Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка прототипу програмного забезпечення для об'єктної локального оповіщення при надзвичайних ситуаціях з використанням технологій .NET

Виконав: студент IV курсу, групи СП-43
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Максим'як Д.Ю.
(підпис) (прізвище та ініціали)

Керівник _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Ревнюк О.А.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Максим'як Данило Юрійович

1. Тема роботи Розробка прототипу програмного забезпечення для об'єктової системи локального оповіщення
віщення при надзвичайних ситуаціях з використанням технологій .NET

Керівник роботи к.т.н. доц. Цуприк Галина Богданівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 06 » квітня 2026 року № 4/9-171

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи Технічне завдання

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ.
1 Аналіз вимог та огляд предметної області.
2. Проєктування
3. Конструювання та тестування.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації, діаграма варіантів використання системи, схема бази даних, схема архітектури застосунку, схема розгортання системи у безсерверному середовищі.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			
Нормоконтроль	Стоянов Ю.М. к.т.н., доц. каф ПІ		

7. Дата видачі завдання ___6 квітня_____ 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	*Виконано*
2.	*Робота над першим розділом « Аналіз вимог до прототипу програмного забезпечення »*	*13.04 – 26.04*	*Виконано*
3.	*Робота над другим розділом « Проєктування та розробка прототипу програмного забезпечення»*	*27.04 – 03.05*	*Виконано*
4.	*Робота над третім розділом «Тестування, впровадження та підтримка»*	*04.05 – 17.05*	*Виконано*
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	*Виконано*
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	*Виконано*
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	*Виконано*
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	*Виконано*
9.	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ Максим'як Д.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка прототипу програмного забезпечення для об’єктової системи локального оповіщення при надзвичайних ситуаціях з використанням технологій .NET // Кваліфікаційна робота освітнього рівня «Бакалавр» // Максим’як Данило Юрійович // ТНТУ ім. І. Пулюя, ФІС, кафедра програмної інженерії, гр. СП–43 // Тернопіль, 2026 // С. 80, рис. – 22, табл. – 15, слайд. – 12, додат. – 6, бібліогр. – 43.

Ключові слова: цивільний захист; об’єктова система локального оповіщення; надзвичайна ситуація; програмний прототип; .NET; ASP.NET Core; SignalR; аудит.

Кваліфікаційна робота присвячена розробці програмного прототипу об’єктової системи локального оповіщення при надзвичайних ситуаціях з використанням технологій .NET.

У першому розділі проаналізовано предметну область, нормативний контекст цивільного захисту, ролі користувачів, та сценарії.

У другому розділі обґрунтовано вибір технологій .NET, ASP.NET Core, Entity Framework Core, SQLite, SignalR та ASP.NET Core Identity, спроєктовано архітектуру.

У третьому розділі описано тестування, локальне розгортання, системні вимоги та верифікацію програмного прототипу.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці оператора системи локального оповіщення, зокрема організацію дій персоналу під час оповіщення.

Об’єкт дослідження: процес організації об’єктового локального оповіщення осіб, які перебувають на території об’єкта або в його охоронних зонах.

Предмет дослідження: програмні моделі, архітектурні рішення та засоби реалізації прототипу об’єктової системи локального оповіщення з використанням технологій .NET.

ABSTRACT

Development of a Software Prototype for a Facility-Level Local Emergency Alerting System Using .NET Technologies // Qualification work for obtaining the Bachelor's degree // Maksymiak Danylo Yuriiovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP–43 // Ternopil, 2026 // 80 pages, 22 figures, 15 tables, 12 slides, 6 appendices, 43 references.

Keywords: civil protection; facility-level local alerting system; emergency situation; software prototype; .NET; ASP.NET Core; SignalR; audit.

The qualification work is devoted to the development of a software prototype for a facility-level local emergency alerting system using .NET technologies.

The first chapter analyzes the subject area, the regulatory context of civil protection, user roles, and alerting scenarios.

The second chapter justifies the choice of .NET, ASP.NET Core, Entity Framework Core, SQLite, SignalR, and ASP.NET Core Identity technologies, and presents the designed system architecture.

The third chapter describes testing, local deployment, system requirements, and verification of the software prototype.

The fourth chapter considers life safety and occupational safety issues related to the work of the local alerting system operator, in particular the organization of personnel actions during alerting.

Object of research: the process of organizing facility-level local alerting of persons located on the facility premises or within its protection zones.

Subject of research: software models, architectural solutions, and implementation tools for a prototype of a facility-level local alerting system using .NET technologies.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

АСЦО – автоматизована система централізованого оповіщення.

АРМ – автоматизоване робоче місце оператора системи.

БД – база даних, у якій зберігаються події оповіщення, отримувачі, повідомлення, спроби доставки та записи аудиту.

ДСНС – Державна служба України з надзвичайних ситуацій.

МВС – Міністерство внутрішніх справ України.

МАСЦО – місцева автоматизована система централізованого оповіщення.

НС – надзвичайна ситуація, що потребує організованого реагування та своєчасного доведення інформації до визначених осіб.

СУБД – система управління базами даних.

ТАСЦО – територіальна автоматизована система централізованого оповіщення.

ЦЗ – цивільний захист як система заходів, спрямованих на запобігання надзвичайним ситуаціям, реагування на них і захист населення та територій.

API – Application Programming Interface, прикладний інтерфейс взаємодії програмних компонентів.

ASP.NET Core – фреймворк платформи .NET для створення вебзастосунків, API та серверних програмних систем.

ASP.NET Core Identity – компонент ASP.NET Core для автентифікації користувачів, керування обліковими записами, ролями та політиками доступу.

C# – мова програмування, що використовується для реалізації серверної логіки програмного прототипу.

DB – Database, англомовне скорочення для позначення бази даних у діаграмах і технічних схемах.

EF Core – Entity Framework Core, об’єктно-реляційний засіб доступу до бази даних у середовищі .NET.

ER-діаграма – Entity-Relationship diagram, діаграма сутностей і зв’язків, яка використовується для подання структури бази даних.

HTTP – HyperText Transfer Protocol, протокол передавання даних у вебсередовищі.

HTTPS – захищена версія протоколу HTTP із використанням шифрування.

ID – Identifier, унікальний ідентифікатор запису, об’єкта або сутності в програмній системі.

JSON – JavaScript Object Notation, текстовий формат обміну структурованими даними.

LINQ – Language Integrated Query, засіб мови C# для формування запитів до колекцій даних і джерел даних.

RBAC – Role-Based Access Control, модель керування доступом на основі ролей користувачів.

SignalR – бібліотека ASP.NET Core для передавання оновлень між сервером і клієнтами в реальному часі.

SMS – Short Message Service, канал коротких текстових повідомлень.

SQL – Structured Query Language, мова роботи з реляційними базами даних.

UML – Unified Modeling Language, мова графічного моделювання програмних систем.

WebUI – вебінтерфейс користувача або вебмодуль програмного прототипу.

Журнал аудиту – сукупність записів про критичні дії користувачів і системні події, що використовуються для контролю та подальшого аналізу роботи системи.

Об’єктова система оповіщення – система, що функціонує на визначеному об’єкті та забезпечує оповіщення керівників, працівників і осіб, які перебувають на його території або в охоронних зонах.

Оповіщення – процес доведення інформації про загрозу або виникнення надзвичайної ситуації до визначених отримувачів.

Програмний прототип – спрощена реалізація програмної системи, призначена для демонстрації основної логіки роботи, архітектурних рішень і функціональних можливостей.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Ефективне реагування на надзвичайні ситуації залежить від своєчасного доведення інформації до осіб, які приймають рішення, персоналу об’єкта та інших людей, що можуть перебувати у зоні небезпеки. У сучасних умовах для цього недостатньо лише технічних засобів на кшталт сирен, гучномовців або світлових покажчиків. Необхідною є програмна підтримка, яка забезпечує реєстрацію події, визначення отримувачів, формування повідомлень, контроль їх доставки, отримання підтверджень та збереження історії виконаних дій.

Відповідно до Кодексу цивільного захисту України, оповіщення полягає у своєчасному доведенні інформації про загрозу або виникнення надзвичайної ситуації до органів управління цивільного захисту, сил цивільного захисту, суб’єктів господарювання та населення. Постанова Кабінету Міністрів України № 733 визначає порядок організації оповіщення та передбачає функціонування, зокрема, об’єктових систем оповіщення, призначених для інформування керівників, працівників та інших осіб, які перебувають на території відповідного об’єкта або в його охоронних зонах. Вимоги до автоматизованого приймання, передавання, підтвердження та протоколювання інформації деталізуються в Інструкції МВС № 93.

Нормативний аналіз засвідчив, що за функціональними межами розробка моделює програмну частину об’єктової системи оповіщення: вона призначена для інформування осіб на території визначеного об’єкта та не заявляється як сертифікована локальна система цивільного захисту.

Актуальність розробки зумовлена тим, що процеси оповіщення на рівні окремого об’єкта часто залишаються фрагментованими. Рішення щодо запуску оповіщення можуть фіксуватися окремо, списки отримувачів підтримуються вручну, повідомлення надсилаються різними каналами, а результати доставки не завжди документуються в єдиному журналі.

Такий підхід ускладнює оперативний контроль, перевірку факту інформування отримувачів і подальший аналіз дій під час надзвичайної ситуації. Програмний прототип дає змогу показати, як ці процеси можуть бути об’єднані в єдину керовану систему.

Метою роботи є розробка програмного прототипу об’єктової системи оповіщення на платформі .NET, який демонструє створення події, визначення отримувачів, формування та імітаційну доставку повідомлень, контроль статусів, підтвердження отримання й аудит дій.

Для досягнення мети необхідно виконати такі завдання:

1. Проаналізувати предметну область об’єктового оповіщення та нормативні вимоги у сфері цивільного захисту.
2. Визначити акторів, ролі, сценарії використання, функціональні та нефункціональні вимоги до програмного прототипу.
3. Спроєктувати архітектуру, модель предметної області, модель даних, механізми доставки повідомлень, безпеки та аудиту.
4. Обґрунтувати вибір технологій реалізації, зокрема .NET, C#, ASP.NET Core, Entity Framework Core, SignalR та ASP.NET Core Identity.
5. Реалізувати програмний прототип об’єктової системи оповіщення.
6. Провести тестування та верифікацію реалізованого функціоналу відповідно до сформованих вимог.
7. Розглянути питання безпеки життєдіяльності та основ охорони праці, пов’язані з роботою оператора системи.

Об’єктом дослідження є процес організації об’єктового оповіщення керівників, працівників та інших осіб, які перебувають на території об’єкта або в його охоронних зонах під час загрози чи виникнення надзвичайної ситуації.

Предметом дослідження є програмні моделі, архітектурні рішення та засоби реалізації прототипу об’єктової системи оповіщення з використанням технологій .NET.

Методика роботи поєднує нормативний аналіз джерел, порівняння аналогів за ролями, зонами, каналами, підтвердженням, аудитом і способом розгортання,

об’єктно-орієнтоване та реляційне моделювання, а також трасування вимог до компонентів і доказів. Перевірка виконана за принципами тестування програмного забезпечення. Концепції надійності й безпечності використано для аналізу залишкових ризиків, але продуктивність, відмовостійкість, browser/e2e-сценарії та реальні канали доставки експериментально не перевірялися.

Практичне значення роботи полягає у створенні контрольованого навчального прототипу для підприємств, установ, навчальних закладів, чергових служб і відповідальних осіб із цивільного захисту. Прототип демонструє створення події, вибір зон, формування переліку отримувачів, підтвердження та аудит; його можна використати як основу для подальшого проєктування повноцінної системи.

Робота має прикладний інженерний характер; наукова новизна як окремий результат не заявляється.

У межах кваліфікаційної роботи реалізується саме програмний прототип. Він не замінює сертифіковану систему цивільного захисту, не керує реальними сиренами, гучномовцями, або спеціалізованим телекомунікаційним обладнанням.

Робота складається зі вступу, чотирьох розділів, висновків, списку джерел і додатків відповідно до методичних вказівок. У розділі 1 проаналізовано предметну область і сформовано вимоги; у розділі 2 подано проєктування та реалізацію програмної системи; у розділі 3 описано тестування, розгортання, підтримку й верифікацію; у розділі 4 розглянуто безпеку життєдіяльності та охорону праці. Додатки містять нормативну матрицю, діаграми, тест-кейси, фрагменти коду, демонстраційний сценарій та перелік електронних матеріалів.

1 АНАЛІЗ ВИМОГ ДО ПРОТОТИПУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У розділі визначено нормативні та предметні межі об’єктового оповіщення, проаналізовано акторів і наявні програмні підходи, після чого сформовано функціональні та нефункціональні вимоги системи. Результатом є узгоджена постановка задачі й критерії, за якими перевіряється реалізація [1].

1.1 Аналіз предметної області

Об’єктові системи оповіщення призначені для своєчасного інформування осіб, які перебувають на території підприємства, установи або іншого об’єкта з масовим перебуванням людей, про загрозу або виникнення надзвичайної ситуації. Відповідно до Постанови Кабінету Міністрів України №733, об’єктова система забезпечує доведення повідомлень до працівників та інших осіб у межах визначеного об’єкта, тоді як локальні системи використовуються на потенційно небезпечних об’єктах, де зона можливого ураження може поширюватися за межі їх території [2]. У межах даної роботи розглядається саме об’єктовий рівень оповіщення.

З точки зору програмного забезпечення процес оповіщення складається з декількох взаємопов’язаних етапів: створення події, визначення отримувачів, формування повідомлень, доставка повідомлень через доступні канали зв’язку, підтвердження отримання та реєстрація виконаних дій у журналі аудиту. Автоматизація цих процесів дозволяє зменшити час реагування, централізувати керування повідомленнями та забезпечити контроль результатів оповіщення.

У розробленому прототипі реалізовано створення подій оповіщення, погодження критичних подій, визначення адресатів за вибраними зонами, формування повідомлень, постановку спроб доставки в чергу, імітацію багатоканальної доставки, повторні спроби надсилання, підтвердження отримання повідомлень та аудит дій користувачів. Основними сутностями предметної області є подія оповіщення (AlertEvent), отримувач (Recipient), повідомлення

(AlertMessage), спроба доставки (DeliveryAttempt), зона (Zone) та журнал аудиту (AuditLog).

Поза межами прототипу залишаються автоматичне виявлення надзвичайних ситуацій за допомогою датчиків, фізичне керування сиренами та гучномовцями, інтеграція з територіальними системами цивільного захисту, а також використання реальних телекомунікаційних шлюзів.

На рисунку 1.1 наведено контекст та межі програмного прототипу об’єктової системи оповіщення.

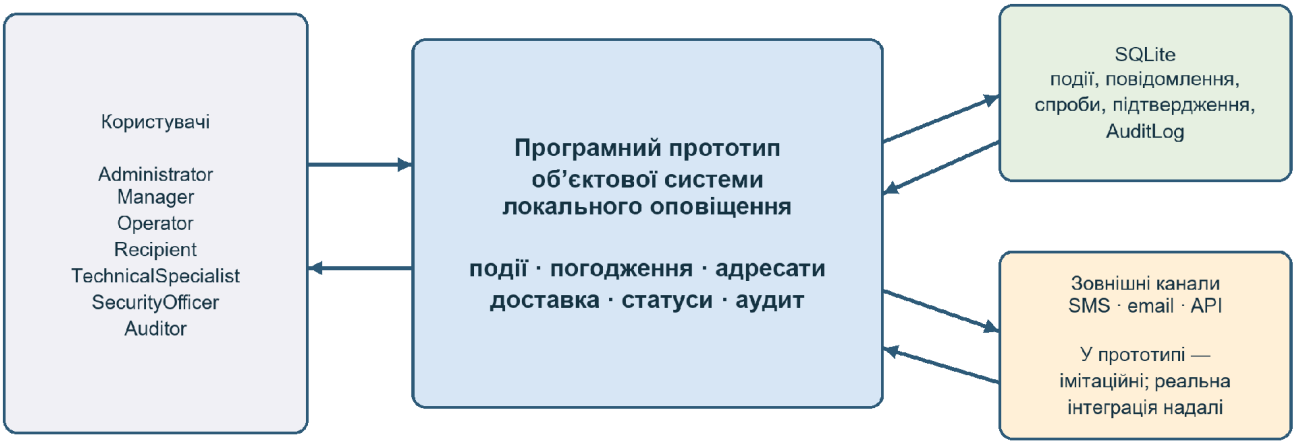

Рисунок 1.1 – Контекст і межі програмного прототипу об’єктової системи оповіщення

Для визначення функціональних меж програмного прототипу проведено аналіз нормативних вимог, які регламентують процес оповіщення та інформування осіб під час надзвичайних ситуацій. Результати аналізу наведено в таблиці 1.1. Розширену матрицю відповідності нормативних вимог і функцій прототипу наведено в додатку А.

Отже, нормативний аналіз дозволив визначити перелік функцій, які доцільно реалізувати у програмному прототипі. Найважливішими серед них є керування подіями оповіщення, підтримка ролей, вибір адресатів за зонами, контроль доставки повідомлень, підтвердження отримання та ведення журналу аудиту. Саме ці функції покладено в основу архітектури та програмної реалізації системи.

Таблиця 1.1 – Нормативні вимоги та їх відображення у прототипі

Вимога	Джерело	Відображення у прототипі
1	**2**	**3**
Своєчасне доведення інформації про НС	[1]	Створення події та запуск повідомлень
Оповіщення осіб на території об’єкта	[2]	Модель зон, отримувачів і груп
Обов’язкова інтеграція об’єктової системи з місцевою або територіальною АСЦО	[2]	Модель майбутньої API-інтеграції; у межах прототипу не є нормативно погодженим підключенням
Наявність технічних засобів оповіщення та інформування	[2]	Не реалізується; змодельовано програмними каналами, реальні засоби винесено в перспективи розвитку
Сценарне або вибіркове передавання	[3]	Шаблони повідомлень і правила вибору отримувачів
Підтвердження прийому	[3]	Сторінка підтвердження та статус повідомлення
Протоколювання дій	[3]	Журнал аудиту
Розмежування доступу	[3]	Ролі та політики авторизації
Доступність повідомлень	[2], [3]	Текстові статуси, короткі однозначні повідомлення, критична інформація не передається лише кольором

Для визначення функціональних можливостей майбутньої системи було проведено аналіз існуючих підходів до організації процесу оповіщення. Порівняння виконано за критеріями, які безпосередньо впливають на проєктування програмного забезпечення: підтримка ролей користувачів, зональна адресація повідомлень, механізми доставки, підтвердження отримання, аудит дій, контроль станів повідомлень та можливість локального розгортання.

Одним із найбільш поширених підходів є використання спеціалізованих платформ масового інформування, зокрема Everbridge Mass Notification, AlertMedia, Rave Alert та CodeRED [9–12]. Такі системи забезпечують багатоканальне надсилання повідомлень, централізоване адміністрування користувачів, моніторинг статусів доставки та формування звітності. Аналіз їх функціональних можливостей дозволив визначити низку вимог до

розроблюваного прототипу, зокрема підтримку ролей користувачів, контроль життєвого циклу повідомлень, аудит дій та можливість відстеження результатів оповіщення.

В Україні для інформування населення широко використовується застосунок «Повітряна тривога» [13], який забезпечує доведення повідомлень про небезпеку для вибраного регіону. Проте подібні рішення орієнтовані на масове інформування населення і не реалізують функцій внутрішнього об’єктового оповіщення. Зокрема, вони не підтримують модель ролей конкретної організації, не забезпечують зональну структуру об’єкта, не ведуть журнал дій оператора та не дозволяють контролювати підтвердження отримання повідомлень визначеними групами користувачів.

Найпростішим способом оповіщення залишається ручне інформування за допомогою телефонного зв’язку, месенджерів або усних повідомлень. Перевагою такого підходу є мінімальні вимоги до інфраструктури.

Порівняльний аналіз показав, що існуючі рішення реалізують окремі функції, необхідні для систем оповіщення, однак для навчального програмного прототипу доцільним є створення власної контрольованої моделі, яка дозволяє дослідити архітектурні та функціональні аспекти побудови об’єктової системи оповіщення. Результати аналізу наведено в таблиці 1.2.

