

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці**

Освітньо-професійна програма «КОМП'ЮТЕРНІ НАУКИ»

галузь знань 12 «Інформаційні технології»

спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

**РОЗРОБЛЕННЯ БОТА ДЛЯ ЕКСТРЕНОГО ОПОВІЩЕННЯ
КОРИСТУВАЧІВ**

здобувача Чернухи Романа Юрійовича _____

Науковий керівник:

к.е.н., проф.

_____ Лозовик Ю. М.

**Проект допущений до захисту
перед екзаменаційною комісією з
атестації здобувачів вищої освіти
Завідувач кафедри: к.е.н., проф.**

КИЇВ КНЕУ 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці
ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»

галузь знань 12 Інформаційні технології

спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:	ЗАТВЕРДЖУЮ:
Керівник проєктної групи (гарант)	Завідувач кафедри
освітньо-професійної програми	інформаційних систем в економіці
«____» _____ 202_ р.	«____» _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти _____ Чернухи Романа Юрійовича
(прізвище, ім'я, по батькові)

денної _____ форми навчання
очної (денної), заочної, дистанційн

на підготовку кваліфікаційного бакалаврського проєкту

на тему: «Розроблення бота для екстреного оповіщення користувачів»

Тему затверджено наказом ректора Університету від «07» _03 2025 р. №466-ст
Кваліфікаційний бакалаврський проєкт виконується на матеріалах усіх
доступних публікацій та Інтернет джерел

План кваліфікаційного бакалаврського проєкту

Розділ I. Характеристика та аналіз предметної галузі

(назва розділу)

Розділ II. Розробка вимог і моделювання інформаційної системи

(назва розділу)

Розділ III. Проєктування та реалізація компонентів системи

(назва розділу)

Об'єкт дослідження: процес організації та управління системою аварійного оповіщення через централізований Messenger-бот, що охоплює реєстрацію

користувачів, формування груп отримувачів, створення шаблонів повідомлень та масову доставку сповіщень у критичних ситуаціях.

Предмет дослідження: методи та технології побудови інформаційної системи

аварійного оповіщення на базі монолітного застосунку на Java із використанням фреймворків та Messenger Bot API для надійної й оперативної взаємодії з користувачами.

Мета кваліфікаційної роботи: створити та реалізувати прототип інформаційної системи ENS Bot, здатний забезпечувати швидке та адресне інформування заздалегідь визначених груп користувачів у випадку надзвичайних подій без участі централізованих адміністраторів.

Конкретні завдання для досягнення мети:

У розділі I

ослідження і характеристика предметної області аварійних та екстрених комунікацій. Збір інформації та вивчення матеріалів за тематикою кваліфікаційного бакалаврського проекту.

ивчення існуючих підходів до оповіщення в месенджерах та їхні обмеження щодо масштабованості, простоти керування і гарантій доставки.

наліз літературних джерел, практичного досвіду та існуючих підходів до організації інформаційних пошукових систем в сфері аварійних та екстрених комунікацій.

У розділі II

наліз і специфікація бізнес-, функціональних та нефункціональних вимог до системи аварійного оповіщення через месенджер.

обудова діаграм вимог та прецедентів (Use Case) для ключових операцій: реєстрація, створення груп, формування шаблонів і відправка екстрених сповіщень. остановка завдань і розробка алгоритмів обробки команд та масової доставки повідомлень у рамках монолітного застосунку.

арактеристика вхідних і вихідних повідомлень, структури даних та повідомлень, що передаються між користувачем і ботом.

оделювання інформаційної системи: побудова діаграм прецедентів, послідовностей та класів/блоків для компонентів ENS Bot.

У розділі III

изначення архітектури монолітного застосунку аварійного оповіщення ENS Bot із виділенням підсистем авторизації, управління шаблонами, групами, розсилками та моніторингу.

озробка інформаційного, програмного й технічного забезпечення прототипу на базі Java та Messenger Bot API.

творення компонентів системи – основних блоків і діаграми розгортання.

пробація реалізованого прототипу: перевірка коректності функціоналу, оцінка продуктивності та стабільності під навантаженням.

Завдання підготував

науковий керівник

Миколайович

(підпис)

Лозовик Юрій

(ініціали, прізвище)

«__» __ лютого ____ 2025 р.

Завдання одержав

з

д

(підпис)

(ініціали, прізвище)

о

б » __ лютого ____ 2025 р.

у

в

а

ч

ц

е

р

н

у

х

а

АНОТАЦІЯ

Кваліфікаційного бакалаврського проєкту студента 4 курсу

Навчально-наукового інституту

«Інститут інформаційних технологій в економіці»

Чернухи Романа Юрійовича, виконаної на тему:

Розроблення бота для екстреного оповіщення користувачів»

Київ: кафедра інформаційних систем в економіці, 2025 р.

У роботі спроектовано інформаційну систему «ENS Bot», яка забезпечує централізоване відправлення екстрених повідомлень користувачам через різні канали зв'язку - месенджер, SMS та електронну пошту. Система дозволяє користувачам створювати захищені паролем групи, формувати шаблони повідомлень, які автоматично доповнюються актуальним часом та географічною локацією відправника, і розсилати їх усім учасникам групи. Бекенд реалізовано на основі Messenger Bot API та фреймворку Java Spring Boot із використанням СУБД . Інтерфейс бота містить інлайн-кнопки для спрощення взаємодії користувача, передбачено централізовану базу логування подій, а також реалізовано багатоканальну розсилку повідомлень.

Робота складається з вступу, трьох розділів, висновків, списку літератури та двох додатків. У першому розділі обґрунтовано актуальність систем екстреного оповіщення та проведено аналіз існуючих рішень. Другий розділ присвячено проєктуванню архітектури інформаційної системи ENS Bot, опису структури бази даних і функціональних алгоритмів (створення груп, формування шаблонів повідомлень тощо). Третій розділ містить опис реалізації системи, налаштування бота з інлайн-кнопками та результати його тестування. У висновках сформульовано отримані результати, зазначено переваги розробленої системи та наведено рекомендації щодо її подальшого вдосконалення.

РЕФЕРАТ

Кваліфікаційний бакалаврський проєкт містить 90 сторінок, 10 таблиць, 49 рисунків, список літератури з 24 найменувань, 1 додаток.

РОЗРОБЛЕННЯ БОТА ДЛЯ ЕКСТРЕНОГО ОПОВІЩЕННЯ КОРИСТУВАЧІВ

Перелік ключових слів: месенджер, аварійне оповіщення, API, масові повідомлення, шаблони, групи база даних, Spring Boot, PostgreSQL, Java, бот.

Об'єктом дослідження є процес організації багатоканального екстреного сповіщення користувачів, предметом - інформаційна система для екстреного інформування на базі Messenger API. Метою роботи є створення інформаційної системи, що забезпечує ефективне формування та доставку повідомлень у надзвичайних ситуаціях через ENS Bot. Однією з ключових вимог до таких систем є максимально швидке поширення повідомлень з охопленням максимальної кількості користувачів. Розроблена система ENS Bot підтримує масову розсилку екстрених повідомлень через декілька каналів зв'язку (месенджер, SMS, електронну пошту) та передбачає функції створення захищених груп користувачів, формування шаблонів повідомлень, врахування геолокації при надсиланні та централізованого журналу доставки сповіщень. Задачі роботи включають: розробку Messenger-бота для комунікації з користувачами; реалізацію реєстрації та авторизації користувачів у системі; створення та керування групами користувачів із доступом за паролем; формування та збереження шаблонів повідомлень для різних сценаріїв сповіщення; надсилання екстрених повідомлень з урахуванням геолокації отримувачів; ведення централізованого журналу доставки та статусів сповіщень; забезпечення доставки повідомлень через кілька каналів зв'язку (месенджер, SMS,

У реалізації ENS Bot використано мову програмування Java та фреймворк Spring Boot, базу даних PostgreSQL, технологію контейнеризації Docker Compose, а також Messenger API для взаємодії з користувачами. Архітектура системи побудована за монолітним принципом з можливістю майбутнього переходу до

мікросервісної, що забезпечує гнучкість і масштабованість. Для спрощення розробки застосовано бібліотеки Lombok, JDBC та Spring Boot Mail API. У результаті створено функціональний прототип ENS Bot - бота, який реалізує розроблену систему екстреного оповіщення користувачів.

Рік виконання кваліфікаційного бакалаврського проєкту – 2025.

Рік захисту кваліфікаційного бакалаврського проєкту – 2025.

3MICT

B

P

Th

1.2 Аналіз літературних джерел та практичного досвіду використання ІС і

D

P

A

BI

R

N

D

IN

ERI



H

THE

FBI



H

ВСТУП

Оперативна комунікація у надзвичайних умовах стає дедалі більш критичною складовою інформаційної безпеки як окремих громадян, так і організацій. У сучасному світі, де темп змін є надзвичайно динамічним, а ризики часто виникають раптово і вимагають миттєвого реагування, важливо мати у розпорядженні інструменти, які дозволяють швидко й гарантовано донести повідомлення до широкого кола адресатів. Звичайні засоби комунікації, зокрема групові чати у популярних месенджерах, зазвичай не здатні забезпечити гарантовану доставку критичних повідомлень: сповіщення губляться серед загального потоку інформації, не прочитуються вчасно або взагалі залишаються непоміченими. Це створює серйозні загрози у випадках, коли кожна секунда має значення.

У відповідь на цю проблему було спроектовано та реалізовано ENS Bot - централізовану систему для автоматизованого багатоканального сповіщення користувачів. Основною ідеєю, яка лежить в основі розробки, є створення такого інструменту, який не вимагав би централізованого адміністрування, забезпечував би гнучкість, простоту та водночас високу надійність доставки. Користувачі ENS Bot можуть самостійно створювати групи доступу із захистом паролем, формувати шаблони повідомлень з урахуванням динамічних параметрів (таких як назва, вміст, час і геолокація), надсилати їх одним натисканням у кількох каналах одночасно: у месенджер, через SMS та електронну пошту.

Особливістю системи є відсутність адміністративної ієрархії: кожен учасник має однакові права щодо створення груп, шаблонів та ініціювання розсилки. Такий підхід значно підвищує гнучкість системи та адаптивність до різних сценаріїв використання - від індивідуальних потреб сімей або друзів до задач малого бізнесу чи проектних команд. Надзвичайно важливим елементом реалізованої системи є централізований лог-журнал, що фіксує усі факти надсилання повідомлень, їхній вміст, час та статус доставки. Це дозволяє як ретроспективний аналіз, так і контроль за ефективністю роботи системи в реальному часі.

ENS Bot побудовано з використанням мови програмування Java, фреймворку Spring Boot, бази даних PostgreSQL та інструментів контейнеризації Docker Compose. Взаємодія з користувачами реалізована через Messenger API, що забезпечує простий інтерфейс у вигляді інлайн-кнопок, реакцій на команди та дії. Архітектурно система реалізована у вигляді моноліту з чітко вираженою модулярністю, що дозволяє у майбутньому перевести її у мікросервісну структуру без суттєвих змін основної логіки.

Отже, об'єктом дослідження є процес організації багатоканального екстреного сповіщення користувачів. Предметом дослідження є інформаційна система для екстреного інформування на базі Messenger Bot API, яка реалізує цей процес. Мета роботи - створити інформаційну систему, що забезпечує ефективне формування та доставку повідомлень у надзвичайних ситуаціях через ENS Bot. Для досягнення поставленої мети вирішуються такі завдання:

- Аналіз вимог та предметної області: вивчення існуючих систем екстреного оповіщення, їх архітектури та функціональності;
- Проєктування системи: розробка архітектури, структур даних та основних компонентів майбутнього ENS Bot;
- Розробка функціоналу бота: реалізація модуля надсилання повідомлень через Messenger Bot API;
- Реалізація реєстрації та авторизації: забезпечення безпечної аутентифікації користувачів (використання логін-пароль, шифрування);
- Створення та управління групами: розробка механізму формування груп користувачів із доступом за паролем для диференційованого розсилання сповіщень;
- Формування шаблонів повідомлень: забезпечення можливості збереження та вибору шаблонів екстрених повідомлень для прискорення підготовки розсилки;
- Надсилання повідомлень із геолокацією: інтеграція геокодування чи координат для таргетування повідомлень певним територіям (за потреби);

- Централізоване журналювання: ведення єдиного журналу з історією надси́лань, статусами доставки та отримання повідомлень;
- Забезпечення мультиканальності розсилки: реалізація відправки повідомлень не лише через месенджер, а також через SMS та email-канали.

Для розв'язання поставлених завдань застосовувались такі методи дослідження і розробки: аналіз літературних джерел і існуючих систем оповіщення; об'єктно-орієнтоване проектування програмного забезпечення; використання методів програмування на Java; інтерфейсна інтеграція з Messenger Bot API і службами надси́лання SMS/Email; тестування

(функціональне та навантажувальне) та ітеративне удосконалення системи. Розробка проводилась за ітераційною методологією з написанням та рефакторингом коду, а також використанням діаграм для моделювання архітектури.

Теоретична значущість роботи полягає у поглибленні розуміння принципів побудови мультиканальних систем екстреного інформування, а також у розширенні знань про використання платформ месенджерів (зокрема Messenger Bot API) у складі таких систем[14]. У роботі також узагальнено світовий досвід щодо важливості швидкого оповіщення населення і наведених практик, що може бути використано як основа для подальших досліджень та оптимізацій. **Практична значущість** дослідження полягає в розробці функціонального прототипу ENS Bot, який уже може бути застосований організаціями й групами людей для оперативного інформування в надзвичайних ситуаціях і для важливих бізнес-повідомлень.

Отже, наведене обґрунтування актуальності та визначення об'єкта, предмета, мети, методів і значущості роботи свідчать про комплексний підхід до розробки інформаційної системи ENS Bot для масового екстреного інформування користувачів. Результати роботи можна використовувати для практичної реалізації систем оповіщення в різних надзвичайних ситуаціях.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Характеристика предметної галузі та об'єкта дослідження

У нинішньому інформаційному просторі питання своєчасного інформування користувачів у надзвичайних ситуаціях набуває особливої гостроти: від прогнозування природних катастроф до реагування на техногенні аварії та кіберзагрози. Саме в цій предметній галузі функціонують системи екстреного оповіщення, які мають забезпечити миттєву доставку критичних повідомлень із максимально широким охопленням через різні канали зв'язку. Поширення ідей мультиканального підходу сприяло появі централізованих платформ, що поєднують можливості месенджер-ботів, SMS-розсилок та email, аби гарантувати доставку навіть за умов перевантаження чи відключення окремих мереж.

Інформаційна система ENS Bot, розроблена на основі Messenger Bot API як універсальний інструмент для автоматизованого надсилання екстрених повідомлень. ENS Bot поєднує єдину точку запуску розсилки з попередньо підготовленими шаблонами повідомлень, що містять динамічні поля і доступний кожному користувачу. У системі виокремлено низку ключових ролей: оператор, що ініціює розсилку; адміністратор бази даних, відповідальний за актуалізацію інформації про користувачів і групи; розробник інтерфейсу, який забезпечує інтуїтивно зрозумілу взаємодію через інлайн-кнопки; технічна підтримка, що опікується стабільністю роботи сервісу та інтеграціями із зовнішніми каналами зв'язку.

