

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

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»**

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: заочна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення інформаційної управляючої системи для автоматизації
процесу зовнішніх нотіфікацій»

здобувача Степаненка Миколая Ігоровича
(ПІБ, підпис)

Науковий керівник: д.е.н., проф. Устенко Станіслав Веніамінович
(науковий ступінь, учене звання, ПІБ)

(підпис)

**Робота допущена до захисту перед екзаменаційною
комісією з атестації здобувачів вищої освіти (ЕК)**

Завідувач кафедри: к.е.н., доцент Тішков Б.О. _____

(підпис)

Київ 2025

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

**Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці**

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА

«Інформаційні управляючі системи і технології»

галузь знань

12 Інформаційні технології

спеціальність

122 Комп'ютерні науки

ПОГОДЖЕНО:

Керівник проєктної групи (гарант)
освітньо-професійної програми

ЗАТВЕРДЖУЮ:

Завідувач кафедри інформаційних
систем в економіці

_____ Устенко С.В.
« ____ » _____ 202_ р.

_____ Тішков Б.О.
« ____ » _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Степаненка Миколая Ігоровича

(прізвище, ім'я, по батькові)

заочної форми навчання

очної (денної), заочної, дистанційної

на підготовку кваліфікаційної магістерської роботи

на тему: «Розроблення інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій»

Тему затверджено наказом ректора Університету від «__» _____ 202_ р. № _____

Кваліфікаційна магістерська робота виконується на матеріалах

План кваліфікаційної магістерської роботи

Розділ I «Дослідження та аналіз підходів до створення інформаційних управляючих систем предметної області»

(назва розділу)

Розділ II «Характеристика інформаційних управляючих систем та методи і моделі»

(назва розділу)

Розділ III «Розроблення проєктних рішень та їх реалізація»

(назва розділу)

Об'єкт дослідження: кваліфікаційної магістерської роботи є процес розробки інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій.

Предмет дослідження: є сукупність теоретичних, методичних і практичних підходів до технології проектування управляючої інформаційної системи

Мета кваліфікаційної магістерської роботи: розробка інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I Необхідно здійснити всебічне дослідження відповідної предметної області, в межах якого визначаються її ключові переваги та наявні проблеми, з виявленням причин їх виникнення та можливих наслідків. Доцільно провести порівняльний аналіз існуючих аналогів інформаційно-управляючих систем з автоматизації процесу зовнішніх нотифікацій, встановити їх сильні та слабкі сторони, а також сформулювати обґрунтовані висновки за результатами аналізу. На основі отриманих даних слід обґрунтувати вибір підходів і методів проектування інформаційно-управляючої системи з автоматизації процесу зовнішніх нотифікацій, із детальним описом кожного підходу, аналізом його переваг і недоліків та формуванням відповідних висновків.

У розділі II необхідно описати інформаційну управляючу систему сповіщень, визначити її мету та основні функції. Слід охарактеризувати архітектуру інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій. Визначити структуру та представити характеристику інформаційної управляючої системи. Описати методи та моделі інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій.

У розділі III У межах розробки інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій необхідно сформулювати проектні рішення та реалізувати їх. Слід спроектувати базу даних для зберігання інформації про повідомлення, користувачів і події, описати сутності, їх атрибути, створити інфологічну та фізичну модель БД із характеристиками кожного атрибута. Варто виконати тестування бази даних з урахуванням вимог до тестових даних. Потрібно обґрунтувати вибір архітектури, технологій і середовища реалізації системи, визначити необхідне програмне та технічне забезпечення. Також слід протестувати готову систему, результати подати у вигляді скріншотів у додатках. Провести аналіз процесу тестування системи. На захисті КМР продемонструвати роботу спроектованої ІУС. Окрему увагу приділити техніко-економічному обґрунтуванню, оцінці ризиків, а також надати рекомендації та висновки на основі проведених досліджень і реалізації підсистеми. Надати висновки, в яких описати основні результати дослідження ІУС, прототипу, подальші дослідження з даною темою.

**Завдання підготував
науковий керівник**

_____ (підпис)

_____ (ініціали, прізвище)

« ____ » _____ 202_ р.

**Завдання одержав
здобувач**

_____ (підпис)

_____ (ініціали, прізвище)

« ____ » _____ 202_ р.

Реферат

Кваліфікаційна магістерська робота містить 116 сторінок, 6 таблиць, 38 рисунків, список використаних джерел з 51 найменувань, 1 додаток.

«Розроблення інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій»

Об'єктом дослідження кваліфікаційної магістерської роботи є процес розробки інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій.

Предметом дослідження є сукупність теоретичних, методичних і практичних підходів до технології проєктування управляючої інформаційної системи

Мета і завдання дослідження. Розробка інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій.

Відповідно до поставленої мети визначені такі *завдання*:

- дослідити предметну область, виконати аналіз існуючих систем обробки та обміну інвойсів та е-інвойсів;

- представити обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій;

- визначити структуру та представити характеристику інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій;

- описати методи та моделі в інформаційній управляючій системі з автоматизації процесу зовнішніх нотифікацій;

- спроектувати базу даних для інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій;

- описати програмне забезпечення для інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій;

- описати технічне забезпечення для інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій;

- представити результати реалізації інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій.

Теоретична, методична та практична значущість отриманих результатів. У ході дослідження було вивчено предметну область систем нотифікацій, проаналізовано існуючі підходи до зберігання та обробки повідомлень, зокрема у вигляді логів, email-розсилок, push- та SMS-сповіщень. Розглянуто формати зберігання даних, способи доступу до них та канали доставки повідомлень користувачам. Визначено потенціал покращення процесів інформування користувачів. Практичні результати дослідження полягають у підвищенні оперативності комунікації, автоматизації поширення інформації серед користувачів і зменшенні людських помилок, що загалом сприяє ефективнішій роботі інформаційної управляючої системи. Впровадження системи нотифікації є логічною та важливою складовою інформаційної управляючої системи (ІУС), оскільки забезпечує оперативну, цільову та автоматизовану комунікацію між системою та її користувачами або іншими зовнішніми учасниками. Система нотифікацій виконує роль інструмента зворотного зв'язку, інформування та реакції на події, що відбуваються всередині бізнес-процесів, які управляються ІУС. З технічної точки зору, нотифікації дозволяють передавати ключові події у реальному часі — наприклад, зміну статусу замовлення, повідомлення про збої, нагадування, підтвердження або оповіщення про

критичні події. Це підвищує оперативність прийняття рішень, знижує ймовірність людських помилок та покращує рівень обслуговування. З управлінської точки зору, система нотифікацій підсилює функціональність ІУС у частині контролю, моніторингу та координації дій. Вона створює прозорий інформаційний канал, через який користувачі (керівники, співробітники, клієнти або партнери) отримують актуальні дані для своєчасної взаємодії з системою. При цьому, інформаційно управляюча система, обирає найоптимальніший канал доставки сповіщень.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: сповіщення, нотифікація, черга сповіщень, брокер повідомлень, провайдер.

Відгук

про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в економіці»

освітньо-професійної програми «Інформаційні управляючі системи і технології»

Степаненка Миколая Ігоровича

(прізвище, ім'я, по батькові)

на тему «Розроблення інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій»

1. Актуальність теми: обумовлена зростаючою потребою у підвищенні оперативності, надійності та автоматизації нотифікацій у сучасних інформаційних управляючих системах. Розробка такої ІУС є відповіддю на виклики бізнесу щодо своєчасного інформування користувачів і контрагентів, що, у свою чергу, сприяє підвищенню ефективності управління та якості обслуговування.
2. Позитивні риси кваліфікаційної магістерської роботи: У даній роботі було здійснено ґрунтовний аналіз і систематизацію інформації щодо побудови системи автоматизованих нотифікацій, із залученням сучасних теоретичних підходів і прикладів практичного впровадження в інформаційно-керованих середовищах. Такий підхід забезпечує наукову обґрунтованість та методологічну коректність дослідження, що підтверджує його відповідність вимогам до прикладних інформаційних систем.
3. Наявність самостійних розробок автора: У роботі представлені самостійно розроблені рішення, зокрема логіка вибору каналу нотифікацій. Ці розробки враховують особливості предметної області та адаптовані під реальні сценарії використання.
4. Цінність теоретичних висновків та практичних рекомендацій: Теоретичні висновки дають чітке розуміння побудови ІУС з автоматизації процесу нотифікацій. Практичні рекомендації дозволяють ефективно впровадити її в реальних умовах.
5. Наявність недоліків: недоліків не виявлено.
6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: _____

Науковий керівник _____

(посада, учене звання, науковий ступінь)

(підпис)

(прізвище, ініціали)

«__» _____ 20__ р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти

Степаненка Миколая Ігоровича

(прізвище, ім'я, по батькові)

Тема Розроблення інформаційної управляючої системи для автоматизації
процесу зовнішніх нотифікацій

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення: Актуальність теми зумовлена зростанням потреби бізнесу в ефективній комунікації з користувачами. Розробка ІУС з автоматизації процесу нотифікацій є доцільною, оскільки дозволяє автоматизувати сповіщення через різні канали та підвищити якість сервісу.

Якість проведеного дослідження: Дослідження виконано на високому рівні з урахуванням сучасних підходів до побудови інформаційних систем. У роботі використано актуальні теоретичні знання та практичні інструменти, що забезпечують її наукову та прикладну цінність.

Позитивні риси кваліфікаційної магістерської роботи Робота вирізняється логічною структурою, чіткою постановкою задач та наявністю самостійних технічних рішень. Вона орієнтована на практичне застосування, що підвищує її цінність для реального впровадження.

Зауваження робота виконана якісно. Зауваження відсутні.

Практична значимість висновків і рекомендацій Практична значимість полягає в підвищенні ефективності ІУС з автоматизації процесу нотифікацій, що сприяє оперативному інформуванню користувачів і покращенню їх взаємодії з додатком. Впровадження запропонованих рішень дозволить знизити затримки в доставці повідомлень та підвищити рівень задоволеності користувачів.

Місце роботи та посада рецензента ФОП „Степаненко Миколай Ігорович“

Науковий ступінь, учене звання (за наявності) Степаненко М.І.

(підпис, ПІБ)

Підпис засвідчую: Software Engineer PHP Степаненко М.І.
(посада, підпис)

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Аналіз існуючих інформаційних систем	12
1.1.1 Дослідження предметної області.....	12
1.1.2 Дослідження інформаційних управляючих систем.....	13
1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи	24
РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ.....	26
2.1 Структура і характеристика системи	26
2.2 Методи та моделі в інформаційних управляючих системах і технологіях..	39
РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ.....	47
3.1. Проєктування бази даних та/або сховища даних для інформаційної управляючої системи	47
3.1.1. Проєктування інфологічної (логічної) моделі БД.....	48
3.1.2. Обґрунтування вибору СКБД та проєктування фізичної моделі БД в середовищі вибраної СКБД.....	51
3.1.3 Контрольний приклад та обрахунок прогнозних обсягів БД	69
3.2 Проєктування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних.....	72
3.2.1 Реалізація інтелектуального аналізу великих даних з метою наповнення бази знань	72
3.2.2 Проєктування та реалізація бази знань інформаційної управляючої системи.	75
3.3 Програмне забезпечення.....	76
3.4 Технічне забезпечення.....	100
3.4.1 Обґрунтування вибору технічного забезпечення	100
3.4.2 Ресурсні вимоги до технічного забезпечення	103
3.5 Реалізація інформаційної системи.....	104
ВИСНОВКИ.....	110
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	112
ДОДАТКИ.....	117

ВСТУП

Сучасні інформаційні системи [1] стають дедалі складнішими, а вимоги до їхньої оперативності, надійності та взаємодії з користувачами зростають. Однією з ключових функцій, яка забезпечує ефективну комунікацію між системою та її користувачами, є система нотифікацій. Саме завдяки нотифікаціям [2] користувачі своєчасно отримують інформацію про зміни у статусах процесів, важливі події [6], помилки, повідомлення безпеки або інші дії, що вимагають їхньої уваги. У масштабних розподілених системах, сервісах електронної комерції, мобільних додатках, банківських платформах і медичних інформаційних системах автоматизовані нотифікації перетворюються на критично важливу складову.

Актуальність теми. Актуальність теми пояснюється стрімким розвитком цифрових сервісів і все більшим значенням персоналізованої, своєчасної та мультиканальної комунікації з користувачем. У час, коли компанії використовують десятки каналів зв'язку — від електронної пошти до push-повідомлень у мобільних додатках — виникає потреба у централізованій, гнучкій і масштабованій системі, яка могла б автоматизовано надсилати повідомлення відповідно до певних подій або бізнес-правил. У свою чергу, автоматизація нотифікацій не лише підвищує зручність для користувача, а й оптимізує роботу персоналу, скорочує витрати на ручну обробку подій, а також знижує ризик людської помилки.

Метою цієї роботи є розробка інформаційної управляючої системи для автоматизації нотифікацій, яка забезпечувала б можливість надсилання повідомлень через різні канали (email, SMS, push), керування шаблонами повідомлень, масштабування та гнучку інтеграцію з подійними механізмами бізнес-логіки.

Об'єктом дослідження є розробка інформаційної управляючої системи, у якій відбуваються різні типи подій, що потребують сповіщення користувачів або адміністраторів.

Предметом дослідження є методологічні засади побудови систем автоматизації нотифікацій, а також програмні підходи та інженерні рішення до проектування й реалізації подібних систем. Особлива увага приділяється застосуванню сучасних архітектурних моделей, принципів об'єктно-орієнтованого проектування, шаблонів програмування, методів інтеграції зі сторонніми сервісами та технічних засобів забезпечення гнучкості, масштабованості та надійності в роботі системи.

Наукова проблема, що розглядається у цій роботі, полягає в необхідності створення універсального підходу до автоматизації надсилання повідомлень у багатокомпонентних інформаційних системах з урахуванням розмаїття типів подій, каналів комунікації та бізнес-правил. Існуючі системи або надто вузькоспеціалізовані (працюють лише з одним каналом), або складні у налаштуванні й інтеграції, що не дозволяє швидко адаптувати їх до нових вимог, подій чи умов масштабування. Проблема також ускладнюється потребою враховувати контекст повідомлення, його пріоритетність, мову, тип одержувача та можливість відкладеної чи повторної доставки. Отже, наукова суть проблеми полягає в розробці моделі та програмного рішення, яке дозволяє гнучко керувати процесом автоматизованого інформування у динамічному середовищі.

Результати дослідження мають прикладне наукове значення й можуть бути використані під час розробки розподілених та мікросервісних систем, де важливо забезпечити централізоване управління нотифікаціями. Запропоноване рішення рекомендовано для впровадження у сервісах електронної комерції, CRM-системах, платформах онлайн-навчання, системах обліку чи моніторингу, де оперативна комунікація з користувачем є ключовою вимогою. Застосування результатів цієї роботи дозволяє зменшити залежність від сторонніх рішень, поліпшити контроль над логікою повідомлень, забезпечити масштабованість та модульність рішення.

Попри зростаючий інтерес до проблем автоматизації комунікаційних процесів у сфері інформаційних технологій, тема комплексної системи автоматизації нотифікацій ще не отримала належного рівня опрацювання у

спеціалізованій науково-технічній літературі. Більшість наявних публікацій зосереджуються на окремих аспектах, таких як надсилання повідомлень через певний канал (наприклад, email або push), або на інтеграції з конкретними сервісами. Водночас бракує системного підходу до побудови гнучкої, розширюваної архітектури, яка дозволяє об'єднувати різні канали доставки, адаптуватися до подієвих сценаріїв системи та враховувати різноманітність бізнес-логік.

Для досягнення поставленої мети в ході дослідження необхідно вирішити такі завдання:

- вивчити наявні підходи до побудови систем нотифікацій та провести їхній порівняльний аналіз;
- описати типові сценарії використання автоматизованих нотифікацій у сучасних програмних продуктах;
- вивести архітектурну модель системи, що дозволяє реалізовувати гнучке керування повідомленнями;
- розробити прототип системи з можливістю налаштування типів подій, шаблонів повідомлень та інтеграцій з API зовнішніх сервісів;
- провести тестування прототипу в умовах, наближених до реальних сценаріїв використання, та оцінити його працездатність і масштабованість.

Особистий внесок автора полягає у формуванні технічного бачення проєкту, розробці архітектури системи, реалізації ключових компонентів та шаблонів взаємодії з зовнішніми сервісами, а також у тестуванні системи у межах реальних сценаріїв. Усі програмні компоненти, описані в роботі, були створені автором самостійно. Також автором проведено аналіз релевантних досліджень і рішень, на основі чого було запропоновано власну модель побудови автоматизованої системи нотифікацій.

У процесі дослідження для досягнення поставленої мети було використано низку методів, які дозволили всебічно проаналізувати предмет дослідження,

здійснити обґрунтоване проєктування системи та провести її реалізацію з урахуванням практичних вимог.

Перш за все, застосовано аналітичний метод, який дозволив дослідити сучасні рішення та підходи до побудови систем нотифікацій, провести порівняльний аналіз їхніх переваг і недоліків, визначити тенденції розвитку галузі та сформулювати вимоги до майбутньої системи.

Метод моделювання був використаний для побудови архітектурної схеми системи, визначення взаємодії її компонентів, каналів повідомлень та потоків даних. Зокрема, використано елементи об'єктно-орієнтованого моделювання (UML-діаграми [40]), що дозволило формалізувати логіку системи та закласти основу її реалізації.

Проєктний метод застосовувався на етапі розробки програмного прототипу системи. Він передбачав розробку структури коду, шаблонів інтеграції, API-інтерфейсів, механізмів налаштування повідомлень та обробки подій.

Також був використаний емпіричний метод, зокрема — тестування програмного продукту на реальних або наближених до реальних сценаріях використання. Це дозволило оцінити працездатність, масштабованість та ефективність реалізованого рішення.

Висновки, отримані у процесі дослідження, мають як теоретичну, так і прикладну значущість. Теоретична цінність полягає у формалізації принципів побудови системи автоматизації нотифікацій, моделюванні її архітектури та визначенні методологічних підходів до інтеграції з зовнішніми каналами зв'язку. Методична значущість проявляється у використанні сучасних принципів проєктування (зокрема, подієвої моделі, інверсії контролю, шаблонів “спостерігач” [3] і “посередник” [4]), які можуть слугувати основою для подальших досліджень або розробки аналогічних систем у різних галузях.

Практична значимість роботи підтверджується тим, що розроблене рішення може бути безпосередньо застосоване у багатьох сферах — зокрема, в e-commerce, банківських системах, CRM-платформах, сервісах онлайн-запису, обліку чи моніторингу. Система є гнучкою, розширюваною, не прив'язана до одного типу

повідомлень або сервісу доставки, що дозволяє адаптувати її до потреб конкретної організації чи проєкту. Принципові рішення, реалізовані в роботі, є прогресивними та демонструють підхід до побудови універсального повідомлювального ядра, яке легко інтегрується в будь-яке середовище через API.

Інформаційною базою дослідження стали наукові публікації вітчизняних та зарубіжних авторів, присвячені питанням побудови систем автоматизації повідомлень, архітектурі програмних систем, інтеграції з зовнішніми сервісами та розробці масштабованих рішень для обробки подій.

Окрему увагу було приділено технічній документації та API-довідникам популярних сервісів нотифікацій, зокрема Twilio, Firebase Cloud Messaging, OneSignal, Brevo, які активно використовуються в індустрії. Інформацію щодо трендів, потреб користувачів та практичного застосування систем повідомлень було отримано з тематичних звітів аналітичних агентств (Gartner, Statista) та блогів провідних IT-компаній (Google, Amazon, Microsoft), а також з офіційної документації open-source фреймворків, які використовувалися при реалізації рішення.

Крім того, у дослідженні враховувалися статистичні дані, опубліковані на офіційних вебсайтах профільних організацій, пов'язаних із цифровими технологіями та комунікаціями, а також результати опитувань користувачів і розробників, що дозволили краще зрозуміти вимоги до сучасних систем нотифікацій. Уся ця інформація стала основою для побудови теоретичної моделі, формулювання вимог та реалізації практичного рішення в межах дослідження.

Структура кваліфікаційної магістерської роботи побудована таким чином, щоб логічно і послідовно розкрити всі аспекти дослідження – від постановки наукової проблеми до реалізації прикладного рішення.

Робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі висвітлено теоретичні основи теми: проаналізовано поняття та класифікацію нотифікацій, розглянуто сучасні підходи до побудови таких систем, а також проведено огляд і порівняння вже наявних рішень і технологій, які використовуються у цій сфері. Метою цього

розділу є закласти наукову основу та окреслити ключові тенденції, на тлі яких сформульовано завдання дослідження.

Другий розділ присвячено методології проєктування системи. У ньому подано постановку задачі, визначено архітектурні та функціональні вимоги до системи, обґрунтовано вибір інструментів і технологій для її реалізації. Окрему увагу приділено моделюванню структури системи, опису логіки її роботи та проєктуванню основних компонентів.

У третьому розділі представлено результати реалізації програмного рішення, описано технічні деталі розробки, приклади використання, протестовано систему на прикладі реальних сценаріїв. Також у цьому розділі проведено оцінку ефективності роботи системи, її масштабованості та зручності інтеграції з іншими сервісами.

Висновки узагальнюють основні результати дослідження, фіксують ступінь досягнення поставленої мети та реалізації завдань, а також визначають напрямки подальших досліджень і можливості вдосконалення системи.

Список використаних джерел містить перелік наукових праць, технічної документації, статистичних матеріалів та інших ресурсів, які були використані під час підготовки роботи. Додатки включають допоміжні матеріали: діаграми, фрагменти коду, таблиці та приклади конфігурацій, що деталізують окремі технічні аспекти реалізації.

Послідовність структурних елементів зумовлена логікою дослідження – від загального до конкретного, від теорії до практики, що забезпечує цілісне сприйняття теми та результатів проєкту.

Таким чином, дана робота має як теоретичну, так і практичну цінність, оскільки поєднує науковий підхід до організації системи комунікації з реальними задачами розробки програмного забезпечення, а також є не лише спробою вирішити актуальну прикладну проблему, але й робить внесок у наукову дискусію щодо оптимізації комунікацій у складних інформаційних системах.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ

У процесі підготовки основної частини кваліфікаційної магістерської роботи було приділено особливу увагу аналізу теоретичних засад побудови систем автоматизації нотифікацій, а також вивченню сучасного стану питання в науковій літературі та практичній розробці. Теоретичне підґрунтя ґрунтується на концепціях проєктування інформаційних систем, принципах архітектурних рішень (зокрема подієвої архітектури), шаблонах програмування та технологіях інтеграції зовнішніх сервісів.

Нотифікації як механізм зворотного зв'язку між системою та користувачем є ключовим елементом у більшості сучасних цифрових продуктів. Теоретичні основи формуються навколо поняття події в системі, яку необхідно зафіксувати, обробити та відповідно повідомити користувача. У цьому контексті широко застосовуються шаблони програмування «спостерігач», «посередник» та концепція publish-subscribe [5], які дозволяють гнучко організувати логіку розсилки повідомлень залежно від типу події.

Серед існуючих рішень особливу увагу було приділено таким сервісам, як Firebase Cloud Messaging [7], Amazon SNS [8], Twilio, OneSignal [10], SendGrid [11] та Brevo [12]. Ці платформи надають універсальні засоби доставки повідомлень, мають широкий спектр функцій, однак у більшості випадків залишають питання логіки генерації нотифікацій на боці користувача. Саме тому розробка гнучкого нотифікаційного ядра, яке керує логікою виникнення та обробки подій, є актуальним завданням.

У роботі також розглядаються підходи до побудови систем на мікросервісній архітектурі [14], застосування черг повідомлень [15] (наприклад, RabbitMQ [16], Kafka [17], Redis Pub/Sub [18]), що дозволяє досягти високої масштабованості та ізольованості логіки. Усі ці підходи та системи були критично

оцінені, а їхні особливості враховано при формуванні вимог до власного програмного рішення, що розроблялося в межах цього дослідження.

Значення теми кваліфікаційної магістерської роботи полягає в її прямій відповідності сучасним викликам цифрового середовища, де оперативна комунікація між системою та користувачем відіграє ключову роль у забезпеченні якості сервісу, підвищенні користувацького досвіду та оптимізації бізнес-процесів. Усе більше цифрових платформ – від банківських застосунків до систем електронної медицини, e-commerce, CRM [19], ERP [20] – потребують ефективних рішень для інформування своїх користувачів про важливі події в реальному часі.

У такому контексті системи автоматизації нотифікацій виступають не лише як інструмент оперативного інформування, а як повноцінний функціональний компонент [21], що впливає на надійність, прозорість та керованість цифрових продуктів. Гнучкість у виборі каналів зв'язку, можливість адаптації під різні сценарії, інтеграція з іншими модулями системи – все це визначає практичну цінність теми.

Окрім того, значущість дослідження посилюється стрімким розвитком технологій та підвищенням вимог до персоналізації повідомлень, дотримання щодо часу доставки, збереження історії повідомлень і відповідності нормативним стандартам. Автоматизована система, розроблена в межах цієї роботи, дозволяє не лише реалізувати ці вимоги, а й формує основу для подальшого розвитку сервісів – включаючи аналітику повідомлень, зворотний зв'язок та адаптивні сценарії розсилки.

На підставі глибокого аналізу існуючих моделей систем нотифікацій, їхньої архітектури, методів обробки подій та способів доставки повідомлень, було здійснено синтез власної автоматизованої системи, орієнтованої на масштабованість, гнучкість у налаштуванні логіки нотифікацій і простоту інтеграції з іншими програмними продуктами. Основною метою синтезу було створення модульної системи з чітко визначеними зонами відповідальності, яка дозволяє ефективно генерувати, обробляти та доставляти повідомлення користувачам різними каналами.

Розроблена система базується на подієво-орієнтованій архітектурі. В основі її роботи лежить модуль Event Dispatcher [23], який приймає події від зовнішніх систем або внутрішніх джерел. Ці події передаються до компонента Notification, який містить правила трансформації події у повідомлення та приймає рішення щодо його створення. Потім сформовані повідомлення потрапляють до Message Queue, де зберігаються до моменту обробки.

Для забезпечення доставки повідомлень передбачено Delivery Manager, який взаємодіє з каналами комунікації (email, SMS, push, webhooks) через абстрактний інтерфейс. Кожен адаптер є окремим підключеним сервісом, який відповідає за специфіку доставки через обраний канал. Зворотний зв'язок про статуси доставки надходить у Delivery Monitor, який фіксує помилки, повторні спроби та успішність доставлення.

Усі дані про повідомлення, користувачів, налаштування та шаблони зберігаються в Data Storage, який складається з бази даних і сховища шаблонів повідомлень. Для адміністрування системи передбачений Admin Panel, що дозволяє налаштовувати шаблони, вмикати або вимикати канали, а також переглядати аналітику.

Таким чином, структурна схема системи включає наступні логічні компоненти:

- джерело подій (Event Source);
- диспетчер подій (Event Dispatcher);
- механізм формування нотифікацій (Notification Engine);
- черга повідомлень (Message Queue);
- менеджер доставки (Delivery Manager);
- адаптери каналів (Channel Adapters);
- моніторинг доставки (Delivery Monitor);
- база даних та сховище шаблонів (Data Storage);
- інтерфейс адміністратора (Admin Panel).

1.1 Аналіз існуючих інформаційних систем

1.1.1 Дослідження предметної області

Інформаційні технології сьогодні є ключовим драйвером розвитку економіки України [24], а сама ІТ-галузь стабільно залишається однією з найдинамічніших у державі [25]. Протягом останніх десяти років спостерігається сталий ріст експорту ІТ-послуг, що свідчить про високий рівень конкурентоспроможності українських компаній на світовому ринку [26]. Ця сфера охоплює широкий спектр напрямків: від розробки програмного забезпечення до хмарних сервісів, систем штучного інтелекту, автоматизації бізнес-процесів, кібербезпеки та аналітики даних.

Водночас, з розвитком галузі зростає і попит на кваліфікованих спеціалістів, що породжує дефіцит людських ресурсів. Це створює додаткові виклики для компаній у боротьбі за таланти та стимулює постійне оновлення підходів до управління, оптимізації внутрішніх процесів та підвищення ефективності команд. У такому контексті важливу роль відіграють інформаційні управляючі системи, що забезпечують автоматизацію внутрішніх процесів, моніторинг продуктивності, комунікацію між командами та контроль за виконанням завдань.

Предметна область інформаційних управляючих систем з автоматизації процесу зовнішніх нотифікацій охоплює методи, засоби та технології, що забезпечують оперативну, надійну та релевантну доставку повідомлень користувачам у відповідь на певні події в системі або у відповідності до заздалегідь визначених сценаріїв. Вона базується на взаємодії між джерелами подій (системи-ініціатори), брокерами повідомлень (системи маршрутизації), каналами доставки (email, SMS, push, in-app, месенджери тощо) та користувачами або іншими системами-отримувачами.

Автоматизація нотифікацій передбачає інтеграцію інформаційної системи з внутрішніми або зовнішніми сервісами (наприклад, Twilio, Firebase, OneSignal, SendGrid) та використання правил, які визначають, кому, коли та яким чином має

бути надіслано повідомлення. Особливу увагу в цій сфері приділяють продуктивності системи, обробці великої кількості подій у режимі реального часу, забезпеченню гарантованої доставки повідомлень, масштабованості, персоналізації контенту повідомлень, а також безпеці та відповідності вимогам конфіденційності.

До основних об'єктів предметної області належать події, типи нотифікацій, шаблони повідомлень, користувачі, їхні налаштування щодо каналів зв'язку, статуси доставки тощо. Важливо також враховувати логіку відкладених або повторних сповіщень, обробку помилок доставки та формування аналітики з ефективності кожного повідомлення або кампанії.

Таким чином, ця предметна область охоплює широкий спектр рішень, що забезпечують швидку реакцію системи на зміни у внутрішньому або зовнішньому середовищі, сприяють покращенню комунікації з користувачем, оптимізації бізнес-процесів та підвищенню загальної ефективності інформаційної системи.

1.1.2 Дослідження інформаційних управляючих систем

У сучасних умовах цифровізації бізнесу та зростаючих вимог до оперативної комунікації з користувачами, особливої актуальності набувають SaaS-рішення (Software as a Service) [27], які дозволяють швидко і з мінімальними витратами інтегрувати механізми нотифікацій у веб- і мобільні додатки. Такі сервіси забезпечують централізовану обробку подій, гнучке налаштування каналів доставки повідомлень (SMS, email, push, webhooks тощо), масштабованість під навантаження, а також аналітику ефективності доставки. У цьому підпункті буде розглянуто найбільш поширені SaaS-системи для автоматизації нотифікацій, їх функціональні можливості, переваги, обмеження та сфери доцільного використання.

Існує низка популярних SaaS-рішень, які спеціалізуються на автоматизації нотифікацій через різні канали зв'язку [39]. Серед найбільш відомих варто виділити такі:

- twilio;
- firebase cloud messaging (fcm);
- onesignal;
- pusher beams;
- amazon simple notification service (sns);
- sendgrid.

Twilio — один із найпопулярніших сервісів, що дозволяє надсилати SMS, голосові повідомлення, push-нотифікації та email. це хмарна платформа комунікацій як послуга (CPaaS), яка надає розробникам API для інтеграції різноманітних каналів зв'язку (SMS, голосові дзвінки, відео, email, WhatsApp, push тощо) безпосередньо у веб- або мобільні додатки. Вона є однією з найпопулярніших у світі і активно використовується для побудови систем сповіщень, автентифікації, контакт-центрів і чат-ботів. Підтримує глобальне покриття і має потужну API-інтеграцію.

Особливістю Twilio є її орієнтація на гнучкість і масштабованість: система надає REST API, які дозволяють керувати повідомленнями, дзвінками чи іншими подіями програмно, без необхідності мати власну інфраструктуру для цього. Крім базових сервісів, платформа включає такі компоненти, як Twilio Studio (візуальний редактор для побудови логіки взаємодії), Twilio Flex (гнучка платформа для створення контакт-центрів), а також можливості для інтеграції зі сторонніми системами CRM або ERP.

На відміну від багатьох вузькоспеціалізованих рішень, Twilio — мультиканальна платформа, яка підтримує інтеграцію одразу кількох каналів в рамках однієї екосистеми. Це робить її особливо корисною для побудови уніфікованих систем нотифікацій.

Серед відмінностей варто виділити високий рівень SLA, потужну документацію, глобальну інфраструктуру з географічно розподіленими серверами, а також моделі ціноутворення на основі використання. Проте до недоліків можна віднести відносно високу вартість у порівнянні з локальними або open-source рішеннями, що може бути критичним для невеликих проєктів. Інформаційна система Twilio наведена на рис. 1.1.

