НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення для візуалізації даних щодо контролю та переробки відходів

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Перевознюк К.А.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Перевознюк Катерині Андріївні

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення для візуалізації даних щодо контролю та переробки відходів

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.05.26
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення, інформації про поточний рівень відходів у регіонах Києва, надання інформації про переробку відходів

Перелік питань що розглядаються:
1. Системний аналіз предметної області
2. Проєктування системи аналізу та контролю переробки відходів
3. Розробка програмного забезпечення моніторингу відходів
4. Впровадження та експлуатація системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Перевознюк К.А.
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП....4
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ....6
1.1 Опис предметної області....6
1.2 Аналіз вимог до системи....10
1.3 Моделювання предметної області....14
1.4 Огляд існуючих програмних рішень....18
1.5 Постановка завдання....23
2 ПРОЄКТУВАННЯ СИСТЕМИ АНАЛІЗУ ТА КОНТРОЛЮ ПЕРЕРОБКИ ВІДХОДІВ....24
2.1 Логічна модель даних у вигляді ER-діаграми....24
2.2 Діаграма класів та кооперацій....26
2.3 Діаграма пакетів....30
2.4 Діаграма компонентів....33
2.5 Діаграма активності....35
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОНІТОРИНГУ ВІДХОДІВ....38
3.1 Архітектура та система управління інформаційною базою....38
3.2 Розробка інформаційної бази....40
3.3 Вибір інструментарію для створення прикладного програмного забезпечення....46
3.4 Алгоритмізація та програмування програмних модулів....49
4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ....53
4.1 Тестування системи....53
4.2 Визначення технічних вимог до використання системи....59
4.3 Склад інсталяційного пакету....59
ВИСНОВКИ....63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ....65
ДОДАТОК А....68
ДОДАТОК Б....70

ВСТУП

Забруднення навколишнього середовища та накопичення відходів є одними з найактуальніших проблем сучасних міст. Зростання кількості населення, обсягів споживання та індустріалізації призводить до постійного збільшення кількості відходів, що потребують контролю та ефективного управління. У цьому контексті впровадження цифрових рішень для збору, аналізу та візуалізації екологічних даних відіграє ключову роль у забезпеченні прозорості процесів та прийнятті обґрунтованих управлінських рішень. Особливо актуальним це питання є для великих урбанізованих регіонів, таких як місто Київ.

Об’єктом дослідження є інформаційна система для екологічного моніторингу та управління відходами в умовах міського середовища.

Предметом дослідження виступають інформаційні технології збору, обробки та візуалізації даних про рівень забруднення та показники переробки відходів.

Метою роботи є розробка веб-орієнтованого програмного забезпечення для інтерактивної візуалізації екологічних даних щодо контролю та переробки відходів, яке забезпечує централізований моніторинг екологічної ситуації в різних регіонах міста, підтримує процес прийняття управлінських рішень та сприяє підвищенню ефективності екологічної політики.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

- проаналізувати предметну область і визначити актуальні інформаційні потреби користувачів системи;
- дослідити наявні програмні рішення у сфері екологічного моніторингу та візуалізації даних;
- сформулювати вимоги до функціональності, інтерфейсу та архітектури системи;

– розробити концептуальну та логічну модель бази даних для зберігання екологічної інформації;
– побудувати UML-діаграми (прецедентів, класів, активностей, компонентів) для опису структури й поведінки системи;
– реалізувати веб-додаток із використанням фреймворку Symfony (PHP), MySQL та технологій HTML/CSS/JavaScript;
– забезпечити тестування основних функцій та проаналізувати результати апробації системи на прикладі районів Києва.

Програмне забезпечення реалізовано у вигляді веб-додатку з адаптивним інтерфейсом і можливістю взаємодії з датчиками, що збирають екологічні показники в реальному часі. Система дозволяє здійснювати перегляд динаміки забруднення, відслідковувати ефективність переробки, генерувати інтерактивні звіти та приймати оперативні рішення на основі аналізу даних.

Результати роботи були апробовані у вигляді повнофункціонального прототипу, що включає реєстрацію, облік, перегляд, аналіз і підтвердження екологічних операцій. Функціональність протестовано відповідно до визначених сценаріїв використання. Розроблена система демонструє потенціал для інтеграції в міську інфраструктуру управління відходами та екологічного планування.

Структура пояснювальної записки охоплює 70 сторінок та включає 4 розділи, 2 додатки, 32 ілюстрації, 1 таблицю та 31 джерело. У першому розділі проведено системний аналіз проблемної області та постановку завдання. У другому – змодельовано предметну область та описано ключові діаграми. Третій розділ присвячено проєктуванню архітектури та реалізації програмної системи. У четвертому подано особливості впровадження та тестування. У висновках узагальнено результати роботи та перспективи подальшого розвитку.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Ефективне управління відходами та моніторинг екологічного стану міського середовища є важливою складовою забезпечення сталого розвитку. У контексті урбанізованих територій, таких як Київ, дедалі актуальнішою стає проблема нагромадження побутових і промислових відходів, недостатньої інфраструктури для їх своєчасної переробки, а також підвищення рівня забруднення повітря, ґрунтів і водних ресурсів. Це створює загрози для здоров’я населення та екосистем, а також ускладнює планування екологічних і управлінських заходів.

Традиційні підходи до управління відходами в багатьох випадках залишаються малоефективними через відсутність механізмів оперативного збору та аналізу даних. Часто такі системи є розрізненими між різними установами, не мають єдиного інформаційного простору й не забезпечують інструментів для візуального представлення екологічної інформації або прогнозування динаміки забруднення. Це ускладнює своєчасне реагування, прийняття рішень щодо розподілу ресурсів і реалізацію екологічної політики. Відсутність актуальних та достовірних даних також створює перешкоди для об’єктивної оцінки ефективності заходів із переробки та ідентифікації проблемних зон, що потребують першочергової уваги [3].

Суттєвим бар’єром є також недостатня прозорість екологічної інформації та обмежений доступ до неї як для органів місцевої влади, так і для громадян. Попри наявність окремих сенсорних пристроїв і локальних систем моніторингу, отримані дані часто не інтегруються в єдину платформу, що унеможливлює їх повноцінне використання для аналізу, порівняння й ухвалення рішень.

В останні роки цифрова трансформація в екологічному моніторингу набирає обертів у всьому світі. Такі технології, як Інтернет речей (IoT), хмарні обчислення та інструменти візуалізації даних, були застосовані для створення

інтелектуальних систем, що підвищують екологічну обізнаність та операційну ефективність. Однак в Україні та зокрема в Києві такі системи все ще перебувають на ранніх стадіях розробки та ще не отримали широкого впровадження.

У глобальному контексті спостерігається активний розвиток цифрових технологій у сфері моніторингу навколишнього середовища. Застосування рішень на базі Інтернету речей, хмарних обчислень та інструментів інтерактивної візуалізації сприяє створенню інтелектуальних екосистем, які забезпечують оперативний контроль за станом довкілля та підвищують ефективність управління. Водночас в Україні, зокрема у Києві, впровадження таких систем відбувається повільно й наразі обмежується локальними або експериментальними проєктами без масштабного застосування.

З огляду на цей контекст, існує очевидна потреба в комплексному програмному рішенні, яке дозволило б:

1. централізований збір даних про навколишнє середовище з датчиків у різних регіонах;
2. візуалізацію рівнів забруднення та обсягів відходів у режимі реального часу;
3. моніторинг та аналіз зусиль з переробки;
4. створення звітів для підтримки прийняття рішень міськими адміністраціями та експертами з охорони навколишнього середовища.

Усуваючи ці прогалини, система, розроблена в цій дисертації, має на меті сприяти більш обґрунтованому, прозорому та проактивному підходу до управління навколишнім середовищем у Києві.

Тематика цієї роботи зосереджена на застосуванні сучасних інформаційних технологій у сфері охорони навколишнього середовища, зокрема в контексті моніторингу рівня відходів та забруднення міського середовища. Зростання обсягів відходів у великих містах, таких як Київ, у поєднанні зі зростанням екологічної стурбованості вимагає більш досконалих та ефективних

інструментів для відстеження стану навколишнього середовища та просування практик сталого управління відходами.

Одним з основних компонентів цієї галузі є моніторинг навколишнього середовища, який включає збір та аналіз даних, пов'язаних з різними показниками забруднення. У контексті цієї системи спеціалізовані датчики розгортаються в різних районах Києва для збору інформації в режимі реального часу про якість повітря та наявність забруднювачів навколишнього середовища. Ці дані слугують основою для виявлення проблемних зон та вжиття відповідних заходів для пом'якшення екологічної шкоди.

Ще одним важливим аспектом є контроль та управління відходами в місті. Моніторинг накопичення твердих відходів, своєчасне планування збору та визначення ділянок з високою концентрацією відходів є важливими кроками у підтримці чистого міського простору. Візуалізуючи ці процеси, система підтримує своєчасне втручання та допомагає уникнути переповнення контейнерів для сміття та нерегульованого скидання відходів [3].

Тісно пов'язаний з контролем відходів процес переробки, який відіграє вирішальну роль у зменшенні впливу на навколишнє середовище. Переробка не лише мінімізує обсяг відходів, що потрапляють на сміттєзвалища, але й сприяє чистішим міським районам та збереженню ресурсів. Система, розроблена в рамках цієї дисертації, дозволяє відстежувати діяльність з переробки в різних районах та інтегрує ці дані в динамічний індекс чистоти – показник, який відображає екологічний стан певного регіону на основі рівня забруднення та ефективності переробки.

В основі цього рішення лежить потужний компонент візуалізації даних. Зібрана інформація перетворюється на інтерактивні графіки, діаграми та таблиці, що полегшує розуміння екологічної ситуації як для фахівців, так і для осіб, які приймають рішення. Інформаційні панелі пропонують комплексне уявлення про стан різних регіонів, дозволяючи користувачам порівнювати тенденції, аналізувати результати та створювати детальні звіти.

З урахуванням викладених проблем, актуальним постає завдання створення комплексного, гнучкого програмного забезпечення, здатного інтегрувати засоби збору екологічних даних з датчиків, забезпечувати їх обробку, аналітичну інтерпретацію та наочне представлення у вигляді графіків, карт і звітів. Таке рішення повинно стати ефективним інструментом для підтримки діяльності міських служб, спеціалістів з охорони навколишнього середовища та керівників підприємств у процесі прийняття управлінських рішень у сфері поводження з відходами.

З технічного погляду система реалізована у вигляді веб-додатку, розробленого з використанням мови програмування PHP і фреймворку Symfony. Для зберігання та керування структурованими даними застосовується реляційна система керування базами даних MySQL. Користувацький інтерфейс створено на основі сучасних фронтенд-технологій, що забезпечує зручну, адаптивну й інтуїтивно зрозумілу взаємодію з візуалізованою інформацією. Запропонований підхід дозволяє користувачам оперативно аналізувати екологічні показники та приймати обґрунтовані рішення на основі актуальних даних.

У табл. 1.1 подано ключові сутності предметної області, які використовуються для моделювання системи екологічного моніторингу в міському середовищі.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	Атрибут	Опис
1	2	3
Регіон	Назва	Назва району або частини міста Києва
	Рівень забруднення	Поточний показник забруднення в регіоні
	Кількість сміття	Обсяг накопиченого сміття
	Індекс чистоти	Загальна оцінка екологічного стану регіону
Сенсор	Тип показника	Який саме показник фіксує сенсор (напр., CO_2, тверді відходи тощо)

Таблиця 1.1 (продовження)

1	2	3
	Поточне значення	Дані, які сенсор зафіксував у реальному часі
	Розташування	Місце встановлення сенсора (вулиця, координати)
	Дата та час фіксації	Коли саме було зафіксовано значення
Відходи	Тип відходів	Категорія сміття (органічне, пластик, скло, тощо)
	Обсяг	Кількість зібраних відходів у певному місці
	Місце накопичення	Де саме були зібрані ці відходи

Представлені сутності охоплюють як статичні об’єкти (регіони, типи відходів), так і динамічні процеси (збір даних, фіксація сенсорами). Атрибути відображають необхідні параметри для побудови інтегрованої системи моніторингу та дозволяють структурувати інформацію з різних джерел у єдиній базі даних.

Таким чином, предметна область включає комплекс взаємопов’язаних об’єктів, пов’язаних із геопросторовими одиницями (районами), фізичними показниками стану середовища (забруднення, обсяги сміття), пристроями збору інформації (сенсорами) та видами відходів. Формалізація цих об’єктів є необхідним кроком для побудови ефективної програмної системи, яка забезпечує аналіз і візуалізацію екологічних даних у режимі реального часу [22].

1.2 Аналіз вимог до системи

Розробка інформаційної системи вимагає чіткого формулювання вимог, що визначають її функціональні можливості, технічні обмеження та особливості взаємодії з користувачами. На цьому етапі проведено узагальнення функціональних, нефункціональних та інтерфейсних вимог до системи візуалізації даних щодо контролю та переробки відходів.

Функціональні вимоги визначають перелік основних можливостей, які повинна забезпечувати система.

- реєстрація, автентифікація та керування обліковими записами користувачів;
- отримання та зберігання даних із сенсорів у режимі реального часу;
- візуалізація екологічних показників у формі графіків, таблиць, карт та індикаторів;
- доступ до історичних даних та генерація звітів за часовими діапазонами;
- налаштовувані сповіщення при перевищенні контрольних параметрів (наприклад, рівень забруднення або заповнення контейнерів);
- аналіз ефективності переробки відходів;
- підтримка різних ролей користувачів (адміністратор, аналітик, оператор, менеджер) з обмеженнями доступу до функціоналу.

Нефункціональні вимоги охоплюють характеристики системи, що не стосуються її функціональності, але визначають якість, продуктивність і надійність роботи:

- масштабованість – можливість додавання нових регіонів, сенсорів та користувачів без зниження продуктивності;
- швидкодія – оперативне відображення оновлених даних на інформаційних панелях;
- надійність – стабільна робота системи з мінімальними ризиками втрати даних;
- безпека – шифрування конфіденційної інформації, автентифікація, контроль доступу за ролями [4];
- доступність – можливість доступу через веб-браузер із різних пристроїв (настільних і мобільних);
- модульність – можливість розширення функціональності шляхом додавання нових компонентів.

Окрему увагу слід приділити вимогам до інтерфейсу користувача, який має бути інтуїтивно зрозумілим, адаптивним і зручним у використанні. Основні вимоги:

- чітка навігація між основними розділами (райони, показники, операції, звіти);
- використання панелі керування з налаштовуваними віджетами;
- підтримка інтерактивних елементів (графіки, карти, індикатори) [6];
- валідація введених даних з повідомленнями про помилки;
- адаптація до мобільних пристроїв (responsive design);
- відображення актуальної інформації без перезавантаження сторінки (реалізація через AJAX/WebSocket).

Інтерфейс користувача системи має забезпечувати просту та зрозумілу взаємодію як для технічних, так і для нетехнічних користувачів. Основні вимоги включають чітку навігацію, адаптивну структуру сторінок, розмежування доступу за ролями та інтуїтивно зрозумілий макет. Особливу увагу слід приділити налаштовуваній інформаційній панелі, що дозволяє персоналізувати відображення ключових показників: обсягу відходів, рівня забруднення, ефективності переробки тощо. Система має підтримувати динамічну візуалізацію у форматі графіків, діаграм, карт і індикаторів, що оновлюються в реальному часі [6].

Крім основних функцій, важливою є підтримка інтерактивних карт, що дозволяють фільтрувати й аналізувати просторові дані за різними критеріями. Інтерфейс має бути повністю адаптивним до мобільних пристроїв, забезпечуючи рівний доступ з будь-якої платформи. Введення й редагування даних повинно відбуватися через зручні форми з вбудованою перевіркою коректності, що підвищує якість даних та зменшує ймовірність помилок під час роботи з системою.

З урахуванням вищезазначених функціональних, нефункціональних та інтерфейсних характеристик було сформовано узагальнену специфікацію вимог до інформаційної системи. Вона охоплює ключові аспекти взаємодії користувачів із системою, технічні параметри її роботи, а також вимоги до зручності та безпеки використання. Деталізовану структуру вимог подано в таблиці 1.2.

Таблиця 1.2

Специфікація вимог до інформаційної системи

№	Категорія	Опис вимоги
1	Функціональна	Користувачі можуть реєструватися та входити в систему
2		Система отримує дані з сенсорів у режимі реального часу
3		Дані відображаються у вигляді графіків, таблиць та інтерактивної карти
4		Користувачі можуть формувати звіти за обраними критеріями
5		Надсилаються сповіщення при перевищенні встановлених екологічних показників
6	Нефункціональна	Система здатна обробляти запити одночасно від ≥100 користувачів
7		Інтерфейс працює однаково на ПК, планшетах і смартфонах
8		Передача даних захищена SSL-шифруванням
9	Інтер-фейсна	Головна панель відображає ключові екологічні індикатори
10		Карта дозволяє фільтрувати дані за регіонами, часом, типом відходів

Специфікація вимог охоплює всі ключові аспекти роботи майбутньої системи: від базової взаємодії користувача з інтерфейсом до масштабованості та безпеки. Вона також враховує можливість адаптації під різні типи користувачів і сценарії застосування. Такий підхід дозволяє забезпечити цілісність, зручність і стійкість системи до навантажень.