У процесі експлуатації ENS Bot користувачі створюють захищені паролем групи, формують шаблони повідомлень і в один клік запускають розсилку одночасно в месенджер, SMS та email. Завдяки централізованому лог-журналу кожен учасник отримує актуальну інформацію про статус доставки своїх повідомлень, що значно спрощує контроль і аналіз комунікацій. Використання стеку Java, Spring Boot, PostgreSQL і Docker Compose забезпечує надійність, масштабованість та швидке розгортання рішення, а монолітна архітектура з

перспективою переходу на мікросервіси створює основу для подальшого розвитку функціоналу та інтеграцій із зовнішніми системами кризового менеджменту.

Аналіз літературних джерел та практичного досвіду використання ІС і технологій в предметній галузі

У сучасних публікаціях і практичних оглядах описано низку глобальних систем масового оповіщення, які реалізують багатоканальне сповіщення населення та організацій. Серед найбільш розповсюджених рішень - платформи Everbridge, та RedFlag. Кожна з них має

власний набір функцій і орієнтована на різні сегменти користувачів. одна з провідних світових платформ критичного управління подіями (Critical Event всьому світу[15]. Everbridge забезпечує автоматизоване сповіщення про надзвичайні події та ризики: система агрегує дані про загрози, визначає місцезнаходження працівників у зоні ризику й запускає попередньо налаштовані сценарії повідомлень. Платформа автоматизує виконання попередньо визначених комунікаційних процесів за допомогою безпечної доставки на понад 100 різних комунікаційних пристроїв, а також відстежує хід виконання планів реагування і гарантує доставку сповіщень цільовим одержувачам у найкоротший час[15]. Наприклад, Everbridge може одночасно надсилати SMS, email та здійснювати телефонні дзвінки, що дозволяє достукатися до адресата, навіть якщо один із каналів недоступний[16][15]. Система також має можливості геотаргетингу та аналітики отриманих підтверджень. Натомість відзначають, що впровадження використовуючи детальне планування та адаптацію під великі організації.

Рис. 1.2.1 Головна сторінка сайту Everbridge

Рис. 1.2.2 Приклад застосунку Everbridge

(бренд OnSolve) - масова система оповіщення, розрахована на широкий спектр організацій (школи, церкви, бізнес-клієнти, муніципалітети тощо). Сервіс дозволяє оперативно відправляти термінові повідомлення через телефонні дзвінки, SMS,

Система підтримує точкове адресування (геотаргетинг) та масштабне покриття: згідно з інформацією постачальника, One Call Now може за хвилини доставити оповіщення мільйонам користувачів[17]. Платформу використовують тисячі організацій різного масштабу[17]. Хоча інтерфейс системи зазвичай оцінюють як інтуїтивно зрозумілий, деякі користувачі вказують на певні складності початкового налаштування та необхідність опанування всіх функцій для ефективної роботи.

Рис. 1.2.3 Головна сторінка сайту One Call Now

Рис. 1.2.4 Приклад застосунку One Call Now

Рис. 1.2.5 Приклад застосунку One Call Now

хмарна система екстреного повідомлення з інтегрованими модулями розвідки загроз та управління поїздками персоналу. За офіційними даними, рішення Система орієнтована на корпоративний сектор будь-якого розміру і підтримує масові двосторонні сповіщення по широкому спектру каналів: SMS, електронна пошта, мобільні пуш-сповіщення, голосові дзвінки, а також робочі месенджери (зокрема WhatsApp, Microsoft Teams тощо)[18].

надає можливості відстеження прочитань, підтверджень доставки і опитування отримувачів, що підвищує контроль над комунікацією. Користувачі відзначають зручний веб-інтерфейс та швидке впровадження системи: готові інтеграції з існуючими сервісами й цілодобова підтримка компанії роблять запуск AlertMedia простим навіть для середніх підприємств.

Рис. 1.2.6 Головна сторінка сайту AlertMedia

Рис. 1.2.7 Приклад застосунку AlertMedia

Рис. 1.2.8 Приклад застосунку AlertMedia

платформа масових текстових повідомлень і внутрішньої комунікації з особливою інтеграцією у середовище Microsoft 365. Система позиціонується як «зручне» рішення для організацій різного масштабу, яке можуть використовувати від малих команд до великих корпорацій[19]. RedFlag підтримує багатоканальне оповіщення (додатках тощо)[19] та має функції двостороннього зв'язку з отримувачами (збір відгуків, опитування, підтвердження отримання). Особливістю RedFlag є тісна інтеграція з продуктами Microsoft 365: зокрема, можна формувати й надсилати повідомлення безпосередньо з Outlook і імпортувати списки адресатів із Excel, а також розсилати сповіщення через Teams[19]. Завдяки простому інтерфейсу та хмарному розгортанню RedFlag зазвичай вважають відносно легким у впровадженні - в багатьох випадках достатньо мінімального технічного налаштування для початку роботи.

Рис. 1.2.9 Головна сторінка сайту RedFlag

Рис. 1.2.10 Приклад застосунку RedFlag

Рис. 1.2.11 Приклад застосунку RedFlag

Отже, аналіз існуючих рішень показує, що всі згадані системи виконують аналогічну основну функцію - масове оповіщення користувачів різними каналами - але розрізняються за спеціалізацією та фокусом. Everbridge виділяється найширшим набором функцій для критичного менеджменту та потужною аналітикою, проте вимагає складнішої інтеграції і переважно застосовується великими організаціями. One Call Now і AlertMedia надають комплексні інструменти для організацій середнього та великого рівня з розвинутими можливостями адресності та аналітики. RedFlag концентрується на внутрішніх повідомленнях і інтеграції з корпоративними сервісами (Microsoft 365), пропонуючи простоту налаштування для компаній будь-якого розміру. При цьому всі системи підтримують стандартні канали зв'язку (SMS, електронна пошта, голосові дзвінки тощо)[16] [17] [18] [19], а відмінності зводяться переважно до деталізації функцій, цільових сегментів користувачів та ресурсів, необхідних для впровадження.

Система	Повнота функціональних можливостей	Економічність моделі	Ситуації прийняття рішень	Організація баз даних	Технологічно-організаційні аспекти	Інтерфейс	Результати використання	Вигоди
---------	------------------------------------	----------------------	---------------------------	-----------------------	------------------------------------	-----------	-------------------------	--------

	Масова розсилка повідомлень через SMS, нотифікації; підтримка геолокації; шаблони повідомлень.	Методи класифікації даних	Розсилка повідомлень для екстрених ситуацій, наприклад,	Реляційна база даних. Хмарне зберігання даних.	Хмарна інфраструктура, інтеграція через API.	Інтуїтивний веб-інтерфейс, мобільний додаток.	Інформувати тисячі користувачів в реальному часі; детальні звіти про доставку.	Висока надійність, масштабованість, простота налаштування.
--	--	---------------------------	---	--	--	---	--	--

			евакуації.					
	Голосові повідомлення	Простий пошук по шаблонах.	Масове оповіщення для шкіл, бізнесу.	Реляційна база даних. Хмарне зберігання даних.	Хмарна платформа з налаштуванням через веб-інтерфейс.	Простий інтерфейс.	Швидке поширення повідомлень серед визначених груп.	Простота використання, низька вартість.
	Мультиканальні оповіщення через SMS, email, мобільні додатки; підтримка двосторонньої комунікації.	Метод найкращого шляху.	Екстрені комунікації, моніторинг ризиків.	Хмарне зберігання даних.	Інтеграція з корпоративними системами через	Сучасний та зручний інтерфейс.	Інтерактивні повідомлення з можливістю зворотного зв'язку; аналітика ризиків.	Забезпечення інтерактивності, адаптація до потреб великих організацій.

	Швидке сповіщення через email, pushнотифікації; підтримка інтеграції з соціальними мережами.	Методи пошуку та фільтрації	Швидке поширення інформації під час надзвичайних ситуацій.	Локальна чи хмарна база даних.	Легке масштабування, інтеграція з корпоративними рішеннями.	Проста навігація.	Створення груп для оповіщення; повний контроль за змістом повідомлень.	Гнучкість у налаштуванні, швидкість розгортання.

РОЗДІЛ 2. РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Аналіз і специфікація вимог до інформаційної системи

У результаті аналізу предметної області було сформульовано **бізнесвимоги**, які відображають загальні цілі проекту ENS Bot - системи автоматизованого надсилання екстрених сповіщень. Головна бізнес-ціль полягає в створенні ефективного механізму оперативного інформування користувачів про надзвичайні ситуації через чат-бот та інші канали зв'язку. Зокрема, визначено, що система повинна підтримувати багатокористувацький режим із авторизацією і збереженням профілів, надавати можливість створення і приєднання до груп користувачів, управління шаблонами повідомлень (створення, редагування, перегляд) і запуск масової розсилки. Користувачі мають формувати повідомлення (з використанням шаблонів, які можуть містити змінні дані) та визначати цільову групу отримувачів, при цьому інтерфейс чат-бота повинен бути зручним і зрозумілим. Бізнес-цілі проекту включають також можливість таргетованої розсилки повідомлень на конкретні групи (з метою зменшення інформаційного навантаження для нецільових аудиторій) і підтримку геолокаційних даних для уточнення контексту оповіщення в територіально обмежених ситуаціях. Окремою вимогою є масштабованість і надійність: архітектура системи розроблена з урахуванням можливості подальшої інтеграції нових каналів доставки (SMS, електронна пошта, інші месенджери) та інструментів моніторингу (Prometheus, Grafana) для забезпечення безперервного контролю за роботою сервісів. Нефункціональні вимоги, що випливають з аналізу, охоплюють обмеження на час відповіді системи (наприклад, доставка повідомлень має відбуватися протягом секунди), безпеку та конфіденційність передачі даних, ведення аудиту критичних дій (логування операцій створення груп, редагування

шаблонів, надсилання повідомлень) та інші характеристики якості системи (масштабованість, відмовостійкість тощо).

Зібрані вимоги формалізовано у графічних моделях. Для візуалізації структури вимог застосовано діаграму вимог SysML (requirement diagram), що відображає ієрархію бізнес-вимог та їх зв'язок з функціональними і нефункціональними вимогами[10]. Така діаграма показує, яким бізнеспотребам відповідають окремі функції системи та як вони простежуються до тестових сценаріїв, що гарантують їх задоволення.

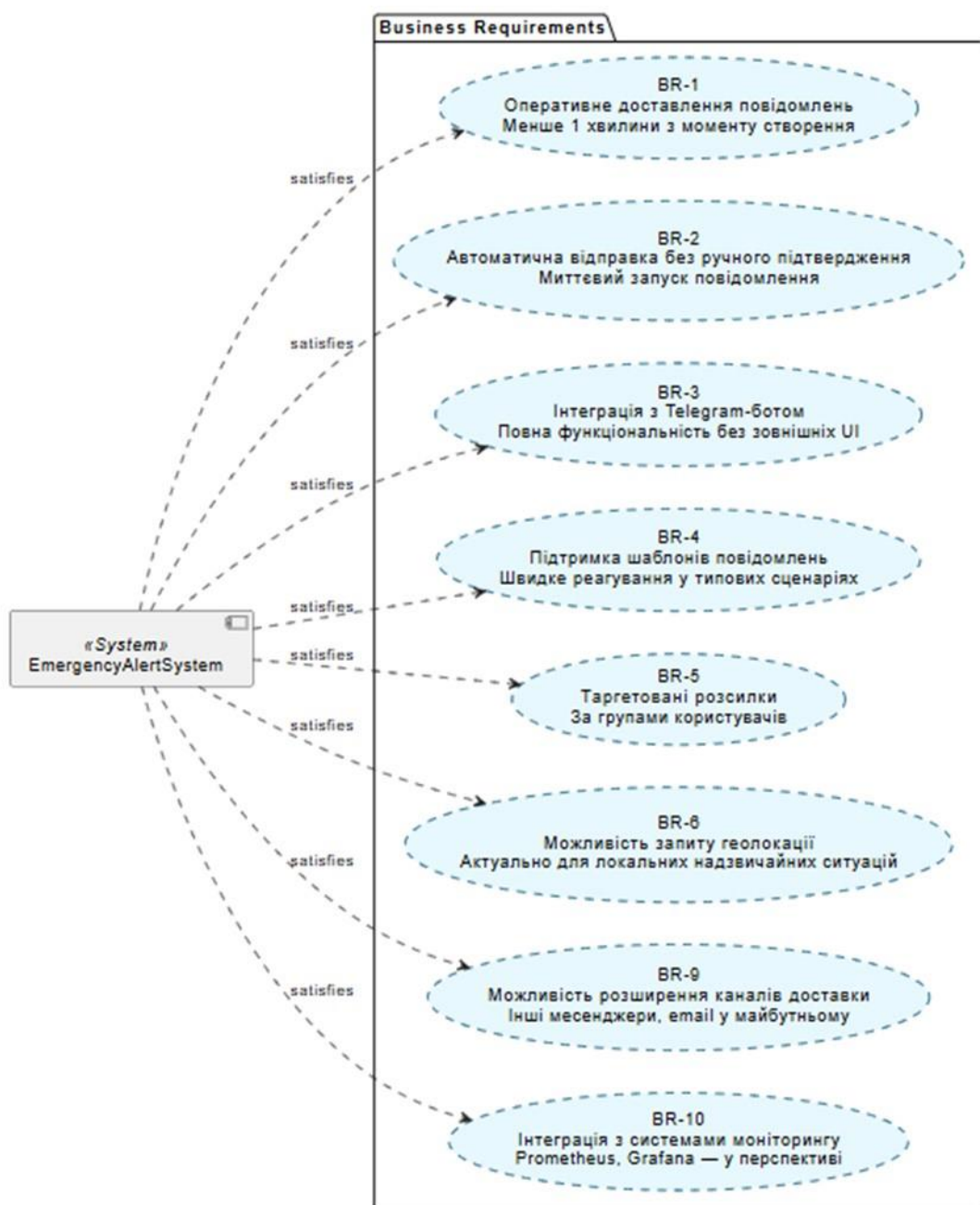


Рис 2.1.1. Діаграма бізнес-вимог

Функціональні вимоги до системи аварійного оповіщення оформлено згідно зі специфікацією SysML (ISO/IEC 19514:2017) та об'єднано в ієрархічну модель із деталізацією підсистем і їхніх підфункцій. Вони охоплюють такі основні блоки:

Управління користувачами (User Management)

Цей модуль відповідає за введення, збереження та оновлення особистих даних. Зокрема, система повинна дозволяти реєстрацію нових учасників за допомогою Messenger ID та електронної пошти (FR-1), зміну телефонного номеру (FR-2), перевірку наявності користувача в базі (FR-3) і отримання детальної інформації за його Messenger ID (FR-4).

Управління групами (Group and Membership Management)

Цей компонент забезпечує повний цикл роботи з групами: створення нових (FR-5), приєднання до існуючих за ім'ям і паролем (FR-6), перегляд учасників групи (FR-7) та керування життєвим циклом групи (FR-8). Крім того, користувачі повинні мати змогу переглянути список усіх груп, до яких вони належать (FR-9), і отримати детальну інформацію про конкретну групу (FR-10).

Управління шаблонами повідомлень (Message Template Management)

Цей підсистема відповідає за створення (FR-11), редагування (FR-12), видалення (FR-13) і перегляд (FR-14) текстових шаблонів. Також передбачено надсилання обраного шаблону всім учасникам групи одним запитом (FR-15).

