕРЕЖА, AWS Cloud, АЛГОРИТМ, ПРОЕКТУВАННЯ, ДОДАТОК, БРАУЗЕР.

ABSTRACT

Explanatory note: 81 p., 31 figures, 8 tables, 3 app., 28 sources.

Object of development: social network implemented on the basis of cloud technologies AWS Cloud.

The purpose of the qualification work: creating a social network with the possibility of registering your profile, adding interests, tags, pictures, finding users by name, sorting by tags, age, location and the ability to start a conversation with them.

The introduction examines the analysis and purpose of the qualification work, the field of its application, specifies the setting of the task and the conditions for its successful completion.

In the first section, the subject area is analyzed, the relevance of the task and the purpose of the development is determined, the task statement is formulated, and the requirements for software implementation, technologies and software tools are specified.

In the second section, available solutions are analyzed, platforms for development are chosen, program design and development is performed, program operation, algorithm and structure of its functioning are described, as well as program calling and loading, input and output data are determined, the composition of technical means parameters is characterized.

In the economic section, the labor intensity of the developed information system is determined, the cost of work on creating the program is calculated, and the time for its creation is calculated.

The practical value is in the creation of a web application that provides the possibility of remote communication between registered users through text chat, storage of chat data, files shared in chats, and database management.

The relevance of the web application is determined by the great demand for similar developments that facilitate the process of communication between people.

List of keywords: SOCIAL NETWORK, AWS CLOUD, ALGORITHM, DESIGN, APP, BROWSER.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

SQL – Structured Query Language.

UML – Unified Modeling Language.

AWS – Amazon Web Services.

S3 – Simple Storage Service.

API – Application Programming Interface.

HTML - Hyper Text Markup Language.

CSS - Cascading Style Sheets.

JS – JavaScript.

CI/CD – Continuous Integration / Continuous Deployment.

MSP – Managed Services Provider.

JVM – віртуальна машина Java.

CLI – Command Line Interface.

ЗМІСТ

ВСТУП

Розроблена інформаційна система призначена для формування соціальної мережі на основі платформи AWS Cloud.

Соціальна мережа - це соціальна структура, утворена індивідами або організаціями. Вона відображає різноманітні зв'язки між ними через різні соціальні взаємовідносини, починаючи з випадкових знайомств і закінчуючи тісними родинними зв'язками.

Соціальна мережа може використовуватися будь-якою компанією для підтримки приватної комунікації між співробітниками, тому розробка такої системи на платформі AWS Cloud, є актуальною та має широке практичне значення.

Метою даної роботи є розробка соціальної мережі.

Умовами виконання поставленого завдання є:

– унікальне посилання на вебдодаток;

– можливість користувачам з та без аккаунтів проглядати список вже зареєстрованих користувачів;

– можливість зареєстрованим користувачам почати текстову бесіду з іншими зареєстрованими користувачами зі списку;

– можливість приховувати та розповсюджувати свою приватну інформацію;

– можливість фільтрувати, сортувати та шукати список зареєстрованих користувачів.

Дана соціальна мережа розроблена за допомогою хмарного середовища AWS Cloud і відрізняється від інших можливістю прямого спілкування з іншими користувачами із завжди наявного списку.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Одними з застосунків, які набули великої популярності за останні десятиліття, є соціальні мережі та інші платформи для спілкування на відстані (особливо закордоном). Люди - істоти соціальні, але не всі мають необхідних навичок для живого спілкування, тут і стають в пригоді соціальні мережі, що скорочують відстань між співрозмовниками.

Соціальна мережа – це соціальна структура, утворена індивідами або організаціями, але в нашому випадку це інтернет-програма, яка допомагає друзям, бізнес-партнерам або іншим особам спілкуватись та встановлювати зв'язки між собою, використовуючи набір інструментів. Яскравим прикладом соціальних мереж є Facebook, Telegram, YouTube та ін.

Легальне визначення терміну «соціальна мережа», як і терміну «акаунт», в українському законодавстві відсутнє. Водночас акаунт у соцмережі можна розцінювати як вебсторінку, визначення якої міститься в Законі України «Про авторське право та суміжні права». Відповідно до вказаного Закону, вебсторінка – це складова вебсайту, що може містити дані, цифрову інформацію, інші об’єкти авторського та/або суміжних прав тощо.

Він може мати неабияку цінність, отже, актуальною темою є належне законодавче регулювання права власності на такий віртуальний актив. Акаунти в соціальних мережах продаються, передаються та навіть здаються в оренду на тіньовому ринку. Не регламентовані також можливість залишення вебсторінки у спадок іншій особі та право сумісної власності на акаунт, наприклад, подружжя (є блогери, які спільно ведуть сторінки у соціальних мережах, отримуючи прибуток). Тож акаунт може мати цілком матеріальну цінність, яка може підлягати оцінці [2].

Розглянувши декілька прикладів популярних соціальних мереж зробив висновки про дизайн і структуру вебдодатку у розробці.

Вже існуюча соціальна мережа Telegram слугувала натхненням при розробці дизайну та загальної структури даного додатку. При цьому деякі функціональні елементи дизайну інтерфейсу були упущені чи видозмінені за відсутності потреби у них. Розроблений додаток від самого початку надає доступ до списку контактів усіх зареєстрованих на платформі користувачів з можливістю фільтрування за ім’ям, віком та місцем проживання. Приклад інтерфейсу Telegram наведено на рис.1.1.

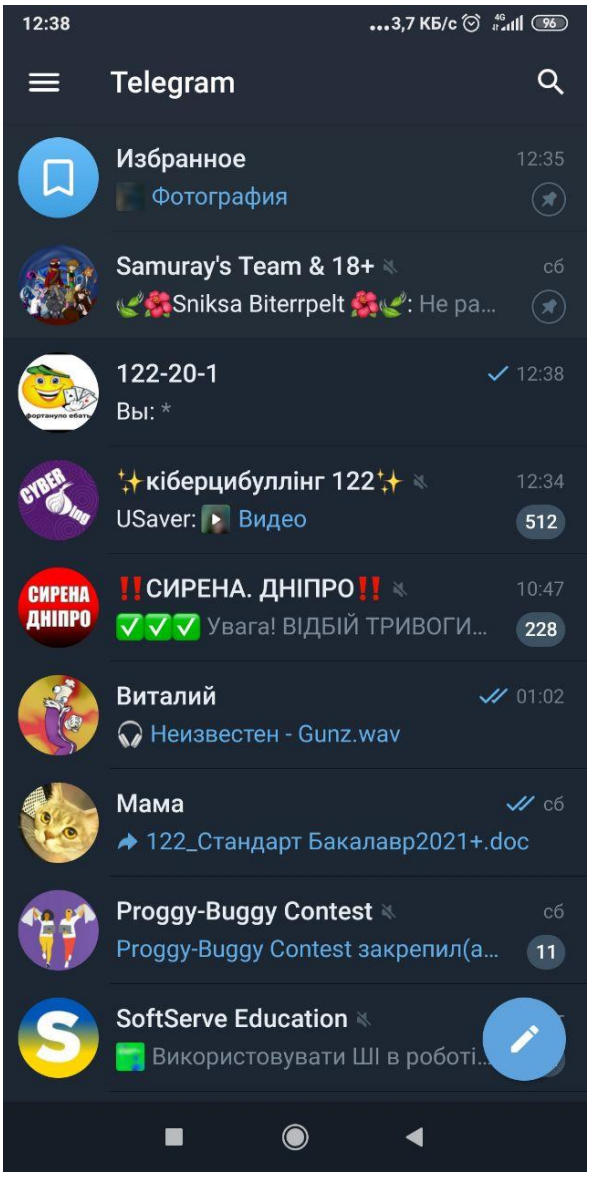


Рис. 1.1. Сторінка Telegram на телефоні

На відміну від Telegram, розроблений додаток побудовано на основі хмарних технологій AWS, що полегшує створення та обслуговування структури додатку, адже сервіси AWS Cloud самі слідкують за станом технічної частини, відповідає за безперебійність та мережу, забеспечує 99,9% стабільності системи. Плата за користування базується на комбінації використання апаратних засобів/ОС/програмного забезпечення/мережевих функцій, вибраних користувачем, а також вимог до доступності, надлишковості, безпеки та додаткових параметрів, тому розробник не переплачує за продукт у періоди низької активності додатку.

Видимих проблем із додатком зафіксовано не було.

1.2. Призначення розробки та галузь застосування

В якості об'єкту розробки розглядається кросплатформний додаток, на якому люди можуть створити профіль, знайти людей за інтересами, віком або місцем проживання та розпочати приватний чат.

Даний додаток повинен містити в собі можливість створювати особистий обліковий запис, додавати контактні дані, створювати нові теги інтересів, які будуть перебувати в публічному доступі, надавати можливість створити чат з іншим користувачем.

Адмінпанеллю служить AWS Console, де команда людей із індивідуальним набором доступу слідкує за навантаженням сервісів та розподілом бюджету.

Платформа розрахована на усіх бажаючих найти співрозмовника за інтересами. Це дозволить робити знайомства більш доступно та сучасно.

Оскільки цей сервіс також розроблявся для студентів нашого університету НТУ "Дніпровська Політехніка", можна надати додаткову можливість використання платформи студентами нашого університету.

1.3. Підстави для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

− освітня програма 122 Комп'ютерні технології;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № _469-с_ від _23-05-2024 р._;

− завдання на кваліфікаційну роботу на тему «Розробка інформаційної платформи для спілкування на великій відстані на AWS Cloud».

1.4. Постановка завдання

Завдання: розробити вебплатформу, реалізовану на основі хмарних технологій AWS Cloud з використанням стеку технологій Java, Python та .NET, HTML & CSS & JS.[4][5]

Функціонал: у табл. 1.1. наведено можливості користувачів (об'єктів) при роботі з вебплатформою.

Таблиця 1.1.

Можливості користувачів (об'єктів) при роботі з вебплатформою

Об'єкт	Можливості
Користувач	реєстрація вхід до системи заповнення профілю створення тегів перегляд контактів створення приватного чату з іншим користувачем
Адміністратор	реєстрація вхід до системи заповнення профілю створення тегів перегляд контактів створення приватного чату з іншим користувачем
Гість	реєстрація перегляд контактів

Також передбачається, що платформа налічуватиме певні об’єкти з певними властивостями (наведено у табл. 1.2.):

Таблиця 1.2.

Об’єкти платформи та їх властивості

Об’єкт	Властивості
Користувач (користувач, адміністратор)	електронна пошта номер телефону ім'я прізвище унікальний код зображення вік людини місто проживання посилання на сторінки в інших соцмережах
Тег	назва посилання
Чат	назва учасники повідомлення

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Вебплатформа “MEGATHON” повинна бути реалізована за допомогою хмарного середовища AWS Cloud у вигляді вебсайту, перейти до якого можна за допомогою браузера (Chrome. Mozilla, Microsoft Edge, Opera, Internet Explorer та ін.).

Для написання платформи передбачається використання наступних технологій: Java, HTML, CSS, JS, Terraform, AWS Lambda, AWS S3, AWS DynamoDB, AWS API Gateway, AWS Cognito.[1]

Вся інформація, яка відображається користувачам і адміністратору, а також обмінюються додатками всередині сервера, повинна знаходитися в базі даних AWS DynamoDB на самому сервері.

Платформа має давати можливість незареєстрованим користувачам переглядати список зареєстрованих контактів, зареєстрованим - взаємодіяти з зареєстрованими контактами.

1.5.2. Вимоги до інформаційної безпеки

Інформаційна безпека вебплатформи є критично важливим аспектом, який включає заходи щодо захисту інформації, запобігання несанкціонованому доступу та збереження конфіденційності даних.

Платформа “MEGATHON” повинна мати такі моменти безпеки:

Аутентифікація та авторизація – вебплатформа повинна мати механізми автентифікації користувачів, щоб переконатися, що лише зареєстровані користувачі чи адміністратори мають право спілкуватися з іншими користувачами.

Захист від злому – платформа має бути захищена від злому. Це може включати застосування оновлень безпеки, захист від відомих вразливостей, використання сильних паролів, захист від атак переповнення буфера та інші методи.

Шифрування – вся інформація, що передається між вебплатформою та її користувачами, має бути зашифрована для захисту від перехоплення та прослуховування. Платформа має використовувати такий протокол як SSL.

Обробка вхідних даних – платформа повинна включати фільтрацію і валідацію вхідних даних, щоб запобігти атакам типу SQL-ін'єкції, міжсайтового скриптингу (XSS) і CSRF-атак.

Резервне копіювання та відновлення – важливо мати механізми резервного копіювання даних, щоб запобігти втраті інформації у разі збоїв або атак.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для доступу до платформи підійде будь-який пристрій: ноутбук, планшет, комп'ютер, телефон, хоча розроблена вона, насамперед, для телефонів. Важливі такі характеристики браузера (наприклад візьмемо популярний - Chrome і телефон з Android):

− версія браузера Chrome 121.0.6167.185 або вище;

− підтримка 32- або 64-розрядного процесора та операційної системи;

− операційна система Android 14 чи вище;

− наявність маніпулятора-сенсорного екрану;

− оперативна пам'ять об'ємом 2 ГБ або вище

− процесор із частотою 1 ГГц або вище;

− жорсткий диск HDD чи SSD об'ємом 10 ГБ або вище;

1.5.4. Вимоги до інформаційної та програмної сумісності

Для ефективної роботи програми на різних пристроях необхідно, щоб ноутбук, планшет, комп'ютер або телефон відповідали таким вимогам:

- операційна система Windows (7+) / Linux / MacOs/Android ;
- веббраузер Google Chrome / Opera / Safari/ Firefox / Microsoft Edge.

Застосунок реалізовано на тенологічному стекі мов програмування JavaScript та TypeScript, мові програмування Java, Python, .NET, HTML & CSS, Terraform та хмарного середовища AWS Cloud, тому для працездатності додатку треба, щоб на пристрої була встановлено JVM.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

У ході виконання кваліфікаційної роботи було розроблено вебплатформу, яка дає змогу людям знайти співрозмовників по інтересам на відстані.

Призначення платформи:

- надати можливість просто, швидко і надійно знайти співрозмовника за інтересами на платформі;
- вести облік зареєстрованих людей;
- надсилати повідомлення особисто іншим користувачам.

Вебплатформа реалізує наступні функції.

1. Загальний функціонал (як для зареєстрованих, так і для незареєстрованих користувачів):
    - Реєстрація;
    - перегляд контактів.
2. Функціонал зареєстрованого користувача:
    - вхід до системи;
    - заповнення/змінення профілю;
    - створення тегів;
    - створення приватного чату з іншим зареєстрованим користувачем;
    - спілкування у створеному чаті з іншим зареєстрованим користувачем.
3. Функціонал зареєстрованого користувача - адміністратор:
    - вхід до системи;
    - заповнення/змінення профілю;
    - створення тегів;
    - створення приватного чату з іншим зареєстрованим користувачем.

- спілкування у створеному чаті з іншим зареєстрованим користувачем.

2.2. Опис застосованих математичних методів

Розроблена платформа надає можливості створювати особистий обліковий запис, додавати контактні дані, створювати нові теги інтересів, які будуть перебувати у публічному доступі, надавати можливість спілкуватися через вбудований месенджер.

У зв'язку з цим, функціональність платформи не включає рішення математичних завдань або використання математичних формул.

2.3. Опис використаних технологій та мов програмування

Розроблену веб платформу було розгорнуто у хмарному середовищі AWS Cloud з використанням Amazon S3, Amazon DynamoDB, Amazon API Gateway, AWS Lambda, Amazon Cognito, підхід – Serverless, OpenAPI, Managed services та Angular.

Нижче наведена діаграма (рис. 2.1), яка ілюструє архітектуру роботи проекту.

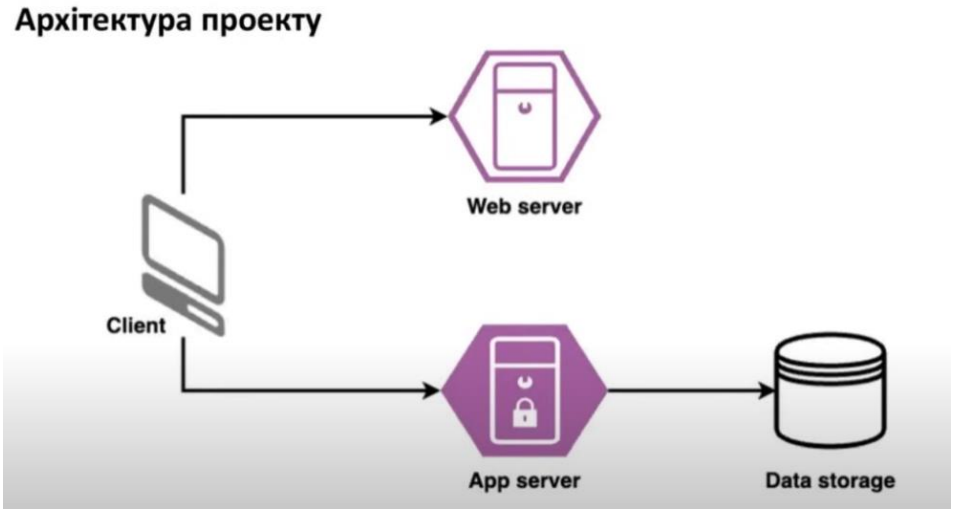


Рис. 2.1. Архітектура проекту на основі хмарних технологій

Переваги AWS Cloud полягають в масштабованості всієї системи роботи з вебсервісами, доступності та надійності, економії на витратах, спрощенні управління та оновлення та безпеці даних. Воно дозволяє вибрати операційну систему, мову програмування, платформу, бази даних та інші сервіси необхідні для безпечної та доступної підтримки повноцінного вебсайту.

Serverless метод - це спосіб надання серверних послуг з урахуванням фактичного використання сервісів. Serverless провайдер надає змогу користувачам писати та розгортати код без потреби налаштовувати базову інфраструктуру. У розробленій веб платформі цього ефекту допомагає досягти Amazon S3.

Переваги Serverless методу:

1. Система автоматично масштабується залежно від навантаження. Це дозволяє програмам автоматично адаптуватися до змінного обсягу роботи без необхідності вручну налаштовувати інфраструктуру.

2. Ви платите лише за те, що ви використовуєте. Оскільки сервери запускаються лише при необхідності виконання певних функцій, не потрібна оплата за простий або ресурси, що не використовуються.

3. Розробники можуть зосередитися на написанні коду, не турбуючись про налаштування та керування інфраструктурою. Це може значно спростити процес розробки та прискорити вихід нових продуктів ринку.

4. Розгортання функцій може бути дуже швидким, оскільки не потрібно докладати зусиль для налаштування серверів або іншої інфраструктури. Можна працювати зі своїм кодом і відразу бачити результати. Serverless дозволяє швидко розгортати та оновлювати код без простоїв або перезавантаження програми.

5. Багато сервісів Serverless мають вбудовану високу доступність, оскільки вони автоматично керують розміщенням вашого коду на різних серверах та регіонах.

6. Serverless-платформи забезпечують широкі можливості інтеграції з іншими хмарними сервісами, такими як бази даних, черги повідомлень і багато іншого.

