НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ _____Голуб Б.Л._____
(підпис) (ПІБ)

"___" червня 2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему
«Програмне забезпечення інформаційної системи територіальної громади»

Спеціальність 121 – «Інженерія програмного забезпечення» ОПП – «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н, доцент_____ _____ __ Вайганг Г.О._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____к.т.н. доцент_____ _____ __ Бородкіна І.Л._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Міцай Руслан Олександрович____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОВИКОРИСТАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерних наук

(назва кафедри)

_____/ Голуб Б.Л. доцент к.т.н./

(підпис)

"___" _____2025р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Міцаю Руслану Олександровичу

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення інформаційної системи територіальної громади» затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 «С».
2. Термін подання завершеної роботи на кафедру 2025 . 06 . ____

(рік, місяць, число)

3. Вихідні дані для роботи: надання інформації про роботу територіальних громад України з жителями у вигляді списків та таблиць.
4. Перелік питань, що розглядаються:
   1. Системний аналіз предметної області
   2. Проєктування інформаційного та програмного забезпечення
   3. Розробка інформаційного та програмного забезпечення
   4. Рекомендації щодо впровадження та експлуатації системи
   5. Висновки

Керівник бакалаврської кваліфікаційної роботи_____/Бородкіна І.Л. /

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/ Міцай Р.О. /

(підпис) (прізвище та ініціали)

Дата подання завдання 2025.01.08

Рік місяць число

ЗМІСТ

ВСТУП

У сучасних умовах розвитку інформаційних технологій дедалі важливішим стає питання автоматизації управлінських процесів у всіх сферах суспільного життя, зокрема в державному та муніципальному управлінні. Територіальні громади, як базові одиниці адміністративно-територіального устрою, потребують сучасних інструментів для ефективного управління ресурсами, оперативного обліку населення, обробки звернень громадян та надання адміністративних послуг. Ручна робота з великими обсягами документів, застарілі підходи до зберігання даних і фрагментованість інформації призводять до затримок, помилок, дублювання та втрати даних, що, своєю чергою, знижує ефективність роботи органів місцевого самоврядування та викликає невдоволення громадян.

Актуальність завдання розробки інформаційної системи для територіальної громади зумовлена як об'єктивними потребами в модернізації управлінських процесів, так і загальнодержавною політикою цифрової трансформації. Зокрема, в нашій країні діють національні програми цифрового врядування, які передбачають впровадження електронних сервісів на місцевому рівні. Проте значна частина громад усе ще використовує паперові носії або фрагментарні рішення, які не забезпечують комплексного охоплення даних про мешканців, житловий фонд, адміністративні одиниці, а також не дозволяють ефективно обробляти звернення громадян.

У цьому контексті виникає потреба у створенні єдиного програмного продукту, який дозволяє централізовано зберігати, обробляти та аналізувати інформацію, необхідну для функціонування громади. Така система має бути простою у використанні, але водночас потужною в частині обробки запитів, побудови звітів, забезпечення безпеки даних та багатокористувацького доступу.

Метою даної бакалаврської роботи є розробка інформаційної системи для територіальної громади, яка дозволить:

1. забезпечити електронний облік мешканців громади та пов’язаних з ними даних (адреси, статуси, типи будинків тощо);
2. автоматизувати процес обробки заяв громадян – від моменту подачі до зміни статусу та завершення обробки;
3. забезпечити фільтрацію та пошук інформації за географічними та адміністративними параметрами (область, район, населений пункт, вулиця, будинок);
4. спростити підготовку звітів та довідок;
5. реалізувати розмежування доступу між різними категоріями користувачів (наприклад, секретарі, оператори).

З технічного погляду, система повинна бути масштабованою, стабільною та зручною для подальшої підтримки й розвитку. Це дозволить у майбутньому інтегрувати її з іншими державними реєстрами або електронними сервісами (наприклад, порталом Дія).

Методи та технології, які застосовувалися під час розробки програмного забезпечення, були обрані з урахуванням таких чинників, як надійність, поширеність, підтримка українськими інституціями, зручність для розробника та кінцевого користувача. Зокрема, були використані такі технології та інструменти:

Мова програмування C# та середовище Microsoft Visual Studio як основа для створення десктопного застосунку з багатим користувацьким інтерфейсом.

1. WinForms – для реалізації графічного інтерфейсу, що дозволяє створювати зручні вікна, форми, елементи взаємодії (ComboBox, TextBox, DataGridView тощо) із користувачем.
2. Система управління базою даних PostgreSQL – як надійна, безкоштовна та потужна система керування базами даних із відкритим вихідним кодом, що підтримує складні зв’язки, транзакції, зберігані процедури.

3. Npgsql – офіційна бібліотека .NET для роботи з PostgreSQL, яка дозволяє зручно реалізовувати підключення, виконання SQL-запитів, додавання, оновлення та видалення даних.
4. RDLC-звіти – для реалізації функціоналу формування звітної документації безпосередньо з даних, які зберігаються у базі.

У процесі розробки було спроєктовано структуру бази даних, реалізовано захист підключення, розмежування доступу за логіном і паролем, створено модулі для додавання, перегляду та оновлення записів, побудовано звітні форми, а також передбачено механізм фільтрації даних за географічною ознакою.

Варто відзначити, що до уваги брались реальні кейси використання, зокрема потреби відділу роботи з громадянами, адміністративно-господарського управління, а також обробки запитів соціального характеру.

Розроблена система є прикладом ефективного застосування сучасних засобів розробки програмного забезпечення для вирішення конкретного завдання у сфері муніципального управління. Вона може бути адаптована під інші громади з незначними змінами або розширенням функціоналу.

Таким чином, бакалаврська робота має на меті не лише технічне вирішення проблеми автоматизації обліку та обробки заяв у громаді, а й демонструє потенціал впровадження цифрових сервісів у публічну сферу.

Результати роботи доповідалися на конференції “ Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025” та “Інформаційні технології в Соціокультурній сфері, освіті та економіці”.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Територіальна громада – це основна ланка адміністративно-територіального устрою, яка відіграє ключову роль у взаємодії громадян із державними структурами. В умовах децентралізації управління територіальні громади отримали значно більше повноважень, а відповідно, і зросла кількість завдань, які покладено на місцеві ради, органи самоврядування та відповідальні відділи. Одним з основних напрямів діяльності є ведення обліку населення, управління житловим фондом, обробка заяв громадян та контроль за виконанням рішень, що приймаються на місцевому рівні.

У традиційному форматі робота з інформацією у громадах здійснюється шляхом використання паперових архівів, окремих електронних таблиць або документів, що не пов’язані між собою. Це призводить до фрагментації інформації, дублювання даних, відсутності узгодженості між структурними підрозділами, а також складнощів у побудові звітності, пошуку інформації та верифікації даних [2].

1.1.1 Облік мешканців. Однією з базових функцій громади є точний і своєчасний облік мешканців. Доцільно зберігати інформацію про:

- ПІБ особи;
- адресу проживання (область, район, населений пункт, вулиця, будинок);

- соціальний статус (пенсіонер, внутрішньо переміщена особа, студент, тощо);
- додаткову інформацію, яка може впливати на пільги чи участь у програмах громади.

На основі цих даних громада може реалізовувати соціальні програми, формувати списки для отримання допомоги, будувати аналітичні звіти про демографічну ситуацію.

1.1.2 Територіальне структурування. У межах однієї територіальної громади можуть бути десятки населених пунктів, розділених на райони, мікрорайони або старостинські округи. Важливо мати чітке територіальне структурування:

- Область – адміністративна одиниця найвищого рівня;
- Район – підрозділ області або громади, що охоплює кілька населених пунктів;
- Населений пункт – місто, село, селище;
- Вулиця – детальна адреса в межах населеного пункту;
- Будинок – окремий об’єкт, до якого можуть бути прив’язані мешканці.

Це структурування потрібне для коректного збереження адресних даних, фільтрації запитів, обліку майна та будівництва, а також при створенні звітів.

1.1.3 Облік житлового фонду. Важливим завданням громади є ведення обліку житлових будинків, їх типів, стану, форми власності тощо. Це необхідно для:

- інвентаризації нерухомості;

- розрахунку комунальних послуг;
- визначення потреб у ремонтах чи благоустрої;
- формування податкових та інших державних звітів.

У контексті предметної області житловий фонд поділяється на категорії: приватні, багатоквартирні, комунальні будинки, тимчасові споруди, гуртожитки тощо.

1.1.4 Робота із заявами. Громадяни часто звертаються до органів місцевого самоврядування з різними заявами: щодо матеріальної допомоги, довідок, внесення змін у реєстр мешканців, технічного обслуговування, тощо. Процес обробки таких заяв включає:

- реєстрацію вхідної заявки;
- фіксацію її змісту, дати подачі, заявника;
- призначення відповідального виконавця;
- зміну статусу заявки в процесі роботи (нова, в обробці, виконано, відхилено);
- формування звітності за період або по конкретному населеному пункту.

Автоматизація цього процесу дозволяє не лише зменшити час на обробку заяв, а й запобігти втраті інформації або затримкам у відповідях.

1.1.5 Побудова звітності. Одним з найважливіших завдань для громади є підготовка звітів для внутрішнього аналізу, керівництва, державних установ. Ручне формування звітів є надзвичайно трудомістким. Звітність може включати:

- кількість мешканців по районах та населених пунктах;

- статистику за типами звернень;
- розподіл заяв по статусах;
- реєстри мешканців з певним соціальним статусом;
- аналіз житлового фонду за типами та розташуванням.

Автоматичне створення звітів за шаблонами дозволяє заощадити ресурси, зменшити ризик помилок і забезпечити актуальність інформації.

1.1.6 Основні проблеми предметної області. У сучасному стані багато територіальних громад стикаються з такими труднощами:

- Відсутність централізованої бази даних – інформація зберігається у вигляді окремих файлів, зошитів, документів.
- Низький рівень автоматизації – велика частина процесів виконується вручну, що спричиняє затримки та помилки.
- Відсутність аналітичного інструменту – громади не мають змоги швидко оцінити ситуацію за певним напрямком (заявки, соціальний статус, структура населення).
- Складність у веденні обліку змін – важко відстежити історію змін у даних.
- Відсутність інтерфейсу користувача – співробітникам громади складно працювати з технічними інструментами без належного візуального середовища.

1.1.7 Потенціал автоматизації. Автоматизована інформаційна система може суттєво змінити підхід до обробки даних у громаді. Вона має включати:

- єдину базу даних із доступом за логіном і паролем;

- механізм фільтрації інформації за районами, вулицями, типами даних;
- інструмент для роботи з заявами (реєстрація, статуси, звіти);
- формування RDLC-звітів і експорт у PDF;
- захист даних та резервне копіювання;
- можливість масштабування та розширення функціоналу.

Таким чином, описана предметна область є критично важливою для ефективної роботи органів місцевого самоврядування. Її комплексна автоматизація не тільки зменшить навантаження на працівників громади, а й покращить якість надання послуг мешканцям, забезпечить прозорість процесів, підвищить точність та оперативність рішень.

1.2 Аналіз вимог до предметної області

Аналіз вимог – це перший і ключовий етап при розробці будь-якого програмного забезпечення. Його завдання – виявити потреби кінцевих користувачів, оцінити обсяг і складність предметної області, визначити основні функції майбутньої системи, а також технічні обмеження й умови реалізації. У контексті даного проєкту, метою є створення програмного модуля для автоматизації обліку мешканців, заяв, адміністративних одиниць та формування звітності для органів місцевого самоврядування.

1.2.1 Функціональні вимоги. Функціональні вимоги описують що саме повинна робити система з погляду користувача. Для предметної області територіальної громади сформульовано такі основні функціональні вимоги:

1) Облік мешканців громади:
   - Додавання, редагування та видалення записів про мешканців.

- Прив’язка мешканця до конкретної адреси: область → район → населений пункт → вулиця → будинок.
- Фіксація додаткової інформації (наприклад номер телефону, електронна пошта).

2) Управління адміністративними одиницями:
   - Збереження списку регіонів, районів, населених пунктів, вулиць і будинків.
   - Забезпечення ієрархічної залежності: вибір області фільтрує доступні райони, район – населені пункти, тощо.
   - Валідація унікальності назв в межах ієрархії.
3) Робота із заявами громадян:
   - Реєстрація нової заяви із зазначенням типу, дати подачі, заявника, опису та статусу.
   - Можливість зміни статусу заявки вручну через інтерфейс.
   - Фільтрація заяв за районом, статусом, датою або типом.
4) Побудова звітів:
   - Автоматичне формування RDLC-звітів на основі фільтрованих даних.
   - Експорт звіту у формат PDF для подальшого друку або архівування.
   - Підтримка шаблонів для різних типів звітності.
5) Аутентифікація користувачів:
   - Підключення до бази даних PostgreSQL із логіном і паролем.
   - Контроль доступу до функцій на основі типу користувача.

1.2.2 Нефункціональні вимоги. Нефункціональні вимоги стосуються якості роботи системи та умов її використання.

1) Інтерфейс користувача:

- Проста, інтуїтивно зрозуміла графічна оболонка на основі WinForms.
- Уніфікований стиль оформлення (кольори, шрифт, розміщення елементів).
- Підтримка кирилиці (українська мова інтерфейсу).

2) Продуктивність
   - Швидке завантаження списків даних у ComboBox-елементах.
   - Миттєва реакція на зміну фільтрів.
   - Формування звіту не більше ніж за кілька секунд навіть на великому обсязі даних.
3) Безпека
   - Непряма передача пароля в рядок підключення (не зберігати пароль у відкритому вигляді).
   - Валідація введених даних для уникнення SQL-ін’єкцій.
   - Можливість у майбутньому додати логування дій користувача.
4) Масштабованість
   - Структура бази даних дозволяє додавати нові категорії (наприклад, інші типи заяв).
   - Можливість розширення системи для роботи з іншими реєстрами.
5) Сумісність
   - Робота з PostgreSQL як на локальному сервері, так і на віддаленій базі.
   - Підтримка ОС Windows 10 і новіших.
   - Відсутність залежностей від комерційних бібліотек (використання Open Source рішень: Npgsql, ReportViewer тощо).

1.2.3 Обмеження. Система створюється в рамках навчального проєкту, тому накладаються деякі обмеження:

- Аутентифікація здійснюється лише через підключення до бази даних.
- Відсутність мобільної або веб-версії – лише настільний клієнт.
- Інтерфейс користувача спрощений – без адаптивності або drag-n-drop елементів.

Проаналізовані вимоги свідчать, що система має задовольнити ключові потреби територіальної громади щодо ведення обліку мешканців, заяв та житлового фонду, а також формування аналітичних звітів. Розробка буде базуватись на сучасних інструментах (PostgreSQL, WinForms, RDLC) та матиме зручний інтерфейс для непідготовленого персоналу. При цьому система залишиться достатньо простою, щоб бути підтримуваною та масштабованою без значних витрат на навчання чи супровід.