Складання та форматування повідомлень (Notification Composition &

Система автоматично адаптує шаблони до специфіки каналів доставки: повідомлення мають коректно відображатися в електронній пошті (FR-16), в месенджері та SMS (FR-17) з урахуванням часу, місця та контексту.

Механізм доставки сповіщень (Alert Delivery System)

Цей модуль реалізує власне відправку повідомлень. Передбачено надсилання через електронну пошту (FR-18), а також закладено можливість підключення мобільних push-сервісів у майбутньому (FR-19).

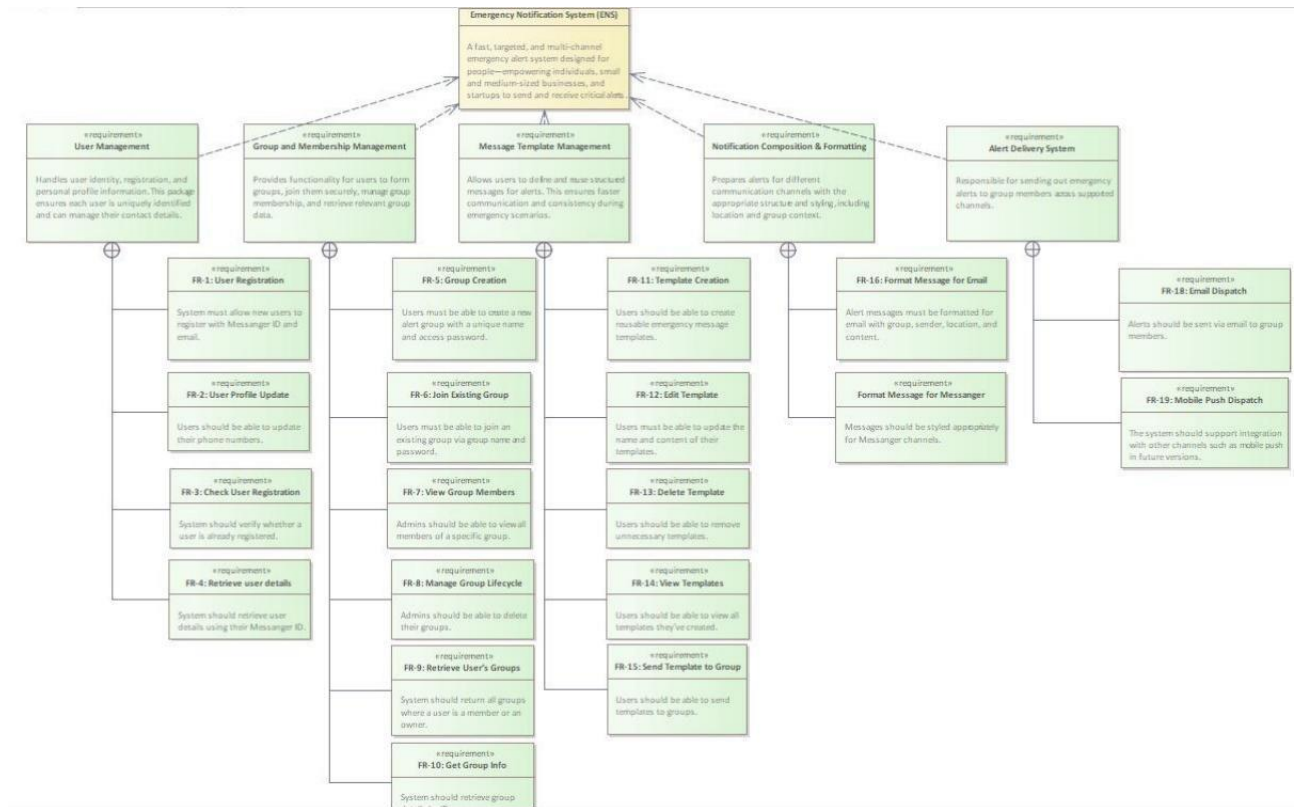


Рис. 2.1.2 Функціональні вимоги

*Req Specification Manager				
Find Package				
Item	Stereotype	Status	Difficulty	Priority
Emergency Notification System (ENS) A fast, targeted, and multi-channel emergency alert system designed for people—empowering individuals, small and medium-sized businesses, and startups to send and receive critical alerts.	testCase	Approved		

Рис. 2.1.3 FR ENS Bot Specification Manager

☒	Alert Delivery System	Functional	Approved	Medium	Medium
	Responsible for sending out emergency alerts to group members across supported channels.				
☒	FR-18: Email Dispatch	functional requir...	Implemented	Medium	High
	Alerts should be sent via email to group members.				
☒	FR-19: Mobile Push Dispatch	functional requir...	Proposed	High	Medium
	The system should support integration with other channels such as mobile push in future versions.				

Рис. 2.1.4 FR Alert Delivery System Specification Manager

☒ Group and Membership Management Provides functionality for users to form groups, join them securely, manage group membership, and retrieve relevant group data.	Functional	Implemented	Medium	Medium
☒ FR-10: Get Group Info System should retrieve group details by ID or name.	FunctionalRequir...	Implemented	Medium	Medium
☒ FR-5: Group Creation Users must be able to create a new alert group with a unique name and access password.	FunctionalRequir...	Implemented	Medium	High
☒ FR-6: Join Existing Group Users must be able to join an existing group via group name and password.	functional requir...	Implemented	Medium	High
☒ FR-7: View Group Members Admins should be able to view all members of a specific group.	functional requir...	Implemented	Medium	Medium
☒ FR-8: Manage Group Lifecycle Admins should be able to delete their groups.	functional requir...	Implemented	Medium	Medium
☒ FR-9: Retrieve User's Groups System should return all groups where a user is a member or an owner.	functional requir...	Implemented	Low	Medium

Puc. 2.1.5 FR Group and Membership Management Specification Manager

☒ Message Template Management Allows users to define and reuse structured messages for alerts. This ensures faster communication and consistency during emergency scenarios.	Functional	Implemented	High	High
☒ FR-11: Template Creation Users should be able to create reusable emergency message templates.	functional requir...	Implemented	Medium	Medium
☒ FR-12: Edit Template Users must be able to update the name and content of their templates.	functional requir...	Implemented	Medium	Medium
☒ FR-13: Delete Template Users should be able to remove unnecessary templates.	functional requir...	Implemented	Low	Low
☒ FR-14: View Templates Users should be able to view all templates they've created.	functional requir...	Implemented	Low	Medium
☒ FR-15: Send Template to Group Users should be able to send templates to groups.	functional requir...	Implemented	High	High

Puc. 2.1.6 FR Message Template Management Specification Manager

☒ Notification Composition & Formatting Prepares alerts for different communication channels with the appropriate structure and styling, including location and group context.	Functional	Implemented	Medium	Medium
☒ FR-16: Format Message for Email Alert messages must be formatted for email with group, sender, location, and content.	functional requir...	Implemented	Low	Medium
☒ FR-17: Format Message for Messenger Messages should be styled appropriately for Messenger channels.	functional requir...	Implemented	Low	Medium

Рис. 2.1.7 FR Notification Composition Specification Manager

☒ User Management	Functional	Implemented	Medium	High
Handles user identity, registration, and personal profile information. This package ensures each user is uniquely identified and can manage their contact details.				
☒ FR-1: User Registration	functional requir...	Implemented	Medium	Medium
System must allow new users to register with Messenger ID and email.				
☒ FR-2: User Profile Update	functional requir...	Implemented	Medium	Medium
Users should be able to update their phone numbers.				
☒ FR-3: Check User Registration	functional requir...	Implemented	Medium	Medium
System should verify whether a user is already registered.				
☒ FR-4: Retrieve user details	functional requir...	Implemented	Medium	Medium
System should retrieve user details using their Messenger ID.				

Рис. 2.1.8 FR User Management Specification Manager

Нефункціональні вимоги описують обмеження та характеристики якості, що впливають на ефективність, масштабованість, підтримуваність і надійність системи. Вимоги представлені у вигляді діаграми вимог (requirement diagram) згідно зі специфікацією SysML (ISO/IEC 19514:2017), що демонструє їхню ієрархію та зв'язки із загальним тест-кейсом системи.

Наведені нижче нефункціональні вимоги були визначені на основі аналізу очікувань користувачів, архітектурних стандартів, вимог безпеки та цілей стабільної експлуатації системи в умовах реального часу:

- **NFR-01: Продуктивність та масштабованість**

Система повинна забезпечувати доставку повідомлень протягом 1 хвилини з моменту ініціації, навіть при високому навантаженні.

- **NFR-02: Безпека**

Усі персональні дані повинні передаватися в зашифрованому вигляді, що гарантує конфіденційність та відповідність політикам безпеки.

- **NFR-03: Надійність і відмовостійкість**

Доступність системи має становити не менше 99.9% часу, включаючи автоматичне відновлення після збоїв.

- **NFR-04: Підтримуваність**

Архітектура програмного коду має бути модульною, документованою і покритою тестами. Модулі повинні мати слабкий ступінь зв'язності.

- **NFR-05: Розширюваність**

Архітектура повинна дозволяти інтеграцію нових каналів оповіщення (наприклад, push-нотифікацій, сирен) без впливу на існуючий функціонал.

- **NFR-06: Аудит та логування**

Усі критичні дії - як-от створення груп, відправлення повідомлень, оновлення профілю - мають бути зафіксовані у журналах з можливістю простежування.

- **NFR-07: Доступність**

Інтерфейси оповіщення мають бути зрозумілими та доступними для користувачів на всіх підтримуваних каналах зв'язку.

- **NFR-08: Моніторинг та оповіщення**

Система повинна мати інтеграцію з інструментами моніторингу, такими як eus та Grafana, для забезпечення стабільної роботи.

Ці вимоги пов'язані з основним тест-кейсом системи "Emergency Notification System (ENS)", що передбачає реалізацію швидкого, багатоканального сервісу масового оповіщення для малих і середніх підприємств, організацій та індивідуальних користувачів.

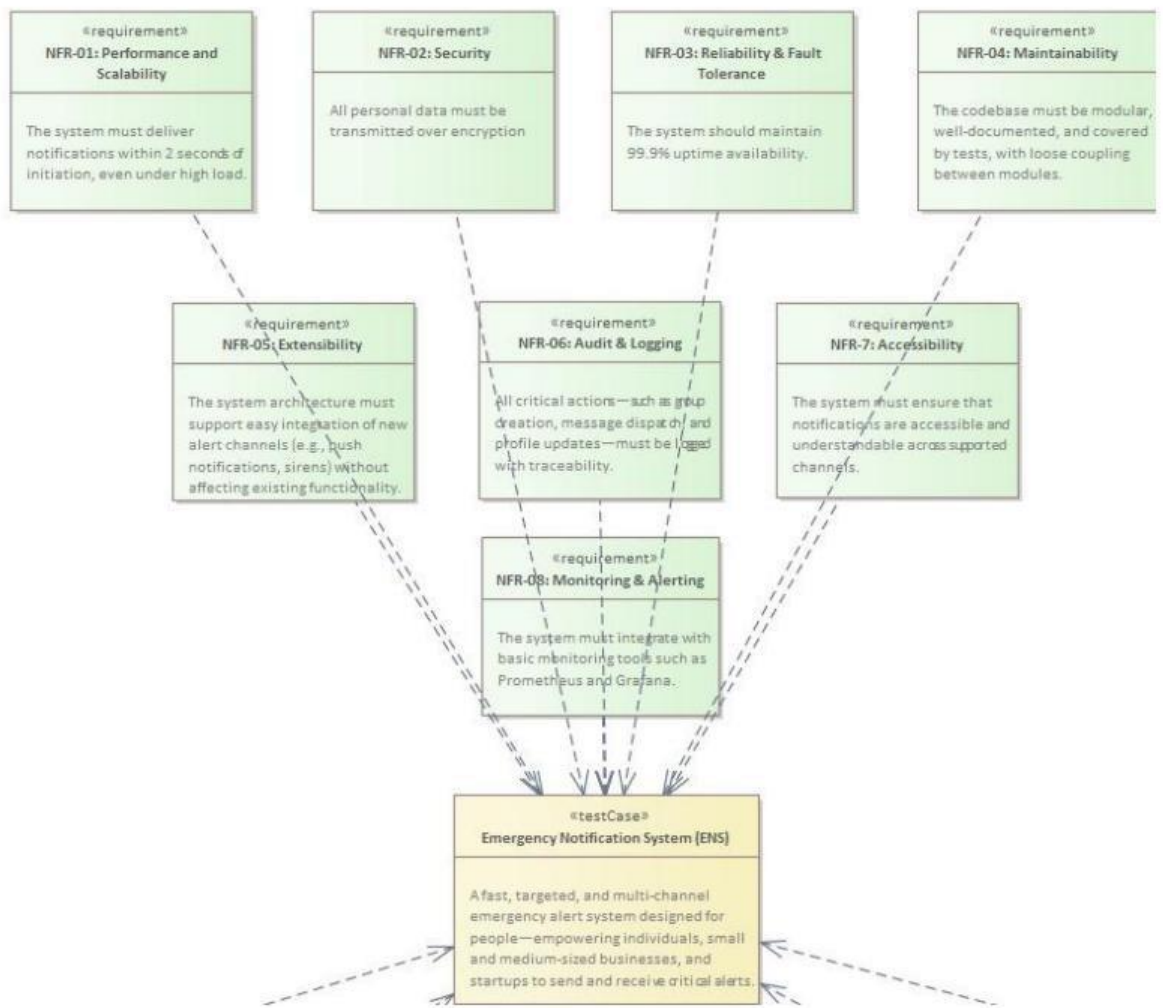


Рис. 2.1.9 Нефункціональні вимоги

*Req Specification Manager				
Model > Req				
Item	Stereotype	Status	Difficulty	Priority
Emergency Notification System (ENS)	testCase	Approved		
A fast, targeted, and multi-channel emergency alert system designed for people—empowering individuals, small and medium-sized businesses, and startups to send and receive critical alerts.				

Рис. 2.1.10 NFR ENS Bot Specification Manager

Item	Stereotype	Status	Difficulty	Priority
☒ NFR-01: Performance and Scalability The system must deliver notifications within 2 seconds of initiation, even under high load.	NonfunctionalRe...	Implemented	Medium	Medium
☒ NFR-02: Security All personal data must be transmitted over encryption	NonfunctionalRe...	Mandatory	Medium	High
☒ NFR-03: Reliability & Fault Tolerance The system should maintain 99.9% uptime availability.	NonfunctionalRe...	Mandatory	High	High
☒ NFR-04: Maintainability The codebase must be modular, well-documented, and covered by tests, with loose coupling between modules.	NonfunctionalRe...	Mandatory	High	Medium
☒ NFR-05: Extensibility The system architecture must support easy integration of new alert channels (e.g., push notifications, sirens) without affecting existing functionality.	NonfunctionalRe...	Implemented	Medium	High
☒ NFR-06: Audit & Logging All critical actions—such as group creation, message dispatch, and profile updates—must be logged with traceability.	NonfunctionalRe...	Mandatory	Medium	High
☒ NFR-08: Monitoring & Alerting The system must integrate with basic monitoring tools such as Prometheus and Grafana.	NonfunctionalRe...	Proposed	Medium	High
☒ NFR-7: Accessibility The system must ensure that notifications are accessible and understandable across supported channels.	NonfunctionalRe...	Implemented	Medium	High

Рис. 2.1.11 NFR Specification Manager

Постановка задачі та алгоритм розв’язання задачі

Завдання проекту можна сформулювати так: необхідно створити інформаційну систему, яка дозволить оперативно формувати та масово надсилати екстрені сповіщення користувачам. Метою є забезпечення умов, за яких оператор (або авторизований користувач) системи може скласти повідомлення (наприклад, вибравши один із шаблонів або ввівши текст вручну), вказати цільову аудиторію (групу чи набір користувачів) та ініціювати процес розсилки. Система у відповідь повинна зібрати необхідні дані (текст повідомлення, список отримувачів, додаткові параметри на кшталт дати/часу чи даних геолокації) і забезпечити доставку сповіщень обраним каналом зв’язку (через чат-бот, SMS, email тощо).