Аналіз вимог до системи дозволив сформулювати повний перелік характеристик, які визначають її функціональність, якість, безпеку та взаємодію з користувачами. Сформована модель вимог є основою для подальшого моделювання архітектури, вибору технологій реалізації та побудови логіки взаємодії всіх компонентів інформаційної системи.

1.3 Моделювання предметної області

Для побудови ефективної програмної системи необхідно чітко визначити структуру та поведінку об’єктів предметної області. У цьому розділі здійснено моделювання основних аспектів системи візуалізації даних щодо контролю та переробки відходів за допомогою уніфікованої мови моделювання (UML). Моделі охоплюють сценарії взаємодії користувачів із системою та порядок обміну повідомленнями між її компонентами.

Діаграма варіантів використання (use case diagram) є одним із базових засобів візуалізації в UML, що відображає взаємодію зовнішніх суб’єктів (акторів) із функціональністю програмної системи [20]. Вона використовується для графічного представлення функцій, які система повинна виконувати на запит користувачів або інших зовнішніх компонентів. Такі діаграми формуються на початкових етапах розробки для формалізації вимог і кращого розуміння системної поведінки з позиції кінцевого користувача [8, 21].

Діаграма варіантів використання описує взаємодію зовнішніх суб’єктів (акторів) із системою, де кожен актор — це особа, пристрій або інша система, що ініціює певні дії для досягнення цілей, таких як перегляд даних чи формування звітів. Вказані дії моделюються як варіанти використання, що відображають функціональні можливості системи. Усі варіанти розміщуються в межах системи, які позначають обсяг її функціональності та відокремлюють її від зовнішнього середовища.

На основі проведеного аналізу вимог до системи було створено діаграму варіантів використання, що графічно відображає основні сценарії взаємодії користувачів із програмним забезпеченням. Ця діаграма представлена на рисунку 1 та охоплює ключові дії, які можуть виконувати різні категорії користувачів залежно від їхніх ролей.

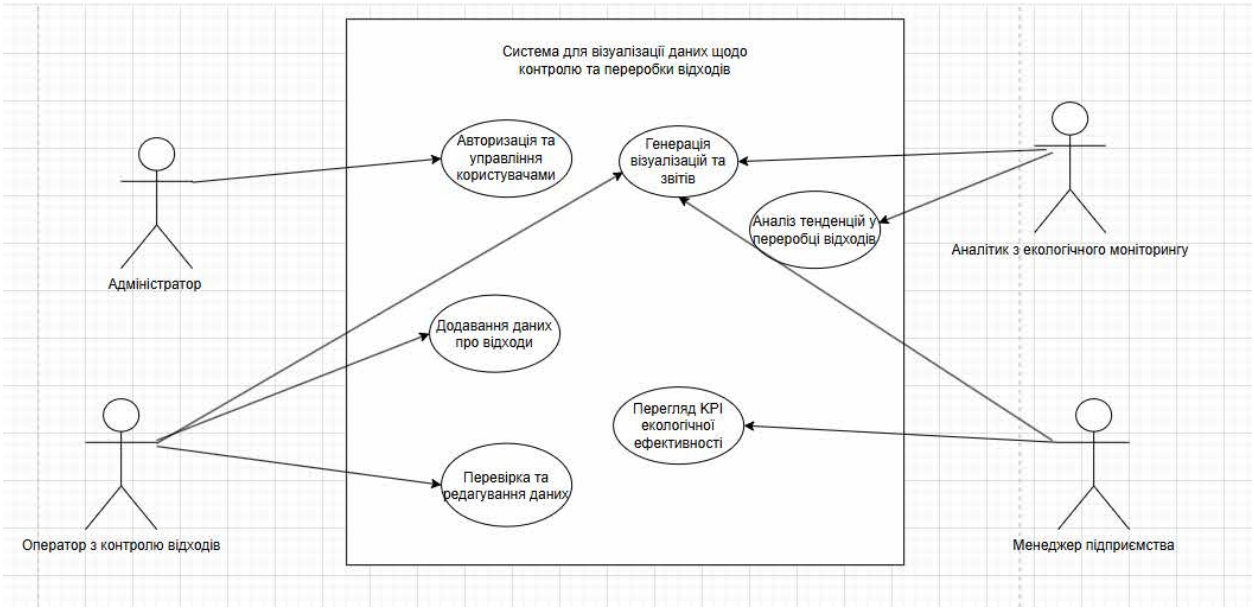

Рис. 1 Діаграма прецедентів

Для побудови діаграми прецедентів було визначено чотири ключові типи користувачів (акторів), кожен з яких має доступ до відповідного функціоналу системи згідно зі своєю роллю. Їх взаємодію з основними варіантами використання узагальнено в таблиці 1.3.

Таблиця 1.3

Актори та варіанти використання системи

Актор	Основні прецеденти
Адміністратор	Авторизація Управління користувачами
Оператор з контролю відходів	Додавання даних Редагування даних Генерація звітів та візуалізацій
Аналітик з екологічного моніторингу	Генерація звітів та візуалізацій Аналіз тенденцій
Менеджер підприємства	Перегляд KPI Ознайомлення зі звітністю та візуалізаціями

Для узагальнення змісту кожного прецеденту в табл. 1.4 наведено стислий опис основних функціональних сценаріїв.

Таблиця 1.4

Сценарії варіантів використання системи

№	Варіант використання	Опис взаємодії
1	Додавання даних про відходи	Оператор фіксує інформацію про тип, обсяг та місце збору сміття
2	Генерація візуалізацій	Аналітик формує графіки й карти за обраними параметрами
3	Перегляд екологічних звітів	Менеджер аналізує ефективність очищення за KPI
4	Керування користувачами	Адміністратор створює та редагує профілі доступу

Представлена діаграма варіантів використання та відповідні таблиці демонструють структуру ролей користувачів у системі та типові сценарії взаємодії з основними функціями. Такий підхід дозволяє формалізувати вимоги до інтерфейсу та поведінки системи, що є основою для подальшого проектування програмної логіки.

Діаграма послідовності відображає порядок обміну повідомленнями між компонентами системи або користувачами під час виконання певного сценарію. Вона демонструє часову послідовність взаємодій, де кожен учасник представлений вертикальною лінією життя, а дії — горизонтальними стрілками [9]. Активні фази об'єктів позначаються прямокутниками на цих лініях. У разі наявності умов або повторень використовуються відповідні графічні конструкції. Такий тип діаграми дає змогу проаналізувати логіку виконання операцій у динаміці.

На рис. 2 зображено діаграму послідовності для основних сценаріїв взаємодії між користувачами (оператор, адміністратор, менеджер, аналітик) та

ключовими компонентами системи: модулем реєстрації відходів, базою даних і системою аналітики та звітності.

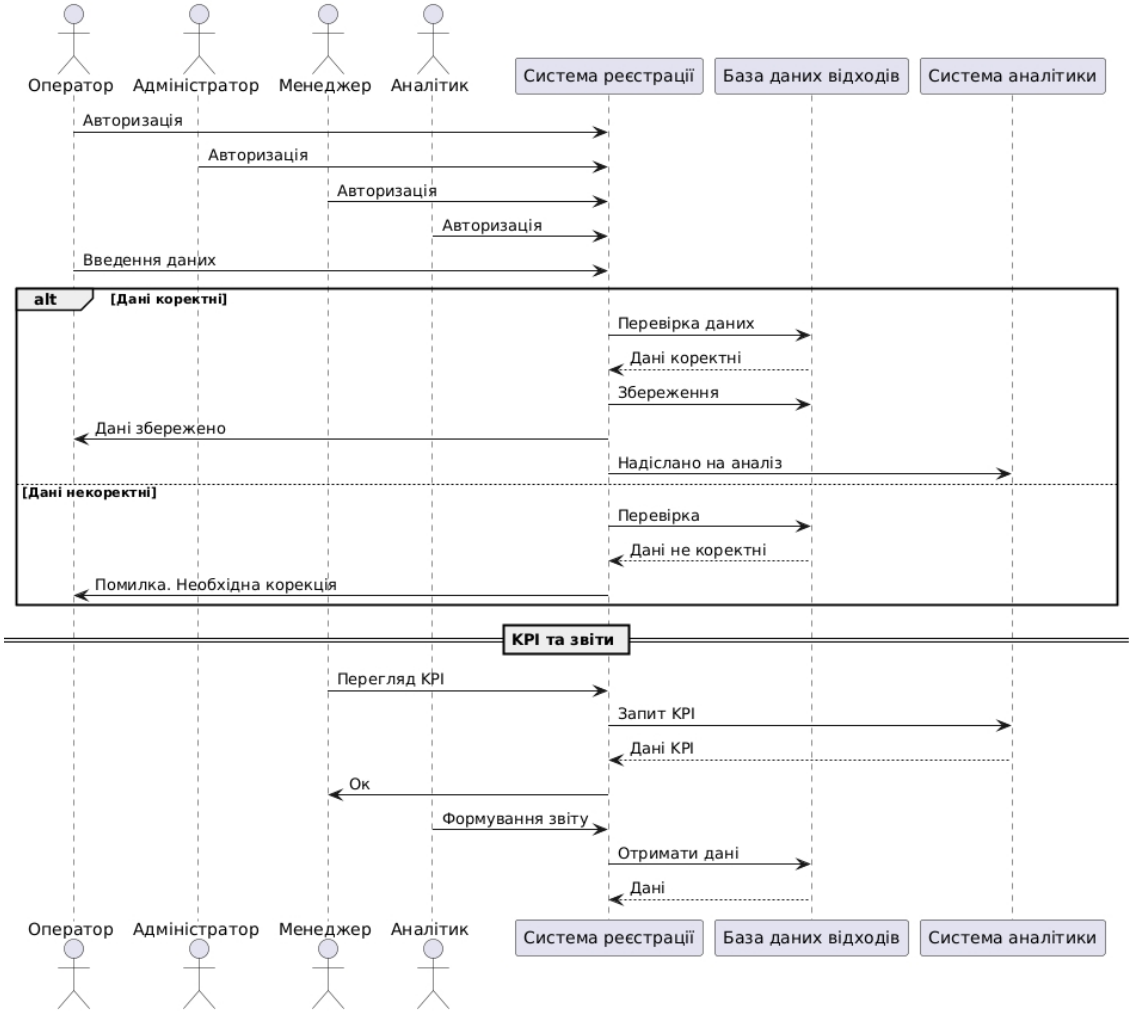

Рис. 2 Діаграма послідовності

Дана діаграма послідовності відображає типову взаємодію користувача із системою: після введення даних через інтерфейс інформація передається на обробку, проходить перевірку, а далі — використовується для формування результату, як-от оновлення в базі даних чи створення звіту, який повертається користувачеві як завершення процесу.

Моделювання предметної області дозволило визначити ключові компоненти системи, їх функціональні зв’язки та порядок взаємодії. Діаграми варіантів використання та послідовності забезпечують чітке уявлення про логіку роботи застосунку та підтримують обґрунтоване проєктування архітектури системи на наступному етапі розробки.

1.4 Огляд існуючих програмних рішень

В умовах інтенсивного зростання урбанізації та збільшення кількості твердих побутових відходів зростає потреба в ефективних інструментах для екологічного моніторингу, прогнозування та управління процесами збору та переробки сміття. Традиційні системи часто не здатні оперативно реагувати на зміну екологічної ситуації, що спонукає до впровадження цифрових рішень нового покоління. У відповідь на ці виклики в різних країнах були запропоновані автоматизовані системи, які інтегрують сучасні технології, зокрема Інтернет речей (IoT), хмарні обчислення та інструменти обробки великих обсягів даних.

Такі системи дозволяють у режимі реального часу відстежувати заповнення сміттєвих контейнерів, оптимізувати логістику вивезення відходів, аналізувати ефективність заходів з переробки та формувати аналітичні звіти для підтримки прийняття рішень органами місцевої влади, екологічними службами та комунальними підприємствами.

Одним із прикладів таких технологічних рішень є система Bigbelly, розроблена для автоматизації процесів управління сміттєзбиральною інфраструктурою в міських середовищах (рис. 3).

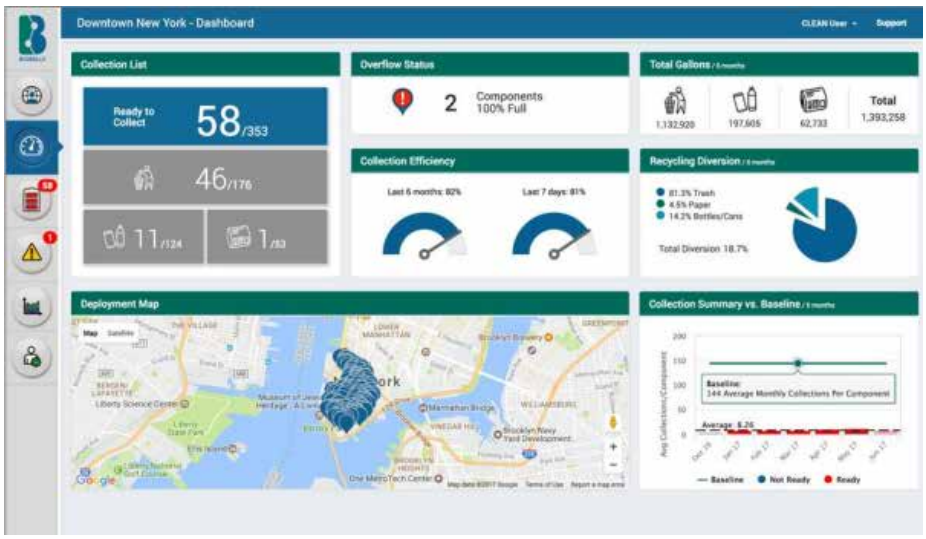

Рис. 3 Інформаційна система “Bigbelly”

Платформа поєднує інтелектуальні сміттєві контейнери, обладнані сенсорами рівня заповнення, з хмарною аналітикою. Основна мета

впровадження цієї системи — зменшення частоти вивезення відходів, зниження експлуатаційних витрат і підвищення ефективності роботи комунальних служб за рахунок динамічного маршрутизування та віддаленого моніторингу стану контейнерів [1].

Система Bigbelly ґрунтується на використанні інтелектуальних контейнерів для збору відходів, які обладнані сенсорами заповнення та живляться від сонячних панелей. Контейнери автоматично надсилають сповіщення про необхідність спорожнення, що дає змогу в реальному часі коригувати маршрути сміттєвозів і зменшити кількість виїздів. Додатково вони оснащені механізмом ущільнення, який дозволяє вміщувати більші об’єми відходів порівняно з традиційними баками, що сприяє зниженню частоти обслуговування та витрат на транспортування.

Разом із фізичною інфраструктурою Bigbelly передбачає програмну платформу для хмарного моніторингу та аналітики. Через мобільний застосунок або веб-інтерфейс оператори можуть відстежувати статус контейнерів, аналізувати маршрути та планувати збір на основі актуальних даних. Це підвищує ефективність управління відходами та підтримує екологічно орієнтоване міське планування [1].

Аналітична платформа Bigbelly надає розширені можливості для оцінювання ефективності збору відходів. Система фіксує ключові показники, зокрема рівень заповнення контейнерів, частоту спорожнення та час обслуговування, що дозволяє органам місцевого самоврядування оперативно аналізувати динаміку накопичення сміття та вдосконалювати логістику. Завдяки виявленню тенденцій утворення відходів, муніципалітети отримують змогу більш точно планувати інфраструктурні потреби та раціональніше використовувати ресурси.

Bigbelly також демонструє позитивний екологічний вплив: зменшення кількості рейсів сміттєвозів сприяє зниженню викидів парникових газів, а автономне живлення від сонячної енергії відповідає принципам енергоефективності. Інтеграція сенсорних технологій із хмарною аналітикою

робить цю систему сучасним інструментом сталого розвитку в урбанізованих середовищах, здатним адаптуватися до масштабів і потреб конкретного міста.

Одним із сучасних прикладів автоматизованого підходу до управління відходами є система Enevo, яка поєднує сенсорні пристрої з хмарною аналітикою для підвищення ефективності збору та обробки сміття (рис. 4). Рішення орієнтоване на потреби муніципалітетів, підприємств і спеціалізованих операторів, забезпечуючи оптимізацію логістики, зниження операційних витрат і мінімізацію екологічного впливу [2].

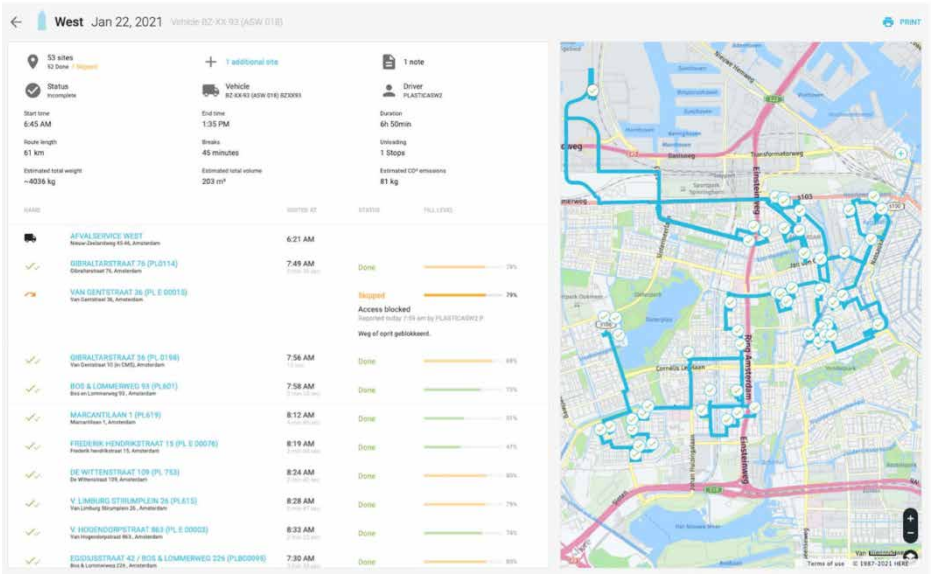

Рис. 4 Інформаційна система “Enevo”

Система Enevo базується на використанні інтелектуальних контейнерів, обладнаних сенсорами для фіксації рівня заповнення в режимі реального часу. Отримані дані передаються на хмарну платформу, що дозволяє оптимізувати графіки та маршрути збору відходів. Завдяки цьому зменшується кількість рейсів, а отже — витрати пального, експлуатаційні витрати та вплив на довкілля. Система надає користувачам зручний інтерфейс для відстеження стану контейнерів та прийняття рішень на основі актуальної інформації.