Managed services метод передбачає передачу набору послуг з обслуговування корпоративної IT-інфраструктури спеціалізованій компанії – MSP (Managed Services Provider). MSP беруть на себе відповідальність за обслуговування, управління, моніторинг та підтримку IT-інфраструктури клієнтів. AWS Cloud, на якому побудована веб платформа, є одним із найуспішніших MSP.

Переваги Managed services методу:

1. Managed services включають в себе адміністрування та управління інфраструктурою, так що розробнику не потрібно стежити за цим аспектом. Це дозволяє зосередитися на розробці програмного забезпечення та вдосконаленні продукту.

2. Багато Managed services пропонують вбудовану високу доступність і масштабованість, що дозволяє підтримувати стабільну роботу створеного продукту без перебоїв у роботі.

3. Managed services зазвичай автоматизують багато рутинних завдань, таких як резервне копіювання, моніторинг та відновлення після збоїв. Це дозволяє ефективно використовувати час розробників і уникати помилок, пов'язаних з людським фактором.

4. Більшість Managed services мають простий і швидкий процес розгортання, що дозволяє швидко створювати та розгортати нові екземпляри програмного забезпечення або змінювати конфігурацію.

5. Багато Managed services мають вбудовані засоби захисту, такі як шифрування даних, мережеві заходи безпеки та автентифікація, що дозволяє зменшити ризик витоку даних або злому системи.

Angular - метод генерації HTML-шаблонів на сервері, що дає кілька переваг. По-перше, він допомагає підвищити рейтинг програми у пошукових системах. По-друге, скорочує час завантаження сторінки та підвищує

продуктивність на мобільних пристроях. Ці плюси призводять до зростання кількості користувачів.

Переваги Angular методу:

1. Angular пропонує структурований підхід до розробки веб-додатків, організовуючи код у модулі, компонентах та сервісах. Це сприяє покращенню читабельності та обслуговування виконуваного коду.

2. Angular надає можливість двостороннього зв'язування даних, що дозволяє автоматично оновлювати відображення даних в шаблонах при зміні їх значень у коді.

3. Angular сприяє модулярній розробці, дозволяючи розділити додаток на невеликі та незалежні компоненти, які можуть бути повторно використані та простіше тестувати.

4. Angular має велику екосистему, що включає в себе багато корисних інструментів, бібліотек та розширень, таких як Angular CLI, Angular Material, NgRx, і багато інших, що полегшує розробку та покращує функціональність вашого додатку.

5. Angular побудований на TypeScript, що дозволяє розробникам використовувати типізацію для виявлення помилок на етапі компіляції та полегшує роботу з великими проектами.

6. Angular дозволяє розробляти як веб-додатки, так і мобільні додатки за допомогою Angular Mobile Toolkit або сторонніх інструментів, таких як Ionic або NativeScript.

OpenAPI - формалізована специфікація та повноцінний фреймворк для опису, створення, використання та візуалізації веб-сервісів REST.

Переваги OpenAPI методу:

1. OpenAPI встановлює стандарт для опису API, що дозволяє розробникам та різним сервісам однаково розуміти та взаємодіяти з API. Це спрощує інтеграцію між різними системами та робить розробку більш прозорою.

2. OpenAPI дозволяє автоматично генерувати документацію для API на основі стандартизованого опису. Це полегшує розуміння та використання API для розробників та користувачів.

3. OpenAPI може використовуватися для валідації запитів, які надходять до вашого API, забезпечуючи відповідність вимогам API. Це допомагає уникнути помилок та забезпечує стабільну роботу сервісу.

4. OpenAPI може бути використаний для автоматизації певних аспектів розробки та управління API, таких як генерація коду, тестування та моніторинг.

5. OpenAPI інтегрується з багатьма різними інструментами для розробки та управління API, що дозволяє розробникам використовувати їхні улюблені інструменти для роботи з API.

6. OpenAPI може бути використаний для опису як простих, так і складних API, що дозволяє створювати різноманітні та масштабовані сервіси.

Рисунок 2.2. ілюструє архітектуру проекту побудованого засобами AWS Cloud.

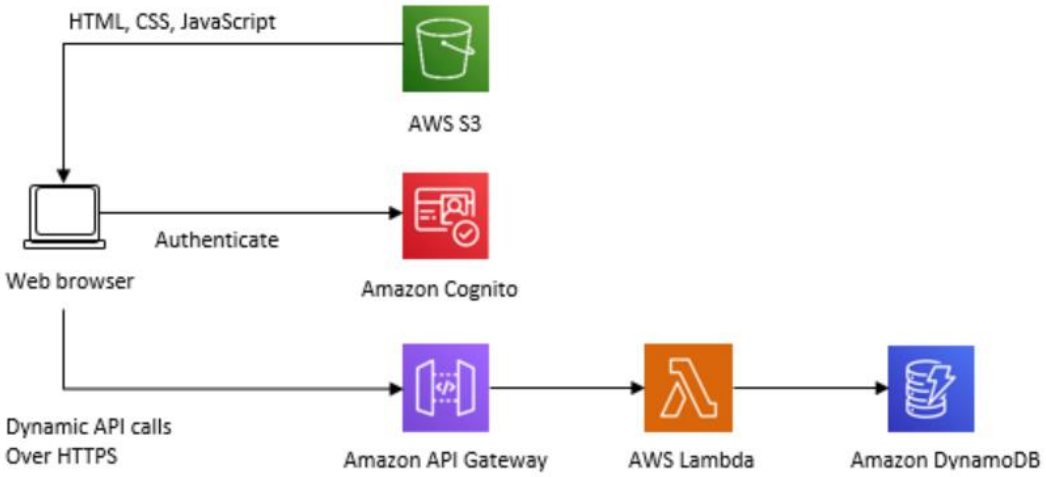


Рис. 2.2. Архітектура проекту з використанням інструментів AWS Cloud

Проект поділений на функціональні частини, кожна з яких відповідає своїм інструментам, наданим платформою AWS Cloud.

На початку роботи з AWS Cloud створюється “root user” акаунт AWS. При створенні акаунту артоматично включається спеціальний пакет “Free Tier”. Free

Tier призначений для практичного досвіду використання ряду послуг AWS безкоштовно. Він надає:

- S3 2000 запитів безкоштовно кожен місяць на 12 місяців (Global-Requests-Tier1)
- S3 20000 запитів безкоштовно кожен місяць на 12 місяців (Global-Requests-Tier2)
- S3 5.0 GB кожен місяць на 12 місяців (Global-TimedStorage-ByteHrs)
- DynamoDB 18600 ReadCapacityUnit-Hrs безкоштовно кожен місяць (EUN1-ReadCapacityUnit-Hrs)
- DynamoDB 18600 WriteCapacityUnit-Hrs безкоштовно кожен місяць (EUN1-WriteCapacityUnit-Hrs)
- AmazonCloudWatch 10 будильники завжди безкоштовні на місяць (Global-CW:AlarmMonitorUsage)
- API Gateway 750000 хвилин безкоштовно кожен місяць на 12 місяців (Global-ApiGatewayMinute)
- Lambda 1000000 запитів завжди безкоштовно кожен місяць (Global-Request)
- Lambda 400000 секунд завжди безкоштовно кожен місяць (Global-Request)

Amazon Simple Storage Service (Amazon S3) - це служба зберігання об’єктів, яка пропонує продуктивність, масштабованість, доступність і безпеку даних. Клієнти будь-якого розміру та галузі можуть зберігати та захищати необхідну кількість даних практично для будь-якого випадку використання. Наприклад, для дата пулів, хмарних та мобільних додатків. Вигідні класи сховищ і прості у використанні інструменти адміністрування дозволяють оптимізувати витрати, упорядковувати дані та налаштувати обмеження доступу відповідно до бізнес-потреб або вимог законодавства.

Amazon DynamoDB - це безсерверний сервіс баз даних NoSQL, що підтримує моделі даних типу ключ-значення та документ. Розробники можуть

використовувати його для створення сучасних безсерверних додатків, які можна запускати в невеликих обсягах і масштабувати по всьому світу. Amazon DynamoDB масштабується для підтримки таблиць практично будь-якого розміру за допомогою автоматичного горизонтального масштабування. Він призначений для виконання високопродуктивних програм у масштабах Інтернету, ефективну роботу яких не в змозі забезпечити традиційні реляційні бази даних. Amazon DynamoDB пропонує необмежену масштабованість, стабільну продуктивність у кілька мілісекунд та доступність до 99,999%[23].

Amazon API Gateway - це повністю керований сервіс для розробників, який спрощує публікацію, обслуговування, моніторинг, захист та використання API у будь-яких масштабах. Це сервіс з оплатою за фактом використання, який вирішує всі трудомісткі завдання щодо безпечної та надійної роботи API у будь-якому масштабі.[24] Він дозволяє створювати API RESTful, використовуючи API HTTP або API REST. API HTTP – найкращий спосіб створювати API, яким не потрібні можливості керування API. API HTTP оптимізовано для безсерверних робочих навантажень та серверної частини HTTP. Порівняно з API REST із API Gateway вони забезпечують зниження витрат майже на 71%, а затримки – на 60%. Для робочих навантажень, яким потрібно проксі-сервер для API та функції управління API, такі як плани використання та ключі API, в одному рішенні, API Gateway надає API REST.

REST – це стиль архітектури взаємодії компонентів у мережі інтернет. Зазвичай RESTful запити являють собою звичайні HTTP-запити (GET, POST, DELETE, PUT, OPTIONS), а дані передаються у вигляді параметрів.

Переваги REST:

1. REST використовує прості HTTP методи, такі як GET, POST, PUT, DELETE, щоб взаємодіяти з ресурсами, що дозволяє легко розуміти та використовувати API для розробників.

2. RESTful сервіси не зберігають стан клієнта між запитами, що дозволяє покращити масштабованість і спрощує розподілену обробку.

3. REST підтримує кешування, що дозволяє зберігати копії ресурсів на клієнтській стороні або на проксі-серверах, що у свою чергу поліпшує продуктивність та зменшує навантаження на сервер.

4. REST дозволяє будувати складні системи з різними типами ресурсів та їх взаємодією за допомогою простих та стандартизованих методів.

5. RESTful API може бути реалізований на будь-якій мові програмування та платформі, що робить його універсальним та доступним для розробників.

API WebSocket в API Gateway — набір маршрутів WebSocket, інтегрованих із внутрішніми кінцевими точками HTTP, функціями Lambda або іншими службами AWS. Ви можете використовувати функції API Gateway, щоб допомогти вам з усіма аспектами життєвого циклу API, від створення до моніторингу робочих API. API Gateway WebSocket є двонаправленими. Клієнт надсилає повідомлення до служби, а служби самостійно надсилають повідомлення клієнтам. Така двонаправлена поведінка забезпечує розширену взаємодію між клієнтом і службою, оскільки служби можуть надсилати дані клієнтам, не вимагаючи від клієнтів робити явний запит. API WebSocket часто використовуються в програмах реального часу, таких як програми чату, платформи для співпраці, багатокористувацькі ігри та фінансові торгові платформи.

Переваги WebSocketAPI:

1. WebSocket дозволяє встановлювати постійне з'єднання між клієнтом та сервером, що зменшує затримку у передачі даних порівняно з традиційними HTTP-запитами.

2. WebSocket забезпечує повноцінний двосторонній зв'язок між клієнтом та сервером, що дозволяє ініціювати взаємодію з обох сторін.

3. WebSocket має низький рівень накладних витрат, оскільки не вимагає повторного встановлення з'єднання для кожного запиту, що дозволяє ефективно передавати великі обсяги даних без зайвого навантаження на сервер.

4. WebSocket використовує бінарний протокол передачі даних, що дозволяє передавати дані у бінарному форматі без додаткового перетворення.

5. WebSocket підтримується більшістю сучасних браузерів і мов програмування, що дозволяє легко розробляти кросс-платформені додатки.

6. WebSocket дозволяє зберігати активне з'єднання між клієнтом та сервером протягом тривалого часу, що дозволяє швидко реагувати на події або оновлення без необхідності повторно встановлювати з'єднання.

AWS Lambda - це обчислювальний сервіс, який запускає код у відповідь на події та автоматично керує обчислювальними ресурсами, що дозволяє швидше перетворити ідею на сучасні виробничі безсерверні програми. Можна створити вебAPI з кінцевою точкою HTTP для функції Lambda за допомогою Amazon API Gateway. API Gateway надає інструменти для створення та документування вебAPI, які спрямовують HTTP-запити до функцій Lambda. Можна також захистити доступ до свого API за допомогою елементів керування автентифікацією та авторизацією. API можуть обслуговувати трафік через Інтернет або можуть бути доступні лише у VPC.

Amazon Cognito – це сервіс управління ідентифікацією клієнтів та доступом (CIAM). Він надає варіанти безпечного сховища та федерації посвідчень, які можуть масштабуватись до мільйонів користувачів. Amazon Cognito підтримує вхід з використанням постачальників посвідчень соціальних мереж та постачальників посвідчень на основі SAML або OIDC, щоб забезпечити задоволеність клієнтів, і пропонує розширені можливості безпеки для захисту ваших клієнтів та підприємств.

Підтримка роботи вебдодатку проводиться за допомогою сервісу Gitlab CI/CD.

CI/CD (Continuous Integration / Continuous Deployment) – це система, призначена для підвищення зручності, частоти та надійності публікації змін програмного забезпечення або продукту, де:

CI - це практика розробки ПЗ, при якій зміни в коді автоматично збираються, тестуються та інтегруються в цільову гілку репозиторію. Основна

ідея — мінімізація розриву між компонентами проекту та швидкий зворотній зв'язок про якість коду завдяки автоматичному складанню та тестуванню.

CD – це продовження CI, яке дозволяє автоматично розгортати успішно зібраний та протестований код на сервері чи іншому середовищі реального застосування, в нашому випадку – AWS Cloud. Мета - автоматизація процесу розробки та розгортання програми або програмного продукту після всіх етапів перевірки та тестування. Розгортання повинно виконуватись після ручного підтвердження деплою, щоб надати додатковий рівень контролю та безпеки.[25]

Gitlab CI/CD - повністю інтегрована в GitLab система для автоматизації складання, тестування та розгортання програмного коду. GitLab CI/CD використовує файл конфігурації YAML в репозиторію проекту для визначення правил роботи на кожному етапі в пайплайні. Підтримує використання Docker-образів для визначення оточення збірки – звідси велика гнучкість та легкість використання коду.[26]

Фронтенд частина розроблена за допомогою Figma.

Figma - хмарний багатоплатформовий сервіс для дизайнерів інтерфейсів і web-розробників, з яким можна працювати як у браузері, так і у відповідному додатку. Figma використовується для розробки прототипів web-сайтів і додатків, окремих елементів інтерфейсу: іконки, кнопки, форми, векторні зображення та ілюстрації, інше.[27]

Використано мови програмування Python, Java, .NET(C#), HTML5, CSS3, JS (Javascript) і TypeScript. Крім цього було використано такі бібліотеки: aws-lambda-java-core, aws-lambda-java-events, aws-java-sdk-dynamodb, lombok, @ctrl/ngx-emoji-mart, @angular, rxjs.

Python - це мова програмування, яка широко використовується в інтернет-додатках, розробці програмного забезпечення, науці про дані та машинному навчанні (ML). Розробники використовують Python, тому що він ефективний, простий у вивченні та працює на різних платформах.

Переваги Python:

1. Python відомий своєю простотою синтаксису, що робить його легко

вивчається для початківців. Код на Python зазвичай коротший та більш зрозумілий, що полегшує розробку та обслуговування програмного забезпечення.

2. Python використовується у багатьох галузях, включаючи веб-розробку, наукові обчислення, штучний інтелект, аналіз даних, автоматизацію та багато іншого. Це робить його універсальним інструментом для різних проектів.

3. Python має велику кількість сторонніх бібліотек і фреймворків, які розширюють його функціональність та сприяють швидкій розробці. Наприклад, Django та Flask для веб-розробки, NumPy та Pandas для аналізу даних, TensorFlow та PyTorch для машинного навчання, тощо.

4. Python відомий своєю високою продуктивністю та швидкістю розробки. Завдяки простому синтаксису та широкій екосистемі бібліотек, розробники можуть швидко створювати та тестувати нові функції.

5. Python має велику та активну спільноту розробників, яка надає велику кількість ресурсів, форумів, та документації для підтримки розробників та вирішення їхніх проблем.

6. Python є крос-платформеною мовою програмування, що означає, що програми, написані на Python, можуть запускатися на різних операційних системах, включаючи Windows, macOS та різні дистрибутиви Linux.

Java – мова програмування, що широко використовується для написання інтернет-додатків. Мова Java широко використовувалася протягом більш як двох десятиліть. Мільйони програм Java використовуються і сьогодні. Java - це багатоплатформна, об'єктно-орієнтована і сетецентрична мова, яка сама по собі може використовуватися як платформа. Це швидка, безпечна та надійна мова програмування для всього: від мобільних додатків та корпоративного ПЗ до додатків для роботи з великими даними та серверних технологій.

Переваги Java:

1. Однією з головних переваг Java є її платформенна незалежність. Програми Java можуть бути виконані на будь-якій платформі, яка підтримує віртуальну машину Java (JVM), що робить їх переносимими між різними

операційними системами.

2. Java відома своєю високою надійністю та безпекою. Вона має вбудовану систему керування пам'яттю, що допомагає уникнути багатьох типових помилок, таких як витоки пам'яті або некоректне використання пам'яті.

3. Java базується на об'єктно-орієнтованій моделі програмування, що сприяє полегшенню розробки, тестування та обслуговування програмного забезпечення шляхом використання об'єктів та їх взаємодії.

4. Java має велику кількість стандартних бібліотек, які включають в себе інструменти для роботи з мережами, роботи з базами даних, обробки тексту, графікою та багато іншого, що дозволяє розробникам швидко створювати різноманітні типи програм.

5. Java має одну з найбільших спільнот розробників у світі, що надає велику кількість ресурсів, документації, форумів та бібліотек для розробки програмного забезпечення на Java.

6. Java підтримує різні типи програм, від малих додатків до великих підприємницьких систем, та має ряд функцій, які полегшують масштабування програм, такі як пакетування та модульність.

.NET(C#) - це платформа з відкритим вихідним кодом для створення настільних, мобільних та веб-застосунків, які можуть працювати в будь-якій операційній системі. Система .NET включає інструменти, бібліотеки та мови, що підтримують сучасну, масштабовану і високопродуктивну розробку програмного забезпечення. Платформу .NET підтримує та обслуговує активна спільнота розробників.

Переваги .NET:

1. .NET підтримує кілька мов програмування, таких як C#, VB.NET, F#, що дозволяє розробникам використовувати мову, яка найкраще підходить для конкретного завдання або їхніх уподобань.

2. .NET має велику кількість бібліотек, фреймворків та інструментів, що дозволяє розробникам швидко розробляти різноманітні програми та застосунки.

3. .NET підтримує різні типи програм, від веб- до мобільних додатків та ігор. Це дозволяє розробникам використовувати одну платформу для розробки різноманітних проектів.

4. Visual Studio, основне інтегроване середовище розробки (IDE) для .NET, надає багато інструментів для швидкої та ефективної розробки програмного забезпечення.

5. .NET має багато функцій, які роблять його привабливим для великих підприємств, таких як висока масштабованість, безпека, підтримка хмарних технологій, а також інструменти для моніторингу та управління.

6. Запуск .NET Core був перехідним етапом до повного переходу до відкритого джерела з .NET 5, що відкрило нові можливості співпраці та розвитку для спільноти розробників.