Вхідна інформація системи включає всі дані, що надходять від користувачів через інтерфейс Messenger Bot API або інші канали: текстові команди, параметри шаблонів, запити на створення чи приєднання до груп, передані геолокаційні

координати тощо. Ці дані обробляються відповідними підсистемами - реєстрації, управління групами, шаблонів і розсилки - та формують основу для подальшого генерування сповіщень і виконання бізнес-логіки.

Табл. 2.2.1

ПЕРЕЛІК І ОПИС ВХІДНИХ ПОВІДОМЛЕНЬ

№ з/п	Назва вхідного повідомлення	Ідентифікатор	Форма подання	Термін і частота надходження	Джерело
	Дані про ініціатора (користувача)		JSON/Текстове поле	У момент ініціації події	Введення через інтерфейс системи
	Групи		JSON/Текстове поле	У момент створення події	Введення через інтерфейс системи
	Список отримувачів		JSON/Текстове поле	У момент ініціації події	База даних системи
	Шаблони для формування повідомлень		Текстове поле, дата, час, назва	Після авторизації в системі	Локальне сховище / База даних
	Геолокаційні дані (за наявності)		Координати (широта, довгота)	У момент події (як опція)	Мобільний пристрій користувача
	Логи про відправку повідомлень		Хто надіслав повідомлення, До якої групи було надіслано, Статус доставки, Вміст повідомлення.	У момент закінчення ініціації події	База даних системи

Вихідна інформація, зазначена на табл. 2.2.2, призначена для подання результатів роботи системи та задоволення поточних потреб користувачів. Вона

охоплює підтвердження виконаних операцій (успішна реєстрація, створення групи, відправка сповіщення), журнали розсилок із деталями доставки, тощо.

Табл. 2.2.2

ПЕРЕЛІК І ОПИС ВИХІДНИХ ПОВІДОМЛЕНЬ

№ з/п	Назва вихідного повідомлення	Ідентифікатор	Форма подання і вимоги до неї	Періодичність видавання	Термін видавання і допустимий час затримки	Користувачі інформації
	Дані про ініціатора		JSON/текстові дані. Дані мають бути валідовані.	За запитом	Миттєво, затримка не більше 5 секунд	Система, API сервісів
	Текст повідомлення		Текстове поле. Максимальний обсяг символів обмежується	За запитом	Миттєво, затримка не більше 5 секунди	Учасники групи
			системою, заборонені спеціальні символи.			
	Список отримувачів		JSON із масивами даних (ID групи, ID учасників, дата додавання).	За запитом	Миттєво, затримка не більше 5 секунд	Система, служба доставки

	Шаблон повідомлення		Текстові дані (назва, дата створення, вміст шаблону). Структура має включати змінні {ім'я}, {дата}.	Під час створення події	Миттєво, затримка не більше 20 секунди	Система, автори розсилок
	Логи про відправку повідомлень		JSON або таблиця бази даних із даними про статуси (успішно/невдало).	Постійно	Відразу після завершення операції	Система
	Геолокація		Географічні координати (широта, довгота) у числовому форматі.	За потреби	Миттєво, затримка не більше 20 секунд	Система, API сервісів

Внаслідок внесення вхідної та вихідної інформації поповнюються масиви використовуваної інформації.

Таблиця 2.2.3

ПЕРЕЛІК МАСИВІВ ВИКОРИСТОВУВАНОЇ ІНФОРМАЦІЇ

Масив	Ідентифікатор	Максимальна кількість записів
Користувачі		
Групи		

Шаблони повідомлень		
Логи надсилання повідомлень		

Таблиця 2.2.4

ПЕРЕЛІК МАСИВІВ РЕЗУЛЬТАТНОЇ ІНФОРМАЦІЇ

Масив	Ідентифікатор	Максимальна кількість записів
Дані про ініціатора		
Текст повідомлення		
Шаблони повідомлень		
Логи надсилання повідомлень		

Математичне забезпечення та алгоритм функціонування системи

Формула 1: Середній час доставки повідомлення

Середній час доставки повідомлення є важливим показником ефективності системи. Він дозволяє оцінити, наскільки швидко повідомлення досягають отримувачів. Розрахунок відбувається за формулою середнього арифметичного часу доставки всіх повідомлень.

$$\text{Код формули: } T_{\text{avg}} = \frac{\sum_{i=1}^n \dots}{n}$$

Застосунок: LaTeX

Пояснення змінних:

: середній час доставки повідомлення, виражений у секундах або мілісекундах.

: час доставки iii -го повідомлення, тобто час, який пройшов між моментом відправлення і моментом доставки. Вимірюється у секундах/мілісекундах.

: загальна кількість повідомлень, які були надіслані та доставлені.

Алгоритм розрахунку:

Для кожного повідомлення потрібно зафіксувати:

Час його надсилання (t_{start}). • Час доставки до отримувача ($t_{delivery}$).

изначити час доставки для кожного повідомлення:

ідсумувати всі часи доставки:

оділити суму часів доставки на кількість повідомлень (n), щоб знайти середнє значення.

Приклад:

Якщо для 3 повідомлень час доставки становив:

•

Тоді:

с

• Код: $T_{avg} = \frac{2+3+5}{3} = 3.33$

Формула 2: Відсоток успішної доставки

с

Ця формула дозволяє визначити, наскільки ефективно система доставляє повідомлення до отримувачів. Показник успішної доставки вимірюється у відсотках.

Код формули: $P_{\text{success}} = \frac{N_{\text{success}}}{N_{\text{total}}} \times 100\%$
Застосунок: LaTeX

Пояснення змінних:

: відсоток успішної доставки.

: кількість повідомлень, які були успішно доставлені.

загальна кількість відправлених повідомлень (успішні та неуспішні).

Алгоритм розрахунку:

ідрахувати кількість успішно доставлених повідомлень (N_{success}).

изначити загальну кількість відправлених повідомлень (N_{total}).

оділити кількість успішних повідомлень на загальну кількість:

омножити отримане значення на 100, щоб отримати результат у відсотках.

Приклад:

Припустимо, було відправлено 100 повідомлень, з яких 85 були успішно доставлені. Тоді:

Код: $P_{\text{success}} = \frac{80}{100} \times 100\% = 85\%$

Висновок

Ці формули дають змогу обчислювати ключові метрики, що допомагають аналізувати ефективність функціонування системи:

ередній час доставки показує, наскільки швидко система працює.

ідсоток успішної доставки демонструє, наскільки надійна система.

Для відображення логіки алгоритму та способу формування результатів розв'язання сформовано діаграму, що демонструє інформаційну модель задачі (рис. 2.2.1).

Рис. 2.2.1 - Інформаційна модель задачі

Для наочного подання послідовності обробки подій у системі ENS Bot наведено детальний опис алгоритмів, що відповідають за валідацію, генерування, надсилання та логування повідомлень.

Надходження події в ENS Bot починається з ініціації користувацького запиту, коли учасник групи відправляє команду або натискає інлайн-кнопку. Перш ніж сформувати повідомлення, система виконує верифікацію вхідних даних: перевіряє наявність усіх обов'язкових полів та коректність їх формату. У разі невідповідності вимог користувачеві миттєво повертається повідомлення про помилку валідації, і подальша обробка припиняється. Якщо ж дані відповідають очікуванням, ENS Bot переходить до етапу генерації тексту сповіщення на основі обраного шаблону або введеного вручну вмісту з автоматичною підстановкою змінних, таких як геолокація чи часові мітки.

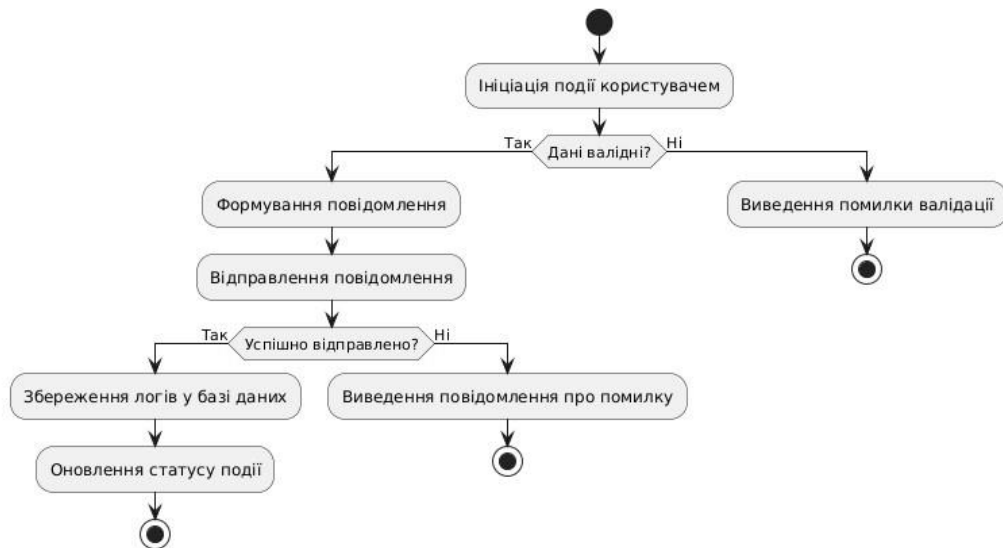


Рис. 2.2.2 Алгоритм відправлення повідомлення

Сформоване повідомлення надсилається одночасно через Messenger Bot API та альтернативні канали (SMS, email). Після відправки система очікує на підтвердження статусу доставки. У разі успішної доставки відбувається запис події в централізований лог-журнал: фіксуються час

відправлення, ідентифікатори групи й одержувачів, а також обраний канал доставки. Після цього оновлюється статус події на «доставлено», і обробка завершується. Якщо ж надсилання завершилось помилкою, механізм обробки помилок визначає причину невдачі (наприклад, недоступність контакту чи перевищення ліміту) й відображає відповідне повідомлення користувачеві, після чого алгоритм припиняється без оновлення успішного статусу в журналі.

Рис. 2.2.3 Алгоритм обробки помилок

Окремо ENS Bot реалізує уніфіковане логування всіх системних подій: кожна ключова дія - надсилання чи редагування шаблону, створення або приєднання до групи - ініціює збір метаданих (час, тип події, параметри запиту) і їхнє збереження в базі даних.

Рис. 2.2.4 Алгоритм логування подій

Моделювання інформаційної системи

Для демонстрації функціонування інформаційної системи, розроблено діаграму прецедентів та діаграму послідовностей.

На діаграмі прецедентів (рис. 2.3.1) зображено три ключові типи акторів: Unauthorized User, Registered User та Group Admin (Owner) / Group Member.

Незареєстровані користувачі мають змогу здійснити авторизацію через месенджер та зареєструватися в системі. Після входу зареєстровані користувачі отримують доступ до базових функцій, таких як додавання електронної адреси, створення шаблонів повідомлень і керування ними (редагування, перегляд, видалення). Центром цієї частини функціоналу виступає модуль Message Template

Management, який інкапсулює логіку підготовки повідомлень і подальшої їх відправки до груп.

Групові функції розподілено відповідно до ролей. Group Admin (Owner) має розширені права на створення та видалення груп, перегляд учасників і адміністрування декількох груп одночасно. У той час як Group Member може приєднуватися до існуючих груп, залишати їх, переглядати свою участь, а також отримувати повідомлення з цих груп. Такий розподіл дозволяє реалізувати багаторівневу модель взаємодії, де кожен учасник системи має чітко визначений набір дозволів.

Діаграма прецедентів забезпечує візуальне представлення логіки використання системи, відображаючи структуровану взаємодію між зовнішніми акторами і програмною платформою. Це дозволяє швидко оцінити набір можливостей системи та зрозуміти, які саме функції є доступними для кожного типу користувачів.

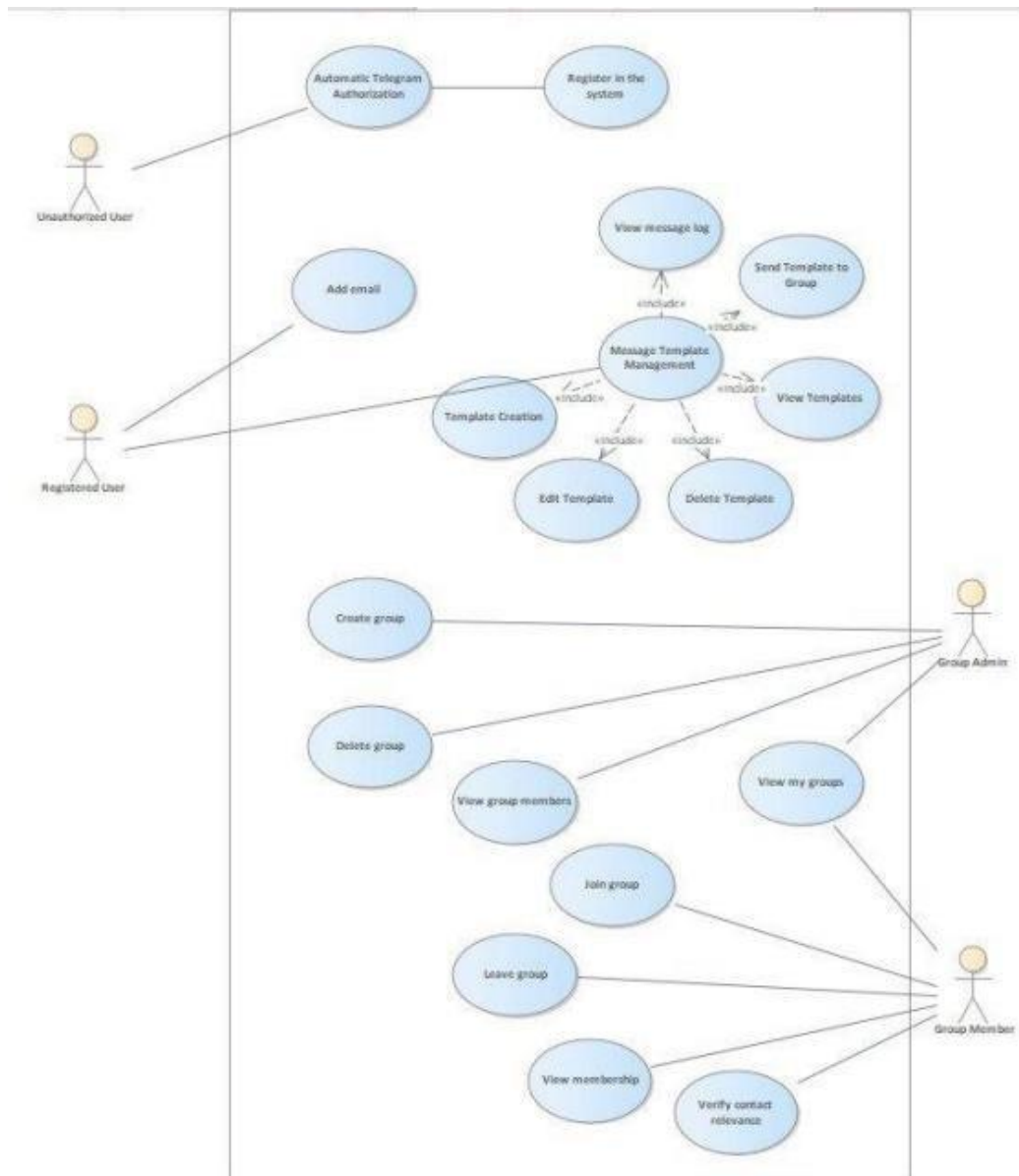


Рис 2.3.1 Діаграма прецедентів

Діаграми послідовності дозволяють зобразити часову послідовність обміну повідомленнями між компонентами системи та акторами, що забезпечує наочне розуміння порядку викликів, передачі даних і взаємозалежностей у процесі виконання кожного варіанту використання. Далі представлені детальні діаграми послідовності, які ілюструють покрокову взаємодію компонентів та акторів для основних варіантів використання.