Аналітичні інструменти Enevo дозволяють проводити глибокий аналіз процесів збору сміття: від рівня заповнення контейнерів до ефективності маршрутів. Наявність історичних даних допомагає прогнозувати утворення відходів та адаптувати стратегії управління відповідно до змінних потреб. Окрім

цього, рішення Enevo добре зарекомендувало себе в публічних просторах, зокрема у парках і під час масових заходів, забезпечуючи автономну роботу контейнерів та зменшуючи ризики переповнення.

Попри значні переваги, впровадження системи потребує початкових інвестицій у сенсорну інфраструктуру. Також можливими є виклики, пов’язані з інтеграцією в наявні муніципальні платформи управління відходами. Проте завдяки поєднанню автоматизації, аналітики та екологічного підходу Enevo пропонує масштабоване рішення для міст, які прагнуть до підвищення ефективності та екологічної сталості у сфері поводження з відходами [2].

Для кращого розуміння можливостей сучасних екологічних IT-рішень було проаналізовано дві провідні системи — Bigbelly та Enevo, що спеціалізуються на автоматизації збору відходів, використовуючи сенсорні технології та хмарну інфраструктуру. Обидва рішення мають схожі цілі, проте відрізняються архітектурними підходами, рівнем автономності та можливостями інтеграції, що відображено в таблиці 1.5.

Таблиця 1.5

Порівняння систем Bigbelly та Enevo

№	Критерій	Bigbelly	Enevo
1	2	3	4
1	Тип рішення	Комплексна система з фізичними контейнерами та ПЗ	Платформа з сенсорами, що інтегруються в стандартні контейнери
2	Живлення	Сонячна енергія (контейнери автономні)	Може використовувати акумулятори або живлення від мережі
4	Механізм ущільнення сміття	Так, вбудований у контейнери	Відсутній
5	Сенсори рівня заповнення	Присутні, вбудовані	Присутні, встановлюються окремо
6	Хмарна аналітика	Так, вбудована у власну платформу	Так, із широкими можливостями аналізу та прогнозування
7	Гнучкість інтеграції	Обмежена — система є повністю закритою	Висока — сенсори можна інтегрувати в будь-які контейнери

Таблиця 1.5 (продовження)

1	2	3	4
8	Цільове середовище використання	Міські зони з інвестиціями в інфраструктуру	Громадські місця, парки, великі заходи, типова міська забудова
9	Мобільний/веб-доступ	Так	Так
10	Екологічний ефект	Зменшення викидів, використання ВДЕ, зменшення кількості рейсів	Оптимізація маршрутів, скорочення споживання палива, зменшення заторів
11	Вартість впровадження	Висока (через вартість обладнання з ущільнювачем і сонячною панеллю)	Середня (сенсори і ПЗ, без зміни контейнерної інфраструктури)
12	Складність обслуговування	Відносно низька, але залежить від стану обладнання	Потребує періодичного технічного обслуговування сенсорів

Проаналізовані програмно-апаратні рішення демонструють високий рівень автоматизації та орієнтацію на сталий розвиток у сфері управління відходами. Система Bigbelly показує ефективність в умовах стабільного фінансування та потреби в автономному функціонуванні. Натомість Enevo пропонує більш адаптивний підхід, придатний для поетапного впровадження в умовах уже наявної інфраструктури.

Разом з тим, жодна з розглянутих систем повною мірою не охоплює специфічні потреби українських міських реалій, зокрема необхідність локалізації, інтеграції з відкритими екологічними реєстрами, підтримки адаптивної звітності та гнучкої взаємодії з користувачами різного рівня. Це обґрунтовує доцільність розробки власного програмного рішення, адаптованого до потреб міського середовища Києва, з урахуванням функцій візуалізації, реального моніторингу та підтримки прийняття рішень на основі даних.

1.5 Постановка завдання

Метою розробки програмної системи є створення інструменту для ефективного контролю та моніторингу процесів поводження з відходами в межах окремого регіону або об'єкта. Система має забезпечувати відображення актуальних екологічних показників, зокрема обсягів відходів, коефіцієнтів переробки та рівнів забруднення, що дозволить уповноваженим користувачам оперативно реагувати на зміни та приймати обґрунтовані управлінські рішення.

Очікується, що система буде корисною для муніципальних служб, екологічних агентств, підприємств і операторів з утилізації, надаючи доступ до таких даних:

- кількість утворених відходів у розрізі регіонів;
- показники ефективності переробки;
- рівні забруднення повітря, ґрунтів або води, зафіксовані датчиками;
- стан інфраструктури з управління відходами (наприклад, рівень заповнення контейнерів або перевищення порогових значень).

До складу програмного забезпечення входитиме база даних, що отримуватиме дані з сенсорів і забезпечуватиме централізований доступ до екологічної інформації в реальному часі. Це дозволить користувачам аналізувати динаміку, формувати звіти та планувати заходи з оптимізації утилізації.

Окремим елементом функціональності стане система сповіщень, яка надсилатиме повідомлення у разі виявлення критичних значень — зокрема, перевищення рівня забруднення або зниження показників переробки. Такі повідомлення слугуватимуть основою для оперативного втручання та підвищення ефективності управління екологічними ризиками.

Інтерфейс взаємодії з користувачем буде побудований у вигляді меню з інтуїтивно зрозумілими діалоговими елементами, що забезпечать швидкий доступ до основних функцій системи. Такий підхід сприятиме зручності використання та зменшенню часу на навчання персоналу.

2 ПРОЄКТУВАННЯ СИСТЕМИ АНАЛІЗУ ТА КОНТРОЛЮ ПЕРЕРОБКИ ВІДХОДІВ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування бази даних є ключовим етапом створення інформаційної системи, що забезпечує надійне зберігання, обробку та доступ до даних. Для побудови структури даних системи аналізу та моніторингу переробки відходів використано ER-модель (модель «сутність-зв’язок»), яка дозволяє формалізувати основні об'єкти предметної області та взаємозв’язки між ними.

ER-модель була побудована на основі функціональних вимог до системи, зокрема таких, як облік відходів за категоріями, збір показників забруднення в окремих регіонах, реєстрація користувачів за ролями, формування звітів тощо [7]. Усі сутності мають унікальні ідентифікатори та логічно пов’язані між собою через типові відношення: «один до багатьох» або «багато до багатьох».

Логічна модель даних для системи контролю та переробки відходів відображає структуру інформації без прив’язки до фізичної реалізації, забезпечуючи узгоджене представлення взаємопов’язаних сутностей [12]. Основу моделі становлять об’єкти: Користувачі, Записи про відходи, Звіти та Ключові показники ефективності (KPI). Кожна з ролей користувачів (оператор, аналітик, адміністратор, менеджер) має чітко визначені повноваження щодо введення, перегляду, аналізу чи звітування. Дані про відходи — тип, обсяг, дата, місце збору — формують базу для подальшого аналітичного оцінювання.

Звіти формуються на основі оброблених даних та KPI, які дозволяють оцінити ефективність переробки, екологічний вплив і динаміку змін. Зв’язки між сутностями побудовані так, що один користувач може створювати кілька записів або звітів, а один запис впливає на низку показників, які використовуються у звітах для прийняття управлінських рішень. Така структура гарантує логічну цілісність і адаптивність системи до вимог моніторингу в динамічному середовищі.

В таблиці 2.1 представлено сутності бази даних, їх ключові атрибути та взаємозв’язки.

Таблиця 2.1

Основні сутності та зв’язки бази даних

№	Сутність	Ключові атрибути	Зв’язки з іншими сутностями
1	Користувач	ID, ім’я, прізвище, роль, логін, пароль, електронна пошта	має доступ до звітів, створює записи
2	Регіон	ID, назва, тип зони, геокоординати	пов’язаний із датчиками, показниками
3	Відходи	ID, тип, обсяг, дата, джерело, статус переробки	належать до регіону, зв’язуються з користувачем
4	Показник	ID, тип показника, значення, одиниця, дата вимірювання	пов’язаний із регіоном та датчиком
5	Датчик	ID, модель, розташування, тип вимірювання	фіксує показники, закріплений за регіоном
6	Звіт	ID, дата створення, тип звіту, автор, період	створюється користувачем, містить відходи/показн.

На основі вказаних сутностей було побудовано ER-діаграму, яка відображає структуру логічної моделі даних (рис. 5).

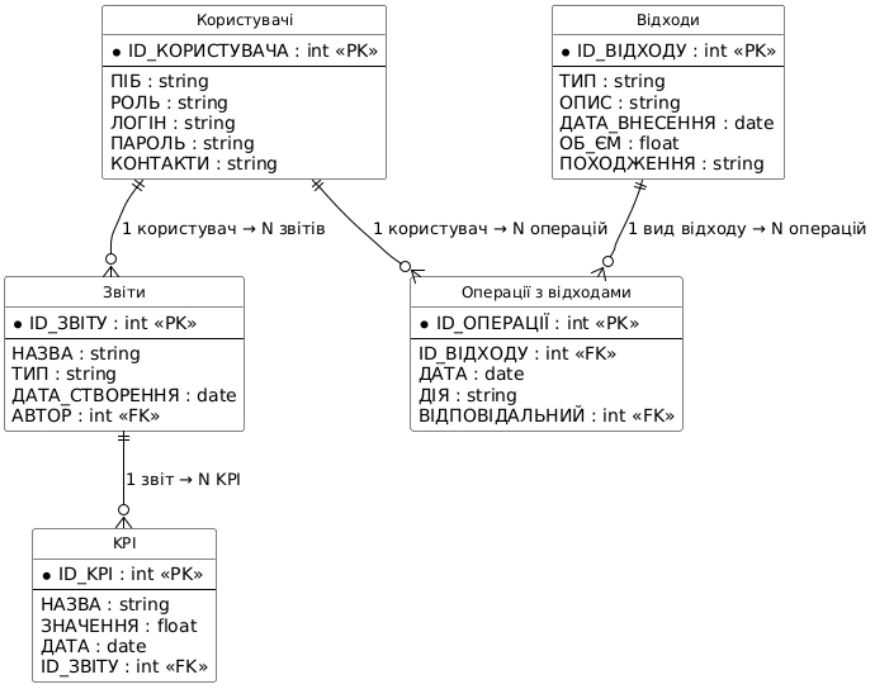

Рис. 5 Логічна ER-діаграма інформаційної системи

На діаграмі відображено основні сутності, зв’язки між ними, типи атрибутів і кардинальності відношень. Наприклад, один користувач може створювати багато звітів; один регіон може містити кілька датчиків і пов’язаний із багатьма записами про відходи; кожен звіт може містити відомості про кілька типів відходів і пов’язані з ними показники.

Нормалізація бази даних у розробленій системі спрямована на забезпечення цілісності, усунення надмірності та логічну впорядкованість структури. Всі таблиці відповідають першій нормальній формі (1НФ), оскільки містять лише атомарні значення без повторюваних груп. У другій нормальній формі (2НФ) забезпечено залежність кожного неключового атрибута від усього первинного ключа. Третя нормальна форма (3НФ) усуває транзитивні залежності: усі неключові поля залежать лише від первинного ключа. Наприклад, у таблицях користувачів і KPI атрибути повністю пов’язані зі своїми унікальними ідентифікаторами [24].

Зв’язки між таблицями реалізовані за допомогою зовнішніх ключів, що відображають відношення «один до багатьох»: один користувач може створювати кілька звітів, а одна операція — бути пов’язаною з кількома показниками. Такий підхід дозволяє зберегти цілісність даних і водночас забезпечує масштабованість системи. Структура бази даних відповідає вимогам 3НФ і придатна до подальшого розширення без ризику дублювання чи втрати узгодженості.

Розроблена ER-модель дозволяє структурувати дані відповідно до вимог системи, забезпечити цілісність інформації та ефективне її використання в процесі моніторингу та аналізу екологічних показників. Побудована модель лягає в основу фізичного проєктування бази даних і подальшої реалізації прикладного програмного забезпечення.

2.2 Діаграма класів та кооперацій

Діаграма класів є одним із базових засобів UML-моделювання, який відображає статичну структуру програмної системи. Вона дозволяє

формалізувати основні класи, їх атрибути, методи та взаємозв’язки, що утворюють основу архітектури застосунку. Такий тип моделі застосовується на етапі проєктування для визначення логічної організації компонентів системи та взаємодії між ними до початку реалізації програмного коду [11, 23].

Діаграма класів у складі UML-моделей відображає статичну структуру об'єктно-орієнтованої системи та є основою для логічного проєктування програмного забезпечення. У контексті системи моніторингу та контролю за відходами вона дозволяє описати основні об’єкти предметної області, їх атрибути, методи та взаємозв’язки. Така модель є необхідною для забезпечення цілісного уявлення про компоненти системи та принципи їх взаємодії до початку реалізації.

Кожен клас на діаграмі відповідає окремому елементу системи: географічному регіону, сенсору, запису даних, дії з переробки, звіту, користувачу тощо. Зв’язки між класами позначають асоціації, які відображають відношення «один до одного» або «один до багатьох». Класи містять як атрибути (наприклад, назва регіону, тип сенсора, обсяг відходів), так і методи (розрахунок, фільтрація, генерація звіту). Узагальнену характеристику ключових класів представлено в таблиці 2.2.

Таблиця 2.2

Основні класи системи та їх призначення

№	Клас	Атрибути (приклади)	Призначення
1	2	3	4
1	Region	Назва, координати, екологічний індекс	Представляє адміністративну зону, пов’язану з датчиками
2	Sensor	ID, тип, дата встановлення, значення	Вимірює показники забруднення в певному регіоні
3	WasteData	Тип відходів, обсяг, дата, джерело	Зберігає інформацію про зібрані або перероблені відходи
4	RecyclingAction	Тип дії, дата, ефективність	Відображає виконані заходи з переробки або зменшення забруднення

Таблиця 2.2 (продовження)

1	2	3	4
5	Report	Назва, період, створено користувачем	Формалізує аналітичні звіти на основі даних
6	User	Ім’я, роль, логін, email	Представляє користувача системи з відповідними повноваженнями
7	Notification	Повідомлення, дата, рівень важливості	Генерується системою у разі перевищення порогових значень

Для візуалізації структури об’єктів, описаних у таблиці 2.2, побудовано діаграму класів, яка відображає основні сутності системи, їх атрибути, методи та зв’язки. Вона надає цілісне уявлення про логіку побудови програмного забезпечення, а також про те, як взаємодіють окремі компоненти в межах архітектури системи.

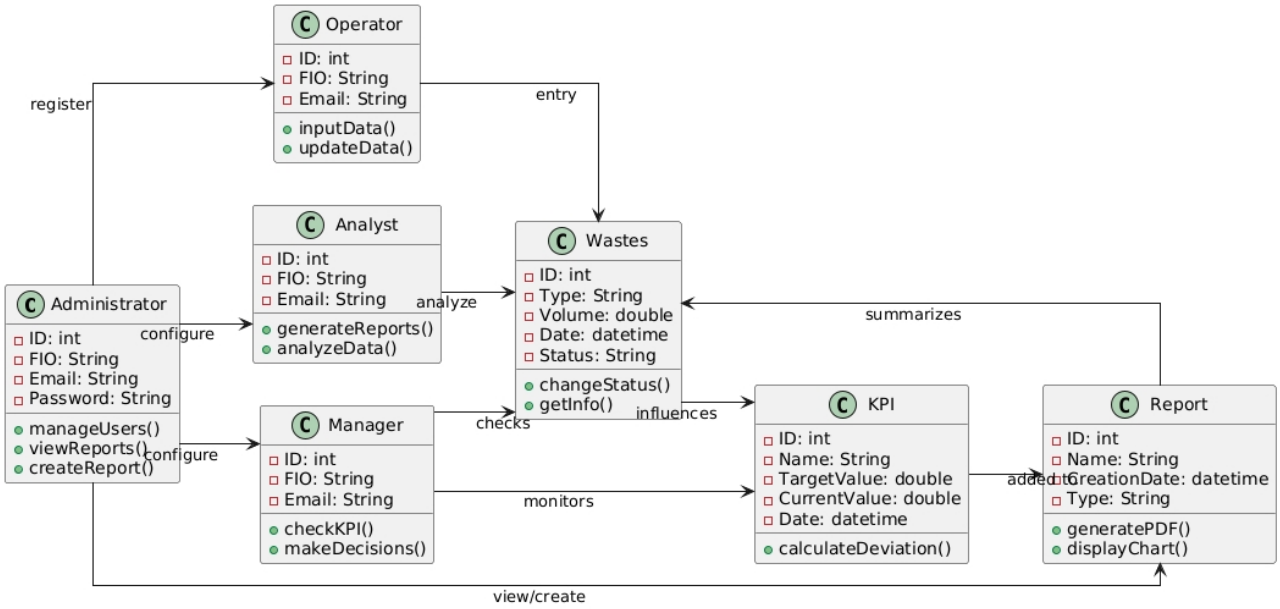

Рис. 6 Діаграма класів системи моніторингу та переробки відходів

У системі контролю та переробки відходів взаємодія між користувачами й об’єктами предметної області реалізується через чітке розмежування ролей. Адміністратор створює облікові записи операторів, менеджерів і аналітиків, визначаючи рівень доступу до функцій системи. Оператори вводять дані щодо

типу, обсягу та дати утворення відходів, які безпосередньо впливають на обчислення ключових показників ефективності (KPI). Менеджери аналізують ці показники з метою контролю екологічної результативності.