Таблиця 1.2 – Порівняння існуючих підходів

Критерій	**Комерційні mass notification [9–12]**	**Регіональні мобільні сповіщення [13]**	**Ручне інформування**	**Навчальний прототип**
1	**2**	**3**	**4**	**5**
Ролі користувачів	Залежить від продукту; склад ролей у відкритій документації не уніфіковано	Об’єктові ролі у відкритій документації не заявлено	Неформальний розподіл	Сім ролей: Administrator, Manager, Operator, Recipient, TechnicalSpecialist, SecurityOfficer, Auditor

Продовження таблиці 1.2

Критерій	Комерційні mass notification [9–12]	Регіональні мобільні сповіщення [13]	Ручне інформування	Навчальний прототип
1	2	3	4	5
Зони оповіщення	Підтримуються	Переважно географічні	За списками	Зони об’єкта; групи зберігаються, але не використовуються RecipientResolver
Підтвердження	Залежить від продукту; можливості наведено в документації [9–12]	Підтвердження конкретним адресатом у відкритій документації не заявлено [13]	Усне або ручне	Автентифіковане підтвердження лише власного Delivered-повідомлення
Аудит	Звітність заявлена; деталізація залежить від продукту [9–12]	Обмежений для користувача	Неповне документування	Append-only контроль AuditLog на рівні DbContext
Багатоканальність	Реальні SMS/email/push	Мобільний канал	Телефон, месенджери, усно	Email/SMS та внутрішній канал імітуються
Локальне розгортання	Залежить від ліцензії	Ні	Так	Так, Windows/.NET/SQLite
Залежність від провайдера	Висока або середня	Висока	Низька	Низька в демо; реальні канали потребуватимуть провайдера
Придатність для навчального прототипу	Низька через складність і вартість	Низька через закриту інфраструктуру	Обмежена через відсутність трасування	Висока: контрольована модель без заяви про перевагу над промисловими системами

1.2 Постановка завдання та цілей

На основі проведеного аналізу предметної області та існуючих підходів до організації процесу оповіщення сформульовано мету та завдання кваліфікаційної роботи.

Метою роботи є розробка програмного прототипу об’єктової системи оповіщення, який забезпечує керування подіями оповіщення, розподіл повідомлень між отримувачами, та аудит виконаних дій.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) провести аналіз предметної області та існуючих програмних рішень;
2) визначити функціональні та нефункціональні вимоги до системи;
3) спроєктувати архітектуру програмного забезпечення;
4) розробити модель даних для зберігання подій, повідомлень, отримувачів та журналу аудиту;
5) реалізувати механізми автентифікації, авторизації та розмежування доступу;
6) реалізувати процес створення, погодження та запуску подій оповіщення;
7) реалізувати механізм визначення адресатів за зонами та формування персоналізованих повідомлень;
8) реалізувати підсистему доставки повідомлень із підтримкою повторних спроб надсилання;
9) забезпечити підтвердження отримання повідомлень користувачами;
10) реалізувати журнал аудиту дій користувачів;
11) провести тестування реалізованого програмного забезпечення.

Розроблювана система повинна забезпечувати підтримку основних сценаріїв роботи з подіями оповіщення, зокрема створення чернетки події, подання її на погодження, запуск розсилання повідомлень, визначення отримувачів за зонами, контроль доставки повідомлень та підтвердження їх отримання. Також система повинна забезпечувати розмежування доступу між ролями користувачів, ведення журналу аудиту та відображення актуального стану подій і повідомлень.

Для реалізації зазначених функцій планується використання платформи ASP.NET Core, механізмів автентифікації та авторизації, засобів фонового виконання задач, а також реляційної бази даних для зберігання інформації про події, повідомлення, спроби доставки та результати їх виконання.

Сформульовані вимоги до системи поділено на функціональні та нефункціональні. Такий підхід відповідає принципам інженерії вимог та забезпечує можливість окремого оцінювання реалізованої функціональності та якісних характеристик програмного забезпечення.

1.3 Вимоги до системи

На основі аналізу предметної області, нормативних вимог та існуючих програмних рішень сформовано функціональні та нефункціональні вимоги до програмного прототипу об’єктової системи оповіщення. Вимоги визначають межі функціональності системи та слугують основою для подальшого проєктування архітектури, моделі даних і компонентів програмного забезпечення.

Таблиця 1.3 – Функціональні вимоги до прототипу

Ідентифікатор	**Функціональна вимога**
1	**2**
FR-01	Система повинна забезпечувати автентифікацію користувачів та авторизацію на основі ролей і політик доступу.
FR-02	Система повинна дозволяти створення чернетки події оповіщення із збереженням усіх необхідних атрибутів події.
FR-03	Система повинна підтримувати подання події на погодження та зміну її стану відповідно до визначеної бізнес-логіки.
FR-04	Система повинна забезпечувати погодження або відхилення критичних подій користувачем із відповідними повноваженнями.
FR-05	Система повинна вимагати введення причини відхилення події під час виконання операції відхилення.
FR-06	Система повинна дозволяти запуск лише погоджених подій оповіщення.
FR-07	Система повинна визначати отримувачів повідомлень на основі вибраних зон оповіщення.
FR-08	Система повинна виключати дублювання адресатів під час формування списку отримувачів.
FR-09	Система повинна враховувати лише активних отримувачів під час формування списку адресатів.
FR-10	Система повинна формувати персоналізовані повідомлення для кожного отримувача.
FR-11	Система повинна створювати окрему сутність доставки для кожного повідомлення та отримувача.

Продовження таблиці 1.3

Ідентифікатор	Функціональна вимога
1	**2**
FR-12	Система повинна виконувати доставку повідомлень у фоновому режимі без блокування основного запиту користувача.
FR-13	Система повинна підтримувати повторні спроби доставки повідомлень у разі виникнення помилки.
FR-14	Система повинна переводити невдалі доставки у стан DeadLetter після досягнення встановленого ліміту повторних спроб.
FR-15	Система повинна відображати актуальні статуси подій та повідомлень користувачам.
FR-16	Система повинна дозволяти підтвердження лише власних доставлених повідомлень отримувачем.
FR-17	Система повинна вести журнал аудиту критичних дій користувачів.
FR-18	Система повинна забезпечувати перегляд довідкової інформації, ролей та демонстраційних даних.
FR-19	Система повинна підтримувати завершення або скасування події відповідно до допустимих переходів між станами.
FR-20	Система повинна блокувати виконання операцій, які порушують правила життєвого циклу події.

Таблиця 1.4 Нефункціональні вимоги до прототипу

Ідентифікатор	Нефункціональна вимога
1	**2**
NFR-1	Система повинна забезпечувати безпечне зберігання облікових даних користувачів та контроль доступу на основі ролей.
NFR-2	Система повинна гарантувати цілісність бізнес-логіки та контроль допустимих переходів між станами подій.
NFR-3	Система повинна забезпечувати надійність доставки повідомлень шляхом використання механізму повторних спроб надсилання.
NFR-4	Система повинна підтримувати фонову обробку задач доставки повідомлень без блокування інтерфейсу користувача.
NFR-5	Система повинна забезпечувати актуальність інформації про статуси подій, повідомлень, спроб доставки та підтверджень.
NFR-6	Система повинна гарантувати незмінність записів журналу аудиту після їх створення.
NFR-7	Система повинна забезпечувати коректне визначення адресатів оповіщення без дублювання записів.

Продовження таблиці 1.4

Ідентифікатор	Нефункціональна вимога
1	**2**
NFR-8	Система повинна забезпечувати можливість підтвердження лише власних доставлених повідомлень користувача.
NFR-9	Архітектура програмного забезпечення повинна підтримувати розділення відповідальностей між компонентами системи.
NFR-10	Структура програмного забезпечення повинна забезпечувати можливість подальшого розширення функціональності та інтеграції нових каналів доставки повідомлень.
NFR-11	Система повинна бути придатною до модульного та інтеграційного тестування.
NFR-12	Програмне забезпечення повинно підтримувати локальне розгортання в середовищі Windows із використанням платформи .NET та СУБД SQLite.

Сформульовані вимоги визначають функціональні межі програмного прототипу та є основою для побудови моделі варіантів використання, проєктування архітектури програмного забезпечення та реалізації компонентів системи.

1.4 Пошук акторів та варіантів використання

На основі аналізу предметної області та сформульованих функціональних вимог визначено основних користувачів програмної системи. Актором системи вважається користувач або роль, що взаємодіє з програмним забезпеченням для виконання визначених функцій. Визначення акторів дозволяє встановити межі відповідальності користувачів та сформувати рольову модель доступу до функцій системи.

У розроблюваному прототипі реалізовано рольовий підхід до керування доступом (RBAC), за якого права користувачів визначаються їх належністю до певної ролі. Такий підхід забезпечує контроль доступу до функцій системи, зменшує ризик несанкціонованого виконання операцій.

Для програмного прототипу визначено ролі користувачів, наведені в зведеній таблиці 1.4.

Таблиця 1.4 – Основні актори системи

Роль	**Ідентифікатор ролі**	**Призначення**	**Основні дії**
1	**2**	**3**	**4**
Адміністратор	Administrator	Загальний доступ до системи	Перегляд довідників, dashboard та журналу аудиту
Керівник	Manager	Контроль критичних подій	Погодження або відхилення Critical-подій
Оператор	Operator	Виконання основних операцій оповіщення	Створення, погодження, запуск, завершення або скасування подій
Отримувач	Recipient	Отримання повідомлень	Підтвердження отримання власних повідомлень
Технічний фахівець	TechnicalSpecialist	Моніторинг роботи системи	Перегляд статусів доставки та dashboard
Відповідальний за безпеку	SecurityOfficer	Контроль виконання процедур оповіщення	Перегляд журналу аудиту та статусів системи
Аудитор	Auditor	Перевірка дій користувачів	Перегляд журналу аудиту та dashboard

Серед визначених акторів найбільш активним користувачем системи є оператор, який виконує основні операції керування подіями оповіщення. Керівник бере участь у процесі погодження критичних подій, отримувач забезпечує завершення процесу оповіщення шляхом підтвердження отримання повідомлення, а контрольні ролі здійснюють моніторинг та аудит роботи системи.

Визначені актори формують основу рольової моделі програмного забезпечення та використовуються під час побудови діаграми варіантів використання, розглянутої в наступному підрозділі.

1.5 Опис ключових варіантів використання

На основі визначених акторів та сформульованих вимог побудовано модель варіантів використання системи. Вона відображає взаємодію користувачів із

програмним забезпеченням під час виконання основних операцій оповіщення та дозволяє формалізувати функціональність системи з точки зору кінцевих користувачів.

Діаграма варіантів використання системи наведена на рисунку 1.2.

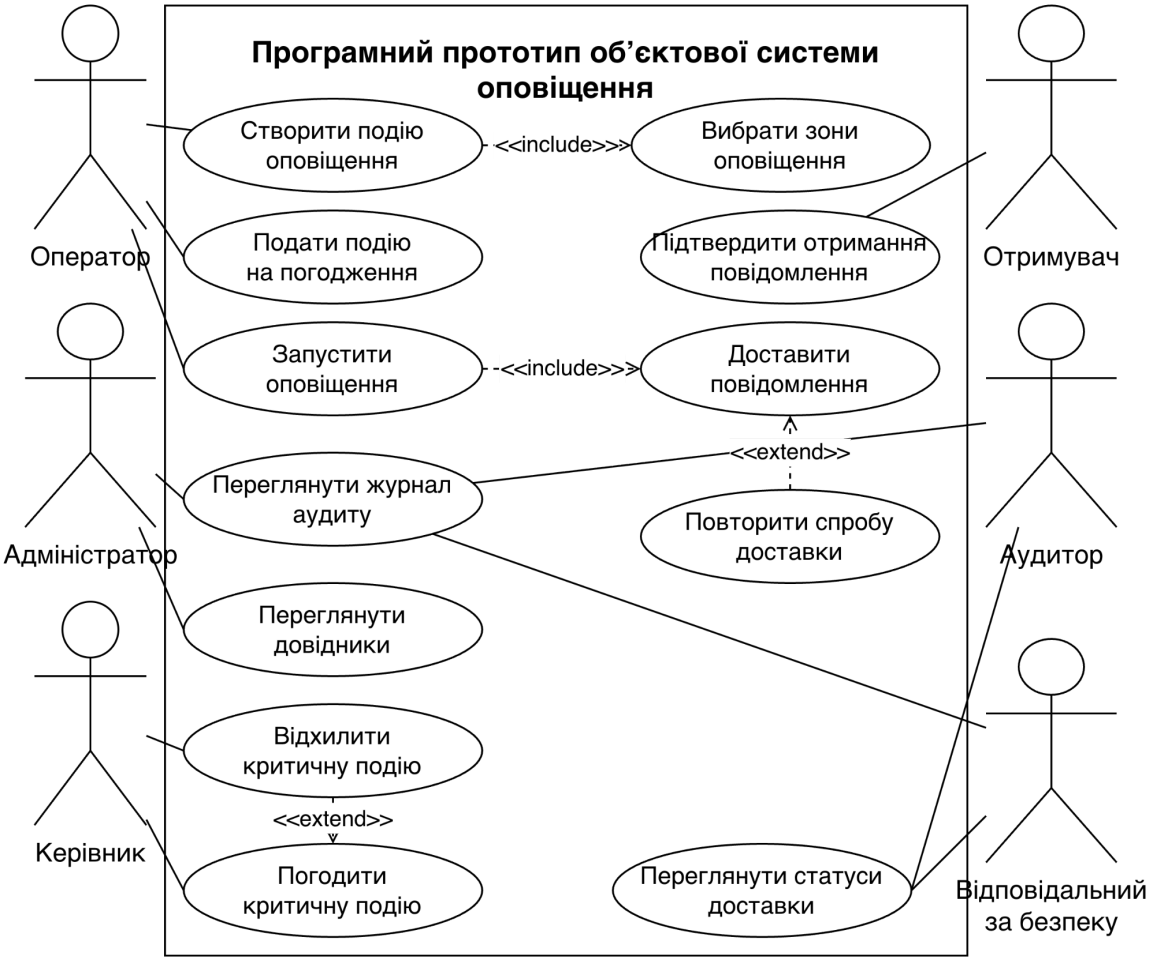

Рисунок 1.2 – Діаграма варіантів використання системи

Основним сценарієм роботи є створення та запуск події оповіщення. Оператор створює нову подію, визначає зони оповіщення та подає її на погодження. Для подій критичного рівня керівник виконує погодження або відхилення. Після погодження система визначає адресатів, формує повідомлення та запускає процес доставки. Отримувач після успішного надсилання повідомлення може підтвердити його отримання. Усі критичні дії користувачів фіксуються в журналі аудиту.

До альтернативних сценаріїв належать відхилення критичної події керівником, повторні спроби доставки повідомлень у разі помилки, блокування недопустимих переходів між станами події, а також відхилення спроб підтвердження повідомлень сторонніми користувачами. У випадку перевищення допустимої кількості повторних спроб доставка переводиться у стан DeadLetter.

Деталізовану діаграму варіантів використання фактичного прототипу наведено в додатку Б.

Використання моделі варіантів використання дозволило формалізувати основні сценарії взаємодії користувачів із системою, визначити межі функціональності програмного забезпечення та підготувати основу для подальшого проєктування архітектури і компонентів системи.

У результаті виконання першого розділу визначено вимоги до програмного прототипу об’єктової системи оповіщення, встановлено його функціональні межі та основні сценарії використання. Проведений аналіз дозволив сформувати рольову модель доступу, визначити ключові функції керування подіями оповіщення, доставки повідомлень, підтвердження отримання та аудиту дій користувачів. На основі отриманих результатів виконуватиметься проєктування архітектури програмного забезпечення, структури бази даних і програмних компонентів системи, що забезпечують реалізацію визначених вимог.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОТОТИПУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У розділі подано проєктні та реалізаційні рішення, використані під час створення програмного прототипу об’єктової системи локального оповіщення при надзвичайних ситуаціях. Розділ послідовно охоплює вибір процесу розробки, архітектуру системи, модель даних, UML-модель класів, технологічний стек, реалізацію основних класів і методів, а також побудову інтерфейсу користувача.

Проєктований продукт є навчально-демонстраційним програмним прототипом. Він моделює програмну підтримку об’єктового оповіщення: створення події, вибір зон, визначення отримувачів, формування повідомлень, постановку спроб доставки в чергу, імітаційну доставку, контроль статусів, підтвердження отримання та аудит дій. Прототип не керує реальними сиренами, гучномовцями, світловими табло або спеціалізованим телекомунікаційним обладнанням і не заявляється як сертифікована система цивільного захисту.

2.1 Вибір процесу розробки

Для розробки програмного прототипу об’єктової системи оповіщення обрано ітераційно-інкрементальний процес розробки. Даний підхід передбачає поступове створення програмного забезпечення шляхом реалізації окремих функціональних інкрементів, кожен з яких розширює можливості системи та формує завершений етап її розвитку.

Вибір такого процесу зумовлений особливостями розроблюваної системи. Прототип охоплює декілька взаємопов’язаних підсистем, серед яких керування подіями оповіщення, рольовий доступ, визначення отримувачів, доставка повідомлень, підтвердження отримання та аудит дій користувачів. Реалізація зазначених компонентів потребувала поступового уточнення архітектурних рішень, структури даних та бізнес-правил, що робить використання каскадної моделі менш доцільним.

На початковому етапі було виконано аналіз предметної області, визначено межі програмного прототипу та сформовано функціональні й нефункціональні вимоги до системи. На основі отриманих результатів було розроблено архітектуру програмного забезпечення, визначено основні програмні компоненти та спроєктовано модель даних.

Подальша розробка виконувалася у вигляді послідовних функціональних інкрементів. Перший інкремент забезпечив реалізацію автентифікації користувачів, рольової моделі доступу та початкових довідкових даних. Другий інкремент охопив створення подій оповіщення, вибір зон та визначення адресатів повідомлень. У межах третього інкремента реалізовано керування життєвим циклом події, включаючи подання на погодження, погодження, відхилення, запуск та завершення оповіщення. Четвертий інкремент був присвячений реалізації підсистеми доставки повідомлень, механізму повторних спроб надсилання, підтвердження отримання повідомлень користувачами та ведення журналу аудиту.

Після реалізації кожного інкремента виконувалося тестування відповідних компонентів системи та перевірка коректності їх взаємодії. Такий підхід дозволив своєчасно виявляти помилки, поступово розширювати функціональність системи та контролювати відповідність реалізації сформульованим вимогам.

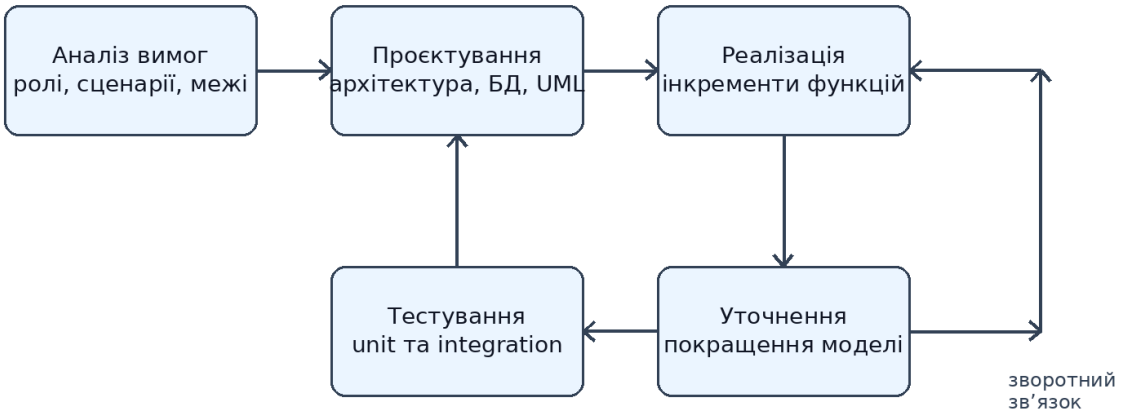

Рисунок 2.1 – Схема ітераційно-інкрементального процесу розробки прототипу

Отже, обраний процес розробки забезпечив поступовий перехід від вимог до працездатного програмного прототипу. Він відповідає навчально-демонстраційному характеру роботи, дає змогу зменшити ризик неузгодженості між компонентами та підтримує поетапне уточнення технічних рішень.

