Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ПРОГРАМНОГО СЕРВІСУ ЕКСТРЕНОЇ ДОПОМОГИ. ЧАСТИНА 1. СЕРВЕРНА ЧАСТИНА.

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ратушняк А. В.
(прізвище та ініціали)
Керівник: доцент каф. КН
Белзецький Р. С.
(прізвище та ініціали)
«07» 06 2024 р.
Рецензент:
Барабан М.В.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"07" 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ратушняку Андрію Валерійовичу

1. Тема роботи: Розробка сервісу екстреної допомоги. Частина 1. Серверна частина.

Керівник роботи: асистент Белзецький Руслан Станіславович.
Затверджені наказом вищого навчального закладу "11" 03 2024 року № ___

2. Термін подання студентом роботи 27.05.2024 р.

3. Вихідні дані до роботи:
Хмарне середовище Amazon Web Services;
Мова програмування C#;
Використання об'єктно-орієнтованого програмування;
Середовище розробки Rider.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки сервісу екстреної допомоги; проектування; проектування сервісу екстреної допомоги; розробка серверної частини сервісу; тестування розроблених програми сервісу екстреної допомоги; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Схема системи розгортання додатку, діаграма архітектури сервісу, інтерфейс додатку Pixel Safety, інтерфейс додатку Life360.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
1-4	Белзецький Р.С.		

7. Дата видачі завдання 07.03.2024р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Пр
		початок	закінчення	
1	Обґрунтування доцільності розробки сервісу екстреної допомоги	06.05.24	10.05.24	
2	Проектування сервісу екстреної допомоги	11.05.24	16.05.24	
3	Розробка серверної частини	17.05.24	22.05.24	
4	Тестування розробленого сервісу екстреної допомоги	23.05.24	28.05.24	
5	Аналіз отриманих результатів та формування висновків	29.05.24	02.06.24	
6	Оформлення бакалаврської дипломної роботи	03.06.24	06.06.24	

Студент (підпис) **Ратушняк** (прізвище та ініці

Керівник роботи (підпис) **Белзецький** (прізвище та ініці

АНОТАЦІЯ

Ратушняк А. В. Розробка сервісу екстреної допомоги. Частина 1. Серверна частина. Бакалаврська дипломна робота складається з 66 сторінок формату А4, на яких є 30 рисунків, список використаних джерел містить 27 найменувань.

Метою даної бакалаврської кваліфікаційної роботи є дослідження застосування сучасних методів розробки програмного забезпечення з використанням хмарних технологій для покращення роботи та відмово стійкості додатку. Зокрема метою роботи є створення сервісу, що дозволить людям вчасно реагувати на надзвичайні події та надавати відповідну допомогу.

Результатом дипломного дослідження є сервер, що реалізований на мові програмування C# у програмному середовищі. А також хмарна інфраструктура для коректної роботи цього сервера

Ключові слова: мікросервіси, C#, ASP .NET, хмарні технології, AWS, Lambda.

ABSTRACT

Ratushnyak A. V. Development of an emergency service. Part 1. The server part. The bachelor thesis consists of 66 pages of A4 format, on which there are 30 pictures, the list of used sources contains 27 names.

The purpose of this bachelor's qualification work is to study the application of modern methods of software development using cloud technologies to improve the performance and fault tolerance of the application. In particular, the goal of the work is to create a service that will allow people to respond to emergency events in a timely manner and provide appropriate assistance.

The result of the diploma research is a server implemented in the C# programming language in a programming environment. And also, the cloud infrastructure for the correct operation of this server

Keywords: microservices, C#, ASP .NET, cloud technologies, AWS, Lambda.

ЗМІСТ

ВСТУП

Актуальність досліджень. Розробка сервісу екстреної допомоги має велику актуальність з огляду на зростаючі вимоги до швидкого і ефективного реагування на надзвичайні ситуації. Сучасні технології інформаційних систем дозволяють створювати потужні інструменти для координації служб екстреної допомоги, забезпечуючи тим самим зменшення часу реагування та підвищення якості надання допомоги. Нижче наведено основні аспекти, які підтверджують актуальність даного дослідження.

Використання сучасних технологій, таких як хмарні обчислення, IoT (Інтернет речей), мобільні додатки та великі дані (Big Data), дозволяє суттєво покращити функціональність та ефективність систем екстреної допомоги.

Однією з ключових задач є забезпечення безпеки та надійності передачі та зберігання даних в умовах екстреної ситуації. Це вимагає розробки надійних серверних рішень, здатних забезпечити захист конфіденційної інформації та безперебійне функціонування системи.

Метою дослідження є розширення функціональних можливостей сервісу екстреної допомоги шляхом створення головного програмного модулю та допоміжних модулів сповіщення.

Об’єктом дослідження є процес створення програмного засобу серверної частини для забезпечення виконання бізнес-логіки сервісу екстреної допомоги.

Предметом дослідження є програмний засіб у вигляді веб-серверу для забезпечення виконання бізнес-логіки сервісу екстреної допомоги.

Задачі дослідження:

1. дослідити методи та технології для створення серверної частини;
2. аргументувати важливість розробки сервісу екстренної допомоги;
3. розробити повну архітектуру серверної частини;
4. проаналізувати та пояснити вибір інструментів для розробки;
5. реалізувати модулі сповіщення, безперебійної комунікації в реальному часі, а також головний монолітний сервер;

6. провести тестування розробленого програмного коду.

Апробація результатів роботи

Результати досліджень було апробовано на LIII науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024) м. Вінниці у 2024 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ СЕРВІСУ ЕКСТРЕНОЇ ДОПОМОГИ

1.1 Аналіз предметної області

У сучасному світі стрімка урбанізація, зміни клімату та соціальні фактори створюють нові виклики для особистої безпеки та ефективного реагування на надзвичайні ситуації. У таких умовах використання мобільних застосунків для забезпечення безпеки, стає дедалі актуальнішим.

Розширення міст і зростання їхньої складної інфраструктури збільшує ризики аварій, злочинів та інших надзвичайних подій. У великих урбаністичних центрах виникає потреба у швидкому доступі до інформації та оперативному реагуванні на ситуації, що загрожують безпеці. Крім того, підвищена мобільність людей, які часто подорожують і переміщуються між містами та країнами, робить застосунки для моніторингу місцезнаходження і забезпечення безпеки не просто зручними, а необхідними.

Сучасні технології, такі як смартфони з постійним підключенням до Інтернету та GPS, роблять ці застосунки доступними для широкого кола користувачів. Завдяки цим технологіям відстеження місцезнаходження в реальному часі та надсилання тривожних сигналів стало можливим практично будь-де і будь-коли.

Соціальні фактори також підкреслюють необхідність використання таких застосунків. Зростання рівня злочинності у багатьох регіонах змушує людей шукати додаткові засоби забезпечення своєї безпеки. Надзвичайні ситуації, як-от природні катастрофи, техногенні аварії та інші непередбачені події, вимагають наявності надійних інструментів для швидкого реагування. Мобільні застосунки, здатні миттєво повідомити про небезпеку та передати точну інформацію про місцезнаходження, стають незамінними у таких випадках.

Для багатьох людей важливим аспектом є турбота про безпеку своїх близьких. Батьки можуть легко відстежувати місцезнаходження своїх дітей, а

родичі — місцезнаходження літніх членів сім'ї, що підвищує їхню безпеку та спокій. Функції автоматичного сповіщення про прибуття чи вихід з певних локацій допомагають підтримувати постійний зв'язок і бути в курсі пересувань своїх рідних.

З економічної точки зору, використання таких застосунків сприяє зниженню витрат на реагування. Швидке і точне визначення місцезнаходження у разі надзвичайної ситуації дозволяє службам екстреної допомоги ефективніше розподіляти ресурси, що зменшує загальні витрати та підвищує ефективність реагування. Крім того, це допомагає швидше вирішувати проблеми, пов'язані з безпекою, що позитивно впливає на загальну продуктивність громадян.

Не менш важливою є проблема захисту та конфіденційності даних. Сучасні застосунки використовують передові технології шифрування та захисту інформації, що забезпечує високий рівень конфіденційності для користувачів. Це особливо важливо в умовах зростаючої кількості кіберзагроз та шахрайських дій.

Отже, застосунки для особистої безпеки, мають значну доцільність у сучасний час. Завдяки технічному прогресу та зростаючим потребам у безпеці, використання таких застосунків продовжує зростати і вдосконалюватися, роблячи їх невід'ємною частиною нашого життя.

Переваги сервісів для забезпечення безпеки:

- Швидка реакція при загрозі.
- Відстеження місцезнаходження в реальному часі.
- Історія маршрутів та аналітика.

Недоліки сервісів для забезпечення безпеки:

- Постійна робота з конфіденційними даними.
- Високе споживання батареї.
- Точність геолокації.

1.2 Огляд аналогів

Personal Safety (рис. 1.1) – андроїд додаток доступний на телефонах Pixel [2].

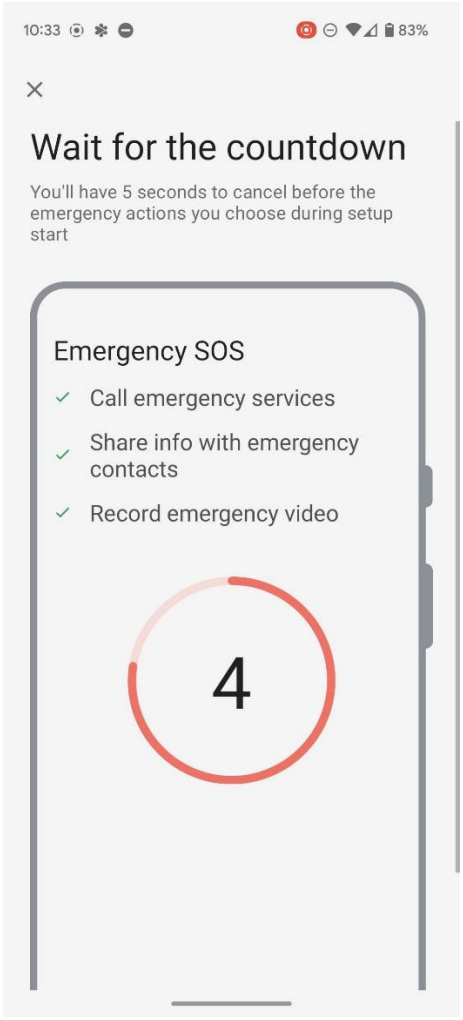


Рисунок 1.1 – Виклики Emergency SOS в додатку Personal Safety

Основні функції Personal Safety включають:

1. Екстрений виклик допомоги. Можливість швидкого виклику екстрених служб (поліція, швидка допомога, пожежна охорона) одним натисканням кнопки.
2. Доступ до матеріалів. Відстеження місця перебування користувача в режимі реального часу та надсилання координат місцезнаходження до екстрених контактів під час виклику допомоги.

3. Надсилання тривожних повідомлень. Автоматичне або ручне надсилання тривожних повідомлень до попередньо визначених контактів (друзі, сім'я).
4. Аудіо- та відеозапис подій. Автоматичний запис аудіо та відео під час активації тривоги. Збереження та передача записів на сервер або екстреним контактам для доказової бази.
5. Функція "Таймер перевірки безпеки". Встановлення таймера, після закінчення якого додаток автоматично надсилає тривожне повідомлення, якщо користувач не підтвердить свою безпеку.

Недоліки:

1. Жорстка прив’язка до моделі телефону. Даний додаток доступний лише на моделях телефонів Google Pixel з 4 версії.
2. Відсутність універсальності. Додаток може не враховувати специфічні потреби або ситуації, характерні для різних регіонів або країн. Наприклад, різні номери екстрених служб або різні сценарії надзвичайних ситуацій.
3. Високе споживання енергії. Постійне відстеження місцезнаходження та робота в фоновому режимі можуть швидко розряджати батарею смартфона, що може бути критичним у надзвичайній ситуації.

Life360 (рис. 1.2) – це мобільний додаток, призначений для відстеження місцезнаходження членів сім'ї або друзів у реальному часі та забезпечення їхньої безпеки. Він дозволяє створювати групи ("кола"), де учасники можуть бачити місцезнаходження один одного, отримувати сповіщення про прибуття чи вихід з певних місць, використовувати SOS-функцію для швидкого виклику допомоги, а також автоматично виявляти аварії та інші надзвичайні ситуації. [3].

Основні функції Classroom включають:

1. Сімейні кола. Створення груп (кіл) для відстеження місцезнаходження членів сім'ї або друзів у реальному часі. Можливість налаштування різних кіл для різних груп людей (сім'я, друзі, колеги).

2. Сповіщення про місцезнаходження. Отримання автоматичних сповіщень, коли члени кола прибувають або залишають певні локації (додому, школу, роботу).
3. SOS-функція. Швидка відправка тривожних повідомлень з точним місцезнаходженням до всіх членів кола у випадку надзвичайної ситуації.
4. Аварійне виявлення. Автоматичне виявлення дорожньо-транспортних пригод і негайне сповіщення екстрених контактів з наданням інформації про місцезнаходження.
5. Моніторинг стану батареї. Перевірка рівня заряду батареї пристроїв членів кола, щоб бути в курсі, коли комусь може знадобитися зарядний пристрій.

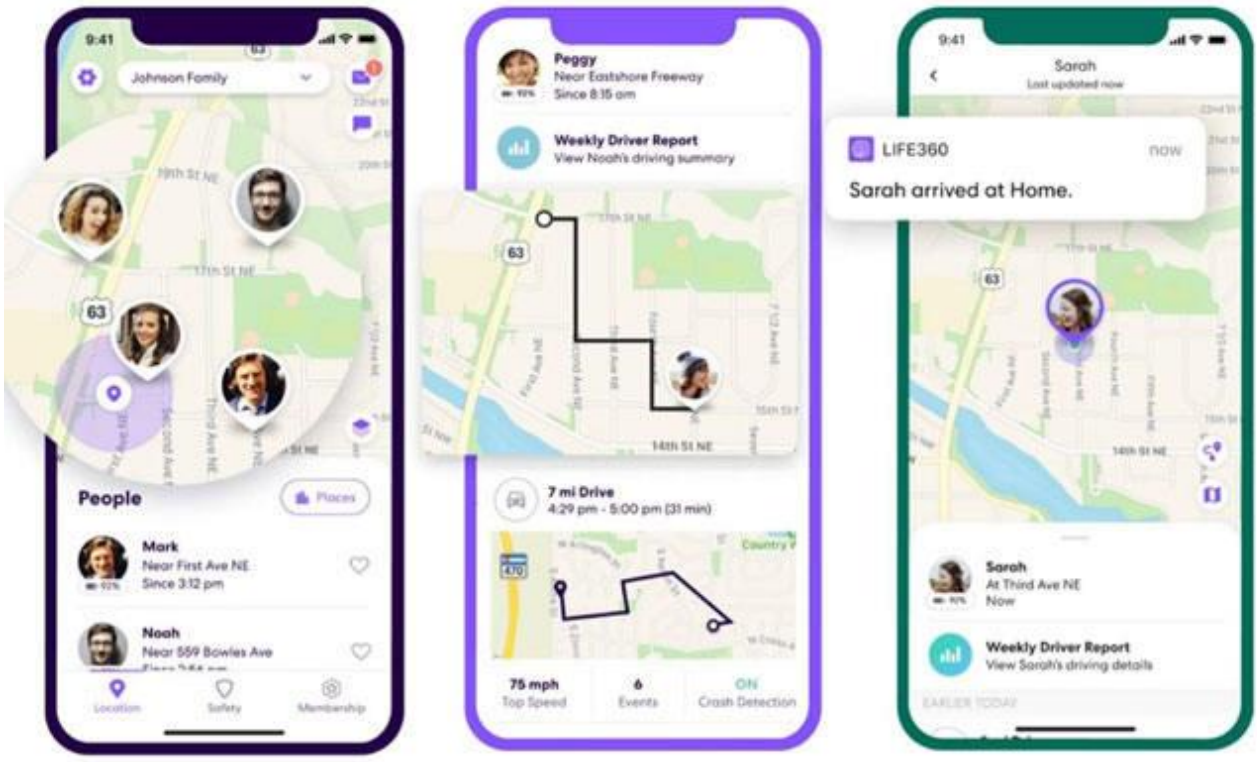


Рисунок 1.2 – Приклади інтерфейсу та використання додатку Life360

Недоліки:

1. Платні функції. Велика кількість корисних функцій доступні лише у платній версії додатка, що може обмежити його використання для користувачів, які не бажають або не можуть дозволити собі оплатити підписку.
2. Потенційна ненадійність сповіщень. Іноді сповіщення можуть приходити із затримкою або не приходити взагалі, що може бути критичним у надзвичайних ситуаціях. Доступний лише один тип сповіщень – push нотифікаціями всередині додатку.
3. Проблеми з точністю геолокації. У деяких випадках можуть виникати проблеми з точністю визначення місцезнаходження, особливо у густонаселених районах або в місцях з поганим GPS-сигналом.