HTML5 - це мова розмітки що використовується, щоб повідомляти вашому браузеру, як відображати веб-сторінки, які ви відвідуєте. Він може бути складним або простим, залежно від того, як веб-дизайнер хоче. HTML складається з ряду елементів, які використовуються щоб вкладати або обертати різні частини контенту, щоб змусити контент відображатись або діяти певним чином.

Переваги HTML5:

1. HTML5 додає підтримку вбудованих мультимедійних елементів, таких як <video> та <audio>, що дозволяє відтворювати відео та аудіо без використання додаткових плагінів.

2. HTML5 має підтримку для створення графіки та анімації за допомогою <canvas> та CSS3. Це дозволяє створювати складні графічні ефекти та інтерактивні анімації без необхідності використання Flash або інших додаткових технологій.

3. HTML5 працює на різних пристроях та платформах, включаючи смартфони та планшети. Також має підтримку респонсивного дизайну, що дозволяє створювати веб-сайти, які адаптуються до різних розмірів екранів.

4. HTML5 має можливості для роботи в офлайн-режимі та локального збереження даних за допомогою технологій, таких як Web Storage та IndexedDB. Це дозволяє створювати веб-додатки, які можуть працювати незалежно від з'єднання з Інтернетом.

5. HTML5 містить нові семантичні елементи, такі як <header>, <footer>, <nav>, <article> та інші, що полегшують створення доступних та структурованих веб-сторінок.

6. HTML5 має покращену продуктивність та швидкість завантаження сторінок за рахунок оптимізації ресурсів та покращеної роботи з асинхронним JavaScript.

CSS3 - мова ієрархічних правил (таблиць стилів), що використовується для представлення зовнішнього вигляду документа, написаного на HTML або XML описує, яким чином елемент повинен відображатися на екрані, на папері, голосом або з використанням інших медіа засобів.

Переваги CSS3:

1. CSS3 надає розширені можливості стилізації, такі як тіні, градієнти, заокруглення кутів та інші ефекти, що раніше були важко досяжні без використання графічних зображень або скриптів.

2. CSS3 додає підтримку анімації та переходів, що дозволяє створювати різноманітні анімаційні ефекти без необхідності використання JavaScript або Flash.

3. CSS3 дозволяє створювати гнучкі та адаптивні макети, що легко адаптуються до різних розмірів екранів та пристроїв.

4. CSS3 включає медіа-запити, що дозволяють розробникам створювати стилі для різних типів пристроїв та екранів, таких як мобільні пристрої, планшети та настільні комп'ютери.

5. CSS3 надає зручні засоби для виконання трансформацій, таких як переміщення, масштабування, обертання та скупчення, що дозволяє створювати складні макети та ефекти.

6. CSS3 дозволяє використовувати веб-шрифти, що відкриває нові можливості для стилізації тексту та поліпшення вигляду веб-сайтів.

Javascript - це легковажна інтерпретована мова програмування з функціями першого класу. Найширше застосування знаходить мову сценаріїв веб-сторінок, але також використовується і в інших програмних продуктах, наприклад Node.js або Apache CouchDB. JavaScript це прототипно-орієнтована, мультипарадигменна мова з динамічною типізацією, яка підтримує об'єктно-орієнтовану, імперативну та декларативну стилі програмування.

Переваги Javascript:

1. Однією з найбільших переваг JavaScript є можливість використання її для створення динамічних та інтерактивних веб-додатків. Вона дозволяє додавати різноманітні функції, валідацію форм, анімацію та інші елементи, що поліпшують користувацький досвід.

2. JavaScript може виконуватися на різних платформах, включаючи браузери, сервери та мобільні пристрої. Це дозволяє розробникам створювати програми, які працюють на різних пристроях та операційних системах.

3. JavaScript має простий синтаксис, що робить її досить легкою для вивчення, особливо для початківців. Вона використовується багатьма веб-розробниками як перша мова програмування.

4. JavaScript має широкий спектр функціональних можливостей, включаючи роботу з DOM (Document Object Model), AJAX (Asynchronous JavaScript and XML), маніпулювання графікою та анімацією, роботу з cookies, роботу з локальним сховищем та багато іншого.

5. JavaScript має велику та активну спільноту розробників, яка надає безліч ресурсів, бібліотек, фреймворків та інших інструментів для полегшення розробки програм.

6. JavaScript дозволяє розробникам швидко прототипувати та розгортати веб-додатки, що дозволяє їм ефективно працювати над проектами та швидко впроваджувати нові функції.

TypeScript - безкоштовна високорівнева мова програмування з відкритим вихідним кодом, розроблена корпорацією Microsoft, яка додає до JavaScript статичний тип із додатковими анотаціями типу. Розроблений для розробки великих додатків і транспілюється на JavaScript. Оскільки TypeScript є надмножиною JavaScript, усі програми JavaScript є синтаксично дійсними TypeScript, але вони можуть не перевіряти тип з міркувань безпеки.

Переваги TypeScript:

1. TypeScript підтримує статичну типізацію, що дозволяє визначати типи даних для змінних, параметрів функцій, властивостей об'єктів тощо. Це дозволяє виявляти та виправляти помилки в коді на етапі розробки, підвищуючи його надійність та облегшуючи рефакторинг.

2. TypeScript базується на стандарті ECMAScript, тому підтримує всі сучасні можливості JavaScript, такі як стрілкові функції, розширений синтаксис класів, розпростереження, деструктуризація та інші.

3. TypeScript надає додаткові можливості для організації та підтримки коду, такі як модульність, інтерфейси, узагальнення та інші. Це дозволяє створювати більш чистий, структурований та повторно використовуваний код.

4. TypeScript інтегрується з багатьма популярними інструментами розробки, такими як Visual Studio Code, WebStorm, Sublime Text та інші, що полегшує розробку та налагодження програм.

5. TypeScript може використовувати існуючий JavaScript код безпосередньо, що дозволяє поступово впроваджувати його у проект без необхідності переписування великої частини коду.

6. TypeScript має велику та активну спільноту розробників, яка надає безліч ресурсів, бібліотек, фреймворків та інших інструментів для полегшення розробки програм.

Бібліотека @angular - відкрита платформа для розробки веб-додатків, написана мовою TypeScript, що розробляється командою Google, а також спільнотою розробників з різних компаній.

Переваги бібліотеки @angular:

1. Angular використовує модель-вид-контролер (MVC) архітектуру, що сприяє розділенню логіки додатку на моделі даних, представлення та управління.
2. Angular надає двостороннє зв'язування даних між моделлю та представленням, що дозволяє автоматично оновлювати відображення при зміні даних та навпаки.
3. Вбудована система внедрення залежностей (Dependency Injection, DI) дозволяє легко керувати залежностями та забезпечує легке тестування додатків.
4. Angular має широкий вибір бібліотек, модулів та інструментів для розширення його функціональності та покращення продуктивності розробки.
5. Angular базується на компонентах, які є будівельними блоками веб-інтерфейсу. Це спрощує створення, тестування та підтримку складних користувацьких інтерфейсів.
6. Angular дозволяє розробляти як веб-додатки, так і мобільні додатки за допомогою Angular Mobile Toolkit та NativeScript.
7. Angular CLI (Command Line Interface) надає набір інструментів для швидкого створення, розгортання та керування Angular додатками.
8. Angular має вбудовані інструменти для тестування, включаючи юніт-тести та енд-ту-енд тести, що дозволяє забезпечити високу якість коду.

Бібліотека Rxjs - бібліотека для складання асинхронних і подієвих програм за допомогою спостережуваних послідовностей. Він надає один основний тип, Observable, супутникові типи і оператори, натхненні методами Array, щоб дозволити обробку асинхронних подій як колекцій.[28]

Rxjs поєднує шаблон Observer із шаблоном Iterator і функціональне програмування з колекціями, щоб задовольнити потребу в ідеальному способі керування послідовністю подій.

Основні концепції в RxJS, які вирішують асинхронне керування подіями:

- Observable: представляє ідею викликаної колекції майбутніх цінностей або подій.

- Observer: це набір зворотних викликів, який знає, як прослуховувати значення, які надає Observable.
- Subscription: представляє виконання Observable, в першу чергу корисна для скасування виконання.
- Operators: це чисті функції, які забезпечують функціональний стиль програмування роботи з колекціями за допомогою таких операцій, як map, filter, concat, reduce тощо.
- Subject: еквівалент EventEmitter і єдиний спосіб групової передачі значення чи події кільком спостерігачам.
- Schedulers: це централізовані диспетчери для керування паралельністю, що дозволяє нам координувати, коли обчислення відбуваються, наприклад, setTimeout або requestAnimationFrame або інші.

Переваги RxJS:

1. RxJS дозволяє працювати з асинхронними операціями та подіями як з потоками даних. Це дозволяє зручно працювати з асинхронним кодом, включаючи події від клієнтського інтерфейсу, запити до сервера та інші асинхронні операції.
2. RxJS надає потужні засоби для композиції та обробки потоків даних, що дозволяє виконувати складні маніпуляції з даними, такі як фільтрація, мапування, складання, з'єднання та інші, у зручний та зрозумілий спосіб.
3. RxJS може використовуватися для різних типів даних та сценаріїв, включаючи роботу з потоками даних, обробку подій, роботу з віддаленими джерелами даних, автоматичне кешування та інше.
4. RxJS дозволяє створювати реактивні програми, які реагують на зміни вхідних даних та автоматично оновлюють відображення інтерфейсу користувача або виконують інші дії.
5. RxJS має великий набір операторів для роботи з потоками даних, що дозволяє виконувати різноманітні операції над даними без необхідності писати власний код.

6. RxJS має велику та активну спільноту розробників, яка надає безліч ресурсів, документації, прикладів коду та підтримує розвиток бібліотеки.

Бібліотека @ctrl/ngx-emoji-mart - порт emoji-mart від missive. Emoji-mart налаштовуваний HTML-компонент вибору емодзі для web.

Переваги @ctrl/ngx-emoji-mart:

1. @ctrl/ngx-emoji-mart надає готові компоненти, які легко інтегрувати в Angular-додатки. Це дозволяє швидко і просто додавати функціональність емодзі у додатку.

2. Бібліотека містить велику кількість різноманітних емодзі, що дозволяє користувачам обирати та використовувати емодзі в своїх повідомленнях або коментарях.

3. @ctrl/ngx-emoji-mart має різні налаштування та опції конфігурації, які дозволяють налаштувати вигляд та поведінку компонентів емодзі відповідно до потреб вашого додатку.

4. Бібліотека організована за категоріями, що полегшує пошук та вибір емодзі. Крім того, вона підтримує можливість швидкого пошуку потрібного емодзі.

5. Компоненти @ctrl/ngx-emoji-mart мають адаптивний дизайн, що дозволяє їм працювати на різних пристроях та розмірах екрану.

6. Бібліотека підтримується спільнотою Angular, що означає, що вона легко інтегрується з Angular-проєктами й отримує регулярні оновлення та підтримку.

2.4. Опис структури системи та алгоритмів її функціонування

Структура вебплатформи наведена на рис. 2.3.

У подробицях структуру проекту можна описати таким чином:

- ci - .yaml-файли, відповідають за логіку розгортання додатку;
- lambda-deployment – архіви з java lambda кодом;

- megathon-dotnet-backend – lambda код на .NET;
- megathon-frontend - відповідає за зовнішній вигляд додатку;
- megathon-java-backend - lambda код на Java;
- terraform - логіка розгортання сервісів AWS Cloud;
- tf-custom-module\lambda – шаблон розгортання lambda з папки terraform;
- .gitlab-ci.yml – конфігураційний файл для CI/CD пайплайнів;
- README.md - файл-tutorial.

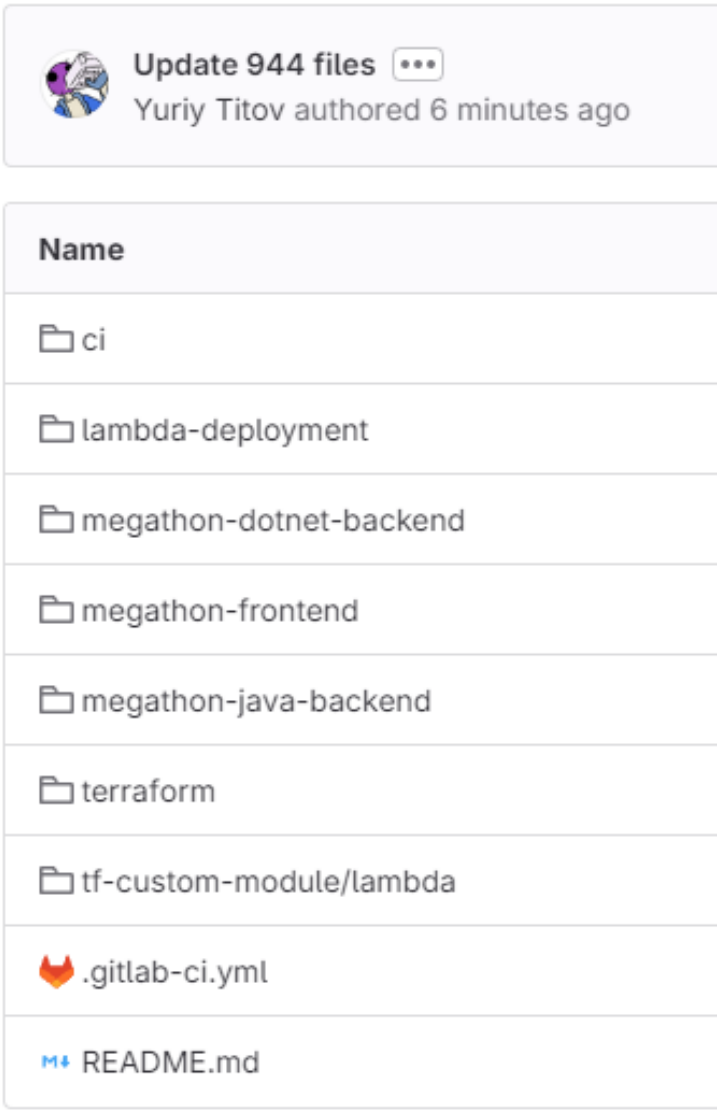


Рис. 2.3. Лістинг файлів директорії проекту у GitLab

Алгоритм функціонування платформи:

1. Платформа слідує архітектурі REST.

2. Для розробки запускається статичний сайт на базі S3 у публічному бакеті it-marathon-v3-frontend.

3. Обробка запитів та надсилання HTTP-відповіді виконується за допомогою Lambda функцій

4. Маршрутизацію запиту здійснює REST API Gateway під назвою it-marathon-v3-api-gateway, що й визначає яка Lambda повинна бути викликана за даної URL-адреси та даного типу запиту.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Розроблена платформа використовує базу даних DynamoDB для зберігання інформації різних моделей. Для взаємодії з даними, їх зберіганням та виведенням використовуються AWS Web sockets, які викликають у разі потреби відповідні Lambda функції.

База даних DynamoDB має NoSQL структуру. Усього для роботи платформи створено три таблиці DynamoDB:

– it-marathon-v3-message-db – таблиця де зберігаються усі записи чатів користувачів

– it-marathon-v3-user-conect-db – таблиця де зберігаються усі створені користувачами з’єднання з вебсокетом

– it-marathon-v3-chat-db – таблиця де зберігаються усі створені чати між користувачами

– it-marathon-v3-user-db – таблиця де зберігається уся інформація зареєстрованих користувачів

На Рис.2.4 показаний вигляд таблиць DynamoDB у AWS Cloud.

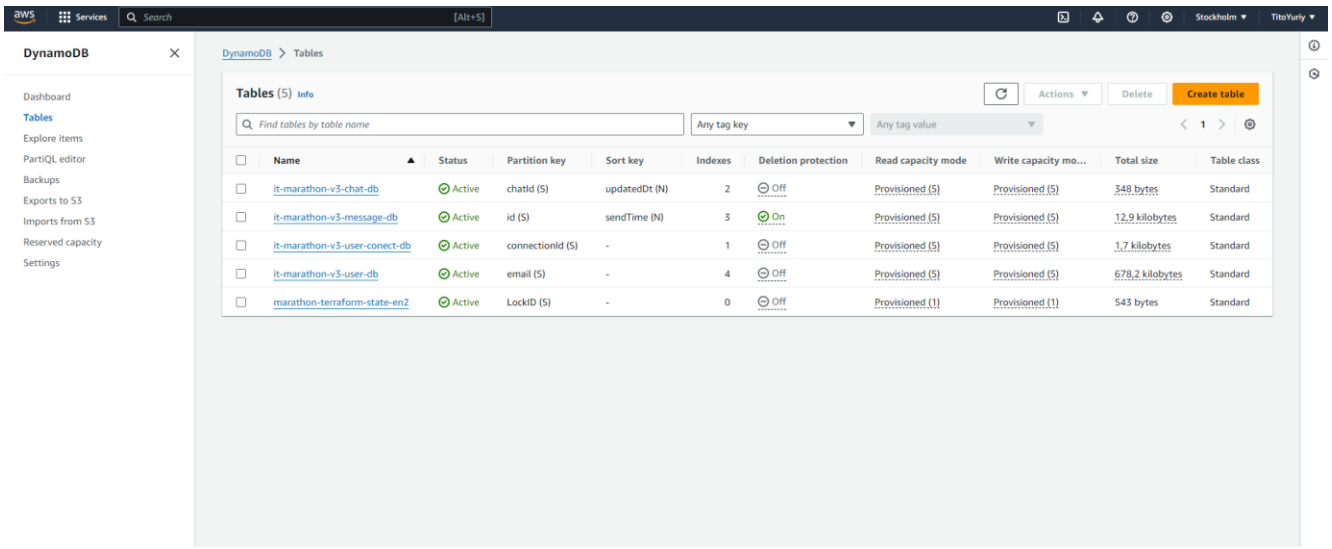


Рис. 2.4. Таблиці DynamoDB у AWS Cloud

Структура таблиць БД. Опис полів усіх таблиць БД наведено у таблицях 2.1-2.4.

Таблиця 2.1

Опис полів таблиці it-marathon-v3-message-db

Назва поля	Тип даних поля	Ціль (значення)	Обмеження (constraints)
id	String	Id чату до якого належить повідомлення	
sendTime	Number	Дата та час надсилання повідомлення	unique=True primaryKey
message	String	Зміст повідомлення	
receiverId	String	Пошта приймача	
seen	Binary	Чи подивився приймач повідомлення	default=False
senderId	String	Пошта відправника	

Таблиця 2.2

Опис полів таблиці it-marathon-v3-chat-db

Назва поля	Тип даних поля	Ціль (значення)	Обмеження (constraints)
chatId	String	Унікальний ідентифікатор	unique=True primaryKey

Продовження таблиці 2.2

updatedDt	Number	Дата та час останньої активності у чаті	
user1	String	Email користувача 1	
user2	String	Email користувача 2	

Таблиця 2.3

Опис полів таблиці it-marathon-v3-user-conect-db

Назва поля	Тип даних поля	Ціль (значення)	Обмеження (constraints)
connectionId	String	Унікальний ідентифікатор	unique=True primaryKey
userId	String	Email користувача	

Таблиця 2.4

Опис полів таблиці it-marathon-v3-user-db

Назва поля	Тип даних поля	Ціль (значення)	Обмеження (constraints)
email	String	Унікальний ідентифікатор	unique=True primaryKey
about	String	Персональний опис користувача	
avatar	Data	URL-код аватару користувача	
birthday	Number	Дата народження користувача	
country	String	Країна проживання користувача	
interests	Map	Список інтересів користувача	
location	String	Місто проживання користувача	
name	String	Ім’я користувача	
privacy	Binary	Чи приватна у користувача інформація	default=False

Продовження таблиці 2.4

registration	Number	Дата реєстрації користувача	
social_media	Map	Посилання на соціальні мережі користувача	

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Розроблена вебплафторма базується у хмарному сервісі AWS Cloud з мінімальними характеристиками потужностей, а тому має змогу обробляти певну кількість клієнтських запитів. Для розробки платформи було використано такі характеристики системи:

- операційна система: Windows;
- процесор з тактовою частотою 1 ГГц або вище;
- RAM об'ємом 1ГБ або вище;
- жорсткий диск об’ємом 10 ГБ або вище;

монітор, миша (тачпад) і клавіатура - для розробки.