2.2 Проектування архітектури системи

Архітектура програмного прототипу реалізована у вигляді багатошарового клієнт-серверного вебзастосунку з використанням підходу Modular Monolith. Такий архітектурний стиль забезпечує логічне відокремлення функціональних компонентів системи при збереженні простоти розгортання та супроводу. На відміну від мікросервісної архітектури, модульний моноліт не потребує окремої інфраструктури для міжсервісної взаємодії, що є доцільним для навчального програмного прототипу.

Програмне рішення складається з окремих проєктів:

1) LocalAlerting.Domain – доменна модель та бізнес-правила системи;
2) LocalAlerting.Application – прикладна логіка та сценарії використання;
3) LocalAlerting.Infrastructure – доступ до даних та інтеграція із зовнішніми компонентами;
4) LocalAlerting.Web – вебінтерфейс користувача та HTTP-взаємодія;
5) LocalAlerting.Worker – фонові процеси обробки доставки повідомлень;
6) LocalAlerting.UnitTests – модульне тестування;
7) LocalAlerting.IntegrationTests – інтеграційне тестування.

Архітектура побудована відповідно до принципів розділення відповідальностей (Separation of Concerns) та інверсії залежностей (Dependency Inversion Principle). Залежності між компонентами спрямовані від зовнішніх шарів до внутрішніх, що дозволяє ізолювати бізнес-логіку від технологічних деталей реалізації.

Рівень Domain містить основні доменні сутності системи, серед яких AlertEvent, AlertMessage, DeliveryAttempt, Recipient, Zone та AuditLog. У цьому шарі реалізовано правила життєвого циклу події оповіщення, перевірку допустимих переходів між станами та інші бізнес-обмеження. Доменний рівень не містить залежностей від бази даних, вебтехнологій або механізмів доставки повідомлень.

Рівень Application реалізує прикладні сценарії роботи системи. У ньому розміщено сервіси керування подіями оповіщення, механізми визначення адресатів, DTO-об’єкти та контракти взаємодії між компонентами. Одним із ключових компонентів цього рівня є RecipientResolver, який виконує вибір отримувачів за зонами оповіщення та усуває дублювання адресатів.

Рівень Infrastructure відповідає за реалізацію технічних механізмів системи. До його складу входять компоненти доступу до бази даних на основі Entity Framework Core, реалізація ASP.NET Core Identity, механізми аудиту, імітаційні канали доставки повідомлень та засоби збереження початкових даних. Саме цей шар забезпечує взаємодію доменної моделі з SQLite та іншими інфраструктурними сервісами.

Рівень Web реалізує користувацький інтерфейс, маршрутизацію запитів, механізми автентифікації та авторизації, а також відображення стану системи. Для оперативного оновлення інформації використовується технологія SignalR, яка забезпечує передавання змін статусів подій та повідомлень у режимі реального часу.

Проєкт Worker реалізує окремий процес фонової обробки доставки повідомлень. Він виконує вибір підготовлених DeliveryAttempt зі статусом Queued, обробляє надсилання повідомлень, змінює статуси доставки та виконує повторні спроби у випадку виникнення помилок. У демонстраційній конфігурації використовується один активний dispatcher, що виключає конкуренцію між кількома процесами під час локального запуску системи.

Основний потік взаємодії між компонентами починається з виконання дії користувачем у вебінтерфейсі. Вебрівень передає запит до прикладного сервісу,

який реалізує відповідний сценарій використання. Прикладний сервіс викликає доменну логіку, а інфраструктурний шар забезпечує збереження змін у базі даних. Під час запуску оповіщення створюються повідомлення та спроби доставки, які обробляються фоновим сервісом Worker. Результати доставки зберігаються в базі даних та передаються до вебінтерфейсу через SignalR.

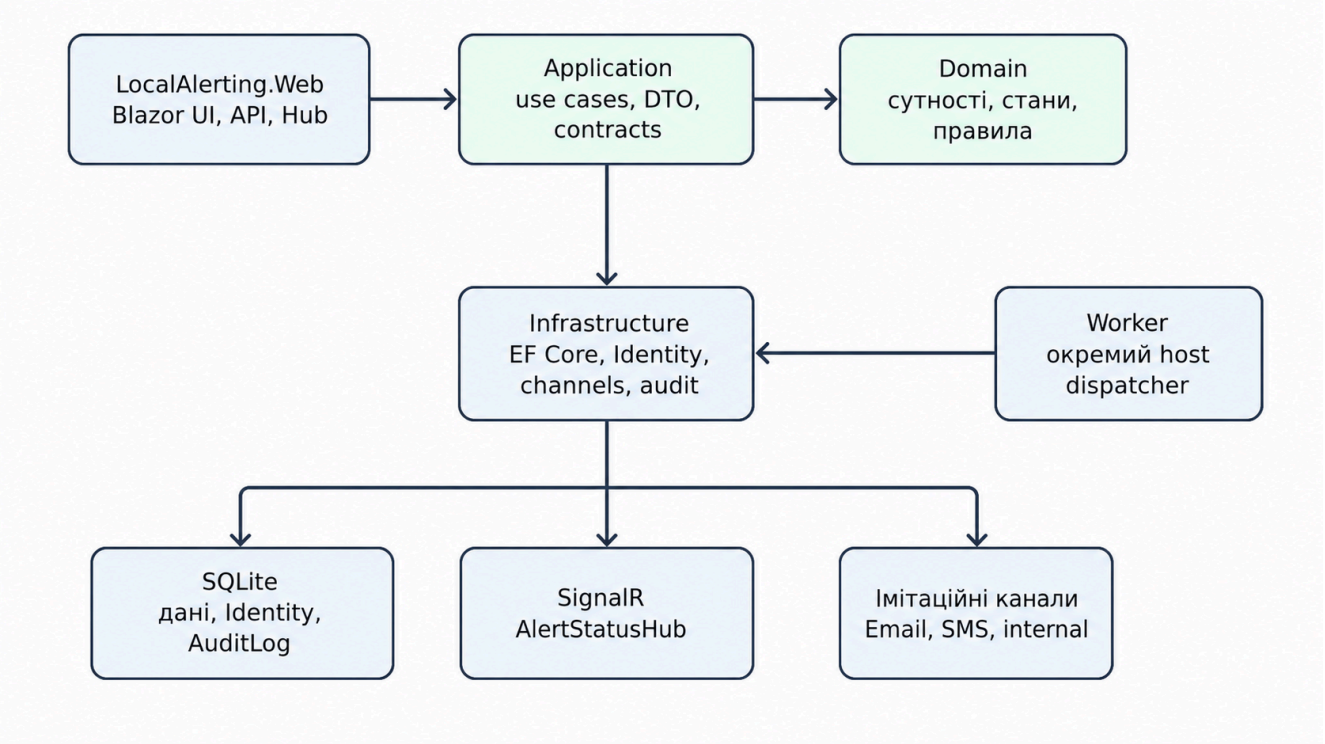

Рисунок 2.2 – Архітектурна схема програмного прототипу системи оповіщення

Деталізовану компонентну архітектуру програмного прототипу наведено в додатку Б.

Для забезпечення безпеки використано механізми ASP.NET Core Identity, рольову модель доступу та політики авторизації. У системі реалізовано ролі Administrator, Manager, Operator, Recipient, TechnicalSpecialist, SecurityOfficer та Auditor. Доступ до критичних функцій контролюється за допомогою політик CanLaunchAlerts, CanApproveAlerts, CanViewAlertDashboard, CanViewAudit та CanManageDictionaries. Такий підхід забезпечує централізоване керування

правами доступу та унеможливлює виконання критичних операцій користувачами без відповідних повноважень.

Обрана архітектура забезпечує незалежність бізнес-логіки від інфраструктурних компонентів, підтримує розширення функціональності системи та дозволяє замінювати окремі технічні реалізації без зміни доменної моделі. Зокрема, імітаційні канали доставки можуть бути замінені реальними SMS або email-провайдерами без модифікації основних бізнес-правил системи.

2.3 Побудова схем бази даних

Для забезпечення збереження даних програмного прототипу спроєктовано реляційну базу даних, яка охоплює всі основні сутності процесу оповіщення та зв’язки між ними. Як систему керування базами даних обрано SQLite, що відповідає локальному характеру розгортання програмного прототипу та не потребує встановлення окремого серверного програмного забезпечення. Взаємодія з базою даних реалізована за допомогою Entity Framework Core із використанням підходу Code First.

Модель даних побудована відповідно до доменної структури системи та складається з декількох логічних підсистем.

Перша підсистема відповідає за зберігання інформації про об’єкт оповіщення та його отримувачів. До неї належать сутності FacilityObject, Zone, Recipient, ContactPoint, RecipientGroup та RecipientGroupMembership. Сутність FacilityObject описує об’єкт, у межах якого здійснюється оповіщення, а сутність Zone використовується для поділу об’єкта на окремі функціональні або територіальні зони. Сутність Recipient містить інформацію про отримувачів повідомлень та може бути пов’язана з обліковим записом користувача через ApplicationUserId. Для зберігання контактних даних використовується сутність ContactPoint, яка визначає тип каналу зв’язку та відповідну адресу доставки повідомлень. Сутності RecipientGroup і RecipientGroupMembership забезпечують

можливість групування отримувачів, однак у поточній реалізації механізм визначення адресатів базується на зональному принципі.

Друга підсистема призначена для опису типів надзвичайних ситуацій та шаблонів повідомлень. Для цього використовуються сутності EmergencyType та AlertTemplate. EmergencyType визначає категорію надзвичайної ситуації та рекомендований рівень критичності повідомлення. AlertTemplate використовується для автоматичного формування заголовка та тексту повідомлення на основі вибраного типу події.

Основу моделі даних становить підсистема керування процесом оповіщення. Центральною сутністю є AlertEvent, яка зберігає інформацію про подію оповіщення, її рівень критичності, поточний стан та часові характеристики життєвого циклу. Для кожного адресата формується окремий запис AlertMessage, що дозволяє відстежувати результати доставки на рівні конкретного отримувача. Сутність DeliveryAttempt використовується для реєстрації окремих технічних спроб надсилання повідомлення через визначений канал зв’язку. Такий підхід забезпечує можливість реалізації механізму повторних спроб доставки та детального аналізу причин помилок. Факт підтвердження отримання повідомлення зберігається в сутності Acknowledgement.

Для забезпечення простежуваності роботи системи використовується сутність AuditLog. Вона призначена для реєстрації критичних дій користувачів та системних подій, зокрема створення, погодження, запуску, завершення або скасування подій оповіщення, результатів доставки повідомлень та підтверджень їх отримання. На рівні прикладної логіки реалізовано механізм, який забороняє зміну або видалення вже створених записів аудиту, що забезпечує збереження історії виконаних операцій.

Під час проєктування схеми бази даних особливу увагу приділено підтримці зв’язків між сутностями та забезпеченню цілісності даних. Спроєктована структура дозволяє простежити повний життєвий цикл повідомлення — від створення події оповіщення та вибору зони до доставки повідомлення

конкретному отримувачу, фіксації результатів доставки та підтвердження отримання.

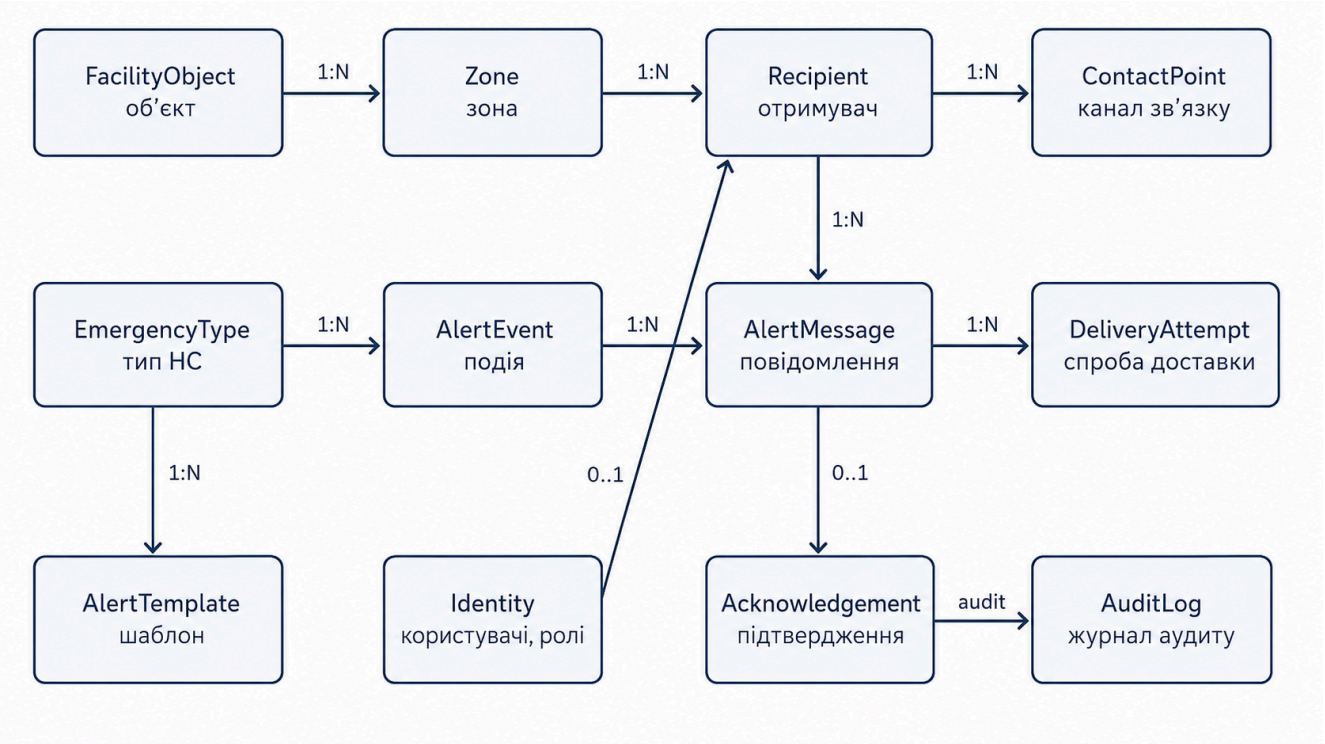

Рисунок 2.3 – ER-діаграма основних сутностей бази даних прототипу

Повну ER-діаграму моделі даних наведено в додатку В.

Отже, спроєктована модель даних забезпечує зберігання інформації про всі етапи процесу оповіщення, підтримує реалізацію основних бізнес-сценаріїв системи та створює основу для подальшого розширення функціональності програмного забезпечення.

2.4 Побудова UML-діаграм класів

Для відображення статичної структури програмного забезпечення побудовано UML-діаграму класів, яка описує основні сутності доменної моделі, їх атрибути, операції та зв'язки між ними. На відміну від ER-діаграми бази даних, UML-діаграма орієнтована на об'єктну структуру програмного забезпечення та

використовується для проєктування взаємодії компонентів на рівні програмного коду.

Основу доменної моделі становлять класи AlertEvent, AlertMessage, DeliveryAttempt, Recipient, Zone, ContactPoint, EmergencyType, AlertTemplate, Acknowledgement та AuditLog. Саме ці класи реалізують основні бізнес-сценарії системи оповіщення та відповідають за керування життєвим циклом подій, повідомлень і результатів їх доставки.

Центральним елементом моделі є клас AlertEvent, який представляє подію оповіщення та містить інформацію про її тип, рівень критичності, поточний стан і часові характеристики життєвого циклу. У межах класу реалізовано операції, що забезпечують керування станами події, зокрема подання на погодження, погодження, відхилення, запуск, завершення та скасування. Реалізація цих операцій дозволяє контролювати допустимі переходи між станами та забезпечує цілісність бізнес-логіки.

Клас AlertMessage представляє персоналізоване повідомлення, сформоване для конкретного отримувача. Він пов’язаний із подією оповіщення, адресатом та зоною, у межах якої виконується оповіщення. Для кожного повідомлення підтримується окремий життєвий цикл доставки та підтвердження отримання.

Технічний процес надсилання повідомлень реалізується за допомогою класу DeliveryAttempt. Кожний його екземпляр відповідає окремій спробі доставки повідомлення через визначений канал зв’язку. Наявність окремої сутності для спроб доставки дозволяє реалізувати механізм повторних спроб надсилання, відстежувати причини помилок та аналізувати результати роботи каналів зв’язку.

Адресна модель системи представлена класами Recipient, Zone та ContactPoint. Клас Recipient описує отримувача повідомлень, Zone використовується для зонального розподілу адресатів, а ContactPoint зберігає контактні дані та параметри каналу доставки. Такий підхід дозволяє відокремити логіку визначення отримувачів від механізму надсилання повідомлень.

Для підтримки шаблонів повідомлень і класифікації надзвичайних ситуацій використовуються класи EmergencyType та AlertTemplate. Вони забезпечують стандартизацію повідомлень та спрощують створення нових подій оповіщення.

Контроль підтверджень отримання повідомлень реалізується класом Acknowledgement, який зберігає інформацію про факт підтвердження та пов’язує його з відповідним повідомленням і користувачем. Це дозволяє забезпечити перевірку отримання повідомлень адресатами.

Окремим елементом доменної моделі є клас AuditLog, призначений для фіксації критичних дій користувачів та системних подій. Він забезпечує простежуваність роботи системи та підтримує реалізацію механізму аудиту.

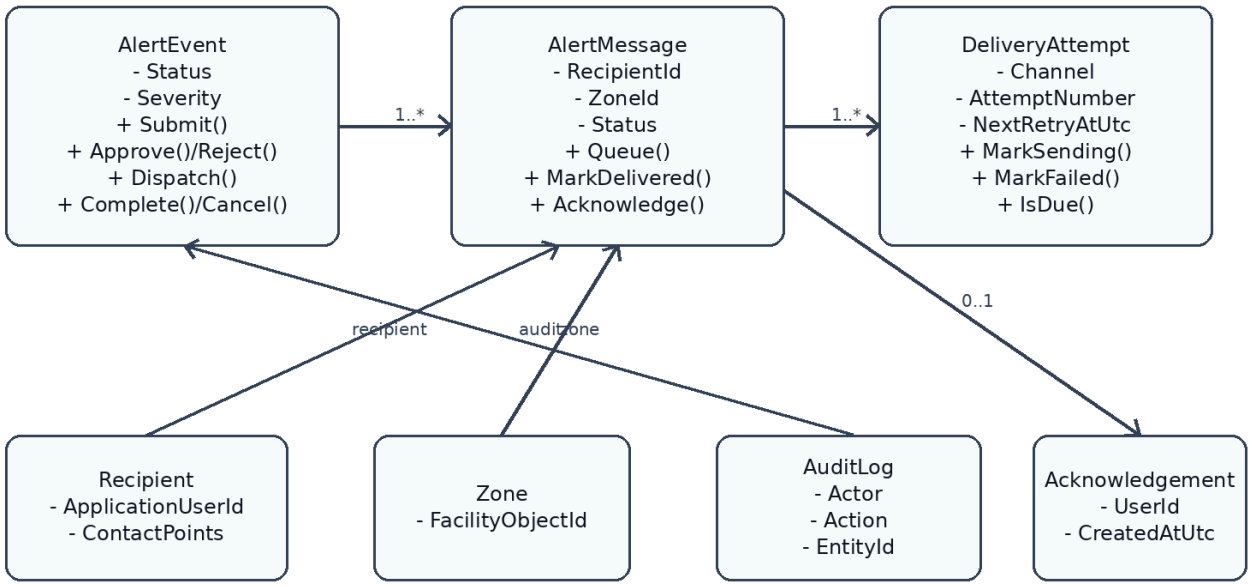

Рисунок 2.4 – UML-діаграма основних класів доменної моделі

Деталізовану UML-діаграму класів програмного прототипу наведено в додатку Б.