Аналітики, працюючи з накопиченими даними, формують звіти різного рівня деталізації. Один набір відходів може бути представлений у кількох звітах, або поки не входити до жодного. Аналогічно, KPI можуть дублюватися у звітах різних типів — наприклад, щомісячному та квартальному — для проведення порівняльного аналізу. Такий підхід забезпечує цілісну оцінку стану системи управління відходами.

Загалом, така модель відображає гнучку та логічно вибудовану структуру зв’язків між користувачами й даними в системі, забезпечуючи ефективний контроль, облік і аналіз у сфері поводження з відходами.

Після визначення структурної організації системи у вигляді діаграми класів доцільно проаналізувати динамічні аспекти взаємодії між об'єктами на основі типових сценаріїв. Для цього використовується діаграма кооперацій, яка деталізує, як елементи, описані на діаграмі класів, обмінюються повідомленнями під час виконання окремих функцій.

Діаграма кооперацій відображає взаємодію між користувачами системи та об’єктами предметної області в процесі збору, обробки та аналізу даних про відходи (рис. 7). Взаємозв’язки між об’єктами супроводжуються нумерованими повідомленнями, що описують порядок дій. Наприклад, адміністратор створює облікові записи операторів, які вводять інформацію про відходи. Менеджер переглядає ключові показники ефективності, а аналітик — формує звіти. Ця модель демонструє, як об'єкти взаємодіють у межах конкретного процесу.

Дана діаграма дозволяє простежити логіку виконання ключових операцій системи та забезпечує підґрунтя для перевірки узгодженості функціональних залежностей. Застосування кооперaційної моделі сприяє підвищенню прозорості механізмів комунікації між компонентами та підтверджує коректність закладеної архітектури системи.

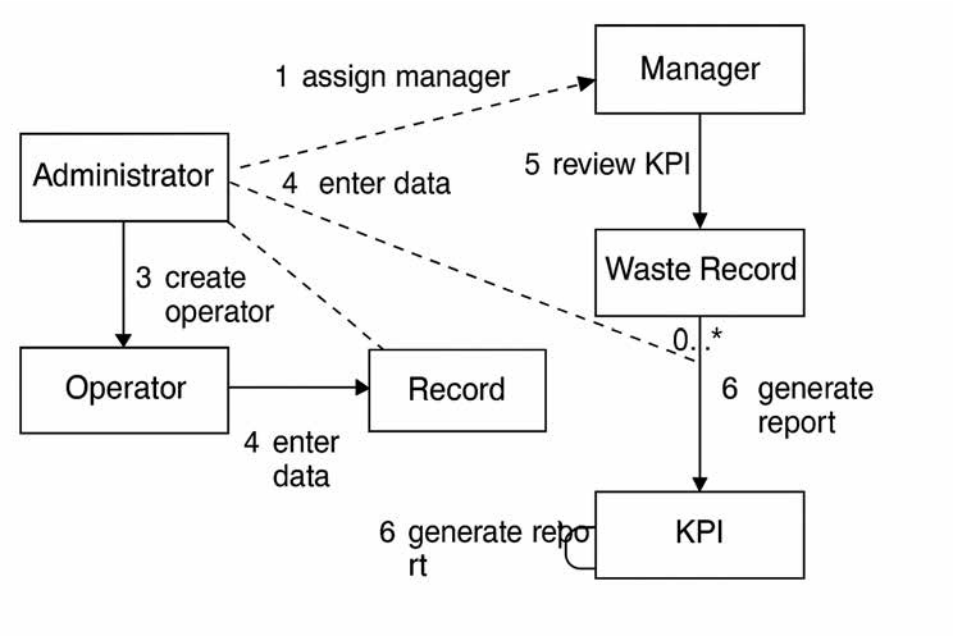

Рис. 7 UML-діаграма кооперацій для сценарію роботи з даними про відходи

Таким чином, побудова діаграми класів та кооперацій дозволила сформувати цілісне уявлення про структуру та взаємодію об’єктів у межах програмної системи. Визначення ключових класів, їх атрибутів, методів і зв’язків забезпечує логічну основу для реалізації функціональності, а кооперaційна діаграма ілюструє динаміку обміну повідомленнями між об’єктами під час виконання типових сценаріїв. Такий підхід сприяє підвищенню узгодженості архітектури системи та забезпечує ефективність її подальшої реалізації.

2.3 Діаграма пакетів

Діаграма пакетів відображає модульну структуру програмного забезпечення, групуючи функціональні компоненти системи у логічно відокремлені пакети. Такий підхід дозволяє впорядкувати архітектуру проєкту, розмежувати відповідальність між окремими частинами системи та забезпечити кращу керованість програмного коду. Кожен пакет відповідає за реалізацію певного підмножини функцій, що сприяє масштабованості, повторному використанню та спрощенню технічного супроводу [16].

Архітектура системи побудована на основі пакетного поділу, що забезпечує логічну організацію коду та спрощує його підтримку. Пакет

інтерфейсу користувача відповідає за взаємодію з користувачем, включаючи введення даних, перегляд KPI та генерацію звітів. Завдяки адаптивному дизайну забезпечується зручність навігації та доступ до візуалізованої інформації.

Пакет бізнес-логіки реалізує основні функції: обробку даних про відходи, розрахунок показників, управління правами доступу та формування звітності. Комунікацію з базою даних забезпечує окремий пакет доступу до даних, який виконує всі операції з читання й запису. Захист інформації здійснюється через пакет безпеки, що включає механізми автентифікації, авторизації та шифрування, дотримуючись принципів ролей та обмеженого доступу.

Для наочного відображення модульної архітектури системи було побудовано діаграму пакетів, що відображає логічні залежності між основними складовими. На рисунку 8 показано структуру системи, розподілену за функціональними пакетами, кожен з яких виконує визначену роль у загальній архітектурі застосунку.

Рис. 8 Діаграма пакетів системи контролю та переробки відходів

Діаграма пакетів ілюструє архітектуру системи, побудовану за трирівневою моделлю: презентаційний рівень, рівень логіки та рівень доступу до даних. Презентаційний рівень охоплює інтерфейси для різних категорій

користувачів (адміністратор, оператор, менеджер, аналітик) та забезпечує введення даних, перегляд звітів і контроль за процесами.

На рівні логіки реалізовано модулі керування користувачами, обробки даних про відходи, розрахунку KPI та генерації звітності. Рівень доступу до даних відповідає за взаємодію з базою, захист інформації та інтеграцію із зовнішніми сервісами. Така структура гарантує модульність, безпеку й ефективність роботи системи.

В таблиці 2.3 подано стислий опис призначення кожного пакета, що відображений на діаграмі.

Таблиця 2.3

Призначення основних пакетів системи

№	Назва пакета	Призначення
1	UI (User Interface)	Реалізує взаємодію з користувачем: введення даних, перегляд KPI, звіти
2	BusinessLogic	Обробляє бізнес-операції: облік відходів, обчислення показників, контроль доступу
3	DataAccess	Забезпечує доступ до бази даних: зчитування, збереження, оновлення записів
4	Security	Містить функції автентифікації, авторизації та контролю доступу
5	Reporting	Генерує звіти, агрегує дані, забезпечує аналітичну обробку

Подана таблиця узагальнює функціональне призначення основних пакетів системи, що дозволяє чітко розмежувати їхню відповідальність у межах архітектури. Такий поділ сприяє зручності супроводу, гнучкості оновлення та масштабуванню програмного забезпечення без порушення його цілісності.

Таким чином, діаграма пакетів відображає логічну структуру системи та її модульну організацію відповідно до принципів трирівневої архітектури. Вона дозволяє впорядкувати компоненти за функціональними ознаками, забезпечуючи гнучке розширення системи, підвищену безпеку та ефективну взаємодію між рівнями.

2.4 Діаграма компонентів

Діаграма компонентів у складі UML-моделей ілюструє фізичну структуру програмної системи, відображаючи її основні складові та зв’язки між ними. Вона дозволяє окреслити, як логічні модулі (інтерфейс, логіка, служби) реалізуються у вигляді окремих компонентів, які взаємодіють між собою через інтерфейси або API.

У системі візуалізації даних про контроль і переробку відходів діаграма компонентів відображає три основні рівні: інтерфейс користувача, контролер і бізнес-логіку, а також компоненти доступу до даних. Наприклад, UI-модуль відповідає за виведення дашбордів та форм, модуль обробки запитів керує логікою запитів і взаємодією з сервісами KPI і звітності, а база даних зберігає всю інформацію про користувачів, відходи та показники.

Для кращого розуміння фізичної структури системи розроблено діаграму компонентів, яка демонструє взаємозв’язки між основними модулями програмного забезпечення (рис. 9). На ній зображено, як користувацький інтерфейс, бізнес-логіка та рівень збереження даних взаємодіють між собою через чітко визначені інтерфейси.

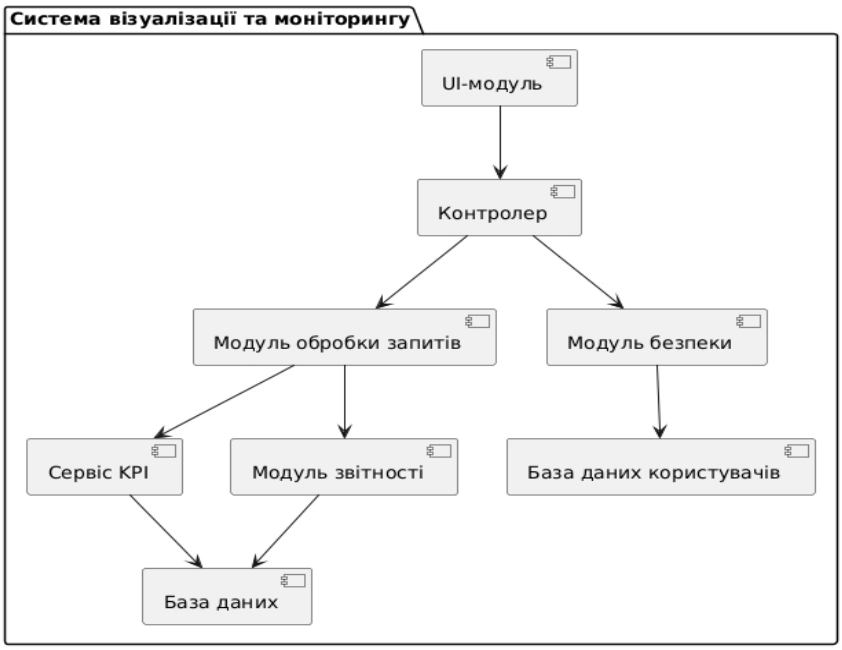

Рис. 9 Діаграма компонентів системи контролю та візуалізації даних про відходи

З метою деталізації елементів, представлених на діаграмі, у таблиці 2.4 подано стислий опис призначення кожного компонента системи, що забезпечує повніше розуміння їх функціонального навантаження.

Таблиця 2.4

Опис компонентів системи

№	Компонент	Призначення
1	UI-модуль	Відображення форм введення, панелей статистики, взаємодія з користувачем
2	Контролер	Приймає запити від UI, спрямовує їх до відповідних сервісів
3	Модуль обробки запитів	Виконує логіку обробки даних, фільтрацію, агрегацію
4	Сервіс KPI	Розраховує ключові показники ефективності на основі наявних даних
5	Модуль звітності	Формує звіти за параметрами часу, регіону, типу відходів
6	Модуль безпеки	Реалізує автентифікацію, авторизацію, шифрування даних
7	База даних	Зберігає інформацію про користувачів, операції, звіти, KPI
8	БД користувачів	Містить облікові записи та ролі доступу

У діаграмі та таблиці чітко відображено розподіл функціональних обов’язків між модулями системи. Взаємодія між інтерфейсом користувача, контролером, сервісами обробки даних та базою даних реалізується через визначені інтерфейси, що забезпечує надійність, масштабованість і простоту супроводу програмного забезпечення.

Таким чином, діаграма компонентів дозволяє представити логічну структуру системи в реальних програмних модулях, що полегшує процес реалізації, тестування та підтримки. Розподіл функціональності між окремими компонентами сприяє масштабованості, підвищує безпеку та забезпечує незалежність модулів у процесі розробки.

2.5 Діаграма активності

Діаграма активності (activity diagram) є інструментом динамічного моделювання в UML, який дозволяє відобразити послідовність дій у межах певного процесу. На відміну від діаграм, що зосереджені на взаємодії об’єктів, цей тип моделі описує загальний хід виконання операцій — від початкового стану до завершення, з урахуванням умов, розгалужень та паралельних дій. Діаграма активності особливо корисна для формалізації робочих процесів системи та виявлення критичних точок прийняття рішень або оптимізації [9].

Діаграма активності починається з початкового вузла, який позначає запуск процесу, та демонструє послідовність дій, що виконуються користувачем або системою. Кожна дія представлена окремим вузлом, а стрілки відображають напрямок потоку. Умовні переходи моделюються через логічні розгалуження, що оформлюються ромбоподібними символами, а завершення процесу позначається кінцевим вузлом.

Для відображення відповідальності учасників процесу діаграма може містити доріжки (swimlanes), які розподіляють дії між користувачами або підсистемами. У разі паралельного виконання операцій використовуються вузли розгалуження та синхронізації, що дозволяє моделювати виконання дій одночасно з подальшим їх об’єднанням. Побудована діаграма активності зображена на рис. 10.

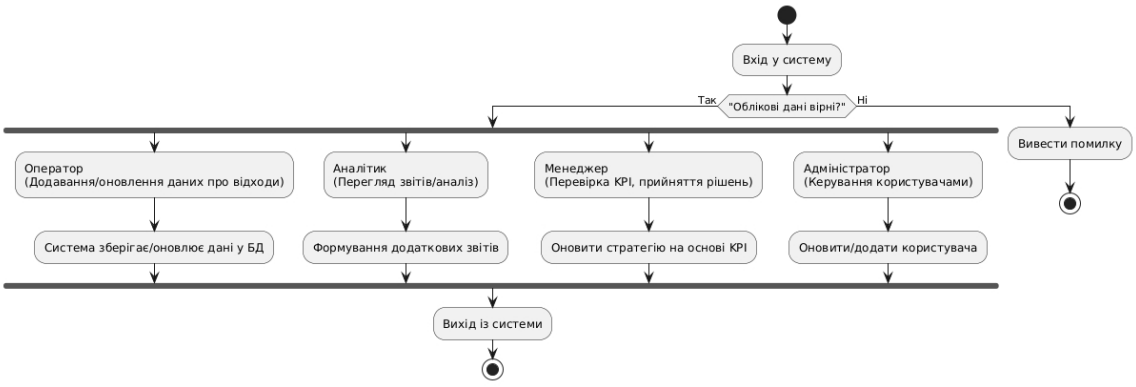

Рис. 10 Діаграма активності

На діаграмі активності представлено загальний сценарій взаємодії користувача із системою — від моменту авторизації до завершення роботи. Після введення облікових даних система перевіряє їх коректність. У разі помилки виводиться повідомлення, і користувач може повторити спробу; якщо дані правильні — процес переходить до основних дій.

У межах системи оператор додає або оновлює записи про відходи, аналітик переглядає дані та формує запити, менеджер аналізує KPI і приймає управлінські рішення, а адміністратор налаштовує доступ користувачів. Інформація зберігається у базі даних, на основі якої формуються звіти. Завершення роботи користувача з системою позначає кінець активного циклу.

Щоб деталізувати етапи виконання основних операцій користувачів та системи, у таблиці 2.5 наведено ключові дії, які відбуваються в межах одного циклу взаємодії. Це дозволяє чітко простежити роль кожного учасника в процесі обробки та аналізу даних.

Таблиця 2.5

Ключові дії у процесі взаємодії користувача із системою

№	Учасник	Дія	Умова / Результат
1	Користувач	Вхід у систему (авторизація)	Вірні дані → доступ надано; помилка → повідомлення про відмову
2	Оператор	Додавання або оновлення даних про відходи	Дані збережено в базі
3	Аналітик	Перегляд запитів, аналіз інформації	Формуються аналітичні звіти
4	Менеджер	Оцінка ключових показників ефективності (KPI)	Прийняття управлінських рішень
5	Адміністратор	Керування доступом користувачів (додавання, редагування профілів)	Зміни збережено в системі
6	Система	Збереження, оновлення та обробка даних у базі	Дані актуалізовано
7	Система	Формування звітів на основі показників KPI	Дані передано менеджеру або аналітику
8	Користувач	Вихід із системи	Завершення сеансу роботи

У таблиці узагальнено основні функції, які виконують користувачі відповідно до своїх ролей, а також внутрішні дії, що виконує система автоматично. Такий опис полегшує інтерпретацію діаграми та дає змогу структурувати функціональність за логічними етапами.