2.6.2. Використані програмні засоби

Під час створення платформи були використані наступні програмні засоби:

- браузер Chrome версії 123.0.6312.106;
- Microsoft Visual Studio Code;
- Eclipse IDE;
- PostrgeSQL;

Як бекенд було використано мову Python версії 3.9, Java версії 1.9, C# версії 2.23. Крім цього було використано такі бібліотеки: aws-lambda-java-core, aws-lambda-java-events, aws-java-sdk-dynamodb, lombok, @ctrl/ngx-emoji-mart, @angular, rxjs.

Фронтенд було розроблено за допомогою векторного онлайн-сервісу розробки інтерфейсів Figma на мові гіпертекстової розмітки HTML5, каскадних таблицях стилів CSS3 і мовах програмування JS (Javascript) і TypeScript angular. Приклад зовнішнього вигляду фронтенду показано на рис. 2.5.

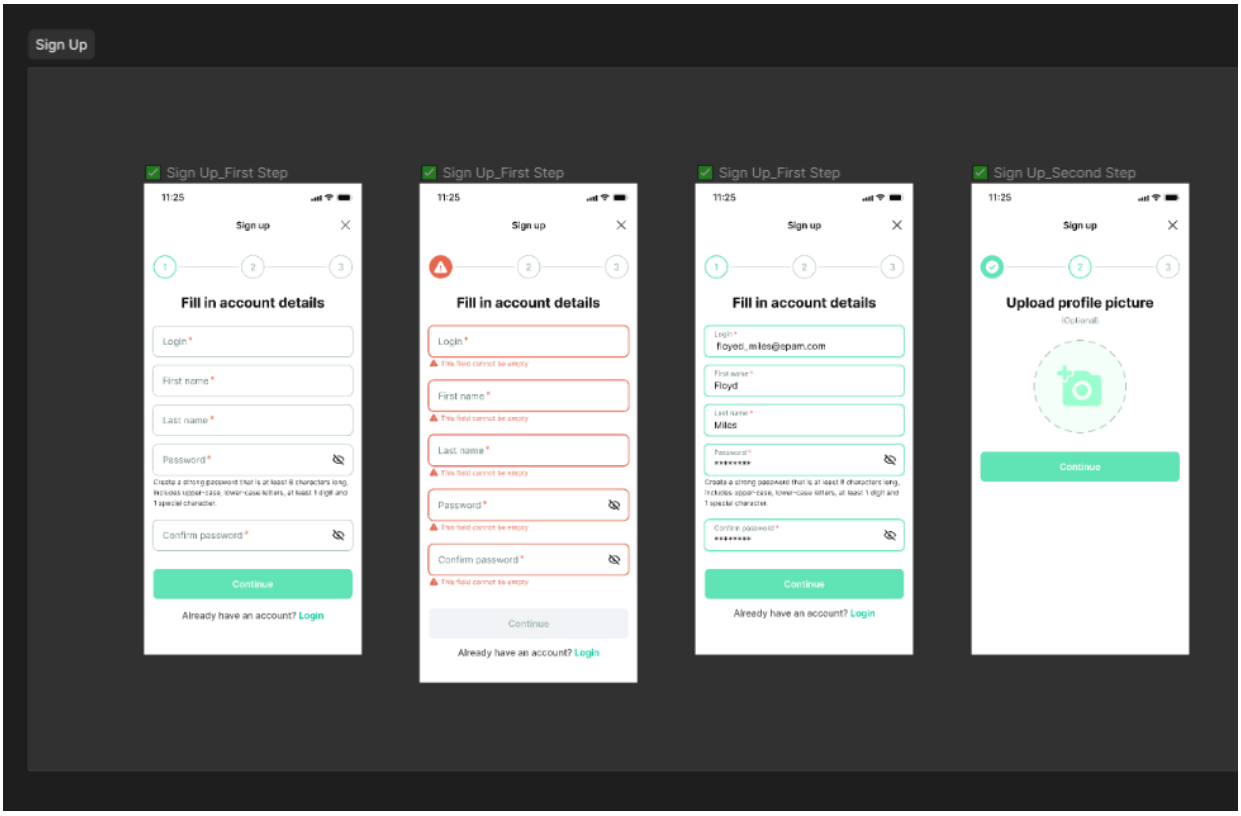


Рис. 2.5. Дизайн сторінок реєстрації у Figma

Деплой вебсайту виконується за допомогою GitLab CI/CD scheduled pipeline “test”. Pipeline розпочинає свою роботу з перевірки налаштувань у файлі “.gitlab-ci.yml”, а потім розподіляє асинхронний деплой з файлу “ci/scheduled_pipeline.yaml”(Рис. 2.6.). На Рис. 2.7-8. приведено зовнішній вигляд роботи pipeline “test”. Деплой проводиться першого числа кожного місяця на гілці “devl”.

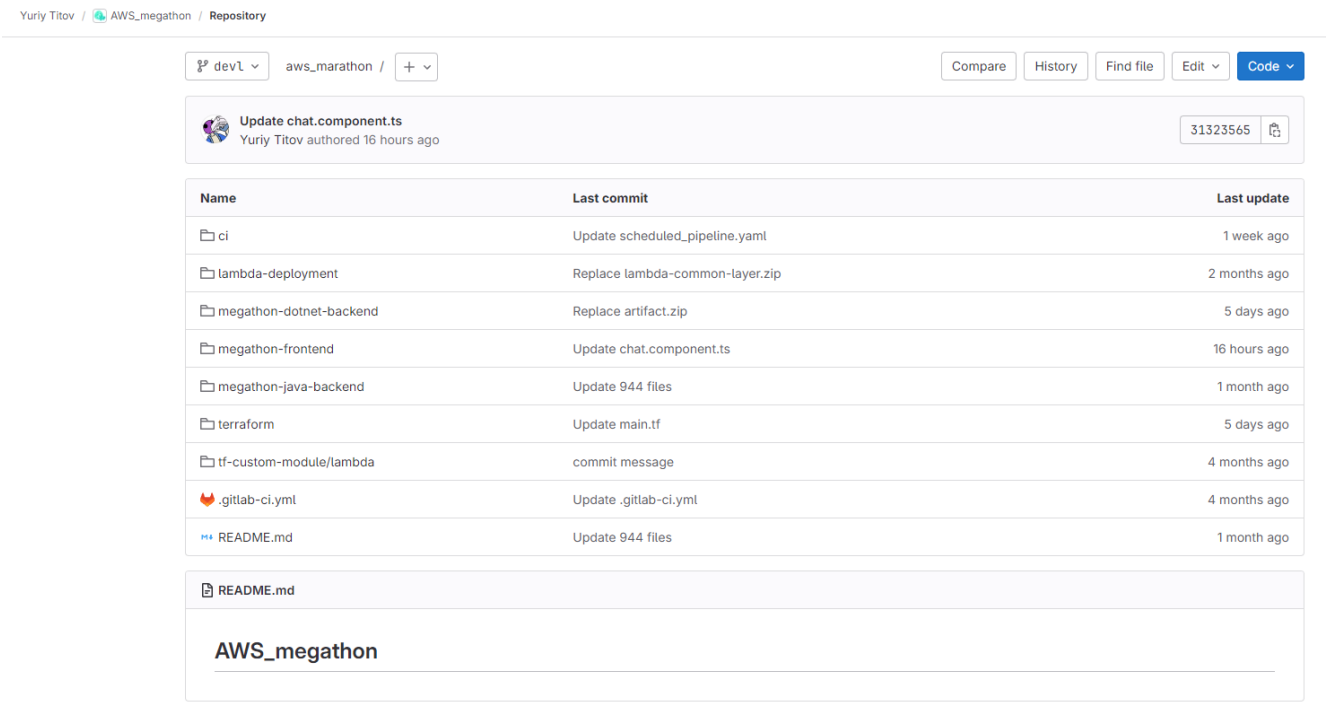


Рис. 2.6. Файлова система проекту у GitLab

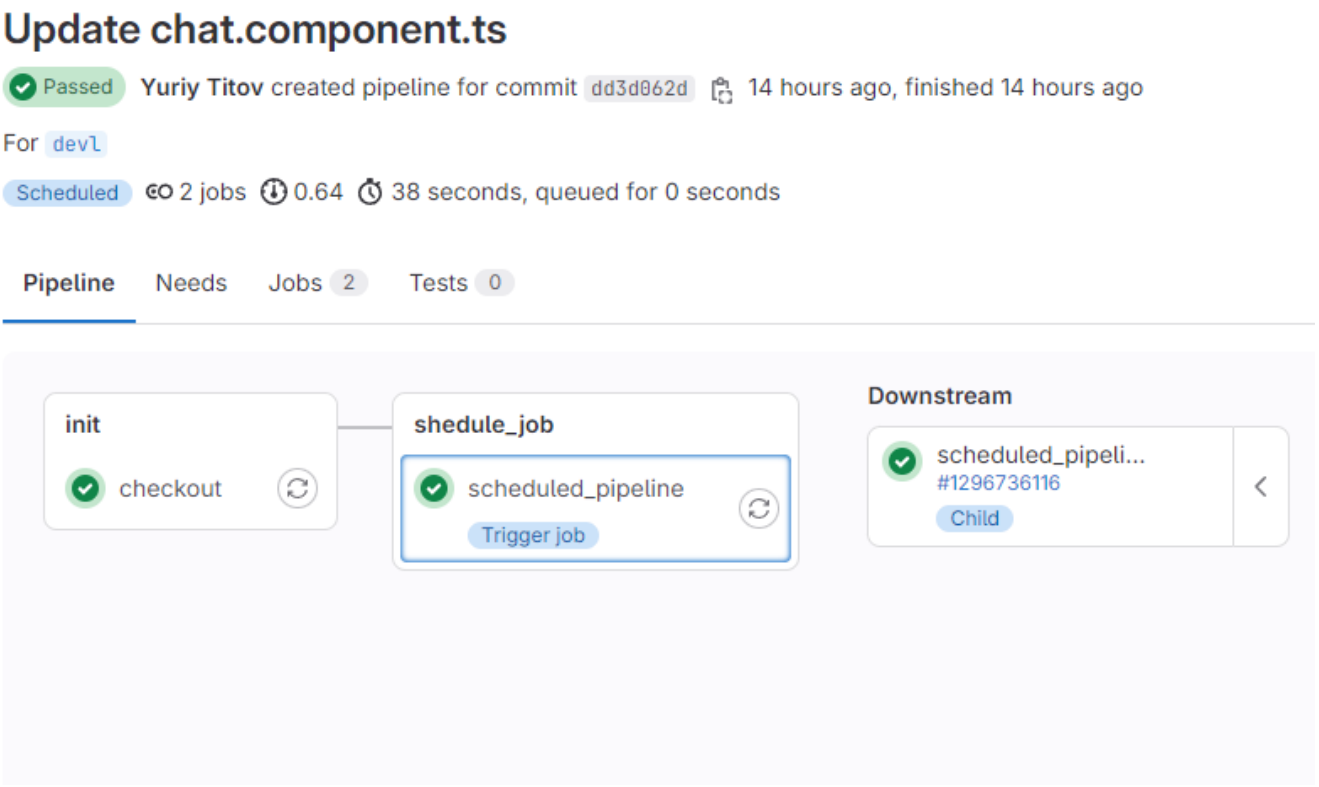


Рис. 2.7. CI/CD pipeline по деплою проекту у GitLab

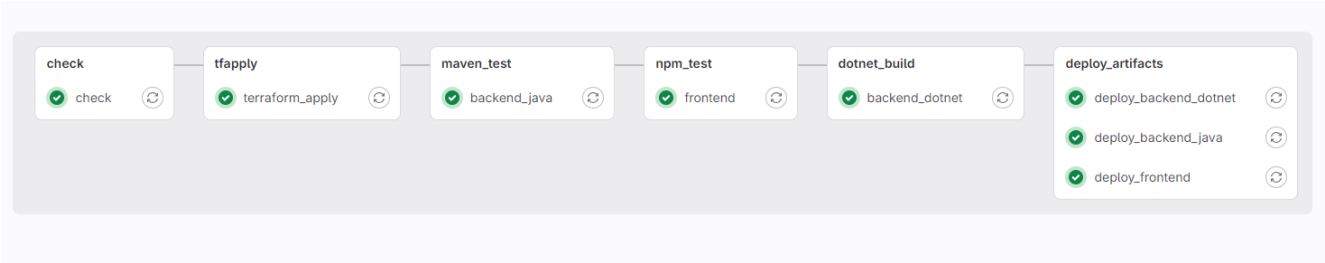


Рис. 2.8. етапи деплою проекту у GitLab

Тестування роботи платформи проводиться за допомогою мови програмування Java, а саме бібліотеки створення тестів TestNG та бібліотеки взаємодії зі структурою вебсайтів Selenium для Apache Maven. На рис. 2.9. показано вигляд створених тестів.

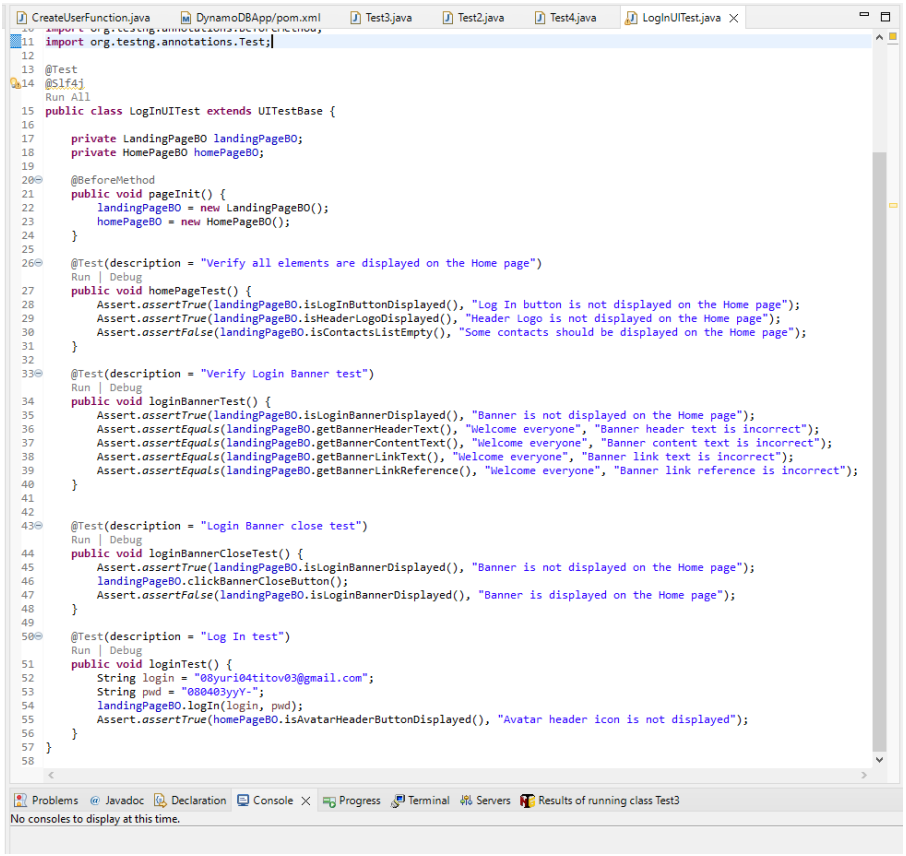


Рис. 2.9. тести у Java

Тестування роботи API Gateway та WebSocket ефективніше всього робити у Postman. Надсилання HTTP-запитів, створення тестів, організація запитів у

колекції, робота зі змінними – все це лише частина функціональності Postman, яка полегшує процес тестування та підвищує його ефективність. (Рис. 2.10.)

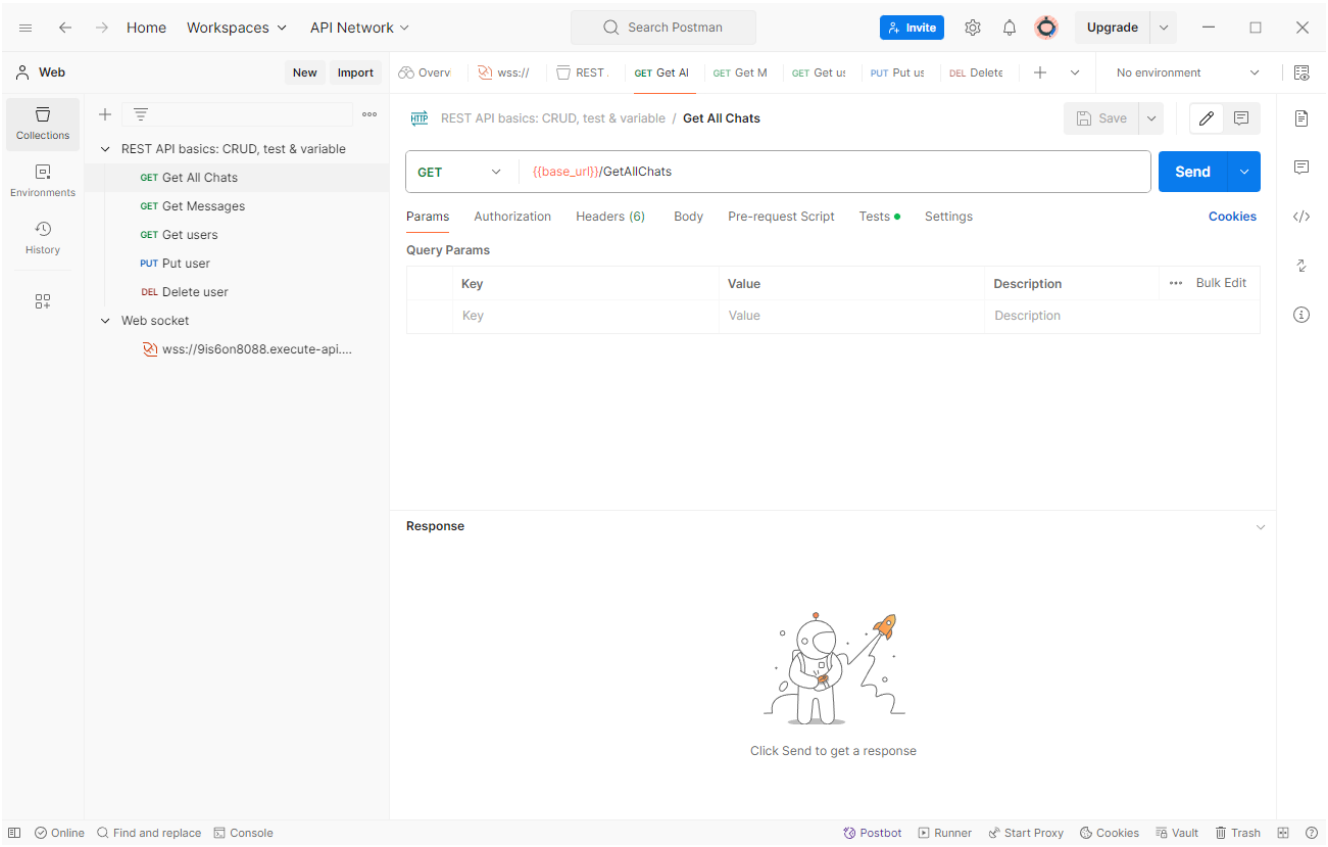


Рис. 2.10. Тести у Postman

2.6.3. Виклик та завантаження програми

Щоб запустити сервер платформи, потрібно:

- відкрити AWS S3 бакет з фронтенд частиною вебсайту під назвою it-marathon-v3-frontend як адмін.
- Для цього необхідно:

1. Зайти на сайт AWS Console Home за наданими логіном та паролем адміну.
2. Вибрати регіон eu-north-1.
3. Перейти до сервісу S3.
4. Відкрити it-marathon-v3-frontend(Рис.2.11.)