Побудована UML-модель відображає структуру доменної області та взаємозв’язки між основними сутностями системи. Її використання дозволило формалізувати бізнес-логіку програмного забезпечення, спростити проєктування компонентів системи та забезпечити узгодженість між архітектурою програмного забезпечення, моделлю даних і програмною реалізацією.

2.5 Вибір мови та середовища розробки

Вибір технологій реалізації здійснювався з урахуванням функціональних та нефункціональних вимог до програмного прототипу. Основними критеріями були підтримка веборієнтованої архітектури, наявність засобів автентифікації та авторизації, можливість локального розгортання, підтримка реляційних баз даних, реалізація фонової обробки задач, оновлення даних у режимі реального часу та можливість автоматизованого тестування [14–27].

2.5.1 Критерії вибору технологій

Під час вибору технологічного стеку враховувалися такі критерії:

1) підтримка об’єктно-орієнтованого програмування та статичної типізації;
2) можливість побудови багатошарової архітектури;
3) підтримка механізмів автентифікації та авторизації;
4) наявність ORM-засобів для роботи з реляційними базами даних;
5) підтримка асинхронної та фонової обробки задач;
6) можливість локального розгортання без додаткової серверної інфраструктури;
7) підтримка автоматизованого тестування;
8) можливість подальшого розширення та інтеграції із зовнішніми сервісами.

2.5.2 Обґрунтування технологічного стеку

Для реалізації програмного прототипу обрано платформу .NET 10 та мову програмування C#. Використання C# забезпечує підтримку сучасних засобів об’єктно-орієнтованого програмування, статичної перевірки типів та побудови складних доменних моделей.

Основою серверної частини системи є ASP.NET Core, який забезпечує маршрутизацію запитів, конфігурацію middleware-компонентів, механізми автентифікації та авторизації, а також підтримку Dependency Injection. Для реалізації користувацького інтерфейсу використано технологію Blazor Web App із

режимом Server Interactivity, що дозволяє створювати інтерактивний вебінтерфейс із використанням мови C# без необхідності розробки окремого JavaScript-клієнта (рис. 2.5).

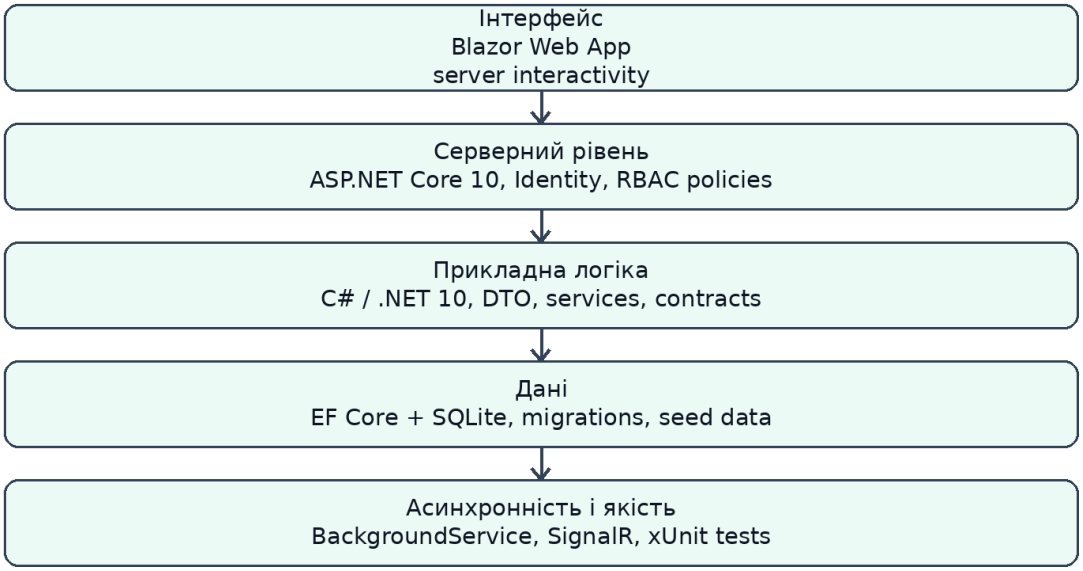

Рисунок 2.5 – Технологічний стек програмного прототипу

Для реалізації механізмів автентифікації та авторизації використано ASP.NET Core Identity. Даний компонент забезпечує керування обліковими записами користувачів, ролями та політиками доступу. Для контролю доступу до функцій системи застосовано політики авторизації CanLaunchAlerts, CanApproveAlerts, CanViewAlertDashboard, CanViewAudit та CanManageDictionaries.

Збереження даних реалізовано за допомогою Entity Framework Core та SQLite. Entity Framework Core забезпечує об’єктно-реляційне відображення доменної моделі на структуру бази даних та підтримує міграції схеми даних. Використання SQLite дозволяє виконувати локальне розгортання системи без встановлення окремого серверного середовища.

Для оновлення статусів подій та повідомлень у режимі реального часу використано SignalR. Дана технологія реалізує двосторонню взаємодію між

сервером та клієнтом і дозволяє передавати зміни без необхідності періодичного опитування сервера.

Фонова обробка доставки повідомлень реалізована за допомогою BackgroundService. Використання окремого фонового сервісу дозволяє виконувати доставку повідомлень незалежно від HTTP-запитів користувачів та забезпечує підтримку повторних спроб надсилання.

Для тестування програмного забезпечення використано фреймворк xUnit, який забезпечує реалізацію модульних та інтеграційних тестів основних компонентів системи.

2.5.3 Реєстр архітектурних рішень

Під час розробки програмного прототипу було прийнято низку архітектурних рішень, що визначають структуру та принципи функціонування системи.

Для побудови програмного забезпечення обрано архітектурний стиль Modular Monolith, який забезпечує логічне розділення компонентів системи при збереженні простоти локального розгортання. Як систему керування базами даних використано SQLite, оскільки вона не потребує окремого серверного середовища та є достатньою для навчального прототипу.

Для оновлення інформації в режимі реального часу використано SignalR замість періодичного опитування сервера, що дозволило зменшити кількість запитів та підвищити оперативність відображення статусів. Реалізацію доставки повідомлень виконано за допомогою BackgroundService без використання брокерів повідомлень, оскільки для локального середовища така архітектура є достатньою та менш складною в супроводі.

Для керування доступом обрано ASP.NET Core Identity та політики авторизації замість власної реалізації механізмів безпеки. Це дозволило використати перевірені механізми автентифікації та зменшити ризик помилок у реалізації критичних функцій безпеки.

Обраний технологічний стек забезпечує реалізацію всіх функціональних вимог програмного прототипу, підтримує подальше розширення системи та відповідає архітектурним рішенням, прийнятим під час проєктування програмного забезпечення.

Окремим напрямом подальшого розвитку може бути інтелектуальна підтримка підготовки повідомлень та довідкових даних. Сучасні дослідження трансформерних нейронних мереж і LLM-підходів показують можливість видобування структурованої інформації з неструктурованих текстів, зокрема службових повідомлень, анкетних даних або інших документів [28–31]. Для систем, що працюють із міжсистемним обміном даними та оцінюванням якості інформації, також важливими є питання інтероперабельності й контролю якості даних в AI-пайплайнах [32]. У межах поточного прототипу такі механізми не реалізовано, однак архітектурне розділення компонентів залишає можливість їх подальшого підключення без зміни основної доменної моделі.

2.6 Реалізація основних класів та методів

Реалізація програмного прототипу виконана відповідно до спроєктованої багатошарової архітектури та передбачає розподіл функціональності між доменними сутностями, прикладними сервісами, компонентами доставки повідомлень і механізмами доступу до даних.

Центральним компонентом системи є сервіс керування подіями оповіщення EfAlertEventService. Він реалізує основні сценарії роботи з подіями, включаючи створення події, подання на погодження, погодження або відхилення, запуск оповіщення, завершення та скасування подій. Сервіс координує взаємодію між доменною моделлю, механізмом визначення адресатів і підсистемою доставки повідомлень.

Для визначення отримувачів повідомлень використовується компонент RecipientResolver. Його призначення полягає у формуванні списку адресатів відповідно до вибраних зон оповіщення та усуненні дублювання отримувачів.

Результатом його роботи є набір унікальних адресатів, для яких надалі формуються повідомлення та спроби доставки.

Основу доменної моделі становлять класи AlertEvent, AlertMessage та DeliveryAttempt. Клас AlertEvent реалізує життєвий цикл події оповіщення та забезпечує контроль допустимих переходів між станами. Клас AlertMessage представляє персоналізоване повідомлення для конкретного отримувача, а DeliveryAttempt використовується для реєстрації окремих спроб доставки повідомлень через визначений канал зв’язку.

Обробка доставки повідомлень виконується компонентом DeliveryDispatcherHostedService, реалізованим на базі BackgroundService. Його основною задачею є вибір підготовлених спроб доставки, передавання повідомлень через відповідний канал зв’язку, оновлення статусів доставки та підтримка механізму повторних спроб у випадку виникнення помилок.

Для забезпечення можливості використання різних каналів доставки застосовано інтерфейс INotificationChannel. Такий підхід відповідає принципу інверсії залежностей та дозволяє замінювати конкретні реалізації каналів без зміни прикладної логіки системи. У поточному прототипі використовуються імітаційні канали доставки, які демонструють роботу механізму оповіщення без підключення зовнішніх сервісів (рис. 2.6).

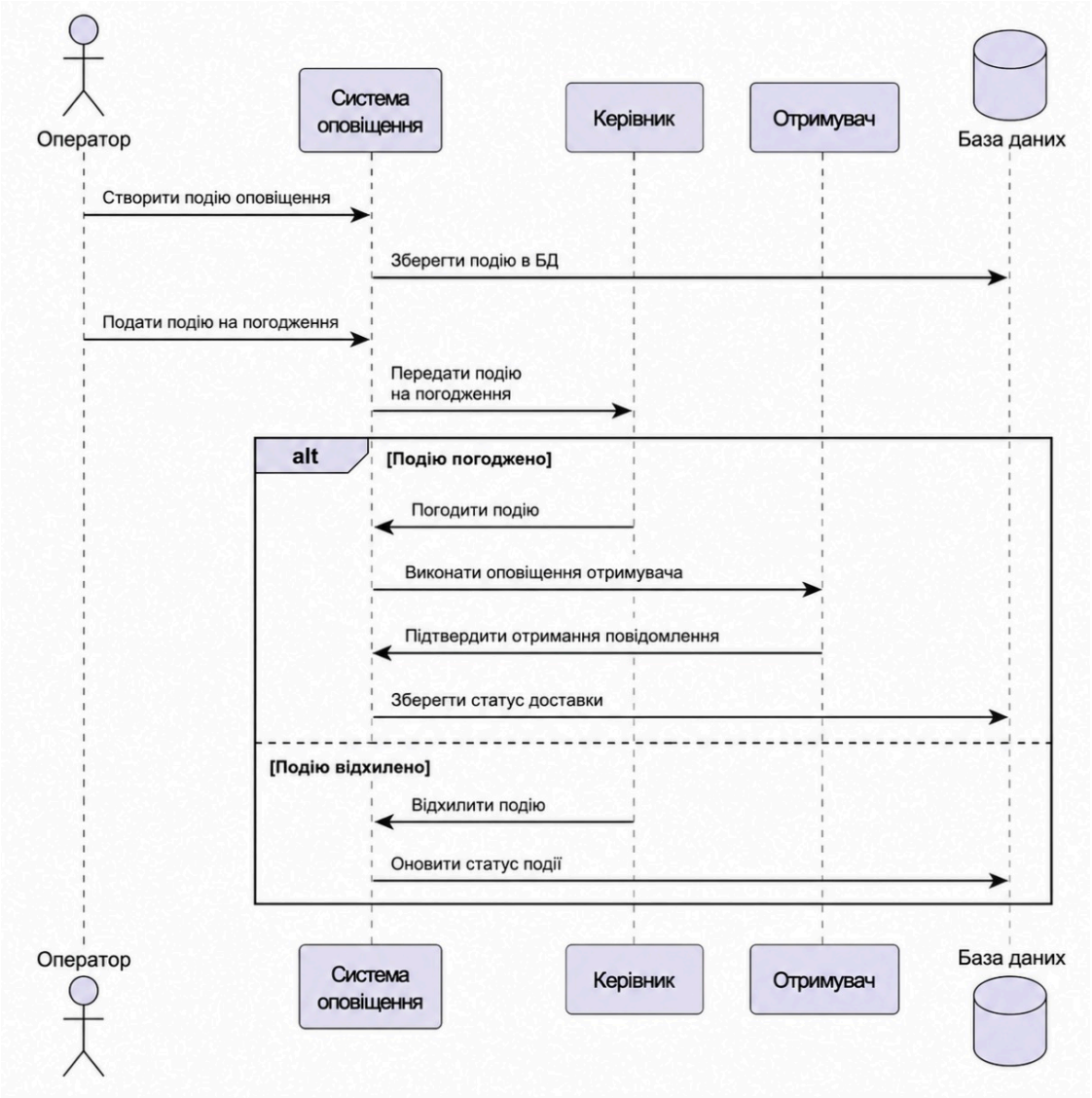

Рисунок 2.6 – Діаграма послідовності оповіщення

Підтвердження отримання повідомлень реалізовано за допомогою сервісу EfAcknowledgementService. Він забезпечує перевірку належності повідомлення поточному користувачеві та реєстрацію факту підтвердження отримання повідомлення.

Для відображення актуального стану системи використовується технологія SignalR. Компоненти AlertStatusHub та IAlertStatusPublisher забезпечують передавання змін статусів подій і повідомлень до вебінтерфейсу в режимі реального часу.

Доступ до бази даних реалізовано через компонент LocalAlertingDbContext. Він відповідає за збереження доменних сутностей, роботу з обліковими записами

користувачів та підтримку цілісності даних. Окремо реалізовано прикладний механізм захисту журналу аудиту, який забороняє зміну або видалення вже створених записів AuditLog.

Деталізовану діаграму послідовності створення та запуску події наведено в додатку Б.

2.6.1 Реалізація доменних правил

Важливою частиною реалізації є контроль бізнес-правил предметної області. Основні обмеження життєвого циклу подій та повідомлень реалізовано безпосередньо в доменній моделі. Такий підхід забезпечує цілісність бізнес-логіки незалежно від способу виклику відповідних операцій.

У системі реалізовано контроль допустимих переходів між станами подій, перевірку коректності підтвердження повідомлень, механізм повторних спроб доставки та автоматичне переведення невдалих доставок у стан DeadLetter після вичерпання ліміту повторних спроб.

Коректність реалізації доменних правил перевіряється за допомогою модульних тестів, які охоплюють життєвий цикл подій, механізм доставки повідомлень, підтвердження отримання та роботу журналу аудиту.

Діаграму станів події оповіщення наведено в додатку Б, рисунок Б.5.

2.6.2 Тестування компонентів

Для перевірки коректності реалізації програмного забезпечення використано модульні та інтеграційні тести. Модульне тестування охоплює доменні сутності та прикладні сервіси, тоді як інтеграційні тести перевіряють взаємодію компонентів системи, роботу механізмів авторизації та доступність основних функціональних можливостей.

Застосування автоматизованого тестування дозволило підтвердити коректність реалізації основних сценаріїв роботи системи та забезпечити відповідність програмного прототипу сформульованим функціональним вимогам.

Попри реалізацію основних сценаріїв оповіщення, поточний прототип має низку обмежень. Зокрема, він не інтегрований із реальними SMS або email-провайдерами, а механізм доставки повідомлень працює в демонстраційному режимі. Проте обрана архітектура дозволяє виконати таку інтеграцію в майбутньому без зміни доменної моделі та прикладної логіки системи.

2.7 Розробка інтерфейсу користувача

Інтерфейс користувача реалізовано у проєкті LocalAlerting.Web на основі технології Blazor Web App із використанням режиму Server Interactivity. Основним призначенням інтерфейсу є забезпечення взаємодії користувачів із програмним прототипом та надання доступу до функціональності відповідно до ролей і повноважень.

Під час розробки інтерфейсу було використано рольовий підхід до організації доступу. Набір доступних сторінок, елементів керування та операцій визначається роллю користувача, що забезпечує виконання вимог безпеки та запобігає несанкціонованому виконанню критичних дій.

Структура інтерфейсу побудована відповідно до основних сценаріїв роботи системи та включає такі функціональні модулі:

1) автентифікація користувачів;
2) створення та редагування подій оповіщення;
3) погодження критичних подій;
4) моніторинг стану подій та повідомлень;
5) підтвердження отримання повідомлень;
6) перегляд журналу аудиту;
7) перегляд довідкових даних.

Навігаційна структура вебзастосунку наведена на рисунку 2.7.

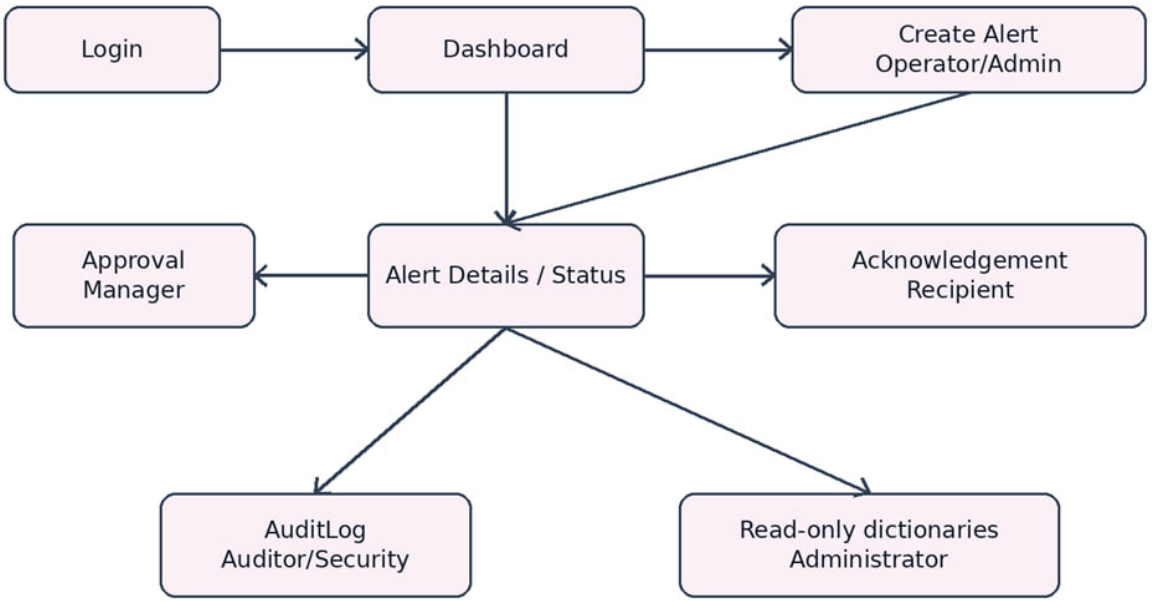

Рисунок 2.7 – Навігаційна структура вебінтерфейсу

Для реалізації рольового доступу визначено набір сторінок та операцій, доступних окремим категоріям користувачів. Відповідність ролей та основних функцій наведено в таблиці 2.6.

Таблиця 2.6 – Рольовий доступ до основних сторінок інтерфейсу

Роль	**Основні сторінки**	**Доступні дії**
1	**2**	**3**
Administrator	Dashboard, AuditLog, read-only довідники	Перегляд контрольних даних, аудит, доступ до демонстраційних довідників
Operator	Create Alert, Alert Details, Dashboard	Створення чернетки, подання події, запуск Dispatch, завершення або скасування
Manager	Approval, Alert Details	Погодження або відхилення Critical-події із зазначенням причини
Recipient	Acknowledgement	Перегляд і підтвердження лише власного Delivered-повідомлення
TechnicalSpecialist	Dashboard, Alert Status	Контроль статусів, кількості спроб і результатів доставки
SecurityOfficer	Dashboard, AuditLog	Контроль критичних дій, перегляд журналу аудиту
Auditor	Dashboard, AuditLog	Перевірка послідовності операцій і результатів оповіщення

Центральним елементом інтерфейсу є форма створення події оповіщення. Вона забезпечує введення основних параметрів події, вибір типу надзвичайної ситуації, визначення зон оповіщення та формування повідомлення. Даний екран використовується оператором для ініціювання процесу оповіщення (рис. 2.8).