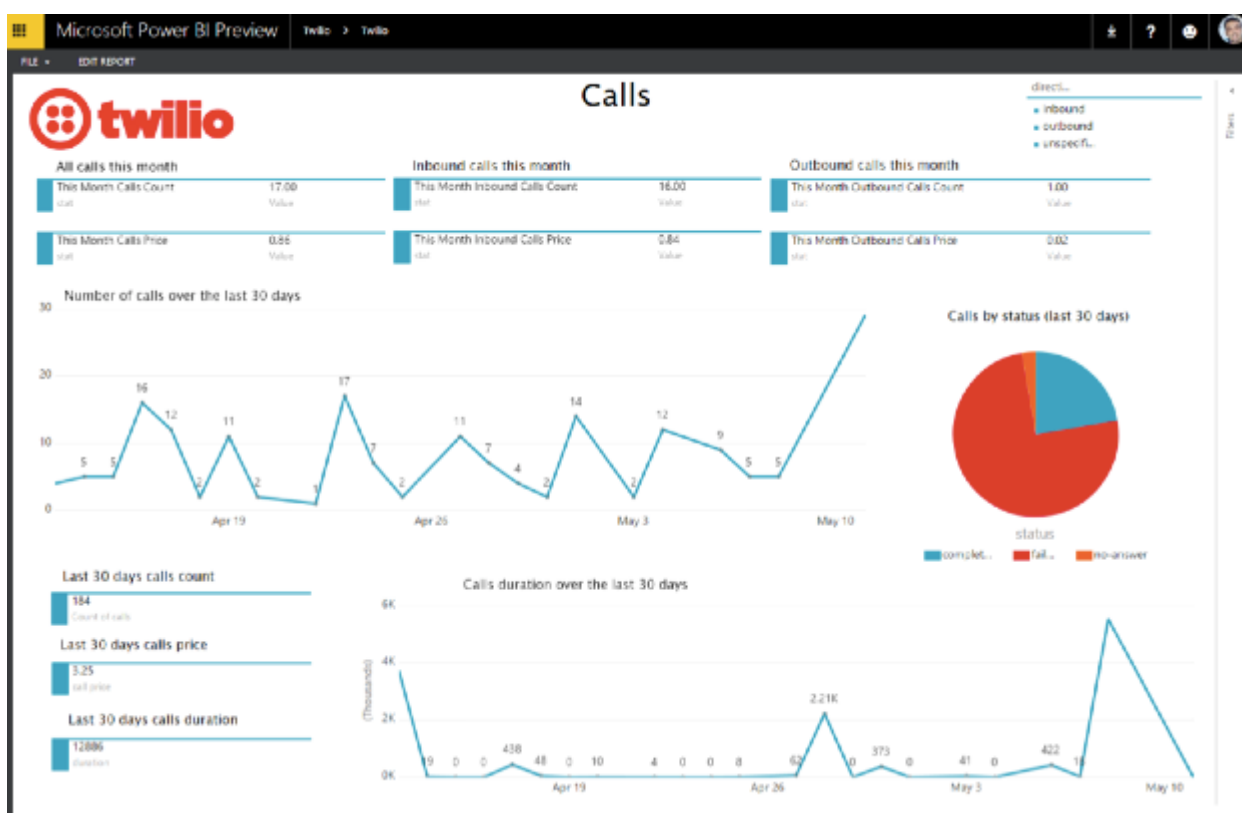


Рисунок 1.1 - Інформаційна система Twilio

Джерело: <https://www.twilio.com/content/dam/twilio-com/global/en/blog/legacy/2015/wilio-monitoring-within-microsoft-power-bi-html/2TwilioMicrosoft.png>

Firestore Cloud Messaging (FCM) — безкоштовний сервіс від Google для надсилання push-нотифікацій на Android, iOS і веб. Широко використовується у мобільних застосунках і легко інтегрується з іншими сервісами Google. Це безкоштовний сервіс від Google, який дозволяє розробникам надсилати push-сповіщення та повідомлення даних на Android, iOS та веб-пристрої. Він є частиною

екосистеми Firebase — платформи для розробки мобільних та вебдодатків, і тісно інтегрується з іншими її сервісами

Основна особливість FCM полягає в простоті інтеграції та орієнтації на мобільні додатки. Для Android-пристроїв він працює особливо ефективно, оскільки є безпосередньо частиною інфраструктури Google Play Services. FCM дозволяє надсилати повідомлення як конкретним пристроям, так і групам, темам або за допомогою користувацьких тегів. Окрім звичайних push-сповіщень, він підтримує "тихі" повідомлення (data-messages), які можуть оброблятися у фоновому режимі.

Важливою перевагою FCM є його безкоштовність, що робить його популярним серед невеликих команд та стартапів. Він також підтримує високу надійність доставки завдяки потужній серверній інфраструктурі Google. Інтеграція з Firebase Analytics дозволяє легко відслідковувати ефективність повідомлень.

Відмінності FCM полягають у фокусі саме на push-повідомленнях, тоді як такі системи, як Twilio, охоплюють ширший спектр каналів (наприклад, SMS, email, голос). Крім того, FCM не надає візуальних інструментів для створення складних сценаріїв сповіщення або управління контакт-центрами. Також обмежено можливість інтеграції з каналами на кшталт Viber або WhatsApp.

Таким чином, FCM — чудове рішення для простого та швидкого додавання push-сповіщень у мобільні додатки, особливо для Android, однак для реалізації мультиканальних систем сповіщення, як-от у корпоративних ІУС, його функціональність може бути недостатньою. Інформаційна система Firebase Cloud Messaging наведена на рис. 1.2.

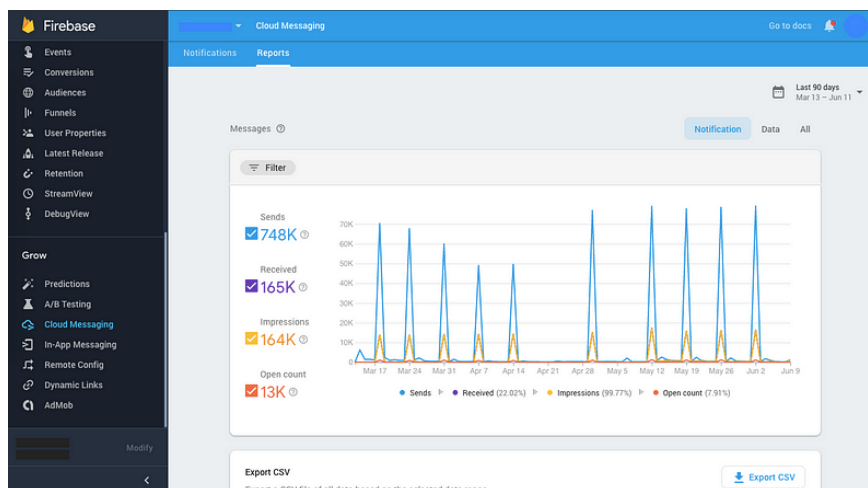


Рисунок 1.2 - Інформаційна система Firebase Cloud Messaging

Джерело: https://cdn-images-1.medium.com/max/800/1*F52mXkjaZkJlhfmocu0Y1w.png

OneSignal — багатоканальний сервіс для відправлення push, email, SMS і in-app повідомлень. Має зручний інтерфейс і можливості для сегментації аудиторії та A/B тестування. Це потужна SaaS-платформа для надсилання push-нотифікацій, email-повідомлень, SMS та in-app повідомлень. Вона орієнтована на розробників мобільних і вебдодатків та маркетингові команди, надаючи зручний інтерфейс, SDK для популярних платформ (Android, iOS, web) і API [38] для інтеграції в бекенд систем.

Ключова особливість OneSignal — це мультиканальність. На відміну від Firebase Cloud Messaging, який зосереджений тільки на push, OneSignal дозволяє надсилати повідомлення по декількох каналах і керувати кампаніями з єдиного інтерфейсу. Це дає змогу реалізовувати складні сценарії взаємодії з користувачем і персоналізовані маркетингові кампанії. Платформа має потужну аналітику, A/B тестування, сегментацію аудиторії, підтримку локалізації та гнучке налаштування тригерів для відправки повідомлень.

Інтеграція OneSignal досить проста, особливо завдяки готовим SDK та документації. Для push-нотифікацій сервіс використовує під капотом Firebase або Apple Push Notification Service, але додає до них додатковий рівень керування, UI, маркетингових функцій та підтримку інших каналів.

Важлива відмінність OneSignal полягає в орієнтації не лише на технічних спеціалістів, а й на маркетинг-команди — вона надає візуальну панель для створення повідомлень, налаштування аудиторій, планування розсилок. У порівнянні з Firebase, який є технічним інструментом, OneSignal — це більш комплексне рішення для взаємодії з користувачами. У порівнянні з Twilio, OneSignal більше сфокусований саме на повідомленнях, тоді як Twilio є універсальною комунікаційною платформою.

Таким чином, OneSignal — це ідеальний вибір для проєктів, де потрібні потужні push-механізми з маркетинговими можливостями, підтримкою різних каналів і гнучкою аналітикою, але він не підходить як універсальний комунікаційний бекенд на рівні інфраструктури, як-от RabbitMQ або Kafka. Інформаційна система OneSignal наведена на рис. 1.3.

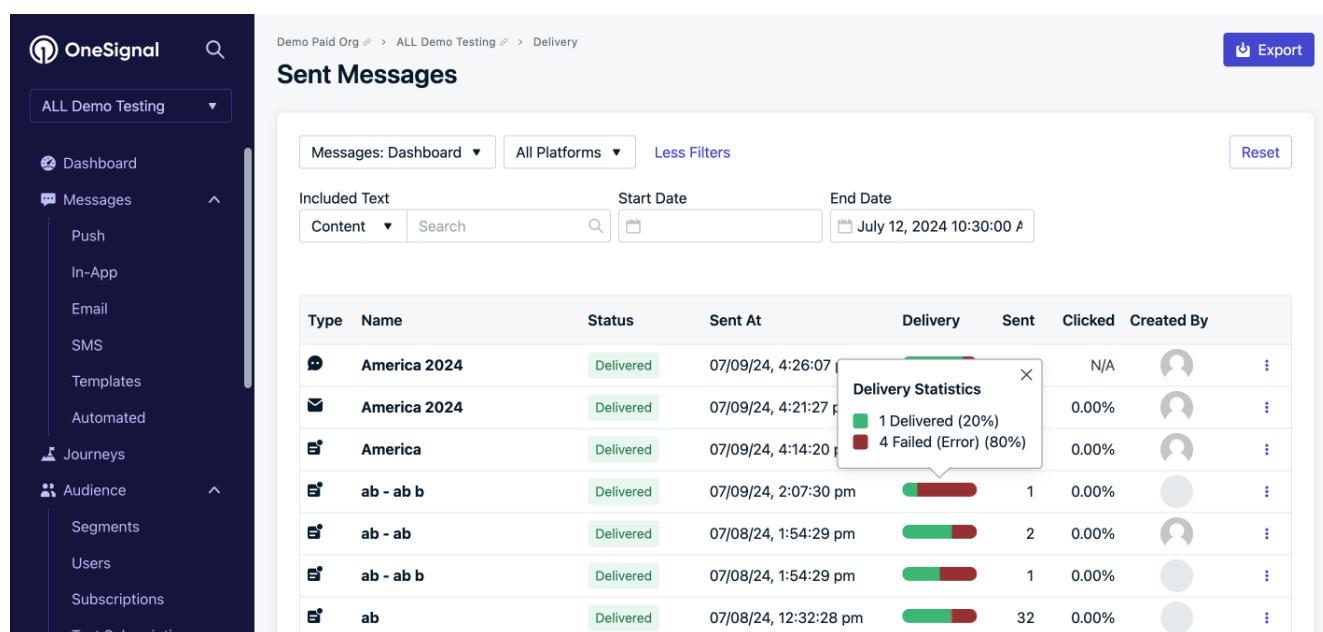


Рисунок 1.3 - Інформаційна система OneSignal

Джерело: https://files.readme.io/d214c95-Sent_message.png

Pusher Beams — спеціалізується на push-повідомленнях у реальному часі, орієнтований на розробників мобільних додатків. Підтримує як персоналізовані, так і широкомовні сповіщення. Це хмарне рішення для надсилання мобільних push-нотифікацій, орієнтоване в першу чергу на персоналізовані та надійні

повідомлення в реальному часі. На відміну від широкого спектру продуктів від Pusher, Beams — це вузькоспеціалізований сервіс, створений виключно для мобільних push-нотифікацій на Android та iOS, з фокусом на transactional notifications (тобто повідомлення, які пов'язані з конкретними діями користувача або подіями в системі).

Основна особливість Pusher Beams полягає в простоті інтеграції та високій доставності повідомлень. Він використовує офіційні канали — Firebase Cloud Messaging (FCM) для Android і Apple Push Notification Service (APNs) для iOS — але надає поверх них додатковий рівень управління, спрощену роботу з користувацькими каналами, а також надійну інфраструктуру для обробки великого потоку подій. Beams підтримує цільову доставку (наприклад, лише одному пристрою конкретного користувача) та дозволяє налаштовувати "interests" — логічні групи підписок.

Відмінність від OneSignal полягає в тому, що Beams не є багатоканальним рішенням. Він не працює з email, SMS або in-app повідомленнями. Його мета — бути максимально надійним та зручним сервісом для саме мобільних push-нотифікацій. Водночас OneSignal надає значно ширший функціонал, але менш вузько сфокусований на якості push-доставки в складних сценаріях.

Порівняно з Firebase Cloud Messaging, Pusher Beams надає значно вищий рівень абстракції та зручності, особливо у випадках, коли потрібно відправляти повідомлення на конкретні пристрої або канали. Він також має хорошу документацію, SDK та панель управління для аналітики.

Таким чином, Pusher Beams — це чудовий вибір для мобільних додатків, яким потрібно надсилати стабільні, персоналізовані push-повідомлення в реальному часі, але не підходить для систем, які потребують багатоканальної комунікації або універсальних повідомлень через email, SMS чи web push. Інформаційна система Pusher Beams наведена на рис. 1.4.

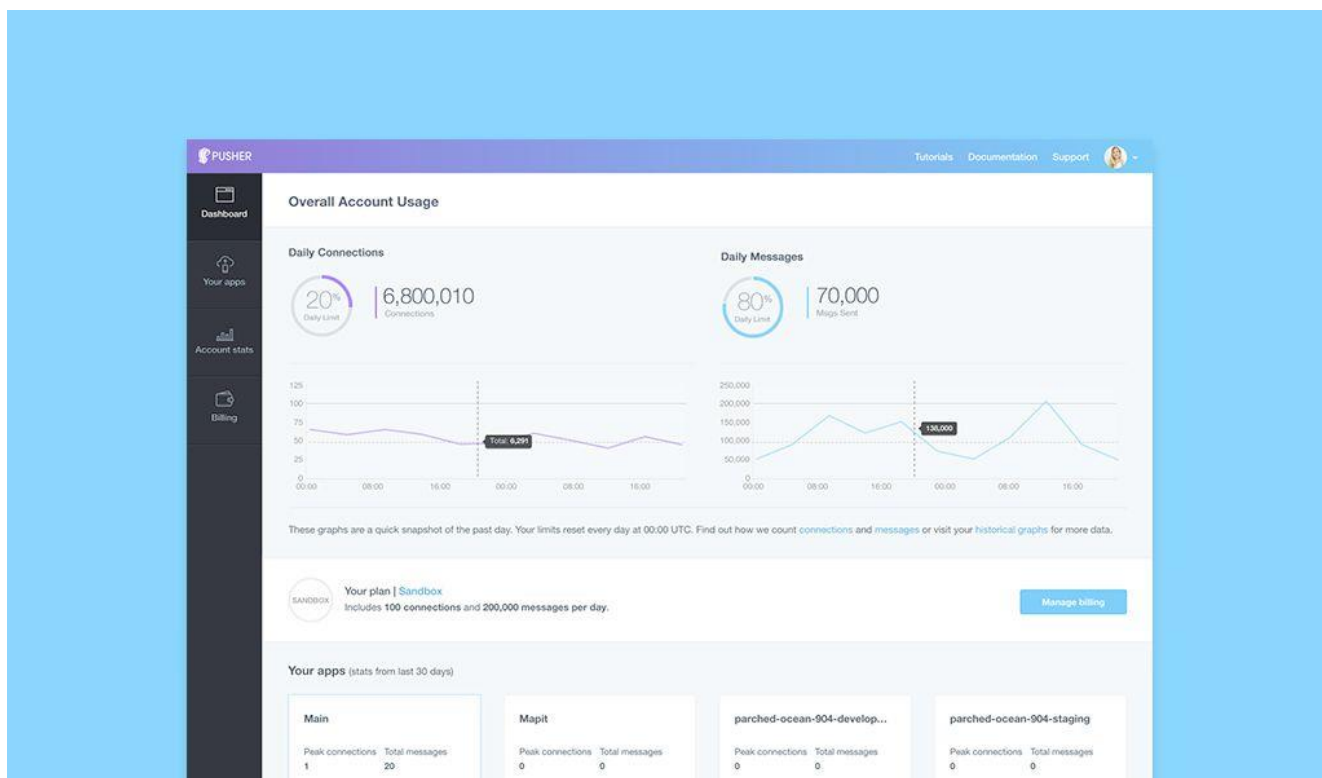


Рисунок 1.4 - Інформаційна система Pusher Breams

Джерело: https://general-pusher-cms.s3.amazonaws.com/blog/new_dashboard_f649c90e4f.jpg

Amazon Simple Notification Service (SNS) — сервіс AWS для масового розсилання повідомлень, що підтримує SMS, email, мобільні push та інші протоколи. Ідеально підходить для масштабних рішень у хмарному середовищі тому, що це керований хмарний сервіс публікації-підписки (pub/sub), який дозволяє масштабовано та асинхронно надсилати повідомлення різним підписникам і системам. Особливістю SNS є його універсальність і глибока інтеграція з іншими сервісами AWS, що робить його ідеальним інструментом для побудови розподілених, подієво-орієнтованих архітектур у хмарі.

Основна відмінність SNS полягає в тому, що це не просто сервіс push-нотифікацій до мобільних пристроїв. SNS дозволяє надсилати повідомлення різними способами: на email, SMS, HTTP/HTTPS endpoints, AWS Lambda, Amazon SQS, а також на мобільні push-платформи. Це надає можливість будувати системи, де повідомлення миттєво надходять до людей (через SMS або email) або запускають логіку в бекенді.

На відміну від більш «фронтенд-орієнтованих» сервісів, таких як OneSignal чи Firebase Cloud Messaging, SNS більше орієнтований на системні та серверні взаємодії. Наприклад, якщо якась подія в системі має викликати повідомлення на кілька підписаних компонентів, SNS ідеально це реалізовує. Він має високий рівень надійності та масштабованості, включаючи гарантовану доставку через повторні спроби та моніторинг з CloudWatch.

У порівнянні з такими рішеннями як Twilio чи Pusher Beams, Amazon SNS не забезпечує "людської" зручності: в нього складніша конфігурація, відсутні шаблони повідомлень або розширена аналітика прямо з коробки. Проте його перевага — в масштабі, гнучкості, безпеці (через IAM політики) та можливості інтегруватися в складні AWS-екосистеми.

Таким чином, Amazon SNS — це потужний вибір для бекенд-систем, що потребують універсального та надійного каналу доставки повідомлень до багатьох споживачів, однак він менш зручний для створення кросплатформних користувацьких push-нотифікацій «з коробки». Інформаційна система Amazon SNS наведена на рис. 1.5.

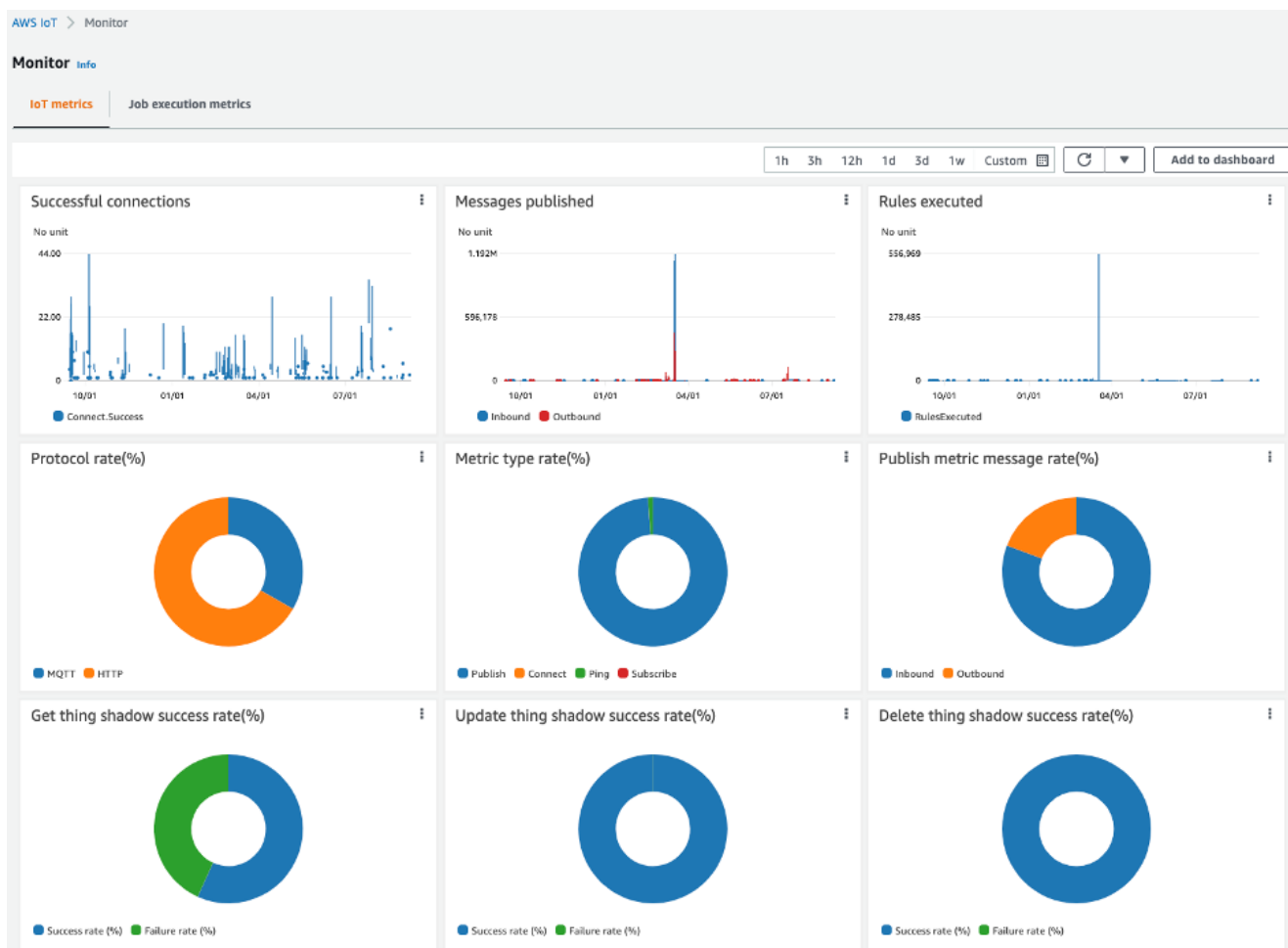


Рисунок 1.5 - Інформаційна система Amazon SNS

Джерело: https://d2908q01vomqb2.cloudfront.net/f6e1126cedebf23e1463aee73f9df08783640400/2022/09/08/iot_monitor_default.png

SendGrid — це хмарний сервіс, який спеціалізується на надсиланні електронної пошти, зокрема транзакційних повідомлень (реєстрації, підтвердження, відновлення паролю) та маркетингових email-кампаній. Його основна мета — забезпечити стабільну, надійну та масштабовану email-інфраструктуру, яку можна легко інтегрувати в будь-який веб-сервіс або мобільний додаток.

Однією з головних особливостей SendGrid є його висока доставлюваність повідомлень, що забезпечується завдяки ретельному контролю за репутацією IP-адрес, підтримці DKIM, SPF та інших механізмів автентифікації електронної пошти. Сервіс також дозволяє створювати й управляти списками підписників,

відстежувати відкриття листів, переходи за посиланнями, відмови та скарги на спам — все це доступно через зручну аналітику.

На відміну від загальних нотифікаційних сервісів, таких як Amazon SNS або Firebase Cloud Messaging, SendGrid спеціалізується виключно на електронній пошті. Це дозволяє йому надавати більш глибокі та професійні інструменти саме для цього каналу: A/B тестування, динамічні шаблони з умовною логікою, сегментацію аудиторії, інструменти для роботи з доставкою великих обсягів повідомлень, ретельний моніторинг активності користувачів.

У порівнянні з такими конкурентами, як Mailgun або Amazon SES, SendGrid виділяється простішим стартом, більш дружнім інтерфейсом і кращою підтримкою саме маркетингових функцій без потреби глибокої технічної інтеграції. Водночас, для високонавантажених проєктів або глибоко кастомізованих сценаріїв, його API дозволяє здійснювати повноцінне керування всіма аспектами email-інфраструктури.

Таким чином, SendGrid — це потужне SaaS-рішення для email-комунікації, яке найбільше підходить у тих випадках, коли важливо забезпечити надійну, масштабовану, гнучку та аналітично керовану email-доставку, особливо якщо мова йде не тільки про транзакційні, але й маркетингові повідомлення. Інформаційна система SendGrid наведена на рис. 1.6.



Рисунок 1.6 - Інформаційна система SendGrid

Джерело: <https://encrypted->

[tbn0.gstatic.com/images?q=tbn:ANd9GcRAEFv0kIX5TfR80i9eoPijBDYaiye8Zz69ow&s](https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRAEFv0kIX5TfR80i9eoPijBDYaiye8Zz69ow&s)

Отже, висновок такий, що ці рішення мають різний фокус — одні краще підходять для email, інші для SMS або push-сповіщень — тому вибір конкретного інструменту залежить від завдань системи, її масштабів і цільової аудиторії.

Проаналізовані вище SaaS-рішення демонструють високу ефективність у своєму напрямку, однак здебільшого зосереджені лише на окремих каналах комунікації — push, email або SMS. Це зумовлює потребу у додатковій інтеграції або використанні кількох сервісів одночасно для забезпечення повного покриття нотифікацій. У зв'язку з цим актуальним постає створення універсальної інформаційної управляючої системи, яка поєднує всі основні типи нотифікацій в єдиному рішенні, забезпечуючи гнучкість, масштабованість і централізоване управління комунікаційними потоками.

1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи

У даному підрозділі розглянуто концептуальні рішення, що лежать в основі створення інформаційної управляючої системи для автоматизації процесу нотифікацій. Запропонована система має забезпечувати обробку великої кількості повідомлень у режимі реального часу, мати модульну архітектуру з можливістю масштабування та використовувати ефективні технології зберігання, маршрутизації й доставки повідомлень. Концептуальна модель будується на мікросервісній архітектурі з використанням брокера повідомлень Redis, що дає можливість організувати асинхронну обробку нотифікацій різних типів — email, SMS, push. В якості серверної платформи обрано мову PHP [28] та фреймворк Symfony [29], які забезпечують необхідну гнучкість, безпеку та продуктивність. Для фронтенду використовується Vue.js 2 [30], який дозволяє будувати динамічний інтерфейс із підтримкою AJAX-запитів [31].

База даних реалізована на MySQL [32] як надійне реляційне сховище, що задовольняє вимоги до ACID-гарантій [33] і дає змогу зберігати користувацькі налаштування, черги повідомлень та лог подій. Організація інфологічної моделі передбачає виділення ключових сутностей системи, зокрема користувачів, повідомлень, каналів доставки та подій, що їх тригерять. Для обміну між мікросервісами застосовується Redis у режимі pub/sub, що мінімізує затримки доставки повідомлень. Система не потребує кешування через природу її роботи в реальному часі та постійне оновлення даних.

Методологічно розробка базується на принципах SOLID [34] і чистого коду, з дотриманням MVC-патерну [35] в архітектурі веб-застосунку. Процес розробки організовано за допомогою методології Agile, що дає змогу гнучко адаптуватися до змін у вимогах та поступово розширювати функціонал. У якості основного інструменту розробки обрано IDE PHPStorm [36], яка має потужну інтеграцію з інструментами налагодження, тестування та аналізу коду.

РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ

В даному розділі йдеться мова про характеристику об'єкта дослідження, визначається структура ІУС, а також притаманні їй особливості, формується постановка задачі та обґрунтовується вибір методів і моделей ІУС.

2.1 Структура і характеристика системи

На основі проведеного аналізу існуючих моделей і систем у сфері автоматизації нотифікацій сформульовано концептуальні рішення, що визначають архітектуру та принципи побудови розроблюваної інформаційної управлінської системи.

Ключовим концептуальним положенням є використання подійно-орієнтованої архітектури, яка забезпечує асинхронну обробку подій у реальному часі. Джерелами подій можуть виступати внутрішні або зовнішні сервіси підприємства, взаємодія з якими реалізується через стандартизовані API або системи брокерів повідомлень.

Основу архітектури складає багаторівнева система [37], що включає такі шари:

- шар прийому та обробки подій;
- шар обробки бізнес-логіки нотифікацій та генерації повідомлень;
- шар управління чергами доставки;
- шар інтерфейсу взаємодії із зовнішніми каналами зв'язку;
- шар зберігання інформації та моніторингу стану системи.

Передбачено побудову системи як набору незалежних мікросервісів із чітко визначеними зонами відповідальності, що забезпечує високу гнучкість масштабування та спрощує процеси оновлення окремих компонентів без зупинки всієї системи. Діаграму функціональних вимог наведено на рис. 2.1.

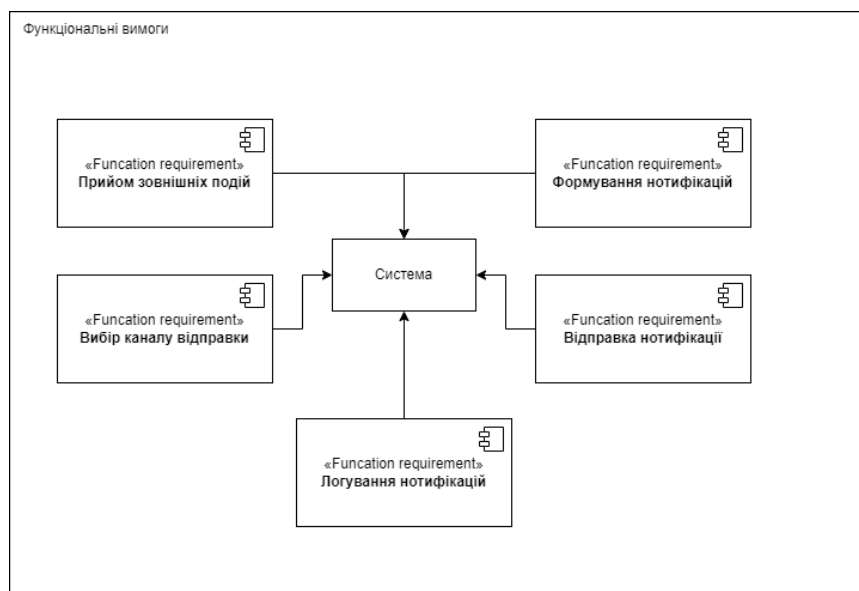


Рисунок 2.1 - Функціональні вимоги

Джерело: розроблено автором

Для забезпечення надійності та відмовостійкості системи впроваджується механізм обробки збоїв із використанням черг повторних спроб доставки повідомлень, а також модуль моніторингу із функціями виявлення та реагування на аномальні ситуації. Діаграма нефункціональних вимог наведена на рис. 2.2

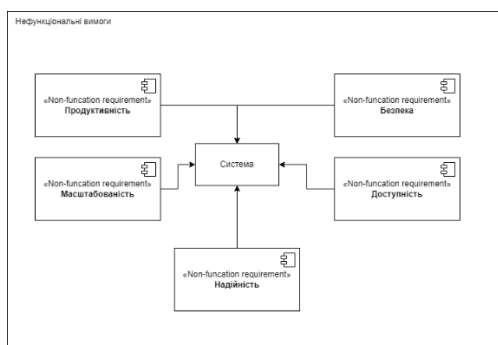


Рисунок 2.2 - Нефункціональні вимоги

Джерело: розроблено автором

Ще одним концептуальним рішенням є підтримка мультимодальних каналів комунікації. Система повинна мати можливість надсилати повідомлення за допомогою різних сервісів: електронної пошти, SMS, push-нотифікацій, месенджерів тощо. Вибір каналу доставки здійснюється автоматично на основі налаштувань користувачів та пріоритетності каналів. Бізнес вимоги наведено на рис. 2.3

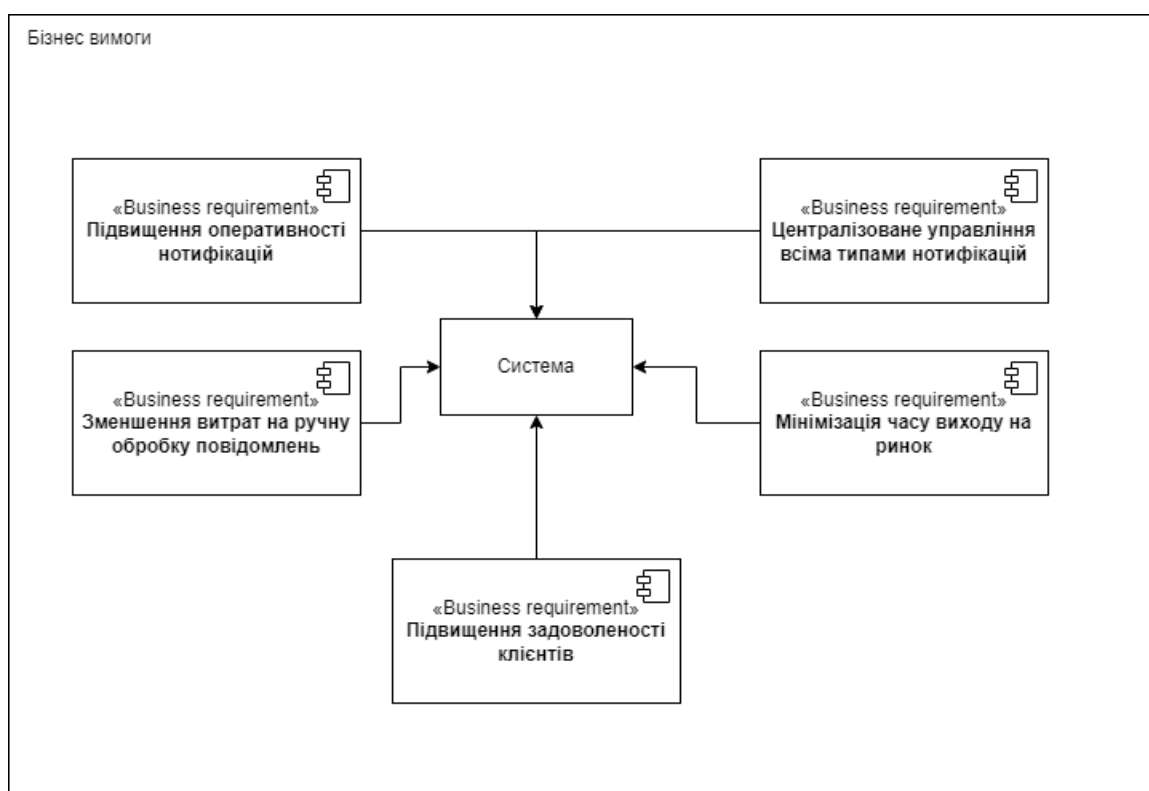


Рисунок 2.3 - Бізнес вимоги

Джерело: розроблено автором

Безпека даних розглядається як важливий аспект системи: передбачається шифрування даних у транспортному середовищі та автентифікація всіх взаємодіючих сервісів.