1.3 Постановка задачі та формулювання вимог

У цій роботі було поставлено завдання розробки мобільного додатку, який забезпечить користувачів ефективними інструментами для забезпечення їхньої особистої безпеки та надання допомоги у надзвичайних ситуаціях.

Для досягнення цього завдання платформа повинен мати наступні функції:

- Можливість реєстрації та авторизації користувачів.
- Система тривожних сигналів: Можливість активувати тривожний сигнал в разі надзвичайних ситуацій за допомогою швидкого натискання кнопки або введенням кодового слова.
- Автоматичне сповіщення контактів: Можливість автоматичного сповіщення визначених контактів у разі активації тривожного сигналу або виникнення надзвичайної ситуації.
- Зручне та гнучке налаштування профілю користувача для надсилання потрібної та актуальної інформації.

1.4 Висновок до розділу 1

У даному розділі проведено детальний порівняльний аналіз існуючих додатків екстреної допомоги з описом їх можливостей та недоліках. Після ознайомлення з різними сервісами, визначено, що розробка такої системи доволі важке завдання для виконання якого потрібно враховувати досвід існуючих застосунків.

Застосунки які надають інструменти для забезпечення персональної безпеки, є надзвичайно актуальними в наш час. Вони не тільки надають широкий спектр функцій для захисту користувачів у різних ситуаціях, але й допомагають зменшити ризики та підвищити загальний рівень безпеки в суспільстві. Завдяки технічному прогресу та зростаючій потребі в безпеці, популярність таких застосунків постійно зростає і вони стають все більш досконалими, перетворюючись на невід'ємну частину нашого життя.

Виконано постановку задачі та опис призначення розроблюваного сервісу, внаслідок чого визначено напрямки подальших досліджень.

2 ПРОЕКТУВАННЯ СЕРВІСУ ЕКСТРЕННОЇ ДОПОМОГИ

2.1 Вибір технологій та проектування розгортання серверної частини

Для більш ефективного, надійного та швидкого розгортання були використані підходити CI/CD [4]. CI (Continuous Integration) — це неперервна інтеграція, а CD (Continuous Delivery) — неперервна доставка. Цей набір методик дозволяє розробникам частіше і надійніше розгортати зміни в програмному забезпеченні.

Більшість сучасних застосунків створюють із використанням різних платформ та інструментів. Тому виникає необхідність у їх злагодженій інтеграції та тестуванні внесених змін. Основна мета CI/CD — як раз забезпечити послідовну й автоматизовану збірку, упаковування і тестування застосунків. Завдяки цьому розробники можуть більше зосередитись на якості коду та безпеці продукту.

В якості віддаленого репозиторію використовувався репозиторій на Github. В якості інструменту для впровадження CI/CD був обраний GitHub Actions [5]. Оскільки GitHub Actions інтегрований безпосередньо в GitHub, його використання значно спрощує налаштування та керування CI/CD процесами без необхідності використовувати сторонні сервіси.

GitHub Actions – це потужний інструмент автоматизації, який дозволяє створювати та керувати процесами Continuous Integration (CI) та Continuous Deployment (CD) безпосередньо в середовищі GitHub. Завдяки своїй інтеграції з GitHub, Actions забезпечує безшовний процес налаштування, виконання та моніторингу CI/CD для проектів.

Основними перевагами використання цього інструменту є:

- Масштабованість – GitHub Actions підтримує масштабованість завдяки можливості використовувати різні середовища для виконання завдань (наприклад, Linux, Windows, macOS), що дозволяє тестувати та розгортати додатки у різних конфігураціях.

- Гнучкість - Actions забезпечує гнучкість у налаштуванні робочих процесів (workflows). Користувачі можуть створювати кастомні workflow-файли, які містять різноманітні дії (actions) та кроки (steps) для досягнення необхідних завдань.
- Безпека – GitHub Actions забезпечує високий рівень безпеки, включаючи підтримку секретів (secrets) для зберігання конфіденційної інформації, обмеження доступу до репозиторіїв та контроль доступу на рівні організації.

Ключові поняття в Github Actions:

- Робочий процес – це автоматизований процес, який буде виконувати одне або кілька завдань. Робочі процеси визначаються файлом YAML, повернутим до репозиторію, і будуть виконуватися під час активації події в репозиторії. Либо їх можна активувати вручну або за певним розписом.
- Подія - це певна дія у репозиторії, яка активує виконання робочого процесу. Наприклад, джерелом дії може бути GitHub, коли хтось створює запит на витягування, відкриває проблему або відправляє фіксацію до репозиторію. Ви також можете активувати робочий процес для запуску за розкладом, розмістивши REST API або вручну.
- Робота - це набір кроків у процесі, який виконується у тому ж засобі виконання. Кожен етап — це скрипт оболонки, який виконуватиметься, чи дію, яке виконуватиметься. Етапи виконуються по порядку та залежать один від одного. Оскільки кожен етап виконується одному засобі виконання, дані кількох етапів може бути загальними. Наприклад, може бути етап, який створює додаток, за ним слідує етап, який перевіряє створений додаток.

Після ознайомлення з теоретичною частиною розглянемо, як влаштований цей процес в проекті. На рисунку 2.1 наведена діаграма розгортання додатку за допомогою Github Actions

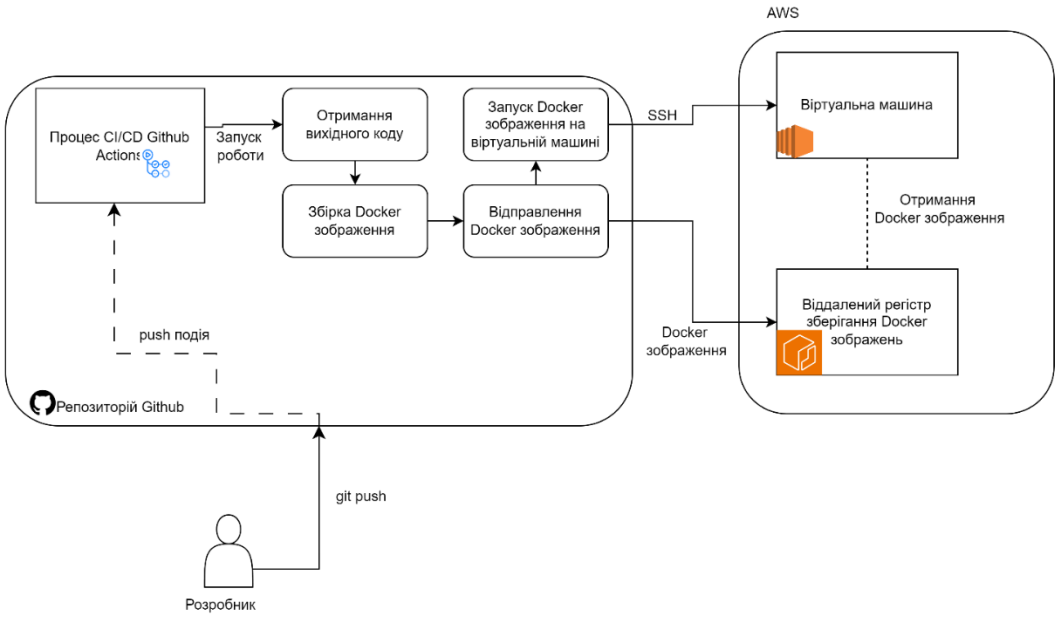


Рисунок 2.1 – Діаграма розгортання серверної частини

Розглянемо процес розгортання детальніше. Першим кроком є відправлення нових змін в коді розробником, після цього репозиторій помічає подію та запускає робочий процес Github Actions серед тригерів якого наявна ця подія. Увесь робочий процес можна поділити на чотири роботи:

– Отримання вихідного коду з репозиторію
– Збірка Docker [6] зображення
– Відправлення Docker зображення в віддалений репозиторій
– Налаштування та підключення по SSH [7] до віртуальної машини. Отримання раніше зібраного Docker зображення та запуск Docker контейнера.

У якості віддаленого репозиторію для зберігання Docker зображень було обраного AWS Elastic Container Registry. А для в якості віртуальної машини AWS

EC2. Про ці та інші використані сервіси від Amazon Web Services буде розказано у наступному розділі.

2.2 Архітектура та хмарна інфраструктура серверної частини

У сучасному світі використання хмарної інфраструктури для додатків стає все більш популярним і обґрунтованим вибором з багатьох причин.

Хмарні провайдери, як-от AWS [8], Google Cloud[9] та Microsoft Azure, дозволяють легко масштабувати ресурси відповідно до потреб вашого додатку. Це означає, що ви можете швидко збільшити або зменшити обчислювальну потужність, зберігання даних та інші ресурси без значних капіталовкладень.

Для вибору хмарного провайдеру було проведено порівняння [10] трьох найпопулярніших представників на ринку:

Таблиця 2.1 – Порівняння хмарних провайдерів

Категорія	AWS (Amazon Web Services)	Azure (Microsoft Azure)	GCP (Google Cloud Platform)
Ринкова частка та досвід	Частка 22%. Заснований у 2006 році	Частка 10%. Заснований у 2010 році.	Частка – 33%. Заснований у 2011 році.
Обширність і різноманітність послуг	Понад 200 сервісів, інтеграція між різними користувачами всередині сервісу.	Понад 200 сервісів. Додатково наявна інтеграція з Microsoft Office 365	Понад 150 сервісів. Інтеграція з різноманітними сервісами Google: Workspace, Maps, Youtube, Google Ads.

Продовження таблиці 2.1

Категорія	AWS (Amazon Web Services)	Azure (Microsoft Azure)	GCP (Google Cloud Platform)
Розробницькі інструменти	Велика кількість SDK, API, потужні CI/CD інструменти, підтримка DevOps.	Відмінна інтеграція з Visual Studio та іншими Microsoft інструментами.	Сильна інтеграція з Google сервісами (BigQuery, TensorFlow).
Ціни та модель оплати	AWS Cost Explorer Гнучка модель оплати, розширені можливості економії (наприклад, Reserved Instances).	Azure Cost Management and Billing Гнучка модель оплати, оплата по мірі використання.	Google Cloud Console Billing. Оплата по мірі використання, найзрозуміліший інтерфейс при оплаті
Глобальна інфраструктура	32 регіони та 102 зони доступності.	47 регіонів та 92 зони доступності.	39 регіонів та 118 зон доступності.

AWS виділяється своєю тривалою історією, великою ринковою часткою, обширністю послуг, та найширшою глобальною інфраструктурою. Це робить його привабливим вибором для розробки сервісу екстренної допомоги.

Після вибору хмарного провайдеру було створено архітектуру всього сервісу, що наведена на рисунку 2.2. У якості архітектури програмної частина було обрано монолітну архітектуру з допоміжними модулями у вигляді AWS Lambda [11].

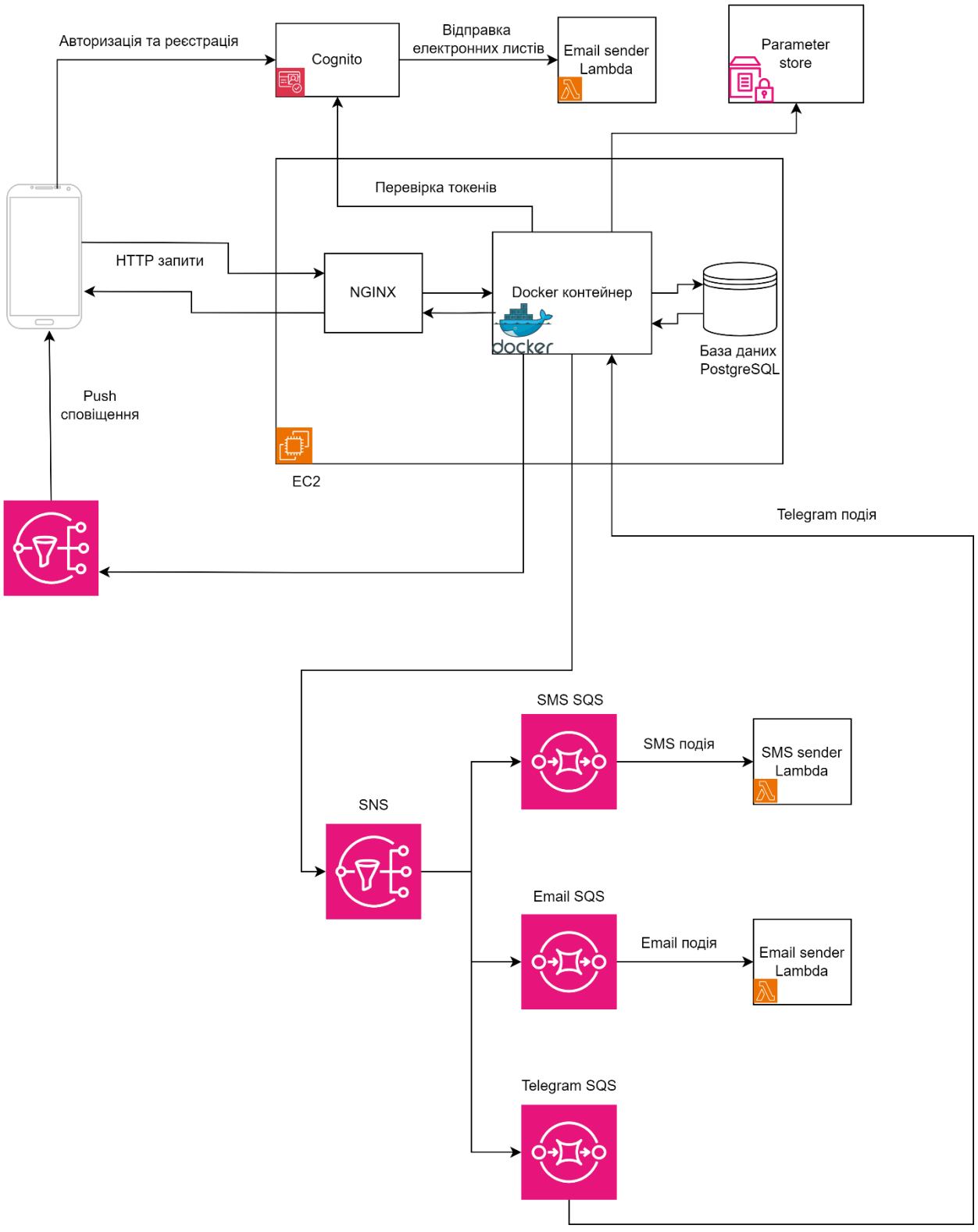


Рисунок 2.2 – Архітектура сервісу екстреної допомоги

Розглянемо основі сервіси, що використовувались при побудові додатку:

- Amazon Cognito – забезпечує аутентифікацію, авторизацію та управління користувачами для веб- та мобільних додатків. Він дозволяє легко додавати реєстрацію, вхід та контроль доступу до ваших додатків. Cognito підтримує функції для соціальної авторизації через провайдери, такі як Google, Facebook, та Amazon, а також традиційні методи аутентифікації через електронну пошту та пароль. Крім того, він пропонує можливості для безпарольного входу через SMS чи електронну пошту, а також багатофакторну аутентифікацію (MFA). Зокрема, Cognito синхронізується з AWS Identity and Access Management (IAM) для детального контролю доступу до ресурсів AWS.
- Lambda – є серверлес обчислювальною послугою, яка дозволяє запускати код у відповідь на події без необхідності управляти серверами. Lambda автоматично масштабується, обробляючи окремі запити паралельно, і вам платять лише за час виконання вашого коду. Lambda підтримує різноманітні мови програмування, включаючи Java, Python, Node.js, C#, Ruby, та Go. Ви можете використовувати Lambda для створення бекендів додатків, обробки потоків даних у реальному часі, виконання ETL задач, інтеграції з іншими сервісами AWS та багато іншого.
- EC2 (Elastic Compute Cloud) [12] – дозволяє користувачам запускати віртуальні сервери, відомі як інстанси, які можна конфігурувати для задоволення специфічних потреб додатку. EC2 пропонує широкий вибір типів інстансів, оптимізованих для різних робочих навантажень, таких як загального призначення, обчислювально-інтенсивні, пам’яттєво-інтенсивні, графічні процесори (GPU) та інші. EC2 інтегрується з іншими сервісами AWS, включаючи S3 для зберігання, RDS для управління базами даних, і VPC для ізоляції мережі, надаючи високий рівень контролю та гнучкості.