Діаграма активності забезпечує наочне відображення динаміки процесів у межах інформаційної системи. Вона дозволяє визначити послідовність дій, ролі користувачів і логіку обробки запитів. Завдяки структурованому підходу можна виявити ключові точки прийняття рішень і синхронізації дій, що сприяє вдосконаленню бізнес-логіки системи та її подальшій оптимізації.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОНІТОРИНГУ ВІДХОДІВ

3.1 Архітектура та система управління інформаційною базою

Архітектура програмного забезпечення системи візуалізації даних про переробку відходів побудована за багаторівневим принципом, що забезпечує модульність, масштабованість та безпечну обробку даних. Основні рівні — представлення, бізнес-логіка, доступ до даних, інтеграція та безпека — взаємодіють між собою через чітко визначені інтерфейси.

Рівень представлення відповідає за інтерфейс користувача, реалізований засобами HTML, CSS, JavaScript і Symfony. Він забезпечує інтуїтивну навігацію, перегляд KPI, формування звітів та внесення даних. На рівні бізнес-логіки реалізовано основні функції: обробка даних, розрахунок показників ефективності, автентифікація та контроль доступу. Цей рівень виступає посередником між інтерфейсом і базою даних.

Рівень даних побудовано на базі СКБД MySQL, де зберігається структурована інформація про користувачів, відходи, KPI, звіти [15]. Забезпечено цілісність даних, підтримку зв’язків між таблицями та швидкий доступ до інформації. Рівень інтеграції відповідає за взаємодію з зовнішніми API — наприклад, службами геолокації або експортом даних до аналітичних платформ. Захист даних забезпечується інтегрованим рівнем безпеки, який включає автентифікацію, шифрування, HTTPS-з’єднання та обмеження доступу на основі ролей користувачів.

Архітектура програмного забезпечення побудована за багатошаровою моделлю, що забезпечує поділ функціональності між окремими рівнями [5]. Такий підхід підвищує гнучкість системи, полегшує її супровід і масштабування. Основні рівні архітектури: представлення (інтерфейс користувача), бізнес-логіка, доступ до даних, інтеграція із зовнішніми сервісами та захист інформації.

Рівні взаємодіють між собою через чітко визначені інтерфейси, що дозволяє забезпечити надійний обмін даними та безпеку обробки запитів. Загальна структура архітектури системи зображена на рисунку 11.

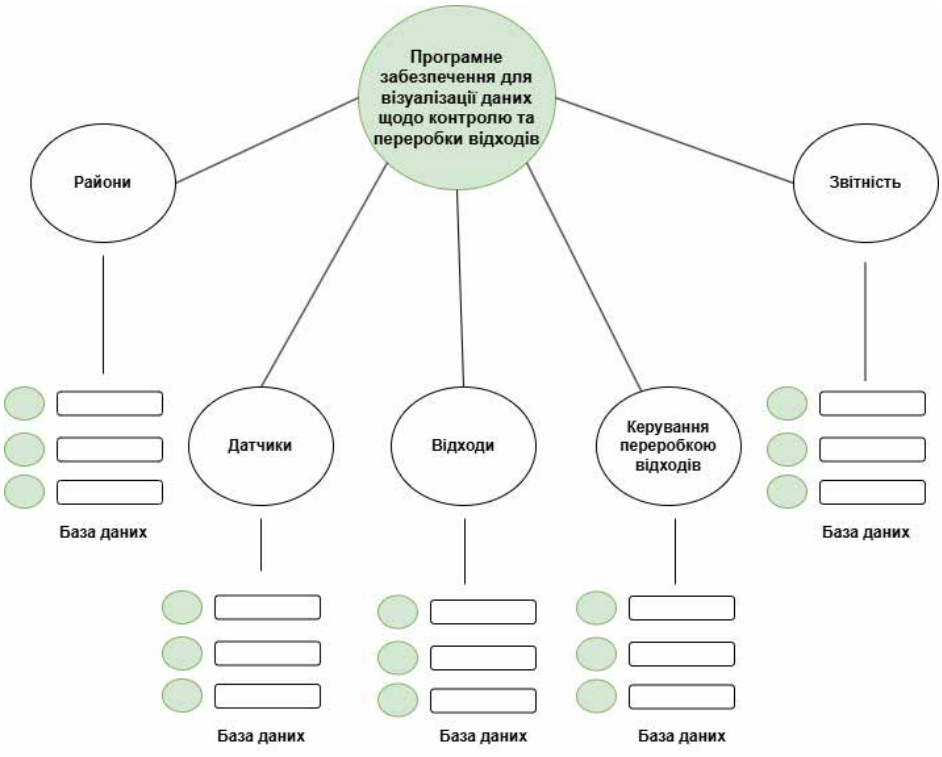

Рис. 11 Багатошарова архітектура

Одним із ключових компонентів розробленої системи є інформаційна база, яка забезпечує зберігання, обробку та цілісність даних про відходи, ключові показники ефективності (KPI), звіти та облікові записи користувачів. Для керування цією базою даних використано реляційну систему керування базами даних (СКБД) MySQL, яка добре інтегрується з обраним фреймворком Symfony, підтримує транзакції, індекси, зовнішні ключі та масштабування.

База даних побудована на основі попередньо розробленої логічної ER-моделі, нормалізованої до третьої нормальної форми, що дозволяє уникнути надмірності даних і забезпечити узгодженість. Основні таблиці бази даних відображають сутності предметної області, їх атрибути та зв’язки між ними (табл.. 3.1).

Таблиця 3.1

Основні таблиці бази даних та їх призначення

№	Назва таблиці	Призначення
1	users	Зберігає дані про користувачів, їх ролі, облікові дані
2	regions	Містить інформацію про географічні регіони, з якими пов’язано датчики
3	sensors	Дані про сенсори, їх розташування та типи вимірювань
4	wastes	Зберігає відомості про типи відходів, їх обсяг, дату, місце збору
5	kpi	Містить показники ефективності, їх цільові та поточні значення
6	reports	Звіти, згенеровані аналітиками на основі KPI і даних про відходи
7	notifications	Сповіщення користувачам про критичні значення або події

Для забезпечення узгодженості даних між таблицями в базі реалізовано зовнішні ключі, які підтримують відношення «один до багатьох» (наприклад, один користувач може створити кілька звітів або внести кілька записів про відходи).

База даних функціонує на сервері, що підтримує MySQL 8.0 і налаштований для роботи з Symfony через ORM-компонент Doctrine. Завдяки чіткому поділу таблиць, використанню індексів, каскадному оновленню та видаленню записів забезпечується цілісність, швидкість доступу та масштабованість.

Система управління інформаційною базою реалізована на основі СКБД MySQL із дотриманням принципів реляційного проєктування та нормалізації. Структура бази повністю відповідає предметній області, дозволяє ефективно зберігати, аналізувати та представляти дані, що є необхідною умовою для надійної роботи інформаційної системи.

3.2 Розробка інформаційної бази

Для забезпечення ефективного функціонування програмного забезпечення з візуалізації даних, пов’язаних із контролем і переробкою відходів, важливо

було обрати надійну систему керування базами даних (СКБД), яка дозволяє безпечно зберігати, швидко обробляти й легко інтегрувати інформацію про користувачів, види відходів, операції з ними, звіти й ключові показники ефективності [13, 25].

Оскільки система працює зі структурованими, чітко взаємопов’язаними даними, було вирішено використовувати реляційну модель зберігання. Вона забезпечує логічну організацію даних у вигляді таблиць і дозволяє гарантувати цілісність через первинні та зовнішні ключі. Серед розглянутих варіантів — PostgreSQL, SQLite та MySQL — перевагу було надано MySQL.

Цей вибір був обґрунтований кількома критеріями. По-перше, MySQL є однією з найпоширеніших СКБД з відкритим кодом, що гарантує наявність широкої спільноти, якісної документації та стабільних оновлень. По-друге, система відзначається сумісністю з обраними технологіями серверної частини, зокрема PHP та фреймворком Symfony, що значно полегшило інтеграцію бази даних у проєкт. Також MySQL показав хорошу продуктивність для проєктів середнього масштабу, забезпечуючи швидкий доступ до інформації навіть при зростанні обсягів даних.

Особливу увагу було приділено безпеці — механізми розмежування доступу, автентифікації користувачів та шифрування даних стали додатковими перевагами цієї системи. Реалізація бази даних включала проєктування структури з урахуванням нормалізації — всі таблиці приведено до третьої нормальної форми. Такий підхід дозволив уникнути дублювання, забезпечити логічну цілісність і зробити систему легкою для масштабування.

Між основними сутностями встановлено чіткі зв’язки: наприклад, кожен користувач може створювати звіти, до одного виду відходів можуть застосовуватись численні операції, а один звіт може містити кілька KPI. Всі ці зв’язки реалізовані через зовнішні ключі, що сприяє надійності логіки зберігання.

Для взаємодії з базою даних використовувалися як SQL-запити, так і ORM-інструменти Symfony, що дало змогу автоматизувати створення,

оновлення та отримання даних у зручному форматі. У результаті MySQL не лише забезпечив стабільну роботу платформи, а й став гнучкою основою для подальшого розвитку системи — з можливістю додавання нових функціональних модулів та вдосконалення візуалізації інформації.

Під час вибору системи керування базами даних (СКБД) для програмного забезпечення, орієнтованого на візуалізацію даних, пов'язаних з контролем відходів та їх переробкою, було враховано кілька важливих факторів. Зрештою, MySQL було обрано завдяки поєднанню продуктивності, сумісності та функцій безпеки, що зробило його ідеальним вибором для проєкту.

Однією з ключових причин вибору MySQL була його популярність та широке використання в галузі. Як одна з найчастіше використовуваних реляційних баз даних, MySQL має велику базу користувачів та сильну спільноту. Це полегшило пошук ресурсів, таких як документація та підтримка, та гарантувало, що система буде підкріплена надійною та добре зарекомендувала себе технологією. Широке впровадження MySQL також означає, що розробники добре знайомі з ним, що скорочує криву навчання та допомагає уникнути потенційних технічних проблем.

Сумісність MySQL з існуючим технологічним стеком була ще одним вирішальним фактором. Програмне забезпечення побудовано за допомогою PHP та фреймворку Symfony, які бездоганно інтегруються з MySQL. Таке узгодження дозволило ефективно впровадити, заощаджуючи як час, так і зусилля. Легкість, з якою MySQL можна було налаштувати для роботи з серверною системою, забезпечила безперебійне управління базою даних та мінімізувала проблеми інтеграції.

MySQL також відомий своєю продуктивністю, особливо в середовищах, де пошук даних є критично важливим. Програмне забезпечення для контролю та переробки відходів може обробляти великі обсяги даних, включаючи типи відходів, їх кількість та інформацію про переробку. Ефективна обробка запитів MySQL гарантує швидке отримання цих даних, що робить програмне

забезпечення чутливим та здатним обробляти зростаючі обсяги даних та трафіку в міру масштабування системи з часом.

Окрім своєї продуктивності, MySQL забезпечує високий рівень цілісності даних, що є критично важливою вимогою для програм, що керують конфіденційною інформацією. Завдяки повній підтримці властивостей ACID (атомарність, узгодженість, ізоляція, довговічність), MySQL гарантує надійну та безпомилкову обробку транзакцій бази даних. Це гарантує, що дані залишаються узгодженими та точними, що особливо важливо для створення звітів, пов'язаних з відходами, та ведення точних записів.

Ще однією значною перевагою MySQL є його гнучкість та простота використання. Система баз даних дозволяє ефективно організовувати та аналізувати різноманітні типи даних. Незалежно від того, чи керуєте ви структурованими даними, такими як облікові записи користувачів, операції з відходами чи ключові показники ефективності (KPI), MySQL спрощує організацію, запит та пошук інформації. Його інтуїтивно зрозумілі інструменти керування, такі як phpMyAdmin та MySQL Workbench, ще більше спрощують процес підтримки бази даних та виконання складних запитів [14].

Безпека була ще одним важливим фактором. Враховуючи, що система оброблятиме персональні дані та операційну інформацію, потужні функції безпеки MySQL були вирішальними. Він пропонує надійні механізми автентифікації користувачів, шифрування даних та контролю доступу, гарантуючи, що лише авторизовані особи мають доступ до конфіденційних даних.

База даних є центральною складовою інформаційної системи й забезпечує збереження, доступ і цілісність даних щодо користувачів, відходів, операцій з ними, звітності та ключових показників ефективності (KPI). Структуру бази даних розроблено на основі логічної моделі, що враховує потреби у зв’язності даних, нормалізацію до ЗНФ та підтримку зовнішніх ключів. Схема взаємозв’язків між основними таблицями зображена на рис. 12.

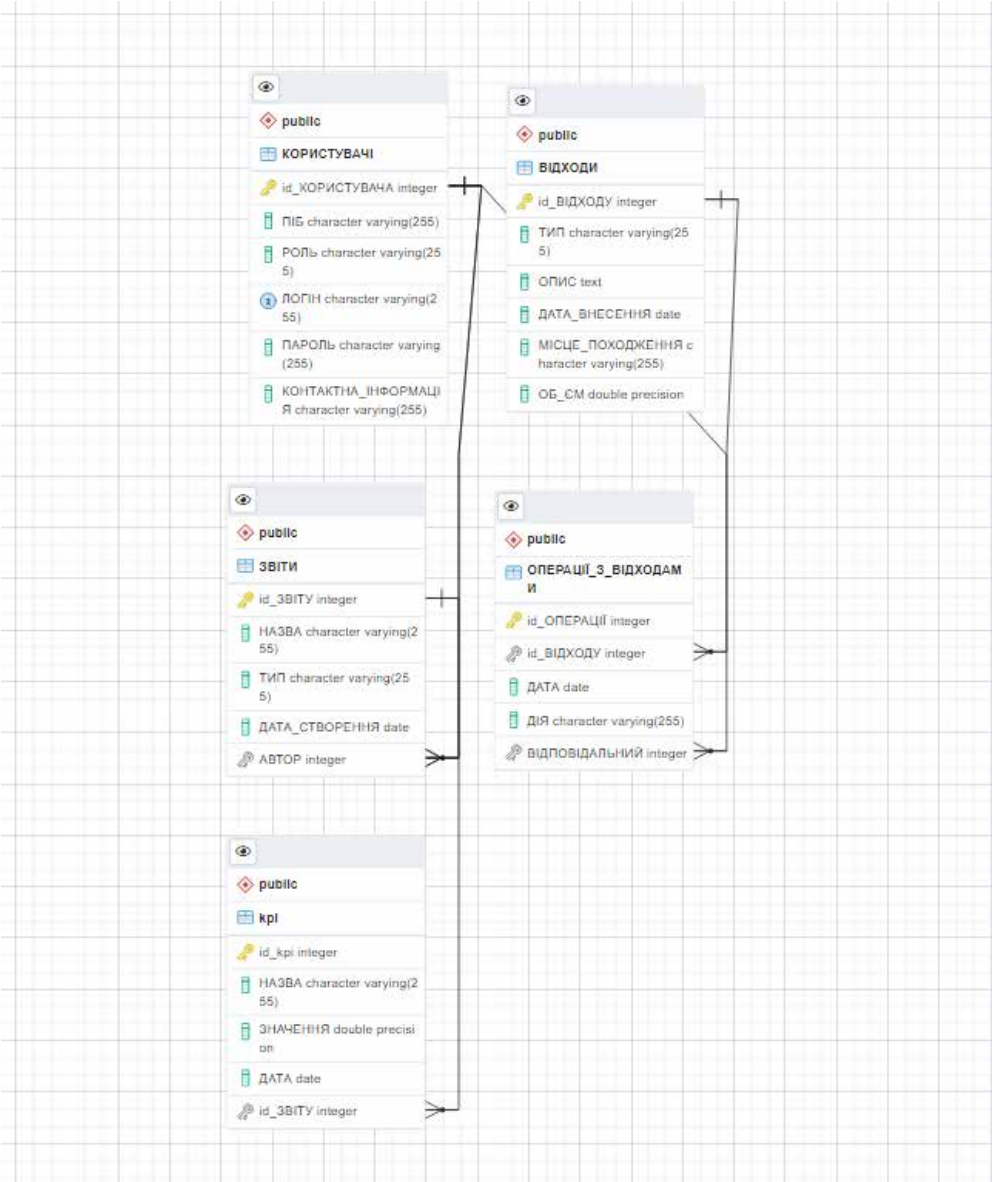

Рис. 12 База даних

На діаграмі зображена структура бази даних, яка складається з кількох таблиць та їх зв'язків між собою. З метою візуалізації логічної структури зберігання інформації у системі доцільно подати перелік таблиць бази даних із зазначенням їхніх ключових полів та функціонального призначення. Це дозволяє краще зрозуміти архітектуру даних, взаємозв’язки між сутностями та загальні принципи побудови сховища екологічної інформації. Структурований перелік основних таблиць наведено у таблиці 3.2.