Для повноцінного опису функціонування розроблюваної інформаційної управлінської системи автоматизації нотифікацій сформульовано моделі її структури та поведінки. Діаграму прецедентів наведено на рис. 2.4.

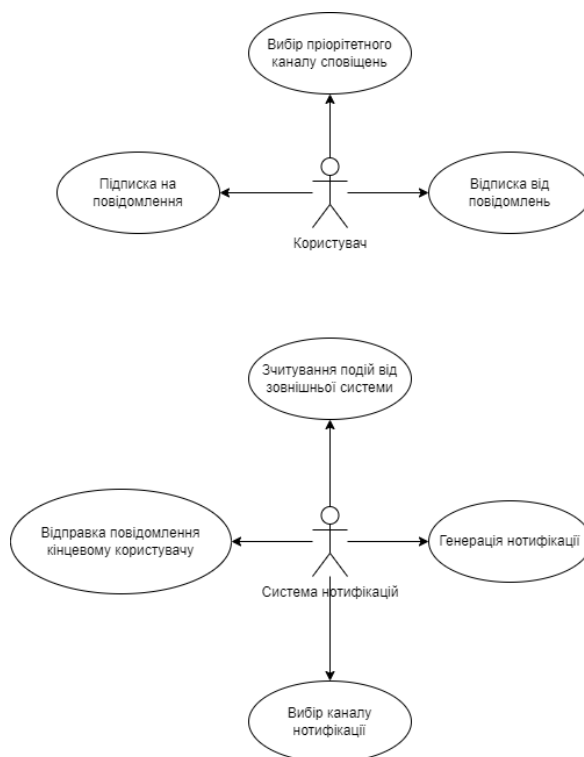


Рисунок 2.4 - Діаграма прецедентів

Джерело: розроблено автором

Структурна модель системи відображає основні компоненти та взаємозв'язки між ними. До складу ІУС входять наступні елементи:

- модуль прийому подій;
- модуль обробки бізнес-логіки нотифікацій;
- модуль шаблонізації повідомлень;
- система управління чергами повідомлень;
- модулі інтеграції з каналами доставки;
- база даних.

Модуль прийому подій забезпечує отримання повідомлень про події із зовнішніх та внутрішніх систем через API або брокери повідомлень.

Модуль обробки бізнес-логіки нотифікацій визначає правила формування повідомлень на основі типу події та заданих сценаріїв.

Модуль шаблонізації повідомлень генерує текстові або мультимедійні повідомлення відповідно до заданих шаблонів.

Система управління чергами повідомлень організовує черговість доставки повідомлень і контроль за їх успішною доставкою.

Модулі інтеграції з каналами доставки забезпечують взаємодію з сервісами електронної пошти, SMS-шлюзами, системами push-нотифікацій тощо.

База даних зберігає інформацію про події, повідомлення, статуси доставок та аналітичні дані.

Між модулями встановлено канали взаємодії через черги повідомлень або внутрішні API, що забезпечує незалежність компонентів і можливість горизонтального масштабування.

Модель поведінки системи описує послідовність реакцій на вхідні події:

- подія надходить до модуля прийому і фіксується у базі даних;
- модуль обробки бізнес-логіки аналізує подію і визначає, чи потрібно створювати нотифікацію;
- у разі потреби формується повідомлення на основі шаблонів;
- повідомлення передається до системи чергування доставки;
- повідомлення доставляється через відповідний канал;
- отримується відповідь про успішність або помилку доставки;
- статус повідомлення оновлюється у базі даних, відповідні події фіксуються у системі логування;
- у випадку помилок повторна доставка ініціюється відповідно до заданих правил повторних спроб.

Поведінкова модель системи може бути формалізована у вигляді діаграм UML, зокрема діаграми діяльності та діаграми станів. Таке формалізоване представлення дозволяє однозначно описати всі можливі сценарії взаємодії компонентів системи та полегшує валідацію правильності її функціонування. Діаграму компонентів наведено на рис. 2.5.

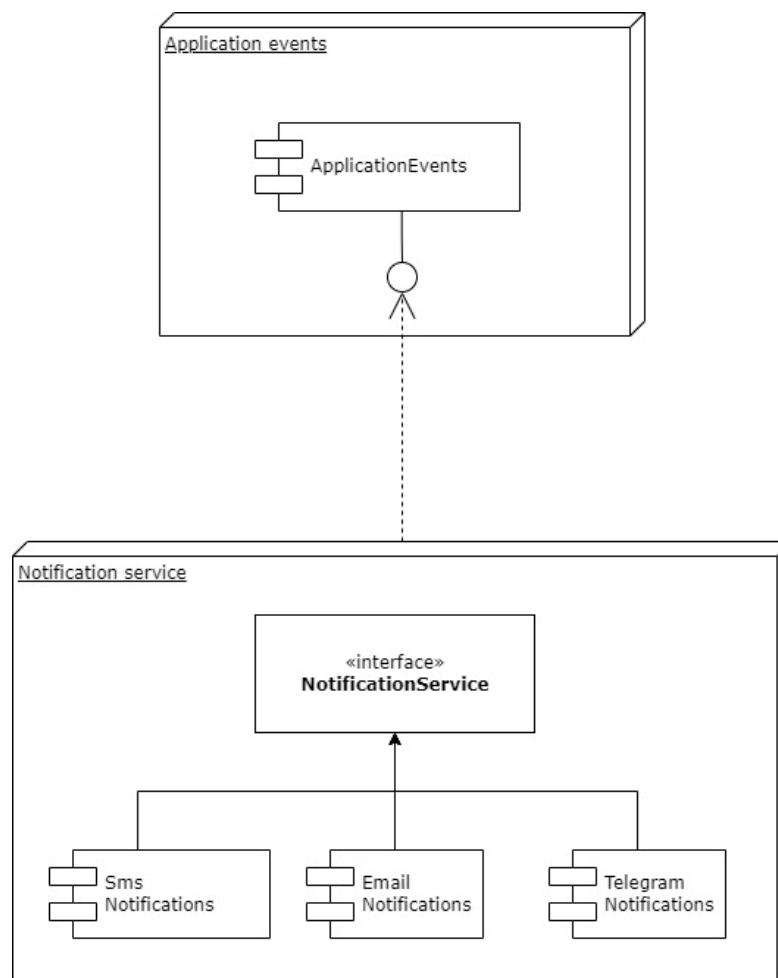


Рисунок 2.5 - Діаграма компонентів

Джерело: розроблено автором

Для поглибленого аналізу структури та функціонування інформаційної управлінської системи автоматизації нотифікацій доцільним є використання формальних моделей, зокрема діаграм UML. Застосування таких моделей забезпечує візуалізацію внутрішніх процесів, структуроване відображення динаміки змін системних станів та об'єктивізацію вимог до функціональності системи.

Одним із базових інструментів формалізації є діаграма діяльності (Activity Diagram), яка ілюструє логіку виконання бізнес-процесів у системі. Діаграма діяльності дозволяє описати алгоритм обробки події від моменту її отримання до доставки повідомлення користувачу або реєстрації помилки. На основі специфіки предметної області, процес обробки події в системі автоматизації нотифікацій

включає етапи прийняття події, аналізу бізнес-правил, створення повідомлення, постановки у чергу, виконання доставки, фіксації результату та ініціювання повторних спроб у разі виникнення збоїв.

Іншим важливим засобом опису поведінки системи є діаграма станів (State Machine Diagram). Вона демонструє зміну станів об'єкта (у даному випадку — повідомлення) під впливом внутрішніх і зовнішніх подій. Типова модель життєвого циклу повідомлення передбачає стани "Створено", "Готове до відправки", "Відправлено", "Доставлено успішно", "Помилка доставки" та "Повторна спроба доставки". Переходи між станами детермінуються результатами дій модулів системи та відповідями зовнішніх сервісів.

Використання діаграм UML для моделювання структури та поведінки системи має такі переваги:

- забезпечення формалізованого, однозначного трактування процесів;
- підвищення прозорості архітектури системи для розробників, замовників та інших зацікавлених сторін;
- можливість виявлення логічних суперечностей та вузьких місць ще на етапі проектування;
- спрощення процесу тестування та валідації розробленого рішення.

Побудова моделей структури та поведінки системи є критично важливою складовою етапу наукового обґрунтування і проектування інформаційної управлінської системи автоматизації нотифікацій.

На етапі деталізації проєктних рішень щодо створення інформаційної управлінської системи (ІУС) автоматизації нотифікацій особливу увагу слід приділити моделюванню поведінки системи на рівні окремих підсистем і задач. З цією метою доцільно застосовувати комплекс моделей UML, що забезпечують багаторівневу візуалізацію взаємодії користувачів із системою, послідовності обробки подій та внутрішньої логіки виконання бізнес-процесів.

Діаграма прецедентів (Use Case Diagram) відображає взаємодію акторів із підсистемою або окремою функціональністю. Вона дозволяє визначити перелік

функціональних можливостей, які має забезпечувати система для різних типів користувачів або зовнішніх систем. Наприклад, для підсистеми нотифікацій буде актуальним опис таких прецедентів, як "Ініціювати створення повідомлення", "Відправити повідомлення користувачу", "Отримати статус доставки", "Повторно надіслати повідомлення при збої".

Діаграма послідовності (Sequence Diagram) детально ілюструє порядок обміну повідомленнями між об'єктами системи у процесі реалізації певного прецеденту. Вона дозволяє відобразити динаміку взаємодії компонентів у часовій послідовності. Наприклад, у випадку надсилання повідомлення діаграма послідовності може показувати процес: створення повідомлення → перевірка правил → постановка у чергу → виклик сервісу доставки → отримання відповіді → оновлення статусу повідомлення.

Діаграма діяльності (Activity Diagram) моделює алгоритм виконання окремої задачі у вигляді послідовності дій, що відображають обробку події на рівні бізнес-логіки. На відміну від діаграми послідовності, що фокусується на взаємодії об'єктів, діаграма діяльності зосереджена на самій логіці процесу: виявлення події → перевірка валідності → розрахунок параметрів повідомлення → маршрутизація повідомлення залежно від його типу та пріоритету.

Комплексне застосування цих трьох типів діаграм забезпечує багатовимірне відображення поведінки підсистеми та дозволяє:

- всебічно охарактеризувати зовнішні вимоги до системи;
- формалізувати внутрішні механізми обробки даних;
- знизити ймовірність виникнення логічних похибок у процесі подальшої реалізації.

Таким чином, використання діаграм прецедентів, послідовності та діяльності є обов'язковим елементом наукового обґрунтування архітектури інформаційної системи на рівні окремих задач і підсистем.

Метою побудови діаграми прецедентів є формалізація взаємодії зовнішніх акторів із функціональністю підсистеми. Основними акторами виступають

"Користувач системи" та "Зовнішній сервіс доставки". Користувач ініціює прецеденти "Створити повідомлення", "Переглянути статус повідомлення", "Ініціювати повторну відправку". Зовнішній сервіс доставки взаємодіє із системою у межах прецеденту "Підтвердити доставку повідомлення" або "Повідомити про помилку доставки". Діаграма також відображає залежності між прецедентами, зокрема розширення ("extend") операції повторної відправки залежно від результату первинної доставки.

Діаграма послідовності моделює етапи взаємодії об'єктів системи під час обробки запиту на надсилання повідомлення. Учасниками взаємодії є об'єкти: "Користувач", "Сервіс створення повідомлень", "Черга повідомлень", "Сервіс доставки", "Сховище повідомлень". Послідовність подій включає: отримання запиту від користувача → формування сутності повідомлення → передача повідомлення у чергу → виклик модуля доставки → обробка відповіді сервісу доставки → запис статусу доставки у базу даних. Діаграма відображає як синхронні, так і асинхронні виклики залежно від природи інтеграцій.

Діаграма діяльності відображає процес управління життєвим циклом повідомлення у відповідь на отримані події від зовнішнього сервісу доставки. Процес розпочинається з перевірки валідності отриманої події. У разі успішної валідації виконується оновлення статусу повідомлення у базі даних. Якщо подія вказує на помилку доставки, ініціюється процедура планування повторної відправки, яка передбачає або негайне повторення, або відкладене пересилання залежно від кількості попередніх спроб. У разі успішної доставки повідомлення завершує свій життєвий цикл.

На етапі побудови логічної моделі інформаційної управлінської системи автоматизації нотифікацій доцільним є застосування діаграм класів UML для відображення її статичної структури. Діаграми класів є ефективним засобом представлення основних сутностей системи, їхніх атрибутів, операцій, а також логічних зв'язків і відношень між об'єктами.

З метою забезпечення прозорості та ієрархічної організації структури моделі, рекомендовано формувати декілька діаграм із різними рівнями деталізації:

- діаграма пакетів класів;
- діаграма підмножини класів і відношень між ними;
- діаграма класів із атрибутами та операціями.

Діаграма пакетів класів. На цій діаграмі зображуються основні логічні групи класів (пакети) та взаємозв'язки між ними. Для системи автоматизації нотифікацій такими пакетами можуть бути: "Керування повідомленнями", "Керування чергою", "Сервіси доставки", "Моніторинг і логування", "Інтеграція із зовнішніми системами".

Діаграма підмножини класів і відношень між ними. Цей тип діаграми деталізує взаємодію класів всередині окремого пакета або між кількома пакетами. Наприклад, у пакеті "Керування повідомленнями" можуть бути класи: Message, MessageTemplate, Recipient, з відношеннями "асоціація", "агрегація" або "композиція" для відображення внутрішньої структури повідомлень і їх зв'язку з отримувачами.

Діаграма класів із атрибутами та операціями. Ця діаграма є найбільш деталізованою і відображає внутрішню структуру класів, включаючи їхні атрибути (з типами даних) та основні методи (операції), необхідні для забезпечення функціональності підсистеми. Наприклад, клас Message може містити атрибути id, content, status, timestamp і методи send(), validate(), retry().

Побудова таких діаграм забезпечує:

- уніфіковане уявлення про архітектуру підсистеми;
- полегшення процесу розробки та тестування;
- можливість виявлення структурних помилок ще на етапі проєктування;
- забезпечення узгодженості між логічною моделлю й фізичною реалізацією системи.

Використання діаграм класів різних рівнів деталізації дозволяє системно організувати інформацію про підсистему та слугує основою для її ефективної реалізації.

У процесі побудови структурної моделі підсистеми автоматизації нотифікацій важливим є виділення основних стереотипів класів, які відіграють специфічні ролі в архітектурі інформаційної системи. До таких стереотипів належать:

- класи-сутність;
- граничні класи;
- класи керування.

Класи-сутності (entity classes) призначені для моделювання об'єктів предметної області з довготривалим життєвим циклом. У контексті підсистеми нотифікацій такими класами є, наприклад, Message, User, NotificationChannel. Ці сутності зберігають основні дані, які системі необхідно обробляти, і прямо відповідають елементам бази даних.

Граничні класи (boundary classes) моделюють інтерфейси взаємодії користувача або зовнішніх систем із підсистемою. Вони відповідають за прийом запитів і передачу результатів. У запропонованій системі до граничних класів можуть належати NotificationAPI, AdminDashboardInterface, WebhookListener. Такі класи реалізують прийом запитів на створення повідомлень, обробку подій доставки тощо.

Класи керування (control classes) координують послідовність подій і логіку виконання прецедентів системи. Вони забезпечують узгоджену роботу між граничними класами та класами-сутностями. У нашій підсистемі такими є, зокрема, NotificationManager, DeliveryScheduler, RetryHandler. Ці класи відповідають за маршрутизацію повідомлень, планування повторних спроб доставки, контроль за статусами повідомлень.

Побудова діаграм класів із чітким виділенням стереотипів дає змогу:

- структурно організувати моделі підсистеми;
- забезпечити високу модульність і масштабованість рішень;
- сприяти полегшенню процесу тестування та супроводження системи.

Таким чином, у даному підрозділі було визначено основні стереотипи класів, що формують структурну основу інформаційної системи автоматизації нотифікацій. Виділення класів-сутностей, граничних класів та класів керування дозволило системно організувати архітектуру підсистеми, забезпечити чітке розмежування відповідальностей між компонентами та підвищити зрозумілість і логічну узгодженість моделі. Застосування стереотипів UML у побудові діаграм класів сприяє підвищенню формалізації процесу проектування системи, полегшує її розширення, супроводження та адаптацію до змін вимог. Створена структурна модель слугує надійною основою для подальшої реалізації інформаційної системи та забезпечує умови для високої якості розробки.

Вибір та обґрунтування технологій. PHP було обрано як основну мову програмування для реалізації системи автоматизації зовнішніх нотифікацій з огляду на її численні переваги, що роблять її зручною, ефективною та економічно вигідною у порівнянні з іншими поширеними мовами.

Перш за все, PHP спеціалізується на веб-розробці і з самого початку була орієнтована на створення динамічних вебсайтів. У цьому середовищі вона має глибоку інтеграцію з веб-серверами (зокрема Apache і Nginx), зручну роботу з HTTP-запитами, формами, сесіями та куками. На відміну від таких мов як Java чи C#, PHP не потребує складної конфігурації або запуску окремого серверного контейнера, що знижує поріг входу і спрощує деплой.

Другою важливою перевагою є розвинена екосистема: PHP підтримує тисячі бібліотек та фреймворків, серед яких особливо виділяється Symfony — модульний і гнучкий фреймворк, який ідеально підходить для побудови масштабованих систем з чіткою структурою.

У порівнянні з Python, який часто використовується для аналітики або розробки REST API, PHP краще інтегрується з веб-хостингами і менш вимогливий до ресурсів у простих вебсистемах. А в порівнянні з Java або Node.js, PHP забезпечує простіший цикл розробки й має нижчі витрати на підтримку інфраструктури.

Загалом, вибір PHP як основної мови зумовлений її орієнтацією на веб, зрілістю, доступністю інструментів та ресурсів, а також простотою масштабування, що ідеально відповідає завданням розробки системи нотифікацій.

У межах реалізації системи автоматизації зовнішніх нотифікацій було обрано систему керування базами даних MySQL. Це рішення ґрунтується на низці технічних, практичних та економічних переваг, які роблять MySQL оптимальним вибором для зберігання та обробки даних у веб-орієнтованому середовищі.

Передусім, MySQL є однією з найпопулярніших реляційних СУБД у світі, і вона відзначається високою стабільністю, продуктивністю та сумісністю з більшістю поширених мов програмування — зокрема, з PHP, яка використовується в цьому проєкті. Її нативна підтримка через розширення `mysqli` або абстракції Doctrine (у Symfony) дозволяє реалізувати безпечну та гнучку взаємодію з базою даних без додаткових накладних витрат на інтеграцію. На відміну від складніших і важчих рішень, як-от Oracle чи Microsoft SQL Server, MySQL має низькі вимоги до ресурсів і легко масштабується горизонтально — особливо у зв'язці з реплікацією або шардуванням. Це особливо важливо у випадку зростання кількості подій, каналів і користувачів у системі нотифікацій. Для систем, де важлива швидка обробка великих обсягів транзакцій (наприклад, логів, черг подій або станів доставки повідомлень), MySQL здатен забезпечити потрібну продуктивність із мінімальним споживанням ресурсів.

Отже, вибір MySQL продиктований її надійністю, продуктивністю, інтеграційною гнучкістю, зрозумілістю для розробників та підтримкою масштабування — саме ці фактори дозволяють побудувати стабільну та ефективну систему нотифікацій, здатну до зростання та адаптації в майбутньому.

2.2 Методи та моделі в інформаційних управляючих системах і технологіях

Методи та моделі є фундаментальними інструментами побудови сучасних інформаційних управляючих систем (ІУС), оскільки саме вони визначають способи обробки даних, формалізації¹ знань, прийняття рішень та реалізації функціональних процесів системи.

Класифікація багатокomпонентних інформаційних систем (ІС) може бути проведена за кількома критеріями, що враховують різноманітність типів подій, каналів комунікації та бізнес-правил:

- ІУС з подієво-орієнтованою архітектурою;
- ІУС з транзакційною моделлю;
- ІУС із потоковою обробкою подій

Для реалізації системи було обрано подієво-орієнтовану архітектуру (EDA). Вона дозволяє ефективно реагувати на динамічні події в реальному часі та забезпечує гнучке масштабування обробки нотифікацій. Завдяки EDA система легко адаптується до нових каналів комунікації та змін у бізнес-логіці без зупинки роботи. Діаграма подієво-орієнтованої архітектури наведена на рис. 2.6.

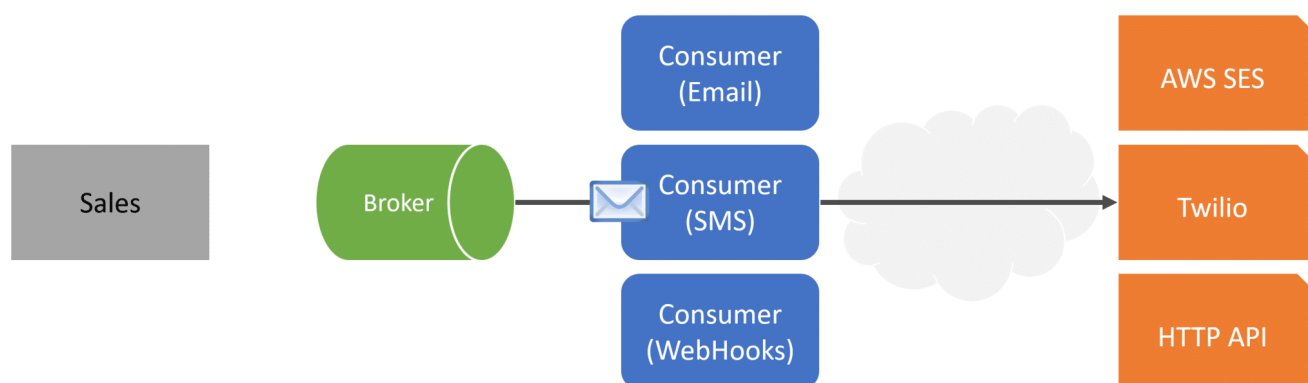


Рисунок 2.6 - Діаграма подієво-орієнтованої архітектури EDA

Джерело: <https://codeopinion.com/wp-content/uploads/2022/05/4.png>

У контексті розробки системи автоматизації нотифікацій використовуються як загальні методи інженерії програмного забезпечення, так і спеціалізовані моделі, що відповідають специфіці побудови розподілених обчислювальних систем.

До основних методів, застосованих у роботі, належать:

- методи об'єктно-орієнтованого проєктування;
- методи моделювання процесів та поведінки системи за допомогою uml;
- методи аналізу та проєктування баз даних;
- методи формалізації бізнес-процесів.

Методи об'єктно-орієнтованого проєктування (ООП), що дозволяють структурувати систему у вигляді взаємодіючих об'єктів із чітко визначеними ролями та відповідальністю. Це забезпечує модульність [22], масштабованість і повторне використання коду.

Методи моделювання процесів та поведінки системи за допомогою UML. Зокрема, діаграми прецедентів, послідовності, діяльності та класифікації застосовані для відображення функціональної та структурної моделей системи.

Методи аналізу та проєктування баз даних, які забезпечують ефективне зберігання та доступ до інформації про повідомлення, користувачів і канали доставки.

Методи формалізації бізнес-процесів на основі BPMN-підходів (Business Process Model and Notation), що дозволяють інтегрувати систему у більш широкий контекст управлінської діяльності підприємства.

Щодо моделей, у підсистемі автоматизації нотифікацій реалізовані такі концептуальні підходи:

- модель подій та реакцій;
- модель централізованої черги повідомлень;
- модель повторної доставки;
- модель оркестрації сервісів.

Система побудована навколо ідеї, що події (тригери) викликають відповідні реакції — надсилання повідомлень, зміни статусів, запис у лог тощо.

Для забезпечення масштабованості та контролю навантаження використовується модель централізованої черги повідомлень, де кожне повідомлення є окремою одиницею обробки.

Оскільки передача повідомлень по різних каналах може бути ненадійною, застосовується стратегія повторних спроб із використанням зростаючих інтервалів.

Підсистема взаємодіє з кількома зовнішніми API, для чого використовуються механізми керування потоками задач і контролю станів.

Використання зазначених методів та моделей забезпечує досягнення високої гнучкості, стійкості до помилок та адаптивності розробленої системи до змін середовища експлуатації. Комплексне поєднання об'єктно-орієнтованого підходу, моделювання процесів та управління потоками даних є запорукою успішної реалізації цілей даного дослідження.

В умовах стрімкого розвитку інформаційних технологій складність завдань в техніці, фінансово-кредитній, а також у соціально-економічній сферах діяльності постійно зростає. Збільшення обсягів даних, ускладнення бізнес-процесів, посилення конкурентного середовища та підвищення вимог до оперативності прийняття рішень зумовлюють необхідність розробки все більш досконалих інформаційних управляючих систем.

Зокрема, в технічній сфері задачі проектування, управління та моніторингу об'єктів набувають комплексного характеру, вимагаючи інтеграції різномірних джерел даних, підтримки роботи в реальному часі та забезпечення високої надійності систем. У фінансово-кредитних структурах значно зросли вимоги до точності аналітичних розрахунків, безпеки обробки персональних і транзакційних даних, адаптивності до змін регуляторного середовища та потреб клієнтів. У соціально-економічній галузі вирішення задач управління великими масивами інформації про населення, послуги, ресурси потребує широкого впровадження систем обробки знань, інтелектуальної аналітики та автоматизованого прийняття рішень.

У зв'язку з цим вимоги до засобів вирішення зазначених завдань суттєво ускладнилися і передбачають: підвищену масштабованість рішень, здатність до

обробки великих обсягів розподілених даних, використання гнучких архітектурних підходів, впровадження механізмів самонавчання та прогнозування. Інформаційні управляючі системи повинні забезпечувати не лише збереження та обробку інформації, але й активно підтримувати процеси аналізу, планування і управління, адаптуючись до змінних умов середовища.

Таким чином, розробка систем автоматизації, зокрема системи автоматизації нотифікацій, має враховувати ці загальні тенденції і бути орієнтованою на забезпечення високої продуктивності, гнучкості, надійності та можливості розширення функціональності. Використання сучасних методів і моделей у побудові ІУС є необхідною умовою для задоволення цих вимог.

У сучасних умовах більшість задач, що виникають у технічній, фінансово-кредитній та соціально-економічній сферах, мають характер багатокритеріальних і багатоцільових задач прийняття управлінських рішень. Вони потребують не лише одночасного досягнення декількох цілей, але й забезпечення узгодженості між ними, навіть у випадках, коли ці цілі є частково або повністю конфліктними. Такий характер задач вимагає комплексного підходу до вибору альтернатив, який передбачає врахування широкого спектра критеріїв, що часто мають різну природу кількісну, якісну, економічну, технічну, соціальну тощо.

У результаті, проектування та реалізація інформаційних управляючих систем має здійснюватися з орієнтацією на підтримку багатокритеріального аналізу і механізмів ухвалення рішень в умовах невизначеності, ризику, нечітких вимог та обмежень. Це зумовлює необхідність використання моделей, які здатні адаптуватися до змін входних параметрів і враховувати суб'єктивні переваги користувача, а також методів, що забезпечують багаторівневу оптимізацію управлінських рішень.

Реалізація систем автоматизації — таких як система управління нотифікаціями — у такому контексті передбачає не лише технічну ефективність, але й здатність гнучко адаптуватися до множинності умов, забезпечуючи підтримку прийняття рішень відповідно до заданих багатокритеріальних моделей.

У розробленій системі автоматизації нотифікацій багатокритеріальність управлінських задач виявляється у необхідності одночасного врахування різних аспектів, таких як терміновість повідомлення, тип користувача, його пріоритет у системі, канал доставки (SMS, електронна пошта, push-нотифікація), а також поточне навантаження на систему. Наприклад, в умовах обмежених ресурсів система має обирати найефективніший канал надсилання, який забезпечить своєчасну доставку без перевантаження зовнішніх сервісів. У разі надсилання великої кількості повідомлень пріоритетним користувачам, система повинна враховувати політику черг і планування навантаження, зокрема шляхом застосування алгоритмів динамічного пріоритетного сортування.

Крім того, у розробці системи реалізовані модулі, які забезпечують гнучку конфігурацію політик сповіщення відповідно до організаційних потреб замовника. Це дозволяє враховувати не лише технічні, а й бізнес-критерії: наприклад, чи є користувач клієнтом із преміальним статусом, чи сповіщення стосується критичних інцидентів, чи має повідомлення юридичну силу тощо. З огляду на такі фактори, у системі застосовується багаторівнева модель правил, що включає логіку прийняття рішень на основі зважених коефіцієнтів і гнучких умовних конструкцій.

Таким чином, система підтримує прийняття складних багатоцільових рішень у реальному часі, що забезпечує високу адаптивність, ефективне управління інформаційними потоками та оптимізацію ресурсів. Запропонована архітектура і логіка роботи системи можуть бути масштабовані та адаптовані до інших задач управління нотифікаціями в межах підприємств або організацій різних типів.

В умовах сучасного інформаційного середовища прийняття ефективних управлінських рішень ускладнюється як через дефіцит релевантних даних, так і через їх надмірну кількість. У першому випадку виникає проблема неповноти, нечіткості або невизначеності вхідної інформації, що потребує застосування спеціалізованих моделей — зокрема, методів нечіткої логіки, байєсівських мереж чи евристичних підходів, які дозволяють ухвалювати рішення в умовах ризику. У другому випадку, коли даних надто багато, основне завдання полягає у фільтрації,

класифікації та аналізі інформації, щоб відокремити релевантні від шумових або хибних даних (зокрема фейкових, дубльованих чи застарілих).

У рамках проєктування системи автоматизації нотифікацій враховано обидва типи викликів. У разі браку інформації про користувача або контекст ситуації система реалізує політику прийняття дефолтних рішень із найменшим ризиком — наприклад, надсилання повідомлення через універсальний канал. У ситуації надлишковості — наприклад, коли одночасно спрацьовує велика кількість тригерів або надходить багато дубльованих повідомлень з різних джерел — система застосовує механізми дедуплікації, пріоритезації та об'єднання повідомлень на основі правил кореляції та контекстного аналізу.

Таким чином, інформаційна управляюча система забезпечує не лише технічну реалізацію надсилання нотифікацій, а й виконує роль "інтелектуального фільтра", який дозволяє приймати виважені рішення навіть за умов неповноти або надмірності інформації. Це підвищує якість обслуговування користувачів, знижує навантаження на канали комунікації та сприяє стабільному функціонуванню системи в умовах складної інформаційної динаміки.

Значна частина задач прийняття рішень є слабоструктурованими або повністю неструктурованими, що означає відсутність чітко визначених правил, алгоритмів чи однозначно формалізованих залежностей між вхідними параметрами та очікуваними результатами. Такі задачі, як правило, супроводжуються високим ступенем невизначеності, неоднозначністю цілей і обмежень, необхідністю інтерпретації контексту та залучення суб'єктивних експертних оцінок. Зокрема, в системах управління повідомленнями ці задачі можуть виникати в ситуаціях кризового інформування, масових збоїв, кіберінцидентів або нових сценаріїв взаємодії, що не мають прецедентів у минулому досвіді.

Наприклад, система може накопичувати історичні дані щодо реакцій користувачів на повідомлення різного типу в різних контекстах та на основі цього формувати рекомендаційні правила. Крім того, за допомогою механізмів логіки першого порядку або онтологічного опису знань можна будувати гнучкі моделі для інтерпретації слабоструктурованих подій.

Урахування слабкої формалізованості дозволяє системі адаптуватися до нових умов без необхідності повного перепроєктування. Таким чином, вона підтримує прийняття управлінських рішень у ситуаціях, де традиційні аналітичні методи є малоефективними або непридатними. Це забезпечує не лише функціональну гнучкість системи, а й її здатність до еволюційного розвитку відповідно до змін вимог та умов експлуатації.

Подолання викликів, пов'язаних із слабоструктурованістю, багатокритеріальністю та невизначеністю задач управління, можливе завдяки застосуванню адекватного математичного апарату, зокрема економіко-математичних методів та моделей. Ці методи дозволяють формалізувати складні процеси прийняття рішень, представити їх у вигляді оптимізаційних, імовірнісних, логіко-структурних або симуляційних моделей, що адаптовані до умов функціонування інформаційних управляючих систем (ІУС).