- Parameter Store – є частиною AWS Systems Manager і надає безпечне зберігання конфігураційних даних та секретів. Це дозволяє централізовано керувати конфігураціями для додатків і служб у хмарі та на локальних серверах. Parameter Store підтримує зберігання plaintext і шифрованих параметрів, інтегрується з AWS Key Management Service (KMS) для шифрування, та забезпечує контроль доступу через IAM. Це корисно для зберігання конфігураційних параметрів, таких як рядки підключення до баз даних, API ключі, паролі та інші секрети.
- Amazon SNS – сервіс для масової доставки повідомлень та сповіщень. Він підтримує відправку повідомлень до кінцевих точок або клієнтів за допомогою різних протоколів, таких як SMS, електронна пошта, HTTP/HTTPS, та AWS Lambda. SNS також підтримує публікацію-підписку (pub/sub) моделі, де повідомлення можуть бути транслювані до великої кількості підписників одночасно. Це корисно для побудови систем оповіщення, сповіщень про зміни в додатках, а також для координації між розподіленими системами.
- Amazon SQS – є повністю керованим сервісом черг повідомлень, що дозволяє розділяти та масштабувати розподілені системи та додатки. SQS дозволяє передавати будь-який обсяг даних одночасно через черги, забезпечуючи високу доступність та надійність передачі повідомлень між різними компонентами системи. Підтримуються дві моделі черг: Standard Queues, які пропонують високу пропускну здатність та принаймні один раз доставку повідомлень, та FIFO Queues (First-In-First-Out), які гарантують упорядковану та саме один раз доставку повідомлень. SQS використовується для побудови масштабованих та стійких систем з розділеними обчислювальними компонентами.– представляє тему, до якої належать матеріали чи завдання.

Початковим кроком при будь-якій роботі користувача є звернення мобільного додатку до сервісу авторизації Cognito для отримання JWT токенів [13] для подальшого звернення до бекенд частини сервісу.

В свою чергу, бекенд частина розміщена на віртуальній машині EC2, разом з веб сервером Nginx [14], що забезпечує дію протоколу HTTPS за допомогою згенерованих та підключених SSL сертифікатів. Бекенд частина знаходиться в Docker контейнері для забезпечення легкого, швидкого та універсального розгортання, що було описано у пункті 2.1. Додатково на віртуальній машині розташована база даних, що використовує СУБД PostgreSQL [15]. В майбутньому планується перенести управління та хостинг бази даних на спеціалізований сервіс Amazon RDS.

Для звернень до бекенд частини використовуються HTTP запити, в заголовках яких міститься JWT токен, отриманий від Cognito, що забезпечує авторизацію та аутентифікацію всіх користувачів. Cognito використовує схему авторизації OAuth 2.0 та OpenIDConnect, тому для перевірки токенів серверна частина мусить запросити публічний ключ, який були створений в парі з приватним ключем. У свою чергу публічний ключ дозволяє перевірити сігнатуру токену та переконатись, що запит був відправлений не зловмисником.

Бекенд частина містить деяку конфіденційну інформації та конфігурацію, як от наприклад API ключі до сторонніх сервісів. Для зберігання цієї інформації був використаний Parameter Store, данні в якому, сервер запитує кожного разу при запуску головної програми.

В якості допоміжних модулів для відправки різноманітних повідомлень та сповіщень було використано AWS Lambda. Це рішення дозволяє не мати постійно запущений сервер. Кожна Lambda тільки тоді як в цьому є необхідність, і одразу вимикається після завершення роботи – це допомагає зменшити кошти на утримання всієї інфраструктури.

Для виклику кожної Lambda використовуються два сервіси по доставці повідомлень. Бекенд відсилає всі повідомлення в одну спільну SNS, яка

всередині себе розподіляє ці повідомлення по трьом різним SQS, в залежності від способу доставки цього повідомлення.

2.3 Висновок до розділу 2

У цьому розділі було обрано хмарний провайдер для сервісу, спроектовано розгортання та хмарну інфраструкту.

Основою для вибору Amazon Web Services у якості провайдеру є його великий досвід на ринку та великий набір сервісів та SDK для розробки.

Запропонована архітектура демонструє ефективну інтеграцію різноманітних сервісів AWS для забезпечення безпечної, масштабованої та гнучкої системи. В майбутньому при розвитку сервісу ця архітектура дозволить легко та майже безшовно додавати нову функціональність та масштабуватись до вимог користувачів.

3 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ

3.1 Аналіз і обґрунтування вибору засобів для реалізації

Для розробки серверної частини необхідно вибрати мову програмування. Проведемо аналіз трьох найпопулярніших мов для створення веб додатків [17].

Java — мова загального призначення, яка може з дивовижною легкістю переходити з однієї комп'ютерної системи в іншу. Тому не дивно, що сьогодні вона входить до десятки найпоширеніших мов програмування у світі (посідає 6 місце). Згідно з опитуванням розробників Stack Overflow 2022, 33,27% розробників активно ним користуються. Остання версія Java – Java 19, вона була випущена у вересні 2022 року.

В даний час Java - це не просто універсальна мова. Це ціла платформа та екосистема, яка поєднує різні технології, що використовуються для різноманітних завдань, починаючи від створення настільних програм і закінчуючи створенням великих веб-порталів і сервісів. Крім того, Java активно використовується для розробки програмного забезпечення для багатьох пристроїв, включаючи ПК, планшети, смартфони, побутову техніку та багато іншого.

Переваги:

- Масштабованість та простота: Java дозволить серверу запускати кілька екземплярів програм одночасно. Завдяки об'єктно-орієнтованому підходу ви зможете створювати великі, гнучкі, масштабовані та розширювані веб-додатки.
- Багатопотоковість: програмний код здатний обробляти запити в незалежних потоках на багатопоточному WEB-сервері. Ось чому ця мова програмування чудово працює з програмами, які потребують величезної потужності процесора.

Недоліки:

- При використанні Java для комерційного використання, необхідно мати ліцензію.

- Java працює повільно через використання комп’ютерної віртуальної машини для компіляції, проблеми з очищенням пам’яті, блокування потоків, погані параметри кешування тощо.
- Розробникам доводиться використовувати сторонні бібліотеки та інструменти для створення нативного дизайну.

C#[18] – це це мова програмування загального призначення високого рівня, розроблена корпорацією Microsoft і випущена в 2001 році. C# не залежить від платформи завдяки своїй компіляції в проміжну мову (IL), яка потім може бути виконана на будь-якій платформі, що підтримує загальну мовну інфраструктуру (CLI). . C# є однією з основних мов для розробки додатків у .NET Framework. Синтаксис C# подібний до C++ і Java.

Переваги:

- Платформа .NET: C# тісно інтегрований з платформою .NET, що надає доступ до великої кількості бібліотек і фреймворків, що значно полегшує розробку додатків різного рівня складності.
- Кросплатформеність: Завдяки .NET Core та Xamarin, C# дозволяє створювати додатки для різних операційних систем, включаючи Windows, macOS, Linux, Android та iOS.
- Автоматичне управління пам'яттю: C# використовує збирач сміття (garbage collector), що автоматично керує виділенням і звільненням пам'яті, зменшуючи ризик витоків пам'яті та інших помилок, пов'язаних з ручним управлінням пам'яттю.
- Підтримка асинхронного програмування: C# пропонує зручний синтаксис для асинхронних операцій (async/await), що робить написання асинхронного коду простішим та зрозумілішим.

Недоліки:

- Великі обсяги середовища виконання: Додатки на C# вимагають установки .NET середовища виконання, що може бути проблемою для невеликих програм або додатків з обмеженими ресурсами.

- Продуктивність: В порівнянні з мовами, такими як C++ або Rust, C# може бути менш продуктивним в задачах, що вимагають високої продуктивності та низького рівня доступу до пам'яті.

PHP – була створена в 1995 році і з тих пір продовжує займати високі позиції в різних рейтингах. За даними WWW Technology Surveys, понад 80% усіх веб-сайтів написані цією мовою програмування. Цим користуються такі гіганти, як Wikipedia, Slack, WordPress, Tumblr, Nvidia, Pinterest і Facebook.

Переваги:

- Автоматизація функцій: Функція створення сценаріїв PHP робить його придатним для створення автоматизації, як-от автентифікація, відображення URL-адрес, керування сеансами тощо.
- Відкритий код та універсальність: Існує багато безкоштовних бібліотек PHP, які прискорюють веб-розробку. Вони дозволяють програмістам використовувати функціональні можливості, які вже реалізовані та перевірені, замість того, щоб витрачати час на написання власного коду.

Недоліки:

- PHP не може ефективно конкурувати з сучасними технологіями розробки, такими як Java або C# через обмежену кількість бібліотек.
- Динамічна типізація – і один і той самий фрагмент коду може поводитися по-різному залежно від контексту, що ускладнює масштабування програм PHP, а іноді й уповільнювати.
- Погана підтримка сучасних архітектур - PHP не був створений для роботи з сучасними архітектурами, такими як мікросервіси. Його традиційна модель працює з кожним запитом окремо, що не завжди оптимально для більш складних систем.
- Безпека: PHP часто критикують за численні уразливості безпеки, зокрема в старих версіях. Розробникам необхідно ретельно

контролювати безпеку коду, щоб уникнути атак, таких як SQL-ін'єкції та XSS.

Для створення бекенд-частини обрано мову програмування C#, оскільки вона є універсальною мовою, що надає ряд переваг як завдяки особливостям самого C#, так і платформі .NET. Основні з них включають:

- Статична типізація. Типи даних змінних та виразів перевіряються на етапі компіляції, що допомагає виявляти помилки типу до запуску програми, зменшуючи час і зусилля на налагодження. Водночас, C# підтримує динамічну типізацію через ключове слово "dynamic", що може бути корисно в окремих випадках.
- Підтримка ООП. Мова повністю підтримує об'єктно-орієнтоване програмування з його основними принципами: абстракцією, наслідуванням, поліморфізмом та інкапсуляцією.
- Продуктивність. C# є компільованою мовою, що означає перетворення коду на машинний до виконання. Це робить її однією з найшвидших мов програмування, доступних сьогодні.
- Набір інструментів для бекенд-розробки. C# пропонує потужні інструменти для бекенд-розробки, такі як ASP.NET Core – високопродуктивний фреймворк, та Entity Framework Core – інструмент для об'єктно-реляційного відображення даних, серед інших.

Отже, C# є оптимальним вибором для розробки бекенду цього програмного застосунку завдяки своїм численним перевагам.

3.2 Розробка системи автоматичного розгортання серверної частини

Для реалізації цієї системи, як уже була зображена на рисунку 2.1, використовувались інструменти CI/CD від сервісу Github – Github Actions.

Початок створення будь-якого автоматичного процесу починається з створення файлу з розширенням .yml за шляхом .github/workflows/ в кореневій папці репозиторію – рисунок 3.1

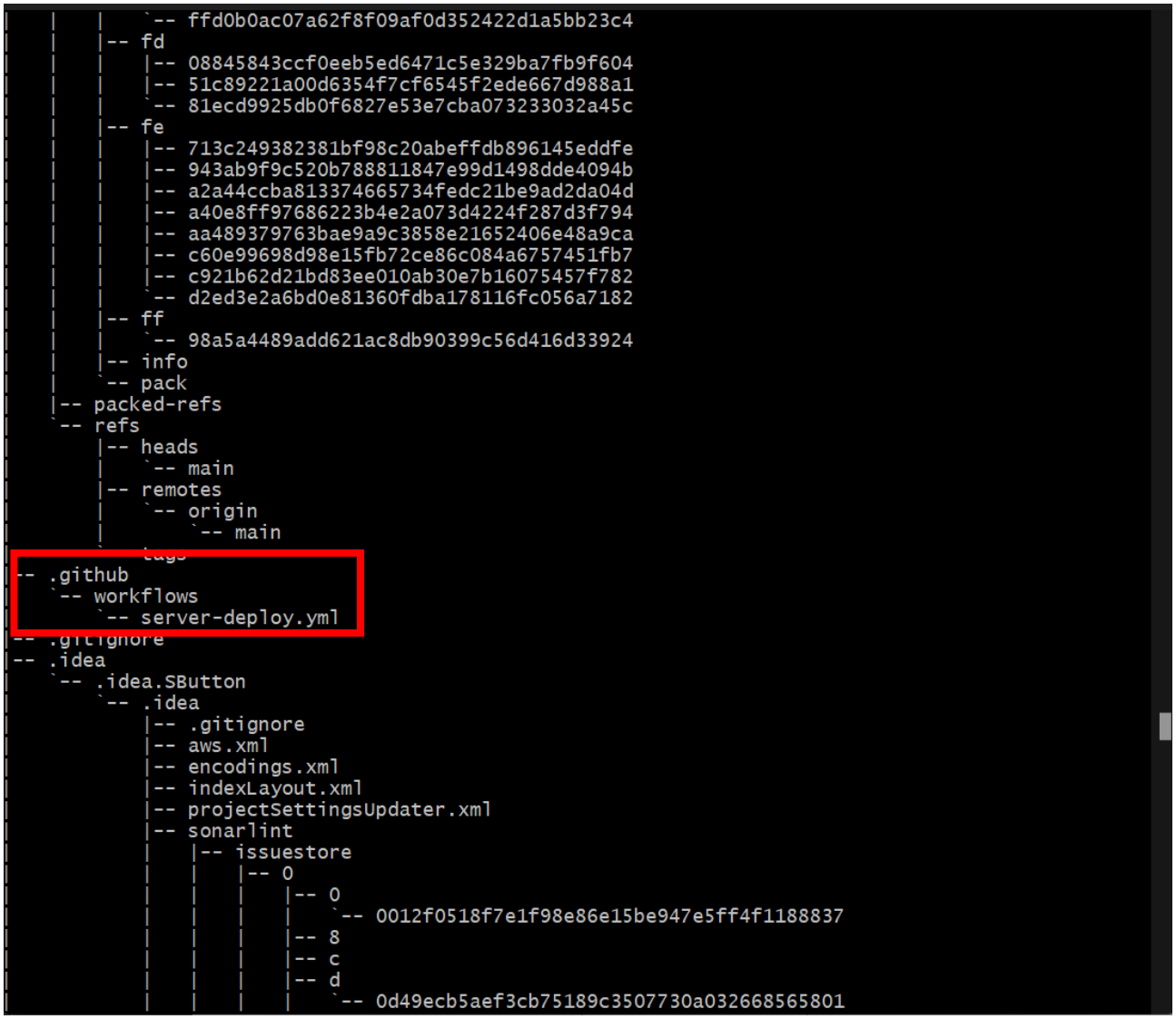
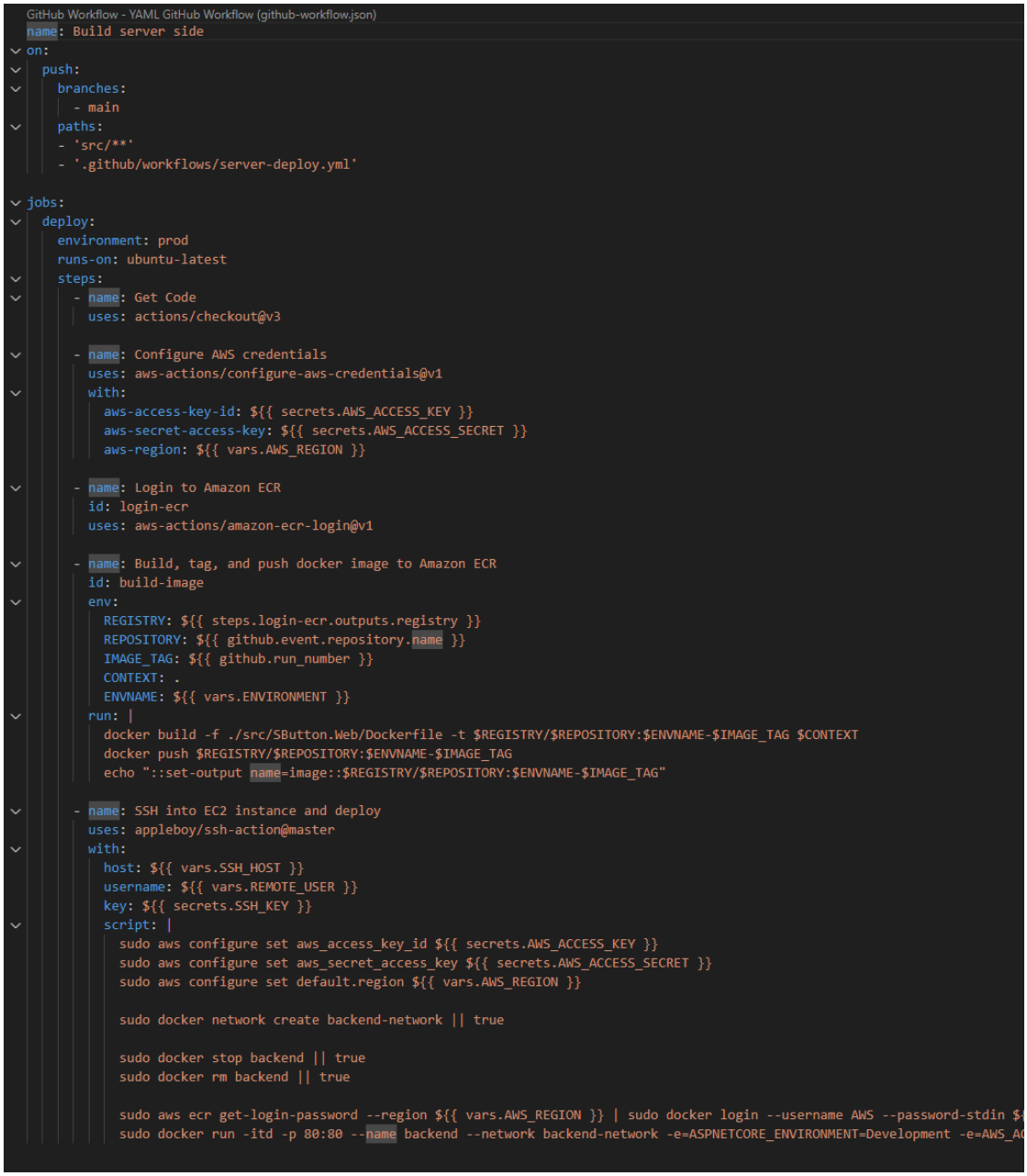