5. Перейти у вкладинку “Properties”.

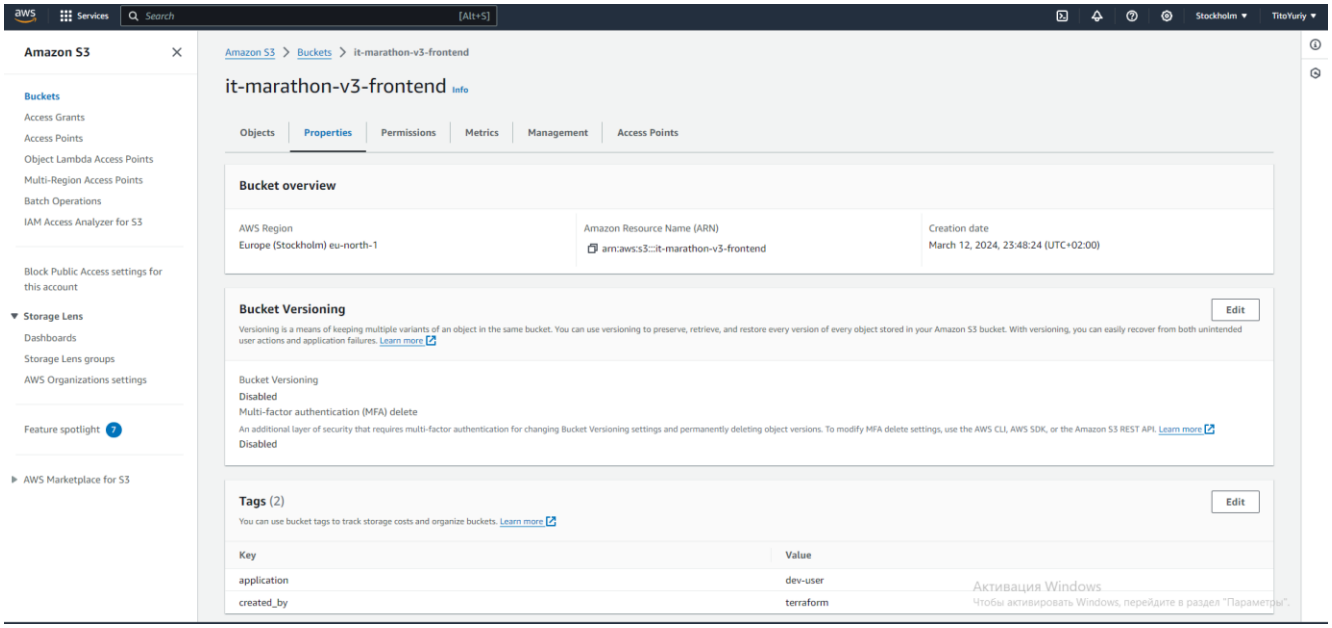


Рис. 2.11. Вкладка “Properties” у it-marathon-v3-frontend

6. Увімкнути “Static website hosting”(Рис.2.12.);

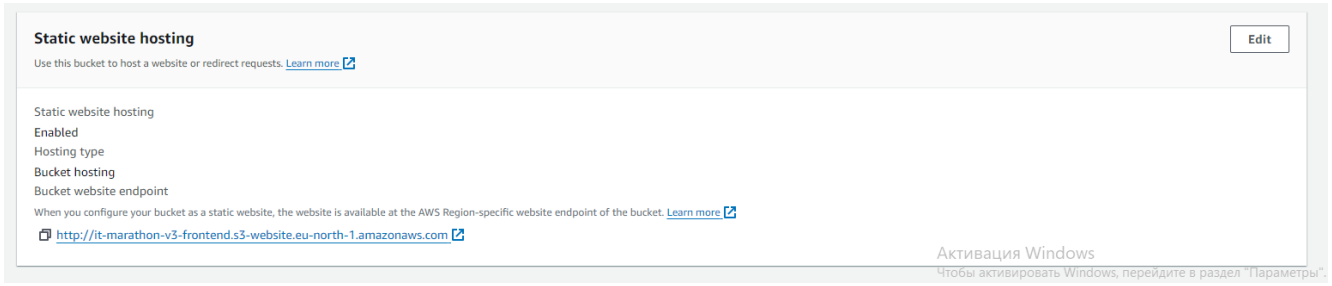


Рис. 2.12. “Static website hosting”

2.6.4. Опис інтерфейсу користувача

Коли користувач відкриває сайт, відображається головна сторінка. На цій сторінці можно переглядати список інших зареєстрованих користувачів, змінювати фільтри пошуку. Також гість може перейти на сторінку логіна/реєстрації, а зареєстрований користувач перейти на свій особистий акаунт або почати текстовий чат з іншим зареєстрованим користувачем.

Основні сторінки зображено на рис. 2.13. – 2.23.

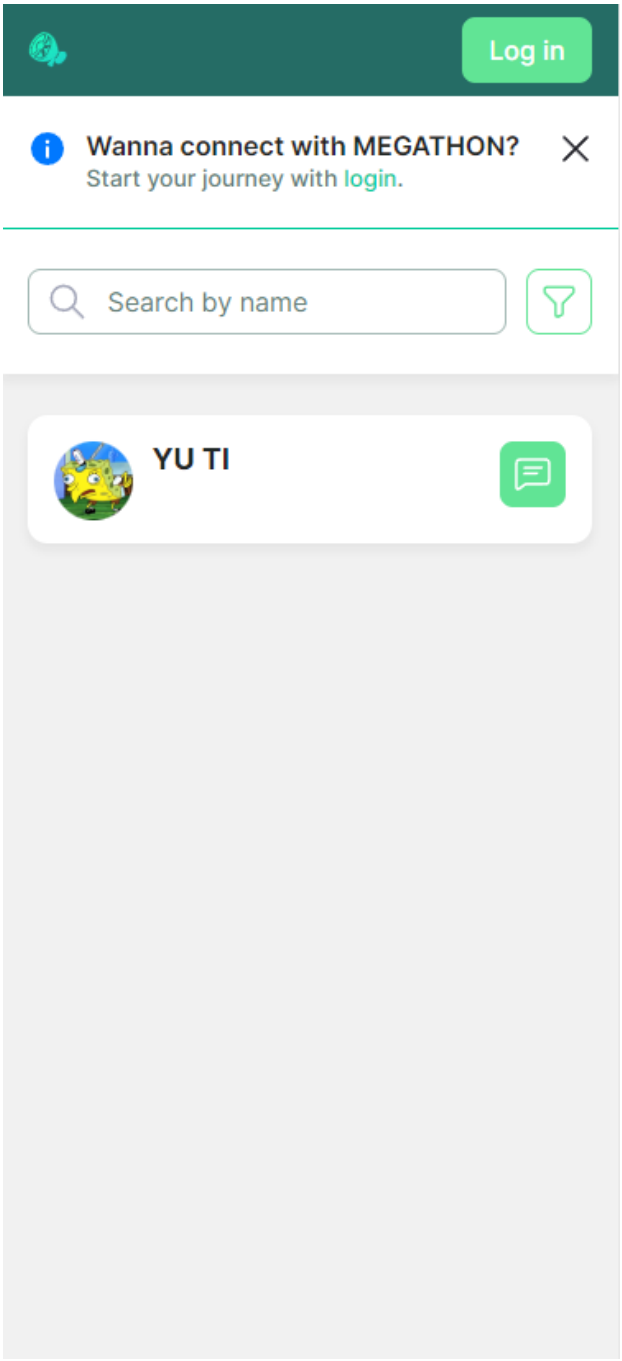


Рис. 2.13. Головна сторінка

При натисканні на посилання "Log in" користувача перенаправляє на сторінку входу. На цій сторінці можна натиснути "Sign Up" щоб перейти на сторінку реєстрації. На рис. 2.14 – 2.19 представлені сторінки входу та реєстрації.

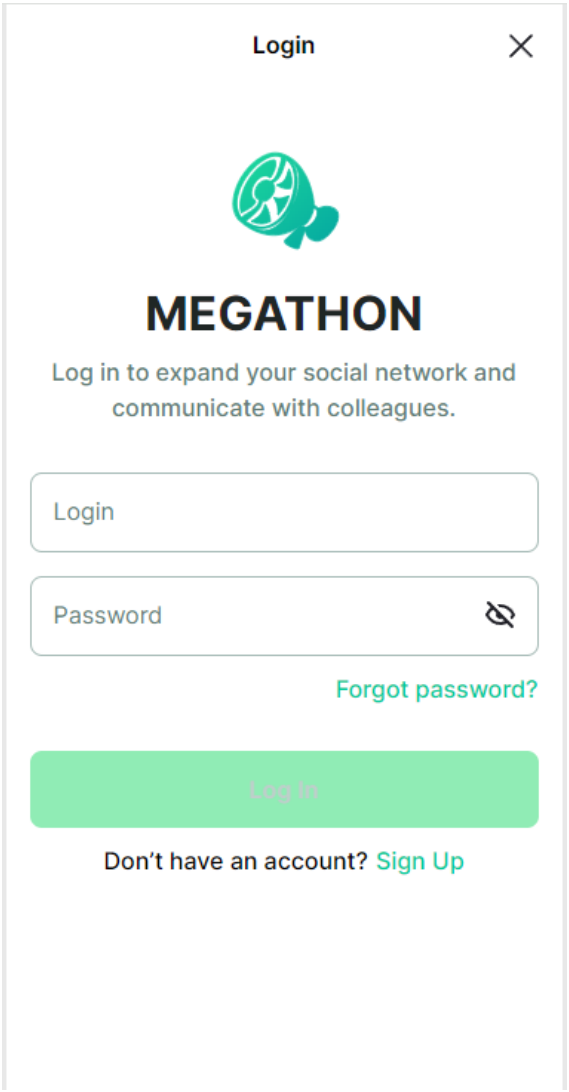
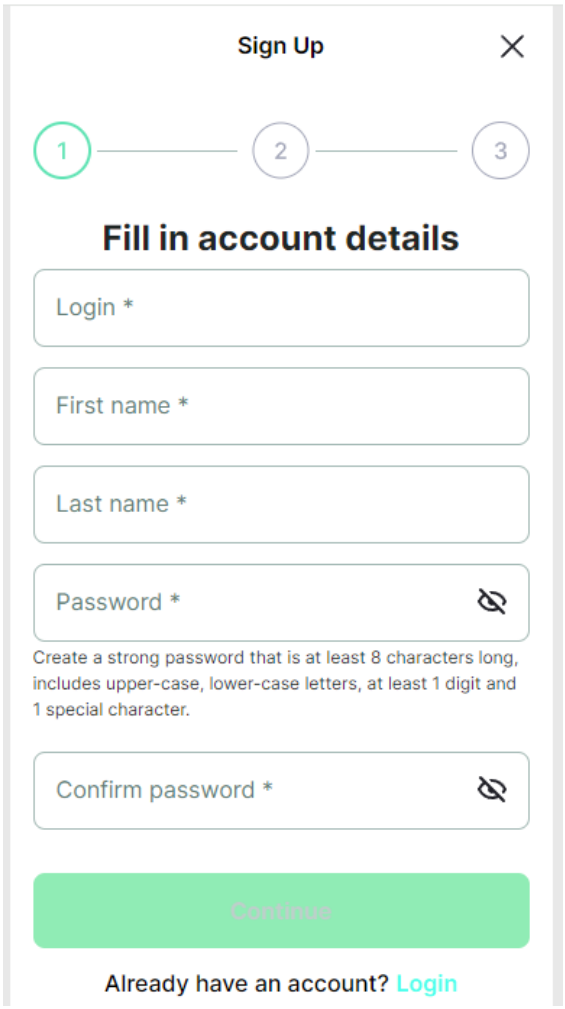


Рис. 2.14. Сторінка входу



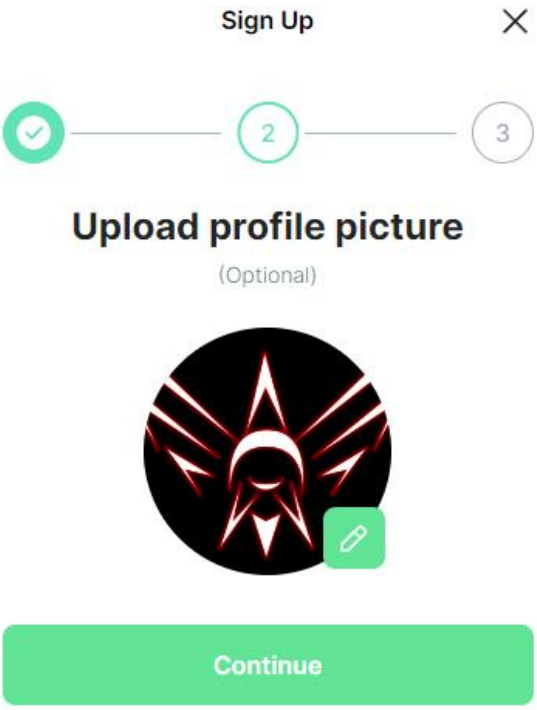


Рис. 2.15.- 2.16. Сторінки реєстрації ім’я, паролю та аватару

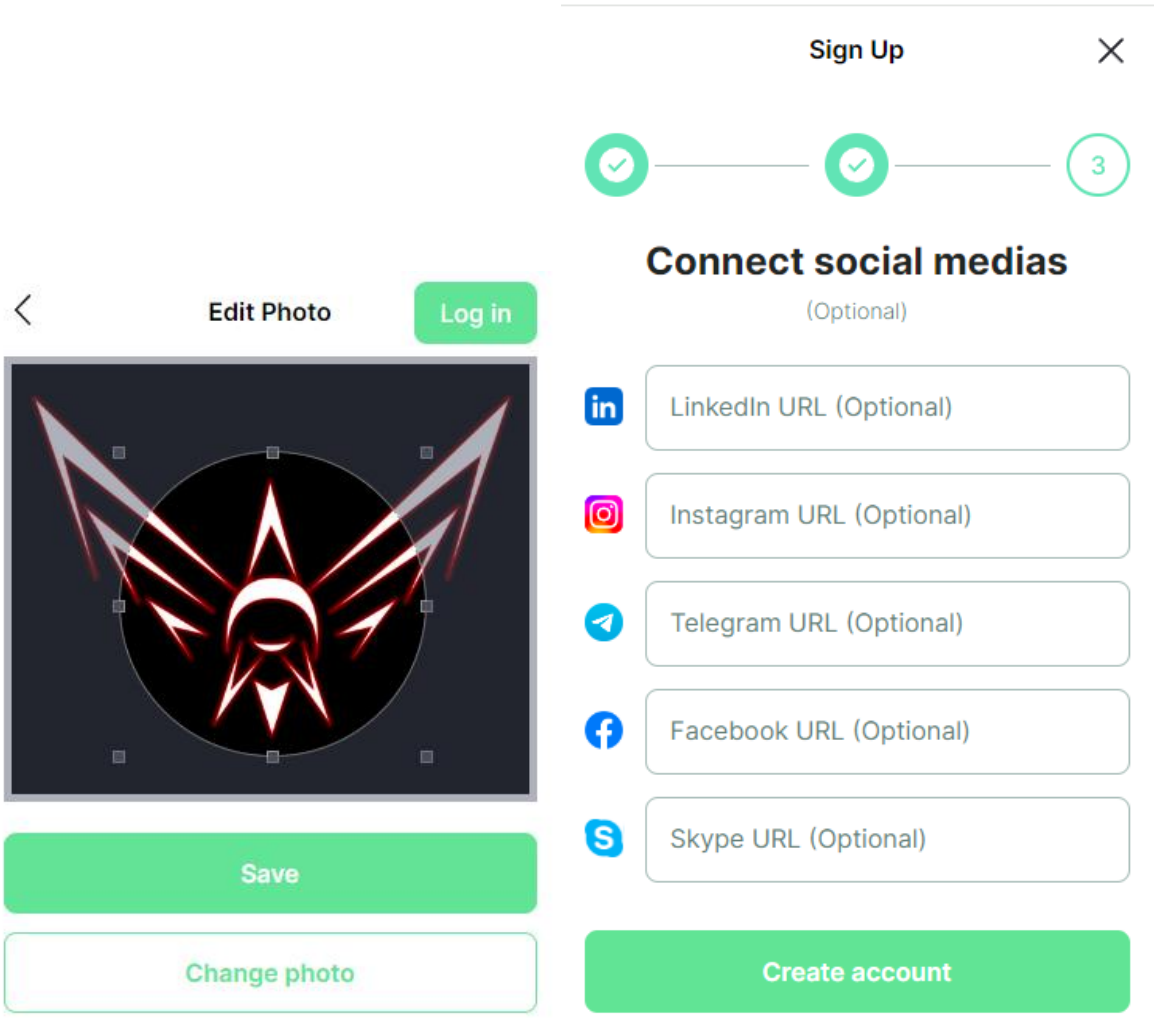


Рис. 2.17.- 2.18. Сторінка реєстрації соціальних мереж та редагування аватару

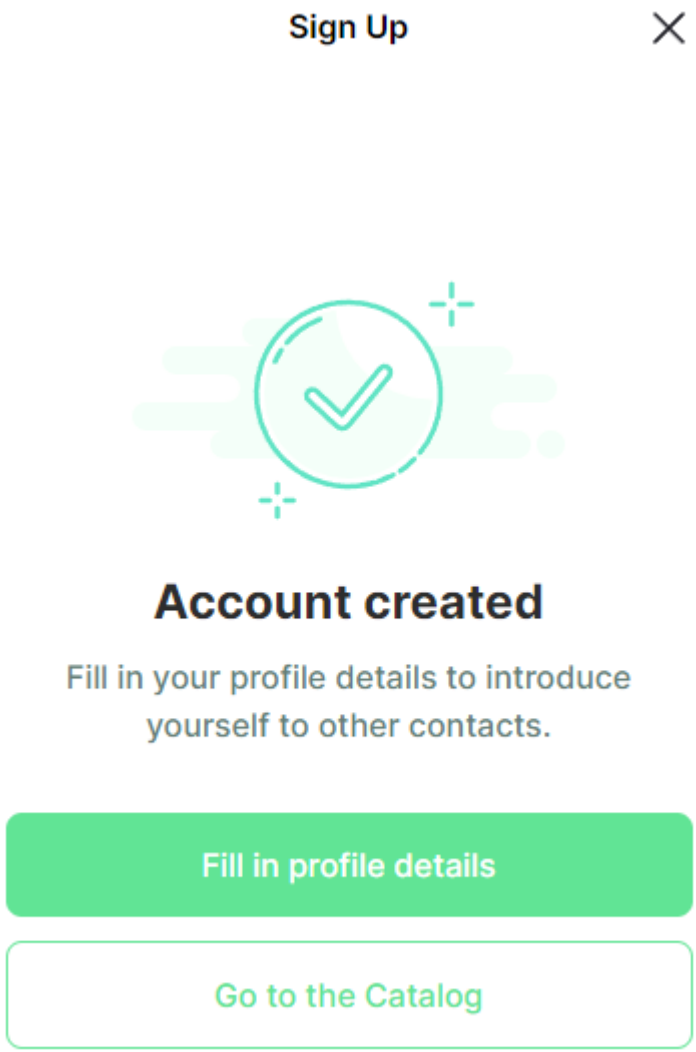


Рис. 2.19. Сторінка сповіщаюча про успішну реєстрацію

При натисканні на кнопку фільтрів можна визначити по яким параметрам буде проводитися пошук користувачів. З варіантів можна вибрати по місцю проживання або по приблизному віку. На рис. 2.20 представлено спливаючу вкладку.