Використання такого математичного апарату дозволяє системі не лише реагувати на події у режимі реального часу, а й здійснювати інтелектуальну підтримку управлінських рішень шляхом генерування альтернатив, оцінки ризиків і вибору оптимального сценарію дій. Отже, інтеграція економіко-математичних моделей у функціональну архітектуру ІУС забезпечує якісно новий рівень її адаптивності, стійкості до інформаційних перевантажень та здатності до функціонування в умовах високої складності управлінського середовища.

Узагальнюючи викладене у підрозділі, можна зробити висновок, що сучасні задачі управління в інформаційних системах характеризуються підвищеною складністю, багатокритеріальністю, невизначеністю та слабкою структурованістю. У таких умовах традиційні засоби прийняття рішень виявляються недостатніми, що зумовлює необхідність застосування спеціалізованих економіко-математичних методів і моделей. Інтеграція таких методів у модулі інформаційних управляючих систем, зокрема з використанням компонентів штучного інтелекту, дозволяє підвищити ефективність обробки даних, сформувати адаптивні механізми підтримки рішень і забезпечити більш точне прогнозування та оцінювання управлінських сценаріїв.

Особливу увагу було приділено аналізу даних з метою оптимізації розсилок та визначення пріоритетних категорій отримувачів. З цією метою я обрав метод ABC-аналізу, який дозволяє класифікувати об'єкти (у цьому випадку — користувачів або події, що викликають нотифікації) за ступенем їхньої важливості. За формулою (3.1) можна розрахувати вибір оптимального каналу доставки.

$$C = \min (telegram, email, sms) \quad (3.1)$$

де C — оптимальний канал нотифікацій, $telegram, email, sms$ — канали нотифікацій.

Розроблена концепція системи автоматизації нотифікацій демонструє, що застосування адекватного математичного апарату є ключовою умовою її успішного функціонування в умовах динамічного та інформаційно насиченого середовища. Побудова модульної структури, використання методів оптимізації, нечіткої логіки та машинного навчання дозволяє системі не лише реагувати на події, а й проактивно формувати оптимальні шляхи інформування користувачів. Це свідчить про наявність у системи необхідних властивостей інтелектуальності, гнучкості та здатності до еволюційного розвитку.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

Даний розділ присвячений практичному етапу дослідження, в якому на основі теоретичних засад, аналізу предметної області та сформульованих вимог здійснюється безпосереднє конструювання і впровадження інформаційної системи. На цьому етапі відбувається трансформація концептуальної моделі у реальне програмне забезпечення з урахуванням архітектурних рішень, вибору технологічного стеку, інструментів розробки, специфіки інтерфейсної взаємодії та алгоритмів роботи модулів.

Особливу увагу в межах цього розділу приділено обґрунтуванню архітектури системи автоматизації нотифікацій, визначенню її функціональних компонентів, структури даних, вибору способів взаємодії між підсистемами, а також реалізації механізмів інтеграції з іншими сервісами. Наведено послідовність етапів розробки, методику тестування, описані технічні рішення, які забезпечують масштабованість, продуктивність та надійність системи.

Таким чином, у даному розділі відбувається перехід від аналітичної фази дослідження до проєктно-технологічної реалізації, що завершується створенням повнофункціонального програмного продукту, здатного ефективно вирішувати поставлені задачі в реальному середовищі експлуатації.

3.1. Проєктування бази даних та/або сховища даних для інформаційної управляючої системи

Даний підрозділ присвячений розробленню структурованої основи для зберігання, обробки та управління даними, що використовуються у процесі функціонування системи автоматизації нотифікацій. База даних є ключовим компонентом інформаційної управляючої системи, оскільки забезпечує

збереження станів, історії подій, параметрів конфігурації, користувацьких налаштувань, розкладів сповіщень та логів виконання.

Проектування бази даних базується на результатах попереднього аналізу предметної області та вимог до системи. Було обрано реляційну модель даних як таку, що найкраще відповідає вимогам до цілісності, консистентності, транзакційності та масштабованості. Структура бази даних реалізована за допомогою системи управління базами даних PostgreSQL, яка забезпечує підтримку складних запитів, індексування та реплікації.

Схема бази даних включає сутності, що відображають основні компоненти предметної області: користувачі, нотифікації, канали доставки, журнали надсилання та інші допоміжні таблиці. Було застосовано принципи нормалізації до третьої нормальної форми (3НФ) з метою усунення надмірності даних та забезпечення логічної цілісності.

3.1.1. Проектування інфологічної (логічної) моделі БД

Проектування інфологічної моделі бази даних є ключовим етапом у створенні інформаційної управляючої системи, оскільки саме на цьому рівні формується уявлення про сутності предметної області, їх атрибути та взаємозв'язки незалежно від конкретної реалізації у тій чи іншій СУБД. Інфологічна модель відображає логічну структуру даних з позицій користувача системи та формалізує інформаційні потреби, що виникають у процесі управління нотифікаціями.

Інфологічна модель є основою для подальшого переходу до даталогічного (реляційного) проектування, де вона трансформується у схему таблиць з урахуванням обмежень цілісності, нормалізації та продуктивності системи. Такий підхід гарантує відповідність структури бази даних інформаційним запитам, забезпечує гнучкість у модифікації даних та сприяє масштабованості розроблюваної системи.

У рамках даного проєкту було виокремлено основні сутності, що беруть участь у процесах формування, обробки та доставки нотифікацій. До таких сутностей віднесено:

- користувач (User);
- журнал доставки (DeliveryLog);
- шаблон повідомлення (Template);
- Параметри користувача (UserSettings);
- Група користувачів (UserGroup);
- подія (Event);
- тип повідомлення (NotificationType);
- повідомлення (Message);
- канал доставки (DeliveryChannel).

Кількість сутностей бази даних можна зменшити, оптимізувавши і декомпозуючи логіку з рівня бази даних на рівень абстракцій об'єктно орієнтованого програмування (ООП). Після оптимізації сутностей, можна виокремити наступні:

- користувач (User);
- журнал доставки (DeliveryLog);
- шаблон повідомлення (Template);
- Параметри користувача (UserSettings);
- Група користувачів (UserGroup).

Зв'язки між сутностями реалізуються у вигляді логічних асоціацій типу "один до багатьох" або "багато до багатьох", залежно від характеру взаємодії. Так, один користувач може мати багато повідомлень, одне повідомлення може належати певному типу, а один канал може використовуватись для доставки багатьох повідомлень. Для зв'язку між користувачами та каналами доставки може бути додатково введена проміжна сутність, яка дозволить зберігати індивідуальні налаштування.

Користувач - об'єкт, що представляє кінцевого отримувача нотифікацій. Подія – це ситуація чи дія в системі, яка ініціює створення нотифікації. Нотифікація - конкретне сповіщення, що створюється на основі події. Тип нотифікації - визначає формат або характер повідомлення. Канал доставки - спосіб, за допомогою якого повідомлення надсилається до користувача (SMS, Email, Push, Webhook тощо). Шаблон повідомлення – це попередньо визначена структура або текст повідомлення.

Кожна з сутностей описується набором атрибутів, які репрезентують ключові характеристики відповідного об'єкта реального світу. Наприклад, сутність "Користувач" має такі атрибути, як *user_id*, *name*, *email*, *phone_number* тощо; сутність "Повідомлення" — *notification_id*, *content*, *status*, *timestamp*, *user_id*, *event_id*. ER-діаграму таблиць бази даних наведено на рис. 3.1.

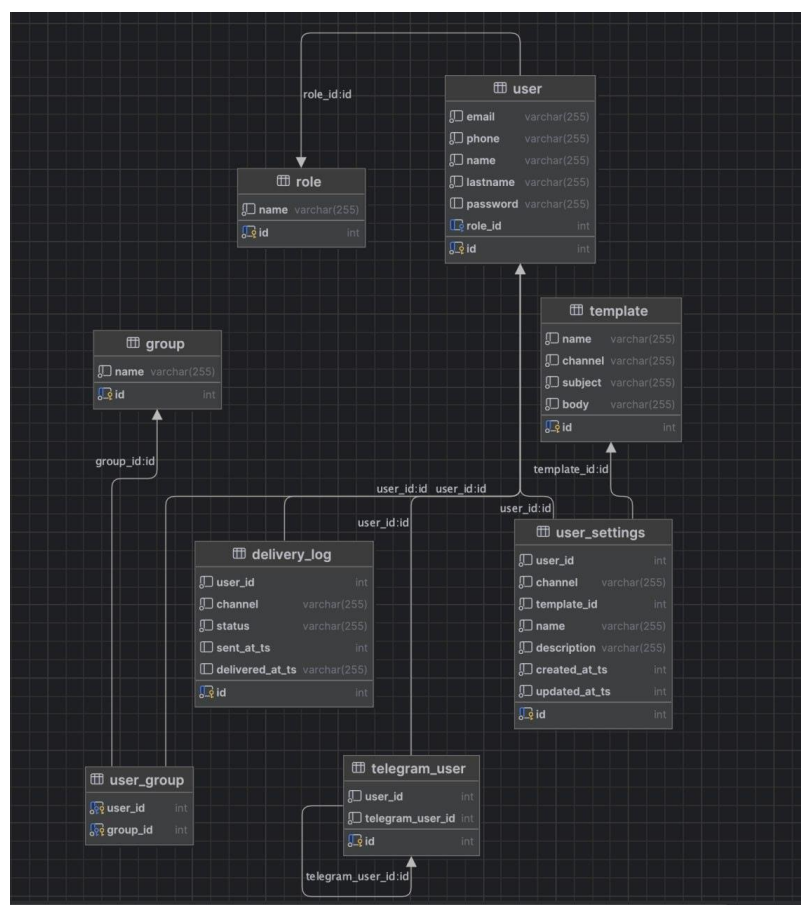


Рисунок 3.1 - ER-діаграма

Джерело: розроблено автором

3.1.2. Обґрунтування вибору СКБД та проєктування фізичної моделі БД в середовищі вибраної СКБД

У підрозділі також подано ER-діаграму, яка відображає взаємозв'язки між сутностями, а також визначено ключові атрибути, типи зв'язків (один-до-багатьох, багато-до-багатьох), обмеження цілісності та індексацію. Окрему увагу приділено аспектам безпеки та ізоляції доступу до даних, зокрема, запровадженню ролей, прав доступу та механізмів аудиту.

База даних є масштабованою, стійкою до збоїв та адаптованою до змін вимог. Вона забезпечує ефективне зберігання та обробку інформації, необхідної для функціонування системи автоматизації нотифікацій у реальному часі.

Реляційні бази даних (РБД) мають низку переваг, що обумовлюють їх широку популярність у розробці інформаційних управляючих систем. Насамперед варто відзначити логічну чіткість моделі: дані організовано у вигляді таблиць, які мають фіксовану структуру й чітко визначені зв'язки, що спрощує як проєктування, так і супровід системи.

Ключовою перевагою РБД є забезпечення цілісності даних: використання первинних та зовнішніх ключів дозволяє контролювати правильність зв'язків між таблицями, запобігаючи виникненню несумісних або дубльованих записів. Це особливо важливо в умовах високих вимог до достовірності та узгодженості інформації в системі.

Завдяки мові SQL реляційні бази даних забезпечують універсальний, стандартизований та потужний інструмент для маніпуляції даними. Це спрощує взаємодію з базою з боку прикладного програмного забезпечення, а також підтримує складну аналітику, агрегацію, фільтрацію й об'єднання даних з декількох таблиць.

Ще однією вагомою перевагою є транзакційність: РБД гарантують виконання операцій у межах транзакцій, які або завершуються повністю, або

повністю скасовуються у разі виникнення помилки. Це гарантує консистентність стану бази навіть у разі збоїв.

Реляційні бази даних також добре масштабуються вертикально, підтримують індексацію, що дозволяє значно пришвидшити вибірку за складними запитамі, і забезпечують механізми безпеки на рівні користувачів, ролей і політик доступу. Основні переваги та недоліки реляційної бази даних MySQL можна підсумувати в табл. 3.1.

Таблиця 3.1 - Переваги та недоліки реляційної бази даних mysql

Параметр	Переваги	Недоліки
Продуктивність	Висока швидкодія при роботі з невеликими й середніми обсягами даних	На великих обсягах зменшується ефективність при складних транзакціях
Простота у використанні	Легка у налаштуванні та адмініструванні	Менш зручна для складних аналітичних завдань порівняно з PostgreSQL
Підтримка транзакцій	Повна підтримка транзакцій (через InnoDB), ACID-сумісність	Обмеження в деяких типах операцій, залежно від типу таблиць
Масштабованість	Добре масштабується вертикально	Горизонтальне масштабування вимагає складних рішень (реплікація, шардінг)
Безпека	Є засоби контролю доступу, SSL-підтримка	Потребує додаткового налаштування для високого рівня безпеки
Ліцензія та вартість	Відкрите ПЗ (GPL), безкоштовне	Комерційна підтримка Oracle може бути дорогою
Підтримка та спільнота	Велика активна спільнота, багато документації	Деякі нові функції з'являються повільніше, ніж у конкурентів

Продовження таблиці 3.1

Параметр	Переваги	Недоліки
Інтеграція	Добра інтеграція з більшістю мов і фреймворків	Іноді складніше інтегрувати з Big Data або AI-системами
Надійність	Стабільна та перевірена роками система	Потребує налаштування резервного копіювання та реплікації вручну

Джерело: сформовано автором на основі виконаного дослідження

Online Analytical Processing (OLAP) та Online Transaction Processing (OLTP) – це два різні підходи до обробки даних, які мають відмінне призначення, архітектуру та характеристики.

OLTP-системи орієнтовані на виконання великої кількості коротких транзакцій, зокрема операцій читання, запису, оновлення та видалення даних у реальному часі. Їх основне завдання — забезпечити швидке обслуговування користувачів, наприклад, в інтернет-магазині, банківській системі або обліковій платформі. OLTP використовує нормалізовану структуру бази даних, що мінімізує дублювання інформації й дозволяє швидко змінювати дані з гарантією цілісності (через транзакції ACID).

На відміну від цього, OLAP-системи призначені для складного аналізу великих обсягів даних. Вони використовуються для створення звітів, аналітики, прогнозування і виведення закономірностей, часто в рамках бізнес-аналітики або наукових досліджень. OLAP-системи працюють з денормалізованими базами (часто у вигляді багатовимірних кубів або спеціалізованих сховищ даних), що дозволяє швидко обробляти запити до агрегованої інформації, але не підходить для частих змін даних.

Узагальнюючи, OLTP — це про оперативність і точність транзакцій, тоді як OLAP — про глибину і швидкість аналітики. Обидві системи можуть співіснувати в межах однієї інформаційної інфраструктури, обслуговуючи різні рівні прийняття

рішень: OLTP — оперативний, OLAP — стратегічний. Порівняльні характеристики можна предствити у вигляді табл. 3.2.

Таблиця 3.2 - Порівняльні характеристики OLTP та OLAP

Критерій	OLTP	OLAP
Призначення	Обробка транзакцій у реальному часі	Аналітична обробка даних для прийняття рішень
Користувачі	Кінцеві користувачі, оператори	Аналітики, менеджери, стратегічне керівництво
Типи операцій	INSERT, UPDATE, DELETE, SELECT (оперативні запити)	SELECT (складні, агреговані запити)
Обсяг даних	Невеликий до середнього	Великий, історичний
Час відповіді	Мілісекунди – секунди	Секунди – хвилини
Моделювання даних	Висока нормалізація (3NF і вище)	Денормалізація, зіркова/сніжинкова схема
Типи запитів	Простий, вузький фокус	Складний, багатовимірний аналіз
Актуальність даних	Поточні (оперативні)	Історичні, агреговані
Зміни в даних	Часті, інтенсивне оновлення	Рідкісні, переважно читання
Транзакції	Часто використовує транзакції з ACID-гарантіями	Не використовує транзакції або рідко
Приклад застосування	CRM-системи, банківські системи, e-commerce	Бізнес-аналітика, фінансове планування, дослідження ринку

Джерело: сформовано автором на основі виконаного дослідження

Таким чином, РБД MySQL є універсальним, стабільним і надійним інструментом зберігання структурованих даних, що поєднує формальну строгість з гнучкістю використання у прикладних системах різного рівня складності.

Додатковими сховищами даних можуть бути NoSQL бази даних. Порівняння NoSQL-сховищ даних слід почати з того, що вони розділяються на кілька основних типів: документо-орієнтовані, колоночні, ключ-значення та

графові. Кожен із них має специфічну модель зберігання даних і призначений для різних сценаріїв використання.

Документо-орієнтовані бази даних, такі як MongoDB чи CouchDB, працюють із напівструктурованими даними у форматі JSON або BSON. Вони ідеально підходять для зберігання складних об'єктів, які можуть змінювати структуру в процесі еволюції застосунку. Такі БД забезпечують гнучкість, масштабованість і потужний механізм запитів. Основний недолік — складність підтримки транзакцій, особливо між колекціями.

Колоночні сховища, наприклад Apache Cassandra або HBase, зберігають дані у вигляді таблиць, де основна увага зосереджена на колонках, а не на рядках. Це робить їх особливо ефективними для аналітичних задач, обробки великих обсягів даних і реалізації масштабованих розподілених систем. Проте вони зазвичай складніші в адмініструванні й мають обмежену підтримку складних операцій.

Брокери повідомлень (message brokers) — це проміжне програмне забезпечення, яке дозволяє компонентам розподіленої системи обмінюватися повідомленнями в асинхронному режимі. Вони виступають посередниками між відправником (producer) і одержувачем (consumer), забезпечуючи буферизацію, маршрутизацію, черги повідомлень, гарантію доставки, масштабованість та розділення відповідальностей між сервісами.

Основна роль брокера — прийняти повідомлення від одного модуля, зберегти його в черзі та передати іншому модулю або групі модулів тоді, коли вони готові його обробити. Це дозволяє уникнути прямої залежності між сервісами, покращує відмовостійкість системи і дає змогу легко додавати нові обробники або змінювати логіку взаємодії.

Можна виділити такі найбільш популярні брокери повідомлень:

- RabbitMQ;
- Apache Kafka;
- Redis (Pub/Sub);
- Amazon SQS.

RabbitMQ — це один із найпопулярніших брокерів повідомлень, який реалізує протокол AMQP (Advanced Message Queuing Protocol). Його основне призначення — забезпечення асинхронної взаємодії між різними компонентами програмної системи через надійне обмінювання повідомленнями. RabbitMQ активно застосовується у мікросервісних архітектурах, обробці подій у реальному часі, системах логістики, фінансових сервісах та в будь-яких проєктах, де критичною є черговість, гарантія доставки та масштабованість.

Архітектура RabbitMQ базується на таких основних поняттях:

- Producer (виробник) — відправник повідомлення;
- Consumer (споживач) — той, хто приймає і обробляє повідомлення;
- Queue (черга) — проміжне сховище, в яке надсилаються повідомлення;
- Exchange (обмінник) — маршрутизатор повідомлень, який приймає повідомлення від producer'ів і направляє їх у відповідні черги згідно з правилами;
- Binding (зв'язок) — правило, яке поєднує обмінник з чергою на основі ключа маршрутизації;
- Routing Key — мітка, яка допомагає вирішити, куди відправити повідомлення.

RabbitMQ підтримує наступні типи обмінників:

- Direct Exchange — маршрутизує повідомлення в чергу з точним відповідником routing key;
- Topic Exchange — дозволяє складні шаблони маршрутизації з використанням символів-узагальнень;
- Fanout Exchange — надсилає кожне повідомлення у всі прив'язані черги (бродкаст);
- Headers Exchange — маршрутизує повідомлення на основі заголовків, а не routing key.

Також RabbitMQ ряд певних особливостей, серед яких можна виділити надійність, гнучкість, масштабованість, підтримку різних протоколів, такі як AMQP, MQTT, STOMP, HTTP, плагіни та панель адміністратора.

Під надійністю розуміється підтримка підтвердження доставки (acknowledgements), стійкість до збоїв, можливість персистентного збереження повідомлень.

Гнучкість означає конфігурування черг, пріоритети, TTL (час життя повідомлення), dead-letter queues.

Масштабованість означає підтримування кластерів та дзеркальних черг для розподілення навантаження.

RabbitMQ має багату екосистему, включно з плагінами для моніторингу, авторизації, інтеграції з іншими сервісами.

Панель адміністратора надає веб-інтерфейс для моніторингу черг, обмінників, підключень та стану вузлів.

Сценарії використання rabbitmq включають сповіщення і нотифікації, черги задач для обробки повідомлень у фоновому режимі, event-driven архітектури з логікою на подіях, взаємодія між мікросервісами та лімітування навантаження між компонентами.

RabbitMQ добре підходить для систем, де важлива черговість обробки повідомлень, гнучкість маршрутизації, підтримка надійної доставки й інтеграція з різними мовами програмування (через клієнтські бібліотеки, наприклад PHP, Python, Java, C#, Go).

Apache Kafka — це розподілена стрімінгова платформа з відкритим кодом, яка використовується для побудови систем реального часу, де ключовими є висока пропускна здатність, низька затримка та надійна доставка повідомлень [13]. Kafka була розроблена у LinkedIn і пізніше передана до Apache Software Foundation як проєкт верхнього рівня. Її призначення — забезпечити публікацію, зберігання, обробку та підписку на потоки подій (streams of records), часто у режимі реального часу.

У програмному забезпеченні Apache Kafka можна виділити наступні ключові компоненти:

- Producer — відправник подій у Kafka;
- Consumer — споживач подій з Kafka;
- Topic — логічна категорія або канал, у який надсилаються повідомлення;
- Partition — фізичний поділ topic, що дозволяє масштабувати Kafka горизонтально;
- Broker — сервер Kafka, який зберігає повідомлення та обробляє запити;
- ZooKeeper (раніше обов'язковий компонент, але починаючи з Kafka 2.8 можлива робота без нього) — координує кластери Kafka;
- Consumer Group — набір споживачів, що працюють разом і спільно споживають записи з topic'ів.

У програмному забезпеченні Apache Kafka можна виділити наступні основні особливості:

- масштабованість;
- продуктивність;
- надійність;
- реальна обробка подій (stream processing);
- реплікація і стійкість до збоїв.

Kafka спроектована для горизонтального масштабування. Додавання нових брокерів або партицій відбувається без простоїв. Також Kafka здатна обробляти мільйони повідомлень на секунду завдяки асинхронному введенню/виведенню та збереженню на диск з оптимізацією через commit log.

Kafka дозволяє зберігати всі повідомлення на диску протягом заданого часу або до досягнення певного обсягу, з можливістю повторного читання, що відрізняє її від традиційних брокерів повідомлень. Інтеграція з Apache Kafka Streams та Apache Flink дозволяє обробляти події на льоту. Також в Kafka є підтримка гарантованої доставки повідомлень лише один раз, що раніше було значною

проблемою у стрімінгових системах. Kafka реплікує партиції між брокерами, що забезпечує відмовостійкість.

Apache Kafka має наступні сфери застосування: сповіщення і нотифікації, потокова аналітика, передача логів, моніторинг та аудит у реальному часі, фінансові транзакції та фрод-детекція та інтеграція мікросервісів. Також Kafka часто виступає як центральна шина даних (data bus), що поєднує десятки й сотні сервісів у великій розподіленій системі.

Redis Pub/Sub — це механізм публікації/підписки, вбудований у Redis — високопродуктивну, in-memory систему зберігання даних. Pub/Sub дозволяє клієнтам надсилати (publish) повідомлення до певних каналів, а інші клієнти, які підписані (subscribe) на ці канали, отримують ці повідомлення миттєво. Цей механізм дуже легкий, з низькою затримкою та мінімальним оверхедом, що робить Redis придатним для сценаріїв миттєвої доставки повідомлень.

Redis не зберігає повідомлення після їх надсилання — повідомлення доставляється лише активним підписникам каналу на момент публікації. Redis підтримує кілька основних команд:

- publish channel message;
- subscribe channel;
- psubscribe pattern;
- unsubscribe channel.

Amazon SQS (Simple Queue Service) — це керований сервіс черг повідомлень від AWS, який дозволяє безпечно передавати будь-які дані між компонентами розподілених програм, не вимагаючи відправника та одержувача працювати одночасно. Основне призначення SQS — асинхронна передача повідомлень з високою надійністю, масштабованістю та відмовостійкістю.

Серед основних характеристик Amazon SQS можна виділити наступні: повністю керований сервіс, висока доступність і надійність, підтримка dead letter queue, підтримка затримки та безпека.

Серед основних переваг сервісу черг Amazon SQS можна виділити наступні, такі як:

- масштабованість;
- надійність;
- інтеграція;
- мінімізація збоїв;
- гнучкість;
- оплата за використання.

Проте незважаючи на ряд позитивних характеристик, особливостей та переваг, Amazon SQS має наступні недоліки:

- обмежений функціонал у порівнянні з Kafka чи RabbitMQ, а саме відсутність складної маршрутизації чи широкомовних повідомлень;
- стандартна черга не гарантує порядок або унікальність повідомлень;
- обмеження розміру повідомлення, а саме 256 KB максимум, але більші можна зберігати через Amazon S3;
- можливі затримки через "visibility timeout", коли повідомлення тимчасово приховане після отримання, але не видалене.

Amazon SQS має наступні сценарії використання:

- сповіщення і нотифікації;
- обробка завдань у фоновому режимі;
- буферизація запитів до обмежених за ресурсами бекенд-сервісів;
- передача подій між мікросервісами;
- розділення модулів великих систем для підвищення надійності;
- запуск AWS Lambda по надходженню повідомлення.

Для обґрунтованого вибору брокера повідомлень, що найкраще відповідатиме вимогам системи нотифікацій, доцільно здійснити порівняльний аналіз їх функціональних характеристик, продуктивності та придатності до конкретних умов використання. Такий аналіз наведено в табл. 3.3.

Таблиця 3.3 – Аналіз брокерів повідомлень

Характеристика	RabbitMQ	Apache Kafka	Redis (Pub/Sub)	Amazon SQS
Модель обміну повідомленнями	Queue + Exchange (routing, topics, fanout)	Log-based (publish-subscribe, distributed log)	Простий publish-subscribe	Queue (Standard / FIFO)
Зберігання повідомлень	Так, з підтримкою підтверджень	Так, з можливістю повторного читання	Ні (миттєва доставка, без збереження)	Так (до 14 днів)
Порядок доставки	Може бути збережений (FIFO можливий)	Збережений	Не гарантується	FIFO тільки в спеціальній черзі
Масштабованість	Обмежена, потребує кластеризації	Висока, розрахований на Big Data	Обмежена	Дуже висока (керується AWS)
Надійність доставки	At-least-once, можливий exactly-once	At-least-once (залежить від конфігурації)	Немає гарантій доставки	At-least-once (або exactly-once для FIFO)
Затримка доставки (latency)	Дуже низька	Низька або середня (залежить від партицій)	Дуже низька (під мілісекунди)	Може бути вища через HTTP API
Придатність до нотифікацій	Висока – складна маршрутизація, retry, TTL	Середня – для масових подій, аналітики	Низька – немає збереження, складно масштабувати	Висока – проста інтеграція, висока надійність
Підтримка fanout (розсилка)	Є	Так (через кілька підписників на топик)	Так, автоматично	Частково (через інтеграцію з SNS або Lambda)

Продовження таблиці 3.3

Характеристика	RabbitMQ	Apache Kafka	Redis (Pub/Sub)	Amazon SQS
Простота інтеграції	Складна конфігурація, багато клієнтів	Потребує налаштування, знань розподілених систем	Дуже проста	Дуже проста, особливо в AWS-середовищі
Використання у хмарі	Можна розгорнути самостійно або через PaaS	Часто вимагає власного керування	Потребує власного інстансу Redis	Нативний хмарний сервіс
Підтримка затримки (delay)	Так	Ні	Ні	Так

Джерело: сформовано автором на основі виконаного дослідження

На підставі проведеного порівняльного аналізу було прийнято рішення обрати Redis як основний брокер повідомлень для розроблюваної інформаційної управляючої системи нотифікацій. Це зумовлено його високою швидкодією, простотою інтеграції, підтримкою механізму Pub/Sub у реальному часі, а також відсутністю складної інфраструктури, що дозволяє мінімізувати накладні витрати в межах даного проєкту. Крім того, Redis добре підходить для систем із помірним навантаженням, де критичним є швидке розповсюдження повідомлень із мінімальною затримкою, що повністю відповідає потребам обраної предметної області. Інфологічну модель системи наведено на рис.3.2.

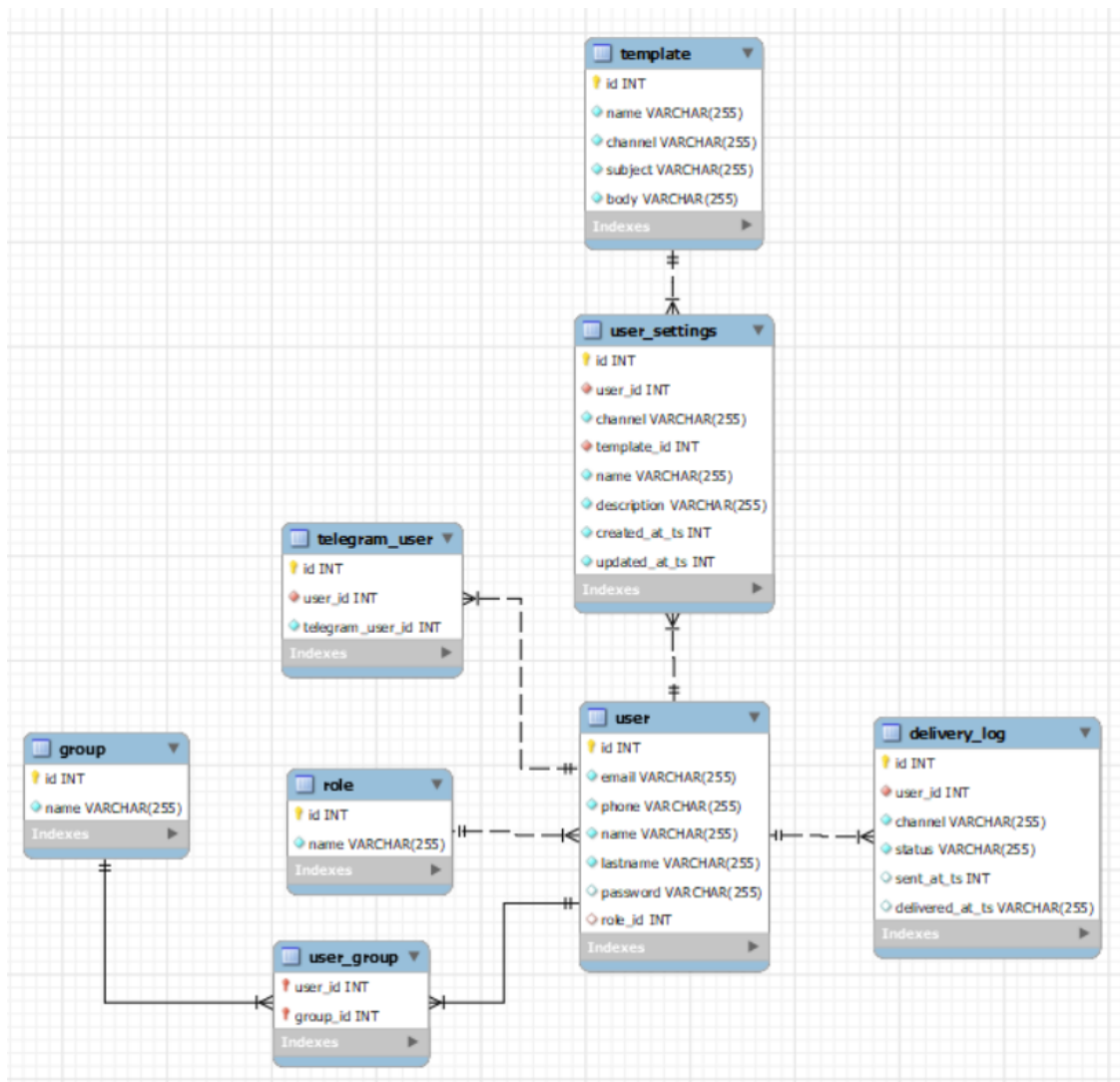


Рисунок 3.2 - Інфологічна модель системи

Джерело: розроблено автором

Фізична модель даних, яка демонструє, наведена на рис. 3.3.