Рисунок 3.1 – Розташування файлу робочого процесу

Розглянемо детальніше файл розгортання для серверної частини (рис. 3.2). Блок «on» вказує на те, що робочий процес (workflow) запускається щоразу, коли відбувається push в гілку main або змінюються файли в директорії src або файл конфігурації server-deploy.yml.

Після цього в файлі роботи наведено список кроків, які будуть виконані при запуску, а саме: Get Code, Configure AWS credentials, Login to Amazon ECR,

Build, tag, and push docker image to Amazon ECR та SSH into EC2 instance and deploy. В загальному розглянемо, що роблять всі написані кроки.



```
GitHub Workflow - YAML GitHub Workflow (github-workflow.json)
name: Build server side
on:
  push:
    branches:
      - main
    paths:
    - 'src/**'
    - '.github/workflows/server-deploy.yml'

jobs:
  deploy:
    environment: prod
    runs-on: ubuntu-latest
    steps:
      - name: Get Code
        uses: actions/checkout@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_ACCESS_SECRET }}
          aws-region: ${{ vars.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push docker image to Amazon ECR
        id: build-image
        env:
          REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          REPOSITORY: ${{ github.event.repository.name }}
          IMAGE_TAG: ${{ github.run_number }}
          CONTEXT: .
          ENVNAME: ${{ vars.ENVIRONMENT }}
        run: |
          docker build -f ./src/SButton.Web/Dockerfile -t $REGISTRY/$REPOSITORY:$ENVNAME-$IMAGE_TAG $CONTEXT
          docker push $REGISTRY/$REPOSITORY:$ENVNAME-$IMAGE_TAG
          echo "::set-output name=image::$REGISTRY/$REPOSITORY:$ENVNAME-$IMAGE_TAG"

      - name: SSH into EC2 instance and deploy
        uses: appleboy/ssh-action@master
        with:
          host: ${{ vars.SSH_HOST }}
          username: ${{ vars.REMOTE_USER }}
          key: ${{ secrets.SSH_KEY }}
          script: |
            sudo aws configure set aws_access_key_id ${{ secrets.AWS_ACCESS_KEY }}
            sudo aws configure set aws_secret_access_key ${{ secrets.AWS_ACCESS_SECRET }}
            sudo aws configure set default.region ${{ vars.AWS_REGION }}

            sudo docker network create backend-network || true

            sudo docker stop backend || true
            sudo docker rm backend || true

            sudo aws ecr get-login-password --region ${{ vars.AWS_REGION }} | sudo docker login --username AWS --password-stdin $
            sudo docker run -itd -p 80:80 --name backend --network backend-network -e=ASPNETCORE_ENVIRONMENT=Development -e=AWS_A
```

Рисунок 3.2 – Виконуючий код робочого процесу розгортання

Get Code – використовує GitHub Action “actions/checkout@v3”, щоб скопіювати (checkout) репозиторій з вихідним кодом в робочий каталог виконавчого середовища. Це дозволяє подальшим крокам отримати доступ до коду проекту.

Configure AWS credentials – використовує GitHub Action aws-“actions/configure-aws-credentials@v1” для налаштування облікових даних AWS. Використовуються секрети, збережені в сховищі секретів GitHub, для безпечного доступу до AWS. Вказується регіон AWS, де будуть виконуватись наступні дії.

Login to Amazon ECR - використовує GitHub Action aws-“actions/amazon-ecr-login@v1”, щоб увійти в Amazon Elastic Container Registry (ECR). Після успішного входу, зберігається вихідний реєстр у змінній registry.

Build, tag, and push docker image to Amazon ECR - використовує Docker для побудови образу контейнера. Параметри середовища (REGISTRY, REPOSITORY, IMAGE_TAG, CONTEXT, ENVNAME) використовуються для визначення місця збереження та тегування образу. Образ будується за допомогою Dockerfile, розташованого в ./src/SButton.Web/Dockerfile. Після побудови образ завантажується в Amazon ECR. Остання команда встановлює вихідний параметр image, що містить повне ім'я образу.

SSH into EC2 instance and deploy**:**

- Використовується GitHub Action “appleboy/ssh-action@master” для SSH-підключення до EC2 інстанса.
- Налаштовуються облікові дані AWS на віддаленому сервері.
- Створюється Docker мережа **backend-network**, якщо вона ще не існує.
- Зупиняється та видаляється старий контейнер **backend**, якщо він існує.
- Виконується вхід до Amazon ECR на віддаленому сервері
- Запускається новий контейнер з побудованим образом, вказуються необхідні змінні середовища.

Цей робочий процес дозволяє автоматизувати процес деплойменту коду на віддалений сервер через EC2 інстанс, включаючи побудову Docker-образу, його завантаження в Amazon ECR та розгортання на сервері.

Протягом усього перебігу робочого процесу на сторінці Github можна побачити логування всього, що відбувається і при невдалому запуску (рис. 3.3) легко та зрозуміло знайти і виправити помилку

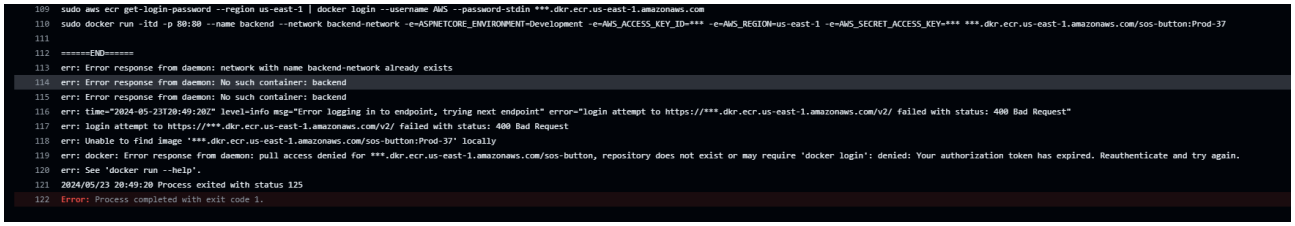


Рисунок 3.3 – Логування виконання робочого процесу

Варто зауважити, що деякі змінні, як наведені у файлі робочого процесу (рис. 3.1), як от: AWS_ACCESS_KEY, AWS_ACCESS_SECRET, SSH_KEY – є конфіденційними та не можуть просто знаходить у відкритому вигляді у коді. Саме для тих випадків в Github Actions є інструмент Repository Variables And Secrets [19]. Ці змінні зберігаються в зашифрованому вигляді, та взагалі не можуть бути прочитані людським оком – рисунок 3.4

Рисунок 3.4 – Repository secrets

На цьому розробку системи автоматичного розгортання серверної частини можна вважати закінченою.

3.3 Розробка і налаштування авторизації та аутентифікації за допомогою AWS Cognito

Використання аутентифікації та авторизації є критично важливим для забезпечення безпеки програмних продуктів. Вони допомагають захистити дані від несанкціонованого доступу, підтримують цілісність системи та забезпечують дотримання вимог законодавства. Впровадження надійних механізмів аутентифікації та авторизації підвищує довіру користувачів і сприяє захисту бізнесу від загроз безпеці.

Аутентифікація - це процес перевірки особи користувача або системи. Її основна мета - підтвердити, що користувач є тим, за кого себе видає. Авторизація - це процес надання або обмеження доступу до певних ресурсів або дій після успішної аутентифікації. Вона визначає, які ресурси та дії дозволені користувачеві.

Перед написанням коду для перевірки користувачів на бекенді, потрібно створити і налаштувати сервіс, який буде забезпечувати коректну, а головне надійну, авторизацію користувачів. В якості такого сервісу використовується AWS Cognito. Під час налаштування Cognito, є можливість налаштування паролю (рис. 3.5) – вибір мінімальної довжини паролю та необхідність певних символів.

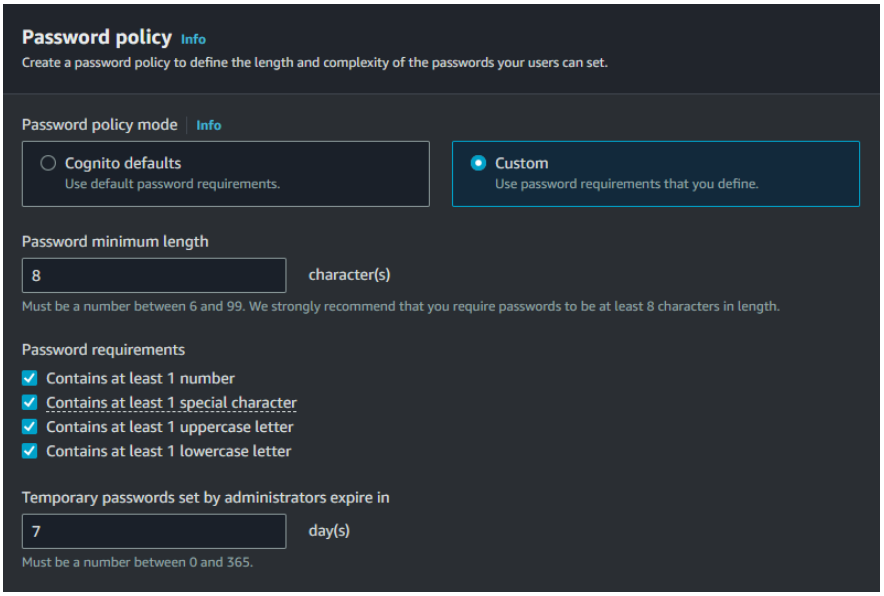


Рисунок 3.5 – Налаштування паролю при створенні Cognito

Підключення методів багатофакторної перевірки (рис. 3.6) – є можливість додати багатофакторну перевірку через спеціальні програми, що побудовані за технологією TOTP або ж за звичайними SMS.

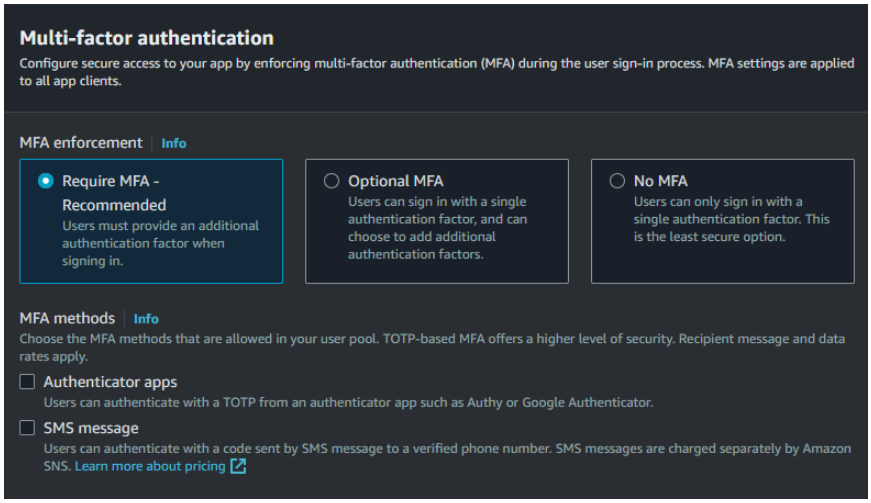


Рисунок 3.6 – Налаштування багатофакторної перевірки

А також можна налаштувати дозволенні методи, за допомогою яких користувач зможе скинути пароль (рис. 3.7) – email та SMS, а також різна варіативність комбінацій цих методів.