1.3 Моделювання предметної області

Процес розробки програмного забезпечення неможливий без попереднього формального моделювання предметної області, адже саме це дозволяє точно зрозуміти, які дії має виконувати система, хто її користувачі, як відбувається обробка інформації та в якій послідовності виконуються операції. У рамках цього проєкту – створення інформаційної системи для обліку мешканців, обробки заяв і генерації звітів у межах територіальної громади – особливе значення має побудова UML-діаграм, які наочно відображають функціональність і логіку роботи майбутнього додатку.

1.3.1 Мета використання UML-діаграм. Метою моделювання є створення візуальних моделей, що описують функціональність, взаємодію та внутрішню логіку системи. Зокрема, акцент робиться на:

- визначенні ролей користувачів і доступних їм сценаріїв взаємодії;

- описі послідовності виконання дій у межах ключових процесів (наприклад, подання та обробки заяв);
- формалізації алгоритмів роботи системи через розгалуження, умови, цикли.

Ці цілі реалізуються через побудову трьох ключових типів UML-діаграм: діаграми прецедентів, діаграми послідовності та діаграми активності [5].

1.3.2 Діаграма прецедентів (Use Case Diagram). Цей тип діаграми дозволяє ідентифікувати основних акторів системи (тобто типи користувачів) та сценарії взаємодії між ними і програмним продуктом. У нашій системі системи територіальної громади основними акторами є:

- Секретар адміністрації – відповідає за наповнення довідників, керування користувачами;
- Оператор адміністрації – виконує введення мешканців, поданих заяв;

Основні прецеденти включають:

- «Створення заяв»;
- «Створення та перегляд звітів»;
- «Змінити статус заяви»;
- «Сформувати звіт»;
- «Переглянути звіт»;
- «Ввести дані про область, район, населений пункт і тд.».

Рис. 1.1 – Діаграма прецедентів для системи територіальної громади

Ця діаграма дозволяє зрозуміти функціональні межі системи та потреби кожного типу користувача.

1.3.3 Діаграма послідовності (Sequence Diagram). Діаграма послідовності показує часовий перебіг взаємодії об'єктів під час виконання певного сценарію. На поданому нижче малюнку зображено повний цикл від

внесення заяви до системи до її обробки. Основні об’єкти діаграми, які беруть участь у процесі:

- Оператор адміністрації
- Секретар адміністрації
- Адміністрації

На діаграмі відображається:

- Подання заяви
- Прийняття заяви для обробки у системі
- Створення та перегляд списку заяв
- Створення звітів та їх перегляд

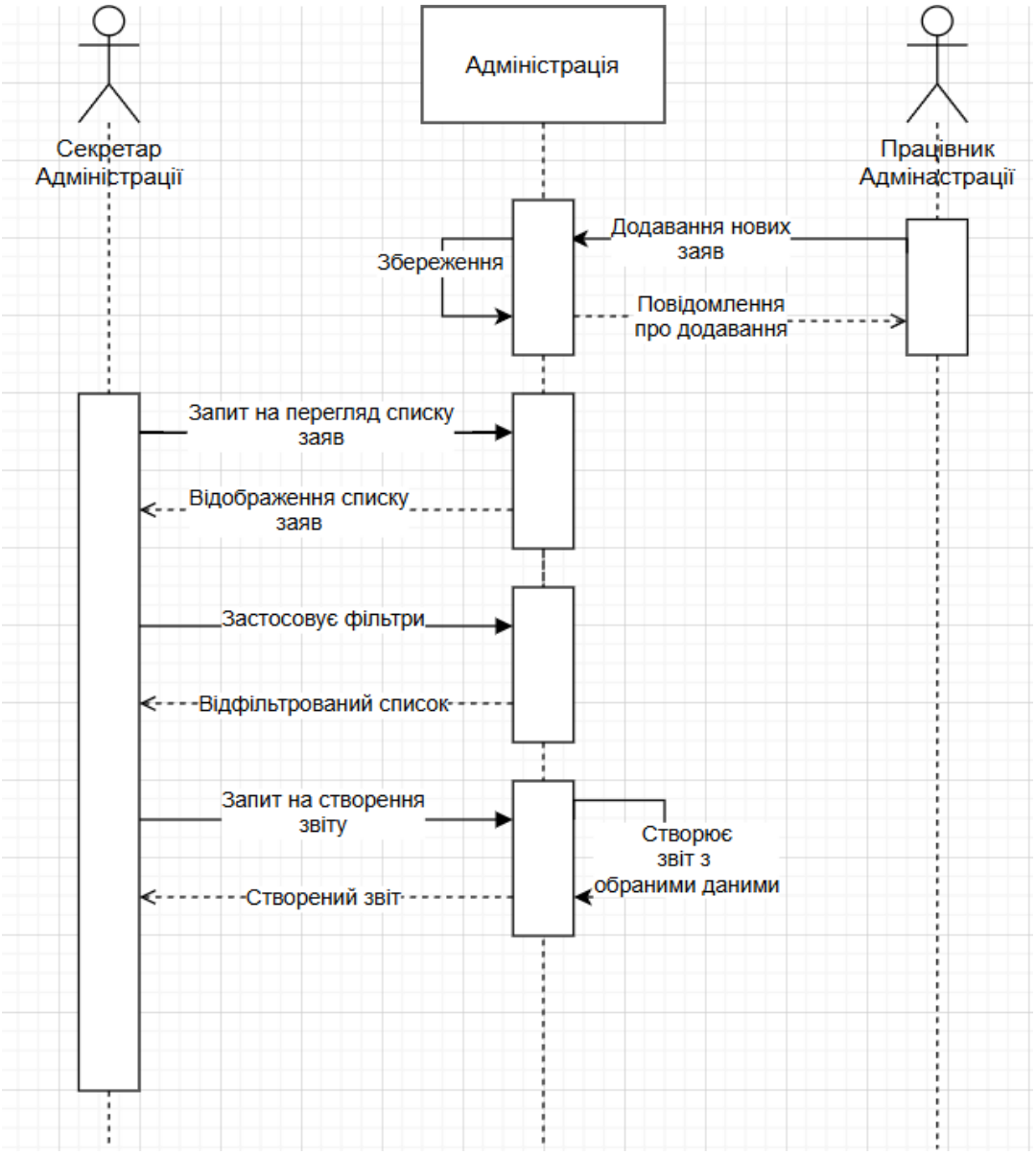

Рис. 1.2 – Діаграма послідовності для системи територіальної громади

Таке моделювання дозволяє розробникам уточнити порядок викликів методів та взаємодію між компонентами на етапі реалізації конкретної функції.

1.3.4 Діаграма активності (Activity Diagram). Цей тип UML-діаграми описує алгоритм виконання дій, включаючи розгалуження, умови та паралельні процеси. У контексті моєї системи є обробка поданої заяви, що включає:

- Прийняття заяви

- Збереження заяви у системі
- Перегляд заяв
- Перегляд опису заяви для подальших дій
- Зміна статусу заяви
- Збереження змін

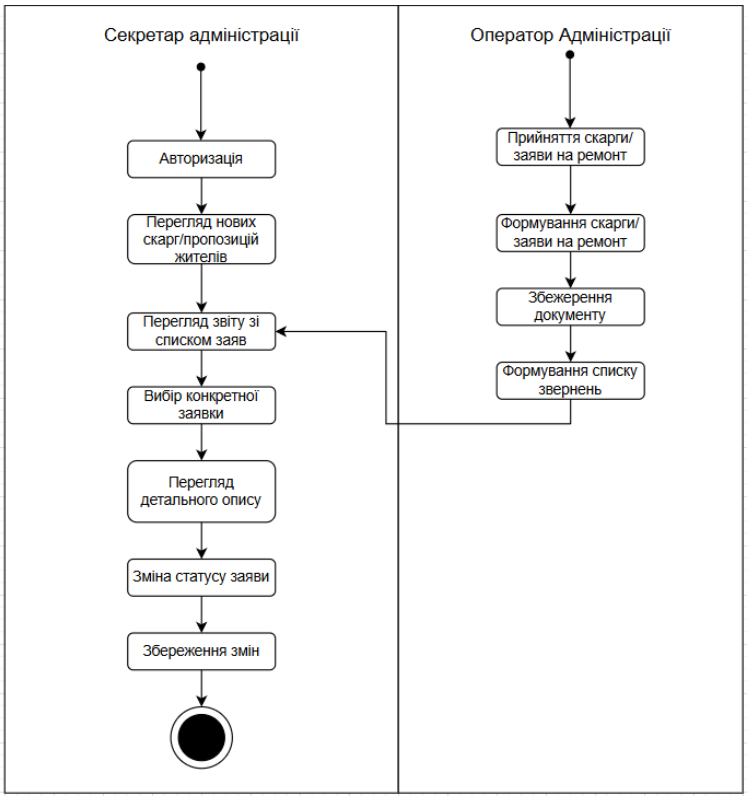

Рис. 1.3 – Діаграма активності системи територіальної громади

Діаграма активності дозволяє зобразити всі варіанти поведінки системи в рамках однієї логічної одиниці – з урахуванням як звичайного, так і альтернативного ходу подій.

1.3.5 Значення для реалізації. Використання UML-діаграм на етапі проєктування дає змогу:

- уникнути двозначностей у розумінні вимог;
- чітко окреслити функціональність та поведінку системи;

- спростити комунікацію між замовником і розробником;
- забезпечити основу для реалізації програмних модулів.

Завдяки створенню діаграм прецедентів, послідовності та активності, стає можливим точне моделювання логіки роботи системи, що дозволяє уникнути суттєвих помилок у процесі її реалізації.

1.4 Огляд інформаційних джерел та існуючих рішень

Розробка інформаційної системи для обліку мешканців, обробки заяв і формування звітності в межах територіальної громади передбачає вивчення як теоретичних основ побудови таких систем, так і аналізу вже реалізованих рішень, що мають подібну функціональність. Даний розділ присвячено аналізу наукових джерел, державних інформаційних систем, а також популярних програмних продуктів у цій галузі.

1.4.1 Науково-методична база. Серед теоретичних джерел, що лягли в основу розробки, варто відзначити праці з тематики проєктування інформаційних систем , баз даних та засобів розробки користувацьких інтерфейсів:

- Класичні праці з інженерії програмного забезпечення (І. Соммервілл, Р. Прессман) описують життєвий цикл програмного забезпечення, етапи проєктування інформаційних систем, принципи нормалізації баз даних та моделювання процесів.
- Практичні посібники з використання технологій .NET, Windows Forms та система управління базою даних PostgreSQL забезпечують технічну базу для реалізації системи.

- Роботи з публічного адміністрування та цифрової трансформації громад (наприклад, дослідження Інституту розвитку місцевої демократії) підкреслюють важливість електронних сервісів для громадян.

Отже, проєкт базується як на технічних принципах побудови інформаційних систем, так і на соціальних потребах сучасного е-врядування.

1.4.2 Аналіз державних інформаційних систем. В Україні функціонує кілька централізованих інформаційних систем, що частково реалізують функції, подібні до запропонованій у моїй системі:

1) Реєстр територіальних громад (РТГ)

Розроблений Міністерством цифрової трансформації, цей реєстр забезпечує базовий облік територіальних одиниць та мешканців. Однак його функціональність часто обмежується наданням інформації органам влади та не забезпечує локалізоване адміністрування на рівні конкретної громади.

2) Єдина інформаційна система соціального захисту населення

Дана система оперує інформацією про отримувачів соціальної допомоги, однак її структура є складною для інтеграції з іншими системами, і вона не забезпечує гнучкого обліку заяв, адресної інформації або мешканців поза контекстом пільг.

3) Системи електронного документообігу громад (наприклад, «АСКОД», «Megapolis.DocNet»)

Ці системи орієнтовані переважно на документообіг у межах органів влади, без фокусування на обліку мешканців або формуванні звітів за заявами.

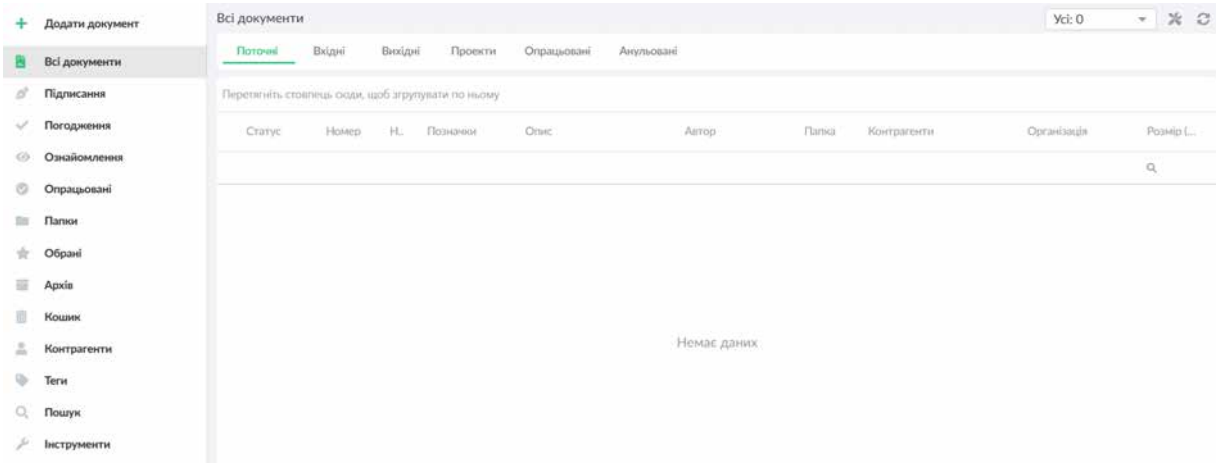

Рис. 1.4 – Інтерфейс програми АСКОД для електронного документообігу

Таким чином, існуючі державні рішення або не охоплюють усі необхідні функції, або є надмірно складними і дорогими для окремих громад.

1.4.3 Комерційні та відкриті рішення у програмному забезпеченні. Серед програмних продуктів, які можна використовувати у муніципальному адмініструванні, варто розглянути:

1) ЦНАП-платформи

Програмні комплекси для Центрів надання адміністративних послуг (наприклад, «Вулик» від e-Governance Academy) мають схожі функції – зокрема, реєстрацію заяв, облік громадян. Але вони часто не дають змоги гнучко змінювати логіку взаємозв’язків між територіальними одиницями, що важливо для кастомізації під потреби конкретної громади[3].

2) ERP-системи для муніципалітетів

Існують спеціалізовані модулі у складі ERP-систем (наприклад, SAP для публічного сектору), які включають управління територіями, мешканцями, послугами. Проте такі рішення надто дорогі та складні для громад середнього та малого розміру.

3) Open Source рішення

Серед відкритих систем зустрічаються окремі спроби автоматизації місцевого управління, однак більшість з них не мають стабільної підтримки або не відповідають українському законодавству, зокрема щодо персональних даних.