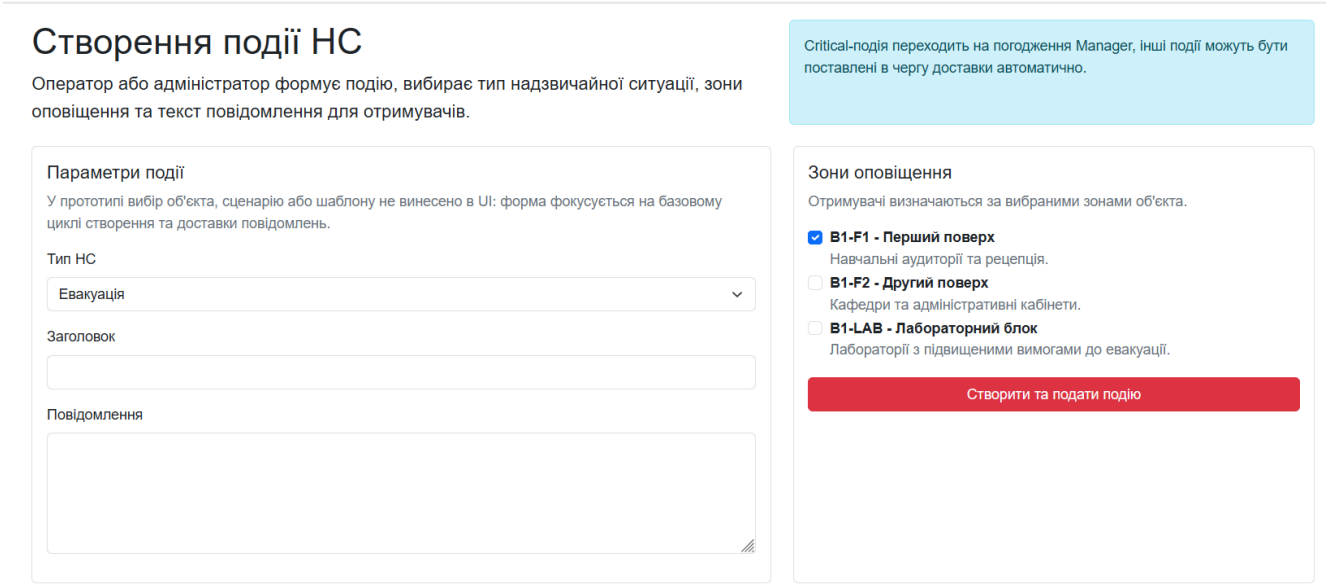

Рисунок 2.8 – Форма створення події надзвичайної ситуації

Для оперативного контролю роботи системи реалізовано інформаційну панель Dashboard.

Вона відображає перелік активних та завершених подій, поточний стан повідомлень, кількість успішних доставок, підтверджень отримання та помилок доставки. Dashboard використовується як основний засіб моніторингу процесу оповіщення (рис. 2.9).

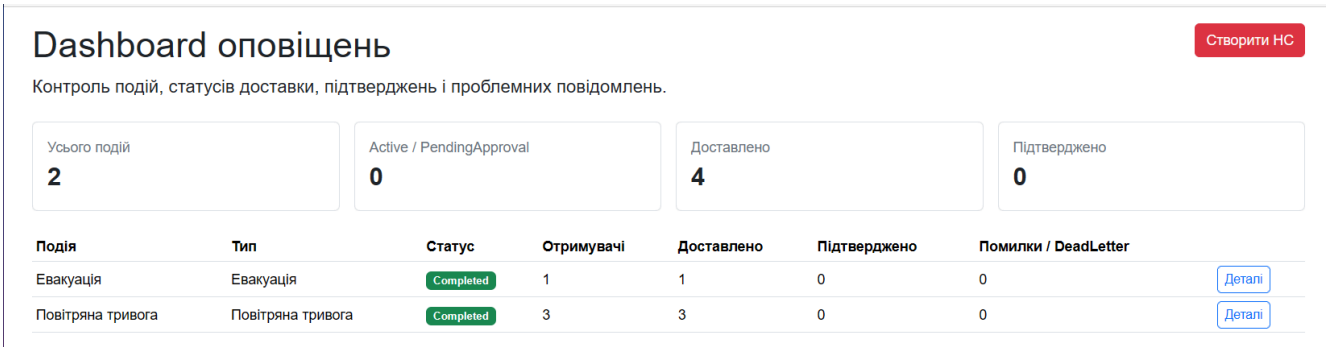

Рисунок 2.9 – Панель моніторингу подій оповіщення

Для детального аналізу окремої події використовується сторінка перегляду події та статусів доставки. Вона надає інформацію про отримувачів, результати доставки повідомлень, історію спроб надсилання та поточний стан події. Залежно від ролі користувача на сторінці можуть бути доступні операції погодження, відхилення, запуску, завершення або скасування події (рис. 2.10).

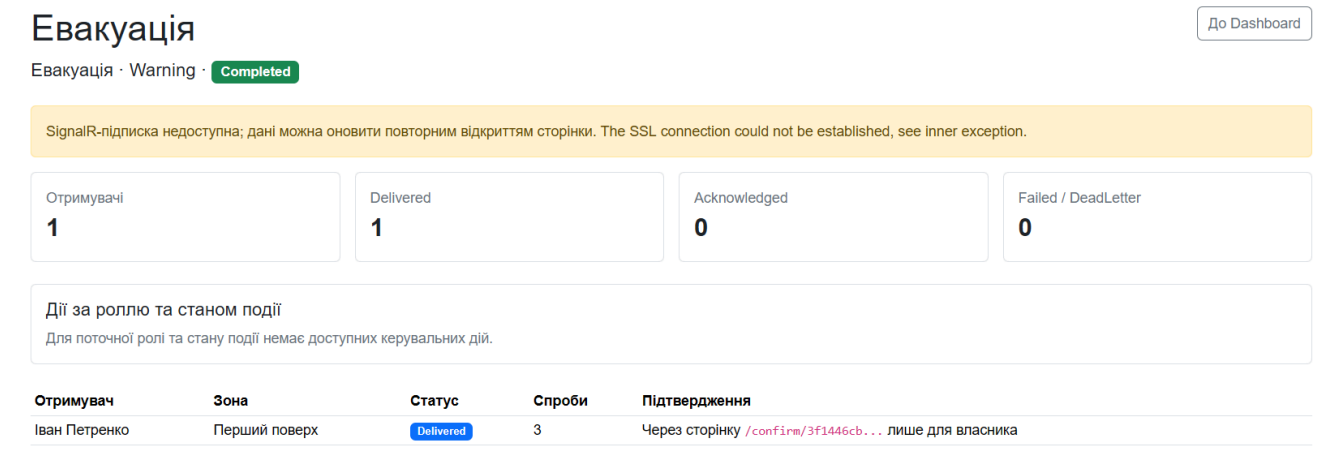

Рисунок 2.10 – Деталізація події та статусів доставки повідомлень

Для отримувачів повідомлень реалізовано окремий сценарій підтвердження отримання. Відповідна сторінка забезпечує перегляд повідомлення та реєстрацію факту його отримання. Система контролює належність повідомлення поточному користувачеві та не допускає підтвердження сторонніх повідомлень.

Для контролю критичних дій користувачів реалізовано сторінку журналу аудиту. Вона містить інформацію про виконані операції, час їх виконання,

користувача та об’єкт, над яким була виконана дія. Журнал аудиту використовується адміністраторами, аудиторами та відповідальними за безпеку для контролю роботи системи (рис. 2.11).

Під час розробки інтерфейсу особлива увага приділялася узгодженості елементів керування, простоті навігації та відповідності рольовій моделі доступу. Критичні операції доступні лише користувачам із відповідними повноваженнями, а набір доступних дій змінюється залежно від поточного стану події.

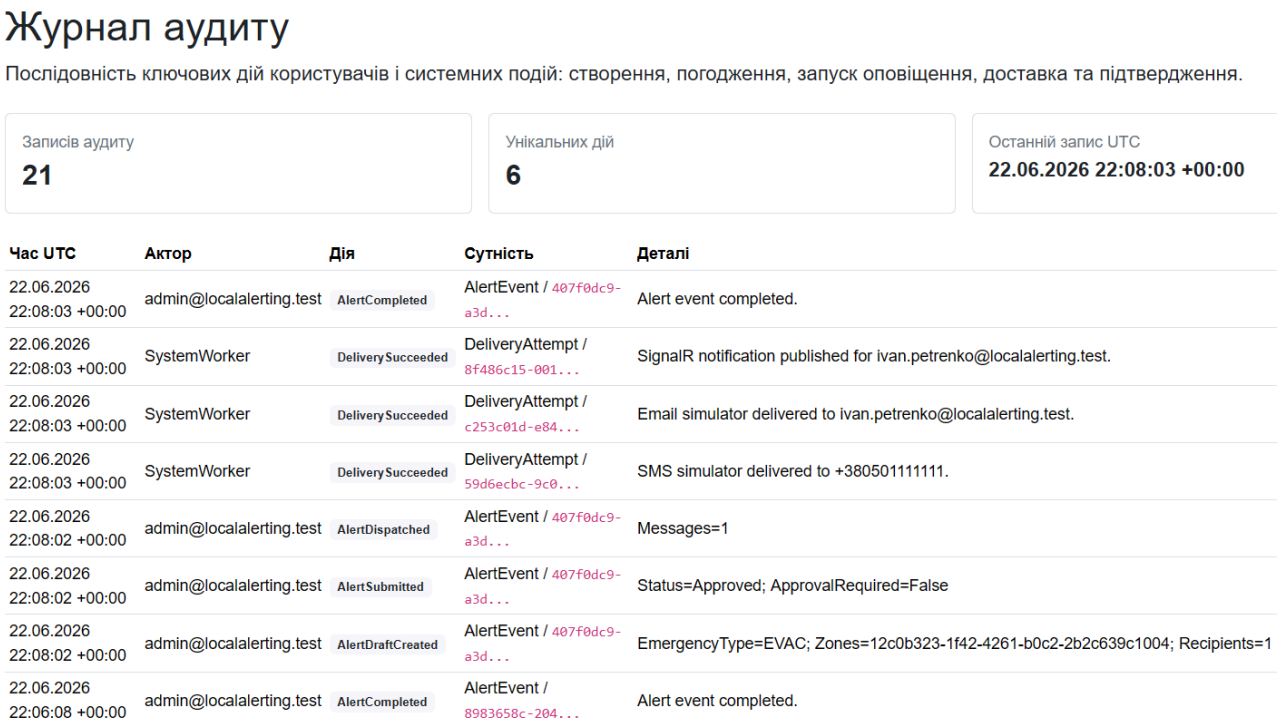

Рисунок 2.11 – Журнал аудиту дій користувачів та системних подій

Отже, розроблений вебінтерфейс забезпечує реалізацію всіх основних сценаріїв роботи програмного прототипу та створює зручне середовище для керування процесом оповіщення, контролю доставки повідомлень, підтвердження їх отримання та аудиту виконаних дій.

У другому розділі виконано проєктування та реалізацію прототипу системи оповіщення. Обґрунтовано ітераційно-інкрементальний підхід та спроєктовано багатошарову архітектуру (Modular Monolith).

Розроблено реляційну модель даних, UML-модель та обрано технологічний стек (.NET 10, ASP.NET Core, Blazor, EF Core, SQLite, SignalR, Identity). Реалізовано основні компоненти системи й вебінтерфейс із рольовим доступом.

Отримані результати підтверджують можливість реалізації прототипу та створюють основу для подальшого тестування й оцінювання.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

У цьому розділі розглянуто процес тестування, локального розгортання та супроводу програмного прототипу об’єктової системи оповіщення. Особливу увагу приділено перевірці відповідності реалізованого програмного забезпечення функціональним і нефункціональним вимогам, сформульованим на етапі аналізу та проєктування системи. Також наведено результати автоматизованого тестування, системні вимоги до середовища виконання та порядок локального запуску програмного прототипу.

Тестування є одним із ключових процесів життєвого циклу програмного забезпечення та використовується для підтвердження коректності реалізованих функцій, виявлення можливих дефектів і перевірки взаємодії програмних компонентів. У межах кваліфікаційної роботи тестування виконує роль верифікації архітектурних рішень, бізнес-логіки та механізмів взаємодії між підсистемами програмного забезпечення.

Розроблений програмний прототип має навчально-демонстраційний характер, тому метою тестування є не підтвердження готовності системи до промислової експлуатації, а перевірка правильності реалізації основних сценаріїв об’єктового оповіщення. До таких сценаріїв належать створення та погодження події, визначення адресатів за зонами, формування повідомлень, виконання доставки, повторні спроби надсилання, контроль станів повідомлень, підтвердження їх отримання та аудит критичних дій користувачів.

3.1. Тестування програмної системи

Тестування є одним із ключових процесів забезпечення якості програмного забезпечення та виконується з метою перевірки відповідності реалізованої системи сформульованим функціональним і нефункціональним вимогам. Для розробленого програмного прототипу тестування використовується як засіб

верифікації коректності реалізації бізнес-логіки, архітектурних рішень і взаємодії між основними компонентами системи.

Особливістю програмного прототипу об’єктової системи оповіщення є наявність критичних сценаріїв, пов’язаних із керуванням життєвим циклом події, доставкою повідомлень, підтвердженням їх отримання та контролем доступу до функцій системи. Тому під час тестування основна увага приділяється перевірці коректності переходів між станами подій, дотриманню рольової моделі доступу, правильності визначення адресатів повідомлень, роботі механізмів доставки та повторних спроб надсилання, а також забезпеченню цілісності журналу аудиту.

Об’єктами тестування є доменні сутності, прикладні сервіси, механізм визначення отримувачів, підсистема доставки повідомлень, механізм підтвердження отримання повідомлень, журнал аудиту, політики авторизації, вебendpoint-и та механізми публікації статусів у режимі реального часу. При цьому враховується, що програмний прототип не інтегрований із реальними SMS- або email-провайдерами, не взаємодіє зі спеціалізованими технічними засобами оповіщення та не проходив промислового навантажувального тестування.

Процес тестування побудовано за принципом поетапної перевірки компонентів системи. Спочатку виконуються модульні перевірки доменної логіки та окремих бізнес-правил, після чого здійснюється тестування прикладних сервісів і механізмів взаємодії між компонентами. Наступним етапом є інтеграційне тестування вебendpoint-ів, механізмів авторизації та політик доступу. Завершальним кроком є аналіз отриманих результатів та їх зіставлення з вимогами, сформульованими у першому розділі роботи.

У межах тестування оцінюються такі характеристики програмного забезпечення: функціональна коректність основних сценаріїв роботи, цілісність доменних правил, коректність обробки помилок, підтримка повторних спроб доставки повідомлень, відповідність рольової моделі доступу, захист критичних операцій, незмінність журналу аудиту та базова працездатність системи після локального розгортання. Нефункціональні характеристики, пов’язані з продуктивністю, відмовостійкістю, масштабуванням та інтеграцією з реальними

каналами доставки повідомлень, розглядаються як напрями подальшого дослідження та не входять до меж підтвердження поточного навчального прототипу.

Рисунок 3.1 – Загальний процес тестування програмного прототипу

Очікуваним результатом тестування є підтвердження коректності реалізації основних бізнес-сценаріїв системи та відповідності програмного забезпечення визначеним вимогам. Результати перевірок також дозволяють оцінити рівень готовності програмного прототипу до подальшого розвитку, визначити обмеження поточної реалізації та сформувати рекомендації щодо його вдосконалення.

3.1.1 Види та план тестування

Для підтвердження якості програмного забезпечення та перевірки відповідності реалізованої системи сформульованим вимогам використано декілька рівнів тестування. Кожен із них орієнтований на перевірку окремого аспекту функціонування програмного прототипу: від коректності реалізації доменної логіки до взаємодії між програмними компонентами та механізмами контролю доступу.

Основний акцент у роботі зроблено на модульному та інтеграційному тестуванні, оскільки саме ці види перевірок безпосередньо відповідають

реалізованому програмному забезпеченню та забезпечують достатню доказову базу для підтвердження працездатності навчального прототипу.

Навантажувальне тестування, тестування із використанням реальних каналів доставки повідомлень не входили до меж поточної кваліфікаційної роботи. Їх реалізація потребує окремого середовища розгортання, зовнішніх провайдерів та додаткових ресурсів, тому вони розглядаються як перспективні напрями подальшого розвитку та оцінювання системи.

Таблиця 3.1 – Види тестування програмної системи

Вид тестування	**Об’єкт перевірки**	**Спосіб виконання**	**Критерій успішності**	**Статус**
1	**2**	**3**	**4**	**5**
Модульне	Доменні переходи, retry/DeadLetter, дедуплікація, підтвердження, аудит, publisher	LocalAlerting.UnitTests	Усі тести завершуються без помилок	Виконано
Інтеграційне	Health endpoint, захист endpoint, RBAC-політики	LocalAlerting.IntegrationTests	Endpoint-и та політики працюють відповідно до очікуваної поведінки	Виконано
Функціональне	Цикл створення, погодження, запуску, доставки та підтвердження	Автоматизовано частково; частково ручна перевірка	Подія проходить лише допустимі стани	Виконано
Обробка помилок	Відхилення без причини, некоректні переходи, чуже підтвердження	Unit-тести	Некоректні дії блокуються	Виконано

Продовження таблиці 3.1

1	2	3	4	5
Продуктивність	Масові повідомлення і паралельні користувачі	Не виконувалося	Потрібно визначити метрики часу відповіді	Заплановано
Реальні канали	SMS/email-провайдери	Не виконувалося	Доставка підтверджується зовнішнім провайдером	Заплановано

План тестування сформовано відповідно до прийнятого підходу забезпечення якості програмного забезпечення. Перевірка виконується поетапно: від тестування окремих компонентів і бізнес-правил до перевірки інтеграційних обмежень та коректності роботи користувацьких сценаріїв. Такий підхід дозволяє локалізувати можливі дефекти та поступово підтверджувати коректність функціонування всієї системи.

Таблиця 3.2 – План тестування програмного прототипу

Етап	**Мета**	**Очікуваний результат**	**Критерій завершення**
1	**2**	**3**	**4**
1. Підготовка середовища	Забезпечити відтворюваний запуск тестів	Середовище готове до dotnet test	Проєкт збирається без критичних помилок
2. Логіка доставки	Перевірити retry, DeadLetter і чергу	Невдалі спроби повторюються або переходять у DeadLetter	Unit-тести доставки Passed
3. Отримувачі	Перевірити вибір активних адресатів без дублювання	Один отримувач не дублюється	RecipientResolver Passed
4. Підтвердження	Переконатися, що користувач підтверджує лише власне повідомлення	Власне повідомлення підтверджується, чуже відхиляється	Acknowledgement Passed

Продовження таблиці 3.2

1	2	3	4
5. Аудит	Перевірити незмінність AuditLog	UPDATE/DELETE блокуються	AuditLog Passed
6. Ручна перевірка UI	Перевірити доступність дій у вебінтерфейсі	Користувачі бачать лише дозволені дії	Passed

Критерієм успішності автоматизованого тестування є успішне проходження всіх модульних та інтеграційних перевірок без виникнення помилок або порушення визначених бізнес-правил. Критерієм завершення тестування в межах навчального прототипу є підтвердження працездатності основних функціональних компонентів системи, коректності реалізації рольової моделі доступу, механізмів доставки повідомлень, підтвердження отримання та журналювання критичних подій.

Водночас результати тестування слід інтерпретувати з урахуванням обмежень поточної реалізації. Зокрема, у межах роботи не виконувалися навантажувальні випробування, тестування відмовостійкості, перевірка інтеграції з реальними каналами передачі повідомлень та повноцінне browser/e2e-тестування. Тому отримані результати підтверджують коректність функціонування програмного прототипу в межах визначених сценаріїв використання, але не можуть розглядатися як підтвердження готовності системи до промислової експлуатації.