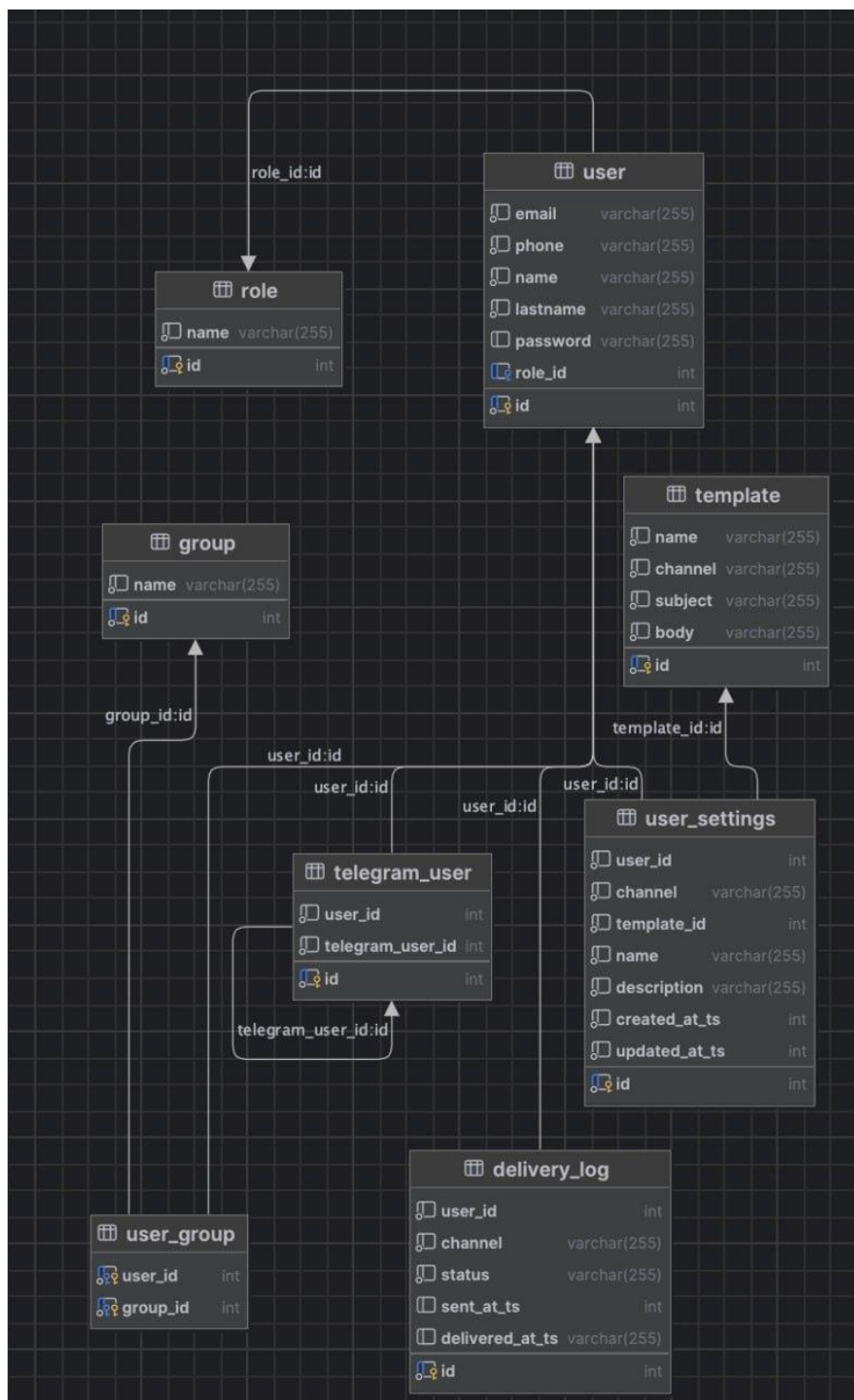


Рисунок 3.3 – Фізична модель даних

Джерело: розроблено автором

Нижче наведено структуру таблиць, необхідних для роботи інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій, які наведено на рис. 3.4. – 3.11.

```
-- auto-generated definition
create table delivery_log
(
    id            int auto_increment
    |            primary key,
    user_id       int            not null,
    channel       varchar(255) not null,
    status        varchar(255) not null,
    sent_at_ts    int            null,
    delivered_at_ts varchar(255) null
)

collate = utf8mb4_unicode_ci;
```

Рисунок 3.4 - Структура таблиці delivery_log

Джерело: розроблено автором

```
-- auto-generated definition
create table `group`
(
    id    int auto_increment
    |    primary key,
    name  varchar(255) not null
)

collate = utf8mb4_unicode_ci;
```

Рисунок 3.5 - Структура таблиці group

Джерело: розроблено автором

```
-- auto-generated definition
create table role
(
    id      int auto_increment
    |      primary key,
    name    varchar(255) not null
)

collate = utf8mb4_unicode_ci;
```

Рисунок 3.6 - Структура таблиці role

Джерело: розроблено автором

```
-- auto-generated definition
create table telegram_user
(
    id          int auto_increment
    |          primary key,
    user_id     int not null,
    telegram_user_id int not null
)

collate = utf8mb4_unicode_ci;
```

Рисунок 3.7 - Структура таблиці telegram_user

Джерело: розроблено автором

```
-- auto-generated definition
create table template
(
    id          int auto_increment
    |          primary key,
    name        varchar(255) not null,
    channel     varchar(255) not null,
    subject     varchar(255) not null,
    body        varchar(255) not null
)

collate = utf8mb4_unicode_ci;
```

Рисунок 3.8 - Структура таблиці template

Джерело: розроблено автором

```
-- auto-generated definition
create table user
(
    id          int auto_increment
    |          primary key,
    email       varchar(255) not null,
    phone       varchar(255) not null,
    name        varchar(255) not null,
    lastname    varchar(255) not null,
    password    varchar(255) null,
    role_id     int          null,
    constraint FK_8D93D649D60322AC
    |          foreign key (role_id) references role (id)
)

collate = utf8mb4_unicode_ci;

create index IDX_8D93D649D60322AC
on user (role_id);
```

Рисунок 3.9 - Структура таблиці user

Джерело: розроблено автором

```
-- auto-generated definition
create table user_group
(
    user_id int not null,
    group_id int not null,
    primary key (user_id, group_id),
    constraint FK_8F02BF9DA76ED395
        foreign key (user_id) references user (id)
        on delete cascade,
    constraint FK_8F02BF9DFE54D947
        foreign key (group_id) references `group` (id)
        on delete cascade
)
collate = utf8mb4_unicode_ci;

create index IDX_8F02BF9DA76ED395
on user_group (user_id);

create index IDX_8F02BF9DFE54D947
on user_group (group_id);
```

Рисунок 3.10 - Структура таблиці user_group

Джерело: розроблено автором

```
-- auto-generated definition
create table user_settings
(
    id int auto_increment
        primary key,
    user_id int not null,
    channel varchar(255) not null,
    template_id int not null,
    name varchar(255) not null,
    description varchar(255) not null,
    created_at_ts int not null,
    updated_at_ts int not null
)
collate = utf8mb4_unicode_ci;
```

Рисунок 3.11 - Структура таблиці user_settings

Джерело: розроблено автором

3.1.3 Контрольний приклад та обрахунок прогнозних обсягів БД

Прогнозування обсягів бази даних є важливим етапом у процесі проектування інформаційної системи, що дозволяє завчасно оцінити потреби у ресурсах та забезпечити масштабованість. У рамках даної інформаційної управляючої системи, яка автоматизує процес доставки нотифікацій, було здійснено попередню оцінку необхідного обсягу зберігання даних, виходячи з логічної моделі, типових сценаріїв використання та характеру зберезуваної інформації.

Розрахунок базувався на наборі ключових сутностей, а саме: User, UserSettings, UserGroup, Template та DeliveryLog. У якості основних параметрів оцінки використовувалися припущення щодо середньої кількості користувачів системи, кількості шаблонів повідомлень, частоти надсилання повідомлень, а також обсягу текстової інформації, що супроводжує кожен запис.

Найбільше навантаження на обсяг даних створює таблиця журналу доставок (DeliveryLog), оскільки кожен запис містить тип каналу, статус, дату надсилання, а також потенційні текстові повідомлення про помилки. В умовах середнього навантаження — до ста повідомлень на одного користувача — прогнозований обсяг лише цієї таблиці може досягати кількох гігабайт. Інші сутності, зокрема користувачі, групи, налаштування та шаблони, формують фоновий обсяг, що є незначним у порівнянні з обсягами журналу, але все ж має враховуватись у повній оцінці.

У результаті прогнозований базовий обсяг зберігання даних становить понад 5 ГБ, що, з урахуванням індексів, системних накладних витрат та майбутнього зростання обсягів даних, може потребувати резервування щонайменше 8–10 ГБ на початковому етапі розгортання системи. Ця оцінка підтверджує доцільність використання високопродуктивного сховища даних, а також передбачає потребу в реалізації механізмів архівації або ротації історичних записів у подальшій експлуатації системи.

Щоб продемонструвати роботу ІУС для автоматизації процесу зовнішніх нотифікацій, було виконано мануальний тест перевірки email-сповіщення під час реєстрації користувача в системі. На рис. 3.12. наведено сторінку реєстрації користувача.

Головна сторінка

Регістрація

Ім'я
Mykolai

Прізвище
Stepanenko

Електронна пошта
Nikolaua36@gmail.com

Номер телефону
+380980191919

Пароль
.....

Повторення паролю
.....

[Зареєструватися](#)

Вже маєте акаунт? [Увійти!](#)

Рисунок 3.12 - Сторінка реєстрації користувача

Джерело: розроблено автором

Після реєстрації користувача, можна побачити отриманий лист на електронній пошті, а також створення запису щодо доставки сповіщення користувачу у таблиці delivery_log. Результати тестування наведені на рис. 3.11 – 3.13.

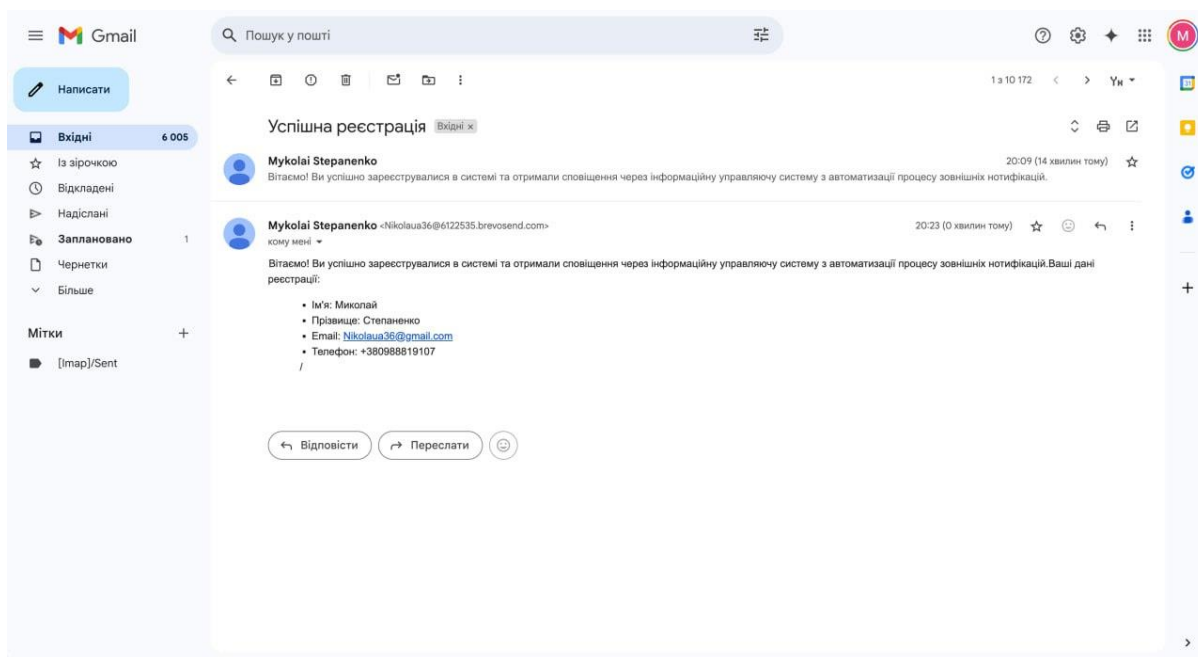


Рисунок 3.13 - Результат отриманого сповіщення з результатами реєстрації користувача

Джерело: розроблено автором

delivery_log [diploma@localhost]		
WHERE		
		1
id	▼	1
user_id	↕	86
channel	↕	email
status	↕	delivered
sent_at_ts	↕	1746984217
delivered_at_ts	↕	1746984217

Рисунок 3.14 - Створення запису щодо успішного надсилання нотифікації користувачу

Джерело: розроблено автором

3.2 Проєктування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних

3.2.1 Реалізація інтелектуального аналізу великих даних з метою наповнення бази знань

З огляду на необхідність систематичного виявлення прихованих закономірностей, патернів і залежностей у великих обсягах вхідних даних, для наповнення бази знань інформаційної системи було реалізовано комплекс заходів з інтелектуального аналізу даних (ІАД) [41]. У контексті цієї роботи основний акцент було зроблено на використанні доступних онлайн-платформ, які забезпечують повноцінну функціональність для виконання ключових етапів глибинної аналітики, без потреби у локальній розробці чи інтеграції з внутрішніми сервісами системи.

Процес ІАД був реалізований у відповідності до міжгалузевого стандарту CRISP-DM [42], що передбачає поетапне виконання таких стадій: розуміння задачі, ознайомлення з даними, їх попередня обробка, побудова аналітичних моделей, оцінка результатів та їх подальше використання. Для кожного з етапів використовувалися онлайн-сервіси, що забезпечують необхідні засоби трансформації даних, побудови моделей та візуалізації результатів. Попереднє очищення, нормалізація, скорочення розмірності та візуалізація даних виконувалися на базі хмарної платформи Google Colab [43].

Таким чином, повний цикл ІАД у рамках цієї роботи був реалізований з використанням доступних онлайн-інструментів, що дозволило уникнути потреби у розробці або локальній інтеграції модулів аналізу, а також забезпечити стандартизованість і прозорість усіх процедур відповідно до вимог CRISP-DM. Отримані аналітичні результати стали підґрунтям для автоматизованого формування бази знань системи.

Процес проєктування БЗ [44] традиційно складається з шести послідовних етапів. Перший з них — ідентифікація бази знань — передбачає чітке визначення

задач, які має вирішувати ІУС, а також їхню неформальну декомпозицію на підзадачі. Описуються ключові поняття, вхідні та вихідні дані, знання, що підлягають формалізації, ситуації, які ускладнюють розв'язання задач, і приклади очікуваних рішень. Цей етап формує цілі побудови БЗ, серед яких: автоматизація рутинних процесів, підвищення якості рішень, створення механізмів для багаторазового використання знань. Також плануються ресурси — зокрема джерела знань, час, інструменти та ролі учасників команди (експерти, інженери знань, користувачі).

Другий етап — концептуалізація БЗ — базується на аналізі предметної області з метою побудови концептуальної моделі. Виявляються основні поняття та зв'язки між ними: ієрархічні, причинно-наслідкові, просторові чи функціональні. Визначається структура знань, типи задач, які покриває система, рівень деталізації, обмеження та вимоги до інтерпретації. Результатом цього етапу є узгоджена схема, яка може бути представлена у вигляді онтології або логічної структури. Для цього доцільно використовувати онлайн-середовища моделювання, зокрема WebProtégé, яке дає змогу створити ієрархію понять, встановити властивості об'єктів і зв'язки, сформувати загальний каркас для наступних етапів.

На третьому етапі — формалізації БЗ — здійснюється перехід від концептуального рівня до формального подання знань. Описуються структури даних і методи логічного виводу, обирається формат подання знань (правила продукцій, фрейми, семантичні мережі, онтології). Формується простір станів і визначається механізм навігації цим простором. У разі застосування OWL або RDF для формального подання, зручно використовувати онлайн-редактори, які автоматично валідують онтологію, дозволяють виконувати запити SPARQL і перевіряти логічну узгодженість моделі. Це забезпечує швидку інтеграцію з іншими системами або модулями, орієнтованими на семантичну обробку інформації.

Четвертий етап — виконання (реалізація) БЗ — передбачає створення прототипу бази знань шляхом її наповнення сутностями, атрибутами, правилами та прикладами. Знання можуть бути систематизовані на основі результатів аналізу

даних або доменної експертизи, використовуючи раніше визначені моделі. Для цього ефективним є залучення сервісів на кшталт OntoText Platform [45], GraphDB [46] або інших середовищ, які дозволяють імпортувати дані, зберігати їх у вигляді графів та організовувати пошук за логічними правилами. Завданням етапу є побудова дієвої структури, яка буде здатна підтримувати процеси логічного висновку, фільтрації та інтерпретації знань відповідно до потреб користувача.

П'ятий етап — тестування — полягає у перевірці логічної узгодженості знань, функціональності запитів та здатності бази генерувати або оцінювати рішення. Використовується репрезентативна вибірка задач з проблемної області, проводяться сесії інтерактивного тестування з використанням онлайн-інтерфейсів або консольних інструментів SPARQL/OWL [47]. Важливим аспектом цього етапу є аналіз пояснень, які система надає для прийнятих рішень — таким чином перевіряється прозорість логіки та доступність знань для інтерпретації.

Останній, шостий етап — дослідна експлуатація — охоплює перевірку зручності та ефективності використання системи кінцевими користувачами. Аналізується інтерфейс, швидкодія, релевантність результатів, простота адаптації системи до нових умов. За результатами цього етапу можуть бути виявлені потреби в коригуванні моделі або в розширенні обсягу знань.

Крім класичної методології, у контексті проекту можливе застосування підходу MDA (Model-Driven Architecture) [48], який дозволяє будувати базу знань через створення множини моделей: CIM — для опису вимог до системи без урахування обчислювального середовища, PIM — для специфікації логіки незалежно від платформи реалізації, PSM — для пристосування до конкретної технологічної бази, та PDM — для опису особливостей обраної платформи. Кожна з моделей може бути реалізована за допомогою відповідних візуальних або текстових редакторів у хмарному середовищі (наприклад, GenMyModel, Modelio Web, Draw.io), що спрощує розгортання та актуалізацію знань у процесі проєктування.

Отже, база знань у контексті ІУС виступає як структурований і формалізований ресурс, що акумулює досвід, алгоритми і логічні зв'язки, необхідні

для функціонування інтелектуальної системи. Реалізація підходу через онлайн-інструменти відкриває можливості для масштабування, обміну знаннями, повторного використання компонентів та прискореного розгортання нових функціональних модулів у системі.

3.2.2 Проектування та реалізація бази знань інформаційної управляючої системи.

Проектування та реалізація бази знань (БЗ) є ключовим етапом побудови інформаційної управляючої системи (ІУС), що орієнтована на інтелектуальну обробку інформації та підтримку прийняття рішень. Формування БЗ потребує системного підходу, який охоплює аналіз проблемної області, структурування знань, їхню формалізацію та забезпечення функціональності у складі ІУС. У сучасних умовах особливу увагу приділяють використанню онлайн-інструментів, які забезпечують доступність, масштабованість і простоту інтеграції знань без необхідності розробки складного програмного забезпечення. База знань, яка призначена для роботи з користувачами наведена на рис. 3.15.

	id	email	phone	name	lastname	password	role_id
▶	71	behew...	+18143...	Vincent	Schaefer	\$2y\$10\$D...	1
	72	pineda...	+18143...	Bertie	Jacobson	\$2y\$06\$Po...	1
	73	norepl...	+15852...	Owain	Frazier	\$2y\$06\$sc...	2
	74	Khalid...	+15853...	Khalid	Armstrong	\$2y\$06\$E...	1
	75	Henrie...	+15853...	Henrietta	Winters	\$2y\$06\$thl...	2
	76	Madeli...	+18143...	Madeline	Gibbs	\$2y\$06\$O...	2
	77	Jacob...	+18143...	Jacob	David	\$2y\$06\$gd...	1
	78	Rosem...	+15852...	Rosem...	Slater	\$2y\$06\$bb...	2
	79	Luqma...	+14842...	Luqman	Clements	\$2y\$06\$ba...	2
	80	Ewan...	+18143...	Ewan	Sparks	\$2y\$06\$W...	1

Рисунок 3.15 – База знань користувачів

Джерело: розроблено автором

Також інформаційна управляюча система з автоматизації процесу зовнішніх нотифікацій використовує базу знань для роботи з нотифікаціями у вигляді журналу нотифікацій. База знань журналу нотифікацій наведена на рис. 3.16.

	id	user_id	channel	status	sent_at_ts	delivered_at_ts
▶	1	71	email	DELIVERED	1572890	1572891
	2	72	email	DELIVERED	1572890	1572891
	3	73	sms	DELIVERED	1572890	1572891
	4	74	telegram	DELIVERED	1572890	1572891
	5	75	email	DELIVERED	1572890	1572891
	6	76	sms	DELIVERED	1572890	1572891
	7	77	sms	DELIVERED	1572890	1572891
	8	78	email	DELIVERED	1572890	1572891
	9	79	email	DELIVERED	1572890	1572891
	10	80	email	DELIVERED	1572890	1572891

Рисунок 3.16 - База знань журналу нотифікацій

Джерело: розроблено автором

3.3 Програмне забезпечення

Для розробки інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій слід обрати архітектуру, згідно якої буде розроблена система. Серед запропонованих на розгляд архітектур можна обрати наступні:

- Цибулева архітектура;
- Чиста архітектура;
- Гексагональна архітектура.

Цибулева (Onion) [49], Чиста (Clean) [50] та Гексагональна (Hexagonal або Ports & Adapters) [51] архітектури — це три споріднені підходи до побудови програмних систем із високим рівнем ізоляції бізнес-логіки від інфраструктурного

коду. Усі три архітектури розв’язують проблему надмірної залежності домену від зовнішніх технологій, але роблять це трохи по-різному.

Цибулева архітектура (Onion Architecture) — це архітектурний підхід, що фокусується на захисті бізнес-логіки від змін у зовнішній інфраструктурі. Його впровадження особливо актуальне в системах, де важлива масштабованість, тестованість і стійкість до змін технологій (наприклад, БД, API, брокери тощо). Діаграма цибулевої архітектури наведена на рис. 3.17.

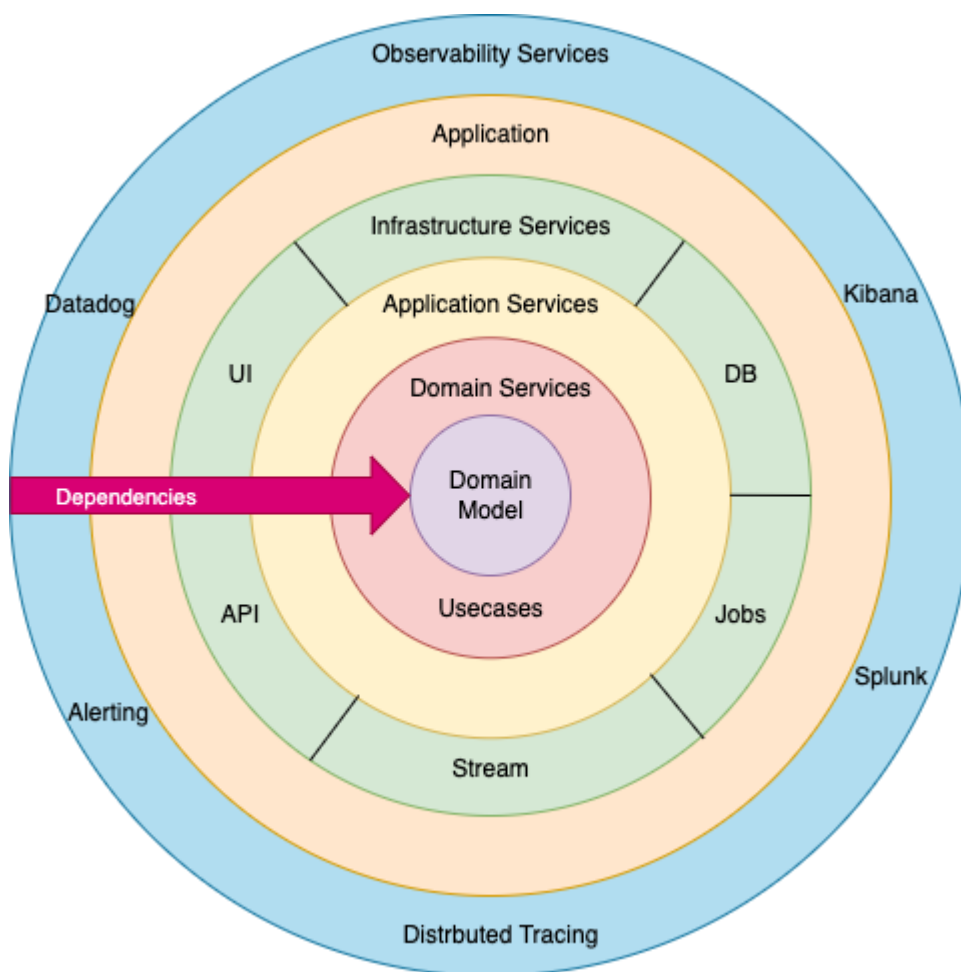


Рисунок 3.17 - Діаграма цибулевої архітектури

Джерело: https://miro.medium.com/v2/resize:fit:962/1*Pd6l7Za_Pl8BI-kltDQukg.png

У центрі цибулевої архітектури — доменна модель, яка не має жодних зовнішніх залежностей. Навколо цієї моделі нарощуються шари, що реалізують прикладну логіку, сервіси, інтерфейси та зовнішні адаптери. Зовнішні шари можуть

залежати від внутрішніх, але не навпаки — це забезпечується через інверсію залежностей (Dependency Inversion Principle). Для цибулевої архітектури можна виділити наступні типові шари:

- Ядро (Domain Model);
- Доменні сервіси (Domain Services);
- Application Services;
- Infrastructure / External Layers.

Ядро (domain model) цибулевої архітектури містить сутності та їх поведінку. Тут немає залежностей ні на БД, ні на HTTP, ні на що інше. Це — серце системи. Наприклад, User, Order, Invoice — з відповідними методами та інваріантами.

Доменні сервіси (domain services) призначені для логіки, яка не прив'язана до конкретної сутності (наприклад, складна взаємодія кількох доменних об'єктів).

На рівні application services є реалізація use-case'ів. Вони використовують доменні сутності та сервіси, але не залежать від деталей зберігання даних, мережі тощо. Тут описується прикладна логіка: що має відбуватися при створенні користувача, оформленні замовлення тощо.

Infrastructure / External Layers – це реалізація інтерфейсів (контрактів), які визначені у внутрішніх шарах. Тут зберігаються репозиторії (реалізація через ORM, SQL), REST-контролери, драйвери брокерів повідомлень, логери тощо.

Узагальнити вище описану інформацію щодо цибулевої архітектури та виокремити переваги та недоліки даної архітектури, можна зобразити у вигляді таблиці (табл. 3.1).

Таблиця 3.4 - Переваги та недоліки цибулевої архітектури

Переваги	Недоліки
Стійкість до змін: внутрішня логіка не залежить від інфраструктури.	Складність реалізації: потребує глибшого розуміння архітектурних принципів.
Тестованість: легко писати юніт-тести для бізнес-логіки.	Велика кількість шарів: додає суттєву кількість інтерфейсів та коду.
Гнучкість у заміні технологій: можна змінити БД, брокер повідомлень тощо.	Зайва складність для простих проєктів: не завжди виправдана для CRUD-застосунків.
Чіткий поділ відповідальностей між шарами.	Більше часу на проектування та реалізацію.
Забезпечує модульність і розширюваність.	Потреба в ретельному управлінні залежностями.

Джерело: сформовано автором на основі виконаного дослідження

Підсумовуючи описані факти, цибулева архітектура — це чудове рішення для систем, які розвиваються, масштабуються або мають високі вимоги до якості бізнес-логіки.

Чиста архітектура (Clean Architecture) — це архітектурний підхід, запропонований Робертом С. Мартіном, який описує спосіб організації програмного забезпечення так, щоб забезпечити незалежність бізнес-логіки від зовнішніх залежностей (UI, бази даних, фреймворків тощо). Діаграму чистої архітектури наведено на рис. 3.18.

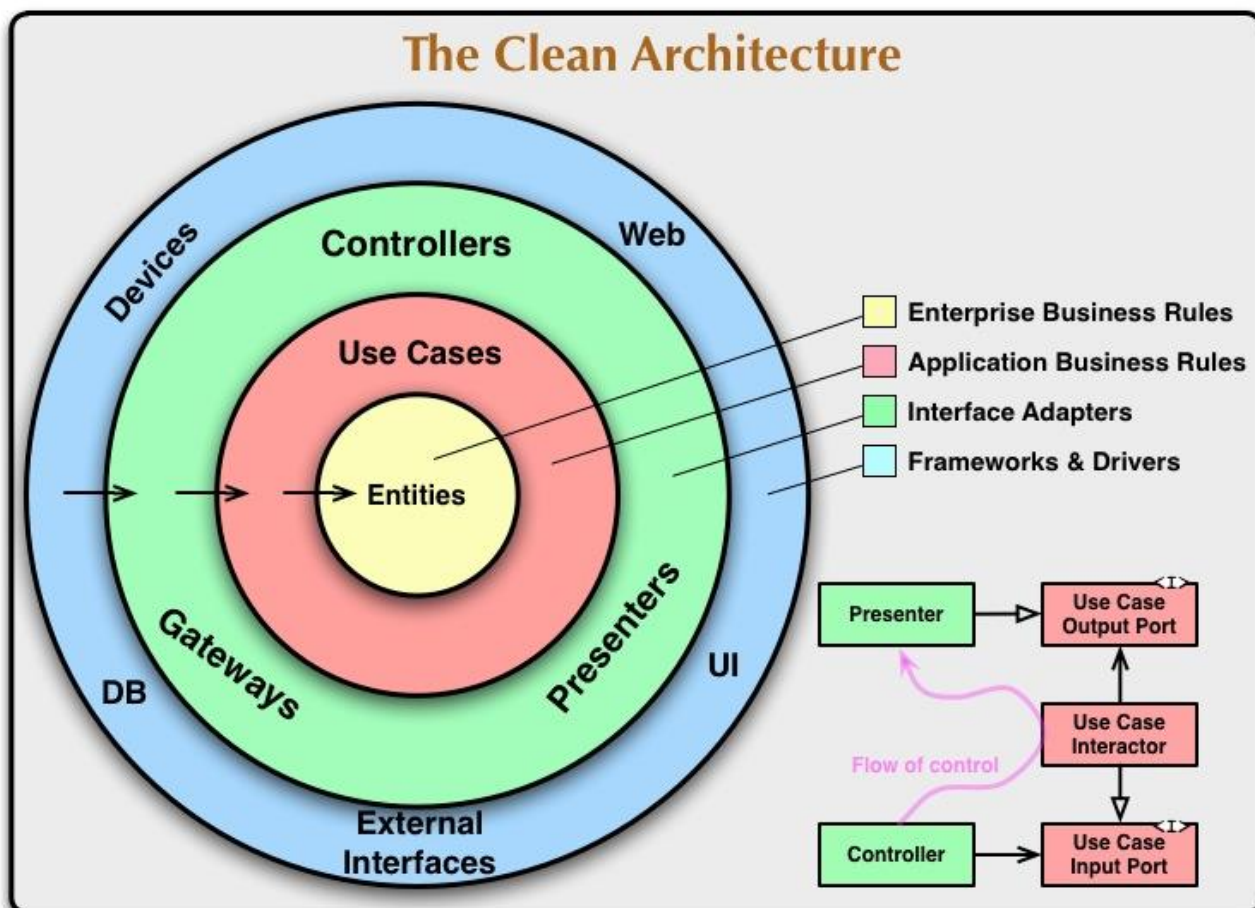


Рисунок 3.18 - Діаграма чистої архітектури

Джерело: [https://encrypted-](https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRu_3Gn5VDMg8qxbV3BrJ-dcIsc6gO4-SS8xA&s)

[tbn0.gstatic.com/images?q=tbn:ANd9GcRu_3Gn5VDMg8qxbV3BrJ-dcIsc6gO4-SS8xA&s](https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRu_3Gn5VDMg8qxbV3BrJ-dcIsc6gO4-SS8xA&s)

Основна мета — зробити систему легкою для тестування, розширення та підтримки. Основні принципи чистої архітектури наступні:

- Централізація бізнес-логіки;
- Шари з чітким розділенням;
- Інверсія залежностей.

Централізація бізнес-логіки описує, що у центрі архітектури знаходяться сутності (Entities) — об'єкти предметної області, які реалізують основні бізнес-правила. Вони не залежать ні від яких зовнішніх систем чи бібліотек.

Шари з чітким розділенням описують, що система поділяється на концентричні кола (шари), де кожен внутрішній шар має вищий рівень абстракції,

а зовнішні — нижчий. Зовнішні шари можуть залежати від внутрішніх, але не навпаки. Зовнішні шари — це контролери, бази даних, UI, сторонні сервіси.

Інверсія залежностей декларує, що комунікація між шарами відбувається через абстракції (інтерфейси), реалізація яких передається зовнішнім шаром. Це забезпечує слабку зв'язність і дозволяє легко змінювати реалізації.

Також чиста архітектура містить наступну типову структуру шарів:

- Entities — сутності предметної області.
- Use Cases (Interactors) — сценарії використання, що описують, як користувач взаємодіє із системою.
- **Interface Adapters** — адаптери, які перетворюють дані від зовнішніх систем до формату, зручного для бізнес-логіки.
- **Frameworks & Drivers** — UI, база даних, веб-сервери, брокери повідомлень тощо.

Узагальнити вище описану інформацію щодо чистої архітектури та виокремити переваги та недоліки даної архітектури, можна зобразити у вигляді таблиці (табл. 3.5).

Таблиця 3.5 - Переваги та недоліки чистої архітектури

Переваги	Недоліки
Незалежність від інфраструктури — логіка не залежить від бази даних, UI, фреймворків	Велика кількість абстракцій — збільшує складність розуміння структури
Тестованість — бізнес-логіку легко покрити юніт-тестами без потреби запуску сервісів	Початкові витрати — потребує більше часу на стартову реалізацію архітектури
Гнучкість — легко змінювати технології або реалізації (наприклад, змінити MySQL на Redis)	Надмірність для простих/малих проєктів, де така абстракція є зайвою

Продовження таблиці 3.5

Переваги	Недоліки
Зрозумілий розподіл відповідальностей між компонентами системи	Зростання кількості файлів/класів — може ускладнювати навігацію в проєкті
Масштабованість — добре підходить для великих і довгострокових проєктів	Необхідність досвіду — розробникам потрібно добре розуміти принципи архітектурного дизайну
Легка підтримка — ізолюваність дозволяє вносити зміни без ризику порушити інші частини	Менше зручності для швидкої розробки MVP

Джерело: сформовано автором на основі виконаного дослідження

Підсумовуючи описані факти, чиста архітектура — це чудове рішення для систем, які розвиваються, масштабуються або мають високі вимоги до якості бізнес-логіки.