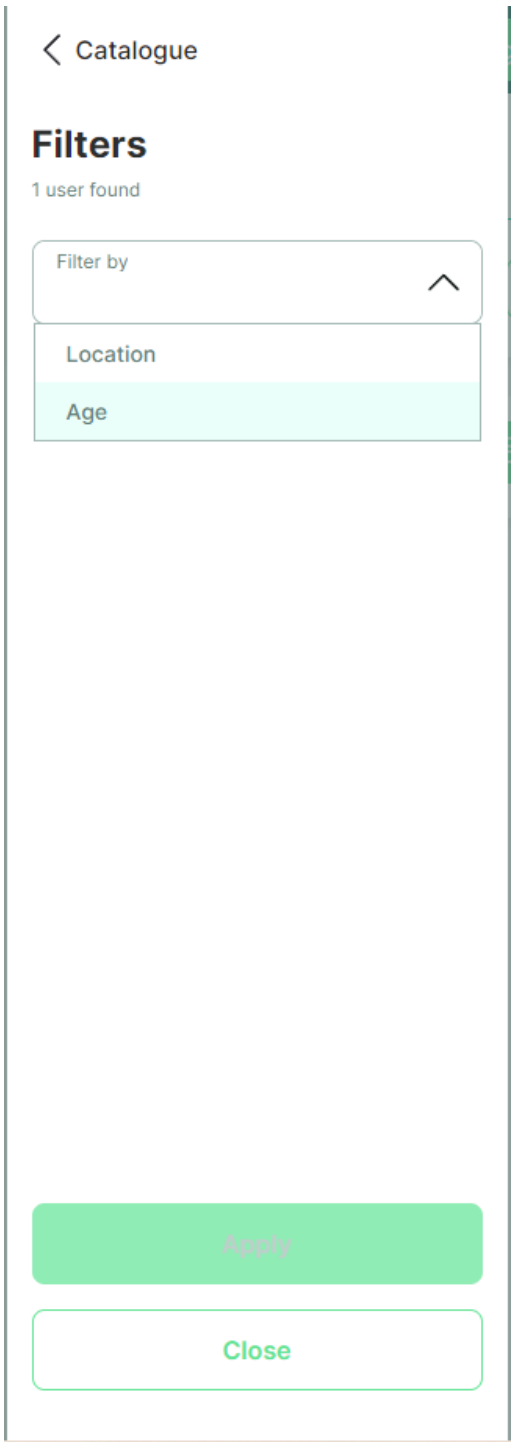


Рис. 2.20. Вкладинка фільтрування користувачів

При натисканні на аватар користувача з’являються опції продивитися свій профіль (Рис. 2.21), меню налаштуваннь (Рис. 2.22) та вихід.

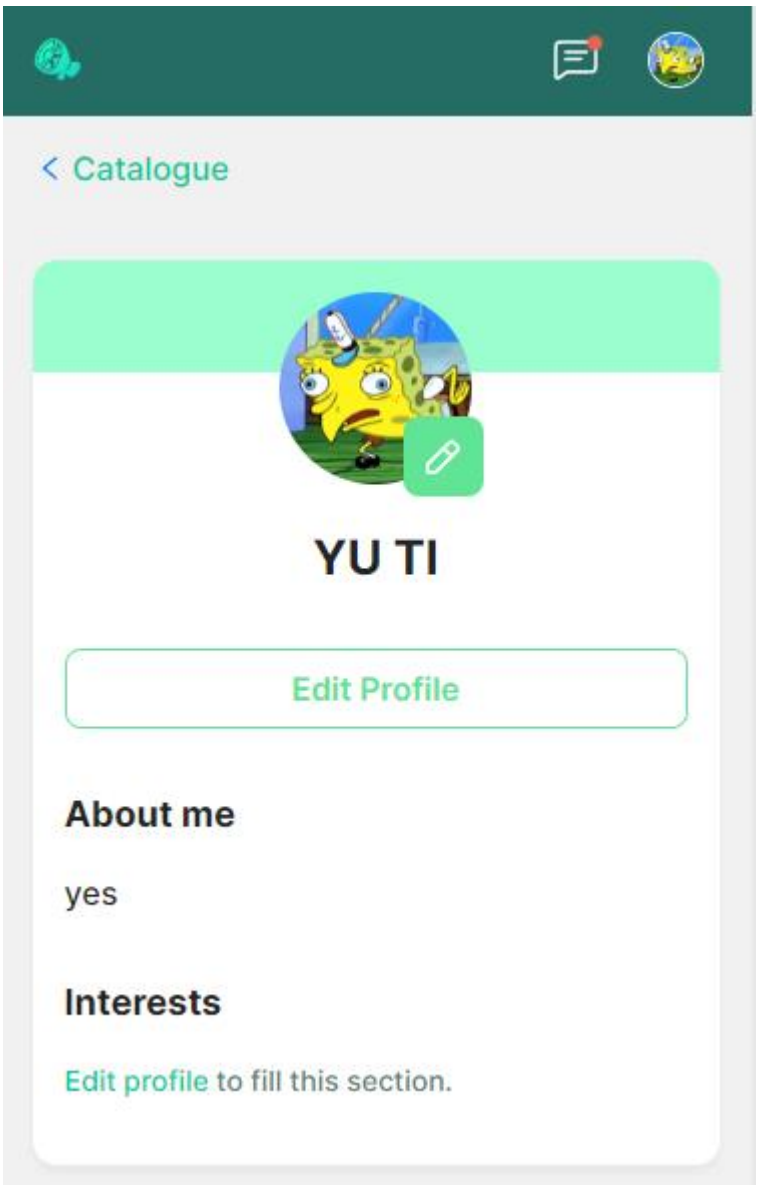


Рис. 2.21. Сторінка користувача

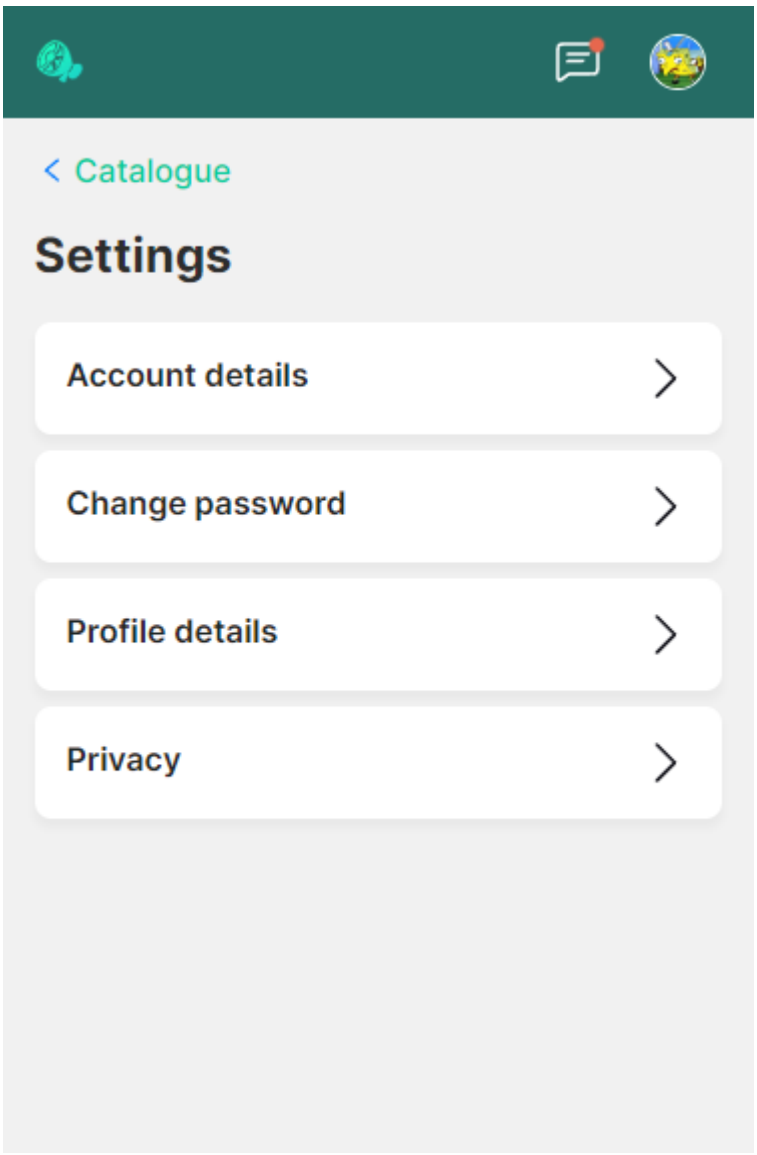
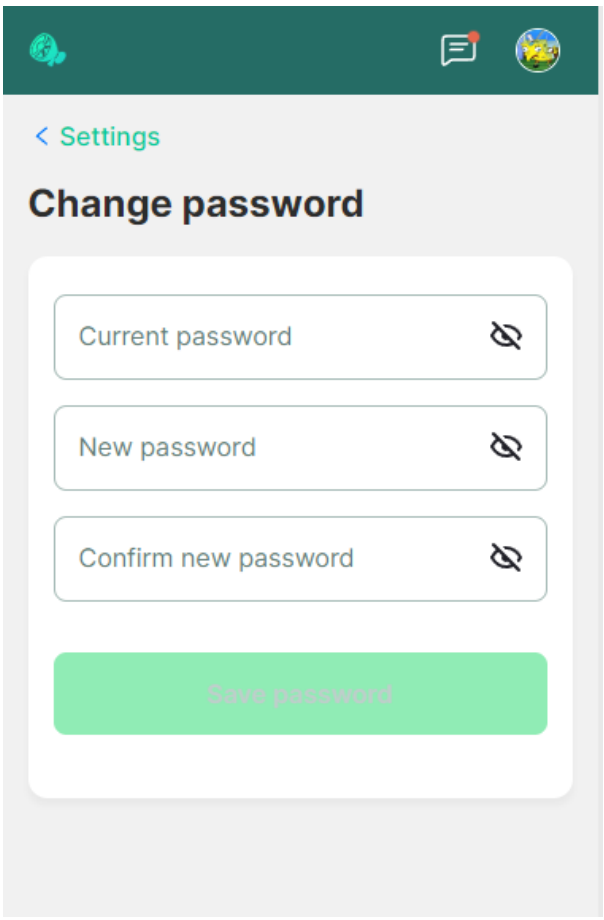


Рис. 2.22. Сторінка налаштувань користувача

У меню налаштувань користувача є опції змінити ім’я, логін(Рис. 2.24) і пароль(Рис. 2.23), деталі профілю(Рис. 2.25) та приватність(Рис. 2.26).



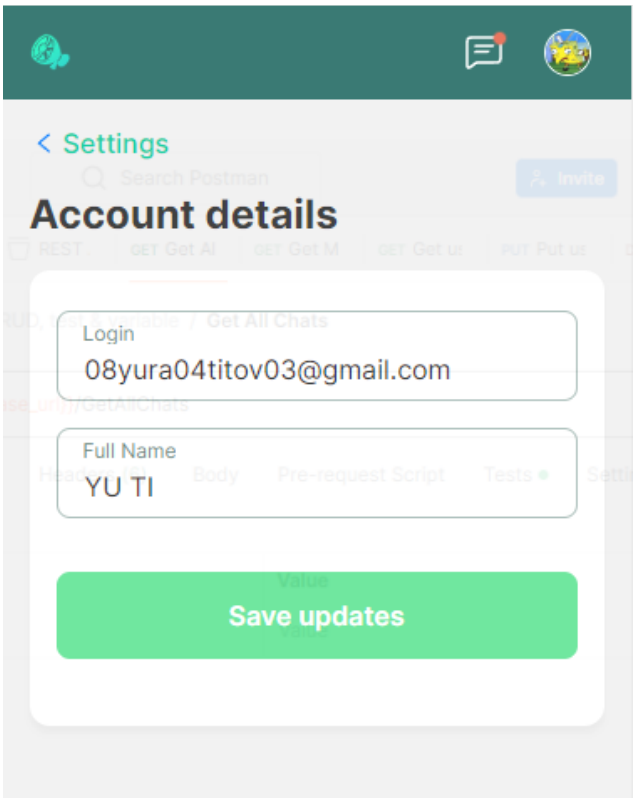


Рис. 2.23.- 2.24. Сторінка реєстрації соціальних мереж та редагування аватару

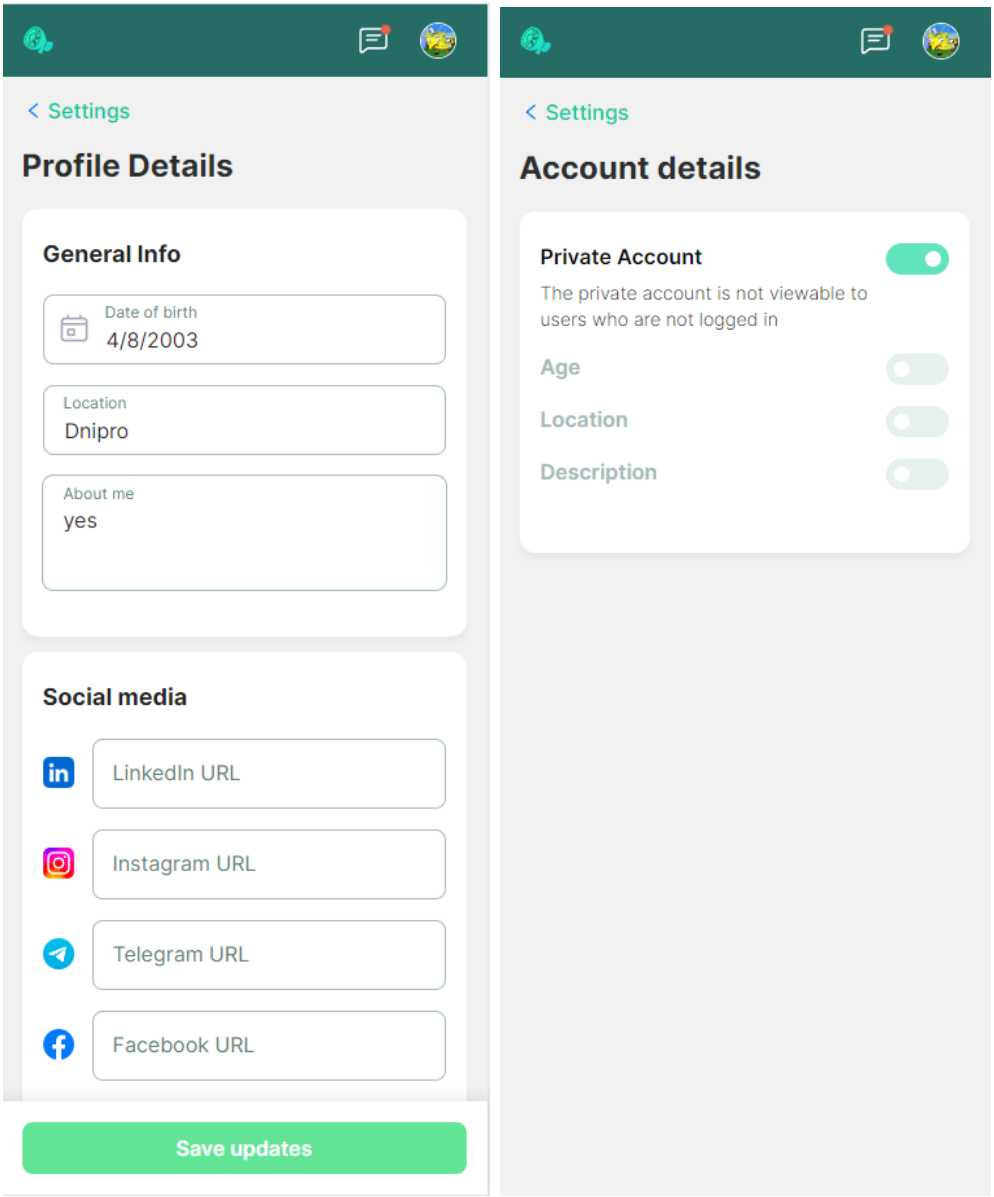


Рис. 2.25.- 2.26. Сторінка реєстрації соціальних мереж та редагування аватару

При виборі співрозмовника відкривається вікно з інформацією про вибраного користувача та кнопкою для початку розмови (Рис. 2.27). Приклад діалогового вікна чату наведено на рис. 2.28.