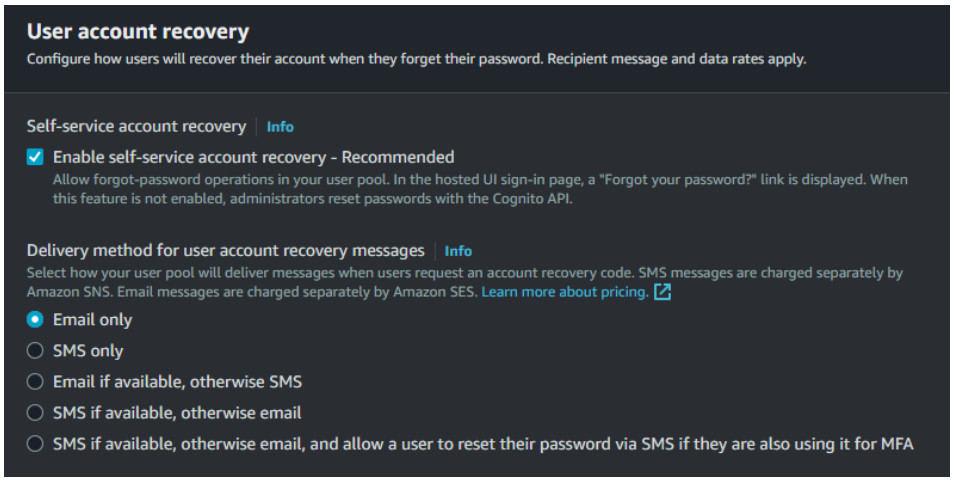


Рисунок 3.7 – Налаштування методів скидання паролю користувачем

Після успішного створення сервісу Cognito, можна буде побачити головне меню управління (рис. 3.8), на якому буде розміщена інформація про зареєстрованих користувачів, і також вкладки на зміни деяких налаштувань

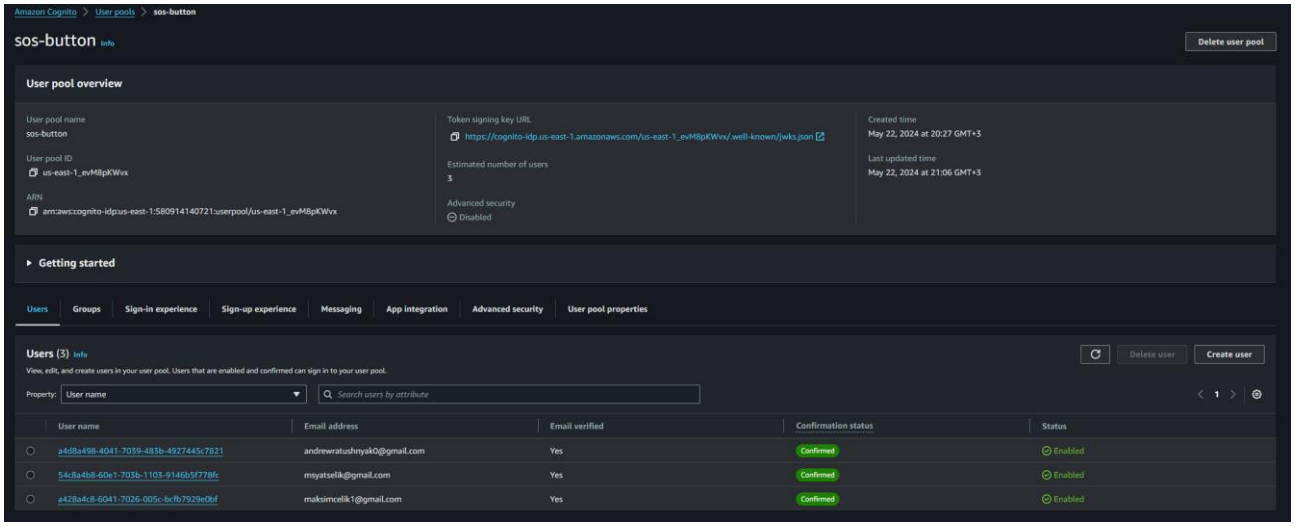


Рисунок 3.8 – Головне меню управління Cognito

Одним із таких налаштувань – є власна система відправлення електронних листів з перевірочними кодами для підтвердження електронної пошти. Оскільки в звичайній конфігурації сервісу лист з кодом буде мати не дуже презентабельний вигляд (рис. 3.9). Тому в якості вирішення цієї проблеми, був створений власний дизайн листа, а також інтегровано сторонній сервіс по надсиланню електронних листів – Postmark [20].

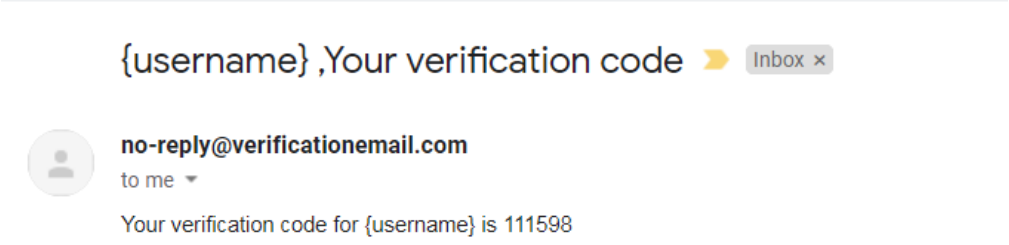


Рисунок 3.9 – Стандартний вигляд листа з Cognito

Обраний сервіс для відправки електронних листів, має зручний інструментарій створення шаблонів для відправки повторюваних листів. Завдяки цьому інструментару можна створювати дизайн електронних листів за допомогою мови гіпертекстової розміти HTML та мови стилів CSS, вихідний код створеного листа наведено у лістингу Б.1.

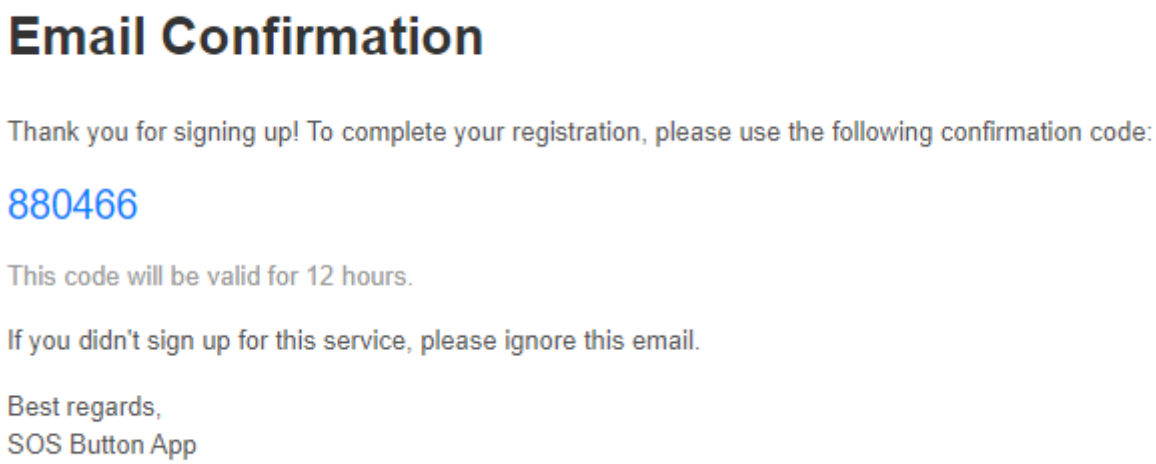


Рисунок 3.10 – Зовнішній вигляд листа для підтвердження пошти

Вся система для відправки листів була побудована лише за допомогою однієї AWS Lambda (рис. 3.11), яка отримує зашифрований код та через SDK Postmark для C# надсилає лист користувачеві.

Метод FunctionHandler є асинхронним і відповідає за основну логіку Lambda-функції, яка обробляє події від Amazon Cognito з метою відправки електронних листів. Починається він з ініціалізації логера, який використовується для запису інформаційних повідомлень та помилок. Далі, ініціалізуються провайдери матеріалів для шифрування та SDK для роботи з шифруванням з політикою, яка забороняє шифрування, але дозволяє розшифровку.

Потім створюється ключове кільце KMS, використовуючи клієнта KMS та ідентифікатор ключа, який отримується з змінної середовища. Наступним

кроком є розшифровка коду підтвердження, який був отриманий у зашифрованому вигляді в події. Спочатку цей зашифрований код конвертується з формату Base64, потім створюється об'єкт для розшифровки з цим шифрованим текстом та ключовим кільцем, і в кінці отриманий розшифрований текст конвертується в рядок коду підтвердження.

Після цього обробляється джерело тригера події. В залежності від того, чи це подія повторного відправлення коду або реєстрації, викликається метод для відправки листа з підтвердженням на вказану адресу електронної пошти. Якщо тригер невідомий, повертається повідомлення про відсутність тригера.

Насамкінець, якщо виникла помилка під час відправки листа, вона записується в лог. Якщо ж все пройшло успішно, записується інформаційне повідомлення про успішну відправку листа. Таким чином, метод обробляє подію від Amazon Cognito, розшифровує код підтвердження, відправляє електронний лист користувачу та логує результати.

```
public async Task FunctionHandler(CustomEmailSenderRequest eventData, ILambdaContext context)
{
    Ratushnyak, 21/02/2024 18:54 • Init for deploy
    var logger = context.Logger;
    var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
    var encryptionSdk = new ESDK(new AwsEncryptionSdkConfig
        { CommitmentPolicy = ESDKCommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT });

    var kmsDecryptKeyringInput = new CreateAwsKmsKeyringInput
    {
        KmsClient = new AmazonKeyManagementServiceClient(),
        KmsKeyId = Environment.GetEnvironmentVariable("KmsKeyArn"),
    };

    var kmsEncryptKeyring = materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);

    var decryptInput = new DecryptInput
    {
        Ciphertext = new(Convert.FromBase64String(eventData.Request.Code)),
        Keyring = kmsEncryptKeyring,
    };

    var decryptOutput = encryptionSdk.Decrypt(decryptInput);
    var code = Encoding.UTF8.GetString(decryptOutput.Plaintext.ToArray());

    var error = eventData.TriggerSource switch
                {
                    "CustomEmailSender_ResendCode" => await SendEmailConfirmationLetter(code, eventData.Request.UserAttributes.Email),
                    "CustomEmailSender_SignUp" => await SendEmailConfirmationLetter(code, eventData.Request.UserAttributes.Email),
                    _ => "No trigger found"
                };

    if (!string.IsNullOrEmpty(error))
        logger.LogError(error);

    logger.LogInformation("Letter was sent successfully");
}
```

Рисунок 3.11 – Код Lambda функції для відправки повідомлень

Після успішної реєстрації та входу користувача, йому надається JWT токен для подальшої авторизації при HTTP запитах до бекенду. Для перевірки токену на бекенді потрібно завантажити ключі, якими цей токен був створений. Код що відповідає за цю функціональність наведено на рисунку 3.12.

```
public static IServiceCollection AddSecurity(this IServiceCollection services, IConfiguration configuration)
{
    var validIssuer :string = configuration // IConfiguration
        .GetSection( key: nameof(CognitoOptions))
        .GetSection( key: nameof(CognitoOptions.Issuer)) // IConfigurationSection
        .Value; // string

    var keys :IEnumerable<JsonWebKey> = JwtHelper.GetSigningKeysByIssuer( issuer: validIssuer); // IEnumerable<JsonWebKey>

    services.AddAuthentication( defaultScheme: JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer( configureOptions: options :JwtBearerOptions =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                IssuerSigningKeyResolver = (_, _, _, _) => keys,

                ValidIssuer = validIssuer,
                ValidateIssuerSigningKey = true,
                ValidateIssuer = true,
                ValidateLifetime = true,
                ValidateAudience = false,
                ClockSkew = TimeSpan.Zero
            };
        }); // AuthenticationBuilder

    services.AddAuthorization(); // IServiceCollection
    return services;
```

Рисунок 3.12 – Додавання перевірки JWT токену на бекенді

Метод AddSecurity є розширюючим методом для інтерфейсу IServiceCollection, який налаштовує аутентифікацію та авторизацію в додатку на основі JWT (JSON Web Token). Цей метод використовується для додавання налаштувань безпеки в контейнер служб залежностей. Ось що робить цей метод, детально:

Спочатку метод отримує значення допустимого видавця (issuer) з конфігурації. Це робиться через звернення до секції конфігурації, що відповідає параметрам Cognito, і отримання значення поля Issuer.

Далі, використовуючи цей допустимий видавець, метод отримує ключі підпису для JWT за допомогою допоміжного класу JwtHelper. Ці ключі використовуються для перевірки підпису токена.

Потім метод додає та налаштовує аутентифікацію в колекцію служб. Він вказує, що використовується схема аутентифікації JwtBearer. Налаштування JwtBearer включають:

- IssuerSigningKeyResolver: ця функція використовується для отримання ключів підпису токена. Вона повертає раніше отримані ключі.
- ValidIssuer: вказує на допустимого видавця токена.
- ValidateIssuerSigningKey: дозволяє перевірку підпису токена.
- ValidateIssuer: дозволяє перевірку видавця токена.
- ValidateLifetime: дозволяє перевірку часу життя токена.
- ValidateAudience: вимикає перевірку аудиторії токена, тобто, кому призначений токен.
- ClockSkew: встановлює зсув часу на нуль, що означає, що не допускається жодних відхилень у часі при перевірці часу життя токена.

Після налаштування аутентифікації, метод додає авторизацію до колекції служб, що дозволяє керувати доступом на основі політик та ролей.

Насамкінець, метод повертає модифіковану колекцію служб для подальшого використання.

Таким чином, метод AddSecurity налаштовує аутентифікацію та авторизацію для додатку, використовуючи JWT токени та параметри безпеки, зчитані з конфігурації.

3.4 Розробка модуля сповіщень про екстрені ситуації шляхом SMS

Для відправки SMS повідомлень було використано сервіс Sinch [20].

Sinch SMS — це одна з основних послуг, яку надає платформа Sinch. Вона дозволяє бізнесам легко інтегрувати функціональність SMS у свої додатки та системи. API для відправки SMS: Sinch надає простий у використанні API, який дозволяє відправляти текстові повідомлення на будь-який мобільний номер у світі. API підтримує як одиничні повідомлення, так і масові розсилки. Партнерські відносини з багатьма операторами зв'язку по всьому світу, дозволяють забезпечувати доставку SMS у більш ніж 200 країн та територій.

Для комунікації з API Sinch також використовувалась AWS Lambda (рис. 3.13)

Рисунок 3.13 – AWS lambda для відправки SMS повідомлень

Ця функція є частиною програмного коду написаною на мові програмування C#, яка призначена для обробки подій, що надходять у чергу Amazon SQS (Simple Queue Service) та надсилання SMS повідомлень через Sinch, використовуючи їх API.

На початку класу Function маємо конструктор, який ініціалізує змінні _sinchClient та _stopFlag. Змінна _sinchClient відповідає за клієнт для взаємодії з API Sinch, тоді як _stopFlag використовується для перевірки наявності необхідних конфігураційних даних для підключення до Sinch. Якщо хоча б один з параметрів, таких як SINCH_PROJECT_ID, SINCH_KEY_ID, або SINCH_KEY_SECRET, не існує або є порожнім, то _stopFlag встановлюється в true, що означає, що Sinch не буде налаштовано та функція буде припинено.

Метод FunctionHandler викликається при отриманні нових подій з черги SQS. Перевіряється, чи встановлено значення _stopFlag. Якщо так, то функція реєструє критичне повідомлення у журналі та припиняє свою роботу. Якщо _stopFlag не встановлено, то функція переходить до обробки кожного повідомлення, що було отримано з черги.

Метод ProcessMessage призначений для обробки окремого повідомлення. Він витягує текст повідомлення та розпаковує його у відповідний об'єкт SmsRequest. Після цього створюється об'єкт SendTextBatchRequest для надсилання SMS через Sinch, використовуючи дані з отриманого повідомлення. Відправлення SMS відбувається через _sinchClient. Після успішної відправки SMS відповідь серіалізується у формат JSON та записується до журналу.

Отже, ця функція виконує важливу роль у обробці подій Amazon SQS та надсиланні SMS повідомлень через Sinch, забезпечуючи таким чином комунікацію у системі.

3.5 Висновок до розділу 3

У цьому розділі аргументовано вибір мови програмування для реалізації проекту, та пояснено основні етапи розробки.

Отже було обрано мову програмування C#, зважаючи на її основні переваги: статична типізація, повна підтримка об’єктно-орієнтованого програмування, наявність великої кількості інструментів для розробки такого застосунку.

Розглянуто розроблену систему для автоматичного розгортання веб серверу на віддаленій віртуальній машині типу EC2.