Гексагональна архітектура (Hexagonal Architecture), також відома як архітектура портів і адаптерів (Ports and Adapters), була запропонована Аластером Кокберном (Alistair Cockburn). Її основна мета — ізоляція ядра додатка (бізнес-логіки) від зовнішніх залежностей через чітко визначені інтерфейси. Це дозволяє досягти високої гнучкості, тестованості та адаптивності до змін зовнішнього середовища.

В даній архітектурі ядро системи (доменна логіка) взаємодіє із зовнішнім світом (UI, БД, API тощо) через "порти", які описують, що система може або повинна робити. "Адаптери" реалізують ці порти для конкретних технологій або інтерфейсів, забезпечуючи ізоляцію логіки від конкретних інструментів реалізації. Основними компонентами даної архітектури є:

- доменна логіка (application core) — головна частина, що вирішує бізнес-задачі;

- порти (ports) — інтерфейси, які описують взаємодію ядра з іншими частинами;
- адаптери (adapters) — реалізації портів, які з'єднують ядро з зовнішнім світом (наприклад, rest api, cli, база даних, ui).

Ключовими ідеями гексагональної архітектури є те, що вона не орієнтована на шари, як у цибулевій або чистій, а на взаємодію через інтерфейси. Будь-який тип клієнта (UI, скрипт, тест, HTTP-клієнт тощо) підключається через адаптери, не змінюючи ядро. А також зовнішні залежності можна легко замінити, не зачіпаючи основну логіку. Діаграма гексагональної архітектури наведена на рис. 3.19.

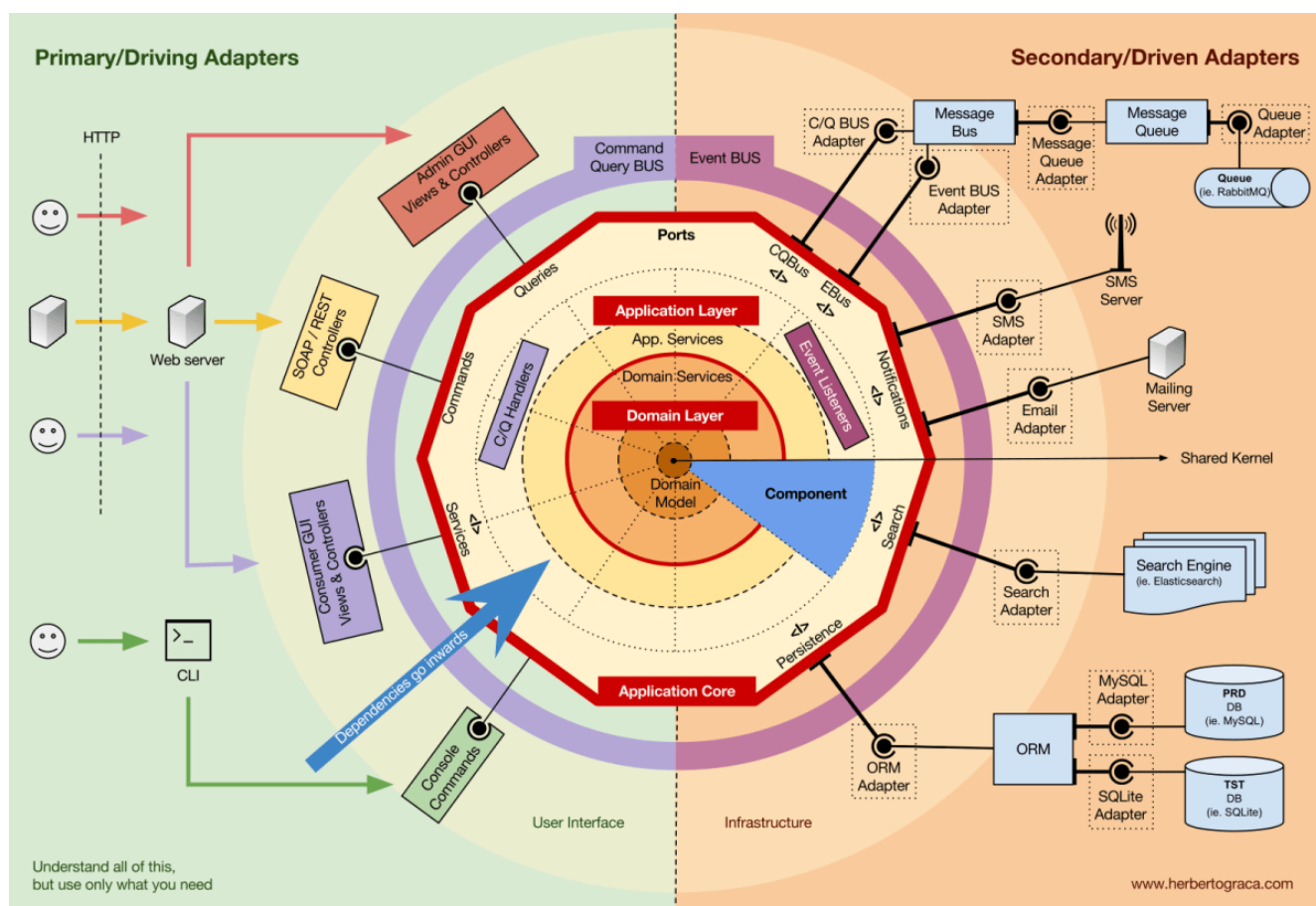


Рисунок 3.19 - Діаграма гексагональної архітектури

Джерело: <https://i0.wp.com/herbertograca.com/wp-content/uploads/2018/11/100-explicit-architecture-svg.png?fit=1200%2C821&ssl=1>

Узагальнити вище описану інформацію щодо гексагональної архітектури та виокремити переваги та недоліки даної архітектури, можна зобразити у вигляді таблиці (табл. 3.6).

Таблиця 3.6 - переваги та недоліки гексагональної архітектури

Переваги	Недоліки
Чітке розділення відповідальностей між ядром системи та зовнішніми компонентами	Ускладнення проєктування, особливо для малих або простих додатків
Полегшує тестування: ядро легко тестувати без зовнішніх залежностей	Вимагає додаткових зусиль для написання інтерфейсів (портів) і адаптерів
Забезпечує високу гнучкість і можливість легко змінювати технології (наприклад, замінити БД або інтерфейс користувача)	У нових командах потребує часу на навчання через незвичний підхід
Покращує підтримку та масштабованість системи у довгостроковій перспективі	Зростає кількість файлів та абстракцій, що ускладнює навігацію
Сприяє багаторазовому використанню ядра системи в різних сценаріях (API, CLI, тести)	Може бути надмірною для CRUD-додатків без складної логіки
Полегшує впровадження Domain-Driven Design (DDD)	Не всі популярні фреймворки підтримують її «з коробки»

Джерело: сформовано автором на основі виконаного дослідження

Після всебічного аналізу трьох архітектурних підходів — цибулевої, чистої та гексагональної — можна дійти до обґрунтованого висновку щодо доцільності використання саме **цибулевої архітектури** у проєкті.

Цибулева архітектура забезпечує чітке відокремлення бізнес-логіки від інфраструктурних компонентів, зберігаючи при цьому відносну простоту структури. Вона дозволяє будувати систему з внутрішнім ядром, яке мінімально залежить від зовнішнього світу, що сприяє легкому тестуванню, розширенню і підтримці. У порівнянні з чистою архітектурою, цибулева є менш абстрактною і більш практично орієнтованою, що знижує поріг входу для розробників та зменшує час на реалізацію. На відміну від гексагональної, вона не потребує створення великої кількості адаптерів і портів, що особливо важливо в контексті проєктів із середньою складністю.

Таким чином, обрання **цибулевої архітектури** є оптимальним компромісом між модульністю, зрозумілістю, підтримуваністю та швидкістю розробки, що цілком відповідає вимогам даної інформаційної управляючої системи.

У процесі розроблення веб-орієнтованої інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій одним із ключових аспектів є вибір мови програмування, що безпосередньо впливає на ефективність розробки, масштабованість рішення, швидкодію, безпеку та підтримуваність коду. У цьому підрозділі буде здійснено аналіз сучасних мов програмування, що найчастіше застосовуються у сфері веб-розробки, з урахуванням їхніх функціональних можливостей, екосистеми, сумісності з архітектурними підходами, а також відповідності вимогам обраної системної архітектури та інфраструктури.

Мови програмування, що використовуються для розробки веб-систем, відрізняються між собою за низкою критеріїв — рівнем абстракції, парадигмою програмування, продуктивністю, зручністю розробки, екосистемою інструментів та бібліотек, а також підтримкою веб-фреймворків. У даному аналізі розглянуто найбільш поширені мови програмування для веб-розробки: JavaScript/TypeScript, PHP, Python, Java, C#, Ruby та Go.

JavaScript / TypeScript. JavaScript — це ключова мова для фронтенд-розробки, яка у поєднанні з фреймворками типу React, Angular чи Vue.js дозволяє створювати інтерактивні інтерфейси. Завдяки Node.js JavaScript став популярним і

на боці серверу, що забезпечує однорідність стека технологій. TypeScript, як надбудова над JavaScript, додає статичну типізацію, що значно підвищує надійність великих проєктів.

PHP. PHP є класичною мовою серверної частини вебу, яка широко застосовується завдяки простоті інтеграції з HTML, великій кількості CMS та фреймворків (Laravel, Symfony). Сильними сторонами є зрілість екосистеми та велика спільнота. PHP ефективний у розробці високонавантажених веб-проєктів, зокрема з широким використанням шаблонних рішень.

Python. Python активно використовується у бекенд-розробці завдяки таким фреймворкам, як Django і Flask. Його лаконічний синтаксис, велика кількість бібліотек і підтримка різноманітних парадигм роблять його хорошим вибором для швидкого створення прототипів та підтримки AI-компонентів, що дедалі частіше інтегруються у веб-системи.

Java. Java є одним із найстабільніших рішень у розробці корпоративних веб-додатків. Завдяки фреймворкам Spring Boot, Jakarta EE тощо, Java пропонує надійну багаторівневу архітектуру, високу продуктивність і масштабованість, що особливо актуально для критично важливих систем з високими вимогами до безпеки.

C#. C# у поєднанні з .NET Core або ASP.NET є популярним рішенням для розробки як внутрішніх корпоративних систем, так і публічних веб-додатків. Мова має добре розвинуту типову систему, багату стандартну бібліотеку та підтримку масштабування. Особливо ефективна у середовищах, де використовується Windows-інфраструктура.

Ruby. Ruby, разом із фреймворком Ruby on Rails, зробив внесок у філософію "convention over configuration". Мова підходить для швидкої розробки MVP та стартапів, однак останніми роками поступається місцем іншим рішенням через меншу продуктивність і зниження популярності.

Go. Go (Golang), розроблений Google, вирізняється високою продуктивністю, простотою синтаксису та зручною паралельною обробкою

завдань. Він ідеально підходить для мікросервісної архітектури, REST API та систем, де важлива надійність, але не критична складність бізнес-логіки.

У результаті проведеного аналізу мов програмування, що застосовуються для розробки веб-систем, з урахуванням вимог до інформаційної управляючої системи, було прийнято рішення використовувати мову PHP. Такий вибір обумовлений кількома вагомими чинниками. По-перше, PHP має високу ступінь зрілості, активно підтримується спільнотою та має багатий набір стабільних фреймворків, зокрема Symfony, який дозволяє впроваджувати сучасні архітектурні підходи. По-друге, мова забезпечує гнучкість та адаптивність, що важливо для реалізації різнорівневих підсистем і взаємодії з повідомленнями, базами даних та зовнішніми API. По-третє, PHP має невисокий поріг входу, що сприяє швидкій інтеграції нових розробників у команду та спрощенню супроводу системи. Усі ці фактори роблять PHP доцільним вибором для проєкту з розробки масштабованої, функціонально насиченої та ефективної ІУС.

Для ефективної розробки програмного забезпечення мовою PHP існує низка інтегрованих середовищ розробки (IDE), кожне з яких має свої особливості, функціональні переваги та цільові аудиторії. Найпоширенішими є наступні інструменти: Visual Studio Code, NetBeans, Eclipse PDT, Sublime Text, PhpStorm.

Visual Studio Code є безкоштовним, легким редактором з широкою екосистемою розширень, зокрема для PHP. Його перевагами є висока швидкодія, гнучкість налаштувань та активна підтримка спільноти. Проте для повноцінної розробки складних проєктів із розвиненим автодоповненням, дебагінгом та аналізом коду потрібне встановлення багатьох додаткових модулів, що потребує додаткової конфігурації.

NetBeans і Eclipse PDT є класичними безкоштовними IDE, які підтримують PHP "з коробки", але мають менш зручний інтерфейс і помітно нижчу швидкодію. Вони підходять для навчання або невеликих проєктів, однак поступаються сучасним комерційним рішенням за глибиною інтеграції з системами контролю версій, засобами рефакторингу та продуктивністю.

Sublime Text позиціонується як потужний редактор коду, але по суті залишається текстовим редактором з розширеннями, тому не може забезпечити повноцінну підтримку складних архітектурних рішень або інтеграцій у порівнянні з професійними IDE.

PhpStorm від JetBrains є повноцінним середовищем розробки, спеціалізованим саме на PHP. Воно забезпечує глибокий аналіз коду, автоматичне завершення з урахуванням контексту, інтеграцію з системами контролю версій, відлагодження, підтримку Docker, баз даних, фреймворків (Symfony, Laravel тощо) та сучасних фронтенд-технологій. PhpStorm також підтримує стандарти кодування, має зручну систему навігації по проєкту та високий рівень стабільності. Незважаючи на те, що PhpStorm є комерційним продуктом, його можливості значно перевершують безкоштовні альтернативи в контексті розробки масштабованої інформаційної системи.

З огляду на вищезазначене, для розробки інформаційної управляючої системи доцільно обрати середовище розробки **PhpStorm**, оскільки воно забезпечує найвищий рівень підтримки екосистеми PHP, сприяє підвищенню продуктивності розробників, мінімізує ризики виникнення помилок і є найкращим вибором для реалізації комплексних рішень з використанням сучасних архітектурних підходів.

Для повного розуміння структури програмного забезпечення та взаємодії між його окремими модулями доцільно використовувати діаграму компонентів. Вона слугує інструментом візуалізації, що відображає логічні частини системи, їхні залежності та способи взаємодії між собою. Діаграма компонентів дозволяє показати, які саме компоненти реалізують функціональність, як вони організовані, які інтерфейси надають для взаємодії з іншими частинами системи та які зовнішні залежності мають.

Побудована діаграма компонентів відображає модульну архітектуру, у якій кожен компонент відповідає за певну логічну підсистему, підтримує чітко визначені інтерфейси взаємодії та забезпечує ізоляцію реалізації. Такий підхід

сприяє дотриманню принципів інженерії програмного забезпечення та полегшує як супровід, так і подальший розвиток системи.

Для забезпечення якісного, масштабованого й підтримуваного програмного забезпечення критично важливим є правильне проєктування системи ще на етапі архітектурного планування. Одним із ключових підходів, що дозволяє досягти цієї мети, є використання принципів SOLID — сукупності п'яти фундаментальних принципів об'єктно-орієнтованого програмування, спрямованих на поліпшення модульності, розширюваності та тестованості коду.

Принцип Single Responsibility Principle (SRP) вимагає, щоб кожен клас мав лише одну відповідальність. Це сприяє спрощенню коду та полегшує його супровід, оскільки зміни в одній сфері функціональності не впливають на інші.

Принцип Open/Closed Principle (OCP) передбачає, що програмні сутності мають бути відкритими для розширення, але закритими для модифікації. Це дозволяє додавати нову поведінку без змін до вже існуючого коду, що особливо важливо для стабільності великих систем.

Принцип Liskov Substitution Principle (LSP) означає, що об'єкти підкласів повинні без проблем замінювати об'єкти батьківських класів. Це забезпечує коректну поведінку програми при використанні поліморфізму і дозволяє створювати узгоджені ієрархії класів.

Принцип Interface Segregation Principle (ISP) наголошує на необхідності створювати вузько спрямовані інтерфейси, специфічні для окремих клієнтів, замість одного загального. Завдяки цьому класи залежать лише від методів, які вони дійсно використовують, що сприяє більш гнучкому і незалежному проєктуванню компонентів.

Принцип Dependency Inversion Principle (DIP) вимагає, щоб високорівневі модулі не залежали від низькорівневих, а обидва типи залежали від абстракцій. Це дозволяє реалізувати інверсію залежностей, що сприяє зниженню зв'язності компонентів і спрощенню їх тестування.

Впровадження принципів SOLID у процес розробки інформаційної управляючої системи забезпечує прозору архітектуру, яка легко адаптується до

змін вимог, сприяє повторному використанню компонентів та полегшує супровід і розвиток проєкту. Це є передумовою створення ефективного й надійного програмного рішення.

У процесі створення якісного програмного забезпечення особливе значення має не лише функціональність системи, а й спосіб, у який реалізовано її програмний код. З цією метою доцільно керуватися принципами чистого коду — сукупністю практик, що забезпечують зручність читання, підтримки та масштабування програмних рішень.

Принципи чистого коду (Clean Code) є основою для створення програмного забезпечення, яке легко читати, зрозуміти, підтримувати і розвивати. Вони не є жорсткими правилами, але слугують рекомендаціями, що ґрунтуються на практичному досвіді професійних розробників. Основні принципи чистого коду охоплюють як стиль написання, так і структурну організацію програм.

У процесі проєктування веб-системи важливим завданням є вибір архітектурного патерна, який забезпечить розділення відповідальностей, спростить підтримку та масштабування коду. Одним із найбільш поширених і перевірених рішень у цьому контексті є патерн Model-View-Controller (MVC). Цей патерн передбачає чітке розділення на три основні компоненти: модель, яка відповідає за бізнес-логіку та доступ до даних; представлення (view), що відображає інформацію користувачу; та контролер, який обробляє вхідні запити, координує модель і представлення.

Серед переваг MVC можна виділити покращену модульність і можливість паралельної роботи над різними компонентами системи. Це дає змогу розробникам концентруватися на своїй частині функціоналу, що особливо ефективно в командній роботі. MVC також забезпечує кращу тестованість, оскільки бізнес-логіка ізольована від користувацького інтерфейсу, і дозволяє повторно використовувати код окремих компонентів у межах різних частин системи. Крім того, такий підхід значно полегшує оновлення інтерфейсу без впливу на логіку застосунку.

Однак, патерн MVC має й свої недоліки. Зокрема, початкова реалізація може здатися складною для невеликих проєктів або для початківців через потребу структурувати код за визначеною архітектурною схемою. Також надлишкова абстракція може призводити до надмірної кількості файлів, що ускладнює навігацію у великій кодовій базі. У випадках зі складною взаємодією між компонентами контролер може стати перевантаженим, що негативно впливає на підтримку.

Незважаючи на зазначені недоліки, було обрано саме патерн MVC як базовий архітектурний шаблон для розробки, оскільки він забезпечує структурованість, масштабованість та зручність у підтримці програмного продукту. Його широке використання у фреймворках, таких як Laravel або Symfony, а також відповідність принципам розділення обов'язків і чистого коду роблять його оптимальним вибором для розробки сучасних веб-застосунків.

У процесі розробки сучасної веб-системи вибір фреймворку відіграє ключову роль, оскільки саме фреймворк визначає архітектуру проєкту, стиль програмування, зручність масштабування та ефективність командної роботи. Тому необхідно здійснити аналіз популярних фреймворків з урахуванням особливостей системи, що проєктується, з метою обґрунтованого вибору найбільш придатного з них.

Laravel — це сучасний PHP-фреймворк з відкритим кодом, створений для спрощення розробки веб-додатків. Його основною особливістю є елегантний синтаксис, орієнтований на розробників, який дозволяє писати чистий, читабельний та підтримуваний код. Laravel реалізує шаблон MVC, забезпечуючи чітке розділення відповідальностей між логікою додатку, його інтерфейсом та контролерами, що координують взаємодію між цими компонентами. Окрім MVC, Laravel також використовує інші патерни, зокрема репозиторій, сервіс-локатор, фасади, інверсію керування (IoC) через контейнер залежностей, а також патерн Observer у системі подій.

Фреймворк Laravel активно використовується у створенні різноманітних типів веб-застосунків — від простих корпоративних сайтів і RESTful API до

складних SaaS-платформ і високонавантажених систем. Його популярність обумовлена широким набором вбудованих засобів, таких як система маршрутизації, ORM Eloquent, підтримка міграцій бази даних, засоби аутентифікації та авторизації, черги, події, кешування, планувальник задач та інтеграція з брокерами повідомлень. Laravel також має розвинену екосистему, включаючи пакети Horizon для моніторингу черг, Nova для адмін-панелей та Jetstream для побудови сучасної аутентифікації.

До переваг Laravel слід віднести його зручну архітектуру, яка підтримує розробку за принципами SOLID, розвинену документацію, активну спільноту, швидкий старт завдяки генераторам коду та шаблонам, а також високу інтеграцію з популярними сервісами й бібліотеками. Завдяки системі міграцій та сидів, розробники можуть легко синхронізувати структуру бази даних між середовищами.

Недоліками Laravel іноді вважають високу абстракцію, яка може ускладнювати розуміння внутрішніх механізмів для новачків, а також підвищені вимоги до ресурсів на складних проєктах, особливо при роботі з Eloquent ORM, що при неправильному використанні може створювати надмірну кількість запитів до бази. Крім того, при зростанні складності проєкту потрібно уважно стежити за розмежуванням бізнес-логіки, щоби уникнути "заплутаних" контролерів або сервісів.

Загалом, Laravel є потужним інструментом для професійної розробки веб-додатків, який поєднує гнучкість, естетику коду та великий функціонал із високою швидкістю розробки.

Symfony — це потужний PHP-фреймворк, орієнтований на розробку складних, масштабованих і гнучких веб-додатків. Він широко використовується у корпоративному секторі, де важливі розширюваність, підтримка високих навантажень та дотримання стандартів. Symfony реалізує низку ключових патернів програмування, зокрема MVC (Model-View-Controller), Dependency Injection, Event Dispatcher та Template Method, що дозволяє створювати чисту, модульну й повторно використововану архітектуру. Однією з основних переваг Symfony є його

модульність: завдяки бандлам та компонентам розробники можуть використовувати лише ті частини фреймворку, які їм потрібні, або навіть інтегрувати окремі компоненти Symfony в інші проєкти, не використовуючи весь фреймворк. До переваг також належать висока гнучкість, активна спільнота, довготривала підтримка LTS-версій та наявність офіційної документації. Однак недоліками Symfony можуть бути складність у вивченні для початківців і вища початкова складність налаштування у порівнянні з іншими PHP-фреймворками, такими як Laravel. Також, через свою архітектурну важкість, він іноді менш продуктивний у розробці невеликих або прототипних рішень. Проте для великих, структурованих веб-проєктів, які потребують суворого контролю якості та довготривалої підтримки, Symfony залишається одним із найкращих рішень.

Інтерфейс `NotificationServiceInterface` є центральним елементом у побудові модульної та масштабованої системи сповіщень. Його проєктування відповідає ключовим принципам SOLID, що забезпечує якісну архітектуру застосунку.

Перш за все, інтерфейс реалізує принцип єдиної відповідальності (SRP) — він відповідає лише за контракт надсилання сповіщення, не включаючи в себе жодних деталей реалізації чи побічних обов'язків, таких як логування або формування повідомлення.

Він також задовольняє принцип відкритості/закритості (OCP): система може бути розширена додаванням нових реалізацій інтерфейсу, наприклад `EmailNotificationService`, `SmsNotificationService`, `TelegramNotificationService`, без необхідності змінювати сам інтерфейс чи існуючий код, що з ним працює.

Принцип підстановки Лісков (LSP) дотримується, оскільки будь-яка реалізація інтерфейсу може бути використана там, де очікується `NotificationServiceInterface`, без порушення логіки клієнтського коду.

Інтерфейс також відповідає принципу сегрегації інтерфейсу (ISP) — він не нав'язує реалізаціям методів, яких ті не потребують. Наприклад, якщо інтерфейс має лише один метод `send(Notification $notification): void`, то він не змушує реалізації підтримувати надлишкову функціональність.

Нарешті, принцип інверсії залежностей (DIP) реалізовано в тому, що клієнтський код залежить не від конкретної реалізації, а від абстракції `NotificationServiceInterface`, що спрощує підключення, заміну або тестування реалізацій через ін'єкцію залежностей.

Таким чином, `NotificationServiceInterface` — це чиста, абстрактна точка розширюваності, яка дозволяє створювати гнучку, легко підтримувану систему з можливістю підключення нових каналів сповіщення без порушення існуючої архітектури.

Клас `EmailNotificationService` є конкретною реалізацією інтерфейсу `NotificationServiceInterface`, відповідальним за надсилання сповіщень через електронну пошту. Він інкапсулює всю логіку, пов'язану з формуванням, адресацією та відправленням email-повідомлень, забезпечуючи тим самим чітке розділення обов'язків у системі сповіщень.

У своїй реалізації клас зазвичай приймає залежності через конструктор — наприклад, сервіс поштового транспорту, шаблонізатор або логер. Метод `send()` (або аналогічний), визначений інтерфейсом, приймає об'єкт повідомлення або набір параметрів, таких як адресат, тема, тіло повідомлення тощо. Далі `EmailNotificationService` формує лист відповідно до заданого формату (часто з використанням шаблонів) та передає його обраній реалізації поштового транспорту.

Таким чином, `EmailNotificationService` ізольовує деталі реалізації email-розсилки, дозволяючи розширювати систему нотифікацій без зміни клієнтського коду, який працює з абстракцією `NotificationServiceInterface`. Це відповідає принципам SOLID, зокрема принципу відкритості/закритості та інверсії залежностей.

Клас `SmsNotificationService` є конкретною реалізацією інтерфейсу `NotificationServiceInterface`, який відповідає за надсилання нотифікацій за допомогою SMS-повідомлень. Основним його завданням є прийняття структурованих вхідних даних — як-от текст повідомлення, отримувач та супутні

параметри — та ініціювання передачі цих даних до стороннього SMS-провайдера (наприклад, Twilio або інший API), з яким відбувається інтеграція.

гідно з принципами SOLID, цей сервіс чітко дотримується принципу єдиної відповідальності (SRP), оскільки займається виключно доставкою SMS і не включає логіку, пов'язану з іншими каналами. Завдяки впровадженню через інтерфейс, `SmsNotificationService` легко підміняється іншими реалізаціями в тестах або при зміні провайдера, що узгоджується з принципом підстановки Лісков (LSP) та залежності від абстракцій (DIP).

Зазвичай у методі `send()` цей клас формує запит до API постачальника SMS, обробляє можливі помилки доставки, а також ініціює запис у систему логів через, наприклад, `DeliveryLogService`. Таким чином, `SmsNotificationService` є важливим елементом загальної архітектури, який забезпечує можливість надійного зв'язку з користувачем через один із традиційних та універсальних каналів комунікації.

`TelegramNotificationService` — це сервіс, який реалізує інтерфейс `NotificationServiceInterface` і відповідає за надсилання повідомлень користувачам через месенджер Telegram. Основною функцією цього класу є взаємодія з Telegram Bot API для передачі текстових нотифікацій конкретним користувачам, які попередньо авторизувались або були пов'язані з ботом системи. Сервіс може використовувати окремий допоміжний сервіс на кшталт `TelegramUserService` для отримання Telegram ID користувача за даними з бази, що забезпечує гнучкість та масштабованість.

Цей сервіс особливо корисний у тих системах, де важлива миттєва доставка інформації без додаткових витрат, характерних для SMS, і коли цільова аудиторія активно користується Telegram. Він дозволяє реалізувати push-механізми на стороні бекенду, спрощуючи взаємодію з користувачем у реальному часі через зручний і популярний канал.

У процесі проектування сучасних веб-систем одним із ключових архітектурних рішень є вибір між монолітною та мікросервісною архітектурою. Кожен з підходів має свої концептуальні особливості, сильні та слабкі сторони, що

визначають спосіб побудови системи, її масштабованість, керованість і супровід у майбутньому.

Монолітна архітектура передбачає реалізацію всієї логіки системи як єдиного цілісного застосунку, де усі модулі та функціональні компоненти тісно пов'язані між собою. Такий підхід дозволяє швидко розпочати розробку, забезпечує просте локальне тестування та розгортання, особливо на ранніх етапах розвитку проєкту. Проте зі зростанням розміру проєкту виникають труднощі з масштабуванням, впровадженням змін та підтримкою, адже будь-яка модифікація вимагає глибокого розуміння всієї системи, що ускладнює командну роботу і підвищує ризики при оновленнях.

Натомість мікросервісна архітектура передбачає розподіл системи на окремі незалежні сервіси, кожен з яких відповідає за чітко визначену функціональність і може розгортатись, масштабуватись та оновлюватись окремо від інших. Це дозволяє будувати гнучкі, розширювані та відмовостійкі системи, які легко масштабуються як горизонтально, так і вертикально. Однак така архітектура потребує більш складної інфраструктури, управління комунікацією між сервісами, забезпечення узгодженості даних та вищої компетенції у команді розробки.

З урахуванням складності предметної області, очікуваного зростання функціональних вимог та потреби в масштабованості й незалежному управлінні окремими частинами системи, було прийнято рішення на користь використання мікросервісної архітектури. Такий підхід забезпечує гнучкість у розробці, дозволяє різним командам працювати автономно над окремими сервісами, спрощує впровадження змін, полегшує тестування й оновлення системи без зупинки її роботи в цілому. Крім того, мікросервісна архітектура створює передумови для підвищення надійності та відмовостійкості, що особливо важливо в контексті реалізації високонавантажених веб-систем.

У сучасних веб-додатках питання безпеки та зручності автентифікації користувачів відіграє ключову роль. Залежно від архітектурних рішень і особливостей взаємодії між клієнтом та сервером застосовуються різні підходи до реалізації механізмів аутентифікації. У даному проєкті для забезпечення

збереження стану автентифікованого користувача та безперервності його взаємодії із системою було обрано традиційний і перевірений часом підхід, заснований на використанні сесій і cookie-файлів. Такий спосіб добре зарекомендував себе для серверно-орієнтованих застосунків і забезпечує достатній рівень контролю над ідентифікацією користувачів із боку бекенду.

Аутентифікація на основі токенів передбачає видачу клієнту унікального маркера (токена) після успішного входу в систему. Найпоширенішим підходом є використання JSON Web Token (JWT), що містить зашифровану або підписану інформацію про користувача і термін дії. Токен надсилається клієнту (наприклад, у заголовок HTTP Authorization) і використовується для підтвердження особи під час кожного запиту без необхідності зберігати стан на сервері.

Головна перевага токенної аутентифікації полягає у безстейтовості: сервер не повинен зберігати сесію, що спрощує масштабування, особливо у мікросервісній архітектурі. Токени можуть бути легко передані між сервісами або використовуватись для авторизації у сторонніх системах (наприклад, у SSO-сценаріях). Також цей механізм легко інтегрується у SPA або мобільні застосунки.

Серед недоліків – складність реалізації безпечної інвалідації токенів, оскільки вони часто живуть самостійно до завершення терміну дії. Крім того, якщо токен потрапляє до зломисника, без механізмів відкликання він може використовуватись до моменту закінчення дії. Також слід бути обережним із розміром токенів, оскільки вони можуть збільшити навантаження на мережу.

З огляду на вимоги до безпеки, контроль доступу та простоту реалізації в умовах обраної архітектури системи, було прийнято рішення використовувати механізм аутентифікації на основі сесій. Такий підхід забезпечує централізоване керування станом користувача, спрощує інвалідацію сеансів у разі виявлення підозрілої активності, а також добре інтегрується з cookie-механізмом, що дозволяє уникати передачі облікових даних у кожному запиті. У контексті монолітного або слабо розподіленого середовища використання сесій є виправданим вибором, оскільки знижує ризики, пов'язані з компрометацією токенів, та зменшує загальну складність інфраструктури.

У контексті розробки системи нотифікацій, яка функціонує в умовах високого навантаження та обробляє повідомлення в режимі реального часу, питання кешування стає другорядним або навіть недоцільним. Основна причина полягає в тому, що такі системи орієнтовані не на повторне використання вже згенерованих даних, а на миттєву обробку нових подій і негайну доставку повідомлень до кінцевих користувачів. Кешування у цьому випадку лише ускладнює архітектуру, не даючи суттєвого виграшу в продуктивності, оскільки повідомлення швидко втрачають актуальність і не мають сенсу для повторного використання. Крім того, запровадження кешу може створити затримки в оновленні інформації, що суперечить вимогам до реального часу. Відтак, при розробці інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій було свідомо відмовлено від використання кешування, зосередивши ресурси на оптимізації черг повідомлень, балансуванні навантаження та масштабуванні системи для забезпечення безперервної роботи з актуальними даними.

У процесі розробки клієнтської частини системи, побудованої з використанням фреймворку Vue.js, активно застосовується технологія AJAX (Asynchronous JavaScript and XML) для організації асинхронної взаємодії з сервером. AJAX дозволяє здійснювати обмін даними з сервером у фоновому режимі без необхідності повного перезавантаження вебсторінки, що забезпечує високу інтерактивність, зменшення затримок у взаємодії користувача з інтерфейсом та покращення загального користувацького досвіду. Такий підхід сприяє більшій чутливості інтерфейсу, дозволяє реалізувати механізми реального часу (наприклад, відображення нотифікацій або статусів), а також знижує навантаження на сервер за рахунок часткової передачі лише необхідних даних. Завдяки цим перевагам технологія AJAX є ключовим інструментом для створення сучасних динамічних веб-застосунків у рамках обраної архітектури.