3.1.2 Розробка тестових сценаріїв

Тестові сценарії сформовано на основі функціональних вимог, визначених у першому розділі роботи, та ключових бізнес-сценаріїв програмного прототипу. Основною метою їх розробки є перевірка коректності реалізації доменної логіки, взаємодії між програмними компонентами та дотримання обмежень, передбачених бізнес-правилами системи.

Набір тестових сценаріїв охоплює повний життєвий цикл події оповіщення, включаючи створення події, погодження критичних повідомлень, запуск процесу

доставки, виконання повторних спроб надсилання, переведення повідомлень у стан DeadLetter, визначення отримувачів за зонами, підтвердження отримання повідомлень, аудит критичних дій користувачів та перевірку механізмів рольового доступу. Окрему увагу приділено негативним сценаріям, які перевіряють коректність реакції системи на недопустимі дії та помилкові умови виконання.

У межах роботи реалізовано автоматизований набір тестів, що складається з 22 перевірок, серед яких 16 модульних та 6 інтеграційних. Модульні тести використовуються для перевірки окремих компонентів доменної моделі та бізнес-правил, тоді як інтеграційні перевірки підтверджують коректність взаємодії між компонентами програмної системи та механізмами авторизації.

У таблиці 3.3 наведено узагальнений перелік основних тестових сценаріїв та очікуваних результатів їх виконання.

Таблиця 3.3 – Узагальнення результатів тестування

Тип перевірки	**Середовище**	**Фактичний результат**	**Доказ**	**Статус**
1	**2**	**3**	**4**	**5**
Build / test	Windows; .NET SDK 10.0.204; SQLite	22/22	dotnet test, 0 failed	Passed
Unit	LocalAlerting.UnitTests	16/16	Workflow, owner acknowledgement, audit, retry, resolver, publisher	Passed
Integration	LocalAlerting.IntegrationTests	6/6	Health, auth, 4 RBAC policy cases	Passed
RBAC policies	Test host	4 automated cases	Operator, Manager, TechnicalSpecialist, Auditor	Passed
Browser/e2e	Browser runner	0	Не виконано	Planned

Продовження таблиці 3.3

1	2	3	4	5
Реальні SMS/email	Реальний провайдер	0	Не підключено	Planned
Load/failover	Окреме навантажувальне середовище	0	Не виконано	Planned

Для аналізу результатів тестування сформовано узагальнений журнал перевірок, який дозволяє відокремити підтверджені автоматизованими тестами характеристики системи від перевірок, що залишаються запланованими на наступні етапи розвитку програмного забезпечення. Такий підхід забезпечує коректну інтерпретацію результатів тестування та дозволяє об’єктивно оцінити рівень готовності програмного прототипу.

За результатами тестування критичних дефектів у межах автоматизованого набору не зафіксовано, оскільки всі 22 перевірки завершилися успішно. Водночас журнал результатів фіксує обмеження тестування: відсутність повного browser/e2e сценарію, реальних SMS/email-провайдерів, навантажувальної перевірки, failover-тестування та перевірки повного SignalR-клієнта. Ці обмеження потрібно враховувати під час формування висновку про готовність системи.

3.2 Розгортання програмної системи та системні вимоги

Розгортання програмного прототипу виконано у локальному демонстраційному середовищі. Такий підхід відповідає навчальному характеру роботи, оскільки дає змогу перевірити основну бізнес-логіку без підключення складної зовнішньої інфраструктури, реальних провайдерів доставки або сертифікованих технічних засобів оповіщення.

Прототип реалізовано як веборієнтовану систему на основі ASP.NET Core. Для доступу до даних використовується Entity Framework Core, а як демонстраційна СУБД застосовується SQLite. База даних зберігає події

оповіщення, повідомлення, спроби доставки, отримувачів, контактні точки, підтвердження та записи аудиту. Для автентифікації і рольового доступу використовується ASP.NET Core Identity, а для публікації оновлень статусів передбачено SignalR.

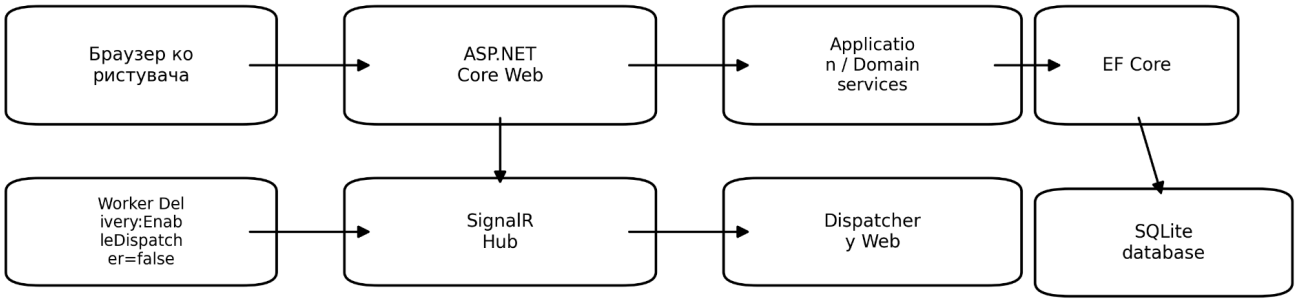

Рисунок 3.2 – Схема розгортання програмного прототипу

Послідовність локального розгортання програмної системи передбачає підготовку робочої станції з Windows та встановленим .NET SDK 10.0.204, отримання вихідного коду проєкту з локального каталогу або репозиторію [потрібно уточнити], перевірку наявності проєктів Domain, Application, Infrastructure, Web, Worker, UnitTests та IntegrationTests, налаштування connection string до SQLite, перевірку параметра Delivery:EnableDispatcher, застосування міграцій бази даних або створення демонстраційної SQLite-бази, запуск вебзастосунку ASP.NET Core, перевірку доступності вебінтерфейсу й health endpoint та виконання команди dotnet test.

Після запуску прототипу необхідно перевірити наявність seed data. Початкові дані мають містити ролі Administrator, Manager, Operator, Recipient, TechnicalSpecialist, SecurityOfficer та Auditor. Окремі облікові записи Recipient повинні бути пов’язані з адресатами, оскільки механізм підтвердження отримання залежить від ApplicationUserId власника повідомлення.

Підтримка програмного прототипу передбачає контроль технічних журналів, резервне копіювання SQLite-бази, перевірку результатів спроб доставки, перегляд записів DeadLetter, актуалізацію довідкових даних отримувачів і зон, а також обережне застосування міграцій бази даних. Для зменшення ризику втрати даних

демонстраційну базу потрібно регулярно копіювати перед внесенням змін у схему або seed data.

Для подальшого переходу від навчального прототипу до промислового рішення необхідно додати моніторинг доступності, централізоване журналювання, інтеграцію з реальними провайдерами доставки, резервування компонентів, контроль секретів, процедури відновлення після збоїв і регламент оновлення залежностей. Окремо потрібно провести навантажувальні та відмовостійкісні тести, оскільки поточне локальне середовище не підтверджує поведінку системи під промисловим навантаженням.

3.3 Верифікація програмної системи

Верифікація програмної системи полягає у перевірці відповідності реалізованого прототипу поставленим вимогам. На відміну від підрозділу 3.1.2, де розглянуто окремі тестові сценарії, у цьому підрозділі результати тестування узагальнюються на рівні вимог, реалізованих компонентів і доказів перевірки.

Основою верифікації є зіставлення функціональних і нефункціональних вимог із програмними компонентами, які їх реалізують, та тестами, які підтверджують правильність поведінки. Такий підхід дає змогу показати, що вимоги не залишилися декларативними, а мають відображення в доменній моделі, прикладних сервісах, базі даних, механізмах доступу, endpoint-ах і тестовому наборі.

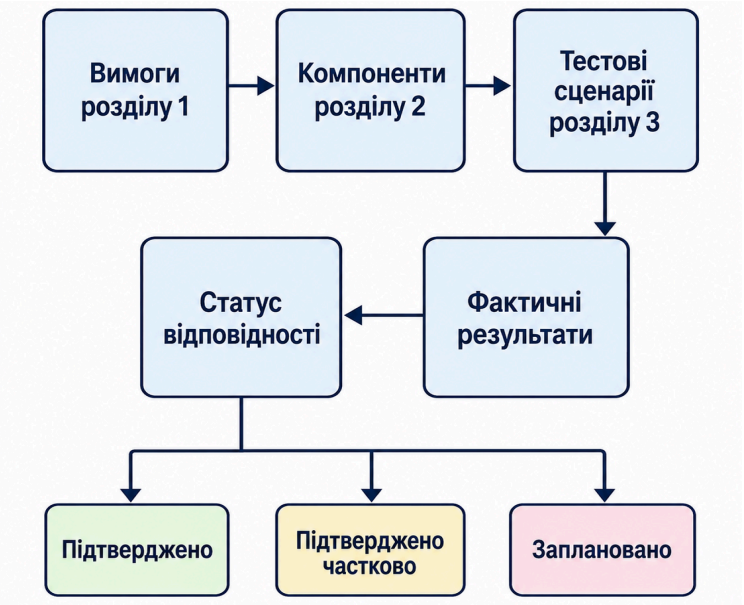

Рисунок 3.3 – Узагальнена схема верифікації програмної системи

Результати верифікації показують, що основні функціональні вимоги прототипу підтверджено автоматизованими тестами або частково підтверджено на рівні компонентів і політик. Найкраще перевіреними є доменні правила життєвого циклу події, логіка retry/DeadLetter, дедуплікація отримувачів, підтвердження власного повідомлення, захист журналу аудиту, health endpoint і частина RBAC-політик. Частково підтвердженими залишаються SignalR-доставка через повний клієнтський сценарій, UI-поведінка для всіх ролей, промислова продуктивність і реальна доставка повідомлень через зовнішніх провайдерів.

Таким чином, програмний прототип можна вважати верифікованим у межах навчально-демонстраційного середовища. Він коректно відтворює основний цикл об’єктового оповіщення, забезпечує контроль станів, рольове розмежування, підтвердження отримання та аудит критичних дій. Водночас для впровадження у реальне експлуатаційне середовище необхідні додаткові перевірки: browser/e2e тестування, підключення SMS/email-провайдерів, навантажувальне тестування, перевірка відмовостійкості, налаштування моніторингу та формалізація регламентів підтримки.

Отже, у розділі 3 систематизовано підхід до тестування, подано види та план перевірок, сформовано тестові сценарії, описано локальне розгортання й системні вимоги, а також виконано верифікацію відповідності прототипу основним функціональним і нефункціональним вимогам. Результати підтверджують працездатність навчального прототипу в локальному середовищі та визначають перелік перевірок, необхідних перед можливим промисловим упровадженням.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У цьому розділі розглянуто питання безпеки життєдіяльності та охорони праці, пов’язані з використанням програмного прототипу об’єктової системи оповіщення. Основну увагу приділено організації оповіщення персоналу, підтримці правильних дій людей під час небезпеки та безпечній роботі оператора з автоматизованим робочим місцем системи.

Під час аналізу умов роботи оператора з автоматизованим робочим місцем враховано положення щодо охорони праці користувачів комп’ютерів [33], а також загальні підходи до безпеки життєдіяльності, що передбачають ідентифікацію небезпечних чинників, оцінювання ризиків та визначення організаційних заходів захисту людини у виробничому середовищі [34].

4.1 Безпека життєдіяльності

У розробленому прототипі ці положення відображено через модель події надзвичайної ситуації, вибір типу події та зони об’єкта, визначення отримувачів, формування повідомлень, контроль статусів доставки, підтвердження отримання та журнал аудиту. Такий підхід не замінює сертифіковану систему оповіщення, але демонструє програмну логіку, яка може підтримувати організоване реагування на об’єктовому рівні.

З позиції безпеки життєдіяльності важливим є не лише факт надсилання сигналу, а й забезпечення зрозумілої поведінкової інструкції для людини: отримувач повинен швидко визначити характер небезпеки, місце її прояву та безпечну дію. Тому у прототипі повідомлення формуються як короткі текстові інструкції, пов’язані із зоною об’єкта, типом події та очікуваною дією персоналу [34].

Дослідження попереджень наголошують на швидкості, зрозумілості та наявності конкретної захисної дії [35–40]. Тому повідомлення має бути коротким і

містити тип загрози, локалізацію та очікувану дію. Оцінювання сприйняття текстів користувачами в межах прототипу не проводилося.

У таблиці 4.1 наведено приклади дій персоналу й оператора для типових подій, які можуть бути використані у демонстраційному сценарії прототипу.

У прототипі фактично реалізовано шаблони, зональну вибірку, статуси, підтвердження й аудит. Навчальний режим, попередній перегляд і перевірка актуальності груп є організаційними рекомендаціями, а не реалізованими функціями.

Узагальнення цих ризиків наведено у таблиці 4.2.

Таблиця 4.1 – Дії персоналу та оператора після отримання повідомлення

Тип події	**Дії персоналу**	**Дії оператора системи**
1	**2**	**3**
Пожежа або задимлення	Припинити роботу, не користуватися ліфтами, рухатися до евакуаційного виходу, діяти відповідно до плану евакуації	Запустити сценарій евакуації, перевірити доставку повідомлень, контролювати підтвердження та повідомити відповідальних осіб
Повітряна тривога	Вимкнути небезпечне обладнання, перейти до укриття, допомогти особам, які потребують супроводу	Запустити сценарій переходу до укриття, контролювати отримання повідомлень працівниками визначених зон
Витік небезпечної речовини	Залишити небезпечну зону, не наближатися до джерела загрози, виконувати інструкції відповідальних осіб	Оповістити персонал небезпечної зони, технічну службу, охорону та керівництво
Аварія інженерних мереж	Не використовувати пошкоджені мережі, повідомити про ознаки аварії, не виконувати самовільних ремонтних дій	Оповістити технічну службу, керівництво та черговий персонал
Тестове оповіщення	Підтвердити отримання, виконати навчальний алгоритм без паніки та без припинення критичних процесів, якщо це не передбачено інструкцією	Запустити тестовий сценарій, перевірити коректність груп отримувачів, зафіксувати результати перевірки

Таблиця 4.2 – Ризики під час оповіщення та засоби їх зниження у прототипі

Ризик	Можливий наслідок	Засіб зниження у прототипі
1	2	3
Затримка запуску оповіщення	Працівники пізно отримують інформацію про небезпеку	Шаблони за типом НС і зональна адресація
Нечіткий текст повідомлення	Неправильне трактування інструкції або паніка	Короткі шаблони з типом події, зоною та потрібною дією
Неповний список отримувачів	Частина персоналу не отримує повідомлення	Визначення активних отримувачів за зонами; групова вибірка не реалізована
Ігнорування повідомлення	Відсутність підтвердженої реакції працівника	Механізм підтвердження отримання повідомлення
Втрата контролю за перебігом оповіщення	Оператор не бачить, кому повідомлення доставлено, а кому ні	Панель стану зі статусами повідомлень
Неможливість подальшого аналізу	Важко встановити послідовність дій під час події	Журнал аудиту та збереження історії подій

4.2 Основи охорони праці

Робота оператора поєднує дефіцит часу та відповідальність за правильність дій. Поточний UI використовує текстові статуси, рольове відображення дій і журнал аудиту; спеціальний preview, навчальний режим та окреме підтвердження критичного запуску не реалізовано й розглядаються як напрями ергономічного посилення.

Вимоги до безпечної роботи користувачів комп’ютерів передбачають раціональну організацію робочого місця, правильне розміщення монітора, клавіатури та миші, дотримання режиму праці й відпочинку, зниження зорового навантаження та врахування психофізіологічної напруги оператора [33]. Для АРМ системи оповіщення це особливо важливо, оскільки оператор працює з критичними повідомленнями в умовах дефіциту часу.

Санітарно-гігієнічні умови також впливають на працездатність оператора. Мікроклімат приміщення визначається температурою, відносною вологістю,

швидкістю руху повітря та тепловим випромінюванням [41]. Для операторської роботи доцільно підтримувати комфортну температуру, помірну вологість, регулярне провітрювання або вентиляцію, відсутність протягів, надмірного шуму та різких перепадів температури. Освітлення має бути рівномірним, без прямого засвічення екрана, різкого контрасту між монітором і навколишнім середовищем та блиску на робочій поверхні.

Електробезпека робочого місця оператора забезпечується справною електромережею, цілими кабелями, правильним розміщенням подовжувачів, відсутністю перевантаження розеток, захистом обладнання від потрапляння рідини та використанням справного обладнання [42]. Для критичного робочого місця системи оповіщення доцільно передбачити джерело безперебійного живлення, яке дає змогу завершити поточні дії або зберегти працездатність системи під час короткочасного зникнення електроживлення. У разі запаху горілого, перегрівання, іскріння, пошкодження кабелю або нестабільної роботи обладнання оператор повинен припинити використання пристрою та повідомити відповідальну особу.

Пожежна безпека робочого місця передбачає недопущення експлуатації пошкоджених розеток, кабелів та електроприладів, відсутність горючих матеріалів біля обладнання, вільні евакуаційні проходи, справність первинних засобів пожежогасіння та знання плану евакуації [43].

У таблиці 4.3 наведено основні небезпечні та шкідливі чинники роботи оператора системи оповіщення і заходи їх зниження.

Таблиця 4.3 – Небезпечні та шкідливі чинники роботи оператора і заходи їх зниження

Чинник	**Можливий прояв**	**Заходи зниження**
1	**2**	**3**
Психоемоційне навантаження	Помилковий вибір сценарію, затримка запуску, перевтома	Навчальні тренування, затверджені шаблони, підтвердження критичних дій, резервний оператор

Продовження таблиці 4.3

Чинник	Можливий прояв	Заходи зниження
1	**2**	**3**
Інформаційне перевантаження	Втрата важливого статусу або неправильна оцінка ситуації	Панель стану, фільтрація подій, текстові статуси, пріоритети повідомлень
Тривала робота з екраном	Втома зору, зниження концентрації, дискомфорт	Регламентовані перерви, правильне розміщення екрана, відсутність відблисків
Незручна робоча поза	Напруження спини, шиї, рук	Ергономічне крісло, достатня робоча поверхня, правильна висота клавіатури й миші
Несприятливий мікроклімат	Втома, зниження уваги, погіршення самопочуття	Підтримання нормативних параметрів температури, вологості та повітрообміну
Несправне електрообладнання	Ураження електричним струмом, зупинка робочого місця	Справні кабелі, неперевантажені розетки, джерело безперебійного живлення, своєчасне повідомлення про несправності
Пожежна небезпека	Загроза оператору, обладнанню та приміщенню	Первинні засоби пожежогасіння, вільні проходи, справна електромережа, дотримання правил пожежної безпеки

Безпека експлуатації залежить від реалізованого рольового доступу, станів і аудиту, а також від організаційних заходів: інструктажів, тренувань, режиму праці, справності обладнання, електро- та пожежної безпеки. Програмний прототип не замінює ці процедури.

ВИСНОВКИ

У кваліфікаційній роботі вирішено завдання розробки програмного прототипу об’єктової системи оповіщення для підтримки процесів створення, погодження, доставки та контролю повідомлень про надзвичайні ситуації. У процесі виконання роботи реалізовано основні етапи життєвого циклу програмного забезпечення, передбачені методологією інженерії програмного забезпечення: аналіз предметної області, визначення вимог, проєктування архітектури, розробку програмних компонентів, тестування та оцінювання отриманих результатів.

Результати роботи підтверджують досягнення поставленої мети та виконання всіх визначених завдань кваліфікаційної роботи. Розроблений програмний прототип демонструє можливість практичної реалізації основних процесів об’єктового оповіщення із застосуванням сучасних технологій розробки програмного забезпечення, принципів багатошарової архітектури та засобів забезпечення якості програмних систем.

На етапі аналізу досліджено предметну область об’єктових систем оповіщення, нормативно-правові вимоги та існуючі програмні рішення. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до системи, визначено ролі користувачів і варіанти використання програмного забезпечення.