Також було розглянуто розробку та налаштування авторизації, відправку email та SMS повідомлень про надзвичайні ситуації. Наведено частини програмної реалізації для кожного модуля. Описано використання нових сторонніх засобів, які використовувались при розробці.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ ЕКСТРЕНОЇ ДОПОМОГИ

4.1 Методи тестування програмного забезпечення

До основних методів тестування належать:

- Юніт-тестування. Тестування окремих модулів або компонентів коду.
- Інтеграційне тестування. Перевірка взаємодії між різними модулями чи компонентами.
- Системне тестування. Тестування всієї системи як єдиного цілого.
- Функціональне тестування. Перевірка, чи відповідає система вимогам користувача або бізнесу.
- Тестування регресії. Перевірка системи після внесення змін для забезпечення того, що новий код не вплинув негативно на існуючий функціонал.
- Тестування надійності. Визначення здатності системи працювати без збоїв протягом певного часу.
- Тестування безпеки. Оцінка системи на вразливості, загрози та ризики безпеки.
- Альфа- та бета-тестування. Перший етап тестування, що виконується розробниками або спеціалізованою групою тестувальників в рамках організації розробника. Тестування системи обмеженою групою користувачів в реальних умовах перед остаточним випуском продукту.

Для тестування розроблених модуль проведемо лише функціональне тестування. Оскільки цей тип дозволить впевнитися у коректності роботи розроблених модулів.

4.2 Функціональне тестування розробленого програмного продукту

Для перевірки коректності роботи програми, є можливість протестувати роботу локально за допомогою HTTP запитів з використанням інструменту Postman.

Postman — це потужний інструмент для тестування API (Application Programming Interface), який дозволяє розробникам легко надсилати HTTP-запити та отримувати відповіді. Postman надає зручний інтерфейс для створення та управління запитами, автоматизації тестування та спрощення співпраці між командами.

Основні можливості Postman:

- Створення та надсилання запитів. Підтримка різних типів запитів: GET, POST, PUT, DELETE, PATCH, HEAD та інші. Можливість додавання параметрів запиту, заголовків (headers), тіла (body) запиту, авторизації та інших налаштувань. Можливість додавання тестових скриптів та попередніх скриптів (pre-request scripts) до колекцій.
- Колекції запитів. Колекції дозволяють групувати пов'язані запити для організації та повторного використання.
- Автоматизація тестування. Можливість написання тестів за допомогою JavaScript для автоматизації перевірок відповіді API.

Отож для проведення функціонального тестування для розробленої програми достатньо надіслати HTTP запит до віддаленого серверу та перевірити відповідь.

Розглянемо запити до ендпоінтів для виконання операцій з користувачем, результати роботи представлені на рисунках 4.1-4.3.

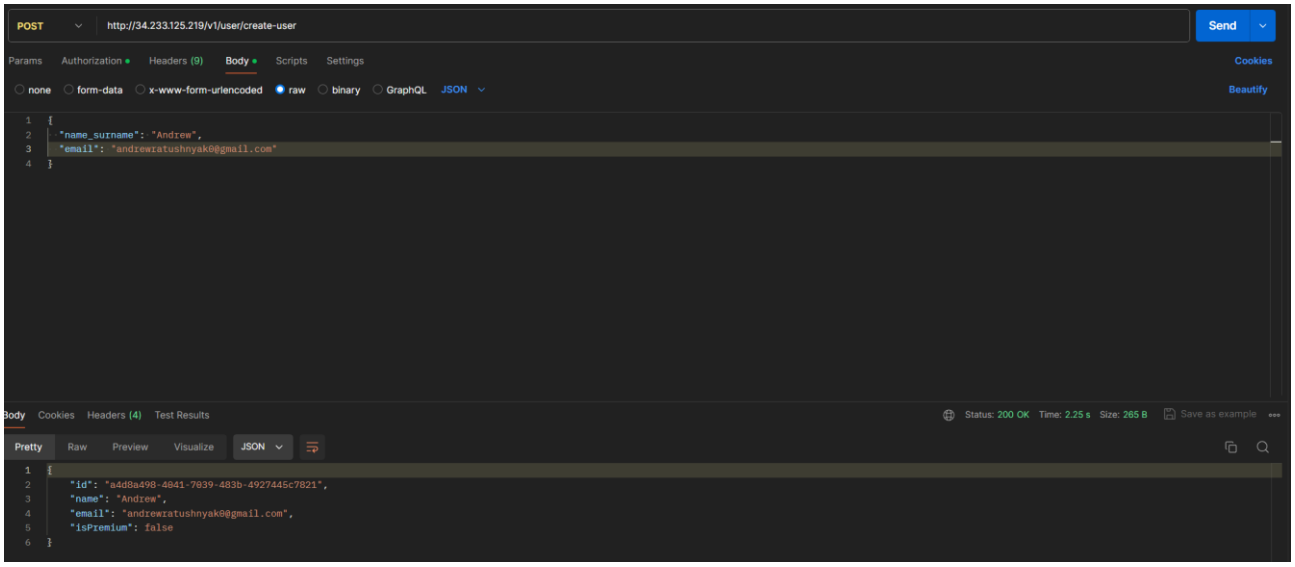


Рисунок 4.1 – Результат запиту до методу CreateUser

Тут виконується POST-запит на сервер, для створення нового запису про користувача. Передається JSON-об’єкт(ім’я, електронна адреса), як тіло запиту, в якості унікального ідентифікатора виступає поле “sub”, що містить у JWT токену (рис. 4.2). В результаті отримуємо повідомлення про успішну реєстрацію та дані користувача.

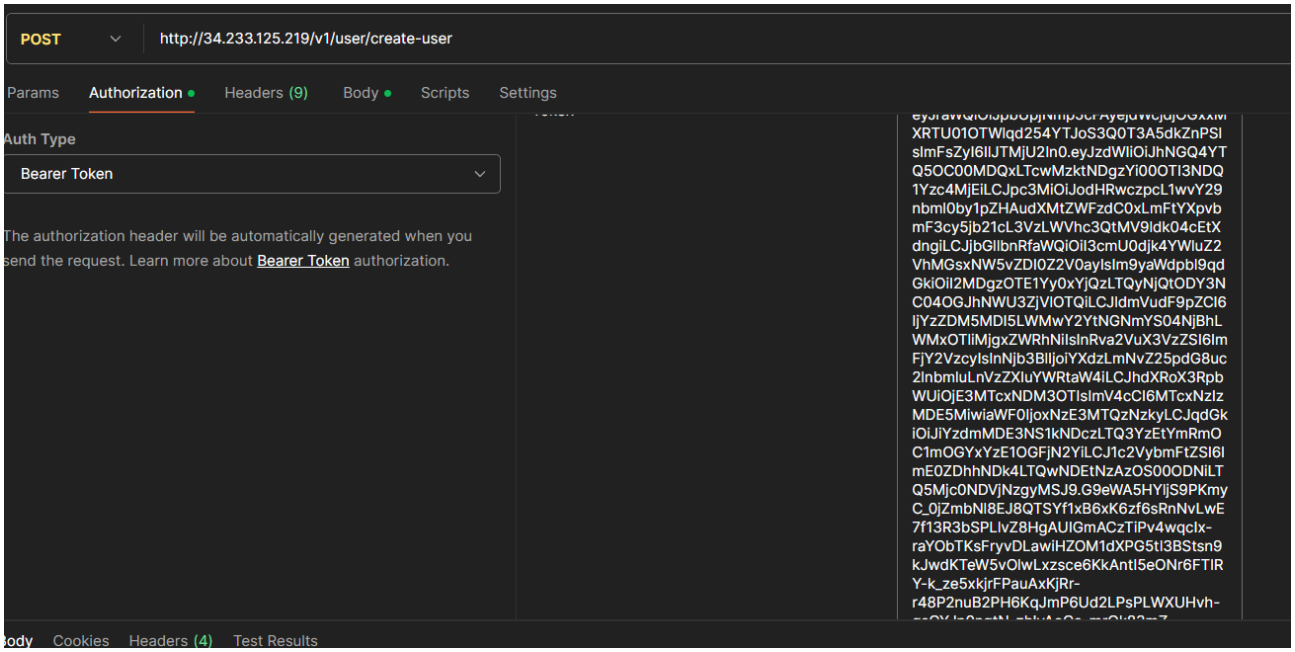


Рисунок 4.2 – Додавання JWT токену до запиту

Для отримання інформації про користувача – рисунок 4.3 – достатньо надіслати GET запит на сервер, взагалі без параметрів. Оскільки Id користувача, для якого здійснюється пошук інформації знаходиться в JWT токену.

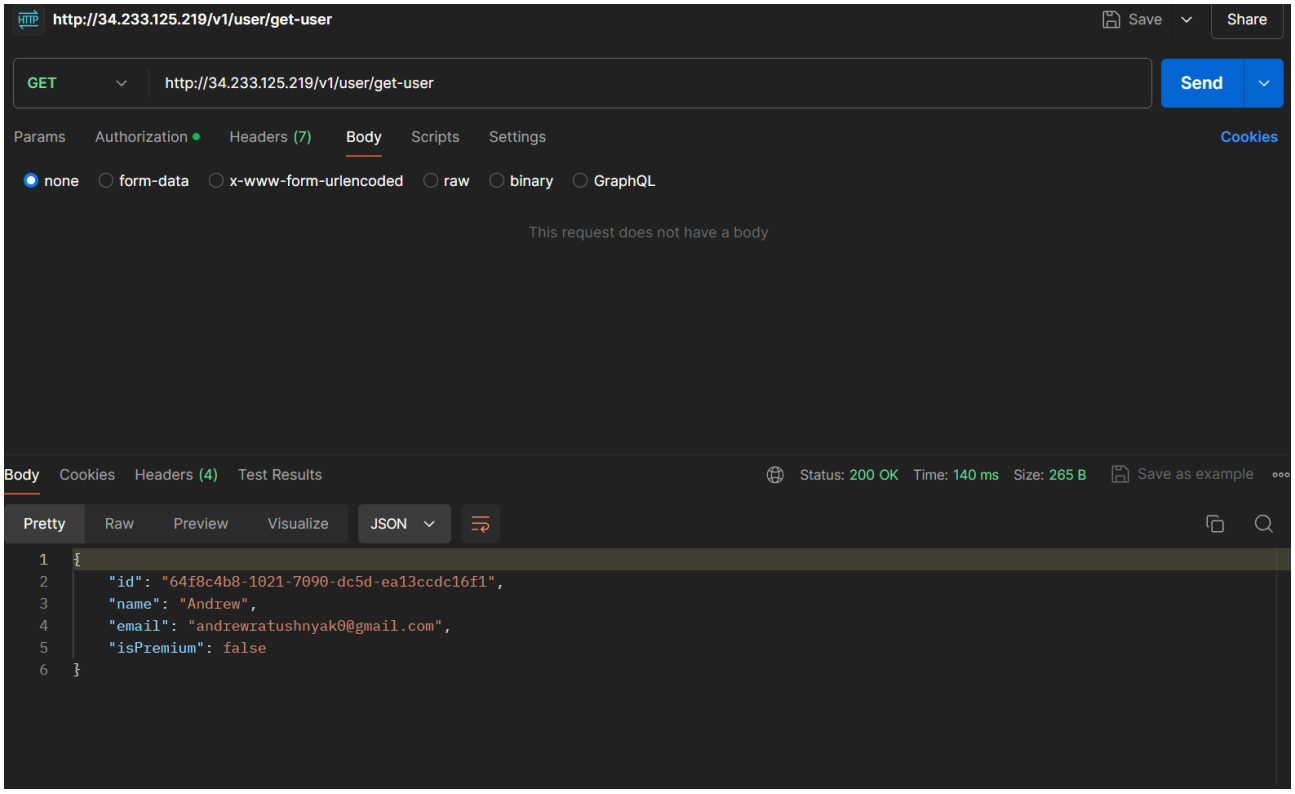


Рисунок 4.3 – Отримання інформації про користувача.

Після успішного створення користувача, потрібно перевірити чи є можливість управління довіреними контактами. Результати цієї перевірки наведені на рисунках 4.3 – 4.4.

Для створення екстреного контакту слід відправити POST запит на сервер, вказавши у тілі запиту – назву контакту, і опціонально номер телефону, пошту або телеграм контакт – рисунок 4.3.

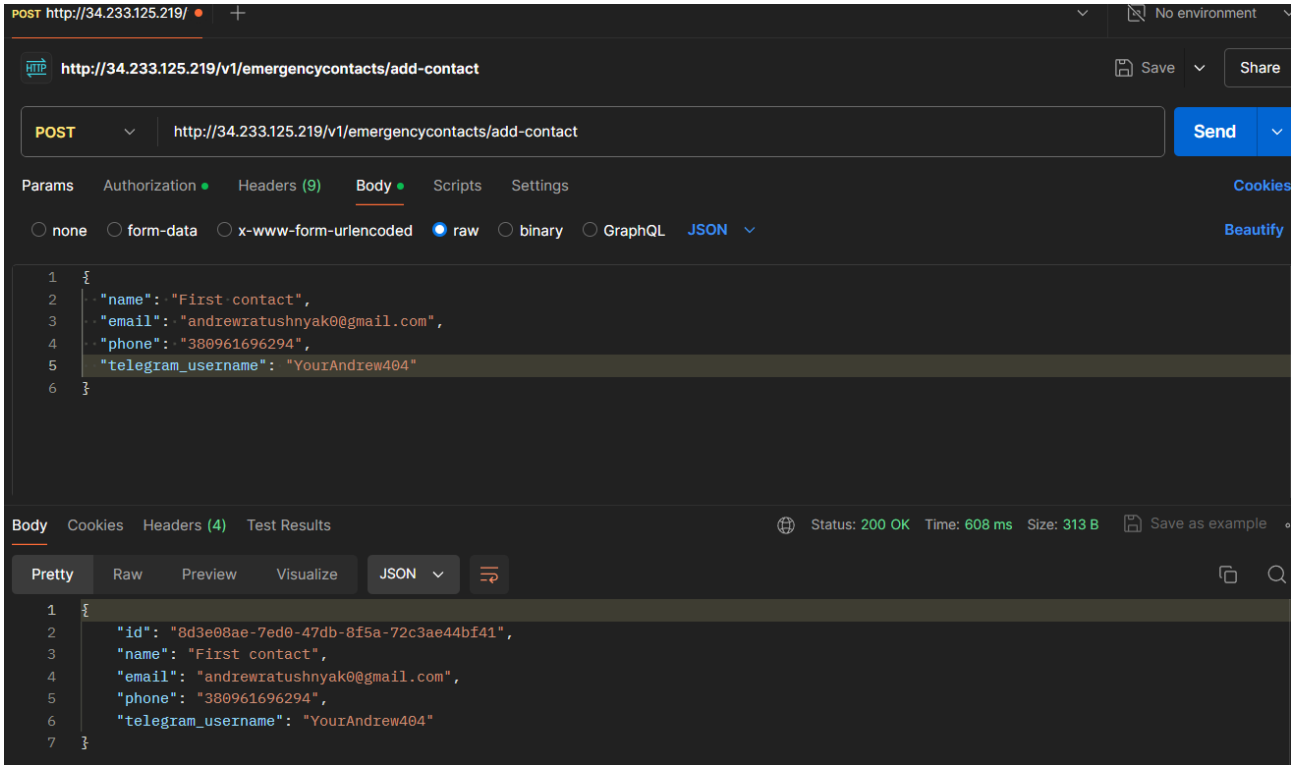


Рисунок 4.3 – Створення екстреного контакту

Для отримання контакту достатньо відправити GET запит та передати у якості параметра Id контакту – рисунок 4.4.