Інструкція користувача є важливою складовою документації проєкту, що забезпечує належне розгортання, підтримку та ефективне використання розробленої інформаційної системи. Далі наведено поетапні вказівки для різних

категорій користувачів, включаючи програмістів, системних адміністраторів та кінцевих користувачів. Кожна частина інструкції орієнтована на конкретні потреби та передбачає детальний опис дій, необхідних для встановлення, налаштування, супроводу або роботи із системою. Дотримання наведених рекомендацій дозволяє забезпечити безперебійну роботу системи, а також уникнути типових помилок під час її експлуатації.

Для програмістів запуск проєкту передбачає стандартний процес налаштування локального середовища розробки. Перш за все, потрібно клонувати репозиторій на свій комп'ютер, перейти в робочу директорію, скопіювати шаблон змінних середовища і створити на його основі власний `.env` файл із відповідними налаштуваннями. Після цього достатньо зібрати й запустити Docker-контейнери, які автоматично піднімуть необхідні сервіси, включно з веб-сервером, базою даних та додатком. У разі використання фреймворків, таких як Laravel або Symfony, додатково виконується команда для запуску міграцій, що створюють структуру бази даних. Для налагодження також можна активувати Xdebug, а логіка додатку розміщується згідно з принципами архітектури MVC, що спрощує навігацію та модифікацію коду.

Системному адміністратору для розгортання проєкту на продакшн-сервері необхідно мати встановлене середовище Docker та Docker Compose, відкриті відповідні порти для HTTP або HTTPS, наявний зареєстрований домен та SSL-сертифікат для захищеного з'єднання. Після копіювання коду на сервер адміністратор повинен налаштувати середовище через змінні у відповідному конфігураційному файлі, запустити контейнери, перевірити журнали роботи сервісів та, за потреби, налаштувати резервне копіювання бази даних. Після оновлення коду виконується оновлення образів контейнерів із повторним запуском. Система також може супроводжуватись моніторингом продуктивності за допомогою сторонніх рішень.

3.4 Технічне забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Для забезпечення функціонування ІУС з автоматизації процесу зовнішніх нотифікацій необхідна наявність сервера. Його можна розгорнути самостійно, орендувати або скористатися хмарними сервісами. На цьому сервері буде розміщено програмне забезпечення системи, яке оброблятиме запити нотифікацій. Доступ до системи здійснюється через веб-інтерфейс, що відкривається у браузері користувача. Це дозволяє працювати із системою з будь-якого пристрою незалежно від операційної системи чи типу платформи — зокрема, з комп'ютерів та мобільних пристроїв, що працюють на Windows, MacOS, Linux чи інших ОС. Схема структури технічного та програмного забезпечення наведена на рис. 3.20.

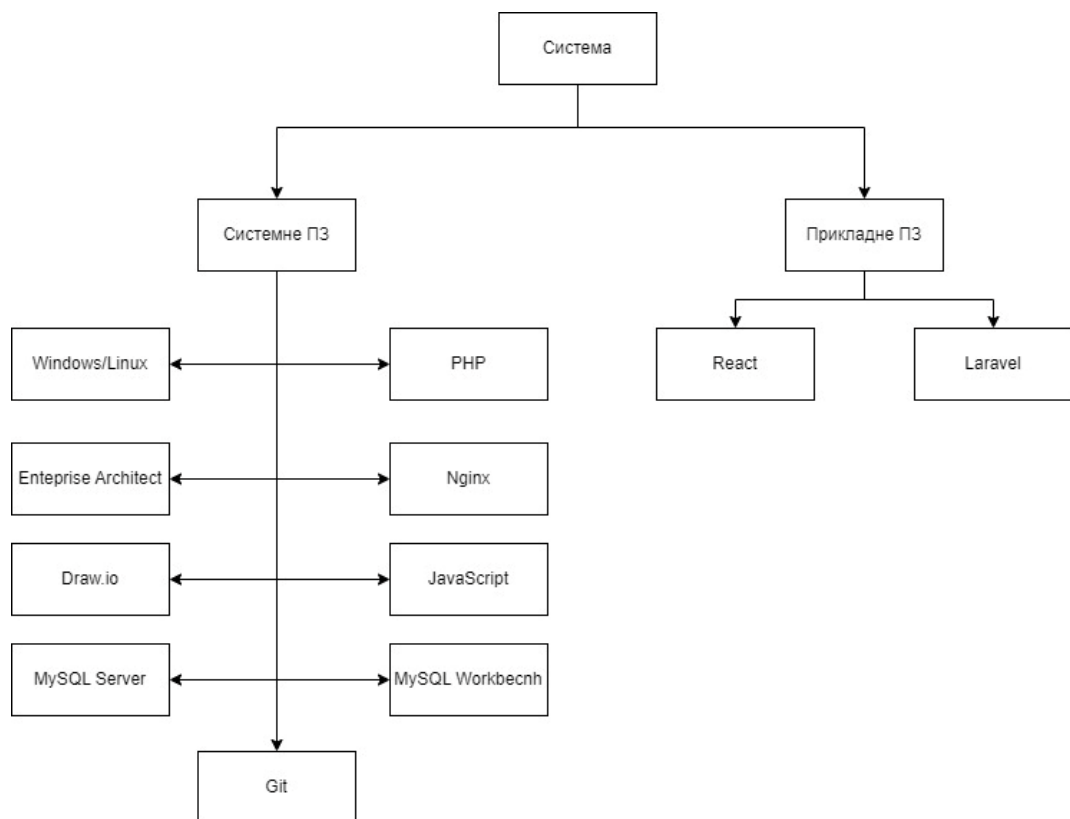


Рисунок 3.20 - Схема структури технічного та програмного забезпечення

Джерело: розроблено автором

Коли користувач заходить на сайт або в систему, його пристрій надсилає запит на сервер. Сервер обробляє цей запит, шукає потрібну інформацію і повертає її назад на пристрій. Усе це відбувається дуже швидко, тож користувач бачить результат майже миттєво. Перевага такої архітектури в тому, що підключитися до системи можна з будь-якого пристрою, який має доступ до Інтернету — незалежно від того, це Android, iPhone, планшет, ноутбук чи звичайний ПК. Головне — мати браузер або застосунок, який може надіслати запит до сервера. Схема клієнт-серверної архітектури технічного забезпечення наведена на рис. 3.21.

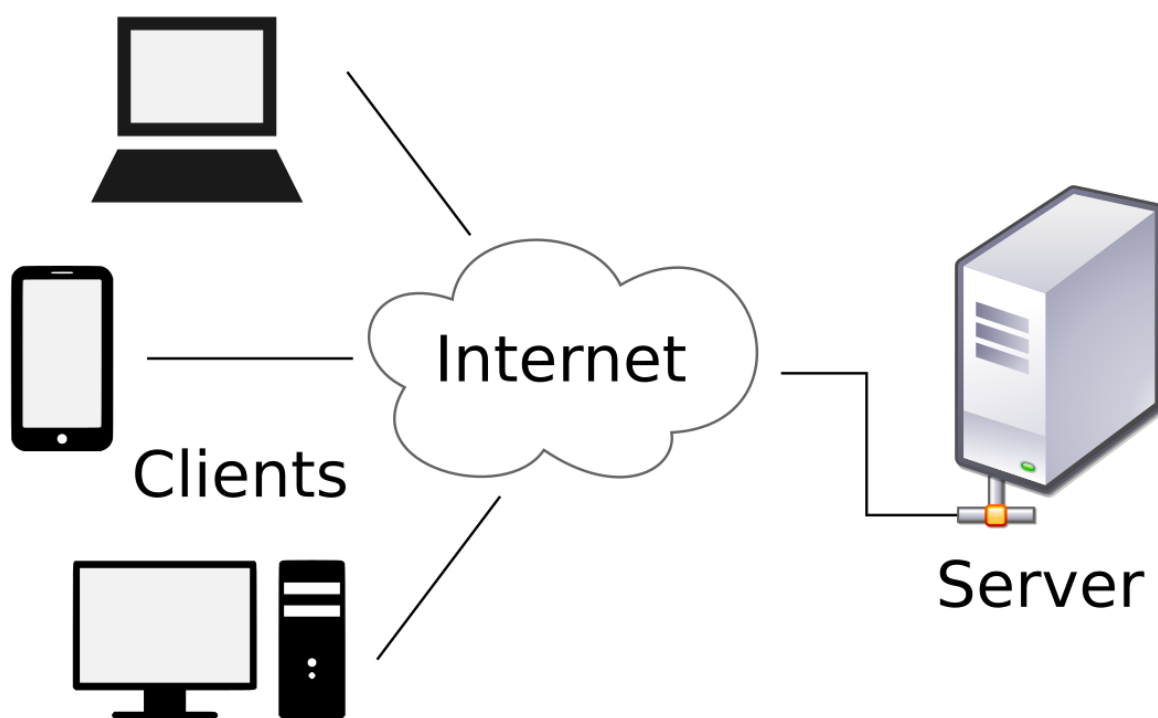


Рисунок 3.21 - Схема клієнт-серверної архітектури технічного забезпечення

Джерело: <https://upload.wikimedia.org/wikipedia/commons/thumb/c/c9/Client-server-model.svg/1200px-Client-server-model.svg.png>

Розробка системи здійснювалась на ноутбуку Lenovo IdeaPad, що є потужним і надійним пристроєм для розробки програмного забезпечення. На ньому була встановлена операційна система Linux Ubuntu, яка є однією з найбільш популярних серед розробників завдяки своїй стабільності, безпеці та зручності

використання. Ubuntu надає великий вибір інструментів для розробки, включаючи підтримку різних мов програмування, фреймворків і бібліотек, а також можливість налаштування середовища для ефективної роботи з такими технологіями, як PHP, JavaScript, а також з Docker, Git і базами даних. Така операційна система є ідеальним вибором для розробки веб-систем, оскільки підтримує всі необхідні інструменти для інтеграції, тестування та налаштування серверного середовища. Зображення технічного забезпечення Lenovo IdeaPad наведено на рис. 3.22.

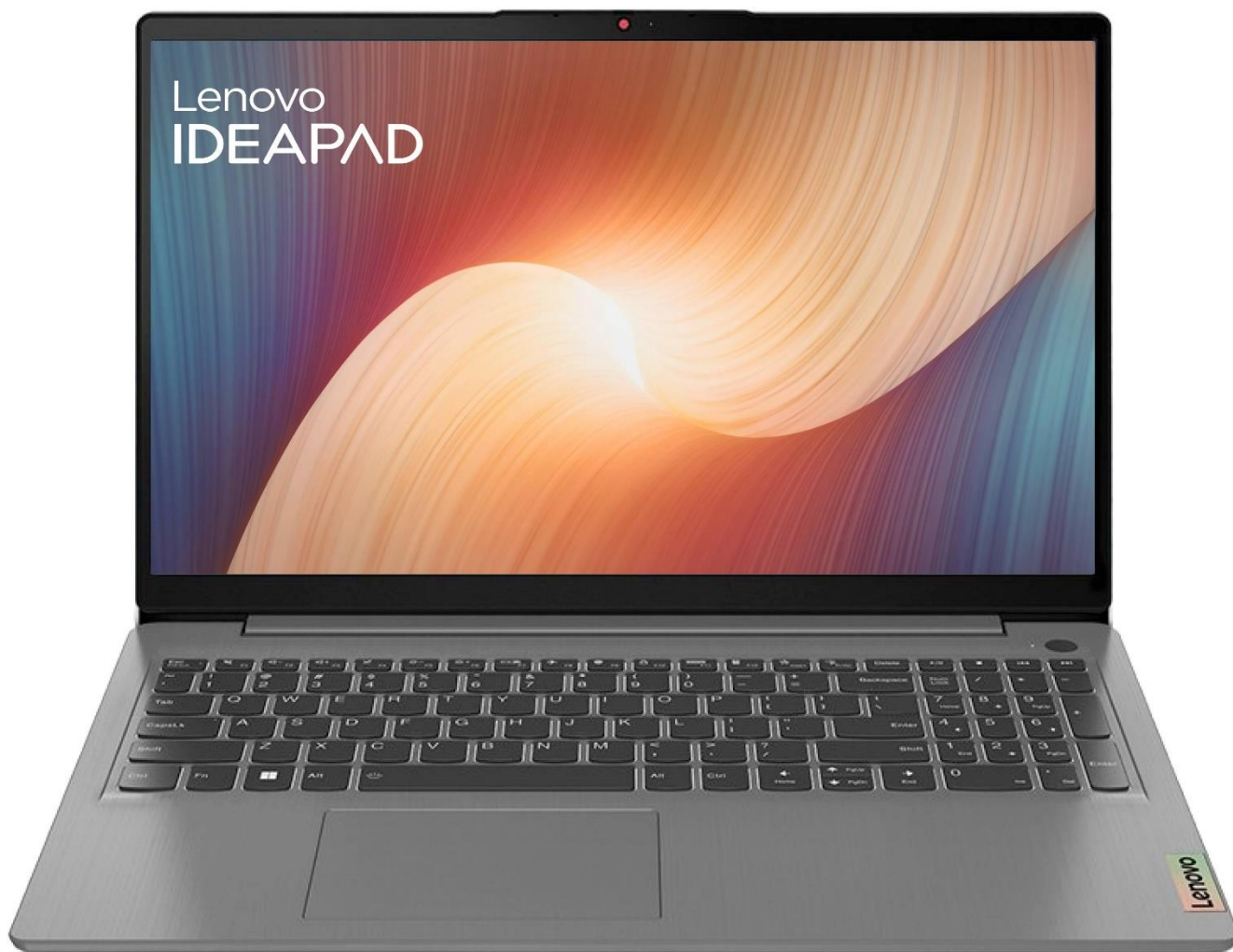


Рисунок 3.22 - Технічне забезпечення Lenovo IdeaPad

Джерело: https://cdn.comfy.ua/media/catalog/product/l/e/lenovo_ideapad_3_15iau7_82rk012lra_arctic_grey_1__1_1.jpg

3.4.2 Ресурсні вимоги до технічного забезпечення

Для запуску та ефективної роботи розробленої інформаційної управляючої системи необхідно забезпечити відповідність технічним вимогам, що гарантують стабільність і продуктивність при обробці великого обсягу трафіку в реальному часі. Мінімальні вимоги до апаратного забезпечення для розробки та експлуатації цієї системи включають:

- процесор (CPU): Для обробки складних обчислювальних задач та багатозадачності рекомендується мінімум чотириядерний процесор, наприклад, Intel Core i5 або еквівалент від AMD (наприклад, Ryzen 5). Це дозволить забезпечити швидку обробку запитів та підтримку декількох одночасних процесів;
- оперативна пам'ять (RAM): Мінімум 8 ГБ оперативної пам'яті для нормальної роботи з великою кількістю відкритих вкладок, інструментів для розробки та баз даних. Для більших навантажень або роботи з великими обсягами даних краще використовувати 16 ГБ RAM;
- жорсткий диск: Для зберігання проекту та бази даних бажано мати SSD диск ємністю від 256 ГБ для швидкого доступу до даних і забезпечення високої продуктивності. SSD значно прискорює роботу в порівнянні з традиційними HDD;
- графічна карта (GPU): Для цієї інформаційної системи не потрібно потужне графічне обладнання. Базової інтегрованої графіки, що є в процесорах Intel або AMD, буде достатньо для підтримки роботи системи;
- операційна система: Linux Ubuntu (або інша подібна система, яка підтримує необхідні інструменти для розробки та деплою). Ubuntu забезпечує всі необхідні інструменти для серверного та локального середовища розробки;

- мережеве з'єднання: Оскільки система працює в реальному часі з великим потоком трафіку, важливо мати стабільне та швидке інтернет-з'єднання. Рекомендується мінімум 100 Мбит/с для забезпечення швидкого обміну даними між сервером і користувачами.

Загалом, вищезазначене технічне забезпечення дозволяє запускати та ефективно працювати з інформаційною системою, зважаючи на її вимоги щодо швидкості та обсягу оброблюваних даних.

3.5 Реалізація інформаційної системи

Результати розробленої системи задовільняють поставлену задачу. Дані результати роботи інформаційної управляючої системи полягають у її здатності ефективно обробляти великі обсяги даних та здійснювати їх аналіз у реальному часі, забезпечуючи своєчасне реагування на події та нотифікації для кінцевих користувачів. Система реалізує управлінську функцію, автоматизуючи процеси моніторингу, обробки даних і прийняття рішень на основі отриманої інформації.

Демонстрація функціоналу надсилання нотифікації, ініційованої після реєстрації користувача, наведено на рис. 3.23 - 3.24.

Реєстрація

Ім'я
Mykolai

Прізвище
Stepanenko

Електронна пошта
Nikolaua36@gmail.com

Номер телефону
+380980191919

Пароль
.....

Повторення паролю
.....

[Зареєструватися](#)

Вже маєте акаунт? [Увійти](#)

Рисунок 3.23 – Демонстрація надсилання нотифікації після реєстрації користувача
Джерело: розроблено автором

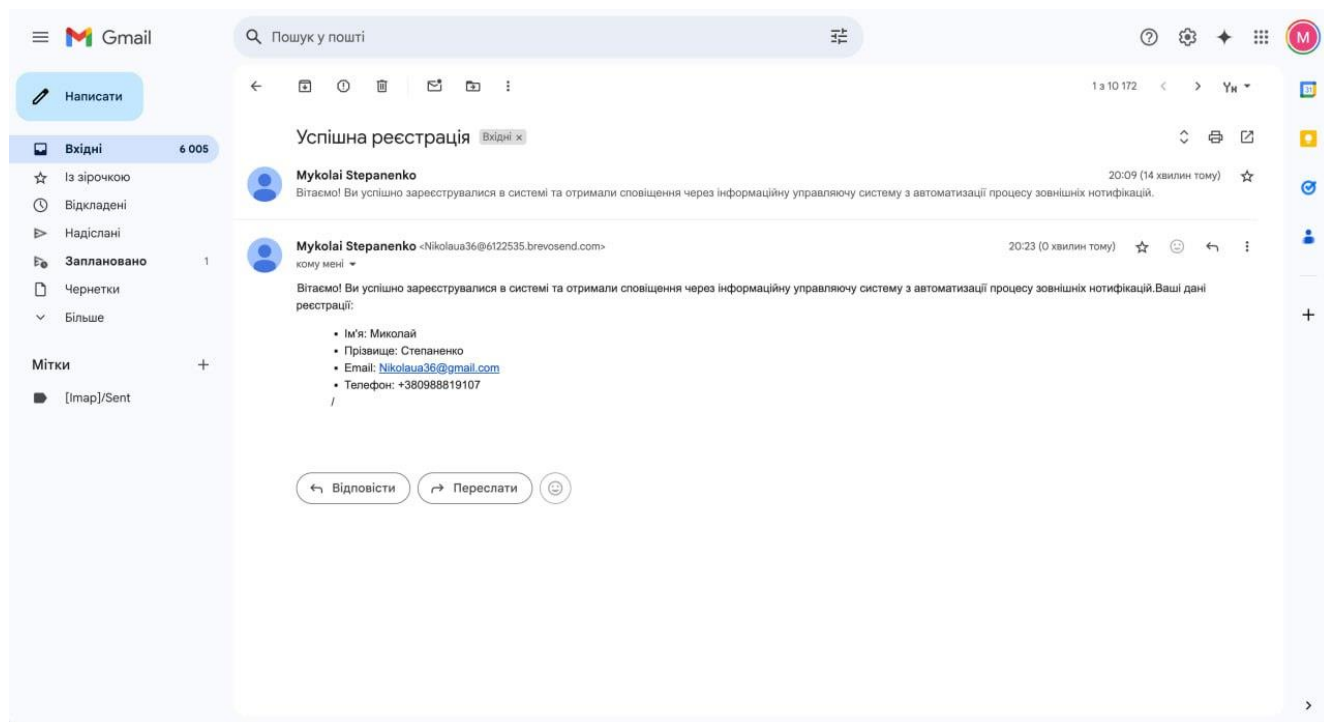


Рисунок 3.24 - Демонстрація надсилання нотифікації після реєстрації користувача
Джерело: розроблено автором

Також на рис. 3.25 – 3.26 наведено демонстрацію обрання іншого каналу, такого як Telegram, та надсилання нотифікації через даний канал сповіщення.

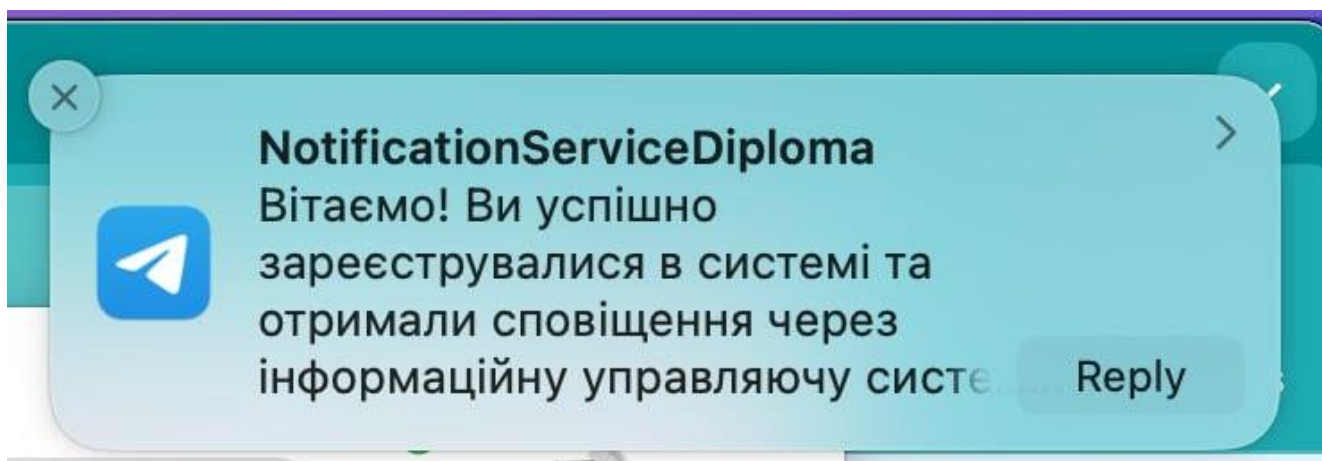


Рисунок 3.25 – Демонстрація надсилання нотифікації після реєстрації користувача в канал нотифікації соц. мережі

Джерело: розроблено автором

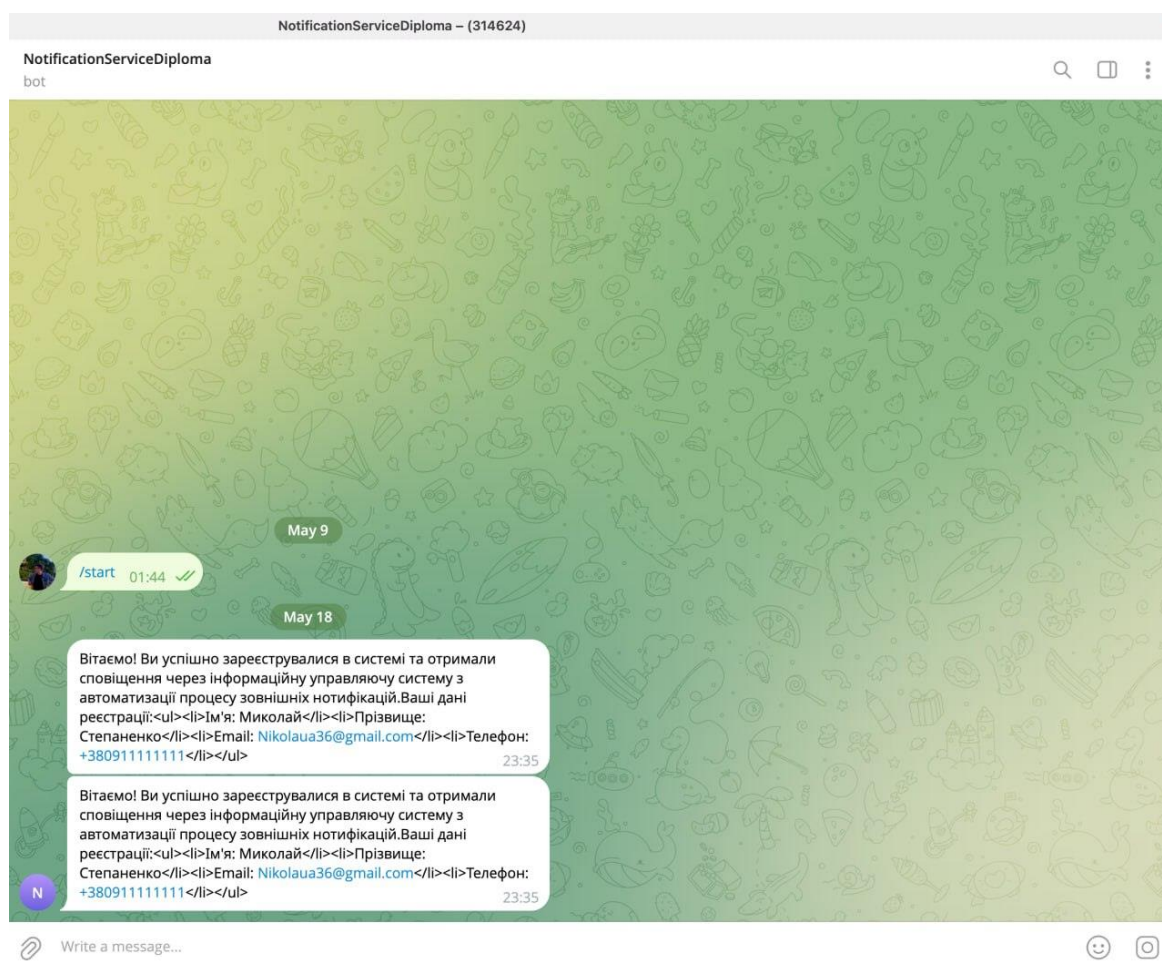


Рисунок 3.26 – Демонстрація надсилання нотифікації після реєстрації користувача в канал нотифікації соц. мережі

Джерело: розроблено автором

Робота інформаційної управляючої системи задовільняє всім поставленим вимогам та виконує всі поставлені задачі.

Завдяки інтеграції різних технологій, таких як AJAX для динамічної взаємодії з користувачем, а також використання сучасних методів аутентифікації, система забезпечує зручність використання та безпеку для користувачів. Реалізація масштабованих механізмів для обробки потоку нотифікацій у реальному часі дозволяє ефективно працювати з великими обсягами трафіку, що критично важливо в умовах високих навантажень. Система може бути застосована в таких сферах як:

- інтернет-комерція — для відслідковування замовлень, автоматичної генерації нотифікацій для користувачів та адміністраторів, а також для управління взаємодією з клієнтами;
- фінансові установи — для моніторингу операцій, відправки повідомлень про стан рахунків, а також для контролю за виконанням фінансових транзакцій у реальному часі;
- медичні заклади — для обробки запитів пацієнтів, відправки термінових повідомлень лікарям та адміністраторам, а також для автоматизації процесів внутрішнього управління;
- логістика та транспорт — для відслідковування відправлень, попередження про затримки або зміни в маршрутах, а також для оптимізації процесів планування та доставки;
- соціальні мережі та медіа — для забезпечення своєчасних сповіщень та повідомлень користувачам про новини, зміни в статусах чи подіях.

Для подальшого вдосконалення інформаційної управляючої системи можна розглянути кілька напрямків покращення, які дозволять підвищити її ефективність, масштабованість і зручність для користувачів.

Масштабування та оптимізація продуктивності: одним із найбільш важливих напрямків є оптимізація обробки великих обсягів даних, що дозволить забезпечити високу швидкість роботи при збільшенні кількості користувачів і

запитів. Для цього можна впровадити горизонтальне масштабування серверів, використовуючи мікросервісну архітектуру для розподілення навантаження на різні компоненти системи. Також можна вдосконалити систему кешування для зменшення затримок у відповідях і підвищення ефективності.

Інтеграція з іншими системами та сервісами: для розширення функціональних можливостей системи варто впровадити інтеграцію з іншими зовнішніми системами через API, що дозволить обмінюватися даними з іншими платформами (наприклад, платіжними системами, ERP-системами або інструментами аналітики). Це дозволить створити більш комплексну екосистему для користувачів.

Покращення UI/UX дизайну: оскільки взаємодія з системою здійснюється через веб-інтерфейс, важливо постійно працювати над покращенням дизайну для забезпечення більш комфортного та інтуїтивно зрозумілого досвіду користувачів. Можна також реалізувати адаптивний дизайн для підтримки різних пристроїв, таких як мобільні телефони та планшети.

Покращення безпеки: оскільки інформаційна управляюча система працює з персональними та конфіденційними даними, важливо забезпечити високий рівень захисту від зовнішніх і внутрішніх загроз. Для цього можна впровадити додаткові заходи безпеки, такі як багатофакторна аутентифікація, шифрування даних на всіх етапах їх обробки та додаткові засоби захисту від атак (наприклад, фільтрація запитів, захист від DoS-атак).

Загалом, покращення системи можна здійснити в кількох напрямках, зокрема в оптимізації її продуктивності, безпеки та інтеграції з іншими платформами. Це дозволить досягти ще більш високої ефективності в управлінні та обробці даних, а також підвищити задоволеність кінцевих користувачів.

Наукова новизна в розробленій інформаційній управляючій системі полягає в інтеграції сучасних підходів до обробки та управління великими обсягами даних, а також у впровадженні новітніх технологій для забезпечення високої ефективності та безпеки роботи системи. Однією з основних інновацій є використання мікросервісної архітектури в поєднанні з брокерами повідомлень для обробки

великих потоків даних у реальному часі, що дозволяє системі працювати з високою масштабованістю і без затримок навіть при великому навантаженні.

Ще одним аспектом новизни є застосування принципів Clean Code та SOLID, що дозволяє не тільки полегшити подальшу підтримку і розвиток системи, а й підвищити її стабільність та ефективність. Використання таких підходів значно зменшує ймовірність виникнення помилок під час експлуатації, що є критичним для систем з високими вимогами до надійності.

Іншою важливою інновацією є застосування сучасних фреймворків, зокрема Symfony, у поєднанні з Vue2 для побудови інтерактивного та швидкого веб-інтерфейсу, що дозволяє забезпечити високий рівень користувацького досвіду при роботі з системою. Впровадження цього технологічного стека дозволяє значно спростити процес розробки і підтримки, знижуючи витрати часу на реалізацію нових функцій та покращень.

Таким чином, наукова новизна даної системи полягає у використанні інноваційних технологій, сучасних методів розробки та підходів до управління даними, що дозволяє створити високоефективну, масштабовану і безпечну систему для роботи з великими потоками інформації в реальному часі.

ВИСНОВКИ

У рамках кваліфікаційної магістерської роботи з розробки інформаційної управляючої системи з автоматизації процесу зовнішніх нотифікацій було всебічно опрацьовано ключові аспекти функціонування такої системи та запропоновано ряд практичних рішень, що сприяють підвищенню її ефективності та надійності. У процесі розробки було обґрунтовано архітектуру системи, визначено оптимальні канали сповіщень, реалізовано механізми налаштування подій та сценаріїв нотифікацій, а також враховано вимоги до безпеки, масштабованості та зручності адміністрування. Впроваджені рішення забезпечують стабільне функціонування системи в умовах високого навантаження, дозволяють оперативно інформувати користувачів про події та підтримують адаптацію до змін бізнес-процесів. У підсумку, результати проекту значно підвищують якість інформаційної взаємодії в межах організації та сприяють її ефективнішому управлінню.

Застосування розроблених у дипломному проекті рішень у сфері систем нотифікацій дозволяє організації суттєво підвищити ефективність інформаційної взаємодії, оперативність реагування на події та якість комунікації з користувачами. Отримані результати демонструють перспективність впровадження сучасних підходів до проектування інформаційних управляючих систем у цій галузі, що забезпечують гнучкість, масштабованість та адаптивність рішення до конкретних потреб бізнесу. Таким чином, проект підтверджує актуальність інтеграції систем нотифікацій як важливої складової цифрової трансформації підприємства.

Реалізація проекту інформаційної управляючої системи нотифікацій є важливим етапом у вдосконаленні внутрішніх процесів організації та підвищенні ефективності управління інформаційними потоками. Система нотифікацій була спроектована з урахуванням потреб і вимог користувачів, що дозволило забезпечити необхідний функціонал для автоматизованої обробки повідомлень, покращення зворотного зв'язку та своєчасного інформування. Це, у свою чергу,

сприяє загальному підвищенню якості обслуговування та ефективності прийняття рішень.

Впровадження розробленої системи нотифікацій дозволить організації оптимізувати комунікаційні процеси, зменшити часові та операційні витрати, а також підвищити загальну ефективність управління інформацією. Система має надійну архітектуру, підтримує масштабування та адаптацію до змін, що забезпечує її готовність до подальшого розвитку відповідно до зростаючих потреб.