1.4.4 Висновки з аналізу. Проведений аналіз дозволяє зробити кілька важливих висновків:

1) Існуючі державні системи мають обмежену функціональність у частині локального управління громадою або вимагають складної інтеграції.
2) Комерційні рішення є занадто дорогими або складними для впровадження у громадах, які мають обмежені фінансові та технічні ресурси.
3) Необхідність розробки власного кастомізованого рішення виправдана тим, що воно дозволяє:
   - максимально врахувати особливості конкретної громади;
   - реалізувати просту, гнучку та зрозумілу логіку взаємодії користувача із системою;
   - забезпечити підтримку української мови, локальних стандартів і законодавчих вимог.

Таким чином, розробка спеціалізованого програмного додатку є актуальною, доцільною та обґрунтованою з практичної точки зору.

1.5 Постановка завдання

1.5.1 Загальна характеристика задачі. Інформаційне середовище сучасних територіальних громад потребує ефективних засобів для ведення обліку адміністративно-територіальних одиниць, мешканців, обробки звернень громадян та формування відповідної звітності. У більшості громад така робота виконується вручну або із використанням застарілих програмних засобів, що призводить до дублювання даних, труднощів при обробці великого обсягу інформації, відсутності єдиної системи звітності, а також значних витрат часу.

Саме тому постає необхідність у створенні сучасного програмного засобу, який забезпечить централізоване зберігання, обробку та аналіз інформації про мешканців, адміністративні одиниці та подані заяви.

1.5.2 Мета проєкту. Метою даної роботи є розробка програмного додатку для територіальної громади, який дозволяє:

- вести облік області, районів, населених пунктів, вулиць, будинків і мешканців;
- здійснювати реєстрацію та обробку заяв мешканців;
- змінювати статуси заяв у процесі обробки;
- формувати звіти за вибраними критеріями (район, тип заяви, статус тощо);
- забезпечити рольовий доступ (оператор, адміністратор).

1.5.3 Функціональні вимоги. Функціональні вимоги визначають дії, які система має бути здатна виконувати. У таблиці нижче наведено функціональні системи.

Таблиця 1.1 Функціональні вимоги

Номер	Назва функції	Опис функціоналу
1	Авторизація користувачів	Користувач має змогу ввести логін і пароль для доступу до системи. Права доступу визначаються відповідно до ролі.
2	Внесення жителів	Дозволяє оператору додати нового жителя із прив’язкою до будинку, вулиці, населеного пункту, району та області.
3	Облік територіальних одиниць	Додавання інформації про області, райони, населені пункти, вулиці та будинки.
4	Подання заяв	Додавання нової заяви від конкретного мешканця з вибором категорії та автоматичним присвоєнням статусу «Нова».
5	Обробка заяв	Зміна статусу заяви («В обробці, «Завершено», «Відхилено»)
6	Генерація звітів	Створення відфільтрованих звітів(наприклад: «Усі нові заяви за обраним районом», «Усі виконані заяви у певному районі»)

1.5.4 Нефункціональні вимоги. Нефункціональні вимоги визначають якість роботи системи, умови експлуатації, продуктивність тощо.

Таблиця 1.2 Нефункціональні вимоги

Номер	Вимога	Опис
1	Зручність інтерфейсу	Інтерфейс має бути зрозумілим, із чітким поділом на функціональні блоки.
2	Продуктивність	Система повинна обробляти запити до бази даних у межах кількох секунд навіть за великого обсягу даних
3	Безпека	Авторизація через логін/пароль. Захист від SQL-ін’єкцій. Користувач бачить лише ті дані які дозволено його роллю.
4	Розширюваність	Архітектура має дозволяти додавання нових довідників або типів заяв без зміни основного коду.
5	Сумісність	Система має коректно працювати на ОС Windows 10+ з .Net Framework.
6	Надійність	Забезпечення стабільної роботи без аварій у випадку помилки користувача.
7	Мова інтерфейсу	Програмний додаток має підтримувати українську мову для всіх елементів UI, повідомлень та форм

1.5.5 Вимоги до середовища розробки. Для розробки додатку планується використання таких інструментів:

- Мова програмування: C (.NET WinForms);

- Система управління базою даних: PostgreSQL;
- Засіб звітності: RDLC (ReportViewer);
- Середовище розробки: Visual Studio 2022;
- Система керування версіями: Git (локально або GitHub);
- Бібліотеки доступу до бази даних: Npgsql для PostgreSQL.

1.5.6 Обмеження. Система, що буде створена також має певні обмеження у зв’язку з описаною вище предметною областю, а саме:

- Система не розрахована на роботу в браузері – лише десктопна версія.
- Підтримується лише одна мова інтерфейсу (українська)[11].

1.6 Висновки по першому розділу

У першому розділі було розглянуто основні теоретичні питання, які стосуються розробки інформаційної системи для обліку даних територіальної громади. Насамперед була показана актуальність теми. Сьогодні багато процесів, пов’язаних з адміністративним обліком, досі виконуються вручну або в застарілих системах. Це ускладнює зберігання інформації, її оновлення, пошук та обробку. Саме тому створення сучасного програмного додатку, який дозволить автоматизувати облік мешканців, будинків, заяв та інших об’єктів громади, є важливим і своєчасним завданням.

Далі було описано предметну область, у якій буде працювати система. Було визначено, які саме об’єкти мають бути враховані в програмі: області, райони, населені пункти, вулиці, будинки, мешканці, а також подані ними заяви. Для кожного з цих об’єктів були описані зв’язки та залежності, що дозволяє краще зрозуміти структуру даних і побудувати на її основі логіку майбутньої системи.

Важливим етапом стало моделювання системи, яке виконувалось за допомогою UML-діаграм. Було розроблено:

- Діаграму прецедентів (use case) – вона показує, які основні дії можуть виконувати користувачі системи (наприклад, додавання мешканця або подання заяви);
- Діаграму активності (activity diagram) – на ній зображено етапи обробки заяви, тобто як система переходить з одного кроку до іншого;
- Діаграму послідовності (sequence diagram) – яка показує, як саме взаємодіють об'єкти системи в певних сценаріях.

Таке візуальне представлення дуже корисне, оскільки дозволяє на ранньому етапі побачити, як працюватиме система, де можуть бути помилки або незрозумілі моменти.

Крім того, у розділі було проведено короткий огляд вже існуючих рішень, що дозволило порівняти функціонал майбутнього додатку з тими системами, які вже існують. Це дає змогу врахувати сильні та слабкі сторони конкурентів і розробити зручніший та ефективніший продукт.

У кінці розділу було сформульовано постановку завдання для майбутньої системи. Було визначено, які функції має виконувати програма, а також які є технічні та нефункціональні вимоги. Усе це створює міцну основу для переходу до наступного розділу, де буде здійснено безпосереднє проєктування системи, розробка структури бази даних, інтерфейсів користувача та основної логіки роботи програми.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

На етапі проєктування інформаційного забезпечення одним з найважливіших завдань є побудова коректної моделі даних, яка забезпечить зберігання, обробку та доступ до інформації у майбутній системі. Для досягнення цієї мети використовується ER-діаграма (Entity-Relationship diagram), що дозволяє візуально представити логічну структуру бази даних та взаємозв’язки між основними об’єктами предметної області.

У межах даної інформаційної системи, призначеної для автоматизованого обліку адміністративно-територіальних одиниць, мешканців та їхніх заяв, мною було виділено такі основні сутності:

- Region (Область) – одиниця найвищого рівня адміністративного поділу.
- District (Район) – підпорядкована певній області.
- Settlement (Населений пункт) – належить до району.
- Street (Вулиця) – знаходиться в межах населеного пункту.
- House (Будинок) – об’єкт житлової інфраструктури, прив’язаний до певної вулиці.
- House_Type (Тип будинку) – класифікація будинків за призначенням або типом.
- Citizen (Мешканець) – особа, яка проживає у будинку.
- Statement (Заява) – документ, який подає мешканець для отримання певної послуги або дії.
- Statement_Status (Статус заяви) – етап обробки заяви: нова, в обробці, виконано тощо.

- Category (Категорія заяви) – тематична класифікація звернення (довідка, скарга, запит тощо).

2.1.1 Загальний опис взаємозв’язків між сутностями:

- Кожна область може містити декілька районів, але кожен район належить лише одній області.
- Кожен район містить декілька населених пунктів, але населений пункт прив’язаний тільки до одного району.
- Кожен населений пункт має декілька вулиць, однак кожна вулиця пов’язана лише з одним населеним пунктом.
- Будинок належить певній вулиці і має вказаний тип з таблиці House_Type.
- Мешканці проживають у будинках, тобто кожен мешканець має зовнішній ключ на будинок.
- Заяви подаються мешканцями, мають прив’язку до категорії і статусу.

Таке моделювання забезпечує логічно послідовну структуру бази даних, яка чітко розділяє сутності й мінімізує дублювання даних.

Таблиця 2.1 Структура таблиць бази даних

Таблиця	**Основні поля**
Region	Region_ID, Region_name
District	District_ID, District_name, Region_ID
Settlement	Settlement_ID, Settlement_name, District_ID
Street	Street_ID, Street_name, Settlement_ID

House	House_ID, Street_ID, Type_ID
House_Type	Type_ID, Type_name
Citizen	ID_Citizen, First_name, Last_name, Patronymic, House_ID
Statement	Statement_ID, ID_Citizen, Category_ID, Status_ID, Create_Time
Statement_status	Status_ID, name
Category	Category_ID, Category_name

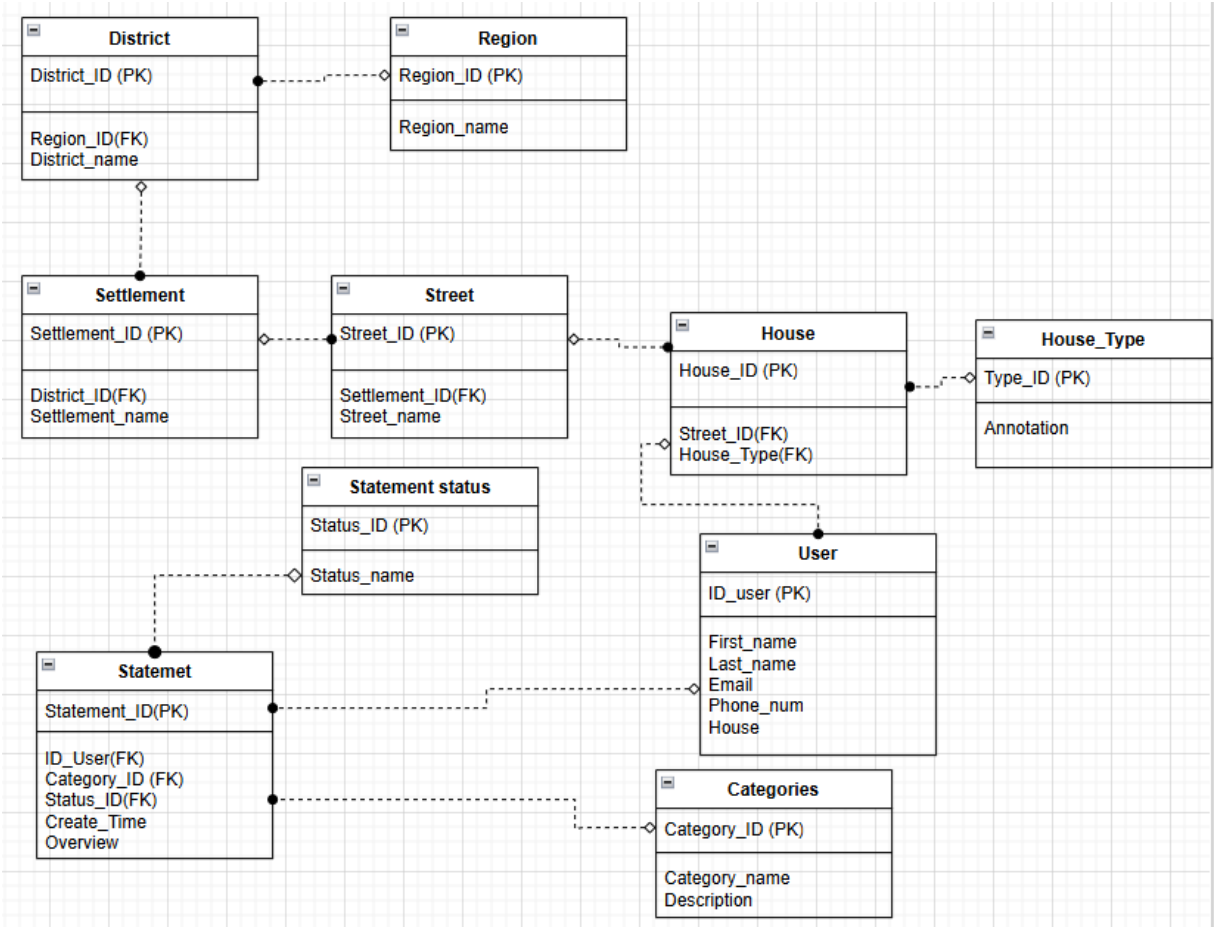

Рис. 2.1 – ER-діаграма розробленої бази даних

2.1.2 Пояснення ER-діаграми. У поданій ER-діаграмі (вставляється користувачем), кожна сутність відображається як прямокутник, а зв’язки між ними – у вигляді ліній зі зазначенням кратності (1:1, 1:N). Наприклад:

- Зв’язок "Область має райони" відображено як 1:N;
- Зв’язок "Мешканець подає заяви" – також 1:N;
- Зв’язок між заявою і категорією, статусом – 1:N, де категорія та статус є довідниками.

Всі поля, які є зовнішніми ключами, мають чітко визначену спрямованість та назви, що відповідають принципам логіки даних.

2.1.3 Відповідність третьої нормальної формі (3НФ). Побудована модель даних відповідає третій нормальній формі (3НФ), що підтверджується такими критеріями:

1. Перша нормальна форма (1НФ): Усі атрибути в таблицях атомарні, тобто кожне поле зберігає лише одне значення. Відсутні повторювані групи даних або списки у вигляді одного поля.
2. Друга нормальна форма (2НФ): Усі неключові атрибути повністю функціонально залежні від первинного ключа. Усі таблиці не містять часткових залежностей.
3. Третя нормальна форма (3НФ): Відсутні транзитивні залежності між неключовими атрибутами. Усі атрибути описують лише ключову сутність, до якої вони належать. Наприклад, у таблиці Citizen атрибути стосуються лише особи, а не будинку чи району.

Таким чином, модель не має надлишкових залежностей, дублювання інформації або аномалій при оновленні, що дозволяє ефективно використовувати її в реальній системі [6] [8].