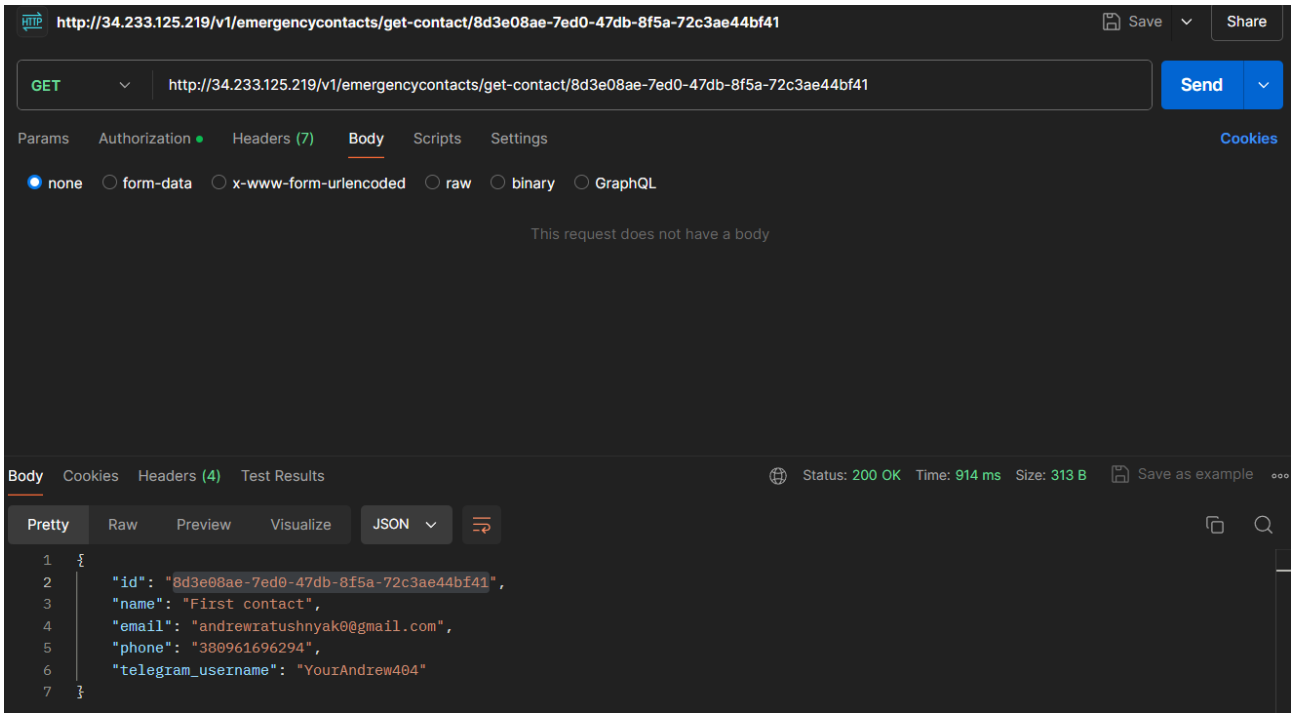


Рисунок 4.4 – Отримання інформації про екстрений контакт

Після створення хоча б одного з контактів – протестуємо надсилання екстрених повідомлень. Для цього в програмі передбачений POST запит – рисунок 4.5.

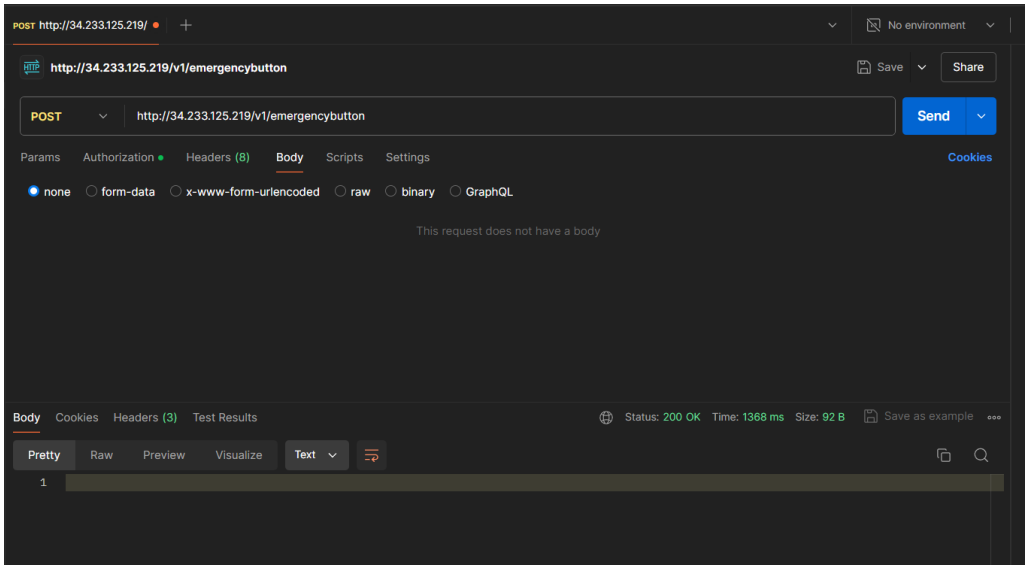
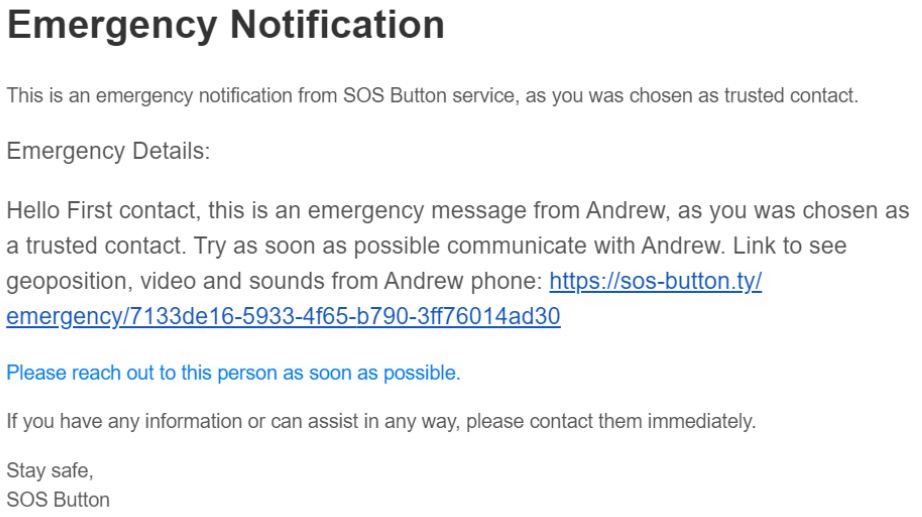


Рисунок 4.5 – Надсилання екстрених повідомлень

При відправленні цього запиту перевіримо кінцеві точки, куди були надіслані повідомлення, а саме електрону пошту, SMS та телеграм – рисунки 4.6-4.8.



Emergency Notification

This is an emergency notification from SOS Button service, as you was chosen as trusted contact.

Emergency Details:

Hello First contact, this is an emergency message from Andrew, as you was chosen as a trusted contact. Try as soon as possible communicate with Andrew. Link to see geoposition, video and sounds from Andrew phone: https://sos-button.ty/emergency/7133de16-5933-4f65-b790-3ff76014ad30

Please reach out to this person as soon as possible.

If you have any information or can assist in any way, please contact them immediately.

Stay safe,
SOS Button

Рисунок 4.6 – Екстрене повідомлення на електронній пошті

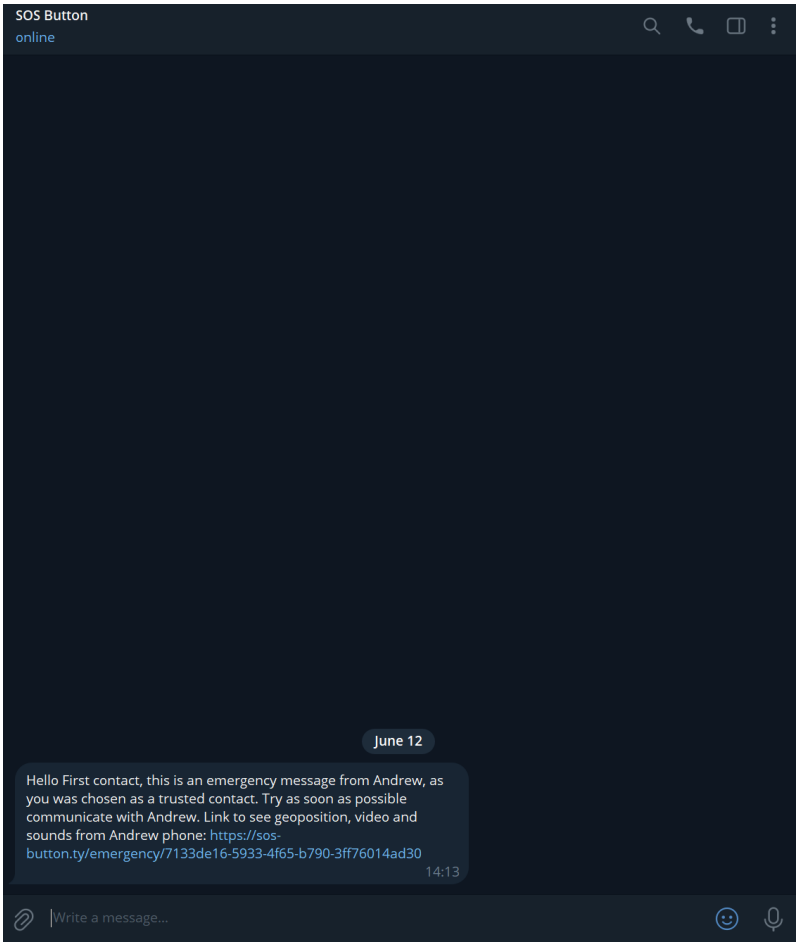


Рисунок 4.7 – Екстрене повідомлення у телеграмі

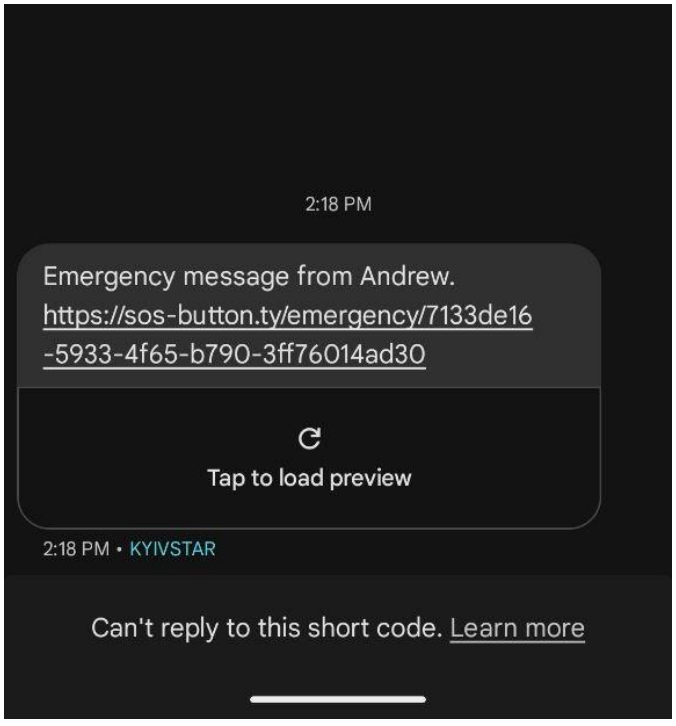


Рисунок 4.8 – Екстрене повідомлення у SMS.

4.3 Висновок до розділу 4

У четвертому розділі були розглянуті різні аспекти тестування, включаючи реєстрацію користувача, отримання інформації про користувача, управління довіреними контактами та надсилання екстрених повідомлень.

Тестування реєстрації користувача: перевірено коректність виконання POST-запиту з передачею JSON-об’єкту, що містить ім’я та електронну адресу, а також забезпечено успішну реєстрацію нового користувача з використанням JWT токену для ідентифікації. Тестування отримання інформації про користувача: перевірено правильність виконання GET запиту без параметрів, що використовує Id користувача з JWT токену для отримання даних.

Тестування управління довіреними контактами: перевірено правильність створення екстреного контакту через POST-запит з передачею назви контакту та опціонально інших даних (номер телефону, пошта, телеграм), а також перевірено отримання екстреного контакту через GET запит з передачею Id контакту.

Тестування надсилання екстрених повідомлень: перевірено коректність виконання POST-запиту для надсилання екстрених повідомлень на електронну пошту, SMS та телеграм. Результати тестування показали, що функціонал додатку працює відповідно до специфікацій і забезпечує коректну взаємодію з користувачем.

Усі перевірені функції працюють належним чином.

ВИСНОВКИ

У бакалаврській дипломній роботі було розглянуто розробку сервісу екстреної допомоги з акцентом на серверну частину. Основними етапами дослідження були аналіз предметної області, проектування системи, розробка серверної частини та її тестування.

Аналіз предметної області показав високу актуальність розробки подібних сервісів в умовах стрімкої урбанізації, зростання кількості надзвичайних ситуацій та необхідності оперативного реагування на них. В результаті було сформульовано вимоги до сервісу, які забезпечують його функціональність та надійність.

Проектування системи включало вибір відповідних технологій, таких як мікросервіси, хмарні обчислення та використання ASP.NET для розробки серверної частини. Було розроблено архітектуру, яка включає хмарну інфраструктуру на базі Amazon Web Services (AWS), що забезпечує відмовостійкість та масштабованість сервісу.

Розробка серверної частини сервісу включала реалізацію основних функціональних модулів, таких як автоматичне розгортання, авторизація та аутентифікація за допомогою AWS Cognito, а також модуль сповіщень про екстрені ситуації. Використання мови програмування C# та середовища розробки Rider дозволило створити ефективний та надійний серверний додаток.

Тестування розробленої серверної частини показало, що всі функціональні модулі працюють коректно і відповідають вимогам специфікації. Було перевірено коректність валідації даних, функціональність реєстрації та авторизації користувачів, а також відправлення сповіщень про екстрені ситуації. Результати тестування підтвердили, що серверний додаток забезпечує надійну та безперебійну роботу сервісу.

Отже, мета дослідження була досягнута: створено серверну частину сервісу екстреної допомоги, яка відповідає сучасним вимогам до функціональності, надійності та безпеки. Результати цієї роботи можуть бути використані для подальшого розвитку та вдосконалення систем екстреної допомоги, що сприятиме підвищенню рівня безпеки та оперативності реагування на надзвичайні ситуації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Андрій Ратушняк, Максим Целік, Руслан Белзецький. “Використання платформи хмарних обчислень Amazon Web Services для розробки додатку екстренної допомоги” в Матеріалах конференції “LIII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)”, Вінниця, 2024. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20910.
2. Pixel Personal Safety [Електронний ресурс]. Режим доступу: https://www.androidpolice.com/pixel-personal-safety-app-explainer/.
3. Додаток Life360 [Електронний ресурс]. Режим доступу: https://www.life360.com/intl/.
4. Continuous Integration Continuous Delivery [Електронний ресурс]. Режим доступу: https://highload.today/uk/blogs/shho-take-ci-cd-yak-vin-pratsyuye-ta-koli-znadobitsya-na-proyekti-lajfhaki-ta-bad-practices/.
5. Github Actions [Електронний ресурс]. Режим доступу: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
6. Docker [Електронний ресурс]. Режим доступу: https://www.docker.com/
7. Secure Shell [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/SSH
8. Amazon Web Services [Електронний ресурс]. Режим доступу: https://aws.amazon.com/?nc1=h_ls
9. Google Cloud Provider [Електронний ресурс]. Режим доступу: https://cloud.google.com/
10. AWS vs Azure vs GCP [Електронний ресурс]. Режим доступу: https://cloudfresh.com/ua/cloud-blog/aws-vs-azure-vs-google-cloud/
11. AWS Lambda [Електронний ресурс]. Режим доступу: https://aws.amazon.com/lambda/

12. Elastic Compute Cloud [Електронний ресурс]. Режим доступу: https://aws.amazon.com/ec2/
13. Json Web Token [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/JSON_Web_Token
14. NGINX Cookbook: Advanced Recipes for High-Performance Load Balancing 3rd Edition // Derek DeJonghe, 2024. 52c.
15. PostgreSQL [Електронний ресурс]. Режим доступу: https://steven-giesel.com/blogPost/
16. The best server side language [Електронний ресурс]. Режим доступу: https://www.qulix.com/about/blog/the-best-server-side-language/
17. Брюс У. Перри. Java сервлеты и JSP. Сборник рецептов [Електронний ресурс]/ Брюс У. Перри — Пер. С англ. — КУДИЦ-Пресс, 2009 — 768 с. — Режим доступу: https://www.ozon.ru/context/detail/id/4434287
18. Троелсен Е. Мова програмування C# 9 і платформа .NET 5. Бібліотека програміста / Е. Троелсен. – Діалектика, 2021. – 126 с
19. Github Actions Secrets [Електронний ресурс]. Режим доступу: https://docs.github.com/en/actions/security-guides/using-secrets-in-github-actions
20. Postmark [Електронний ресурс]. Режим доступу: https://postmarkapp.com/
21. Sinch [Електронний ресурс]. Режим доступу: https://www.sinch.com/

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного сервісу екстреної допомоги. Частина 1. Серверна частина

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІІТА
(кафедра, факультет)

5 Показники звіту подібності Unicheck

Оригінальність 94,36% Схожість 5,64%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Ратушняк А.В.