Таблиця 3.2

Структура основних таблиць бази даних та опис їхніх полів

№	Назва таблиці	Поля	Опис
1.1	КОРИСТУВАЧІ	id_КОРИСТУВАЧА (integer)	унікальний ідентифікатор користувача;
1.2		ПІБ (character varying, довжина 255)	повне ім'я користувача.
1.3		РОЛЬ (character varying, довжина 25)	роль користувача (наприклад, адміністратор, оператор);
1.4		ЛОГІН (character varying, довжина 255)	логін для входу в систему;
1.5		ПАРОЛЬ (character varying, довжина 255)	пароль для автентифікації;
1.6		КОНТАКТНА_ІНФОРМАЦІЯ (character varying, довжина 255)	контактні дані (телефон, email тощо).
2.1	ВІДХОДИ	Id_ВІДХОДУ (integer)	унікальний ідентифікатор відходу;
2.2		ТИП (character varying, довжина 25)	тип відходу;
2.3		ОПИС (text)	опис відходу;
2.4		ДАТА_ВНЕСЕННЯ (date)	дата внесення запису;
2.5		МІСЦЕ_ПОХОДЖЕННЯ (character varying, довжина 255)	місце, звідки виник;
2.6		ОБ_ЄМ (double precision)	обсяг.
3.1	ОПЕРАЦІЇ_З_ВІДХОДАМИ	id_ОПЕРАЦІЇ (integer)	унікальний ідентифікатор операції;
3.2		id_ВІДХОДУ (integer)	зовнішній ключ до таблиці ВІДХОДИ;
3.3		ДАТА (date)	дата проведення операції;
3.4		ДІЯ (character varying, довжина 255)	опис дії;
3.5		ВІДПОВІДАЛЬНИЙ (integer)	зовнішній ключ до таблиці КОРИСТУВАЧІ.
4.1	ЗВІТИ	id_ЗВІТУ (integer)	унікальний ідентифікатор звіту;
4.2		НАЗВА (character varying, довжина 255)	назва звіту;
4.3		ТИП (character varying, довжина 25)	тип звіту;
4.4		ДАТА_СТВОРЕННЯ (date)	дата створення звіту;
4.5		АВТОР (integer)	зовнішній ключ до таблиці КОРИСТУВАЧІ.
5.1	KPI	id_kpi (integer)	унікальний ідентифікатор KPI;
5.2		НАЗВА (character varying, довжина 255)	назва ключового показника ефективності;
5.3		ЗНАЧЕННЯ (double precision)	числове значення показника;
5.4		ДАТА (date)	дата фіксації значення;
5.5		id_ЗВІТУ (integer)	зовнішній ключ до таблиці ЗВІТИ.

У моделі реалізовано кілька зв’язків типу «один до багатьох», що забезпечують логічну цілісність:

- таблиця ОПЕРАЦІЇ_З_ВІДХОДАМИ пов’язана з таблицею ВІДХОДИ через зовнішній ключ id_ВІДХОДУ;
- та сама таблиця також пов’язана з КОРИСТУВАЧІ через поле ВІДПОВІДАЛЬНИЙ, що посилається на id_КОРИСТУВАЧА;
- таблиця ЗВІТИ містить зовнішній ключ АВТОР, який пов’язаний із таблицею КОРИСТУВАЧІ;
- таблиця KPI прив’язана до таблиці ЗВІТИ через поле id_ЗВІТУ.

Запропонована структура бази даних забезпечує логічну цілісність, дозволяє ефективно реалізувати функції аналізу, звітності, моніторингу та контролю відходів. Використання зовнішніх ключів сприяє збереженню коректності зв’язків між сутностями та забезпечує масштабованість бази при розширенні системи.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Для створення прикладного програмного забезпечення системи візуалізації та контролю процесів поводження з відходами було обрано сучасний стек технологій, що забезпечує надійність, продуктивність і підтримку масштабування. Комбінація інструментів охоплює засоби для розробки серверної частини, клієнтського інтерфейсу, бази даних, управління версіями та прототипування інтерфейсу користувача.

Серверна частина системи реалізується з використанням мови програмування PHP, яка забезпечує обробку запитів, взаємодію з базою даних та реалізацію бізнес-логіки на основі фреймворку Symfony. Основною мовою для реалізації бекенд-логіки обрано PHP, яка є стабільним і широко підтримуваним рішенням для розробки веб-застосунків. PHP забезпечує роботу з HTTP-запитами, авторизацію, сесійну взаємодію та зв’язок із базами даних. Для

організації архітектури системи було використано фреймворк Symfony, що реалізує шаблон Model-View-Controller (MVC). Його застосування дозволяє чітко розділити логіку, представлення та обробку даних.

Symfony має вбудовані механізми маршрутизації, шаблонізації (Twig), обробки форм і безпечної взаємодії з ORM. У системі застосовуються такі компоненти Symfony, як Security, Validator, Form, що полегшує впровадження контролю доступу, перевірки даних і взаємодії з користувачем. Завдяки повторному використанню компонентів і підтримці стандартів PSR, фреймворк є масштабованим рішенням для проєктів корпоративного рівня.

Робота з базами даних у системі організована на основі реляційного підходу, де для зберігання структурованої інформації використовується СКБД MySQL у поєднанні з ORM-інструментом Doctrine. Для зберігання та обробки структурованих даних (користувачі, відходи, KPI, звіти) використовується MySQL версії 5.7 або новішої. Це надійна реляційна СКБД, що підтримує ACID-транзакції, індекси, зовнішні ключі та інструменти оптимізації запитів. Вибір MySQL обумовлений її сумісністю з PHP і широкою підтримкою ORM-рішень.

Для взаємодії з базою даних використовується Doctrine ORM — потужний інструмент об'єктно-реляційного відображення, що абстрагує SQL-запити. Doctrine дозволяє працювати з сутностями як з об’єктами, підтримує каскадні операції, lazy loading і міграції. Це спрощує супровід проєкту та покращує читабельність коду, особливо при розширенні моделі даних.

Клієнтська частина (фронтенд) відповідає за взаємодію користувача із системою та реалізована з використанням стандартних веб-технологій, які забезпечують адаптивність, доступність і інтерактивність інтерфейсу. Фронтенд системи побудований на стандартних веб-технологіях: HTML5, CSS3 та JavaScript [10]. HTML забезпечує семантичну структуру сторінок, CSS відповідає за стилізацію та адаптивне відображення, а JavaScript надає інтерфейсу інтерактивності — наприклад, для фільтрації таблиць, оновлення графіків без перезавантаження сторінки тощо.

Для швидкої стилізації використано CSS-фреймворки Bootstrap або Tailwind CSS, що дозволяють створити візуально привабливий і доступний дизайн без потреби ручної обробки стилів. За потреби проєкт може бути розширений з використанням JavaScript-фреймворків React або Vue.js, які дають змогу реалізувати складні реактивні компоненти.

Проєктування інтерфейсу користувача здійснювалося з урахуванням принципів зручності, доступності та візуальної послідовності, що забезпечується використанням спеціалізованих інструментів для UI/UX-дизайну. Для проєктування та прототипування інтерфейсу було використано Figma — онлайн-інструмент для UI/UX-дизайну, що дозволяє командній роботі над макетами у реальному часі. Figma забезпечує створення інтерактивних прототипів, узгодження кольорових схем, перевірку юзабіліті та зручне документування візуальних компонентів.

Інструменти розробки та супроводу програмного забезпечення обрані з урахуванням вимог до ефективної реалізації, зручного налагодження коду та командної роботи над проєктом. Розробка програмного коду здійснюється в інтегрованих середовищах PhpStorm та Visual Studio Code, які забезпечують підсвічування синтаксису, автодоповнення, роботу з Git та зручну інтеграцію з тестовими фреймворками. Система контролю версій побудована на Git, із розміщенням репозиторію на платформах GitHub або GitLab, що дозволяє зручно відстежувати зміни, працювати в команді та реалізовувати CI/CD.

Тестування є важливим етапом у процесі розробки, який забезпечує стабільність, надійність та відповідність функціональності програмного забезпечення заявленим вимогам. Для перевірки якості коду застосовано модульне тестування:

– PHPUnit — для тестування серверної логіки;

– Jest або Mocha — для перевірки фронтенд-компонентів.

Наявність тестів гарантує стабільність системи під час оновлень і дозволяє швидко виявляти помилки.

Обраний інструментарій забезпечує повний цикл розробки, тестування та супроводу програмного забезпечення. Його складові — PHP, Symfony, MySQL, Doctrine, HTML/CSS/JS, Figma та Git — дозволяють реалізувати масштабовану, надійну та зручну у використанні інформаційну систему, що відповідає функціональним і нефункціональним вимогам проєкту.

3.4 Алгоритмізація та програмування програмних модулів

Розробка програмної системи включала проєктування ключових алгоритмів, що реалізують функціональність моніторингу та аналізу даних про відходи. Алгоритми було реалізовано модульно, згідно з принципами структурного та об’єктно-орієнтованого програмування на основі фреймворку Symfony з використанням мови PHP.

Основними модулями системи є:

1. модуль автентифікації користувачів;
2. модуль введення даних про відходи;
3. модуль розрахунку KPI;
4. модуль генерації звітів;

5. модуль сповіщень.

Для реалізації функціональності системи було виділено ключові програмні модулі, кожен з яких відповідає за окрему частину логіки або обробки даних. Їх структура, призначення та взаємозв’язки наведено в таблиці 3.3.

Таблиця 3.3

Основні програмні модулі системи

№	Назва модуля	Функціональність
1	AuthModule	Реєстрація, вхід, автентифікація користувачів, контроль прав доступу
2	WasteDataModule	Введення/редагування даних про відходи, валідація даних, збереження в БД
3	KpiCalculationModule	Автоматичний розрахунок показників ефективності (KPI) за збереженими даними
4	ReportingModule	Генерація звітів у вигляді PDF/таблиць/графіків, експорт
5	NotificationModule	Формування системних повідомлень (перевищення порогів, нові звіти)

Процес розрахунку ключових показників ефективності виконується після оновлення або додавання даних про відходи. Алгоритм визначає співвідношення переробленого обсягу до загального, порівнює результати з цільовими значеннями та виводить результат у відсотках.

Один із ключових етапів функціонування системи — автоматичний розрахунок ключових показників ефективності (KPI) на основі актуальних даних про відходи. Цей процес реалізовано у вигляді послідовного алгоритму, який виконується кожного разу після оновлення або внесення нових записів до бази даних.

Алгоритм починається з отримання системою інформації про загальний обсяг зібраних відходів та частину, що була перероблена. Після цього обчислюється значення KPI за формулою:

$$KPI = (\text{перероблено} / \text{зібрано}) \times 100,$$

що відображає відсоткову ефективність переробки.

Далі результат порівнюється із заздалегідь встановленим цільовим значенням. У разі, якщо фактичне значення KPI є нижчим за допустиму норму, система автоматично формує відповідне попередження для зацікавлених осіб. Незалежно від результату, значення KPI фіксується в базі даних, після чого передається до модуля звітності для подальшого аналізу, візуалізації або включення до зведених документів. Послідовність етапів цього процесу наведено на блок-схемі алгоритму розрахунку KPI (рис. 13).

Як показано на блок-схемі (рис. 13), алгоритм включає послідовні етапи обробки даних, що реалізуються за допомогою відповідних параметрів. Для детальнішого розуміння функціонування модуля KPI у таблиці 3.4 наведено перелік його основних вхідних та вихідних параметрів.

Рис. 13 Блок-схема алгоритму розрахунку KPI

Таблиця 3.4

Вхідні та вихідні параметри модуля KPI

Параметр	Тип даних	Опис
total_waste	float	Загальний обсяг зібраних відходів (кг)
processed_waste	float	Обсяг перероблених відходів (кг)
target_percentage	float	Цільове значення ефективності (%)
kpi_value	float	Розраховане значення KPI (%)
alert_generated	boolean	Ознака перевищення або недосягнення норми

Програмні модулі розроблено відповідно до MVC-архітектури Symfony. Логіка кожного модуля винесена в окремі сервіси, що полегшує повторне використання коду, тестування та розширення. Зв’язок із базою даних реалізовано через Doctrine ORM, що дозволяє працювати з об’єктами як із таблицями бази без прямого SQL-коду.

Виконано алгоритмізацію основних функціональних процесів системи: від введення даних до розрахунку KPI та формування звітів. Реалізація програмних модулів здійснена на основі структурованого підходу з дотриманням принципів модульності, повторного використання та розмежування відповідальності, що підвищує надійність та масштабованість розробленого ПЗ.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування інформаційної системи є критично важливим етапом, що дозволяє перевірити відповідність реалізованого функціоналу технічним вимогам, виявити можливі помилки та оцінити стабільність роботи програмного забезпечення в умовах реального використання. На цьому етапі особливу увагу приділено перевірці коректності авторизації користувачів, достовірності збору та відображення екологічних показників, цілісності даних у базі, а також ефективності реалізованих механізмів візуалізації та звітності.

Запустивши систему, потрапляємо на форму авторизації, тут нам треба ввести логін та пароль користувача (рис. 14).

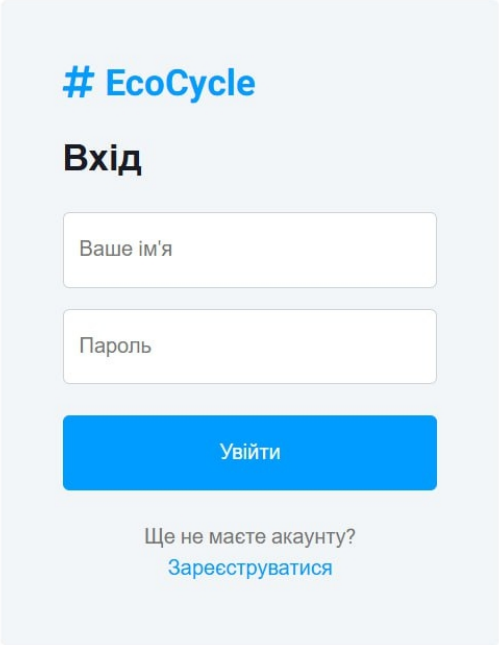

Рис. 14 Форма авторизації

Спочатку нас зустрічає сторінка з інформаційною панеллю. Тут показуються актуальні дані щодо аналізу районів та контролю відходів у вигляді карток (Рис. 15).

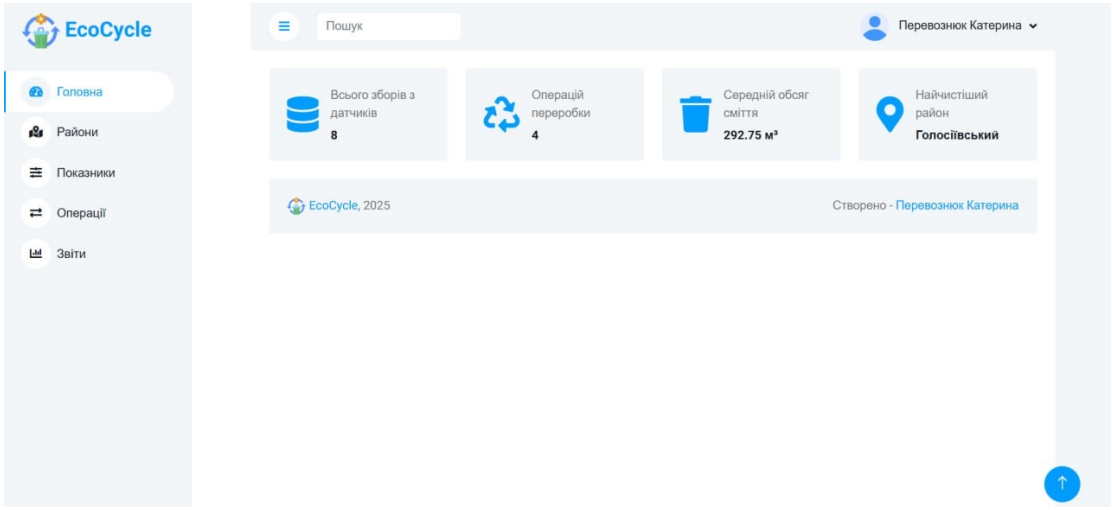

Рис. 15 Інформаційна панель

Перейдемо на сторінку "Райони". Тут у нас представлені всі райони Києва у вигляді блоків, на яких відображається назва та поточний рівень чистоти (Рис. 16).

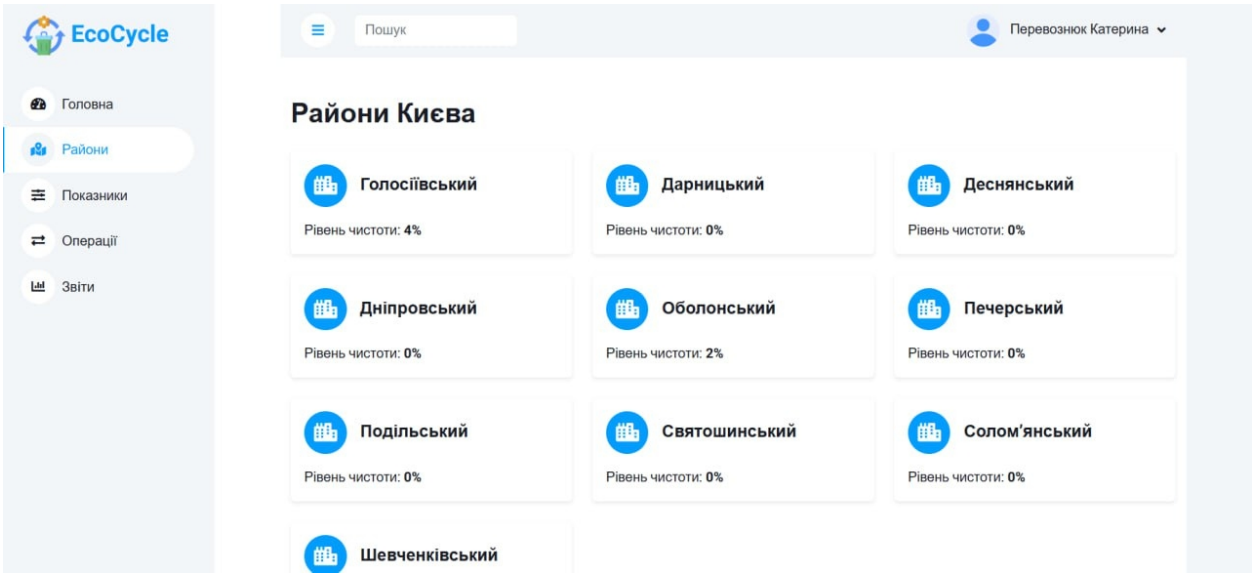

Рис. 16 Сторінка “Райони”

Ми можемо натиснути на будь-який обраний район, і нам відкриється сторінка з його детальною інформацією (Рис. 17).