2.2 Діаграма класів та кооперацій

Для побудови якісної інформаційної системи недостатньо лише зібрати вимоги до її функціонування. Надзвичайно важливо також спроєктувати архітектуру програмного забезпечення: визначити основні класи (об’єкти), їхні атрибути, методи та взаємозв’язки. UML-діаграми дозволяють зробити це наочно, зручно та структуровано.

У цьому підпункті розглянемо дві важливі моделі: діаграму класів, яка описує структуру системи, та діаграму кооперацій, що демонструє, як об’єкти взаємодіють під час виконання певної задачі.

2.2.1 Діаграма класів. На діаграмі класів зображено основні об’єкти, які беруть участь у процесі подання та обробки заяв мешканців громади. Кожен клас має свої атрибути – характеристики, які визначають його сутність, – а також методи, тобто функції, які він виконує. Також на діаграмі показано асоціації між класами, які демонструють логіку їх зв’язків.

Основні класи:

1) Житель (Citizen). Зберігає інформацію про ПІБ, адресу та контактні дані мешканця. Має метод подати заяву(), що ініціює створення нової заявки.
2) Заява (Statement). Один з ключових класів, що містить номер заявки, дату подання, статус, опис. Має методи:
    - Встановити категорію()
    - Отримати деталі()
    - Отримати статус()

3) Оператор Адміністрації (AdminOperator). Працівник, який приймає заяви від мешканців і вносить їх у систему. Має методи:
    - Прийняти заяву()
    - Внести заяву до системи()
4) Секретар Адміністрації (AdminSecretary). Відповідальний за опрацювання заяв, зміну їхнього статусу. Методи:
    - Обробити заяву()
    - Отримати статус()
5) Категорія (Category). Містить назву та опис категорії заяв. Забезпечує методи:
    - Додати нову категорію()
    - Оновити опис()

Зв’язки між класами:

- Житель подає 1 або більше заяв.
- Кожну заяву приймає Оператор Адміністрації (1 або більше).
- Секретар Адміністрації обробляє заяви (може бути прив’язаний до кількох заяв).
- Кожна заява належить до однієї категорії (зв’язок із класом Категорія).

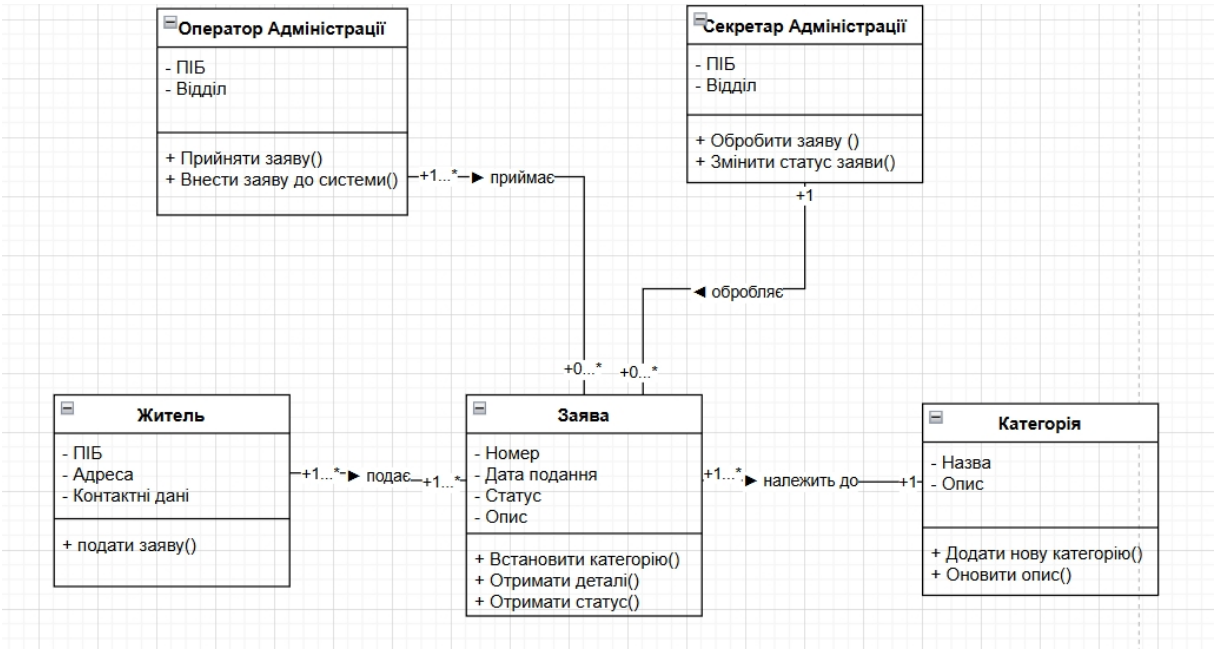

Рис. 2.2 – Діаграма класів системи обробки заяв у територіальній громаді

Ця діаграма дозволяє детально описати структуру системи на об'єктному рівні. Завдяки їй можна легко уявити, як виглядають класи в програмному коді, які мають поля (атрибути), і які методи реалізуються для обробки даних [12].

2.2.2 Діаграма кооперацій.

У той час як діаграма класів описує статичну структуру системи, діаграма кооперацій дозволяє показати, як об'єкти взаємодіють у рамках конкретного процесу, обмінюючись повідомленнями. Вона корисна для моделювання поведінки системи в реальному часі.

Для створюваного мною програмного комплексу було розроблено три окремі кооперації, що охоплюють повний цикл обробки звернення мешканця:

- Подання заяви мешканцем
- Прийом заяви працівником адміністрації
- Обробка заяви секретарем адміністрації

1. Подання заяви мешканцем

- У цьому сценарії задіяні такі об'єкти:
- Житель (ініціатор процесу)
- Заява, яка створюється
- Категорія, що визначає тип звернення

Процес взаємодії: Житель викликає метод подати заяву().

В результаті створюється об'єкт Заява з відповідними атрибутами.

Об’єкт Заява зв’язується з певною Категорією, яка визначає напрямок розгляду.

Категорія надається за допомогою методу належить до.

2. Прийом заяви адміністрацією

Цей процес відображає взаємодію працівника адміністрації з системою при прийомі звернення.

Учасники:

- Працівник Адміністрації
- Заява

Процес:

- Працівник адміністрації приймає заяву за допомогою методу прийняти заяву().
- Далі вносить її до системи, використовуючи метод внести заяву до системи().

Таким чином, відбувається первинна реєстрація звернення в системі, що фіксується в базі даних.

3. Обробка заяви адміністрацією

Останній етап охоплює роботу секретаря адміністрації, який відповідальний за опрацювання звернень.

Об'єкти:

- Секретар Адміністрації
- Заява

Процес:

- Секретар отримує звернення та обробляє його, викликаючи метод обробити заяву().
- Після цього змінює її статус через метод змінити статус заяви() – наприклад, з "Нова" на "В обробці".

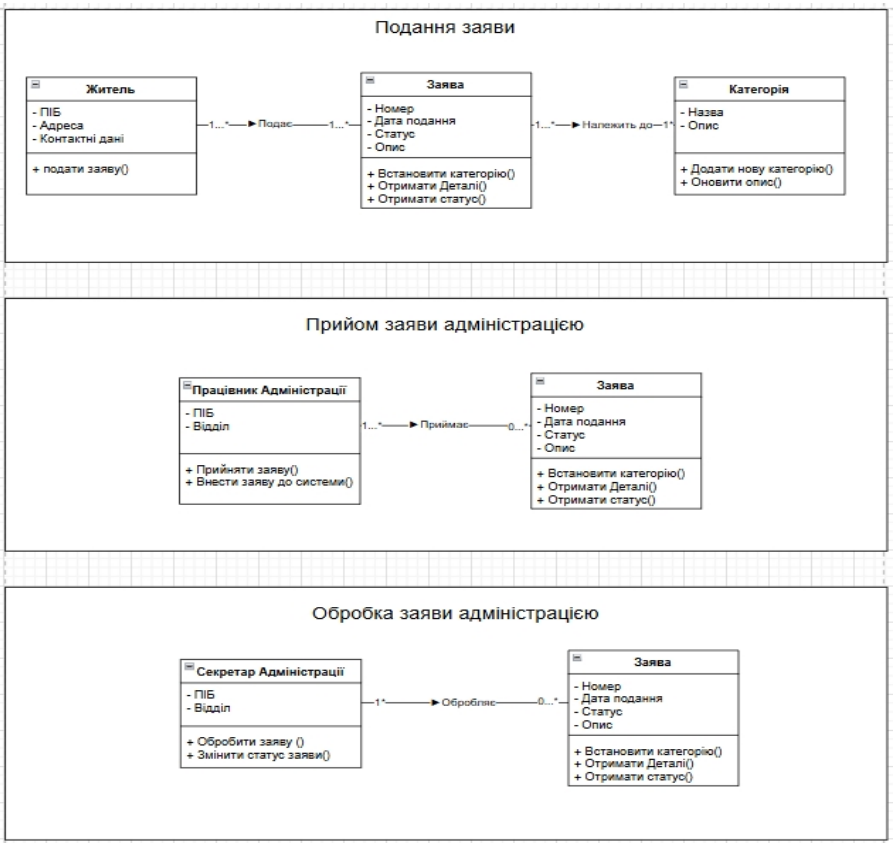

Рис. 2.3 – Діаграма простих кооперацій системи територіальної громади

2.3 Діаграма пакетів

У процесі проєктування складних інформаційних систем важливо не лише розуміти взаємодію окремих класів, а й мати загальне уявлення про логічну структуру всієї програми. Саме для цього використовуються діаграми пакетів (package diagrams), які є частиною стандарту UML і дозволяють візуально відобразити модульну архітектуру програмного продукту.

Діаграма пакетів ілюструє групування класів у логічні пакети, а також залежності між цими пакетами. Це дає змогу:

- сформувати чітке уявлення про ієрархію підсистем;
- виділити функціональні області (UI, бізнес-логіка, доступ до даних тощо);
- спростити підтримку та масштабування системи;
- визначити межі відповідальності кожного модуля;
- забезпечити слабке зв’язування між компонентами (loose coupling).

2.3.1 Загальна структура. Для розробленої системи створено діаграму, яка складається з чотирьох основних логічних блоків (пакетів):

1) Інтерфейс користувача (UI)
2) Логіка застосунку (Application Logic)
3) Доступ до даних (Data Access)
4) База даних (Database)

Кожен з цих пакетів містить відповідні класи, які реалізують певні функціональні можливості системи.

2.3.2 Опис компонентів

1. Пакет “Інтерфейс користувача”

Цей пакет відповідає за взаємодію системи з кінцевим користувачем. Він містить елементи графічного інтерфейсу та обробники подій.

Класи:

- ГоловнеВікно – основне вікно застосунку, з якого користувач має доступ до ключових функцій (наприклад, подання нової заяви, перегляд статусів, вхід/вихід).
- ФормаПоданняЗаяви – спеціалізована форма, яка дозволяє мешканцям створювати і подавати заяви до адміністрації. Тут реалізовано валідацію введених даних та ініціацію запитів до логіки застосунку.
- Пакет має залежність від пакету "Логіка застосунку", оскільки викликає методи обробки заяв або управління користувачами.

2. Пакет “Логіка застосунку”

Цей модуль містить основні сервіси, що реалізують бізнес-логіку системи – тобто ту частину, яка відповідає за обробку подій, перевірку прав доступу, зміну статусів, облік та інші алгоритми.

Класи:

- ОбробкаЗаяв – реалізує логіку створення, обробки, перевірки статусу та маршрутизації заяв.
- УправлінняКористувачами – відповідає за автентифікацію, авторизацію користувачів, управління ролями (мешканець, працівник, секретар), облік історії активності.

Пакет звертається до "Доступу до даних", використовуючи репозиторії для збереження або читання даних.

3. Пакет “Доступ до даних”

Цей модуль реалізує шаблон Data Access Object (DAO) або Repository, ізольовуючи прикладну логіку від специфіки роботи з базою даних. Завдяки цьому вдається легко змінювати джерело зберігання (наприклад, перейти з PostgreSQL на SQLite) без зміни основного коду логіки.

Класи:

- РепозиторійЗаяв – виконує операції читання/запису об’єктів Заява, включно з фільтрами (за категорією, статусом тощо).
- РепозиторійКористувачів – надає доступ до даних про мешканців, співробітників, їх ролі та облікові записи.

Пакет взаємодіє з модулем "База даних", де фактично зберігається інформація.

4. Пакет “База даних”

У цій діаграмі база даних представлена як окремий абстрактний блок. Хоча конкретні класи не зазначені (оскільки мова йде про сторонній компонент – система управління базою даних), саме тут фізично зберігаються таблиці, зв’язки, індекси та вся інформація, якою оперує система.

Взаємозв’язки між пакетами

Діаграма передбачає чітку ієрархічну структуру, де:

- Інтерфейс користувача використовує Логіку застосунку, що логічно, адже дії користувача ініціюють виконання бізнес-операцій.
- Логіка застосунку звертається до Доступу до даних, делегуючи йому збереження або отримання інформації.

- Доступ до даних, своєю чергою, взаємодіє з Базою даних, відправляючи SQL-запити або інші типи транзакцій.

Такий підхід дозволяє забезпечити принцип розділення обов’язків (Separation of Concerns) і високу модульність, що є базовими принципами сучасної розробки програмного забезпечення.