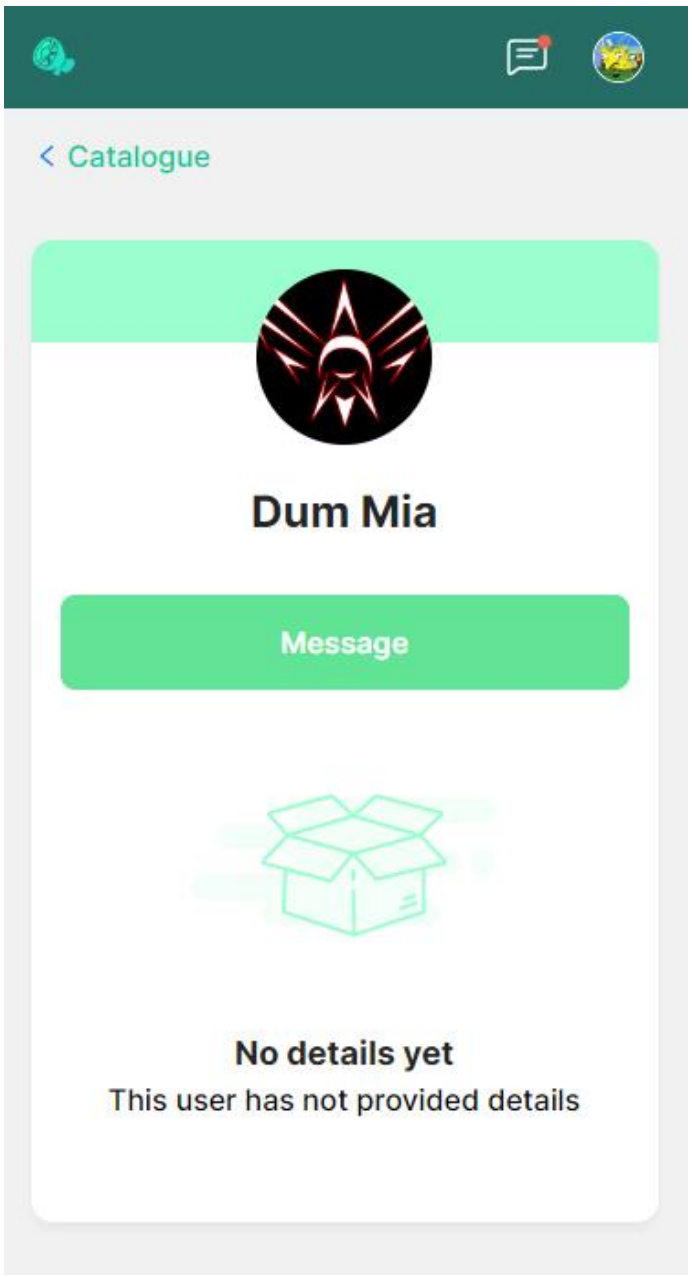


Рис. 2.27. Сторінка інформації про користувача

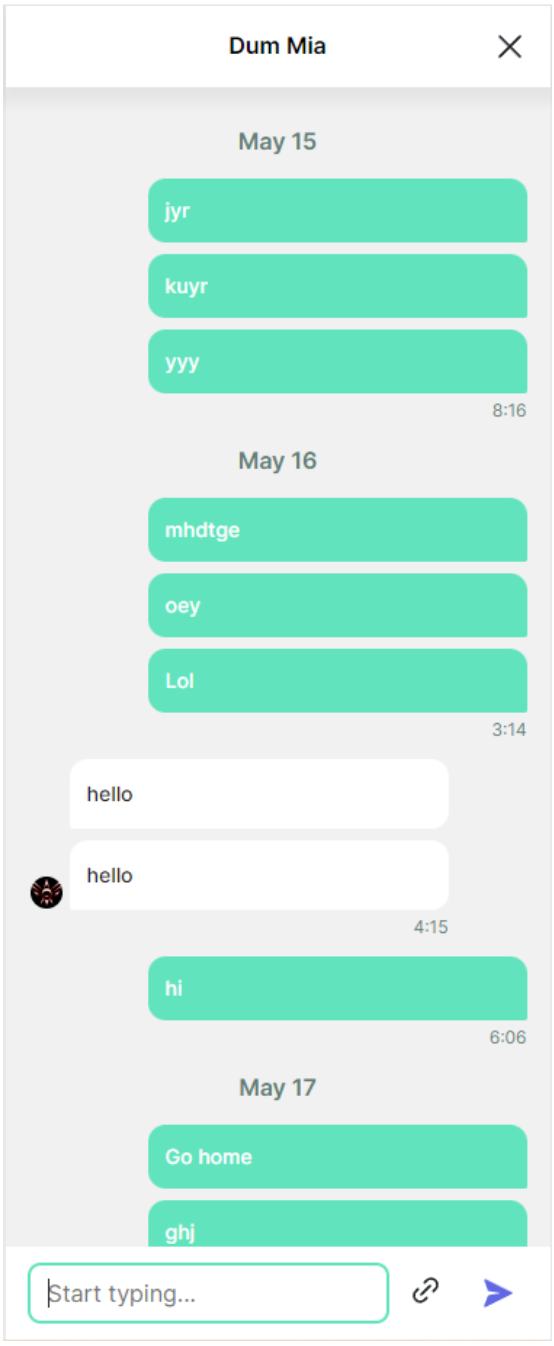


Рис. 2.28. Сторінка чату

При натисканні на кнопку emoji відкривається діалогове вікно вибору різних emoji(Рис. 2.29.). При натисканні на emoji воно додається до текстового поля вводу повідомлення. Присутнє також поле пошуку яке надає більш швидкий спосіб пошуку потрібних emoji.

Рис. 2.29. Вікно вибору emoji

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного забезпечення

Вихідні дані:

1. передбачуване число операторів програми – 2318;
2. коефіцієнт складності програми – 1,25;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата розробника – 235,58 грн/год; [6]

Відповідно до інформації на сайті jooble.org (https://ua.jooble.org/) і провівши аналіз даних, можна зробити висновок, що середня місячна заробітна плата web developer становить 1000,00 дол.США. На момент написання цієї записки, Офіційний курс гривні Національного банку України щодо долара становить 39,5771 гривень за 1 Долар США [11](див. рис. 3.1.). Тобто, при заробітній платі в розмірі 1000 дол.США, це еквівалентно 39 577,10 грн. на місяць. Робочий графік розробника в середньому складається з 21 робочого дня по 8 годин на день. Отже, погодинна ставка становитиме (39577,10 грн / 21 робочий день) / 8 годин = 235,58 грн. на годину.

Код цифровий	Код літерний	Кількість одиниць валюти	Назва валюти	Офіційний курс
840	USD	1	Долар США	39,5771

Рис. 3.1. Курс долар/гривня

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;
7. вартість машино-години ЕОМ – 0,69 грн/год.

При написанні кваліфікаційної роботи додаткове обладнання та приміщення не використовувалося.

Для розрахунку вартості машино-годин ПОЕМ були враховані витрати на електроенергію та домашній інтернет. Згідно з даними Мінфіну на момент написання роботи тариф на електроенергію для населення становила 2,64 грн/кВт-год. Домашній інтернет коштував 80,00 грн. на місяць. Ноутбук споживав 80 Вт, виходячи з цього, витрати на електроенергію за місяць використання в робочий час становила 80 * 2,64 * 8 * 21 / 1000 = 35,48 гривень. Таким чином, вартість машино-години комп'ютера становила (80,00 + 35,48) / 168 = 0,69 грн.

Беручи до уваги, творчий характер роботи web розробника, нормування праці в процесі розробки програмного забезпечення не можна розглядати стандартно. Тому, трудомісткість розробки сайту може бути розрахована з використанням системи моделей, що пропонують різні рівні точності оцінки. Трудомісткість розробки web сайту може бути визначена за наступною формулою:

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д \text{, людино-годин,}$$

(3.1)

де t_o - витрати праці на підготовку та опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p)$$

(3.2)

де q - передбачуване число операторів;

С - коефіцієнт складності програми;

р - коефіцієнт кореляції програми в ході її розробки.

Після підстановки відповідних значень в формулу (3.2) умовне число операторів дорівнює:

$$Q = 2318 * 1,25* (1 + 0,05) = 3042,38$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_и = \frac{Q*B}{(75..85)*k}, \text{ людино-годин,}$$

(3.3)

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності, стаж роботи 1 рік, тому коефіцієнт кваліфікації програміста = 1,1.

Будемо вважати збільшення витрат праці внаслідок недостатнього опису завдання як 1,2 (B = 1,2).

Після підставлення значень в формулу (3.3) маємо:

$$t_u = (3042,38 *1,2) / (85* 1,1) = 39,05 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25)*k}, \text{ людино-годин.}$$

(3.4)

Підставивши значення в формулу (3.4):

$$t_a = 3042,38 / (25 * 1,1) = 110,63 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі розраховується за формулою:

$$t_{\text{п}} = \frac{Q}{(20..25)*k}$$, людино-годин.

(3.5)

Після підставлення значень в формулу (3.5) отримуємо витрати на складання програми по готовій блок-схемі:

$t_п$ = 3042,38 / (25 * 1,1) = 110,63 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)*k}$$, людино-годин.

(3.6)

Підставивши значення в формулу (3.6):

$t_{отл}$ = 3042,38 / (5 * 1,1) = 553,16 людино-годин.

– за умови комплексного налагодження завдання:

$$t^{k}_{\text{отл}} = 1{,}5 * t_{\text{отл}}$$, людино-годин.

(3.7)

Підставивши значення в формулу (3.7):

$t^k_{отл}$ = 1,5 * 553,16 = 829,74 людино-годин.

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}$$, людино-годин,

(3.8)

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{\text{др}} = \frac{Q}{(15..20)*k}$$, людино-годин.

(3.9)

Підставивши значення в формулу (3.8):

$t_{др}$ = 3042,38 / (20 *1,1) = 138,29 людино-годин.

$T_{до}$ - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 * t_{др}, \text{ людино-годин.}$$

(3.10)

Підставивши значення в формулу (3.9):

$t_{до}$ = 0,75 · 138,29 = 103,72 людино-годин.

Виходить що витрати праці на підготовку документації:

$t_{д}$ = 138,29 + 103,72 = 242,01 людино-годин.

Отже підставивши всі найдені значення у формулу (3.1) маємо:

t = 50 + 39,05 + 110,63 + 110,63 + 553,16 + 242,01 = 1105,48 людино-години.

3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення web сайту Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{грн}$$

(3.11)

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{грн}$$

(3.12)

де: t - загальна трудомісткість, людино-годин;

$С_{пр}$ - середня годинна заробітна плата програміста, грн/година

Середня плата за одну годинну роботи web розробника становить 235,58 грн., тому:

$$З_{ЗП} = 1105,48 * 235,58 = 260428,12 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} * С_{мч}, \text{грн}$$

(3.13)

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (0,69 грн/год).

Підставивши значення у формулу (3.13):

$$З_{мв} = 553,16 * 0,69 = 381,68 \text{ грн.}$$

Отже витрати на створення програмного продукту будуть складати:

$$К_{ПО} = 260428,12 + 381,68 = 260809,80 \text{ грн.}$$

Формула для розрахунку очікуваного періоду створення ПЗ:

$$Т = \frac{t}{B_k * F_p}, \text{міс}$$

(3.14)

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Очікуваний період створення ПЗ:

$$Т = 1105,48 / 176 \approx 6,28 \text{ міс.}$$

Висновок: Остаточні витрати на розробку цього програмного продукту становлять 260809,80 гривень, що є мінімальною вартістю. Розрахунки також показують, що термін створення становить 6,28 місяців. Цей період містить час на тестування, приведення у відповідність завдань та ін.

ВИСНОВКИ

Під час виконання даної роботи було розроблено вебплатформу для спілкування студентів і молодих людей на основі спільних інтересів. Платформу реалізовано на основі хмарних технологій AWS Cloud з використанням мов програмування Java, Python та .NET.

Додаток створений за допомогою HTML5/CSS3 та мов програмування JavaScript та TypeScript.

Відповідно до завдання платформа реалізує такий функціонал:

- має унікальне посилання у мережі інтернет;
- забезпечує можливість користувачам з та без аккаунтів проглядати список вже зареєстрованих користувачів;
- забезпечує можливість зареєстрованим користувачам почати текстову бесіду з іншими зареєстрованими користувачами зі списку;
- забезпечує можливість ховати та розповсюджувати свою приватну інформацію;
- забезпечує можливість фільтрувати, сортувати та шукати список зареєстрованих користувачів.

Працездатність даного програмного продукту підтверджується вдалими експлуатаційними випробуваннями.

База даних Amazon DynamoDB забезпечує швидкий і надійний пошук необхідних записів за ключовими словами.

Під час виконання даної кваліфікаційної роботи також було встановлено складність розробленої системи і здійснено розрахунок вартості роботи зі створення програми та її підтримки у робочому стані, використовуючи вартість експлуатації сервісів хмарної платформи AWS Cloud та середню заробітну плату DevOps розробника.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Створення безсерверного інтернет-додатку [Інтернет ресурс] URL: https://aws.amazon.com/ru/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/# (дата звернення: 07.04.2024.)

2. Правова природа сторінок у соціальних мережах. [Інтернет ресурс] URL: https://yur-gazeta.com/publications/practice/zahist-intelektualnoyi-vlasnosti-avtorske-pravo/pravova-priroda-storinok-u-socialnih-merezhah.html (дата звернення: 23.02.2024.)

3. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

4. Розробка Lambda функцій на Java [Інтернет ресурс] URL: https://docs.aws.amazon.com/lambda/latest/dg/lambda-java.html (дата звернення: 07.04.2024.)

5. Розробка Lambda функцій на C# [Інтернет ресурс] URL: https://docs.aws.amazon.com/lambda/latest/dg/lambda-csharp.html (дата звернення: 07.04.2024.)

6. Програміст: середня зарплата у Дніпро https://ua.jooble.org/salary/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%96%D1%81%D1%82/%D0%94%D0%BD%D1%96%D0%BF%D1%80%D0%BE#hourly

7. WebSocketSubject: документація rxjs/webSocket [Інтернет ресурс] URL: https://rxjs.dev/api/webSocket/webSocket (дата звернення: 13.05.2024.)

8. Get started with GitLab CI/CD [Інтернет ресурс] URL: https://docs.gitlab.com/ee/ci/ (дата звернення: 13.05.2024.)

9. Figma документація з розробки дизайну [Інтернет ресурс] URL: https://help.figma.com/hc/en-us/categories/360002051613 (дата звернення: 13.05.2024.)

10. Introduction to the Angular docs [Інтернет ресурс] URL: https://angular.io/docs (дата звернення: 13.05.2024.)

11. Національний банк України, Офіційний курс гривні щодо іноземних валют [Інтернет ресурс] URL: https://bank.gov.ua/ua/markets/exchangerates (дата звернення: 16.05.2024.)

12. Doug Lowe. Java All-in-One For Dummies 7th Edition, 2023.

13. Stefan Baumgartner. TypeScript Cookbook, 2023.

14. Elisabeth Robson & Eric Freeman. Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages. " O'Reilly Media, Inc.", 2012.

15. Mark Wilkins. Learning Amazon Web Services (AWS): A Hands-On Guide to the Fundamentals of AWS Cloud. Addison-Wesley, 2019.

16. ANDREW TROELSEN & Philip Japikse. C# 6.0 and the .NET 4.6 Framework. Apress, 2016.

17. Eric Matthes. Python Crash Course. No Starch Press, 2016.

18. Tanmoy Sarkar. Building Modern Serverless Web APIs. 2021.

19. Salvatore Agostino Romeo PhD. Design, UX, and Mockups for Frontend Developers: The Definitive Guide: Learn the techniques to create any mockup in Figma. Master the essential design and UX rules through examples and use cases. 2022.

20. Jerry N. P. AWS Command Line Interface: Easy Guide on AWS CLI. 2016.

21. Jason Beaird, Alex Walker, and James George. Principles of Beautiful Web Design. SitePoint, 2020.

22. @ctrl/ngx-emoji-mart [Інтернет ресурс] URL: https://www.npmjs.com/package/@ctrl/ngx-emoji-mart (дата звернення: 24.05.2024.)

23. Amazon DynamoDB features [Інтернет ресурс] URL: https://aws.amazon.com/dynamodb/features/?nc1=h_ls (дата звернення: 21.05.2024.)

24. What is Amazon API Gateway? [Інтернет ресурс] URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html (дата звернення: 21.05.2024.)

25. Порівняння сучасних рішень для налаштування CI та CD [Інтернет ресурс] URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/294ada88-adcd-46ed-af79-bdadf9c05a4a/content (дата звернення: 21.05.2024.)

26. Jenkins VS Gitlab CI/CD [Інтернет ресурс] URL: https://itedu.center/ua/blog/comparisons/jenkins-vs-gitlab-ci-cd/ (дата звернення: 21.05.2024.)

27. Що таке Figma: функції, інструменти та переваги [Інтернет ресурс] URL: https://wezom.academy/ua/chto-takoe-figma-funktsii-instrumenty-ipreimuschestva/#:~:text=Figma%20%2D%20%D1%86%D0%B5%D1%85%D0%BC%D0%B0%D1%80%D0%BD%D0%B8%D0%B9%20%D0%B1%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%BE%D0%B2%D0%B8%D0%B9%20%D1%81%D0%B5%D1%80%D0%B2%D1%96%D1%81,%D0%BE%D0%B4%D0%BD%D0%B5%20%D0%B7%20%D0%B2%D0%B0%D0%B6%D0%BB%D0%B8%D0%B2%D0%B8%D1%85%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D0%B0%D0%B3%20%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B8. (дата звернення: 21.05.2024.)