У межах проєктування розроблено багатошарову архітектуру програмного забезпечення за принципами Modular Monolith, спроєктовано реляційну модель даних та UML-модель основних класів доменної області. Для реалізації системи обрано сучасний технологічний стек на основі .NET 10, ASP.NET Core, Blazor Web App, Entity Framework Core, SQLite, SignalR та ASP.NET Core Identity.

У результаті реалізації створено програмний прототип, який підтримує повний цикл обробки події оповіщення: створення події, погодження критичних повідомлень, визначення адресатів за зонами, формування повідомлень, доставку через програмні канали, підтвердження отримання повідомлень та аудит

виконаних дій. У системі реалізовано сім ролей користувачів: Administrator, Manager, Operator, Recipient, TechnicalSpecialist, SecurityOfficer та Auditor, для яких передбачено розмежування доступу відповідно до функціональних обов’язків.

Для перевірки коректності роботи програмного забезпечення виконано автоматизоване тестування основних компонентів системи. Реалізовано 22 тести, з яких 16 модульних та 6 інтеграційних. Результати тестування підтвердили коректність реалізації доменних правил, механізмів доставки повідомлень, підтвердження отримання, рольового доступу та взаємодії між основними компонентами програмного забезпечення.

Практичним результатом роботи є працездатний програмний прототип об’єктової системи оповіщення, який може використовуватися як основа для подальшої розробки спеціалізованих інформаційних систем у сфері цивільного захисту, безпеки підприємств, установ та організацій. Запропоновані архітектурні рішення забезпечують можливість масштабування системи та інтеграції з реальними сервісами передачі повідомлень.

Подальший розвиток роботи доцільно спрямувати на інтеграцію з реальними SMS-, email- та push-провайдерами, розширення механізмів моніторингу й аудиту, впровадження засобів забезпечення відмовостійкості та резервування, проведення навантажувального тестування, а також комплексного тестування системи в умовах, наближених до реальної експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кодекс цивільного захисту України : Закон України № 5403-VI, [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/5403-17/print (дата звернення: 13.04.2026).

2. Про затвердження Положення про організацію оповіщення про загрозу виникнення або виникнення надзвичайних ситуацій та організації зв'язку у сфері цивільного захисту : Постанова Кабінету Міністрів України № 733, [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/733-2017-%D0%BF/print (дата звернення: 14.04.2026).

3. Про затвердження Інструкції щодо практик чи процедур проектування, дослідження, введення в експлуатацію, експлуатації та технічного обслуговування (супроводження) автоматизованих систем централізованого оповіщення : Наказ МВС України № 93, [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/z0418-19/print (дата звернення: 25.04.2026 .

4. Ammann P., Offutt J. Introduction to Software Testing. 2nd ed. Cambridge : Cambridge University Press, 2017. 364 p. DOI: 10.1017/9781316771273.

5. Avizienis A., Laprie J.-C., Randell B., Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing. 2004. Vol. 1, No. 1. P. 11-33. DOI: 10.1109/TDSC.2004.2.

6. Knight J. C. Safety critical systems: challenges and directions. Proceedings of the 24th International Conference on Software Engineering. New York : ACM, 2002. P. 547-550. DOI: 10.1145/581339.581406.

7. Leveson N. G., Turner C. S. An investigation of the Therac-25 accidents. Computer. 1993. Vol. 26, No. 7. P. 18-41. DOI: 10.1109/MC.1993.274940.

8. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 - Інженерія програмного забезпечення / Д. М. Михалик, Г. Б. Цуприк, В. М. Бревус. Тернопіль : ТНТУ, 2024. 45 с. URL: https://elartu.tntu.edu.ua/handle/lib/50317 (дата звернення: 25.05.2026).

9. Everbridge Mass Notification [Електронний ресурс]. URL: https://www.everbridge.com/resource/mass-notification-data-sheet/ (дата звернення: 18.04.2026).

10. AlertMedia Mass Notification System [Електронний ресурс]. URL: https://www.alertmedia.com/products/emergency-mass-notification/ (дата звернення: 18.04.2026).

11. Rave Alert - Mass Notification System [Електронний ресурс]. URL: https://www.motorolasolutions.com/enus/products/command-center-software/rave-alert.html (дата звернення: 19.04.2026).

12. CodeRED Emergency Notification for Public Safety [Електронний ресурс]. URL: https://codered.crisis24.com/ (дата звернення: 19.04.2026).

13. Повітряна тривога [Електронний ресурс]. URL: https://www.ukrainealarm.com/ (дата звернення: 20.04.2026).

14. Overview of ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/overview?view=aspnetcore-10.0 (дата звернення: 27.04.2026).

15. Entity Framework Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 29.04.2026).

16. Logging in .NET and ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-10.0 (дата звернення: 04.05.2026).

17. Introduction to Identity on ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-10.0 (дата звернення: 30.04.2026).

18. Role-based authorization in ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-10.0 (дата звернення: 30.04.2026).

19. ASP.NET Core Data Protection Overview [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/introduction?view=aspnetcore-10.0 (дата звернення: 01.05.2026).

20. Create web APIs with ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-10.0 (дата звернення: 05.05.2026).

21. Background tasks with hosted services in ASP.NET Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-10.0 (дата звернення: 06.05.2026).

22. Overview of ASP.NET Core SignalR [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-10.0 (дата звернення: 07.05.2026).

23. Introduction to .NET [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/dotnet/core/introduction (дата звернення: 27.04.2026).

24. C# language documentation [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 28.04.2026).

25. Spring Boot [Електронний ресурс]. URL: https://spring.io/projects/spring-boot (дата звернення: 28.04.2026).

26. NestJS Documentation [Електронний ресурс]. URL: https://docs.nestjs.com/ (дата звернення: 29.04.2026).

27. FastAPI Features [Електронний ресурс]. URL: https://fastapi.tiangolo.com/features/ (дата звернення: 29.04.2026).

28. Олянін Д., Цуприк Г., Говорущенко Т., Багрій-Заяць О., Андрущак І. Transformer Neural Networks in Industry 4.0. Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. Ternopil : Ternopil Ivan Puluj National Technical University, 2025. URL: https://ceur-ws.org/Vol-4057/ (дата звернення: 02.05.2026).

29. Tsupryk H., Olianin D. Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security. 2025. P. 125–130. DOI: https://doi.org/10.32782/IT/2025-2-13.

30. Олянін D., Цуприк Н. Огляд ролі трансформерних нейронних мереж у видобуванні інформації із неструктурованих даних. Measuring and Computing Devices in Technological Processes. 2025. Vol. 82, No. 2. P. 360–364. DOI: https://doi.org/10.31891/2219-9365-2025-82-52.

31. Olianin D., Tsupryk H. LLM-based Extraction from Resumes. Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. International Scientific Unity, 2025. P. 72–76.

32. Kotov Y., Yavorska E., Tsupryk H., Dzierżak R., Reshetnik O., Bokovets V. Evaluating interoperability and data quality in FHIR-based AI assessment pipelines. Proc. SPIE 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025. Article 140091F. DOI: https://doi.org/10.1117/12.3100561.

33. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

34. Запорожець О.І. Безпека життєдіяльності : підручник. 2-ге вид. Київ : Центр учбової літератури, 2020. 448 с.

35. Mileti D. S., Sorensen J. H. Communication of Emergency Public Warnings: A Social Science Perspective and State-of-the-Art Assessment. Oak Ridge : Oak Ridge National Laboratory, 1990. 166 p.

36. Bean H. et al. The study of mobile public warning messages: a research review and agenda. Review of Communication. 2015. Vol. 15, No. 1. P. 60-80. DOI: 10.1080/15358593.2015.1014402.

37. Wood M. M. et al. Communicating actionable risk for terrorism and other hazards. Risk Analysis. 2018. Vol. 38, No. 4. P. 601-615.

38. Wogalter M. S. (ed.). Handbook of Warnings. Mahwah, NJ : Lawrence Erlbaum Associates, 2006. 864 p.

39. Lindell M. K., Perry R. W. The Protective Action Decision Model: theoretical modifications and additional evidence. Risk Analysis. 2012. Vol. 32, No. 4. P. 616-632. DOI: 10.1111/j.1539-6924.2011.01647.x.

40. Sutton J. et al. Warning tweets: serial transmission of messages during the warning phase of a disaster event. Information, Communication & Society. 2014. Vol. 17, No. 6. P. 765-787. DOI: 10.1080/1369118X.2013.862561.

41. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/go/va042282-99 (дата звернення: 18.05.2026).

42. Про затвердження Правил безпечної експлуатації електроустановок споживачів : Наказ Держнаглядохоронпраці № 4 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/z0093-98 (дата звернення: 19.05.2026).

43. Про затвердження Правил пожежної безпеки в Україні : Наказ МВС України № 1417 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/go/z0252-15 (дата звернення: 20.05.2026).

ДОДАТКИ

ДОДАТОК А

МАТРИЦЯ ВІДПОВІДНОСТІ НОРМАТИВНИХ ВИМОГ І ФУНКЦІЙ ПРОТОТИПУ

Таблиця А.1 – Матриця відповідності нормативних вимог і функцій

№	Вимога	Критерій	Компонент	Тест	Результат
1	Workflow	Лише допустимі переходи	AlertEvent	7 unit	Passed
2	Manager approval	Critical без approval не dispatch	ApproveAsync + policy	Unit + перевірка складу policy	Частково підтверджено
3	Owner confirmation	Чуже повідомлення заборонено	AcknowledgementService	2 unit	Passed
4	Deduplication	Унікальні адресати	RecipientResolver	2 unit	Passed
5	Retry/DeadLetter	Ліміт спроб	DeliveryAttempt	2 unit	Passed
6	Append-only audit	UPDATE/DELETE заборонено	DbContext	1 unit	Passed
7	SignalR publisher	Виклик publisher без реального Hub-клієнта	Channel + publisher abstraction	UT-15, UT-16	Частково автоматизовано
8	Protected API	Анонімний доступ відхилено	Policy	1 integration	Passed
9	Health	HTTP 200	/health	1 integration	Passed

Матриця відображає відповідність на рівні програмного прототипу. Вона не підтверджує сертифікаційну придатність системи, не означає наявність реально погодженої інтеграції з місцевою або територіальною системою оповіщення та не замінює технічні засоби оповіщення.

ДОДАТОК Б

UML-ДІАГРАМИ ТА АРХІТЕКТУРНІ ДІАГРАМИ ПРОТОТИПУ

Діаграми відображають проєктну модель прототипу, взаємодію ролей із системою, структуру компонентів, основні послідовності та життєві цикли подій і повідомлень.