2.3.3 Візуалізація

На рисунку нижче зображено відповідну діаграму пакетів системи територіальної громади, яка демонструє логічну структуру системи, а також зв’язки між її ключовими компонентами:

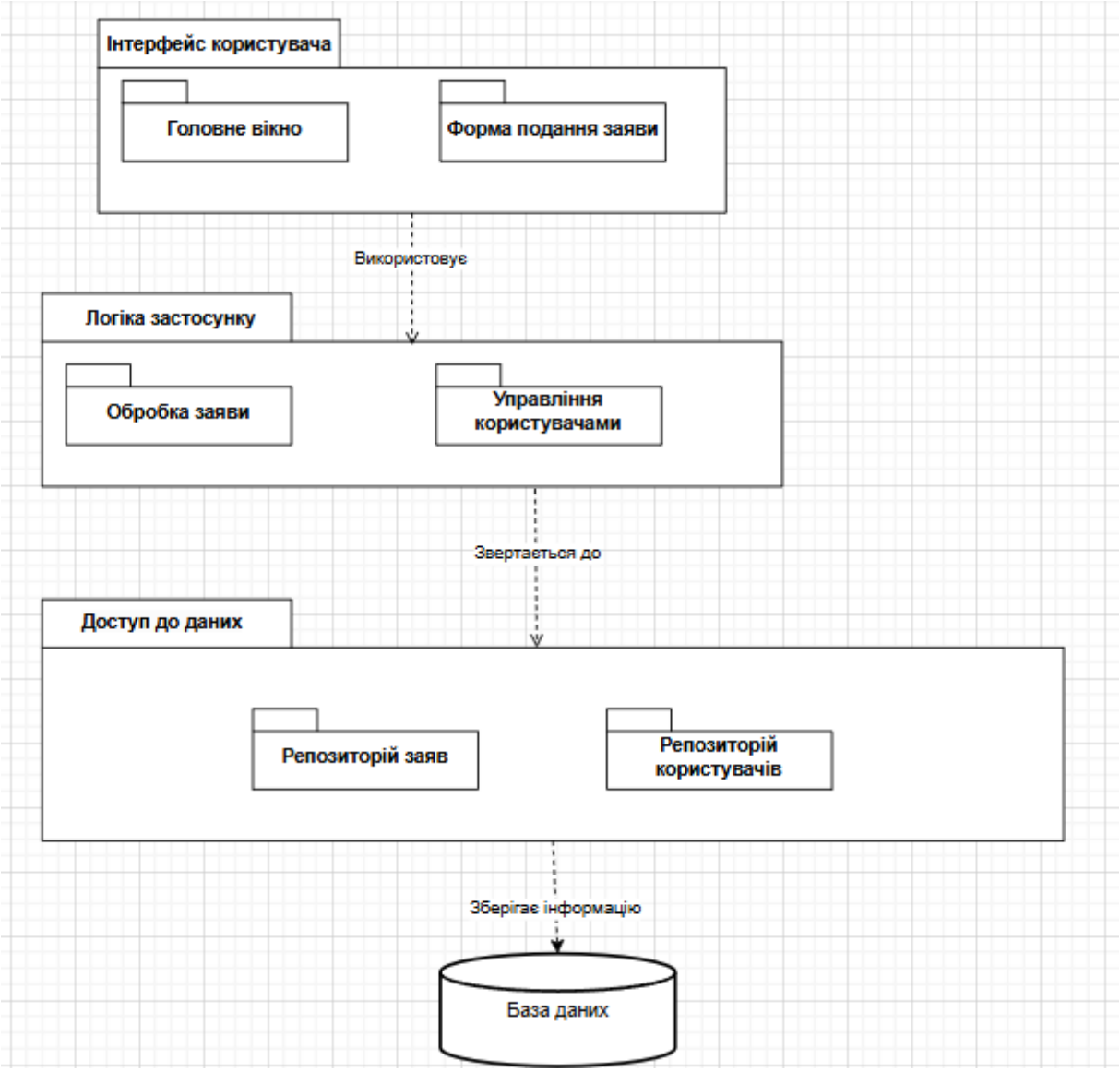

Рис. 2.4 – Діаграма пакетів системи територіальної громади

Отже, діаграма пакетів дозволяє розробнику легко орієнтуватися в загальній структурі системи, розуміти зони відповідальності різних модулів, оцінювати залежності між компонентами. Це особливо важливо на етапах:

- первинного проєктування архітектури;
- аналізу впливу змін;
- розподілу задач між розробниками;
- підготовки технічної документації.

Завдяки чіткому поділу системи на логічні пакети, можна швидко масштабувати проєкт, підключати нові компоненти або змінювати внутрішню реалізацію без порушення загальної структури.

2.4 Діаграма компонентів

У структурі інформаційної системи територіальної громади важливою є чітка модульність, яка забезпечує зрозумілий розподіл обов'язків між функціональними блоками, спрощує підтримку, тестування, а також дає змогу розширювати функціонал у майбутньому. Одним з ключових способів представлення архітектури системи є діаграма компонентів, що дозволяє візуалізувати взаємозв'язки між програмними компонентами системи.

2.4.1 Структура та взаємодія компонентів

Діаграма включає такі ключові компоненти:

- Інтерфейс користувача – основна точка взаємодії між користувачем (мешканцем або працівником громади) та системою. Через нього здійснюється доступ до функціоналу системи, зокрема – введення даних, перегляд статусу заяв, формування звітів.
- Система авторизації – відповідає за перевірку облікових даних користувача. Після успішної аутентифікації користувач отримує доступ до інших модулів.
- Логіка обробки даних – центральний обробник, що забезпечує координацію всіх запитів, їхню маршрутизацію до відповідних підсистем, а також контроль за послідовністю виконання бізнес-процесів.
- Керування заявками – обробка звернень мешканців, реєстрація нових заяв, зміна статусів та взаємодія з базою даних.

- Управління користувачами – дозволяє адміністраторам додавати нових користувачів, редагувати існуючих, призначати ролі та права доступу.
- Управління категоріями – підтримує класифікацію заяв за типами, що полегшує аналіз та обробку звернень.
- Модуль логування – реєструє всі важливі події, зміни, дії користувачів для забезпечення прозорості та безпеки.
- Модуль звітності – формує підсумкові документи, статистику, аналітичні звіти на основі даних, збережених у системі.
- База даних – централізоване сховище інформації, що містить записи про користувачів, заявки, категорії, логи та інші важливі об'єкти.

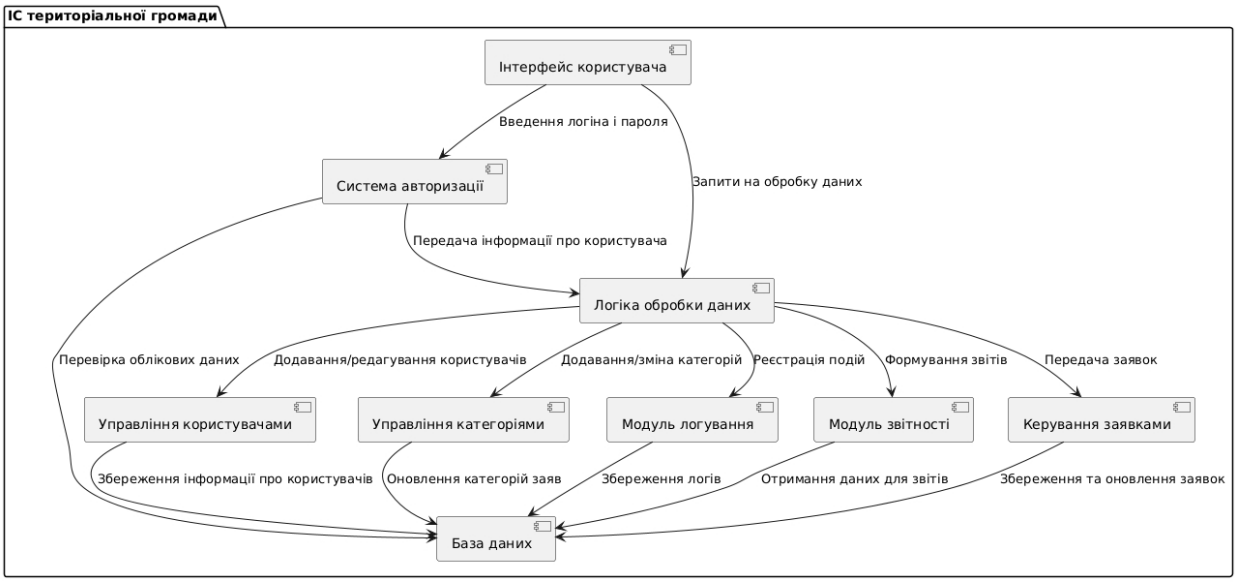

Рис.2.5 Діаграма компонентів системи територіальної громади

2.4.2 Аналіз діаграми компонентів

Ця діаграма дозволяє відстежити основні шляхи взаємодії між компонентами. Вона вказує, що:

- усі ключові операції проходять через логіку обробки даних, яка виступає ядром системи;

- інтерфейс користувача напряму не взаємодіє з базою даних, що відповідає принципу безпеки та розділення обов’язків;
- система авторизації виконує лише початкову перевірку, а далі – передає контроль основному логічному модулю;
- звітність і логування реалізовані як окремі компоненти, що можна масштабувати незалежно від основної логіки;
- база даних є єдиним сховищем для всієї системи, до якого мають доступ лише серверні компоненти.

2.4.4 Переваги архітектурного підходу

Модульність: кожен компонент реалізує обмежений набір функцій, що знижує складність системи.

Розширюваність: у майбутньому можна легко додати нові модулі (наприклад, API для зовнішніх сервісів).

Безпека: чіткий контроль доступу до бази даних лише через внутрішні компоненти.

Зручність супроводу: у разі помилки в одному модулі – не порушується робота всієї системи.

2.5 Висновки по другому розділу

У даному розділі було проведено детальний аналіз архітектури інформаційної системи територіальної громади. Розглянуто основні функціональні можливості системи, які охоплюють реєстрацію мешканців, обробку заяв, управління доступом, ведення категорій звернень, формування звітів і логування подій. Це дозволяє зробити висновок, що система покликана

не лише автоматизувати адміністративні процеси, а й забезпечити контроль, прозорість і зручність у взаємодії між громадянами та органами місцевого самоврядування.

Окрему увагу було приділено побудові діаграми компонентів, яка відображає логічну структуру взаємодії між ключовими модулями системи. Визначено центральні елементи – інтерфейс користувача, модуль авторизації, логіка обробки даних, управління заявами та користувачами, база даних тощо. Дана структура відповідає вимогам сучасних підходів до проєктування програмного забезпечення, таких як модульність, масштабованість і чіткий розподіл відповідальностей.

Представлена архітектура створює підґрунтя для ефективної реалізації системи як з технічного, так і з організаційного погляду. Завдяки зрозумілій структурі компонентів можна забезпечити швидке впровадження, спрощену підтримку та можливість подальшого розширення функціоналу. Такий підхід дозволяє будувати надійну та адаптивну інформаційну систему, здатну задовольнити потреби як місцевої влади, так і мешканців громади.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

3.1.1 Загальні положення

Одним з основних компонентів будь-якої інформаційної системи є система управління базою даних, яка забезпечує збереження, обробку, пошук та оновлення великого обсягу інформації. Для реалізації інформаційної системи територіальної громади в цьому проєкті було обрано PostgreSQL – потужну об’єктно-реляційну систему управління бази даних з відкритим вихідним кодом.

PostgreSQL є надійним, масштабованим та безпечним рішенням, яке підтримує всі основні функції, необхідні для побудови сучасної інформаційної системи. Його активне використання в державному та комерційному секторі підтверджує його стабільність і відповідність сучасним вимогам до зберігання даних.

3.1.2 Причини вибору PostgreSQL

Вибір саме PostgreSQL зумовлений низкою технічних і практичних переваг, а саме:

- Відкритий вихідний код – дозволяє безкоштовно використовувати систему управління базою даних без ліцензійних обмежень, що особливо важливо для муніципальних проєктів.
- Підтримка транзакцій – забезпечує цілісність і послідовність даних навіть у разі збоїв.

- Розширені можливості – підтримка збережених процедур, тригерів, JSON-структур, повнотекстового пошуку та реплікації.
- Безпека – підтримка аутентифікації, шифрування, контролю доступу до таблиць і представлень.
- Масштабованість – PostgreSQL ефективно працює як із малими базами даних, так і з великими (мільйони записів, десятки гігабайтів і більше).

Ці характеристики роблять PostgreSQL оптимальним вибором для системи, яка повинна зберігати персональні дані мешканців, історію звернень, журнали подій тощо.

3.1.3 Основна структура бази даних

У рамках моєї системи база даних є ядром, яке об’єднує інформацію з різних підсистем. Основні таблиці бази включають:

- Users – зберігає інформацію про зареєстрованих користувачів системи (логін, пароль, роль, ПІБ, контактні дані).
- Roles – визначає типи доступу (адміністратор, працівник, мешканець).
- Statements – заявки громадян із полями: тип, опис, дата подачі, статус, виконавець.
- StatementCategories – категорії звернень (соціальні, житлово-комунальні, земельні тощо).
- Regions, Districts, Streets, Houses – довідники географічного поділу для точного визначення адрес.
- Residents – перелік мешканців із прив’язкою до адреси.
- Logs – таблиця для зберігання системних подій: вхід у систему, зміни даних, дії користувачів.
- Reports – збережені шаблони або результати сформованих звітів.

Ця структура забезпечує нормалізацію даних, зменшує дублювання інформації та дозволяє швидко формувати вибірки за різними параметрами.

3.1.4 Забезпечення цілісності та продуктивності

PostgreSQL надає потужні засоби для забезпечення цілісності даних:

- Зовнішні ключі (foreign keys) – гарантують зв'язок між таблицями (наприклад, між Residents і Houses).
- Обмеження (constraints) – наприклад, NOT NULL, UNIQUE, CHECK для валідації введених даних.
- Індекси – використовуються для прискорення пошуку, особливо в таблицях із великою кількістю записів (наприклад, Statements).
- Тригери – автоматизують оновлення пов'язаних даних (наприклад, автоматичне оновлення статусу заявки після перегляду).

Крім того, PostgreSQL підтримує журнали транзакцій (Write-Ahead Logging), які дозволяють швидко відновити систему після аварій, зберігаючи послідовність змін.

3.1.5 Взаємодія з прикладним програмним забезпеченням

Система територіальної громади розробляється з використанням мови програмування (наприклад, C# або Java), яка взаємодіє з PostgreSQL через спеціальні бібліотеки (наприклад, Npgsql для .NET). Це дозволяє:

- Створювати запити до бази (SQL SELECT, INSERT, UPDATE, DELETE).
- Використовувати параметризовані запити для захисту від SQL-ін’єкцій.
- Працювати з транзакціями на рівні коду.

- Підключатися до бази з різних модулів (модуль авторизації, звітності тощо).

PostgreSQL виступає як надійна й продуктивна система управління базою даних для реалізації системи. Завдяки широкому набору функцій, вона дозволяє зберігати велику кількість даних, забезпечує гнучку структуру з високим рівнем безпеки та може масштабуватись відповідно до зростаючих потреб громади. Вибір цієї системи створює фундамент для стабільної роботи всієї інформаційної платформи [4].

3.2 Розробка інформаційної бази

Проєктування та реалізація інформаційної бази є одним із ключових етапів розробки будь-якої інформаційної системи. У контексті моєї системи інформаційна база виконує функцію централізованого сховища, що містить дані про користувачів, адреси, типи будинків, категорії звернень, статуси заяв, а також самі заяви.

Створення інформаційної бази надає нам можливості деякі з яких:

- забезпечення структурованого та цілісного зберігання всіх необхідних даних;
- підтримку цілісності, узгодженості та достовірності інформації;
- надає ефективного доступу до даних для подальшої обробки;
- дає можливість розширення бази без порушення наявної структури.

3.2.1 Структура бази даних

Інформаційна база спроєктована з урахуванням реляційного підходу, який забезпечує логічне представлення сутностей та їх взаємозв’язків. Загальна

модель побудована відповідно до нормалізованої схеми, що дозволяє уникнути надмірності даних і забезпечити ефективність при виконанні запитів.

База даних включає наступні групи таблиць:

- Адміністративно-територіальна структура
- Адресні об’єкти
- Користувачі
- Категорії та статуси
- Заяви

Кожна з цих груп представлена окремими таблицями, об’єднаними за змістом та функціональністю.