Керівник роботи Белзецький Р.С.

ДОДАТОК Б

ЛІСТИНГИ ПРОГРАМНОГО КОДУ

Лістинг Б.1 – Електронний лист для підтвердження пошти

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Email Confirmation</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      margin: 0;
      padding: 0;
    }

    .container {
      max-width: 600px;
      margin: 30px auto;
      padding: 20px;
      background-color: #ffffff;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    h1 {
      color: #333333;
    }

    p {
      color: #555555;
    }

    .confirmation-code {
      font-size: 1.5em;
      color: #007bff;
      margin-top: 10px;
    }

    .expiration-info {
      color: #999999;
      margin-top: 10px;
    }

    .cta-button {
      display: inline-block;
      padding: 10px 20px;
      background-color: #007bff;
```

```
      color: #ffffff;
      text-decoration: none;
      border-radius: 5px;
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Email Confirmation</h1>
    <p>Thank you for signing up! To complete your registration, please use the following
confirmation code:</p>
    <div class="confirmation-code">
     {{ confirmation_code }}
    </div>
    <p class="expiration-info">This code will be valid for 12 hours.</p>
    <p>If you didn't sign up for this service, please ignore this email.</p>
    <p>Best regards,<br>SOS Button App</p>
  </div>
</body>
</html>
```

Лістинг Б.2 – Контролер для додавання довірених контактів

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SButton.Core.Helpers;
using SButton.Core.Services.Interfaces;
using SButton.Web.DTO.Requests;
using SButton.Web.DTO.Responses;

namespace SButton.Web.Controllers
{
    [ApiController]
    [Route("v1/[controller]/")]
    [Authorize]
    public class EmergencyContactsController : ControllerBase
    {
        private readonly IEmergencyContactService _emergencyContactService;

        public EmergencyContactsController(IEmergencyContactService emergencyContactService)
        {
            _emergencyContactService = emergencyContactService;
        }

        [HttpPost]
        [Route("add-contact")]
        public async Task<IActionResult> AddEmergencyContact([FromBody]
EmergencyContactRequest request)
        {
```

```
        if (string.IsNullOrEmpty(string.Concat(request.Email, request.TelegramUsername,
request.Phone)))
            return BadRequest(ApiResponseObject.Fail("You must provide at least one method to
communicate with contact"));

        var userSub = User.GetUserSub();
        var resultContact = await _emergencyContactService.CreateEmergencyContact(userSub,
request.ToModel());

        if (resultContact.IsFailure)
            return BadRequest(ApiResponseObject.Fail(resultContact.Error));

        return Ok(EmergencyContactResponse.FromModel(resultContact.Value));
    }

    [HttpDelete]
    [Route("delete-contact/{id:guid}")]
    public async Task<IActionResult> DeleteEmergencyContact([Required] Guid id)
    {
        var userSub = User.GetUserSub();
        var resultContact = await _emergencyContactService.RemoveEmergencyContact(userSub,
id);

        if (resultContact.IsFailure)
            return BadRequest(ApiResponseObject.Fail(resultContact.Error));

        return Ok();
    }

    [HttpGet]
    [Route("get-contact/{id:guid}")]
    public async Task<IActionResult> GetEmergencyContact([Required] Guid id)
    {
        var userSub = User.GetUserSub();
        var resultContact = await _emergencyContactService.GetContact(userSub, id);

        if (resultContact.IsFailure)
            return BadRequest(ApiResponseObject.Fail(resultContact.Error));

        return Ok(EmergencyContactResponse.FromModel(resultContact.Value));
    }

    [HttpPut]
    [Route("update-contact/{id:guid}")]
    public async Task<IActionResult> UpdateEmergencyContact([FromBody]
EmergencyContactRequest request, [FromRoute] [Required] Guid id)
    {
        if (string.IsNullOrEmpty(string.Concat(request.Email, request.TelegramUsername,
request.Phone)))
            return BadRequest(ApiResponseObject.Fail("You must provide at least one method to
communicate with contact"));
```

```
            var userSub = User.GetUserSub();
            var emergencyContactModel = request.ToModel();
            emergencyContactModel.Id = id;

            var resultContact = await _emergencyContactService.UpdateEmergencyContact(userSub,
emergencyContactModel);

            if (resultContact.IsFailure)
                return BadRequest(ApiResponseObject.Fail(resultContact.Error));

            return Ok(EmergencyContactResponse.FromModel(resultContact.Value));
        }

        [HttpGet]
        [Route("short-list")]
        public async Task<IActionResult> ShortList()
        {
            var userSub = User.GetUserSub();
            var resultContactsList = await _emergencyContactService.ListContacts(userSub);

            if (resultContactsList.IsFailure)
                return BadRequest(ApiResponseObject.Fail(resultContactsList.Error));

            return Ok(new GetEmergencyContactShortList(resultContactsList.Value));
        }
    }
}
```

Лістинг Б.3 – Сервіс для комунікації через телеграм

```
using CSharpFunctionalExtensions;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using SButton.Core.Options;
using SButton.Core.Services.Interfaces;
using TdLib;

namespace SButton.Core.Services
{
    public class TelegramService : ITelegramService
    {
        private readonly TdClient _client;
        private readonly ILogger<TelegramService> _logger;
        private readonly TelegramOptions _options;
        private readonly ManualResetEventSlim _readyToAuthenticate = new();

        private bool _authNeeded;
        private bool _passwordNeeded;

        public TelegramService(TdClient client, IOptions<TelegramOptions> options,
ILogger<TelegramService> logger)
        {
```

```
        _client = client;
        _logger = logger;
        _options = options.Value;

        _client.Bindings.SetLogVerbosityLevel(0);
        _client.UpdateReceived += async (_, update) => { await ProcessUpdates(update); };

        StartAuthorizationFlow().Wait();
    }

    public async Task<bool> IsUserExists(string userName)
    {
        var chat = await _client.SearchPublicChatAsync(userName);
        return true;
    }

    public async Task<Result> EnterConfirmationCodeAndPassword(string confirmationCode,
string password = null)
    {
        _readyToAuthenticate.Wait();
        if (!_authNeeded)
            return Result.Failure("Auth is not needed");

        if (string.IsNullOrEmpty(confirmationCode))
            return Result.Failure("Confirmation code is not provided");

        await _client.CheckAuthenticationCodeAsync(confirmationCode);

        if (!_passwordNeeded)
            return Result.Success();

        if (string.IsNullOrEmpty(password))
            return Result.Failure("Password is not provided");

        await _client.ExecuteAsync(new TdApi.CheckAuthenticationPassword { Password =
password });

        return Result.Success();
    }

    private async Task StartAuthorizationFlow()
    {
        _readyToAuthenticate.Wait();

        if (!_authNeeded)
        {
            _logger.LogInformation("Auth is not needed");
            return;
        }

        _logger.LogInformation("Auth is needed");
```

```
            await _client.ExecuteAsync(new TdApi.SetAuthenticationPhoneNumber { PhoneNumber =
_options.PhoneNumber });

            _authNeeded = true;
        }

        private async Task ProcessUpdates(TdApi.Update update)
        {
            switch (update)
            {
                case TdApi.Update.UpdateAuthorizationState { AuthorizationState:
TdApi.AuthorizationState.AuthorizationStateWaitTdlibParameters }:
                {
                    var filesLocation = Path.Combine(AppContext.BaseDirectory, "db");

                    await _client.ExecuteAsync(new TdApi.SetTdlibParameters
                    {
                        ApiId = Convert.ToInt32(_options.ApiId),
                        ApiHash = _options.ApiHash,
                        DeviceModel = "PC",
                        SystemLanguageCode = "en",
                        ApplicationVersion = "1.0.0",
                        DatabaseDirectory = filesLocation,
                        FilesDirectory = filesLocation
                    });
                    break;
                }

                case TdApi.Update.UpdateAuthorizationState { AuthorizationState:
TdApi.AuthorizationState.AuthorizationStateWaitPhoneNumber }:
                case TdApi.Update.UpdateAuthorizationState { AuthorizationState:
TdApi.AuthorizationState.AuthorizationStateWaitCode }:
                    _authNeeded = true;
                    _readyToAuthenticate.Set();
                    break;

                case TdApi.Update.UpdateAuthorizationState { AuthorizationState:
TdApi.AuthorizationState.AuthorizationStateWaitPassword }:
                    _authNeeded = true;
                    _passwordNeeded = true;
                    _readyToAuthenticate.Set();
                    break;

                case TdApi.Update.UpdateUser:
                    _readyToAuthenticate.Set();
                    break;
            }
        }
    }
}
```

ДОДАТОК Г.

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА СЕВРВІСУ ЕКСТРЕННОЇ ДОПОМОГИ ЧАСТИНА 1. СЕРВЕРНА ЧАСТИНА

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ратушняк А. В.
(прізвище та ініціали)
Керівник:
Белзецький Р.С.
(прізвище та ініціали)
«01» 06 2024 р.

Вінниця ВНТУ – 2024 рік

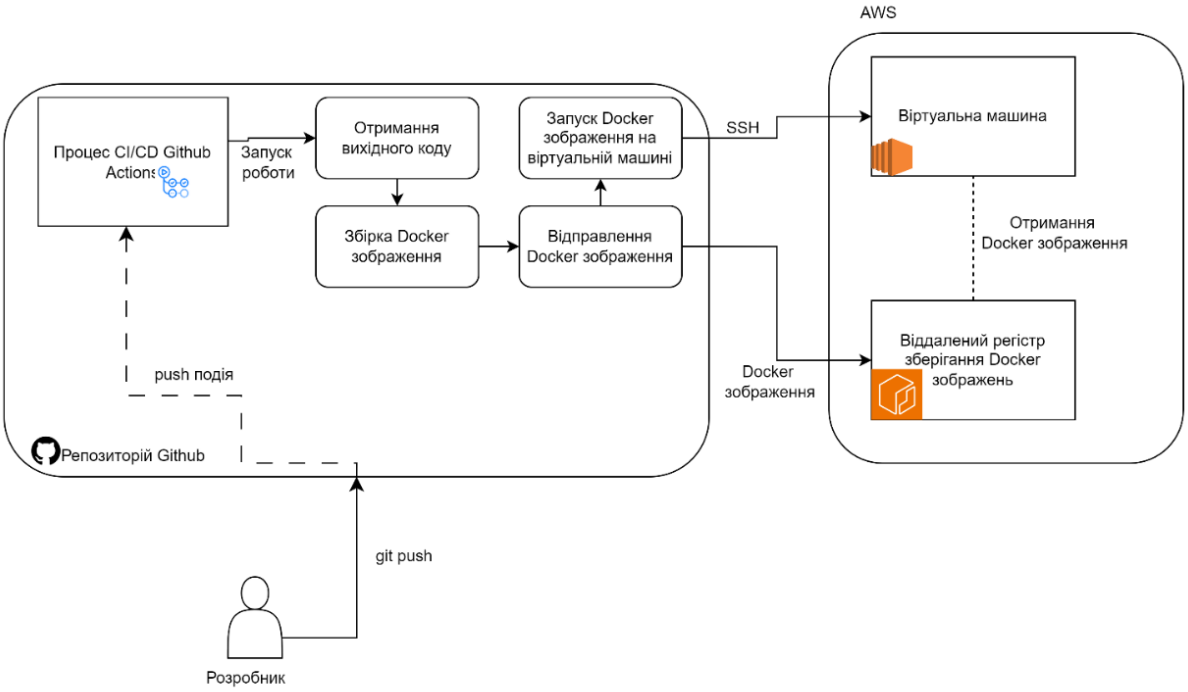


Рисунок Г.1 – Система розгортання додатку

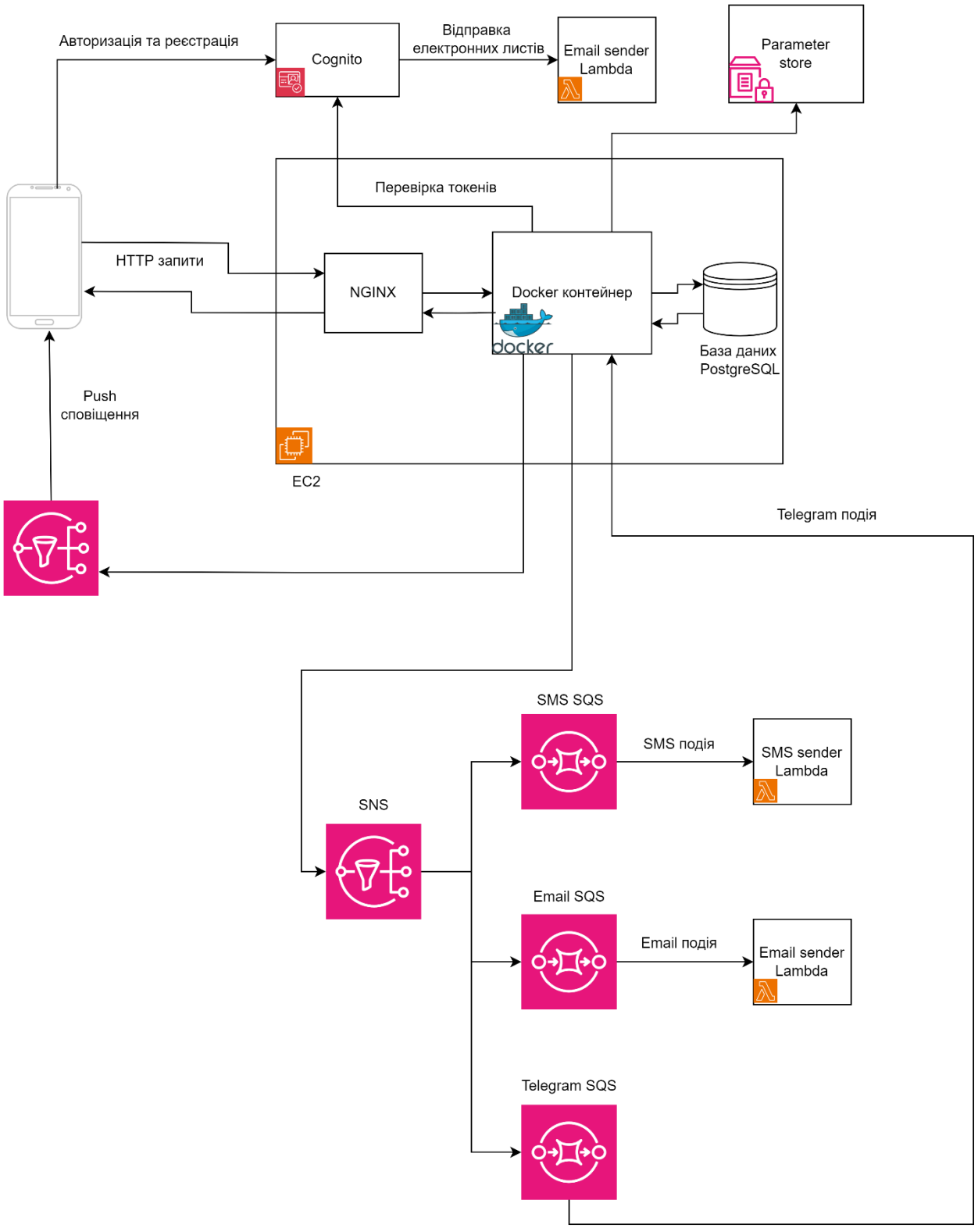


Рисунок Г.2 – Архітектура сервісу