28. RxJS Overview [Інтернет ресурс] URL: https://rxjs.dev/guide/overview (дата звернення: 21.05.2024.)

ДОДАТОК А

КОД ПРОГРАМИ

chat.component.ts

```
import { AfterViewInit, Component, OnInit, ViewChild, ElementRef } from '@angular/core';
import { CommonModule, NgOptimizedImage, ViewportScroller } from '@angular/common';
import { FormBuilder, FormControl, FormGroup, ReactiveFormsModule } from '@angular/forms';
import { Title } from '@angular/platform-browser';
import { Router } from '@angular/router';

import { ChatForm, ExpandedUserDetailed, Message, User, ChatProper } from '../../main.model';
import { APP_ROUTER_NAME, BUTTON_THEMES, ICON_NAMES, INPUT_PLACEHOLDERS } from
'../../../app.config';
import { HeaderService } from '../../../services/header.service';
import { EpmChatMessageComponent } from '../epm-chat-message/epm-chat-message.component';
import { EpmInputComponent } from '../../../shared/components/epm-input/epm-input.component';
import { EpmButtonComponent } from '../../../shared/components/epm-button/epm-button.component';
import { ToggleDisplayMessageTimePipe } from '../../../shared/pipes/toggle-display-message-time.pipe';
import { ToggleDisplayChatStartDayPipe } from '../../../shared/pipes/toggle-display-chat-start-day.pipe';
import { ChatStartDayPipe } from '../../../shared/pipes/chat-start-day.pipe';
import { CHAT_MOCK, USERS_MOCK } from '../../../../mocks/mock-data';
import { SetUserAvatarPipe } from '../../../shared/pipes/set-user-avatar.pipe';
import { WsService } from '../../../services/ws.service';
import { ApiService } from '../../../services/api.service';
import { take } from 'rxjs';
import { MainApiService } from '../../services/main-api.service';
import { CommunicationService } from 'src/app/services/communication.service';
import { PickerComponent } from '@ctrl/ngx-emoji-mart'

@Component({
  selector: 'epm-chat',
  standalone: true,
  imports: [
    CommonModule,
    EpmChatMessageComponent,
    EpmInputComponent,
    EpmButtonComponent,
    ReactiveFormsModule,
    NgOptimizedImage,
    ToggleDisplayMessageTimePipe,
    ToggleDisplayChatStartDayPipe,
    ChatStartDayPipe,
    SetUserAvatarPipe,
    PickerComponent
  ],
  templateUrl: './chat.component.html',
  schemas:[],
  styleUrls: ['./chat.component.scss']
})
export class ChatComponent implements OnInit, AfterViewInit {
  @ViewChild('messageInput') messageInputRef!: EpmInputComponent;

  receiver: ExpandedUserDetailed = window.history.state.receiver;
  chatForm!: FormGroup<ChatForm>;

  userId = "";
  chatId = "";


  messages: Message[] = [];
```

```
interlocutor: User = this.receiver;



readonly buttonThemes: typeof BUTTON_THEMES = BUTTON_THEMES;
readonly iconNames: typeof ICON_NAMES = ICON_NAMES;
readonly inputPlaceholders: typeof INPUT_PLACEHOLDERS = INPUT_PLACEHOLDERS;

get message(): FormControl {
  return this.chatForm.controls.message;
}

constructor(
  private title: Title,
  private router: Router,
  private headerService: HeaderService,
  private fb: FormBuilder,
  private wsService: WsService,
  private apiService: ApiService,
  private mainApiService: MainApiService,
  private communicationService: CommunicationService,
  private viewportScroller: ViewportScroller,
  private elementRef: ElementRef
) {}

ngOnInit(): void {
  this.initForm();
  this.userId = this.receiver.email;
  console.log("Receiver: "+this.userId);

  this.getChat();
  this.communicationService.getAction().subscribe(action => {
    this.updateChat();
  });


  const currentUser = this.apiService.currentUser();
  let currentUserEmail = "";
  if(currentUser != null){
    currentUserEmail = currentUser.email;
  }



  this.wsService.connect(currentUserEmail);
  this.wsService.messages$.subscribe((message: any) => {
    console.log(message);
  });
}

ngAfterViewInit(): void {
  this.focusOnInput();
  setTimeout(() => this.setTitles(), 0);
  setTimeout(() =>this.scrollToBottomList(), 500);
  setTimeout(() =>this.scrollToBottomList(), 500);
  setTimeout(() =>this.scrollToBottomList(), 500);
}

getChat():void{
  const currentUser = this.apiService.currentUser();
  let currentUserEmail = "";
  if(currentUser != null){
```

```
        currentUserEmail = currentUser.email;
      }

      let chatsList: ChatProper[] = [];
      this.mainApiService.getChatsList(currentUserEmail).pipe(take(1)).subscribe(
        (response: ChatProper[]) => {
          chatsList = response;
          console.log(chatsList);
          console.log("ChatCount: "+chatsList.length);
          const filteredChat = chatsList.filter(ch => ((ch.User2.Email == this.userId) || (ch.User1.Email == this.userId)))[0];
          console.log("filteredChat: "+filteredChat);
          if(filteredChat){
            this.chatId = filteredChat.ChatId;
            console.log("ChatId: "+this.chatId);
            this.updateChat();


          }
          this.scrollToBottomList();
        }
      );
    }

    updateChat(): void{
      this.mainApiService.getMessagesList(this.chatId).subscribe(response => {
            console.log(response);
            const upMessages: Message[] = [];
            response.forEach(function (msg) {
              const date = new Date(parseInt(msg.SentDtm, 10) * 1000);
              const formattedDate = date.toISOString().slice(0, 16);
              const resultMsg: Message = {
                id: "1",
                chatId: msg.Id,
                sender: msg.SenderId,
                time: formattedDate,
                text: msg.Message,
                read: msg.Seen
              };
              upMessages.push(resultMsg);
            });

            this.messages = upMessages;
            console.log(this.messages);
            this.scrollToBottomList();
          });

    }

    onSend(): void {
      const currentUser = this.apiService.currentUser();
      console.log("CurrentUser: "+currentUser);
      if (this.message.value && currentUser) {
        const requestBody = {
          SenderName: currentUser.name,
          SenderAvatar: currentUser.avatar,
          SenderId: currentUser.email,
          ReceiverId: this.receiver.email,
          Message: this.message.value
        };
        console.log(requestBody);
```

```
        this.wsService.sendWithResponse<any>(requestBody).subscribe(
          (response: any) => {
            console.log('WS response recieved');
            this.updateChat();
          },
          (error: any) => {
            console.error('WebSocket error:', error);
          }
        );
        const qAdd:Message = this.messages[0];
        qAdd.id="1";
        qAdd.chatId=this.chatId;
        qAdd.sender=requestBody.SenderId;
        qAdd.time=this.getCurrentTime();
        qAdd.text=requestBody.Message;
        qAdd.read=false;



        this.messages.push(qAdd);


        this.message.reset();
        this.focusOnInput();
      }
    }

    getCurrentTime(): string {
    const now = new Date();
    const year = now.getFullYear();
    const month = String(now.getMonth() + 1).padStart(2, '0');
    const day = String(now.getDate()).padStart(2, '0');
    const hours = String(now.getHours()).padStart(2, '0');
    const minutes = String(now.getMinutes()).padStart(2, '0');

    return `${year}-${month}-${day}T${hours}:${minutes}:00`;
  }

    onOpenUser(): void {

this.router.navigateByUrl(`${APP_ROUTER_NAME.Main}/${APP_ROUTER_NAME.Contact}/${this.receiver.email
}`);
    }

    trackByMessageId(_index: number, message: Message): string {
      return message.id;
    }

    private setTitles(): void {
      console.log("SetTitles: "+this.interlocutor.name);
      this.title.setTitle(`MEGATHON | ${this.interlocutor.name} chat`);
      this.headerService.addTitle(this.interlocutor.name);
    }

    private initForm(): void {
      this.chatForm = this.fb.group({
        message: new FormControl<string>('', { nonNullable: true })
      });
    }

    private scrollToBottom(): void {
      window.scrollTo(0, document.body.scrollHeight);
```

```
  }

  @ViewChild('listElement') list!: ElementRef<HTMLUListElement>;

  private scrollToBottomList(): void {
    const firstListItem = this.list.nativeElement.querySelector('li:first-child');

    firstListItem!.scrollIntoView({ behavior: 'smooth', block: 'end' });
  }

  private focusOnInput(): void {
    this.messageInputRef.epmInputRef.nativeElement.focus();
  }

  addEmoji(event: any) {
    const emoji = event.emoji.native;
    this.focusOnInput();
    this.chatForm.controls.message.setValue(this.message.value+""+emoji);
    console.log(this.message.value);
  }

  showEmoji: boolean = false;

  setShowEmoji(){
    this.showEmoji = !this.showEmoji;
  }

}
```

chat.component.html

```
<div class="chat__wrapper">
  <ul #listElement class="chat__list">
    <!-- TODO add logic to show it if websocket messaging that user tipes -->
    @if (false) {
    <li class="chat__list__element time__hidden">
      <div class="message__wrapper">
        <div class="avatar__wrapper"></div>
        <epm-chat-message [message]="true" />
      </div>
    </li>
    } @for (msg of messages; track trackByMessageId(i, msg); let i = $index) {

    <li
      class="chat__list__element"
      [ngClass]="{ time__hidden: (msg | toggleDisplayMessageTime: messages : i), own: msg.sender !== userId
}"
    >
      <div class="message__wrapper">
        <div class="avatar__wrapper">
          <a class="avatar__link" (click)="onOpenUser()">
            <img width="24" height="24" [src]="interlocutor.avatar | setUserAvatar" alt="avatar of the interlocutor"
/>
          </a>
        </div>
        <epm-chat-message [ngClass]="{ own: msg.sender !== userId }" [message]="msg" />
      </div>
      <span class="time">{{ msg.time | date: 'h:mm' }}</span>
    </li>
    @if (msg | toggleDisplayChatStartDay: messages : i) {
    <span class="chat__start-day">
      {{ msg.time | chatStartDay: 'LLLL d' }}
    </span>
```

```
        } }
      </ul>
    </div>
    @if (showEmoji){
      <div class="picker">
        <emoji-mart title="Pick your emoji…" [exclude]="['recent']" emoji="point_up" (emojiSelect)="addEmoji($event)"></emoji-mart>

        <!--<Picker data={data} onEmojiSelect="addEmoji($event)"></Picker>-->
      </div>
    }
    <form class="chat__form" [formGroup]="chatForm">
      <epm-input #messageInput class="chat__form_input" [inputPlaceholder]="inputPlaceholders.Chat" [control]="message" />

      <epm-button
        type="submit"
        class="chat__form_button"
        [class]="[buttonThemes.IconOnly, buttonThemes.Small]"
        [iconName]="iconNames.Plus"
        (click)="setShowEmoji()"
      />

      <epm-button
        type="submit"
        class="chat__form_button"
        [class]="[buttonThemes.IconOnly, buttonThemes.Small]"
        [iconName]="iconNames.SendMessage"
        (click)="onSend()"
      />
    </form>
```

main.model.ts

```
    import { FormControl, FormGroup } from '@angular/forms';
    import { RouterStateSnapshot, UrlTree } from '@angular/router';
    import { Observable } from 'rxjs';

    import { SOCIAL_ICONS } from '../app.config';


    export interface User {
      name: string;
      email: string;
      avatar: string;
    }

    export interface ChatLastMessage extends User {
      lastMessage: string;
      lastMessageDateTime: string;
    }

    export interface UserDetailed extends User {
      birthday: number | null;
      registration: number;
      about: string;
      interests: string[];
      location: string;
      socialMedia: SocialLink;
      privacy: UserPrivacy;
    }

    export interface SocialLink {
```

```
  instagram: string;
  linkedin: string;
  facebook: string;
  skype: string;
  telegram: string;
}

export interface UserPrivacy {
  description: boolean;
  location: boolean;
  account: boolean;
  age: boolean;
}

export interface DetailedSocialLink {
  priority: number;
  link: string;
  type: SOCIAL_ICONS;
}

export interface ExpandedUserDetailed extends Omit<UserDetailed, 'socialMedia'> {
  socialMedia: DetailedSocialLink[];
}

export type GetLink = (userName: string) => string;

export interface ChatForm {
  message: FormControl<string>;
}

export interface Message {
  id: string;
  chatId: string;
  sender: string;
  time: string;
  text: string;
  read: boolean;
}

export interface ReqestMessage {
  Id: string;
  Message: string;
  ReceiverId: string;
  Seen: boolean;
  SenderId: string;
  SentDtm: string;
}

export interface ChatProper {
    ChatId: string;
    UpdateDt: number;
    User1: {
        Email: string;
        Avatar: string;
        Name: string;
    };
    User2: {
        Email: string;
        Avatar: string;
        Name: string;
    };
}
```

```
export interface PagMessage {
  PaginationToken: string;
  Messages: ReqestMessage[];
}

type SimpleFormType<T, U> = {
  [key in keyof T]: FormControl<U>;
};

export interface AccountDetailsData {
  email: string;
  name: string;
}

export type AccountDetailsForm = SimpleFormType<AccountDetailsData, string>;

export interface ChangePasswordData {
  currentPassword: string;
  newPassword: string;
  confirmNewPassword: string;
}

export type ChangePasswordForm = SimpleFormType<ChangePasswordData, string>;

export interface SocialMedias {
  linkedin: FormControl<string>;
  instagram: FormControl<string>;
  telegram: FormControl<string>;
  facebook: FormControl<string>;
  skype: FormControl<string>;
}

export interface RegisterForm {
  email: FormControl<string>;
  givenName: FormControl<string>;
  familyName: FormControl<string>;
  password: FormControl<string>;
  repeatPassword: FormControl<string>;
  avatar: FormControl<string>;
  socialMedia: FormGroup<SocialMedias>;
}

export interface PrivacyItemsControlsData {
  age: boolean;
  location: boolean;
  description: boolean;
}

export interface PrivacyItemsControls {
  age: FormControl<boolean>;
  location: FormControl<boolean>;
  description: FormControl<boolean>;
}

export interface PrivacyStatusForm {
  account: FormControl<boolean>;
  privacyItems: FormGroup<PrivacyItemsControls>;
}

export interface ProfileDetailsForm {
  birthday: FormControl<string>;
```

```
  location: FormControl<string>;
  about: FormControl<string>;
  socialMedia: FormGroup<SocialMedias>;
 }

 export interface AgeRange {
  minAge: number;
  maxAge: number;
 }

 export interface CanComponentDeactivate {
  canDeactivate?: (nextState: RouterStateSnapshot) => Observable<boolean> | Promise<boolean> | boolean;
 }

 export type CanDeactivateType = Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean |
UrlTree;
```

api.service.ts

```
 import { Injectable, signal, Signal, WritableSignal} from '@angular/core';
 import { environment } from '../../environments/environment.prod';
 import { HttpClient, HttpHeaders} from '@angular/common/http';
 import { Observable, take, tap, map } from 'rxjs';
 import { UserDetailed, Message, ChatProper, PagMessage, ReqestMessage} from '../main/main.model';
 import { FilterQueryConfig } from '../main/main.config';

 const BASE_URL = environment.apiBaseUrl;

 @Injectable({
  providedIn: 'root'
 })
 export class ApiService {
  private readonly currentUser$: WritableSignal<UserDetailed | null> = signal(null);

  readonly currentUser: Signal<UserDetailed | null> = this.currentUser$.asReadonly();

  constructor(private http: HttpClient) {}

  headerDict = {
     'Content-Type': 'application/json'
  };



  getUserList(limit: number, hashKey: string): Observable<UserDetailed[]> {
    return
this.http.get<UserDetailed[]>(`${BASE_URL}/users?limit=${limit}&hashkey=${hashKey}`).pipe(take(1));
  }

  getFilteredUserList(query: FilterQueryConfig, limit: number, rangekey: string): Observable<UserDetailed[]>
{
    return      this.http.post<UserDetailed[]>(`${BASE_URL}/match?limit=${limit}&rangekey=${rangekey}`,
query).pipe(take(1));
  }

  getUser(id: string): Observable<UserDetailed> {
    return this.http.get<UserDetailed>(`${BASE_URL}/users/${id}`).pipe(take(1));
  }

  getCurrentUser(id: string): Observable<UserDetailed> {
    return this.getUser(id).pipe(
      tap(user => {
        this.currentUser$.set(user);
```

```
      })
    );
  }

  updateUser(id: string, userData: Partial<UserDetailed>): Observable<string> {
    return this.http.put<string>(`${BASE_URL}/users/${id}`, userData).pipe(take(1));
  }

  createUser(userData: UserDetailed): Observable<string> {
    return this.http.post<string>(`${BASE_URL}/users`, userData).pipe(take(1));
  }

  clearUserData(): void {
    this.currentUser$.set(null);
    sessionStorage.clear();
  }

  requestOptions = {
    headers: new HttpHeaders(this.headerDict)
  };

  getMessagesList(chatId: string): Observable<ReqestMessage[]> {
    return this.http.get<PagMessage>(`${BASE_URL}/GetMessages?chatId=${chatId}`).pipe(
      map(response => response.Messages),
      tap(response => {console.log("Actual response: "+response);})
    );
  }

  getChatsList(userId: string): Observable<ChatProper[]> {
    return this.http.get<ChatProper[]>(`${BASE_URL}/GetAllChats?userId=${userId}`).pipe(take(1));
  }
}
```

ws.service.ts

```
import { Injectable } from '@angular/core';
import { catchError, delayWhen, EMPTY, Observable, retryWhen, Subject, switchAll, tap, timer, map} from
'rxjs';
import { webSocket, WebSocketSubject } from 'rxjs/webSocket';
import { CommunicationService } from './communication.service';
import { environment } from '../../environments/environment';

const BASE_URL = environment.wsUrl;

@Injectable({
  providedIn: 'root'
})
export class WsService {
  readonly RECONNECT_INTERVAL = 2000;

  socket$!: WebSocketSubject<any>;
  private reactSocket$: WebSocket | null;

  constructor(
    private communicationService: CommunicationService
  ) {

    this.reactSocket$ = null;
  }

  private messagesSubject$: Subject<any> = new Subject();

  messages$ =  this.messagesSubject$.pipe(
```

```
    switchAll(),
    catchError(e => {
      throw e;
    })
  );

  connect(userId: string, config: { reconnect: boolean } = { reconnect: false }) {
    if (!this.reactSocket$ || this.reactSocket$.readyState !== WebSocket.OPEN) {
      this.reactSocket$ = new WebSocket(`${BASE_URL}?userId=${userId}`);
      this.reactSocket$.addEventListener('open', () => {console.log('ReactWebSocket connected succesfully');})
      this.reactSocket$.addEventListener('close', () => {console.log('ReactWebSocket connection ended succesfully'); this.connect(userId, { reconnect: true });})
      this.reactSocket$.addEventListener('message', () => {
        console.log('ReactWebSocket received message');
        this.communicationService.sendAction({ type: 'messageSent' });
      })
    }

    this.reactSocket$.onopen = () => {
      console.log('ReactWebSocket connection established.');
    };

    this.reactSocket$.onerror = (error) => {
      console.error('ReactWebSocket error:', error);
    };

    this.reactSocket$.onclose = (event) => {
      console.log('ReactWebSocket connection closed:', event.reason);
    };
  }

  reconnect(observable: Observable<any>) {
    return observable.pipe(
      retryWhen(errors =>
        errors.pipe(
          tap(val => console.log('[WS Service] Try to reconnect', val)),
          delayWhen(_ => timer(this.RECONNECT_INTERVAL))
        )
      )
    );
  }

  close() {
    this.socket$.complete();
    this.socket$ = null!;
    if (this.reactSocket$ && this.reactSocket$.readyState === WebSocket.OPEN) {
      this.reactSocket$.close();
    }
  }

  private responses = new Map<string, Observable<any>>();

  sendWithResponse<T>(msg: {
     SenderName: string;
     SenderAvatar: string;
     SenderId: string;
     ReceiverId: string;
     Message: string;}
    ): Observable<T> {

    console.log("Message to send: "+msg.Message);
    msg.SenderAvatar = "/mocks/img/cat.jpg";
```

```
  const body = JSON.stringify({
    action: 'send',
    SenderName: msg.SenderName,
    SenderAvatar: msg.SenderAvatar,
    SenderId: msg.SenderId,
    ReceiverId: msg.ReceiverId,
    Message: msg.Message
  });


  const id = Math.random().toString(36).substring(7);
  const responseSubject = new Subject<T>();
  this.responses.set(id, responseSubject);

  if (this.reactSocket$ && this.reactSocket$.readyState === WebSocket.OPEN) {

    this.reactSocket$.send(body);
    console.log("React message was sent");
  }

  return responseSubject.asObservable();
  }

 sendMessage(msg: {
  SenderName: string;
  SenderAvatar: string;
  SenderId: string;
  ReceiverId: string;
  Message: string;
 }){
  console.log("Message to send: "+msg.Message);
  msg.SenderAvatar = "/mocks/img/cat.jpg";

  const body = JSON.stringify({
    action: 'send',
    SenderName: msg.SenderName,
    SenderAvatar: msg.SenderAvatar,
    SenderId: msg.SenderId,
    ReceiverId: msg.ReceiverId,
    Message: msg.Message
  });
  if (this.reactSocket$ && this.reactSocket$.readyState === WebSocket.OPEN) {

    this.reactSocket$.send(body);
    console.log("React message was sent");
  }
 }

 private getSocket(userId: string) {
  console.log("UserId of a socket: "+userId);

  return webSocket({
    url: `${BASE_URL}?userId=${userId}`,
    openObserver: {
      next: () => {
        console.log('connection ok');
      },
      error: (err) => console.error('WebSocket open error:', err)
    },
    closeObserver: {
      next: () => {
```

```
        console.log('connection closed');
        this.socket$ = null!;
        this.connect(userId, { reconnect: true });
      }
    }
  });
 }
}
```

Решта коду додається окремо.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка програмного забезпечення мережі

на базі платформи AWS Cloud»

студента групи 122-20-1 Тітова Юрія Ярославовича

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Тітов Ю.Я.диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Тітов Ю.Я.диплом.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
Тітов Ю.Я.диплом.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Тітов Ю.Я.диплом.ppt	Презентація кваліфікаційної роботи