Б.1 Діаграма варіантів використання

Рисунок Б.1 – Діаграма варіантів використання фактичного прототипу

Б.2 Діаграма компонентів / архітектури системи

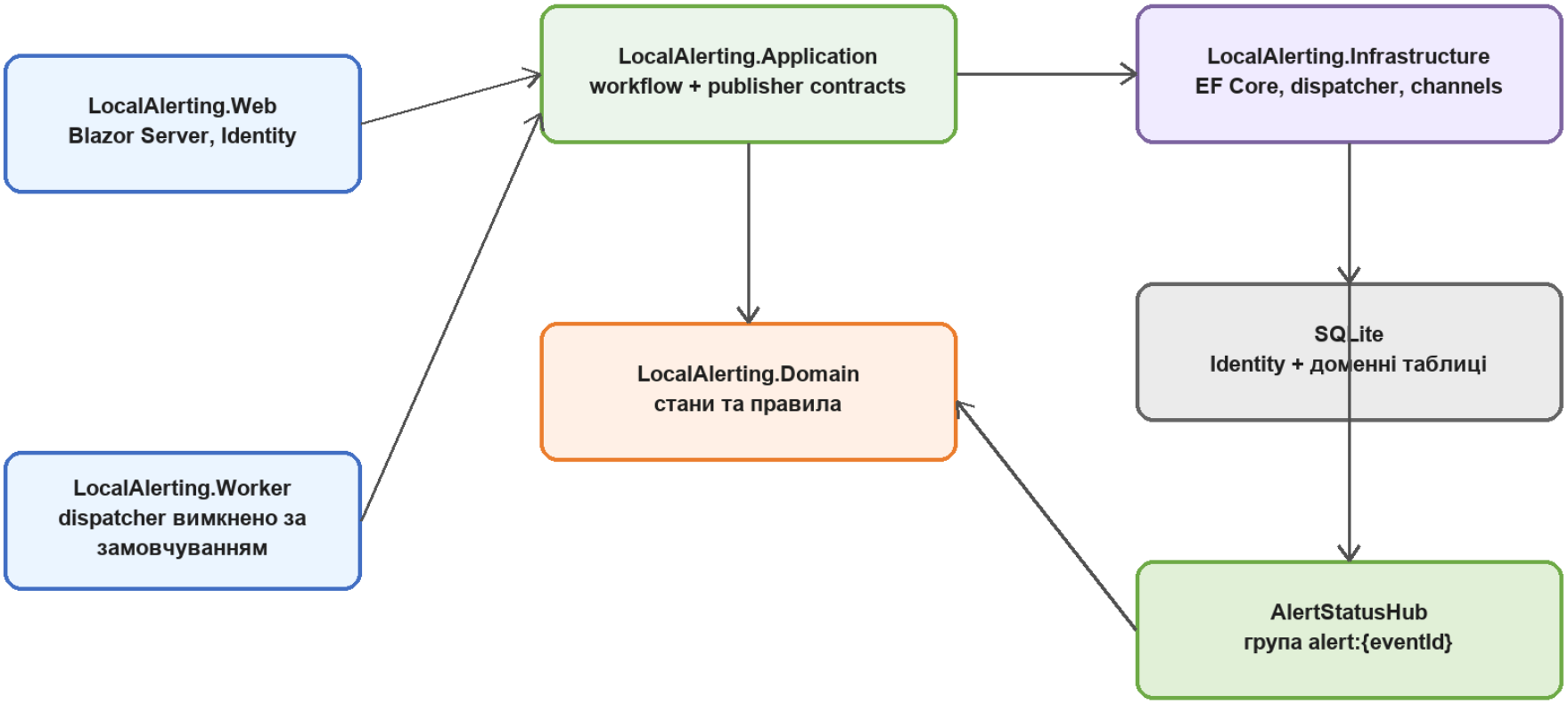

Рисунок Б.2 – Компонентна архітектура фактичного прототипу

Б.3 Діаграма класів доменної моделі

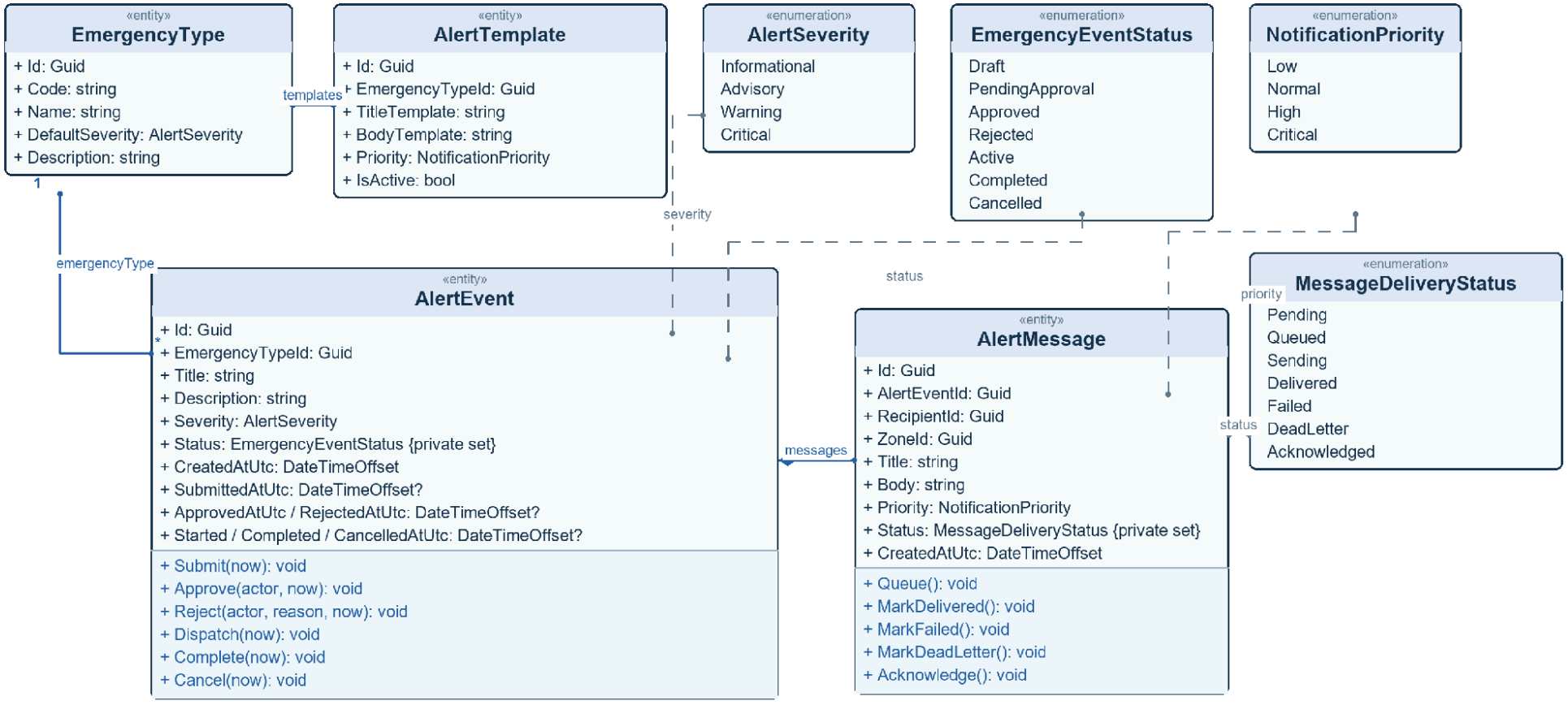

Рисунок Б.3 – Основні сутності фактичної доменної моделі

Б.4 Діаграма послідовності запуску оповіщення

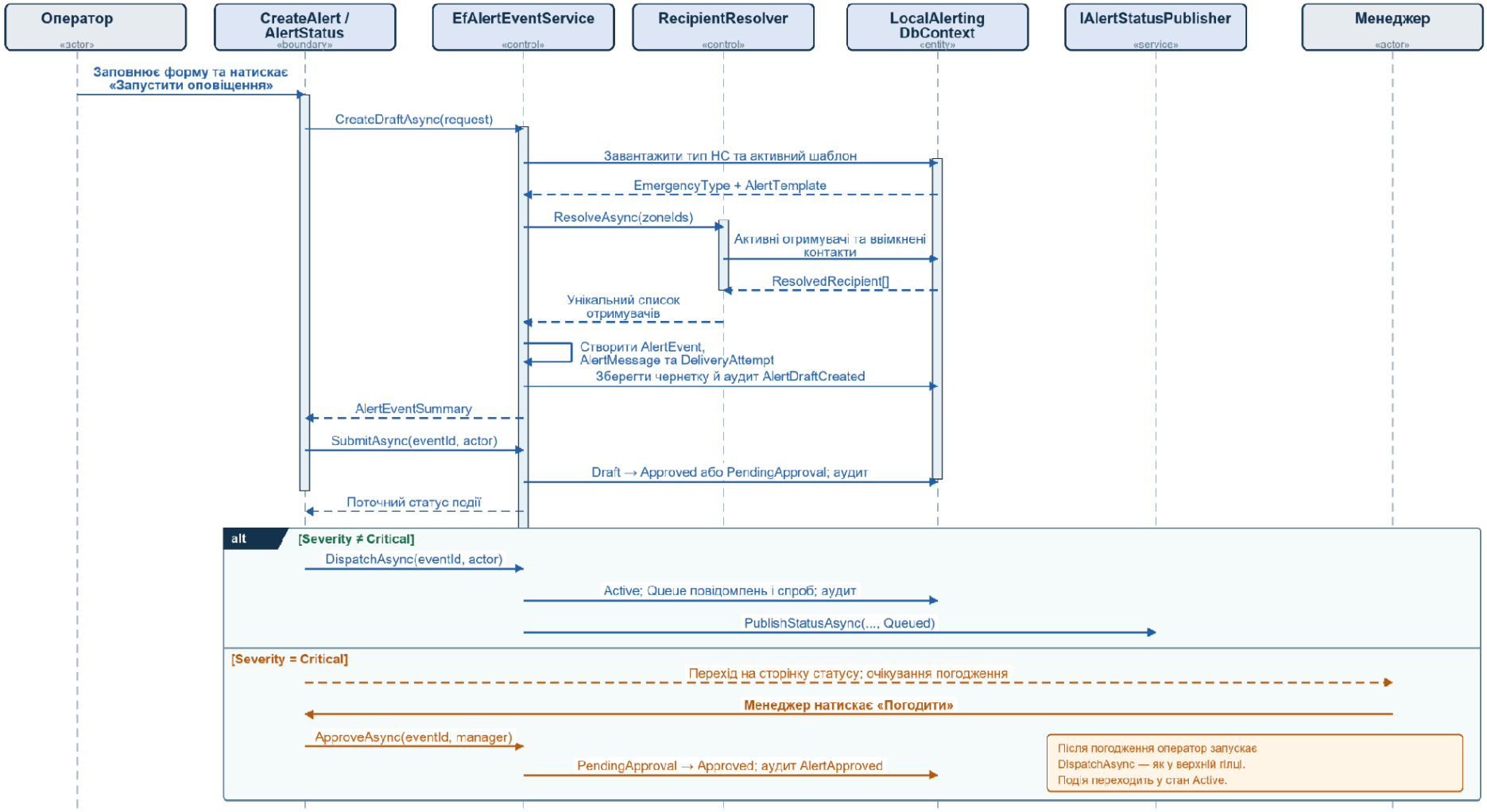

Рисунок Б.4 – Послідовність створення і запуску події

Б.5 Діаграма станів події НС

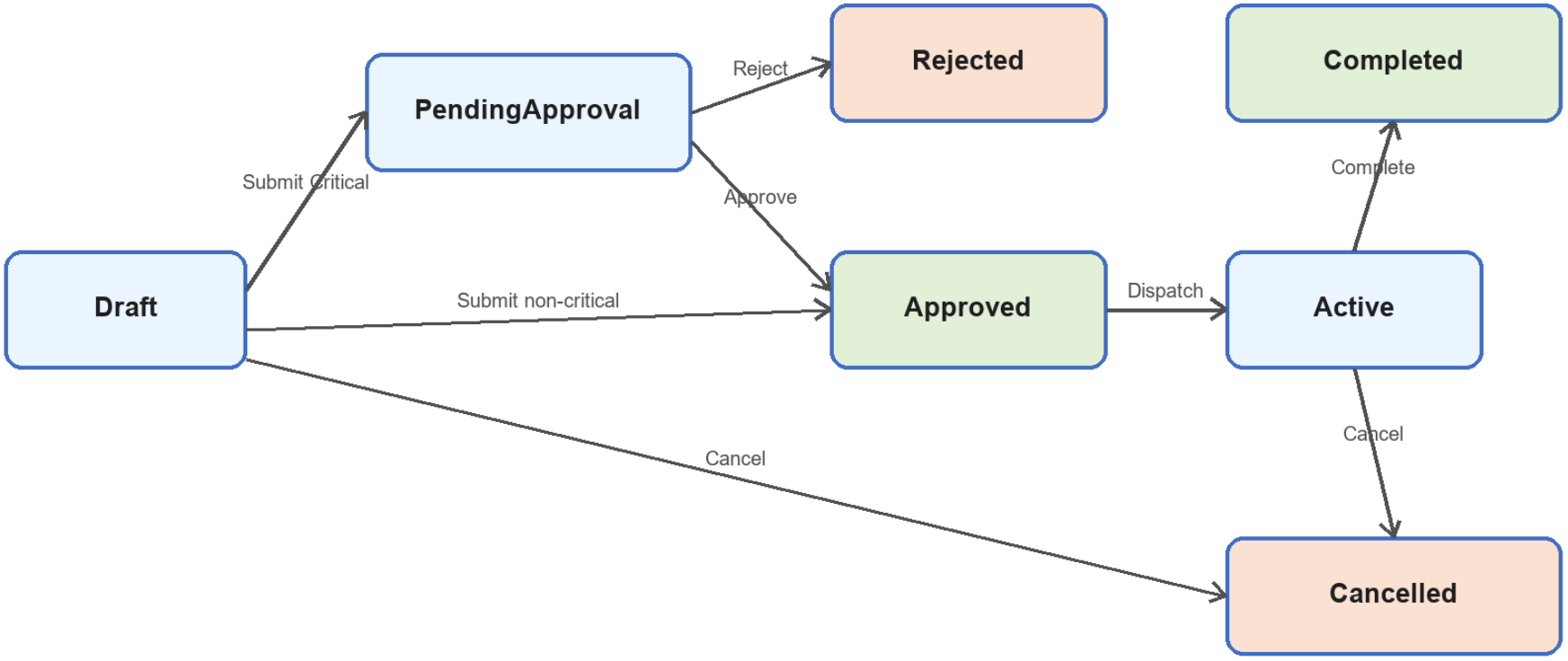

Рисунок Б.5 – Фактичні стани події

ДОДАТОК В

ER-ДІАГРАМА БАЗИ ДАНИХ

ER-діаграма відображає проєктну модель даних прототипу, основні сутності предметної області та зв’язки між подіями, повідомленнями, отримувачами, каналами доставки й аудитом.

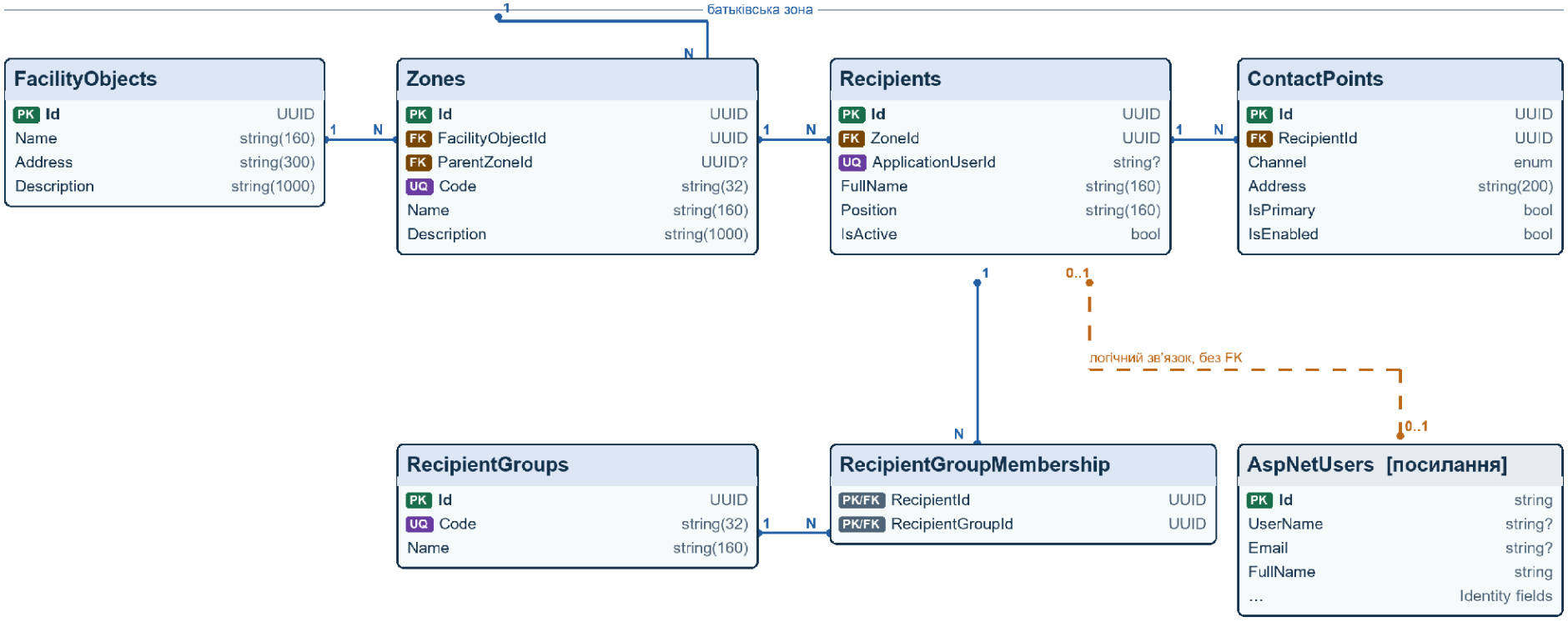

Рисунок В.1 – ER-діаграма фактичної моделі дани

Таблиця В.1 – Основні проєктні таблиці бази даних

Таблиця / набір	**Фактичне призначення**
FacilityObjects	Об’єкти системи
Zones	Зони об’єкта
Recipients	Отримувачі
RecipientGroups	Групи
RecipientGroupMemberships	Зв’язок отримувачів і груп
ContactPoints	Контактні точки
EmergencyTypes	Типи НС
AlertTemplates	Шаблони
AlertEvents	Події
AlertMessages	Повідомлення
DeliveryAttempts	Спроби доставки
Acknowledgements	Підтвердження
AuditLogs	Журнал аудиту
ASP.NET Core Identity tables	Користувачі, ролі, claims, logins, tokens

ДОДАТОК Г

ТЕСТ-КЕЙСИ ТА РЕЗУЛЬТАТИ ТЕСТУВАННЯ

Автоматизований набір після покращень містить 16 модульних і 6 інтеграційних тестів, разом 22 успішні результати dotnet test. Ручні та заплановані перевірки наведено окремо. Фактичні ролі: Administrator, Manager, Operator, Recipient, TechnicalSpecialist, SecurityOfficer та Auditor.

Матриця нижче фіксує набір сценаріїв, за якими перевіряється відповідність прототипу функціональним, інтеграційним і безпековим вимогам.

Таблиця Г.1 – Автоматизовані модульні тести

ID	Тип	Передумова	Дія	Фактично	Статус
1	2	3	4	5	6
UT-01	Unit	Підготовлені дані	Submit_NonCritical_ApprovesWithoutManager	Правило виконано	Passed
UT-02	Unit	Підготовлені дані	Submit_Critical_RequiresManagerApproval	Правило виконано	Passed
UT-03	Unit	Підготовлені дані	Approve_PendingCritical_AllowsDispatch	Правило виконано	Passed
UT-04	Unit	Підготовлені дані	Reject_RequiresReason	Правило виконано	Passed
UT-05	Unit	Підготовлені дані	Reject_PendingCritical_IsTerminal	Правило виконано	Passed
UT-06	Unit	Підготовлені дані	Complete_WhenDraft_Throws	Правило виконано	Passed
UT-07	Unit	Підготовлені дані	Cancel_WhenCompleted_Throws	Правило виконано	Passed
UT-08	Unit	Підготовлені дані	MarkFailed_BeforeMaxAttempts_SchedulesRetry	Правило виконано	Passed
UT-09	Unit	Підготовлені дані	MarkFailed_AtMaxAttempts_MovesToDeadLetter	Правило виконано	Passed
UT-10	Unit	Підготовлені дані	ResolveAsync_DeduplicatesRecipients	Правило виконано	Passed

Продовження таблиці Г.1

1	2	3	4	5	6
UT-11	Unit	Підготовлені дані	ResolveAsync_WhenNoZones_ReturnsEmptyResult	Правило виконано	Passed
UT-12	Unit	Підготовлені дані	Acknowledge_OwnMessage_UsesRecipientIdentity	Правило виконано	Passed
UT-13	Unit	Підготовлені дані	Acknowledge_ForeignMessage_IsRejected	Правило виконано	Passed
UT-14	Unit	Підготовлені дані	AuditLog_Modification_IsRejected	Правило виконано	Passed
UT-15	Unit	Підготовлені дані	CriticalWorkflow_RequiresApproval_ThenQueuesMessages	Правило виконано	Passed
UT-16	Unit	Підготовлені дані	SignalRChannel_PublishesNotification	Правило виконано	Passed

Таблиця Г.2 – Автоматизовані інтеграційні тести

ID	Тип	Передумова	Дія	Фактично	Статус
IT-01	Integration	Test host	Health_ReturnsOk	Correct	Passed
IT-02	Integration	Test host	AlertSummary_RequiresAuthentication	Correct	Passed
IT-03	Integration	Test host	RBAC CanLaunchAlerts/Operator	Correct	Passed
IT-04	Integration	Test host	RBAC CanApproveAlerts/Manager	Correct	Passed
IT-05	Integration	Test host	RBAC Dashboard/TechnicalSpecialist	Correct	Passed
IT-06	Integration	Test host	RBAC Audit/Auditor	Correct	Passed

ДОДАТОК Д

ФРАГМЕНТИ ПРОГРАМНОГО КОДУ ОСНОВНИХ МОДУЛІВ

Фрагменти коду в цьому додатку наведені для ілюстрації основних модулів прототипу: доменної моделі, статусів повідомлень, каналів доставки, фонової обробки, SignalR-оновлень, політик авторизації та аудиту.

Лістинг Д.1 – Клас AlertEvent

```
public sealed class AlertEvent
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public Guid EmergencyTypeId { get; set; }
    public EmergencyType? EmergencyType { get; set; }
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public AlertSeverity Severity { get; set; } =
AlertSeverity.Warning;
    public EmergencyEventStatus Status { get; private set; } =
EmergencyEventStatus.Draft;
    public DateTimeOffset CreatedAtUtc { get; set; } =
DateTimeOffset.UtcNow;
    public DateTimeOffset? SubmittedAtUtc { get; private set; }
    public DateTimeOffset? ApprovedAtUtc { get; private set; }
    public DateTimeOffset? RejectedAtUtc { get; private set; }
    public string? ApprovedBy { get; private set; }
    public string? RejectedBy { get; private set; }
    public string? RejectionReason { get; private set; }
    public DateTimeOffset? StartedAtUtc { get; private set; }
    public DateTimeOffset? CompletedAtUtc { get; private set; }
    public DateTimeOffset? CancelledAtUtc { get; private set; }
    public List<AlertMessage> Messages { get; set; } = [];

    public void Submit(DateTimeOffset now)
    {
        if (Status != EmergencyEventStatus.Draft)
        {
            throw new InvalidOperationException("Only draft alert
events can be submitted.");
        }

        SubmittedAtUtc = now;
        Status = Severity == AlertSeverity.Critical
            ? EmergencyEventStatus.PendingApproval
            : EmergencyEventStatus.Approved;
    }
```

```
    public void Approve(string actor, DateTimeOffset now)
    {
        if (Status != EmergencyEventStatus.PendingApproval)
        {
            throw new InvalidOperationException("Only pending alert
events can be approved.");
        }

        Status = EmergencyEventStatus.Approved;
        ApprovedBy = actor;
        ApprovedAtUtc = now;
    }

    public void Reject(string actor, string reason, DateTimeOffset
now)
    {
        if (Status != EmergencyEventStatus.PendingApproval)
        {
            throw new InvalidOperationException("Only pending alert
events can be rejected.");
        }

        if (string.IsNullOrWhiteSpace(reason))
        {
            throw new ArgumentException("A rejection reason is
required.", nameof(reason));
        }

        Status = EmergencyEventStatus.Rejected;
        RejectedBy = actor;
        RejectionReason = reason.Trim();
        RejectedAtUtc = now;
    }

    public void Dispatch(DateTimeOffset now)
    {
        if (Status != EmergencyEventStatus.Approved)
        {
            throw new InvalidOperationException("Only approved alert
events can be dispatched.");
        }

        Status = EmergencyEventStatus.Active;
        StartedAtUtc = now;
    }

    public void Complete(DateTimeOffset now)
    {
        if (Status != EmergencyEventStatus.Active)
        {
            throw new InvalidOperationException("Only active alert
events can be completed.");
        }
```

```
        Status = EmergencyEventStatus.Completed;
        CompletedAtUtc = now;
    }

    public void Cancel(DateTimeOffset now)
    {
        if (Status is EmergencyEventStatus.Completed or
EmergencyEventStatus.Cancelled or EmergencyEventStatus.Rejected)
        {
            throw new InvalidOperationException("Completed,
cancelled, or rejected alert events cannot be cancelled.");
        }

        Status = EmergencyEventStatus.Cancelled;
        CancelledAtUtc = now;
    }
}
```

Лістинг Д.2 – Перелік статусів повідомлення

```
public enum MessageDeliveryStatus
{
    Pending = 0,
    Queued = 1,
    Sending = 2,
    Delivered = 3,
    Failed = 4,
    DeadLetter = 5,
    Acknowledged = 6
}
```

Лістинг Д.3 – Інтерфейс каналу доставки

```
public interface INotificationChannel
{
    NotificationChannel Channel { get; }
    Task<DeliveryResult> SendAsync(
        AlertMessage message,
        ContactPoint contactPoint,
        CancellationToken cancellationToken = default);
}
```

Лістинг Д.4 – Основний цикл фонового диспетчера

```
var candidateAttempts = await dbContext.DeliveryAttempts
    .Include(x => x.AlertMessage)
    .Include(x => x.ContactPoint)
    .Where(x => x.Status == MessageDeliveryStatus.Queued ||
                x.Status == MessageDeliveryStatus.Failed)
    .ToListAsync(cancellationToken);

var attempts = candidateAttempts
    .Where(x => x.IsDue(now))
    .OrderBy(x => x.CreatedAtUtc)
    .Take(25)
    .ToArray();

foreach (var attempt in attempts)
{
    await DispatchAttemptAsync(dbContext, attempt, retryPolicy,
cancellationToken);
}
```

Лістинг Д.5 – SignalRNotificationChannel, AlertStatusHub та publisher

```
public sealed class SignalRNotificationChannel(IAlertStatusPublisher
publisher)
    : INotificationChannel
{
    public NotificationChannel Channel =>
NotificationChannel.SignalR;

    public async Task<DeliveryResult> SendAsync(
        AlertMessage message, ContactPoint contactPoint,
        CancellationToken cancellationToken = default)
    {
        await publisher.PublishNotificationAsync(
            message.AlertEventId, message.Id, message.Title,
message.Body,
            cancellationToken);
        return DeliveryResult.Success(
            $"SignalR notification published for
{contactPoint.Address}.");
    }
}

[Authorize(Policy = "CanViewAlertDashboard")]
public sealed class AlertStatusHub : Hub
{
    public Task JoinEventGroup(Guid alertEventId) =>
        Groups.AddToGroupAsync(Context.ConnectionId,
$"alert:{alertEventId:N}");
}
```

```
public sealed class
SignalRAlertStatusPublisher(IHubContext<AlertStatusHub> hubContext)
    : IAlertStatusPublisher
{
    public Task PublishStatusAsync(Guid eventId, Guid messageId,
        MessageDeliveryStatus status, CancellationToken token =
default) =>
        hubContext.Clients.Group($"alert:{eventId:N}")
            .SendAsync("MessageStatusChanged", messageId,
status.ToString(), token);
}
```

Лістинг Д.6 – Політики рольового доступу

```
CanLaunchAlerts: Administrator, Operator; CanApproveAlerts: Manager;
CanViewAlertDashboard: Administrator, Manager, Operator,
TechnicalSpecialist, SecurityOfficer, Auditor; CanViewAudit:
Administrator, Manager, Operator, TechnicalSpecialist,
SecurityOfficer, Auditor; CanManageDictionaries: Administrator.
```

Лістинг Д.7 – Структура запису журналу аудиту

```
public sealed class AuditLog
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public DateTimeOffset CreatedAtUtc { get; set; } =
DateTimeOffset.UtcNow;
    public string Actor { get; set; } = string.Empty;
    public string Action { get; set; } = string.Empty;
    public string EntityType { get; set; } = string.Empty;
    public string EntityId { get; set; } = string.Empty;
    public string Details { get; set; } = string.Empty;
}
```

ДОДАТОК Е

ДИСК АБО ЕЛЕКТРОННІ МАТЕРІАЛИ З РОБОТОЮ