Таблиця 3.1 Розподіл таблиць бази даних

Структура	Таблиця	Пояснення
Адміністративно-територіальна	Region	Зберігає інформацію про області. Первинний ключ використовується для зв’язку з нижчими рівнями.
	District	Містить назви районів. Має зовнішній ключ Region_ID, що зв’язує район із відповідною областю.
	Settlement	Описує населені пункти. Зовнішній ключ District_ID вказує, до якого району належить населений пункт.
Адресна	Street	Зберігає назви вулиць. Кожна вулиця пов’язана з населеним пунктом через Settlement_ID.

Структура	Таблиця	Пояснення
	House_Type	Довідкова таблиця, що містить типи будинків (приватний, квартира).
	House	Визначає конкретні будинки за адресою. Поля Street_ID і House_Type пов’язують будинок із вулицею та типом.
Жителі	Citizen	Містить дані про мешканців: ім’я, прізвище, по батькові, email, номер телефону та адресу проживання (через House_ID). Ключ ID_user є унікальним ідентифікатором жителя, що також використовується в таблиці заяв.
Категорії та статуси	Categories	Містить категорії заяв (наприклад, освітлення, благоустрій, транспорт). Є реєстром типів звернень з описом.
	Statement Status	Довідник статусів заяв: «Нова», «В обробці» тощо.
Заяви	Statement	Центральна таблиця, зберігає всі подані заяви. До неї входять ключові поля з таблиць Statement_Status, Categories та Citizen, для точного розуміння про що ця заява та від кого надійшла

Як вказано у таблиці вище, розроблена база даних розділена на структури, вони у свою чергу дають нам краще розуміння, як розподілені таблиці. Пояснити

це можна так: Адміністративно-територіальний блок – описує ієрархічну структуру місцевості, що дозволяє нам точніше фіксувати місця проживання людей та виникнення проблем. Адресна структура ж, дає нам змогу максимально точно ідентифікувати місце проживання людини яка подає нам заяву.

3.2.2 Міжтабличні зв’язки

У структурі бази реалізовано численні зовнішні ключі (FOREIGN KEY), що дозволяють підтримувати референційну цілісність. Це гарантує, що, наприклад:

- не можна створити запис про район без наявної області;
- не можна додати мешканця без адреси;
- не можна створити заяву без відповідної категорії та користувача.

Таким чином, логіка структури є послідовною та цілісною.

3.2.2 SQL-структура бази

На рисунках нижче представлено SQL-код створення таблиць, що реалізує описану вище структуру.

```sql
CREATE TABLE Region (
    Region_ID CHAR(10) PRIMARY KEY,
    Region_name VARCHAR(255) NOT NULL
);

CREATE TABLE District (
    District_ID CHAR(10) PRIMARY KEY,
    Region_ID CHAR(10),
    District_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (Region_ID) REFERENCES Region(Region_ID)
);

CREATE TABLE Settlement (
    Settlement_ID CHAR(10) PRIMARY KEY,
    District_ID CHAR(10),
    Settlement_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (District_ID) REFERENCES District(District_ID)
);

CREATE TABLE Street (
    Street_ID CHAR(10) PRIMARY KEY,
    Settlement_ID CHAR(10),
    Street_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (Settlement_ID) REFERENCES Settlement(Settlement_ID)
);
```

Рис. 3.1 – SQL-код створення таблиць бази даних

Створена база враховує можливість:

- розширення довідників (типів будинків, категорій, статусів);
- модифікації рівнів адресної деталізації;
- обробки великої кількості заяв без втрати продуктивності;
- інтеграції з іншими системами (через ключі та чіткі зв’язки).

Усі поля мають відповідний тип, довжину, а також обмеження (NOT NULL, PRIMARY KEY, FOREIGN KEY), що забезпечує дотримання вимог до даних.

Структура розробленої інформаційної бази дозволяє ефективно організувати дані, що використовуються у системі управління заявами. Адже

вона побудована на принципах логічної зв’язності, нормалізації, розширюваності та узгодженості.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка прикладного програмного забезпечення передбачає не лише реалізацію функціональності, а й вибір ефективних засобів для створення, налагодження, тестування та подальшого обслуговування системи. У процесі створення інформаційної системи територіальної громади заявами було обрано перевірене програмне середовище розробки – Microsoft Visual Studio, а також фреймворк для створення десктопного інтерфейсу – Windows Forms (WinForms).

3.3.1 Обґрунтування вибору Microsoft Visual Studio

Visual Studio є потужним середовищем розробки, що підтримує багато мов програмування, має інтегровані засоби налагодження, моделювання, тестування, підтримує підключення до баз даних та роботу з системами контролю версій (наприклад, Git). Переваги, які зумовили вибір саме цього середовища:

- Інтеграція з .NET Framework і .NET Core, що забезпечує підтримку сучасних стандартів розробки.
- Зручний графічний редактор форм, що дозволяє швидко створювати інтерфейс користувача без необхідності вручну описувати розмітку.
- Наявність вбудованих засобів підключення до бази даних, що спрощує інтеграцію з PostgreSQL через сторонні бібліотеки (наприклад, Npgsql).
- Великий набір інструментів для рефакторингу, тестування та профілювання коду.

- Можливість створення модульної структури проєкту, з розділенням на класи, бібліотеки та окремі компоненти (UserControl).

Visual Studio забезпечує стабільну та зручну роботу як з мовою C, так і з іншими інструментами екосистеми Microsoft. Крім того, вона дозволяє ефективно налагоджувати застосунок, переглядати змінні під час виконання, працювати з винятками та діагностувати помилки.

3.3.2 Обґрунтування вибору Windows Forms

Для побудови графічного інтерфейсу було обрано Windows Forms (WinForms) – одну з найстаріших, але надійних технологій для створення настільних додатків на базі .NET. Незважаючи на появу більш сучасних рішень (таких як WPF або MAUI), вибір на користь WinForms був зумовлений наступними перевагами:

- Швидкість розробки: можливість створювати форми, розміщувати елементи керування (кнопки, поля, ComboBox, DataGridView тощо) шляхом перетягування.
- Простота інтеграції з базою даних: доступ до PostgreSQL реалізується через бібліотеку Npgsql, яка легко інтегрується в WinForms-проєкт.
- Підтримка модульності: використання власних UserControl дозволяє створити роздільні компоненти інтерфейсу для різних функціональних частин (додавання заяв, перегляд, фільтрація, формування звітів тощо).
- Стабільність та сумісність: WinForms чудово підходить для невеликих і середніх систем, що не потребують надмірної графічної складності, але мають багатий набір форм і операцій [1].

3.3.3 Інші використані інструменти та бібліотеки

Npgsql – драйвер для підключення до PostgreSQL з .NET-додатків. Забезпечує підтримку SQL-запитів, параметризованих команд, зчитування результатів та транзакцій.

RDLC (Report Definition Language Client-side) – засіб для створення та візуалізації звітів без необхідності зовнішнього сервера звітності. Застосовується для генерації PDF-звітів прямо в інтерфейсі користувача.

System.Windows.Forms.DataVisualization – бібліотека для створення графіків і діаграм у межах WinForms (за потреби).

Git – для контролю версій коду під час розробки.

Комбінація Microsoft Visual Studio та Windows Forms забезпечує швидку, надійну й ефективну розробку прикладного програмного забезпечення для настільних систем. Простота роботи з формами, доступ до бази даних, підтримка звітності та модульна структура дозволили реалізувати повноцінний функціонал управління заявами у зручному для користувача інтерфейсі .

3.4 Алгоритмізація та програмування програмних модулів

Під час розробки прикладного програмного забезпечення для системи територіальної громади було реалізовано низку функціональних модулів, кожен з яких відповідає за окремий аспект взаємодії користувача з системою та забезпечує зручність і надійність виконання операцій.

Для реалізації модулів використовувалась технологія Windows Forms, що дозволила створити окремі UserControl для кожної основної функції програми. Модулі поєднуються через головне вікно програми, яке забезпечує навігацію між ними.

3.4.1 Модульність системи

Модуль 1. Додавання регіонів та районів

Цей модуль дозволяє адміністраторам додавати нові регіони та райони до системи. Форма містить два ComboBox:

- Перший – для вибору або додавання нового регіону.
- Другий – для додавання району, який буде пов'язаний із вибраним регіоном.

Основна логіка модуля:

- Завантаження списку регіонів із таблиці Region.
- Після вибору регіону – збереження нового району в таблиці District із вказівкою зовнішнього ключа на Region_ID.
- Перевірка унікальності назви району в межах одного регіону.

Модуль 2. Додавання будинків

Цей модуль реалізує функцію додавання будинків у систему з прив’язкою до вулиці, населеного пункту, району та області.

Особливості:

- Каскадна фільтрація: після вибору області – фільтруються райони, далі населені пункти, потім вулиці.
- Вибір типу будинку з таблиці House_Type.
- Після заповнення форми новий запис додається в таблицю House.

Логіка взаємозв’язку:

Region – District – Settlement – Street – House

Модуль 3. Додавання мешканців (громадян)

Цей модуль відповідає за реєстрацію мешканців, які можуть подавати заяви. Дані, що вносяться:

- Ім’я, прізвище, по батькові.
- Email, телефон.
- Прив’язка до конкретного будинку.

Функціонал модуля:

- Всі поля є обов’язковими, крім Email.
- Після вибору будинку – мешканець зберігається в таблиці Citizen.
- Додавання унікального ID користувача(В нашому випадку це РНОКПП).

Модуль 4. Подання заяви

Ключовий функціональний модуль для користувача. Дає змогу створювати нову заяву зі вказівкою:

- Категорії заяви (із таблиці Categories).
- Опису (overview).
- Статусу, який за замовчуванням встановлюється як "Нова".

Особливості:

- Після подання зберігається в таблиці Statement.
- Фіксується час створення (поле Create_Time).
- Після надсилання користувач бачить підтвердження.

Модуль 5. Перегляд і зміна статусу заяв

Модуль призначений для адміністраторів. Дозволяє:

- Переглядати список заяв з будь-яким статусом (через уявлення vw_Statements_Info).

- Змінювати статус обраної заяви на "В обробці", "Завершена", тощо.

Технічна реалізація:

- ComboBox зі статусами (дані з Statement_status).
- DataGridView із фільтрацією за районом і регіоном.
- Кнопка “Оновити статус” змінює значення поля Status_ID у таблиці Statement.

Модуль 6. Генерація звітів

Для формування звітів у форматі PDF використовується ReportViewer із підтримкою RDLC-звітів. Користувач обирає:

- Область.
- Район.
- Звіт генерується на основі вмісту представлення vw_Statements_Info.

Можливості:

- Перегляд на екрані.
- Збереження у PDF-файл.

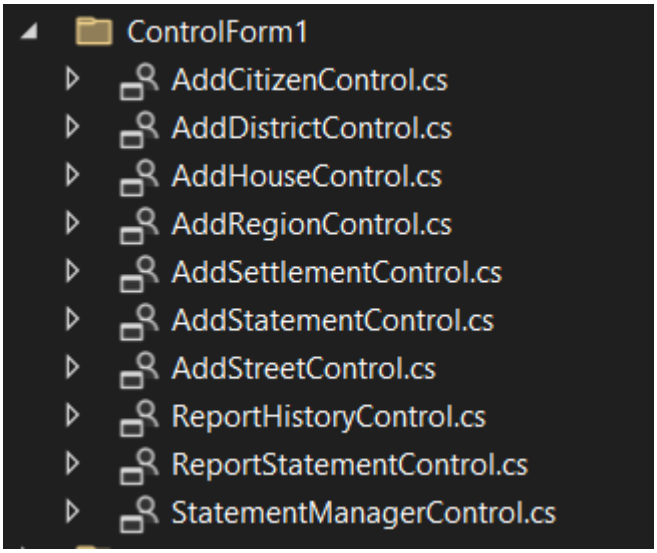

Рис. 3.2 – Структурні модулю застосунку

3.4.2 Логічна структура модулів

Усі модулі побудовані на основі єдиної архітектури з чітким розділенням шарів:

- Презентаційний шар: WinForms і UserControls.
- Логічний шар: методи, які виконують обробку даних, перевірку, фільтрацію.
- Доступ до даних: реалізовано через SQL-запити з використанням Npgsql.
- База даних: PostgreSQL, нормалізована структура, підтримка зовнішніх ключів.

Завдяки модульній структурі, кожен компонент системи легко підтримувати, розширювати або адаптувати під нові потреби. Кожен модуль реалізовано із врахуванням зручності для користувача, а також із забезпеченням цілісності даних через зв’язки з базою даних. Такий підхід значно полегшує масштабування проєкту та гарантує його стабільну роботу [7].

3.5 Висновки по третьому розділу

У третьому розділі було детально розглянуто процес створення прикладного програмного забезпечення інформаційної системи територіальної громади. Проведено аналіз та вибір відповідного інструментарію, зокрема середовища розробки Visual Studio та технології WinForms, які забезпечили ефективну реалізацію користувацького інтерфейсу та логіки програми.

Було розроблено інформаційну базу даних з використанням PostgreSQL, що забезпечує збереження, цілісність і структурованість даних. Представлена

структура таблиць і їх взаємозв’язків свідчить про дотримання принципів нормалізації та підтримки зовнішніх ключів.

Алгоритмізація та програмування основних функціональних модулів охоплює всі етапи життєвого циклу заяви: від реєстрації громадян до подання та обробки заяв, а також генерації звітів. Створена система є гнучкою, масштабованою та орієнтованою на зручність кінцевого користувача.

Загалом, реалізоване програмне забезпечення задовольняє вимоги, поставлені на етапі проєктування, і демонструє приклад ефективного застосування сучасних засобів розробки для створення облікових інформаційних систем.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Після розробки програмного забезпечення важливо переконатися в його коректності, надійності та відповідності функціональним вимогам. Для цього було проведено всебічне тестування створеної системи територіальної громади. Основна мета тестування – виявити помилки, недоліки в логіці програми та перевірити взаємодію між модулями та з базою даних.

4.1.1 Цілі тестування

- Перевірка коректності введення та збереження даних.
- Перевірка функціональності інтерфейсу.
- Перевірка обробки помилок та виключних ситуацій.
- Перевірка взаємодії з базою даних PostgreSQL.
- Перевірка відповідності фактичної поведінки програмного забезпечення очікуваним результатам.

4.1.2 Методи тестування

У процесі тестування були використані наступні методи:

- Модульне тестування – перевірка окремих функціональних компонентів, таких як модулі обробки заяв, взаємодії з базою даних, реєстрації користувача тощо.
- Інтеграційне тестування – перевірка взаємодії між модулями
- Системне тестування – тестування повної системи в умовах, наближених до реального середовища.
- Тестування користувацького інтерфейсу (UI testing) – перевірка зручності, логіки та відповідності інтерфейсу очікуваному функціоналу[9].