У підсумку, створення і впровадження системи нотифікацій є важливим елементом цифрової трансформації, який сприятиме підвищенню організаційної гнучкості, покращенню сервісу та досягненню стратегічних цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шингера Н. Я. ІНФОРМАЦІЙНО-УПРАВЛЯЮЧІ СИСТЕМИ. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024.
2. Що таке нотифікація. <https://slovotvir.org.ua/words/notyfikatsiia>. *Що таке нотифікація*. 12.03.2025.
3. Шаблон проектування "Спостерігач". <https://refactoring.guru/uk/design-patterns/observer>. *Шаблон проектування "Спостерігач"*. 11.03.2025. URL: <https://refactoring.guru/uk/design-patterns/observer> (дата звернення: 17.04.2025).
4. Шаблон проектування "Посередник". <https://refactoring.guru/uk/design-patterns/mediator>. *Шаблон проектування "Посередник"*. 16.01.2025. URL: <https://refactoring.guru/uk/design-patterns/mediator>.
5. System Design Patterns: Publisher Subscriber pattern. <https://dsysd-dev.medium.com/system-design-patterns-publisher-subscriber-pattern-ae281d5598f5>. URL: <https://dsysd-dev.medium.com/system-design-patterns-publisher-subscriber-pattern-ae281d5598f5>.
6. Особливості event-driven architecture на прикладах з практики. *DOU*. 20.05.2022. URL: <https://dou.ua/forums/topic/38182/>.
7. Firebase Cloud Messaging. <https://firebase.google.com/docs/cloud-messaging>. URL: <https://firebase.google.com/docs/cloud-messaging>.
8. Benefits of Amazon Simple Notification Service. *aws.amazon.com*. URL: <https://aws.amazon.com/ru/sns/>.
9. The Twilio Customer Engagement platform combines powerful communications APIs with AI and first-party data.. *twilio.com*. URL: <https://www.twilio.com/en-us>.
10. Firebase Cloud Messaging. <https://firebase.google.com/docs/cloud-messaging>. URL: <https://firebase.google.com/docs/cloud-messaging>.
11. Turn emails into revenue with a fully integrated email platform. *sendgrid.com*. URL: <https://sendgrid.com/en-us>.

12. Best Email Marketing Service. *brevo.com*. URL:
https://www.brevo.com/landing/best-email-marketing-service/?utm_source=adwords&utm_medium=cpc&utm_content=Competitors&utm_extension=&utm_term=sendgrid&utm_matchtype=e&utm_campaign=20030961979&utm_network=g&km_adid=753087712485&km_adposition=&km_device=c&utm_adgroupid=156532867856&gad_source=1&gad_campaignid=20030961979&gbr aid=0AAAAADjx0RbnRHMRKWeH46XDz8weD7iww&gclid=CjwKCAjwraVBBh BjEiwAIr30VHSmCX8nVbCuADxwin3ZX2eUHcvklEkkc4VlbiDFqUb6PnMxlujV 4hoC_4cQAvD_BwE.
13. Як реалізувати системи з високим навантаженням на PHP: особистий досвід. *highload.tech*. URL: <https://highload.tech/uk/blogs/yak-realizuvati-sistemi-z-visokim-navantazhennyam-na-php-osobistij-dosvid/>.
14. Що таке архітектура мікросервісів?. *payproglobal.com*. URL:
<https://payproglobal.com/uk/%D0%B2%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D1%96/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D0%BC%D1%96%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D1%96%D1%81%D0%BD%D0%B0-%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0/>.
15. What is a message queue?. *ibm.com*. URL:
<https://www.ibm.com/think/topics/message-queues>.
16. Part 1: RabbitMQ for beginners - What is RabbitMQ?. *www.cloudamqp.com*. URL:
<https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>.
17. Apache Kafka More than 80% of all Fortune 100 companies trust, and use Kafka.. *kafka.apache.org*. URL: <https://kafka.apache.org/>.
18. Redis Pub/Sub How to use pub/sub channels in Redis. *redis.io*. URL:
<https://redis.io/docs/latest/develop/interact/pubsub/>.

19. Що таке CRM?. *www.creatio.com*. URL: <https://www.creatio.com/ua/crm/what-is-crm>.
20. Що таке ERP?. *www.oracle.com*. URL: <https://www.oracle.com/ua/erp/what-is-erp/>.
21. Функціональні компоненти інформаційних систем. *studfile.net*. URL: <https://studfile.net/preview/5064248/page:4/>.
22. Модульність. *studfile.net*. URL: <https://studfile.net/preview/9445425/page:52/>.
23. Observer. *refactoring.guru*. URL: <https://refactoring.guru/design-patterns/observer>.
24. Updated data: IT is the only growing export industry in Ukraine. https://itukraine.org.ua/en/updated-data-it-is-the-only-growing-export-industry-in-ukraine/?utm_source=chatgpt.com. URL: https://itukraine.org.ua/en/updated-data-it-is-the-only-growing-export-industry-in-ukraine/?utm_source=chatgpt.com.
25. Ukraine's booming IT sector can drive positive change. *www.atlanticcouncil.org*. URL: https://www.atlanticcouncil.org/blogs/ukrainealert/ukraines-booming-it-sector-can-drive-positive-change/?utm_source=chatgpt.com.
26. In 2023, Ukrainian IT Services Export Faced Its First Decline in Years. *itukraine.org.ua*. URL: https://itukraine.org.ua/en/in-2023-ukrainian-it-services-export-faced-its-first-decline-in-years/?utm_source=chatgpt.com.
27. Що таке SaaS (програмне забезпечення як послуга)?. <https://www.oracle.com/ua/applications/what-is-saas/>. URL: <https://www.oracle.com/ua/applications/what-is-saas/>.
28. Що таке php?. <https://freehost.com.ua/ukr/faq/wiki/что-такое-php/>. URL: <https://freehost.com.ua/ukr/faq/wiki/что-такое-php/>.
29. Symfony. <https://symfony.com/>. URL: <https://symfony.com/>.
30. Vue.js як JavaScript-фреймворк. <https://foxminded.ua/vue-js/>. URL: <https://foxminded.ua/vue-js/>.
31. Що таке AJAX та його застосування в реальних проектах. <https://foxminded.ua/ajax/>. URL: <https://foxminded.ua/ajax/>.
32. Mysql. <https://www.mysql.com/>. URL: <https://www.mysql.com/>.
33. Операції ACID. *www.solix.com*. URL: <https://www.solix.com/uk/kb/acid-transactions/>.

34. Принципи SOLID. Навіщо вони потрібні програмістам та як з ними працювати.
<https://campus.epam.ua/ua/blog/602>. URL: <https://campus.epam.ua/ua/blog/602>.
35. Побудова динамічних вебзастосунків за допомогою MVC.
<https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc>. URL:
<https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc>.
36. IDE: что это и какие возможности дает в разработке.
<https://orbitsoft.com/ru/blog/ide/>. URL: <https://orbitsoft.com/ru/blog/ide/>.
37. Про архітектуру додатків. <https://foxminded.ua/arkhitektura-zastosunku/>. URL:
<https://foxminded.ua/arkhitektura-zastosunku/>.
38. Що таке API?. <https://qalight.ua/baza-znaniy/shho-take-api/>. URL:
<https://qalight.ua/baza-znaniy/shho-take-api/>.
39. Що таке push-сповіщення та як правильно їх використовувати?.
<https://wezom.com.ua/ua/blog/push-notification>. URL:
<https://wezom.com.ua/ua/blog/push-notification>.
40. Уніфікована мова моделювання (Unified Modeling Language - UML).
<https://www.maxzosim.com/unifikovana-mova-modeluvannia/>. URL:
<https://www.maxzosim.com/unifikovana-mova-modeluvannia/>.
41. Дізнайтеся що таке data mining (інтелектуальний аналіз даних), як цей процес відбувається, та чому він є таким важливим для сучасного суспільства та бізнесу зокрема.. *futurenow.com.ua*. URL: <https://futurenow.com.ua/shho-take-data-mining-analiz-danyh/>.
42. Методологія CRISP-DM – класика, яка не застаріває.
<https://www.education.ua/blog/45378/>. URL: <https://www.education.ua/blog/45378/>.
43. Налаштування середовища виконання в Google Colab.
<https://foxminded.ua/google-colab/>. URL: <https://foxminded.ua/google-colab/>.
44. Що таке база знань та навіщо вона потрібна?.
<https://helpcrunch.com/blog/uk/baza-znan/>. URL:
<https://helpcrunch.com/blog/uk/baza-znan/>.
45. Enhance your LLM with a Knowledge Graph!. <https://www.ontotext.com/>. URL:
<https://www.ontotext.com/>.

46. What Is a Graph Database?. *aws.amazon.com*. URL: <https://aws.amazon.com/nosql/graph/>.
47. SPARQL/OWL. *owlready2.readthedocs.io*. URL: <https://owlready2.readthedocs.io/en/latest/sparql.html>.
48. MDA® - The Architecture Of Choice For A Changing World. <https://www.omg.org/mda/>. URL: <https://www.omg.org/mda/>.
49. Onion Architecture. *medium.com*. URL: <https://medium.com/expedia-group-tech/onion-architecture-deed8a554423>.
50. The Clean Architecture. *blog.cleancoder.com*. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>.
51. Hexagonal Architecture, there are always two sides to every story. *medium.com*. URL: <https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c>.

ДОДАТКИ

Додаток А

```
<?php
```

```
namespace App\Entity;
```

```
use App\Repository\DeliveryLogRepository;
```

```
use Doctrine\ORM\Mapping as ORM;
```

```
#[ORM\Entity(repositoryClass: DeliveryLogRepository::class)]
```

```
class DeliveryLog
```

```
{
```

```
    #[ORM\Id]
```

```
    #[ORM\GeneratedValue]
```

```
    #[ORM\Column]
```

```
    private ?int $id = null;
```

```
    #[ORM\Column]
```

```
    private int $userId;
```

```
    #[ORM\Column]
```

```
    private string $channel;
```

```
    #[ORM\Column]
```

```
    private string $status;
```

```
    #[ORM\Column(nullable: true)]
```

```
    private int $sentAtTs;
```

```
#[ORM\Column(nullable: true)]
```

```
private string $deliveredAtTs;
```

```
public function getId(): ?int
```

```
{
```

```
    return $this->id;
```

```
}
```

```
public function getUserId(): int
```

```
{
```

```
    return $this->userId;
```

```
}
```

```
public function setUserId(int $userId): self
```

```
{
```

```
    $this->userId = $userId;
```

```
    return $this;
```

```
}
```

```
public function getChannel(): string
```

```
{
```

```
    return $this->channel;
```

```
}
```

```
public function setChannel(string $channel): self
```

```
{
```

```
    $this->channel = $channel;
```

```
    return $this;
```

```
}
```

```
public function getStatus(): string  
{  
    return $this->status;  
}
```

```
public function setStatus(string $status): self  
{  
    $this->status = $status;  
    return $this;  
}
```

```
public function getSentAtTs(): int  
{  
    return $this->sentAtTs;  
}
```

```
public function setSentAtTs(int $sentAtTs): self  
{  
    $this->sentAtTs = $sentAtTs;  
    return $this;  
}
```

```
public function getDeliveredAtTs(): string  
{  
    return $this->deliveredAtTs;  
}
```

```
public function setDeliveredAtTs(string $deliveredAtTs): self  
{  
    $this->deliveredAtTs = $deliveredAtTs;
```

```

        return $this;
    }
}

namespace App\Entity;

use App\Repository\GroupRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: GroupRepository::class)]
#[ORM\Table(name: '`group`')]
class Group
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    private string $name;

    #[ORM\ManyToMany(targetEntity: User::class, mappedBy: 'groups')]
    private Collection $users;

    public function __construct()
    {
        $this->users = new ArrayCollection();
    }
}

```

```
public function getUsers(): Collection  
{  
    return $this->users;  
}
```

```
public function addUser(User $user): void  
{  
    if (!$this->users->contains($user)) {  
        $this->users[] = $user;  
    }  
}
```

```
public function removeUser(User $user): void  
{  
    $this->users->removeElement($user);  
}  
}
```

```
namespace App\Entity;
```

```
use App\Repository\RoleRepository;  
use Doctrine\ORM\Mapping as ORM;
```

```
#[ORM\Entity(repositoryClass: RoleRepository::class)]  
class Role  
{  
    #[ORM\Id]  
    #[ORM\GeneratedValue]  
    #[ORM\Column]
```

```
private ?int $id = null;
```

```
public function getId(): ?int
```

```
{
```

```
    return $this->id;
```

```
}
```

```
#[ORM\Column]
```

```
public string $name;
```

```
public function getName(): string
```

```
{
```

```
    return $this->name;
```

```
}
```

```
public function setName(string $name): self
```

```
{
```

```
    $this->name = $name;
```

```
    return $this;
```

```
}
```

```
}
```

```
namespace App\Entity;
```

```
use App\Repository\TelegramUserRepository;
```

```
use Doctrine\ORM\Mapping as ORM;
```

```
#[ORM\Entity(repositoryClass: TelegramUserRepository::class)]
```

```
class TelegramUser
```

```
{
```

#[ORM\Id]

#[ORM\GeneratedValue]

#[ORM\Column]

private ?int \$id = null;

#[ORM\Column]

private int \$userId;

#[ORM\Column]

private int \$telegramUserId;

public function getId(): ?int

```
{  
    return $this->id;  
}
```

public function getUserId(): int

```
{  
    return $this->userId;  
}
```

public function setUserId(int \$userId): void

```
{  
    $this->userId = $userId;  
}
```

public function getTelegramUserId(): int

```
{  
    return $this->telegramUserId;  
}
```

```
public function setTelegramUserId(int $telegramUserId): void
{
    $this->telegramUserId = $telegramUserId;
}
}
```

```
namespace App\Entity;
```

```
use App\Repository\TemplateRepository;
```

```
use Doctrine\ORM\Mapping as ORM;
```

```
#[ORM\Entity(repositoryClass: TemplateRepository::class)]
```

```
class Template
```

```
{
```

```
    #[ORM\Id]
```

```
    #[ORM\GeneratedValue]
```

```
    #[ORM\Column]
```

```
    private ?int $id = null;
```

```
    #[ORM\Column]
```

```
    private string $name;
```

```
    #[ORM\Column]
```

```
    private string $channel;
```

```
    #[ORM\Column]
```

```
    private string $subject;
```

```
    #[ORM\Column]
```

```
private string $body;
```

```
public function getId(): ?int  
{  
    return $this->id;  
}  
}
```

```
public function updateFailed(DeliveryLog $deliveryLog, bool $withFlush = true):  
DeliveryLog  
{  
    $deliveryLog->setStatus(DeliveryStatusEnum::FAILED->value);  
    $deliveryLog->setDeliveredAtTs(time());  
    if ($withFlush) {  
        $this->entityManager->flush();  
    }  
  
    return $deliveryLog;  
}  
}
```

```
namespace App\Service;
```

```
use App\Repository\TelegramUserRepository;
```

```
class TelegramUserService  
{  
    public function __construct(private readonly TelegramUserRepository  
$telegramUserRepository) {}
```

```

    public function getTelegramIdByUserId(int $userId): int
    {
        $telegramUser = $this->telegramUserRepository->findOneBy(['userId' =>
$userId]);
        return $telegramUser->getTelegramUserId();
    }
}

```

```

namespace App\Service;

```

```

use App\DTO\UserDto;
use App\Entity\User;
use App\Enum\RoleEnum;
use App\Message\UserRegistered;
use App\Repository\RoleRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\Messenger\MessageBusInterface;
use Symfony\Component>PasswordHasher\Hasher\UserPasswordHasherInterface;

```

```

class UserService

```

```

{
    public function __construct(
        private readonly EntityManagerInterface $entityManager,
        private readonly MessageBusInterface $messageBus,
        private readonly UserPasswordHasherInterface $passwordHasher,
        private readonly RoleRepository $roleRepository,
    ) {}

```

```

    public function create(UserDto $dto, bool $withFlush = true): User
    {

```

```
$userRole = $this->roleRepository->findOneBy(['name' => RoleEnum::USER-  
>value]);
```

```
$user = (new User())  
->setRole($userRole)  
->setEmail($dto->getEmail())  
->setName($dto->getName())  
->setLastname($dto->getLastname())  
->setPhone($dto->getPhone());
```

```
$hashedPassword = $this->passwordHasher->hashPassword(  
    $user,  
    $dto->getPlaintextPassword(),  
);
```

```
$user->setPassword($hashedPassword);
```

```
$this->entityManager->persist($user);
```

```
if ($withFlush) {  
    $this->entityManager->flush();  
}
```

```
$this->messageBus->dispatch(  
    new UserRegistered(  
        $user->getId(),  
        $user->getName(),  
        $user->getLastname(),  
        $user->getEmail(),  
        $user->getPhone(),  
    ),
```

```

        );

        return $user;
    }
}

namespace App\Service;

use App\DTO\UserSettingsDto;
use App\Entity\UserSettings;
use App\Repository\UserSettingsRepository;
use Doctrine\ORM\EntityManagerInterface;

class UserSettingsService
{
    public function __construct(
        private readonly UserSettingsRepository $userSettingsRepository,
        private readonly EntityManagerInterface $entityManager,
    ) {}

    public function getUserSettings(int $userId): ?UserSettings
    {
        return $this->userSettingsRepository->findOneBy(['userId' => $userId],
        ['createdAts' => 'DESC']);
    }

    public function createDefault(UserSettingsDto $dto, bool $withFlush = true):
    UserSettings
    {
        $userSettings = (new UserSettings())

```

```
->setUserId($dto->getUserId())
->setChannel($dto->getChannel())
->setTemplateId($dto->getTemplateId())
->setName($dto->getName())
->setDescription($dto->getDescription())
->setCreatedAtTs(time())
->setUpdatedAtTs(time());
$this->entityManager->persist($userSettings);

if ($withFlush) {
    $this->entityManager->flush();
}

return $userSettings;
}
}
```



Звіт подібності

метадані

Назва організації

Kyiv National Economic University named after Vadym Hetman KNEU

Заголовок

Розроблення інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій

Автор

Науковий керівник / Експерт

Степаненко Миколай Ігорович Устенко С.В.

підрозділ

кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагиат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

25

Довжина фрази для коефіцієнта подібності 2



КП 2

20357

Кількість слів



КЦ

163691

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		2
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		15

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагиату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз			Колір тексту
ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://ir.kneu.edu.ua/bitstreams/6ccdb206-8499-47f8-b15b-5d42b9b14ce2/download	28 0.14 %	
2	Розроблення інформаційної управляючої системи для оптимізації діяльності цифрового інноваційного хабу 5/16/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	26 0.13 %	
3	https://ir.kneu.edu.ua/bitstreams/6ccdb206-8499-47f8-b15b-5d42b9b14ce2/download	25 0.12 %	



Звіт подібності

метадані

Назва організації

Kyiv National Economic University named after Vadym Hetman KNEU

Заголовок

Розроблення інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій

Автор

Науковий керівник / Експерт

Степаненко Миколай Ігорович Устенко С.В.

підрозділ

кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагиат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

25

Довжина фрази для коефіцієнта подібності 2



КП 2

20357

Кількість слів



КЦ

163691

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		2
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		15

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагиату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.kneu.edu.ua/bitstreams/6ccdb206-8499-47f8-b15b-5d42b9b14ce2/download	28 0.14 %
2	Розроблення інформаційної управляючої системи для оптимізації діяльності цифрового інноваційного хабу 5/16/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	26 0.13 %
3	https://ir.kneu.edu.ua/bitstreams/6ccdb206-8499-47f8-b15b-5d42b9b14ce2/download	25 0.12 %

2	https://repository.sspu.edu.ua/collections/82f028f5-bd8a-46a8-8649-dc944aa54df3	13 (2) 0.06 %
3	https://wezm.com.ua/ua/blog/laravel-vs-yii-framework-cto-luchshe-dlja-vashego-biznesa	12 (1) 0.06 %
4	https://revolution.allbest.ru/audit/00395885_0.html	10 (1) 0.05 %
5	https://www.studocu.com/uk/document/kiiivs%D1%8Ckiy-natsional%D1%8Cniy-ekonomichnyy-universitet-imeni-vadima-get%D1%8Cman/finansi-bankivs%D1%8Cka-sprava-ta-strakhuvannya/individual%D1%8Cni-investori-na-fondovomu-rinku/78336831	9 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)
------------------	-------	---------------------------------------

Публікації

ІНФОРМАЦІЙНА УПРАВЛЯЮЧА СИСТЕМА ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗОВНІШНІХ НОТИФІКАЦІЙ

Степаненко Миколай Ігорович

Здобувач вищої освіти інституту
інформаційних технологій в економіці
*Київський національний економічний
університет імені Вадима Гетьмана, Україна*

**Науковий керівник: Устенко Станіслав
Веніамінович**

доктор економічних наук, професор кафедри
інформаційних систем в економіці
*Київський національний економічний
університет імені Вадима Гетьмана, Україна*

Огляд сучасних наукових робіт з автоматизації зовнішніх нотифікацій свідчить про активне впровадження цифрових технологій у різних сферах [1].

Firebase Cloud Messaging (FCM) — це один із найпопулярніших сервісів для надсилання сповіщень, який розробила компанія Google. Він широко використовується у мобільних додатках, веб-додатках і навіть у системах IoT [2].

FCM забезпечує надійну доставку повідомлень з мінімальними затримками. Він дає змогу надсилати як звичайні сповіщення для відображення на пристрої, так і приховані повідомлення для фонової обробки. Крім того, система підтримує таргетинг за групами пристроїв, темами або за конкретними токенами користувачів [2].

Якщо не впровадити інформаційну управляючу систему для автоматизації зовнішніх нотифікацій, компанія може зіткнутися з рядом серйозних проблем, які вплинуть на ефективність роботи, якість обслуговування клієнтів і навіть фінансові показники [3].

Одна з головних проблем — затримка в комунікації. Якщо повідомлення відправляються вручну, це потребує значних часових і людських ресурсів. Наприклад, якщо співробітники мають самотійно надсилати кожен email або SMS,

це займає багато часу і може призвести до запізнень у важливих сповіщеннях [3]. Клієнт може не отримати інформацію про зміну статусу свого замовлення, що викличе роздратування і знизить рівень довіри до компанії.

Ще одна серйозна проблема – ризик людських помилок [4]. При ручному введенні та надсиланні повідомлень можливі помилки у введенні даних, неправильно вказані номери телефонів або електронні адреси, дублювання або, навпаки, відсутність повідомлення у потрібний момент. Це може призвести до ситуацій, коли клієнти отримують неактуальну або некоректну інформацію, що може негативно позначитися на репутації компанії.

Відсутність автоматизованої системи також ускладнює масові розсилки та персоналізацію повідомлень. Якщо компанія намагається індивідуально комунікувати з кожним клієнтом вручну, це забирає багато часу і ресурсів. У той же час, відправлення однакових шаблонних повідомлень без персоналізації може сприйматися як спам і не викликати у клієнтів зацікавленості або довіри. Це може знизити ефективність маркетингових кампаній і комунікацій загалом.

Інформаційна управляюча система для автоматизації процесу зовнішніх нотифікацій потрібна для ефективної та своєчасної комунікації між компанією та її клієнтами, партнерами або співробітниками. У сучасному світі швидкість передачі інформації має велике значення, а ручне розсилання повідомлень займає багато часу і ресурсів. Автоматизована система дозволяє налаштовувати повідомлення таким чином, щоб вони надходили в потрібний час, через оптимальний канал і з урахуванням індивідуальних потреб кожного одержувача [5]. Також однією з основних переваг такої системи є можливість мультиканальної комунікації. Це означає, що повідомлення можуть бути надіслані через різні канали – SMS, email, месенджери, push-нотифікації – залежно від того, який варіант зручніший для користувача або найбільш ефективний у конкретному випадку [6]. Крім того, система може автоматично вибирати найбільш відповідний канал залежно від доступності абонента або важливості повідомлення. Ще один важливий аспект –

персоналізація та адаптація повідомлень [7]. Замість стандартних шаблонних розсилок система може формувати унікальні повідомлення з урахуванням імені отримувача, його попередніх дій, історії покупок чи інших важливих даних. Це підвищує довіру до повідомлень та сприяє кращому залученню клієнтів. Також можна додавати вкладення, що особливо корисно для надсилання рахунків, договорів або інших документів. Для наочного подання ключових переваг інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій, доцільно структурувати їх у вигляді таблиці. Нижче наведено основні переваги інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій (табл 1).

Таблиця 1

Переваги інформаційної управляючої системи для автоматизації процесу зовнішніх нотифікацій

Перевага	Опис
Уніфікований інтерфейс	Один вхідний інтерфейс для різних каналів (SMS, email, push), що спрощує інтеграцію
Легка розширюваність	Можна додати нові канали або провайдерів без зміни бізнес-логіки
Незалежність від провайдерів	Сервіс абстрагує сторонні API, що дозволяє легко змінити постачальника послуг
Вбудована валідація	Дані перевіряються перед надсиланням, що знижує ризик помилок
Повторне використання	Повідомлення можна шаблонізувати та повторно використовувати в різних контекстах
Логування та моніторинг	Централізоване логування дає змогу відслідковувати статус доставки по всіх каналах

Затримка та планування	Можна реалізувати відкладену відправку, черги та повторні спроби надсилання
Централізовані налаштування	Конфігурація каналів, API-ключів тощо зберігається в одному місці (наприклад, у Symfony config)
Можливість тестування	Система допускає ізоляцію логіки для тестування без фактичної відправки повідомлень
Безпека	Вразливості сторонніх сервісів не впливають на бізнес-логіку додатку

Автоматизація процесу відправки дозволяє значно знизити людський фактор і виключити затримки у відправленні критично важливих повідомлень. Наприклад, система може надсилати нагадування про майбутні зустрічі, сповіщати клієнтів про зміну статусу їхніх замовлень або автоматично надсилати повідомлення про заборгованість. Завдяки цьому компанії можуть підтримувати постійний зв'язок з клієнтами без додаткових зусиль з боку персоналу.

Контроль і моніторинг процесу відправлення є ще однією ключовою функцією такої системи. Вона дозволяє відстежувати статус доставки кожного повідомлення, перевіряти, чи було воно відкрито, і навіть аналізувати ефективність комунікації. Якщо повідомлення не доставлено, система може запропонувати альтернативний канал зв'язку або повторну відправку. Усі дані логуються, що дозволяє легко знайти проблеми та оптимізувати процеси.

Інформаційна управляюча система для автоматизації процесу зовнішніх нотифікацій також легко інтегрується з іншими програмними рішеннями, такими як CRM, ERP, білінгові платформи, що дає змогу автоматично отримувати необхідні дані для формування повідомлень.

Загалом, автоматизована система нотифікацій допомагає компаніям покращити взаємодію з клієнтами, підвищити ефективність роботи та мінімізувати

витрати на комунікацію. Вона дозволяє швидко та безпечно доставляти важливу інформацію, зменшуючи ризик людських помилок і підвищуючи рівень задоволеності користувачів.

Список використаних джерел:

1. Атаманчук С. М., Подкопаєв С. В., Батиргарєєва В. С. Цифровізація кримінального провадження: стан та перспективи. *Науково-дослідний інститут вивчення проблем злочинності імені академіка В. В. Сташиса Національної академії правових наук України*. Харків : Науково-дослідний інститут вивчення проблем злочинності імені академіка В. В. Сташиса Національної академії правових наук України, 2024.
2. Firebase Cloud Messaging. URL: <https://firebase.google.com/docs/cloud-messaging>.
3. Десять переваг поведінкової автоматизації для медіа сайтів: історія успіху компанії з Колумбії. <https://gravitec.net/ua/blog/desyat-perevag-povedinkovoyi-avtomaty-zatsiyi-dlya-media-sajtiv/>.
4. Скрипець В. О. Сервіс прогнозування курсу валют на основі моделі машинного навчання з системою підписок та нотифікацій / КПП ім. Ігоря Сікорського. КПП ім. Ігоря Сікорського, 2024. 96 с.
5. ТОП-8 сервісів для розсилки push-повідомлень. URL: <https://hostiq.ua/blog/ukr/push-messages/>.
6. Reasons to Run Automated Notifications and Alerts With a Tracking Software and How?. URL: <https://ezo.io/ezofficeinventory/blog/automated-notifications/>.
7. Reasons to Automate Push Notifications for Your Business. URL: <https://pushninja.com/automate-push-notifications>.

КАНАЛИ НОТИФІКАЦІЙ ІНФОРМАЦІЙНОЇ УПРАВЛЯЮЧОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗОВНІШНІХ НОТИФІКАЦІЙ

Степаненко Миколай Ігорович

Здобувач вищої освіти інституту
інформаційних технологій в економіці
*Київський національний економічний
університет імені Вадима Гетьмана, Україна*

**Науковий керівник: Устенко Станіслав
Веніамінович**

доктор економічних наук, професор кафедри
інформаційних систем в економіці
*Київський національний економічний
університет імені Вадима Гетьмана, Україна*

У сучасних інформаційних системах ефективна доставка сповіщень є важливою складовою взаємодії з користувачами. Основними каналами є електронна пошта, SMS та push-повідомлення [1].

Електронна пошта є універсальним каналом для передачі великих обсягів інформації, формальних повідомлень, документів та квитанцій. Її основна перевага полягає у можливості прикріплення файлів і налаштуванні структурованого формату листа. Однак часом вона може мати низьку оперативність, оскільки користувачі не завжди миттєво перевіряють пошту [2].

SMS-повідомлення забезпечують високу швидкість доставки та зручність для коротких термінових повідомлень. Вони не потребують підключення до інтернету, що робить їх особливо актуальними у критичних ситуаціях. Проте обмеження на обсяг тексту і вартість відправки роблять їх менш ефективними для складних або багатокomпонентних сповіщень [3].

Push-повідомлення у мобільних додатках є зручним інструментом для залучення уваги користувачів у реальному часі. Вони дозволяють інтерактивно взаємодіяти з повідомленням (наприклад, одразу перейти до певної сторінки в

додатку). Головним обмеженням є необхідність попереднього встановлення додатку та надання дозволу на отримання сповіщень [4].

Одним із прикладів сучасних рішень для багатоканальної доставки сповіщень є платформа SendPulse. Вона надає можливість надсилати повідомлення через електронну пошту, SMS, веб-push, Viber та чат-ботів у месенджерах [5]. Завдяки цьому користувачі можуть обирати найбільш ефективний канал комунікації залежно від типу повідомлення, аудиторії та контексту. Наприклад, для нагальних сповіщень про безпеку SendPulse дозволяє швидко доставити SMS, а для рекламних кампаній — налаштувати масову розсилку електронною поштою або через Viber із можливістю додавання медіафайлів. Таким чином, використання багатоканальної стратегії в SendPulse забезпечує вищу ймовірність доставки повідомлення та підвищує загальну ефективність комунікації.

Для кращого розуміння особливостей різних каналів доставки сповіщень доцільно провести їх порівняльний аналіз за основними критеріями. Нижче наведено ключові характеристики електронної пошти, SMS-повідомлень та push-сповіщень, що дозволяє оцінити переваги та обмеження кожного каналу залежно від конкретних умов використання (табл. 1).

Таблиця 1

Ключові характеристики електронної пошти, SMS-повідомлень та push-сповіщень

Критерій	Електронна пошта	SMS-повідомлення	Push-повідомлення
Швидкість доставки	Середня	Висока	Висока
Обсяг повідомлення	Великий	Обмежений (~160 символів)	Короткі повідомлення
Вартість	Низька або безкоштовна	Висока (особливо при масових розсилках)	Низька (переважно безкоштовно)
Залежність від інтернету	Так	Ні	Так

Форматування тексту	Підтримується (HTML, вкладення)	Немає форматування	Мінімальне форматування
Доступність користувачеві	Перевіряється не завжди оперативно	Миттєве отримання	Миттєве отримання за наявності додатку
Технічні вимоги	Email-клієнт або браузер	Наявність телефону з мобільним зв'язком	Встановлений додаток і дозвіл на сповіщення
Використання в практиці	Офіційні листи, новинні розсилки	Критичні сповіщення, банківські коди	Нагадування, маркетинг, залучення у застосунках

Наявність різних каналів для надсилання сповіщень вирішує одну з ключових проблем сучасної цифрової комунікації — підвищення ефективності доставки інформації до користувачів у різних умовах [6].

У реальному житті користувачі перебувають у різному середовищі: хтось має доступ до інтернету лише через смартфон, інший може не мати встановленого потрібного застосунку, хтось рідко перевіряє електронну пошту або перебуває у зоні поганого мобільного зв'язку.

Використання кількох каналів дозволяє:

1. Забезпечити максимальну досяжність;
2. Адаптувати стиль і зміст повідомлення;
3. Оптимізувати витрати.

Таким чином, різноманітність каналів комунікації — це відповідь на потребу в гнучкості, надійності та економічній ефективності систем сповіщень у складному інформаційному середовищі.

Список використаних джерел

1. Що таке SMS-розсилки через API та як їх налаштувати Джерело: <https://hub.kyivstar.ua/articles/shho-take-rozsilki-povidomlen-cherez-api-ta-yak-yih->

nalashtuvati. hub.kyivstar.ua. URL: <https://hub.kyivstar.ua/articles/shho-take-rozsilki-povidomlen-cherez-api-ta-yak-yih-nalashtuvati>.

2. Що таке SMS-розсилки через API та як їх налаштувати Джерело: <https://hub.kyivstar.ua/articles/shho-take-rozsilki-povidomlen-cherez-api-ta-yak-yih-nalashtuvati>. hub.kyivstar.ua. URL: <https://hub.kyivstar.ua/articles/shho-take-rozsilki-povidomlen-cherez-api-ta-yak-yih-nalashtuvati>.
3. Oludare O., Olaniyan A., Olalekan E. SMS-Based Event Notification System. IISTE. IISTE, 2023.
4. Сенів А. В. Метод керування Push-сповіщеннями у гібридному мобільному додатку на основі MAUI.NET Framework. WUNU. Тернопіль : ЗУНУ, 2023.
5. Дієві маркетингові інструменти для розвитку бізнесу. URL: <https://sendpulse.ua/>.
6. SMS vs. Email Marketing: Which Is More Effective?. URL: <https://info.joinsubtext.com/blog/sms-vs-email-marketing-which-is-more-effective>.