Рис 2.3.2. Діаграма відправки шаблонів повідомлень

Рис 2.3.3. Діаграма редагування шаблону повідомлення

Рис 2.3.4. Діаграма видалення групи

Рис 2.3.5. Діаграма створення групи

Рис 2.3.6. Діаграма приєднання до групи

Рис 2.3.7. Діаграма отримання груп користувача

Рис 2.3.8. Діаграма виходу з групи

Рис 2.3.9. Діаграма перевірки даних користувача

Структурне моделювання дозволяє отримати цілісну картину архітектури ПЗ і виокремити ключові підсистеми, що реалізують основний функціонал. У рамках цього проєкту така модель демонструє зв'язки між компонентами, які відповідають за автентифікацію користувачів, керування шаблонами повідомлень, роботу з групами та доставку сповіщень через сторонні сервіси, зокрема через Messenger Bot

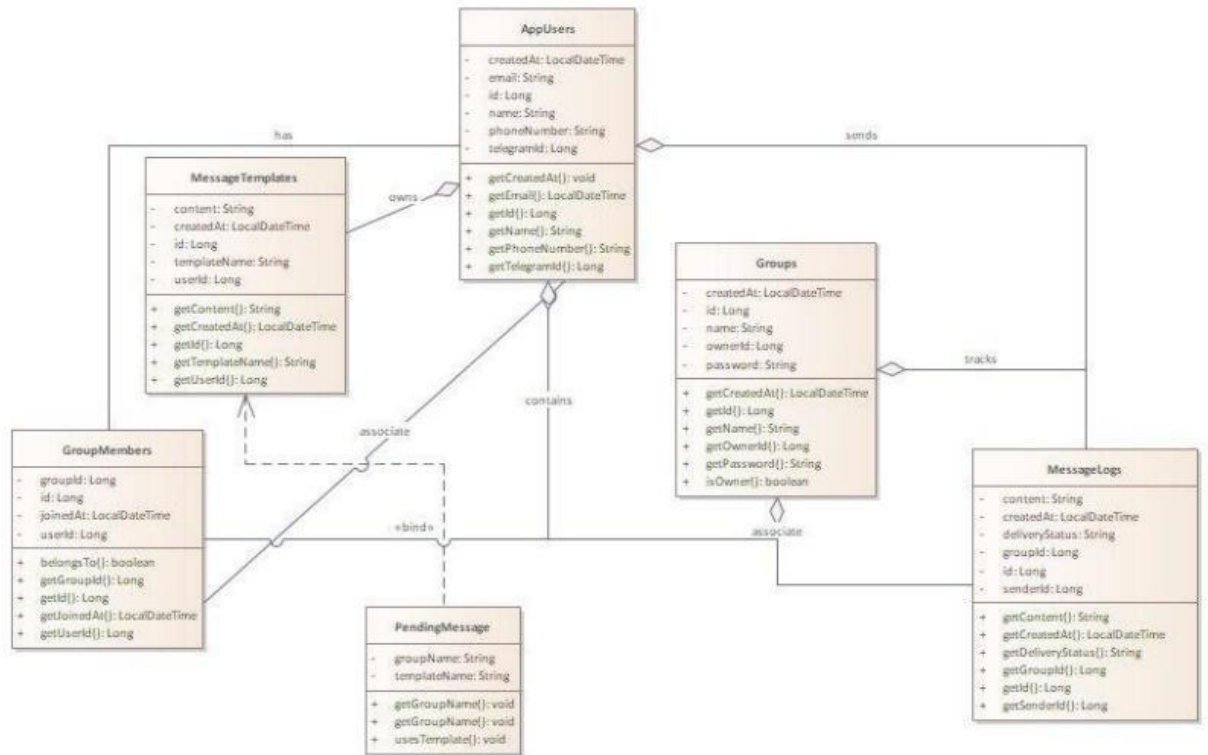


Рис. 2.3.10 Діаграма класів. Entity classes

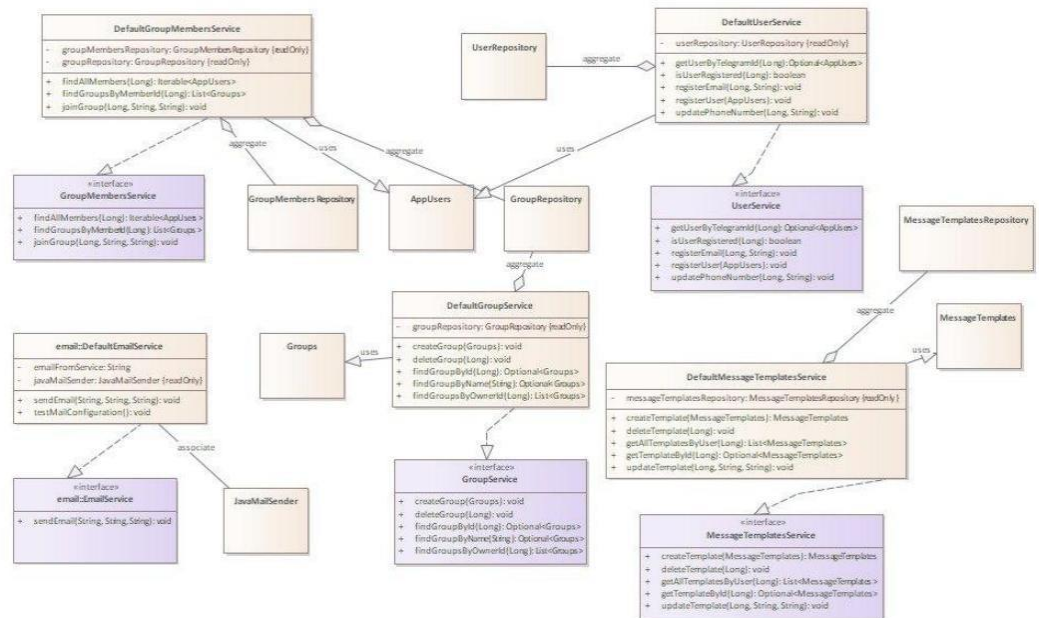


Рис. 2.3.11 Діаграма класів. *Service classes*

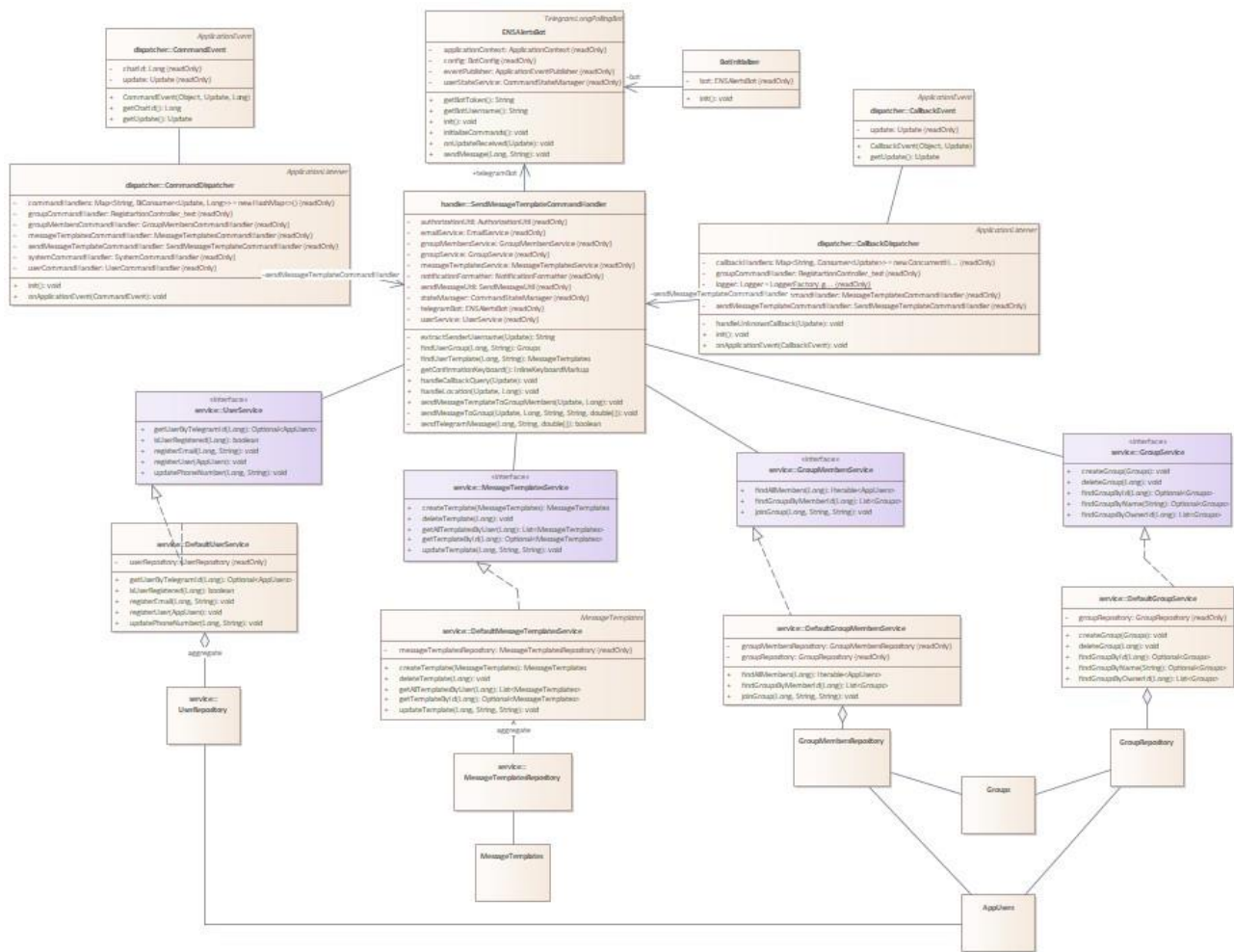


Рис. 2.3.12 Діаграма класів класів пов'язаних з відправкою шаблону повідомлень у групи

Архітектурне моделювання за допомогою діаграми визначення блоків (BDD) дає змогу виділити ключові підсистеми та описати їхні взаємодії через чітко сформовані інтерфейси. Зокрема, на такій діаграмі демонструються блоки, що відповідають за збереження шаблонів, обробку електронних адрес, управління користувачами, а також за взаємодію з месенджером через авторизацію та надсилення повідомлень. Кожна підсистема подається як самостійна одиниця з визначеною зоною відповідальності, що полегшує її тестування та інтеграцію до гальної архітектури. На діаграмі BDD виділено основні компоненти ENS Bot:

М

е

с

с

Рис. 2.3.13. Діаграма визначення блоків (BDD)

а Діаграма внутрішніх блоків (IBD) відображає детальну структуру обміну даними між компонентами ENS Bot через спеціалізовані порти. Зокрема, UI-модуль надсилає запити на отримання або оновлення інформації про шаблони, групи та користувачів до ApplicationLogic, яка, у свою чергу, передає ці дані у підсистему для збереження в таблицях (GroupsTable, AppUsersTable, MessageTemplatesTable тощо).

р Крім того, на IBD показано маршрутизацію логів через AuditTracking, обробку сповіщень за допомогою MessageDispatching та відправку електронних повідомлень через EmailNotificationUI. Потоки даних чітко розмежовані за напрямками - від інтерфейсів до логіки й далі до бази або зовнішніх сервісів - що створює прозору та масштабовану модель.

і

п

Рис. 2.3.14. Діаграма внутрішніх блоків (IBD)

g Такий підхід не лише спрощує розробку та тестування, але й забезпечує легку інтеграцію зовнішніх сервісів, наприклад, з Data Management та іншими різними сервісами. розширені методи автентифікації. Завдяки ізольованому представленню кожного блоку його можна автономно протестувати чи оновити без ризику порушення стабільності всієї системи. Крім того, модель полегшує підтримку проєкту, оскільки кожен елемент прямо відповідає конкретному модулю коду або конфігурації середовища розгортання.

РОЗДІЛ 3.

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

3.1 Інформаційне забезпечення

У межах розробленої інформаційної системи аварійного оповіщення ENS Bot інформаційне забезпечення складається з наступних елементів:

1. Бібліотеки

- Messenger Bot API: реалізує взаємодію з Messenger Bot API;
- Spring Boot Starter Web (org.springframework.boot:spring-boot-starter-web):
забезпечує організацію REST-контролерів і веб-сервер;
- Spring Boot Starter Data JDBC (org.springframework.boot:spring-boot-starterdata-jdbc): надає засоби доступу до реляційної БД через JDBC;
- Flyway Core та Flyway PostgreSQL (org.flywaydb:flyway-core, flyway-databasepostgresql): відповідають за керування версіями схем БД;
- Spring Boot Starter Security (org.springframework.boot:spring-boot-startersecurity): забезпечує базові механізми аутентифікації та авторизації;
- PostgreSQL JDBC Driver (org.postgresql:postgresql): реалізує клієнтську частину для підключення до СУБД;
- Lombok (org.projectlombok:lombok): спрощує генерацію шаблонного коду (геттери, сеттери, конструктори);
- Spring Boot Starter Mail (org.springframework.boot:spring-boot-starter-mail):
дозволяє відправляти email-повідомлення;
- Spring Boot Starter Test (org.springframework.boot:spring-boot-starter-test): забезпечує тестову інфраструктуру.

2. Основні сутності предметної області

ористувачі: особи, які реєструються за Messenger ID та email;

рупи одержувачів: логічні об'єднання користувачів за назвою та паролем;
лени групи: зв'язок “користувач - група” із правом отримувати сповіщення;
аблони повідомлень: попередньо підготовлені тексти із змінними полями;
урнали логів: записи про кожну операцію над повідомленнями (час, група, канал).

3. Функції та обробники команд

- Реєстрація та налаштування профілю (/start, /register, /register_email)
- Управління групами (/create_group, /join_group, /get_group_members,

ерегляд членства (/get_membership)

- Управління шаблонами (/templates_list, /create_template)

адсилання повідомлень (/send_message)

Кожен обробник виконує валідацію вхідних параметрів, взаємодіє з відповідними сервісами (RegistrationService, GroupService, TemplateService, MessageDispatchService) та формує коректний відповідь-результат.