4.1.3 Тестові сценарії

Сценарій 1. Реєстрація нового громадянина

Вхідні дані: Ім’я, прізвище, по батькові, email, телефон, вибір адреси.

Очікуваний результат: Жителя зареєстровано, інформація зберігається в таблиці Citizen.

Результат тесту: Успішно.

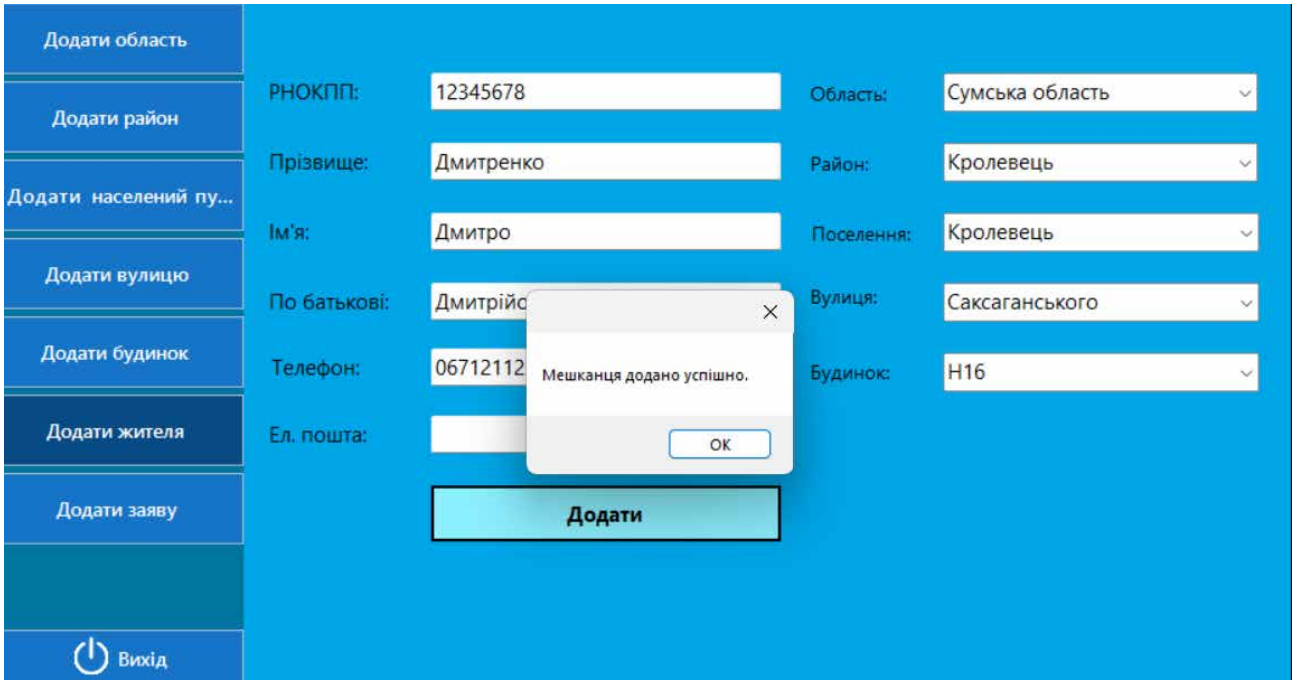

Рис. 4.1 – Додавання нового жителя до системи

Сценарій 2. Подання заяви

Вхідні дані: Вибір жителя, вибір категорії, опис проблеми.

Очікуваний результат: Заява створена, статус – «Нова», інформація потрапляє в таблицю Statement.

Результат тесту: Успішно.

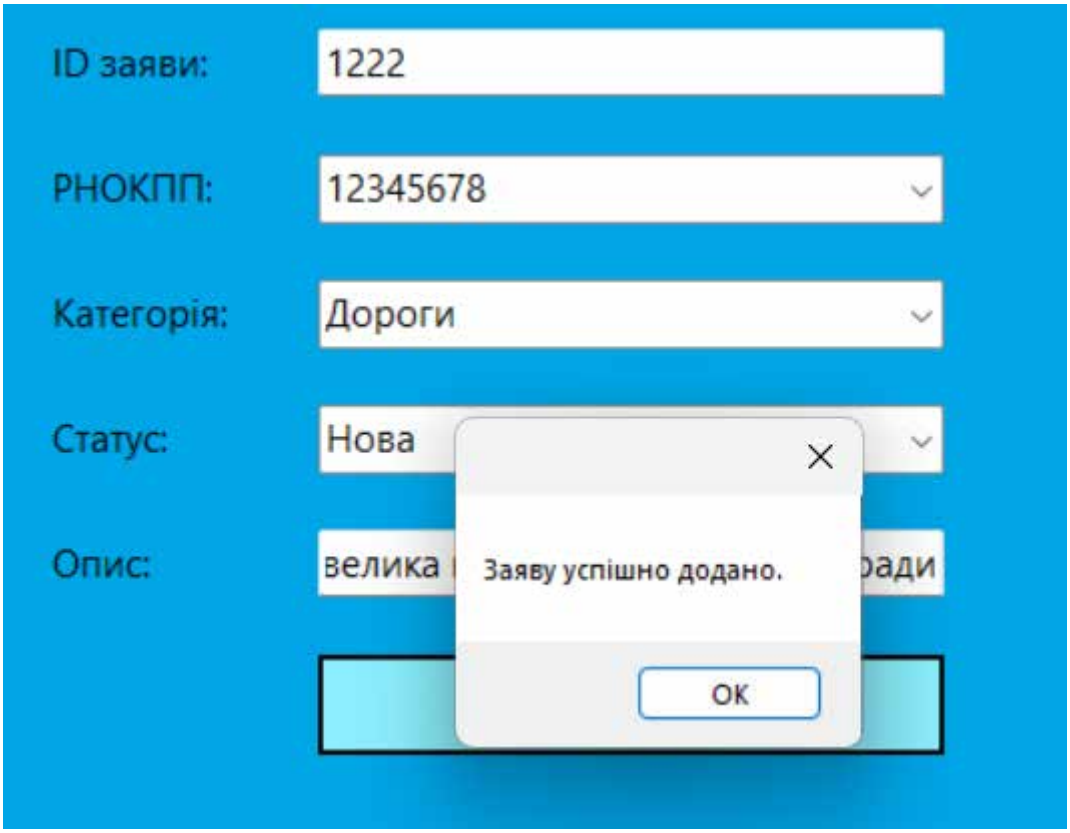

Рис 4.2 – Створення заяви у системі

Сценарій 3. Обробка заяви секретарем

Дії: Зміна статусу заяви на «В обробці»

Очікуваний результат: Дані оновлено в таблиці Statement, зміни видно в інтерфейсі.

Результат тесту: Успішно.

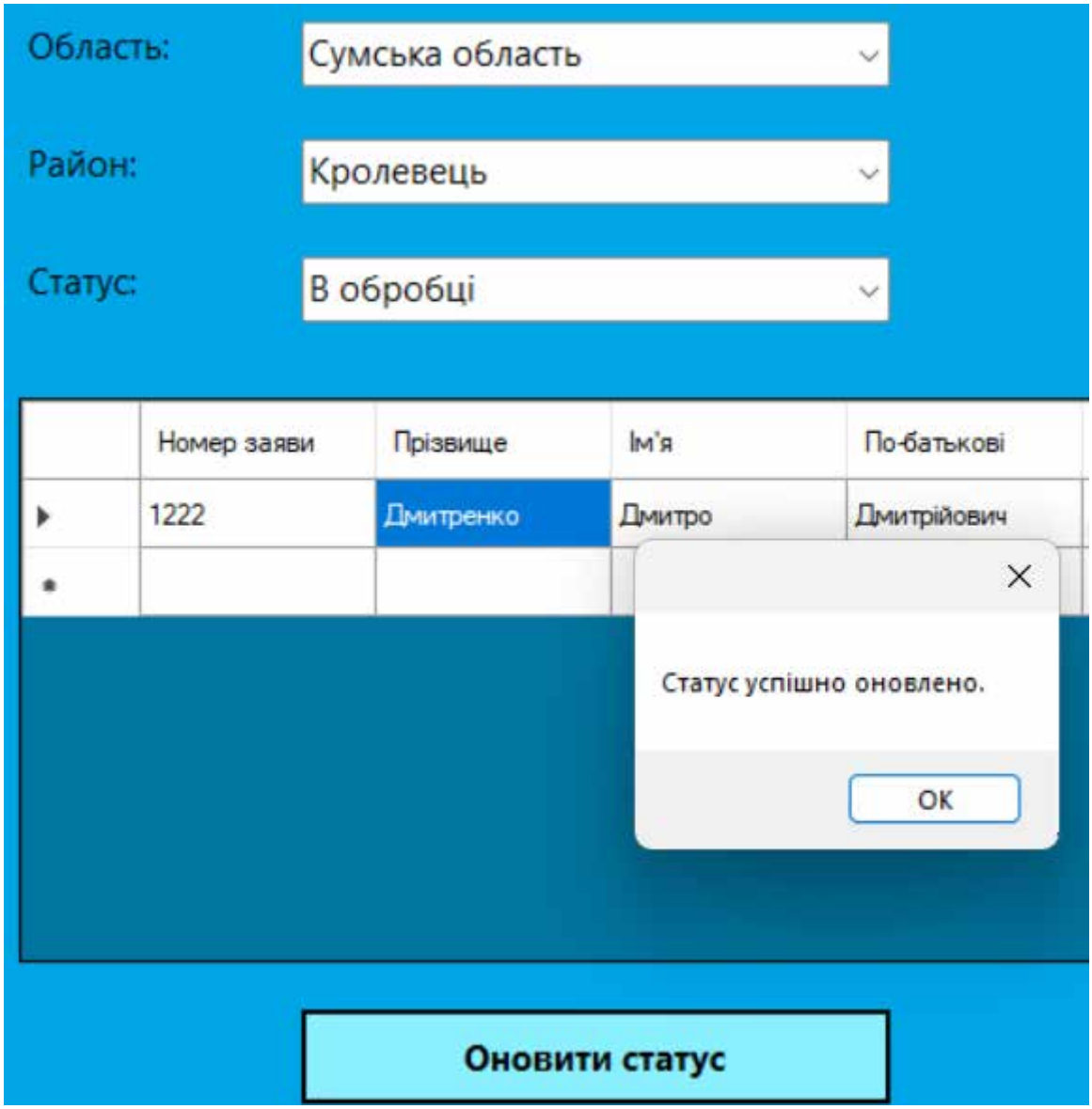

Рис. 4.3 – Зміна статусу заяви секретарем

Сценарій 4. Некоректне введення даних для входу до програми

Симуляція: Користувач.

Очікуваний результат: Виведення повідомлення про помилку підключення.

Результат тесту: Успішно.

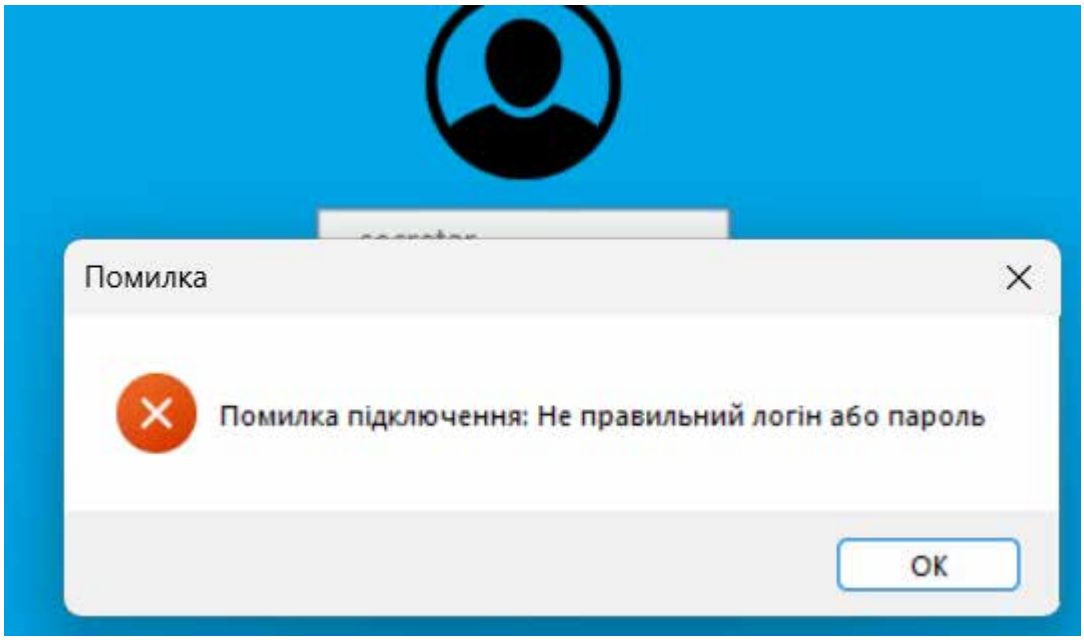

Рис 4.4 – Спроба підключення з неправильним логіном або паролем

4.1.4 Інструменти, використані для тестування

- Вбудовані можливості Visual Studio (режим налагодження).
- MessageBox-інформування для відображення помилок.
- PostgreSQL Console для перевірки даних у базі даних.
- Логування в консоль та базу даних для відстеження транзакцій.

4.1.5 Результати тестування

Усі базові функціональні можливості системи були протестовані та продемонстрували стабільну роботу. Система успішно виконує:

- збереження та редагування даних;
- коректну фільтрацію (наприклад, районів за обраною областю);
- зміну статусів заяв;
- перевірку валідності введених даних.

Виявлені помилки у ході тестування стосувались в основному:

- неправильного відображення повідомлень при незаповнених обов’язкових полях;
- неповного оновлення елементів ComboBox після змін в базі даних.

Ці помилки були оперативно усунуті, і система була стабілізована.

Тестування показало, що розроблена система є працездатною та відповідає вимогам. Завдяки модульному підходу реалізації та логічному структуруванню компонентів забезпечується надійна робота та легка підтримка програмного забезпечення. Система готова до розгортання у середовищі кінцевого користувача.

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення ефективної роботи готового програмного продукту, створеного для обробки заяв, необхідно врахувати низку технічних вимог. Вони охоплюють як апаратне забезпечення, так і програмне середовище в якому функціонує система. Вимоги, за стандартом, поділяються на мінімальні – для запуску та тестування, та рекомендовані – для повноцінного та стабільного використання в реальному середовищі.