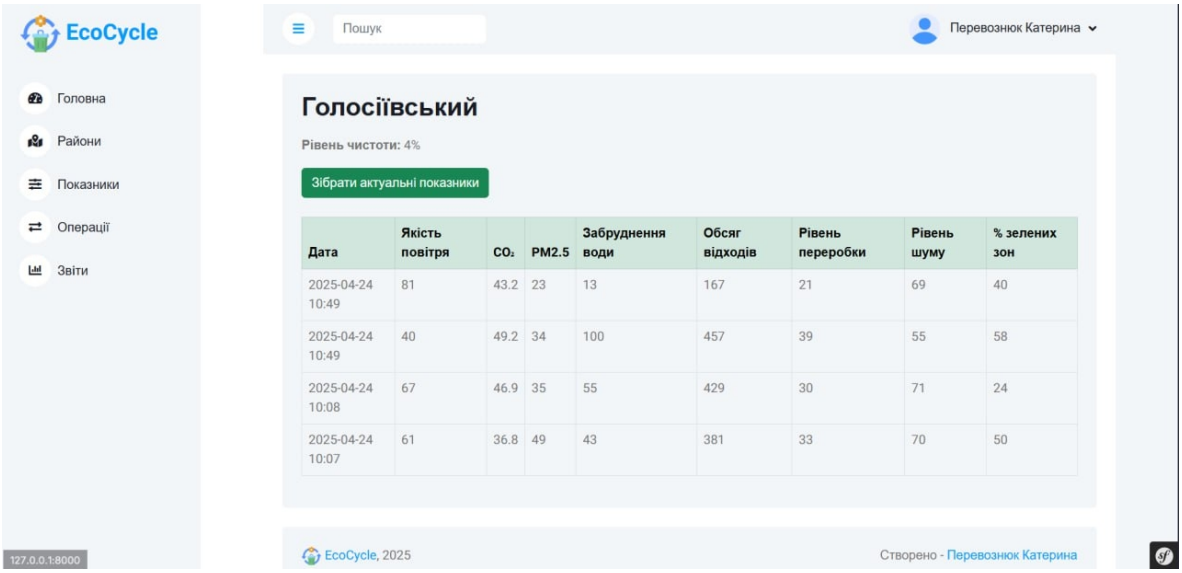

Рис. 17 Вибраний район

Тут у нас показується таблиця з усіма зборами показників по цьому району, а також сама кнопка збору показників. Можемо натиснути на неї і перевірити, які дані будуть зібрані в даний момент (рис. 18).

Дата	Якість повітря	CO_2	PM2.5	Забруднення води	Обсяг відходів	Рівень переробки	Рівень шуму	% зелених зон
2025-04-25 09:01	96	55.8	46	22	435	30	80	22

Рис. 18 Нові забрані дані з датчику

Тепер перейдемо на сторінку "Показники". Тут у нас представлена таблиця з усіма показниками по всіх районах, тут ми можемо детально подивитися всі дані (Рис. 19), а також видалити будь-який запис або підтвердити переробку та очищення відходів (Рис. 20).

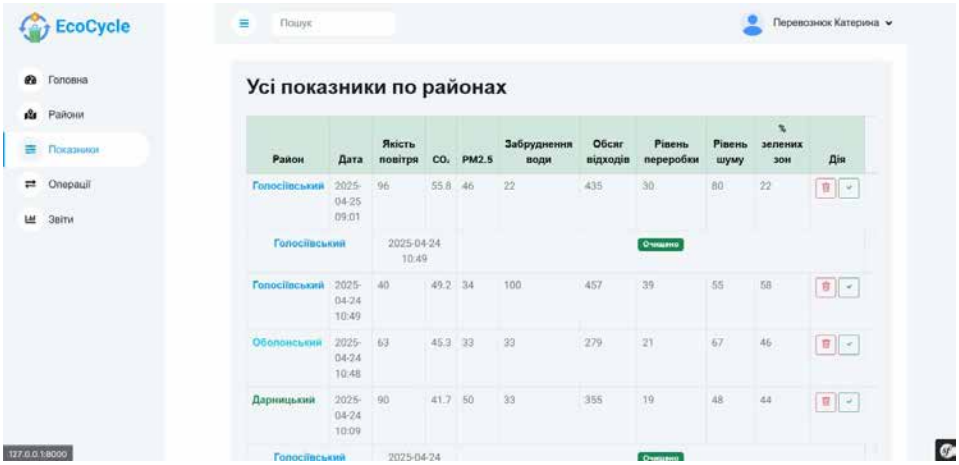

Рис. 19 Сторінка “Показники”

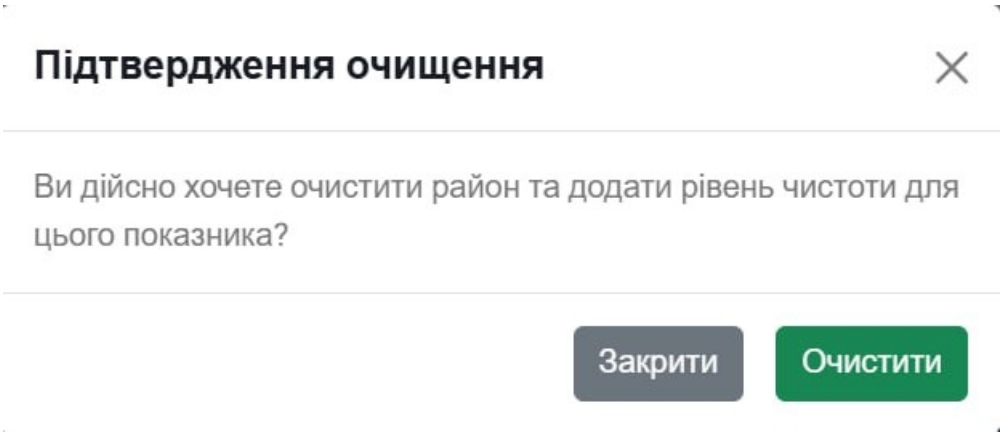

Рис. 20 Модальна форма підтвердження переробки відходів

Підтверджує дію переробки, і тепер у нас з'явився новий запис зі статусом "Очищено" (Рис. 21).

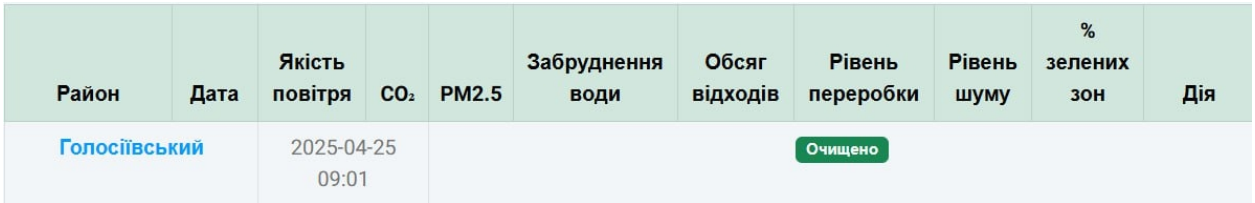

Рис. 21 Новий запис щодо переробки відходів

Тепер переходимо на сторінку "Операції". На ній у нас сторінка розділена на 2 вкладки – збори показників (Рис. 22) та операції переробки (Рис. 23).

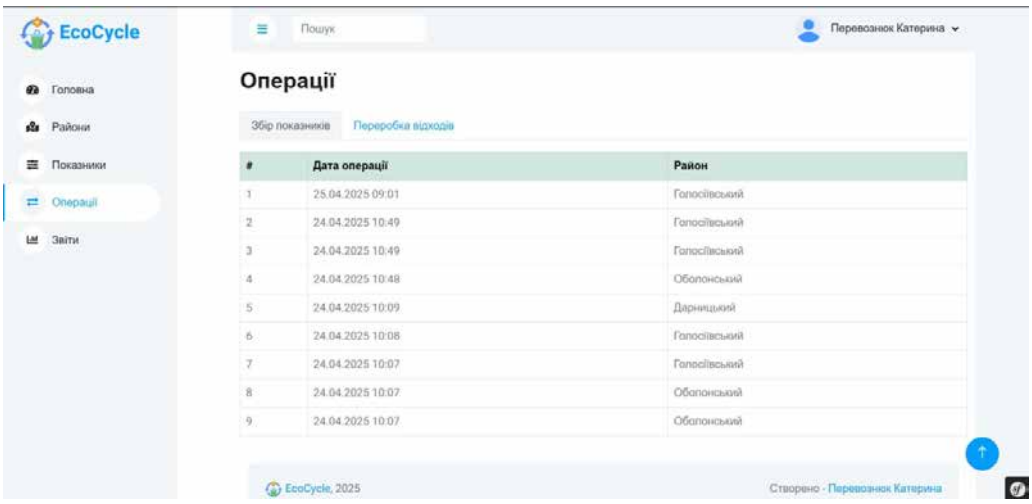

Рис. 22 Сторінка “Операції”

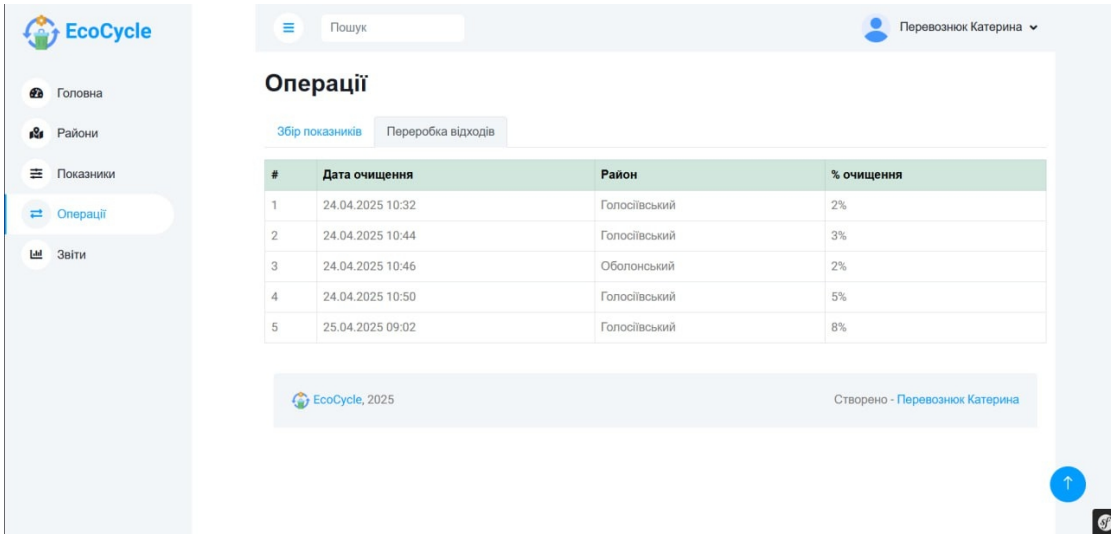

Рис. 23 Сторінка “Операції”

Остання сторінка у нас це "Звіти". Тут працівник може за вибраним діапазоном дат зібрати потрібну йому статистичну інформацію, яка буде виведена сторінці (Рис. 24).

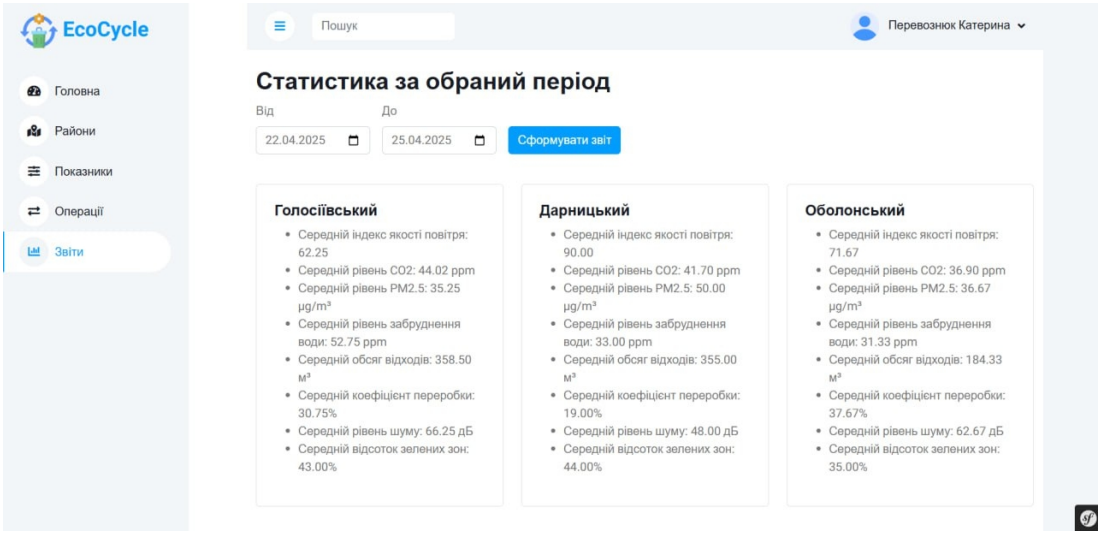

Рис. 24 Сторінка “Звіти”

Також дані виводяться ще й у вигляді графіків (рис. 25).

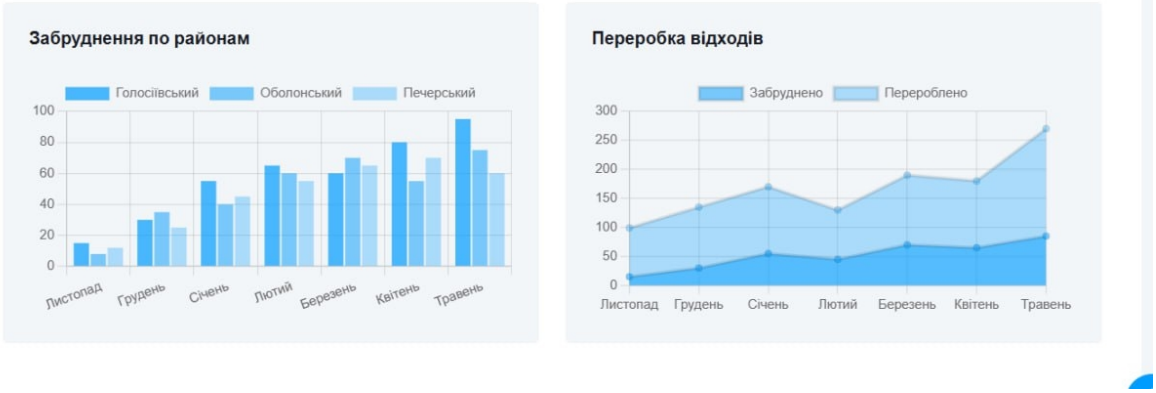

Рис. 25 Сторінка “Звіти”

Проведене тестування підтвердило працездатність основних функціональних модулів системи та її здатність забезпечувати інтерактивну взаємодію з користувачами, своєчасне оновлення екологічних показників і формування звітності. Інтерфейс продемонстрував стабільну роботу в сучасних браузерах, а логіка обробки даних — відповідність очікуваній поведінці. Отримані результати свідчать про готовність програмного забезпечення до впровадження в реальні умови експлуатації.

4.2 Визначення технічних вимог до використання системи

Ефективне функціонування розробленої інформаційної системи для візуалізації даних щодо контролю та переробки відходів значною мірою залежить від правильно сформованих технічних вимог. Вони охоплюють як апаратне, так і програмне забезпечення, необхідне для розгортання, обслуговування та використання системи.

З боку апаратного забезпечення основну роль відіграє серверна інфраструктура. Для забезпечення стабільної обробки запитів, взаємодії з базою даних і одночасної роботи кількох користувачів рекомендується використовувати багатоядерний процесор, оперативну пам’ять обсягом не менше 8 ГБ (для виробничих середовищ — 16 ГБ і більше) та SSD-накопичувач ємністю від 100 ГБ. Також важливо забезпечити пропускну здатність мережі не менше ніж 100 Мбіт/с, особливо в умовах інтенсивної роботи з великими наборами даних.

Клієнтська частина системи не потребує високопродуктивного обладнання: достатньо двоядерного процесора, 4 ГБ оперативної пам’яті та сучасного браузера (Chrome, Firefox, Safari, Edge) на будь-якій операційній системі — Windows, Linux чи macOS. Це дозволяє користувачам працювати з вебзастосунком без необхідності встановлення додаткового програмного забезпечення.