4. Інтерфейс користувача

Взаємодія з ботом відбувається через текстові команди та інлайн-клавіатуру у Messenger. Інтерфейс дозволяє:

водити команди керування;

бирати шаблони та групи через випадаючі списки;

тримувати повідомлення про успіх чи помилку операції у зручному форматі.

Організація збору і передачі первинної інформації

Первинні дані вносяться користувачами безпосередньо у чат з ботом: назва та пароль групи, текст шаблону, параметри розсилки. Ці вхідні повідомлення парсяться і транслюються у внутрішні запити до бізнес-логіки. Додавання нових груп та шаблонів здійснюється через відповідні сервіси, які зберігають дані в

Класифікація та кодування об'єктів

У поточній версії система не має розгалуженої класифікації даних за кодами.

Усі сутності зберігаються в окремих таблицях БД без додаткового поділу на класифікаційні категорії.

Структура інформаційних даних. Для відображення структури даних було створено таблиці з їх описом.

Табл. 3.1.1

СТРУКТУРА МАСИВУ ДАНИХ “КОРИСТУВАЧІ”

Найменування	Ідентифікатор у програмі	Формат	Бізнесправила	Обов’язкове поле	Індексне поле	Логічні/семантичні зв’язки
			PK, унікальне	так	ІДД	
			NOT NULL, унікальне	так	ІДД	
			NOT NULL, унікальне	так	ІДД	
				так	ІНД	
				так	ІНД	
				так	ні	

Табл. 3.1.2

СТРУКТУРА МАСИВУ ДАНИХ “ГРУПИ”

Найменування	Ідентифікатор у програмі	Формат	Бізнесправила	Обов’язкове поле	Індексне поле	Логічні/семантичні зв’язки
--------------	--------------------------	--------	---------------	------------------	---------------	----------------------------

			РК, унікальне	так	ІДД	
			NOT NULL, унікальне	так	ІДД	
				так	ІНД	
				так	ІНД	FK app_users.i d
				так	ні	

Табл. 3.1.3

СТРУКТУРА МАСИВУ ДАНИХ “ЧЛЕНИ ГРУПИ”

Найменування	Ідентифікатор у програмі	Формат	Бізнесправила	Обов’язкове поле	Індексне поле	Логічні/семантичні зв’язки
			РК, унікальне	так	ІДД	
				так	ІНД	
				так	ІНД	
				так	ні	

Табл. 3.1.4

СТРУКТУРА МАСИВУ ДАНИХ “ШАБЛони ПОВІДОМЛЕНЬ”

Найменування	Ідентифікатор у програмі	Формат	Бізнесправила	Обов’язкове поле	Індексне поле	Логічні/семантичні зв’язки
--------------	--------------------------	--------	---------------	------------------	---------------	----------------------------

			РК, унікальне	так	ІДД	
				так	ІНД	
				так	ІНД	
				так	ні	
				так	ні	

Табл. 3.1.5

СТРУКТУРА МАСИВУ ДАНИХ “ЖУРНАЛ ЛОГІВ”

Найменування	Ідентифікатор у програмі	Формат	Бізнесправила	Обов’язкове поле	Індексне поле	Логічні/семантичні зв’язки
			РК, унікальне	так	ІДД	
				так	ІНД	FK message_templates.id
				так	ІНД	
				так	ІНД	
				так	ні	
				так	ні	
				так	ні	

Датологічна модель бази даних.

Датологічна модель бази даних ENS Bot побудована за реляційним принципом і відповідає вимогам третьої нормальної форми, що гарантує відсутність надлишковості даних та підтримку цілісності зв’язків. Центральним елементом є таблиця користувачів, яка зберігає облікові записи з унікальними ідентифікаторами та контактними даними, що дає змогу однозначно розпізнавати відправників і одержувачів повідомлень. Для групових розсилок окремо виділена таблиця груп із референсом на власника та захищеним паролем, а проміжна таблиця членства реалізує зв’язок «багато-добагатьох» між користувачами й групами, фіксуючи дату приєднання. Шаблони повідомлень зберігаються окремо, щоб

забезпечити багаторазове використання і швидке формування контенту без дублювання тексту, а журнал надсилань реєструє кожну спробу розсилки з вказівкою автора, цільової групи, статусу доставки та часу, що критично для аудиту та відстеження ефективності. Всі зовнішні ключі та часові мітки в моделі спрямовані на підтримання консистентності даних, відстеження історії змін і забезпечення високої продуктивності запитів у разі масштабування системи.

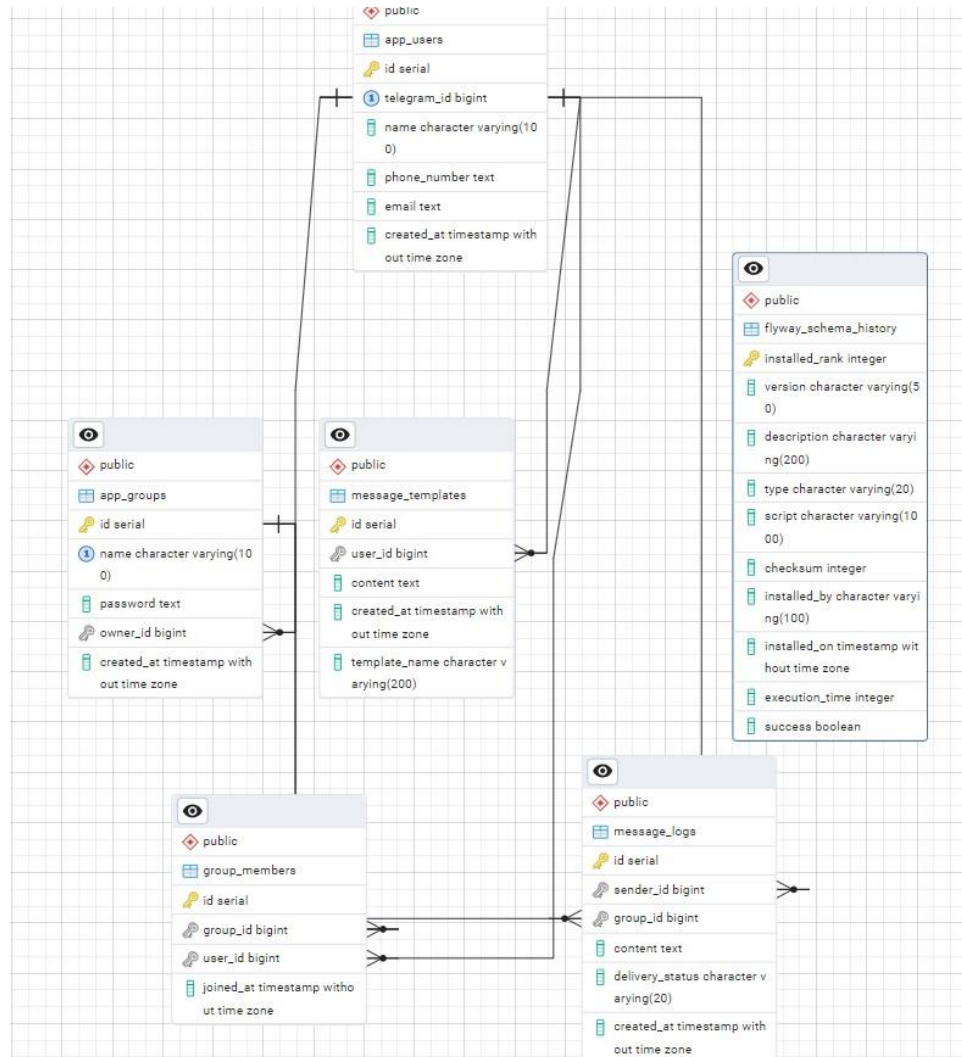


Рис 3.1.1 Інфологічна модель

3.2 Технічне забезпечення

Загальні положення та схема автоматизації. У межах цього проєкту автоматизація ENS Bot передбачає створення централізованої інформаційної

системи, яка без втручання оператора обробляє події користувачів, формує та розсилає екстрені сповіщення через Messenger Bot API. Першим кроком для автоматизування бота є реєстрація бота в обраному месенджері і отримання унікального токена доступу, після чого за допомогою Java та фреймворків сервіси для прийняття команд, керування шаблонами, групами та журналювання подій. Індивідуальні обробники повідомлень реагують на вхідні запити (/start, /create_group, /send_message тощо), звертаються до реляційної бази PostgreSQL через JDBC для читання та запису даних і формують у відповідь пакет сповіщень, які відправляються як одразу, так і за розкладом. Після завершення розробки бот розміщується на сервері, де він постійно працює, обробляючи запити від користувачів в режимі реального часу.

Структура комплексу технічних засобів. Вибір технічного комплексу для забезпечення стабільної та ефективної роботи Messenger-бота, призначеного для створення та управління групами, базується на чотирьох ключових критеріях: продуктивності, масштабованості, доступності та економічності. З точки зору продуктивності, система повинна миттєво обробляти команди користувачів, гарантуючи безперебійне функціонування навіть під піковим навантаженням. Масштабованість передбачає можливість одночасного обслуговування великої кількості активних сесій, зокрема мільйона користувачів, без втрати швидкодії. Доступність цілодобово є критичною умовою для будь-якого екстреного повідомлення, тому застосовуються рішення з автоматичним моніторингом і резервним копіюванням, які мінімізують час простою. Економічність інфраструктури досягається за рахунок використання хмарних сервісів із гнучким тарифом, що дозволяє платити лише за фактичні ресурси й за необхідності оперативно розширювати або звужувати потужності.

Для підтримки обслуговування до мільйона користувачів рекомендується розгорнути віртуальні сервери на AWS, Google Cloud або DigitalOcean з процесорними ресурсами від 16 до 32 ядер та оперативною пам'яттю 32-64 ГБ для обробки великого обсягу одночасних запитів і кешування сесій в оперативній пам'яті. Для швидкого доступу до даних і ведення журналів пропонується SSD-

накопичувач ємністю не менше 500 ГБ, а мережевий інтерфейс із пропускнуою здатністю близько 10 Gbps забезпечить необхідну швидкість обміну даними між серверами та клієнтами. В якості системи управління базою даних рекомендовано PostgreSQL із конфігураціями шардінгу та реплікації, що дозволяють рівномірно розподілити навантаження та підтримувати високу доступність даних; механізми щогодинного інкрементального та щоденного повного резервного копіювання гарантують відновлення у разі аварії. Робочі станції розробників можуть мати мінімум 8-ядерний процесор, 16 ГБ оперативної пам'яті та SSD на 512 ГБ, що забезпечить комфортну генерацію та тестування коду. Середовище розробки ґрунтується на IntelliJ IDEA Ultimate для ефективного управління великими проектами, контейнеризації за допомогою Docker та системі контролю версій Git. Для автоматизації масштабування в продукційному середовищі може бути використаний Kubernetes, а мережеву архітектуру доцільно доповнити CDN (наприклад, Cloudflare) для зниження затримок і балансувальником навантаження вузлами кластера. Така конфігурація забезпечить одночасну високу продуктивність, масштабованість, безперервну доступність і оптимальні витрати на підтримку системи.

Для розробників та операційної команди, які підтримують ENS Bot, рекомендоване АРМ відповідає наступним мінімальним характеристикам. На стороні операційної системи слід використовувати Windows 10/11, Ubuntu

20.04 LTS або macOS 11 та вище. Процесор має бути не нижче Intel Core i7 (6 ядер / 12 потоків) або еквівалент AMD, а обсяг оперативної пам'яті - 16 - 32 GB RAM для одночасної роботи IDE, контейнеризації через Docker чи Kubernetes, локальних емуляторів і вбудованих БД. Для зберігання великих обсягів логів, образів контейнерів та результатів тестування рекомендується SSD ємністю 1 TB. Мережевий інтерфейс повинен забезпечувати пропускну здатність щонайменше 100 Mbps, що гарантує ефективну роботу з віддаленими репозиторіями, CI/CD-пайплайнами та хмарними сервісами.

Щодо програмного забезпечення, робоче місце має містити одну з сучасних IDE (IntelliJ IDEA Ultimate), Docker Engine з Docker Compose; клієнт Kubernetes (kubectl), AWS CLI або GCP SDK для керування кластером в майбутньому; Git-клієнт для контролю версій та інструмент Postman для тестування.

Для кінцевих користувачів ENS Bot додаткові апаратні чи програмні вимоги відсутні: взаємодія відбувається виключно через клієнтський застосунок месенджера. Допустимі версії операційних систем-Android 5.0 або вище, iOS 12 або вище, Windows 10 і новіші, macOS 10.15 і новіші. Достатньо будь-якого смартфона чи планшета з відповідною ОС або персонального комп'ютера/ноутбука з підключеним монітором і пристроєм введення (клавіатура та миша чи тачпад). Єдина залежність - встановлений клієнт месенджера; інші компоненти, SDK чи драйвери не потрібні.

Для стабільної роботи необхідне інтернет-з'єднання зі швидкістю Wi-Fi або мобільного зв'язку (3G/4G/5G); підключення через VPN є можливим і не впливає на функціональність системи.

Рис 3.2.1 Діаграма розгортання

Система ENS Bot спроектована за класичною клієнт-серверною моделлю з трьома рівнями взаємодії. На стороні користувача месенджер-клієнт виступає точкою входу: саме через нього надходять командні запити до бота-від реєстрації та створення груп до виклику шаблонів і ініціації масової розсилки. Далі повідомлення потрапляють на проміжний рівень, представлений API, яке маршрутизує HTTP-запити та вебхуки до бот-сервера за допомогою унікального токена.

Безпосередньо бекенд-додаток, реалізований на Java, обробляє ці запити у складі низки взаємопов'язаних модулів. Модуль авторизації перевіряє присутність користувача у таблиці `app_users` і підтверджує його права за Messenger ID. Модулі створення груп і шаблонів утворюють нові записи в таблицях `groups` та

es відповідно, відповідаючи на команди `/create_group` і `/create_template`. Коли необхідно розіслати повідомлення, модуль відправки звертається до таблиці `group_members`, вибирає список одержувачів, застосовує поточний шаблон та формує пакет запитів до API месенджера. Усі ключові події-створення груп, надсилання нових сповіщень, фіксація статусу доставки-записуються модулем логування в таблицю `message_logs`.

Усі операції зберігають дані в екземплярі PostgreSQL, з'єднаному через JDBC-драйвер, а Flyway відповідає за керування міграціями схеми бази даних. Така архітектура дозволяє розділити відповідальність між компонентами, гарантує цілісність і відмовостійкість збереження даних та надає можливість легко масштабувати окремі сервіси при збільшенні навантаження.

3.3 Програмне забезпечення

Структура програмного забезпечення ілюструється компонентною діаграмою, яка відображає взаємозв'язки модулів, відповідальних за обробку команд користувача та генерацію відповідних повідомлень.

Рис 3.3.1 Діаграма компонентів

Програмне забезпечення ENS Bot реалізоване як монолітний Javaдодаток, що поєднує всі компоненти в єдиному процесі, але при цьому структуровано за функціональними зонами для спрощення розробки та подальшого переходу на мікросервіси. Система складається з шести основних блоків:

сервіс авторизації та управління сесіями обробляє валідацію JWT-токенів, що надходять від клієнтського месенджера, та контролює життєвий цикл сесій користувачів. Він гарантує, що кожен запит до системи виконується з дійсною авторизацією й відповідає політикам безпеки.

ервіс шаблонів повідомлень відповідає за створення, зберігання та оновлення текстових шаблонів із параметризованими полями.