4.2.1 Апаратна складова

Для початку розглянемо серверну частину, тобто де буде розміщеного нашу базу даних для подальшого для неї підключення. Я визначив такі потрібні характеристики:

Таблиця 4.1 Мінімальні вимоги до сервера

Компонент	Мінімальні	Рекомендовані
Процесор	2 ядра, частота 2.0 ГГц	4 ядра, частота 2.5ГГц або вища
Оперативна пам’ять	4 ГБ	8-16 ГБ
Накопичувач	20 ГБ SSD	Від 50 ГБ SSD (для бекапів)
Мережа	100 Мбіт/с	1 Гбіт/с
Операційна система	Linux	Linux з автоматизованим резервним копіюванням

Наступним ми визначимо вимоги для комп’ютера користувача який буде використовувати програму

Таблиця 4.2 Клієнтська частина

Компонент	Мінімальні	Рекомендовані
Процесор	Intel core I3 або аналогічний	Intel core I5 / AMD Ryzen 5 і вище
ОЗУ	4 ГБ	8 ГБ або більше
Місце на диску	Вільні 4 ГБ	Вільні 4 ГБ
Операційна система	Windows 10 (64-bit)	Windows 11 (64-bit)

.Net Framework	Не нижче 4.7.2	Сучасна стабільна версія

4.2.2 Програмне забезпечення

Також я визначив потрібне програмне забезпечення для серверу та користувача:

Таблиця 4.3 Серверна інфраструктура

Програмне рішення	Призначення або версія
PostgreSQL	Не нижче 17-ї версії
pgAdmin	Інтерфейс керування базою даних (версія 6 і новіше)
Npgsql драйвер	Потрібен для взаємодії WinForms із PostgreSQL
SSH-сервер	Для безпечного доступу до сервера ззовні
Планувальний (cron)	Для регулярного створення резервних копій

Таблиця 4.4 Клієнтське програмне забезпечення

Програмне забезпечення	Функціональне призначення
Windows (ОС)	Платформа для запуску додатку
Microsoft .Net Framework	Середовище для виконання застосунку
Visual C++ Redistributable	Бібліотека для деяких компонентів системи

WinForms-додаток	Основний застосунок для взаємодії з даними

Таблиця 4.5 Мережеве забезпечення

Компонент	Опис вимог
З’єднання з сервером	Швидкий інтернет для доступу до бази даних
Швидкість передавання	Від 10 Мбіт/с для комфортної роботи з системою
Стабільність зв’язку	Бажаний аптайм – не менше 99% часу
Захист мережі	Наявність фаєрволу, автентифікація, шифрування даних

4.2.3 Забезпечення надійності та безпеки

Щоб унеможливити втрату інформації та зберегти конфіденційність даних, рекомендується дотримуватися таких заходів:

- Налаштувати регулярне резервне копіювання даних бази даних
- Забезпечити робочі станції антивірусним програмним забезпеченям

Система побудована так, щоб працювати в клієнт-серверній архітектурі. Завдяки помірним вимогам до апаратної частини, витрати на її впровадження мінімальні. Дотримання написаних мною параметрів дозволить забезпечити

стабільну та безпечну експлуатацію системи при звичайному та інтенсивному навантаженні.

4.3 Склад інсталяційного пакету

Для забезпечення зручного розгортання розробленого прикладного програмного забезпечення користувачам було створено інсталяційний пакет за допомогою вбудованих засобів Visual Studio. Такий підхід дозволяє автоматизувати процес встановлення програми, включити всі необхідні залежності та гарантувати коректну інтеграцію з операційною системою Windows.

4.3.1 Засоби створення інсталяційного пакету

Інсталяційний пакет було зібрано за допомогою шаблону Publish в Visual Studio з вибором методу публікації – Folder з наступним застосуванням опції "ClickOnce". Цей підхід дозволив створити інсталятор, що включає виконуваний файл програми, файл встановлення (setup.exe), а також папку Application Files, яка містить усі додаткові ресурси, бібліотеки та конфігураційні файли.

Процес публікації включав такі етапи:

- Налаштування параметрів проєкту публікації.
- Визначення способу розгортання (ClickOnce, окремий каталог).
- Автоматичне формування пакету Visual Studio.
- Створення каталогу, який містить усі інсталяційні файли.

4.3.2 Основні компоненти інсталяційного пакету

Після виконання інсталяції користувач отримує доступ до наступної структури файлів і папок:

- setup.exe – це головний файл запуску встановлення. При запуску setup.exe відбувається:
  1) Перевірка системи на наявність необхідного програмного забезпечення
  2) Ініціалізація встановлення ClickOnce.
  3) Завантаження усіх необхідних компонентів з папки Application Files.
  4) Реєстрація ярликів, елементів автозавантаження (якщо передбачено), а також можливість автоматичного оновлення (за потреби).
- TSprog.exe – це основний виконуваний файл прикладного програмного забезпечення – безпосередньо сама WinForms-програма для роботи з базою даних заяв. Саме він запускається користувачем для повсякденної роботи. Файл містить:
  1) Реалізований графічний інтерфейс користувача (GUI).
  2) Логіку взаємодії з PostgreSQL через Npgsql.
  3) Засоби авторизації, фільтрації, перегляду, редагування даних.
  4) Механізми оновлення статусів заяв, генерації звітів тощо.
- Папка Application Files – У цій папці зберігаються всі необхідні залежності, компоненти та ресурси для роботи програми. Структура папки зазвичай включає одну або кілька вкладених директорій, які називаються відповідно до версії застосунку. Ці файли потрібні ClickOnce для відновлення, перевірки цілісності програми та її коректного встановлення на клієнтську машину.

Серед важливих складових:

- .deploy-файли – за замовчуванням ClickOnce перейменовує всі файли, додаючи розширення .deploy для захисту від прямого виконання з файлової системи.
- .manifest – XML-файл, який містить мета-інформацію про застосунок: дозволи, залежності, ресурси, версії тощо.
- Файли ресурсів – зображення, іконки, звукові або текстові ресурси, які використовуються в інтерфейсі програми.

4.3.3 Вміст метаданих та конфігурацій

Поряд з основними файлами до складу інсталяції входять:

- TSprog.application – ClickOnce-документ, який описує логіку оновлень, місце встановлення, URL-адресу джерела публікації (якщо використовується).
- config.xml або app.config – конфігураційні файли з параметрами підключення до бази даних, шляхами до звітів або іншим середовищем виконання.
- Журнали встановлення, створювані під час інсталяції (при включеному логуванні).

4.3.4 Рекомендації з використання інсталяційного пакету

Перед встановленням рекомендується переконатися у наявності на комп’ютері:

- NET Framework 4.7.2 або новішої версії.
- Visual C++ Redistributable (при використанні сторонніх бібліотек).
- Доступу до інтернету (для автоматичних оновлень).

Встановлення здійснюється шляхом запуску setup.exe з правами адміністратора. Після встановлення програма автоматично створює ярлик у меню «Пуск» або на робочому столі.

4.3.5 Переваги використання ClickOnce

Застосування ClickOnce дало змогу:

- Забезпечити простоту встановлення «в один клік».
- Автоматично додавати всі потрібні залежності до пакету.
- Налаштувати оновлення програми з одного джерела.
- Гарантувати цілісність та безпечність пакету за рахунок підпису та перевірки контрольних сум[10].

4.4 Висновки по четвертому розділу

У четвертому розділі було здійснено повний цикл тестування розробленого прикладного програмного забезпечення. Визначено технічні вимоги до апаратної й програмної частин системи, а також представлено структуру інсталяційного пакету. Проведене тестування засвідчило стабільну роботу ключових функціональних модулів, таких як введення, перегляд, фільтрація й оновлення заяв, формування звітів та взаємодія з базою даних PostgreSQL. Система демонструє надійну роботу як у локальному середовищі, так і при мережевому використанні.

Було встановлено мінімальні та рекомендовані вимоги до серверного й клієнтського обладнання, що дає змогу забезпечити належний рівень продуктивності за різних умов експлуатації. Окрему увагу приділено програмним залежностям, необхідним для коректного функціонування застосунку, серед яких — платформа .NET Framework, драйвер Npgsql та інші необхідні компоненти.

Інсталяційний пакет, сформований за допомогою інструментів Visual Studio, забезпечує простий і надійний процес розгортання системи. Завдяки використанню технології ClickOnce, інсталяція та оновлення програмного забезпечення доступні навіть для користувачів без спеціальної технічної підготовки.

Підсумовуючи, можна стверджувати, що розроблена система є технічно завершеною, відповідає поставленим вимогам і готова до впровадження в реальному середовищі.

ВИСНОВКИ

У ході виконання цього проєкту мені вдалося на практиці реалізувати всі етапи створення прикладного програмного забезпечення — від аналізу вимог до розгортання та тестування готової системи. Проєктування та реалізація програми управління заявами дало змогу поєднати теоретичні знання з прикладними навичками, що мають важливе значення для подальшої професійної діяльності.

Особливо корисним було вивчення архітектури програмного забезпечення та принципів побудови сучасних інформаційних систем. Робота над базою даних, яка містить багаторівневу структуру (області, райони, населені пункти, вулиці, будинки, громадяни, заяви тощо), дозволила зрозуміти особливості реляційної моделі даних, нормалізації, зв’язків між таблицями та ефективної організації запитів. Застосування системи керування базами даних PostgreSQL дало змогу забезпечити високу надійність і продуктивність.

Інтерфейс програми був реалізований засобами Windows Forms, що дало змогу створити зручне робоче середовище для користувачів. Завдяки модульній структурі програми стало можливим швидке доопрацювання та розширення функціональності, що робить її гнучкою та придатною для адаптації під інші потреби.

Окрему увагу було приділено тестуванню функціональності та оцінці працездатності системи в умовах різного навантаження. Це дозволило виявити потенційні помилки та усунути їх ще до впровадження. Завдяки цьому сформувалося розуміння важливості контролю якості програмного забезпечення.

У підсумку, виконана мною робота підтверджує, що я маю практичні навички створення, налагодження та підтримки інформаційних систем. Проєкт демонструє мою здатність аналізувати завдання, приймати технічні рішення та реалізовувати їх у формі повноцінного програмного продукту, який може бути використаний у реальному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Create a Windows Forms app with C# - Visual Studio (Windows). *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/visualstudio/IDe/create-csharp-winform-visual-studio?view=vs-2022 (Дата звернення: 11.05.2025).
2. Батанов О. В. Територіальна громада. *Енциклопедія Сучасної України*. URL: https://esu.com.ua/article-879248 (Дата звернення: 11.05.2025).
3. Інформаційна система «Вулик». *Інформаційна система «Вулик»*. URL: https://vulyk.gov.ua/ (Дата звернення: 12.05.2025).
4. PostgreSQL: About. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (Дата звернення: 13.05.2025).
5. Посібник із побудови схем UML і моделювання баз даних. *Посібник із побудови схем UML і моделювання баз даних*. URL: https://www.microsoft.com/uk-ua/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling (Дата звернення: 16.05.2025).
6. Реляційні бази даних. *Relational databases*. URL: https://rdb.dp.ua/uk/chapter_03 (Дата звернення: 16.05.2025).
7. Голуб Б.Л. Методичний посібник до вивчення дисципліни «Програмування та алгоритмічні мови». Навчальне видання Голуб Б.Л., Щукайло Є.М.-К-, Видавничий центр НАУ. 2003. – 64 с

8. Introduction of ER Model - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/introduction-of-er-model/ (Дата звернення: 16.05.2025).
9. Singureanu C. Системне тестування - типи, процес, інструменти та багато іншого!. *ZAPTEST*. URL: https://www.zaptest.com/uk/що-таке-системне-тестування-глибоке-з (Дата звернення: 23.05.2025).
10. Create an App Installer file with Visual Studio - MSIX. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/windows/msix/app-installer/create-appinstallerfile-vs (Дата звернення: 23.05.2025).
11. Технічне завдання (ТЗ) на розробку програмного забезпечення: Все, що вам потрібно знати | Блог на Stfalcon. *Custom Software Development Company | Stfalcon.com*. URL: https://stfalcon.com/uk/blog/post/statement-of-work-software-development (Дата звернення: 23.05.2025).
12. Моралес Дж. What is UML Class Diagram including UML Class Diagram Maker. *MindOnMap*. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (Дата звернення: 23.05.2025).

ДОДАТКИ

Додаток А

Код створеної бази даних

```
DROP TABLE IF EXISTS Statement;
DROP TABLE IF EXISTS Statement_status;
DROP TABLE IF EXISTS Categories;
DROP TABLE IF EXISTS Citizen;
DROP TABLE IF EXISTS House;
DROP TABLE IF EXISTS House_Type;
DROP TABLE IF EXISTS Street;
DROP TABLE IF EXISTS Settlement;
DROP TABLE IF EXISTS District;
DROP TABLE IF EXISTS Region;


CREATE TABLE Region (
    Region_ID CHAR(10) PRIMARY KEY,
    Region_name VARCHAR(255) NOT NULL
);

CREATE TABLE District (
    District_ID CHAR(10) PRIMARY KEY,
    Region_ID CHAR(10),
    District_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (Region_ID) REFERENCES Region(Region_ID)
);

CREATE TABLE Settlement (
    Settlement_ID CHAR(10) PRIMARY KEY,
```

```
    District_ID CHAR(10),
    Settlement_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (District_ID) REFERENCES District(District_ID)
);

CREATE TABLE Street (
    Street_ID CHAR(10) PRIMARY KEY,
    Settlement_ID CHAR(10),
    Street_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (Settlement_ID) REFERENCES Settlement(Settlement_ID)
);

CREATE TABLE House_Type (
    Type_ID SERIAL PRIMARY KEY,
    Type_Name VARCHAR(255)
);

CREATE TABLE House (
    House_ID CHAR(10) PRIMARY KEY,
    Street_ID CHAR(10),
    House_Type INT,
    FOREIGN KEY (Street_ID) REFERENCES Street(Street_ID),
    FOREIGN KEY (House_Type) REFERENCES House_Type(Type_ID)
);

CREATE TABLE Citizen (
    ID_user CHAR(10) PRIMARY KEY,
    First_name VARCHAR(255) NOT NULL,
    Last_name VARCHAR(255) NOT NULL,
    Patronymic VARCHAR(255) NOT NULL,
    Email VARCHAR(255),
    Phone_num VARCHAR(20),
    House CHAR(10),
    FOREIGN KEY (House) REFERENCES House(House_ID)
);

CREATE TABLE Categories (
    Category_ID SERIAL PRIMARY KEY,
```

```
    Category_name VARCHAR(255) NOT NULL,
    Description TEXT
);

CREATE TABLE Statement_status (
    Status_ID SERIAL PRIMARY KEY,
    Status_name VARCHAR(255) NOT NULL
);

CREATE TABLE Statement (
    Statement_ID CHAR(10) PRIMARY KEY,
    ID_User CHAR(10),
    Category_ID INT,
    Status_ID INT,
    Create_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Overview TEXT,
    FOREIGN KEY (ID_User) REFERENCES Citizen(ID_user),
    FOREIGN KEY (Category_ID) REFERENCES Categories(Category_ID),
    FOREIGN KEY (Status_ID) REFERENCES Statement_status(Status_ID)
);
```