У табл. 4.2 наведено узагальнені технічні вимоги до серверного та клієнтського середовища.

З боку програмного забезпечення сервер повинен функціонувати на базі надійної операційної системи — Linux (Ubuntu, CentOS) або Windows Server. Розгортання вебзастосунку здійснюється за допомогою Apache або Nginx, із підтримкою PHP (версії 7.4 або вище). Для ефективної обробки запитів використовується фреймворк Symfony, який реалізує модульну архітектуру, дозволяє організувати маршрутизацію, роботу з шаблонами та взаємодію з базою даних через ORM Doctrine.

Таблиця 4.2

Технічні вимоги до функціонування системи

Категорія	Компонент	Мінімальні вимоги	Рекомендовані вимоги
1	2	3	4
Сервер	Процесор	4 ядра	8 і більше ядер
	Оперативна пам’ять	8 ГБ	16 ГБ і більше
	Сховище	100 ГБ SSD	256 ГБ SSD
	Пропускна здатність мережі	100 Мбіт/с	1 Гбіт/с
	Операційна система	Ubuntu 20.04 / CentOS 8 / Windows Server	Ubuntu LTS / Windows Server (остання версія)
	Веб-сервер	Apache / Nginx	Apache + PHP-FPM
	Мова програмування	PHP 7.4 або новіше	PHP 8.x
	База даних	MySQL 5.7+	MySQL 8.x
	ORM	Doctrine ORM	Doctrine з підтримкою міграцій
	Фреймворк	Symfony	Symfony 6.x
Клієнт	Процесор	2 ядра	4 ядра
	Оперативна пам’ять	4 ГБ	8 ГБ
	Вільне місце на диску	1 ГБ	2–5 ГБ
	Операційна система	Windows/Linux/macOS	Актуальні версії ОС
	Браузер	Chrome, Firefox, Edge, Safari (останні версії)	Оновлені версії з підтримкою HTML5/JS

Сховище даних базується на MySQL, що забезпечує збереження інформації про відходи, KPI, користувачів і звіти. Doctrine ORM абстрагує роботу з SQL-запитами, надаючи зручні інтерфейси для розробника та підвищуючи надійність програмного коду.

Інтерфейс користувача реалізовано на основі HTML5, CSS3 та JavaScript, з можливістю використання фреймворків (React, Vue.js) у майбутньому. Для стилізації інтерфейсу застосовуються Bootstrap або Tailwind CSS. Проєктування UI-макетів виконувалося у Figma, що забезпечило швидке узгодження вигляду та взаємодії.

Розробка та супровід програмного коду здійснюється в середовищах PhpStorm, VS Code або Sublime Text із використанням Git. Проєкт розміщено на GitHub/GitLab, що забезпечує контроль версій і командну взаємодію. Для тестування використано PHPUnit (бекенд) і Jest або Mocha (фронтенд), що гарантує якість та стабільність системи.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення включає всі необхідні компоненти для повноцінного розгортання та запуску системи в обраному середовищі. До його складу входять:

- вихідні файли проєкту (каталоги src, public, templates, config);
- налаштування маршрутизації, шаблонів, безпеки (routes.yaml, security.yaml);
- файл конфігурації бази даних .env з параметрами підключення;
- SQL-дамп бази даних (schema.sql);
- скрипти ініціалізації та міграцій (migrations/);
- інструкція зі встановлення README.md;
- Docker-файл (Dockerfile) та docker-compose.yml (у разі контейнерного розгортання);
- файли тестів (tests/) та документація до API (якщо застосовується);
- підключені залежності (через composer.json, package.json).

Розгортання системи передбачає кілька альтернативних варіантів залежно від технічного середовища замовника. Зокрема, для хмарного хостингу можуть бути використані такі сервіси, як Amazon Web Services (AWS), DigitalOcean або Heroku. Ці платформи забезпечують високий рівень доступності, масштабування, автоматичний моніторинг та підтримку інструментів для балансування навантаження, що є важливим для забезпечення стабільної роботи в умовах зростаючого обсягу даних та користувачів.

У випадку локального розгортання або тестування застосунок може бути запущений за допомогою Docker, що дозволяє контейнеризувати серверну частину, базу даних та супутні сервіси. Це забезпечує відтворюваність середовища, ізоляцію процесів та простоту перенесення з розробки до продакшену. Використання docker-compose дозволяє одночасно запускати необхідні служби (Symfony, MySQL, phpMyAdmin) із заздалегідь визначеними параметрами.

Для зручності деплойменту та візуального уявлення структури системи створено діаграму розгортання (рис. 26), яка демонструє розміщення основних програмних компонентів на сервері, взаємодію з клієнтськими пристроями, а також підключення до бази даних та зовнішніх API.

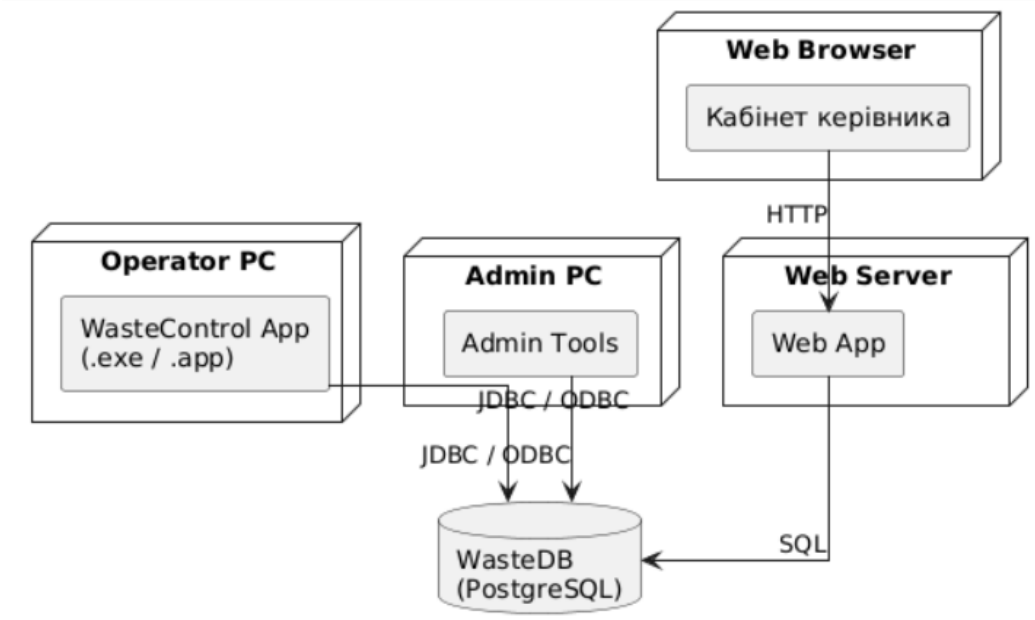

Рис. 26 Діаграма розгортання

Таким чином, інсталяційний пакет містить усі необхідні компоненти для повноцінного розгортання, налаштування та запуску системи в різних середовищах — як у хмарних інфраструктурах, так і на локальних серверах. Використання контейнеризації, підтримка масштабованості та наявність чіткої структури пакету забезпечують стабільну роботу програмного забезпечення, його портативність та зручність у супроводі.

ВИСНОВКИ

Спираючись на початкові цілі проєкту, програмне забезпечення для візуалізації даних про контроль відходів та переробку розроблено не лише для того, щоб надати уявлення про поточні практики управління відходами, але й для стимулювання значущих змін у галузі. Зручний інтерфейс системи дозволить зацікавленим сторонам — від операторів з контролю відходів до державних установ та екологічних неурядових організацій — легко відстежувати діяльність та тенденції, пов'язані з відходами, забезпечуючи швидке та ефективне прийняття обґрунтованих рішень.

Завдяки інтеграції інструментів візуалізації даних система перетворить складну статистику відходів на зрозумілі візуальні формати, що полегшить виявлення закономірностей, відстеження ефективності та визначення областей для покращення. Незалежно від того, чи йдеться про відстеження цілей щодо скорочення відходів, чи про забезпечення дотримання екологічних норм, користувачі матимуть доступ до показників у режимі реального часу, які допоможуть їм вимірювати успіх своїх ініціатив та оптимізувати майбутні зусилля.

Обраний технологічний стек — PHP, Symfony, MySQL, Doctrine ORM та фронтенд-технології, такі як HTML, CSS та JavaScript — гарантує, що додаток буде високочутливим, надійним та безпечним. Ці технології не лише забезпечують безперебійний користувацький досвід, але й гарантують, що програмне забезпечення є достатньо гнучким для майбутніх оновлень, додаткових модулів або навіть інтеграції з іншими екологічними системами чи технологіями.

Більше того, модульна конструкція цього програмного забезпечення, що підтримується потужними бекенд-технологіями, позиціонує його як рішення, яке можна розширити за межі контролю та переробки відходів, потенційно пропонуючи аналогічні рішення для інших галузей, орієнтованих на сталий розвиток. Акцент на безпеці даних та надійності системи гарантує, що програмне

забезпечення зможе безпечно обробляти конфіденційні екологічні дані, забезпечуючи користувачам спокій.

Успішне впровадження цього програмного рішення є важливою віхою в галузі моніторингу навколишнього середовища та управління ресурсами. Воно служить практичним інструментом для підвищення ефективності управління відходами, зменшення впливу на навколишнє середовище та сприяння більш сталому підходу до утилізації та переробки відходів. У майбутньому програмне забезпечення може розвиватися, включаючи передову аналітику, штучний інтелект або алгоритми машинного навчання, щоб забезпечити ще глибше розуміння тенденцій управління відходами, що ще більше розширить можливості користувачів проактивно вирішувати проблеми, пов'язані з відходами.

Завдяки врахуванню як поточних, так і майбутніх технологічних досягнень, проєкт узгоджується з глобальними зусиллями щодо створення більш сталого майбутнього та пропонує модель для інших секторів у використанні технологій для вирішення нагальних екологічних проблем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bigbelly. Clean Software [Електронний ресурс]. – Режим доступу: https://bigbelly.com/products/clean-software
2. Enevo. Waste Sensor [Електронний ресурс]. – Режим доступу: https://enevo.com/waste-sensor/
3. Deployment Diagram in UML: Definition, Examples & Components [Електронний ресурс]. – Режим доступу: https://study.com/academy/lesson/deployment-diagram-in-uml-definition-examples-components.html
4. What is a data flow diagram? Lucidchart [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/data-flow-diagram
5. Microsoft Docs. Layered architecture pattern [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/layered
6. What is Component Diagram? Visual Paradigm [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
7. Entity Relationship Diagram – Data Modeling [Електронний ресурс]. – Режим доступу: https://www.visualparadigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html
8. UML 2 Tutorial – Package Diagram. Sparx Systems [Електронний ресурс]. – Режим доступу: https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html
9. Bootstrap Documentation [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/documentation
10. W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com

11. What is Sequence Diagram? Visual Paradigm [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
12. UML – Activity Diagrams. TutorialsPoint [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm
13. Побудова діаграми класів. Flexberry [Електронний ресурс]. – Режим доступу: https://flexberry.github.io/ru/gpg_class-diagram.html
14. UML-діаграми класів [Електронний ресурс]. – Режим доступу: https://prog-cpp.ua/uml-classes/
15. Багаторівнева архітектура [Електронний ресурс]. – Режим доступу: https://simpleone.ua/glossary/mnogourovnevaya-arhitektura/
16. The Clean Architecture [Електронний ресурс]. – Режим доступу: https://medium.com/clean-code-channel/clean-architecture-the-solution-to-have-a-reusable-flexible-and-testable-code-ac7e296d1a75
17. Типи архітектури програмного забезпечення [Електронний ресурс]. – Режим доступу: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724
18. What is Unified Modeling Language (UML)? Visual Paradigm [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
19. Проєктування ER-діаграми [Електронний ресурс]. – Режим доступу: https://nationalteam.worldskills.ua/skills/proektirovanie-er-diagrammy/
20. Діаграма варіантів використання (Use Case Diagram) [Електронний ресурс]. – Режим доступу: https://flexberry.github.io/ru/fd_use-case-diagram.html
21. Проєктування Use Case діаграми. Визначення функціональних можливостей системи [Електронний ресурс]. – Режим доступу: https://nationalteam.worldskills.ua/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/

22. Валачіч Дж. С., Джордж Дж. Ф., Хоффер Дж. А. Сучасний системний аналіз та проєктування. 8-е вид. Бостон: Pearson, 2016. 544 с.
23. Соммервіль І. Програмна інженерія. 10-е вид. Бостон: Pearson, 2015. 816 с.
24. Коронел К., Морріс С., Крокетт К., Блюетт С. Принципи баз даних: Основи проєктування, впровадження та управління. 3-є вид. Бостон: Cengage Learning, 2020. 928 с.
25. Елмасрі Р., Наватхе Ш. Основи систем баз даних [Електронний ресурс]. 7-е вид. Бостон: Pearson, 2017. 1280 с. Режим доступу: https://www.pearson.com/store/p/fundamentals-of-database-systems/P100000639062

ДОДАТОК А

Фрагменти програмного коду. Функція збору показників району з датчиків

```
<?php

namespace App\Controller;

use App\Entity\District;
use App\Entity\Indicators;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class DistrictController extends AbstractController
{
    #[Route('/districts', name: 'app_districts')]
    public function index(EntityManagerInterface $entityManager): Response
    {
        $districts = $entityManager->getRepository(District::class)->findAll();

        return $this->render('districts/index.html.twig', [
            'districts' => $districts,
        ]);
    }

    #[Route('/districts/{id}', name: 'app_district_show')]
    public function show(int $id, EntityManagerInterface $entityManager): Response
    {
        $district = $entityManager->getRepository(District::class)->find($id);

        $indicators = $entityManager->getRepository(Indicators::class)
            ->findBy(['district' => $district], ['createdAt' => 'DESC']);

        return $this->render('districts/show.html.twig', [
            'district' => $district,
            'indicators' => $indicators,
        ]);
    }

    #[Route('/districts/{id}/check', name: 'app_district_check', methods: ['POST'])]
    public function checkIndicators(int $id, EntityManagerInterface $entityManager): Response
    {
        $district = $entityManager->getRepository(District::class)->find($id);

        $indicator = new Indicators();
        $indicator->setDistrict($district);
        $indicator->setCreatedAt(new \DateTimeImmutable());
```

```
        $indicator->setAirQualityIndex(rand(30, 100));
        $indicator->setCo2Level(mt_rand(300, 600) / 10);
        $indicator->setPm25(mt_rand(10, 50));
        $indicator->setWaterPollutionLevel(mt_rand(0, 100));
        $indicator->setWasteVolume(mt_rand(50, 500));
        $indicator->setRecyclingRate(mt_rand(10, 90));
        $indicator->setNoiseLevel(mt_rand(40, 80));
        $indicator->setGreenZonePercentage(mt_rand(10, 60));

        $newPurity = $district->getDistrictPurity() - 2;

        if ($newPurity < 0) {
            $newPurity = 0;
        }

        $district->setDistrictPurity($newPurity);

        $entityManager->persist($indicator);
        $entityManager->persist($district);
        $entityManager->flush();

        return $this->redirectToRoute('app_district_show', ['id' => $id]);
    }
}
```

ДОДАТОК Б

Фрагменти програмного коду. Функція переробки відходів

```
<?php

namespace App\Controller;

use App\Entity\Indicators;
use App\Entity\Operation;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Bundle\SecurityBundle\Security;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class IndicatorController extends AbstractController
{
    #[Route('/indicators', name: 'app_indicators')]
    public function index(EntityManagerInterface $entityManager): Response
    {
        $indicators = $entityManager->getRepository(Indicators::class)
            ->findBy([], ['createdAt' => 'DESC']);

        return $this->render('indicators/index.html.twig', [
            'indicators' => $indicators,
        ]);
    }

    #[Route('/indicators/delete/{id}', name: 'app_indicator_delete', methods: ['POST'])]
    public function delete(int $id, EntityManagerInterface $entityManager): Response
    {
        $indicator = $entityManager->getRepository(Indicators::class)->find($id);

        $entityManager->remove($indicator);
        $entityManager->flush();

        return $this->redirectToRoute('app_indicators');
    }

    #[Route('/indicators/clean/{id}', name: 'app_indicator_clean', methods: ['POST'])]
    public function cleanIndicator(
        int $id,
        EntityManagerInterface $entityManager,
        Request $request,
        Security $security
    ): Response {
        $indicator = $entityManager->getRepository(Indicators::class)->find($id);
```

```
        if (!$indicator) {
            return $this->redirectToRoute('app_indicators');
        }

        if (!$this->isCsrfTokenValid('clean' . $indicator->getId(), $request->request->get('_token'))) {
            return $this->redirectToRoute('app_indicators');
        }

        $indicator->setCleaned(true);

        $district = $indicator->getDistrict();

        $newPurity = $district->getDistrictPurity() + rand(2, 10);

        if ($newPurity > 100) {
            $newPurity = 100;
        }

        $district->setDistrictPurity($newPurity);

        $operation = new Operation();
        $operation->setIndicator($indicator);
        $operation->setUser($security->getUser());
        $operation->setOperationDate(new \DateTime());
        $operation->setCleanedPercent($newPurity);

        $entityManager->persist($operation);
        $entityManager->persist($district);
        $entityManager->flush();

        return $this->redirectToRoute('app_indicators');
    }
}
```