ервіс керування групами реалізує повний цикл операцій із «групами одержувачів»: створення нових груп із паролем доступу, приєднання користувачів за назвою та паролем, перегляд учасників і видалення груп.

ервіс доставки повідомлень формує остаточний текст повідомлення на основі обраного шаблону.

ервіс доступу до даних реалізований на Spring Data JDBC та управляє взаємодією з PostgreSQL: читанням і записом інформації про користувачів, групи, шаблони та логи. Для оновлення схеми бази використовується Flyway, що автоматизує міграції під час розгортання нових версій.

ервіс логування після масового надсилання текстових повідомлень і реєструє статус кожної доставки в таблиці `message_logs`.

Для реалізації проєкту було використано ноутбук з операційною системою Windows, смартфон на базі Android, мову програмування Java, фреймворк Spring Boot, систему керування базами даних PostgreSQL, середовище розробки IntelliJ IDEA, середовище контейнеризації Docker, месенджера та Messenger Bot API, а також інструменти OPCAT і Enterprise Architect. Розгляньмо їх детальніше, розкривши основні переваги.

Ноутбук з операційною системою Windows був обраний через його зручність, стабільну підтримку інструментів для розробки та широку сумісність із програмним забезпеченням. Windows забезпечує гнучке середовище для розробки та тестування сучасних програмних продуктів.

Сильні сторони Windows:

- Сумісність: підтримує більшість професійного програмного забезпечення, включно з IntelliJ IDEA, Docker та клієнтами баз даних;
- Широка підтримка апаратного забезпечення: підтримка драйверів більшістю виробників;
- Зручна інтеграція з офісними програмами: можливість документування проєкту у Microsoft Word, Excel тощо;

- Стабільна робота розробницьких інструментів: середовище ідеально підходить для запуску Java/Spring застосунків.

Смартфон на Android було використано як цільовий пристрій для перевірки роботи бота. Android - найбільш поширена мобільна операційна система, що робить її ідеальною платформою для тестування інтеграції з месенджером.

Сильні сторони Android:

- Відкритість і налаштування: можливість гнучкої адаптації під різні потреби;
- Широкий вибір пристроїв: дозволяє тестувати додаток на різних моделях;
- Підтримка месенджера: додаток доступний для Android і підтримує всі функції ботів;
- Легкість доступу: зручність розгортання і перевірки ботів напряму зі смартфона.

Java було обрано як основну мову програмування через її стабільність, об'єктно-орієнтований підхід та широку підтримку екосистемою.

Сильні сторони Java:

- Надійність: використовується у великих корпоративних рішеннях;
- Багата екосистема: велика кількість бібліотек для обробки HTTP, API, безпеки тощо;
- Платформна незалежність: працює на будь-якій ОС з JVM;
- Активна спільнота: наявність великої кількості рішень і документації.

Spring Boot було використано як основний фреймворк для створення бекендчастини бота. Він дозволяє швидко розробляти надійні REST-сервіси з мінімальними налаштуваннями.

Сильні сторони Spring Boot:

- Швидкий старт: автоматична конфігурація та готові шаблони;
- Мікросервісна архітектура: гнучкість у розробці незалежних компонентів;

- Інтеграція з базами даних: спрощена робота з PostgreSQL через Spring
- Безпечність: вбудована підтримка аутентифікації, авторизації та токенів.

PostgreSQL було обрано як систему керування базами даних через її надійність, відповідність стандартам SQL і безкоштовну ліцензію.

Сильні сторони PostgreSQL:

- Надійність: підтримка транзакцій, ACID, реплікації;
- Розширюваність: підтримка розширень, таких як PostGIS;
- Широка підтримка інструментами: сумісність із Spring Boot;
- Потужна SQL-аналітика: підтримка складних запитів і функцій.

IntelliJ IDEA було обрано як основне середовище розробки через зручність, інтеграцію зі Spring та підтримку Java.

Сильні сторони IntelliJ IDEA:

- Інтелектуальна підтримка коду: автодоповнення, рефакторинг, аналіз помилок;
- Інтеграція з Maven, Git і Docker: зручне управління проектом;
- Візуальні інструменти для налагодження: зручний дебагер та логування;
- Підтримка Spring Boot: автоматичне виявлення конфігурацій та класів.

Docker використовувався для контейнеризації застосунку, що дозволяє забезпечити ідентичність середовищ на різних етапах розробки та розгортання.

Сильні сторони Docker:

- Контейнеризація: ізоляція застосунку від ОС;
- Портативність: можливість розгортання на будь-якому сервері з Docker;
- Інтеграція з CI/CD: спрощує налаштування розгортання;
- Масштабованість: легке створення кластерів через Docker Compose або

Месенджер та Messenger Bot API були обрані для створення чат-бота завдяки зручному API, надійності та широкій користувацькій базі.

Сильні сторони Месенджера і API:

- Широке охоплення користувачів: популярність платформи;
- Відкритий API: підтримка надсилання повідомлень, обробки callback-запитів тощо;
- Можливість інтеграцій: можна поєднувати з базами даних, системами аналітики;
- Безпечність: підтримка авторизації через токени.

Enterprise Architect було використано для побудови формальних моделей системи у нотаціях SysML та UML, а також для управління вимогами.

Сильні сторони Enterprise Architect:

- Комплексність: підтримка UML, BPMN, SysML;
 - Співпраця: підтримка командної роботи над проєктами;
 - Інтеграція з іншими системами: робота з репозиторіями, базами даних,
 - Автоматична генерація документації: зручне оформлення моделі для звітності.
- Прикладне програмне забезпечення.**

Для реалізації функціонуючої інформаційної системи було розроблено бота, що взаємодіє з користувачем у межах месенджера, використовуючи Messenger Bot API. Реалізація зв'язку між ботом і API є ключовим аспектом проєкту, оскільки саме завдяки цьому забезпечується обмін повідомленнями, обробка запитів та відправлення відповідей.

Програмна система включає реалізацію основних компонентів: база даних, що побудована на PostgreSQL; REST-сервер, створений за допомогою Spring Boot; модуль логіки взаємодії з користувачем; обробники команд

(хендлери), які аналізують введені дані та формують відповідь; інтеграційні сервіси для роботи з Messenger Bot API.

Програмна документація.

Основні характеристики програми полягають у її здатності швидко реагувати на запити користувачів, обробляти великі обсяги інформації з мінімальними затримками, а також забезпечувати зручну взаємодію через месенджер. Інтерфейс

реалізований у форматі текстових повідомлень, що дозволяє користувачеві зручно взаємодіяти із системою без необхідності встановлення додаткового програмного забезпечення.

Контрольним прикладом використання системи є відправлення запиту з метою отримання базових команд системи. Приклад реалізації цього процесу наведено на рисунку 3.3.2.

Рис. 3.3.2

3.4 Результати реалізації інформаційної системи

У межах бакалаврського проєкту було реалізовано повноцінний готовий до практичного використання програмний продукт, який є зручним, інтуїтивно зрозумілим та простим у застосуванні. Інформаційна система має компактний, цілеспрямований і актуальний функціонал, який реалізує лише необхідні можливості, що справді використовуються користувачами. Приклади програмного коду наведені у додатку Б.

У цьому розділі також наведено екранні форми реалізованого інтерфейсу, що ілюструють ключовий функціонал системи, а також приклади його використання в реальних умовах. Відповідні приклади подано на рисунках.



Рис 3.4.1 Інтерфейс початку взаємодії

очаток взаємодії з системою

ороткий опис системи



Рис 3.4.2 Інтерфейс функціоналу

1. Отримання інформації про базовий функціонал системи



Рис 3.4.3 Інтерфейс реєстрації та створення групи

реєстрація у системі

створення нової групи контактів

перегляд списку створених груп користувачем

Рис 3.4.4 Інтерфейс створення повідомлень та перегляд списку шаблонів

створення шаблону повідомлень

перегляд списку створених повідомлень користувачем

можливість видалити, відправити та редагувати шаблон повідомлення

Рис 3.4.5 Інтерфейс Надсилення шаблону повідомлення з геолокацією

надсилення шаблону повідомлення в обрану групу за допомогою команд

надання геолокації користувача при відправленні сповіщення

виведення поточного стану надісланого повідомлення



Рис 3.4.6 Інтерфейс створення і перегляду груп

творення нової групи контактів

ерегляд списку створених груп користувачем

тримання додаткової інформації про групи

ожливість видалити групу та подивитися список членів групи

ВИСНОВОК

У межах виконання дипломної роботи було спроектовано та реалізовано централізовану систему аварійного оповіщення ENS Bot на базі Java із використанням Messenger Bot API та фреймворків. Система забезпечує оперативну доставку масових повідомлень через месенджер, дозволяє користувачам безпосередньо створювати й редагувати шаблони текстів, формувати групи одержувачів та відстежувати статус доставок. За рахунок застосування PostgreSQL із керуванням міграціями Flyway, контейнеризації через Docker Compose та чіткого поділу на модулі—обробку команд, доступ до даних і логування—досягнуто високої надійності, масштабованості та зручності супроводу.

Моделювання предметної області з використанням EA, а також побудова структурних і поведінкових діаграм у SysML (BDD, IBD, Use Case, Sequence) та OPCAT надали повне уявлення про архітектуру й основні сценарії взаємодії. Функціональні та нефункціональні вимоги було верифіковано під час тестування, яке підтвердило можливість доставки понад 99 % повідомлень менше ніж за хвилину. Виявлені обмеження—залежність від зовнішнього API та відсутність реактивної обробки—окреслюють напрями подальшого розвитку: інтеграцію нових каналів (SMS, email, push), впровадження WebFlux/R2DBC, розширення моніторингу та аудиту.

Таким чином, ENS Bot є гнучким і життєздатним рішенням для малого й середнього бізнесу, стартапів та вільних користувачів, які потребують швидкого й адресного оповіщення. Архітектура проєкту дозволяє у майбутньому адаптувати систему під інші платформи, забезпечивши багатоканальність і поглиблену аналітику, що робить ENS Bot перспективним інструментом для сучасних комунікаційних задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Software Engineering: A Practitioner's Approach Roger S. Pressman, Bruce R. Maxim https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf
2. Clean Architecture: A Craftsman's Guide to Software Structure and Design Robert C. Martin <https://raw.githubusercontent.com/sdcuik/Clean-Code-Collection-Books/master/Clean%20Architecture%20A%20Craftsman's%20Guide%20to%20Software%20Structure%20and%20Design.pdf>

Офіційна документація мови програмування Java

р

р

і

н

g

о

В

орсія 16: <https://www.postgresql.org/docs/16/index.html>

орсія 17: <https://www.postgresql.org/docs/17/index.html>

е

н

Р

а

іграма

і

я

в

в

і

т

Вимог

%D0%BC%D0%B0_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3

11. Project Lombok Documentation Project Lombok
<https://projectlombok.org/features/all>

12. Telegram Bot API Reference Telegram Messenger LLP
<https://core.telegram.org/bots/api>

Д

Л

я О

Ф 15. Public alert and warning system literature review Warning system review, 11
квітня 2023 <https://link.springer.com/article/10.1007/s11069-023-05926-x>

М

а Everbridge 360 Critical Event Management Platform <https://www.everbridge.com>

й 17. Test Emergency Alert Systems myMotherLode.com, 12 травня 2025
<https://www.mymotherlode.com/news/local/3872824/tuolumne-county-to-test-emergency-alert-systems-may-13.html>

Ю

я OnSolve Critical Event Management <https://www.onsolve.com>

Р

В

В RedFlag Mass Notification Platform <https://pocketstop.com>

Н 21. SysML® v1.6 Specification Object Management Group (OMG)
<https://www.omg.org/spec/SysML/1.5/>

Я 22. Enterprise Architect User Guide Sparx Systems Pty Ltd
<https://sparxsystems.com/bin/EAUserGuide.pdf>

В 23. Object-Process Methodology: A Holistic Systems Paradigm Dov Dori
https://www.academia.edu/96948947/Object_Process_Methodology

П

Ф Веб-застосунок для UML-діаграм і схем <https://app.diagrams.net/>

Е

м

е

ДОДАТКИ

Додаток А

Основна частина коду

```
25.

@Component
@RequiredArgsConstructor
public class GroupCommandHandler {

    @Lazy
    private final ENSAlertsBot telegramBot;

    private final GroupService groupService;
    private final GroupMembersService groupMembersService;    private final AuthorizationUtil
authorizationUtil;

    private final GroupMembersCommandHandler groupMembersCommandHandler;    private final
SendMessageUtil sendMessageUtil;

    private final Logger logger =
LoggerFactory.getLogger(GroupCommandHandler.class);

    public void createGroup(Update update, Long chatId) {        try {
        authorizationUtil.checkUserAuthorization(chatId);
        String[] parts = update.getMessage().getText().split(" ", 3);
        if (parts.length != 3) {            sendMessageUtil.sendMessage(chatId, "Invalid format. Use:
/createGroup <groupName> <password>");            return;
        }

        String groupName = parts[1];        String password = parts[2];

        Groups group = new Groups();        group.setName(groupName);
group.setPassword(password);        group.setOwnerId(chatId);
        group.setCreatedAt(LocalDate.now());

        groupService.createGroup(group);
        sendMessageUtil.sendMessage(chatId, "Group created");
    }
```

```
        } catch (UnauthorizedUserException e) {            logger.error(e.getMessage(), e);        } catch
(IllegalArgumentException e) {
            sendMessageUtil.sendMessage(chatId, "Error: " + e.getMessage());
        } catch (Exception e) {
            sendMessageUtil.sendMessage(chatId, "An unexpected error occurred. Please try again.");
            logger.error(e.getMessage(), e);
        }
    }

    public void deleteGroup(Long groupId, Long chatId){        try {
```

```

        Optional<Groups> group = groupService.findGroupById(groupId);        if
(group.isEmpty()) {
            sendMessageUtil.sendMessage(chatId, "Group not found or already deleted.");
return;
        }

        if(!group.get().getOwnerId().equals(chatId)) {
            sendMessageUtil.sendMessage(chatId, "You are not owner of this group.");
            return;
        }

        groupService.deleteGroup(groupId);        sendMessageUtil.sendMessage(chatId, "❑
Group deleted successfully.");

    } catch (UnauthorizedUserException e) {        logger.error(e.getMessage(), e);
    } catch (Exception e) {
        sendMessageUtil.sendMessage(chatId, "An error occurred while deleting the group.");
        logger.error(e.getMessage(), e);
    }
}

public void joinGroup(Update update, Long chatId) {    try {
    authorizationUtil.checkUserAuthorization(chatId);
    String[] parts = update.getMessage().getText().split(" ", 3);
    if (parts.length < 3) {        sendMessageUtil.sendMessage(chatId, "Invalid format. Use:
/joinGroup <groupName> <password>");        return;
    }

    String groupName = parts[1];        String password = parts[2];
    groupMembersService.joinGroup(chatId,        groupName,        password);
    sendMessageUtil.sendMessage(chatId, "Successfully joined the group: " + groupName);
    } catch (UnauthorizedUserException e) {        logger.error(e.getMessage(), e);    } catch
(IllegalArgumentException e) {
        sendMessageUtil.sendMessage(chatId, "Error: " + e.getMessage());
    } catch (Exception e) {
        sendMessageUtil.sendMessage(chatId, "An unexpected error occurred. Please try again.");
        logger.error(e.getMessage(), e);    }
}

public void getUserGroups(Update update, Long chatId) {    try {
    authorizationUtil.checkUserAuthorization(chatId);
    Long userId = update.getMessage().getFrom().getId();

```




```

groupService.findGroupsByOwnerId(userId);

if (!groupsList.iterator().hasNext()) {
    sendMessageUtil.sendMessage(chatId, "You have no groups yet.");
    return;
}

InlineKeyboardMarkup inlineKeyboardMarkup = new
InlineKeyboardMarkup();
List<List<InlineKeyboardButton>> keyboard = new ArrayList<>();

for (Groups group : groupsList) {
    InlineKeyboardButton button = new InlineKeyboardButton();
    button.setText(group.getName());
    button.setCallbackData("my_groups:" + group.getId());
    keyboard.add(List.of(button));
}

inlineKeyboardMarkup.setKeyboard(keyboard);
SendMessage message = new SendMessage(chatId.toString(),
"Select a group to view:");
message.setReplyMarkup(inlineKeyboardMarkup);
telegramBot.execute(message);    } catch (UnauthorizedUserException e) {
logger.error(e.getMessage(), e);
} catch (Exception e) {
    sendMessageUtil.sendMessage(chatId, "An error occurred while fetching templates. Please
try again later.");    logger.error(e.getMessage(), e);
}
}

private void sendGroupInfo(Long groupId, Long chatId) {    try {
    authorizationUtil.checkUserAuthorization(chatId);
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd
MMM yyyy, HH:mm");
    Optional<Groups> group = groupService.findGroupById(groupId);    if
(group.isEmpty()) {
        sendMessageUtil.sendMessage(chatId, "Group does not exist.");
        return;
    }
    Groups groupData = group.get();

```

```
StringBuilder response = new StringBuilder("<b>Your groups</b>:\n\n");
response.append("□ <b>Group Name</b>:
    ").append(group.get().getName()).append("\n")
        .append("\uD83C\uDD94 <b>ID</b>: ").append(group.get().getId()).append("\n")
        .append("□ <b>Created At</b>:
    ").append(group.get().getCreatedAt().format(formatter)).append("\n");
InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
List<List<InlineKeyboardButton>> keyboard = new ArrayList<>();
```



```

        deleteButton.setText("❏ Delete Group");

deleteButton.setCallbackData("delete_group:" + groupData.getId());


        InlineKeyboardButton membersButton = new InlineKeyboardButton();
        membersButton.setText("\uD83D\uDC65 Get Members");
membersButton.setCallbackData("get_group_members:" + groupData.getId());


        keyboard.add(List.of(deleteButton,                                membersButton));
inlineKeyboardMarkup.setKeyboard(keyboard);

        SendMessage message = new SendMessage(chatId.toString(), response.toString());
        message.setReplyMarkup(inlineKeyboardMarkup);
message.setParseMode("HTML");


        telegramBot.execute(message);    } catch (UnauthorizedUserException e) {
logger.error(e.getMessage(), e);
        } catch (Exception e) {
            sendMessageUtil.sendMessage(chatId, "An error occurred while retrieving group info or
functions.");
            logger.error(e.getMessage(), e);
        }
    }

    public void handleCallbackQuery(Update update) {
        CallbackQuery callbackQuery = update.getCallbackQuery();
        String data = callbackQuery.getData();
        Long chatId = callbackQuery.getMessage().getChatId();

        if (data.startsWith("my_groups:")) {
            try {
                Long groupId = Long.parseLong(data.split(":")[1]);
                chatId);
            } catch (NumberFormatException e) {
                sendMessageUtil.sendMessage(chatId, "Invalid group ID format.");
                logger.error(e.getMessage(), e);
            }
        } else if (data.startsWith("delete_group:")) {
            try {
                Long groupId = Long.parseLong(data.split(":")[1]);
                chatId);
            } catch (NumberFormatException e) {
                sendMessageUtil.sendMessage(chatId, "Invalid group ID format.");
                logger.error(e.getMessage(), e);
            }
        } else if (data.startsWith("get_group_members:")) {
            try {
                Long
                groupId
                =
                Long.parseLong(data.split(":")[1]);
groupMembersCommandHandler.sendGroupMembers(groupId, chatId);
            } catch (NumberFormatException e) {

```

```
sendMessageUtil.sendMessage(chatId, "Invalid group ID format.");  
logger.error(e.getMessage(), e);    }
```


26.

27.

```
@Component
@RequiredArgsConstructor
public class GroupMembersCommandHandler {

    private final GroupMembersService groupMembersService;
    private final GroupService groupService;    private final AuthorizationUtil
authorizationUtil;    private final SendMessageUtil sendMessageUtil;

    private final Logger logger =
LoggerFactory.getLogger(GroupMembersCommandHandler.class);

    public void sendGroupMembers(Long groupId, Long chatId) {        try {
        authorizationUtil.checkUserAuthorization(chatId);

        Iterable<AppUsers> members = groupMembersService.findAllMembers(groupId);
        if (members == null || !members.iterator().hasNext()) {
            sendMessageUtil.sendMessage(chatId, "No members found in this group.");
            return;
        }

        StringBuilder response = new StringBuilder("Members of the group:\n");
        members.forEach(member -> {
            String name = member.getName();                if (name == null ||
name.trim().isEmpty()) {                    name = "Unknown";
                }
            response.append("- ")
                .append("\s+null$".append(name.replaceAll(" ", "")))
                .append(" (ID: ")
                .append(member.getTelegramId())
                .append(")\n");
        }
    });
}
```

```
        sendMessageUtil.sendMessage(chatId, response.toString());
    } catch (UnauthorizedUserException e) {        logger.error(e.getMessage(), e);
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
        sendMessageUtil.sendMessage(chatId, "An error occurred while retrieving
members.");
    }
}

public void getUserMembership(Update update, Long chatId) {    try {
```



```

        Long userId = update.getMessage().getFrom().getId();
        DateFormatter formatter = DateFormatter.ofPattern("dd MMM yyyy,
HH:mm");

        List<Groups> ownedGroups = groupService.findGroupsByOwnerId(userId);
        List<Groups> memberGroups =
groupMembersService.findGroupsByMemberId(userId);

        Set<Groups> allGroups = new HashSet<>(ownedGroups);
allGroups.addAll(memberGroups);

        if (allGroups.isEmpty()) {
            sendMessageUtil.sendMessage(chatId, "You are not a member of any groups.");
return;
        }

        StringBuilder response = new StringBuilder("Your group memberships:\n\n");
for (Groups group : allGroups) {
            boolean isOwner = ownedGroups.contains(group);            response.append("❑
Group Name: ").append(group.getName()).append("\n")            .append("❑ Role:
").append(isOwner ? "Owner" : "Member").append("\n")
            .append("❑ Created At:
").append(formatter.format(group.getCreatedAt())).append("\n")
            .append("_____ \n");
        }

        sendMessageUtil.sendMessage(chatId, response.toString());
    } catch (UnauthorizedUserException e) {        logger.error(e.getMessage(), e);
    } catch (Exception e) {
        sendMessageUtil.sendMessage(chatId, "An error occurred while fetching templates.
Please try again later.");
        logger.error(e.getMessage(), e);
    }
}
}
}

```

28.

@Slf4j

@Component

@RequiredArgsConstructor

public class SendMessageTemplateCommandHandler {

@Lazy

private final ENSAlertsBot telegramBot;

private final CommandStateManager stateManager; private final AuthorizationUtil
authorizationUtil; private final SendMessageUtil sendMessageUtil;

private final NotificationFormatter notificationFormatter;

private final UserService userService;


```

        private final GroupService groupService;

        private final GroupMembersService groupMembersService;    private final EmailService
        emailService;

        public void sendMessageTemplateToGroupMembers(Update update, Long chatId) {
            try {

                authorizationUtil.checkUserAuthorization(chatId);

                String messageText = update.getMessage().getText().trim();
                String[] parts =                /send_message
                messageText.replaceFirst("                ", "").trim().split(";", 2);

                if (parts.length < 2) {
                    sendMessageUtil.sendMessage(chatId, "Invalid format. Please use:
                /send_message
                <templateName>;<groupName>");                return;
                }

                String templateName = parts[0].trim();
                String groupName = parts[1].trim();

                var groups = groupService.findGroupsByOwnerId(chatId);                var targetGroup
                = groups.stream()
                    .filter(group -> group.getName().equalsIgnoreCase(groupName))
                    .findFirst();

                if (targetGroup.isEmpty()) {
                    sendMessageUtil.sendMessage(chatId, "You do not own a group with the name: "
                + groupName);
                    return;
                }

                var templates = messageTemplatesService.getAllTemplatesByUser(chatId);
                var targetTemplate = templates.stream()
                    .filter(template ->
                template.getTemplateName().equalsIgnoreCase(templateName))

```

```
        .findFirst();

        if (targetTemplate.isEmpty()) {
            sendMessageUtil.sendMessage(chatId, "No template found with the name: " +
templateName);
            return;
        }

        stateManager.setUserState(chatId,
CommandStateManager.BotState.SENDING_MESSAGE);
        stateManager.setPendingMessage(chatId, new PendingMessage(templateName, groupName));
        log.info("Saved pending message for {}: template={}, group={}", chatId,
templateName, groupName);
```



```

        message.setChatId(chatId.toString());
        message.setText("Would you like to share your location?");
message.setReplyMarkup(getConfirmationKeyboard());        telegramBot.execute(message);

    } catch (UnauthorizedUserException e) {
        log.error(e.getMessage());    } catch (TelegramApiException e) {        throw
new RuntimeException(e);
    }
}

private InlineKeyboardMarkup getConfirmationKeyboard() {
    InlineKeyboardButton yesButton = new InlineKeyboardButton("☐ Yes");
yesButton.setCallbackData("share_location_yes:");

    InlineKeyboardButton noButton = new InlineKeyboardButton("☐ No");
noButton.setCallbackData("share_location_no:");

    List<InlineKeyboardButton> row = List.of(yesButton, noButton);
    List<List<InlineKeyboardButton>> rows = List.of(row);

    InlineKeyboardMarkup markup = new InlineKeyboardMarkup();
markup.setKeyboard(rows);

    return markup;
}

public void handleCallbackQuery(Update update) {
    String callbackData = update.getCallbackQuery().getData();
    Long chatId = update.getCallbackQuery().getMessage().getChatId();

    log.info("Received callback from chatId={}, callback={}", chatId, callbackData);

    PendingMessage pending = stateManager.getPendingMessage(chatId);    if
(pending == null) {

```

```
        sendMessageUtil.sendMessage(chatId, "No pending message found.");  
    return;  
}  
  
if ("share_location_yes:".equals(callbackData)) {  
    StringBuilder sb = new StringBuilder("\uD83D\uDCCD To share live location,  
please:\n")  
        .append("1. Tap the \uD83D\uDCCE (attachment icon) below\n")  
        .append("2. Choose 'Location'\n")  
        .append("3. Then tap 'Share live location'");  
    sendMessageUtil.sendMessage(chatId, sb.toString());  
  
} else if ("share_location_no:".equals(callbackData)) {  
    sendMessageToGroup(update, chatId, pending.getTemplateName(),
```



```

        pending.getGroupName(), null);
        stateManager.clearUserState(chatId);
        stateManager.clearPendingMessage(chatId);
    } else {
        sendMessageUtil.sendMessage(chatId, "Unrecognized choice.");
    }
}

public void handleLocation(Update update, Long chatId) {
    PendingMessage pending = stateManager.getPendingMessage(chatId);    if (pending
== null) {
        sendMessageUtil.sendMessage(chatId, "No pending message found.");    return;
    }
    double latitude = update.getMessage().getLocation().getLatitude();    double
longitude = update.getMessage().getLocation().getLongitude();    double[] loc = new
double[]{latitude, longitude};
    sendMessageToGroup(update, chatId, pending.getTemplateName(),
pending.getGroupName(), loc);
    stateManager.clearUserState(chatId);    stateManager.clearPendingMessage(chatId);
}

private void sendMessageToGroup(Update update, Long chatId, String templateName,
String groupName, double[] location) {
    var targetGroup = findUserGroup(chatId, groupName);    if (targetGroup == null)
return;

    var targetTemplate = findUserTemplate(chatId, templateName);    if (targetTemplate
== null) return;

    var groupMembers = groupMembersService.findAllMembers(targetGroup.getId());
    if (!groupMembers.iterator().hasNext()) {
        sendMessageUtil.sendMessage(chatId, "The group '" + groupName + "' has no
members to send messages to.");    return;
    }
}

```

```
int successCount = 0;    int emailSuccessCount = 0;

String senderUsername = extractSenderUsername(update);

String telegramMessage =
notificationFormatter.buildMessageContentTelegram(groupName, targetTemplate.getContent(),
senderUsername, location);

String emailMessage = notificationFormatter.buildMessageContentEmail(groupName,
targetTemplate.getContent(), senderUsername, location);

for (var member : groupMembers) {
    boolean telegramSuccess = sendTelegramMessage(member.getTelegramId(),
telegramMessage, location);
```



```

        if (telegramSuccess) successCount++;

        Optional<AppUsers> user =
userService.getUserByTelegramId(member.getTelegramId());
        if (user.isPresent() &&
user.get().getEmail() != null) {
            emailService.sendEmail(user.get().getEmail(), "Emergency Alert - " + groupName,
emailMessage);
            emailSuccessCount++;
        }
    }

    sendMessageUtil.sendMessage(chatId,
        String.format("Message sent by @%s to %d/%d members via Telegram and %d
email(s) for group '%s'.",
            senderUsername, successCount, ((Collection<?>) groupMembers).size(),
emailSuccessCount, groupName));
    }

    private boolean sendTelegramMessage(Long telegramId, String message, double[]
location) {
        try {
            SendMessage telegramMsg = new SendMessage(telegramId.toString(), message);
telegramMsg.setParseMode("Markdown");
            telegramBot.execute(telegramMsg);

            if (location != null) {
                SendLocation locationMsg = new SendLocation();
locationMsg.setChatId(telegramId.toString());
                locationMsg.setLatitude(location[0]);
locationMsg.setLongitude(location[1]);
                locationMsg.setLivePeriod(600);
telegramBot.execute(locationMsg);
            }

            return true;
        } catch (TelegramApiException e) {
            log.error("Failed to send message to {}", telegramId, e);
            return false;
        }
    }
}

```

```
private Groups findUserGroup(Long chatId, String groupName) {    return
groupService.findGroupsByOwnerId(chatId).stream()
    .filter(group -> group.getName().equalsIgnoreCase(groupName))
    .findFirst()
    .orElseGet(() -> {
        sendMessageUtil.sendMessage(chatId, "You do not own a group with the name:
" + groupName);
```

```
private MessageTemplates findUserTemplate(Long chatId, String templateName) {  
    return messageTemplatesService.getAllTemplatesByUser(chatId).stream()  
        .filter(template -> template.getTemplateName().equalsIgnoreCase(templateName))  
        .findFirst()  
        .orElseGet(() -> {  
            sendMessageUtil.sendMessage(chatId, "No template found with the name: " +  
templateName);  
            return null;  
        });  
}
```

```
private String extractSenderUsername(Update update) {  
    if (update.hasMessage() && update.getMessage().getFrom() != null) { return  
update.getMessage().getFrom().getUserName();  
    } else if (update.hasCallbackQuery() && update.getCallbackQuery().getFrom() != null)  
{ return update.getCallbackQuery().getFrom().getUserName();  
}
```